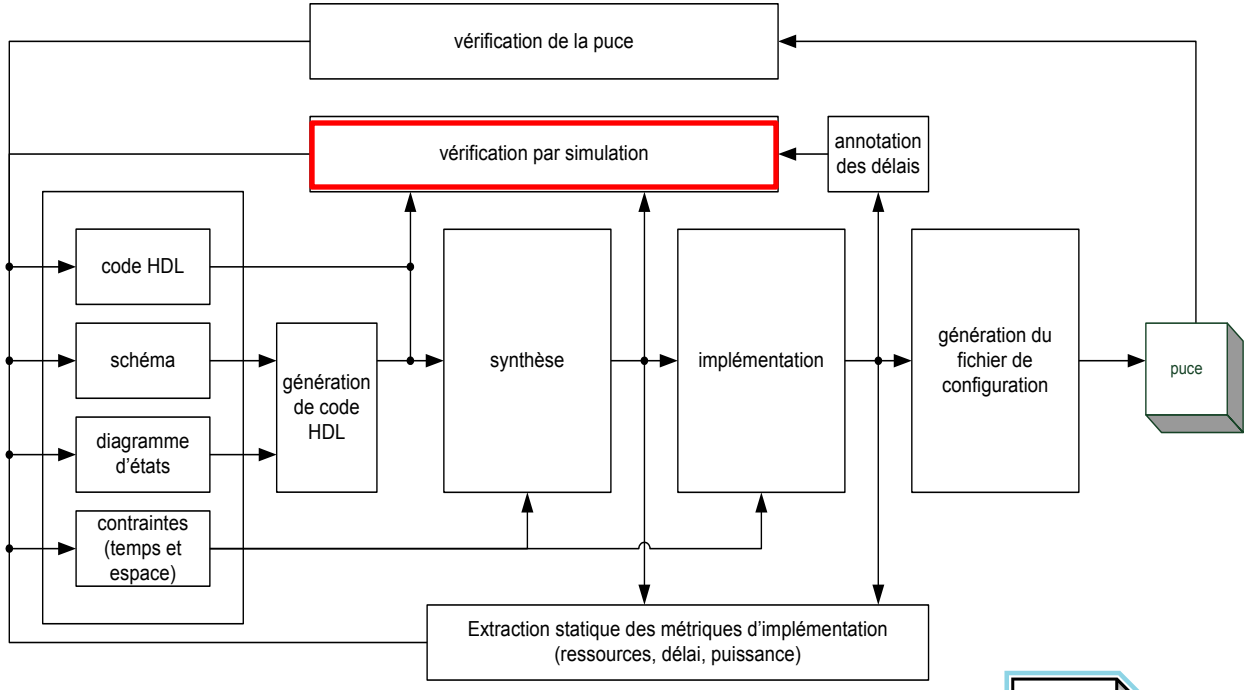
Vérification de circuits numériques: principes généraux

Vérification de circuits numériques: principes généraux
Sujets de ce thème

- Rappel: vérification et bancs d'essai
- Le plan de test: une stratégie pour vérifier un circuit
- 5 qualités d'un bon ensemble de vecteurs de test
- L'analyse statique du code
- Le suivi des bogues

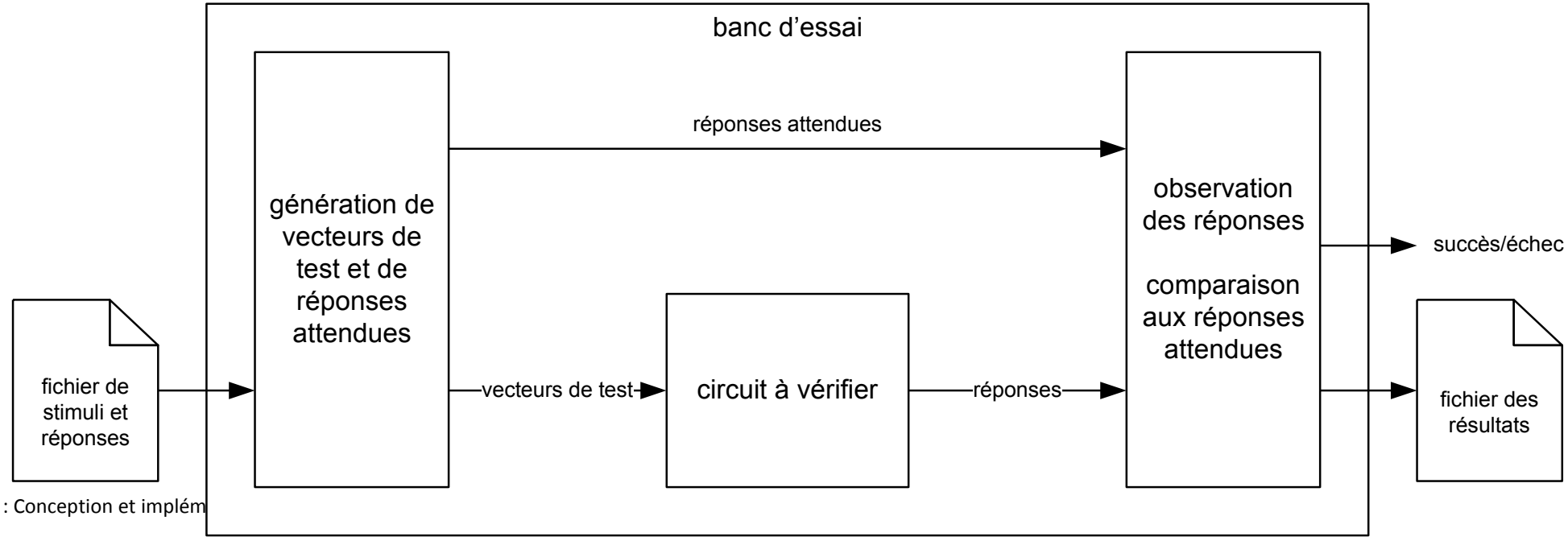
La vérification d'un circuit

- La vérification a pour but de confirmer qu'un circuit rencontre bien ses spécifications.
- La vérification complète d'un circuit est normalement un problème très difficile.
- Dans l'industrie de la conception numérique, on considère en général que le processus de vérification nécessite autant d'efforts que le processus de conception lui-même.
- La vérification d'un circuit est un art qui repose sur la maîtrise de trois principes:
  - la compréhension de la spécification;
  - le contrôle des entrées et de signaux internes du circuit à vérifier; et,
  - l'observation des sorties, des signaux internes et de l'état du circuit à vérifier.



Vérification par banc d'essai

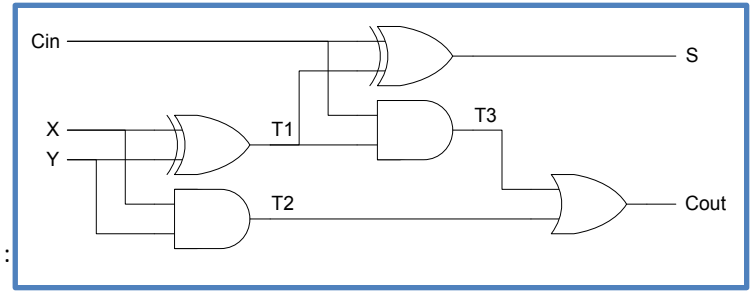
- Un banc d'essai doit effectuer les tâches suivantes :
  - instancier le circuit à vérifier;
  - générer des vecteurs de test et les appliquer aux ports d'entrée du circuit;
  - [utile]: générer automatiquement des réponses attendues aux vecteurs de test;
  - [utile]: comparer les réponses du circuit aux réponses attendues,
    et indiquer toute différence entre les deux par une condition d'erreur;
  - [facultatif]: lire des stimuli d'un fichier et écrire les réponses dans un fichier.



Banc d'essai complet avec observation et évaluation des réponses

```
entity add3bitsTB is
end add3bitsTB;

architecture arch5 of add3bitsTB is

signal Cin, X, Y : std_logic;
signal Cout, S : std_logic;

function somme3bits(vec: std_logic_vector(2 downto 0))
return std_logic_vector is
variable lasomme : std_logic_vector(1 downto 0);
begin
   case vec is
      when "000" => lasomme := "00";
      when "001" | "010" | "100"  => lasomme := "01";
      when "011" | "110" | "101"  => lasomme := "10";
      when "111" => lasomme := "11";
      when others => lasomme := "XX";
   end case;
   return lasomme;

end somme3bits;
```

```
begin

   UUT : entity add3bits(flotdonnees)
         port map (Cin, X, Y, Cout, S);

   process
   variable stim : std_logic_vector(2 downto 0);
   begin
      for k in 0 to 7 loop
         (Cin, Y, X) <= to_unsigned(k, 3);
         wait for 10 ns;

         assert somme3bits(Cin & Y & X) = Cout & S
            report "erreur pour l'entrée "
            & integer'image(k) severity warning;

      end loop;

      report "simulation terminée" severity failure;

   end process;

end arch5;
```



Vérification hiérarchique

- Une approche hiérarchique correspond à la structure du système et simplifie l'effort de vérification.
- Trois niveaux:
  - Modules (p. ex. additionneur ou décodeur):
    - comportement simple, pleine visibilité, test exhaustif
    - {*génie logiciel: test unitaire*}
  - Unités (p. ex. une UAL):
    - choisir des vecteurs de test à partir des spécifications
    - bonne visibilité de l'interface entre les modules
    - vérification des interactions entre les modules
    - {*génie logiciel: test d'intégration*}
  - Système:
    - vérifier les fonctionnalités de haut niveau
    - vérifier que les interconnections entre les unités sont correctes
    - {*génie logiciel: test de système*}

Élaboration d'un plan de test

- Un plan de test détaille la stratégie employée pour effectuer la vérification d'un système.
- La complexité et le niveau de détail d'un plan de test dépendent de la taille du module, de l'unité ou du système à vérifier.
- Un plan de test complet détaille:
  - la couverture du test: quels fonctionnalités de la spécification doivent être vérifiées par le test?
  - les méthodes de test: comment le système sera-t-il vérifié? quelles sont les conditions de réussite et d'échec des cas de test?
  - les responsabilités de test: qui est responsable de quels tests?

- Un plan de test peut être élaboré en trois étapes :
  1. Extraire les fonctionnalités requises du système à partir de ses spécifications;
  2. Prioriser les fonctionnalités à vérifier; et,
  3. Créer des vecteurs de test.

Élaboration d’un plan de test
1. Extraire les fonctionnalités requises du système

- La spécification architecturale décrit l’interface du système avec le monde extérieur et ses relations d’entrées et sortie.
- La spécification architecturale ne spécifie pas comment le comportement interne du système doit être réalisé.
- Il est difficile d’extraire toutes les fonctionnalités requises à partir de la spécification. Il est encore plus difficile d’automatiser cette extraction.
- Certaines fonctionnalités sont implicites. D’autres fonctionnalités ou comportements ne doivent pas être présents.

Concevoir et décrire une file d'attente en VHDL. Le nombre maximum d'éléments dans la file et la largeur des données en bits doivent être paramétrables. Un signal d'horloge synchronise toutes les activités de la file. Le signal reset permet de vider la file. Les ports din et dout sont respectivement l'entrée et la sortie de la file. Les ports empty et full indiquent respectivement que la file est vide ou bien qu'elle est pleine. Quand le signal wr_en (*write enable*) est activé, la valeur placée sur le port din sera insérée dans la file lors de la prochaine transition active du signal d'horloge clk. Quand le signal rd_en (*read enable*) est activé, la valeur qui est dans la file depuis le plus longtemps sera placée sur le port dout lors de la prochaine transition active du signal d'horloge clk. Il doit être possible d'écrire une nouvelle valeur dans la file et de lire la valeur la plus ancienne simultanément. Quand la file est pleine, le signal wr_en n'a aucun effet - il n'est pas possible d'écrire par dessus le contenu présent de la file. Quand la file est vide, le signal rd_en n'a aucun effet. Il doit être possible d'écrire une nouvelle valeur dans la file et de lire la valeur la plus ancienne simultanément.

Élaboration d'un plan de test
1. Extraire les fonctionnalités requises du système

Quatre catégories de fonctionnalités à vérifier	Exemple de fonctionnalité
1. Il est possible de placer le système dans son état de départ valide à partir de n'importe quel état.	Retour à l'état de départ quand le signal reset est activé.
2. À partir d'un état valide et étant donnée une entrée valide, le système doit se placer dans le bon état valide.	Écriture dans la file (quand la file n'est pas pleine).
3. À partir d'un état valide et étant donnée une entrée valide, le système ne doit pas se placer dans un état non valide ou un état incorrect.	Écriture dans la file (quand la file n'est pas pleine).
4. À partir d'un état valide et étant donnée une entrée non valide, le système ne doit pas se placer dans un état non valide ou un état incorrect.	Écriture dans la file (quand la file est pleine).

Élaboration d’un plan de test
2. Prioriser les fonctionnalités à vérifier

- Il y a souvent une très grande quantité de fonctionnalités à vérifier.
- Il peut être utile de les placer en ordre de priorité.
  1. Les fonctionnalités qui correspondent à des contraintes de sécurité devraient être considérées en premier.
  2. Les fonctionnalités qui correspondent à des besoins fondamentaux du système identifiés dans les spécifications devraient être vérifiées de façon prioritaire.
  3. Les fonctionnalités qui faciliteraient l’utilisation du système peuvent avoir une priorité plus faible.
  4. Enfin, les fonctionnalités facultatives ou futures du système peuvent avoir une priorité encore plus faible.

Élaboration d'un plan de test
3. Créer un ensemble de vecteurs de test

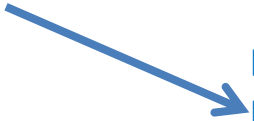
- On crée un ensemble de vecteurs de test pour chaque fonctionnalité à vérifier.
- Il faut établir comment vérifier la fonctionnalité et comment établir qu'elle est satisfaite ou non à partir des signaux du système.
- On détermine les vecteurs de test spécifiques qui permettent de stimuler la fonctionnalité désirée.
- Le plan de test peut comporter un tableau dans lequel on identifie
  - les fonctionnalités à vérifier
  - leur priorité
  - la personne responsable
  - le statut : en développement, débutée, écrite, appliquée, révisée, etc.

Fonctionnalité	Priorité	Responsable	Statut
1. Réinitialisation	1	Armand	Assignée
2. Écrire dans la pile	2	Armand	Écrite
3. Lire de la pile	2	Asma	Écrite
4. Écriture et lecture simultanées	2	Alexis	En développement
5. Pile pleine a. Fonctionnement du signal 'full' b. Écriture impossible	2	Alexis	En développement
6. Pile vide a. Fonctionnement du signal 'empty' b. Lecture impossible	2	Armand	Assignée
7. Signaux 'overflow' et 'underflow'	4	Armand	Assignée

Cinq qualités d'un bon ensemble de vecteurs de test

- Les cinq qualités principales d'un bon ensemble de vecteurs de test sont:
  - Efficace pour découvrir des bogues, c'est-à-dire que chaque vecteur de test vérifie plusieurs fonctionnalités en même temps, et donc que peu de vecteurs de tests sont nécessaires.
  - Identifie la source des bogues, pour aider à leur éradication.
  - Reproductible : il est facile de recréer le bogue.
  - Automatisé à l'aide d'un banc d'essai.
  - Exécutable dans un temps raisonnable.
    - dépend de la taille et de la complexité du circuit et de l'ampleur du projet

- Le problème de la création d'un ensemble de vecteurs de test n'est pas simple.
- Les vecteurs de test choisis doivent permettre de vérifier que le circuit rencontre ses spécifications.



Laboratoire de 3 heures: quelques minutes
Projet intégrateur: de plusieurs minutes à quelques heures
Système de grande envergure: de plusieurs heures à quelques jours

Analyse statique du code

- La meilleure façon de réduire le nombre de bogues dans un design est de réduire les chances de leur introduction dans celui-ci.
- En adoptant des pratiques de codage favorisant la vérification, on augmente les chances d'atteindre ce but.
- Guides de codage:
  - développés avec le temps en se basant sur l'expérience des concepteurs;
  - restreignent l'utilisation d'un langage à un sous-ensemble robuste et sécuritaire.
- Il existe des outils de vérification du respect des guides de codage qui effectuent une analyse statique du code.
- Le premier programme de la sorte s'appelait 'lint', et ce terme est souvent appliqué aux outils comme au processus.
- En VHDL et dans les autres HDL, une analyse statique peut détecter des erreurs potentielles et augmenter la qualité du code.
- Ces erreurs potentielles peuvent se situer à plusieurs niveaux:
  - Opérandes de largeurs différentes dans une expression;
  - Énoncés conditionnels dont tous les cas ne sont pas couverts et qui créent des états implicites;
  - Énoncés conditionnels qui se recoupent;
  - Nom d'entité différent du nom du fichier;
  - Insertion de signaux de contrôle dans le chemin d'une horloge;
  - Signaux ou variables qui ne sont pas initialisés avant d'être utilisés;
  - Signaux ou variables auxquels on n'assigne jamais de valeur ou qui ne sont jamais utilisés; et,
  - Expression constante dans une structure de condition.

Détection, identification et suivi des bogues

- Le but de la sélection des vecteurs de test et leur application au circuit en simulation est de découvrir les bogues du circuit.
- Un graphique de la fréquence d'apparence de nouveaux bogues dans un système donne une indication générale de la fiabilité de celui-ci.

- Quand un bogue survient:

1. Identifier les conditions où le bogue est devenu apparent: version du code, suite de vecteurs de tests qui a mené à la défaillance, paramètres de simulation.
2. Déterminer la source du bogue: une stratégie consiste à réduire la portée du circuit ou de l'ensemble de vecteurs de test.
3. Documenter le bogue dans un système de suivi: enregistrer le bogue, les conditions pour le produire, son statut, sa priorité, les actions prises pour le résoudre.

Vous devriez maintenant être capable de …

- Le plan de test: expliquer et appliquer les trois étapes à suivre. (B2, B3)
- Expliquer les cinq qualités d'un bon ensemble de vecteurs de test. (B2, B3)
- Décrire les principes d'analyse statique du code et du suivi des bogues. (B2)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance – mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.