Placement & Routage

Initialisation et utilisation de `cgt`

Pour initialisation de l'environnement, sourcer le script `coriolis2.sh`:

```
> . /soc/coriolis2/etc/coriolis2/coriolis2.sh
Switching to Coriolis 2.x (Release.Shared)
>
```

(le "`.`" est équivalent au mot clé `source`)

Exécution directe d'un script `Stratus` par `cgt`

1. Lancer la commande `cgt`.
2. Sélectionner le menu `"Tools" --> "Python Script"`, une boite de dialogue apparaît. Saisir le nom du script `Stratus`, *sans extension*, à exécuter.

Remarque: un circuit ne peut être chargé/généré qu'une seule fois en mémoire, ce qui à pour conséquence que l'étape *2* ne peut être effectuée qu'une seule fois pour un circuit donné. Lorsque vous modifiez votre placement, il est donc nécessaire de *quitter* `cgt` puis de le relancer.

Déboguage: Si votre script python contient des erreurs, `cgt` affichera une fenêtre *pop-up* indiquant que l'éxecution s'est mal passée, mais sans indications précises. L'erreur python complète est affichée sur la sortie standart dans le terminal depuis lequel `cgt` a été lancé.

Routage

Sélectionner le menu `"P&R" --> "Kite - Route"`. Explication des résultats retournés par le routeur:

Dans l'illustration çi-dessus, le routage a échoué car il reste 7 segments que le routeur n'arrive pas à router. Cela se retrouve dans le taux de routage ({{{Wire Length Completion Ratio}}} qui n'est que de `99.77%`. Le routeur indique aussi la longueur totale de fil ayant été routée `275368` et celle restante `622`, en *lambdas*. Au fur et à mesure que vous allez améliorer le placement, le taux de routage atteindra les `100%`, et la longueur de fil totale va (fortement) décroitre.

Le Chemin de données du MIPS micro-programmé

L'archive çi après vous fourni la *netlist* du chemin de données d'un MIPS micro-programme. Le fichier principal, à faire exécuter par `cgt` est `design.py`.

- MIPS.Microprogramme.tar.bz2

Dans le cas des chemins de données, le problème du placement se réduit à un problème à une dimension. Autrement dit, placer un chemin de donnée revient simplement à définir l'ordre dans lequel les opérateurs (chaque opérateur est équivalent à une colonne) sont placés. Dans notre cas, on adopte l'ordre de gauche à droite.

```
class contest ( Model ) :
  def Interface ( self ) :
    # [...]
    return

  def Netlist ( self ) :
    # [...]
    return

  def Layout (self):
    Place      ( self.acc_buff,      NOSYM, XY(0,0) )  # First column: absolute placement.
    PlaceRight ( self.acc_reg,       NOSYM )           # Second column, right of first.
    PlaceRight ( self.inv_np,        NOSYM )           # Thirst column, right of second.
    PlaceRight ( self.inv_ra,        NOSYM )           # ...
    # [...]
    return
```

Description en schéma-bloc du chemin de données. Cette description ne comprend ni tous les blocs, ni tous les signaux.

Travail Demandé

Q.1 Placement d'une colonne

Le placement de l'opérateur `x_const` ne vous est pas fourni. a vous de le compléter. Vous pouvez vous inspirer des fichiers fournis pour les autres opérateurs.

Q.2 Placement complet

Trouver un placement du chemin de données du `MIPS` minimisant la longueur totale de fil, sans provoquer de saturation.

Avant de vous lancer dans l'ordonnancement des colonnes, il vous est fortement recommandé de faire un dessin complet ou partiel de la *netlist* opérateur par opérateur (et pour un bit de donnée) pour mieux apréhender ou se trouvent les possibilités d'optimisation.

Compte Rendu

Votre fichier Python de placement du `MIPS`, afin que le correcteur puisse le ré-exécuter. Facultativement, une page décrivant les choix que vous avez fait pour le placement.