Dessin de Cellule

Contents

Introduction

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

Dans les TME 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.

Vous devez travailler dans l'environnement ALLIANCE . Vérifier que cette variable est bien positionnée :

```
etudiant@pc:TME3> echo $RDS_TECHNO_NAME
```

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

```
etudiant@pc:TME3> export RDS_TECHNO_NAME=/opt/alliance/etc/cmos.rds
```

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 :

```
/soc/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 :

```
etudiant@pc:TME1> 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 :

```
etudiant@pc:TME1> export MBK_IN_PH=ap
etudiant@pc:TME1> export MBK_OUT_LO=al
etudiant@pc:TME1> cougar -t file1 file2
```

Yagle et Proof

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.

```
etudiant@pc:TME1> export MBK_IN_LO=al
etudiant@pc:TME1> ~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 :

```
etudiant@pc:TME1 proof file1 file2
```

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 (centrés à 3 et 47 lambdas en Y); 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 (centré à 39 lambdas en Y).

- 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.
- La largeur minimale de CALU1 est de 2 lambdas, plus 1 lambda pour l'extension.

Le schéma de la figure suivante présente un résumé de ces contraintes :

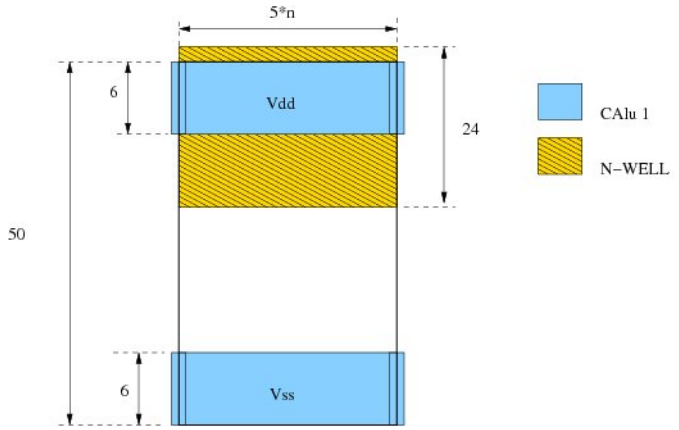

Figure 1: Figure 1 -- Gabarit **sxlib**

Travail à Effectuer

Le schéma théorique du NAND2 est présenté dans la figure suivante :

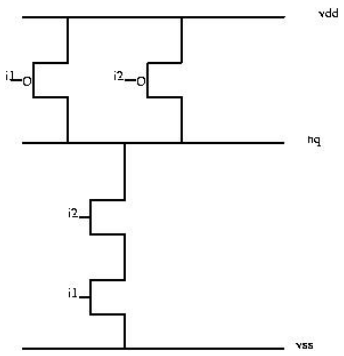

Figure 2: Figure 2 -- Schéma en transitors du NAND2

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 page de manuels (`man`) existent ...

Compte Rendu

Vous rédigerez un petit compte-rendu pour ce TME dans lequel vous expliquerez :

- Les choix effectués pour la création de la cellule NAND ainsi que la démarche de validation,
- Le `Makefile` de vérification de votre cellule

Vous joindrez vos fichiers source sans oublier les fichiers `Makefile`.