Proiectare comportamentală (secvenţială)

Nu întotdeauna este posibil să descriem direct comportarea unui circuit logic prin utilizarea unei instrucţiuni concurente. Pentru majoritatea descrierilor comportamentale este necesară folosirea unor elemente suplimentare de limbaj.

Instrucţiunile secvenţiale se utilizează în procese, funcţii şi proceduri.

1. Un element de descriere comportamentală cheie este **procesul**. Un proces (*process*) este o colecţie de instrucţiuni secvenţiale care se execută in paralel cu alte instrucţiuni concurente şi cu alte procese.

Instrucţiunea *process* are următoarea sintaxă:

process (nume-semnal, nume-semnal, …, nume-semnal)
declaraţii de tipuri
declaraţii de variabile
declaraţii de constante
definiţii de funcţii
definiţii de proceduri
begin
instrucţiune-secvenţială
….
instrucţiune-secvenţială
end process;

Intr-un proces sunt vizibile numai tipurile, semnalele, constantele, funcţiile şi procedurile care fac parte din aceeaşi arhitectură cu procesul, insă toate aceste elemente cu excepţia semnalelor pot fi definite şi local in proces.

In cadrul unui proces, variabilele au rolul de a păstra stări, ele nu sunt vizibile in afara procesului. În funcţie de modul de utilizare, o variabilă poate genera sau nu un semnal correspondent într-o implementare fizică a circuitului modelat.

Sintaxa *VHDL* pentru definirea unei variabile :

variable *nume-variabile : tip-variabile ;*

Semnalele care se află în paranteze alături de cuvântul cheie *process,* determină dacă procesul rulează sau dacă este suspendat, această listă de semnale poartă numele de listă de sensibilitate.

Presupunem că un proces este suspendat iniţial. Dacă unul dintre semnalele aflate în lista de sensibilitate îşi schimbă valoarea, procesul intră în execuţie, începând cu prima instrucţiune şi până la ultima. Dacă valoarea oricărui semnal din lista de sensibilitate se modifică datorită execuţiei procesului, acesta se execută din

nou. Astfel procesul va rula până în momentul în care nici unul dintre semnale nu-şi mai schimbă valoarea. Toate evenimentele din cadrul unui proces au loc, în cadrul unei simulări într-un "timp de simulare" egal cu zero.

Un proces descris corect în *VHDL* se va suspenda după una sau mai multe rulări, trebuie să se evite modelarea proceselor care nu se suspendă niciodată. Lista de sensibilitate este opţională, procesele care nu au listă de sensibilitate încep să ruleze în cadrul simulării la momentul zero, aceste procese sunt utile pentru modelarea *test bench*-urilor.

Limbajul *VHDL* are câteva tipuri de instrucţiuni secvenţiale.

1. ***Atribuirea secvenţială de semnal*** (sequential signal-assignment):

nume-semnal **<=** *expresie ;*

Instrucţiunea are aceeaşi sintaxă ca şi varianta concurentă, dar se utilizează doar în interiorul procesului, şi nu în arhitectură.

2. ***Atribuirea secvenţială de variabilă***:

nume-variabilă **: =** *expresie ;*

Exemplu. Arhitectura detectorului de numere prime rescrisă ca proces.

```
architecture   prime5_arch of prime is
        begin
           process (N)
                variable T1, T2,T3,T4 :  STD_LOGIC;
            begin
                T1  :=  not N(3) and N(0) ;
                T2  := not N(3) and not  N(2) and N(1) ;
                T3  := not N(2) and N(1) and N(0) ;
                T4  := N(2) and not N(1) and N(0) ;
                F    <= T1 or T2 or T3 or T4;
            end process ;
  end prime5_arch;
```

În cadrul acestei arhitecturi (*prime5_arch*) există doar o singură instrucţiune concurentă, aceasta este instrucţiunea *process.* În lista de sensibilitate a procesului apare intrarea *N*, iar în cadrul procesului se definesc variabile, definirea de semnale nu este permisă. Ieşirile porţilor AND trebuie definite ca variabile, deoarece definiţiile de semnale nu sunt acceptate într-un proces.

1. Instrucţiunea if

Sintaxa:

a) **if** expresie-booleană **then** instrucţiune secvenţială
end if;

Expresia booleană este testată şi, dacă ea este adevărată (TRUE), se va executa o instrucţiune secvenţială.

b) **if** expresie-booleană **then** instrucţiune secvenţială ;
else instrucţiune secvenţială ;
end if;

În această formă se mai adaugă şi clauza *else* urmată de o altă instrucţiune secvenţială care se execută în cazul în care expresia testată este falsă (nu se verifică).

c) **if** expresie-booleană **then** instrucţiune secvenţială ;
elsif expresie-booleană **then** instrucţiune secvenţială ;
. . .
elsif expresie-booleană **then** instrucţiune secvenţială ;
end if;

d) **if** expresie-booleană **then** instrucţiune secvenţială ;
elsif expresie-booleană **then** instrucţiune secvenţială ;
. . .
elsif expresie-booleană **then** instrucţiune secvenţială ;
else instrucţiune secvenţială **;**
end if;

În aceste forme se introduce cuvântul cheie *elsif*. O instrucţiune secvenţială condiţionată de clauza *elsif* se execută dacă expresia booleană căreia i s-a aplicat aceasta este adevărată şi toate expresiile precedente sunt false. Instrucţiunea secvenţială corespunzătoare clauzei finale *else* se execută dacă toate expresiile booleene precedente au fost false.

Exemplu. Codul VHDL pentru detectorului de numere prime folosindu-se instrucţiunea *if*.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity prime is
        port ( N: in STD_LOGIC_VECTOR (3 downto 0);
                F: out STD_LOGIC) ;
end prime;
architecture   prime6_arch of prime is
```

```
begin
  process (N)
     variable NI : INTEGER;
       begin
          NI  :=  to_integer(unsigned (N));
  if NI=1  or NI=2  or   NI=3 or NI=5  or NI=7  or NI=11  or NI=13  then F<='1' ;
    else F <=        '0' ;
  end if ;
  end process ;
end prime6_arch;
```

În acest caz se foloseşte o variabilă *NI* pentru a păstra valoarea întreagă (de tip integer) rezultată în urma convertiri intrării *N*.

2. ***Instrucţiunea case***

Se foloseşte când trebuie aleasă o singură alternativă din multitudinea de alternative oferite de valorile pe care le poate lua un semnal sau o expresie.

Sintaxa instrucţiunii *case:*

```
case expresie  is
        when opţiuni  => instrucţiuni-secvenţiale ;
        . . .
        when opţiuni  => instrucţiuni-secvenţiale ;
        end case ;
```

Această instrucţiune evaluează o expresie dată, alege valoarea care se potriveşte din una din opţiuni (*choices*) şi execută instrucţiunea secvenţială corespunzătoare.

Opţiunile (*choices*) sunt reprezentate de o singură valoare sau de un set de valori separate prin bare verticale (|), aceste opţiuni trebuie să se excludă una pe cealaltă şi trebuie să includă toate valorile posibile ale expresiei evaluate, altfel se foloseşte clauza *others*.

Exemplu. Arhitectura pentru detectorul de numere prime, folosind instrucţiunea *case*.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity prime is
        port ( N: in STD_LOGIC_VECTOR (3 downto 0);
             F: out STD_LOGIC) ;
end prime;
```

```
architecture   prime7_arch of prime is
begin
          process (N)
            begin
               case to_integer(unsigned (N))   is
                  when 1  => F <= '1' ;
                  when 2  => F <= '1' ;
    when 3 | 5 | 7 |  11 |  13    => F <= '1' ;
                   when others  =>  F <= '0' ;
               end case ;
          end process ;
 end prime7_arch;
```

3. **Instrucţiunile *for loop, while loop* şi *loop***

Se utilizează pentru a indeplini mai multe iteraţii în procese, funcţii şi proceduri.

Sintaxa:

a) **for** identificator **in** domeniu **loop**
instrucţiune-secvenţială;
. . .
instrucţiune-secvenţială;
end loop;

Variabila *identificator* se declară implicit şi este de acelaşi tip cu domeniul (*range*), pe care îl parcurge de la stânga la dreapta câte o valoare la fiecare iteraţie.

Domeniul (range) poate fi:

5 **downto** 0
0 **to** 5

În limbajul VHDL, instrucțiunea ***loop*** are mai multe particularități, care o deosebesc de alte limbaje de programare. În unele limbaje de programare, identificatorului din interiorul ciclului i se poate atribui o valoare oarecare. În limbajul VHDL acest lucru nu este posibil.

O altă particularitate este faptul că identificatorul este declarat doar în interiorul instrucțiunii ***for loop*** și nu este necesară declararea explicită a lui în proces, funcție sau procedură. Din această cauză identificatorul nu poate fi utilizat în calitate de valoare returnată în funcții și proceduri. Dacă în interiorul procedurii, dar în afara ciclului ***loop***, există o altă variabilă cu acelaşi nume, ea este tratată ca o variabilă aparte.

process (i)

```
begin
  ...
 x <= i+1;   -- x este un semnal
 …
  for  i   in 1 to 10   loop
   q(i) := a;   -- q este o variabilă
  end loop;
 …
end process;
```

Exemplu. Un generator al bitului de paritate pentru un cuvânt binar de 10 biţi, utilizând instrucţiunea **for-loop.**

```
library ieee;
use ieee.std_logic_1164.all;
entity   parity10  is
  port  (D: in std_logic_vector(0 to 9);
          ODD: out std_logic);
          constant WIDTH: integer := 10;
end parity10;

architecture par of parity10 is
  begin
    process  (D)
      variable x: Boolean;
        begin
         x := false;
          for i in 0 to D'length - 1 loop  -- se utilizează atributul length
            if D(i) = '1' then
              x := not x;
           end if;
         end loop;
            if x  then
              ODD <= '1';
            else
              ODD <= '0';
            end if;
 end process;
 end par;
```

Programul va genera bitul de imparitate, dacă variabila x va primi iniţial valoarea TRUE.

b) **while** expresie-booleană **loop**
instrucţiune-secvenţială;
. . .
instrucţiune-secvenţială;
end loop;

În acest tip de instrucţiune expresia booleană este testată înaintea de fiecare iteraţie, iar bucla se execută numai dacă valoarea expresiei este adevărată.

Instrucţiunea se foloseşte pentru descrierea proceselor secvenţiale în programe speciale de generare a stimulilor de intrare, numite test-bench. Nu se recomandă utilizarea ei în programele de descriere a circuitelor, deoarece compilatorul nu întotdeauna poate sintetiza instrucţiunea.

Exemplu. Generator al semnalului de ceas, care poate fi utilizat într-un test-bench. Semnalul de ceas se va genera până când fanioanele ***error_flag*** sau ***done*** nu vor fi egale cu 1.

```
process
  begin
    while error_flag /= ‘1’ and done /= '1’ loop
        Clock <= not Clock;
        wait for CLK_PERIOD/2;
  end loop;
end process;
```

c) loop **--** ciclu infinit, se utilizează cu instr. next sau exit
instrucţiune-secvenţială;
. . .
instrucţiune-secvenţială;
end loop;

La fel ca şi instrucţiunea while-loop nu poate fi sintetizată.

4. Instrucţiunile *exit* şi *next*

Sunt instrucţiuni secvenţiale care se pot executa în cadrul unei instrucţiuni *loop.*

Instrucţiunea ***exit*** **t**ransferă comanda către instrucţiunea ce urmează imediat după sfârşitul buclei. Poate fi utilizată pentru toate tipurile de instrucţiuni ***loop.***

Poate fi:

- necondiţională: ***exit***
- condiţională: ***exit when***

Sintaxa:

exit [label] [when condition] ;

Exemplu. Generator al semnalului de ceas, care poate fi utilizat într-un test-bench în care se foloseşte instrucţiunea ***exit***.

```
process
  begin
    loop
       Clock <= not Clock;
       wait for CLK_PERIOD/2;
       if done = '1' or error_flag = '1' then
          exit;
       end if;
    end loop;
end process;
```

Exemplu. Utilizarea instrucţiunii condiţionale ***exit***.

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

Instrucţiunea ***next*** face ca toate instrucţiunile rămase până la sfârşitul buclei să fie ocolite şi să înceapă o nouă execuţie a buclei.

Sintaxa:

next [label] [when condition] ;

Exemplu. Un numărător de unităţi pentru cuvinte binare de 16 biţi.

```
library ieee;
use ieee.numeric_bit.all;
entity count_ones is
```

```
 port (v: in bit_vector (15 downto 0);
 count: out signed (3 downto 0));
end count_ones;
architecture functional of count_ones is
begin
process (v)
 variable result: signed (3 downto 0);
 begin
 result := (others => '0');
 for i in v'range loop
 next when v(i) = '0';
 result := result + 1;
 end loop;
 count <= result;
 end process;
end functional;
```

7. Instrucţiunea *wait*

În locul unei liste de sensibilitate, un proces poate conţine o instrucţiune wait. Utilizarea unei instrucţiuni wait are două scopuri:

• Suspendarea execuţiei unui proces;

• Specificarea condiţiei care va determina activarea procesului suspendat.

La întâlnirea unei instrucţiuni wait, procesul în care apare această instrucţiune este suspendat. Atunci când se îndeplineşte condiţia specificată în cadrul instrucţiunii wait, procesul este activat şi se execută instrucţiunile acestuia până când se întâlneşte din nou instrucţiunea wait.

Limbajul VHDL permite ca un proces să conţină mai multe instrucţiuni wait. Atunci când se utilizează pentru modelarea logicii combinaţionale în vederea sintezei, un proces poate conţine însă o singură instrucţiune wait.

Dacă un proces conţine o instrucţiune wait, acesta nu poate conţine o listă de sensibilitate. Formele instrucţiunii wait:

- wait on *listă_de_sensibilitate*. Procesul se execută când apare o modificare a valorii semnallor din lista de sensibilitate; Exemplu: ***wait on*** a,b;

- wait until *expresie_condiţională*; Procesul se suspendă până când condiţia specificată devine adevărată datorită modificării unuia din semnalele care apar în expresia condiţională. Dacă nici un semnal din această expresie nu se modifică, procesul nu va fi activat, chiar dacă expresia condiţională este adevărată. Exemplu: ***wait until*** ((x*y)<100);

- wait for *expresie_de_timp.* Permite suspendarea execuţiei unui proces pentru un timp specificat, de exemplu: **wait for** 10 ns;

Exemplele următoare prezintă mai multe forme ale instrucţiunii wait until:

wait until *semnal* **=** *valoare*;

wait until *semnal***'event and** *semnal* **=** *valoare*;

wait until not *semnal***'stable and** *semnal* **=** *valoare*;

unde *semnal* este numele unui semnal, iar *valoare* este valoarea care se testează. Dacă semnalul este de tip bit, atunci pentru valoarea '1' se aşteaptă frontul crescător al semnalului, iar pentru '0' frontul descrescător.

Instrucţiunea wait until se poate utiliza pentru implementarea unei funcţionări sincrone. În mod obişnuit, semnalul testat este un semnal de ceas. De exemplu, aşteptarea frontului crescător al unui semnal de ceas se poate exprima în următoarele moduri:

wait until clk = '1';

wait until clk'**event and** clk = '1';

Pentru descrierile destinate sintezei, instrucţiunea wait until trebuie să fie prima din cadrul procesului. Din această cauză, logica sincronă descrisă cu o instrucţiune wait until nu poate fi resetată în mod asincron.

Exemplu. Implementarea unui circuit cu resetare sincronă, utilizând instrucţiunile ***wait*** şi ***loop.*** Semnalul RESET trebuie verificat imediat după fiecare instrucţiune ***wait***.

```
process
  begin
    RESET_LOOP: loop
      wait until CLOCK'event and CLOCK = '1';
      next RESET_LOOP when (RESET = '1');
        X <= A;
      wait until CLOCK'event and CLOCK = '1';
      next RESET_LOOP when (RESET = '1');
        Y <= B;
  end loop RESET_LOOP;
end process;
```

Exemplu. Utilizarea procesului cu listă de sensibilitate şi fără listă de sensibilitate, dar cu instrucţiunea wait.

```
 PROCESS (clk)
   VARIABLE last_clk : std_logic := 'X';
 BEGIN
   IF (clk /= last_clk ) AND (clk = '1') THEN
    q <= din AFTER 25 ns;
 END IF;

 last_clk := clk;

 END PROCESS;
PROCESS
  VARIABLE last_clk : std_logic := 'X';
BEGIN
  IF (clk /= last_clk ) AND (clk = '1') THEN
   q <= din AFTER 25 ns;
  END IF;

  last_clk := clk;

  WAIT ON clk;
END PROCESS;
```

Instrucțiunea wait este situată la sfărșitul procesului pentru a permite instrucțiunilor din proces de fi executate o dată.