Présentation de notre simulateur

Nicolas Blanchard, Axel Davy et Marc Heinrich

22 octobre 2012

1 Comment exécuter notre simulateur

Pour executer le programme, simplement lancer ocamlbuild -use-menhir main.native, ce qui génère l'executable main.native que l'on peut lancer avec les options désirées.

2 Fonctionnement

Le simulateur est articulé autour d'un main qui gère la liste des options et lance dans l'ordre les différentes étapes indépendantes :

- Lecture de Netlist (par le module fourni)
- Transformation par le scheduler (pas avec le même résultat que celui fourni sur le traitement des registres)
- conversion de ce résultat dans un format (mprogram) permettant d'améliorer les performances ultérieures
- Execution de ce mprogram.

3 Options disponibles

syntaxe : main (options) filename

- n *entier* : correspond au nombre d'étapes à executer (par défaut égal à -1 ce qui est équivalent à illimité)
- noprint : n'imprime pas le résultat du scheduler
- debug : lance en mode débug (permet de faire du pas à pas)
- clock *flottant* : permet de régler la fréquence d'horloge simulée, pas encore implémentée)
- verbose : imprime à l'écran les sorties à chaque pas

4 Difficultés rencontrées

Après la première séance du TD, nous avions une fonction scheduler qui permettait de faire un tri topologique et de mettre ainsi dans l'ordre les instructions d'un fichier netlist source. Mais nous avons eu beaucoup de mal à avoir une fonction qui marche aussi pour les registres. En effet il faut que la fonction ne détecte pas de cycle combinatoire quand il y a une boucle causée par un registre. De plus si deux registres sont chacuns connectés à la sortie de l'autre, on ne peut en exécuter un avant l'autre. Nous avons donc décidés d'enlever les liaisons des

registres dans le graphe du tri topologique, ainsi le tri se fait sans considérer les registres, se qui coupe les éventuels cycles combinatoire qui auraient été répérés à cause d'une boucle de registre. Nous avons aussi fait en sorte que les registres donnent leurs sorties au début de l'éxécution et capturent leur entrées à la fin de l'éxécution. Pour cela nous avons mis les registres dans un liste pour donner leur sortie au début de l'exécution et nous les avons mis à la fin de la liste à exécuter pour qu'ils puissent capturer leur entrée. Comme le type "programme" était incomplet à notre goût on a crée un nouveau type Mprogramme : On souhaitait une liste des cas à traiter à part (comme les registres) et nous voulions que les équations soient composées de clés indiquant la valeur de leur arguments dans un tableau.