TP4 : AM2901

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.

Vous avez dans les TPs 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, **lvx**, pour comparer la netlist extraite à la netlist initiale ainsi que l'analiseur temporel **TAS**.

L'usage d'un **Makefile** est obligatoire.

1 Architecture interne du circuit Am2901

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

Nous décomposons le circuit en 2 blocs : la partie contôle, 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 founies).
- 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'appel 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

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

- Placez le chemin de données aux coordonnées (0, 0) : fonction *Place()*.
- Agrandissez la boite d'aboutement du coeur : fonction *ResizeAb()*. Cette étape est utile pour réserver la place nécessaire aux cellules de la partie contrôle qui seront placées de façon automatique par la suite. Vous pouvez par exemple dans un premier temps doubler sa hauteur et modifier votre choix une fois que vous aurez visualisé la partie contrôle placée.
- Placez les rails de rappels d'alimentation dans le coeur : fonctions *AlimVerticalRail()* et *AlimHorizontalRail()*.
- Placez les connecteurs du coeur : fonction *AlimConnectors()*.
- Modifiez l'appel à la fonction *Generate* dans le chip de façon à générer la vue physique du coeur.
- Faites appel à la méthode *View* pour visualiser le résultat.

3.2 Placement du coeur et de la couronne de plots

Introduisez les étapes suivantes dans la méthode *Layout* du fichier *am201çchip.py* décrivant le circuit AM2901 :

- Définissez 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()*. Vous pouvez commencer par définir une boite d'aboutement de 5000 par 5000 et essayer ensuite de la réduire.
- Placez le coeur du circuit au centre de la boîte d'aboutement du chip : fonction *PlaceCentric()*.
- Définissez sur quelle face et dans quel ordre placer les plots, cela se fait à l'aide des 4 fonctions : *PadNorth()*, *PadSouth()*, *PadEast()* et *PadWest()*. Les plots devront être placés en regard des cellules auxquelles ils sont connectés.
- Visualisez 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** de la chaîne **Coriolis** 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". Dans votre cas, seules les cellules de la partie contrôle restent à placer.

- Appelez le placeur **mistral** : fonction *PlaceGlue ()*. 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

car l'outil de placement automatique place les cellules non placées en se basant sur les attirances vers les cellules déjà placées ET vers les plots.
- 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

Le routeur automatique de la chaîne **Coriolis** n'étant pas encore opérationnel, vous devez utiliser **nero**, le routeur d' **Alliance**. 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**. L'appel à **cougar** effectue une extraction au niveau catalogue.

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

4 Analyse temporelle

4.1 Extraction

Pour pouvoir utiliser **TAS**, il vous faire faire une nouvelle extraction avec **cougar**, cette fois si au niveau transistor (cette extraction vous permet également de connaitre le nombre de transistors de votre circuit). Cette extraction nécessite de cibler une technologie réelle, il faut donc initaliser la variable d'environnement **RDS_TECHNO_NAME** :

```
> export RDS_TECHNO_NAME=/users/soft/techno/labo/035/extract/prol035.rds
> export MBK_OUT_LO=al
> cougar -t amd2901_chip_r amd2901_chip_r_t
```

4.2 Chaines longues

TAS est un analyseur temporel. Il permet d'obtenir les temps de propagation minimaux et maximaux entre les points de référence (c'est à dire les **connecteurs externes** et les **points mémorisants**) d'un circuit. **TAS** travaille sans stimuli, c'est pourquoi il donne des délais **pire-cas** pour les chaînes longues.

L'environnement doit être correctement initialisé de façon à pouvoir utiliser **TAS** :

- Initialisez la variable d'environnement **ELP_TECHNO_NAME** :

```
export ELP_TECHNO_NAME=/users/soft/techno/labo/035/elp/prol035.elp
```

- Précisez le format d'entrée (**.al**) dans la variable d'environnement MBK_IN_LO.
- Mettez en place l'environnement pour l'analyse de timing :

```
source avt_env.sh
```

- Il suffit ensuite de l'ancer l'outil **TAS** :

```
tas -t am2901_chip_r_t
```

- Consulter le man de **TAS** et essayer les différentes options pour comprendre le fonctionnement de **TAS**.
- Utiliser **XTAS** qui permet d'interpréter les résultats de **TAS**. Disposant lui aussi d'un man, **XTAS** est agrémenté d'une aide en ligne.

```
xtas
```

XTAS vous permet de visualiser les chaines longues entre les points de référence du circuit et leur temps (entrées, registres, sorties). Pour les nostalgiques de l'écran vert il existe un outil similaire en ligne de commande qui s'appelle **ETAS** (voir l'aide en ligne qui s'affiche avec la commande help sous **ETAS**).

5 Chemin de test

- Effectuez l'appel à *scapin* après la synthèse sur la partie contrôle.
- Utilisez la bascule **Dpgen** avec chemin de test dans le chemin de données.
- Connectez les chemins de test contrôle et chemin de données dans le coeur.
- Ajoutez les plots de test dans le chip et connectez les avec le coeur.

6 Rapport

Vous décrirez dans votre rapport le flot de conception que vous avez suivi pour créer votre circuit ainsi que les résultats obtenus (taille du circuit, fréquence, nombre de transistors ...)

Vous répondrez également aux questions suivantes :

- Quelles différences y a-t-il entre la description de la partie contrôle et celle du chemin de données ?
- Quels avantages y a-t-il à faire des colonnes d'opérateurs pour le chemin de données ?
- Votre circuit est-il *pad limited* ou *core limited* ?
- Par où passe la chaîne longue ?
- Quelles modifications pourriez vous apporter à votre circuit de façon à améliorer la chaîne longue ? Quelles étapes de conception peuvent être amenées à être ré étudiées ?

N'oubliez pas de fournir les fichiers source ou de donner 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**.