Communication Carte Nexys 4 / Joystick PmodJSTK™ via protocole SPI

[Mini-Projet Individuel]

1 Présentation du protocole SPI (adapté de Wikipedia)

Une liaison SPI (Serial Peripheral Interface) permet à un Maître et un ou plusieurs Esclave(s) d'échanger de manière bi-directionnelle un ensemble d'octets. Chaque octet est envoyé bit à bit (série).

Elle est réalisée selon le schéma suivant :

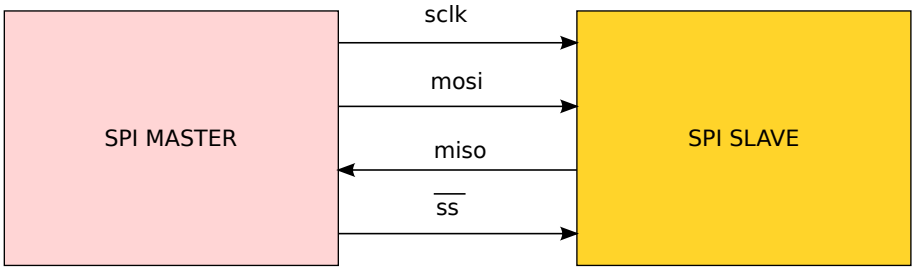

Figure 1 – Communication Maître-Esclave

Le bus SPI utilise les 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 octets (trame), 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 l'échange du premier octet ainsi qu'un temps d'attente entre l'échange de deux octets.

Pour échanger un octet, le Maître génère l'horloge `sclk` (seulement pour la durée de l'échange). La fréquence de `sclk` est réglée selon des caractéristiques propres aux périphériques. `sclk` est au repos à '1'.

À chaque top de `sclk` (front montant et descendant), le Maître et l'Esclave s'envoient 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 configurable (polarité et phase de `sclk`) ; il est le plus souvent utilisé en mode 0, ce qui signifie que :

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

Dans un premier temps, on se propose de développer un Maître SPI qui échange plusieurs octets avec un Esclave fourni. Ce développement est divisé en deux parties :

— un sous-composant du Maître (`er_1octet`) permettant l'échange d'un octet entre un Maître et un esclave. Sa spécification est donnée dans la section 2.
— un Maître SPI échangeant plusieurs octets (une trame) avec l'esclave fourni ; il utilise le composant `er_1octet`. Sa spécification est donnée dans la section 3.

Dans un second temps, on se propose de développer un autre Maître SPI qui permet d'échanger une trame avec le Joystick PmodJSTK. Celui-ci utilise également le composant `er_1octet`. Sa spécification est donnée dans la section 4.

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 lors de ce projet.

Il présente l'interface suivante :

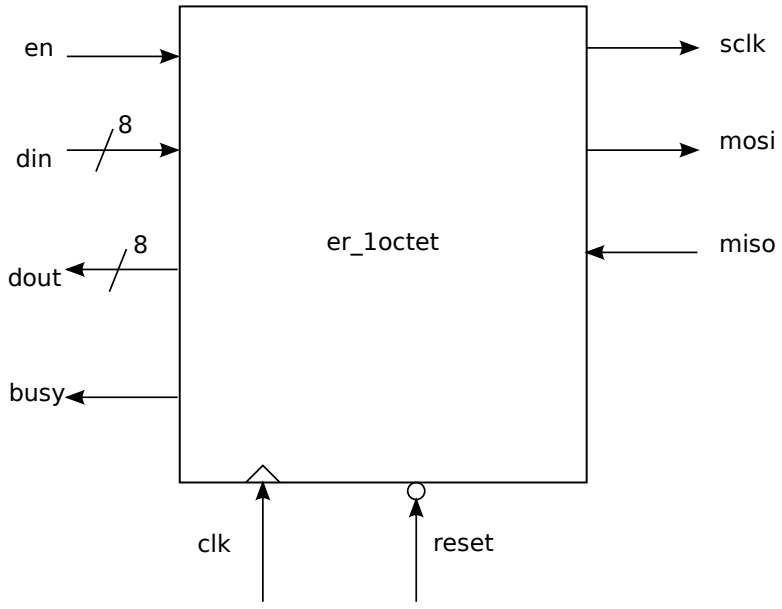

Figure 2 – Interface du composant d'échange d'1 octet

On retrouve les 3 signaux `sclk`, `mosi` et `miso` du protocole SPI, le signal `ss` n'apparaissant pas à ce niveau (ce sera au niveau d'un maître lors de la transmission d'une trame).

Les signaux supplémentaires sont :

- `clk` et `reset`, l'horloge et la remise à zéro de ce composant synchrone,
- la commande `en` (`enable`) qui indique qu'un ordre d'échange (émission/réception) d'1 octet est passé (actif à '1'),
- `din`, l'octet à émettre, sa valeur est fournie au moment où le composant est au repos et `enable` passe à '1' (elle n'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 7 et 8 en annexe et son comportement peut être décrit sous forme d'un automate à états.

Travail à réaliser

Développez le composant `er_1octet` et validez-le avec le testbench fourni.
Des assertions sont présentes dans le test pour les signaux en sortie du composant `er_1octet`, ce qui facilite la validation.

3 Module OpL

Il s'agit de développer et de tester un Maître SPI qui, couplé avec un esclave (**fourni**), permet d'effectuer des opérations logiques entre deux octets.

3.1 Principe

La trame échangée entre le module Maître `MasterOpL` et l'esclave `SlaveOpL` est constituée de 3 octets.

Pendant que le Maître envoie 2 octets "opérandes" et un octet non utilisé, l'esclave lui renvoie les résultats du "et", du "ou" et du "xor" entre les deux opérandes de la transmission précédente.

La trame de cet exemple est donc constituée de 3 octets.

Les 3 octets reçus par le Maître lors de la première transmission ne correspondent bien entendu pas aux résultats d'un calcul mais sont des valeurs "en dur" positionnées dans l'esclave (repérez-les dans le code de l'esclave fourni).

Enfin ce fonctionnement nécessite de toujours faire une transmission de plus pour obtenir les derniers résultats qui nous intéressent.

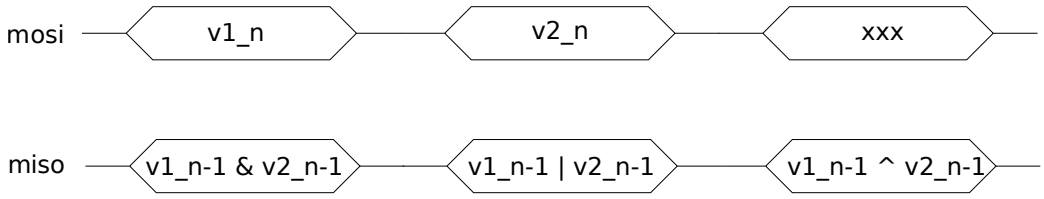

Figure 3 – trame n : décalage entre les données et les résultats

3.2 MasterOpL

Interface

L'interface de ce composant est donnée dans la figure 4 ; le fichier VHDL de ce composant avec l'interface et une architecture vide vous est aussi fourni.

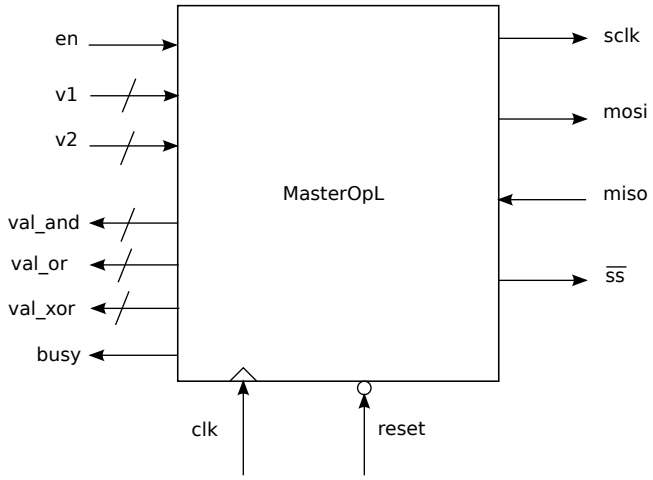

Figure 4 – Interface du MasterOpL

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 `MasterOpL`
   — `en` (`enable`) indique qu'un ordre de transmission est donné,
   — `v1` et `v2`, en entrée, sont les deux octets opérandes,
   — `val_and`, `val_or` et `val_xor`, en sortie, sont les octets des opérations logiques entre les opérandes de la transmission précédente,
   — `busy` indique que le composant `MasterOpL` est occupé.

Fonctionnement

Lorsqu'un ordre est passé au composant `MasterOpL`, il initialise la transmission en mettant le signal `ss` du protocole SPI à '0' et indique qu'il est "busy".

Une attente de 10 cycles de l'horloge `clk` est nécessaire pour que l'esclave soit prêt ; quand l'échange commence, l'esclave a positionné son premier bit sur le signal `miso` du protocole SPI.

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

Une attente de 3 cycles de l'horloge `clk` est nécessaire avant l'échange du deuxième octet.

Le deuxième octet (opérande 2) est envoyé du maître vers l'esclave pendant que le "ou" de l'opération précédente est envoyé de l'esclave vers le maître.

Une attente de 3 cycles de l'horloge `clk` est nécessaire avant l'échange du troisième octet.

Le troisième octet (non utilisé) est envoyé du maître vers l'esclave pendant que le "xor" de l'opération précédente est envoyé de l'esclave vers le maître.

Le signal `ss` repasse à '1' une fois la transmission terminée et le composant n'est plus "busy".

Le composant `MasterOpL` a, cela va de soi, comme sous-composant, le composant `er_1octet` que vous avez développé précédemment ; son comportement peut être décrit sous forme d'un automate à états.

3.3 SlaveOpL

Ce composant vous est fourni. Vous n'avez aucune raison de modifier ce composant.

Son comportement se décompose en 2 parties implémentées en 2 `process` synchrones sur l'horloge `sclk` et avec un reset `ss = '1'`.

En début de transmission le Maître positionne `ss` à '0' ce qui a pour effet de faire sortir l'Esclave de son attente et ses process vont pouvoir réagir aux fronts de `sclk`.

1. `process` de capture : à chaque front montant de `sclk`, le bit reçu sur `mosi` est rangé dans `v1` pour la première opérande, `v2` pour la seconde ; `v3` pour l'octet non significatif n'est pas stocké. Les résultats des opérations logiques entre `v1` et `v2` sont alors calculées.
2. `process` d'émission : à chaque front descendant de `sclk`, un bit est positionné sur `miso`, en commençant par l'octet `val_and`, puis `val_or`, et enfin `val_xor`.

Quand la transmission est terminée, le Maître fait passer `ss` à '1' ce qui réinitialise les deux process de ce composant.

Travail à réaliser

Développez le composant `MasterOpL`, générez un testbench `TestOpL` et complétez-le pour tester un circuit complet (Master + Slave) avec le simulateur (on ne se contentera pas d'un seul calcul !).

Pour travailler à distance, il vous est fortement conseillé de travailler avec le script d'exécution des outils de simulation.

4 Module Joystick PmodJSTK™

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

Figure 5 – Le joystick

Il se branche sur les ports Pmod :

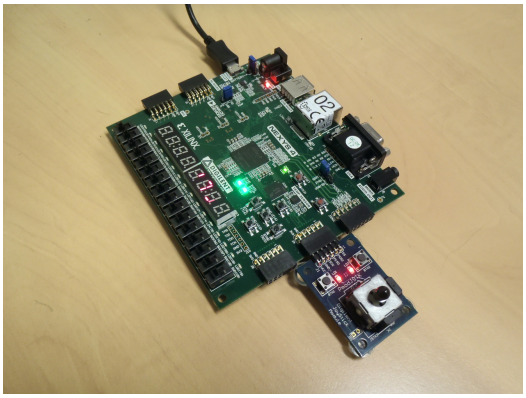

Figure 6 – Connexion du joystick

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`, sur le manuel de référence et sur votre expertise acquise lors du développement du composant `MasterOpL`, développez le composant `MasterJoystick` qui permettra de faire communiquer la carte Nexys4 avec le composant PmodJSTK.

Pour vérifier le fonctionnement correct de cette communication, vous pouvez, par exemple :
- afficher la position du joystick sur les 7 segments,
- commander les leds du joystick avec des switchs ou des boutons,
- allumer des leds quand vous appuyez sur les boutons du joystick.

Nous vous fournissons des versions complètes et correctes de composants donnés ou développés en TP (`dec7seg`, `diviseurClk`, `All7Segments`). Il est inutile de les modifier.

5 Déroulé / Instructions / Fichiers à rendre

Tout est indiqué dans la section Moodle consacrée à ce Mini-Projet.

annexes

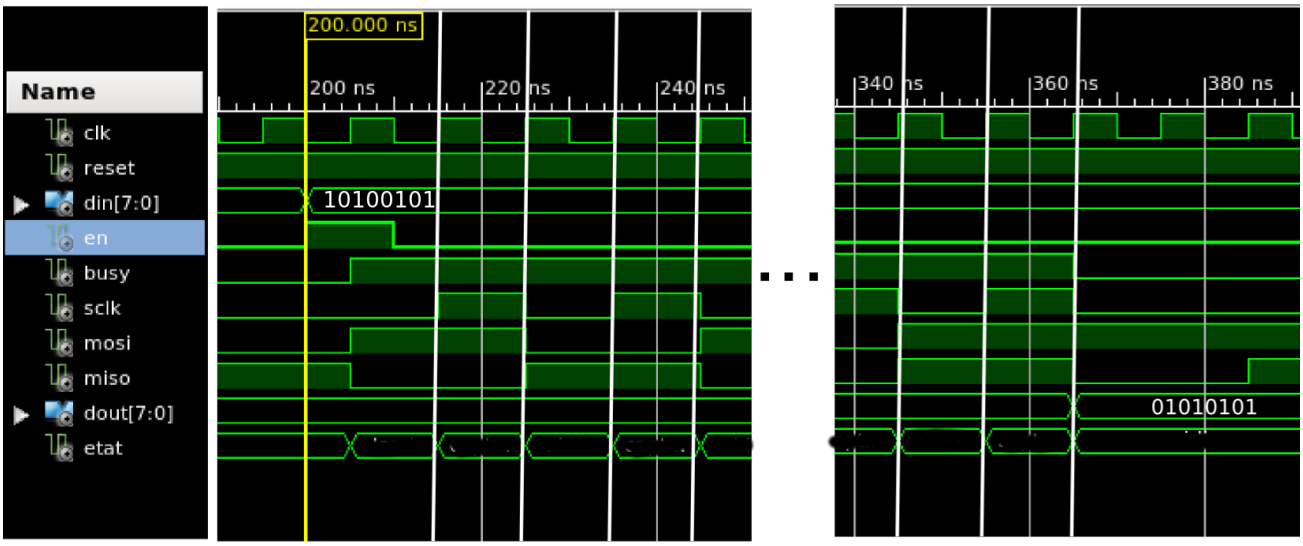

Figure 7 – émission de "10100101" et réception de "01010101" (début et fin)

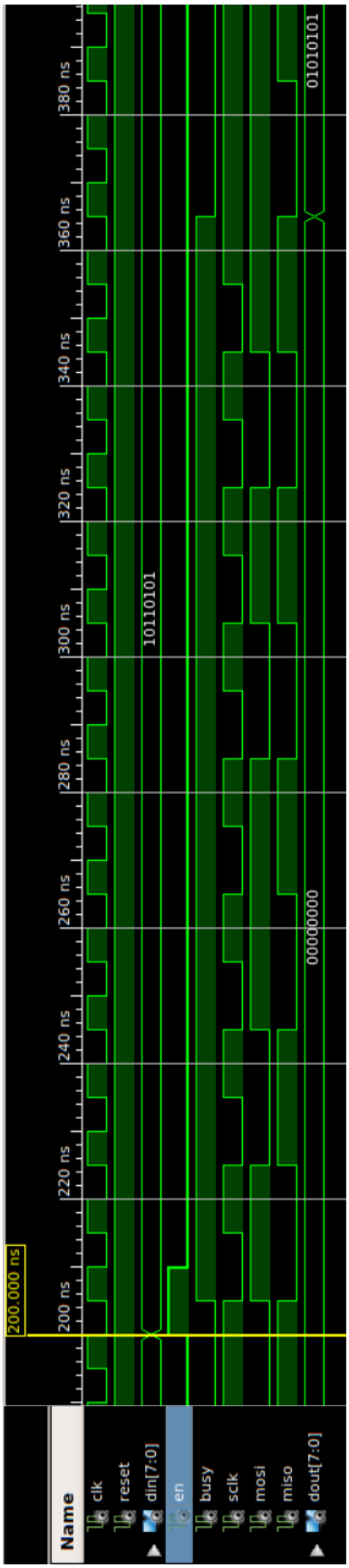

Figure 8 – émission de ”10100101” et réception de ”01010101”