

# Modélisation Transactionnelle des Systèmes sur Puces en SystemC Ensimag 3A — filière SLE Grenoble-INP Notions Avancé en SystemC/TLM

Matthieu Moy

Matthieu.Moy@imag.fr

2011-2012



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 1 / 31 >

## Planning approximatif des séances

- 1 Introduction : les systèmes sur puce
- 2 Introduction : modélisation au niveau transactionnel (TLM)
- 3 Introduction au C++
- 4 Présentation de SystemC, éléments de base
- 5 Communications haut-niveau en SystemC
- 6 Modélisation TLM en SystemC
- 7 TP1 : Première plateforme SystemC/TLM
- 8 Utilisations des plateformes TLM
- 9 TP2 (1/2) : Utilisation de modules existants (affichage)
- 10 TP2 (2/2) : Utilisation de modules existants (affichage)
- 11 **Notions Avancé en SystemC/TLM**
- 12 TP3 (1/3) : Intégration du logiciel embarqué
- 13 TP3 (2/3) : Intégration du logiciel embarqué
- 14 TP3 (3/3) : Intégration du logiciel embarqué
- 15 Intervenant extérieur : Laurent Maillet-Contoz (ST)
- 16 Perspectives et conclusion



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 2 / 31 >

## Sommaire

- 1 Bug, or not Bug ?
- 2 Optimisations de Performances
- 3 Questions de Sémantique
- 4 Évaluation de consommation



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 3 / 31 >

## What is a bug ?

- 1 Launch a SystemC/TLM simulation
- 2 It produces incorrect result

### Question



Good news or bad news ?



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 5 / 31 >

## A Few Kinds of Model/Simulator's Bugs

- Hardware bug :
  - 1 **Simulator** design bug, corresponding to a real hardware bug 😊
  - 2 Simulator programming error, irrelevant in real hardware 😞
- **Software** bug : software doesn't run properly on simulator
  - 1 Because software is buggy ? 😊
  - 2 Because simulator is not faithful ? 😞



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 6 / 31 >

## What Can we Expect from the Model ?

Software **runs** on TLM  $\Rightarrow$  Software **runs** on real chip



Software **doesn't run** on real chip  $\Rightarrow$  Software **doesn't run** on TLM



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 7 / 31 >

## Another Kind of "Bug"

- Up to now :
  - Hardware Model doesn't work
  - Software doesn't run on hardware model
- What about :
  - Software **does** run on hardware model, but **not** on real chip

Hiding bugs  $\neq$  Fixing bugs



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 8 / 31 >

## Sets of Behaviors, Faithfulness

- An ideal TLM model ...
  - Should exhibit **all** behaviors of the real system
  - May exhibit **more** behaviors than the real system
  - Should not exhibit "**too many**" unrealistic behaviors



Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2011-2012

< 9 / 31 >

## (Counter) Examples

TLM should exhibit **all** behaviors of the real system (1/2)

### Software CPU1

```
compute_img(&buf);  
img_computed = 1;
```

### Software CPU2

```
while (img_computed != 1)  
    continue;  
read_img(&buf);
```

### Question



Are all interleavings correct ?



## (Counter) Examples

TLM should exhibit **all** behaviors of the real system (2/2)

### (Incorrect) Software CPU1

```
img_computed = 1;  
compute_img(&buf);
```

### Software CPU2

```
while (img_computed != 1)  
    continue;  
read_img(&buf);
```

### Question



Are all interleavings correct ?

### Question



Will we see the bug in a simulation ?



## (Counter) Examples

TLM may exhibit **more** behaviors than the real system

### Software CPU1

```
compute_img(&buf);  
img_computed = 1;
```

### Software CPU2

```
count=0;  
while (img_computed != 1)  
    count++;  
assert(count == 3);  
read_img(&buf);
```



## (Counter) Examples

TLM should not exhibit **"too many"** unrealistic behaviors

### Software CPU1

```
dest = stg_fast();
```

### Software CPU2

```
stg_very_slow();  
do_stg_with(dest);
```

- No explicit synchronization between `dest = ...` and access to `dest ...`
- ... but do we want to see this bug ?



## Set of Behaviors and Non-Determinism

- TLM models should exhibit **several** behaviors
- Several possibilities  $\Rightarrow$  non-determinism
- Implementing non-determinism :
  - Formal verification approach : exhaustive exploration
  - Simulation approach : random



## An Example of Non-Determinism : Loose Timing

```
// generates a pseudo-random float  
// between 0.0 and 0.999...  
float randfloat()  
{  
    return rand() / (float(RAND_MAX)+1);  
}  
  
// loose timing  
void pv_wait (x) {  
    wait(x*(randfloat()+0.5));  
}
```

### Question



What does it do ?



## Transaction bloc

### Avant

```
for (ensitlm::addr_t ad = ad0;  
     ad < ad0+size; ad++) {  
    write(socket, ad,  
          data[ad]);  
}
```

- Grosse granularité
- Beaucoup plus rapide en simulation

### Après

```
// 1 echange atomique  
block_write(socket, ad, data, size);
```

### Question



Perd-t-on en fidélité ?



## Timing approximé et découplage temporel

- Constat : les context-switch sont lents.
- Conséquence 1 : `wait` coûte cher !
- Conséquence 2 : on évite de mettre des `wait`.



## Timing approximé et découplage temporel

- **Problème :**
  - ▶ En PVT, granularité de temps fine.
  - ⇒ 1 wait pour chaque avancement du temps.
  - ⇒ simulation lente.
- **Solution proposée en TLM 2 :**
  - ▶ Chaque processus peut être « en avance » sur le temps global.
  - ▶ (L'avance correspond à l'argument `sc_time` des méthodes transport de TLM-2, ignoré dans ENSITLM)
  - ▶ Quand faire les `wait()` (i.e. laisser le reste de la plate-forme rattraper notre avance) ?
    - ★ **Quantum Keeping :** Si on est plus « en avance » que le quantum (constante de temps choisie par l'utilisateur)
    - ★ **Synchronisation explicite :** Avant (ou après ?) les points de synchronisation



## Timing approximé et découplage temporel

### Exemple

```
sc_time t = SC_ZERO_TIME; // local advance over SystemC time
local_computation(); // Instantaneous wrt SystemC
t += sc_time(12, SC_NS); // don't wait() now
other_computation();
t += sc_time(3, SC_NS);
socket.write(addr, data, t); // may update t
wait(t); t = SC_ZERO_TIME; // Catch-up with SystemC time
send_interrupt();
```

### Question



Quel sont les problèmes ?



## Parallélisation de SystemC

- **Paradoxe :**
  - ▶ Les systèmes sur puces sont parallèles
  - ▶ SystemC a une notion de processus
  - ▶ SystemC n'exploite qu'un processeur !
- **Solution (très) naïve :**
  - ▶ 1 SC\_THREAD → 1 pthread
  - ▶ On lance tout en parallèle.
  - ⇒ beaucoup de pthreads, ne passe pas à l'échelle.
- **Solution moins naïve :**
  - ▶  $N$  processeurs →  $\approx N$  pthreads.
  - ▶ Gestion du mapping « processus SystemC » ↔ pthread dans le kernel SystemC.

### Question



Où est le problème ?



## Parallélisation de SystemC

- Si on veut faire les choses proprement :
  - ▶ Analyse statique des dépendances de données
  - ▶ Prise en compte à l'exécution
  - ▶ (DRT de Yussef Bouzouzu—2 ans de travail)
  - ⇒ on sort du principe « SystemC, c'est facile à compiler, g++ le fait très bien ».
- **Problème restant :**
  - ▶ Parallélisation « à l'intérieur du  $\delta$ -cycle », mais exécuter un parallèle des processus censés s'exécuter à différents instants de `simu` = difficile.
- **Solutions envisageables :**
  - ▶ Profiter du découplage temporel pour paralléliser
  - ▶ Tâches avec durée (cf. jTLM)



## Parallélisation de SystemC : conclusion

- C'est dur
- La plupart des gens qui le font ne se soucient pas de préservation de la sémantique
- Solution en pratique : lancer  $N$  simulations sur  $< N$  machines !
- Il faut peut-être un autre langage ?



## Accélération des simulations SystemC

- Code interne aux composants = C++  
⇒ g++ -O3 et le tour est joué (ou pas)
- Context-switch = cher :
  - ▶ Scheduling
  - ▶ Sauvegarde/restauration de tous les registres
  - ▶ Changement du pointeur de pile ⇒ cache-miss
  - ▶ Changement du compteur programme ⇒ vidage de pipeline
- Transactions = cher :
  - ▶ Plusieurs appels de méthodes virtuelles (⇒ non inline-ables)
  - ▶ Décodage d'adresse
  - ⇒ Là où la vraie plateforme fait un `load/store`, le code TLM exécute du code difficile à optimiser.



## Accélération des simulations SystemC Minimiser le coût du au context-switch

- Utilisation de `SC_METHOD` à la place des `SC_THREAD` (pas toujours possible)
- Minimisation du nombre de `wait` à exécuter
- Scheduling statique (<http://www.cprover.org/scoot/> : plus de travail à la compilation, simulation 2 à 6 fois plus rapide sur des exemples)



## Accélération des simulations SystemC Minimiser le coût du aux transactions

- DMI = Direct Memory Interface : on récupère un pointeur sur la zone mémoire intéressante, et on fait des accès sans passer par le bus.

### Question



Quel est le problème ?

- Techniques de compilation spécifiques : lancer une passe d'optimisations après l'élaboration (prototype basé sur LLVM développé au LIAMA et à Verimag)



## Vérification Formelle de SystemC

- Plusieurs approches :
  - ▶ Compilation de SystemC vers des langages sources des outils de preuve
    - ★ Front-end SystemC dédié
    - ★ Modélisation de toutes les constructions SystemC
    - ★ (c'était ma thèse ...)
  - ▶ Exploration exhaustive de l'espace d'état à l'exécution
    - ★ stateless : on explore tous les ordonnancements de taille  $< N$
    - ★ statefull : on ajoute la possibilité de mémoriser et de comparer des états ( $\Rightarrow$  construction de l'espace d'état entier)
  - ▶ En général, problème de passage à l'échelle (state explosion)

### Question



Quel est le problème ?

## Exposé Docea Power

### Question



Qu'en avez-vous retenu ?