

# TP3 : Vue Physique

1. 1 Dessin de cellule
  1. 1.1 Introduction
  2. 1.2 Outils utilisés
    1. Graal
    2. Cougar
    3. Yagle et Vasy
    4. Proof
  3. 1.3 Le gabarit sxlib
  4. 1.4 Travail à effectuer
2. 2 Routage manuel
  1. 2.1 Introduction
  2. 2.2 Travail à réaliser
3. 3 Compte rendu

## 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 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 par la variable d'environnement **RDS\_TECHNO\_NAME** qui doit être positionnée à */users/outil/alliance/Linux.SLSoc5x/etc/cmos.rds* :

Vérifier que cette variable est bien positionnée :

```
> echo $RDS_TECHNO_NAME
```

Si ce n'est pas le cas, positionnez la :

```
> 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**. Vérifiez qu'elle est bien positionnée à `/users/outil/alliance/Linux.SLSoc5x/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'un circuit au format **.vbe** à partir d'une netlist au format **.al** si celle-ci est au niveau transistor.

```
> export MBK_IN_LO=al
> ~encadr/yagle -s file1 file2
```

## 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 CAu1 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 figure 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. N'hésitez pas à lancer **DRUC** au fur et à mesure de façon à détecter les erreurs rapidement !!
- 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 la 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 le full-adder vu lors du TP précédent. Il contient donc 3 portes Nand2 et deux portes Xor. Regardons plus précisément les caractéristiques de ces deux cellules :

- La *na2\_x1* a une largeur de 4 pitches,
- le signal d'entrée i0 est accessible sur 7 pistes de routage,
- le signal d'entrée i1 est accessible sur 6 pistes de routage,
- le signal de sortie nq est accessible sur 7 pistes de routage.
  
- La *xr2\_x1* a une largeur de 9 pitches,
- le signal d'entrée i0 est accessible sur 7 pistes de routage,
- le signal d'entrée i1 est accessible sur 7 pistes de routage,

- le signal de sortie *q* est accessible sur 6 pistes de routage.

## 2.2 Travail à réaliser

- Saisir sous **graal** le dessin du bloc *fulladder* en instanciant les 5 portes utiles. Réfléchissez bien au placement.
- 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*. Vous pouvez également utiliser l'outil **druc** en ligne de commande.
- 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 *.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 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.