STRATUS: UN ENVIRONNEMENT DE DÉVELOPPEMENT DE CIRCUITS

Sophie Belloeil, Jean-Paul Chaput, Roselyne Chotin-Avot, Christian Masson, Habib Mehrez

Université Pierre et Marie Curie – Paris 6
Laboratoire d'Informatique de Paris 6 (LIP6)
Département SOC
4 place Jussieu
F-75252 PARIS cedex 05
E-mail : sophie.belloeil@lip6.fr

1. INTRODUCTION

Vu l'essor considérable de la technologie, les circuits deviennent de plus en plus complexes et peuvent désormais intégrer des systèmes complets sur une seule puce. Pour développer ces systèmes, des nouveaux outils et des méthodes de conception sont mis en œuvre pour réduire le temps de conception et accroître le taux de fiabilité et de vérifications des systèmes. Pour répondre à ces besoins, nous avons développé un environnement de description des architectures matérielles génériques, permettant un haut degré de réutilisabilité des modules conçus. Cet environnement fait partie de la chaîne Coriolis [1] du LIP6 et s'interface aisément avec la chaîne Alliance [2]. De plus, cet environnement est utilisé dans les enseignements du MASTER ACSI (Architectures et Conception des Systèmes Intégrés).

Nous commencerons par expliquer les choix qui ont été faits pour la spécification de cet environnement. Nous décrirons ensuite en détail ses différentes fonctionnalités dans le cadre de son utilisation comme expériences pédagogiques pour le développement d'un processeur MIPS R3000 et des projets.

2. UN LANGAGE DE DESCRIPTION DE CIRCUITS

Il existe deux catégories de langage de description de circuits : les langages généralistes tels que C , C++, Python, enrichis avec des fonctions ad-hoc, et les langages spécialisés. Les langages spécialisés comme le VHDL ont leur propre syntaxe, sont moins portables, ce qui rend les langages enrichis beaucoup plus attractifs. Ce choix avait déjà été adopté pour Genlib, le langage de la chaîne Alliance (développé au-dessus du langage C).

Un second critère de choix était de réduire le plus possible le temps du cycle de conception (compilation, édition de liens, visualisation du résultat). Notre choix s'est donc porté sur les langages interprétés.

Enfin, nous voulions créer un META langage bien adapté à la description de circuits et basé sur la chaîne Coriolis et sa base de données Hurricane écrite en C++. L'utilisation d'un langage objet était donc naturelle.

Pour toutes ces raisons, nous avons retenu le langage Python pour écrire Stratus. Python est un langage objet développé depuis 1989 par Guido van Rossum. C'est un langage portable, dynamique, qui permet une approche modulaire et orientée objet de la programmation. De plus, la syntaxe simple de Python en fait un langage idéal d'apprentissage. Notre choix a été conforté par l'apparition de langages tels que MyHDL [3], qui permet d'implanter des

concepts pour la description de circuits. Ce langage a été créé avec l'idée qu'un langage généraliste tel que Python peut être utilisé dans la conception de circuits au même titre que des langages spécialisés comme VHDL.

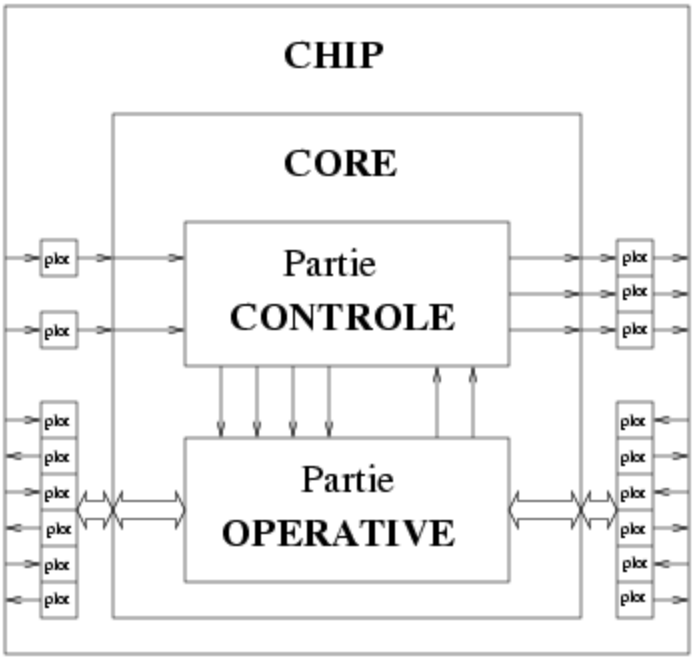
Nous avons donc développé un langage qui est un ensemble de classes, méthodes et fonctions Python dédié à la création de circuits. Du point de vue de l'utilisateur, Stratus est un langage de description de circuits qui utilise toutes les fonctionnalités de Python de façon à gérer des objets VLSI. La caractéristique principale de Stratus est de fournir différents niveaux de conception en fonction de ce que veut faire l'utilisateur : bas niveau de description pour créer un circuit, et haut niveau de description pour, par exemple, faire de la simulation.

3. DÉVELOPPEMENT DU PROCESSEUR MIPS R3000

L'unité d'enseignement « Réalisation VLSI du processeur MIPS R3000 » du MASTER ACSI est un module très pratique organisé autour de l'implantation VLSI d'un micro-processeur RISC 32 bits. Le projet est réalisé en 3 semaines par des équipes de 5 étudiants. Le cours décrit en détail l'architecture du processeur Mips R3000 et en présente une réalisation micro-programmée. L'objectif du cours est de fournir aux étudiants les éléments leur permettant de concevoir en 3 semaines le processeur Mips R3000 sur une puce.

3.1. Introduction à l'architecture du Mips

Nous décomposons le Mips R3000 en 2 blocs : la partie opérative (chemin de donnée), commandée par la logique de la partie contrôle.



Le chemin de données contient les parties régulières du Mips, 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.

Le chemin de données est décrit par l'étudiant en Stratus et la description de la partie de contrôle est obtenue par la synthèse de sa description comportementale.

3.2. Travail sur le chemin de données

Dans le langage Stratus, chaque cellule est définie comme une classe, héritant de la classe `Model`, dont chaque méthode décrit une vue. On trouvera par exemple les méthodes `Interface`, `Netlist` (vue structurelle) et `Layout` (vue physique).

```
class cellule ( Model ) :
        def Interface ( self ) :
            ...
        def Netlist ( self ) :
            ...
        def Layout ( self ) :
            ...
```

Par héritage de la classe `Model`, chaque cellule peut utiliser les deux méthodes suivantes :

- `View` : ouverture d'un éditeur pour voir la cellule
- `Save` : sauvegarde sur disque

Notons qu'il est de ce fait très aisé pour un utilisateur de rajouter des fonctionnalités à Stratus sans avoir à modifier ce dernier : il suffit de créer une classe dérivant de la classe `Model`, de lui ajouter les fonctionnalités désirées, et cette classe sera la nouvelle classe de référence pour les cellules.

3.2.1 Vue structurelle

Dans la vue structurelle, on effectue l'instanciation et la connectique des cellules présentes dans la cellule.

Par exemple, l'instanciation d'une cellule logique ET à deux entrées avec sa connectique s'effectue comme suit :

```
instance = Inst ( 'a2_x2'
                , map = { 'i0'  : net_in0
                        , 'i1'  : net_in1
                        , 'q'   : net_output
                        , 'vdd' : net_vdd
                        , 'vss' : net_vss
                        }
                )
```

où `a2_x2` est le nom de la cellule de la bibliothèque cible, et `map` décrit la connectique entre les ports de cette cellule (`i0`, `i1` ...) et les signaux de la cellule que l'on est en train de décrire (`net_in0`, `net_in1` ...).

Dans le cas du Mips R3000, nous avons recours à la bibliothèque de générateurs DPGEN (une bibliothèque d'opérateurs réguliers pré-placés) de façon à pouvoir générer des colonnes de 32 bits. Il faut tout d'abord générer (méthode `Generate`) l'opérateur avec les paramètres appropriés (dictionnaire `param`) de façon à pouvoir ensuite l'instancier.

Par exemple, pour une porte ET avec les signaux sur 32 bits, on génère le modèle comme suit :

```
Generate ( 'DpgenAnd2', 'and2_32bits'
         , param = { 'nbit' : 32 } )
```

où `DpgenAnd2` est le modèle du générateur et `and2_32bits`, le nom du modèle généré. La clé `'nbit'` du dictionnaire `param` indique que la largeur est fixée à 32 bits.

Une fois le générateur créé, il suffit alors de l'instancier, comme précédemment, avec le nom du modèle généré (dans l'exemple : `and2_32bits`).

3.2.2 Vue physique

Il reste à effectuer un placement des opérateurs réguliers. Ce placement se résume à un ordonnancement des colonnes entre elles. De la qualité de cet ordonnancement dépendra la qualité et même la faisabilité du routage.

Le placement en Stratus se fait de deux façons :

- Soit avec des coordonnées absolues :

```
Place ( instance, NOSYM, XY ( 0, 0 ) )
```

- Soit par technique d'aboutement par rapport à une instance de référence (typiquement, la dernière instance à avoir été placée) :

```
PlaceTop ( instance, SYM_Y )
```

Notons que la notion de symétrie est très importante car c'est elle qui permet la validation des règles de dessin, notamment en permettant d'abouter correctement les rails d'alimentation.

3.3. Travail sur le cœur

Le cœur du Mips contient le chemin de données et la partie contrôle. Il faut donc instancier ces deux cellules.

D'un point de vue physique, on utilise, d'une part le pré-placement du chemin de données, et d'autre part le placement automatique obtenu par les outils de la chaîne Coriolis pour la partie contrôle.

Il faut ensuite effectuer le placement des rails d'alimentation, avec les fonctions `AlimVerticalRail` et `AlimHorizontalRail` (ces fonctions ont comme arguments les coordonnées auxquelles l'utilisateur veut placer les rails). Il faut enfin effectuer le placement des connecteurs, avec la fonction `AlimConnectors`.

3.4. Travail sur le circuit complet

Le circuit contient le cœur et tous les plots. Les différentes étapes du flot de conception sont les suivantes :

- Placer le cœur au milieu de la boîte d'aboutement du circuit (il faut donc choisir une taille au préalable qui sera ensuite diminuée autant que possible)
- Répartir les plots sur les quatre faces du circuit, avec les fonctions `PadNorth`, `PadSouth`, `PadEast` et `PadWest`
- Router les alimentations, avec la fonction `PowerRing`
- Effectuer le placement automatique de la logique irrégulière (qui se base sur l'emplacement des plots et les attirances entre cellules pour effectuer un placement optimal) avec la fonction `PlaceGlue`
- Router les signaux d'horloge avec la fonction `RouteCk`
- Router les signaux restants avec la fonction `Route`

3.5. Synthèse

Cet enseignement permet de familiariser les étudiants avec la plupart des étapes nécessaires pour obtenir la vue en masque d'un circuit réalisé en cellules pré-caractérisées avec pré-placement des parties régulières.

Stratus fournit toutes les fonctions nécessaires à la réalisation de ce circuit. L'étudiant a donc une approche claire et simple des différentes étapes de travail.

4. TRAVAUX RÉALISÉS DANS LE CADRE DE PROJETS

Un autre aspect de Stratus est la capacité d'effectuer une description haut niveau d'un circuit. Basée sur ce principe, une bibliothèque d'opérateurs arithmétiques est en train d'être développée au sein du laboratoire. Grâce à la modularité de Stratus, il est facile de lui rajouter des fonctionnalités sans avoir à modifier le cœur de l'outil.

Dans le cadre de l'unité d'enseignement « Projet ACSI », des étudiants ont rajouté des fonctionnalités à Stratus et développé de nouveaux opérateurs .

4.1. Fonctionnalités fournies par la description haut niveau

4.1.1. Surcharge

Nous utilisons un des mécanismes des langages objets pour la description haut niveau : la surcharge de méthode. Cela permet d'écrire des affectations telles que :

```
net_output <= net_in1 & net_in2
```

Une telle affectation consiste à redéfinir le symbole & pour qu'il appelle l'opérateur ET approprié.

Les surcharges fournies sont les fonctions booléennes : ET (&), OU (|), OU-EXCLUSIF (^) et, NON (~), ainsi que les opérations arithmétiques : l'addition (+), la soustraction (-), multiplication (*) et la division (/).

4.1.2 Méthodes fournies

Différentes méthodes sont également fournies dans Stratus pour faciliter le travail de description :

- Méhode `Mux` : Cette méthode permet l'instanciation d 'un opérateur de choix multiples

```
out <= cmd.Mux ( { "0-2"   : in1
                 , "3,5-7" : in2
                 , "4"     : in3
                   } )
```

Ici, on instancie un multiplexeur dont la sortie `out` prend une des entrées `in1`, `in2`, `in3`, en fonction de la valeur de la commande `cmd`.

- Méthode `Shift` : Cette méthode permet l'instanciation d'un opérateur de décalage en choisissant le sens et le type du décalage à effectuer

```
out <= cmd.Shift ( in, "right", "logical" )
```

Ici, on instancie un décaleur dont la sortie `out` est le résultat d'un décalage logique à droite de l'entrée `in` de la valeur du signal `cmd`.

4.1.3 Bibliothèque virtuelle

Un mécanisme très pratique fourni par Stratus est le mécanisme de bibliothèque virtuelle. En effet, ce mécanisme permet de modifier la bibliothèque cible d'un circuit sans avoir à réécrire ce dernier.

De plus, changer de bibliothèque virtuelle, ou ajouter une nouvelle cellule, est fait de façon aisée en écrivant un fichier de correspondance. Ce fichier est écrit en langage Xml, et décrit le lien entre la cellule virtuelle et la cellule de la bibliothèque cible en termes de nom de modèle et d'interface.

4.2 Réalisations effectuées en projet

Dans le cadre de l'unité d'enseignement « Projet ACSI », Stratus a été utilisé pour la conception de circuits et pour l'ajout de fonctionnalités à l'outil. Cette

UE (unité d'enseignement) permet aux étudiants de se confronter à un problème "non académique" dont la solution nécessite une réalisation matérielle ou logicielle de taille significative : environ deux demi-journées de travail par semaine pendant trois mois. Le but de cette UE est de permettre à chaque étudiant de démontrer qu'il a correctement assimilé les connaissances dispensées en cours et qu'il est capable de les mettre en oeuvre pour aboutir à la solution d'un problème.

4.2.1 Conception de circuits

Différents circuits ont été réalisés dans le cadre des projets :

- Un opérateur de DCT
- Un filtre numérique
- Un multiplieur multi-formats
- Un opérateur de division/racine carrée

La description haut niveau fournie par Stratus, notamment la surcharge des opérateurs et les différentes méthodes fournies, ont grandement facilité le travail des concepteurs.

4.2.2 Ajout de fonctionnalité

Deux autres projets ont été effectués dans le but d'ajouter de nouvelles fonctionnalités à l'outil :

- Un algorithme de nettoyage de la description structurelle : suppression des signaux non connectés
- Un procédé de génération automatique de vecteurs de tests et de simulation

5- CONCLUSION

Nous avons présenté un nouvel environnement de développement de circuits. Un nouveau langage a été créé ayant la particularité de permettre une description haut niveau de la vue structurelle et également une description bas niveau de la vue physique. Cela permet l'apprentissage d'un seul langage, et l'utilisation d'une partie des différentes fonctionnalités fournies en fonction des besoins.

Le langage Stratus a été créé au-dessus du langage Python, qui, de par sa syntaxe simple, est parfaitement adapté pour l'apprentissage de la conception de circuits.

De plus, d'un point de vue pédagogique, une approche complète des étapes nécessaires à la conception de la vue en masques a été réalisée : des traitements spécifiques de routage pour les alimentations et les horloges ont été présentés de façon à familiariser les étudiants avec ces procédés.

Références :

[1] Christophe Alexandre and Hugo Clement and Jean-Paul Chaput and Marek Sroka and Christian Masson and Remy Escassut, « TSUNAMI: An Integrated Timing-Driven Place And Route Research Platform », DATE '05: Proceedings of the conference on Design, Automation and Test in Europe (NB : Tsunami est l'ancien nom de la chaîne Coriolis)

[2] Alain Greiner and Frédéric Pétrot, « Using C to write portable CMOS VLSI module generators », EURO-DAC '94: Proceedings of the conference on European design automation

[3] Jan Delacuwe « MyHDL: a python-based hardware description language », journal Linux n° 127, 2004