

TP4 : AM2901

1. 1 Architecture interne du circuit Am2901
2. 2 Etude des fichiers fournis
 1. 2.1 Partie contrôle
 2. 2.2 Chemin de données
3. 3 Placement / Routage
 1. 3.1 Préplacement des structures régulières
 2. 3.2 Placement du coeur et de la couronne de plots
 3. 3.3 Routage des alimentations
 4. 3.4 Placement de la logique irrégulière
 5. 3.5 Routage des signaux d'horloge
 6. 3.6 Routage des signaux logiques
 7. 3.7 Validation
4. 4 Rapport

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 TP précédents, pour générer le dessin des masques du circuit AM2901.

Vous avez dans les TP précédents appris à utiliser le langage **Stratus** pour décrire des netlists hiérarchiques et des directives de placement manuel.

Vous allez maintenant utiliser **Stratus** pour définir des directives de placement/routage automatiques.

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

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

1 Architecture interne du circuit Am2901

La description générale du processeur AM2901 est donnée par : [?ftp://asim.lip6.fr/pub/amd2901/amd2901.pdf](ftp://asim.lip6.fr/pub/amd2901/amd2901.pdf).

Nous décomposons le circuit en 2 blocs : la partie contrôle, et la partie opérative ou chemin de données.

- Le chemin de données contient les parties régulières de l'Amd2901 c'est à dire les registres et l'unité arithmétique et logique.
- La partie contrôle contient la logique irrégulière, c'est à dire le décodage des instructions et le calcul des "drapeaux" (indicateurs, ou "Flags").

Le *coeur* instancie le chemin de données et la partie contrôle, le *chip* instancie le coeur et les plots, donnant la hiérarchie présentée ci-après.



2 Etude des fichiers fournis

Les Fichiers fournis sont les suivants :

- description du comportement de la partie contrôle de l'AM2901
- description logique de la partie chemin de données de l'AM2901
- description logique du coeur de l'AMD2901

- description logique du circuit contenant les plots et le coeur de l'AM2901
- script python de création du circuit AM2901
- le fichier de vecteurs de test de l'AMD2901
- Catalogue des modèles

2.1 Partie contrôle

- Etudiez le fichier `amd2901_ctl.vbe` fourni (vous pouvez entre autres vérifier qu'il correspond bien aux données fournies).
- Générez la vue structurelle de l'AM2901 avec le script python fourni.
- Lancez la simulation avec **asimut** (Vérifiez que le fichier CATAL indique bien au simulateur qu'il faut utiliser la description comportementale (`.vbe`) de la partie controle).

```
> asimut amd2901_chip pattern resultat
```

- Utilisez les outils de synthèse de la chaîne **Alliance** pour réaliser la synthèse logique avec les cellules pre-caractérisées de **sxlib**.
- Utilisez de nouveau **asimut** pour valider le schéma obtenu en simulant le circuit complet avec les vecteurs de test fournis. Pensez à remplacer la vue comportementale de la partie contrôle par la vue structurelle en ôtant le nom `amd2901_ctl` du fichier CATAL (Notez que l'on réalise une simulation "zero délai" de la netlist).

```
> asimut -zerodelay amd2901_chip pattern resultat
```

- Automatisez ces étapes à l'aide d'un **Makefile**.

En cas de problème(s), n'hésitez pas à utiliser **xpat**.

2.2 Chemin de données

Le chemin de données est formé de la logique régulière du circuit.

Afin de profiter de cette régularité, on utilise les opérateurs vectoriels de la bibliothèque **Dpgen**. Cela permet d'optimiser le schéma en utilisant plusieurs fois le même matériel. Par exemple, les amplificateurs des signaux de commande d'un multiplexeur sur n bits sont partagés par les n bits ...

Le chemin de données de l'Am2901 peut être schématisé par les figures ci-dessous.



- Etudiez Le fichier fourni décrivant le chemin de données : cef fichier comporte non seulement la description de la netlist du chemin de données mais aussi le placement explicite des colonnes représentant les différents opérateurs 4 bits du chemin de données les unes par rapport aux autres.



- L'applier à la méthode *View* dans le script python permet de visualiser le placement du chemin de données. Vous pouvez également visualiser ce placement en utilisant **graal**.
- Etudiez le placement choisi : vérifiez entre autres que les colonnes ayant un grand nombre d'interconnexions communes sont *proches*.

3 Placement / Routage

3.1 Préplacement des structures régulières

Introduire les étapes suivantes dans la méthode *Layout* du fichier *am2901_core.py* décrivant le coeur du circuit AM2091 :

- Placer le chemin de données : fonction *Place()*.
- Agrandir la boîte 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()*.
- Modifier l'appel à la fonction *Generate* dans le chip de façon à générer la vue physique du coeur.
- Faire appel à la méthode *View* pour visualiser.

3.2 Placement du coeur et de la couronne de plots

Dans le fichier *amd2901_chip.py* fourni, les plots sont instanciés dans la méthode *Netlist*. Il vous faut donc :

- 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 boîte 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()*.
- Visualiser le résultat.

3.3 Routage des alimentations

- Créez la grille d'alimentation : fonction *PowerRing()*.
- Visualisez le résultat.

3.4 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".



- Appelez 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.)
- Visualisez le résultat.
- Effectuez le placement automatique de cellules de bourrage : fonction *FillCell()*.
- Visualisez le résultat.

3.5 Routage des signaux d'horloge

- Construisez le réseau maillé correspondant au signal d'horloge interne : fonction *RouteCk()*.
- Visualisez le résultat.

3.6 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 *netlist*, le troisième est le nom du fichier résultat.

3.7 Validation

- Validez le routage en utilisant les les outils **druc**, **cougar** et **lvx**.

```
> druc amd2901_chip_r
> export MBK_OUT_LO=a1
> cougar -f amd2901_chip_r
> lvx vst a1 amd2901_chip amd2901_chip_r -f
```

- Resimulez 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=a1
```

- Pour connaître le nombre de transistors, vous pouvez effectuer une extraction au niveau transistors :

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

4 Rapport

TODO