Interpréteur netlist

Florentin Guth

27 octobre 2016

1 Fonctionnement général

L'interpréteur prend en entrée un nom de fichier contenant la *net-list* du circuit et, de manière optionnelle, le nombre de cycles sur lesquels effectuer la simulation. On utilise alors le module `Netlist` pour obtenir la liste d'équations du programme, que l'on trie topologiquement. Dans le tri topologique, les instructions `REG` sont ignorées car elles seront exécutées à la fin du cycle, et l'instruction `RAM` ne dépend que de `read_addr` puisque l'écriture est également exécutée à la fin du cycle (pour éviter les boucles combinatoires).

Le programme est alors évalué cycle par cycle. La structure d'un cycle se décompose comme suit :

— On demande les `inputs` à l'utilisateur (en refusant toute valeur non conforme au type attendu de l'entrée).

— On évalue chaque équation dans l'ordre. Lorsqu'une instruction `REG` ou `RAM` est rencontrée, on ajoute à la liste des tâches la variable qui devra être mise à jour ainsi que les paramètres utiles pour la calculer (sans chercher à les évaluer, leur valeur n'est peut-être pas encore calculée pour ce cycle).

— On affiche la valeur de toutes les entrées.

— On effectue chaque tâche « simultanément »(cf. section sur l'environnement) afin de se préparer pour le cycle suivant.

2 Gestion de la `RAM`

La `RAM` est représentée par un tableau de taille $2^{\texttt{addr\_size}}$ contenant des tableaux de `word_size` bits. On vérifie au préalable que la *net-list* effectue une utilisation consistante de la `RAM` ou `ROM`, i.e. que la taille des adresses et des mots est la même pour chacune des instructions concernées. Les adresses reçues sont converties en entier afin d'accéder à la valeur demandée. Au début de l'évaluation, la `RAM` est initialisée avec des 0.

3 Gestion de l'environnement

L'environnement est représenté par une table de hachage `env` dont les clés sont de type `string` (identificateurs) et qui contient des valeurs de type `Netlist_ast.value`. Celle-ci est mise à jour à chaque équation évaluée, et est utilisée en cours de calcul pour obtenir les valeurs des variables intermédiaires déjà calculées. Lorsqu'on cherche à obtenir la valeur d'une variable qui n'est pas présente dans la table, ce qui correspond à la première itération d'un registre (les autres cas ont été écartés par la détection des cycles combinatoires ou par le compilateur MiniJazz), on renvoie 0 ou un tableau de 0 suivant le type de la variable considérée.

Pour traiter les tâches « simultanément »(i.e. les instructions `x = REG y` et `y = REG x` doivent échanger les valeurs de `x` et `y` à la fin du cycle), on utilise un deuxième environnement `env'` :

— A chaque fin de cycle, on commence par vider `env'`.

— Lors de l'évaluation de chaque tâche en fin de cycle, on stocke les valeurs calculées dans `env'`, mais en utilisant les valeurs des variables contenues dans `env`.

— Une fois toutes les tâches effectuées, on recopie les valeurs que contient `env'` dans `env`.