

# Systèmes interconnectés – SINT

## Sujet de TP 5 : CAN et FPGA (version alternative)

Centrale Nantes

P.-E. Hladik, pehladik@ec-nantes.fr

—

Version bêta (27 novembre 2023)

## 1 Objectifs pédagogiques

Cette version de TP est une version alternative du sujet initial. Elle propose de développer une partie d'un contrôleur CAN et le succès n'est pas assuré...

- spécifier une machine séquentielle à l'aide d'un automate
- simuler et tester une implémentation en VHDL
- intégrer plusieurs entités dans un projet

## 2 Rendu

À la fin du TP déposez sur Hippocampus une archive avec un compte rendu en pdf (voir les éléments demandés dans le texte).

## 3 Prise en main

Cette première partie a pour but de reprendre en main Vivado et le VHDL.

### 3.1 Création d'un projet

Commencez par créer un nouveau projet sous Vivado (voir le [document en ligne](#) ou la vidéo). Pour cette première étape vous aurez besoin des fichiers `clock_div.vhd` en tant que *Design Sources*, de `testbench_div.vhd` comme *Simulation Sources* et `basys3.xdc` comme contraintes. Ces fichiers sont dans l'archive sur Hippocampus. Vous pouvez importer les fichiers à la création du projet ou bien après.

Pour la carte Basys3, il faut sélectionner le composant « xc7a35tcpg236-1 » que l'on peut trouver avec le filtre :

- Family : Artix-7
- Package : cpg236
- Speed : -1

### 3.2 Simulation

Vérifiez que vous avez bien importé `testbench_div.vhd` dans *Simulation Sources* et qu'il est bien au *top level* (sinon clic droit et *Set as Top*). Ce fichier simule le comportement d'une entité de type `clock_div`.

Lancez la simulation comportementale (voir vidéo) et observez. Pour l'instant seul le signal `reset` est simulé. Ajoutez le code nécessaire pour simuler l'horloge à une période de 10 ns et lancez la simulation.

**Rapport :** mettez l'extrait de code du testbench ainsi qu'une capture de la simulation.

Quel est le rapport cyclique si le diviseur est impair (3 par exemple) ? Observez en simulation le comportement et expliquez les raisons en allant lire le code de `clock_div.vhd`.

**Rapport :** mettez une capture de la simulation avec un diviseur de 3 et expliquez le comportement observé à partir du code.

### 3.3 Synthèse

Assurez-vous que vous avez bien importé `display_sint.vhd` comme *Design Sources* et `basys3.xdc` comme contraintes.

Créez un nouveau fichier de conception et mettez le comme *Top Level*. Implémentez un système qui fait clignoter une led toutes les 500 ms. Pour cela utilisez le diviseur d'horloge sachant que l'horloge du système a une période de 10 ns. Les signaux physiques dont vous aurez besoin sont :

- `led0` en sortie,
- `reset` en entrée,
- `clock_100Mhz` en entrée.

Ils doivent être déclarés comme ports de votre entité comme dans l'exemple ci-dessous :

```
entity blink is
  Port (
    reset          : in std_logic;
    clock_100Mhz   : in std_logic;
    led0           : out std_logic
  );
end blink;
```

et seront ensuite liés aux signaux physiques dans le fichiers de contraintes (vérifiez que les lignes nécessaires dans le fichiers de contraintes `basys.xdc` sont bien décommentées).

Ecrivez le code, faites la synthèse et exécutez le résultat sur la carte.

**Rapport :** mettez le code produit.

## 4 Affichage

Importez dans votre projet le fichier `display_sint.vhd`.

L'entité `display_basys3` permet d'afficher une valeur en hexadécimale sur l'afficheur 7 segments de la carte. Les ports sont :

- `clock_100Mhz` (entrée) : l'horloge physique de la carte,
- `reset` (entrée) : le signal physique de reset de la carte,
- `anode_activate` (entrée) : les signaux physiques des anodes,
- `led_out` (entrée) : les signaux physiques des leds de l'afficheur,
- `displayed_number` (entrée) : la valeur à afficher.

Créez un nouveau fichier et implantez un code qui met à jour toutes les secondes l'afficheur en incrémentant une valeur (vous aurez une horloge hexadécimale). Servez-vous de ce que vous avez fait avant pour le compteur en le copiant dans le nouveau fichier (ou utilisez ce que vous avez fait avant).

Si vous passez par un entier pour coder la valeur à afficher, vous pouvez utiliser

```
n <= std_logic_vector( to_unsigned(val, n'length))
```

pour convertir l'entier non signé `val` et l'affecter à un signal `n` de type `std_logic_vector` de taille `n'length`.

Pour les contraintes, voir les commentaires dans le fichier `display_sint.vhd` et décommentez les lignes dans le fichier de contraintes.

**Rapport :** mettez le code produit.

## 5 Contrôleur CAN

L'objectif est de concevoir un contrôleur CAN sur le FPGA pour envoyer un message de type donnée avec un octet. L'identifiant sera fixé ainsi que la fréquence d'envoi (125000 Hz). La trame sera donc fixe pour la partie en-tête mais changera pour la partie donnée. Il faudra donc recalculer le CRC.

Pour réaliser cela il faut mettre au point plusieurs dispositifs :

1. émettre la trame
2. gérer le bit-stuffing
3. calculer un CRC
4. détecter que la ligne est disponible pour émettre
5. gérer les priorités
6. gérer les erreurs (non acquitement)

Pour les derniers points il faut donc aussi écouter la ligne Rx.

Pour chacun de ces composants, décrivez leur interface ainsi que l'automate de leur comportement. Tester les en simulation puis en réelle.

Pour tester en réel il vous faudra un *transceiver*, une breadboard et quelques fils. Vous allez utiliser un MCP2562. La documentation est fournie avec le sujet. La communication entre le CAN et le transceiver se fera sur une broche d'un PMOD de la Basys 3 (à vous de la définir). Le composant MCP2562 ayant besoin d'une alimentation en 5V (et la sortie sur les PMOD étant à

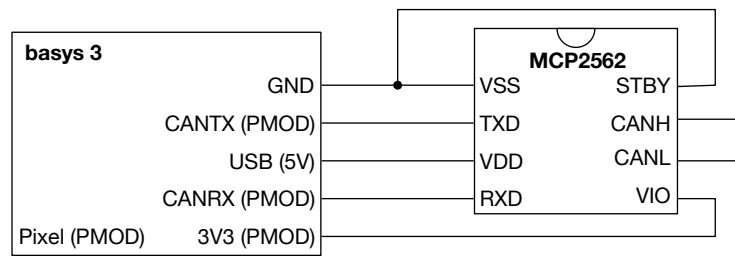


FIG. 1 – Schéma de connexion entre le basys 3 et le MCP2562

3.3V) vous pouvez repiquer sur la breadboard l'USB qui sort de la carte en utilisant l'adaptateur micro USB qui vous sera fourni (voir figure 1).

Suivez les conseils suivants pour réaliser l'émetteur CAN :

**Test d'envoi d'une trame statique.** Pour faire les choses progressivement commencer par émettre une trame statique vers un récepteur CAN (Feather ou Teensys) et vérifier que la trame est bien reçue. Vous pouvez aussi utiliser l'analyseur logique pour vérifier que votre trame est correcte.

**Bit-stuffing.** Ajouter ensuite la gestion du bit-stuffing en testant toujours par une émission vers un récepteur ou l'analyseur logique.

**Calcul du CRC.** Passer ensuite au calcul du CRC avec une trame dont la partie donnée est variable et émettez le résultat. Utiliser l'analyseur logique pour vérifier que votre CRC est bon.

**Détection de l'état de la ligne.** Réaliser un composant pour détecter que la ligne est libre. Pour cela, connecter votre dispositif à un émetteur CAN qui envoie périodiquement des messages. Sortez sur un port PMOD un signal indiquant si la ligne est libre ou non et vérifier avec l'analyseur que le comportement est correct.

**Composition émission et état de la ligne.** Composez vos modules d'émission avec le module de détection d'état de la ligne. Tester en interconnectant votre système avec un autre émetteur qui envoie périodiquement des messages et vérifier que votre émetteur n'émet pas quand la ligne n'est pas disponible.

**Gestion des priorités.** Réaliser un composant qui vérifie que vous l'émetteur à la priorité pendant son émission. Pour tester votre dispositif, composer un système sur el FPGA avec deux émetteurs synchrones et sortez une ligne pour indiquer qui est prioritaire. Vérifiez à l'aide de l'analyseur logique que tout se passe bien.

**Gérer les erreurs.** Si vous en êtes là c'est qu'a priori vous êtes aptes à imaginer comment ajouter et tester ce composant...

**Rapport :** pour chacun des composants mettez dans votre rapport l'automate ainsi que le code produit et les résultats de tests réalisés.