TP2 : Modélisation structurelle avec Stratus

Dans ce TP, nous souhaitons réaliser un générateur de circuit addaccu amélioré avec comme paramètre, entre autres, le nombre de bits.

Nous verrons dans ce TP **Stratus** comment permet de décrire des netlists paramétrables et de les utiliser.

1 Introduction

1.2 Circuit addaccu

Dans le circuit **addaccu** sont instanciés trois blocs **mux**, **reg** et **add**.

Les deux blocs **mux** et **reg** sont des générateurs paramétrable décrits dans le langage **Stratus**, ce sont des interconnexions de portes de bases, fournies par la bibliothèque de cellules pré-caractérisées SXLIB.

Le bloc **add**, également décrit dans le langage **Stratus**, instancie un bloc **full_adder**, lui même étant une interconnexion de porte SXLIB, décrit en **Stratus**.

Le circuit addaccu a donc trois niveaux de hiérarchie.

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.

1.2 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 visualiser le dessin des masques de ces cellules en utilisant l'éditeur graphique de la chaîne ALLIANCE **GRAAL**.

1.3 Schéma des blocs

1.3.1 Multiplexeur

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 la cellule *mx2_x2* : mx2_x2.vbe.

1.3.2 Registre

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.

1.3.3 Additionneur

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	s	r
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
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) :

2 Travail à effectuer

1.1 Bloc mux

- Récupérer les deux fichiers permettant de créer le bloc **mux** et les étudier :
  - Netlist en "'Stratus'" du bloc "'mux'"
  - Script pour la création de la netlist

  Ce bloc a la fonctionnalité suivante :

  ```
  si (cmd==0) alors s <= i0 sinon s <= i1
  ```

  i0, i1 et s ayant un nombre de bit paramétrable

- Créer une instance de mux sur 4 bits. Pour ce faire, il faut exécuter le script fourni avec le bon paramètre, soit en exécutant la commande suivante :

  ```
  > python gen_mux.py -n 2
  ```

  , soit en modifiant les droits du fichier :

  ```
  > chmod u+x gen_mux.py
  > ./gen_mux.py -n 2
  ```

Si le script s'effectue sans erreur, un fichier **.vst** est normalement généré. Vous pouvez vérifier qu'il décrit bien le circuit voulu.

1.2 Bloc registre

- En s'inspirant du multiplexeur, écrire le bloc **reg** avec **Stratus** en utilisant exclusivement les cellules de la bibliothèque **sxlib** Ce bloc prend lui aussi comme paramètre le nombre de bits. En outre, ils véri?ent que leur paramètre est compris entre 2 et 64 (ce n?est pas fait dans mux).

- Ecrire le script python permettant de créer l'instance du registre.

1.3 Bloc additionneur

- Ecrire le bloc **full_adder** en utilisant exclusivement les cellules de la bibliothèque **sxlib**.
- Ecrire le script python permettant de créer l'instance du full_adder.
- Ecrire le bloc **adder** instanciant le full_adder créé. Ce bloc prend comme paramètre le nombre de bits.
- Ecrire le script python permettant de créer l'instance de l'additionneur.

1.2 Circuit addaccu

- Ecrire le circuit **addaccu** avec **Stratus**. Ce circuit instancie les trois blocs précédents (**mux**, **add** et **reg**). Le circuit **addaccu** prend également comme paramètre le nombre de bits.
- Ecrire le script python permettant de créer des instances de l'addaccu.

- Ecrire un ?chier **Makefile paramétrable** permettant de produire chaque composant et le circuit addaccu en choisissant le nombre de bits.

- Générer le circuit sur 4 bits.
- Visualiser la netlist obtenue avec **xsch**.

1.3 Circuit addsubaccu

Maintenant, nous souhaitons que l?addaccu puisse effectuer soit des additions, soit des soustractions. Un nouveau paramètre sera donc à apporter pour choisir la fonction à effectuer (Vous avez le choix pour le nom et les valeurs possibles de ce paramètre). Ce nouveau composant sera sur le même schéma que le précédent, avec des modifications à apporter au circuit et/ou ses composants.

- Créer un nouveau composant, appelé **addsubaccu** qui prend en compte cette nouvelle contrainte.
- Ecrire le script python permettant de créer des instances de l'addsubaccu.
- Ecrire un ?chier **Makefile** paramétrable permettant de produire chaque composant et le circuit addsubaccu.

1.4 Fonction Generate

Il n'est pas toujours très pratique d'avoir à générer avec plusieurs scripts les différents blocs d'un circuit. Le langage **Stratus** fournit donc une alternative : la fonction **Generate**.

Par exemple, pour générer une instance du multiplexeur fourni, il suffit d'ajouter la ligne suivante dans le fichier **Stratus** décrivant le circuit instanciant le multiplexeur :

```
Generate ( "mux.mux", "mux_%d" % self.n, param = { 'nbit' : self.n } )
```

Dans cette fonction, le premier argument représente la classe **Stratus** créée (format : *nom_de_fichier.nom_de_classe*), le deuxième argument est le nom du modèle généré, le dernier argument est un dictionnaire initialisant les différents paramètres de cette classe.

- Modifier le fichier décrivant l'addsubaccu et le Makefile de façon à pouvoir créer les instances de ce circuit en n'ayant besoin que d'un script.

1.5 Description de patterns

La chaîne de CAO **ALLIANCE** fourni un outil permettant de décrire des séquences de stimuli : l*outil **GENPAT**. **Stratus** fournit le même service pour la chaîne de CAO **Coriolis**. De plus, **Stratus** encapsule l'appel au simulateur **ASIMUT**.*

- Récupérer les deux fichiers décrivant le bloc mux avec création du fichier de patterns et simulation, et les étudier :
    - ♦ Netlist en "'Stratus'" du bloc "'mux'" avec la description des patterns
    - ♦ Script pour la création de la netlist, du fichier de patterns et du lancement du simulateur
- Créer les patterns et effectuer la simulation des deux autres blocs de la même façon.
- Une fois tous les sous blocs validés, créer les patterns et effectuer la simulation du bloc addsubaccu.

1.6 Bibliothèque DPGEN

TODO

2 Compte rendu

Vous rédigerez un compte-rendu d'une page maximum pour ce TP. Vous expliciterez **en détail** les choix que vous avez fait pour modifier le circuit **addaccu** et/ou ses composants de façon à créer le circuit **adddsubaccu**.

Vous fournirez tous les fichiers écrits, avec les **Makefile** permettant d'effectuer la génération des deux circuits (et l'effacement des fichiers générés).