Eléments de base du langage VHDL

1 - Introduction

VHDL : Very high speed intregrated circuit, Hardware Description Langage (norme IEEE 1076)

C'est un langage qui décrit le comportement d'une fonction numérique.

Il est principalement utilisé :

⇒ Pour programmer les composants de type PLD (PAL, GAL), CPLD , FPGA.
Description de la fonction en VHDL ⇒ Compilation ⇒ fichier JEDEC

⇒ Concevoir des modèles de simulations numériques.

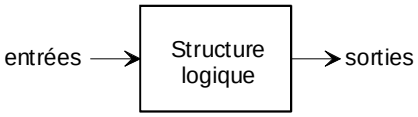
Dans nos applications courantes, les outils mis à disposition permettront de suivre la démarche suivante :

- Edition de la description de la fonction en VHDL .
- Simulation après édition des stimuli appliqués sur les entrées.
- Compilation et génération du fichier JEDEC pour programmer le composant cible.

On peut rencontrer de type de simulation :

- Une simulation à partir de la description en VHDL : description fonctionnelle (ne prend pas en compte le composant cible).
- Une simulation à partir du fichier JEDEC : prend en compte les contraintes du composant cible.

2 – Structure d'une description VHDL



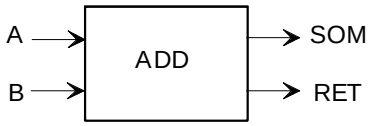
entité

```
entity    nom_entité  is
end   nom_entité ;
```

architecture

```
architecture    nom_arch    of nom_entité  is
zone de déclaration
begin
description de la structure logique
end nom_arch ;
```

exemple : additionneur 1 bit

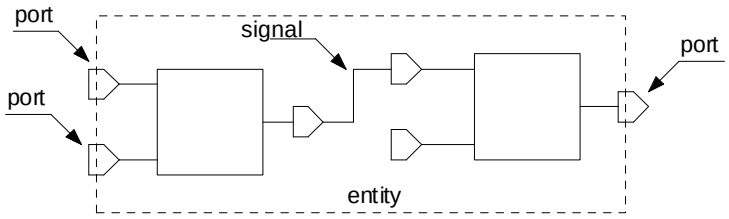


A	B	SOM	RET
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
entity ADD  is
port (A,B : in bit ; SOM,RET : out bit) ;
end ADD ;

architecture arch_add of ADD is
begin
SOM  <= A xor B ;
RET  <= A and B ;
end arch_add ;
```

3 – Ports et signaux

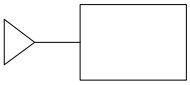
Un port est une entrée ou une sortie d'une entité. Un signal est interne à l'entité.



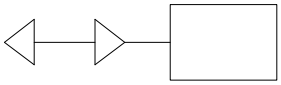
Un signal interne possède un nom, un type et éventuellement un ensemble de valeurs.

Sens d'un port

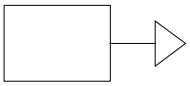
IN : entrée (monodirectionnelle)



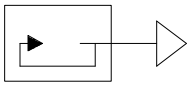
INOUT : entrée/sortie (bidirectionnelle)



OUT : sortie (monodirectionnelle)



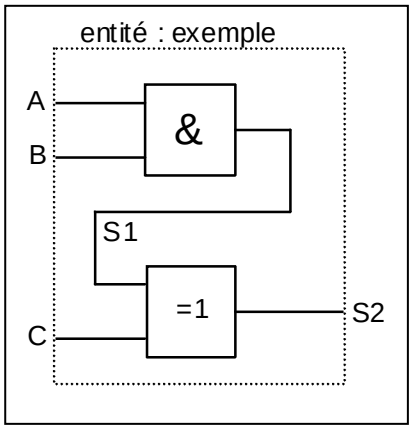
BUFFER : sortie/entrée (monodirectionnelle)



Principaux types
BIT, BOOLEAN, BIT_VECTOR

Exemple :

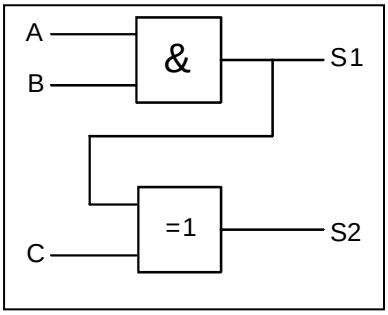
```
entity exemple  is
port (A,B,C : in bit ; S2 : out bit) ;
end exemple ;

architecture arch of exemple is
signal S1 : bit ;
begin
S2     <= S1 xor C ;
S1     <= A and B ;
end arch ;
```



4 – Instructions concurrentes

Les sorties évoluent de façon indépendante et parallèle en fonction des entrées. L'ordre des instructions, dans la description VHDL, n'a pas d'importance.
On parle d'instructions concurrentes (contrairement aux instructions séquentielles de type **if** … **then** ... **else** …).

Exemple :



Avec instructions concurrentes

```
S2 <= C xor S1 ;
S1 <= A and B ;

Equivalent à :

S1 <= A and B ;
S2 <= C xor S1 ;
```

3 types d'instruction concurrente : booléenne (**and**, **or**, **xor**, etc…), **when….else** , **with…select…when**

exemple : multiplexeur

instruction when…else

```
entity mux is
port (A,B,C : in bit ; SEL : in bit_vector (1 to 0) ; S : out bit) ;
end mux ;
architecture arch1 of mux is
begin
    S  <= A  when (SEL= "00"  )   else
          B  when (SEL= "01"  )   else
          C  when (SEL= "10"  )   else
          '0' ;
end arch1 ;
```

instruction with…select…when

```
entity mux is
port (A,B,C : in bit ; SEL : in bit_vector (1 to 0) ; S : out bit) ;
end mux ;
architecture arch2 of mux is
begin
    with SEL select
    S  <= A  when    "00",
          B  when    "01",
          C  when    "10",
          '0' when   others;
end arch2 ;
```

5 – Instructions séquentielles

Instructions **if…then…else**

```
if      condition1   then  instruction1 ;
elsif   condition2   then  instruction2 ;
elsif   condition3   then  instruction3 ;
else    instruction4 ;
end if ;
```

Boucle **for**

```
for    paramètre    in  intervalle loop   instruction ; end loop ;
```

exemple :

```
for    i    in  0 to 3     loop
       if (A=i) then S<= B ;
       end if ;
end loop ;
```

Boucle **while**

```
while condition     loop   instructions  ;    end loop ;
```

instruction **case**

```
case  signal     is
   when     valeur1   =>     instruction1 ;
   when     valeur2   =>     instruction2 ;
   when     valeur3   =>     instruction3 ;
end case ;
```

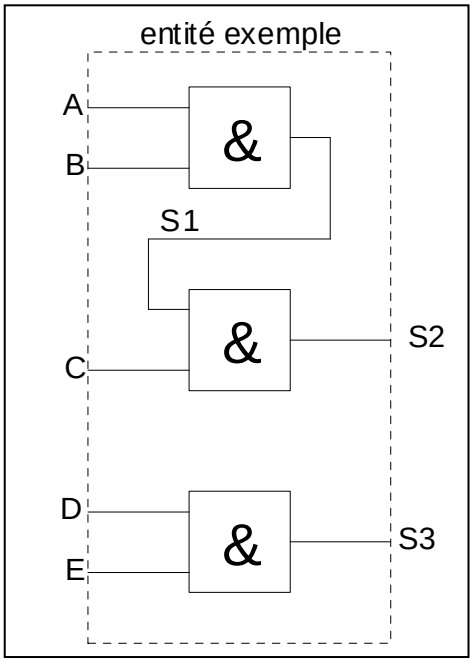
6 – Les process

L'exécution d'un process est concurrente même si les instructions qu'il contient sont séquentielles.

- Le process s'exécute à chaque changement d'état d'un des signaux déclarés sensibles.
- Les instructions à l'intérieur du process s'exécute séquentiellement
- Les modifications des signaux prennent effet à la fin du process.

Exemple

```
entity exemple  is
port (A,B,C,D,E : in bit ; S2,S3 : out bit) ;
end exemple ;

architecture arch of exemple is
signal S1 : bit ;
begin
S3     <= D and E ;
    Process (A,B,C)
    begin
        if (S1='1' and C='1')    then S2 <= '1' ; else S2 <= '0' ;
        end if ;
        if ( A='1'  and  B='1')   then S1 <= '1' ; else S1 <= '0' ;
        end if ;
    end process ;
end arch ;
```



Annexe : Les attributs

Le langage VHDL propose un outil appelé attribut que l'on peut, en quelque sorte, assimiler à des fonctions. Placé auprès d'un signal, l'attribut " 'event " va, par exemple, retourner une information vraie ou fausse (donc de type booléen) sur le fait que le signal en question ait subi ou non un événement (un changement de valeur). Le fonctionnement de cet attribut ressemble au comportement d'une fonction dont le paramètre de retour serait de type booléen, à ceci près que l'attribut 'event en question possède une notion de mémoire. En effet il détecte un changement d'état qui nécessite d'avoir au préalable mémorisé l'état précédent. Il existe en réalité deux sortes d'attributs.

- Les attributs destinés à retourner des informations concernant le comportement d'un signal (événements, dernière valeur...).
- Les attributs destinés à retourner des informations concernant les caractéristiques d'un vecteur (longueur, dimension...).

Pour ces deux sortes d'attributs, on parle d'attributs prédéfinis, au même titre que l'on parle de types prédéfinis (bit, boolean, etc.). Voici une liste non exhaustive des différents attributs prédéfinis proposés par le langage VHDL.

'high ⇒ Placé près d'un nom de tableau, il retourne la valeur du rang le plus haut (la valeur de retour est de type entier).

'low ⇒ Placé près d'un nom de tableau, il retourne la valeur du rang le plus bas (la valeur de retour est de type entier).

'left ⇒ Placé près d'un nom de vecteur ou tableau, il retourne la valeur du rang le plus à gauche (la valeur de retour est de type entier).

'right ⇒ Placé près d'un nom de vecteur ou tableau, il retourne la valeur du rang le plus à droite (la valeur de retour est de type entier).

'range ⇒ Placé près d'un signal, il retourne la valeur de l'intervalle spécifié par l'instruction range lors de la déclaration du signal ou du type utilisé.

'reverse_range ⇒ Placé près d'un signal, il retourne la valeur de l'intervalle inverse spécifié par l'instruction range lors de la déclaration du signal ou du type utilisé.

'length ⇒ Retourne X'high - X'low + 1 (sous la forme d'un entier).

'event ⇒ Vrai ou faux si le signal auquel il est associé vient de subir un changement de valeur.

'active ⇒ Vrai ou faux si le signal auquel il est associé vient d'être affecté.

'last_value ⇒ Retourne la valeur précédente du signal auquel il est associé.

'last_event ou **'last-active** ⇒ Retournent des informations de temps concernant la date d'affectation ou de transition des signaux auxquels ils sont affectés

'stable(T) ⇒ Vrai ou faux si le signal auquel il est associé n'est pas modifié pendant la durée T.

'quiet ⇒ Vrai ou faux si le signal auquel il est associé n'est pas affecté pendant la durée T.

'transaction ⇒ Retourne une information de type bit qui change d'état lorsque le signal auquel il est associé est affecté.

Exercices sur le langage VHDL
Comparateur 4 bits

Objectif : Décrire de différentes manières le fonctionnement d'un comparateur 4bits.

Structure logique : Comparateur 4 bits . Deux mots de 4 bits (A et B) sont comparés. Une sortie (AegalB) passe à 1 lorsque les 2 mots d'entrées sont égaux.

Compléter les descriptions ci-dessous.

1er cas : Description à l'aide d'opérateurs logiques :

```
entity compare is port(
A,B :   in bit_vector(0 to 3);
AegalB : out bit);
end compare;

architecture archcompare of compare is
begin
--par des équations logiques
AegalB <=
```

;

Commentaires :

- *les entrées pour les mots A et B sont notés A(0) , A(1), A(2) ,….B(3)*
- *pour les comparateurs logiques il est préférable d'utiliser des opérateurs XOR ou XNOR*
- *les commentaires sont précédés par --*

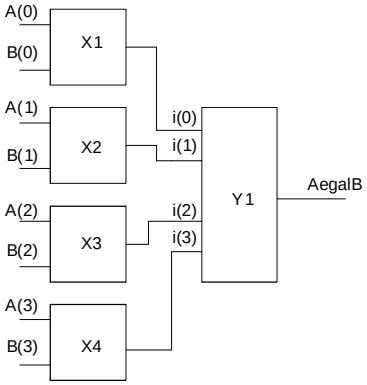
2ème cas : Description comportementale avec instruction WHEN ….ELSE :

```
entity compare is port(A,B :   in bit_vector(0 to 3);   AegalB : out bit);
end compare;

architecture archcompare of compare is
begin
```

3ème cas : Description à l'aide d'un processus et des instructions séquentielles :

```
entity compare is port( A,B :   in bit_vector(0 to 3);   AegalB : out bit);
end compare;

architecture archcompare of compare is
begin



end archcompare;
```

4ème cas : Description modulaire sans package



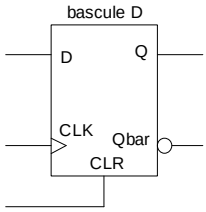
commentaires :

- *les composants de base sont décrits séparément, ils sont alors réutilisables*
- *leur utilisation nécessite une déclaration* **component** *dans la description principale*
- *dans la description principale on relit les différentes fonctions par l'instruction* **port map**
- *les signaux intermédiaires (i(0) , i(1), ;…) sont déclarés dans la zone de déclaration de l'architecture.*

```
entity XNOR2 is port(x,y: in bit; z: out bit);
end XNOR2;
architecture arch of XNOR2 is
begin
z <= not(x xor y);
end arch;

entity AND4 is port (a,b,c,d : in bit; e :out bit);
end AND4;
architecture arch of AND4 is
begin
e <= (a and b) and (c and d);
end arch;

entity compare is port( A,B : in bit_vector(0 to 3); AegalB : out bit);
end compare;

architecture archcompare of compare is
component AND4 port(a,b,c,d : in bit; e: out bit);
end component;

component XNOR2 port(x,y : in bit; z : out bit);
end component;

signal i: bit_vector(0 to 3);

begin
X1 : XNOR2 port map (A(0),B(0),i(0));
X2 : XNOR2 port map (A(1),B(1),i(1));
X3 : XNOR2 port map (A(2),B(2),i(2));
X4 : XNOR2 port map (A(3),B(3),i(3));
Y1 : AND4 port map (i(0),i(1),i(2),i(3),AegalB);
end archcompare;
```

Exercices sur le langage VHDL
Bascule D - Compteur

Bascule D



Commentaires :

- *l'attribut 'even renvoie l'information vrai si le signal auquel il est affecté vient de subir un changement de valeur.*
- *on peut aussi utiliser les instructions* **wait** et **until** :
  wait until CLK'event and CLK='1' (certains outils de synthèse impose cette syntaxe)
- le process s'exécute lorsque le signal CLK ou le signal CLR change d'état. Il n'est pas nécessaire de définir D comme signal sensible
- il faut évidemment que le composant cible accepte une entrée synchrone (D) et une entrée asynchrone (CLR) . Ce qui n'est pas le cas avec un GAL16V10, mais valable pour le GAL22V10.

```
library ieee;
use ieee.std_logic_1164.all;
entity bascule_D is
port (D, CLK, CLR : in std_logic;
Q,Qbar : out std_logic);
end bascule_D;

architecture arch_D of bascule_D is
begin
process (CLK,CLR)
begin
if CLR='1' then  Q <= '0'; Qbar <= '1' ;
elsif (CLK'event and CLK='1') then Q <= D ; Qbar <= not(D);
end if;
end process;
end arch_D;
```

Compteur 4 bits avec remise à zéro

⇒ Compléter la description ci-dessous pour obtenir un compteur synchrone de 4 bits, avec une remise à zéro asynchrone active sur niveau haut. Le comptage est effectué sur front montant d'horloge. Deux cas sont envisagés : 1ercas : Q est déclaré comme BUFFER – 2èmecas : Q déclaré comme sortie, et on utilise un signal interne déclaré dans l'architecture.
Remarque : Pour utiliser des instructions arithmétiques (comme l'addition de nombre entier), il faut prévenir le compilateur d'utiliser la librairie std_arith.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity COMPT4B is
port (CLK, CLR : in std_logic;
Q : buffer std_logic_vector(3 downto 0));
end COMPT4B;

architecture arch of COMPT4B is
begin
process (




end process;
end arch;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity COMPT4B is
port (CLK, CLR : in std_logic;
Q : out std_logic_vector(3 downto 0));
end COMPT4B;

architecture arch of COMPT4B is
signal X : std_logic_vector (3 downto 0) ;
begin
process (




end process;
Q <= X ;
end arch;
```

COMPTEUR /DECOMPTEUR BCD 4 BITS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity CD4B is
port (CLK, CLR,UD : in std_logic; Q : buffer std_logic_vector(3 downto 0));
end CD4B;

architecture arch of CD4B is
begin
process (CLK,CLR)
begin
if CLR='1' then Q <= "0000";

elsif (CLK'event and CLK='1') then
    if (Q="1001" and UD='1') then Q <= "0000";
    elsif (Q="0000" and UD='0') then Q <= "1001";
    elsif (UD='1')then Q <= Q + 1 ;else Q <= Q-1;
    end if;
end if;
end process;
end arch;
```