Chap 2. Les instructions du mode séquentiel.

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 (opposition aux instructions concurrentes présentées au chp 1).

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**.

Syntaxe :

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

2 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***.

3 L'assignation conditionnelle :

L'instruction conditionnelle sont exécutées en séquentielles sont *internes* aux processus, aux procédures et aux fonctions.

Syntaxe :

if condition then
instructions
[elsif condition then instructions]
[else instructions]
end if ;

Il est possible d'imbriquer plusieurs boucles IF les unes dans les autres. (exemple 2)

```
IF ... THEN
    IF ... THEN
    ELSIF ... THEN
    END IF;
ELSE
END IF;
```

Remarque: ne pas confondre => (implique) et <= (affecte).
Ces deux instructions vont être utilisées dans la suite du

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
-- Note : CLK'event Détection du front d'un signal d'horloge

if (CLK'event and CLK ='1') then
S <= D;
end if;
end process PRO_BASCULED;
end DESCRIPTION;
```

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;
Q : 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
   Q<= ‘0’;
  elsif (SET =’1’)then
   Q <= ‘1’;
  else
   Q <= D;
 end if;
end if;
end process PRO_BASCULEDSRS;
end DESCRIPTION;
```

Example N° 3 “Compteur simple” :

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*.

```
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;
```

4 L’assignation sélective

L’instruction sélecative sont exécutées en séquentielles sont *internes* aux processus, aux procédures et aux fonctions.

Syntaxe

case signal_de_slection is
when valeur_de_sélection => instructions
[when others => instructions]
end case;

Exemple

Additionneur de deux nombres à deux bits:

```
library ieee;
use ieee.std_logic_1164.all;
entity AddLog is
port (x1, x0, y1, y0 : in std_logic;
s2, s1, s0 : out std_logic);
end AddLog;

architecture table of AddLog is
signal entree: std_logic_vector(3 downto 0);
begin
entree <= x1 & x0 & y1 & y0;
process (entree)
begin
case entree is
when "0000" => s2 <= '0';
            s1 <= '0';
            s0 <= '0';
when "0001" => s2 <= '0';
            s1 <= '0';
            s0 <= '1';
.
.
.
.
when "1111" => s2 <= '1';
            s1 <= '1';
            s0 <= '0';
when others => null;
end case;
end process;
end table;
```

5 Les boucles :

Il y a deux types de boucles en VHDL : les boucles while et les boucles for. La boucle for parcourt un vecteur d'indices et effectue à chaque pas toutes les instructions délimitées par l'instruction end.

Syntaxe « for » :

for *parametre* in *minimum* to *maximum* loop

séquence d'instructions

end loop;

Ou :

for *parametre* in *maximum* downto *minimum* loop

séquence d'instructions

end loop [*etiquette*] ;

Syntaxe " while " :

while *condition* loop

séquence d'instructions

end loop [*etiquette*] ;

Note : les instruction next et exit sont des instruction plus utiliser dans les boucles

Syntaxe :

next when *condition*;

Permet de passer à l'itération suivante d'une boucle.

exit when *condition*;

Permet de provoquer une sortie de boucle.

Exemple code

```
test : process
variable i, k : integer;
variable l : line;
begin
  for i in 2 downTo 0 loop
     for k in 2 downTo 0 loop
          next when (k=1);
          write (l, STRING'(" i = "));
         write (l, i);
         write (l, STRING'(" k = "));
         write (l, k);
         writeLine (output, l);
      end loop;
   end loop;
wait;
end process test ;
```

Resultats sortie simulation :

```
# i = 2 k = 2
# i = 2 k = 0
# i = 1 k = 2
# i = 1 k = 0
# i = 0 k = 2
# i = 0 k = 0
```

Programme precedent
Ligne : next when (k=1); Remplacee par : exit when (k=1);
Resultats sortie simulation :
i = 2 k = 2
i = 1 k = 2
i = 0 k = 2

6 Instruction generate

Une instruction generate permet de convertir les instructions sont exécutées séquentiellement a l'exécution concurrentes ou permet de dupliquer un bloc d'instructions concurrentes un certain nombre de fois, ou de créer un tel bloc si une condition est vérifiée.

Syntaxe :

-- structure répétitive :

etiquette : for *variable* in *debut* to *fin* generate

instructions concurrentes

end generate [*etiquette*] ;

ou :

-- structure conditionnelle :

etiquette : if *condition* generate

instructions concurrentes

end generate [*etiquette*] ;

`Example 1`

Dans l´exemple précédent, les quatre instanciations de bascules peuvent ainsi être écrit plus simplement en considérant une variable de boucle i variant de 0 à 3. Cette variable rentre dans la définition des signaux nécessaires au câblage mais aussi à la génération automatique des étiquettes d´instanciation des composants.

Exemple :

```
gen_for : for i in 0 to 3 generate
```

```
end generate gen_for ;
```

7 Instruction "wait"

L'instruction "wait" provoque la suspension du process (ou de la procédure):

Wait [on nomSignal1, nomSignal2…] [until expressionBooléenne] [for expressionTemps];

L'instruction wait suspend le sous-programme ou le process tant qu'aucun événement ne survient sur l'un des signaux de la liste de sensibilité (nomSignal1, nomSignal2…) et que l'expression booléenne n'est pas évaluée à TRUE. Tous les signaux de la liste de sensibilité doivent être accessibles en lecture.

Exemple 1 :
wait ; -- attendre a jamais

Exemple 2 :
wait for 200 ns – attendre 200 ns

Exemple 3 :
--expressions strictement équivalentes
wait on h until h = '0'; -- attendre un transition sur le signal "h" vers l'etat '0'
wait until h = '0';

Exemple 4 :
wait on h until nClear='1'; -- attendre tant que un evenement sur le signal "h" ne coïncide pas avec une valeur sur le signal nClear egale a '1'

Exemple 5 :
une bascule D en utilisant l'instruction wait :
pBasc : process
begin
wait until (h'event and h = '1');
q <= d;
end process pBasc;