Communication Carte Nexys 4 / Joystick PmodJSTK™ via protocole SPI

[Projet en Monôme]

1 Présentation du protocole SPI (adapté de Wikipedia)

Une liaison SPI (Serial Peripheral Interface) avec un Maître et un seul Esclave est réalisée selon le schéma suivant :

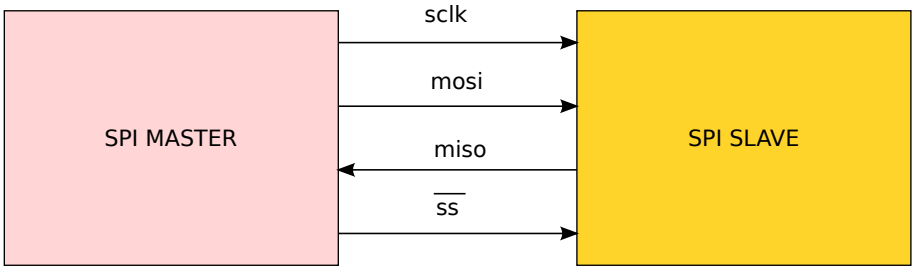

Figure 1 – Communication Maître-Esclave

Le bus SPI spécifie l'usage des 4 signaux suivants :

1. `sclk` : serial clock (produit par le Maître),
2. `mosi` : master output, slave input (produit par le Maître),
3. `miso` : master input, slave output (produit par l'Esclave),
4. `ss` : slave select (actif à niveau bas, produit par le Maître).

Une transmission, d'un ou plusieurs octects, est initiée lorsque `ss` passe de 1 (repos) à 0, et s'arrête lorsque `ss` repasse à 1. L'esclave requiert généralement un temps d'attente avant que la transmission des données commence ainsi qu'un temps d'attente entre la transmission de chaque octet.

Pour transmettre un octet, le Maître génère l'horloge `sclk`.

À chaque top de `sclk`, le Maître et l'Esclave s'échangent un bit (poids fort vers poids faible). Après huit périodes de `sclk`, le Maître et l'Esclave se sont donc échangé 1 octet.

Le protocole SPI est le plus souvent utilisé en mode 0, ce qui signifie que :

— un bit est capturé (par le Maître et par l'Esclave) lors du front montant de `sclk`,
— un nouveau bit est présenté (par le Maître et par l'Esclave) lors du front descendant de `sclk`.

2 Échange d'un octet

Le composant `er_1octet` (émission-réception 1 octet) a pour rôle d'échanger 1 octet et sera donc utilisé par les différents Maîtres développés.

Il présente l'interface suivante :

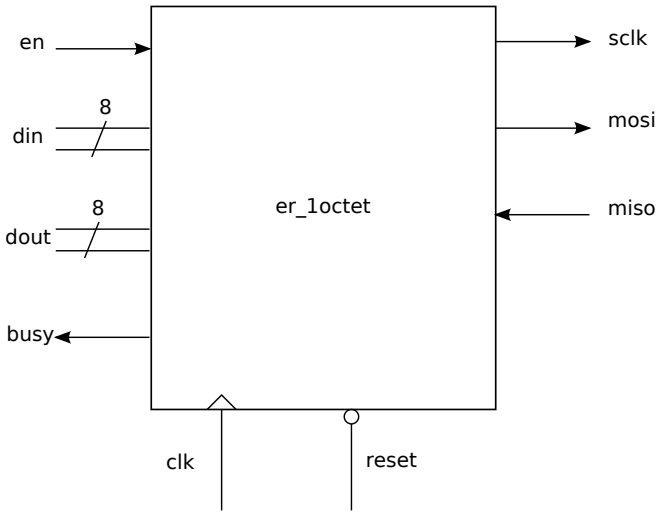

FIGURE 2 – Interface du composant d'échange d'1 octet

On retouve les 3 signaux `sclk`, `mosi` et `miso` du protocole SPI, le signal `ss` n'apparaissant pas à ce niveau.

Les signaux supplémentaires sont :

- `clk` et `reset`, l'horloge et la remise à zéro de ce composant synchrone,
- `en` (`enable`) qui indique qu'un ordre d'émission/réception d'1 octet est donné (actif à '1'),
- `din`, l'octet à émettre, sa valeur est fournie au moment où `enable` passe à '1' (elle est assurée d'être valide qu'à ce moment),
- `busy` qui à '1', indique que le composant est occupé à émettre/réceptionner,
- `dout` qui contient l'octet reçu une fois l'émission/réception terminée.

Son fonctionnement est donné par les chronogrammes des Figures 3 et 4.

Son comportement peut être décrit sous forme d'un automate à états.

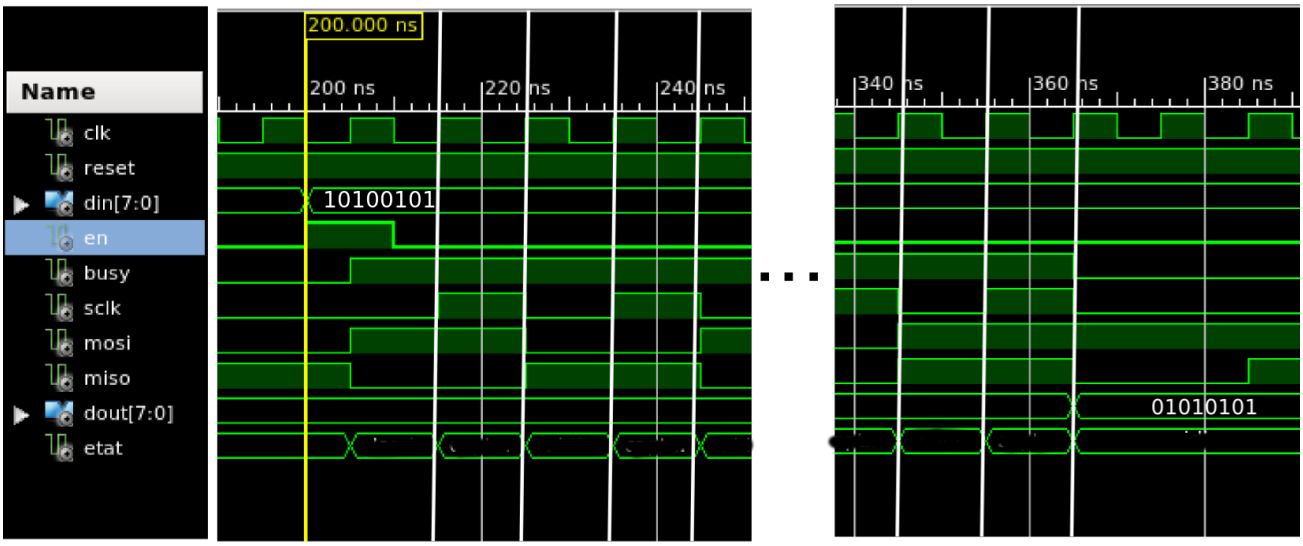

Figure 3 – émission de "10100101" et réception de "01010101" (début et fin)

Travail à réaliser

Développez le composant `er_1octet` et testez-le avec le simulateur (on ne se contentera pas de l'échange d'un seul octet !).
Dans le test, il est important d'écrire

- un process qui gèrera la commande et l'octet à envoyer (`en` et `din`) ;
  le principe de ce process est de positionner `din`, de passer l'ordre d'échange `en` puis d'attendre que l'échange se termine avant de recommencer.
  L'attente peut être une attente dépendant du temps de l'échange (qu'on sait calculer en fonction de la période de l'horloge) ou une attente liée à l'évolution du signal `busy`.
- un process synchrone qui génère `miso` (le plus simple est de changer la valeur de `miso` à chaque période) ; on se débrouillera pour qu'au cours des différents échanges, les signaux `dout` n'aient pas la même valeur.

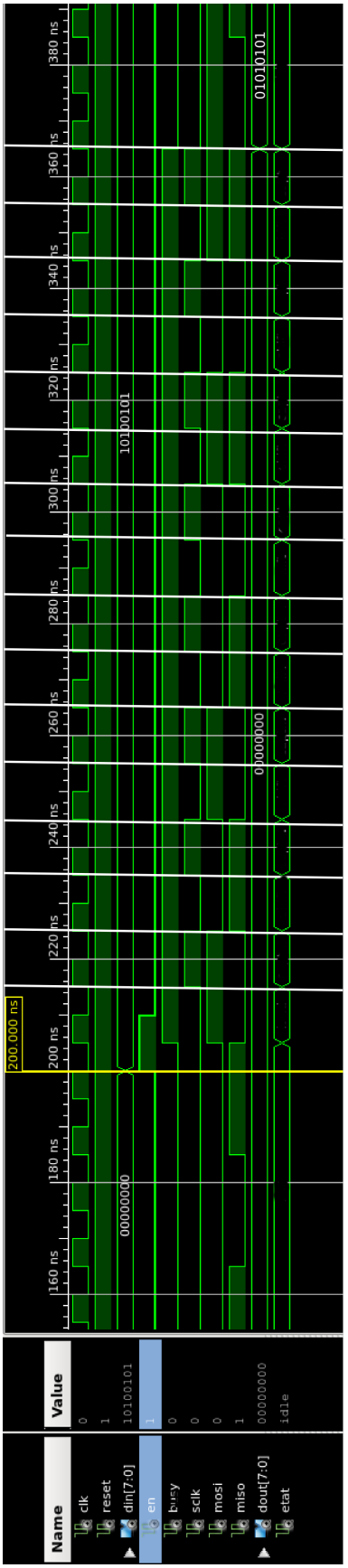

Figure 4 – émission de "10100101" et réception de "01010101"

3 Module Sum

Il s'agit de développer et de tester un nouveau composant Master qui, couplé avec un esclave (fourni), permet de faire la somme de deux octets.

3.1 Principe

Lors d'une transmission, le module Maître `MasterSum` envoie les 2 octets à sommer au module `SlaveSum` pendant que celui-ci lui renvoie la somme et la retenue, résultats de la transmission précédente.

3.2 MasterSum

Interface

L'interface de ce composant est la suivante :

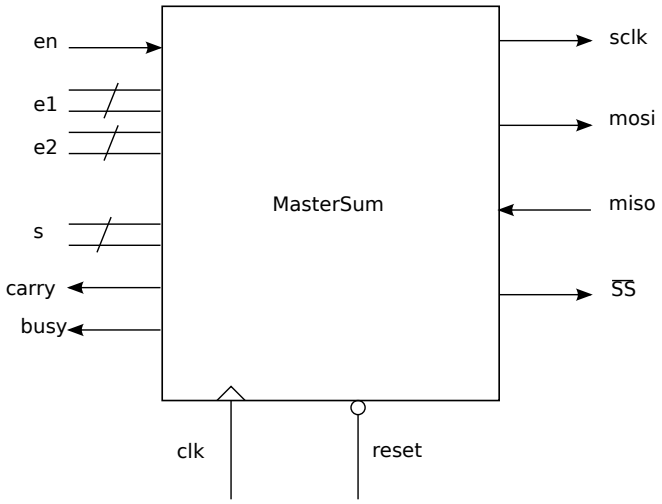

FIGURE 5 – Interface du MasterSum

Les signaux de ce composant se décomposent en trois groupes :

1. `clk` et `reset` sont l'horloge et la remise à zéro de ce composant synchrone,
2. les signaux connectés à l'esclave : `sclk`, `miso` et `mosi` qui ont le même rôle que pour le composant `er_1octet` et `ss` pour compléter les signaux du protocole SPI.
3. les signaux connectés au composant (ou process) qui donne des ordres au `MasterSum`
   — `en` (`enable`) indique qu'un ordre de transmission est donné,
   — `e1` et `e2` sont les deux octets à sommer,
   — `s` est l'octet somme de la transmission précédente,
   — `carry` est la retenue de la transmission précédente,
   — `busy` indique que le composant `MasterSum` est occupé.

Fonctionnement

Lorsqu'un ordre est passé au composant `MasterSum`, il initialise la transmission en mettant le signal `ss` du protocole SPI à '0'.

Un délai de 5 cycles de l'horloge est nécessaire pour attendre que l'esclave soit prêt (il aura positionné à ce moment son premier bit sur le signal `miso` du protocole SPI).

Le premier octet (opérande 1 de la somme) est envoyé du maître vers l'esclave pendant que l'octet somme de l'opération précédente est envoyé de l'esclave vers le maître.

Un délai de 2 cycles est nécessaire avant l'échange du second octet.

Le second octet (opérande 2 de la somme) est envoyé du maître vers l'esclave pendant que la retenue de l'opération précédente est envoyée de l'esclave vers le maître (bit de poids fort de l'octet, les autres bits ne sont pas significatifs).

Le signal `ss` repasse à '1' une fois la transmission terminée.

Le composant `MasterSum` a, cela va de soi, comme sous-composant, le composant `er_1octet` et son comportement peut être décrit sous forme d'un automate à états.

3.3 SlaveSum

Ce composant vous est fourni.

Son comportement se décompose en 2 parties implémentées en 2 `process` :

1. `process` de capture : à chaque front montant de `sclk`, le bit reçu sur `mosi` est rangé dans `v1` pour la première opérande puis `v2` pour la seconde. Dès que les deux octets sont reçus leur somme (`sum`) et la retenue (`retenue`) sont calculées
2. `process` d'émission : quand la transmission est initiée (`ss` passant à '0'), le bit de poids fort de `sum` est positionné sur `miso`, puis à chaque front descendant de `sclk`, un nouveau bit est positionné. Le second octet envoyé contient la retenue en poids fort et '0' pour les autres bits (choix arbitraire pour les 7 bits non significatifs).

Vous n'avez aucune raison de modifier ce composant.

Travail à réaliser

Développez le composant `MasterSum` et testez un circuit complet (Master + Slave) avec le simulateur (on ne se contentera pas d'une seule somme !).

4 Module Joystick PmodJSTK™

Le module PMODJSTK communique avec la carte avec le protocle SPI.

FIGURE 6 – Le joystick

Il se branche sur les ports PMOD :

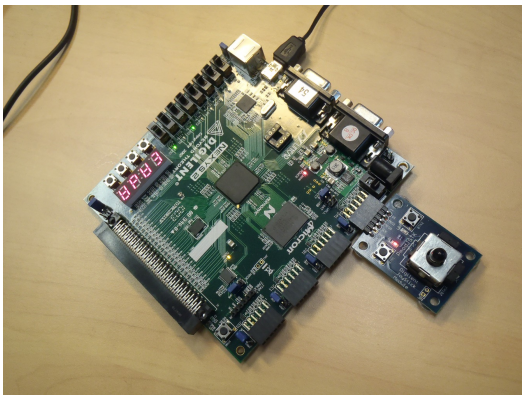

FIGURE 7 – Connexion du joystick

Le fichier `PmodJSTK_rm_RevC.pdf` est le manuel de référence de ce composant.

Travail à réaliser

En vous appuyant sur le composant `er_1octet` et sur le manuel de référence, développez le composant `MasterJoystick` qui permettra de faire communiquer la carte NEXYS2 avec le composant PMODJSTK.

5 Documents à rendre

Vous serez validé au fur et à mesure du développement des composants demandés (`er_1octet` et son test, `MasterSum` et son test, `MasterJoystick` et son circuit d'implantation sur la carte `Nexys4`).

Nous vous demandons de rendre :

1. les schémas des vues structurelles du `MasterSum`, `MasterJoystick` et du circuit implanté sur la carte `Nexys4` (utilisation de logiciels de dessins et format `pdf` imposés, pas de dessins à la main),
2. tous les codes développés en respectant la dénomination des composants / fichiers :
   (a) `er_1octet.vhd`
   (b) `test_er_1octet.vhd`
   (c) `MasterSum.vhd`
   (d) `TestSum.vhd`
   (e) `MasterJoystick.vhd`
   (f) `Nexys4Joystick.vhd`
   (g) `Nexys4Joystick.ucf`