Module Master: VTF

Designs complexes sur FPGA

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

ReDS
Reconfigurable & Embedded Digital Systems

2ème partie

Etienne Messerli

24 septembre 2010

Contenu de la présentation

- Evolution des technologies des PLDs
- Méthodologies de conception
  - ✓évolution des méthodologies
  - ✓conception full-synchrone
- Rappel sur le VHDL pour la synthèse automatique
  - ✓paquetage Numeric_Std (Addition, comparaison)
  - ✓Instruction *process* et variable
  - ✓éléments mémoires et systèmes séquentiels
- Design re-use

Designs complexes sur FPGA

VHDL pour la synthèse automatique

Rappel notions de base

Fausses idées sur le langage VHDL

Le langage en tant que tel ne garantit pas :

- ✓qualité des descriptions
- ✓portabilité des descriptions
- ✓descriptions soient synthétisables
- ✓optimum (quantité de logique)
- ✓performance (fréquence maximum)

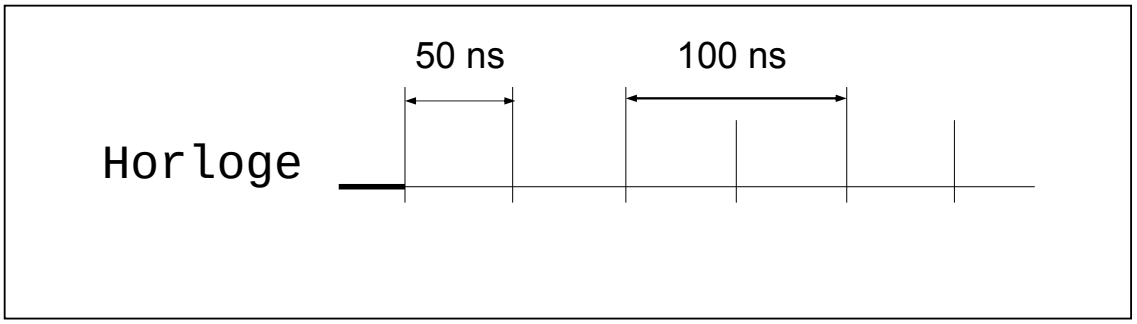
Mauvais concepteur + VHDL = **catastrophe**

Exemple de description VHDL

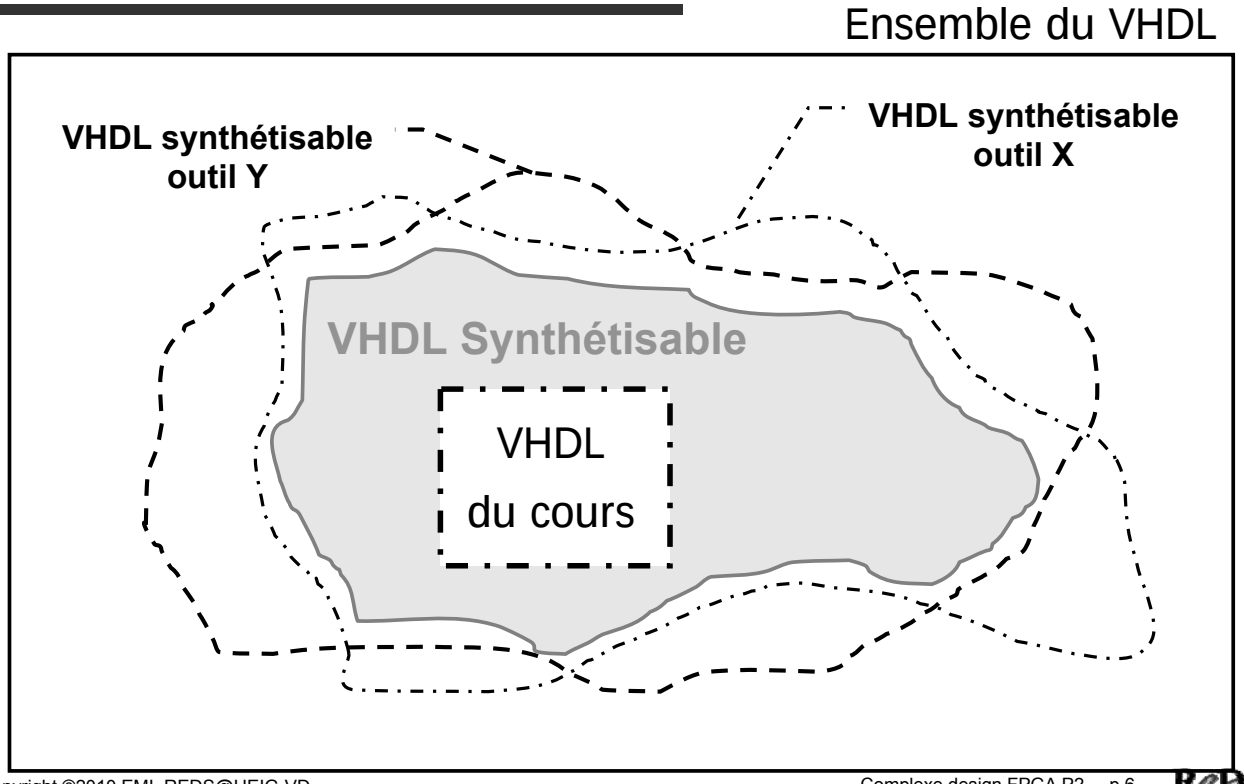
Soit la description VHDL suivante :

```
Horloge <= not Horloge after 50 ns;
```

Cela correspond au chronogramme suivant :



Ensemble synthétisable du VHDL



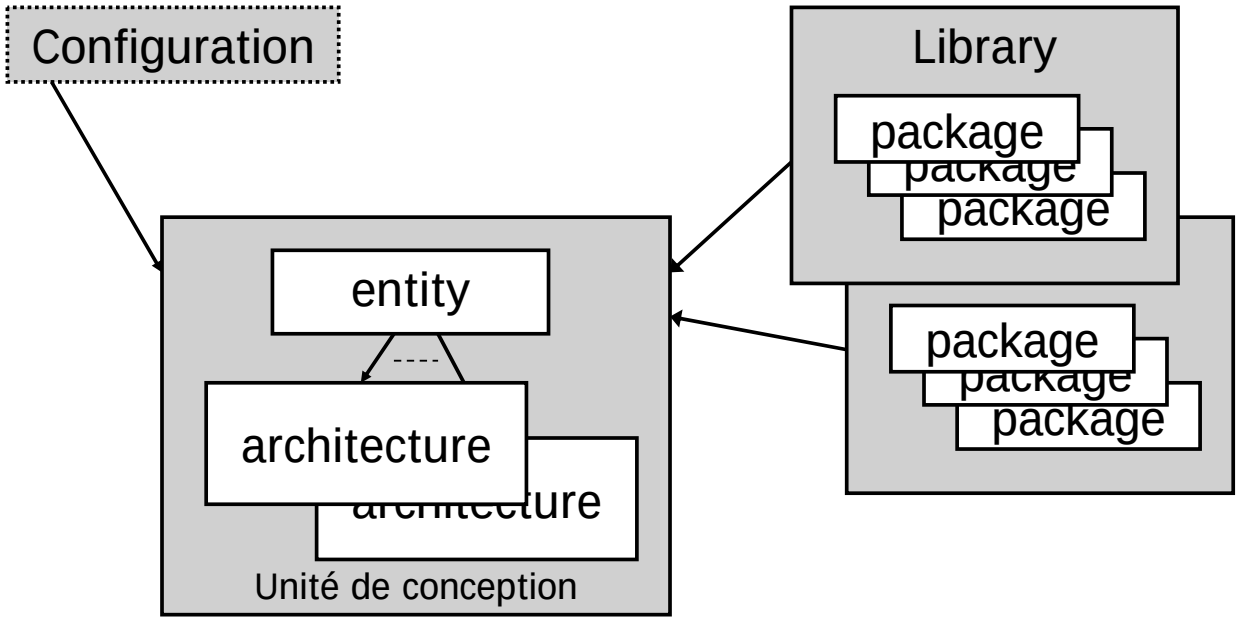
Conception avec le VHDL

- Il faut penser CIRCUIT.
- Une bonne conception commence par une décomposition du système (hiérarchie).
- Il faut imaginer l'architecture physique du système.
- Le VHDL n'est pas un outil de CONCEPTION.
- Le VHDL est un outil de DESCRIPTION.

La portabilité des descriptions

- Afin de garantir une bonne portabilité des descriptions :
  - ✓méthode indispensable
  - ✓une seule fonction par module VHDL
  - ✓faire des descriptions simples et lisibles
  - ✓expliciter les éléments mémoires
  - ✓utiliser uniquement les bibliothèques standardisées IEEE

Unité de conception



Les bibliothèques recommandées

- Portabilité des descriptions assurées en utilisant les bibliothèques IEEE :
  - ✓`Std_Logic_1164` types et fonctions de bases
  - ✓`Numeric_Std` opérations arithmétiques
- Bibliothèques propres (générale, projet):
  - ✓disposer des sources
  - ✓maîtriser le contenu
  - ✓pièces et boite à outils personnel

Les objets du langage

Objet : information manipulée par le langage

Ils sont répartis en quatre classes :

- ✓ **constantes** : objets de valeurs fixes
- ✓ **variables** : objets appartenant au monde *software*
- ✓ **signaux** : objets appartenant au monde *hardware*
- ✓ **fichiers** : objets servant à stocker des valeurs

On parle fréquemment de **classes** d'objets

Les types du langage

- scalaires :
  - ✓ énumérés (Boolean, Std_Logic,..),
    entiers (Integer, ...), physiques (Time),
    flottant (Real)
- composites : tableau (array)

  type Std_Logic_Vector **is** array(natural range<>) **of** Std_Logic;
- accès : pointeur pour accéder à des objets (dynamique)
- fichier : séquence de valeur d'un type donné

Affectation d'un signal

- Syntaxe de l'affectation d'un signal :

```
signal  <=  expression;
```

L'affectation représente un lien définitif entre le signal et le circuit générant l'expression (connexion)

L'affectation du signal ne modifie pas la valeur courante mais les valeurs **futures**

... affectation d'un signal

- Affecter une expression à un signal correspond à connecter un signal sur la sortie d'une porte.
- Plusieurs affectations sur un même signal correspond à un

court-circuit !

Affectation de variable

- Syntaxe de l'affectation d'une variable :

```
variable  :=  expression;
```

L'affectation de la variable est instantanée, ensuite il n'existe plus aucun lien entre la variable et l'expression

Le type Std_uLogic (type énuméré)

Défini par le paquetage IEEE.Std_Logic_1164

```
type Std_uLogic is (
      'U',-- état non initialisé
      'X',-- état inconnu fort
      '0',-- état logique 0 fort
      '1',-- état logique 1 fort
      'Z',-- état haute impédance
      'W',-- état inconnu faible
      'L',-- état logique 0 faible
      'H',-- état logique 1 faible
      '-' -- état indifférent, don't care );
```

… type Std_uLogic …

Etats utilisables pour l'affectation d'un signal en synthèse

```
type Std_uLogic is (
        …
       'X',-- pour cas de simulation
       '0',-- état 0
       '1',-- état 1
       'Z',-- état haute impédance
        …
        …
        …
       '-' -- état indifférent, don't care );
```


… type Std_uLogic

Etats utilisables pour tester l'état d'un signal en synthèse

```
type Std_uLogic is (
        …
        …
       '0',-- état 0
       '1',-- état 1
        …
        …
        …
        …
        …                                    );
```

… type Std_uLogic …

Etats utilisables pour l'affectation d'un signal en spécification et en simulation

```
type Std_uLogic is (
        …
       'X',-- pour cas de simulation
       '0',-- état 0
       '1',-- état 1
       'Z',-- état haute impédance
        …
       'L',-- état 0 faible => pull-down
       'H',-- état 1 faible => pull-up
       '-' -- état indifférent, don't care );
```


Std_Logic et Std_ulogic

- **Std_ulogic** : L'interconnexion entre deux signaux est interdite (unresolved).
  - ✓ cas des sorties standards des circuits numériques. Elles ne peuvent pas être connectées ensembles.
- **Std_Logic** : Interconnexion possible grâce à l'utilisation d'une fonction de résolution
  - ✓ correspond aux sorties spéciales (trois états, collecteur ouvert, ..) qui sont interconnectées.

Type Std_Logic généralement utilisé dans l'industrie

=> utilisé pour le cours

Fonction de résolution pour type Std_Logic

- La valeur affectée au point d'interconnexion de deux sorties Std_logic est régie par la table de résolution suivante:

```
constant resolution_table : stdlogic_table := (
--  ---------------------------------------------------------
--  |  U    X    0    1    Z    W    L    H    -        |   |
--  ---------------------------------------------------------
    ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
    ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
    ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
    ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
    ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
    ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
    ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
                                                    );
```


Type utilisé dans l'industrie :

Le type le plus couramment utilisé est le

Std_Logic

En cas d'interconnexion erronée :

- ✓ pas de verrouillage par le langage VHDL
- ✓ détecté en simulation par un état 'X'
- ✓ détecté lors de la synthèse par une erreur : multiples *drivers*

Déroulement concurrent et séquentiel

- Dans un langage informatique :
  - ✓les instructions ont un déroulement séquentiel
- Dans un circuit :
  - ✓toutes les portes fonctionnent simultanément
  - ✓tous les signaux évoluent de manière concurrente
- Le langage VHDL dispose d'instructions concurrentes pour la description de circuits (matériel)

Structure de l'entité (entity)

```
-- Librairie IEEE
library IEEE;
use IEEE.Std_Logic_1164.all; --Defini type Std_Logic

entity Exemple is
  port(Entree_i  : in  Std_Logic;
       Vecteur_i : in  Std_Logic_Vector(3 downto 0);
       Sortie_o  : out Std_Logic;
       BiDir_io  : inout Std_Logic);
end Exemple;
```

Structure de l'architecture (architecture)

```
architecture Style_Description of Exemple is
  --zone de déclaration
begin

  --Instructions concurrentes  .......

  process (Liste_De_Sensibilité)
  begin
     --Instructions séquentielles  .......
  end process;

end Style_Description;
```


Architecture: zone de déclaration

- Déclaration de signaux internes

```
signal Interne_s : Std_Logic;
signal Vect_s    : Std_Logic_Vector(4 downto 0);
```

- Déclaration de constantes

```
constant Val_c : Std_Logic_Vector(2 downto 0):="101";
```

- Déclaration de composants
- Déclarations de types, de procédures et de fonctions

Architecture, zone de description

- Déroulement concurrent :
  - ✓ toutes les instructions concurrentes
  - ✓ les processus (instr. concurrente !)
- Si plusieurs processus :
  - ✓ exécution concurrente entre les processus
- Déroulement séquentiel :
  - ✓ **UNIQUEMENT** à l'intérieur d'un processus

Styles de description RTL

Styles de descriptions	Abréviations
• Equations logiques	Logique
• Table de vérité	TDV
• Flot de données	Flot_Don
• Comportementale	Comport
• Machine d'états	M_Etat
• Structurelle	Struct
•	

Autres types de description

Types de descriptions	Abréviations
• Spécification	Specif
• Banc de test	Test_Bench
•	

Les instructions concurrentes

- Affectation (`Y <= A and C;`)
- Affectation avec condition
  (`Y <=.. when .. else .. ;`)
- Instruction de sélection
  (**`with .... select`**)
- Instanciation de composants (**`port map`**).
- Processus (**`process`**)

Instruction d'affectation …

- Syntaxe de l'affectation simple :

```
Signal <= Expression;
```

Exemples d'expression :

```
Expression Operator Expression
Identifier
Aggregat
…
```

Exemples d'affectation :

```
Sortie  <= Sel0 or Sel1 or Sel2;
Signal  <= Vect_8Bits(3);
Vect(0) <= (A or B) and C;
Signal  <= Entree;
```


… instruction d'affectation …

- Syntaxe de l'affectation avec une condition :

```
Signal1 <= Expression_True when Condition else
           Expression_False;
```

Exemples :

```
Egal <= '1' when (Valeur="1001") else
        '0';

Result <= A or B when (c='1') and (En='1') else
          '0';
```

… instruction d'affectation

- Syntaxe de l'affectation avec plusieurs conditions :

```
 Signal1 <= Expression when Cond_booleen else
            Expression when Cond_booleen else
            ...
            Expression;
```

Exemple :

```
 Out <= '1'      when (Force1 = '0') else
        A and B when (FctAnd = '1') else
        A ;
```

Remarque : L'instruction when…else implique une notion de priorité entre les différentes conditions

Instruction de sélection …

- Syntaxe de l'instruction de sélection :

```
with Signal_Commande select
  Signal_Affecte <= Expression1 when "Com_Etat1",
                    Expression2 when "Com_Etat2",
                                ...
                    ExpressionN when others;
```

Remarque : le terme **`others`** définit toutes les autres combinaisons possibles de l'état du signal de commande

… instruction de sélection

Exemple :

```
with Val_Sel select
  Sortie <= Entr_A when "00",
            Entr_B when "01",
            Entr_C when "10",
            Entr_D when "11",
            'X'    when others; --simulation
```

Remarque : le terme **others** représente les combinaisons "UH", "ZZ", "X0", … du signal Val_Sel

Instanciation d'un composant …

Dans la zone déclaration de l'architecture :

```
Component Nom_Composant is -- is accepte en VHDL93
  port (Ports_Entree : in Std_Logic;
          .....
          Ports_Sortie : out Std_Logic);
end component;
for all : Nom_Composant use
          entity work.Nom_Entity(Style_Description);
```

Dans la zone de description de l'architecture :

```
[Label:] Nom_Composant port map
          (Signal_Entrees => Signal_Achitecture1_s,
           Signal_Sorties => Signal_Achitecture2_s);
```

… instanciation d'un composant …

```
architecture Struct of Exemple is
  component Porte_Et is -- is accepte en VHDL93
           port (A_i, B_i : in  Std_Logic;
                 Z_o      : out Std_Logic);
  end component;
  for all : Porte_Et use
              entity work.Porte_Et(Flot_Don);
begin
   ..
  U1: Porte_Et port map (A_i => Entr_i,
                         B_i => Signal_s,
                         Z_o => Sortie_o);
   ..
end Struct;
```


… instanciation d'un composant

Cas d'une sortie non utilisée (unconnected) :

```
architecture Struct of Exemple is
  component Module
           port (  …
                 Sortie_o  : out Std_Logic
                   …       );
  end component;
  for all : Module use entity  work.Module(Comport);
begin
   ..
  U1: Module port map (   …
                       Sortie_o => open,
                       --Sortie_o => pas connecté
                          …            );
   ..
end Struct;
```

Visseries et astuces

- Voici des éléments très pratique pour les descriptions VHDL :
  - ✓opérateur de concaténation "&"
  - ✓notation par agrégat et mot clé others
  - ✓affectation en hexadécimal
  - ✓les attributs les plus courants
  - ✓comparaison de vecteur et pièges

L'opérateur de concaténation "&"

Exemples de regroupement de vecteurs :

```
Vect4 <= D & C & B & A;
Vect8 <= Vect10(9 downto 6) & A & B & "00";
```

Exemples de décomposition de vecteurs :

```
Vect4 <= Vect8(5 downto 2);
Signal <= Vect8(4);
Vect8(5 downto 3) <= "010";
```

Affectation par notation d'agrégat

- La notation par agrégat permet d'indiquer la valeur d'un type composite, comme par exemple les tableaux (*array*). Cela est très utile pour l'affectation des vecteurs (type *array*). Voici des exemples :

```
--Soit la déclaration suivante:
signal Vecteur : Std_Logic_Vector(3 downto 0);

--Nous pouvons affecter le vecteur comme suit :
Vecteur <= ('1','0','0','1') ; --idem <="1001";
```


Agrégat et mot clé others …

Le mot clé "others" permet d'indiquer que tous les autres éléments de l'agrégat sont affectés par une même valeur.

Si le mot clé "others" est utilisé seul, cela permet d'affecter TOUS les éléments avec la même valeur. Exemples :

```
Vect_A <= (others => '0'); --tout à 0
Vect_C <= (others => 'Z'); --tout à Z

--Ou affecté avec le même signal My_Signal
Vect_4 <= (others => My_Signal);
```

… agrégat et mot clé others

La notation par agrégat permet d'indiquer le numéro de l'élément affecté. Combiné avec le mot clé "others", cela permet des affectations très pratique. Exemples :

```
--affectation : MSB à '1', autres bits à '0'
Vect8 <= (7 => '1', others => '0');

--affectation : MSB à '0', autres bits à '1'
Vect8 <= (7 => '0', others => '1');

--Affectation : LSB à '1', autres bits à '0'
Vect8 <= (0 => '1', others => '0');
```


Affectation en hexadécimal …

- En VHDL93 il est possible de donner des valeurs en hexadécimal.
- Exemple d'affectation de vecteurs:
  - ✓`Vecteur_8Bits <= x"6A";`
  - ✓`Vecteur_24Bits <= x"2A_4F5C"`
  - ✓`Vecteur_4Bits <= x"8";`
- Attention valable uniquement pour des vecteurs de longueurs : 4, 8, 12, 16 ...

  donc des **multiples de 4 !!**

… affectation en hexadécimal

- Cas de vecteurs de longueur non multiple de 4 :

  Déclarer une constante ayant la taille multiple de 4 supérieur

```
constant Cst16 : Std_Logic_vector(15 downto 0)
                                        := x"2A04";
```

  puis

```
 Vect14bits <= Cst16(13 downto 0);
```

  Ou

```
 Vect14bits <= "10" & x"A04";
```


Attributs prédéfinis pour tableaux (*array*) …

Ces attributs sont très utiles pour rendre les descriptions paramétrables:

- ✓ permettent de manipuler des *array* (exemple : Std_logic_Vector(7 downto 0))
- ✓ nécessaires pour rendre les descriptions paramétrables (indépendantes de la taille des tableaux)
- ✓ à utiliser avec l'opérateur de concaténation & et la notation par agrégat
- ✓ utiles pour la synthèse, les spécifications et les test benchs

… les attributs pour les *array*…

Voici les principaux attributs pour les *array*:

- ✓'left : indice de gauche
- ✓'right : indice de droite
- ✓'high : indice supérieur (MSB)
- ✓'low : indice inférieur (LSB)
- ✓'length : longueur du tableau (array)
- ✓'range : intervalle des indices
- ✓'reverse_range : intervalle inverse des indices

… les attributs pour les *array*

- Exemple d'utilisation des attributs :

```
Data : Std_Logic_Vector(7 downto 0);
```

Dans ce cas :	correspond à
`Data'left = Data'high`	`7`
`Data'right = Data'low`	`0`
`Data'length`	`8`
`Data'range`	`7 downto 0`
`Data'reverse_range`	`0 to 7`

Exemples utilisation attributs *array* ...

- Déclaration d'un signal interne :

```
signal Compteur: Std_Logic_Vector(7 downto 0);
signal Cpt_Int: Unsigned(Compteur'range);
```

- Décalage à droite :

```
Vect_SHR <= '0' & Vecteur(vecteur'high downto 1);
```

- Rotation à gauche :

```
Vect_ROL <= Vect(Vect'high-1 downto 0) &
                                    Vect(Vect'high);
```

- Initialiser un vecteur signé à la valeur min

```
Vect <= (Vect'high => '1', others => '0');
```


Comparaison Std_Logic_Vector ...

- La comparaison se fait en comparant bit à bit les deux vecteurs en commençant par les MSB !
- Exemple de comparaison :

```
signal REG : Std_Logic_Vector(4 downto 0);
signal CNT : Std_Logic_Vector(3 downto 0);
--   REG   >   CNT   résultat
-- "01111" > "0100"  True      correct
-- "01111" > "1000"  False     ERRONÉ
```

Voir : paquetage Numeric_Std

Comparaison Sdt_Logic_Vector et '-' …

L'exemple ci-dessous n'est pas correct :

```
signal Adresse : Std_Logic_Vector(3 downto 0);
signal CS : Std_Logic;

begin
  CS <= '1' when (Adresse = "1---") else '0';
    -- Résultat comparaison toujours FAUSSE
```

Le signal CS est toujours à '0' !

… compar. Sdt_Logic_Vector

Voici la bonne description :

CS doit être actif lorsque Adresse = 1ΦΦΦ,

donc lorsque Adresse varie de 1000 à 1111

```
signal Adresse : Std_Logic_Vector(3 downto 0);
signal CS : Std_Logic;

begin
  CS <= '1' when (Adresse(3) = '1') else '0';
  -- Description correcte au cahier des charges
```

Paquetage Numeric_Std

- Bibliothèque IEEE
- Norme IEEE-1076.3, 1997
  - ✓paquetages Numeric_Bit & Numeric_Std
- Déclaration :

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;
```


… paquetage Numeric_Std …

- But du paquetage :
  - ✓types numériques et fonctions arithmétiques pour la synthèse
- Défini 2 types numériques :

  UNSIGNED : vecteur représentant un nombre non signé

  SIGNED : vecteur représentant un nombre signé

  Remarques :

  il s'agit de nombres ENTIER en binaire

  ces 2 types sont basés sur le `Std_Logic`

… paquetage Numeric_Std

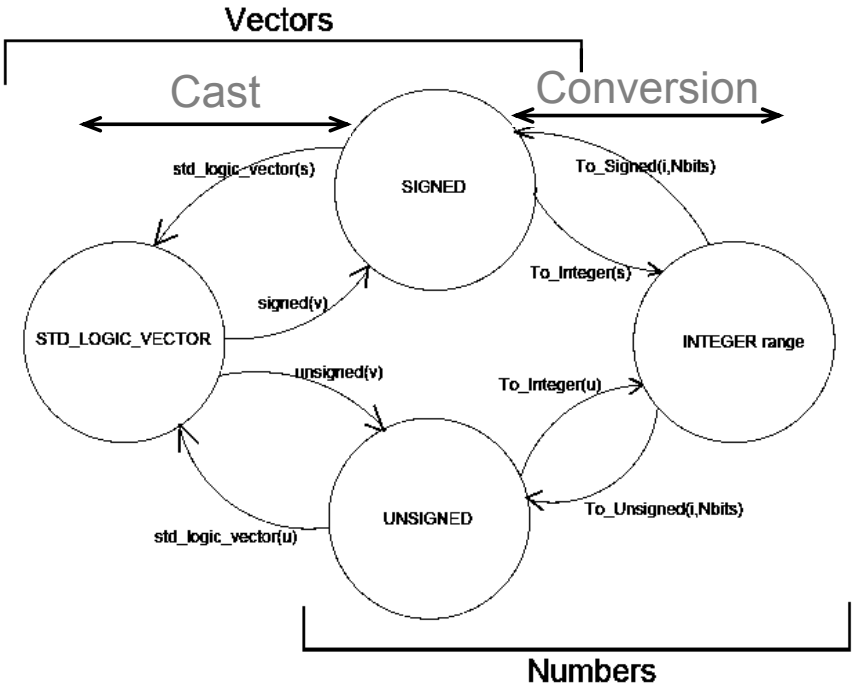
- Bit le plus à gauche :
  - ✓bit le plus significatif, MSB (Most Significant Bit)
- Représentation nombre signé :
  - ✓représentation en complément à 2
- Paquetage contient :
  - ✓surcharge des opérateurs arithmétiques pour les types UNSIGNED et SIGNED
  - ✓fonctions de conversion et d'adaptation

Types UNSIGNED et SIGNED

- Basé sur le type : Std_Logic
- Std_Logic_Vector <> Unsigned ou Signed :
  - ✓spécifie une interprétation différente
    - →vecteur binaire
    - →nombre entier en binaire (signé ou non signé)
- Définition :

```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED   is array (NATURAL range <>) of STD_LOGIC;
```

Représentation : vecteurs/nombres



Adaptation de type (*cast*)

```
-- Déclaration des signaux :
signal Vecteur : Std_Logic_Vector(7 downto 0);
signal Nombre  : Unsigned(7 downto 0);

--Exemples d'adaptation de type (cast):
Nombre  <= Unsigned(Vecteur);
Vecteur <= Std_Logic_Vector(Nombre);

--Basé sur le Std_Logic, donc :
Vecteur(i) <= Nombre(i);   --correct
Nombre(i)  <= Vecteur(i);  --correct
```

Fonctions de conversion ...

- Fonction To_Integer :

  To_Integer (ARG: Unsigned) return Natural
  To_Integer (ARG: Signed) return Integer

- Fonction To_Unsigned :

  To_Unsigned (ARG, SIZE: Natural) return Unsigned

- Fonction To_Signed :

  To_Signed (ARG: Integer; SIZE: Natural) return Signed

... fonctions de conversion

```
-- Déclaration des signaux :
signal Nbr_Entier : Natural;
signal Nbr_Bin : Unsigned(7 downto 0);

--Exemples de conversion :
Nbr_Entier <= To_Integer(Nbr_Bin);

Nbr_Bin <= To_Unsigned(Nbr_Entier,8);

-- en utilisant l'attribut : 'length
Nbr_Bin <=
      To_Unsigned(Nbr_Entier,Nbr_Bin'length);
```

Fonctions "+" et "-"

- Caractéristiques :

  2 opérandes de tailles différentes
  résultat aura la taille du plus grand
  Result :MAX(L'LENGTH, R'LENGTH)-1 downto 0

- Combinaisons des 2 opérandes (L & R):

  (L, R: UNSIGNED) return UNSIGNED
  (L: UNSIGNED; R: NATURAL) return UNSIGNED
  (L: NATURAL; R: UNSIGNED) return UNSIGNED
  (L, R: SIGNED) return SIGNED
  (L: INTEGER; R: SIGNED) return SIGNED
  (L: SIGNED; R: INTEGER) return SIGNED

Exemples d'addition

```
-- Déclaration des signaux :
signal N_A, N_B : Unsigned(7 downto 0);
signal Somme    : Unsigned(7 downto 0);

--Exemples d'addition :
Somme <= N_A + N_B;
Somme <= N_A + "0001";
Somme <= N_B + 1;
Somme <= N_A + "00110011";
--Erroné : Somme <= N_B + '1';
```

Fonctions de comparaison

- Caractéristiques :

  Opérandes de tailles différentes
  Résultat de type booléen

- Combinaisons des 2 opérandes (L & R):

  (L, R: UNSIGNED) return BOOLEAN
  (L: UNSIGNED; R: NATURAL) return BOOLEAN
  (L: NATURAL; R: UNSIGNED) return BOOLEAN
  (L, R: SIGNED) return BOOLEAN
  (L: INTEGER; R: SIGNED) return BOOLEAN
  (L: SIGNED; R: INTEGER) return BOOLEAN

Exemples de comparaison ...

```
--déclaration des signaux
signal nSgn_A,nSgn_B : Unsigned(3 downto 0);

--résultat de la comparaison est un booléen
  nSgn_A =  "1010"  ou  nSgn_A = 10

  nSgn_A /= nSgn_B
```

… exemples de comparaison

```
--déclaration des signaux
signal Sgn_A, Sgn_B : Signed(3 downto 0);

--résultat de la comparaison est un booléen
  Sgn_A < "1100"  ou  Sgn_A < -4

  Sgn_A > Sgn_B
```


Exercices : Paquetage Numeric_Std

- Compte sur le réseau :
  - ✓utilisateur : Reds_user
  - ✓mot de passe : REDS
  - ✓domaine : einet
- Logiciels utilisés :
  - ✓synthétiseur : Precision
  - ✓puis simulation avec QuestaSim
- Exercice : ADD7
- Directory de travail :
  - ✓D:\VTF\Exos_VHDL\Add7

Déroulement concurrent et séquentiel

- Dans un langage informatique classique :
  - ✓ les instructions ont un déroulement séquentiel
- Dans un circuit :
  - ✓ toutes les portes fonctionnent simultanément
  - ✓ tous les signaux évoluent de manière concurrente
- Le langage VHDL dispose d'instructions concurrentes pour décrire le comportement des circuits réels (matériel)

Processus concurrents ...

- Toutes les portes d'un circuit sous tension fonctionnent continuellement →
  un programme principal VHDL est constitué d'un ensemble de processus concurrents
- L'ordre dans lequel les processus apparaissent dans le listing est indifférent

```
D <= not C;
C <= A and B;
```

est identique à

```
C <= A and B;
D <= not C;
```

…processus concurrents …

- Des processus simples peuvent être déclarés de façon implicite, en les décrivant avec une affectation simple, conditionnelle ou sélectionnée

```
-- une affectation simple,
-- hors d'une declaration de process
A <= B or C ;
```

revient au même que :

```
process (B,C) begin
  A <= B or C ;
end process;
```


…processus concurrents

```
A <= '1' when B='1' else
     '1' when C='1' else '0';
-- ATTENTION : l'affectation conditionnelle avec when else
-- ne peut etre utilisee qu'a l'exterieur d'une declaration
-- de process (instruction strictement concurrente)
```

revient au même que :

```
process (B,C)
begin                          -- ATTENTION :
  if B='1' then                -- la structure conditionnelle
    A <= '1';                  -- if then else ne peut etre
  elsif C='1' then             -- utilisee qu'a l'interieur
    A <= '1';                  -- d'une declaration de process
  else                         -- (strictement sequentielle)
    A <= '0';
  end if;
end process;
```

L'instruction *process* …

- Le langage VHDL dispose d'une instruction *process* dont la syntaxe est :

```
process (Liste_De_Sensibilité)
  --zone de déclaration
begin

  --Zone pour instructions
  --              séquentielles ....

end process;
```


…instruction *process* …

- Déclenchement:
  - ✓ s'active lorsqu'un des signaux de sa liste de sensibilité (ses entrées) change de valeur,
  - ✓ et se rendort lorsque toutes les instructions séquen-tielles ont été évaluées une fois (*end process*)
- Fonctionnement:
  - ✓ exécution séquentielle des instructions à l'intérieur d'un *process*
  - ✓ Le temps simulé est stoppé durant l'exécution d'un *process*
- Exécution:
  - ✓ les affectations aux signaux ne prennent effet qu'après l'endormissement du process (fin de l'exécution de l'algorithme)

L'instruction *process* : résumé

- Activation lorsqu'un signal de la liste de sensibilité change
- Exécution séquentielle à l'intérieur
  Utilisation des instructions séquentielles !
- Le temps **ne progresse pas** durant l' exécution
- Variables utilisables uniquement à l'intérieur d'un processus
- Signaux mis à jour à la fin du process

Processus, zone de déclaration

- Déclaration de :
  - ✓variables
  - ✓constantes
  - ✓types et sous-types
  - ✓fonctions
  - ✓procédures
- Déclaration de signaux ou de composants interdite

Processus et liste de sensibilité …

- Avec liste de sensibilité :
  - ✓instruction *wait* interdite
  - ✓utilisé pour les descriptions synthétisables
  - ✓processus activé **uniquement** lorsqu'un signal de la liste de sensibilité change
  - ✓processus endormi à la fin (*end process*)
  - ✓permet de décrire avec un algorithme :
    - →système combinatoire ou
    - →système séquentiel ou
    - →model, spécification.

… processus et liste de sensibilité …

- Sans liste de sensibilité :
  - ✓ utilisation de l'instruction *wait*
  - ✓ utilisé pour fichiers de simulation & spécification
  - ✓ processus activé à l'instant 0 ns
  - ✓ processus endormi à chaque *wait*, puis activé selon *wait*
  - ✓ réactivé automatiquement à la fin (end process)

  Attention :
  - ✓ processus sans liste de sensibilité et sans *wait* à une durée d'exécution nul !

Bloque le simulateur

… processus et liste de sensibilité

- Liste de sensibilité ou instruction wait :

```
process (Signal,…)
begin

   ...

end process;
```

ou

```
process
begin
   ...
   wait on Signal,…;
   ...
   wait on Signal,…;
   ...
end process;
```


Les instructions séquentielles

Les instructions séquentielles doivent être placées à l'intérieur d'un processus

- Affectation (`Y <= A and C;`)
  - ✓ le signal Y sera actualisé à la fin du processus
- Affectation de variable (`Var1 := D and X;`)
  - ✓ la variable sera actualisée immédiatement
- Instruction de test (**`if ... then else`**)
- Instruction de choix (**`case ... is when`**)

Syntaxe de l'instr. d'affectation d'un signal

Affectation d'un signal :

```
Signal <= Sign_A fonct_logique Sign_B;
```

Le signal sera affecté à **l'endormissement** du processus (**`end process`** ou **`wait ...`**)

Remarque :

L'affectation avec condition (`..<= ..when..`) n'est pas utilisable à l'intérieur d'un *process*. Ce n'est pas une instruction séquentielle

Syntaxe de l'instr. d'affectation d'une variable

- Affectation d'une variable :

```
Variable1 := Var2 fonct_logique Var3;
```

- Exemple :

```
Temp := Var0 or Var1 or Var3;
```

La variable est actualisée **immédiatement**

L'utilisation des variables est possible seulement à l'intérieur de l'instruction process

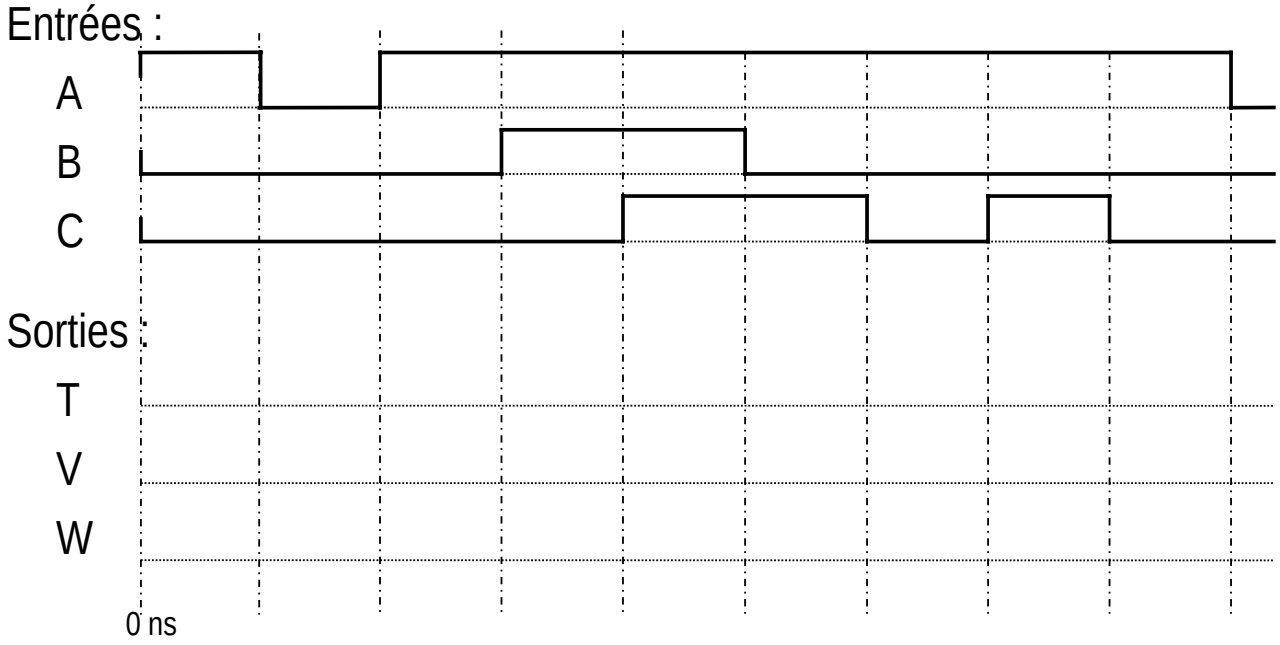
Exemple de processus I

Description d'un système combinatoire avec un processus

```
process (A, B, C)
begin
  T <= A and W;
  V <= C and T;
  W <= C or  B;
end process;
```

Complétez le chronogramme ci-après

Chronogramme I



Remarque sur processus I

Cette description est non synthétisable

```
process (A, B, C)
begin
  T <= A and W;
  V <= C and T;
  W <= C or  B;
end process;
```

Quelle erreur fait que cette description est non synthétisable ?

page volontairement laissée vide

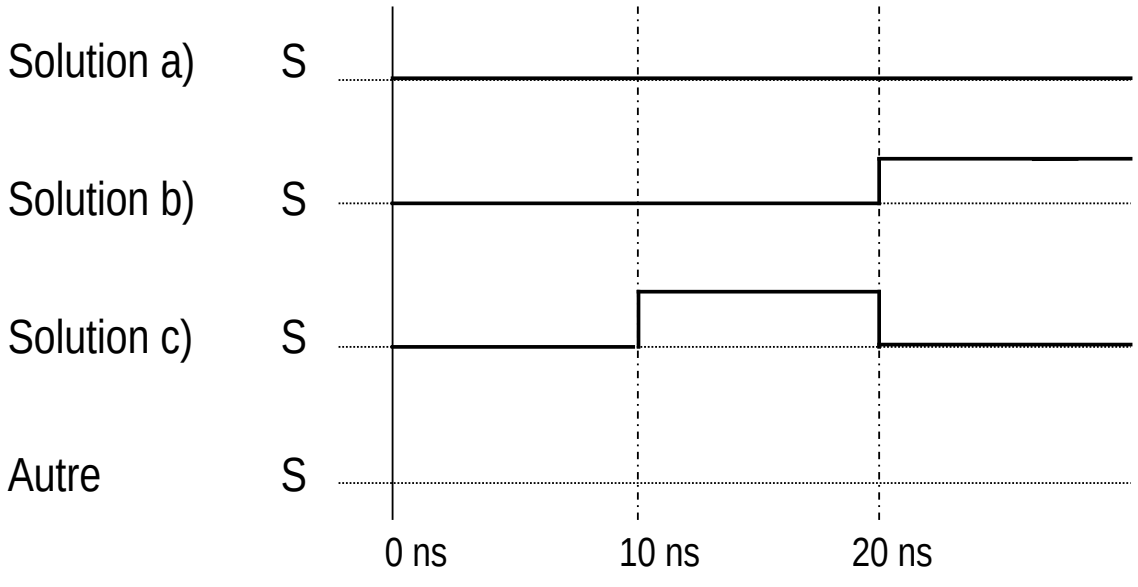
Exemple de processus II

Comportement séquentielle à l'intérieur d'un process

```
process
begin
  S <= '0';
  S <= '1' after 20 ns;
  S <= '1' after 10 ns, '0' after 20 ns;
  wait ;
end process;
```

Complétez le chronogramme ci-après

Quel chronogramme est correct ?



Syntaxe de l'instruction conditionnelle

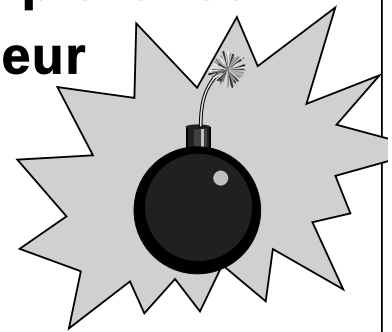
```
process ( ..... )
begin
  if Condition1 then
     --ZoneInstructions_1
  elsif Condition2 then
     --ZoneInstructions_2

    ...

  else
     --ZoneInstructions_n
  end if;
end process;
```


Exemple d'instruction conditionnelle

```
process (Sel, Enable)
begin

  if Enable = '1' then
    if Sel = "00" then
      Sorties <= "0001";
    elsif Sel = "01" then
      Sorties <= "0010";
    elsif Sel = "10" then
      Sorties <= "0100";
    else
      Sorties <= "1000";
    end if;

  end if;
end process;
```

Démultiplexeur

Exemple avec erreur



Syntaxe de l'instruction de choix

```
process ( ..... )
begin
 case Expression is
    when Valeur1   => --ZoneInstructions_1;
    when Valeur2   => --ZoneInstructions_2;
       ...
       ...
    when others    => --ZoneInstructions_n;
 end case;
end process;
```


Exemple d'instruction de choix

Décodeur d'adresse

```
process(adresse)
begin
  CS_A <= '0'; --Valeur par defaut
  CS_B <= '0'; --Valeur par defaut
  case Adresse is
     when "000"|"001"|"010" => CS_A <= '1';
     when "011"|"100"       =>  null;
     when "101"|"110"|"111" => CS_B <= '1';
     when others => CS_A <= 'X';CS_B <= 'X';
                    --cas pour simulation
  end case;
end process;
```

Description SLC avec un process

IMPORTANT (SLC : Système Logique Combinatoire)

- Toutes les entrées du SLC **doivent** être dans la liste de sensibilité.
- Toutes les sorties du SLC doivent être initialisées **au début** du processus.
  ou
  *Toutes les sorties du SLC doivent êtres affectées dans **toutes** les branches des instructions séquentielles.*
- **Interdit** d'utiliser, dans le process, les sorties générée par celui-ci ! (Si nécessaire utiliser une variable dans le process)

Description algorithmique DMUX …

- Description du comportement du démultiplexeur :
  Si Enable est actif alors
  la sortie correspondante à Sel est activée
  toutes les autre sorties sont inactives
- autre variante :
  Toutes les sorties sont inactives
  Si Enable est actif alors
  la sortie correspondante à Sel est activée

Description algorithmique DMUX …

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity DMUX1_4 is
  port(Sel_i : in  Std_Logic_Vector(1 downto 0);
       En_i  : in  Std_Logic;
       Y_o   : out Std_Logic_Vector(3 downto 0)
      );
end DMUX1_4;
```

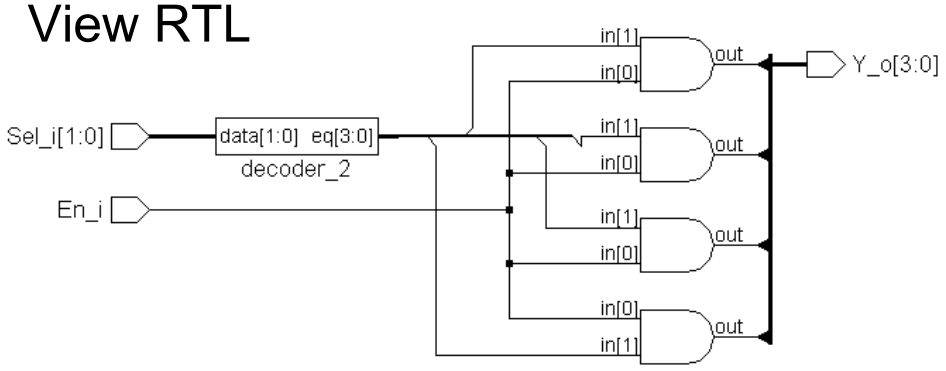
ATENTION : description non synthétisable

… description algorithmique DMUX

```
architecture Comport of DMUX1_4 is
begin

  process(Sel_i ,En_i)
  begin
    Y_o <= (others => '0'); -- Valeur par défaut

    Y_o(To_Integer(Unsigned(Sel_i))) <= En_i;

  end process;
end Comport;
```

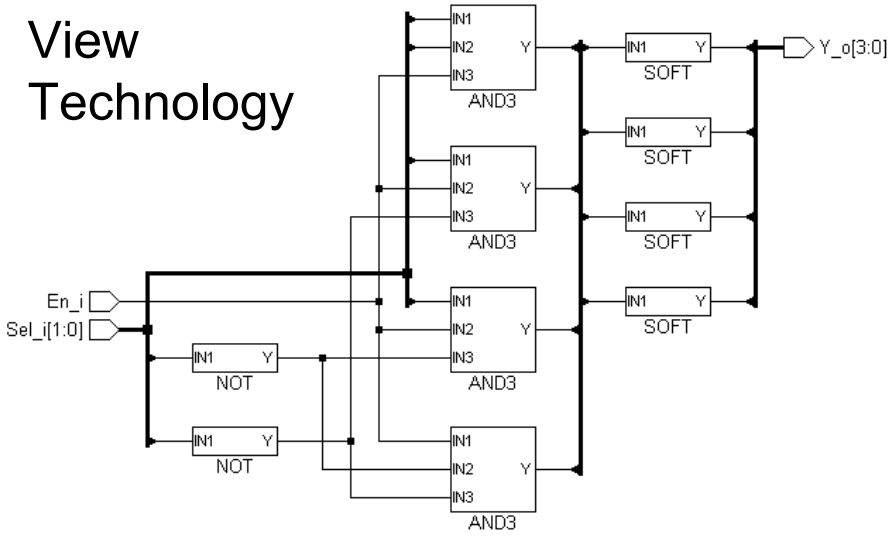
Synthèse DMUX avec Leonardo …

View RTL



… synthèse DMUX avec Leonardo

View Technology



Synthèse DMUX avec Max+plus II (ALTERA)

Equations fournies dans fichier *.rpt

```
Y_o0    = LCELL( _EQ001 $  GND);
 _EQ001 =  En_i & !Sel_i0 & !Sel_i1   ← terme correct
         #  Sel_i0 &  Y_o0
         #  Sel_i1 &  Y_o0                 termes
         #  En_i & !Sel_i1 &  Y_o0         inutiles !
         #  En_i & !Sel_i0 &  Y_o0;

Y_o1    = LCELL( _EQ002 $  GND);
 _EQ002 =  En_i &  Sel_i0 & !Sel_i1   ← terme correct
         #  Sel_i1 &  Y_o1
         # !Sel_i0 &  Y_o1                termes
         #  En_i &  Sel_i0 &  Y_o1         inutiles !
         #  En_i & !Sel_i1 &  Y_o1;
```


Analyse description algorithmique DMUX

```
process(Sel_i ,En_i)
begin
  Y_o <= (others => '0'); -- Valeur par défaut
  Y_o(To_Integer(Unsigned(Sel_i))) <= En_i;
end process;
```

Indice variable

donc :

description NON SYNTHETISABLE !

Variante description algorithmique DMUX

```
architecture Comport of DMUX1_4 is
begin

  process(Sel_i ,En_i)
  begin
    Y_o <= (others => '0'); -- Valeur par défaut
    for I in 0 to Y_o'length-1 loop
      --la boucle for permet d'expliciter les
      --comparaisons (test =)
      if I = To_Integer(Unsigned(Sel_i)) then
        Y_o(I) <= En_i;
      end if;
    end loop;
  end process;

end Comport;
```

pas d'indice variable

Synthèse avec Max+plus II (ALTERA)

Equations fournies dans fichier *.rpt

```
Y_o0    = LCELL( _EQ001 $  GND);
 _EQ001 =  En_i & !Sel_i0 & !Sel_i1;

Y_o1    = LCELL( _EQ002 $  GND);
 _EQ002 =  En_i &  Sel_i0 & !Sel_i1;

Y_o2    = LCELL( _EQ003 $  GND);
 _EQ003 =  En_i & !Sel_i0 &  Sel_i1;

Y_o3    = LCELL( _EQ004 $  GND);
 _EQ004 =  En_i &  Sel_i0 &  Sel_i1;
```

Equations correctes !

Description d'éléments mémoires

- Pas de déclaration explicite en VHDL
- Description indique au synthétiseur :
  signal correspond à un élément mémoire

La description doit :

Diriger correctement le synthétiseur

afin d'obtenir l'élément mémoire désiré

Avertissement

Avec le langage VHDL :

- Il est possible d'obtenir un élément mémoire
  lorsque vous ne le voulez pas !

INVERSEMENT

- Il est possible de ne pas avoir d'élément mémoire
  lorsque vous le voulez !

Comportement par défaut du VHDL

Lors de l'utilisation d'instructions séquentielles :

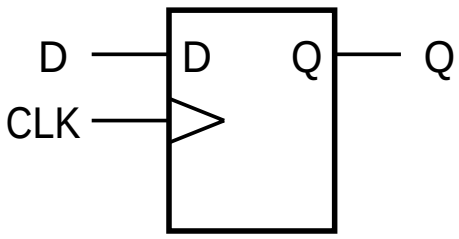
Cas non traité $\Rightarrow$ VHDL maintien l'état du signal

- Cette fonctionnalité servira de base pour écrire les éléments mémoires
- Cette fonctionnalité sera aussi un piège par l'obtention de *latchs* non désirés

page volontairement laissée vide

Description d 'un flip-flop

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Flip_Flop is
   port (D       : in Std_Logic;
         Clock   : in Std_Logic;
         Q       : out Std_Logic);
end Flip_Flop;

architecture Comport of Flip_Flop is
begin
  process(Clock)
  begin
    if Rising_Edge(Clock) then
       Q <= D;
    end if;
  end process;
end Comport;
```

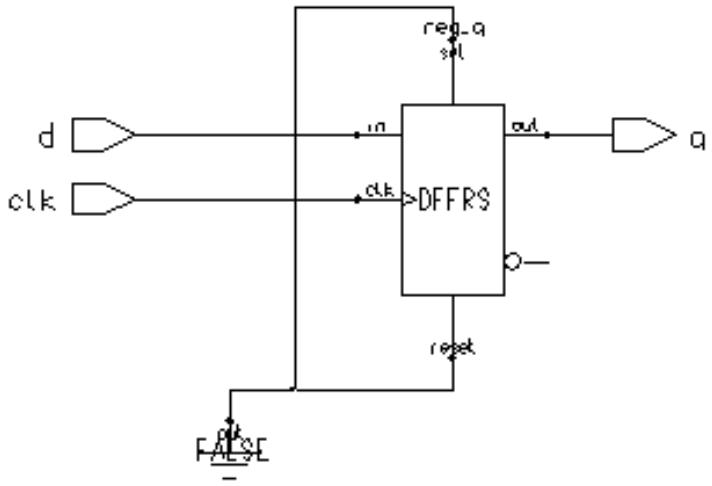


Synthèse du flip-flop

- Cette description se synthétise parfaitement sur tous les synthétiseurs actuels
- Deux informations confirment la notion de flip-flop :
  - ✓le processus ne réagit que sur l'horloge CLK
  - ✓la condition du test spécifie clairement une action dynamique (Rising_Edge)

Synthèse DFF avec Leonardo

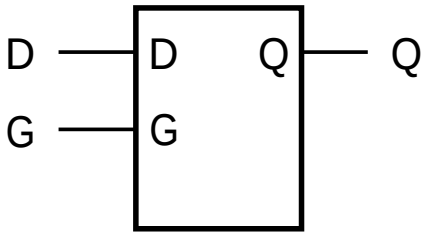
- Vue RTL



page volontairement laissée vide

Description d'un latch

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Latch is
   port (D      : in Std_Logic;
         G      : in Std_Logic;
         Q      : out Std_Logic);
end Latch;

architecture Comport of Latch is
begin
  process(G, D)
  begin
    if G = '1' then
      Q <= D;
    end if;
  end process;
end Comport;
```

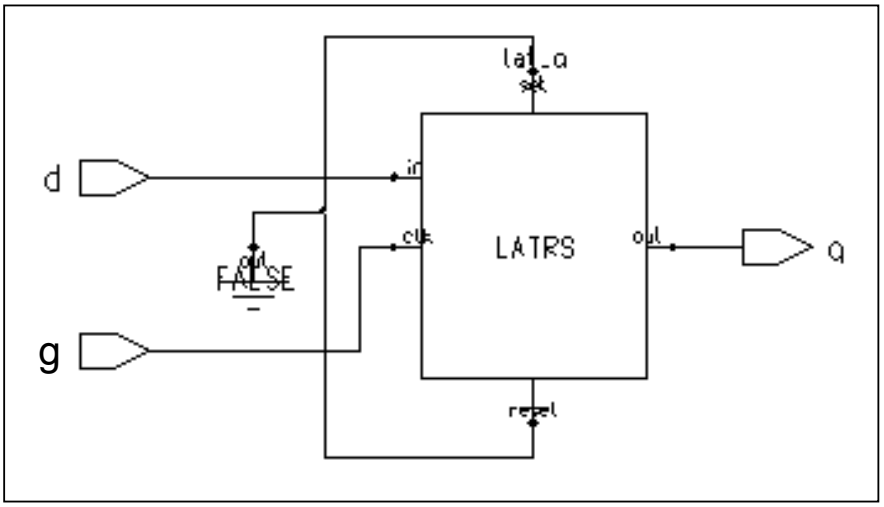


Synthèse du latch

- Cette description se synthétise parfaitement sur tous les synthétiseurs actuels
- Deux informations confirment la notion de latch :
  - ✓le processus réagit sur les deux signaux G et D
  - ✓la condition du test (=) spécifie clairement une action sur un niveau

Synthèse latch avec Leonardo

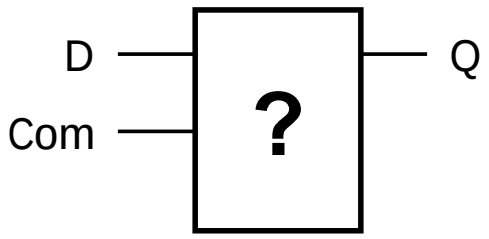
- Vue RTL



page volontairement laissée vide

Type de bascule ?

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Bascule is
   port (D   : in Std_Logic;
         Com : in Std_Logic;
         Q   : out Std_Logic);
end Bascule;

architecture Comport of Bascule is
begin
  process(Com)
  begin
    if Com = '1' then
       Q <= D;
    end if;
  end process;
end Comport;
```



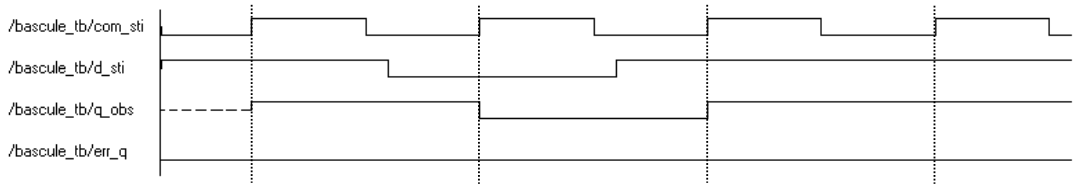
Synthèse de cette bascule

- Cette description est ambiguë.
- Deux informations se contredissent :
  - ✓le processus ne réagit que sur Com
    => action dynamique => flip-flop
  - ✓la condition du test spécifie un niveau (=)
    => action sur un niveau => latch

Type de bascule ?

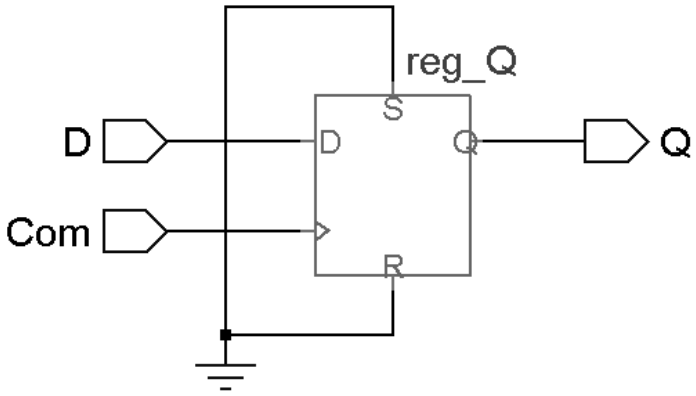
- Compréhension par le simulateur, inter- prétation stricte de la description VHDL:
  - ✓une seule réponse : .....
- Compréhension par le synthétiseur qui va traduire la description VHDL en logique:
  - ✓priorité à la liste de sensibilité : .....
  - ✓priorité à la condition de test : .....

Résultat de la simulation (ModelSim)



Le comportement correspond bien a celui d'un flip-flop D

Synthèse avec Léonardo

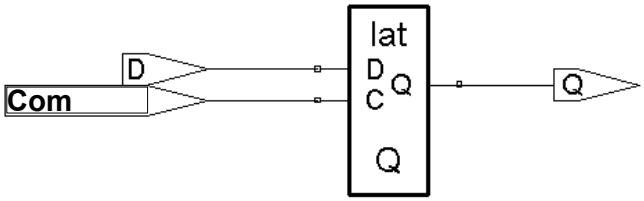


Le résultat de la synthèse est un **flip-flop**

Synthèse avec Synplify : 3 *WARNING*

```
Synthesizing work.bascule.comport
@W:"f:\en_cours\synplify\bascule.vhd":30:2:30:8
  Incomplete sensitivity list - assuming
  completeness
@W:"f:\en_cours\synplify\bascule.vhd":33:10:33:10
  Referenced variable d is not in sensitivity list
Post processing for work.bascule.comport
@W:"f:\en_cours\synplify\bascule.vhd":32:4:32:5
  Latch generated from process for signal q,
  probably caused by a missing assignment in an if
  or case stmt
@END
```

Synthèse avec Synplify



Le résultat de la synthèse est un LATCH

Conclusion sur cette description

- Le comportement en simulation peut-être différent du matériel obtenu
- Matériel obtenu dépend du synthétiseur, résultat pas identique

Cette description est **NON SYNTHETISABLE**

Interdit par la norme IEEE 1076.6-1999

Etat initial d'une bascule

- A l'instant 0 ns :
  Tous les signaux sont à l'état 'U'
- Une initialisation est indispensable
- Comment forcer l'état initial d'une bascule en VHDL ?

Initialisation d'un flip-flop

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Flip_Flop is
   port (D       : in Std_Logic;
         Clock   : in Std_Logic;
         Reset   : in Std_Logic;
         Q       : out Std_Logic);
end Flip_Flop;

architecture Comport of Flip_Flop is
begin
  process(Clock, Reset)
  begin
    if Reset = '1' then
      Q <= '0';
    elsif Rising_Edge(Clock) then
      Q <= D;
    end if;
  end process;
end Comport;
```

Flip-flop avec actions asynchrones

La norme IEEE-1076.6 "Standard for VHDL Register Transfer Level (RTL) Synthesis" définit le sous-ensemble utilisable en sysnthèse.

- Voici la syntaxe de l'instruction *process* pour un flip-flop avec action asynchrone

```
process(Clock, Action_Asynchrone)
begin
  if Action_Asynchrone = '1' then
    Q <= Etat_Initial;
  elsif Rising_Edge(Clock) then
    Q <= D;
  end if;
end process;
```


Remarque synthétiseur PRECISION

- Synthèse conforme LRM (LRM compliance checks)
  - ✓ Menu "Tools" → "Set Options …"
    - →Dans le sous menu "Input" (actif par défaut)
    - →Sélectionner l'option : 2004c Compile Mode
- Meilleure visibilité des connections dans la vue RTL
  - ✓ Menu "Tools" → "Set Options …"
    - →Sélectionner sous menu "Schématique Viewer"
    - →Désélectionner les options
      - Show Bundled Instances
      - Show Net Buses
    - →Cliquer sur "Appliquer" puis "OK"

Exercices :

- Compte réseau à utiliser :
  - ✓ utilisateur : REDS_User
  - ✓ mot de passe : REDS
  - ✓ domaine : einet
- Directory de travail :
  - ✓ D:\VTF\Exos_VHDL\EleMem

Exe : description latch, flip-flops (DFF, DFFE, TFF)

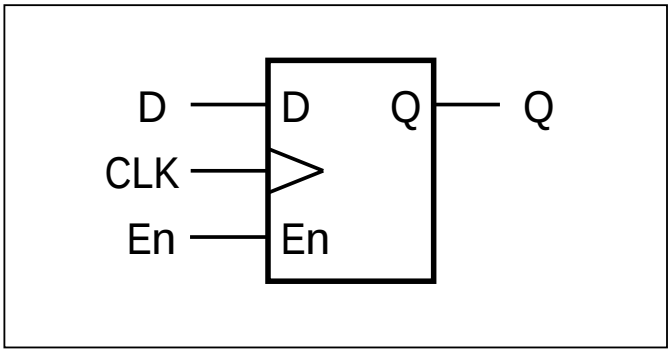
- Logiciels utilisés :
  - ✓ synthétiseur : Precision
  - ✓ simulateur : QuestaSim
- Exercices : Latch_D, DFF, DFFE, Flip-flop T; Bascule RS

Démarche :

- ✓ compléter les descriptions puis les synthétiser et vérifier le matériel obtenu (vues *RTL* et *Technology*)
- ✓ simuler les descriptions pour vérifier le fonctionnement

Description DFFE

- Description d'un flip-flop D avec maintien (Enable)
- Symbole DFFE :



Description DFFE, proposition I

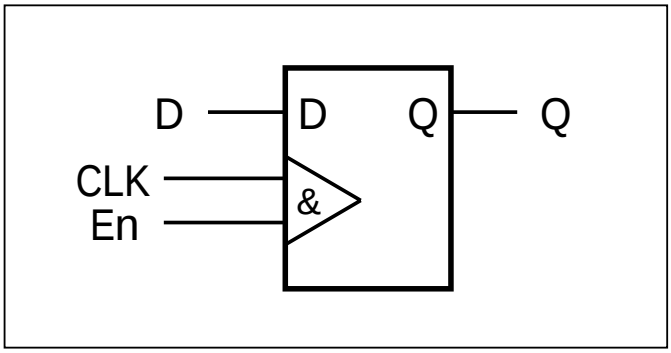
```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  DFFE_I is
   port (D,En    : in Std_Logic;
         Clock   : in Std_Logic;
         Q       : out Std_Logic);
end DFFE_I;

architecture Comport of DFFE_I  is
begin
  process(Clock)
  begin
    if Rising_Edge(Clock) and En = '1' then
       Q <= D;
    end if;
  end process;
end Comport;
```

Description DFFE, proposition II

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  DFFE_II is
   port (D, En   : in Std_Logic;
         Clock   : in Std_Logic;
         Q       : out Std_Logic);
end DFFE_II;

architecture Comport of DFFE_II is
begin
  process(Clock)
  begin
    if Rising_Edge(Clock) then
      if En = '1' then
        Q <= D;
      end if;
    end if;
  end process;
end Comport;
```


Description DFFE, proposition I

- Représentation matériel :

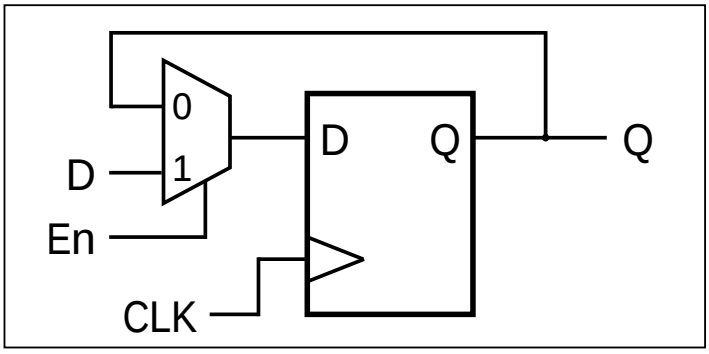


Ce type de composant n'existe pas

Cette description est non synthétisable !

Description DFFE, proposition II

- Représentation matériel :



Les composants utilisés existent

Cette description est synthétisable

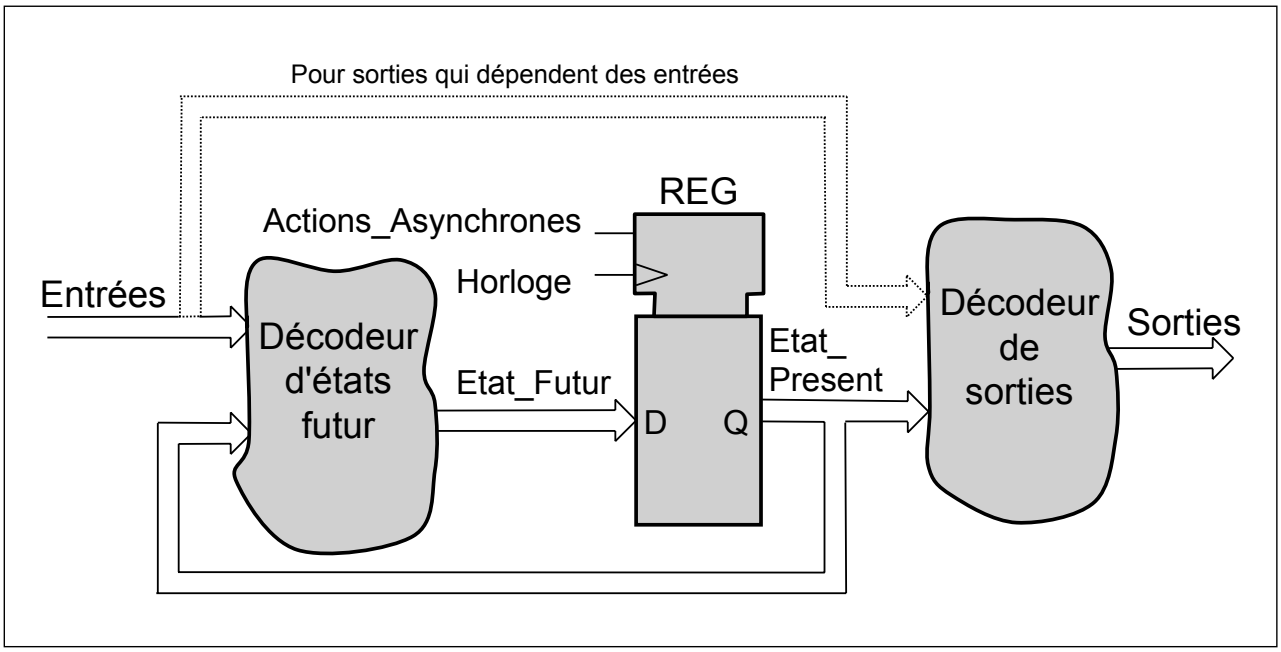
page volontairement laissée vide

Description systèmes séquentiels

Principe :

- Suivre la décomposition logique d'un système séquentiel
- Décomposition visible dans la description
- Avantages :
  - ✓ description fiable (synthèse)
  - ✓ description lisible
  - ✓ modifie seulement la partie combinatoire (décodeur d'état futur)
- Inconvénients :
  - ✓ description un peu plus longue (plus de lignes)

Décomposition d'un système séquentiel



Description d'un système séquentiel

Description décomposée en trois parties :

- Décodeur d'états futur, combinatoire :
  - ✓ description concurrente ou séquentielles (*process*)
- Elément mémoire, séquentiel :
  - ✓ description avec un processus
  - ✓ toujours identique
- Décodeur de sortie, combinatoire :
  - ✓ description concurrente ou séquentielles (*process*)

Processus de mémorisation

- La liste de sensibilité contient l'horloge et les actions asynchrones.
- Ce processus ne contient **que** la mémorisation.
- Description identique pour tout système séquentiel

Syntaxe conforme à la norme IEEE 1076.6-1999

Description lisible, fiable et portable

Syntaxe description système séquentiel

```
architecture Comport of Sys_Seq is
begin
  -- Description concurrente du décodeur d'état futur
     . . .

   Mem: process (Horloge, Entree_Action_Asynch)
  begin
    if (Entree_Action_Asynch = '1') then
      Etat_Present <= Etat_Initial;
    elsif Rising_Edge(Horloge) then
      Etat_Present <= Etat_Futur;
    end if;
  end process;
  -- Description concurrente du décodeur de sorties
     . . .
end Comport;
```


Opérations arithmétiques + et -

Deux possibilités :

- Utiliser le paquetage IEEE (Numeric_Std) qui définit les opérations arithmétiques pour le type Std_Logic.
  - ✓ simple d'emploi.
  - ✓ laisse le choix au synthétiseur.
  - ✓ permet l'utilisation de macros fonctions.
- Décomposer en opérations 1 bit (équations).
  - ✓ nécessaire pour s'adapter à la technologie.

Description d'un compteur …

- Compteur 4 bits
- Reset asynchrone actif haut
- Actions synchrones :

Charge	Compte	Description	
'1'	-	charge	Cpt = Valeur
'0'	'1'	compte	Cpt = Cpt + 1
'0'	'0'	maintien	Cpt = Cpt

Ordre de priorité : charge, compte, maintien

… description d'un compteur …

Traduction "textuelle" du fonctionnement synchrones

Si Charge actif alors Cpt = Valeur
sinon Compte actif alors Cpt = Cpt + 1
sinon Cpt = Cpt

description aisée avec une instruction when…else

```
Cpt_Futur <=
  Valeur          when Charge = '1' else
  Cpt_Present +1  when Compte = '1' else
  Cpt_Present;
```

… description compteur …

```
library IEEE;
use IEEE.Std_Logic_1164.all;
--definit les operations arithmetiques
use IEEE.Numeric_Std.all;

entity Compteur is
   port(Charge, Compte : in Std_Logic;
        Horloge, Reset : in Std_Logic;
        Valeur : in  Std_Logic_vector(3 downto 0);
        Cpt    : out Std_Logic_Vector(3 downto 0)
        );
end Compteur;
```


… description compteur …

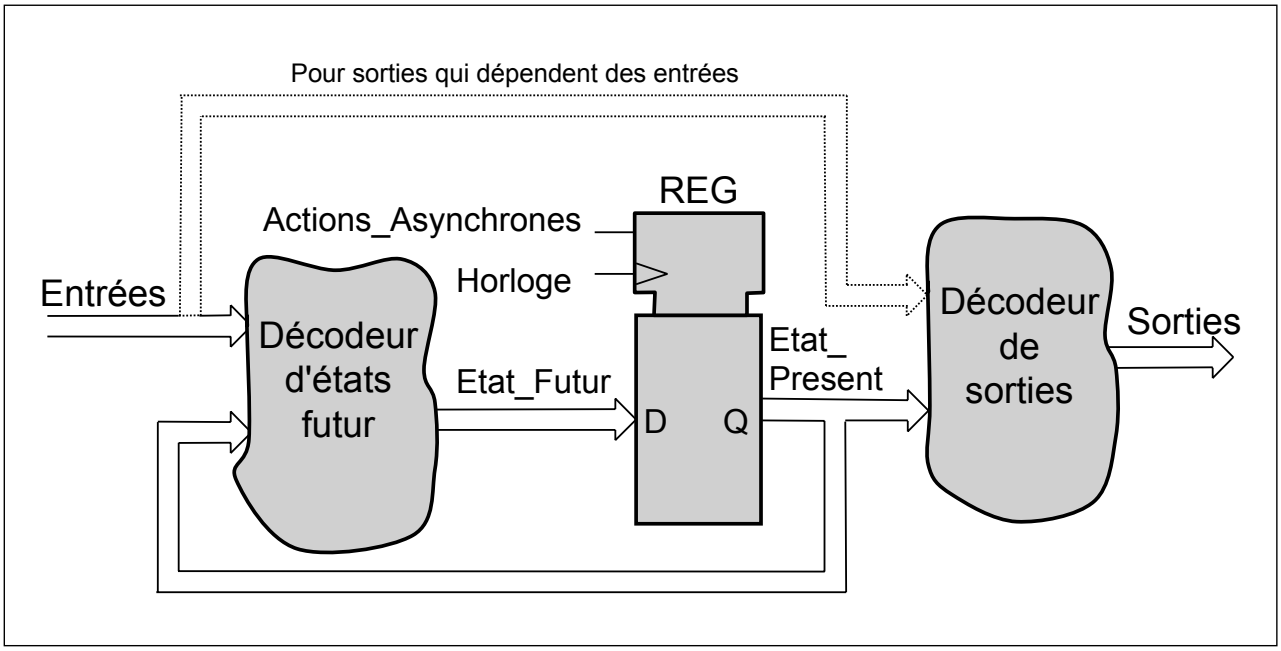
- Description concurrente du décodeur d'états futur

```
architecture Comport of Compteur is
 signal Cpt_futur, Cpt_Present : unsigned(3 downto 0);
begin
  --Description concurrente du décodeur d'états futur
  --ordre de priorite : charge, compte, maintien
  Cpt_Futur <=
        Unsigned(Valeur) when (Charge = '1') else
        Cpt_Present +1   when (Compte = '1') else
        Cpt_Present; -- maintien
```

… description compteur

```
  Mem: process (Horloge, Reset)
  begin
    if (Reset = '1') then
      Cpt_Present <= (others => '0');
    elsif Rising_Edge(Horloge) then
      Cpt_Present <= Cpt_Futur;
    end if;
  end process;

  --Mise a jour de l'etat du compteur
  Cpt <= Std_Logic_Vector(Cpt_present);

end Comport;
```


Décomposition d'une machine d'état



Description d'une machine d'état

Description décomposée en trois parties :

- Décodeur d'états futur, combinatoire :
  - ✓ description à l'aide d'un *process* et de l'instruction *case*
- Elément mémoire, séquentiel :
  - ✓ description avec un processus
  - ✓ toujours identique
- Décodeur de sortie, combinatoire :
  - ✓ cas simple: description avec des instructions concurrentes, sinon
  - ✓ description à l'aide d'un *process* et de l'instruction *case* (parfois combiné avec le décodeur d'état futur)

Description d'une machine d'état ...

```
architecture M_Etat of Exemple is
-- Machine d'etat avec n bits d'etat
  signal Etat_Present, Etat_Futur :
                              Std_Logic_Vector(n downto 0);

--Les constantes permettent de fixer le codage
--Si modification : seul les constantes sont a corriger.
  constant Etat_0 : Std_Logic_Vector(n downto 0) := "0..00";
  constant Etat_1 : Std_Logic_Vector(n downto 0) := "0..01";
     "
begin

  Fut: process (Toutes_Les_Entrees, Etat_Present)
  begin
    Etat_Futur <= Etat_0;  --valeur par defaut
    Sortie_A <= '0'; Sortie_B <= '0'; --valeur par defaut
```

… description machine d'état …

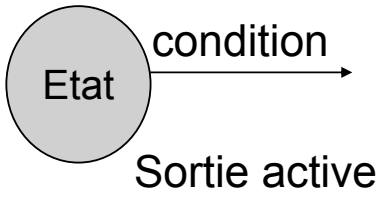
```
case Etat_Present is
  when Etat_0 =>
    Sortie_A <= '1'; --sortie de type Moore
    if (condition) then
      Etat_Futur <= Etat_0;
      Sortie_B <= '1'; --sortie de type Mealy
    else
      Etat_Futur <= Etat_1;
      Sortie_B <= '0'; --sortie de type Mealy
    end if;
  when Etat_1 =>  ...

    "

  when others =>
    Sortie_A <= '0'; Sortie_B <= '0';
    Etat_Futur <= Etat_0;
end case;
```


… description machine d'état …

```
  end process;

  Mem: process (Horloge, RAZ)
  begin
     if (RAZ = '1') then
        Etat_Present <= Etat_0;
     elsif Rising_Edge(Horloge) then
        Etat_Present <= Etat_Futur;
       end if;
  end process;

end Comport;
```

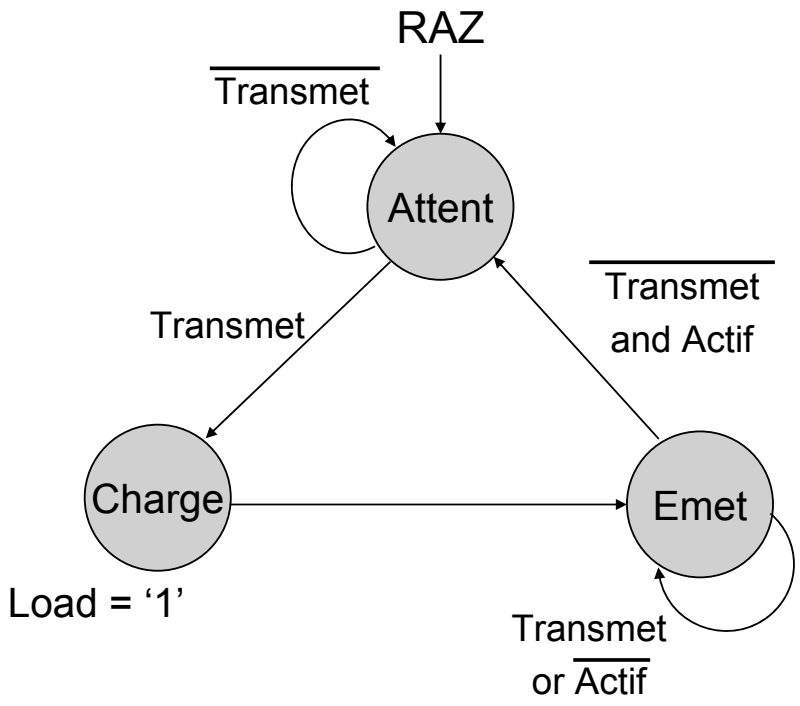
Exemple de machine d'état …

Convention graphe



Entrées:
- Transmet, Actif
- Load

Sortie:
- Load



… exemple de machine d'état …

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity UC_Emetteur is
  port ( Transmet, Actif : in  Std_Logic;
         Clock, Reset :    in  Std_Logic;
         Load :            out Std_Logic);
end UC_Emetteur;

architecture M_Etat of UC_Emetteur is

  signal Etat_Pres, Etat_Fut : Std_Logic_Vector(1 downto 0);

--Les constantes permettent de fixer le codage
--Si modification : seul les constantes sont a corriger.
  constant Attent : Std_Logic_Vector(1 downto 0) := "00";
  constant Charge : Std_Logic_Vector(1 downto 0) := "01";
  constant Emet   : Std_Logic_Vector(1 downto 0) := "10";
begin
```

```
--process combinatoire, calcul etat futur
Fut:process (Transmet, Actif, Etat_Present)
begin
  Etat_Futur <= Attent; -- valeur par defaut
  case Etat_Present is
    when Attent =>
    if Transmet = '1' then
      Etat_Futur <= Charge;
    else
      Etat_Futur <= Attent;
    end if;
    when Charge =>
    Etat_Futur <= Emet;
    when Emet =>
    if Transmet = '0' and Actif = '0' then
      Etat_Futur <= Attent;
    else
      Etat_Futur <= Emet;
    end if;
    when others =>
    Etat_Futur <= Attent;
  end case;
end process;
```


… exemple de machine d'état

```
--processus de memorisation
Mem:process (Clock, Reset)
begin
  if Reset = '1' then
    -- reset asynchrone prioritaire
    Etat_Present <= Attent;
  elsif Rising_Edge(Clock) then
    Etat_Present <= Etat_Futur;
  end if;
end process;

-- equations combinatoire de sortie
Load <= '1' when Etat_Present = Charge else '0';

end M_Etat;
```

Remarque synthétiseur PRECISION

- Meilleure visibilité des connections dans la vue RTL
  - ✓ Menu "Tools" → "Set Options …"
    - →Sélectionner sous menu "Schématique Viewer"
    - →Désélectionner les options
      - Show Bundled Instances
      - Show Net Buses
    - →Cliquer sur "Appliquer" puis "OK"

FIN présentation VHDL !

Questions