Journées Doctorales 2009

Conception d'un outil de débogage formel pour systèmes logiciels ou matériels selon l'approche "*Debug as Design*"

Nicolas Vallée – UEI

27 Janvier 2009

Plan

- Intérêts de la vérification de systèmes
  - Coûts liés aux bugs
  - Nécessité d'outils complexes de vérification
- En quoi consiste mon travail ?
- Résultats expérimentaux
  - Cas d'école : un compteur
  - Vérification d'un bus pour STMicroElectronics
- Bibliographie

Intérêts de la vérification de systèmes

Coûts des bugs

Sonde Mariner 1 pour Vénus (1962)

oubli d'un trait d'union dans un programme FORTRAN

⇒ 80 millions de dollars

Fusée Ariane 5 – vol 501 (1996)

réutilisation sans adaptation d'un programme fonctionnant parfaitement ave Ariane 4, mais taille des données 4 fois plus grande…

⇒ 1 milliard de dollars

Imaginez que l'ordinateur de vol d'un Airbus s'arrête en plein décollage…

Intérêts de la vérification de systèmes

Nécessité d'outils complexes de vérification

Indécidabilité : impossible qu'un programme puisse prouver qu'un programme quelconque soit sans bug

2 phénomènes contradictoires :

- Complexité croissante des systèmes
- Baisse des durées de conception/production (time to market)

Vérification des spécifications fonctionnelles

tache difficile souvent réalisée à la fin de la conception

$\Rightarrow$ corrections coûteuses

Attentes de l'industrie

- systeme de conception modulaire
- possibilité d'effectuer des tests poussés très tôt

En quoi consiste mon travail ?

Démystification du sujet de thèse (1/2)

débogueur formel

- Débogeur = outil capable de
  - détecter des erreurs
  - localiser les erreurs
- Formel = inutile de faire fonctionner réellement

  ⇒ indépendant des conditions expérimentales

  ⇒ certifier dans tous les cas possibles

systèmes logiciels ou matériels

analyse de description de systèmes

- Logiciel → code source
- Circuit électronique → description dans un langage HDL

En quoi consiste mon travail ?
Démystification du sujet de thèse (2/2)

approche ”Debug as Design”

utiliser des outils de vérification dès la conception

- trouver et corriger rapidement les petites erreurs
- réutiliser les résultats des analyses des sous-composants pour analyser un composant dès que possible
- abstraction progressive de l'analyse
  - au début, vérification des assertations et détection de problèmes locaux
  - puis, trouver les différences entre la spécification et la réalisation du système
  - au final, analyse fonctionnelle du système complet

En quoi consiste mon travail ?

Que devrait pouvoir détecter cet outil ?

- Violation de protocole ou de spécification
- Violation des assertions
- Accès incorrect aux données
  - hors de bornes d'un tableau
  - attribut privé d'un objet
- Utilisation avant définition / initialisation
- Informations calculées mais inutilisées
- Dépendances aberrantes
- Etreintes fatales
- Mauvaise implantation de fonctionnalités

En quoi consiste mon travail ?

Un peu de théorie

Systèmes complexes

- nombre de sous-composants important
- éventuelle utilisation de différentes technologies
- niveau d'abstraction différent suivant les composants

Différentes techniques de vérification formelle

- Model-checking (logique temporelle)
- Interprétation abstraite
- Démonstrateurs de théorèmes

Nécessité d'une représentation commune et adaptée à chaque technique de vérification

$\Rightarrow$ hypergraphes fractals

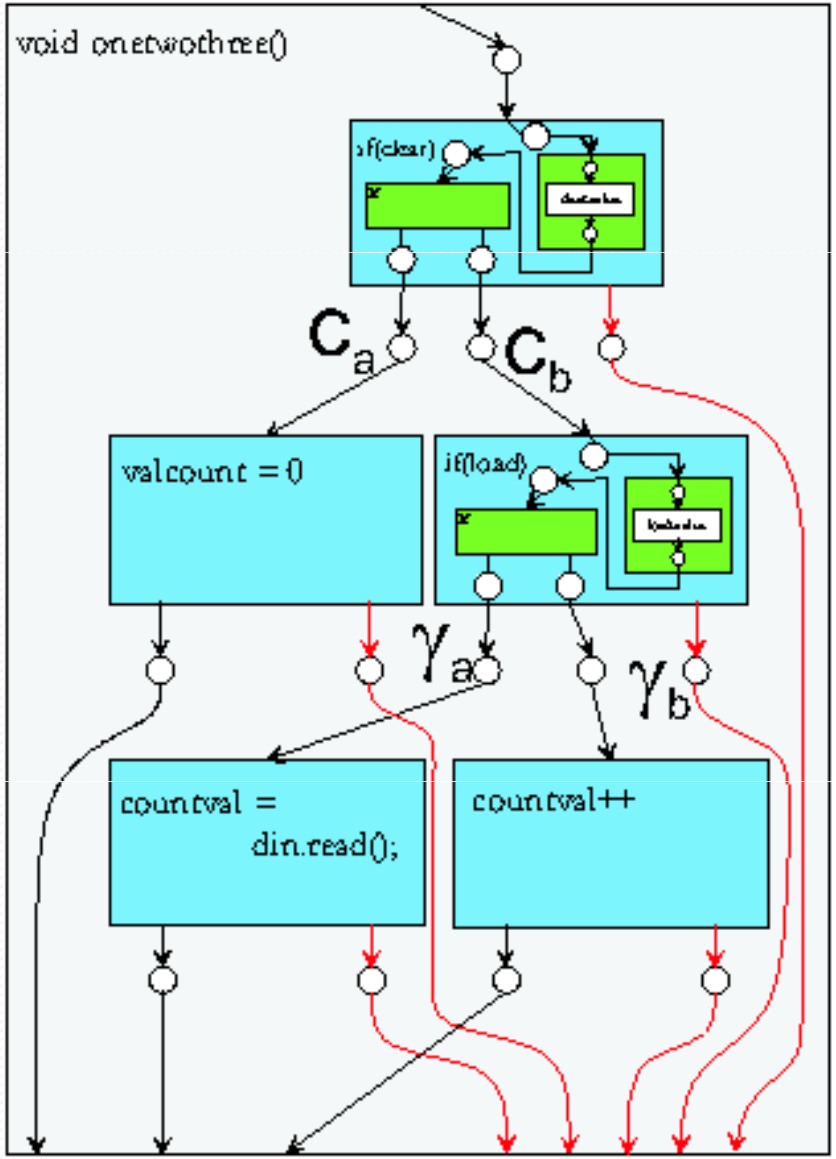
automate dont les transitions peuvent être des sous-automates

Résultats expérimentaux

Cas d'école : un compteur

```
SC_MODULE(counter) {
  sc_in<bool> clock, load, clear;
  sc_in<sc_int<8> > din;
  sc_out<sc_int<8> > dout;
  sc_int<8> countval;

  SC_CTOR(counter) {
    SC_METHOD(onetwothree);
    sensitive_pos(clock);
  }
  void onetwothree() {
    if(clear)
      countval = 0;
    else if(load)
      countval = din.read();
    else  countval ++;
    dout = countval;
  }
};
```



Etapes :

- Parsage description HDL vers hypergraphe fractal
- Exécution symbolique valeurs formelles de toutes les données du système
- Interprétation abstraite via insertion compteurs

Résultats :

$$c_a \in \{0;1\} \;\; ; \; c_b \in \{0;1\}$$

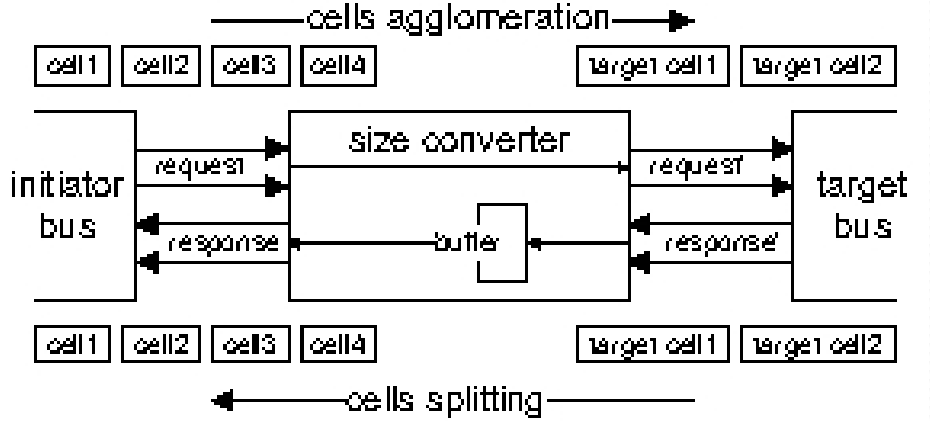
$$\gamma_a \in \{0;1\} \;\; ; \; \gamma_b \in \{0;1\}$$

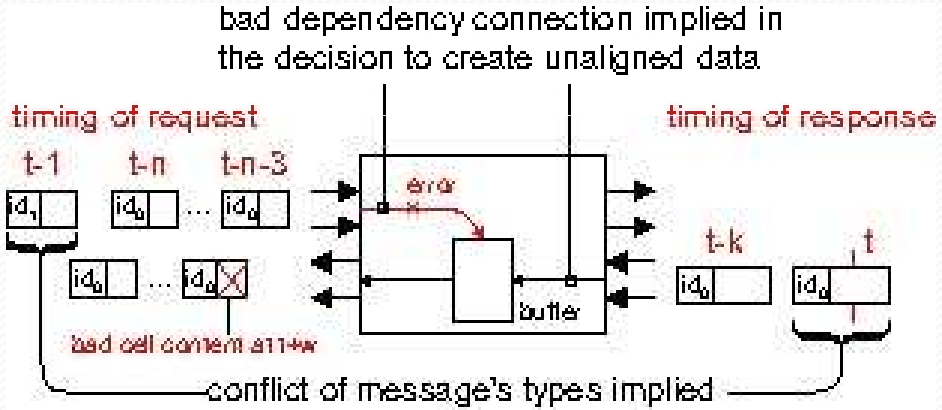
$$c_a = clear \;\; ; \; c_a + c_b = 1 \;\; ; \; \gamma_a + \gamma_b = c_b$$

$$din.value = v$$

$$valcount = \begin{cases} v_a & \text{à l'entrée} \\ (v_a + 1)c_b + v\gamma_a & \text{à la sortie} \end{cases}$$

Résultats expérimentaux
Cas industriel : bus avancé chez STMicroElectronics





- Détection d'une erreur
  - après analyse manuelle, convertisseur de taille
  - aucune erreur sur les actions élémentaires effectuées
- Difficulté ?
  - non-alignement ne viole aucun des protocoles
  - erreur détectée plus tard, donc origine difficile à déterminer
- Comment l'a-t-on détecté ?
  - ordre des données envoyées dépendait à la fois de l'émetteur et du récepteur

Bibliographie

On the design of a Formal Debugger for System Architecture
Bruno Monsuez, Franck Védrine, Nicolas Vallée
DATICS'08 (Crète – Juillet 2008)

How an "Incoherent Behavior" inside generic hardware component characterizes functional errors ?
Bruno Monsuez, Franck Védrine, Micaela Mayero, Nicolas Vallée
CISST'09 (Chine – Janvier 2009)

Beyond Hierarchical graphs : Fractal Hypergraphs
Bruno Monsuez, Nicolas Vallée, Franck Védrine
(en attente de publication)

Fin

Des Questions ?