6GEI228 – Systèmes Digitaux

Laboratoire #6

Introduction au VHDL

Hiver 2018

1. Objectifs

- Apprendre les bases du VHDL
- Apprendre à implémenter de l'arithmétique simple

2. Méthodologie

Au cours de ce laboratoire, vous aurez à utiliser le langage de description matériel (HDL, ou hardware description language en anglais) appelé VHDL (la lettre V signifie VHSIC qui signifie Very High Speed Integrated Circuits). C'est une approche qui est privilégiée dans le domaine de la conception de circuits parce qu'elle nous permet de concevoir plus rapidement et plus efficacement qu'avec des dessins de portes logique.

Une fois que la description des circuits aura été faite en VHDL, le reste des étapes est semblable à ce qui s'est fait dans les autres laboratoires : compilation, simulation (au besoin), assignation des PINS, recompilation et programmation.

3. Préparation, théorie et tutoriel

Le langage VHDL est un langage qui a été conçu pour la simulation, il y a plus de 20 ans. Elle nous permettait de spécifier n'importe quel genre de comportement et un logiciel existait pour vérifier, sur ordinateur, le comportement des circuits sans avoir à les construire physiquement. Durant les années 1990, il y a eu une technologie qui s'est développée qui permettait de transformer une description VHDL en une série de portes logiques : la synthèse logique. C'est le développement de la synthèse, en partie, qui a permis à plusieurs entreprises de fabriquer des systèmes très rapidement.

Le problème, cependant, est que la synthèse ne supporte pas toutes les commandes VHDL. Les outils ne sont capables d'identifier que certaines structures pour les

transformer en portes logiques. Il faut donc faire attention d'écrire nos programmes selon certaines règles. De plus, ceci est encore plus important puisqu'un circuit mal décrit peut donner des résultats de simulation qui sont différentes de ce qui est synthétisé. Nous n'avons donc pas la possibilité de programmer de n'importe quelle manière.

Partie 1. Additionneur de 4 bits

Avant de parler du langage VHDL, commencez par ouvrir le logiciel Quartus Prime. Créez un nouveau projet appelé *add4*. Maintenant, créez un nouveau fichier de type VHDL.

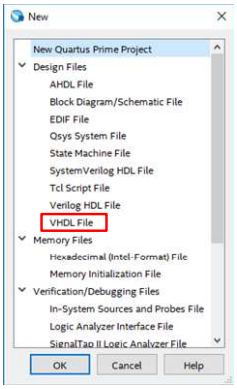

Figure 1. Ouverture d'un fichier VHDL.

Un code VHDL contiendra généralement 3 parties :

- Les librairies
- L'entité
- L'architecture

Les librairies spécifient quelles fonctions sont nécessaires au bon fonctionnement de notre circuit. En général, on pourrait tout simplement **copier les 3 lignes suivantes** : (copiez-le et collez-le dans votre code VHDL)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Les majuscules n'ont pas d'importance. Habituellement, nous utilisons des majuscules pour des termes qui sont réservés au VHDL et les minuscules pour des noms que nous donnons.

L'entité c'est ce qui décrit le module quand on le voit de l'extérieur (un bloc avec entrées et sorties). Dans l'entité, il faut définir le nom et la taille des signaux qui vont entrer et

sortir d'un système. Un exemple d'une entité (ENTITY) vous est donné copiez-le et collez-le dans votre code:

```
ENTITY add4  IS
        PORT (
                entree1: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                entree2: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                somme: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
END add4;
```

L'entité ne décrit pas ce qu'il y a à l'intérieur mais plutôt comment le circuit est vu de l'extérieur. En regardant la description, on voit que le nom du module est *add4* et qu'il y a 2 entrées (*entree1* et *entree2*) et 1 sortie (*somme*). Le terme STD_LOGIC_VECTOR veut simplement dire que c'est un vecteur de plusieurs bits. Dans ce cas-ci, les bits vont de 3 à 0, donc il y a 4 bits. Pour avoir un seul bit, il faudrait utiliser STD_LOGIC à la place de STD_LOGIC_VECTOR.

La description précédente de l'entité se traduirait en un diagramme comme le suivant :

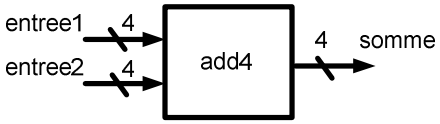

Figure 2. Exemple d'une entité

Finalement, la troisième partie d'un code VHDL typique c'est l'architecture. L'architecture c'est l'endroit où l'on décrit l'intérieur du circuit. Un exemple de code pour l'architecture se trouve ci-dessous (copiez-le et collez-le dans votre code) :

```
ARCHITECTURE rtl OF add4 IS

        BEGIN
                somme <= entree1 + entree2;
        END;
```

On voit juste après le mot *architecture*, il y a le mot *rtl* en minuscule : *rtl* est le nom que je donne à l'architecture de ma conception (design) *add4*. Le terme *rtl* veut dire « transfert de registre » (ou register transfer level, RTL) qui est une expression qui est souvent utilisé pour décrire du VHDL synthétisable. Un design peut parfois avoir plusieurs architectures possibles et donc, le langage est fait pour accommoder ce genre de choses. Pour nos besoins actuels, ce n'est pas très utile mais on doit quand même donner un nom à notre architecture.

Après le mot BEGIN se trouve le cœur du programme. Dans ce cas-ci, le code dit qu'il y a *entree1* et *entree2* qui sont additionnés ensemble et qui sont mis dans *somme*.

Pour vérifier que ça fonctionne, compilez-le. En même temps, créez aussi un bloc et insérez-le dans un schéma quelconque (File→ Create/Update→Create Symbol Files from Current File). L'idée ici est de passer rapidement au travers du cycle de design pour se convaincre que c'est un processus simple.

En plus de l'addition, le langage VHDL offre aussi d'autres opérations mathématiques possibles tels que la soustraction (entree1 – entree2), la multiplication (entree1 * entree2) et les fonctions logiques qui peuvent être spécifiés avec AND, OR, NOT, et XOR.

Pour la multiplication, il faut se rappeler que le résultat est typiquement de plus grosse taille que l'entrée. De façon plus précise, les nombres de bits des entrées vont s'additionner. Par exemple, en multipliant un nombre de 4 bits par un nombre de 4 bits, on obtiendrait un nombre de 4+4=8 bits.

Partie 2. Calcul d'une distance parcourue

Dans cette partie du laboratoire, il sera question d'implanter une fonction qui calcule la position d'une personne qui marche à une vitesse *vitesse*, qui commence à marcher à la position *depart* et qui marche pendant *temps* secondes. L'équation de la position est la suivante :

$$position = vitesse \cdot temps + depart$$

Commencez par vous créer un nouveau projet appelé distance et ajoutez un nouveau fichier VHDL. Copiez les 3 lignes pour déclarer les librairies.

Si *vitesse* et *temps* étaient des vecteurs de 4 bits chaque, combien de bits devrait-on avoir pour *position*? Il n'y a pas qu'une seule bonne réponse, mais soyez au moins conscient qu'il faut décider de ces tailles.

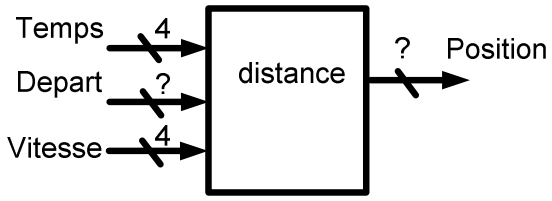

Figure 3. Entité distance

Avec cette information, écrivez le code pour l'entité.

L'allure générale du système devrait ressembler à ceci :

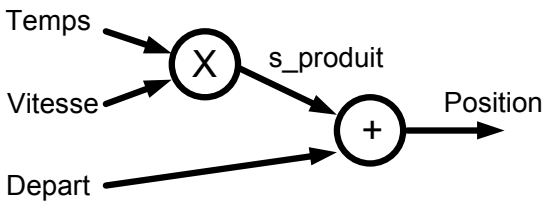

Figure 4. Diagramme de la fonction.

Nous allons vouloir multiplier *temps* et *vitesse* ensemble pour donner une variable intermédiaire *s_produit*. Par la suite, nous allons additionner *s_produit* à *depart* pour nous donner *position*.

Il faut commencer par définir un nœud intermédiaire *s_produit* dans notre code VHDL. Pour ce faire, il faut ajouter une ligne ENTRE la ligne ARCHITECTURE et la ligne BEGIN. Cette ligne devrait dire que *s_produit* est un SIGNAL d'une TELLE grosseur. Par exemple, SI LE VECTEUR s_produit était de taille 6 (**ce ne l'est pas!**), on écrirait ceci :

```
ARCHITECTURE rtl OF distance IS

SIGNAL s_produit : STD_LOGIC_VECTOR(5 DOWNTO 0);

BEGIN
```

La suite du code s'écrirait de la façon suivante (c'est la suite du code après le BEGIN) :

```
s_produit <= vitesse * temps ;
position <= s_produit + depart ;

END ;
```

Votre code est maintenant complet. Compilez-le pour vous assurer qu'il n'y a pas de problèmes. Par la suite, simulez-le en mettant des valeurs de *temps*, de *vitesse* et de *depart* différents. Vérifiez que ça donne la bonne réponse. Par exemple, avec *vitesse* de 5, *depart* de 3 et *temps* de 2, on devrait obtenir *position* de 13. Assurez-vous que la sortie donne cette valeur.

Il est important de savoir que TOUTES LES OPÉRATIONS SE PRODUISENT EN PARALLÈLE ET SE PRODUISENT EN TOUT TEMPS. Ceci est complètement différent de la programmation en langage séquentielle conventionnelle (comme le C++).

Dans le code précédent, il aurait été possible d'écrire les lignes de façon inversée et ça ne changerait absolument rien. C'est-à-dire que le code suivant donnerait exactement les mêmes résultats :

```
position <= s_produit + depart ;
s_produit <= vitesse * temps ;
```

Pour illustrer le principe de façon plus poussée, prenons un autre exemple. Prenons le code suivant (qui n'a aucun rapport avec le code pour calculer la position, en passant) :

```
led1<= NOT switch1;
led1<= NOT switch2;
```

Dans un langage séquentiel, le système assignerait *NOT switch1* dans *led1* en premier et par la suite, il mettrait *NOT switch2*. En VHDL, ce n'est pas le cas. Le logiciel fera en sorte que les deux opérations se produisent en même temps tel qu'illustré à la figure suivante.

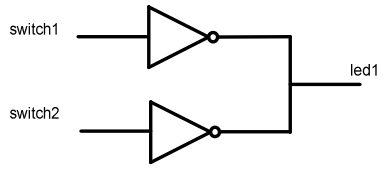

Figure 5. Circuit NOT.

Puisque ceci n'est pas une pratique acceptée, le logiciel vous donnera un message d'erreur du style :

Error (10028): Can't resolve multiple constant drivers for net "led1" at nonet.vhd(19)

4. Travail demandé

Dans cette partie du laboratoire, vous aurez à concevoir un système qui calcule la position avec un paramètre de plus : l'accélération. Vous aurez donc à implanter l'équation suivante :

$$position = \frac{1}{2} acceleration \cdot temps^2 + vitesse_0 \cdot temps + depart$$

Pour cette partie du laboratoire, vous aurez en entrée l'accélération, la vitesse de départ (*vitesse*$_0$) et la position de départ (*depart*). Pour avoir la valeur du temps, vous aurez à inclure un module fourni qui aura besoin d'une horloge de 50MHz en entrée pour générer

une valeur à la sortie qui représente les secondes. C'est-à-dire que la sortie va augmenter de 1 à chaque seconde.

Pour se sauver de l'effort, utilisez des tailles de 2 bits pour l'accélération, la vitesse de départ et la position de départ.

Concevez le design et implémentez-le sur FPGA. Affichez la sortie *position* sur les DELs et PROUVEZ au chargé de laboratoire que ça fonctionne. C'est à vous de montrer explicitement que votre système est bon.