TP2 : Modélisation structurelle avec Stratus

Avant propos

Dans ce TP, nous souhaitons réaliser un générateur de circuit addaccu amélioré avec comme paramètre, entre autres, le nombre de bits.

Le circuit addaccu a trois niveaux de hiérarchie : dans **addaccu** sont instanciés trois blocs **mux**, **reg** et **add**. Les blocs **mux** et **reg** doivent être décrits par une netlist paramétrable de cellules **sxlib**, dans le langage **Stratus**. Le bloc **add** est lui aussi décrit par une netlist paramétrable en **Stratus** et instancie le bloc *full_adder.vst* créé dans les TP précédents.

Nous verrons dans ce TP que **Stratus** permet de décrire des netlists paramétrables.

1 Travail à effectuer

1.1 Familiarisation avec Stratus

- Récupérer les deux fichiers permettant de créer le bloc **mux** et les étudier :
    - La netlist en "'Stratus"' du bloc "'mux"'
    - Script pour la création de la netlist

  Ce bloc a la fonctionnalité suivante :

  ```
  si (cmd==0) alors s <= i0 sinon s <= i1
  ```

  i0, i1 et s ayant un nombre de bit paramétrable.

- Créer une instance de mux sur 4 bits. Pour ce faire, il faut exécuter le script fourni avec le bon paramètre, soit en exécutant la commande suivante :

  ```
  > python gen_mux.py -n 2
  ```

  , soit en modifiant les droits du fichier :

  ```
  > chmod u+x gen_mux.py
  > ./gen_mux.py -n 2
  ```

Si le script s'effectue sans erreur, un fichier **.vst** est normalement généré. Vous pouvez vérifier qu'il décrit bien le circuit voulu. Ce bloc peut lui-même être instancié dans une netlist grâce à la méthode **Inst**.

Note : Stratus étant issu du langage Python, il faut apporter une grande importance à l'indentation. Un bon conseil, n'utilisez pas de tabulations (ou alors configurez vos éditeurs pour qu'ils transforment automatiquement les tabulations en espaces).

1.2 Circuit addaccu

- Ecrire les blocs **add** et **reg** avec **Stratus** en utilisant exclusivement les cellules de la bibliothèque **sxlib**. Ces deux blocs prennent comme paramètre le nombre de bits. En outre, ils véri?ent que leur paramètre est compris entre 2 et 64 (ce n?est pas fait dans mux). S'inspirer des descriptions structurelles décrites la semaine précédente.
- Ecrire les deux scripts python permettant de créer les instances de l'additionneur et du registre.

- Ecrire le circuit **addaccu** avec **Stratus**. Ce circuit instancie les trois blocs précédents (**mux**, **add** et **reg**). Le circuit **addaccu** prend également comme paramètre le nombre de bits.
- Ecrire le script python permettant de créer des instances de l'addaccu.

- Ecrire un ?chier **Makefile paramétrable** permettant de produire chaque composant et le circuit addaccu en choisissant le nombre de bits.

- Générer le circuit sur 4 bits.
- Visualiser la netlist obtenue avec **xsch**.
- Tester la netlist avec **asimut** (utiliser le fichier **.pat** de la semaine précédente).

1.3 Circuit addsubaccu

Maintenant, nous souhaitons que l?addaccu puisse effectuer soit des additions, soit des soustractions. Un nouveau paramètre sera donc à apporter pour choisir la fonction à effectuer (Vous avez le choix pour le nom et les valeurs possibles de ce paramètre). Ce nouveau composant sera sur le même schéma que le précédent, avec des modifications à apporter au circuit et/ou ses composants.

- Créer un nouveau composant, appelé **addsubaccu** qui prend en compte cette nouvelle contrainte.
- Ecrire le script python permettant de créer des instances de l'addsubaccu.

- Ecrire un ?chier **Makefile** paramétrable permettant de produire chaque composant et le circuit addsubaccu.

1.4 Fonction Generate

Il n'est pas toujours très pratique d'avoir à générer avec plusieurs scripts les différents blocs d'un circuit. Le langage **Stratus** fournit donc une alternative : la fonction **Generate**.

Par exemple, pour générer une instance du multiplexeur fourni, il suffit d'ajouter la ligne suivante dans le fichier **Stratus** décrivant le circuit instanciant le multiplexeur :

```
Generate ( "mux.mux", "mux_%d" % self.n, param = { 'nbit' : self.n } )
```

Dans cette fonction, le premier argument représente la classe **Stratus** créée (format : *nom_de_fichier.nom_de_classe*), le deuxième argument est le nom de l'instance générée, le dernier argument est un dictionnaire initialisant les différents paramètres de cette classe.

- Modifier le fichier décrivant l'addsubaccu et le Makefile de façon à pouvoir créer les instances de ce circuit en n'ayant besoin que d'un script.

1.5 Description de patterns

L'outil **genpat** étudié lors de la semaine précédente fait parti de la chaîne de CAO **ALLIANCE**. **Stratus** fournit de même service pour la chaîne de CAO **Coriolis**. De plus, **Stratus** encapsule l'appel au simulateur **asimut**.

- Récupérer les deux fichiers décrivant le bloc mux avec création du fichier de patterns et simulation, et les étudier :
    - ♦ La netlist en "'Stratus"' du bloc "'mux"' avec la description des patterns
    - ♦ Script pour la création de la netlist, du fichier de patterns et du lancement du simulateur
- Créer les patterns et effectuer la simulation des deux autres blocs de la même façon.
- Une fois tous les sous blocs validés, créer les pattenrs et effectuer la simulation du bloc addsubaccu.

2 Compte rendu

Vous rédigerez un compte-rendu d'une page maximum pour ce TP. Vous expliciterez **en détail** les choix que vous avez fait pour modifier le circuit **addaccu** et/ou ses composants de façon à créer le circuit **adddsubaccu**.

Vous fournirez tous les fichiers écrits, avec les **Makefile** permettant d'effectuer la génération des deux circuits (et l'effacement des fichiers générés).