TP3 : Vue Physique

1 Dessin de cellule

1.1 Introduction

Le but de cet exercice est le dessin sous **graal** d'une Nand à 2 entrées. Les notions de cellules précaractérisées et de de gabarit seront introduites.

Dans les TP précédents nous avons utilisé des cellules d'une bibliothèque. Cette bibliothèque peut être enrichie de nouvelles cellules grâce à l'éditeur **graal**.

graal est un éditeur de layout symbolique intégrant le vérificateur de règles de dessin **druc**.

Cet exercice a pour objectif de dessiner une cellule en tenant compte des règles de dessin fournies.

Remarque : certains outils utilisent un environnement technologique particulier. Il est désigné la variable d'environnement **RDS_TECHNO_NAME** qui doit être positionnée à *opt/alliance/etc/cmos.rds* :

```
> export RDS_TECHNO_NAME=/opt/alliance/etc/cmos.rds
```

1.2 Outils utilisés

Graal

L'éditeur de layout **graal** manipule plusieurs types d'objets différents que l'on peut créer avec le menu **create** :

- les instances (importation de cellules physiques),
- les boîtes d'aboutement qui définissent les limites de la cellule,
- les segments : DiffN, DiffP, Poly, Alu1, Alu2 ...
- le CAluX est utilisé pour désigner une portion possible pour les connecteurs,
- les VIAs ou contacts :ContDiffN, ContDiffP, ContPoly et Via Metal1/Metal2,
- les Big VIAs,

- les transistors : NMOS ou PMOS.

graal utilise la variable d'environnement **GRAAL_TECHNO_NAME**. Elle doit être positionnée à */opt/alliance/etc/cmos.graal* :

```
> export GRAAL_TECHNO_NAME=/opt/alliance/etc/cmos.graal
```

Cougar

L'outil **cougar** est capable d'extraire la netlist d'un circuit aux formats **.vst** ou **.al** à partir d'une description au format **.ap**.

Pour extraire au niveau transistor, la commande à utiliser est :

```
> cougar -t file1 file2
```

cougar utilise les variables d'environnement **MBK_IN_PH** et **MBK_OUT_LO** suivant les formats d'entrée et de sortie. Par exemple pour générer une netlist au format **.al** à partir d'une description **.ap** il faut écrire :

```
> export MBK_IN_PH=ap
> export MBK_OUT_LO=al
> cougar -t file1 file2
```

Yagle et Vasy

L'outil **yagle** est capable d'extraire la description VHDL comportementale d'uncircuit au format **.vhd** à partir d'une *netlist* au format **.al** *si celle-ci est au niveau transistor*.

L'outil **vasy** permet de convertir une description VHDL comportementale du format **.vhd** au format **.vbe**. La commande à utiliser est :

```
> export MBK_IN_LO=al
> export YAGLE_BEH_FORMAT=vbe
> yagle -s file1 file2
> vasy -a -I vhd file1 file2
```

Avant tout vous devez utiliser la commande :

```
> source avt_env.sh
```

avec le fichier avt_env.sh. Cette commande permet de mettre en place l'environnement nécessaire à l'utilisation de **yagle**.

Les documentations pour cet outil se trouvent en :**/users/soft/AvtTools2003/doc** .

Proof

Lorsqu'on veut prouver l'équivalence de deux descriptions comportementales de type *dataflow* d'un même circuit à n entrées, on peut simuler par **asimut** des vecteurs pour les deux descriptions et les comparer. Cette solution devient vite coûteuse en temps CPU et il vaut mieux faire appel à un outil de preuve formelle qui effectue la comparaison *mathématique* des deux réseaux booléens. **proof** réalise cette opération entre les description file1.vbe et file2.vbe par la commande :

```
> proof file1 file2
```

1.3 Le gabarit sxlib

- Les cellules de la bibliothèque **sxlib** ont toutes une hauteur de 50 lambdas et une largeur multiple de 5 lambda.
- Les alimentations Vdd et Vss sont réalisées en Calu1 ; elles ont une largeur de 6 lambdas et sont placées horizontalement en haut et en bas de la cellule.
- Les transistors P sont placés près du rail Vdd tandis que les transistors N sont placés près du rail Vss.
- Le caisson N doit avoir une hauteur de 24 lambdas :
- Les segments spéciaux CAluX (CAlu1, Calu2, CAlu3...) forment l'interface de la cellule et jouent le rôle de connecteurs "étalés". Ils doivent obligatoirement être placés sur une grille de 5x5 lambdas et peuvent se trouver n'importe où à l'intérieur de la cellule.
- Les segments spéciaux TAlux (TAlu1, TAlu2, ...) servent à désigner les obstacles au routeur Lorsque vous voulez protéger des segments AluX, il faut les recouvrir ou les entourer de TAlux correspondant (même couche). Les TAluX sont placés sur une grille au pas de 5 lambdas.
- La largeur minimale de CAlu1 est de 2 lambdas, plus 1 lambda pour l'extension.
- Les caissons N et P doivent être polarisés. Il faut donc les relier respectivement à Vdd et à Vss.

Le schéma de la ?gure suivante présente un résumé de ces contraintes :

1.4 Travail à effectuer

Le schéma théorique du Nand2 est présenté dans la figure suivante :

Réaliser les étapes suivantes :

- Décrire le comportement de la cellule dans un fichier au format **.vbe**.
- Dessiner sur papier un stick-diagram.
- Saisir sous **graal** le dessin de la cellule en respectant le gabarit SXLIB.
    - ♦ On utilisera les largeurs suivantes pour les transistors : WN = WP = 10.
- Valider les règles de dessin symbolique en lançant la commande **DRUC** sous graal.
- Utilisez la commande **EQUI** pour vérifier la connectivité des équipotentielles.
- Extraire la netlist de l'inverseur au format **.al** avec **cougar**.
- Utiliser les outils **yagle** et **proof** pour vérifier le comportement.
- Créer un **Makefile** pour automatiser les différentes étapes.

N'oubliez pas que les mans existent ...

2 Routage manuel

2.1 Introduction

Nous avons vu dans l'exercice précédent comment dessiner une cellule précaractérisée respectant le gabarit de le bibliothèque **sxlib**. Le but de cet exercice est de réaliser le dessin des masques d'un bloc combinatoire résultant de l'interconnexion de plusieurs cellules de la bibliothèque **sxlib**. On va donc effectuer *à la main* (c'est à dire en utilisant l'éditeur interactif **graal**), le placement et le routage de ce bloc, pour mieux comprendre les problèmes que doivent résoudre les outils de placement/routage automatiques.

Le bloc à réaliser est l'additionneur 4 bits vu lors du TP précédent. Il contient donc 4 Full-adders. Regardons plus précisément les caractéristiques de cette cellule :

- La cellule a une largeur de x pitchs.
- Le signal d'entrée x est accessible sur x pistes de routage.
- Le signal de sortie x est accessible sur x pistes de routage.

2.2 Travail à réaliser

- Saisir sous **graal** le dessin du bloc *adder* en instanciant les 4 portes *fulladder_x2*.
- Dessiner les fils de routage sous **graal**.
- Utiliser la commande *equi* pour vérifier la connectivité de chacun des signaux.
- Vérifier l'absence de violation des règles de dessin en lançant la commande *druc* sous **graal**. Pour que cette vérification soit significative, il faut préalablement "mettre à plat" le bloc, en utilisant la commande *real flat*.

- Extraire la netlist du bloc au format **.al** avec l'outil **cougar** mais sans descendre au niveau des transistors : On veut obtenir une netlist de cellules, et non une netlist de transistors.
- Vérifier que la netlist obtenue *adder.al* et la netlist au format *.vst* obtenue au TP précédent sont isomorphes en utilisant l'outil **lvx**.
- Créer un fichier **Makefile** automatisant la procédure de validation.

3 Placement de l'addaccu

3.1 Introduction

Le TP précédent vous a permis d'utiliser le langage **Stratus** pour décrire la netlist hiérarchique d'un addaccu.

On va maintenant utiliser le langage **Stratus** pour introduire des directives de placement dans le fichier .py décrivant la netlist de l'addaccu à base de générateurs de la bibliothèque **dpgen**.

En effet, la régularité des générateurs **dpgen** a été exploitée en imposant un placement en colonnes : tous les bits de cahque opérateur sont placés en colonne. Il est donc possible d'imposer un placement relatif des colonnes les unes par rapport aux autres.

3.2 Fonctions de placement

Pour définir les directives de placement, le langage **Stratus** fournit les fonctions suivantes :

- Place()
- PlaceRight(), PlaceTop(), PlaceLeft(), PlaceBottom()
- SetRefIns()
- DefAb(), ResizeAb()

Vous pouvez consulter le manuel de **Stratus** en ligne :

?https://www-asim.lip6.fr/recherche/coriolis/doc/en/html/stratus/index.html

Toutes ces fonctions doivent être utilisées dans la méthode *Layout* associée au bloc considéré.

Ensuite pour générer le fichier *.ap*, il faut rajouter l'appel à la méthode *Layout* dans le fichier *.py* générant le bloc et ne pas oublier de sauvegarder le résultat sur disque (argument **PHYSICAL** à l'appel de la méthode **Save**). Pour visualiser le placement, vous pouvez, soit utiliser **graal** sur le fichier *.ap* soit utiliser le visualisateur de Coriolis en ajoutant la méthode **View** dans le script avant l'appel de la méthode **Save**.

3.3 Travail à faire

Il vous est demandé ici d'effectuer un placement pour l'opérateur *addaccu* déjà réalisé.

4 Compte rendu

Vous rédigerez un compte-rendu de deux pages maximum pour ce TP dans lequel vous expliquerez :

- les choix effectués pour la création de la cellule Nand ainsi que la démarche de validation,
- la façon dont vous avez routé l'additionneur,
- la façon dont vous avez placé les colonnes de votre addaccu.

Vous joindrez vos fichiers source sans oublier les fichiers Makefile.