Microcontrôleur 32 bits / STM32
Majeure Numérique
TP1 : Liaison SPI - Les bases

Y. Bornat V. Lebret

April 2, 2025

Pour les enseignements spécialisés, les TPs de microcontrôleurs utiliseront la liaison SPI comme base de travail. Il s'agit par contre de la suite directe des enseignements de tronc commun effectués au S7. N'hésitez pas à ressortir les textes de TPs ainsi que vos codes précédents. On commencera par de la scrutation pour repartir sur des bases saines, mais il y aura pas mal d'accès matériels.

1 Introduction

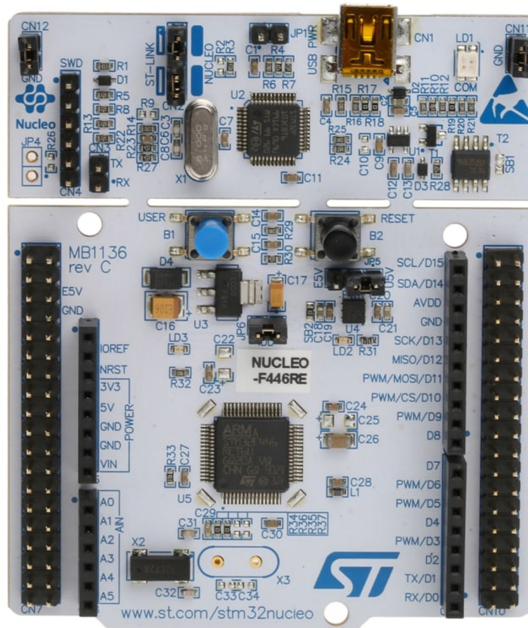

Figure 1: La carte *Nucleo-F446RE*

1.1 Deux cartes pour le prix d'une !

Étant donné que la carte STM32 est assez pauvre en application, nous allons communiquer avec la carte *basys MX3* que vous connaissez déjà des TPs de C++. L'accent sera mis sur la communication avec cette carte car c'est un très bon support de manipulation. La carte *basys MX3* doit donc être vue comme un périphérique controlée par le micro-contrôleur STM32. La liaison entre les deux cartes est un bus SPI qui sera détaillé plus tard.

1.2 La carte Nucleo F446RE

Vous conaissez déjà la carte, présentée figure 1 donc la présentation sera rapide. La liaison SPI se fait sur le connecteur de droite. Normalement, elle est déjà fonctionnelle.

Cette carte se programme à l'aide de l'environnement *STM32 Cube IDE*. Cet environnement est spécifique à la marque ST, il n'y a donc aucun intérêt pédagogique de s'attarder sur ses fonctionnalités. Des projets fonctionnels sont fournis pour que vous puissiez être opérationnel au plus tôt.

1.3 La carte *basys MX3*

Il s'agit d'une carte microcontrôleur qui a le bon goût d'avoir énormément de modules d'application embarqués, mais le mauvais goût d'être en obsolescence. L'usage que nous verrons de cette carte n'est pas natif. Il s'agit d'une application que les enseignants ont développé pour que cette carte reste utilisable dans des circonstances autres que la programmation de son microcontrôleur. Vous savez déjà que son usage a été pensé pour être accessible au moyen de registres 8 bits.

Le protocole mis en place n'est pas générique, mais il est très proche de ce qui existe sur d'autres composants. A l'issue des TPs, vous n'aurez donc pas appris à utiliser cette carte. Juste à communiquer avec une application qui tourne dessus.

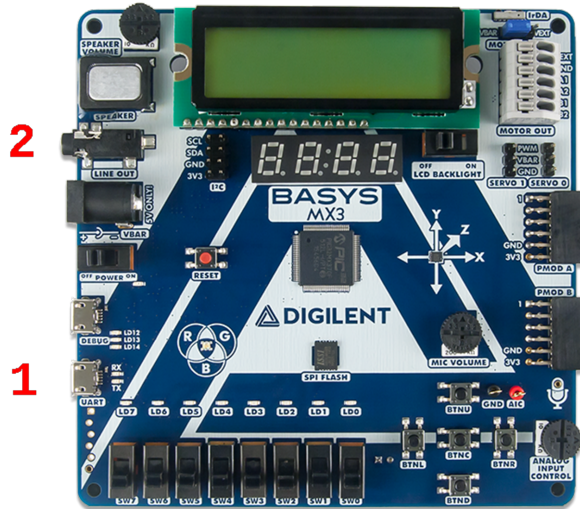

Figure 2: La carte *basys MX3*

La suite de cette partie est un simple référence technique. Dans un premier temps, vous pouvez passer directement à la partie 1.4

1.3.1 Registres de la carte MX3

Pour rappel, le tableau 1 rassemble les registres disponibles sur la carte. L'accès précise si le registre concerné est en lecture seule, écriture seule ou lecture écriture. Une lecture sur un registre en écriture seule renverra toujours la valeur 0.

L'accès à la plupart des fonctions est assez intuitif, du moins nous l'espérons. Pour les switchs ou les LEDs, chaque bit correspond à un switch ou une LED. Pour la LED *RGB*, chacune des trois valeurs code la composante de couleur. Vous trouverez ci-après quelques précisions concernant certains périphériques.

1.3.2 L'écran LCD

Pour utiliser l'écran LCD, il suffit d'écrire dans l'un des 32 registres associés. Chaque registre correspond à un caractère visible à l'écran. Ainsi les adresses `MX3ADDR_LCD_START` à `MX3ADDR_LCD_START+15` correspondent aux caractères de la ligne du haut, et les adresses `MX3ADDR_LCD_START+16` à `MX3ADDR_LCD_START+31` à la ligne du bas.

Dec	Hex	ID	R/W	fonction
00	00	MX3ADDR_TEST	RO	Valeur de test : 75
01	01	MX3ADDR_SW	RO	Valeur des switches
02	02	MX3ADDR_LED	R/W	Valeur des LEDs
03	03			
04	04	MX3ADDR_FLASH_RFIFO	RO	Lecture Incrémentale de la mémoire Flash
05	05	MX3ADDR_FLASH_PTR_L	R/W	Index de lecture de la mémoire Flash, remis à jour si lecture dans MX3ADDR_FLASH_RFIFO. Si modifié, la nouvelle valeur n'est prise en compte qu'à l'écriture de MX3ADDR_FLASH_PTR_H
06	06	MX3ADDR_FLASH_PTR_M	R/W	
07	07	MX3ADDR_FLASH_PTR_H	R/W	
08	08	MX3ADDR_RGBLED_R	R/W	Composante rouge pour la LED RGB
09	09	MX3ADDR_RGBLED_G	R/W	Composante verte pour la LED RGB
10	0A	MX3ADDR_RGBLED_B	R/W	Composante bleue pour la LED RGB
11	0B	MX3ADDR_SND_WFIFO	WO	Écriture incrémentale des échantillons sonores
12	0C	MX3ADDR_SND_ELTS_L	RO	Nombre d'échantillons sonores présents dans la FIFO
13	0D	MX3ADDR_SND_ELTS_H	RO	
14	0E	MX3ADDR_UPTIME_L	RO	Nombre de secondes écoulées depuis la derniere initialisation de la carte
15	0F	MX3ADDR_UPTIME_H	RO	
16	10	MX3ADDR_7SEG_DEC_L	R/W	Valeur de l'afficheur à 7 segments. Ces registres activent le mode décimal.
17	11	MX3ADDR_7SEG_DEC_H	R/W	
18	12	MX3ADDR_7SEG_HEX_L	R/W	Valeur de l'afficheur à 7 segments. Ces registres activent le mode hexadécimal.
19	13	MX3ADDR_7SEG_HEX_H	R/W	
20	14	MX3ADDR_7SEG_MAP_0	R/W	Manipulation fine des afficheurs à 7 segments Chaque registre identifie un afficheur, chaque bit contrôle un segment. Ces registres activent le mode 'map'.
21	15	MX3ADDR_7SEG_MAP_1	R/W	
22	16	MX3ADDR_7SEG_MAP_2	R/W	
23	17	MX3ADDR_7SEG_MAP_3	R/W	
24	18	MX3ADDR_B_IMAT_LL	RO	Numéro d'identification individuel de la carte. Le numéro de série hexadécimal figure également sur l'étiquette au dos de la carte.
25	19	MX3ADDR_B_IMAT_LH	RO	
26	1A	MX3ADDR_B_IMAT_HL	RO	
27	1B	MX3ADDR_B_IMAT_HH	RO	
28	1C			
29	1D			
30	1E			
31	1F	MX3ADDR_STATUS	R/W	Registre de Statut de la carte. Cf tableau 3 pour les détails.
32 ... 63	20 ... 3F	MX3ADDR_LCD_START - -	R/W ... R/W	Caractères affichés sur l'écran LCD

Table 1: Tableau descriptif des registres accessibles sur la carte Basys MX3

ATTENTION : : L'utilisation de l'écran LCD entraine une certaine latence à la prise en charge des écritures. Consultez la section 1.5.2 pour plus de détails.

1.3.3 La sortie audio

La carte *basys MX3* est dotée d'une sortie audio (haut parleur ou sortie jack, référencée 2 sur la figure 2) dont le niveau sonore est réglable (potentiomètre). La valeur de sortie est définie sur 8 bits signés par échantillon, à une fréquence de 11,025 KHz (défaut) ou de 22,05KHz. Hstoire de simplifier l'usage, il n'y a que 3 registres à manipuler (dont un registre 16 bits qui occupe donc deux adresses).

- Registre de *status* : seuls les deux bits de poids faible sont utiles pour la sortie audio. Le bit 0 permet d'activer (1) ou désactiver (0) la sortie. Le bit 1 permet de définir la vitesse d'échantillonnage à 22,0kHz (1) ou à 11,02kHz (0).
- MX3ADDR_SND_WFIFO est le point d'entrée d'une File d'attente (FIFO) de 64ko. Si la sortie audio est activée (bit 0 du *status* à 1), les élements de la FIFO sont consommés au rythme de

l'échantillonnage sonore. Sinon, les échantillons attendent.

- `MX3ADDR_SND_ELTS_L` / `MX3ADDR_SND_ELTS_H` contiennent le nombre d'échantillons stockés dans la FIFO. Étant donné que la FIFO a une taille de 64ko, le nombre d'échantillons est sur 16 bits, et la valeur est accessible sur deux registres (`L` pour *low*, l'octet le moins significatif et `H` pour *high*, l'octet le plus significatif)

1.3.4 La mémoire Flash

La carte est équipée d'une mémoire Flash de 4Mo. L'usage est assez basique : il y a un registre d'adresse sur 22 bits et un registre de lecture. Pour lire une donnée de la mémoire Flash, il suffit d'écrire l'adresse de lecture (dans la mémoire Flash) sur les registres `MX3ADDR_FLASH_PTR_L`, `MX3ADDR_FLASH_PTR_M` et `MX3ADDR_FLASH_PTR_H`. La valeur contenue à cette adresse est alors accessible dans le registre `MX3ADDR_FLASH_RFIFO`. Une fois la valeur lue, le registre d'adresse est incrémenté, il est donc possible de lire plusieurs fois `MX3ADDR_FLASH_RFIFO` pour récupérer le contenu de la mémoire.

Techniquement, il n'y a pas de FIFO derrière le registre `MX3ADDR_FLASH_RFIFO`, contrairement à ce qu'il se passe pour la sortie sonore. Mais il porte le nom de FIFO car il se comporte de la même manière : chaque lecture consomme la donnée qui a été lue, et plusieurs lectures successives permettent de récupérer plusieurs données successives.

Lors de la modification du registre d'adresse, il est essentiel d'écrire entièrement la nouvelle adresse, car la mise à jour est effectuée à l'écriture dans le registre le plus significatif `MX3ADDR_FLASH_PTR_H`. Toute modification des deux autres registres sera annulée à la moindre lecture de `MX3ADDR_FLASH_WFIFO` à cause de l'incrémentation automatique.

Le contenu de la mémoire est fourni par le tableau 2. Il n'est pas possible d'écrire dans la mémoire Flash (Ou pour être plus rigoureux, le programme qui tourne sur le microcontrôleur ne le permet pas).

@ (Déc)	@ (Hex)	Taille	Description
0	0x000000	32	Texte de confirmation de lecture de la Flash
40	0x000028	7	Numéro de série de la carte en représentation ASCii Ce numéro est également disponible dans les registres `MX3ADDR_B_IMAT_XX` au format binaire
50	0x000032	1323000	Morceau musical de 60 secondes, au format signé, 8 bits par échantillon, échantillonné à 22,05kHz

Table 2: Contenu de la mémoire Flash

A partir du moment où on manipule la mémoire Flash, il faut devenir très précis dans le vocabulaire. En effet, vous serez confrontés aux adresses des registres de la carte (sur 8 bits), et aux adresses des données de la mémoire Flash (sur 22 bits). Sans compter qu'il y a aussi les adresses mémoire sur l'ordinateur. Bref, Les raccourcis de langage risquent fort de brouiller le message, plutôt que de l'expliquer.

1.3.5 Afficheur 7 segments

Il existe trois façon d'utiliser l'afficheur 7 segments.

- pour afficher une valeur décimale
- pour afficher une valeur hexadécimale
- pour afficher des symboles quelconques (mode *map*)

Dans les deux premiers modes, il suffit d'écrire une valeur non signée sur 16 bits. Selon que cette valeur est écrite aux adresses `MX3ADDR_7SEG_DEC_X` ou `MX3ADDR_7SEG_HEX_X`, cela activera le mode de représentation décimal ou hexadécimal respectivement. Mais il s'agit du même registre. Vous relirez donc la même valeur dans les deux séries de registres.

Dans le dernier mode (*map*), il est possible d'afficher n'importe quel symbole puisque chacun des quatre registres `MX3ADDR_7SEG_MAP_X` est associé à un afficheur, et que chaque bit de ces registres est associé à un segment particulier. La figure 1.3.5 indique quel segment est associé à quel bit. Écrire dans ces registres active le mode *map* automatiquement.

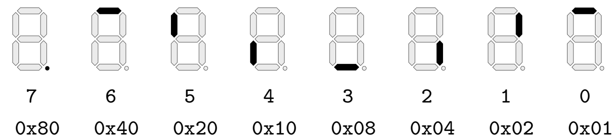

Figure 3: Répartition bit à bit des segments, la ligne supérieure indique le numéro du bit associé au segment référencé. La ligne inférieure indique le masque correspondant

1.3.6 Registre de statut (*status*)

Parmi tous les registres du tableau 1, `MX3ADDR_STATUS` a un fonctionnement particulier car chacun de ses bits a une fonction spécifique. Le tableau 3 en précise la signification.

bit	ID	R/W	fonction
0	`MX3BIT_STATUS_AUD_EN`	R/W	0 : sortie son désactivée 1 : Sortie son active
1	`MX3BIT_STATUS_AUD_FREQ`	R/W	Fréquence d'échantillonage sonore : 0 : 22,05 kHz 1 : 11,025kHz
2-3	`MX3BIT_STATUS_7SEG_MODE`	R/W	Mode des afficheurs 7 segments : 00 : éteint 01 : map 10 : décimal 11 : hexadécimal
4-7			Réservés - Lus comme 0

Table 3: Tableau descriptif des bits du registre `MX3ADDR_STATUS`

1.4 La liaison SPI

La liaison SPI (Serial Peripheral Interface) est le support de communication que nous utiliserons entre les deux cartes. Ses propriétés principales sont :

- Maître/Esclave : la communication n'est pas symétrique. Un composant (Maître) a le contrôle total de la communication. Les autres composants ou périphériques ne peuvent que répondre à des requêtes du maître. Dans notre cas, la carte *Nucléo* est le maitre, la carte *basys MX3* est l'esclave[1].
- Synchrone : Les données sont envoyées avec une horloge.
- Full duplex : Les données peuvent circuler dans les deux sens en même temps. Dans le cas de la liaison SPI, c'est systématique. Le contraire est une liaison *Half Duplex* pour laquelle un seul composant peut envoyer des données à un moment donné.

Comme illustré par la figure 4, elle est généralement composée de quatre signaux : une horloge (SCK), un signal de sélection (SS), deux signaux de données (un dans chaque sens). Ceux qui s'y connaissent un peu en électronique auront remarqué que la masse n'est pas inclue dans ces signaux, alors qu'elle est vitale pour que le transfert se passe correctement. C'est parce que cette liaison est pensée pour être locale à une carte élèctronique. Le montage que nous utilisons ici n'est donc, au mieux, que du bricolage.

La communication ne s'effectue que quand l'esclave est sélectionné par le signal SS (généralement actif à l'état bas). Les mots sont généralement transférés avec le poids fort en premier. Les transferts se font par octets.

[1]Dans certains pays qui n'assument pas leur passé, cette terminologie pose problème. Les concepteurs ont proposé Contrôleur/Périphérique, car il est apparemment moins dégradant d'être un périphérique qu'un esclave. D'autres ont proposé (en V.O.) Main/Sub, qui a l'avantage de ne pas modifier les acronymes utilisés depuis quarante ans. Dans ce texte, le parti pris est que la qualification d'esclave ne heurtera pas la sensibilité des composants. En tous cas, pas plus que périphérique ou sous-fifre.

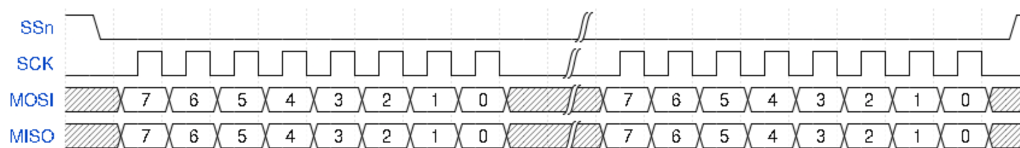

Figure 4: liaison SPI

1.4.1 Avantages

Rapide La liaison SPI peut atteindre des fréquences de l'ordre de 100 MHz. Ce sont généralement les capacités du composant cible qui limitent.

Légère Cette liaison demande peu de ressources matérielles. Elle permet donc des composants moins coûteux et qui consomment peu.

Flexible Il existe de nombreuses possibilités pour contrôler plusieurs périphériques. En parallèle : Les périphériques ont les même signaux d'horloge et de données, mais ils ont chacun leur signal SS spécifique. En série : La sortie de données du premier pariphérique est connectée à l'entrée du second, la sortie du second à l'entrée du troisième, ..., la sortie du dernier à l'entrée du maître. En bordel : n'importe quelle combinaison des deux cas précédents.

1.4.2 Désavantages

Un standard, pas une norme Et oui, la liaison SPI n'est qu'un standard... et encore ! Il s'agit d'une tentative d'uniformiser la terminologie sur des concepts qui existaient bien avant. Le résultat est que, d'une façon ou d'une autre, toutes les liaisons séries synchrones basiques peuvent rentrer dans le cadre de la liaison SPI. Par conséquent, ça ne veut pas dire grand chose. Dans la galerie des monstres, on trouve :

- Des polarités ou phases d'horloge différentes : Celui-ci est un grand classique. Pour certains composants, les données changent sur front montant, sur d'autres, c'est sur front descendant. Pour certains, l'horloge est au repos à l'état haut, pour d'autres, c'est à l'état bas ! Pour certains, les données sont valides dès l'activation de SS, pour d'autres, il faut attendre le premier front d'horloge. Ces propriétés ont été définies/résumées comme phase et polarité d'horloge. Il y a quatre possibilités, donc quatre *modes* de fonctionnement.
- Les deux signaux de données ne sont pas toujours présents. Cela semble assez logique : pour un ADC, on a juste besoin de récupérer les données, pour un DAC, juste besoin de les envoyer. On n'est donc plus en *full duplex*, mais en *simplex*, mais c'est pas grave, c'est du SPI quand même.
- Dans certains cas, c'est la même ligne de données qui est utilisée dans les deux sens de communication. On est donc en *half duplex*, mais il parait que c'est du SPI quand même, si, si, c'est le constructeur qui le dit.
- On peut se retrouver avec des mécanismes ou il y a plusieurs maîtres, techniquement, ce ne sont donc plus des maîtres, mais des pairs. Mais on dira que c'est du SPI quand même, on n'est plus à ça prêt !

Nommages fantaisistes A ce niveau, vous aurez compris qu'il y a un peu de tout derrière ce standard, mais qu'on appelle ça SPI pour rassurer le client. Le souci, c'est que parfois, on a une vrai liaison SPI sans surprise technique, mais les signaux ont des noms peu pratiques ou peu reconnaissables.

- L'horloge s'appellera généralement SCK (officiel), mais aussi CLK, CK ou tout simplement C, rien de bien compliqué ici
- la ligne de sélection est généralement SSn (Slave Select negated), $SS\#$, ou $\overline{SS}$. Cette dernière notation est moins utilisée à cause de la faible prise en charge par les outils de CAO. Pour éviter d'utiliser le mot *slave*, certains vendeurs utilisent plutôt CSn (Chip Select negated), $CS\#$ ou

encore $\overline{SS}$, ça va, on peut encore s'y retrouver. Chez certains constructeur, cela s'appellera plutôt *Sel*, parce que ... pourquoi pas ?

- Pour les signaux de donnée, ça devient vraiment compliqué. Nous ne considéreons que le signal allant du microcontrôleur au périphérique. Le standard préconise *MOSI* (Master Output Slave Input, ou, pour ceux qui privilégient le politiquement correct, Main Output Sub input). Une tendance fait apparaitre *COPI* (Controller Output Peripheral Input), toujours pour le politiquement correct. Là ou ça devient compliqué, c'est que chez certain on a *SDO* (Serial Data Out), *SO* (Serial Output), *DO* (Data Output)... Ces dernières appellation sont au mieux, dangereuses, car elles font référence au composant, mais pas à la ligne de communication. On se retrouve donc avec une ligne electrique entre le *DO* du maître au *SDI* de l'esclave, à ne pas confondre avec celle qui relie le *DI* du maître avec le *SDO* de l'esclave... Logique !

1.5 Propriétés spécifiques à la carte *Basys MX3*

La partie précédent explique comment circulent les données. Cette partie explique comment elles sont organisées. Nous quittons ce qui est standard pour aborder une organisation spécifique au projet.

1.5.1 Protocole

La présence de la ligne SSn permet de séparer des paquets de données sans ajouter de protocole particulier. C'est un avantage. Lors d'échange de données par gros volumes, il n'est pas nécessaire de définir une taille à l'avance. On informe le périphérique esclave que le paquet en cours en terminé en relevant le bit SSn.

De la même façon, lorsque le bit SSn passe à zero, l'esclave sait qu'on commence un nouveau paquet. Dans notre cas, chaque paquet commence par une *instruction* et est suivi par un nombre quelconque d'octets de données.

Une *instruction* est un octets composé de trois champs :

- un bit R/W (7) : Ce bit permet de définir si le maitre du bus veut lire ou écrire des données.
  - 0 : écriture.
  - 1 : lecture.
- un bit MAP/FIFO (6) : Ce bit décrit comment gérer l'adresse s'il y a plusieurs octets de données dans le paquet.
  - 0 (FIFO) Tous les accès de ce paquet seront effectués à la même adresse.
  - 1 (MAP) L'adresse de destination sera incrémentée pour chaque nouvel octet de donnée.
- une adresse de registre sur 6 bits (5-0) : Pour déterminer sur quel registre de l'esclave doit se faire l'opération.

Les octets suivants sont des octets de données. Leur usage est assez évident pour une instruction d'écriture. On se retrouve alors avec le chronogramme suivant de la figure 5.

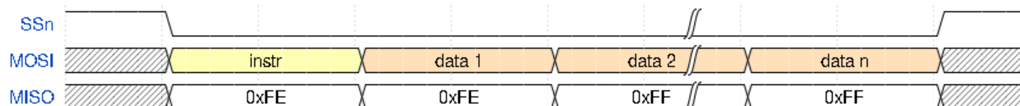

Figure 5: Succession des données pour une écriture sur la carte *basysMX3*

Dans le cas d'une lecture, la procédure se complique un peu. Il reste indispensable d'envoyer des données pour activer l'horloge, et permettre à l'esclave de renvoyer un résultat. La valeur de ces données émises par le maître n'a aucune importance. L'esclave renvoie les données demandées, mais nécessite un certain temps pour réagir à l'instruction qu'il a reçue. Il ne peut donc pas répondre tout de suite. C'est pour cela que, lors d'une lecture, la séquence des trois premiers octets renvoyés par l'esclave sera toujours la même. Ce n'est qu'au quatrième octet que le maitre commence à récupérer les données qui l'intéressent. La figure 6 illustre la succession des données.

Figure 6: Succession des données pour une lecture depuis la carte *basysMX3*

1.5.2 timings

Le principe d'une liaison SPI ne couvre que les formes des signaux, mais ne détermine rien de spécifique concernant les temps à prendre en compte. La figure 7 présente donc les temps à respecter dans le cas de la carte MX3[2]. Le respect de ces temps est crucial, en effet, s'ils ne sont pas respectés, le système pourrait fonctionner, mais sans garantie de fiabilité, ce qui, pour debugger, est pire qu'un système qui ne fonctionne juste pas.

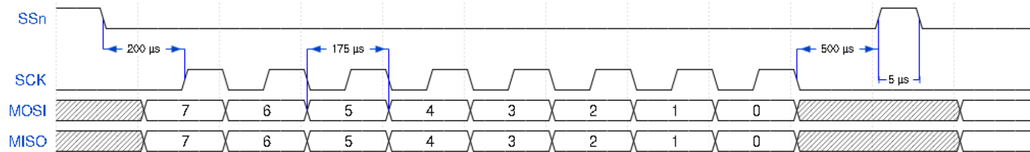

Figure 7: Temps minimaux à respecter pour la communication avec la carte *basys MX3*

Certains temps doivent également faire l'objet d'une attention particulière. En effet, dans la carte *basys MX3*, la tâche qui gère la communication SPI ne fait que gérer les flux de données. Le traitement de ces dernières est effectué par une tâche moins prioritaire. Ainsi, il est normal que la relecture d'un registre immédiatement après son écriture renvoie l'ancienne valeur. Ce comportement est volontaire, il permet de savoir si la donnée a été prise en compte. S'ensuivent les précautions suivantes :

- Quand il n'y a pas d'opération sur l'écran LCD, Les données écrites sont prises en comptes dans un délai de 1 ms.
- Chaque écriture linéaire sur l'écran LCD augmente ce délai de 450 µs. une écriture linéaire consiste à écrire les caractères sur une même ligne, dans l'ordre d'adressage (de gauche à droite).
- Chaque écriture non linéaire sur l'écran LCD fait passer ce délai à 900 µs. La différence vient de la nécessité de repositionner le curseur.
- Certaines opérations d'écriture altèrent d'autres registres. Par exemple, une écriture dans la FIFO audio, modifie le registre qui informe de son remplissage. Là aussi, il faut tenir compte du délai de traitement, ou calculer la marge d'erreur sur la valeur lue par rapport à la valeur réelle.
- La communication avec la mémoire flash de la carte *basys MX3* n'est pas protégée contre les accès concurents pour des raisons de performance. Si vous venez de modifier l'adresse de lecture, il faut toujours attendre que la propagation soit terminée, soit 1 ms, éventuellement majoré par les accès à l'écran LCD. En cas d'interférence, les données seront erronnées, mais redéfinir une adresse résoudra le problème

2 Manipulation *à la main*

Le code fourni en exemple sur la page du TP vous propose une version basique de la communication avec la carte. Ce dernier vous permettra, en cas de problème, de vérifier votre compréhension du protocole. Il s'agit d'une fonction de configuration (`SPI_init_BB()`) à appeler une seule fois au début, et d'une fonction d'échange (`SPI_exc_BB()`). Cette dernière reçoit un `unsigned char` qui sera envoyé à l'esclave, pendant le transfert, la fonction récupère la donnée envoyée par l'esclave. Quand le transfert est terminé, cette fonction retourne la valeur reçue de l'esclave au format `unsigned char`.

[2]Il faudrait plutôt parler de l'application qui tourne sur cette carte. En effet, on pourrait obtenir des temps plus courts avec un autre protocole, une moindre variété de registres et de périphériques, ou tout simplement le même programme compilé de façon plus optimale

Cette technique, appelée *bit banging* ne présente qu'un seul avantage : elle est acessible sans (trop) lire la documentation du système, et est donc facile à debugger. Pour le reste, ouvrons la boite de pandore :

- Mauvaises performances : Au delà de toute considération sur l'utilisation des ressources du processeur, cette fonction qui n'effectue jamais d'attente ne dépasse pas 1 MHz sur un processrur dont la fréquence est de 82 MHz. Le module matériel qui permet de communiquer en SPI fonctionne nativement à la moitié de la fréquence du système, soit 42 MHz dans notre cas. Sans compter la consommation, l'écart de performances est donc d'un facteur supérieur à 40 !
- Instabilité temporelle : Dans le code fourni, aucun temps n'est maitrisé. Bien sûr, on connait les temps minimum liés aux calculs, mais il suffit qu'une interruption se déclenche, et la trame de communication sera mise en pause. Certains composants matériels ne le supportent pas.
- Limites techniques : Vous aviez remarqué que le signal MOSI doit changer au même moment que le front descendant de l'horloge ? Le processeur ne peut faire qu'une chose à la fois, il sera donc difficile de faire les deux transitions simultanément. Ou plutôt, ce ne sera possible que si les deux signaux sont connectés sur le même port d'entrée/sortie.
- Gaspillage de ressources : En laissant un module matériel faire le travail, le processeur peut, au choix, faire autre chose en parallèle, se mettre en veille et économiser de l'énergie.
- Applications limitées : La technique du *bit banging* ne peut pas être utilisée sur tous les moyens de communication. Il faut que ce soit une liaison synchrone (le manque de précision temporelle rend impossible son utilisation sur l'UART), et en tant qu'esclave, il faudrait que l'horloge soit particulièrement lente.

2.1 Bonjour le monde

Juste pour se remettre dans le bain, on utilisera d'abord la fonction `SPI_Exc_BB()` pour communiquer. Pour cela, il y a besoin de la fonction `SPI_init_BB()` qui initialise les entrées/sorties en mode I/O classique. Il faudra manipuler le signal SSn indépendamment, comme vous l'aviez fait pour les LEDs au S7. C'est à dire :

- `SPI1_SSn_GPIO_Port->BSRR = SPI1_SSn_Pin;` pour mettre le signal à 1.
- `SPI1_SSn_GPIO_Port->BSRR = SPI1_SSn_Pin << 16;` pour mettre le signal à 0.

Dans les extraits de code précédents, `SPI1_SSn_GPIO_Port` est le port d'entrées/sorties auquel la broche SSn est ratachée et `SPI1_SSn_Pin` est le masque de la broche concernée.

Activité 1

Écrivez un programme qui affiche une valeur prédéfinie sur les LEDs. Vous pouvez également faire une animation, à condition qu'elle soit assez lente pour être visible (période de 100 ms minimum)

Si vous avez réussi à écrire, la lecture est à votre portée. La vraie différence est conceptuelle : deux données inutiles doivent être insérées entre l'instruction de lecture et le moment où la donnée est reçue.

Activité 2

Écrivez un programme qui lit la valeur des switchs et la recopie sur les LEDs dans une boucle infinie.

A ce niveau, vous êtes capable de faire des échanges basiques. Très bien. poussons un peu le protocole. Les échange de données peuvent se faire sur un nombre indéfini d'octets, il suffit de continuer les transferts sans relever le signal SSn.

Activité 3

- Écrivez une fonction `SPI_exc_mult()` qui reçoit un tableau de `tab` de `unsigned char` ainsi que sa longueur. Cette fonction transfère toutes les données de `tab` vers l'esclave, et modifie `tab` en y écrivant les données reçues de l'esclave.
- Écrivez un programme qui lit le registre `UPTIME` en un seul accès (échange de 5 octets donc), et recopie cette valeur sur les afficheurs 7 Segments, toujours en un seul accès (échange de 3 octets).

3 Utilisation d'un module matériel

Nous allons maintenant passer aux choses plus sérieues : l'utilisation d'un module matériel pour gérer la liaison.

La principale difficulté est que, si rien ne fonctionne, on ne sait pas pourquoi. Il est donc essentiel d'avancer par étapes et de façon méthodique.

Le SSn reste manipulé "à la main" car le matériel ne peut pas deviner le sens qu'on lui donne. Certains composants comme les ADC ou les DAC sont insensible aux montées et descentes de SSn, le signal permet simplement de savoir s'ils sont concernés ou non par le transfert. D'autres comme les mémoires Flash, crtains accéléromètres ou les cartes SD utilisent l'activité du signal SSn pour séparer des instructions, comme c'est le cas pour la carte *basys MX3*

Les autres signaux seront manipulés par le module SPI directement, ils ont donc besoin d'une configuration spécifique, comme pour l'UART au S7.

3.1 Configuration minimale

3.1.1 Activer les périphériques

Il faut tout d'abord que les périphériques soient alimentés et reçoivent l'horloge. C'est le rôle des accès sur les registres APB2ENR et AHB1ENR. Ces lignes sont fournies pour gagner du temps.

3.1.2 Configurer les Entrées/Sorties

Les entrées/sorties doivent êtres connectées directement au module qui va gérer la communication. Pour les configurer, le principe est analogue à ce qui a été fait au S7. La liste des registres du module GPIO et leur signification est donnée à partir de la page 12 de la datasheet (numérotée 186). Le tableau 4 précise la connectivité.

Fonction	Broche	Port	Numero
SSn	PB4	B	4
SCK	PB5	B	5
MISO	PB6	B	6
MOSI	PB7	B	7

Table 4: Correspondance des broches et des fonctions sur la carte *nucleo*

La configuration de SSn ne change pas par rapport à celle dans `SPI_init_BB()` car on le pilote de la même manière.

Pour chacune des trois autres broches, il faut les configurer comme étant pilotées par des *alternate functions*. Leur liste et leur signification est donnée page 59 de la datasheet du microcontrôleur (ne pas confondre avec la datasheet des modules). Il vous faudra modifier les registres MODER (pour dire qu'il faut fonctionner en mode *alternate function*) et AFR[3] (pour dire *quelle* fonction doit être sélectionner). Les autres registres vous seront inutiles ici. Pour le reste, RTFM ! Vous n'êtes plus des débutants.

[3] Techniquement, il y a deux registres AFR. Dans la datasheet, ils sont référencés AFRL (low) et AFRH (high). Dans le code, il s'agit d'un tableau de deux registres. Donc AFRL correspond à AFR[0], et AFRH à AFR[1].

ATTENTION : au moment ou on active la liaison SPI, il faut que SSn soit à 1 sinon, le module se met en echec (entrée en *Mode Fault*). Il vous appartient donc de mettre SSn à 1 dès le début et avant d'activer le module SPI.

3.1.3 Configurer le module SPI lui-même

Pour que le module SPI fonctionne, il suffit de se soucier du registre CR1, et de mettre à zero les registres CR2 et I2SCFGR. Un heureux hasard fait que la documentation du module SPI est en ligne sur la page dédiée au TP. Le registre CR1 y est détaillé page 41 (numérotée 886). Petites choses pour vous aider :

- On fonctionne en mode *full duplex*, donc pas en mode bidirectionnel.
- Le CRC est un code de détection d'erreur, le module peut le calculer automatiquement si besoin... mais on ne s'en servira pas
- Les échanges de données sont sur 8 bits
- Pour le comportement du bit SSn, bien qu'il soit généré à part, il faut tout de même s'assurer qu'une mauvaise configuration ne bloque pas le module. Nous conseillons le mode *software*. Dans ce mode, le bit SSI doit être mis à 1 pour éviter que le module ne se bloque en erreur.
- On envoie le bit de poids fort en premier (du vrai SPI quoi!)
- Pour fonctionner, le module doit être activé (Sans blague ? si si !). Plus sérieusement, c'est bien caché, mais la datasheet recommande de d'abord configurer le module inactivé, puis, de faire passer à 1 le bit d'activation.
- $f_{PCLK} = 82MHz$, pour le reste, cf partie 1.5.2.
- La carte *nucleo* est maître sur le bus
- On fonctionne en mode SPI 0 (phase = 0, polarité = 0)

Une fois la configuration terminée, écrire dans le registre `SPI1->DR` envoie un donnée. Lire ce même registre permet d'accéder à la dernière donnée reçue.

3.1.4 Yapluka

Activité 4

Écrivez un programme qui configure le module SPI1. Pour vérifier le bon fonctionnement, vérifiez si ça bouge en écrivant dans `SPI_DR` ... Pas besoin de bouger SSn, on veut juste voir de l'activité, et que cette activité corresponde à la valeur stockée dans `SPI_DR` (à la bonne fréquence, évidemment).

3.2 Transfert de données fonctionnel, mais brutal

Il s'agit maintenant d'écrire une fonction de communication SPI fonctionnelle. Cela consiste à envoyer une donnée, attendre *suffisament longtemps* pour que le transfert se produise, lire la donnée obtenue en retour.

Pour attendre *suffisament longtemps*, il se pourrait que la fonction `delay_us_timer6()` soit pratique. Contrairement à la fonction `wait_timer1()` qui attendait l'expiration d'une période. `delay_us_timer6()` attend *pendant* le temps qui est exprimé en microsecondes.

Activité 5

- Écrivez la fonction `SPI_exc()` analogue à `SPI_exc_BB()`, mais qui repose sur l'usage du module SPI matériel
- reprenez le programme 3 en utilisant `SPI_exc()`.

3.3 Un peu plus subtil

Jetez un oeil au registre SR du module SPI. Il y a tout un ensemble de bits qui donnent des informations sur le statut du module. Notamment, le bit RXNE vout dit si une donnée a été reçue de l'esclave, ou le bit BUSY vous dit si un transfert est encore en cours ...

ATTENTION : le bit RXNE peut passer à 1 AVANT que le cycle de l'horloge ne soit terminé car les données sont lues sur front montant. Il y a donc quelques précautions à prendre pour être bien conforme aux contraintes temporelles.

Activité 6

Reprenez le programme précédent, mais en utilisant les bits de status pour que les attentes soient dimensionnées juste comme il faut.

3.4 Transfert de données, mode performant

Il existe un dernier niveau de rafinement. Si l'horloge `SPI_SCK` est rapide, le temps que met le processeur pour gérer un nouveau transfert est non négligeable (consulter DR, stocker sa valeur, récupérer la données suivante et l'écrire dans DR). Cela se traduit par une pause dans la comunication et donc un eperte de performances. En s'affranchissant de la fonction `SPI_exc()`, on peut gérer les écritures sur DR à l'échelle du paquet de données, et les lectures de façon indépendantes.

Cette fois, on n'attend pas que la donnée n soit reçue avant d'envoyer la donnée $n+1$. Les échanges sont entrelacés et on évite un temps de pause entre les transferts.

Activité 7

Reprenez (encore) le programme précédent, cette fois, utilisez les lectures et écritures sur DR dès que possible pour limiter l'attente entre deux envois.

Cette dernière question est purement théorique, car la carte *basys MX3* est limitée en nombre de transferts par seconde, et il ne sert à rien d'optimiser les attentes entre les échanges.

4 conclusion

A ce stade, le pilotage d'une liaison SPI n'a plus (trop) de secrets pour vous. Mais... on n'a toujours rien à proposer pour réduire la consommation ou optimiser le temps de calcul. En effet, même si la communication est gérée au niveau matériel, le processeur reste toujours occupé à gérer les transactions en permanence. Impossible d'effectuer une tâche de fond, ou de se mettre en sommeil pour économiser de l'énergie.

Rhââââ... ce n'était que le TP1 ...