Réalisation du simulateur de netlists

Baptiste Lefebvre, Li-yao Xia, Antonin Delpeuch

18 janvier 2013

1 Algorithme

Nous avons implémenté l'algorithme suggéré au cours du TP1. La netlist représentée sous forme d'arbre de syntaxe abstraite définit un graphe des dépendances : un arc joint a à b si connaître la valeur de a est nécessaire pour calculer celle de b. On réalise un tri topologique pour déterminer dans quel ordre il faut calculer les valeurs des variables de sorte qu'on n'ait besoin que de valeurs déjà calculées à chaque étape.

Dans les exemples suivants, on a représenté les variables d'entrée dans le graphe, en pratique elles n'ont pas besoin d'y figurer.

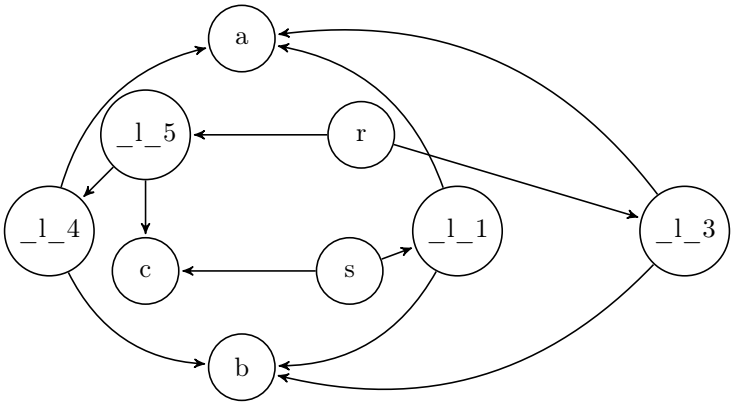

FIGURE 1 – Le graphe obtenu à partir de la netlist `fulladder.net`

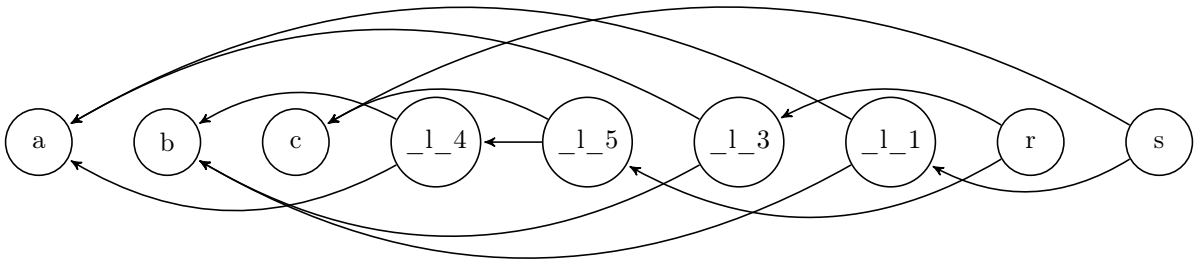

FIGURE 2 – Le même graphe après tri topologique

2 Implémentation

Nous avons réutilisé le parser et le lexer fournis dans le TP1, ainsi que les types définis dans `netlist_ast.ml`. Les variables sont converties en entiers afin de faciliter la simulation (leurs valeurs

étant stockées dans un tableau), au détriment d'une écriture plus complexe pour faire le lien avec leur représentation initiale. Nous avons fait le choix de ne pas interpréter chaque équation à chaque cycle, mais de les interpréter lors d'une phase initiale en les gelant en fonctions du type `unit -> unit` et de les lier aux variables correspondantes. Choix justifié par le nombre d'appels que présenterait une fonction globale aux tests mutliples. Nous avons écrit le reste, en séparant le code (dans la mesure du possible) en modules indépendants :

- `Graph` : représentation des graphes et tri topologique ;
- `Netlist_proxy` : traduction des noms de variables en entiers, extraction des dépendances ;
- `Tape` : gestion des opérations `RAM` et `ROM` ;
- `Scheduler` : jointure entre `Graph` et `Netlist_proxy`.

3 Problèmes rencontrés

Notre programme simule effectivement les circuits simples comportant des registres.

- L'ajout de registres et de nappes de fils n'a pas posé de grosse difficulté.
- Nous avons aussi tenté de prendre en compte les RAM et ROM, apparement en avance par rapport au calendrier, mais nous n'avons pas compris leur fonctionnement dans la netlist ;
- Le fichier ram.net pose problème : $\_l\_14 = CONCAT(\_l\_9\_19, *)$, $\_l\_9\_19 = AND(\_l\_7\_21, *)$, $\_l\_7\_21 = SELECT(0, \_l\_14)$, où $\_l\_7\_21$ et $\_l\_9\_19$ semblent au final correspondre à un même fil dans la nappe $\_l\_14$.

4 Améliorations possibles

- Support des opérations RAM et ROM ;
- Mode de traitement par lot (pour calculer d'un seul coup les sorties après n cycles sur un jeu d'entrées) ;
- Lecture des données plus propre (elle est en boucle infinie pour l'instant) ;
- Mieux séparer les fonctions dans des modules différents ;
- Optimisation de notre programme en vitesse d'exécution. En particulier l'interprétation initiale des fonctions pourrait être poussée plus encore avant le gel.