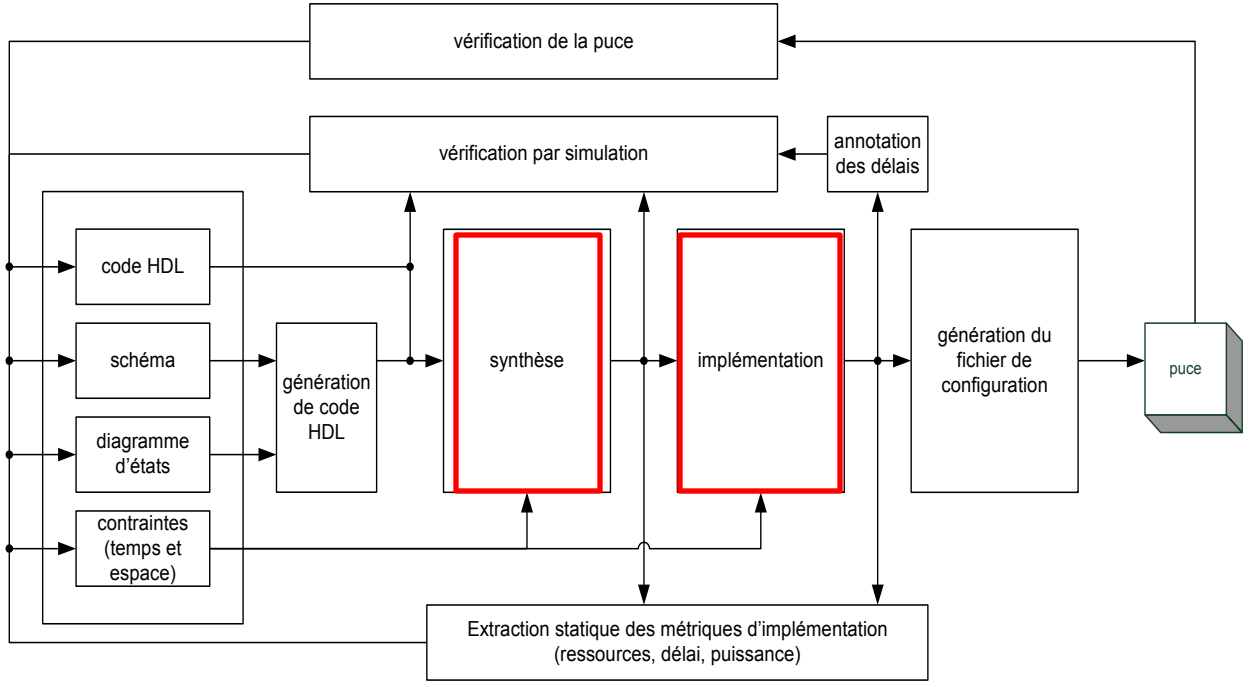
Synthèse et implémentation d'un circuit combinatoire

Sujets de ce thème

- La synthèse et l’implémentation, c’est quoi
- Étapes de la synthèse et de l’implémentation
- Codes synthétisables et non synthétisables

La synthèse et l'implémentation d'un modèle VHDL

- La synthèse du code VHDL est effectuée par un synthétiseur.
- Le produit du synthétiseur est communément appelé «liste des interconnexions» (*netlist*), qui inclut:
  - la liste des composantes de base utilisées; et,
  - les liens qui les relient.
- Après le processus de synthèse, il est possible d'obtenir un estimé de la performance et des ressources utilisées par le circuit.
- L'implémentation consiste à choisir des ressources spécifiques sur le FPGA où instancier les composantes et les interconnexions identifiées par le synthétiseur.



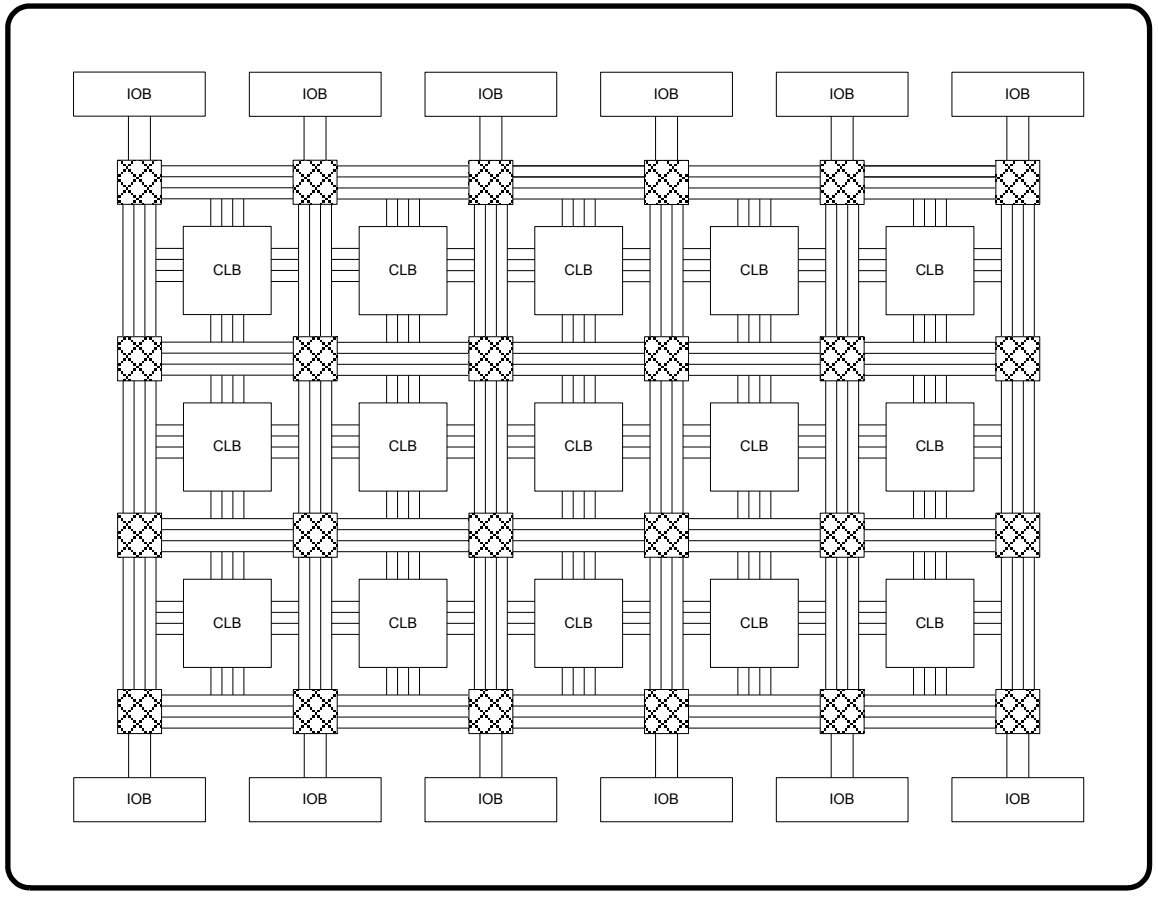
Défis de la synthèse à partir d'un modèle VHDL

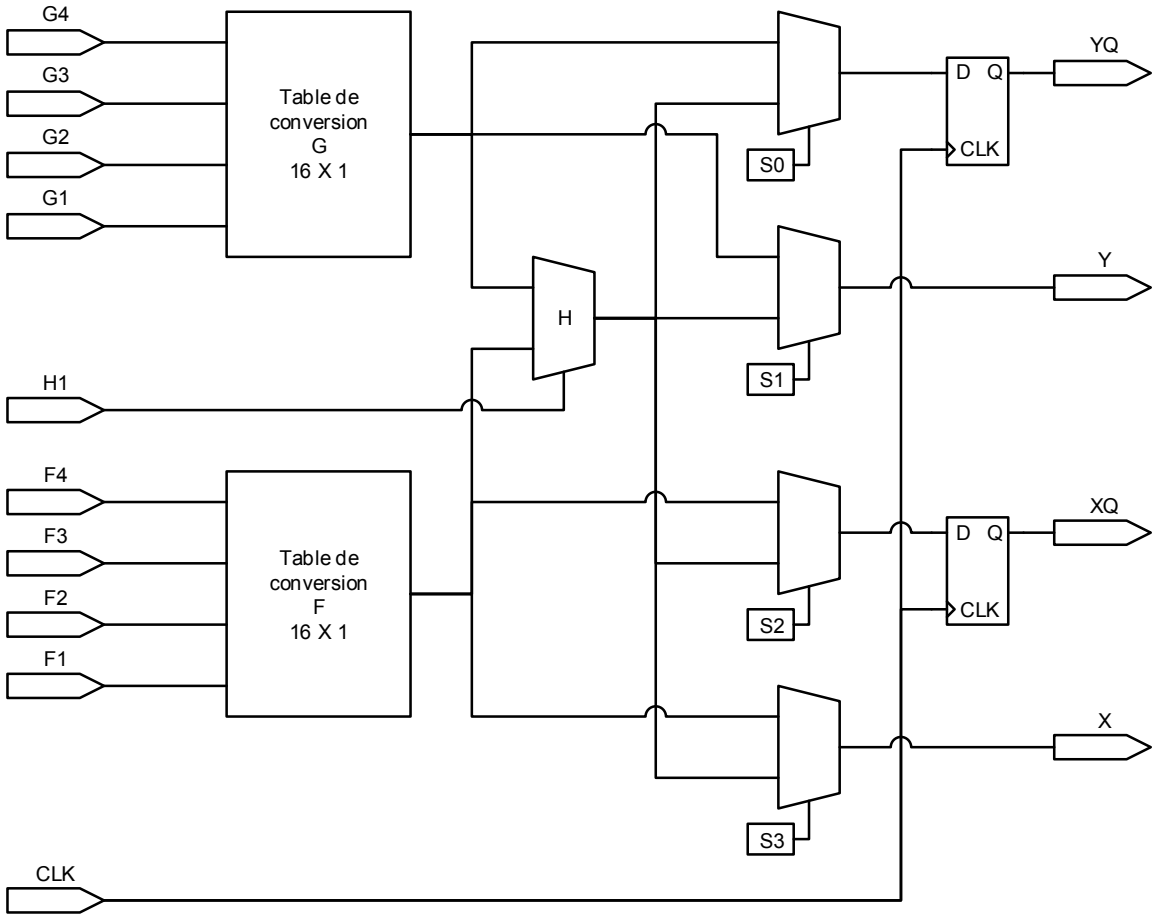
- Le défi du synthétiseur est d'inférer un circuit conforme au code.
- Le défi du concepteur de circuits numériques est de décrire ses intentions d'une façon non ambigüe pour le synthétiseur.
- La norme 1076.6-2004 de l'IEEE définit le sous-ensemble de VHDL qui est (devrait être …) supporté par les outils de synthèse commerciaux.
- La documentation accompagnant le synthétiseur utilisé est pratique pour connaître exactement les parties du langage qui sont supportées ou non, les formulations préférées pour représenter une structure donnée, ainsi que les directives particulières supportées par le synthétiseur.

Étapes de la synthèse d'un modèle combinatoire décrit en VHDL

1. Identifier les entrées et sorties du circuit.
2. Pour chaque sortie, trouver l'équation booléenne ou la table de vérité correspondante. Réduire l'équation si nécessaire.
3. Si la cible est un FPGA, découper les équations de sortie selon la taille des tables de correspondance disponibles.
4. Implémentation: choisir des ressources spécifiques sur la puce pour les ports et les fonctions logiques.

Rappel: ressources d'un FPGA





Code synthétisable – exemple 1

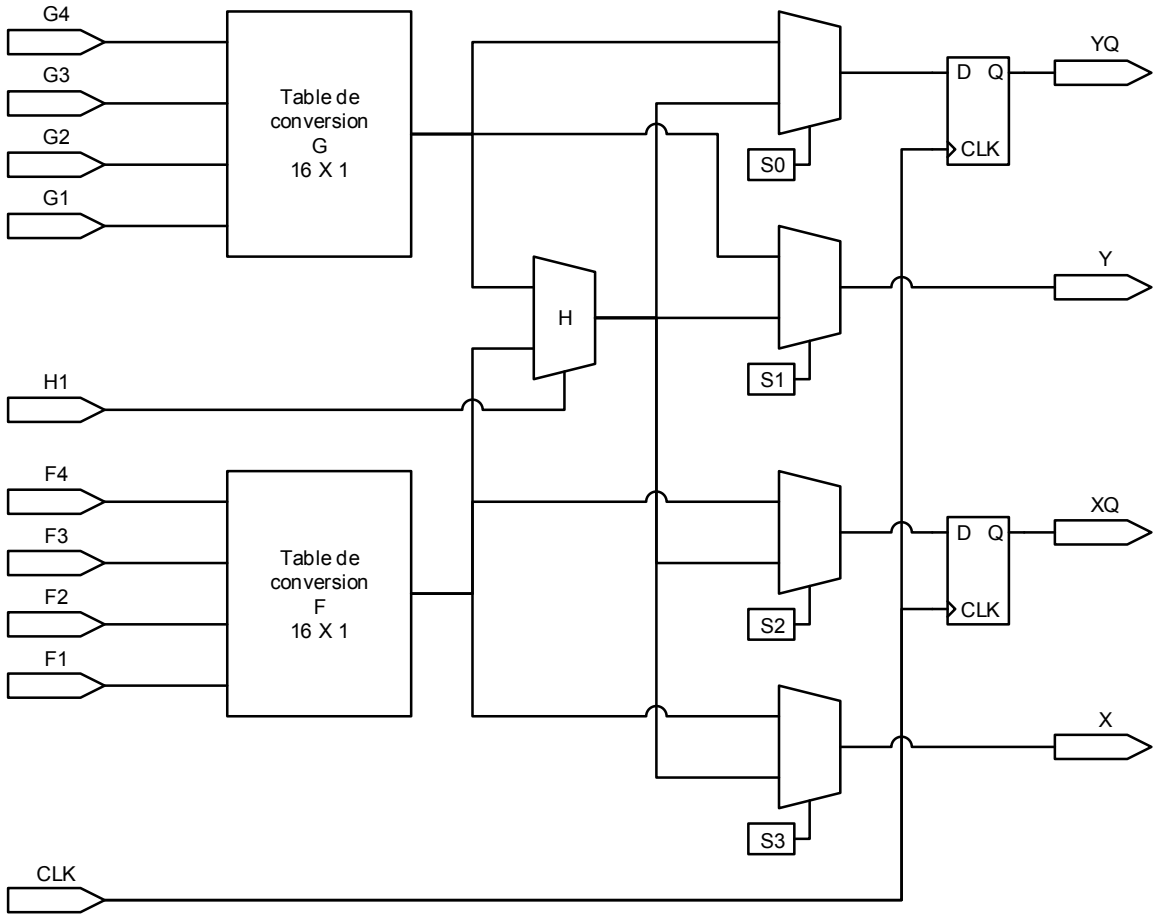
```
library ieee;
use ieee.std_logic_1164.all;

entity porteET4 is
   port (
      I : in std_logic_vector(3 downto 0);
      F : out std_logic
   );
end porteET4;

architecture comportementale1 of porteET4 is
begin
   process (I)
   begin
      F <= I(3) and I(2) and I(1) and I(0);
   end process;
end comportementale1;
```

1. Identifier les entrées et sorties du circuit.
2. Pour chaque sortie, trouver l'équation booléenne ou la table de vérité correspondante. Réduire l'équation si nécessaire.
3. Si la cible est un FPGA, découper les équations de sortie selon la taille des tables de correspondance disponibles.
4. Implémentation: choisir des ressources spécifiques sur la puce pour les ports et les fonctions logiques.

Code synthétisable – exemple 1 – porte ET à 4 entrées



G4 (I(3))	G3 (I(2))	G2 (I(1))	G1 (I(0))	G (F)
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Code synthétisable – exemple 2

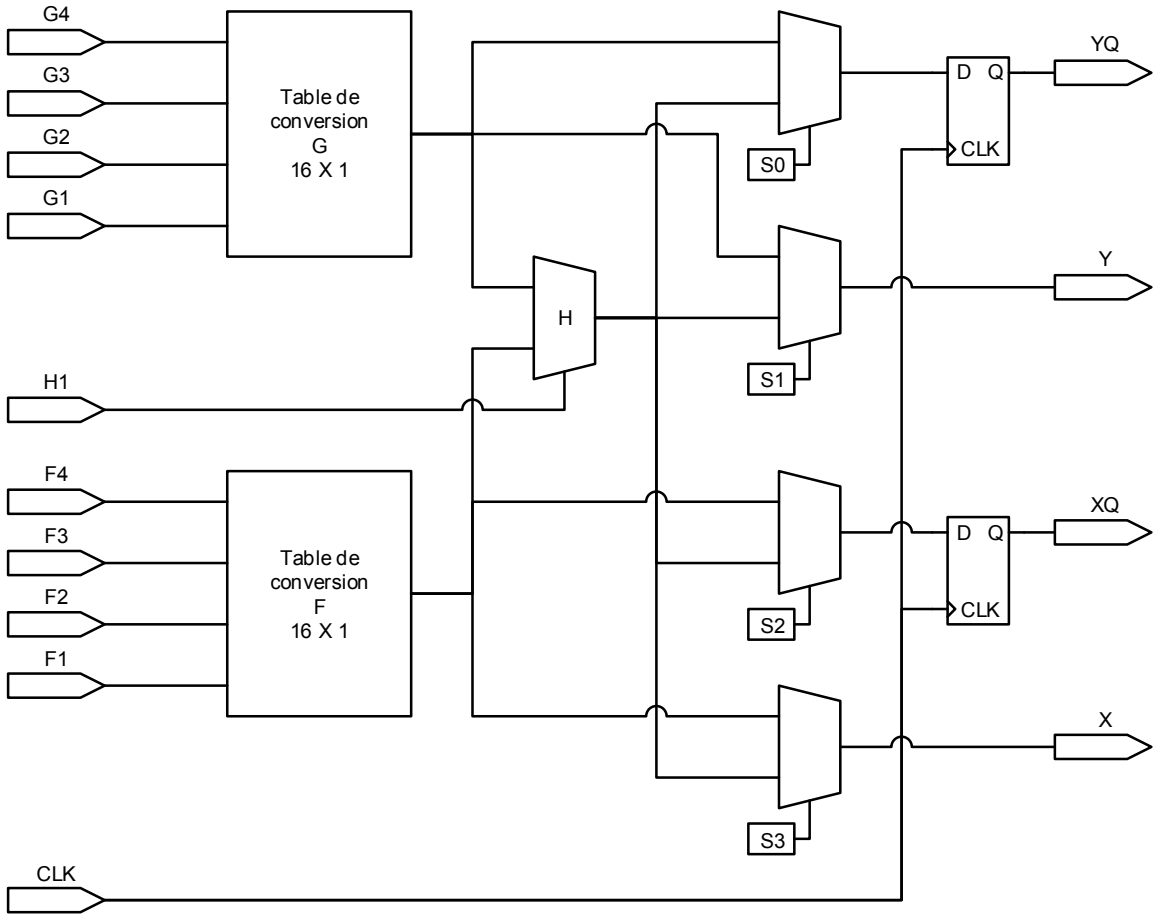
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity vote is
   port (
      lesvotes: in std_logic_vector(3 downto 0);
      approbation : out std_logic
   );
end vote;

architecture flotdonnees1 of vote is
begin
   with lesvotes select
      approbation <=
           '1' when "0111",
           '1' when "1011",
           '1' when "1101",
           '1' when "1110",
           '1' when "1111",
           '0' when others;
end flotdonnees1;
```

1. Identifier les entrées et sorties du circuit.
2. Pour chaque sortie, trouver l'équation booléenne ou la table de vérité correspondante. Réduire l'équation si nécessaire.
3. Si la cible est un FPGA, découper les équations de sortie selon la taille des tables de correspondance disponibles.
4. Implémentation: choisir des ressources spécifiques sur la puce pour les ports et les fonctions logiques.

Code synthétisable – exemple 2 – problème du vote



F4 (lesvotes(3))	F3 (lesvotes(2))	F2 (lesvotes(1))	F1 (lesvotes(0))	F (approbation)
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Synthèse de boucles et de conditions

- À l'intérieur d'un processus, on peut utiliser des boucles et des conditions pour modéliser le comportement d'un circuit.
- Les boucles sont une manière compacte de représenter plusieurs énoncés reliés logiquement entre eux.
- Les paramètres d'exécution de la boucle doivent prendre des valeurs statiques au moment de la synthèse.
- Les boucles sont implémentées en les déroulant: les énoncés d'assignation qu'elles contiennent sont répliqués, un pour chaque itération de la boucle.

- Pour les circuits combinatoires, les conditions permettent d'effectuer un choix.
  - L'énoncé `case` a l'avantage de représenter des choix qui sont mutuellement exclusifs et qui ont la même préséance. Il correspond assez exactement à l'action d'un multiplexeur.
  - L'énoncé `if`, est plus général avec les clauses `elsif` ainsi qu'une clause `else`. Il est possible de l'utiliser pour donner préséance à certaines conditions par rapport à d'autres. Cela peut résulter en un circuit plus complexe que nécessaire, parce que le comportement décrit peut être plus restrictif que ce que le concepteur a en tête.

`Case` est un meilleur choix quand on n'a pas besoin de priorité!

Code synthétisable

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

entity porteET4 is
   port (
      I : in std_logic_vector(3 downto 0);
      F : out std_logic
   );
end porteET4;
```

```
architecture comportementale1 of porteET4 is
begin
   process (I)
   begin
      F <= I(3) and I(2) and I(1) and I(0);
   end process;
end comportementale1;
```

```
architecture comportementale2 of porteET4 is
begin
   process (I)
   variable sortie : std_logic;
   begin
      sortie := '1';
      for k in 3 downto 0 loop
         sortie := sortie and I(k);
      end loop;
      F <= sortie;
   end process;
end comportementale2;
```

Code synthétisable et non-synthétisable

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

entity porteET is
   generic (
      W : positive := 8 -- le nombre d'entrées
   );
   port (
      W2 : in positive;
      I : in std_logic_vector(W - 1 downto 0);
      F : out std_logic
   );
end porteET;
```

La valeur de W est connue au moment de l'instanciation du module.

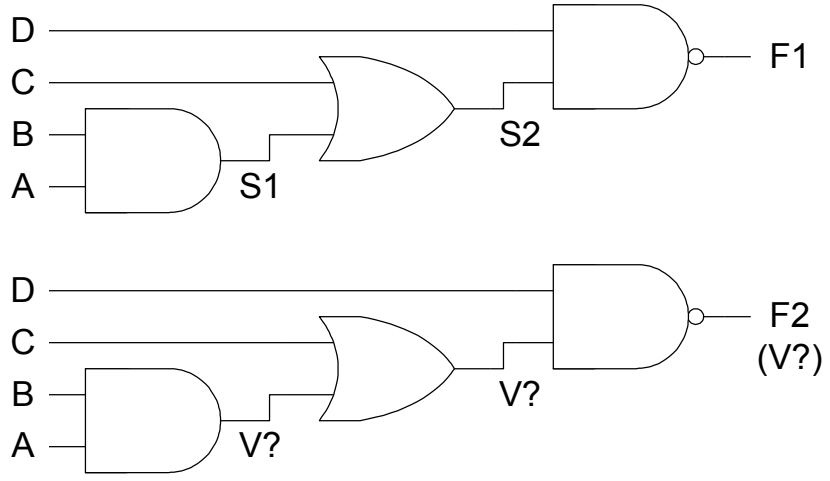
La valeur de W2 n'est pas connue au moment de l'instanciation du module.

```
architecture comportementale1 of porteET is
begin
   process (I)
   variable sortie : std_logic;
   begin
      sortie := '1';
      for k in W - 1 downto 0 loop
         sortie := sortie and I(k);
      end loop;
      F <= sortie;
   end process;
end comportementale1;
```

```
architecture comportementale2 of porteET is
begin
   process (I)
   variable sortie : std_logic;
   begin
      sortie := '1';
      for k in W2 - 1 downto 0 loop
         sortie := sortie and I(k);
      end loop;
      F <= sortie;
   end process;
end comportementale2;
```

Fils modélisés par les catégories `signal` et `variable`

- En général, l'utilisation de la catégorie `signal` résulte en un fil concret dans un module.
- Une équivalence directe est plus difficile à faire pour la catégorie `variable`.




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

entity demoSignalVariable is
   port (
      A, B, C, D: in std_logic;
      F1, F2 : out std_logic
   );
end demoSignalVariable;

architecture demo of demoSignalVariable is
signal S1, S2 : std_logic;
begin

   S1 <= A and B;
   S2 <= S1 or C;
   F1 <= S2 nand D;

   process(A, B, C, D)
   variable V : std_logic;
   begin
      V := A and B;
      V := V or C;
      V := V nand D;
      F2 <= V;
   end process;

end demo;
```

Synthétiser un banc d'essai?

- Les bancs d'essai ne sont pas synthétisables en général:
  - Énoncé `wait`
  - Clause `after`
  - Énoncé `assert – report`
  - Modèles de haut niveau de spécifications
  - Utilisation de types comme `real`.

```
...

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;

...
```

Vous devriez maintenant être capable de …

- Expliquer comment on peut effectuer la synthèse d'un module combinatoire décrit en VHDL. (B2)
- Effectuer la synthèse et l'implémentation du code d'un module combinatoire décrit en VHDL. (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.