ORDONNANCEMENT STATIQUE VERSUS PILOTAGE ÉVENEMENTIEL

R. BUCHMANN, F. PETROT, A. GREINER

Département ASIM, laboratoire LIP6, Université Pierre et Marie CURIE
4, place Jussieu, 75252 Paris Cedex 05
buchmann.richard@lip6.fr
Frederic.Petrot@lip6.fr
Alain.Greiner@lip6.fr

RÉSUMÉ : Les systèmes intégrés, de plus en plus complexes, nécessitent, pour un temps de developpement identique, des composants réutilisables ainsi qu'un simulateur performant facilitant l'exploration architecturale. Un niveau de simulation intéressant est le niveau cycle, qui permet de modéliser exactement le comportement d'un composant avec des performances acceptables. L'intérêt sucité dans le domaine de la simulation par SystemC nous a poussé à l'utiliser comme outil de simulation cycle, et à le comparer au simulateur CASS[1] développé dans notre équipe.

I – INTRODUCTION

Les systèmes que l'on cherche à modéliser comportent un petit nombre de composants (une centaine au plus). Ils sont principalement synchrones et potentiellement multi-horloges. Un exemple est presenté sur la figure 1.

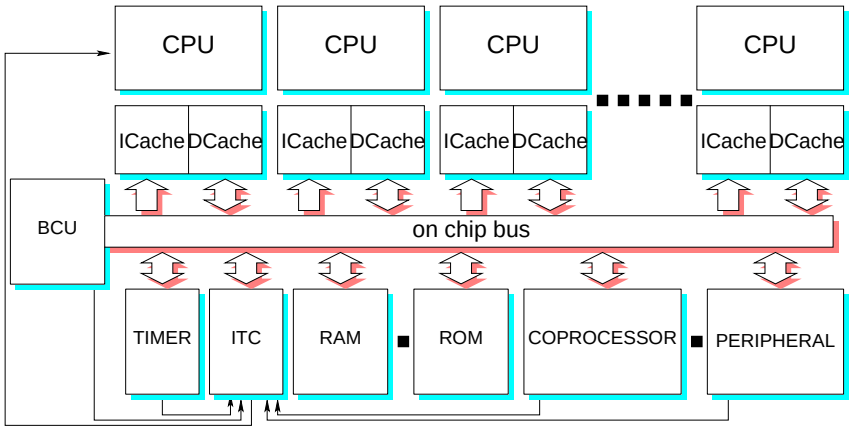

FIG. 1 – Un système intégré typique

L'objet de ce travail est de comparer l'efficacité de la simulation de modèles décrits de manière identique sous SystemC et CASS. Nous décrivons d'abord les contraintes de modélisation nécessaires à une simulation cycle sans propagation d'évènements. Ensuite, nous présentons comment on peut implementer ces mécanismes d'une

part sous CASS et d'autre part sous SystemC. Ces deux simulateurs prennent en entrée des modèles C strictement identiques. Finalement, nous comparons les performances obtenues.

II – MODÉLISATIONS DES COMPOSANTS EN AUTOMATES

Un système complet peut être modélisé par un ensemble d'automates à états finis synchrones, et communicants entre eux par des signaux. À chaque composant, on fait correspondre un automate[2].

Afin de permettre une simulation cycle, on partitionne chaque automate en deux parties, comme présentées figure 2.

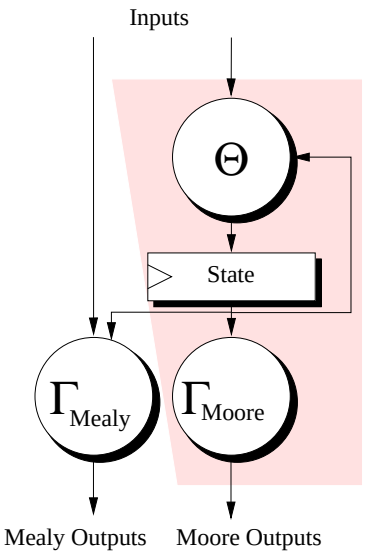

FIG. 2 – Partition d'un automate en deux parties

La fonction de transition et la fonction de génération de Moore ne doivent être évaluées que lorsque l'horloge subit un front de montée. En conséquence elles ne seront évaluées qu'une fois par cycle. On regroupe ces fonctions dans une unique fonction dite *Sequentielle*.

La fonction de génération de Mealy possède une liste de sensibilité qui exclu l'horloge. Elle doit être évaluée jusqu'à stabilité de ses entrées, ce qui peut nécessiter de nombreuses itérations pour un cycle donné. Cette fonction est dite *Combinatoire*.

III – CASS

Pour illustrer l'intérêt de ce niveau de modélisation, nous avons développé, à partir de 1996, CASS (Cycle Accurate System Simulator) un simulateur au cycle et au bit près.

CASS exploite les propriétés du modèle proposé de la façon suivante :

- les fonctions *Sequentielles* peuvent être évaluées dans n'importe quel ordre ;
- les fonctions *Combinatoires* se présentent en deux cas distincts :
  - si la fonction fait partie d'une boucle combinatoire, elle doit être évaluée itérativement dans la boucle (algorithme de relaxation) ;
  - sinon, elle peut être ordonnancée statiquement ;

Pour identifier toutes les boucles combinatoires, nous dessinons un graphe où chaque noeud est la fonction *combinatoire* d'un automate de Mealy et chaque arc symbolise la présence d'au moins un signal reliant un de ses ports d'entrée à un autre automate de Mealy. On effectue la condensation du graphe par extraction de ses composantes fortement connexes. Le graphe résultat est un graphe dirigé acyclique que l'on peut trier topologiquement[3]. Il suffit ensuite d'évaluer la (si pas de boucle combinatoire) ou les (si présence de boucles combinatoires) parties *combinatoires* correspondant au nœud du graphe dans l'ordre croissant de l'étiquetage.

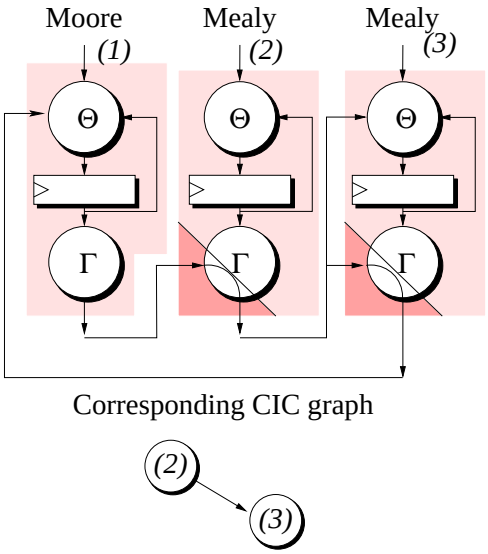

FIG. 3 – Système ordonnançable statiquement

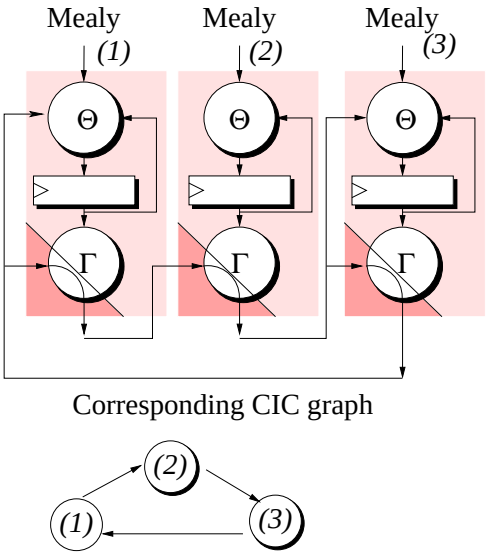

FIG. 4 – Système nécessitant la relaxation

La figure 3 présente un cas où l'ordonnancement statique est possible et la figure 4 illustre un cas ou l'on doit recourir à la relaxation.

IV – SystemC

SystemC est un simulateur de modèles C++. Contrairement à CASS, celui-ci vise plusieurs niveaux d'abstraction, du haut niveau jusqu'au RTL. Le moteur de simulation de SystemC utilise un algorithme event-driven comme un simulateur RTL classique. Une liste de sensibilité aux évenements est associée à chaque composant.

Nous souhaitons contraindre le scheduling event-driven de SystemC afin de le faire fonctionner comme un simulateur par cycle. Nous devons alors décomposer chaque cycle en trois étapes :

- Execution des fonctions *combinatoires* jusqu'à stabilité (Mealy) ;
- Execution unique de chaque fonction *séquentielle* (Moore) ;
- Copie des signaux dans les registres (barrière de synchronisation).

Il suffit d'établir une liste de sensitibilité particulière pour obtenir ce type de fonctionnement. Nous choisissons arbitrairement :

- Au front descendant :
  - Execution des fonctions *combinatoires* jusqu'à stabilité ;
- Au front montant :
  - Execution unique de chaque fonction *séquentielle* ;
  - Copie des signaux dans les registres (la barrière de synchronisation est gérée automatiquement par SystemC : l'évenement du front montant reveille au même instant toutes les fonctions sequentielles ; les registres sont mis à jour au delta-cycle suivant).

Voici un exemple d'implémentation :

```
SC_METHOD (Sequential);
sensitive_pos << CLK;
SC_METHOD (Combinational);
sensitive_neg << CLK;
sensitive << (liste de signaux de Mealy);
```

Avec SystemC, nous obtenons ainsi un ordonnancement dynamique de nos modèles CASS initialement prévus pour la simulation cycle par cycle.

V – RÉSULTATS

Après avoir mis en place le système proposé, nous avons simuler différents programmes sur une architecture simple comprenant :

- 1 CPU : MIPS R3000
- 1 BUS : PIBUS
- 4 blocs de RAM
- 1 Terminal

Système/Simulateur	CASS	SystemC 2.0	CASS/SystemC 2.0
PGCD	356.693 cycles/s	50.633 cycles/s	7,0
LEE2	1.090.068 cycle/s	139.664 cycles/s	7,8
MEMTEST	1.160.000 cycle/s	136.000 cycles/s	8,5

Les performances sont, au minimum, de l'ordre d'un facteur 7 en faveur de CASS. Ceci est dû à l'échéancier, aux propagations d'évenements rendus inutiles, et aux temps d'accès aux ports d'entrées/sorties. Le gain supplémentaire induit par le seul ordonnancement, est plus difficile à apprécier car dépendant des implémentations. Les écarts de performance augmentent avec l'importance de l'ordonnacement statique. Malheureusement, faute de temps, la serie de tests n'a pu être terminée pour mettre suffisament en valeur l'impact de l'ordonnancement statique.

VI – CONCLUSION

L'augmentation de la taille des systèmes et de leurs composants devient un véritable problème pour le concepteur. Aujourd'hui, celui-ci recherche de la part du simulateur une rapidité accrue et recontre un problème : l'algorithme de simulation. En effet, l'évaluation de l'ensemble des composants jusqu'à stabilisation des signaux peut rapidement devenir un gouffre avec un ordonnancement dynamique. Etablir un ordonnancement intelligent necessite d'imposer certaines contraintes quant au choix de l'implémentation. Ces choix sont succeptibles d'améliorer considérablement les performances globales de simulation mais cela a un prix... Ces contraintes sont :

- Dissociation de la partie de MOORE et de MEALY
- Génération puis compilation de l'ordonnancement

Le seul fait de s'affranchir de la gestion d'un ordonnanceur de type event-driven permet, vu les quelques exemples que nous avons developpé, d'obtenir un facteur 7.

RÉFÉRENCES

[1] Denis Hommais, Une méthode d'évaluation et de synthès des communications dans les systèmes intégrés matériel-logiciel. Thèse de doctorat de l'université Paris 6, 2001.

[2] Denis Hommais, Frédéric Pétrot, and Alain Greiner. Un environnement de simulation pour les systèmes embarqués. In Bernard Courtois, editor, *Actes du premier colloque CAO de circuits et systèmes*, pages 66–69, Grenoble, France, January 1997.

[3] C. Berge. *Graphes et hypergraphes*, chapter 2, page 26. Dunod, 2nd edition, 1973.