TP2 : Modélisation Structurelle VHDL

Objectifs

Le but de cette seconde séance de TP est d'utiliser le langage VHDL pour décrire, puis simuler une description structurelle du composant *addaccu*, présenté dans le premier TP, en utilisant une bibliothèque de cellules pré-caractérisées (en anglais *standard cells library*).

Le but final est d'aboutir à une description structurelle du composant addacu qui soit un schéma en portes logiques, utilisant une bibliothèque de cellules pré-caractérisées. Ceci sera fait dans le TP3. Dans ce TP2, nous allons franchir une étape intermédiaire, en décomposant le circuit *addaccu* en trois sous-blocs fonctionnels : le bloc **mux**, le bloc **adder**, et le bloc **accu**.

Un deuxième objectif de ce TP2 est d'introduire le langage de description de stimuli **genpat**.

A) Génération procédurale des stimuli

Dans le TP1, vous avez écrit "a la main" le fichier *stimuli.pat* décrivant les valeurs à appliquer sur les entrées du circuit. Cette méthode est assez fastidieuse, et elle est source d'erreurs. Pour faciliter la description des scénarios de simulation, vous pouvez utiliser le langage **genpat**, qui est un ensemble de fonctions écrites en langage C, et qui apporte au concepteur de circuit toute la puissance d'expression du langage C (boucles, expression conditionnelles, etc.) pour décrire les scénarios de simulation.

On rappelle que vous pouvez obtenir des informations détaillées sur n'importe quel outil de la chaîne de CAO *ALLIANCE* en tapant (par exemple) la commande :

```
>man genpat

>man AFFECT
```

Les noms des fonctions **genpat** sont en majuscules. La fonction la plus importante du langage **genpat** est la fonction AFFECT() qui permet d'assigner une nouvelle valeur à un signal particulier X à une certaine date T. Cette fonction permet donc de spécifier des *événements*. Chaque fonction du langage **genpat** possède son propre man :

```
>man AFFECT
```

Il faut donc écrire un fichier *new_stimuli.c* respectant la syntaxe du langage C, et c'est l'exécution de ce programme C qui générera le fichier *new_stimuli.pat* utilisable par **asimut**. On utilise pour la commande suivante pour générer le fichier *new_stimuli.pat* :

```
>genpat new_stimuli
```

Voici deux suggestions utiles pour écrire le fichier *new_stimuli.c* :

- Ecrire une fonction C indépendante pour le signal d'horloge, qui est très régulier (on conservera une période de 10 ns, avec un rapport cyclique de 50%).
- faites en sorte que la valeur stockée dans l'accumulateur possède une valeur bien définie, en sélectionnant l'entrée a du multiplexeur (au moyen de la commande sel) dans les tous premiers cycles.

Vérifiez que le fichier 'new_'stimuli.pat *généré correspond à ce que vous attendez en utilisant l'outil* ***xpat****.*

```
>xpat new_stimuli
```

Vous pouvez vérifier votre scénario en simulant son exécutionc sur le modèle comportemental du composant *addaccu* provenant du TP1 :

```
>asimut -b -zd addacu new_stimuli new_result
```

L'option -zd signifie que vous souhaitez que **asimut** effectue une simulation zéro-délay : même si la description comportementale contient des constructions AFTER, celles-ci ne seront pas prises en compte.

B) Description structurelle au niveau blocs

On va maintenant décrire le composant *addaccu* comme l'instanciation de trois blocs fonctionnels : le bloc **mux**, le bloc **adder**, et le bloc **accu**.

- Le bloc adder est un additionneur 4 bits (2 mots de 4 bits en entrée, un mot de 4 bits en sortie).
- Le bloc mux est un multiplexeur 4 bits qui sélectionne un mot de 4 bits parmi 2.
- Le bloc accu est un registre 4 bits constitué de ' bascules à échantillonnage sur front montant de CK.

Puisqu'il s'agit d'une description structurelle, le fichier VHDL comportera l'extension *.vst* (Vhdl STructurel)

Il faut donc utiliser la construction VHDL qui permet d'instancier un bloc.

C) Modélisation Comportementale des blocs

Pour chacun des trois blocs, il faut maintenant écrire un modèle VHDL comportemental. Il y a donc trois fichiers à écrire : adder.vbe, mux.vbe, et accu.vbe.

On pourra évidemment s'inspirer du style d'écriture VHDL utilisé dans le fichier addaccu.vbe.

D) Simulation logique

Compte-rendu