Introduction à la Synthèse logique

V.H.D.L.

Lycée Julliot de la Morandière – GRANVILLE – 2001
Philippe LECARDONNEL & Philippe LETENNEUR

SOMMAIRE

I) Introduction.

Pourquoi avoir créé un langage de description de structures électroniques (**H.D.L. Hardware Description language**) **VHDL** ?

L'abréviation **VHDL** signifie **VHSIC** **H**ardware **D**escription **L**anguage (**VHSIC** : **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit). Ce langage a été écrit dans les années 70 pour réaliser la simulation de circuits électroniques. On l'a ensuite étendu en lui rajoutant des extensions pour permettre la conception (synthèse) de circuits logiques programmables (**P.L.D. P**rogrammable **L**ogic **D**evice).

Auparavant pour décrire le fonctionnement d'un circuit électronique programmable les techniciens et les ingénieurs utilisaient des langages de bas niveau (**ABEL, PALASM, ORCAD/PLD**,..) ou plus simplement un outil de saisie de schémas.

Actuellement la densité de fonctions logiques (portes et bascules) intégrée dans les **PLDs** est telle (plusieurs milliers de portes voire millions de portes) qu'il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui.

Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir **VHDL** et **VERILOG**. Ces deux langages font abstraction des contraintes technologies des circuits **PLDs**.

Ils permettent au code écrit d'être portable, c'est à dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit.

Il faut avoir à l'esprit que ces langages dits de haut niveau permettent de matérialiser les structures électroniques d'un circuit.

En effet les instructions écrites dans ces langages se traduisent par une configuration logique de portes et de bascules qui est intégrée à l'intérieur des circuits **PLDs**. C'est pour cela que je préfère parler de description **VHDL** ou **VERILOG** que de langage.

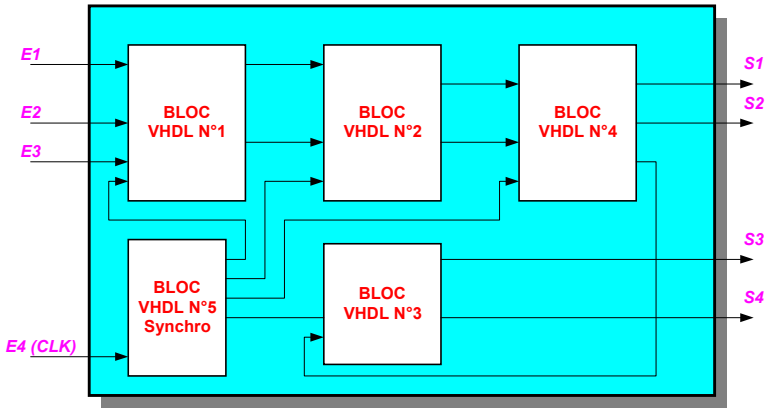
Dans ce polycopié, je m'intéresserai seulement à **VHDL** et aux fonctionnalités de base de celui-ci lors des phases de conception ou synthèse (c'est à dire à la conception de **PLDs**).

II) Relation entre une description VHDL et les circuits logiques programmables.

Décrire le fonctionnement d'un circuit logique programmable c'est bien, mais comment faire le lien avec la structure de celui-ci ?

L'implantation d'une ou de plusieurs descriptions **VHDL** dans un PLD va dépendre de l'affectation que l'on fera des broches d'entrées / sorties et des structures de base du circuit logique programmable.

II.1) Schéma fonctionnel d'implantation de descriptions VHDL dans un circuit logique programmable.



Ci-dessus le schéma représente un exemple d'implantation de descriptions **VHDL** ou de blocs fonctionnels implantés dans un PLD. Lors de la phase de synthèse chaque bloc sera matérialisé par des portes et/ou des bascules. La phase suivante sera d'implanter les portes et les bascules à l'intérieur du circuit logique.

Cette tâche sera réalisée par le logiciel placement/routage (« **Fitter** »), au cours de laquelle les entrées et sorties seront affectées à des numéros de broches.

On peut remarquer sur le schéma la fonction particulière du bloc **VHDL N°5.** En effet dans la description fonctionnelle d'un PLD on a souvent besoin d'une fonction qui sert à cadencer le fonctionnement de l'ensemble, celle-ci est très souvent réalisée par une machine d'états synchronisée par une horloge.

II.2) L'affectation des broches d'entrées sorties.

Elle peut se faire de plusieurs manières :

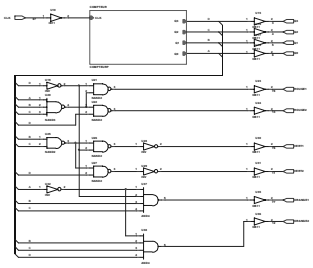
1) L'affectation automatique.

On laisse le synthétiseur et le ***Fitter*** du fabriquant (« ***Fondeur*** » : ***Xilinx, Lattice, Altéra, Cypress...***») du circuit implanter la structure correspondant à la description **VHDL**. Les numéros de broches seront choisis de façon automatique.

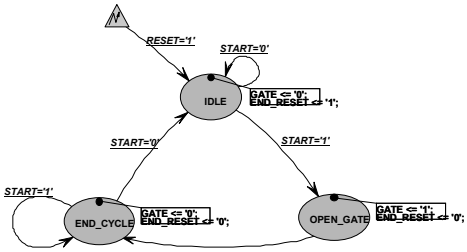
2) L'affectation manuelle.

On définit les numéros de broches dans la description **VHDL** ou sur un schéma bloc définissant les liaisons entre les différents blocs **VHDL** ou dans un fichier texte propre au fondeur. Les numéros de broches seront affectés suivant les consignes données. (voir **page N° 8**).

II.3) Organisation fonctionnelle de développement d'un PLD.



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

entity CNT_4B is
    port (
        CLK: in STD_LOGIC;
        RESET: in STD_LOGIC;
        ENABLE: in STD_LOGIC;

        FULL: out STD_LOGIC;
        Q: out STD_LOGIC_VECTOR
(3 downto 0)
```



Entrée schématique

Entrée syntaxique langages: - VHDL

Diagrammes d'états

VHDL *VHDL* *VHDL*

Vérification des erreurs de schéma et de syntaxe

STIMULI ou TESTBENCH

Simulation comportementale

VHDL

SYNTHETISEUR

Il convertit en fonction du ciruit cible (CPLD ou FPGA: suivant le modèle choisit) le projet en portes logiques et bascules de bases.
De plus on peut lui spécifier des contraintes de technologies, par exemple la vitesse de fonctionnement

VITAL, VHDL

STIMULI ou TESTBENCH

Simulation après synthèse

SYNTHETISEURS CONNUS:

-FPGA Expres (Synosys).
-EXEMPLAR (Exemplar).
-SYNPLICITY (Synplicity).

EDIF, XNF

Outils de placement et routage interne au circuit. Cet outil est propre à chaque fabriquant

Optimisation Placement / routage dans le circuit choisi

VITAL, VHDL, SDF

STIMULI ou TESTBENCH

Simulation temporelle

JEDEC

La programmation du circuit peut se faire soit:
- En utilisant un programmateur.
- En téléchargement le code JEDEC directement sur l'objet technique (JTAG ou ISP)

CIRCUIT FPGA CPLD

III) Structure d'une description VHDL simple.

Une description **VHDL** est composée de 2 parties **indissociables** à savoir :

- ***L'entité*** (***ENTITY***), elle définit les entrées et sorties.
- ***L'architecture*** (**ARCHITECTURE**), elle contient les instructions **VHDL** permettant de réaliser le fonctionnement attendu.

Exemple : Un décodeur 1 parmi 4.

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

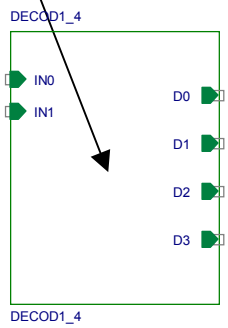
Déclaration des bibliothèques

Commentaires, en VHDL ils commencent par - -

```
-- décodeur
-- Un parmi quatre
```

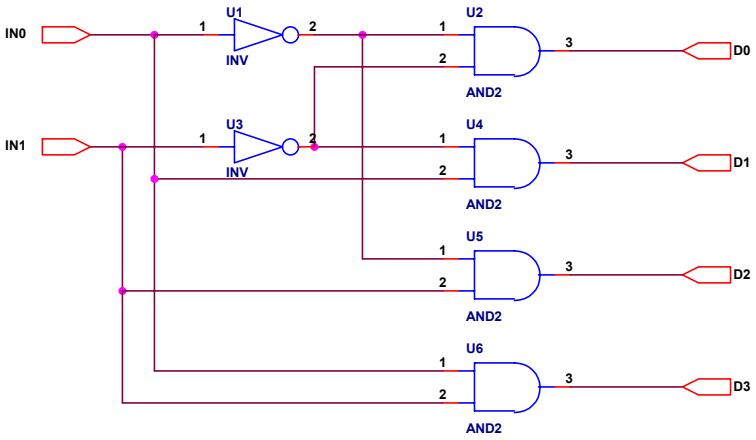
Déclaration de l'entité du décodeur
Correspondance schématique

```
entity DECOD1_4 is
      port(IN0, IN1: in std_logic;
           D0, D1, D2, D3: out std_logic);
end DECOD1_4;
```



Déclaration de l'architecture du décodeur
Correspondance schématique

```
architecture DESCRIPTION of DECOD1_4 is
begin
      D0 <= (not(IN1) and not(IN0));
      D1 <= (not(IN1) and IN0);
      D2 <= (IN1 and not(IN0));
      D3 <= (IN1 and IN0);
end DESCRIPTION;
```



III.1) Déclaration des bibliothèques.

Toute description **VHDL** utilisée pour la synthèse a besoin de bibliothèques. L'**IEEE** (**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers) les a normalisées et plus particulièrement la bibliothèque **IEEE1164**. Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,...

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
-- cette dernière bibliothèque est souvent utilisée pour l'écriture de compteurs
```

La directive `Use` permet de sélectionner les bibliothèques à utiliser.

III.2) Déclaration de l'entité et des entrées / sorties (I/O).

Elle permet de définir le NOM de la description **VHDL** ainsi que les entrées et sorties utilisées, l'instruction qui les définit c'est `port :`

Syntaxe:

```
entity NOM_DE_L_ENTITE is

     port ( Description des signaux d'entrées /sorties …);

end NOM_DE_L_ENTITE;
```

Exemple :

```
entity SEQUENCEMENT is
port (
     CLOCK      : in std_logic;
     RESET      : in std_logic;
     Q          : out std_logic_vector(1 downto 0)
     );
end SEQUENCEMENT;
```

Remarque : Après la dernière définition de signal de l'instruction `port` il ne faut jamais mettre de point virgule.

L'instruction ***port*** .

```
Syntaxe:   NOM_DU_SIGNAL   : sens     type;

Exemple:   CLOCK: in       std_logic;
           BUS   : out     std_logic_vector (7 downto 0);
```

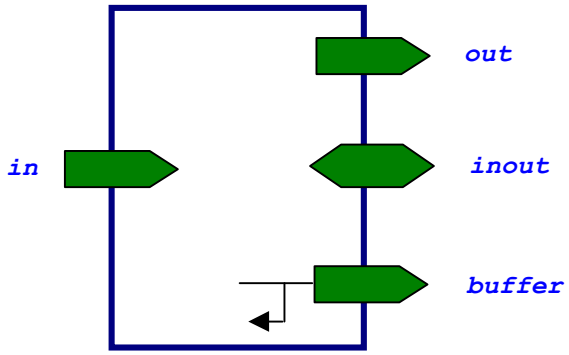
On doit définir pour chaque signal : le ***NOM DU SIGNAL***, le ***sens*** et le ***type***.

III.2.1) Le *NOM_DU_SIGNAL*.

Il est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code. **VHDL** n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.

III.2.2) Le *SENS* du signal.

- ***in*** : pour un signal en entrée.
- ***out*** : pour un signal en sortie.
- ***inout*** : pour un signal en entrée sortie
- ***buffer*** : pour un signal en sortie mais utilisé comme entrée dans la description.



III.2.3) Le *TYPE*.

Le ***TYPE*** utilisé pour les signaux d'entrées / sorties est :

- le ***std_logic*** pour un signal.
- le ***std_logic_vector*** pour un bus composé de plusieurs signaux.

Par exemple un bus bidirectionnel de 5 bits s'écrira :

```
LATCH : inout std_logic_vector (4 downto 0) ;
```

Où ***LATCH(4)*** correspond au **MSB** et ***LATCH(0)*** correspond au **LSB**.

Les valeurs que peuvent prendre un signal de type ***std_logic*** sont :

- ***'0' ou 'L'*** : pour un niveau bas.
- ***'1' ou 'H'*** : pour un niveau haut.
- ***'Z'*** : pour état haute impédance.
- ***'-'*** : Quelconque, c'est à dire n'importe quelle valeur.

III.2.4) Affectation des numéros de broches en utilisant des attributs supplémentaires:

La définition des numéros de broches par la définition d'attributs supplémentaires dépend du logiciel utilisé pour le développement.

Exemples :

Avec Warp de Cypress :

```
entity AFFICHAGE is port(
CLK, RESET, DATA: in std_logic;
S:                buffer std_logic_vector(9 downto 0));

      -- C'est la ligne ci-dessous qui définit les numéros de broches
      attribute pin_numbers of AFFICHAGE:entity is "S(9):23 S(8):22 S(7):21
      S(6):20 S(5):19 S(4):18 S(3):17 S(2):16 S(1):15 S(0):14";

end AFFICHAGE;
```

Avec Express d'Orcad :

```
entity DECODAGE is
port (
      ADRESSE           : in std_logic_vector (15 downto 10);
      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);

      -- Ce sont les lignes ci-dessous qui définissent les numéros de --
      broches
      -- Assignation manuelle des broches PAL16V8
      attribute PLDPIN:string;
      attribute PLDPIN of ADRESSE:    signal is "7,6,5,4,3,2";
      -- Attention : LSB en premier !
      attribute PLDPIN of RAM0:   signal is "19";
      attribute PLDPIN of RAM1:   signal is "18";
      attribute PLDPIN of RAM2:   signal is "17";
      attribute PLDPIN of RAM3:   signal is "16";
      attribute PLDPIN of ROM:    signal is "15";
      attribute PLDPIN of INTER1: signal is "14";
      attribute PLDPIN of INTER2: signal is "13";
      attribute PLDPIN of INTER3: signal is "12";

end DECODAGE;
```

III.2.5) Exemples de description d'entités:

```
entity COUNT is
     port(CLK, RST:  in std_logic; CNT:  inout std_logic_vector(2 downto 0));
end COUNT;


entity MAGNITUDE is
      port( A, B: in std_logic_vector(7 downto 0);
      AGRB: buffer std_logic));
end MAGNITUDE;


entity 7SEG is
      port (
            A     : in std_logic;
            B     : in std_logic;
            C     : in std_logic;
            D     : in std_logic;
            SA    : out std_logic;
            SB    : out std_logic;
            SC    : out std_logic;
            SD    : out std_logic;
            SE    : out std_logic;
            SF    : out std_logic;
            SG    : out std_logic );
end 7SEG;


entity IMAGE3 is
      port (CLK : in std_logic; S,I : out std_logic);
end IMAGE3;


entity COMP4BIT is
      port (A,B :in std_logic_vector(3 downto 0);
      PLUS,MOINS,EGAL :out std_logic);
end COMP4BIT;


entity COMPT_4 is
      port (H,R :in std_logic;
      Q :out std_logic_vector(3 downto 0));
end COMPT_4;


entity DECAL_D is
      port (H,R,IN_SERIE :in std_logic;
      OUT_SERIE :out std_logic);
end DECAL_D;
```

III.3) Déclaration de l'architecture correspondante à l'entité : description du fonctionnement.

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit.

En effet le fonctionnement d'un circuit est généralement décrit par plusieurs modules **VHDL**. Il faut comprendre par module le couple *ENTITE/ARCHITECTURE*. Dans le cas de simples **PLDs** on trouve souvent un seul module.

L'architecture établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel voire les deux séquentiel et combinatoire.

Exemples :

```
-- Opérateurs logiques de base
entity PORTES is
      port (A,B :in std_logic;
      Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end PORTES;

architecture DESCRIPTION of PORTES is
begin
      Y1 <= A and B;
      Y2 <= A or B;
      Y3 <= A xor B;
      Y4 <= not A;
      Y5 <= A nand B;
      Y6 <= A nor B;
      Y7 <= not(A xor B);
end DESCRIPTION;

-- Décodeurs 7 segments
entity DEC7SEG4 is
      port (DEC :in std_logic_vector(3 downto 0);
      SEG:out std_logic_vector(0 downto 6));
end DEC7SEG4;

architecture DESCRIPTION of DEC7SEG4 is
begin
      SEG <= "1111110" when DEC = 0
      else "0110000" when DEC = 1
      else "1101101" when DEC = 2
      else "1111001" when DEC = 3
      else "0110011" when DEC = 4
      else "1011011" when DEC = 5
      else "1011111" when DEC = 6
      else "1110000" when DEC = 7
      else "1111111" when DEC = 8
      else "1111011" when DEC = 9
      else "-------";
end DESCRIPTION;
```

```
-- Décodage d'adresses
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;
```

IV) Les instructions de base (mode « concurrent »), logique combinatoire.

Qu'est ce que le mode ***« concurrent »*** ? Pour une description **VHDL** toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. L'ordre dans lequel elles sont écrites n'a aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description **VHDL** et un langage informatique classique.

Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres.

Avec **VHDL** il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, cela n'est pas toujours évident.

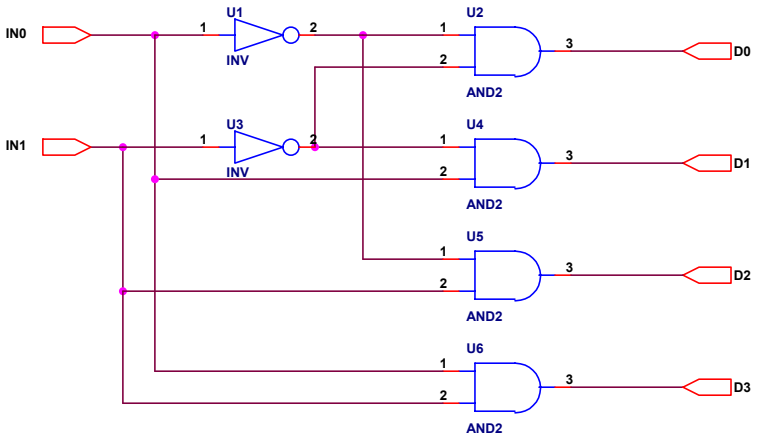
Exemple : Pour le décodeur 1 parmi 4 de la page 5, l'ordre dans lequel seront écrites les instructions n'a aucune importance.

```
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;
```

L'architecture ci dessous est équivalente :

```
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;
```

L'instruction définissant l'état de D0 à été déplacée à la troisième ligne, la synthèse de cette architecture est équivalente à la première.



IV.1) Les opérateurs.

IV.1.1) L'affectation simple : <=

Dans une description **VHDL**, c'est certainement l'opérateur le plus utilisé. En effet il permet de modifier l'état d'un signal en fonction d'autres signaux et/ou d'autres opérateurs.

Exemple avec des portes logiques : `S1 <= E2 and E1 ;`

Les valeurs numériques que l'on peut affecter à un signal sont les suivantes :
- **'1'** ou **'H'** pour un niveau **haut** avec un signal de **1** bit.
- **'0'** ou **'L'** pour un niveau **bas** avec un signal de **1** bit.
- **'Z'** pour un état haute impédance avec un signal de **1** bit.
- **'-'** pour un état quelconque, c'est à dire **'0'** ou **'1'**. Cette valeur est très utilisée avec les instructions : `when … else` et `with …. Select ….`

- Pour les **signaux** composés de plusieurs bits on utilise les guillemets **" … "** , voir les exemples ci dessous :

- Les bases numériques utilisées pour les bus peuvent être :
  BINAIRE, exemple : `BUS <= "1001" ;  -- BUS = 9 en décimal`
  HEXA, exemple : `BUS <= X"9" ;  -- BUS = 9 en décimal`
  OCTAL, exemple : `BUS <= O"11" ;  -- BUS = 9 en décimal`

Remarque : La base décimale ne peut pas être utilisée lors de l'affectation de signaux. On peut seulement l'utiliser avec certains opérateurs, comme **+** et **–** pour réaliser des compteurs (voir le chapitre sur les compteurs).

Exemple:

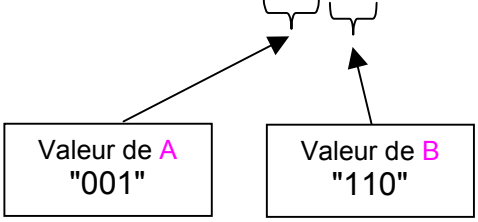
```
Library ieee;
Use ieee.std_logic_1164.all;

entity AFFEC is
port (
      E1,E2              : in std_logic;
      BUS1,BUS2,BUS3     : out std_logic_vector(3 downto 0);
      S1,S2,S3,S4        : out std_logic);
end AFFEC;

architecture DESCRIPTION of AFFEC is

begin
S1 <= '1'; -- S1 = 1
S2 <= '0'; -- S2 = 0
S3 <= E1;  -- S3 = E1
S4 <= '1' when (E2 ='1') else 'Z'; -- S4 = 1 si E1=1 sinon S4
                                   -- prend la valeur haute impédance
BUS1 <= "1000";                    -- BUS1 = "1000"
BUS2 <= E1 & E2 & "10";            -- BUS2 = E1 & E2 & 10
BUS3 <= x"A";                      -- valeur en HEXA -> BUS3 = 10(déc)

end DESCRIPTION;
```

IV.1.2) Opérateur de concaténation : &.

Cet opérateur permet de joindre des signaux entre eux .

Exemple :

```
--   Soit A et B de type 3 bits et S1 de type 8 bits
--   A = "001"  et B ="110"
S1 <= A & B & "01" ;
-- S1 prendra la valeur suivante après cette affectation
-- S1 = "001110 01"
```



IV.1.3) Opérateurs logiques.

Opérateur	VHDL
ET	and
NON ET	nand
OU	or
NON OU	nor
OU EXCLUSIF	xor
NON OU EXCLUSIF	xnor
NON	not
DECALAGE A GAUCHE	sll
DECALAGE A DROITE	srl
ROTATION A GAUCHE	rol
ROTATION A DROITE	ror

Exemples :

```
S1 <= A sll 2 ;   -- S1 = A décalé de 2 bits à gauche.

S2 <= A rol 3 ;   -- S2 = A avec une rotation de 3 bits à gauche

S3 <= not (R);    -- S3 = R
```

Remarque : Pour réaliser des décalages logiques en synthèse logique, il est préférable d'utiliser les instructions suivantes :

Décalage à droite :

```
-- 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
```

Décalage à gauche :

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

IV.1.4) Opérateurs arithmétiques.

Opérateur	VHDL
ADDITION	+
SOUSTRACTION	-
MULTIPLICATION	*
DIVISION	/

Remarque N°1 : Pour pouvoir utiliser les opérateurs ci-dessus il faut rajouter les bibliothèques suivantes au début du fichier **VHDL**:

```
Use ieee.numeric_std.all ;
Use ieee.std_logic_arith.all ;
```

Exemples :

```
S1 <= A - 3 ; -- S1 = A - 3
              -- On soustrait 3 à la valeur de l'entrée / signal A

S1 <= S1 + 1 ; -- On incrémente de 1 le signal S1
```

Remarque N°2 : Attention l'utilisation de ces opérateurs avec des signaux comportant un nombre de bits important peut générer de grandes structures électroniques.

Exemples :

```
S1 <= A * B ;-- S1 = A multiplié par B : A et B sont codés sur 4 bits

S2 <= A / B ;-- S2 = A divisé par B : A et B sont codés sur 4 bits
```

VI.1.5) Opérateurs relationnels.

Ils permettent de modifier l'état d'un signal ou de signaux suivant le résultat d'un test ou d'une condition. En logique combinatoire ils sont souvent utilisés avec les instructions :

- ***when … else …***
- ***with …. Select ….***

Voir ci-dessous.

Opérateur	VHDL
Egal	=
Non égal	/=
Inférieur	<
Inférieur ou égal	<=
Supérieur	>
Supérieur ou égal	>=

IV.2) Les instructions du mode « concurrent ».

IV.2.1) Affectation conditionnelle :

Cette instruction modifie l'état d'un signal suivant le résultat d'une condition logique entre un ou des signaux, valeurs, constantes.

```
SIGNAL <= expression when condition
          [else expression when condition]
          [else expression];
```

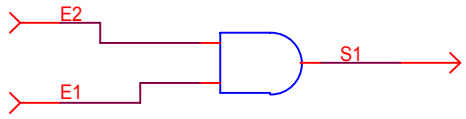
Remarque : l'instruction `[else expression]` n'est pas obligatoire mais elle fortement conseillée, elle permet de définir la valeur du `SIGNAL` dans le cas où la condition n'est pas remplie.

on peut mettre en cascade cette instruction voir l'exemple N°2 ci-dessous

Exemple N°1 :

```
-- S1 prend la valeur de E2 quand E1='1' sinon S1 prend la --
valeur '0'
S1 <= E2 when ( E1= '1') else '0';
```

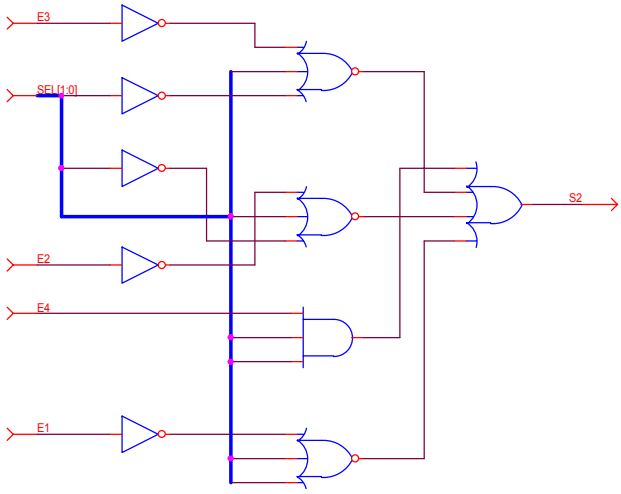
Schéma correspondant :



Exemple N°2 :

```
-- Structure évoluée d'un multiplexeur 4 vers 1
S2 <= E1 when (SEL="00" ) else
      E2 when (SEL="01" ) else
      E3 when (SEL="10" ) else
      E4 when (SEL="11" )
      else '0';
```

Schéma correspondant après synthèse:



IV.2.2) Affectation sélective :

Cette instruction permet d'affecter différentes valeurs à un signal, selon les valeurs prises par un signal dit de sélection.

```
with SIGNAL_DE_SELECTION select
     SIGNAL <=      expression when valeur_de_selection,
                    [expression when valeur_de_selection,]
                    [expression when others];
```

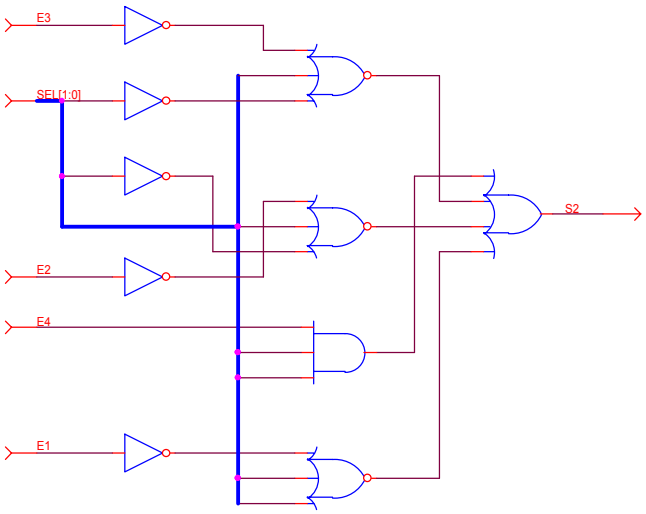
Remarque: l'instruction *[expression when others]* n'est pas obligatoire mais fortement conseillée, elle permet de définir la valeur du *SIGNAL* dans le cas où la condition n'est pas remplie.

Exemple N°1 :

```
-- Multiplexeur 4 vers 1
with SEL select
    S2 <=   E1  when "00",
            E2  when "01",
            E3  when "10",
            E4  when "11",
            '0' when others;
```

Remarque: *when others* est nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.

Schéma correspondant après synthèse:



En conclusion, les descriptions précédentes donnent le même schéma, ce qui est rassurant. L'étude des deux instructions montre toute la puissance du langage **VHDL** pour décrire un circuit électronique, en effet si on avait été obligé d'écrire les équations avec des opérateurs de base pour chaque sortie, on aurait eu les instructions suivantes :

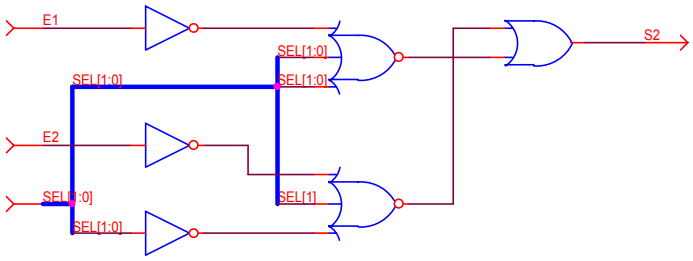
```
S2 <= (E1 and not(SEL(1)) and not(SEL(0))) or (E2 and not SEL(1) and
(SEL(0)) or (E3 and SEL(1) and not(SEL(0))) or (E4 and SEL(1) and SEL(0));
```

L'équation logique ci-dessus donne aussi le même schéma, mais elle est peu compréhensible, c'est pourquoi on préfère des descriptions de plus haut niveau en utilisant les instructions **VHDL** évoluées.

Exemple N°2 : Affectation sélective avec les autres cas forcés à '0'.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity TABLE1 is
      port (
            E1,E2 : in std_logic;
            SEL   : in std_logic_vector(1 downto 0);
            S2    : out std_logic);
end TABLE1;
architecture DESCRIPTION of TABLE1 is
begin
      with SEL select
            S2 <= E1  when "00",
                  E2  when "01",
                  '0' when others;   -- Pour les autres cas de SEL S2
                                     -- prendra la valeur 0 logique
end DESCRIPTION;
```

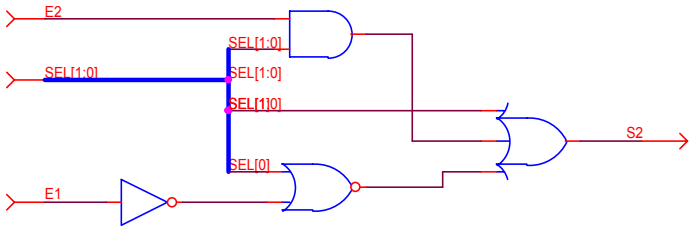
Schéma correspondant après synthèse:



Exemple N°3 : Affectation sélective avec les autres cas forcés à un '1'.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity TABLE2 is
      port (
            E1,E2 : in std_logic;
            SEL   : in std_logic_vector(1 downto 0);
            S2    : out std_logic);
end TABLE2;
architecture DESCRIPTION of TABLE2 is
begin
      with SEL select
            S2 <= E1  when "00",
                  E2  when "01",
                  '1' when others;   -- Pour les autres cas de SEL S2
                                     -- prendra la valeur 1 logique
end DESCRIPTION;
```

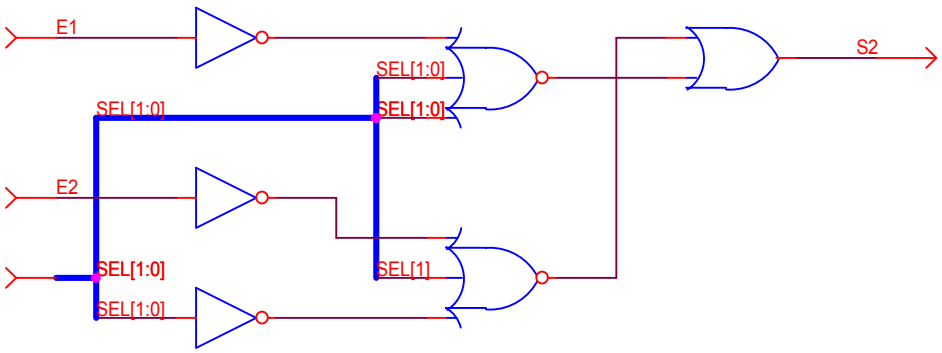
Schéma correspondant après synthèse:



Exemple N°4 :Affectation sélective avec les autres cas forcés à une valeur quelconque '-'.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity TABLE3 is
      port (
            E1,E2 : in std_logic;
            SEL   : in std_logic_vector(1 downto 0);
            S2    : out std_logic);
end TABLE3;
architecture DESCRIPTION of TABLE3 is
begin
      with SEL select
            S2 <= E1  when "00",
                  E2  when "01",
                  '-' when others;        -- Pour les autres cas de SEL S2
                                          -- prendra la valeur quelconque
end DESCRIPTION;
```

Schéma correspondant après synthèse:



V) Les instructions du mode séquentiel.

V.1) Définition d'un PROCESS.

Un **process** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement c'est à dire les unes à la suite des autres.

Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standard de la programmation structurée comme dans les systèmes à microprocesseurs.

L'exécution d'un **process** est déclenchée par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans **la liste de sensibilité** lors de la déclaration du **process**.

```
[Nom_du_process :] process(Liste_de_sensibilité_nom_des_signaux)
Begin
     -- instructions du process
end process [Nom_du_process] ;
```

Remarque: Le nom du *process* entre crochet est facultatif, mais il peut être très utile pour repérer un *process* parmi d'autres lors de phases de mise au point ou de simulations.

Règles de fonctionnement d'un process :

1) L'exécution d'un *process* a lieu à chaque changement d'état d'un signal de la liste de sensibilité.
2) Les instructions du *process* s'exécutent séquentiellement.
3) Les changements d'état des signaux par les instructions du *process* sont pris en compte à la **fin** du *process*.

V.2) Les deux principales structures utilisées dans un process.

L'assignation conditionnelle	L'assignation sélective

`if condition then`
`  instructions`
`[elsif condition then instructions]`
`[else instructions]`
`end if ;`
 	
Exemple:
`if (RESET='1') then SORTIE <= "0000";`
`end if ;`
	`case signal_de_slection is`
	`when valeur_de_sélection => instructions`
	`[when others => instructions]`
	`end case;`
	
	Exemple:
	`case SEL is`
	`  when "000" => S1 <= E1;`
	`  when "001" => S1 <= '0';`
	`  when "010" | "011" => S1 <='1';`
	`  -- La barre | permet de réaliser`
	`  -- un ou logique entre les deux`
	`  -- valeurs "010" et "011"`
	`  when others => S1 <= '0';`
	`end case;`

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

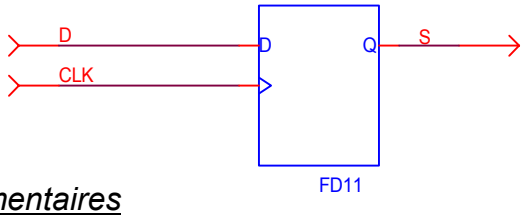
Ces deux instructions vont être utilisées dans la suite du polycopié.

V.3) Exemples de process :

Exemple N°1 : Déclaration d'une bascule D.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity BASCULED is
      port (
            D,CLK : in std_logic;
            S     : out std_logic);
end BASCULED;
architecture DESCRIPTION of BASCULED is
begin
      PRO_BASCULED : process (CLK)
      begin
            if (CLK'event and CLK ='1') then
                  S <= D;
            end if;
      end process PRO_BASCULED;
end DESCRIPTION;
```

Schéma correspondant après synthèse:



Commentaires

- Seul le signal *CLK* fait partie de la liste de sensibilité. D'après les règles de fonctionnement énoncées précédemment, seul un changement d'état du signal *CLK* va déclencher le process et par conséquent évaluer les instructions de celui-ci.

- L'instruction *if (CLK'event and CLK='1') then* permet de détecter un front montant du signal *CLK*. La détection de front est réalisée par l'attribut *event* appliqué à l'horloge *CLK*. Si on veut un déclenchement sur un front descendant, il faut écrire l'instruction suivante : *if (CLK'event and CLK='0').*

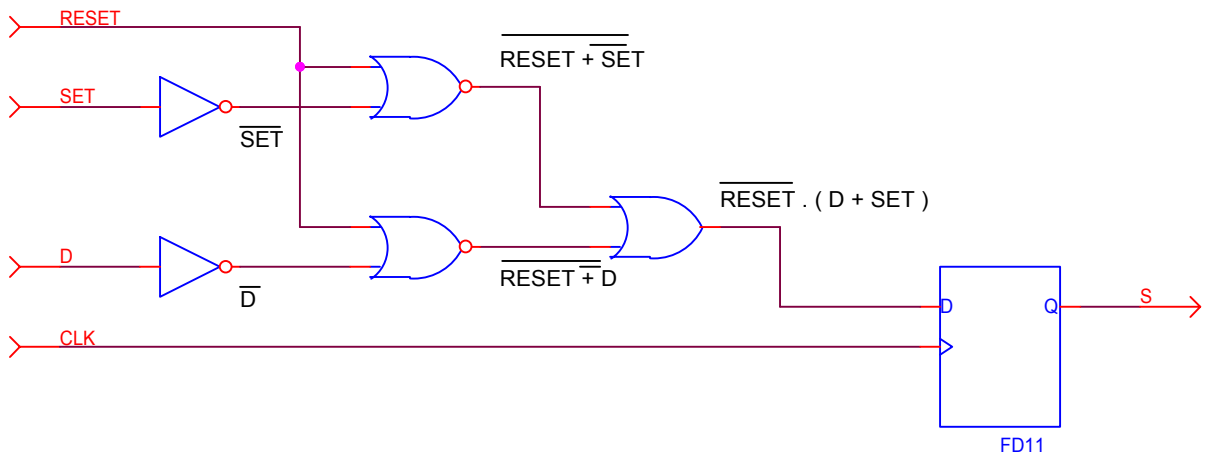
- Les bibliothèques IEEE possèdent deux instructions permettant de détecter les fronts montants) *rising_edge(CLK)* ou descendants *falling_edge(CLK)*.

- Si la condition est remplie alors le signal de sortie *S* sera affecté avec la valeur du signal d'entrée *D*.

Exemple N°2 : Même exemple que précédemment mais avec des entrées de présélections de mise à zéro *RESET* prioritaire sur l'entrée de mise à un *SET*, toutes les deux sont synchrones de l'horloge *CLK*.

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

entity BASCULEDSRS is
      port (
            D,CLK,SET,RESET   : in std_logic;
            S                 : out std_logic);
end BASCULEDSRS;

architecture DESCRIPTION of BASCULEDSRS is
begin
      PRO_BASCULEDSRS : process (CLK)
      Begin
            if (CLK'event and CLK ='1') then
                  if (RESET ='1') then
                        S <= '0';
                  elsif (SET ='1')then
                        S <= '1';
                  else
                        S <= D;
                  end if;
            end if;
      end process PRO_BASCULEDSRS;
end DESCRIPTION;
```

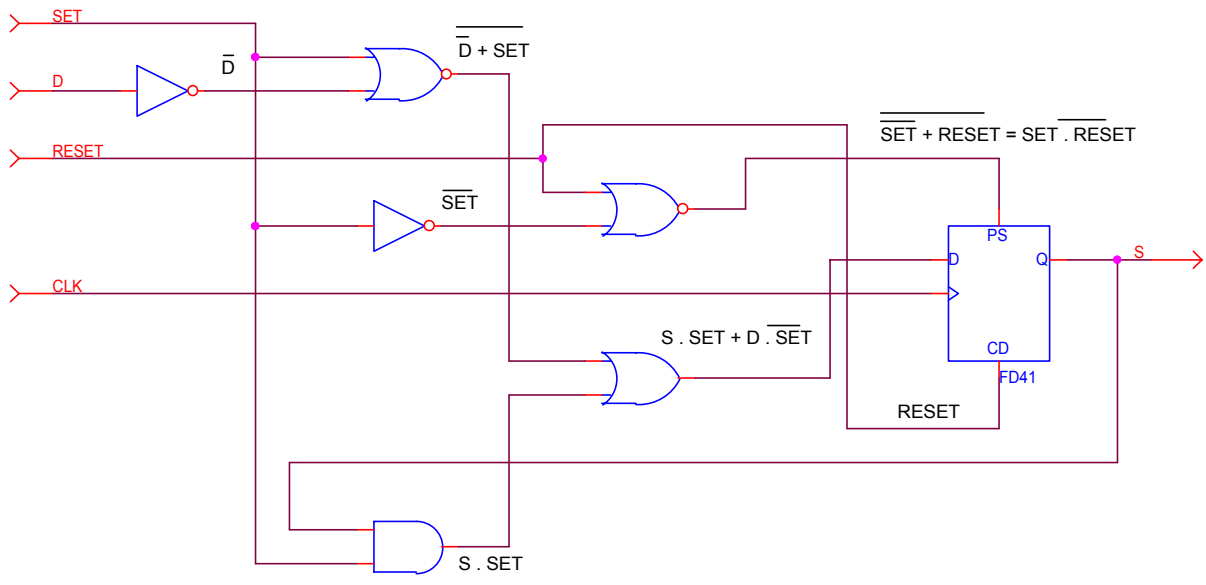
Schéma correspondant après synthèse:



Exemple N°3 : Même exemple que précédemment mais avec des entrées de présélections, de mise à zéro *RESET* prioritaire sur l'entrée de mise à un *SET*, toutes les deux sont asynchrones de l'horloge *CLK*.

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

entity BASCULEDSRA is
      port (
            D,CLK,SET,RESET   : in std_logic;
            S                 : out std_logic);
end BASCULEDSRA;

architecture DESCRIPTION of BASCULEDSRA is
begin
      PRO_BASCULEDSRA : process (CLK,RESET,SET)
      Begin
            if (RESET ='1') then
                  S <= '0';
            elsif (SET ='1')then
                  S <= '1';
            elsif (CLK'event and CLK ='1') then
                  S <= D;
            end if;
      end process PRO_BASCULEDSRA;
end DESCRIPTION;
```

Schéma correspondant après synthèse:



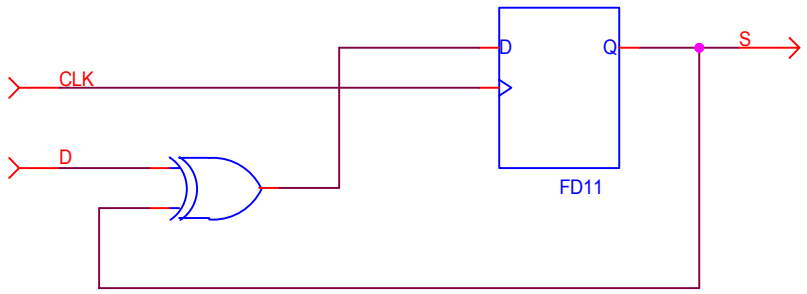
Commentaire

L'entrée *RESET* est prioritaire sur l'entrée *SET* qui est à son tour prioritaire sur le front montant de l'horloge *CLK*. Cette description est asynchrone car les signaux d'entrée *SET* et *RESET* sont mentionnés dans la liste de sensibilité du *process*.

Exemple N°4 : Bascule T.

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

entity BASCULET is
      port (
            D,CLK       : in std_logic;
            S           : out std_logic);
end BASCULET;

architecture DESCRIPTION of BASCULET is
signal S_INTERNE : std_logic; -- Signal interne
begin
      PRO_BASCULET : process (CLK)
      Begin
            if (CLK'event and CLK ='1') then
                  if (D ='1') then
                        S_INTERNE <= not (S_INTERNE);
                  end if;
            end if;
      end process PRO_BASCULET;
      S <= S_INTERNE;
end DESCRIPTION;
```

Schéma correspondant après synthèse:



Commentaires :

La description ci-dessus fait appel à un signal interne appelé ***S_INTERNE***, pourquoi faire appel à celui-ci ? La réponse est que le signal ***S*** est déclaré comme une sortie dans l'entité, et par conséquent on ne peut pas utiliser une sortie en entrée.
Pour contourner cette difficulté on utilise un signal interne qui peut être à la fois une entrée ou une sortie. Avec cette façon d'écrire, les signaux de sortie d'une description ne sont jamais utilisés comme des entrées. Cela permet une plus grande portabilité du code.

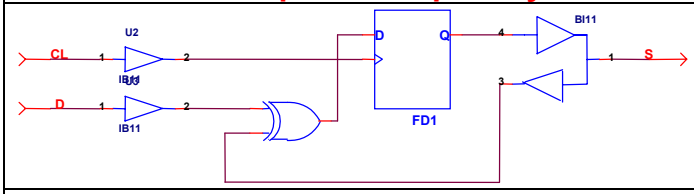
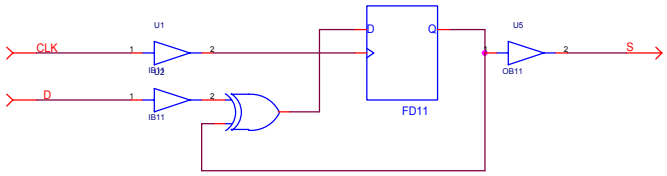
Si on ne déclare pas de signal interne, le synthétiseur renverra certainement une erreur du type : ***Error, cannot read output: s; [use mode buffer or inout].***
Le synthétiseur signale qu'il ne peut pas lire la sortie ***S*** et par conséquent, celle-ci doit être du type ***buffer*** ou ***inout***.

Remarque : Si on souhaite ne pas utiliser de variable interne on peut déclarer le signal ***s*** de type ***buffer*** ou ***inout***.

Ce qui donne pour descriptions :

Avec *s* de type *inout*	**Avec *s* de type *buffer***

`Library ieee;`
`Use ieee.std_logic_1164.all;`
`Use ieee.numeric_std.all;`
`Use ieee.std_logic_unsigned.all;`
 	
`entity BASCULET is`
`port (`
`D,CLK : in std_logic;`
`S : inout std_logic);`
`end BASCULET;`
 	
`architecture DESCRIPTION of BASCULET is`
`begin`
`PRO_BASCULET : process (CLK)`
`Begin`
`if (CLK'event and CLK='1') then`
`if (D='1') then`
`S <= not (S);`
`end if;`
`end if;`
`end process PRO_BASCULET;`
`end DESCRIPTION;`
	`Library ieee;`
	`Use ieee.std_logic_1164.all;`
	`Use ieee.numeric_std.all;`
	`Use ieee.std_logic_unsigned.all;`
	
	`entity BASCULET is`
	`port (`
	`D,CLK : in std_logic;`
	`S : buffer std_logic);`
	`end BASCULET;`
	
	`architecture DESCRIPTION of BASCULET is`
	`begin`
	`PRO_BASCULET : process (CLK)`
	`Begin`
	`if (CLK'event and CLK='1') then`
	`if (D='1') then`
	`S <= not (S);`
	`end if;`
	`end if;`
	`end process PRO_BASCULET;`
	`end DESCRIPTION;`

Schéma correspondant après synthèse:	**Schéma correspondant après synthèse:**
	
Commentaires : On peut constater que ***s*** est bien du type ***inout***.	**Commentaires :** On peut constater que ***s*** est bien du type ***buffer***.

V.4) Les compteurs :

Ils sont très utilisés dans les descriptions **VHDL**. L'écriture d'un compteur peut être très simple comme très compliquée. Ils font appels aux ***process***.

V.4.1) Compteur simple :

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

entity CMP4BITS is
PORT (
      CLOCK : in std_logic;
      Q     : inout std_logic_vector(3 downto 0));
end CMP4BITS;

architecture DESCRIPTION of CMP4BITS is
begin
  process (CLOCK)
  begin
    if (CLOCK ='1' and CLOCK'event) then
      Q <= Q + 1;
    end if;
  end process;
end DESCRIPTION;
```

Commentaires :

La description ci-dessus est très simple. Le déclenchement du ***process*** se fera sur un changement d'état du signal ***CLOCK***, l'incrémentation de la sortie ***Q*** se fera sur le front montant de l'horloge ***CLOCK***.

Cette description fort simple, appelle quelques commentaires :

- L'incrémentation du compteur est réalisée par l'opérateur **+** associé à la valeur **1**. Cela est logique, mais elle l'est beaucoup moins pour les PLDs et les synthétiseurs, pourquoi ? La réponse est que les entrées et sorties ainsi que les signaux sont déclarés de type ***std_logic*** ou ***std_logic_vector***, et par conséquent on ne peut pas leur associer de valeur entière décimale.

- Un signal peut prendre comme valeur les états `'1'` ou `'0'` et un bus n'importe quelle valeur, du moment qu'elle est écrite entre deux guillemets `"1010"` ou `X"A"` ou `o"12"`, mais pas une valeur comme par exemple `1,2,3,4`. Ces valeurs décimales sont interprétées par le synthétiseur comme des valeurs entières (`integer`), on ne peut pas par défaut additionner un nombre entier `1` avec un bus de type électronique (`st_logic_vector`), c'est pour cela que l'on rajoute dans la partie déclaration des bibliothèques les lignes :

```
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
```

  Ces deux bibliothèques ont des fonctions de conversions de types et elles permettent d'associer un entier avec des signaux électroniques. Elles permettent d'écrire facilement des compteurs, décompteurs, additionneurs, soustracteurs, …..

Remarque : Il ne faut pas oublier de les rajouter dans les descriptions.

- Le signal `Q` est déclaré dans l'entité de type `inout`, cela est logique car il est utilisé à la fois comme entrée et comme sortie pour permettre l'incrémentation du compteur. Ce type d'écriture est peu utilisé car elle ne permet pas au code d'être portable, on préféra utiliser un signal interne, celui-ci peut être à la fois une entrée et une sortie.

Même description en utilisant un bus interne:

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

entity CMP4BITS is
PORT (
      CLOCK : in std_logic;
      Q     : out std_logic_vector (3 downto 0));
end CMP4BITS;

architecture DESCRIPTION of CMP4BITS is
signal Q_BUS_INTERNE : std_logic_vector(3 downto 0));
begin
  process (CLOCK)
  begin
    if (CLOCK ='1' and CLOCK'event) then
      Q_BUS_INTERNE <= Q_BUS_INTERNE + 1;
    end if;
  end process;

  Q <= Q_BUS_INTERNE;   -- affectation du bus interne au
                        -- signal de sortie Q
end DESCRIPTION;
```

V.4.2) Compteur mise à un SET et mise à zéro RESET :

V.4.2.1) Compteur 3 bits avec remise à zéro 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;
```

V.4.2.2) Compteur 3 bits avec remise à zéro 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;
```

Quelle différence entre les deux descriptions ? Dans la deuxième le signal ***RESET*** n'est plus dans la liste de sensibilité, par conséquence le ***process*** ne sera déclenché que par le signal ***CLOCK***. La remise à zéro ne se fera que si un front montant sur ***CLOCK*** a lieu.

V.4.3) Compteur / Décompteur à entrée de préchargement :

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

entity CMP4BITSLUD is
PORT (
      RESET, CLOCK, LOAD, UP: in std_logic;
            DATA  : in std_logic_vector (3 downto 0);
      Q     : out std_logic_vector (3 downto 0));
end CMP4BITSLUD;

architecture DESCRIPTION of CMP4BITSLUD is
signal CMP: std_logic_vector (3 downto 0);
begin
  process (RESET,CLOCK)
  begin
    if RESET ='1' then
      CMP <= "0000";          -- Remise à zero asynchrone du compteur
    elsif (CLOCK ='1' and CLOCK'event) then
if (LOAD ='1') then
            CMP <= DATA;            -- Préchargement synchrone
      else
            if (UP ='1') then
                  CMP <= CMP + 1;   -- Incrémentation synchrone
            else
                  CMP <= CMP - 1;   -- Décrémentattion synchrone
            end if;
      end if;
    end if;
  end process;
  Q <= CMP;
end DESCRIPTION;
```

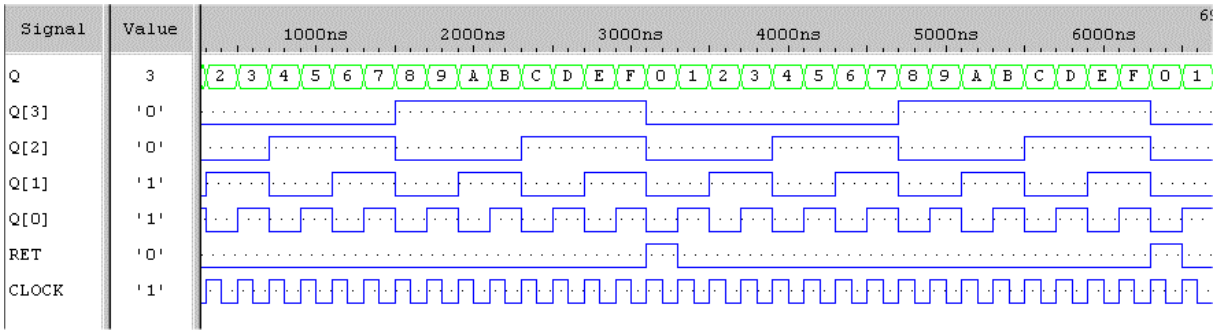
Remarque : La mise à zéro des sorties du compteur passe par l'instruction :
`CMP <= "0000";`
une autre façon d'écrire cette instruction est :
`CMP <= (others => '0') ;`
Cette dernière est très utilisée dans les descriptions car elle permet de s'affranchir de la taille du bus. En effet `others=>'0'` correspond à mettre tous les bits du bus à zéro quelque soit le nombre de bits du bus. De la même façon on peut écrire `others=>'1'` pour mettre tous les bits à un.

V.4.4) Les erreurs classiques avec l'utilisation de *process* :

Exemple : compteur avec retenue (fonctionnement incorrect).

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP4BITSRET is
PORT (
      RESET, CLOCK      : in std_logic;
      RET               : out std_logic;
      Q                 : out std_logic_vector (3 downto 0));
end CMP4BITSRET;
architecture DESCRIPTION of CMP4BITSRET is
signal CMP: std_logic_vector (3 downto 0);
begin
  process (RESET,CLOCK)
  begin
    if RESET ='1' then
            CMP <= "0000";
    elsif (CLOCK ='1' and CLOCK'event) then
      CMP <= CMP + 1;
      if (CMP = "1111") then
            RET <= '1';
      else
            RET <= '0';
      end if;
    end if;
  end process;
  Q <= CMP;
end DESCRIPTION;
```

Signaux de simulation obtenus :



Les résultats de simulation appellent quelques commentaires :

On s'aperçoit que le signal de retenue ***RET*** passe au niveau logique haut quand la valeur du compteur vaut ***0***, pourquoi ?Il ne faut pas oublier la règle 3 des ***process*** (voir page N°21) qui dit que les valeurs de signaux à l'intérieur d'un ***process*** ne sont mis à jour qu'à la fin de celui-ci.

Dans notre cas, prenons l'état où ***CMP=14,*** au coup d'horloge suivant on incrémente le compteur ***CMP***, mais la nouvelle valeur ne sera affectée à ***CMP*** qu'à la fin du ***process***, donc quand le test pour valider le signal de retenue est effectué, la valeur de ***CMP*** est égale à ***14***, et celui-ci n'est pas valide.

Au coup d'horloge suivant ***CMP=15*** et ***CMP*** est incrémenté donc prendra la valeur ***0*** à la fin du ***process***., mais la condition ***CMP= "1111"*** sera valide et le signal de retenue ***RET*** passera au niveau logique un.

Comment faire pour pallier à ce problème ? deux solutions :

1) Il faut anticiper d'un coup d'horloge, on valide la retenue quand la valeur du compteur vaut ***14***, c'est à dire ***n-1***.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP4BITSRET is
PORT (
      RESET, CLOCK      : in std_logic;
      RET               : out std_logic;
      Q                 : out std_logic_vector (3 downto 0));
end CMP4BITSRET;
architecture DESCRIPTION of CMP4BITSRET is
signal CMP: std_logic_vector (3 downto 0);
begin
  process (RESET,CLOCK)
  begin
    if RESET ='1' then
      CMP <= "0000";
    elsif (CLOCK ='1' and CLOCK'event) then
      CMP <= CMP + 1;
      if (CMP = "1110") then
            -- La retenue passera à un  quand CMP = 14 décimal
            RET <= '1';
      else
            RET <= '0';
    end if;
    end if;
  end process;
  Q <= CMP;
end DESCRIPTION;
```

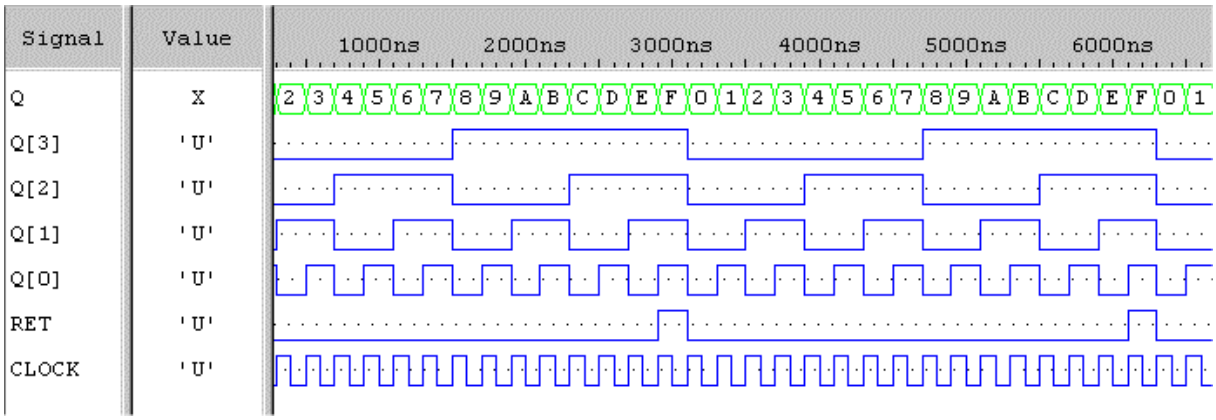
Remarque : Dans ce cas la validation de la retenue s'effectue de façon synchrone car elle est dans le ***process***, mais la description est peu lisible.

2) Le test de validation de la retenue est effectuée en dehors du process.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP4BITSRET is
PORT (
      RESET, CLOCK      : in std_logic;
      RET               : out std_logic;
      Q                 : out std_logic_vector (3 downto 0));
end CMP4BITSRET;
architecture DESCRIPTION of CMP4BITSRET is
signal CMP: std_logic_vector (3 downto 0);
begin
  process (RESET,CLOCK)
  begin
    if RESET='1' then
      CMP <= "0000";
    elsif (CLOCK ='1' and CLOCK'event) then
      CMP <= CMP + 1;
    end if;
  end process;
  Q <= CMP;
  -- Validation de la retenue
  RET <= '1' when (CMP = "1111") else '0';
end DESCRIPTION;
```

Remarque : Dans ce cas la validation de la retenue s'effectue de façon asynchrone car elle est en dehors du ***process***, mais la description est lisible.

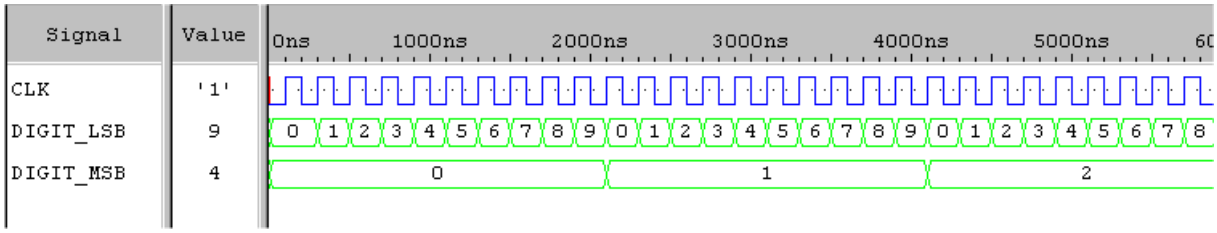
Signaux de simulation obtenus :



V.4.5) Compteur BCD deux digits :

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

entity CMP_BCD8 is
      PORT (
            RESET, CLK : in std_logic;
            Q      : out std_logic_vector (7 downto 0));
end CMP_BCD8;

architecture DESCRIPTION of CMP_BCD8 is
signal DIGIT_MSB, DIGIT_LSB: std_logic_vector (3 downto 0);
begin
  process (RESET,CLK)
  begin
    if RESET ='1' then
      DIGIT_LSB <= (others=>'0');
      DIGIT_MSB <= (others=>'0');
    elsif (CLK ='1' and CLK'event) then
          if DIGIT_LSB < 9 then
             DIGIT_LSB <= DIGIT_LSB + 1;
          else
             DIGIT_LSB <= (others=>'0');
             if DIGIT_MSB < 9 then
                  DIGIT_MSB <= DIGIT_MSB + 1 ;
             else
                  DIGIT_MSB <= (others=>'0');
             end if;
          end if;
    end if;
  end process;
  Q <= DIGIT_MSB & DIGIT_LSB;
end DESCRIPTION;
```

Signaux de simulation obtenus :



VI) Syntaxe résumée du langage VHDL.

Ces deux pages présentent de manière très condensée la syntaxe du langage VHDL. Il est fortement conseillé de se reporter au document complet pour plus de détails.

Commentaires relatifs au fichier VHDL

```
-- Nom des auteurs
-- Description du rôle du fichier VHDL
-- Circuit logique programmable utilisé
```

Définition des librairies utilisées

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
-- librairie additionnelle pour les opérations arithmétiques
```

Description de l'entité : interface externe.
NOMENTITE permet d'identifier l'entité

Début de la description des signaux

```
entity NOMENTITE is
-- Description des signaux d'entrée et de sortie
    port (
     ENTREE            : in std_logic;          -- signal simple
     SORTIE            : out std_logic;         -- signal simple
     ENTREESORTIE      : inout std_logic;       -- signal simple
     BUFFER            : buffer std_logic;      -- signal simple
     BUS :       in std_logic_vector (3 downto 0);   -- signaux regroupés en un bus
     A,B :       in std_logic_vector (3 downto 0);   -- 2 bus 4 bits en entrée
     C,D,E,F     : out std_logic -- 4 signaux de même type PAS DE POINT VIRGULE !
          );
```

Fin de la description des signaux

```
-- En plus des lignes précédentes, il est nécessaire de spécifier le brochage
-- retenu pour le circuit utilisé.
-- La syntaxe suivante est propre à OrCad Express pour un SPLD
-- La définition du brochage doit respecter les possibilités du circuit :
--       IN ≠ CLOCK    IO ≠ OUT
```

Affectation automatique des broches d'un SPLD

```
-- Affectation automatique
attribute pldtype : string;          -- définition de l'attribut « Type de broche »
attribute pldtype of HORLOGE : signal is "CLOCK";   -- ne pas confondre avec IN
attribute pldtype of ENTREE : signal is "IN";
attribute pldtype of SORTIE : signal is "OUT";
attribute pldtype of ENTREESORTIE : signal is "IO"; -- ne pas confondre avec INOUT
attribute pldtype of VALIDATION : signal is "EN";
attribute pldtype of A,B,C : signal is "IN"; -- signaux simples ou bus de même type
attribute pldtype of D,E,F : signal is "IO"; -- signaux simples ou bus de même type
```

Affectation manuelle des broches d'un SPLD

```
-- Affectation manuelle
attribute pldpin : string;          -- définition de l'attribut « Numéro de broche »
attribute pldpin of NOMSIGNAL: signal is "1";          -- une seule broche
attribute pldpin of NOMBUS: signal is "23,22,21,20";       -- LSB en premier !
```

```
end NOMENTITE;
```

Fin de l'entité : NOMENTITE correspond au nom donné dans la déclaration

```
architecture NOMARCHITECTURE of NOMENTITE is

signal NOMAUX : STD_LOGIC_VECTOR (3 downto 0);

begin

-- partie Combinatoire
NOMBUS <= NOMAUX ; -- affectation du signal auxiliaire au bus de sortie

SORTIE <=    ENTREE0 when ( ENTREECOMMANDE =  "00") else
             ENTREE1 when( ENTREECOMMANDE =  "01") else
             ENTREE2 when( ENTREECOMMANDE =  "10") else ENTREE3 ;

with ENTREECOMMANDE select -- signal de sélection
  SORTIE <= ENTREE0 when "00", -- quand EntreeCommande = "00", Sortie vaut Entree0
  ENTREE1 when "01",           -- quand EntreeCommande = "01", Sortie vaut Entree1
  ENTREE2 when "10",           -- quand EntreeCommande = "10", Sortie vaut Entree2
  ENTREE3 when others;         -- Sortie vaut Entree3 dans les autres cas

-- partie Séquentielle
process (SIGNAL1, SIGNAL2,... )
  begin
 if SIGNAL1'event and SIGNAL1 = '1'-- détection d'un front montant sur Signal1
   then if NOMAUX >= x"9"          -- test d'inégalité
          then NOMAUX  <= x"0";    -- affectation simple
          else NOMAUX <= NOMAUX + 1;    -- incrémentation et affectation (autorisé
       end if; -- seulement avec la librairie ieee.std_logic_unsigned.all)
   end if;

 case ENTREECOMMANDE is -- signal de sélection
    when "00" => SORTIE <= ENTREE0; --quand EntreeCommande = "00", Sortie vaut Entree0
    when "01" => SORTIE <= ENTREE1; --quand EntreeCommande = "01", Sortie vaut Entree1
    when "10" => SORTIE <= ENTREE2; --quand EntreeCommande = "10", Sortie vaut Entree2
    wheN others => SORTIE <= ENTREE3; --dans les autres cas, Sortie vaut Entree3
 end case;
end process ;

end NOMARCHITECTURE;

-- REMARQUES
-- « Valeurs Numériques »
--  '1' : binaire pour un bit
--  "11" : binaire pour un bus
--  x"11" : heXadécimal pour un bus
--  11 : décimal (ne peut pas s'appliquer à tous les types de signaux)

-- « Tests »
-- = égal    /=  différent              <  inférieur                  >  supérieur
-- >=  supérieur ou égal                <=  inférieur ou égal
```

Description de l'architecture : fonctionnement interne.
NOMARCHITECTURE permet d'identifier l'architecture.
NOMENTITE correspond au nom donné dans la déclaration de l'entité.

Déclaration facultative de signaux auxiliaires

Début de l'Architecture

Description combinatoire : mode « Concurrent »

Affectation simple

Affectation conditionnelle

Affectation sélective

Description séquentielle : PROCESS suivi de la liste des signaux pouvant déclencher le process

Assignation Conditionnelle : if ... then ... else...

Assignation Sélective : case ... is ... when...

Fin du Process

Fin de l'Architecture : NOMARCHITECTURE correspond au nom donné dans la déclaration