Simulateur de netlist

Lucas Pesenti

Le présent projet est un simulateur de netlist minimal. Il prend en compte les différents opérateurs, les registres, les nappes de fils, les RAM et les ROM. Il est rédigé intégralement en langage OCaml.

La netlist est simulée cycle par cycle en triant dans un certain ordre ses instructions. L'environnement associant à chaque variable sa valeur courante est ainsi modifié à chaque instruction. L'ordre choisi permet d'utiliser un environnement unique et de ne modifier la valeur d'une variable dans celui-ci qu'au plus une fois par cycle. Pour cela, on sépare les instructions en deux parties indépendantes :

- Les instructions temporelles : ce sont les écritures en RAM, et les registres. Le tri fait en sorte que chaque variable est modifiée après toutes celles qui en dépendent. Les écritures en RAM dépendent de **write_enable**, de **write_address** et de **data**. Les registres dépendent de leur unique paramètre. On ne considère que les arêtes qui pointent vers des instructions temporelles qui ne sont pas une écriture en RAM (car la valeur retour d'une instruction RAM ne dépend que de la phase de lecture). On ne fait pas attention aux cycles combinatoires (on ignore tous les arcs de remontée dans la représentation de Tarjan du parcours en profondeur). De cette manière, on peut compiler les netlists de la forme :

  ```
  u = REG(v)
  v = REG(u)
  ```

  (qui, même s'ils elles n'ont pas vraiment d'intérêt, me paraissent constituer un programme netlist valide)
- Les instructions intemporelles : toutes les autres (dont les lectures en RAM et en ROM). À l'inverse du premier type, chaque variable est modifiée avant toutes celles qui en dépendent. Les dépendances sont naturelles pour la plupart des opérations, et la lecture en RAM/ROM dépend uniquement de **read_addr**. Le graphe des dépendances comporte un cycle si, et seulement si le circuit a un cycle combinatoire.

L'ordre global des instructions est obtenu en concaténant la liste triée des instructions intemporelles avec la liste triée des instructions temporelles.

Si jamais il y avait ultérieurement des difficultés de performance, l'ordre ainsi choisi donnerait une compilation naturelle vers un langage impératif tel que le C, sans jamais utiliser la moindre variable intermédiaire. Étant donné qu'on ne peut encore savoir si les performances du simulateur sont un point crucial pour le fonctionnement du projet final, on a ici choisi de garder le code simple, de manière à minimiser ses erreurs.

De même, on a fait attention à garder une complexité temporelle quasi-linéaire en le nombre de portes, et ce dans la totalité du code du simulateur. Comparé à d'autres projets de compilateurs de netlist, celui-ci présente d'ailleurs des performances honorables en pratique sur des grandes entrées générées aléatoirement.

Les principales difficultés rencontrées proviennent de l'absence de spécifications claires des netlists. Il a notamment fallu expérimenter sur les codes compilés par Minijazz, mais cela n'a pas permis de tirer des conclusions certaines. En particulier :

- **MUX a b c** est interprété comme **if c then b else a**.
- Les écritures en RAM se font à la fin du cycle, et les lectures en cours de cycle.
- On utilise un fichier externe pour initialiser la ROM (s'il n'est pas spécifié, on initialise tout à zéro).
- L'adressage de la RAM est géré comme dans les langages de bas niveau. On peut lire et écrire à une adresse non alignée sur un multiple de la taille du mot. Les données sont interprétées en big-endian.
- En cas d'utilisation fourbe de la RAM (par exemple, deux écritures à la même adresse dans le même cycle), le comportement du simulateur est indéfini.

— Le simulateur de netlist est laxiste et laisse à l'utilisateur le soin de gérer la mémoire. En particulier, la RAM et la ROM sont allouées statiquement dans le simulateur au début, de manière à en garantir l'efficacité, donc il faudra faire en sorte à ne pas prendre **addr_size** trop grand.
— On a choisi d'étendre les opérateurs binaires comme le ou bit à bit à des nappes de fils de tailles différentes, en interprétant dans ce cas les variables de la forme **VBit** comme des nappes de 1 fil.
— Les erreurs sont gérées de manière élémentaire, et les messages affichés sont très inexpressifs.

L'utilisation du simulateur est détaillée dans le fichier README.