Concepts intermédiaires de VHDL

- Objets: constantes, signaux, variables
- Structure de contrôle
- Attributs de signaux

Les objets en VHDL

- Constant (et generic): peut contenir une valeur unique qui ne change pas. Un objet generic est une constante spéciale permettant d'appliquer un paramètre à une entité lors de son instanciation.

- Signal et port: peuvent contenir une liste de valeurs dans le temps. Un objet signal correspond, en général, à un fil d'un circuit. Un port d'une entité est implicitement un signal pour cette entité.

- Variable: peut contenir une valeur temporaire. Les objets variables sont utiles pour stocker des valeurs intermédiaires dans les calculs.

- File: un objet dans lequel on peut lire et écrire des valeurs, et qui correspond à un fichier du système d'exploitation.

Lors de la déclaration d'un objet, on spécifie sa catégorie, son identificateur et son type.
On peut aussi lui assigner une valeur initiale.

Constant et generic

- Constant (et generic): peut contenir une valeur unique qui ne change pas.
- Un objet generic est une constante spéciale permettant d'appliquer un paramètre à une entité lors de son instanciation.
- Un objet constant peut être défini pour une architecture ou un process.

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

entity vote is
generic (
            w: positive:=4
           );
port (
       lesvotes: in std_logic_vector(w-1 downto 0);
       approbation: out std_logic
      );
end vote;
```

```
architecture comportementale of vote is
constant nMin: natural := w/2;
begin
Process(lesvotes)
variable compte: natural range 0 to lesvotes'slength;
begin
compte:=0;
for k in lesvotes'srange loop
    if lesvotes(k)='1' then
       compte:=compte+1;
    end if;
end loop;
    if compte > nMin then
       approbation <= '1';
    else
        approbation <= '0';
    end if;
End process;
End comportementale;
```

Signal et port

- Un signal est déclaré dans la partie déclarative d'une architecture et est visible partout dans celle-ci.
- Un signal représente un fil ou un groupe de fils dans un circuit.
- Un port d'une entité est implicitement un signal dans toutes les architectures de cette entité.
  - un port in est un signal qui ne peut pas être écrit.
  - un port out est un signal qui ne peut pas être lu.
- Pour assigner une valeur à un port, on utilise l'opérateur <=.

```
entity vote is
generic (
            w: positive:=4
           );
port (
      lesvotes: in std_logic_vector(w-1 downto 0);
       approbation: out std_logic
       );
end vote;

architecture flotdedonnee of vote is
 signal A,B,C,D : std_logic;

Begin
  (A,B,C,D) <= lesvotes;
  approbation <=
                    (not( A) and B and C and D) or ( A and not( B) and C and D)
                 or (A and B and not© and D) or (A and B and C and not(D))
                  or (A and B and C and D) ;
End flotdedonnee;
```

Les déclarations et les assignations des signaux vecteurs

gauche droite

```
signal a : std_logic_vector (0 to 7);
signal b : std_logic_vector (7 downto 0);

a <= "11110000";
b <= "11110000";
```

MSB (toujours à gauche) LSB (à droite)

a(0)	a(1)	a(2)	a(3)	a(4)	a(5)	a(6)	a(7)
1	1	1	1	0	0	0	0

MSB (toujours à gauche) LSB (à droite)

b(7)	b(6)	b(5)	b(4)	b(3)	b(2)	b(1)	b(0)
1	1	1	1	0	0	0	0

La notation downto est la plus conventionnelle pour les applications de synthèse.

variable

- Une variable peut contenir une valeur temporaire.
- Les variables ne peuvent être utilisées qu'à l'intérieur des processus.
- Les objets variables sont utiles pour stocker des valeurs intermédiaires dans les calculs.
- Pour assigner une valeur à une variable, on utilise l'opérateur :=

```
architecture comportementale of vote is
constant nMin: natural := w/2;
begin
process(lesvotes)
variable compte: natural range 0 to lesvotes'slength;
begin
compte:=0;
for k in lesvotes'srange loop
    if lesvotes(k)='1' then
       compte:=compte+1;
    end if;
end loop;
    if compte > nMin then
       approbation <= '1';
    else
        approbation <= '0';
    end if;
End process;
End comportementale;
```

Structures de contrôle

Les structures (for, while, if-then-else et case) ne s'utilisent qu'à l'intérieur d'un processus.

- les instructions de test (if, case)
- les boucles (loop, for loop, while loop)

L’instruction IF

L' instruction IF repose sur le test d'une condition qui génère un booléen.

Syntaxe:

```
if condition1 then instruction;
 [elsif condition2 then instruction;]
 ...
 [else instruction;]
 end if;
```

N.B:

Pour coder un processus combinatoire, l'utilisation du ELSE est obligatoire de façon à traiter toutes les combinaisons possibles (sinon il y a mémorisation donc c'est de la logique séquentielle).

Exemple

```
begin
process(EN,D)
begin
if (EN = '1') then
Q <= D;
end if;
end process;
```

≠

```
begin
process(EN,D)
begin
if (EN = '1') then
Q <= D;
else Q <= 0;
end if;
end process;
```

Exemple 1:

```
architecture comportementale of vote is
constant nMin: natural := w/2;
begin
Process(lesvotes)
variable compte: natural range 0 to lesvotes'slength;
begin
compte:=0;
for k in lesvotes'srange loop
    if lesvotes(k)='1' then
        compte:=compte+1;
    end if;
end loop;
    if compte > nMin then
        approbation <= '1';
    else
         approbation <= '0';
    end if;
End process;
End comportementale;
```

Exemple 2:

```
entity thermostat  is
port
( temp_desir, temp_reel : in integer;
   chauffage_on : out boolean );
end entity thermostat;

architecture exemple of thermostat is
begin
control : process (temp_desir, temp_reel)
begin
If temp_reel < temp_desir   then
          chauffage_on <= true;
elsif temp_reel > temp_desir   then
          chauffage_on <= false;
end if;
end process control;
end architecture exemple;
```

Sensitivity list
(liste de sensibilité)

La liste de sensibilité est l'ensemble des signaux auxquels le processus est sensible.

Lorsqu'un signal quelconque de cette liste change de valeur, le processus s'exécute (séquentiellement).

Une fois la dernière instruction est exécutée, le processus est suspendu en attendant un autre changement d'un signal de la liste de sensibilité.

L’instruction CASE

L' instructions CASE repose sur le test d'un signal ou d'une variable. En fonction de la valeur, une instruction spécifique est exécutée.

Syntaxe:

```
case var/signal is
when valeur_1 =>Instruction_1;
                Instruction_2;
                 …;
 when valeur_2 => Instruction_3;
                 Instruction_4;
                 …;
      …
when others => Instruction_n;
                Instruction_m;
                 …;
end case;
```

```
case A is
when -7 => B := 10;
                C :='0‘;
 when -3 => B :=15;
                C :='0';
when others => B :=2;
                      C :='1';
end case;
```

Même remarque que pour le IF. (when others est obligatoire)

Les boucles

- Boucle simple :

[label :] **loop**
instructions;
end loop[label];

- Boucle while :

[label :] **while expression loop**
instructions;
end loop[label];

- Boucle for :

[label :] **for var in exp1 to exp2 loop**
instructions;
end loop[label];

Remarques:

- Les indices de boucles ne sont pas à déclarer.
- Les étiquettes de boucles sont optionnelles.
- L'exécution peut être altérée avec les clauses next et exit.

Exemples:

Exemple 1:

```
signal Clock : BIT := '0';
...
Clk_1: process (Clock)
begin
  L1: loop
      Clock <= not Clock after 5 ns;
      end loop L1;
end process Clk_1;
```

Le process (qui est un générateur de signal d'horloge) boucle indéfiniment.

Exemple 2:

```
L2: loop
    A:= A+1;
    exit L2 when A > 10;
    end loop L2;
```

Exemple 3:

```
Shift_3: process (Input_X)
variable i : POSITIVE := 1;
begin
  L3: while i <= 8 loop
        Output_X(i) <= Input_X(i+8) after 5 ns;
        i := i + 1;
      end loop L3;
end process Shift_3;
```

Exemple 4:

```
Shift_4: process (Input_X)
begin
  L4: for count_value in 1 to 8 loop
       Output_X(count_value) <= Input_X(count_value + 8) after 5 ns;
      end loop L4;
end process Shift_4;
```

Exemple 5:

```
Shift_5: process (Input_X)
subtype Range_Type is POSITIVE range 1 to 8;
begin
  L5: for count_value in Range_Type loop
       Output_X(count_value) <= Input_X(count_value +8) after 5 ns;
      end loop L5;
end process Shift_5;
```

Autre Exemple:

```
architecture comportementale of
vote is
constant nMin: natural := w/2;
begin
Process(lesvotes)
variable compte: natural range 0
to lesvotes'slength;
begin
compte:=0;
for k in lesvotes'srange loop
    if lesvotes(k)='1' then
        compte:=compte+1;
    end if;
end loop;
    if compte > nMin then
        approbation <= '1';
    else
        approbation <= '0';
    end if;
End process;
End comportementale;
```

Le type " time"

- VHDL possède un type prédéfini appelé "time“, utilisé pour représenter les temps dans les simulations et les délais/retards "delays“.

exemple:

5 ns, 22 us, 471.3 ms

- Les identificateurs d’unités valides sont:

  fs, ps, ns, us, ms, sec, min, hr

- Les opérations arithmétiques et la fonction "abs" peuvent s’appliquer sur les valeurs "temps" :

  18 ns / 2.0 = 9 ns, 33 ns / 22 ps = 1500

Différences entre variables et signaux

- Les variables sont toujours locales à un processus: il n'y a pas de variables globales.

- Les signaux peuvent être déclarés n'importe où dans le programme, sauf à l'intérieur d'un processus. S'ils sont déclarés au début de l'architecture, ils sont communs à toute l'architecture.

- Contrairement aux signaux, les variables sont mises à jour de façon instantanée, au moment de l'affectation, sans retard possible.

- Les instructions à l'intérieur d'un processus sont toujours exécutées en séquence, y compris les affectations de signaux. Toutefois, les signaux sont mis à jour en même temps, au moment d'un wait.
- En absence d'un wait, le processus est exécuté sans arrêt, sans que les signaux soient mis à jour.

C'est à dire: lorsqu'on exécute les instructions à l'intérieur d'un processus, on utilise les valeurs initiales des signaux pour tous les calculs et, à la fin, on met à jour les nouvelles valeurs.

- En absence d'un autre wait, il faut au moins

wait for 0 ns;

à la fin d'un processus pour s'assurer de la mise à jour des signaux.

Toutefois, cette phrase n'est pas acceptée par les synthétiseurs

Exemple avec des variables

```
Entity ex_var  is
End ex_var;

Architecture var of ex_var is
Signal trigger, sum : integer := 0;
begin
process
Variable var1 : integer := 1;
Variable var2 : integer := 2;
Variable var3 : integer := 3;
begin
wait on trigger;
var1 := var2 + var3;
var2 := var1;
var3 := var2;
sum <= var1 + var2 + var3;
end process;
End var;
```

Si trigger change à t=10, alors:
var1=5, var2=5, var3=5
et à t=10+Δ, sum=15

Exemple avec des signaux

```
Entity ex_sig is
End ex_sig;

Architecture sig of ex_sig is
Signal trigger, sum : integer := 0;
Signal sig1 :integer := 1;
Signal sig2 : integer := 2;
Signal sig3 : integer := 3;
begin
process
begin
wait on trigger;
sig1 <= sig2 + sig3;
sig2 <= sig1;
sig3 <= sig2;
sum <= sig1 + sig2 + sig3;
end process;
End sig
```

Si trigger change à t=10, tous les signaux sont mis à jour à t=10+Δ:
sig1=5, sig2=1, sig3=2
Et Sum =6

Attributs en VHDL

- En VHDL, les attributs permettent d'obtenir de l'information à propos des signaux, variables, etc.

- VHDL inclut des attributs prédéfinis, et il est possible d'en définir de nouveaux.

- Les attributs s'utilisent dans des expressions.

Attributs	Définitions - informations de retour
'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.

```
…
Process(lesvotes)
variable compte: natural range 0 to lesvotes'length;
begin
compte:=0;
for k in lesvotes'range loop
    …
```

Exemple :

type BIT_VECTEUR is array (INTEGER range <>) of BIT;

variable MON_VECTEUR : BIT_VECTEUR (5 downto -5);

Attribut	**Valeur renvoyée**
MON_VECTEUR'left	5
MON_VECTEUR'right	–5
MON_VECTEUR'high	5
MON_VECTEUR'low	–5
MON_VECTEUR'length	11
MON_VECTEUR'range	(5 downto –5)
MON_VECTEUR'reverse_range	(–5 to 5)