CHAPITRE 3

VHDL en Logique Combinatoire

Plan du Chapitre

- Les opérateurs en VHDL
- Instructions du mode « CONCURRENT »
- Instructions du mode « SÉQUENTIEL »

Les opérateurs en VHDL

Plusieurs opérateurs logiques peuvent être utilisés pour la description VHDL d'un circuit logique:

- **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.

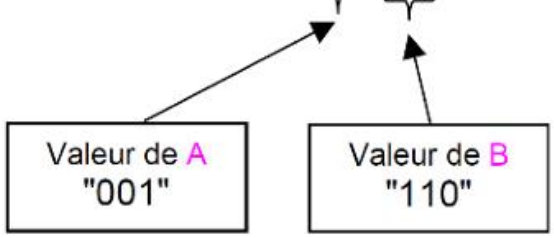
Exemple: *S1 <= E2 and E1;*

- **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"
```



Les opérateurs Logiques en VHDL :

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
```

Les opérateurs Logiques en VHDL :

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
```

Les opérateurs Arithmétiques en VHDL :

Opérateur	VHDL
ADDITION	+
SOUSTRACTION	-
MULTIPLICATION	*
DIVISION	/

- 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 ;
```

Exemple:

```
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

S1 <= A * B ;-- S1 = A multiplié par B

S2 <= A / B ;-- S2 = A divisé par B
```

- Attention l'utilisation des opérateurs + et * avec des signaux comportant un nombre de bits important peut générer de grandes structures électroniques.

Les opérateurs Relationnels en VHDL :

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

Ils permettent de modifier l'état d'un signal ou des signaux suivant le résultat d'un test ou d'une condition.

Ils sont souvent utilisés, en logique combinatoire, avec les instructions :

- when … else …
- with …. select ….

Signal, variable et constante

- Le type **signal** est utilisé pour des signaux intermédiaires. A définir dans l'en-tête de l'architecture.

```
signal NOM_DU_SIGNAL : type;
-- type = std_logic ou std_logic_vector
```

- Une **variable** est déclarée et uniquement valide dans un process (avant le begin du process). Affectation immédiate lors de :=

```
variable NOM_DE_VARIABLE : type [:= exp];
-- type = integer (souvent),...
```

- Une **constante** se déclare dans l'en-tête de l'architecture.

```
constant NOM_DE_CONSTANTE : type := exp;
-- type = integer, boolean,...
```

Application 1:

Coder en VHDL un additionneur complet à 1 bit en utilisant l'**opérateur de concaténation &.**

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

entity full_add1 is
  port (
    a, b, cin : in  std_logic;
    s, cout   : out std_logic
    );
end entity;

architecture arc of full_add1 is

  signal resultat : unsigned(1 downto 0);

begin

  resultat <= ('0' & a) + ('0' & b) + ('0' & cin);
  s        <= resultat(0);
  cout     <= resultat(1);

end arc;
```

Dans ce programme VHDL, on a utilisé:

- Un signal intermédiaire 'resultat'
- La concaténation de signaux pour créer un bus.

Instructions du mode «CONCURRENT»

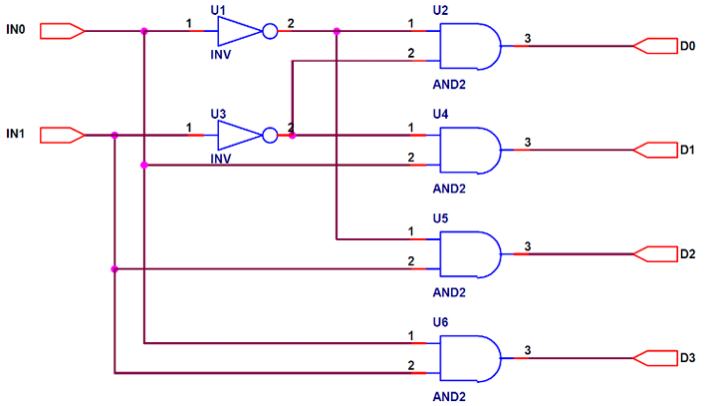
- Qu’est-ce que un mode « **concurrent** » ?

Comme tout langage de description de matériel, le VHDL décrit des structures par assemblage d'instructions **concurrentes** dont l'ordre d'écriture n'a aucune importance, et affectent les signaux de sortie en même temps.

Avec l‘instructions du mode «CONCURRENT»:

- ✓ L’ordre des instructions concurrentes est indifférent,
- ✓ Les instructions se situent entre BEGIN et END de l’architecture,
- ✓ L’objets manipulés par les instructions concurrentes sont les signaux.

Exemple: un décodeur 2 vers 4:




```
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;
```

Cette architecture est équivalente à:

```
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 a été déplacée à la troisième ligne, la synthèse de cette architecture est équivalente à la première.

Instructions du mode «CONCURRENT»

Les objets manipulés par les instructions concurrentes sont les signaux ’signal’ qui disposent chacun d'un échéancier de façon à effectuer une simulation d'instructions concurrentes.

Il existe 2 principales modes d’**affectations concurrentes** des signaux:

- Affectation conditionnelle
  when … else …
- Affectation sélective
  with …. select ….

Affectation conditionnelle

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

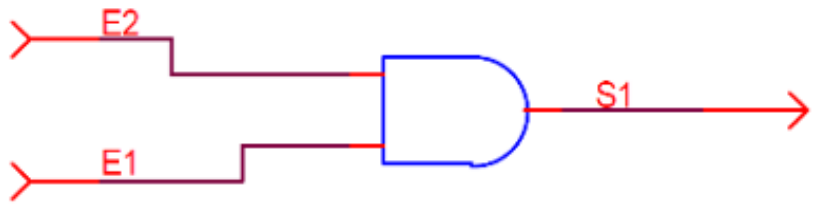
```
SIGNAL <= expression when condition
          [else expression when condition]
          [else expression];
```

- l'instruction [**else expression**] permet de définir la valeur du SIGNAL dans le cas où la condition n'est pas remplie.

Exemple : avec la porte logique AND : S1 <= E1 AND E2;

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

Schéma correspondant :



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];
```

- L'instruction [expression when others] permet de définir la valeur du SIGNAL dans le cas où la condition n'est pas remplie.
- 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.

Exemple: Description d'un décodeur, 4 vers 10, actif à l'état bas avec l'instruction **With … Select**:

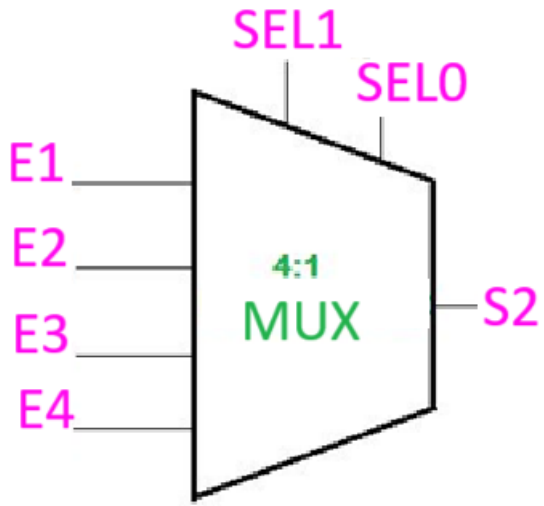
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity decodeur is
port ( A : in std_logic_vector(3 downto 0);
       S : out std_logic (9 downto 0));
End decodeur;

Architecture Arch of decodeur is
Begin
 With A select
    S <="1111111110" when "0000",
        "1111111101" when "0001",
        "1111111011" when "0010",
        "1111110111" when "0011",
        "1111101111" when "0100",
        "1111011111" when "0101",
        "1110111111" when "0110",
        "1101111111" when "0111",
        "1011111111" when "1000",
        "0111111111" when "1001",
        "1111111111" when others;
end Arch;
```

Application 2:

1. Dresser la table de vérité d'un multiplexeur 4 vers 1:



2. Déterminer l'équation générale de la sortie S2,
3. Coder en VHDL ce multiplexeur,
4. Coder en VHDL ce multiplexeur en utilisant les formes conditionnelles lors des affectations :
    - when … else
    - with … select

Application 2:

1. Dresser la table de vérité d'un multiplexeur 4 vers 1:

Sélection (SEL1 & SEL0)	Entrée sélectionnée (E)	Sortie (S2)
00	E1	E1
01	E2	E2
10	E3	E3
11	E4	E4

La combinaison des bits de sélection (SEL1 et SEL0) détermine quelle entrée (E0, E1, E2 ou E3) est connectée à la sortie S2.
Par exemple, si les bits de sélection sont "**01**", alors l'entrée E2 est sélectionnée et la sortie S2 prend la valeur de E2.

Application 2:

2. S2 <= ((not SEL0) and (not SEL1) and E1) or (SEL0 and (not SEL1) and E2) or ((not SEL0) and SEL1 and E3) or (SEL0 and SEL1 and E4);

3. Code VHDL d'un multiplexeur 4 vers 1:

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

entity MUX is
    Port(
            E1,E2,E3,E4,SEL0,SEL1: in  STD_LOGIC;
            S2 : out  STD_LOGIC);
end MUX;


architecture Flot_MUX of MUX is
begin
      S2 <= ((not SEL0) and (not SEL1) and E1) or (SEL0 and (not
      SEL1) and E2) or ((not SEL0) and SEL1 and E3) or (SEL0 and
      SEL1 and E4);

end Flot_MUX;
```

Application 2:

4. Coder en VHDL un multiplexeur 4 vers 1 en utilisant **l'affectation conditionnelle.**

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

entity MUX is
    Port(
            E1,E2,E3,E4 : in  STD_LOGIC;
            SEL: in  STD_LOGIC_VECTOR (1 downto 0);
            S2 : out  STD_LOGIC);
end MUX;

architecture Arc_MUX1 of MUX is
begin
      S2 <= E1 when (SEL="00" ) else
            E2 when (SEL="01" ) else
            E3 when (SEL="10" ) else
            E4 when (SEL="11" )
            else '0';
end Arc_MUX1;
```

Application 2:

4. Coder en VHDL un multiplexeur 4 vers 1 en utilisant **l'affectation sélective.**

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

entity MUX is
    Port(
            E1,E2,E3,E4 : in  STD_LOGIC;
            SEL: in  STD_LOGIC_VECTOR (1 downto 0);
            S2 : out  STD_LOGIC);
end MUX;

architecture Arc_MUX2 of MUX is
begin
     with SEL select
          S2 <=   E1   when "00",
                  E2   when "01",
                  E3   when "10",
                  E4   when "11",
                  '0'  when others;

end Arc_MUX2;
```

Application 2: Coder en VHDL un multiplexeur 4 vers 1:

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

entity MUX is
    Port(
            E1,E2,E3,E4,SEL0,SEL1: in  STD_LOGIC;
            S2 : out  STD_LOGIC);
end MUX;

architecture Arc_MUX of MUX is
begin
```

```
S <= E1 when (SEL0='0' and SEL1='0') else
     E2 when (SEL0='1' and SEL1='0') else
     E3 when (SEL0='0' and SEL1='1') else
     E4 when (SEL0='1' and SEL1='1')
     else '0';
```

```
with SEL1&SEL0 select
S <= E1 when '00',
     E2 when '01',
     E3 when '10',
     E4 when '11',
    '0' when others;
```

```
end Arc_MUX;
```

Instructions du mode «SÉQUENTIEL»

- Qu'est-ce que un mode « **séquentiel** » ?

Les instructions **séquentielles** sont exécutées les unes à la suite des autres, comme c'est le cas du langage C.

Il existe 2 principales structures de contrôle utilisées avec les instructions séquentielles :

- L'assignation conditionnelle
  IF … THEN … ELSE
- L'assignation sélective
  CASE …IS WHEN …

Assignation conditionnelle

Syntaxe:

```
if cond_1 then
        instructions_1;
elsif cond_2 then
        instructions_2;
   .
   .
   .
else instructions_n;
end if;
```

Exemple : avec la porte logique OR : s <= a OR b;

```
IF a = '1' THEN
     s <= '1';
ELSIF b = '1' THEN
     s <= '1';
ELSE
     s <= '0';
END IF;
```

Assignation sélective

Syntaxe: ***case signal_de_selection is***

when valeur_ de_selection1 => instructions_1;

when valeur_ de_selection2 => instructions_2;

.
.
.

when others => instructions_n;

end case;

Exemple :

```
CASE entrée IS
WHEN "11" => s <= '1';
WHEN "01" => s <= '0';
WHEN "10" => s <= '0';
WHEN OTHERS => s <= '1';
END CASE;
```

- Ne pas confondre => (implique)
  <= (affecte)