TP3 : Bibliothèque de cellules pré-caractérisées

A) Objectifs

Le principal objectif de ce TP est d'utiliser le langage VHDL pour définir et simuler un schéma complet du composant *addaccu*, en utilisant une bibliothèque de cellules pré-caractérisées.

Pour cela, nous allons décrire les trois blocs **adder4**, **mux4** et **accu4**, comme une interconnexion de portes de bases, fournies par la bibliothèque de cellules pré-caractérisées SXLIB.

Une cellule pré-caractérisée (en anglais "standard cell") est une fonction élémentaire pour laquelle on dispose des différentes "vues" permettant son utilisation par des outils CAO:

- vue *physique* : dessin des masques, permettant d'automatiser le placement et le routage.
- vue *logique* : schéma en transistors permettant la caractérisation (surface, consommation, temps de propagation).
- vue *comportementale* : description VHDL permettant la simulation logique des circuits utilisant cette bibliothèque.

Créez un répertoire de travail *tp3*, et recopiez dans ce répertoire les fichiers *addaccu.vst*, adder4.vst, accu4.vst, mux4.vst, et *stimuli.pat* du TP2.

B) La bibliothèque SXLIB

La bibliothèque de cellules utilisée dans ce TP est la bibliothèque SXLIB, développée par le laboratoire LIP6, pour la chaîne de CAO *ALLIANCE*. La particularité de cette bibliothèque est d'être "portable" : le dessin des masques de fabrication utilise une technique de *dessin symbolique*, qui permet d'utiliser cette bibliothèque de cellules pour n'importe quel procédé de fabrication CMOS possédant au moins trois niveaux d'interconnexion.

Evidemment les caractéristiques physiques (surface occupée, temps de propagation) dépendent du procédé de fabrication. Les cellules que vous utiliserez dans ce TP ont été caractérisées pour un procédé de fabrication CMOS 0.35 micron.

La liste des cellules disponibles dans la bibliothèque SXLIB peut être obtenue en consultant la page man :

```
>man sxlib
```

Comme vous pourrez le constater, il existe plusieurs cellules réalisant la même fonction logique. Les deux cellules na2_x1 et na2_x4 réalisent toutes les deux la fonction NAND à 2 entrées, et ne diffèrent entre elles que par leur puissance électrique: La cellule na2_x4 est capable de charger une capacité de charge 4 fois plus grande que la cellule na2_x1. Evidemment, plus la cellule est puissante, plus la surface de silicium occupée est importante.

Vous pouvez par exemple visualiser le dessin des masques de deux inverseurs de la la bibliothèque SXLIB, en utilisant l'éditeur graphique de la chaîne alliance :

```
> graal -l inv_x1
```

puis

```
> graal -l inv_x4
```

C) Schéma des blocs

C1) schéma du bloc adder4

On cherche à définir le schéma du bloc **adder4**. Un additionneur 4 bits peut être réalisé en interconnectant 4 additionneurs 1 bit suivant le schéma ci-dessous:

Un additionneur 1 bit (encore appelé *Full Adder*) possède 3 entrées a,b,c, et deux sorties s et r. La table de vérité est définie par le tableau ci-dessous. Le bit de "somme" s vaut 1 lorsque le nombre de bits d'entrée égal à 1 est impair. Le bit de "report" est égal à 1 lorsqu'au moins deux bits d'entrée valent 1.

a	b	c	r	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Ceci donne les expressions suivantes :

- s <= a XOR b XOR c
- r <= (a AND b) OR (a AND c) OR (b AND c)

Il existe plusieurs schémas possibles pour réaliser un Full Adder. Nous vous proposons d'utiliser le schéma ci-dessous, qui utilise trois cellules na2_x1 (NAND 2 entrées), et deux cellules xr2_x1 (XOR 2 entrées):

Vous pouvez consulter le modèle comportemental de ces cellules : na2_x1.vbe et xr2_x1.vbe. Les étudiants curieux peuvent visualiser le dessin des masques de ces cellules en utilisant **graal** :

```
> graal -l na2_x1
```

Ecrivez, en langage VHDL structurel, les deux fichiers *adder4.vst*, et *full_adder*.vst, correspondant aux deux schémas ci-dessus.

C2) schéma du bloc accu

Un registre 4 bits peut être réalisé en utilisant 4 cellules sff1_x4 suivant le schéma ci-dessous:

La cellule sff1_x4 est une bascule D à échantillonnage sur front montant. Vous pouvez consulter le modèle comportemental de cette cellule : sff1_x4.vbe .

Les étudiants curieux peuvent visualiser le dessin des masques de la cellule sff1_x4 en utilisant **graal**: graphique de la chaîne alliance :

```
> graal -l sff1_x4
```

Ecrivez, en langage VHDL structurel, le fichier accu4.vst, correspondant à ce schéma.

C3) schéma du bloc mux

Un multiplexeur 4 bits peut être réalisé en utilisant 4 cellules mx2_x2 suivant le schéma ci-dessous:

Vous pouvez consulter le modèle comportemental de cette cellule : mx2_x2.vbe .

Les étudiants curieux peuvent visualiser le dessin des masques de la cellule mx2_x2 en utilisant **graal** :

```
> graal -l mx2_x2
```

Ecivez, en langage VHDL structurel, le fichier mux4.vst, correspondant à ce schéma.

D) simulation zero-delay

On peut maintenant valider fonctionnellement ce schéma en applicant sur cette description structurelle hiérarchique multi-niveaux les mêmes stimuli que ceux qui on été définis dans le TP2 (fichier stimuli.pat). On utilise pour cela le simulateur **asimut** en mode zero-delay, puisque la validation fonctionnelle ne s'intéresse pas aux temps de propagation.

Pour valider fonctionnellement chacun des blocs **adder**, **accu** et **mux**, il faut remplacer progressivement la description comportementale d'un bloc (fichier *.vbe*) par sa description structurelle (fichier '.vst*), en modifiant progressivement le fichier CATAL. Il faut relancer la simulation à chaque étape, en vérifiant qu'on conserve le même comportement.*

```
>asimut -zd addaccu stimuli result
```

On passera par les étapes suivantes:

- validation du bloc **mux4** : on supprime le nom du composant **mux4** du fichier CATAL.
- validation du bloc **accu4** : on supprime le nom du composant **accu4** du fichier CATAL.
- validation du bloc **adder4** : on supprime le nom du composant **adder4** du fichier CATAL.

Lorsque le fichier CATAL est vide, cela signifie que le circuit est simulé comme un schéma hiérarchique multi-niveaux entièrement décrit avec des cellules de la bibliothèque SXLIB.

E) simulation logico-temporelle

Le modèle VHDL comportemental des cellules de la bibliothèque SXLIB contient des informations temporelles. On peut donc lancer une simulation logico-temporelle prenant en compte les temps de propagation dans les cellules de la bibliothèque. On va chercher a observer la propagation du report dans le bloc *adder4*, en observant les signaux *c0_1*, *c1_2*, *c2_3* et *cout*. Il faut modifier le fichier *stimuli.pat* pour observer ces signaux internes, en les

designant par le *cheminom*: Si le nom d'instance du bloc *adder4* est *my_adder*, on designera le signal *c1_0* par *my_adder.c1_0*.

```
>asimut addaccu new_stimuli new_result
```

F) Compte-Rendu

Il vous est demandé un rapport d'une page, au format .pdf. Vous joindrez les fichiers *adder.vst* et *full_adder.vst* en annexe, ainsi que le chronogramme résultat de la simulation logico-temporelle.