TP7 : Placement et routage du circuit AM2901

1. 1 Objectif
2. 2 Variables d'environnement
3. 3 Fonctions de placement fournies par **Stratus**
4. 4 Travail à effectuer
    1. 4.1 Placement explicite des opérateurs du chemin de données
    2. 4.2 Placement du coeur
    3. 4.3 Placement de la couronne de plots autour du coeur
    4. 4.4 Routage des alimentations
    5. 4.5 Placement de la logique irrégulière
    6. 4.6 Routage des signaux d'horloge
    7. 4.7 Routage des signaux logiques
    8. 4.8 Validation
5. 5 Compte rendu
6. 6 Conclusion

1 Objectif

Le but de ce TP est d'utiliser les outils de placement / routage automatique du flot Coriolis/Alliance? ainsi que tous les outils de vérification vus dans les TPs précédents, pour générer le dessin des masques du circuit AM2901.

Les TPs 3 et 4 vous ont permis d'utiliser le langage **Stratus** pour décrire la netlist hiérarchique du circuit AM2901.

On va maintenant utiliser le langage **Stratus** pour introduire des directives de placement dans les différents fichiers *.py* décrivant la netlist.

Il est par exemple possible d'exploiter la régularité des opérateurs du chemin de données pour imposer un placement en colonnes : tous les bits d'un même opérateur sont placés en colonne, et il est possible d'imposer un placement relatif des colonnes les unes par rapport aux autres. On va également définir le placement des plots d'entrée/sortie sur la périphérie du circuit.

Par ailleurs, on va également utiliser **Stratus** pour effectuer le routage de certains signaux particuliers comme les alimentations Vdd et Vss.

Le routage final sera effectué par l'outil **nero**.

Vous utiliserez aussi **cougar** pour obtenir une netlist extraite, et **lvx**, pour comparer la netlist extraite à la netlist initiale.

Vous utiliserez conjointement les cellules de la bibliothèque **SXLIB** et les macro-cellules génériques de la bibliothèque **DP_SXLIB**.

2 Variables d'environnement

Vous devez positionner les variables d'environnement suivantes :

```
> export VH_MAXERR=10
> export MBK_WORK_LIB=.
> export MBK_CATA_LIB=$ALLIANCE_TOP/cells/sxlib
> export MBK_CATA_LIB=$MBK_CATA_LIB :$ALLIANCE_TOP/cells/dp_sxlib
> export MBK_CATA_LIB=$MBK_CATA_LIB :$ALLIANCE_TOP/cells/pxlib
```

```
> export MBK_CATA_LIB=$MBK_CATA_LIB :.
> export MBK_CATAL_NAME=CATAL
> export MBK_IN_LO=vst
> export MBK_OUT_LO=vst
> export MBK_IN_PH=ap
> export MBK_OUT_PH=ap
> export CRL_OUT_LO=vst
> export CRL_OUT_PH=ap
> export PYTHONPATH=/opt/coriolis/lib/python2.3/site-packages/stratus
> export PYTHONPATH=/opt/coriolis/lib/python2.3/site-packages/isobar :$PYTHONPATH
> export PYTHONPATH=/opt/coriolis/lib/python2.3/site-packages :$PYTHONPATH
```

NB : Ces variables d'environnement sont positionnées par défaut, mais il est utile de les vérifier.

D'une manière générale, les fichiers décrivant une netlist logique doivent porter le même nom que le fichier correspondant décrivant la vue physique.

C'est à dire que le fichier am2901_dpt.vst (vue logique) doit correspondre au fichier am2901_dpt.ap (vue physique). Il en va de même pour am2901_core.

3 Fonctions de placement fournies par Stratus

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.

A titre d'exemple, on donne le code suivant pour le fichier circuit.py décrivant la cellule présentée dans le TP4 :

```
#!/usr/bin/env python
from stratus import *
# definition du bloc de nom "circuit"
class circuit ( Model ):
...
# on suppose que les instances i1, i2, i3 ont été créées
  def Layout ( self ):
    Place ( self.i1, NOSYM, XY ( 0, 0 ) )
    PlaceRight ( self.i2, NOSYM )
    PlaceRight ( self.i3, NOSYM )
```

Ensuite pour générer le fichier circuit.ap, il faut rajouter l'appel à la méthode *Layout* dans le fichier *test_circuit.py* et ne pas oublier de sauvegarder le résultat sur disque :

```
# creation de la vue physique (placement)
my_circuit.Layout()

# sauver les fichiers ''mon_circuit.vst'' et ''mon_circuit.ap''
my_circuit.Save ( PHYSICAL )
```

4 Travail à effectuer

4.1 Placement explicite des opérateurs du chemin de données

Reprendre le fichier *am2901_dpt.py* du TP4. Pour l'instant, ce fichier ne comporte qu'une description de la netlist, qui a permis de générer un fichier *am2901_dpt.vst*.

- Placer explicitement les colonnes représentant les différents opérateurs 4 bits du chemin de données les unes par rapport aux autres, en ajoutant une méthode *Layout* dans ce fichier.
- Modifier l'appel à la fonction *Generate* dans le coeur de façon à créer la vue physique du chemin de données.
- Faire appel à la méthode *View* pour visulaliser le placement généré.

4.2 Placement du coeur

Reprendre le fichier *am2901_core.py* décrivant le coeur du circuit AM2901 et introduire les étapes suivantes dans la méthode *Layout* :

- Placer le chemin de données : fonction Place()
- Agrandir la boite d'aboutement du coeur : fonction ResizeAb() Cette étape est utile pour réserver la place nécessaire au placement des cellules de la partie contrôle. La logique "irrégulière" constituant la partie contrôle n'a pas besoin d'être placée explicitement. Cela sera fait automatiquement par la suite !
- Placer les rails de rappels d'alimentation dans le coeur : fonctions AlimVerticalRail() et AlimHorizontalRail()
- Placer les connecteurs du coeur : fonction AlimConnectors()
- Vérifier le résultat :

```
> ./execute_am2901_core.py
```

4.3 Placement de la couronne de plots autour du coeur

Reprendre le fichier *am2901_chip.py* du TP4 (circuit complet avec les plots), et introduire les étapes suivantes dans la méthode *Layout* :

- Définir la taille de la boîte d'aboutement globale du circuit de façon à ce que les plots puissent être placés à la périphérie : fonction DefAb() (On peut commencer par définir une boite d'aboutement de 4000 par 4000 et essayer ensuite de la réduire)
- Placer le coeur du circuit au centre de la boîte d'aboutement du chip : fonction PlaceCentric()
- Définir sur quelle face et dans quel ordre placer les plots. Cela se fait à l'aide des 4 fonctions : PadNorth(), PadSouth(), PadEast() et PadWest().
- Vérifier le résultat :

```
> ./execute_amd2901_chip.py
```

4.4 Routage des alimentations

- Créer la grille d'alimentation : fonction PowerRing()
- Vérifier le résultat :

```
> ./execute_amd2901_chip.py
```

4.5 Placement de la logique irrégulière

C'est le placeur **mistral** qui se charge de placer automatiquement les cellules non encore placées. Il détecte quelles sont les cellules qui n'ont pas été placées et complète le placement en utilisant les zones "vides".

- Appeler le placeur **mistral** : fonction *PlaceGlue ()*

Attention : Pour pouvoir placer automatiquement la logique "irrégulière", il faut avoir préalablement défini la position des plots d'entrée/sortie sur les 4 faces du circuit. L'outil de placement automatique place les cellules non placées en se basant sur les attirances vers les plots ainsi que vers les cellules déjà placées.

- Vérifier le résultat :

```
> ./execute_amd2901_chip.py
```

- Effectuer le placement automatique de cellules de bourrage : fonction FillCell()

- Vérifier le résultat :

```
> ./execute_amd2901_chip.py
```

4.6 Routage des signaux d'horloge

- Construire le réseau maillé correspondant au signal d'horloge interne : fonction RouteCk()
- Vérifier le résultat :

```
> ./execute_amd2901_chip.py
```

4.7 Routage des signaux logiques

L'appel au routeur automatique **nero** n'est pas encore intégré dans le langage **Stratus**. Pour effectuer le routage de tous les signaux autres que le signal d'horloge et les signaux d'alimentation, il faut lancer **nero** de la manière suivante :

```
> nero -V -p amd2901_chip amd2901_chip amd2901_chip_r
```

L'option -p indique que vous fournissez un fichier de placement en argument. Le deuxième argument est le fichier définissant la *net-list*, le troisième est le nom du fichier résultat.

4.8 Validation

- Valider le routage en utilisant les les outils **druc**, **cougar** et **lvx**, comme cela a été fait dans le TP7.

```
> druc amd2901_chip_r
> export MBK_OUT_LO=al
> cougar -f amd2901_chip_r
> lvx vst al amd2901_chip amd2901_chip_r -f
```

- Resimuler la netlist extraite avec **asimut**. Préciser le format de la netlist dans la variable d'entrée **MBK_IN_LO** avant la simulation.

```
> export MBK_IN_LO=al
```

- Pour connaitre le nombre de transistors, on peut effectuer une extraction au niveau transistors :

```
> cougar -v -t amd2901_chip_r amd2901_chip_r_t
```

5 Compte rendu

Vous rendrez un compte-rendu explicitant les différentes étapes de ce TP :

- Décrivez précisément le flot de conception.
- Quels choix avez-vous retenus pour le placement des colonnes du chemin de données ?
- votre circuit est-il limité par les plots ou par la taille du coeur (pad limited ou core limited) ?
- Quels sont les résultats donnés par lvx ?
- etc ...

Les schémas sont appréciés.

Fournissez les fichiers source de ce TP ou donnez vos logins, vos noms et prénoms, et vos répertoires de travail (dans ce cas laissez libre accès à vos répertoires en lecture !).

Pensez à fournir le fichier Makefile automatisant toutes les étapes. **La première étape de correction consistera à effectuer *make clean; make*.**

6 Conclusion

Ce TP vous a permis de passer par la plupart des étapes nécessaires à la conception physique d'un circuit réalisé en cellules précaractérisées avec préplacement des parties régulières.

Ces mêmes outils seront utilisés pour laréalisation du processeur MIPS R3000.