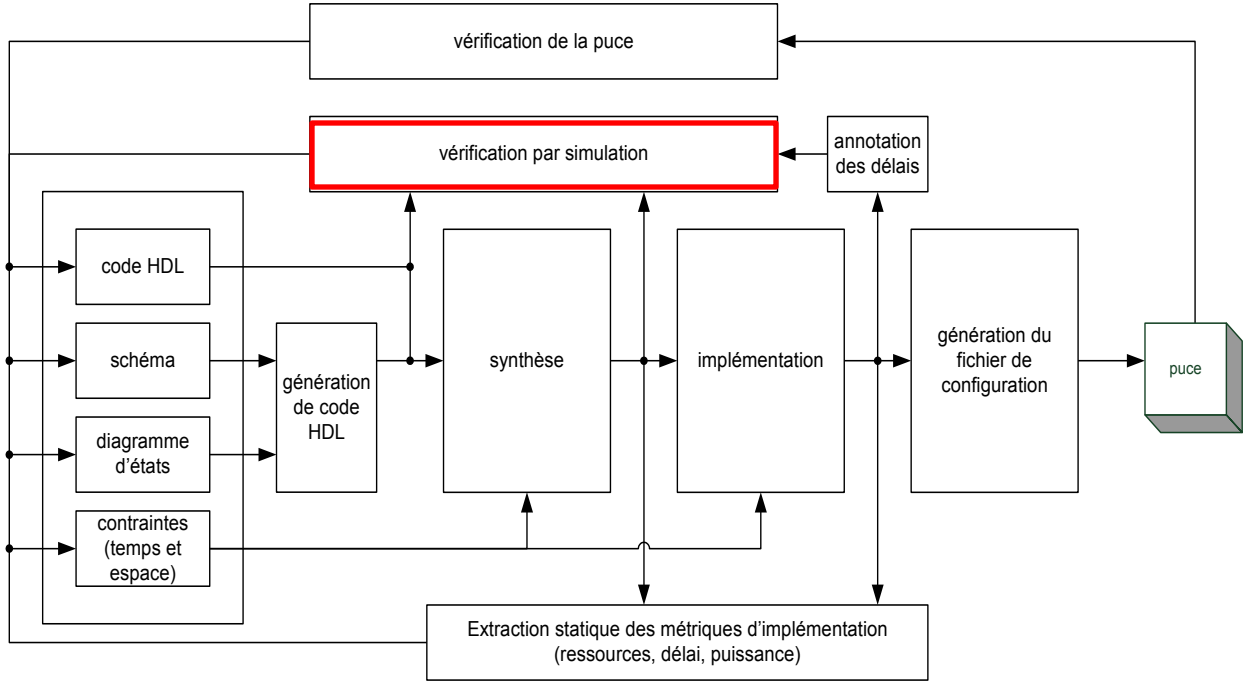
Banc d’essai pour un circuit combinatoire

Sujets de ce thème

- Structure d'un banc d'essai
- Assignation concurrente avec clause `after`
- Énoncé `wait`
- Énoncés `assert`, `report` et `severity`
- Banc d’essai qui stimule un circuit
- Banc d’essai qui stimule et vérifie un circuit

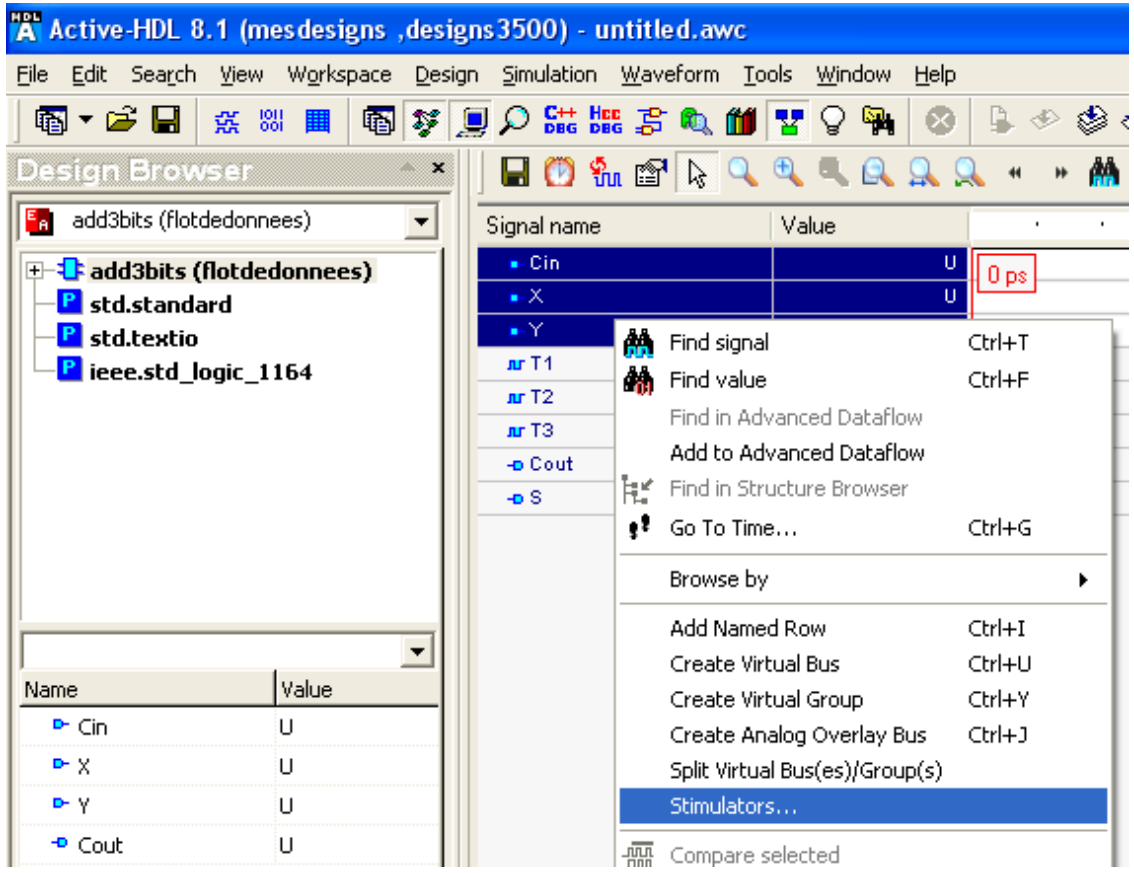
La vérification d'un circuit

- 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.



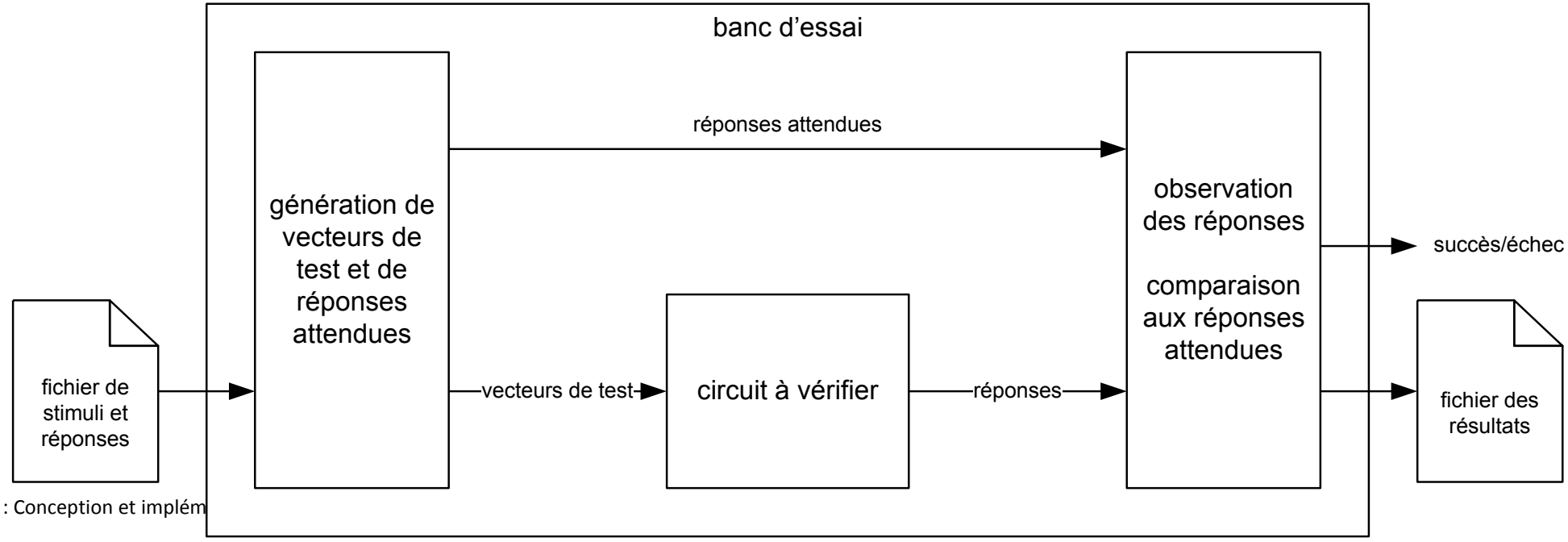
Vérification à l'aide d'un simulateur

- La vérification sans banc d'essai est satisfaisante pour les circuits peu complexes.
- Un simulateur permet de stimuler rapidement un circuit et d'en inspecter les sorties.
- Différents types de stimulateurs pour les ports d'entrée:
  - horloge
  - formule
  - valeur
  - compteur
  - valeur ou forme prédéfinie
  - séquence pseudo aléatoire



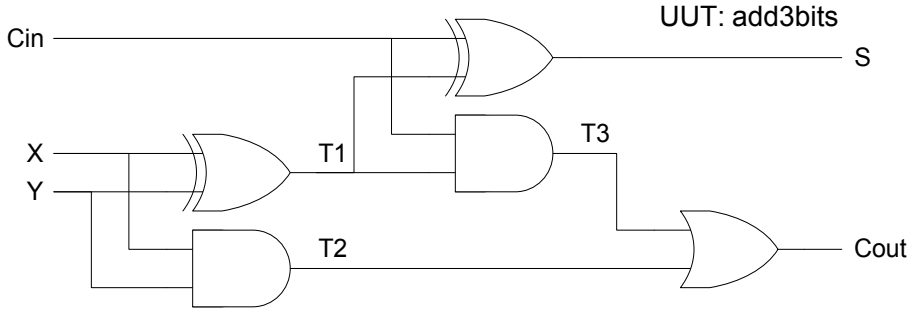
Vérification par banc d'essai

- 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.

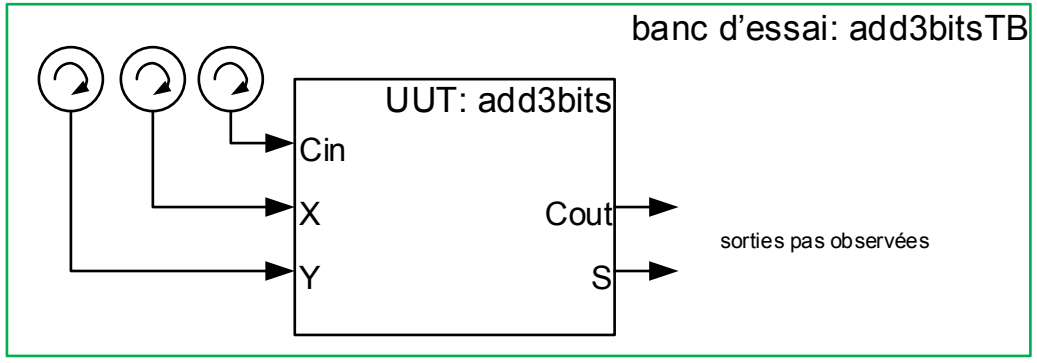


Exemple 1: additionneur à 3 bits et un banc d'essai pour le stimuler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add3bits is
   port (
      Cin, X, Y : in std_logic;
      Cout, S : out std_logic
   );
end add3bits;

architecture flotdonnees of add3bits is
signal T1, T2, T3 : std_logic;
begin
   S <= T1 xor Cin;
   Cout <= T3 or T2;
   T1 <= X xor Y;
   T2 <= X and Y;
   T3 <= Cin and T1;
end flotdonnees;
```



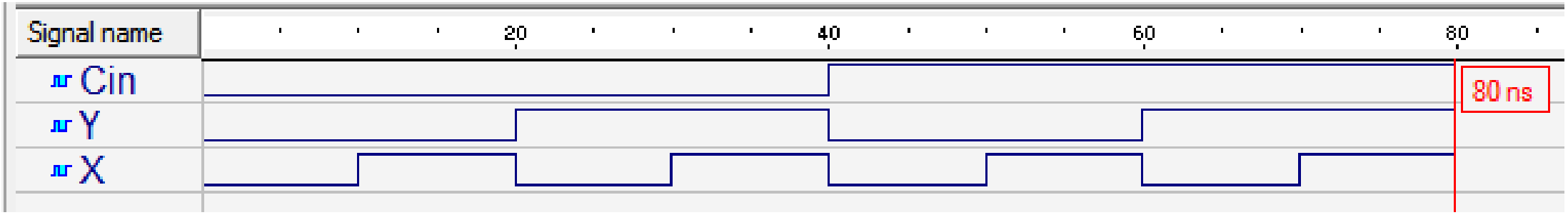

```
library ieee;
use ieee.std_logic_1164.all;

entity add3bitsTB is
end add3bitsTB;

architecture arch1a of add3bitsTB is

signal Cin, X, Y : std_logic; -- signaux pour les vecteurs de tests
signal Cout, S : std_logic; -- signaux pour les réponses

begin

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

   Cin <= '0' after 0 ns, '1' after 40 ns;
   Y <= '0' after 0 ns, '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
   X <= '0' after 0 ns, '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
        '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns;

end arch1a;
```



Clause `after` dans l'assignation d'un signal

- La génération des vecteurs de test peut être faite à l'aide de la clause `after` associée à des assignations concurrentes.
- La clause `after` permet de spécifier le moment auquel un signal doit prendre une valeur.
- La clause `after` comporte une expression de temps, composée d'une quantité et d'une unité.
- L'unité « ns » signifie nanoseconde.
- La simulation débute au temps T = 0 s.

```
Cin <= '0' after 0 ns, '1' after 40 ns;
Y <= '0' after 0 ns, '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
X <= '0' after 0 ns, '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
     '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns;
```



Banc d'essai amélioré: énumération des vecteurs de test

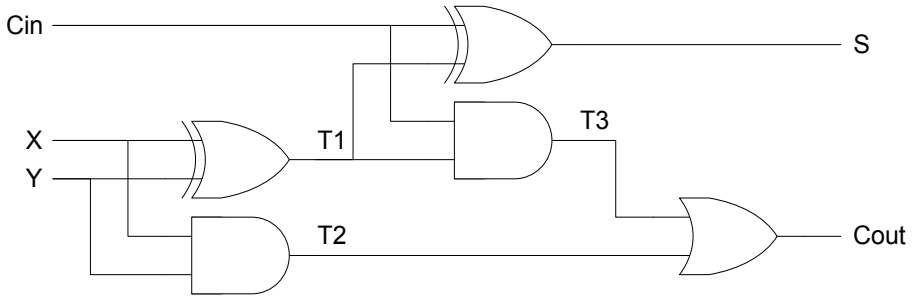


Tableau constant contenant les vecteurs de test

Application de tous les vecteurs à l'aide d'une boucle

Assignation combinée par concaténation

L'énoncé `wait` suspend l'exécution, force les signaux à se propager dans l'UUT, permet d'observer la réponse

L'énoncé `report` affiche un message à la console, et la clause severity interrompt la simulation.

```
library ieee;
use ieee.std_logic_1164.all;

entity add3bitsTB is
end add3bitsTB;

architecture arch2 of add3bitsTB is

signal Cin, X, Y : std_logic; -- signaux pour les vecteurs de tests
signal Cout, S : std_logic; -- signaux pour les réponses

type tableauSLV3 is array (natural range <>) of std_logic_vector(2 downto 0);
constant vecteurs : tableauSLV3 :=
             ("000", "001", "010", "011", "100", "101", "110", "111");

begin

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

   process -- application des vecteurs de test emmagasinés dans le tableau
   begin
      for k in vecteurs'low to vecteurs'high loop
         (Cin, Y, X) <= vecteurs(k);
         wait for 10 ns;
      end loop;

      report "simulation terminée" severity failure;

   end process;

end arch2;
```

Banc d'essai amélioré: stimulation exhaustive systématique

Package `numeric_std` pour le type `unsigned` et la fonction `to_unsigned()`.

On détermine arbitrairement qu'il y a 8 combinaisons d'entrées correspondant aux nombre 0 à 7.

1. Conversion de type d'entier à `unsigned`
(k est implicitement entier parce que c'est un compteur de boucle).

2. L'objet composite (Cin, Y, X) est un tableau de `std_logic`.

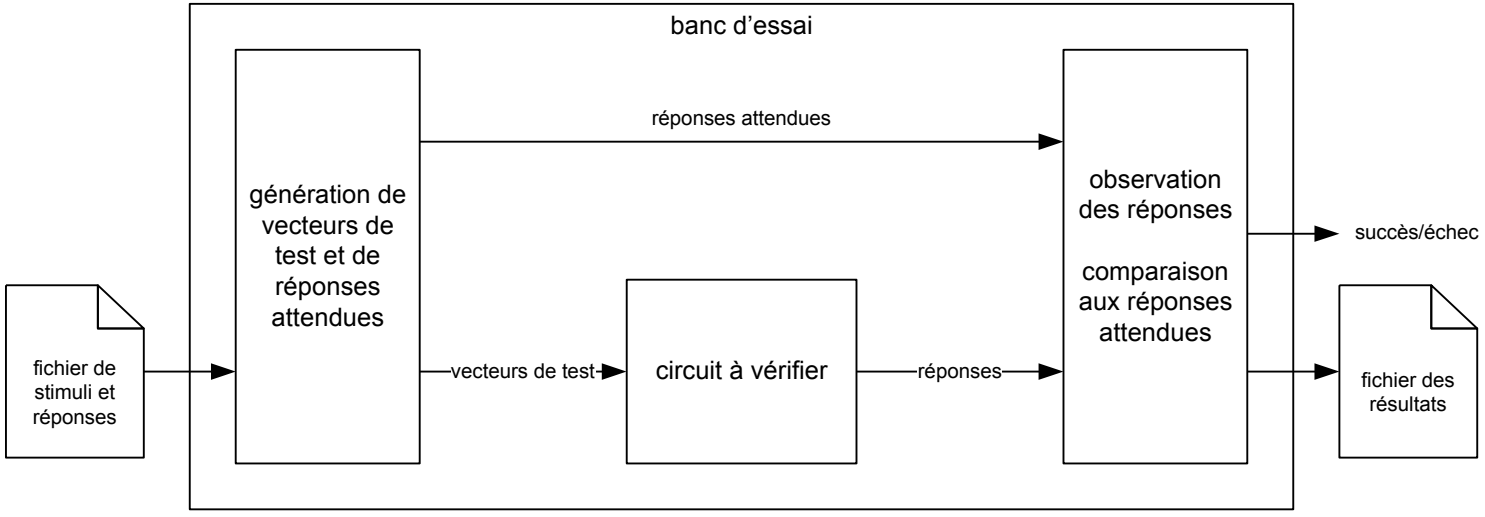
3. Le type `unsigned` est défini comme un tableau de `std_logic` alors l'assignation est permise.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity add3bitsTB is
end add3bitsTB;

architecture arch3 of add3bitsTB is

signal Cin, X, Y : std_logic; -- signaux pour les vecteurs de tests
signal Cout, S : std_logic; -- signaux pour les réponses

begin

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

   process -- application exhaustive des vecteurs de test
   begin
      for k in 0 to 7 loop
         (Cin, Y, X) <= to_unsigned(k, 3);
         wait for 10 ns;
      end loop;

      report "simulation terminée" severity failure;

   end process;

end arch3;
```

Banc d’essai complet avec observation et évaluation des réponses

- L’observation et la comparaison automatisées des réponses du module à vérifier est une approche très puissante qui économise beaucoup de temps pour tous les circuits non triviaux.
- Pour toute condition où le circuit ne produit pas les réponses attendues, le banc d’essai devrait générer un message d’avertissement indiquant
  - le moment où l’erreur s’est produite;
  - la valeur du vecteur de test appliqué;
  - la valeur des réponses observées; et,
  - la valeur des réponses attendues.

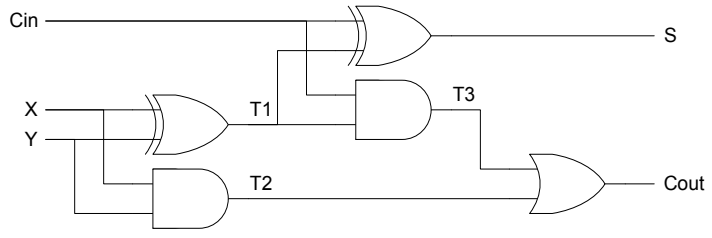
- La réponse attendue devrait être générée par un algorithme différent de celui utilisé dans l’unité à vérifier.



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;
```



L'opérateur & signifie la concaténation.

`integer'image(k)` retourne une chaîne de caractères représentant l'entier k

Décortiquer l'énoncé `assert – report` (1)

- L'énoncé `assert` (en anglais: prétendre) est utilisé pour faire un test d'équivalence entre deux expressions.
  On pourrait le remplacer par un énoncé if.
- Si les deux expressions ne sont pas équivalentes, alors l'énoncé `report` est exécuté.

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

Égalité?

Si non, alors le message est affiché à la console.

On associe un niveau de sévérité au message.
Il y en a quatre: `Note`, `Warning`, `Error` et `Failure`.
La couleur du message affiché peut varier en fonction du niveau de sévérité, et la simulation peut être interrompue.

Décortiquer l'énoncé `assert – report` (2)

- Un énoncé est nécessaire pour que la simulation se termine.
- En VHDL, les processus relancés automatiquement après chaque exécution. Si cet énoncé n'est pas présent, après la dernière itération de la boucle l'exécution du processus est automatiquement relancée.
- C'est le niveau de `severity` et la configuration du simulateur qui déterminent si la simulation se termine ou non.

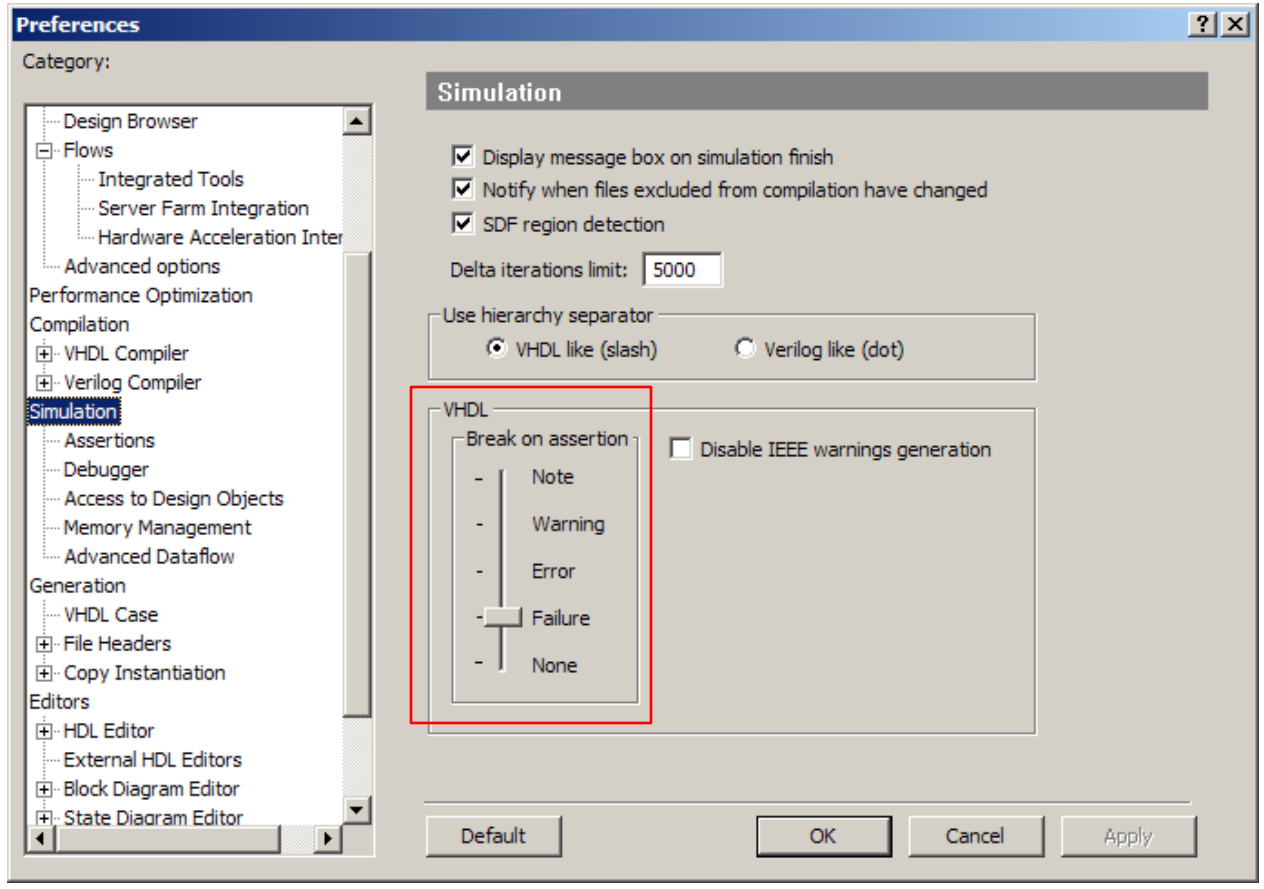
```
report "simulation terminée" severity failure;
```

est équivalent à

```
assert false report "simulation terminée" severity failure;
```

Quatre niveaux de `severity` pour une assertion

- Définis dans le package `standard`, disponible dans le manuel de référence du langage VHDL, norme 1076-2002 (section 14.2)
  `type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);`
- On peut spécifier au simulateur à quel niveau terminer la simulation.



Exemple 2: module qui identifie les nombres premiers

- Le module `detecteurPremier` accepte en entrée un vecteur de 6 bits I représentant un nombre entier non signé.
- Sa sortie F indique si le nombre est premier ou non.
- La définition comporte une erreur: le nombre 63 n'est pas premier.
- On souhaite composer un banc d'essai pour stimuler le module de façon exhaustive et vérifier sa sortie automatiquement.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity detecteurPremier is
   port (
      I : in unsigned(5 downto 0);
      F : out std_logic
   );
end detecteurPremier;

architecture flotdonnees of detecteurPremier is
begin
   with to_integer(I) select
      F <=
      '1' when 2 | 3 | 5 | 7 | 11 | 13 | 17 |
            19 | 23 | 29 | 31 | 37 | 41 | 43 |
            47 | 53 | 59 | 61 | 63, -- erreur!
         '0' when others;
end flotdonnees;
```

Exemple 2: module qui identifie les nombres premiers – banc d'essai

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
use ieee.math_real.all;

entity detecteurPremierTB is
end detecteurPremierTB;

architecture arch1 of detecteurPremierTB is

signal I : unsigned(5 downto 0);
signal F : std_logic;

function estPremier(n: integer) return boolean is
variable reponse : boolean := false;
begin
   if (n <= 1) then
      reponse := false; -- 0 et 1 ne sont pas premiers
   elsif (n <= 3) then
      reponse := true; -- 2 et 3 sont premiers
   else
      reponse := true;
      for k in 2 to integer(ceil(sqrt(real(n)))) loop
         if (n mod k = 0) then
            reponse := false;
            exit;
         end if;
      end loop;
   end if;
   return reponse;
end estPremier;
```

Modèle non synthétisable de la fonction à vérifier.
Ce modèle est-il correct?

```
begin

   UUT : entity detecteurPremier(flotdonnees) port map (I, F);

   process
   constant kmax : integer := 63;
   begin

      for k in 0 to kmax loop

         I <= to_unsigned(k, I'length);
         wait for 10 ns;

         assert (estPremier(k) = (F = '1'))
            report "erreur pour l'entrée "
            & integer'image(k) severity warning;

      end loop;

      report "simulation terminée" severity failure;

   end process;

end arch1;
```

`integer'image(k)` retourne une chaîne de caractères représentant l'entier k

Vous devriez maintenant être capable de …

- Décrire la structure d'un banc d'essai à l'aide d'un diagramme. (B2)
- Utiliser un énoncé d'assignation concurrente comportant une clause `after`. Utiliser l'énoncé `wait`. (B3)
- Utiliser les énoncés `assert`, `report` et `severity` dans un banc d'essai. (B3)
- Donner le code VHDL du banc d'essai d'un circuit combinatoire. (B3)
- Donner le chronogramme correspondant à la simulation d'un module combinatoire par un banc d'essai. (B3)

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.