TP6 : Dessin de cellules précaractérisées

1. 1 Objectif
2. 2 Outils CAO utilisés
    1. 2.1 GRAAL
    2. 2.2 COUGAR
    3. 2.3 YAGLE
    4. 2.4 PROOF
3. 3 schéma des cellules à réaliser
4. 4 Le gabarit sxlib
5. 5 Réalisation de l'inverseur
6. 6 Réalisation d'une porte Nand2

1 Objectif

Le but de ce TP est de vous apprendre à utiliser l'éditeur graphique **graal**, et différents outils de vérification, pour réaliser le dessin des masques de deux cellules précaractérisées respectant le gabarit de la bibliothèque **SXLIB**.

On dessinera tout d'abord la cellule **inv_x1**, puis on dessinera la cellule **na2_x1**.

2 Outils CAO utilisés

Le dessin des masques des cellules manipulées par la chaîne de CAO **Alliance** utilise la technique de dessin symbolique sur grille fixe, pour permettre la migration technologique.

?ftp://asim.lip6.fr/pub/amd2901/symb_rules00-1.pdf

Tous les fichiers contenant la vue physique (dessin des masques) possèdent l'extension **.ap** (comme Alliance Physique).

2.1 GRAAL

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

- Les instances (importation de cellules pré-existantes)
- Les segments : DiffN, DiffP, Poly, Alu1, Alu2 ...
- Le CAluX est utilisé pour les *connecteurs*, c'est à dire pour les segments faisant partie de l'interface de la cellule
- Les VIAs ou contacts : ContDiffN, ContDiffP, ContPoly? et Via Metal1/Metal2.
- Les Big VIAs, qui correspondent à des matrices de vias.
- Les transistors : NMOS ou PMOS
- Les boîtes d'aboutement qui définissent l'encombrement de la cellule

graal permet d'appeler directement le vérificateur de règles de dessin, en lançant _ sous **graal** la commande DRUC. Le vérificateur des règles de dessin **druc** peut également être utilisé indépendamment de **graal**. **graal** permet

également de vérifier la connectivité d'une équipotentielle, au moyen de la commande EQUI.

L'outil **graal** utilise les variables d'environnement **GRAAL_TECHNO_NAME**. et **RDS_TECHNO_NAME**. Elles doivent être positionnée aux valeurs suivantes :

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

2.2 COUGAR

L'outil **cougar** est un extracteur, capable d'extraire la *net-list* d'un circuit à partir du dessin des masques. L'extracteur **cougar** permet de générer deux types de descriptions :

- il peut générer une *net-list* décrite au niveau transistors (les éléments terminaux sont des transistors)
- il peut générer une *net-list* décrite au niveau cellules (les éléments terminaux sont des cellules décrites comme des boîtes noires).

cougar utilise les variables d'environnement **MBK_IN_PH** et **MBK_OUT_LO** pour spécifier les formats d'entrée et de sortie. Pout représenter une *net-list* au niveau transistors, on utilisera de préférence le format **.spi**, qui respecte la syntaxe imposée par les simulateurs **spice** ou **eldo**. Pour représenter une *net-list* au niveau celules, on utilisera de préférence le format *.vst* (VHDL structurel).

Par exemple, pour extraire au niveau transistor, il faut utiliser l'option -t de **cougar**, et écrire :

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

2.3 YAGLE

L'outil **yagle** a été conçu par le laboratoire LIP6, et est actuellement commercialisé par la société **avertec**. L'outil d'abstraction fonctionnelle **yagle** permet de générer une description comportementale de type *réseau Booléen*, à partir d'une *net-list* de transistors au format **.spi**.

Le fichier au format **.vhd** généré par YAGLE respecte la syntaxe VHDL imposée par la chaîne de CAO commerciale **synopsys**. L'outil **VASY** permet de convertir cette description VHDL comportementale au format **.vhd** vers le format **.vbe** de la chaîne **alliance**. La commande à utiliser est :

```
> source avt_env.sh
> export MBK_IN_LO=spi
> yagle -s file1 file2
> vasy -a -I vhd file1 file2
```

La commande avt_env.sh permet de mettre en place l'environnement nécessaire à l'utilisation de **yagle**. La documentation pour cet outil se trouvent en : /users/soft/AvtTools2003/doc .

2.4 PROOF

Lorsqu'on veut prouver l'équivalence de deux descriptions comportementales de type réseau Booléen (assignations concurrentes) on peut faire de la cosimulation avec **asimut** Cette solution devient vite coûteuse en temps CPU et il vaut mieux faire appel à un outil de preuve formelle qui effectue la preuve *mathématique* de l'équivalence des deux réseaux Booléens.

On lance l'outil **proof** par la commande:

```
>proof file1 file2
```

3 schéma des cellules à réaliser

Le schéma de l'inverseur et du nand2 sont présentés ci dessous :

On utilisera les largeurs suivantes pour les transistors de la cellule inv_x1 : WN =5 / WP = 10

On utisera les largeurs suivantes pour les transistors de la cellule na2_x1 : WN = WP = 10

4 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. Ils doivent obligatoirement être placés sur la grille de routage de 5x5 lambdas et peuvent se trouver n'importe où à l'intérieur de la cellule
- Les segments spéciaux TAlux (TAlu1, TAlu2, ...) servent à indiquer d'éventuels obstacles dans les couches métalliques plus élévées que Alu1. Le routeur n'a pas connaissance des segments internes à la cellule. Si un segment Alux a été utilisé à l'intérieur de la cellule, cet obstacle doit être signalé au routeur qui va utiliser la cellule par un segment TAlux recouvrant le segment Alux.
- 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

5 Réalisation de l'inverseur

- Décrire le comportement de la cellule inv_x1 dans un fichier au format **.vbe**
- Dessiner sur papier un *stick-diagram* pour l'inverseur
- Saisir sous **graal** le dessin de la cellule en respectant le gabarit **SXLIB**
- 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 **.spi** avec **cougar**
- Utiliser les outils **yagle** et **proof** pour vérifier le comportement

6 Réalisation d'une porte Nand2

- Décrire le comportement de la cellule na2_x1 dans un fichier au formar **.vbe**
- Dessiner sur papier un *stick-diagram* pour cette porte
- Saisir sous **graal** le dessin de la cellule en respectant le gabarit **SXLIB**
- 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 **.spi** avec **cougar**
- Utiliser les outils **yagle** et **proof** pour vérifier le comportement

N'oubliez pas que les mans existent...