Master 2 - SEM

Concepts avancés d'Architecture

Cours 1.2 : VHDL

Année 2022-2023

Pr. R. BOUDOUR

Doing noting is very hard to do, you never know when you're finished.

Rappel

- L'avantage du code RTL est qu'il est indépendant des technologies utilisées dans la cible à programmer,
- alors qu'au niveau « gate », il faut prendre en compte la technologie du circuit cible, car tout n'est pas permis.

Principales librairies

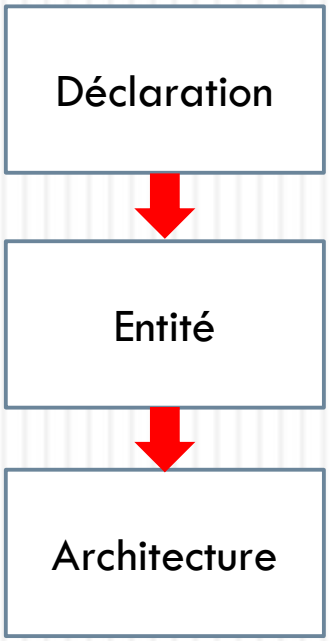
- Tout d'abord, la librairie principale (en général, IEEE).
- Ensuite, le mot clé « use », qui indique quel package de la librairie sera utilisé.
- Après cela, le nom du package. Enfin, le .all signifie que l'on souhaite utiliser tout ce qui se trouve dans ce package.
  - Lorsque le nom de la librairie est précédé de IEEE., cela signifie que c'est une librairie qui est définie dans la norme IEEE, et que l'on retrouvera donc normalement dans tout logiciel.
  - A l'inverse, il faut se méfier des librairies qui ne sont pas IEEE, car elles sont en général spécifiques à un logiciel.

Principales librairies

- Les librairies IEEE principales sont :
  - IEEE.standard
  - IEEE.std_logic_1164
  - IEEE.numeric_std
  - IEEE.std_logic_arith

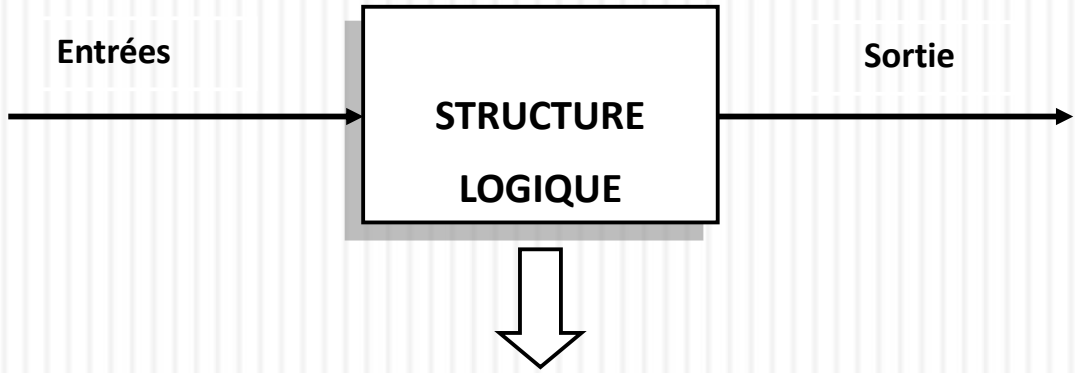
Remarque : *Il ne faut pas utiliser les librairies numeric_std et std_logic_arith en même temps.*
La librairie std_logic_arith est en fait une version améliorée de la numeric_std.

Structure d'un programme VHDL



- Tout programme **VHDL** comporte au moins trois parties :
  - la première est la **déclaration de la ou des bibliothèques** que l'on va utiliser par la suite,
  - la deuxième est une **entité** dont l'objectif est de définir quelles sont les entrées et les sorties ainsi que leurs noms,
  - la troisième est une **architecture** dont l'objectif est de décrire le fonctionnement.
- **Remarque** : *la bibliothèque est associée à l'entité qui la suit, et à cette entité **seulement**. Si un fichier source comporte plusieurs entités, il faudra déclarer autant de fois les bibliothèques qu'il y a d'entités !*

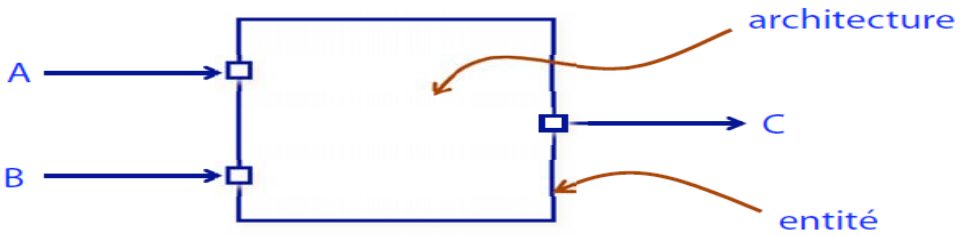
Structure d'un programme VHDL



- L'entité donne les informations concernant les signaux d'entrées et sorties de la structure ainsi que leurs noms et leurs types.
- L'architecture décrit le comportement de l'entité.

Remarque : *Il est possible de créer plusieurs architectures pour une même entité. Chacune de ces architectures décrira l'entité de façon différente.*

Structure logique



ENTITY *Nom de l'entité* **IS**

Description des entrées et des sorties de la structure en explicitant pour chacune d'entre elles le nom, la direction (IN, OUT et INOUT) et le type.

END *Nom de l'entité* **;**

ARCHITECTURE

ARCHITECTURE *Nom de l'architecture* **OF** *Nom de l'entité* **IS**

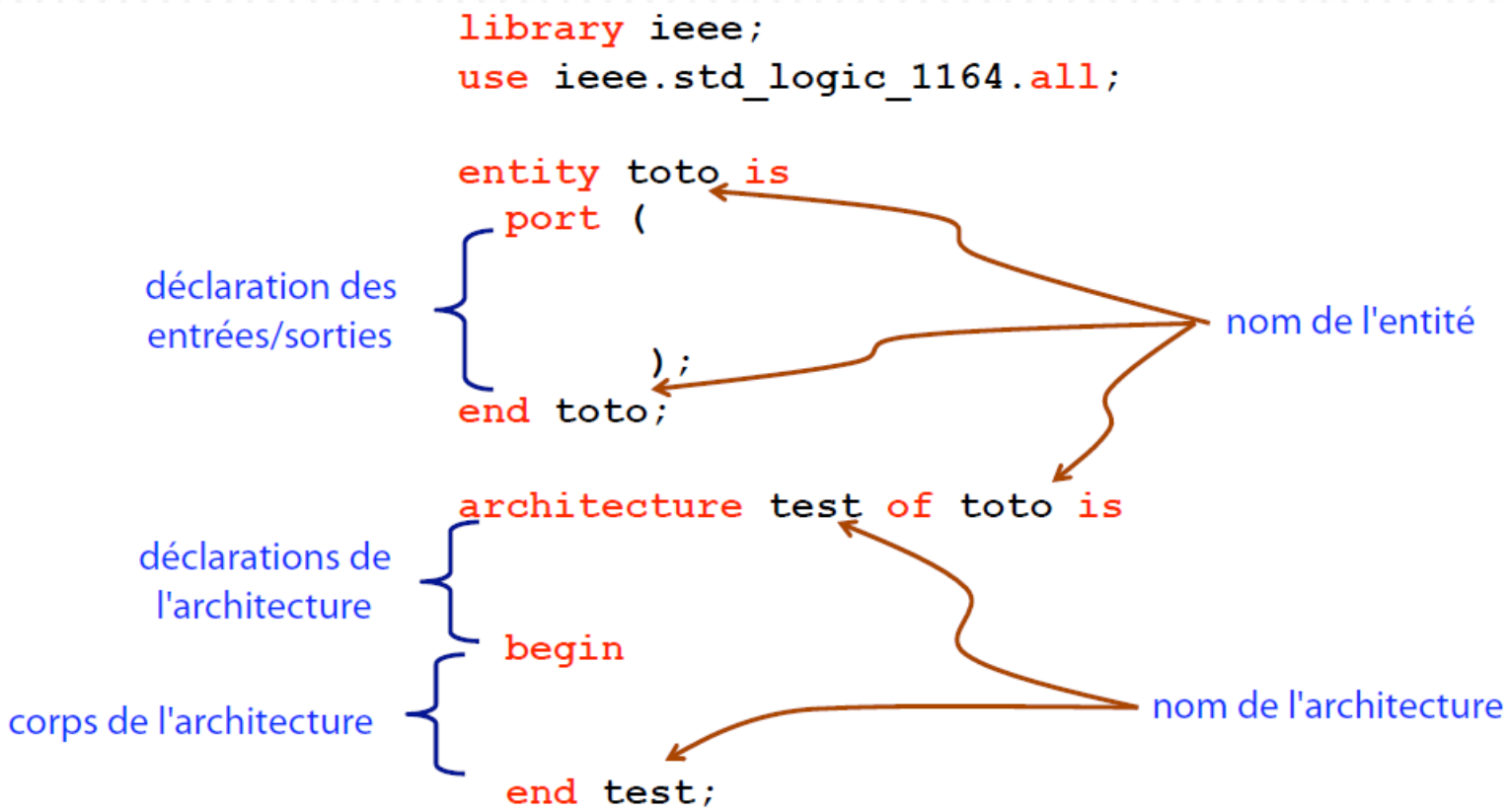
Zone de déclaration

BEGIN

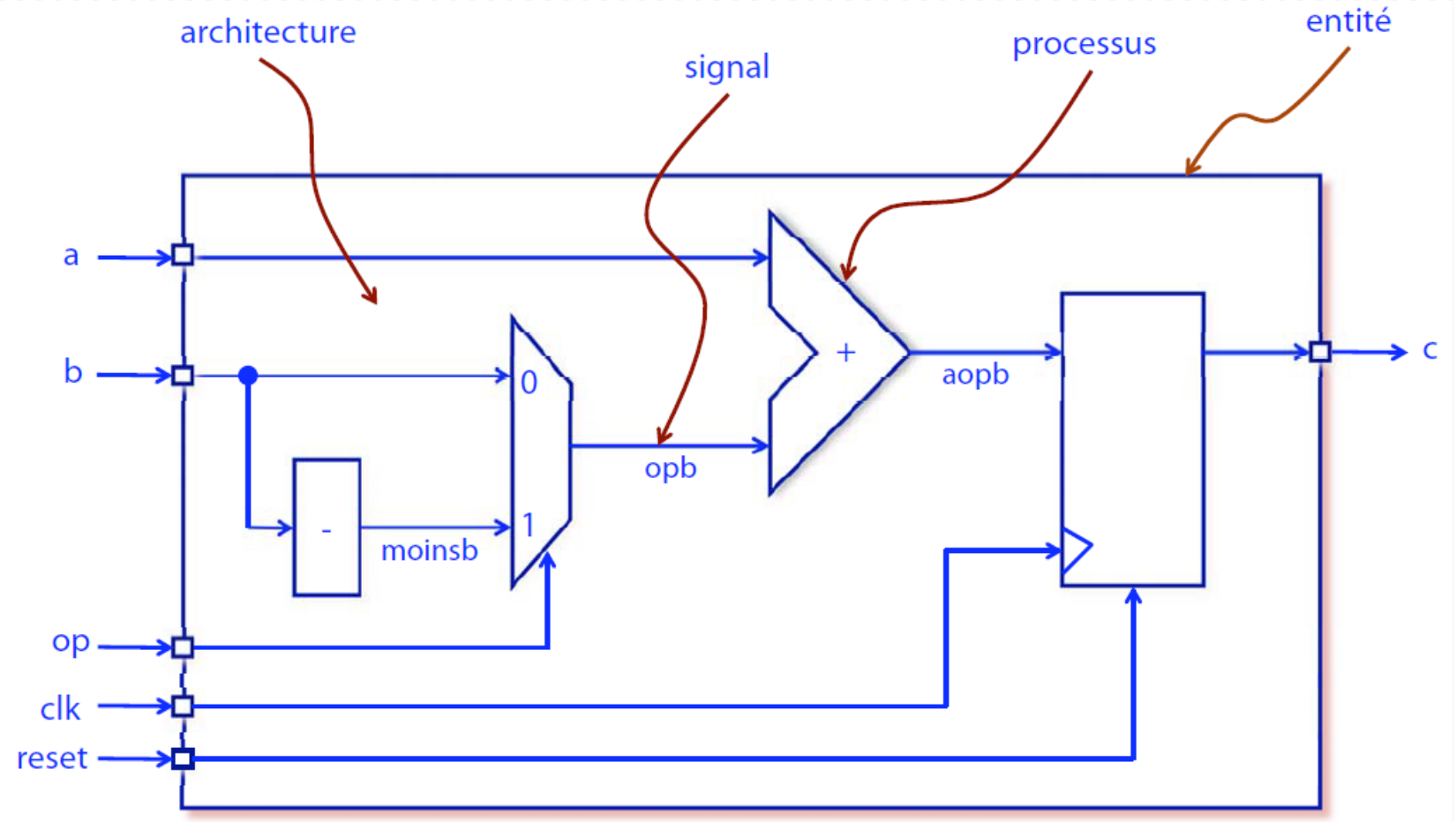
Description de la structure logique.

END *Nom de l'architecture* **;**

Structure logique



Exemple de structure



Exemple de programme

bibliothèque IEEE

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

entité

```
entity exemple is
  port (a, b            : in std_logic_vector(3 downto 0);
        op, clk, reset  : in std_logic;
        c               : out std_logic_vector(3 downto 0));
end exemple;
```

architecture

```
architecture test of exemple is
  signal moinsb, opb, aopb : std_logic_vector(3 downto 0);
begin
moinsb <= not b + "0001";

opb <= b when op='0'
       else moinsb;

aopb <= a + opb;

process (reset, clk)
  begin
  if reset='0'
    then c <= "0000";
    else if (clk'event and clk='1')
           then c <= aopb;
         end if;
  end if;
  end process;
end test;
```

processus implicites

processus explicite

Règles d'écriture

- Le langage VHDL est très strict sur la syntaxe.
  - Un signal doit être défini par son type et sa taille.
  - Tout objet doit être déclaré avant d'être utilisé.
  - Les noms de signaux, labels, etc. ne doivent contenir que des lettres, des nombres et underscore. Ils doivent impérativement commencer par une lettre, et ne pas terminer par un underscore (certains logiciels ne le supportent pas, même si ce n'est pas explicite dans la norme).
  - Le langage VHDL est case insensitive : un signal « abc » peut être appelé en tapant « Abc » ou « ABC ».
  - Pour introduire des commentaires, on fait commencer le commentaire par " -- ". Le commentaire se termine à la fin de la ligne, on ne peut pas faire de commentaires sur plusieurs lignes sans remettre à chaque fois les 2 traits.

Règles d'écriture

- Il ne faut jamais écrire des choses qui vont créer un feedback zéro délai :

  OP <= OP + Y ;

- Cette écriture est syntaxiquement correcte, mais va être refusée par la plupart des synthétiseurs, et fera planter la majeure partie des simulateurs.
- Pour réaliser cette fonction, il faut la faire dans un process :

```
Process (OP, Y)
Begin
OP <= OP + Y ;
End process ;
```

Pas de boucle zéro délai, car la nouvelle valeur de OP ne sera affectée qu'une fois le process est exécuté, cela induit donc un délai de propagation

Règles d'écriture

- Il n'est pas recommandé de jouer sur les signaux présents ou non dans la liste de sensibilité pour essayer de faire des mémorisations implicites.
  - En effet, les synthétiseurs vont de toute façon réintégrer dans la liste de sensibilité tous les signaux oubliés.
  - Ne pas le faire soi-même dès le début présente deux risques :
    - la simulation et la synthèse risquent de ne pas avoir le même résultat.
    - le synthétiseur peut mal interpréter la mémorisation implicite espérée, et synthétiser une solution matérielle complètement différente du résultat voulu.

Instructions séquentielles

- Les fonctions suivantes ne peuvent être écrites qu'à l'intérieur d'un process
- **if**

```
If CONDITION then
-- séquence d'opération
End if ;
```

La séquence ne sera exécutée que si la CONDITION est réalisée.

```
If CONDITION then
-- séquence1 d'opération
Else
-- séquence2 d'opération
End if ;
```

La séquence1 sera exécutée si la CONDITION est réalisée, sinon c'est la séquence2 qui sera exécutée.

```
If CONDITION1 then
-- séquence1 d'opération
Elsif (CONDITION2) then
-- séquence2 d'opération
Elsif (CONDITION3) then
-- séquence3 d'opération
Else
-- séquence_opt d'opération
End if ;
```

Encodage prioritaire : si la CONDITION1 n'est pas réalisée, on teste la CONDITION2,
et si elle n'est pas elle-même réalisée, on testera alors la CONDITION3.
Si les conditions 1 et 3 sont réalisées, par exemple, seule la séquence 1 sera executée, du
fait de l'encodage prioritaire

Instructions séquentielles

- **Case**

```
Case expression is
When VALUE_1 =>
-- séquence1 d'opération
When VALUE_2 to VALUE_5 =>
-- séquence2 d'opération
When VALUE_6 | VALUE_8 =>
-- séquence3 d'opération
When others =>
-- séquence3 d'opération
End case ;
```

L'évaluation des conditions doit être impérativement disjointe :
par exemple, on ne peut avoir « when VALUE_2 to VALUE_5 » et when « VALUE_5 to VALUE_6 ».

Il faut toujours mettre un « when others ». Cela permet de couvrir tous les cas de figure, et donc de ne pas avoir des états indéterminés.

Instructions séquentielles

- **For loop**

```
Signal OP,IP : std_logic_vector(3 downto 0) ;
(…)
For I in 0 to 3 loop
OP(I) <= IP(3 – I) ;
End loop ;
```

La variable I ne doit pas être déclarée, elle l'est déjà par la fonction for..loop. C'est une variable locale à la fonction loop.

Instructions concurrentes

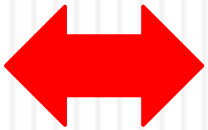
- Le VHDL décrit des structures par assemblage d'instructions **concurrentes** dont l'ordre d'écriture n'a aucune importance
- Il existe 3 principales instructions concurrentes :
  - Les **processus,** qui offrent la possibilité d'utiliser des instructions séquentielles.
  - Les **instanciations de composants** (étudiées à propos du VHDL structurel)
  - les **affectations concurrentes** de signaux, qui peuvent être conditionnelles

Assignation concurrente

- **Ecriture équivalente à un if hors process**
  - Cette écriture permet de remplacer un process simple qui ne contient qu'une boucle if avec affectation sur un seul signal.

```
Process (A, B, C, T)
Begin
If (T > 5) then
OP <= A ;
Elsif (T < 5) then
OP <= B ;
Else
OP <= C ;
End if ;
End process ;
```

Affectation dans un process



```
OP <= A when T > 5
Else B when T < 5
Else C ;
```

Affectation hors process

Assignation concurrente

- ***Ecriture équivalente à un case hors process***
  - Cette écriture permet de remplacer un process simple qui ne contient qu'une boucle if avec affectation sur un seul signal.

```
Process (A, B, C, T)
Begin
Case T is
When 0 to 4 => OP <= B ;
When 5 => OP <= C ;
When others OP <= A ;
End case ;
End process ;
```

Affectation dans un process

```
With T select
OP <= B when 0 to 4,
      C when 5,
      A when others ;
```

Affectation hors process

Variables vs signaux

- Dans un process, un signal n'aura sa nouvelle valeur affectée qu'à la fin du process.
- A l'inverse, la variable aura sa nouvelle valeur affectée immédiatement.
- C'est pourquoi, à l'intérieur d'un process, on préfèrera utiliser des variables intermédiaires plutôt que des signaux intermédiaires.
- De plus, l'utilisation de variable intermédiaire soulage l'écriture du process, car étant déclarée à l'intérieur du process, on ne les déclare pas dans la liste de sensitivité.

Variables vs signaux


```
Signal A, B, P : integer ;
Begin
Process ( A, B)
Variable VM, VN : integer ;
Begin
VM := A ;
VN := B ;
P <= VM + VN ;
End process ;
```

Cas 1 : utilisation de variables

```
Signal A, B, P : integer ;
Signal SM, SN : integer ;
Begin
Process ( A, B, SM, SN)
Begin
SM <= A ;
SN <= B ;
P <= SM + SN ;
End process ;
```

Cas 2 : utilisations de signaux

Dans le cas 1, P est affecté en une itération du process.
Dans le cas 2, P est affecté en deux itérations : une première itération va affecter SM et SN en fin de process, ce qui va re-déclencher l'exécution pour que P reçoive la somme de SM et SN.

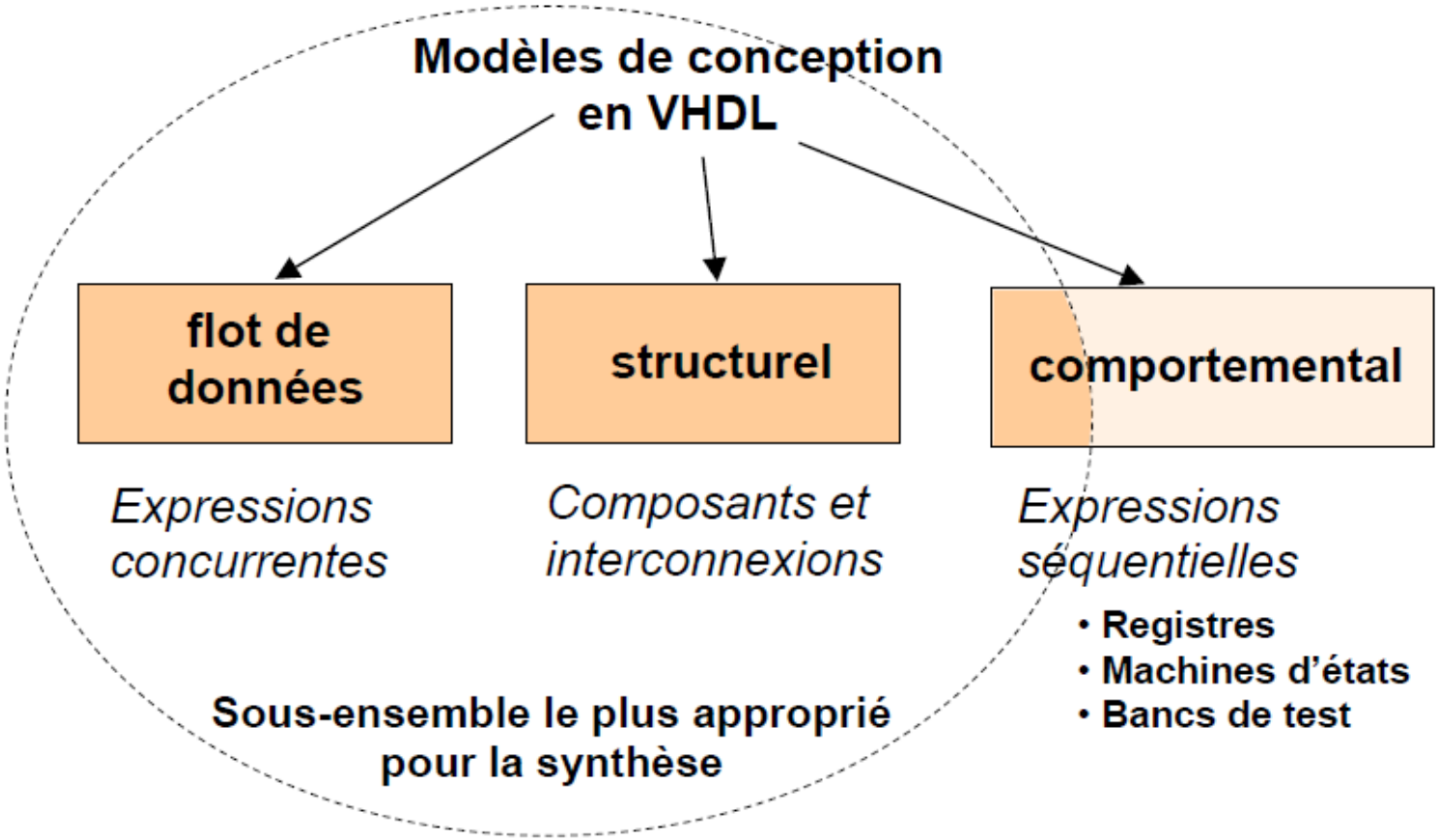
Styles d'architecture

- Trois grands formalismes coexistent pour décrire les architectures :
  - **Flot de données** : On écrit explicitement **les fonctions booléennes (équations logiques)** que l'on veut voir implémentées (à réserver aux plus petits circuits pour des raisons de lisibilité) ; c'est lui qu'on a utilisé pour implémenter le demi-additionneur ;
  - **Structurel** : On décrit le circuit comme une série de boîtes noires interconnectées au moyen de signaux (utilisé pour des circuits moyens ou grands) ; on procèdera de cette manière pour synthétiser un additionneur complet à l'aide de deux demi-additionneurs ; c'est un assemblage de sous blocs similaire à la liste d'interconnexion d'un schéma logique (netlist)
  - **Comportemental** : Algorithmique de manière très semblable à un langage de programmation informatique, on précise le fonctionnement voulu à l'aide d'une suite d'instructions de contrôles plus ou moins évoluées (conditions, boucles, etc.), dans un processus.

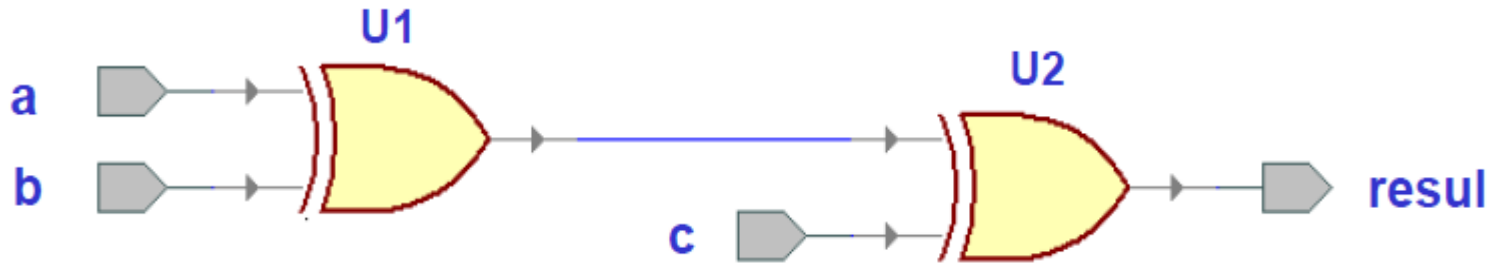
Styles d'architecture

- **En d'autres termes :**
  - Le style "structurel" : Interconnexion de composants, chacun d'eux étant une instance de couple entité/architecture.
  - Le style « flot de données ou dataflow" (s'apparente au niveau d'abstraction RTL) : Ensemble d'instructions sur des signaux qui décrivent les connexions entre portes logiques et les chargements de registres.
  - Le style "comportemental" : Ensemble de processus qui expriment le comportement du système.

Styles d'architecture ou Modèles

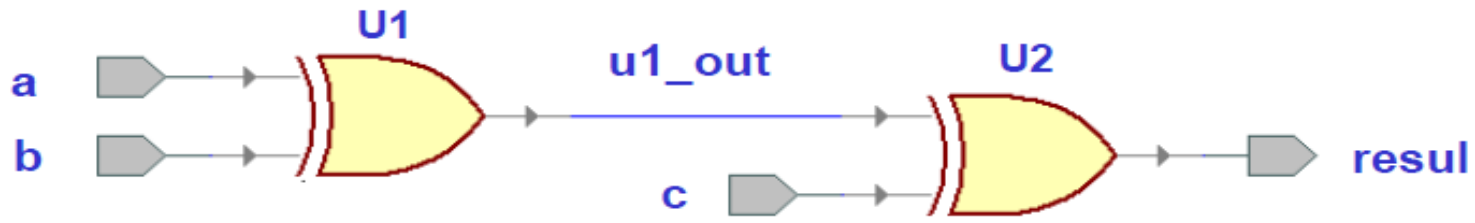


Exemple




```
ENTITY xor3 IS
  PORT(
    a      : IN STD_LOGIC;
    b      : IN STD_LOGIC;
    c      : IN STD_LOGIC;
    resul  : OUT STD_LOGIC
    );
END xor3;
```

Modèle flot de données



Nom d'architecture

Nom d'entité

```
ARCHITECTURE xor3_flotdon OF xor3 IS
  SIGNAL u1_out: STD_LOGIC;
BEGIN
  u1_out <= a XOR b;
  resul  <= u1_out XOR c;
END xor3_flotdon;
```

Déclaration du signal interne

Flot de données


```
-- Déclaration des bibliothèques utilisées
library IEEE;
use IEEE.std_logic_1164.all;

-- Déclaration de l'entité du demi-additionneur (half-adder).
entity HA is
    port(A, B: in  std_logic;
         S, R: out std_logic);
end HA;

-- Déclaration de l'architecture en flot de données.
architecture arch_HA_flow of HA is
begin
    S <= A xor B;
    R <= A and B;
end arch_HA_flow;
```

Modèle flot de données

- On décrit simplement les équations booléennes que l'on veut implémenter.
- Exemple : un multiplexeur 4x1 en portes logiques

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

entity MUX is port(
    E0, E1, E2, E3, SEL0, SEL1: in std_logic;
    S: out std_logic);
end;

architecture FLOT_MUX of MUX is
begin
    S <= ((not SEL0) and (not SEL1) and E0) or
         (SEL0 and (not SEL1) and E1) or
         ((not SEL0) and SEL1 and E2) or
         (SEL0 and SEL1 and E3);
end FLOT_MUX;
```

Modèle flot de données

- Le modèle « Flot de données » décrit les relations internes entre les données dans le module.
- La description de l'architecture basée sur ce modèle utilise les expressions concurrentes pour réaliser la logique. Ces expressions sont évaluées en même temps (en parallèle), donc leur ordre n'est pas important !
- Le modèle « Flot de données » est le plus utile, si la logique peut être représentée par des fonctions booléennes.

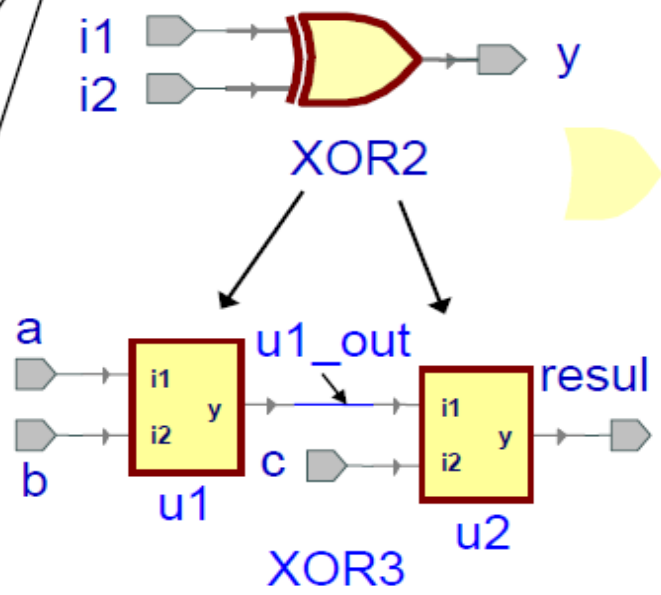
Modèle structurel


```
ARCHITECTURE xor3_struct OF xor3 IS
  SIGNAL u1_out: STD_LOGIC;
  COMPONENT xor2
    PORT(
         i1 : IN STD_LOGIC;
         i2 : IN STD_LOGIC;
          y : OUT STD_LOGIC
        );
  END COMPONENT;
BEGIN
  u1: xor2 PORT MAP (i1 => a,
                     i2 => b,
                      y => u1_out);

  u2: xor2 PORT MAP (i1 => u1_out,
                     i2 => C,
                      Y => resul);
END xor3_struct;
```

Déclaration du signal interne

Déclaration du composant (qui est défini ailleurs)

Instanciations du composant



Déclaration de composant

- Association des connexions par leur noms (recommandée)

```
COMPONENT xor2 IS
  PORT(
       i1 : IN STD_LOGIC;
       i2 : IN STD_LOGIC;
       y  : OUT STD_LOGIC
      );
END COMPONENT;

u1: xor2 PORT MAP (i1 => a,
                   i2 => b,
                   y  => u1_out);
```

Déclaration du composant
(dans la partie « déclarations » de l'architecture)

Instanciation du composant
(dans le corps de l'architecture)

Nom de l'instanciation ***Nom du composant***

Déclaration de composant


```
entity AND_3 is
 port(
 e1 : in bit;
 e2 : in bit;
 e3 : in bit;
 s : out bit
 );
end entity;

architecture arc of AND_3 is

signal z : bit;
component and2
        port (
        a : bit;
        b : bit;
        s : bit);
end component;

begin
 inst1 : and2 port map (a=>e1, b=>e2 , s=>z);
 inst2 : and2 port map (z, e3, s);
end arc
```

Dans cet exemple , 2 instances de composant "and2" sont appelées pour créer une porte ET à 3 entrées.

L'association des ports du composants aux signaux de l'instance se fait à l'aide de la clause `port map`.

La syntaxe des associations est soit

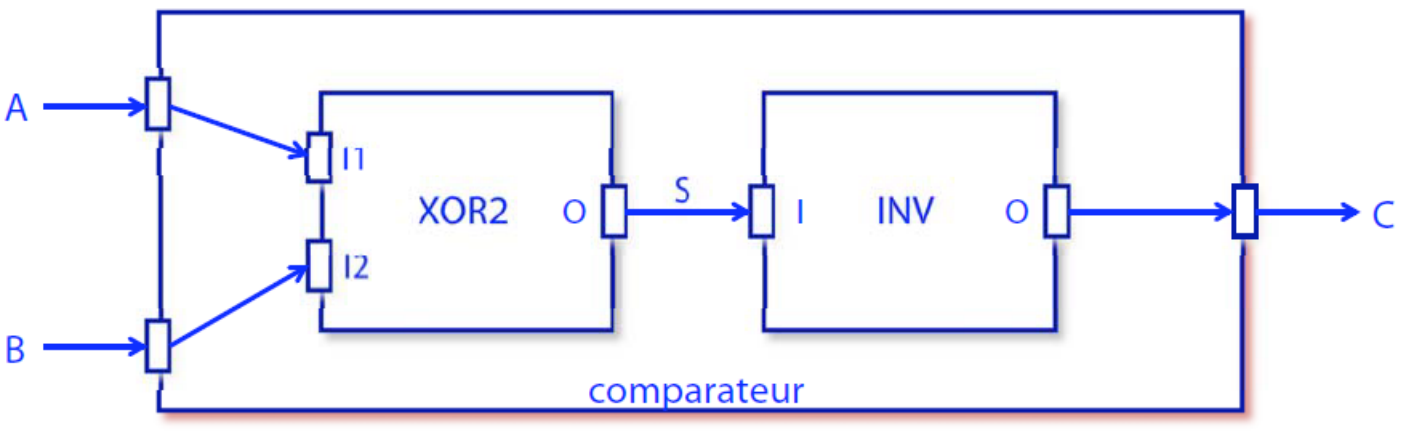
1. par nom où chaque broche du composant est associée à un signal : cas de inst_1
2. positionnelle où l'ordre des signaux correspond à l'ordre des broches : cas de inst_2

Exemple 2 : Comparateur

- C'est la vue externe du système, avec une déclaration de ses ports 'canaux de communication entrée / sortie)


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

Exemple 2 : Modèle structurel



Exemple 2 : Modèle structurel


```
architecture structurelle of comparateur is

  component XOR2
    port (O : out std_logic; I1, I2 : in std_logic);
  end component;
  component INV
    port (O : out std_logic; I : in std_logic);
  end component;

  signal S : std_logic;

  begin
    C1 : XOR2 port map (O => S, I1 => A, I2 => B);
    C2 : INV port map (C, S);
  end structurelle;
```

Exemple 3

- On assemble ici une série de boîtes noires déjà écrites pour former un circuit plus complexe.
- Ici, pour l'exemple, on se limitera à synthétiser un additionneur complet sur la base de deux demi-additionneurs et d'une porte OR.
- On aura besoin de deux instances du composant demi-additionneur et d'une seule du composant OR.
- On note qu'il faut répéter les bibliothèques avant chaque couple entité-architecture.

Exemple 3


```vhdl
-- Entité demi-additionneur
library ieee;
use ieee.std_logic_1164.all;
entity HA is port(
    A, B: in  std_logic;
    R, S: out std_logic);
end HA;

architecture FLOW of HA is
begin
    S <= A xor B;
    R <= A and B;
end FLOW;

-- Entité OR
library ieee;
use ieee.std_logic_1164.all;
entity P_OR is port(
    C, D: in  std_logic;
    E:    out std_logic);
end P_OR;

architecture FLOW of P_OR is
begin
    E <= C or D;
end FLOW;
```

```vhdl
-- Entité additionneur complet
library ieee;
use ieee.std_logic_1164.all;
entity FA is port(
    A, B, Ri: in  std_logic;
    R, S:     out std_logic);
end;

architecture FA_ARCH of FA is
    component HA
        port(A, B: in  std_logic;
             R, S: out std_logic);
    end component HA;

    component P_OR
        port(C, D: in  std_logic;
             E:    out std_logic);
    end component P_OR;

    signal S1, S2, S3: std_logic;
begin
    i1: HA  port map(A,  B,  S1, S2);
    i2: HA  port map(S2, Ri, S3, S);
    i3: P_OR port map(S3, S1, R);
end FA_ARCH;
```

Modèle structurel

- Le modèle structurel est le plus simple à comprendre. Il est le plus proche à la saisie du schéma : il utilise les blocs simples pour composer des fonctions logiques.
- Les composants peuvent être interconnectés d'une manière hiérarchique.
- Dans le modèle structurel nous pouvons connecter les ports simples ou les composants complexes et abstraits.
- Le modèle structurel de l'architecture est utile pour la réalisation d'une conception où les sous-blocs sont connectés d'une manière naturelle.

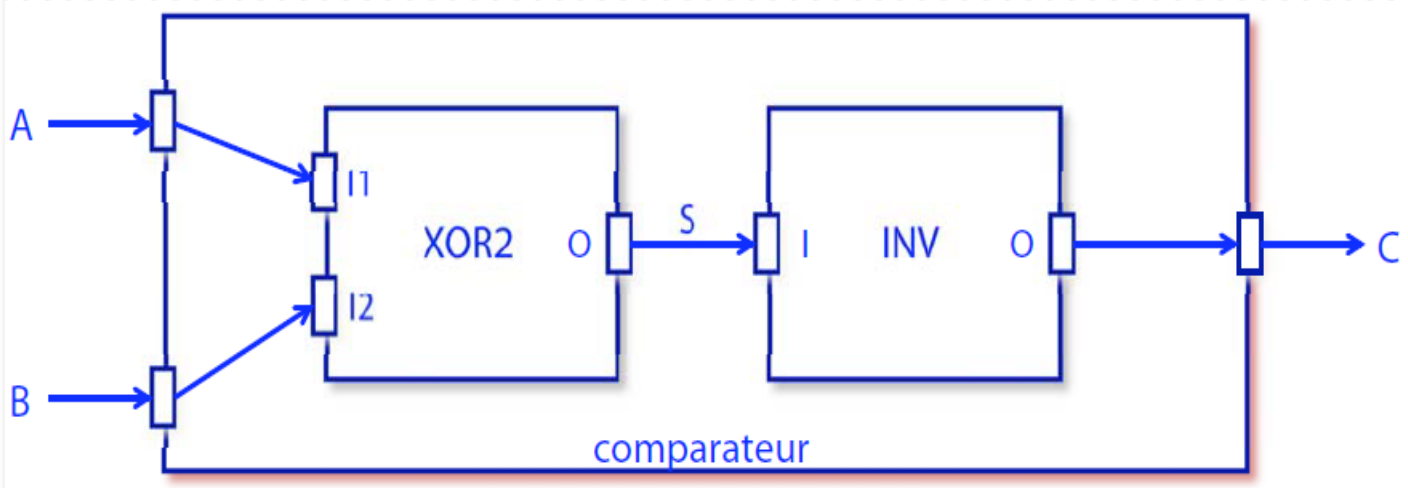
Modèle comportemental


```
ARCHITECTURE xor3_comp OF xor3 IS
BEGIN
   xor3_proc: PROCESS (a, b, c)
   BEGIN
      IF ((a XOR b XOR c) = '1') THEN
         resul <= '1';
      ELSE
         resul <= '0';
      END IF;
   END PROCESS xor3_proc;
END xor3_comp;
```

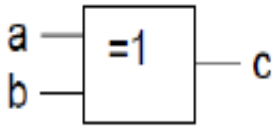
Comportemental

- On décrit cette architecture comme une suite d'instructions, comme en programmation informatique traditionnelle. On définit des process ainsi que leur liste de sensibilité : dès qu'un symbole change de valeur dans cette liste, le code est réexécuté. C'est utile notamment en logique séquentielle, pour lancer le code à chaque coup de l'horloge. Les instructions sont exécutées de manière séquentielle ; cependant, les signaux ne changent de valeur qu'à la fin du process.
- On peut utiliser diverses constructions bien connues : le if-then-else, le case-is when, etc.
- Rappel : Les signaux sont définis dans l'en-tête de l'architecture ; les variables ne sont valables que dans un process. En dehors d'un process, tout se produit de manière concurrente : deux process peuvent être exécutés simultanément. Par contre, dans un process, toutes les instructions sont exécutées de manière séquentielle et les affectations sont réalisées à la fin.

Exemple 1



a	b	c
0	0	0
0	1	1
1	0	1
1	1	0



Exemple 1 : Architecture comportementale


```
architecture comportementale of comparateur is
  begin
    process (A, B)
      begin
      if (A = B) then
        C <= '1';
      else
        C <= '0';
      end if;
    end process;
  end comportementale;
```

Exemple 2 : un multiplexeur 4x1 avec when

- On peut cependant améliorer la syntaxe à l'aide de when, qui indique la valeur que doit prendre le signal ou la sortie dans tel ou tel cas. Cette écriture est déjà plus lisible, elle correspond beaucoup plus directement à la définition en français du multiplexeur.

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

entity MUX is port(
    E0, E1, E2, E3, SEL0, SEL1: in std_logic;
    S: out std_logic);
end;

architecture FLOT_MUX of MUX is
begin
    S <= E0 when (SEL0='0' and SEL1='0') else
         E1 when (SEL0='1' and SEL1='0') else
         E2 when (SEL0='0' and SEL1='1') else
         E3;
end FLOT_MUX;
```

Exemple 2' : Multiplexeur 4x1 avec when

- On peut alors employer un don't care pour gérer tous les cas restants, ce qui permet de n'avoir une sortie contrôlée que si les entrées de sélection forment une adresse correcte.

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

entity MUX is port(
    E0, E1, E2, E3, SEL0, SEL1: in std_logic;
    S: out std_logic);
end;

architecture FLOT_MUX of MUX is
begin
    S <= E0 when (SEL0='0' and SEL1='0') else
         E1 when (SEL0='1' and SEL1='0') else
         E2 when (SEL0='0' and SEL1='1') else
         E3 when (SEL0='1' and SEL1='1') else
         '-';
end FLOT_MUX;
```

Exemple 3 : Multiplexeur avec with-select

- On peut encore l'améliorer en restant très proche des équations avec un with-select et une concaténation des deux bits d'entrée de sélection.
- exemple : un multiplexeur 4-1 avec un SELECT

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

entity MUX is port(
    E0, E1, E2, E3, SEL0, SEL1: in std_logic;
    S: out std_logic);
end;

architecture FLOT_MUX of MUX is
begin
    with SEL1 & SEL0 select
        S <= E0 when "00",
             E1 when "01",
             E2 when "10",
             E3 when others;
end FLOT_MUX;
```

Exemple 4 : Un multiplexeur comportemental avec IF

- On peut implémenter un multiplexeur à l'aide d'une série de if, une manière très commode de procéder pour tout qui vient de la programmation informatique.

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

entity MUX is port(
     E0, E1, E2, E3: in std_logic;
     SEL: in std_logic_vector(1 downto 0);
     S: out std_logic);
end;

architecture CMP of MUX is
begin
     process (E0, E1, E2, E3, SEL)
     begin
          if SEL="00" then
               S <= E0;
          elsif SEL="01" then
               S <= E1;
          elsif SEL="10" then
               S <= E2;
          elsif SEL="11" then
               S <= E3;
          else
               S <= '-';
          end if;
     end process;
```

Exemple 5 : Un multiplexeur comportemental avec case

- On peut y préférer un case pour des raisons de lisibilité, cela explicite bien qu'on travaille toujours avec le même vecteur de bits.

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

entity MUX is port(
    E0, E1, E2, E3: in std_logic;
    SEL: in std_logic_vector(1 downto 0);
    S: out std_logic);
end;

architecture CMP of MUX is
begin
    process (E0, E1, E2, E3, SEL)
    begin
        case SEL is
            when "00" => S <= E0;
            when "01" => S <= E1;
            when "10" => S <= E2;
            when "11" => S <= E3;
        end case;
    end process;
end FLOT_MUX;
```

Fonctions logiques combinatoires

- ***Fonction logique combinatoire - définition***
  - *La valeur à la sortie d'une fonction logique combinatoire ne dépend que des valeurs de signaux des entrées (leur combinaison) et (contrairement à la logique séquentielle) ne dépend pas de l'état interne de la fonction*
  - *Exemple :* $Y = f(A, B, C) = A + B + C$

Y dépend que de A, B, et C

- ***Fonctions combinatoires de base***
  - *Fonctions logiques simples*
  - *Générateurs de parité*
  - *Multiplexeurs, démultiplexeurs*
  - *Codeurs, décodeurs*
  - *Fonctions arithmétiques simples (addition, soustraction, etc.)*
  - *Comparateurs*
  - *Fonctions d'entrées/sorties avec la logique trois états*

Implantation des fonctions combinatoires

Instructions concurrentes

- Affectation inconditionnelle d'un signal

  signal **<=** *expression (avec les signaux et/ou avec les constantes)*;

- Affectation conditionnelle d'un signal

  signal **<=** *expression1* **WHEN** *condition* **ELSE** *expression2*;

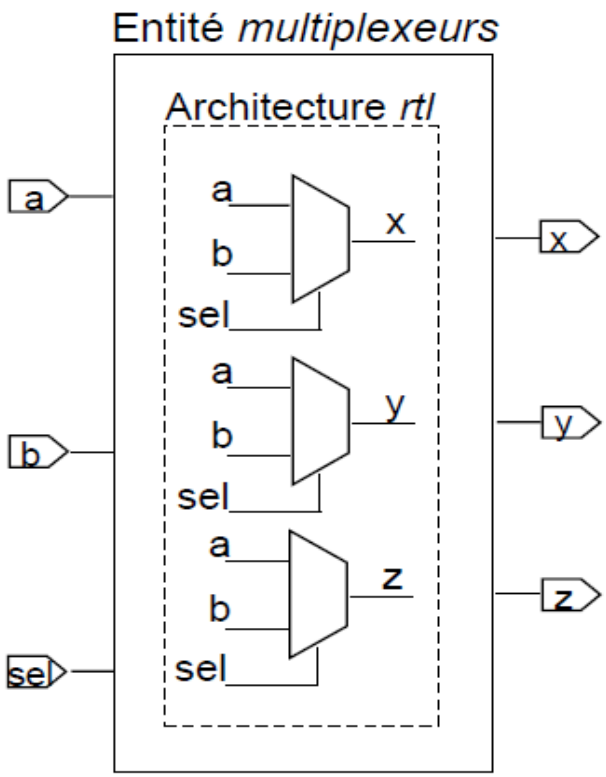
- Affectation sélective d'un signal

  WITH *selecteur* **SELECT**

  signal **<=** *expression1* **WHEN** *valeur_selecteur*, ...;

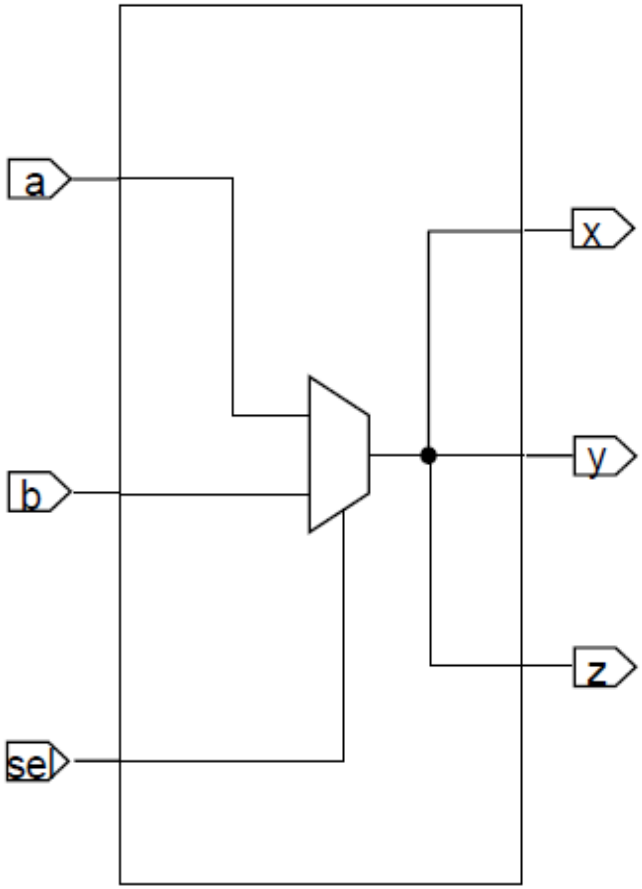
Exemples : instructions concurrentes


```
ENTITY multiplexeurs IS
PORT (a, b, sel : IN bit;
      x, y, z   : OUT bit);
END multiplexeurs;

ARCHITECTURE rtl OF multiplexeurs IS
BEGIN
-- affectation inconditionnelle
  x <= (a AND NOT sel) OR (b AND sel);

-- affectation conditionnelle
  y <= a WHEN sel='0' ELSE
       b;
-- affectation sélective
  WITH sel SELECT
    z <= a  WHEN '0',
         b  WHEN '1',
        '0' WHEN OTHERS;
END rtl;
```



instructions concurrentes (suite)

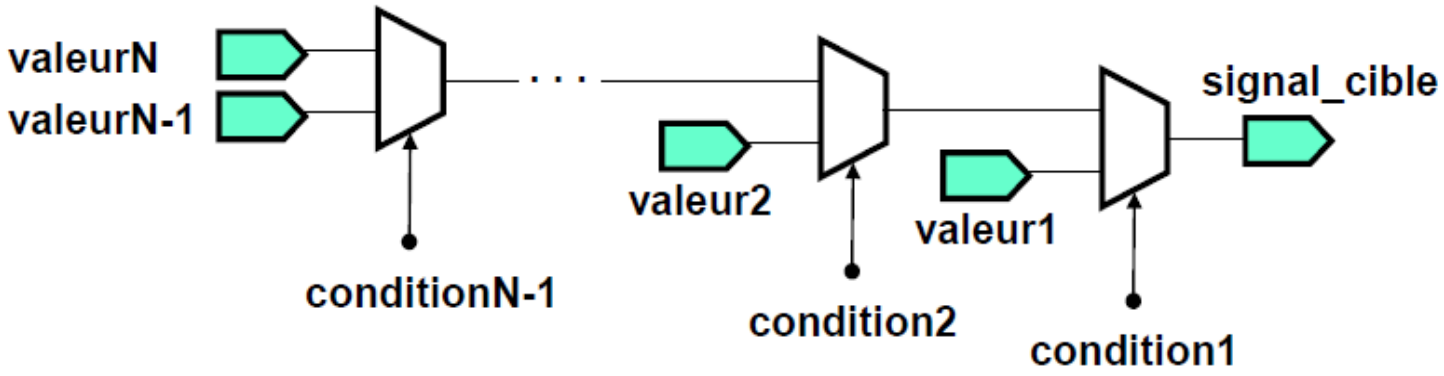
- *Puisque les trois descriptions décrivent la même structure logique, l'architecture* `rtl` *sera implanté dans le matériel de la façon suivante - deux structures redondantes seront supprimées*



instructions concurrentes (suite)

- ***Assignation conditionnelle***

```
signal_cible <= valeur1 WHEN condition1 ELSE
                valeur2 WHEN condition2 ELSE
                . . .
                valeurN-1 WHEN conditionN-1 ELSE
                valeurN;
```

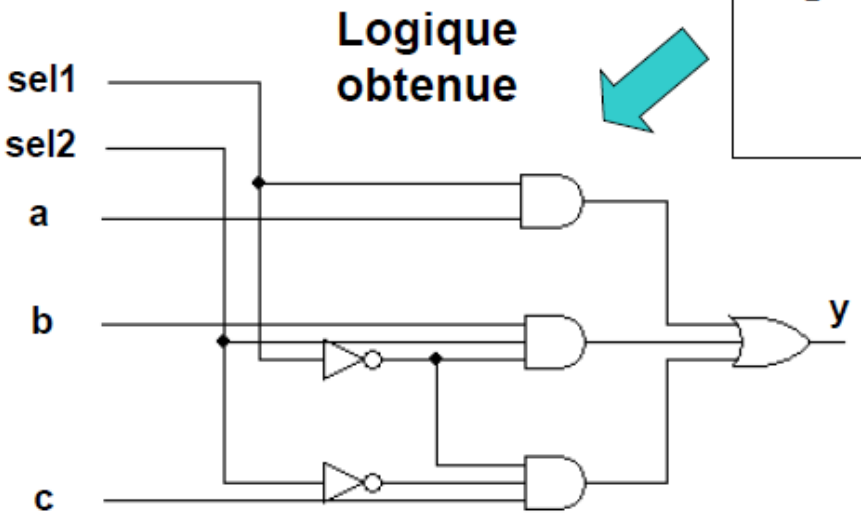


Conclusion : Codage par priorité – prudence !

instructions concurrentes (suite)

Codage par priorité d'une affectation conditionnelle :

```
-- affectation conditionnelle
  y <= a WHEN sel1 = '1' ELSE
       b WHEN sel2 = '1' ELSE
       c;
```



La sélection du signal `a` *avec le signal* `sel1` *est prioritaire devant* `b` *et* `c` *!*

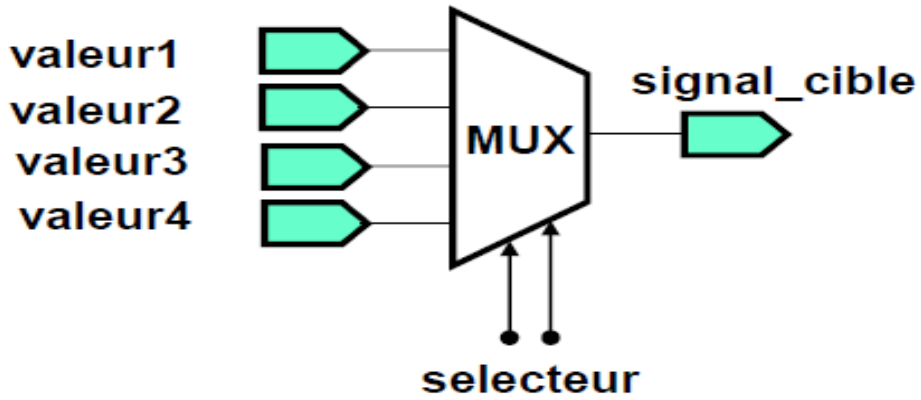
```
q = (sel1 AND a)
    OR ((NOT sel1) AND sel2 AND b)
    OR ((NOT sel1) AND (NOT sel2) AND c)
```

instructions concurrentes (suite)

- ***Affectation sélective***

```
WITH selecteur SELECT
   signal_cible <=  valeur1 WHEN valeur_selecteur,
                    valeur2 WHEN valeur_selecteur,
                    . . .
                    valeurN WHEN OTHERS;
```

Exemple d'application : multiplexeur



Implantation des fonctions simples

- *Implantation basée sur l'utilisation d'opérateurs de base*
  AND, OR, NOT, XOR, NAND, NOR
- *Quelques exemples d'implantation de la fonction ET logique en VHDL*
  - *Ver 1 :*

```
y <= a AND b;
```

  - *Ver 2 :*

```
y <= '1' WHEN (a='1' AND b='1') ELSE '0';
```

  - *Ver 3 :*

```
y <= '0' WHEN (a='0' OR b='0') ELSE '1';
```

  - *Ver 4 :*

```
s <= a & b;
WITH s SELECT
   y <= '1' WHEN "11"
        '0' WHEN OTHERS;
```

Concaténation de a et b

N'oubliez pas les branches alternatives !!!

Implantation des fonctions évoluées

- *Utilisation de l'affectation conditionnelle si certains signaux sont prioritaires* (**sélecteurs avec priorité**)
  - *Ex. :*

```
y <= a WHEN sel(0)='1' ELSE
     b WHEN sel(1)='1' ELSE
     c;
```

- *Utilisation de l'affectation sélective pour obtenir une structure régulière sans priorité* (**multiplexeurs**)
  - *Ex. :*

```
WITH sel SELECT
   y <= a WHEN "00"
        b WHEN "01";
        c WHEN OTHERS;
```

N'oubliez pas les branches alternatives !!!

Générateur de Parité

- *Générateur de parité – génèrent le bit de parité qui permet de détecter les erreurs survenues pendant la transmission ou la sauvegarde (dans la mémoire) de données*
- *Principe – le bit de parité est égal à un si le nombre de uns dans la donnée est impaire – pour une donnée de N bits on peut le réaliser avec une fonction logique OU EXCLUSIF à N entrées*

A noter : le nombre paire d'erreurs n'est pas détecté …

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY parity IS
PORT (a        : IN std_logic_vector(7 DOWNTO 0);
      par_out : OUT std_logic);
END parity;

ARCHITECTURE rtl OF parity IS
BEGIN
-- assignation inconditionnelle
    par_out <= a(7) XOR a(6) XOR a(5) XOR a(4)
               XOR a(3) XOR a(2) XOR a(1) XOR a(0);
END rtl;
```

Générateur de Parité paramétrable

- ***Fonction générique*** **– *fonction universelle, adaptable aux besoins – la largeur du générateur peut être définie par le paramètre N***

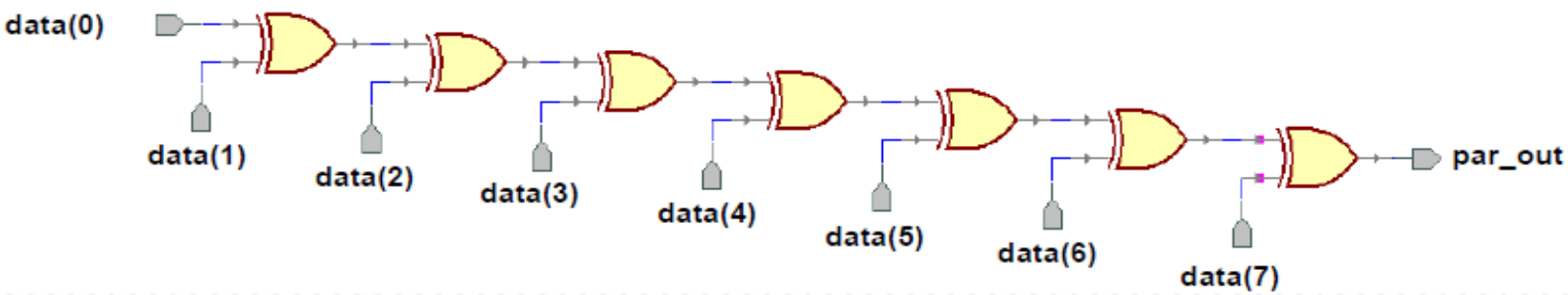
- ***Exemple de la réalisation*** **– *structure FOR … GENERATE :***

étiquette : **FOR** *variable_de_boucle* **IN** *intervalle* **GENERATE**
[déclarations]
BEGIN
{assignation(s) concurrente(s)}
END GENERATE *étiquette*;

Optionnelles

Étiquette obligatoire

Structure à réaliser pour N = 8 :



Générateur de Parité paramétrable


```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY parity IS
GENERIC (WIDTH : INTEGER := 8);
PORT (data    : IN std_logic_vector(WIDTH-1 DOWNTO 0);
      par_out : OUT std_logic);
END parite;

ARCHITECTURE rtl OF parite IS
SIGNAL par_int : std_logic_vector(WIDTH-1 DOWNTO 0);

BEGIN
  par_int(0) <= data(0);
  par_out <= par_int(WIDTH-1);
Parity_gen:
  FOR i IN 1 TO (WIDTH-1) GENERATE
    par_int(i) <= data(i) XOR par_int(i-1);
  END GENERATE Parity_gen;
END rtl;
```

On peut modifier la largeur du générateur en modifiant le paramètre WIDTH

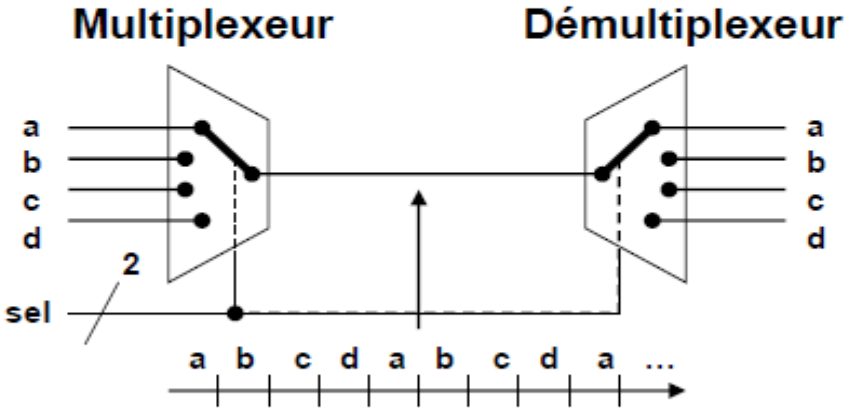
Nous pouvons utiliser le signal par_int(WIDTH-1) avant qu'il soit affecté (structure concurrente)

```
par_out = data(0) XOR data(1) XOR data(2) XOR data(3)
          XOR data(4) XOR data(5) XOR data(6) XOR data(7)
```

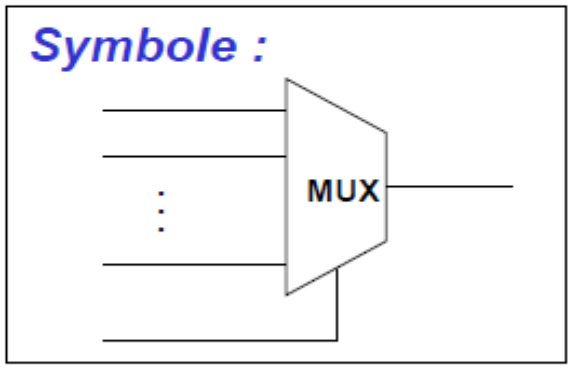
Multiplexeurs et Démultiplexeurs

- *Multiplexeurs – permettent de sélectionner un parmi plusieurs signaux d'entrée*
- *Démultiplexeurs – permettent de rediriger la valeur présente à l'entrée vers une parmi plusieurs sorties*

Schéma du principe :



Symbole :



- *Exemple d'utilisation – transfert de plusieurs données (4 dans notre cas) par une liaison série – une tranche de temps est attribuée périodiquement à chaque canal*

Implantation d'un multiplexeur 4x1


```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY multiplexer IS
PORT (a, b, c, d : IN std_logic;
      sel        : IN std_logic_vector(1 DOWNTO 0);
      y          : OUT std_logic);
END multiplexer;

ARCHITECTURE rtl OF multiplexer IS
BEGIN
-- assignation sélective
  WITH sel SELECT
    y <= a  WHEN "00",
         b  WHEN "01",
         c  WHEN "10",
         d  WHEN OTHERS;
END rtl;
```

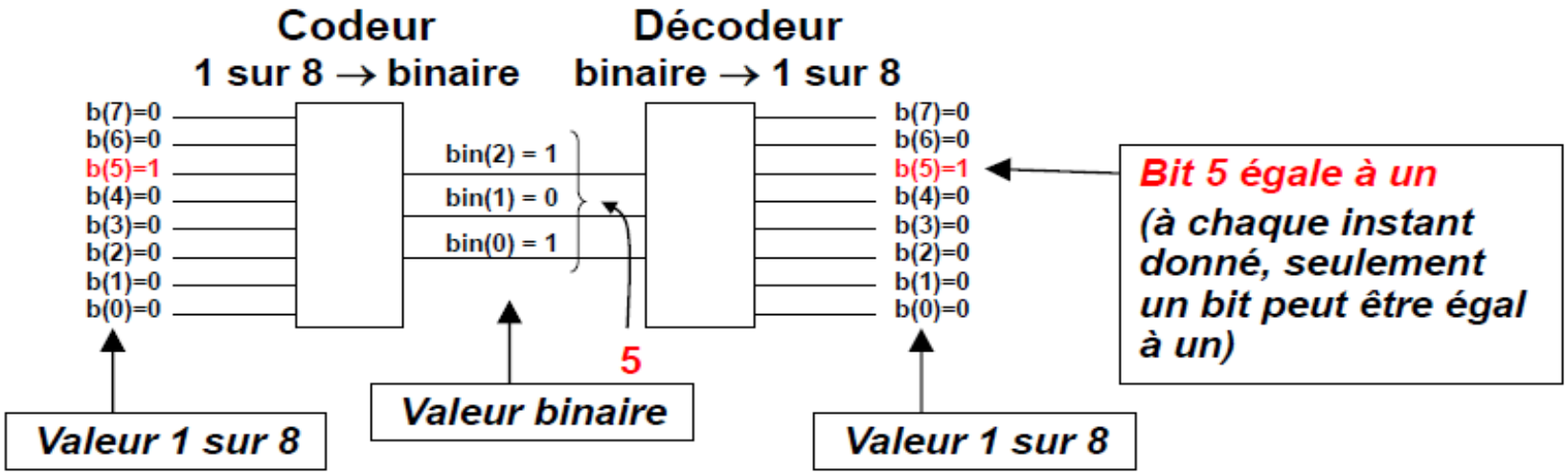
Sélection sans priorité !

```
y = (a AND (NOT sel(1)) AND (NOT sel(0)))
     OR (b AND (NOT sel(1)) AND sel(0))
     OR (c AND sel(1) AND (NOT sel(0)))
     OR (d AND sel(1) AND sel(0))
```

Codeurs et Décodeurs

- ***Codeurs*** *– permettent de coder une donnée (binaire) d'une manière plus efficace ou d'une manière plus avantageuse*
- ***Décodeurs*** *– réalisent une opération inverse par rapport au codage et permettent de représenter une information d'une manière mieux compréhensible ou plus facile à utiliser*

Exemple : Codeur 1 sur N → binaire



Codeurs et Décodeurs

- ***Codeur avec priorité*** **– donne le numéro (en binaire) du canal égal à un ; si plusieurs canaux sont égaux à un, c'est le numéro du canal prioritaire, qui est donné à la sortie**

Exemple – Codeur « 1 sur 3 → binaire » avec priorité

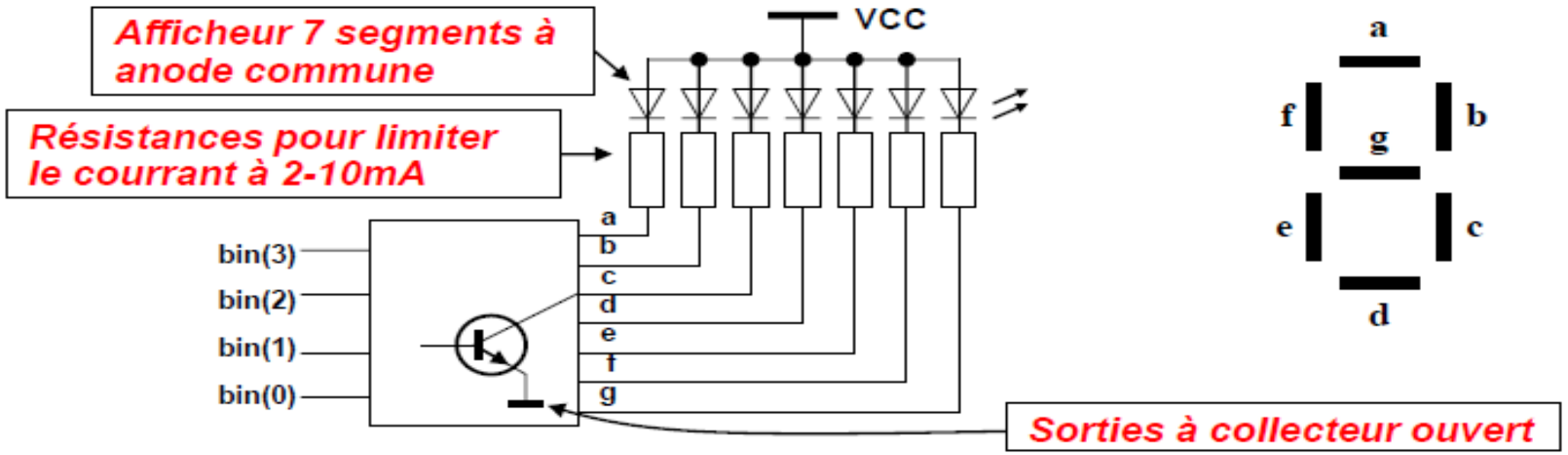
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY priority_encod IS
PORT (a, b, c : IN std_logic;
      bin_out : OUT std_logic_vector(1 DOWNTO 0));
END priority_encod;

ARCHITECTURE rtl OF priority_encod IS
BEGIN
-- assignation conditionnelle
    bin_out <= "01" WHEN a = '1' ELSE
               "10" WHEN b = '1' ELSE
               "11" WHEN c = '1' ELSE
               "00";
END rtl;
```

a est prioritaire devant b, qui est prioritaire devant c

Codeurs et Décodeurs

- ***Décodeur « code binaire → code afficheur sept segments »***
  – permet d'afficher une valeur binaire de 4 bits en hexadécimal sur un afficheur (avec les diodes LED) de sept segments

Schéma d'interconnexion :



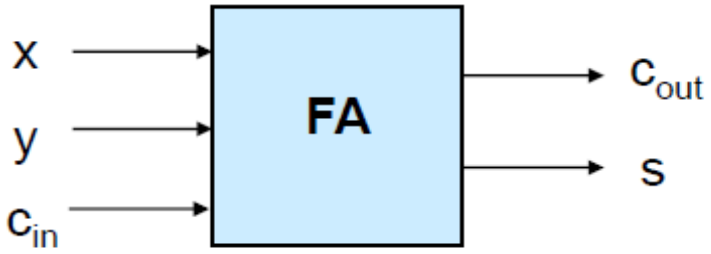
Codage de segments :

Affichage de caractères hexadécimaux :

Blocs arithmétiques

- *Opérations arithmétiques peuvent être réalisées en VHDL*
  - *Au niveau bits* (niveau bas d'abstraction)
  - *Au niveau d'opérateurs arithmétiques* (seulement les opérateurs d'addition et de soustraction sont supportés pour la synthèse)
- *Opérateur d'addition bit par bit* – additionneur complet *Full Adder* - FA

Symbole



Somme :

$s = x \text{ XOR } y \text{ XOR } c_{in}$

Retenue :

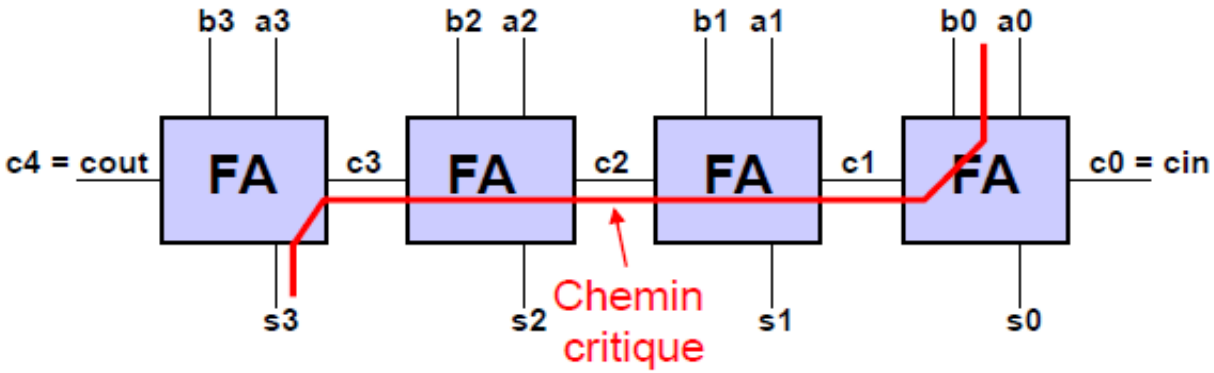
$c_{out} = (x \text{ AND } y) \text{ OR } (c_{in} \text{ AND } X) \text{ OR } (c_{in} \text{ AND } Y)$

Tableau de vérité

x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Blocs arithmétiques

- ***Réalisation d'un additionneur de quatre bits basé sur un additionneur complet***



- ***Chemin critique** – le chemin le plus long dans le flot de données (entre une entrée et une sortie quelconque) – il correspond à la fonction logique du bloc la plus complexe*
  - *Ici – nous avons 5 sorties, donc 5 fonctions logiques (c_{out}, s_0, s_1, s_2, s_3), les fonctions les plus complexes sont c_{out} et s_3. – elles représentent les chemins critiques du bloc*

Blocs arithmétiques

- ***Réalisation d'un additionneur de quatre bits* en utilisant un opérateur arithmétique (haut niveau d'abstraction)**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY addition IS
PORT (a, b     : IN  std_logic_vector(3 DOWNTO 0);
      s        : OUT std_logic_vector(3 DOWNTO 0);
      c_out    : OUT std_logic);
END addition;

ARCHITECTURE rtl OF addition IS
  SIGNAL s_int : std_logic_vector(4 DOWNTO 0);
BEGIN
  s_int <= ('0' & a) + ('0' & b);
  s     <= s_int (3 DOWNTO 0);
  c_out <= s_int(4);
END rtl;
```

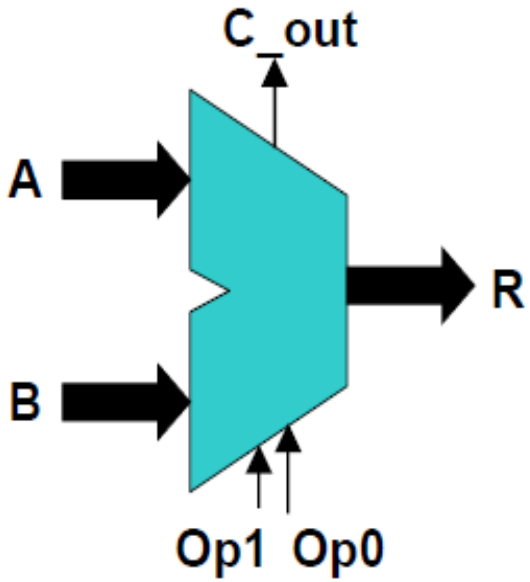
Paquetage nécessaire pour les opérations arithmétiques

Extension du *a* et *b* sur 5 bits (le cinquième bit sera la retenue)

A noter :
En réalité, le compilateur traduit la description de haut niveau d'abstraction (abstrait du matériel) en une structure d'addition bit par bit (probablement en utilisant l'additionneur complet), qui correspond à la structure logique disponible dans le matériel

Blocs arithmétiques

- *Unité arithmétique et logique*



Op1	Op0	Opération
0	0	R = A + B
0	1	R = A - B
1	0	R = A and B
1	1	R = A or B

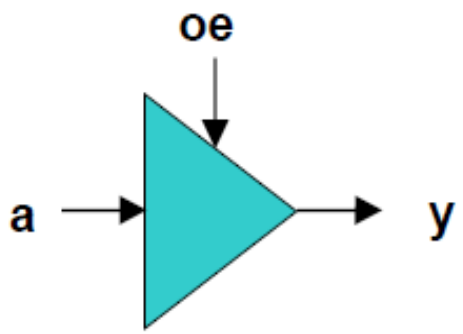
Blocs arithmétiques


```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

ENTITY alu IS
PORT (a, b      : IN std_logic_vector(7 DOWNTO 0);
      op1, op0  : IN std_logic;
      r         : OUT std_logic_vector(7 DOWNTO 0);
      c_out     : OUT std_logic);
END alu;

ARCHITECTURE rtl OF alu IS
SIGNAL oper     : std_logic_vector(1 DOWNTO 0);
SIGNAL int      : std_logic_vector(8 DOWNTO 0);

BEGIN
  oper   <= op1 & op0;  -- code operation
  c_out  <= int(8);
  r      <= int(7 DOWNTO 0);
  WITH oper SELECT
    int <= (('0'& a) + ('0'& b))   WHEN "00",
           (('0'& a) - ('0'& b))   WHEN "01",
           ('0' & (a AND b))       WHEN "10",
           ('0' & (a OR b))        WHEN OTHERS;
END rtl;
```

Paquetage nécessaire pour les opérations arithmétiques signées

Sorties commandées

- *Sorties trois états*

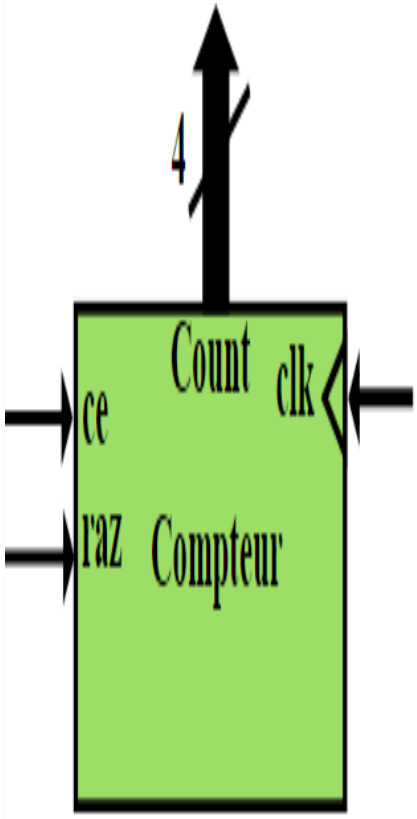


a	oe	y
0	0	Z
0	1	0
1	0	Z
1	1	1

```
-- assignation conditionnelle
  y <= a WHEN oe = '1' ELSE
       'Z';
```

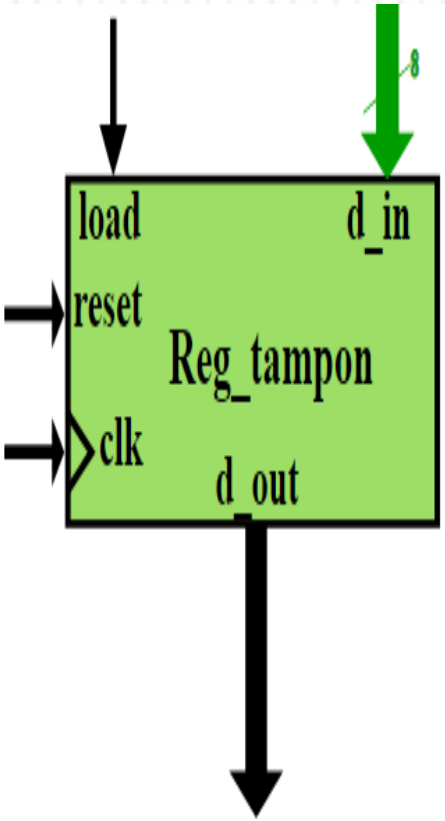
y doit être déclaré comme un signal de la logique standard et non pas comme un bit !

Compteur



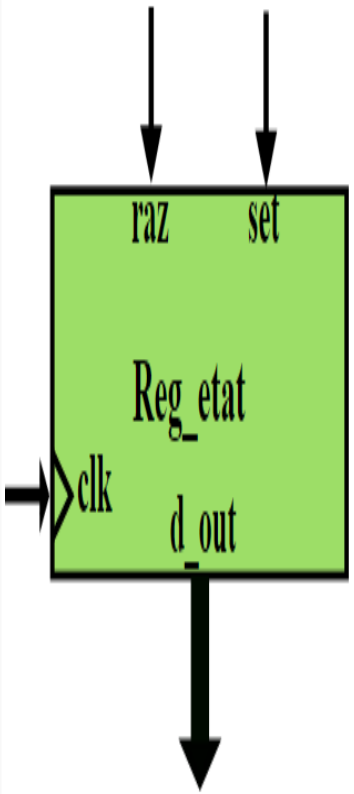

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity compteur is
port(   clk,raz,ce      :   in  std_logic;
    count           :   out std_logic_vector(3 downto 0));
end compteur;

architecture arch_compteur of compteur is
signal count_int : unsigned (3 downto 0);
begin
process(clk,raz)
    begin
    if rising_edge(clk) then
        if raz='1' then count_int <= (others => '0');
        elsif ce='1' then
            if count_int="1111" then count_int <= (others => '0'); -- fin
            else count_int <= count_int + 1; -- "+"(unsigned,int)
            end if;
        end if;
    end if;
end process;

count <= std_logic_vector(count_int); -- count copie de count_int

end arch compteur;
```

Registre tampon




```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity registre_tampon is
port(   clk,reset,load  :   in std_logic;
    d_in        :   in std_logic_vector(7 downto 0);
    d_out       :   out std_logic_vector(7 downto 0));
end registre_tampon;

architecture arch_registre_tampon of registre_tampon is
begin
    process(clk,reset)
        begin
        if reset='1' then d_out <= (others => '0');
        elsif rising_edge(clk) then
            if load='1' then d_out <= d_in;
            end if;
        end if;
    end process;
end arch_registre_tampon;
```

Registre d'état



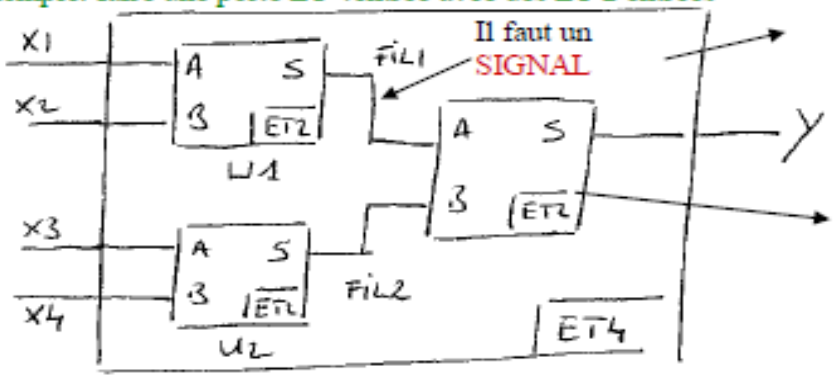

```
library IEEE;
use IEEE.std_logic_1164.all;

entity registre_etat is
port(   clk,raz,set :   in std_logic;
    d_out       :   out std_logic_vector(7 downto 0));
end registre_etat;

architecture arch_registre_etat of registre_etat is

begin
    process(clk)
        begin

        if rising_edge(clk) then
            if raz='1' then d_out <= (others => '0');
            elsif set='1' then d_out(0) <= '1'; d_out(7 downto 1) <= (others
            end if;
        end if;
    end process;

end arch_registre_etat;
```

Bilan

Les questions à se poser

- On identifie les fonctions et on les dessine *sur papier*
- On repère et nomme les entrées de chaque blocs (on évite d'utiliser les mêmes noms)
- On répertorie les signaux INTERNES (mot clé SIGNAL)
- Le bloc est-il combinatoire ou séquentiel?
  - ✓ Si séquentiel alors description avec le mot clé PROCESS + instructions autorisées
- Le bloc est-il utilisé plusieurs fois
  - ✓ Si oui il vaut mieux créer un composant (entity+ architecture)
  - ✓ Sinon le bloc est synthétiser par les lignes de codes directement

Exemple: faire une porte ET 4entrée avec des ET 2 entrées



ET4 est un composant (entity+architecture)

On créera 1 composant ET2 (entity+architecture)

Utilisé 3 fois pour décrire ET4

Bilan

Les conseils

- Pour décrire des systèmes combinatoires les instructions types « concurrentes » seront préférées
- L'ordre des instructions est SANS IMPORTANCE (car en parallèle)
- Il est souhaite de scinder les projets en composants simples
  - ✓ APPROCHE METHODOLOGIQUE TOP-DOWN
- Utilisation des bibliothèques IEEE

Squelette de description VHDL

```
--les libraries
library IEEE;
use IEEE.std_logic_1164.all;
……..

ENTITY LENIVEAUTOP (
……..)
End ENTITY

ARCHITECTURE …..

COMPONENT Truc
…
END COMPONENT
COMPONENT Machin
…
END COMPONENT

SIGNAL: …….
SIGNAL:  ……..

XX<="1110";
YY<= A AND B;
U1: Truc PORT MAP( …….);
S<= "10" when (A=B) else
      "00";
U2: Machin PORT MAP( …….);

With (Toto) select
 G<= ……

END ARCHITECTURE
```

Déclaration de composants créés

Utilisation des ressources disponibles

Environnements

❑ Environnements de développement intégré

Éditeur	Produit	Licence	Synthétiseur	Simulateur	Remarques
Xilinx	**ISE** Webpack	Propriétaire, gratuite, illimitée	Oui	Oui	Simulateur ModelSim XE Starter gratuit
Altera	**Quartus II** Web Edition	Propriétaire, gratuite, 6 mois renouvelable	Oui	Oui	Simulateur ModelSim Altera Starter Edition gratuite

Résumé de syntaxe VHDL

Résumé de syntaxe VHDL


Thank's

Question 1

- **Précisez les identificateurs corrects et incorrects :**
  - **NON_ET, Bascule_JK, NE555, P_**
  - **A#2, 2A, A$2, A__2, _P1, With**

Réponse 1

- **Précisez les identificateurs corrects et incorrects :**
  - **NON_ET, Bascule_JK, NE555, P_**
  - **A#2, 2A, A$2, A__2, _P1, With**

Question 2

- **Ces commentaires sont-ils corrects ?**
  - **-- Ceci est un commentaire !!!**
  - **q <= data2 after 10 ns; -- affectation de la sortie**
  - **-- attention l'instruction suivante sera ignorée : q <= data2 after 10 ns;**
  - **-- Pour être un bon étudiant, il ne faut pas rater ses cours, et pour ne pas rater ses cours, ne pas trop veiller.**

Réponse 2

- **Ces commentaires sont corrects sauf le dernier**
  - **-- Ceci est un commentaire !!!**
  - **q <= data2 after 10 ns; -- affectation de la sortie**
  - **-- attention l'instruction suivante sera ignorée : q <= data2 after 10 ns;**
  - **-- Pour être un bon étudiant, il ne faut pas rater ses cours, et pour ne pas rater ses cours, ne pas trop veiller.**

Question 3

- Indiquez le type des données
  - **12 0 1E6**
  - **12.0 0.0 0.456**
  - **1.23E-12 1.0e+6**
- **Donnez la forme générale d'écriture d'un entier basé**
  - **2#11111111# 16#FF# --Entiers de valeur 255**
- **Trouvez l'erreur ?**
  - **100 ps 3ns 5 v**

Réponse 3

- 12 0 1E6 Entiers
  12.0 0.0 0.456 Réels
  1.23E-12 1.0e+6 Réels avec exposant
- format : base#valeur#
  base comprise entre 2 et 16
  2#11111111# 16#FF# Entiers de valeur 255
- 100 ps 3ns 5 v
  Il faut toujours un espace entre la valeur et l'unité

Question 4

- **Précisez le type de littéral :**
  - Notés entre apostrophes : 'x' 'P' '2' '"' ' '
  - Notés entre guillemets, minuscules et majuscules y sont significatives : "Bonjour... " " " "23BVC$$Ld"
  - Utilisés pour affecter des signaux de type bit_vector, indiquez aussi la base

    X"FFF"

    B"1111_1111_1111"

    O"17"

Réponse 4

- **Précisez le type de littéral :**
  - **Notés entre apostrophes : 'x' 'P' '2' '"' ' ‘**

    caratère
  - **Notés entre guillemets, minuscules et majuscules y sont significatives : "Bonjour... " " " "23BVC$$Ld"**

    chaine de caractères
  - **Utilisés pour affecter des signaux de type bit_vector, indiquez aussi la base** chaine de bits

    X"FFF" -- base hexadécimale, longeur 12 bits

    B"1111_1111_1111" -- base binaire , longeur 12 bits

    O"17" -- base octale, longeur 6 bits

Question 5

- **Soit :**

  type GAMME is (DO,RE,MI,FA,SOL,LA,SI);

  subtype INDEX is integer range 16 downto 0 ;

Donnez l'attribut sur le type :

- **INDEX'left = ?**
- **INDEX'right = ?**
- **INDEX'low = ?**
- **INDEX'high = ?**
- **GAMME'right =?**
- **GAMME'high = ?**
- **GAMME'val(INDEX'leftof(0)) = ?**

GAMME'val(3) = ?
GAMME'pos(FA) = ?
GAMME'pos(DO) = ?
GAMME'pred(RE) = ?
INDEX'succ(13) = ?
INDEX'rightof(13) = ?
GAMME'succ(SI) = ?

Réponse 5

- **Soit :**

  type GAMME is (DO,RE,MI,FA,SOL,LA,SI);

  subtype INDEX is integer range 16 downto 0 ;

Attributs sur le type :

- INDEX'left = 16
- INDEX'right = 0
- INDEX'low = 0
- INDEX'high = 16
- GAMME'right =SI
- GAMME'high =SI
- GAMME'val(INDEX'leftof(0)) =RE

GAMME'val(3) = FA

GAMME'pos(FA) = 3

GAMME'pos(DO) = 0

GAMME'pred(RE) = DO

INDEX'succ(13) =14

INDEX'rightof(13) =12

GAMME'succ(SI) = Erreur

Il s'agit de caractéristiques de types ou d'objet qu'il est possible d'utiliser dans le modèle. Ils sont représentés de cette façon : <OBJET>'<ATTRIBUT>

Il existe des attributs sur les types, sur les objets de type tableau et sur les signaux.

Question 6

- Soit un signal A défini de la manière suivante :

  Signal A : std_logic_vector(5 downto 0) ;

  - A'high renvoie ?
  - A'low renvoie ?
  - A'left renvoie ?
  - A'right renvoie ?
  - A'range renvoie ?
  - A'reverse_range renvoie ?
  - A'length renvoie ?
  - A'ascending renvoie ?

Réponse 6

- Soit un signal A défini de la manière suivante :

  Signal A : std_logic_vector(5 downto 0) ;

  - A'high renvoie '5'
  - A'low renvoie '0'
  - A'left renvoie '5'
  - A'right renvoie '0'
  - A'range renvoie '5 downto 0'
  - A'reverse_range renvoie '0 to 5'
  - A'length renvoie '6'
  - A'ascending renvoie 'false'

Question 7

- Ces 2 architectures sont-elles équivalentes ?

```
architecture DESCRIPTION of DECOD1_4 is
begin
D0 <= (not(IN1) and not(IN0));        -- première instruction
D1 <= (not(IN1) and IN0);             -- deuxième instruction
D2 <= (IN1 and not(IN0));             -- troisième instruction
D3 <= (IN1 and IN0);                  -- quatrième instruction
end DESCRIPTION;
```

```
architecture DESCRIPTION of DECOD1_4 is
begin
D1 <= (not(IN1) and IN0);             -- deuxième instruction
D2 <= (IN1 and not(IN0));             -- troisième instruction
D0 <= (not(IN1) AND not(IN0));        -- première instruction
D3 <= (IN1 AND IN0);                  -- quatrième instruction
end DESCRIPTION;
```

Réponse 7

- Ces 2 architectures sont équivalentes
  - Penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description
  - L'ordre dans lequel seront écrites les instructions n'a aucune importance.

Question 8

- Ecrire l'instruction de décalage équivalente pour chacune des 4 instructions S1 :

```
-- Si A est de type std_logic_vector(7 downto 0)
S1 <= '0' & A(7 downto 1); -- décalage d'un bit à droite
S1 <= "000" & A(7 downto 3); -- décalage de trois bits à droite
S1 <= A(6 downto 0) & '0'; -- décalage d'un bit à gauche
S1 <= A(4 downto 0) & "000"; -- décalage de trois bits à gauche
```

Réponse 8

- Instruction de décalage équivalente pour chacune des instructions S1

```
-- Si A est de type std_logic_vector (7 downto 0)
```

- S1 <= A srl 1 ;
- S1 <= A srl 3 ;
- S1 <= A sll 1 ;
- S1 <= A sll 3 ;

Remarque: ne pas confondre **=>** (implique) et **<=** (affecte).

Question 9

- **L'instruction :** `if (CLK'event and CLK='1') then`
  permet de détecter un front montant du signal *CLK*
- Ecrire l'instruction qui permet de détecter un front descendant du signal CLK
- Ecrire une instruction équivalente à la première pour détecter un front montant de clk.
- Ecrire une instruction équivalente à la deuxième pour détecter un front descendant du signal CLK

Réponse 9

- *if (CLK'event and CLK = '1') then*
- *if (CLK'event and CLK = 'O') then*
- if(*rising_edge(CLK)*)
- *if(falling_edge(CLK))*

Les bibliothèques IEEE possèdent deux instructions permettant de détecter les fronts montants) ***rising_edge(CLK)*** ou descendants ***falling_edge(CLK).***

Question 10

- Qui suis-je ?
  - L'exécution a lieu à chaque changement d'état d'un signal de la liste de sensibilité.
  - Les instructions s'exécutent séquentiellement.
  - Les changements d'état des signaux par les instructions sont pris en compte à la **fin**.

Réponse 10

- Qui suis-je ?
  - L'exécution a lieu à chaque changement d'état d'un signal de la liste de sensibilité.
  - Les instructions s'exécutent séquentiellement.
  - Les changements d'état des signaux par les instructions sont pris en compte à la **fin**.
- Je suis un process

Question 11

- Que fait ce programme ?

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (
        CLOCK : in std_logic;
        RESET : in std_logic;
        Q     : out std_logic_vector(2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
   process (RESET,CLOCK)
   begin
      if RESET ='1' then
        CMP <= "000";
      elsif (CLOCK ='1' and CLOCK'event) then
        CMP <= CMP + 1;
      end if;
   end process;
   Q <= CMP;
end DESCRIPTION;
```

Réponse 11

- Ce programme décrit un compteur 3 bits avec remise à 0 asynchrone

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (
        CLOCK : in std_logic;
        RESET : in std_logic;
        Q     : out std_logic_vector(2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
   process (RESET,CLOCK)
   begin
      if RESET ='1' then
        CMP <= "000";
      elsif (CLOCK ='1' and CLOCK'event) then
        CMP <= CMP + 1;
      end if;
   end process;
   Q <= CMP;
end DESCRIPTION;
```

Question 12

- Que fait ce programme ?

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (
       CLOCK : in std_logic;
       RESET : in std_logic;
       Q     : out std_logic_vector (2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
  process (CLOCK)
  begin
    if (CLOCK ='1' and CLOCK'event) then
      if RESET ='1' then
         CMP <= "000";
      else
         CMP <= CMP + 1;
      end if;
    end if;
  end process;
  Q <= CMP;
end DESCRIPTION;
```

Réponse 12

- Ce programme décrit un compteur 3 bits avec remise à 0 synchrone

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (
        CLOCK : in std_logic;
        RESET : in std_logic;
        Q     : out std_logic_vector (2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0);
begin
   process (CLOCK)
   begin
      if (CLOCK ='1' and CLOCK'event) then
         if RESET ='1' then
            CMP <= "000";
         else
            CMP <= CMP + 1;
         end if;
      end if;
   end process;
   Q <= CMP;
end DESCRIPTION;
```

Question 13

- Que fait ce programe ?

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity DECODAGE is
      port (
            A15, A14, A13, A12, A11, A10  : in std_logic;
            RAM0                          : out std_logic;
            RAM1                          : out std_logic;
            RAM2                          : out std_logic;
            RAM3                          : out std_logic;
            ROM                           : out std_logic;
            INTER1                        : out std_logic;
            INTER2                        : out std_logic;
            INTER3                        : out std_logic);
end DECODAGE;

architecture DESCRIPTION of DECODAGE is

      signal ADRESSE: std_logic_vector(15 downto 0);

begin

ADRESSE <= A15 & A14 & A13 & A12 & A11 & A10 & "----------";
-- definition du bus d'adresses

ROM   <= '0' when (ADRESSE >= x"E000") and (ADRESSE <= x"FFFF") else '1';
RAM0  <= '0' when (ADRESSE >= x"0000") and (ADRESSE <= x"03FF") else '1';
RAM1  <= '0' when (ADRESSE >= x"0400") and (ADRESSE <= x"07FF") else '1';
RAM2  <= '0' when (ADRESSE >= x"0800") and (ADRESSE <= x"0BFF") else '1';
RAM3  <= '0' when (ADRESSE >= x"0C00") and (ADRESSE <= x"0FFF") else '1';

INTER1 <= '0' when (ADRESSE >= x"8000") and (ADRESSE <= x"8001") else '1';
INTER2 <= '0' when (ADRESSE >= x"A000") and (ADRESSE <= x"A001") else '1';
INTER3 <= '0' when (ADRESSE >= x"C000") and (ADRESSE <= x"C00F") else '1';

end DESCRIPTION;
```

Réponse 13

- Ce programme décode des adresses sur 16 bits

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity DECODAGE is
        port (
                A15, A14, A13, A12, A11, A10  : in std_logic;
                RAM0                          : out std_logic;
                RAM1                          : out std_logic;
                RAM2                          : out std_logic;
                RAM3                          : out std_logic;
                ROM                           : out std_logic;
                INTER1                        : out std_logic;
                INTER2                        : out std_logic;
                INTER3                        : out std_logic);
end DECODAGE;

architecture DESCRIPTION of DECODAGE is

        signal ADRESSE: std_logic_vector(15 downto 0);

begin

ADRESSE <= A15 & A14 & A13 & A12 & A11 & A10 & "----------";
-- definition du bus d'adresses

ROM     <= '0' when (ADRESSE >= x"E000") and (ADRESSE <= x"FFFF") else '1';
RAM0    <= '0' when (ADRESSE >= x"0000") and (ADRESSE <= x"03FF") else '1';
RAM1    <= '0' when (ADRESSE >= x"0400") and (ADRESSE <= x"07FF") else '1';
RAM2    <= '0' when (ADRESSE >= x"0800") and (ADRESSE <= x"0BFF") else '1';
RAM3    <= '0' when (ADRESSE >= x"0C00") and (ADRESSE <= x"0FFF") else '1';

INTER1  <= '0' when (ADRESSE >= x"8000") and (ADRESSE <= x"8001") else '1';
INTER2  <= '0' when (ADRESSE >= x"A000") and (ADRESSE <= x"A001") else '1';
INTER3  <= '0' when (ADRESSE >= x"C000") and (ADRESSE <= x"C00F") else '1';

end DESCRIPTION;
```