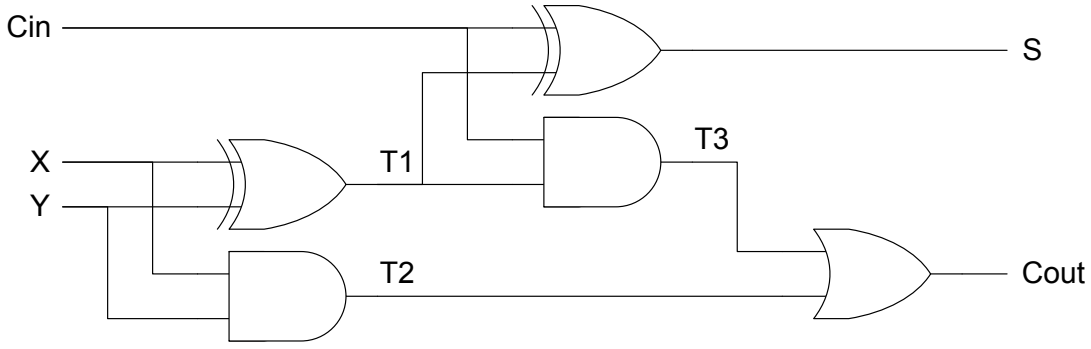
Description d'un circuit combinatoire en VHDL

Sujets de ce thème

- Description structurale.
- Description par flot de données.
- Description comportementale.
- Passer d’un style de description à l’autre.

Rappel: modèle VHDL d'un circuit combinatoire simple

- Librairie: définition de types, fonctions, etc.
- Entité: interface avec le monde extérieur
  - Définit les ports, leur type et leur direction
- Architecture: partie déclarative et corps
  - Définit le comportement du module
- Principe de la concurrence
  - L'ordre est sans importance dans le corps de l'architecture.




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

entity add3bits is

port (
   Cin : in std_logic;
   X : in std_logic;
   Y : in std_logic;
   Cout : out std_logic;
   S : out std_logic
);

end add3bits;

architecture flotDeDonnees of add3bits is

signal T1 : std_logic;
signal T2 : std_logic;
signal 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 flotDeDonnees;
```

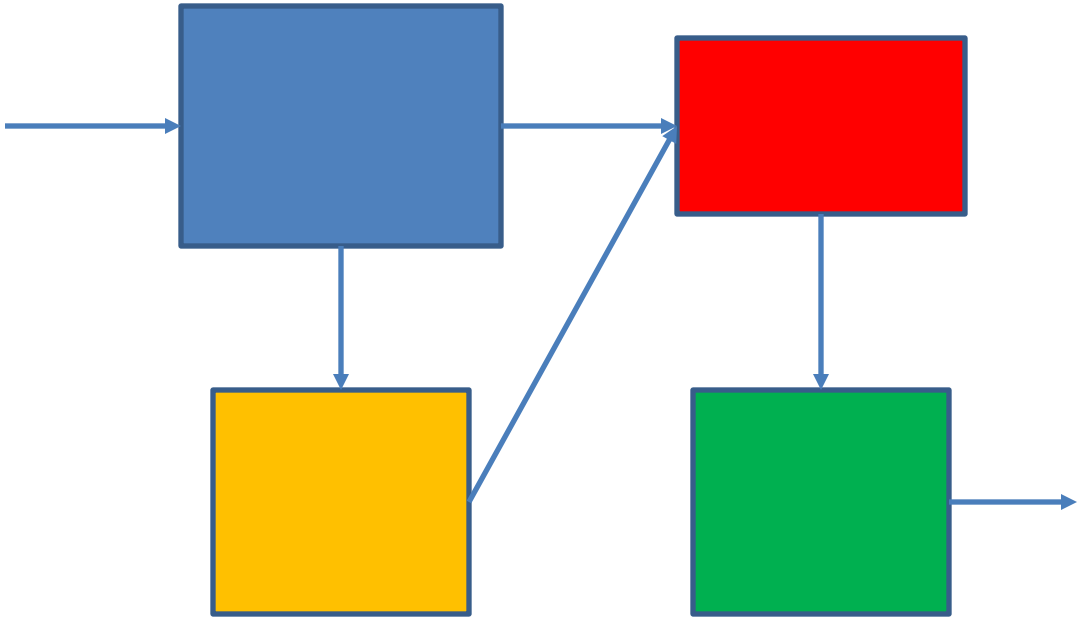
Trois styles de description d'un module

- À chaque style correspond un type d'énoncés concurrents utilisés.

Style de description	Énoncés concurrents
1. Structurale	Instanciations de composantes
2. Par flot de données	Assignations de signaux concurrentes, choisies et conditionnelles
3. Comportementale	Processus

1. Description structurale

- Un circuit numérique peut être défini par sa structure, c'est-à-dire par un assemblage de blocs.
- Une description structurale correspond à une description par schéma, où les instanciations de composantes et leurs interconnexions sont énumérées avec du texte.
- Une description structurale est appropriée pour relier entre eux différents sous-systèmes d’un système numérique.
- En général, il est préférable d’utiliser un éditeur de schéma pour décrire un tel circuit, et laisser un outil générer automatiquement le code VHDL structural.

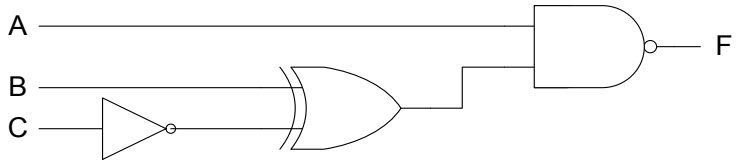


1. Description structurale

```
entity NAND2 is
   port(
      I0, I1 : in std_ulogic;
      O : out std_ulogic
   );
end NAND2;

architecture arch of NAND2 is
begin
   O <= not (I0 and I1);
end arch;
```

```
entity INV is
   port (
      I : in std_logic;
      O : out std_logic
   );
end INV;

architecture arch of INV is
begin
   O <= not I;
end arch;
```

```
entity XOR2 is
   port(
      I0, I1 : in std_ulogic;
      O : out std_ulogic
   );
end XOR2;

architecture arch of XOR2 is
begin
   O <= I0 xor I1;
end arch;
```



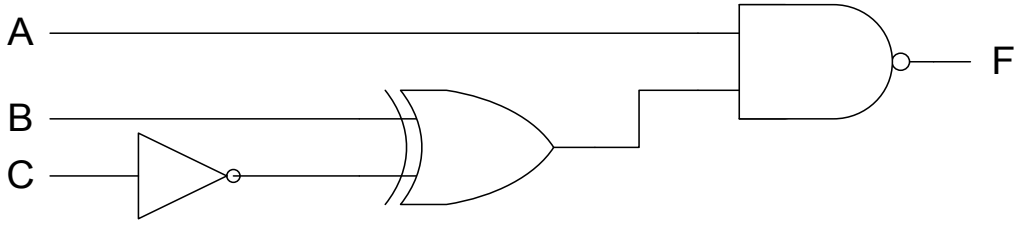

```
library ieee;
use ieee.std_logic_1164.all;

entity combinatoire1 is
   port (
      A, B, C : in std_logic;
      F : out std_logic
   );
end combinatoire1;

architecture structurale of combinatoire1 is

component INV
   port (I : in std_logic; O : out std_logic);
end component;

component NAND2
   port (I0, I1 : in std_logic; O : out std_logic);
end component;

component XOR2
   port (I0, I1 : in std_logic; O : out std_logic);
end component;

signal NET18, NET37 : std_logic;

begin
U1 : NAND2 port map(I0 => NET37, I1 => A, O => F);
U2 : XOR2 port map(I0 => NET18, I1 => B, O => NET37);
U3 : INV port map(I => C, O => NET18);

end structurale;
```

2. Description par flot de données

- Le modèle d'un circuit numérique par flot de données décrit sa fonction sans nécessairement définir sa structure.
- Les valeurs des signaux et ports du circuit sont établies par des assignations concurrentes de valeurs (*concurrent signal assignment*).
- Trois types d'énoncés concurrents:
  - avec des opérateurs logiques : `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`;
  - choisi (équivalent à un énoncé *case*): `with-select`;
  - conditionnel (équivalent à *if-else*): `when-else`.

2. Description par flot de données

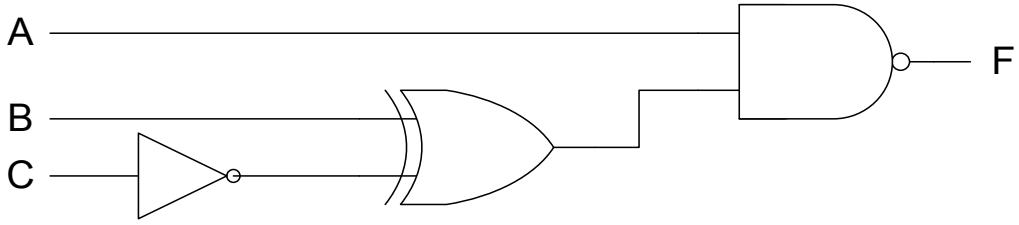
- Le modèle d'un circuit numérique par flot de données décrit sa fonction sans nécessairement définir sa structure.
- Les valeurs des signaux et ports du circuit sont établies par des assignations concurrentes de valeurs (*concurrent signal assignment*).
- Trois types d'énoncés concurrents:
  - avec des opérateurs logiques : `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`;
  - choisi (équivalent à un énoncé *case*): `with-select`;
  - conditionnel (équivalent à *if-else*): `when-else`.




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

entity combinatoire1 is
   port (
      A, B, C : in std_logic;
      F : out std_logic
   );
end combinatoire1;

architecture flotDeDonnees1 of combinatoire1 is
begin
   F <= not(A and (B xor not(C)));
end flotDeDonnees1;
```

2. Description par flot de données

- Le modèle d'un circuit numérique par flot de données décrit sa fonction sans nécessairement définir sa structure.
- Les valeurs des signaux et ports du circuit sont établies par des assignations concurrentes de valeurs (*concurrent signal assignment*).
- Trois types d'énoncés concurrents:
  - avec des opérateurs logiques : `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`;
  - choisi (équivalent à un énoncé *case*): `with-select`;
  - conditionnel (équivalent à *if-else*): `when-else`.

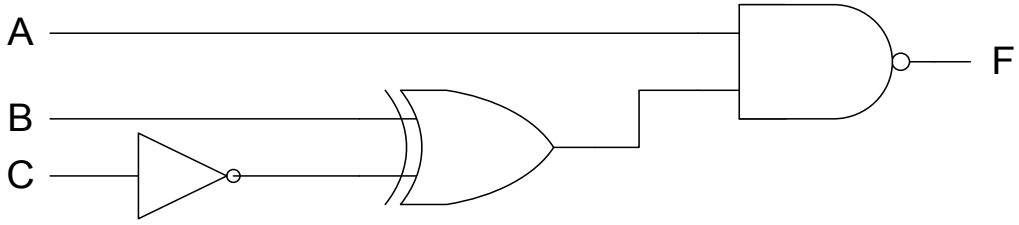



```
architecture flotDeDonnees2 of combinatoire1 is
signal entree : std_logic_vector(2 downto 0);
begin
   entree <= (A, B, C);
   with entree select
      F <=
         '1' when "000",
         '1' when "001",
         '1' when "010",
         '1' when "011",
         '0' when "100",
         '1' when "101",
         '1' when "110",
         '0' when "111",
         '0' when others;
end flotDeDonnees2;
```

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

2. Description par flot de données

- Le modèle d'un circuit numérique par flot de données décrit sa fonction sans nécessairement définir sa structure.
- Les valeurs des signaux et ports du circuit sont établies par des assignations concurrentes de valeurs (*concurrent signal assignment*).
- Trois types d'énoncés concurrents:
  - avec des opérateurs logiques : `and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`;
  - choisi (équivalent à un énoncé *case*): `with-select`;
  - conditionnel (équivalent à *if-else*): `when-else`.




```
architecture flotDeDonnees3 of combinatoire1 is
begin
   F <= '1' when (A = '0' or B /= C) else '0';
end flotDeDonnees3;
```

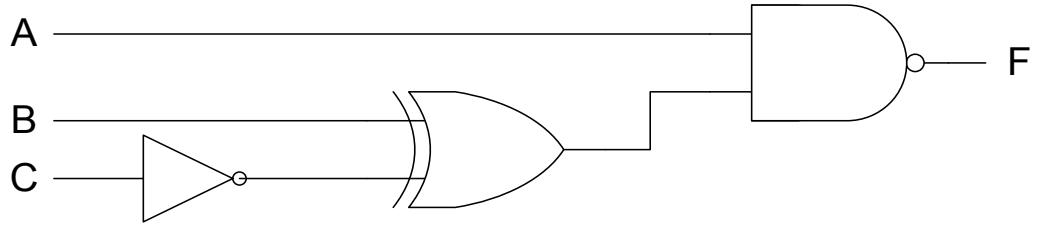
3. Description comportementale

- Une description comportementale utilise des énoncés similaires à ceux d'un langage procédural comme C et Java, incluant les structures de condition et de répétition.
- On peut abstraire le comportement du circuit à un niveau élevé, et donc définir un système complexe en peu de temps, de façon concise, paramétrable et plus facilement modifiable.
- Défi: garder en tête la nature du circuit désiré et l'inventaire de composantes matérielles disponibles pour que la description soit synthétisable.

- Les descriptions comportementales en VHDL se font à l'aide de l'énoncé `process` à l'intérieur d'une architecture.
- Un processus décrit une partie du circuit qui s'exécute de façon concurrente à d'autres processus et à des assignations concurrentes de signaux.

3. Description comportementale: exemple 1

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

entity combinatoire1 is
   port (
      A, B, C : in std_logic;
      F : out std_logic
   );
end combinatoire1;
```




```
architecture comportementale1 of combinatoire1 is
begin

   process (A, B, C)
   begin
      F <= not(A and (B xor not(C)));
   end process;

end comportementale1;
```

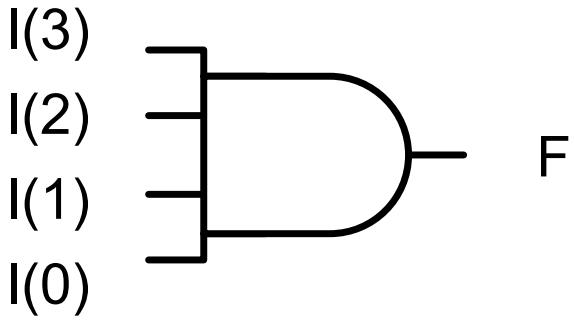
Similaire à une description par flot de données avec opérateurs logiques.

```
architecture comportementale2 of combinatoire1 is
begin

   process (A, B, C)
   begin
      if A = '0' or B /= C then
         F <= '1';
      else
         F <= '0';
      end if;
   end process;

end comportementale2;
```

Similaire à une description par flot de données avec énoncé conditionnel.

3. Description comportementale: exemple 2 – porte ET à 4 entrées

```
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;
```

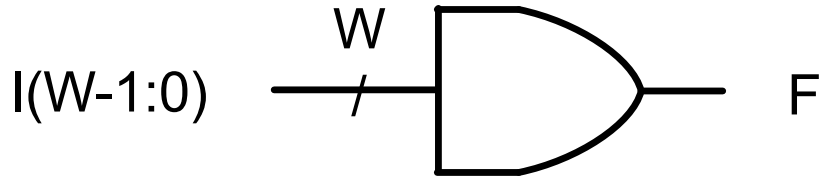
Similaire à une description par flot de données.

```
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;
```

3. Description comportementale: exemple 3 – porte ET à *W* entrées

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

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

```
architecture comportementale 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 comportementale;
```

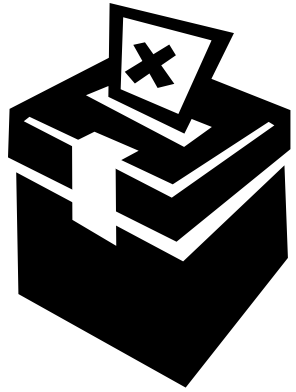


Retour: le problème du vote

Un comité composé de quatre personnes a besoin d'un mécanisme de vote secret pour les amendements sur la constitution du comité.

Un amendement est approuvé si au moins 3 personnes votent pour.

Concevoir un circuit logique qui accepte 4 entrées représentant les votes. La sortie du circuit doit indiquer si l'amendement est accepté.



Problème du vote – description par flot de données, énoncé `with-select`

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

```
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;

-- table de vérité réduite
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;
```

Problème du vote – description par flot de données, énoncés concurrents

$$F = A'BCD + AB'CD + ABC'D + ABCD' + ABCD$$
$$= BCD + ACD + ABD + ABC$$

```
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;
```

```
-- équation non réduite
architecture flotdonnees2 of vote is

signal A, B, C, D : std_logic;

begin
   (A, B, C, D) <= lesvotes; -- pour simplifier l'écriture
   approbation <=
      (not(A) and B and C and D)
      or (A and not(B) and C and D)
      or (A and B and not(C) and D)
      or (A and B and C and not(D))
      or (A and B and C and D);
end flotdonnees2;
```

```
-- équation réduite
architecture flotdonnees3 of vote is

signal A, B, C, D : std_logic;

begin
   (A, B, C, D) <= lesvotes; -- pour simplifier l'écriture
   approbation <=
      (B and C and D)
      or (A and C and D)
      or (A and B and D)
      or (A and B and C);
end flotdonnees3;
```

Problème du vote – description comportementale

- Ce code est indépendant du nombre de personnes qui votent.
- La synthèse de ce code résulte en l'utilisation de fonctions arithmétiques plutôt que logiques – nous allons voir comment plus tard.

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

entity vote is
   generic (W : positive := 4);
   port (
      lesvotes: in std_logic_vector(W - 1 downto 0);
      approbation : out std_logic
   );
end vote;
```

```
architecture comportementale2 of vote is
begin

   process(lesvotes)
   variable compte : natural range 0 to lesvotes'length;
   begin

      compte := 0;
      for k in lesvotes'range loop
         if lesvotes(k) = '1' then
            compte := compte + 1;
         end if;
      end loop;

      if compte > lesvotes'length / 2 then
         approbation <= '1';
      else
         approbation <= '0';
      end if;

   end process;

end comportementale2;
```

Vous devriez maintenant être capable de …

- Analyser le code VHDL d'un module combinatoire *simple* décrit par flot de données et donner son circuit correspondant. Donner le code VHDL (par flot de données) correspondant à un circuit combinatoire *simple*. Utiliser les assignations concurrentes, les énoncés `with-select` et `when-else`. (B2, B3)
- Analyser le code VHDL d'un module combinatoire *simple* décrit structuralement et donner son circuit correspondant. Donner le code VHDL structural correspondant à un circuit combinatoire *simple*. Comprendre et utiliser les déclarations et instanciations de composantes et l'assignation de signaux à des ports. (B2, B3)
- Donner la description par flot de données correspondante à une description structurale et vice-versa. (B2, B3)
- Comprendre et utiliser le style par description comportementale. Comprendre et utiliser l'énoncé `process` et sa liste de sensitivité. Comprendre le déroulement séquentiel de la description avec un processus. Donner le circuit correspondant à une description comportementale et vice versa. (B2, B3)
- Utiliser des énoncés `generic` pour paramétrer la description d'un module. (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.