TP8 : Placement et routage du circuit AM2901

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.

Le TP4 vous a 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 *net-list*.

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 relatifn 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 VSS et VDD.

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

Vous utiliserez aussi **cougar** pour obtenir une net list extraite, et **lvx**, pour comparer la *net-list* extraite à la *net-list* 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 *net-list* 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 vous disposez des fonctions de '**STRATUS**' 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 :

```
#!/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* :

```
#!/usr/bin/env python
from stratus import *
from circuit import circuit

# creation du circuit
my_circuit = circuit ( "mon_circuit" )

# creation de l?interface
my_circuit.Interface() # creation de l?interface

# creation de la vue structurelle (netlist)
my_circuit.Netlist()
```

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

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

4 Placement explicite des opérateurs du chemin de données

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

Il s'agit maintenant de placer explicitement les colonnes représentants les différents opérateurs 4 bits du chemin de données les unes par rapport aux autres.

Après avoir modifié le fichier *am2901_dpt.py* générez le fichier de placement *am2901_dpt.ap* :

```
> ./execute_amd2901_dpt.py
```

5 placement des blocs réguliers dans le coeur

à partir du fichier de description structurelle *am2901_core.py* décrivant le coeur du circuit AM2901, introduisez 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
```

6 Placement de la couronne de plots autour du coeur

Reprenez le fichier *am2901_chip.py* (circuit complet avec les plots), et ajoutez une méthode Layout comme indiqué ci-dessous.

Dans le fichier *amd2901_chip.py*, les plots sont instanciés dans la méthode Netlist :

```
def Netlist ( self ) :
...
  Inst ( "pck_px", "p_ck"
        , map = { ?pad? : self.ck
                , ?ck? : cki
                , ?vddi? : self.vdd
```

```
                    , ?vssi? : self.vss
                    , ?vdde? : self.vdde
                    , ?vsse? : self.vsse
                    }

        )
```

Il faut donc, 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 vous souhaitez placer les plots. Cela se

  fait à l'aide des 4 fonctions : PadNorth (), PadSouth (), PadEast () et PadWest ().

- Vérifier le résultat :

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

7 Routage des alimentations

Vous devez utiliser la fonction PowerRing () pour créer la grille d'alimentation, et Vérifier le résultat :

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

8 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". Pour appeler le placeur **Mistral**, vous devez faire appel à la 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.

Vérifiez le résultat :

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

Le placement automatique se termine par l'appel à la fonction FillCell () qui effectue le placement automatique de cellules de bourrage.

Vérifiez le résultat :

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

6.4 Routage des signaux d'horloge

Vous devez utiliser la fonction RouteCk () qui route le signal d'horloge. Vérifiez le résultat :

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

6.5 Routage des signaux logiques

Routez automatiquement tous les signaux autres que le signal d'horloge et les signaux d'alimentation en utilisant NERO de la manière suivante :

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

L'option -p indique que vous transmettez un placement, à savoir celui du chip. Le troisième argument est la netlist du chip, le quatrième est le fichier résultat.

NOTA BENE : La variable **MBK_CATA_LIB** ne doit contenir qu'une seule fois les chemins d'accès aux bibliothèques.

6.6 Validation du chip

- On validera le travail de **NERO** avec les outils **DRUC**, **COUGAR** et **LVX**.

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

- Simulez à nouveau la netlist extraite avec **ASIMUT**. Précisez le format de la netlist

dans la variable d'entrée **MBK_IN_LO** avant la simulation.

```
> export MBK_IN_LO=al
```

Faites attention au fichier CATAL'"

- Pour connaitre le nombre de transistors, on effectue une extraction du circuit au niveau

transistor :

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

Conclusion

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

Ces mêmes outils seront utilisés pour laréalisation du processeur MIPS R3000. Le compte-rendu du TP doit comporter :

Vos logins, vos noms et prénoms, et vos répertoires de travail pour ce TP (laissez libre accès à vos répertoires en lecture !). Une description exacte de la méthodologie employée, incluant les éventuels probl èmes rencontrés.

Pour l'amd2901, décrivez 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... Les schémas sont appréciés.

Les Makeles du flot total. (Les Makefiles seront testés à la fin de ce TP) NE PAS JOINDRE DE LISTINGS DE FICHIERS (SAUF LES MAKEFILES). Merci et bon courage !