

TP3 : Bibliothèque de cellules pré-caractérisées

1. Objectifs
2. A) La bibliothèque SXLIB
3. B) Schéma des blocs
 1. B1) schéma du bloc adder
 2. B2) schéma du bloc accu
 3. B3) schéma du bloc mux
4. C) simulation zero-delay
5. D) simulation logico-temporelle
6. Compte-Rendu

Objectifs

Le principal objectif de ce TP est d'utiliser le langage VHDL pour écrire une description structurelle hiérarchique multi-niveaux du composant *addaccu*, en utilisant une bibliothèque de cellules précaractérisées.

Pour cela, nous allons décomposer les trois blocs **adder**, **mux** et **accu**, définis dans le TP2, et nous allons décrire chacun de ces blocs 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
- vue *logique* : schéma en transistors
- vue *comportementale* : description VHDL

On dit que ces cellules sont pré-caractérisées, car on connaît leurs caractéristiques physiques:

- surface occupée
- consommation
- temps de propagation

A) 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 de métallisation.

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 0.35micron.

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: Lacellule na2_x4 est capable de charger une capacité de charge 4 fois plus grande que la cellule na2_x1. Evidemment, plus cellule est puissante, plus la surface de silicium occupée est importante.

B) Schéma des blocs

B1) schéma du bloc adder

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 :

- $s \leq a \text{ XOR } b \text{ XOR } c$
- $r \leq (a \text{ AND } b) \text{ OR } (a \text{ AND } c) \text{ OR } (b \text{ AND } c)$

Il existe plusieurs schémas possibles pour réaliser un Full Adder. Nous vous proposons d'utiliser le schéma ci-dessous, qui utilise les deux cellules na2_x1 (NAND 2 entrées), et nxr2_x1 (XOR 2 entrées):



Les étudiants curieux peuvent visualiser le dessin de la cellule na2_x1 en utilisant l'éditeur graphique de la chaîne alliance :

```
> graal -l na2_x1
```

Il faut donc écrire explicitement, en langage VHDL structurel, les deux fichiers adder.vst, et half_adder.vst, correspondant aux deux schémas ci-dessus.

B2) schéma du bloc accu

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



La cellule sff1_x4 réalise une bascule D à échantillonnage sur front montant. Les étudiants curieux peuvent visualiser le dessin de la cellule sff1_x4 en utilisant l'éditeur graphique de la chaîne alliance :

```
> graal -l sff1_x4
```

Il faut donc écrire explicitement, en langage VHDL structurel, le fichier `accu.vst`, correspondant à ce schéma.

B3) schéma du bloc mux

Il existe plusieurs façons de réaliser un multiplexeur. Un multiplexeur 4 bits peut être réalisé en interconnectant 4 cellules `mx2_x2` suivant le schéma ci-dessous:



Les étudiants curieux peuvent visualiser le dessin de la cellule `mx2_x2` en utilisant l'éditeur graphique de la chaîne alliance :

```
> graal -l mx2_x2
```

Il faut donc écrire explicitement, en langage VHDL structurel, le fichier `accu.vst`, correspondant à ce schéma.

C) simulation zero-delay

On peut maintenant valider fonctionnellement ce schéma en appliquant sur cette description structurelle hiérarchique multi-niveaux les mêmes stimuli que ceux qui ont été définis dans le TP2 (fichier `new_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 new_stimuli new_result_zd
```

On passera par les étapes suivantes:

- validation du bloc **mux** : on supprime le nom du composant **mux** du fichier CATAL.
- validation du bloc **accu** : on supprime le nom du composant **accu** du fichier CATAL.
- validation du bloc **adder** : on supprime le nom du composant **adder** 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.

D) simulation logico-temporelle

Le modèle VHDL comportemental des cellules de la bibliothèque SXLIB contient des informations temporelles. Vous pouvez par exemple visualiser le modèle VHDL de la cellule `na2_x1` en cliquant [ici](#)?

On peut donc lancer une simulation logico-temporelle prenant en compte les temps de propagation dans les cellules de la bibliothèque.

```
>asimut addaccu new_stimuli new_result
```

Compte-Rendu