Vérification de circuits

Problèmes et Solutions,

exemple de TestBuilder

Gilles BADOIL

26 Mars 2002

① Introduction
② Principaux outils disponibles
③ Présentation de la solution TestBuilder
④ Conclusion

INTRODUCTION

Illustration du problème

Intel Pentium FDIV 1994	➪	Perte > 400 Millions de dollars
Motorola 68040	➪	Plusieurs mois de retard
Intel Pentium PIII	➪	65 erreurs répertoriées
AMD Athlon	➪	6 erreurs répertoriées

Or les circuits sont aujourd'hui *omniprésents*

➜ Informatique, communication, téléphonie

avec de plus en plus de *responsabilité*

➜ Transports, santé, sécurité, etc.

Comment éviter ou limiter de telles erreurs de conception ?

PROCESSUS DE CONCEPTION

Principales étapes de conception

- ➜ Spécifications (textuelles)
- ➜ Description HDL ➪ RTL

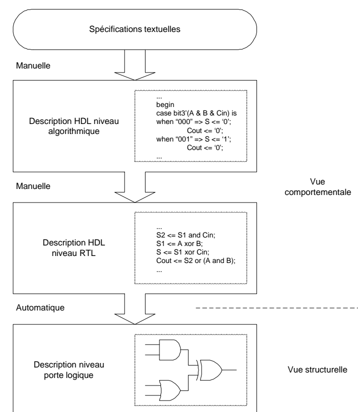
  HDL :Hardware Description Language

  RTL : Register Transfer Level
- ➜ Synthèse logique ➪ netlist
- ➜ Placement/Routage

  ➪ masque (layout)

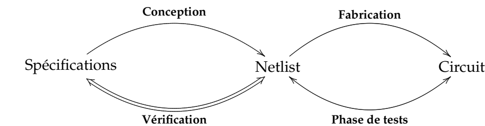
Deux HDL standards

- ➜ VHDL
- ➜ Verilog



LA VÉRIFICATION

Vérification vs phase de test



La vérification porte sur un circuit qui n'existe pas physiquement

Deux approches

- ➜ Méthodes formelles
  - Forte « teneur mathématique »
  - Méthodes exhaustives : valables quelles que soient les entrées
- ➜ La simulation
  - Permet l'interaction avec un modèle du circuit
  - Nécessite un banc de tests

Importance et coût de la vérification

Nécessité de découvrir les erreurs au plus tôt

- ➜ Plus une erreur est découverte tard, plus le coût est élevé
- ➜ Etape de fabrication très coûteuse

Différences avec le logiciel

- ➜ La correction d'une erreur coûte très cher (notion de *révisions*)
- ➜ Aucun espoir de vendre plus facilement la version n+1

Quelques chiffres [Bergeron2000]

- ➜ Jusqu'à 70% de l'effort de conception
- ➜ Plus d'ingénieurs de vérification que de concepteurs
- ➜ Jusqu'à 80% des lignes de code d'un projet

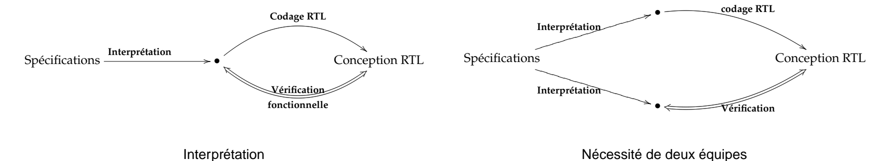
Un enjeu économique important

Les difficultés (1/2)

Le facteur humain

Spécifications
Interprétation
Codage RTL
Conception RTL
Vérification fonctionnelle

Interprétation



Nécessité de deux équipes

Les contraintes du marché

- ➜ Nécessité d'être le premier sur le marché
- ➜ Cycle de vie des produits de plus en plus court

Les difficultés (2/2)

Des circuits de plus en plus complexes

- ➜ Augmentation de la densité d'intégration ➪ Millions de transistors
- ➜ Miniaturisation ➪ SoC (System on a Chip)
- ➜ Logiciel embarqué
- ➜ Techniques d'optimisation (pipeline)

Simulation *exhaustive* impossible dans la pratique

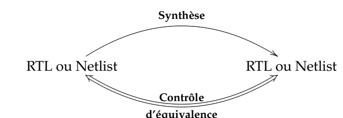
- ➜ Explosion combinatoire
- ➜ Quand arrêter les tests ?

➪ Besoin d'outils performants

Le contrôle d'équivalence

Prouver *mathématiquement* l'équivalence de deux descriptions

- ➜ Travaille au niveau RTL et/ou Netlist
- ➜ Vérification de la synthèse
- ➜ Vérification d'optimisations manuelles



Intérêts

- ➜ Vérification *mathématique* valide pour toutes les entrées
- ➜ Méthode automatisée

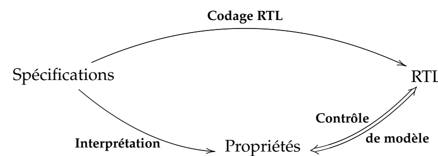
Limitations

- ➜ Ne peux travailler que sur des portions de circuits
- ➜ Prouve l'équivalence. N'aide pas à trouver les erreurs fonctionnelles

Le contrôle de modèle

Principe de fonctionnement

- ➜ Un modèle
- ➜ Des propriétés
- ➜ Un algorithme



Avantages

- ➜ Méthode en partie automatisée
- ➜ Preuve *mathématique* indépendante des entrées
- ➜ Capacité de fournir des contre-exemples
  ➪ utile pour dépister des erreurs

Difficultés

- ➜ Propriétés difficiles à établir
- ➜ Problème d'explosion combinatoire

La simulation

Principe

- ➜ Interaction avec un modèle informatique ➪ Bancs de tests

Un processus fondamentalement *lent*

Principaux outils de simulation

- ➜ Le simulateur
  - Objectif : reproduire le comportement du circuit
  - Travaille directement sur une description HDL
- ➜ Le visualiseur : le *débogueur* de la simulation
  - Analyse *graphique* de la simulation
  - Permet de travailler *a posteriori*
- ➜ Les outils de couverture de code
  - Évaluer les sections de code non exécutées
  - Un indice de confiance, une mesure de progression

Écriture des bancs de tests

Utilisation des HDL

- ➜ Principe : utiliser les possibilités du langage (E/S, etc.)
- ➜ Avantage : un même langage pour la modélisation et la vérification
- ➜ Inconvénients : les HDL ne sont pas spécialement adaptés à cela

Utilisation de langages externes

- ➜ Les HDL proposent des API (Interfaces de programmation)
  ➪ possibilité d'interagir avec la simulation depuis C/C++, Perl ou TCL

Un mécanisme puissant mais contraignant

HVL : langages spécialisés dans la vérification

(HVL : Hardware Verification Language)

Objectifs

- ➜ Masquer la complexité des appels aux API
- ➜ Intégrer les facilités des langages de programmation traditionnels
- ➜ Offrir des facilités pour les bancs de tests
- ➜ Améliorer la couverture *fonctionnelle*

Les différents langages

e/Specman de Verisity — *Rave/Quickbench* de Forte

Vera/Testbench de Synopsys — *Jeda* – licence GPL

Plusieurs solutions, pas de standard

TestBuilder

Présentation

- ➜ Librairie de classes C++ développée par Cadence®
- ➜ Solution récente (version 1.0 sortie en avril 2001)
- ➜ Licence Open-source

Objectif : les avantages d'un HVL avec un langage standard

Moyens mis en œuvre

① Concepts liés au matériel
② Facilités pour l'écriture des bancs de tests
③ Notion de transaction

Implémentation de concepts matériels

Les Signaux

- ➜ tbvSignal2StateT
- ➜ tbvSignal4StateT

```
tbvSignal2StateT myOpCode(3,0);
tbvSignal2StateT myOpA(15,0);
if (myOpCode == INCREMENT) ++myOpA;
```

Les signaux HDL

- ➜ tbvSignalHDL2StateT
- ➜ tbvSignalHDL4StateT

```
tbvSignalHdl4StateT regA("v.regA");
regA = 3;
```

Synchronisation

- ➜ Synchronisation

```
tbvWait(5); tbvWaitCycle(regA);
```

Relation avec le modèle HDL

- ➜ On ne gère pas manuellement les API des HDL
- ➜ `$tbv_main` lance TestBuilder en parallèle avec la simulation

Facilités pour l'écriture des bancs de tests (1/2)

Vérification de propriétés temporelles

- ➜ Définition d'une fenêtre d'observation
- ➜ Vérification de la réalisation ou non d'une propriété
- ➜ Lancement d'une procédure en cas d'échec
- ➜ Exemple : Le bus est accordé *t* cycles après une demande

Un environnement multithread

- ➜ Gestion des threads et mécanismes de synchronisation

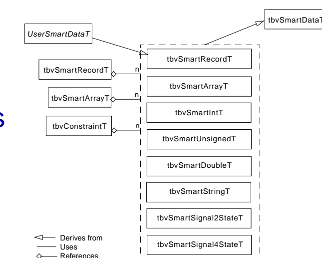
Structure de données

- ➜ Nombreux modèles de classes, dans l'esprit de la STL (Standard Template Library)
- ➜ Principe de généricité avec les patrons de classes C++

Facilités pour l'écriture des bancs de tests (2/2)

Les smart-data

- ➜ Ensemble de classes
- ➜ Diverses facilités



Génération de tests avec contraintes

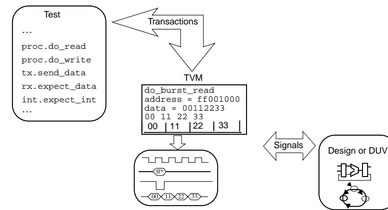
- ➜ Objectif : Automatisation des tests dans un contexte réaliste
- ➜ Méthode `randomize`
- ➜ Méthodes de sélection de plages (keepOnly, keepOut)
- ➜ Classe tbvConstraintT

La notion de transaction

Principe

Découpage en couches

- ➜ Les tests
- ➜ Les transactors (ou TVMs) et les tâches



TVM : Transaction Verification Model

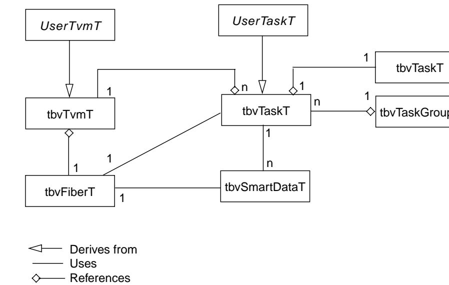
DUV : Design Under Verification

Avantages

- ➜ Division des responsabilités
- ➜ Amélioration de la concision, de la lisibilité et de l'efficacité des tests
- ➜ Possibilités de réutilisation

Principe d'*encapsulation* appliqué aux signaux et protocoles

Les transactions dans TestBuilder



- ➜ Les TVMs et tâches sont des classes
- ➜ Un test est un appel cohérent de tâches (`run`, `spawn`)
- ➜ Un mécanisme permet de garder une trace des transactions (`tbvFiberT`)
- ➜ On peut relier les tâches (notion de succession)
- ➜ On peut regrouper les tâches (`tbvTaskGroupT`)

Conclusion sur TestBuilder

Le choix de C++

- ➜ Jeu de classes cohérent
- ➜ De plus en plus utilisé pour la modélisation de circuits

Comparaison avec les HVL

- ➜ Des fonctionnalités similaires
- ➜ Langage standard et bibliothèque open-source
  - ➪ Pas de dépendance directe vis-à-vis d'un éditeur
  - ➪ Pérennité
  - ➪ D'avantage de possibilités d'adaptation
- ➜ Une relative complexité

Une base standard pour les outils de vérification ?

Conclusion

La simulation

- ➜ La méthode la plus utilisée
- ➜ Un processus fondamentalement lent
  - ➪ Bancs de tests efficaces, techniques de programmation modernes
- ➜ Plusieurs langages mais pas de standard

Les méthodes formelles

- ➜ Méthodes exhaustives
- ➜ Deux méthodes aujourd'hui exploitées par les outils commerciaux
- ➜ Nombreuses recherches dans ce domaine

A court terme

- ➜ Évolution des HDL, techniques d'accélération pour la simulation ?
- ➜ Généralisation du C++ pour la modélisation ?
- ➜ Généralisation de composants de vérification ?
- ➜ Méthodes semi-formelles ?