Vérification fonctionnelle des systèmes numériques

Introduction

Yann Thoma

Reconfigurable and Embedded Systems Institute
Haute Ecole d'Ingénierie et de Gestion du canton de Vaud

Octobre 2010

Plan

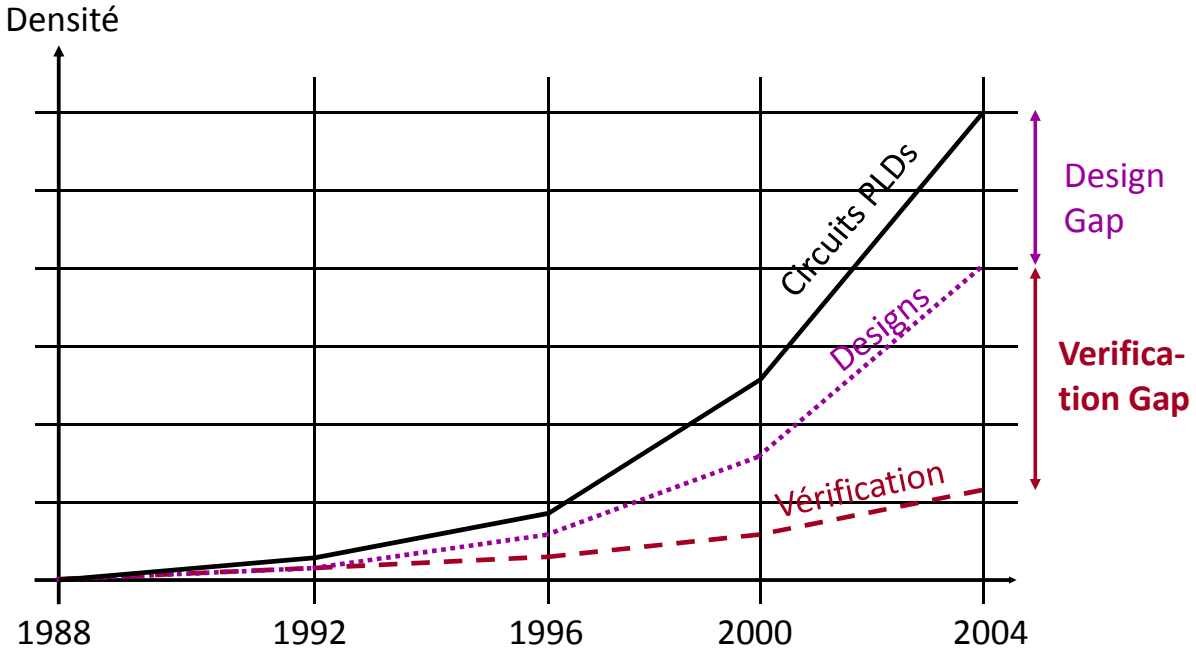
Vérification: contexte

- Dépannage d'un PLD programmé
  - Très coûteux
  - Voir impossible
- Dépannage d'un ASIC
  - Très coûteux (en temps et argent)
  - Refonte du circuit
- Vérification indispensable pour valider le système
  - Nécessite des outils performants
  - Une méthodologie rigoureuse
- La vérification est *le défi* de la conception des systèmes numériques

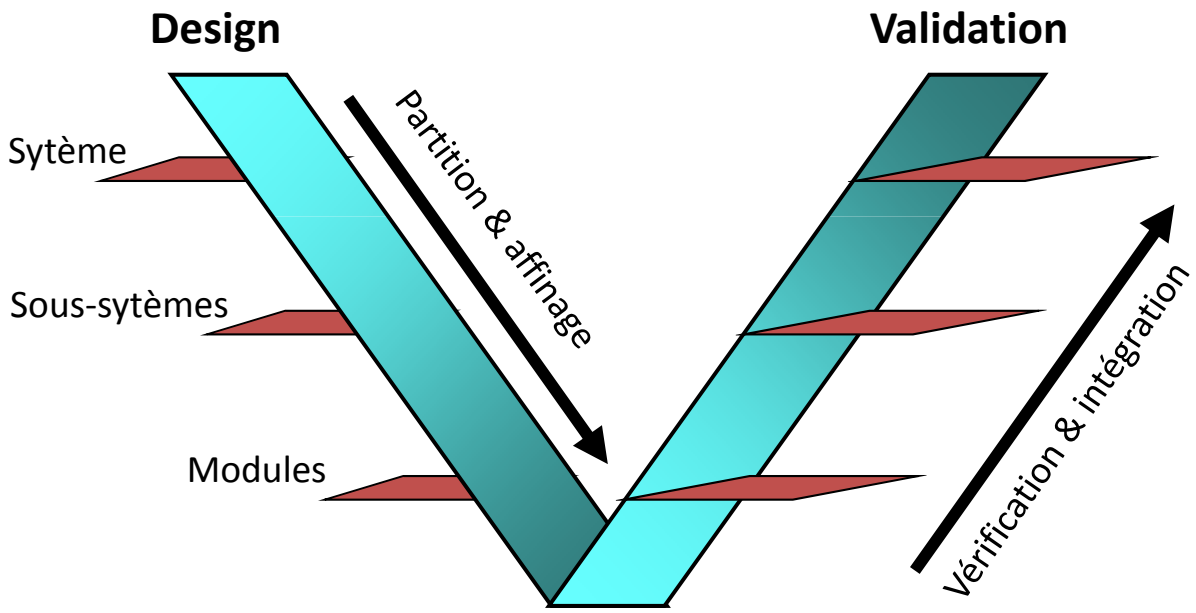
Vérification: concept

- Buts
  1. Vérifier la conformité du système avec le cahier des charges
  2. Détecter les erreurs au plus vite
  3. Assurer un fonctionnement correct après synthèse et intégration
- Difficulté
  1. Maîtriser les coûts de la vérification
     - Environ 70% du temps ingénieur
  2. Garantir un fonctionnement correct

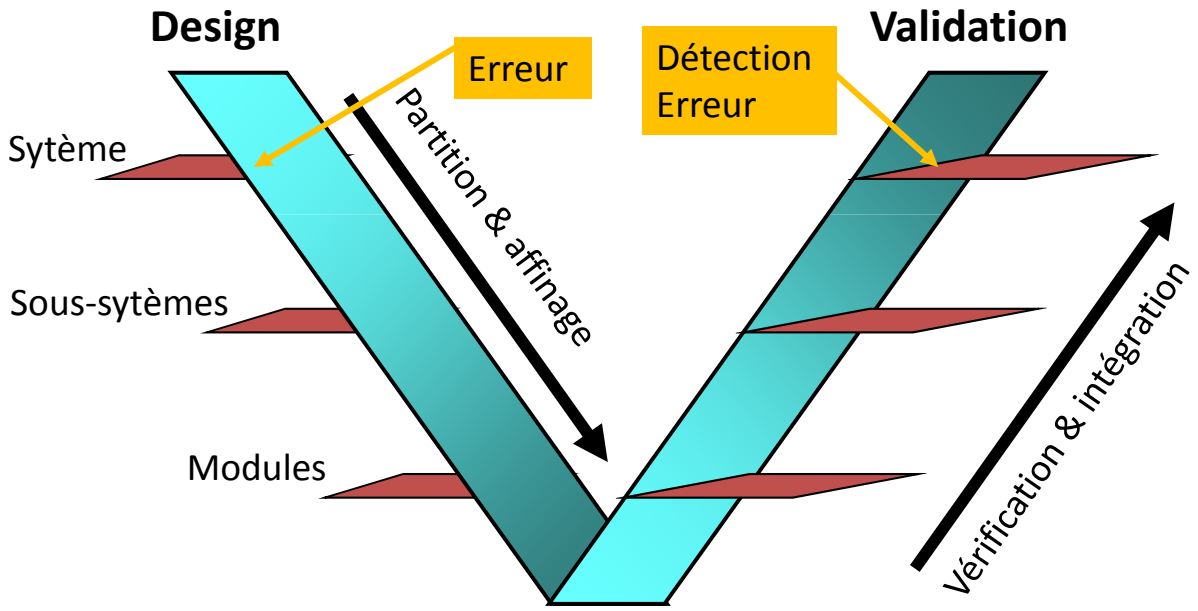
Vérification: évolution de la complexité



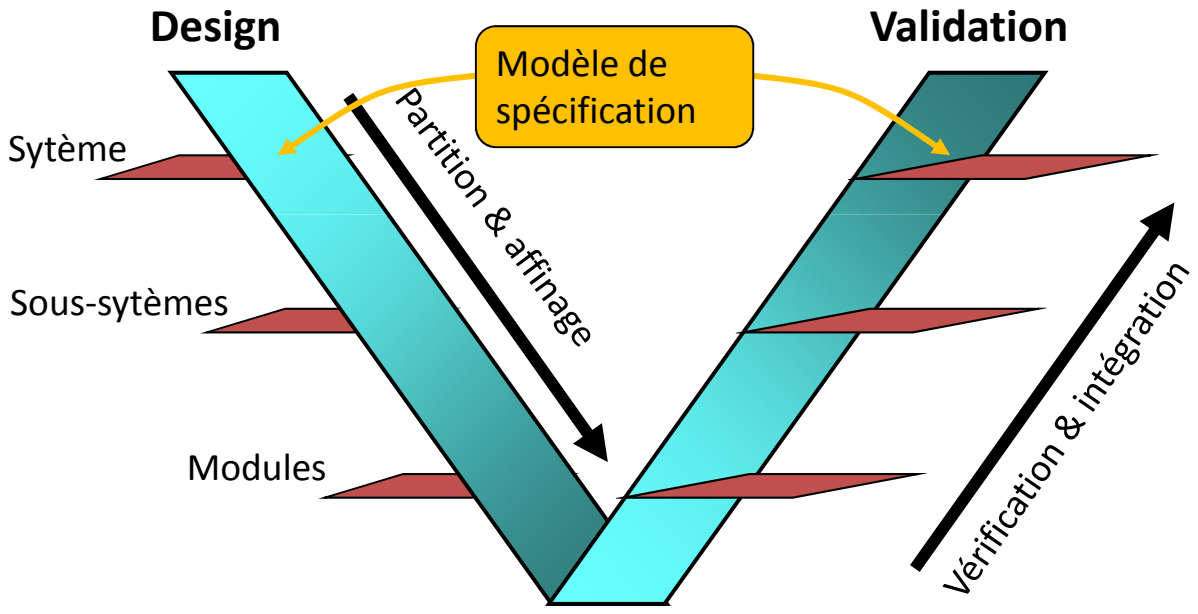
Vérification: décomposition



Vérification: décomposition



Vérification: décomposition



Vérification: les 2 questions

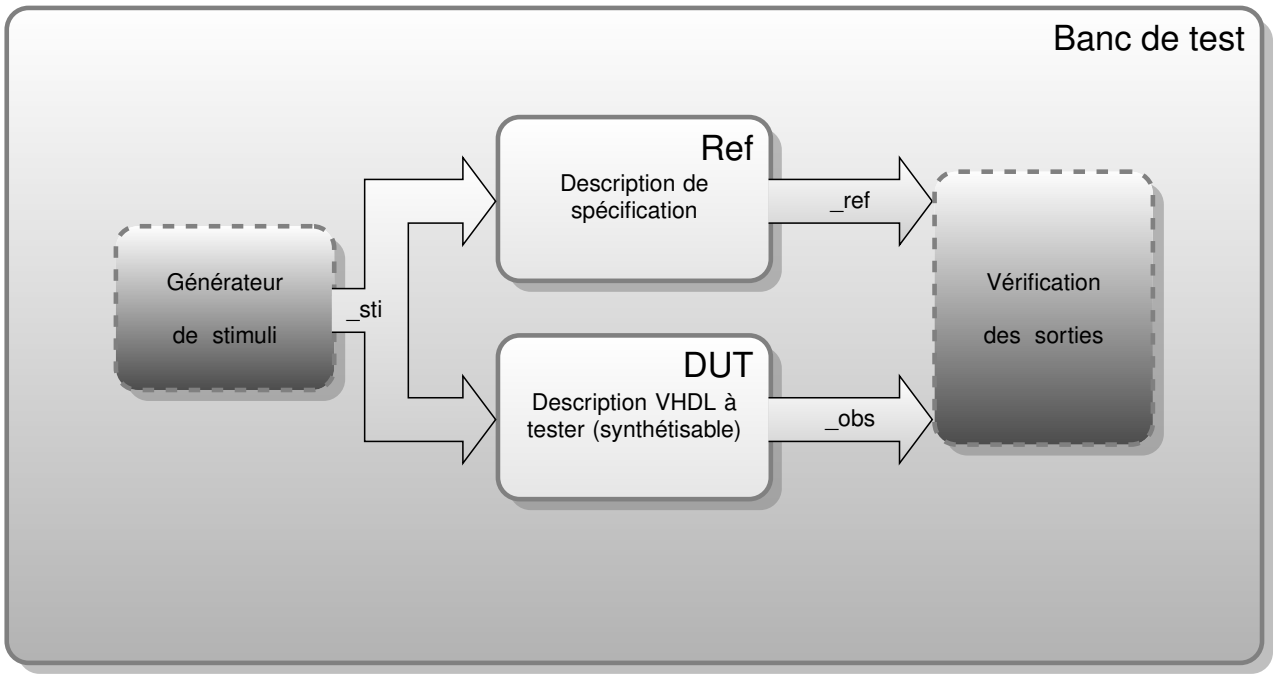
- Les deux grandes questions de la vérification
  1. Est-ce que ça marche?
     - Est-ce que le système fonctionne correctement?
  2. Est-ce qu'on est sûr? (a-t'on fini?)
     - A-t'on réalisé assez de tests?

Vérification black/white/gray box

- But d'un design: que les sorties soient correctes
- Certaines erreurs du design pourraient être ignorées
  - Erreurs jamais activées
  - Erreurs qui ne se propagent pas jusqu'à la sortie
  - Erreurs qui s'annulent l'une l'autre
- Black box
  - La fonctionnalité n'est vérifiée qu'aux frontières du design
  - Typiquement comparées à un modèle de référence

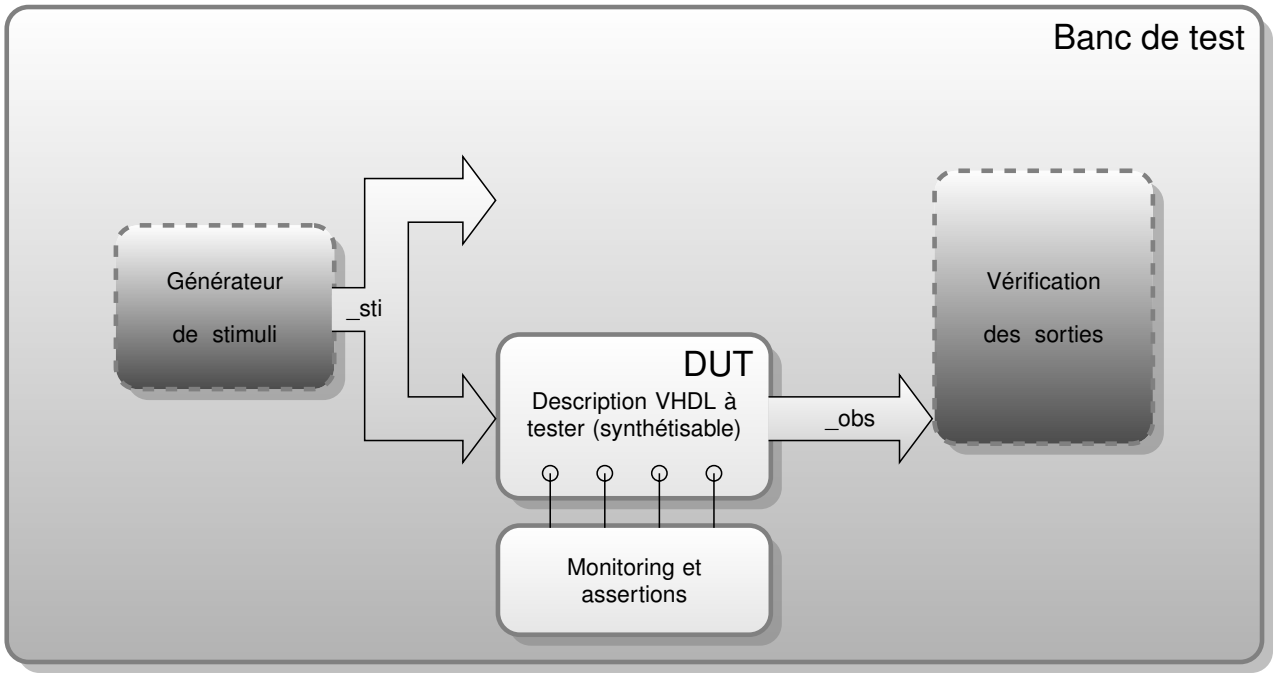
Black box

- Seules les sorties sont analysées
  - Difficulté à trouver les sources d'erreurs
  - Nécessité d'avoir un modèle de référence



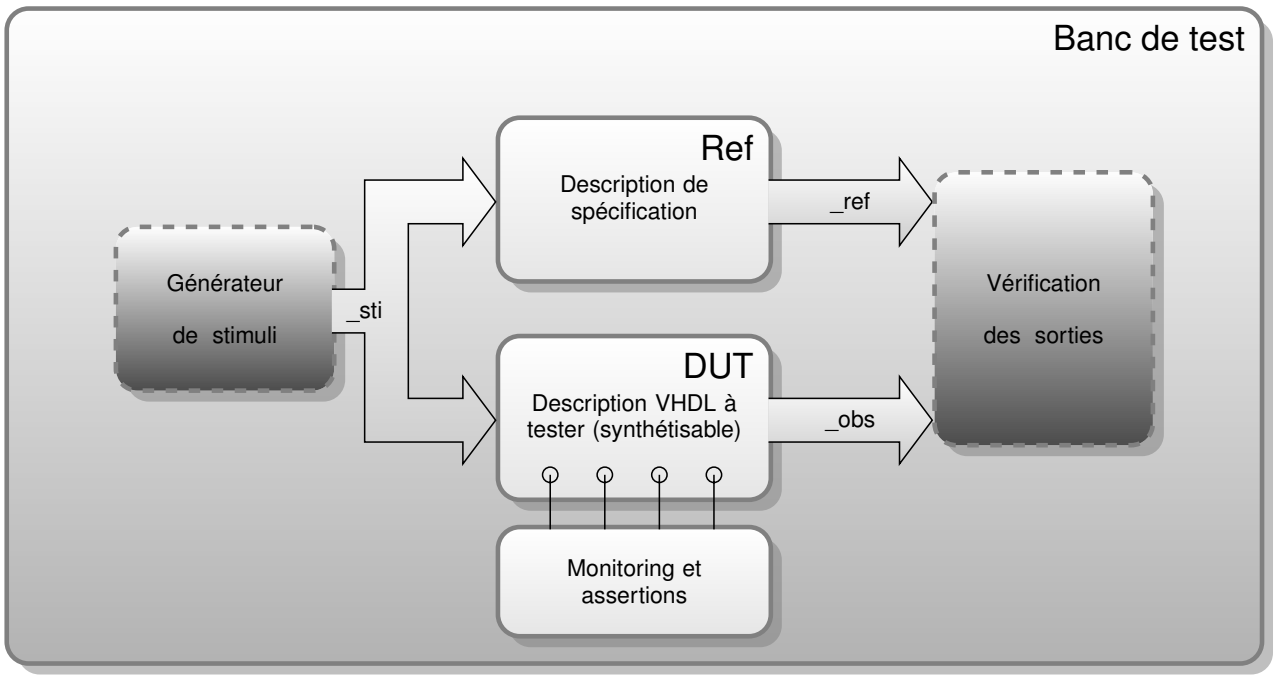
White box

- L'intérieur du design est analysé
  - Recherche d'erreurs facilité
  - Peut être limité pour des grands systèmes



Grey box

- Les sorties et l'intérieur du design sont analysées
  - Compromis entre les deux autres approches
  - Permet de limiter la vérification des sorties par l'analyse interne



White/grey/black box: Effort

Challenge	**Black**	**White**	**Grey**
Création d'un modèle de référence	Haut	Non	Moyen
Ajout de moniteurs et assertions	Non	Haut	Bas
Tracer un bug de la sortie à la source	Haut	Bas	Bas
Vérifier l'implémentation des features	Haut	Bas	Bas

Défis de la vérification

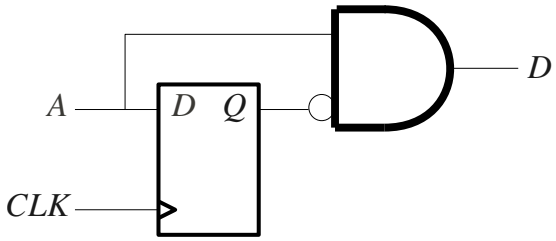
- Le choix d'une solution pour la vérification doit prendre en compte les paramètres suivants:
  - Complétude
    - Maximiser les fonctionnalités (scénarios) de design testée
  - Réutilisabilité
    - Maximiser la réutilisabilité du code pour de futurs projets
  - Efficience
    - Minimiser l'effort à fournir, et automatiser un maximum
  - Productivité
    - Maximiser le travail produit manuellement
  - Performance du code
    - Minimiser le temps de calcul nécessaire à la vérification

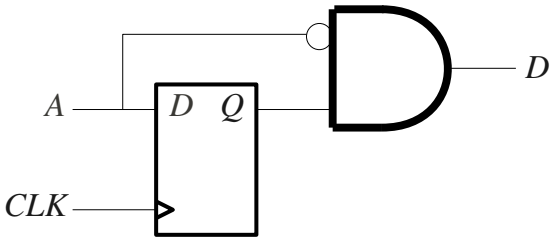
Vérification fonctionnelle: approches

- Vérification formelle
  - Tentative de prouver mathématiquement le bon fonctionnement du système
- Vérification basée sur la simulation
  - Tentative de prouver le bon fonctionnement du système en le simulant
- Vérification par émulation
  - Tentative de prouver le bon fonctionnement du système en l'émulant

Vérification formelle: Property checking

- Un détecteur de flanc montant
- Propriété à vérifier:
  - Si $A = 1$ alors au cycle d'horloge suivant: $D = 0$
    $\Leftrightarrow (A_{t-1} = 1) \Rightarrow D_t = 0$
    $\Leftrightarrow \overline{A_{t-1}} + \overline{D_t} = 1$

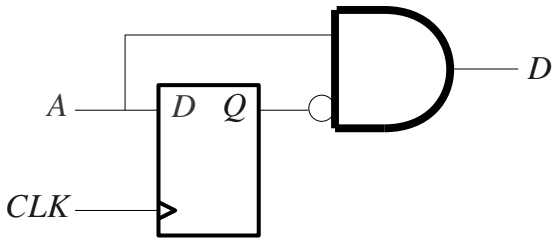


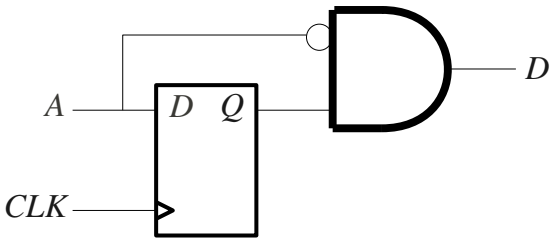


- Quel design est le bon?

Vérification formelle: Property checking

- Propriété à vérifier: $\overline{A_{t-1}} + \overline{D_t} = 1$





- $D_t = A_t \cdot \overline{A_{t-1}}$
- Donc

$$\begin{aligned} \overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + A_t \cdot \overline{A_{t-1}} \\ &= \overline{A_{t-1}} + \overline{A_t} + A_{t-1} \\ &= 1 \end{aligned}$$

cqfd.

- $D_t = \overline{A_t} \cdot A_{t-1}$
- Donc

$$\begin{aligned} \overline{A_{t-1}} + \overline{D_t} &= \overline{A_{t-1}} + \overline{A_t} \cdot A_{t-1} \\ &= \overline{A_{t-1}} + A_t + \overline{A_{t-1}} \\ &= \overline{A_{t-1}} + A_t \neq 1 \end{aligned}$$

Propriété non vérifiée

Vérification par simulation: approches

- Plusieurs manières d'aborder la simulation:
  - Bancs de test dirigés (Directed testbench)
    - Scénarios prévus
    - Par exemple: tests exhaustifs (!)
  - Basé sur la couverture et l'aléatoire (Coverage-driven random-based)
    - Génération de stimuli aléatoires
    - Test de couverture pour décider de la fin de la simulation
  - Vérification basée sur les assertions (assertion-based)
    - Assertions peuvent servir à une preuve formelle
    - Ou sont vérifiées par simulation
- Les outils (langages) sont plus ou moins adaptés à l'une ou l'autre approche

Manuelle vs. automatique

- Vérification manuelle
  - Forcer à la main les signaux d'entrée
  - Grands risques d'erreur
  - Difficulté à réexécuter le même scénario
  - Vérification visuelle par le développeur, par le chronogramme de simulation
  - Fastidieux (temps perdu)
- Vérification semi-automatique
  - Les signaux d'entrée sont forcés automatiquement
  - Vérification du fonctionnement par le développeur
  - Risque de mauvaise analyse par le "vérificateur"

Vérification automatique

- But: vérification la plus complète possible du système
- Résultat Go/No Go (OK, KO)
- Indication de l'instant d'une erreur
- Possibilité de stopper la simulation sur une erreur
- Indication du nombre d'erreurs total
- Permet une re-vérification complète (Go/no Go) après une modification ou une correction
- Les interactions utilisateur doivent être minimisées
  - Nécessaire pour garantir la similarité des tests effectués

Simulation automatique: scripts

- Une simulation doit pouvoir être lancée via des scripts
- Evite des erreurs de manipulation
- Simplifie la vie

Exemple: sim.do

```
# définition de la librairie de travail
vlib work
# compilation du design à tester
vcom counter.vhd
# compilation du banc de test
vcom counter_tb.vhd
# lancement de la simulation
vsim work.counter_tb
# ajout de tous les signaux au chronogramme
add wave -r *
# exécution de la simulation
run -all
```

Objectifs de la simulation

- Vérifier le fonctionnement logique du module ou du système
- Vérifier les caractéristiques dynamiques du module ou du système (temps d'exécution, fréquence maximum, ...)
- Dans les deux cas, un preuve écrite du résultat de simulation doit être générée

Méthodologie de vérification

- Vérification exhaustive
  - Problème NP-dur
  - En général: impossibilité de tester toutes les possibilités, pour un système combinatoire
  - Pour un système séquentiel: encore pire
- Vérifications classiques
  - Test de la description (transparent)
    - Vérification de toutes les lignes de la description (taux de couverture)
  - Test du fonctionnement (boite noire)
    - Vérification du fonctionnement du système sans tenir compte de la description interne
    - Devrait être réalisé par un autre ingénieur
- Nouvelles approches
  - Assertions, coverage-driven, ...

Méthodologie de vérification

- Il faut éviter qu'un banc de test ne détecte pas une erreur de fonctionnement ou de description
  - Il est facile de réaliser un banc de test qui ne détecte pas d'erreur...
  - Rigueur nécessaire pour la mise au point du banc de test
  - Auto-test du banc de test
- Avec le langage VHDL, l'ordre de la séquence de test est important
  - Un processus est activé par la liste de sensibilité

Validation de la simulation

- En vérification dirigée, la validation est donnée par:
  - La liste des stimuli générés et les vérifications exécutées (fichier test-bench)
  - Et la preuve écrite du résultat des vérifications (fenêtre log du simulateur, fichier texte, HTML, ...)
- En vérification aléatoire, la validation est donnée par:
  - La liste des scénarios testés et les vérifications exécutées (fichier test-bench)
  - Le taux de couverture
  - Et la preuve écrite du résultat des vérifications (fenêtre log du simulateur, fichier texte, HTML, ...)

Vérification des timings

- La vérification des timings est compliquée avec une simulation VHDL
- Une meilleure méthodogie:
- Conception Full synchrone
  - Simulation fonctionnelle
  - Analyse statique des timings
- $\Rightarrow$ Fonctionnement garanti
  - Attention toutefois aux entrées/sorties

Fonctionnement d'un banc de test

- Assignation des entrées du circuit à tester
- Détermination des valeurs des sorties attendues (références)
- Attente d'un délai
  - Permet l'évaluation des sorties par le circuit (temps de propagation)
- Vérification des sorties
  - Comparaison avec la référence
  - Indication claire en cas d'erreur

Langages pour la vérification

- Projets de petite et moyenne taille
  - Langage VHDL
- Projets de taille moyenne à élevée
  - Besoin de
    - Modèles décrits à très haut niveau (transaction)
    - Génération de stimuli aléatoires
    - Vérification par assertions
  - Solutions actuelles
    - SystemC
    - PSL (assertions)
  - Nouveau langage
    - SystemVerilog

SystemVerilog

- Langage orienté objet
  - Réutilisation
- Génération aléatoire
  - Non contrainte
  - Contrainte
- Gestion des assertions
- Fonctions de couverture
- Idéal pour de la vérification aléatoire
- Des méthodologies ont été développées

Méthodologies SystemVerilog

- Cadence: VMM (Verification Methodology Manual)
  - Version 1.2
  - `http://www.vmmcentral.org/`
- Mentor: AVM (Advanced Verification Methodology)
  - Version 3.0
  - Terminé
- Ensemble: OVM (Open Verification Methodology)
  - Version 2.1
  - `http://www.ovmworld.org/`
- UVM (Universal Verification Methodology)
  - Unification de VMM et OVM
  - Version 1.0
  - `http://www.uvmworld.org/`