Descrierea circuitelor secvenţiale în VHDL

Semnalele generate la ieşirile unui circuit secvenţial depind atât de semnalele de intrare, cât şi de starea circuitului.

Starea prezentă a circuitului este determinată de o stare anterioară şi de valorile semnalelor de intrare.

Circuitele secvenţiale conţin elemente de memorare şi circuite combinaţionale.

Efectul de memorare se datorează unor legături inverse (bucle de reacţie) prezente în schemele logice ale acestor circuite.

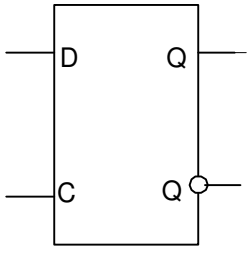
În cazul circuitelor secvenţiale sincrone, modificarea stării se poate realiza la momente bine definite de timp sub controlul unui semnal de ceas. În cazul circuitelor secvenţiale asincrone, modificarea stării poate fi cauzată de schimbarea aleatoare în timp a valorii unui semnal de intrare.

Comportamentul unui circuit asincron este mai puţin sigur, evoluţia stării fiind influenţată şi de timpii de întârziere ai componentelor circuitului. Trecerea între două stări stabile se poate realiza printr-o succesiune de stări instabile, aleatoare.

Circuitele secvenţiale sincrone sunt mai fiabile şi au un comportament predictiv. Toate elementele de memorare ale unui circuit sincron îşi modifică simultan starea, ceea ce elimină apariţia unor stări intermediare instabile. Prin testarea semnalelor de intrare la momente bine definite de timp se reduce influenţa întârzierilor şi a eventualelor zgomote.

Bistabile

Descrierea unui bistabil sincron de tip D acţionat pe nivelul semnalului de ceas (latch).



Tabelul de tranziţie

D_t	Q_{t+1}	nQ_{t+1}
0	0	1
1	1	0

Program VHDL comportamental

```
library IEEE;
use IEEE.std_logic_1164.all;
entity D_latch is
port (D, Clk : in std_logic;
                Q , nQ : out std_logic);
end entity D_latch;
architecture D_comp of D_latch is
begin
p0: process (Clk) is
begin
if (Clk = '1') then
Q <= D;
```

```
nQ <= not D;
end if;
end process p0;
end architecture D_comp;
```

Pornind de la această descriere, compilatorul VHDL „deduce logic" un circuit latch, întrucât codul nu arată cum trebuie de procedat în cazul în care clk nu este 1. Bistabilul creat va păstra valoarea ***Q*** între apelările procesului.

În general, un compilator VHDL deduce un circuit latch pentru un semnal căruia i se atribuie o valoare în cadrul unei instrucţiuni *if* sau *case*, dacă nu se iau în consideraţie toate combinaţiile de intrare.

Majoritatea schemelor digitale modelate cu VHDL sunt sisteme sincrone, realizate cu bistabile active pe front (flip-flop). Pentru descrierea comportării activate pe front se foloseşte atributul ***event***. Atributul event poate fi ataşat unei denumiri de semnal pentru ca valoarea rezultată să fie *true* dacă semnalul îşi schimbă valoarea şi *false* în caz contrar.

Folosind atributul event putem crea un model de bistabil activ pe front. Construcţia ***if (clk`event)*** întoarce valoarea true pe oricare front al semnalului de tact. Construcţia ***if (clk`event and clk=`1`)*** întoarce valoarea true doar pe frontul crescător al semnalului de tact.

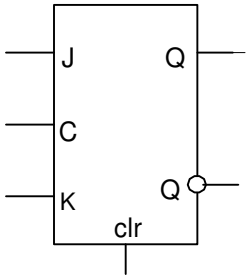
Codul VHDL pentru un bistabil sincron de tip D acţionat pe frontul crescător al semnalului de ceas cu intrare asincronă de resetare.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity vdff is
port (D, Clk, Clr : in std_logic;
                Q , nQ : out std_logic);
end entity vdff;
architecture D_comp of vdff is
begin
p0: process (Clk, Clr) is
variable state: std_logic;
begin
      if clr='1' then Q <= '0'; nQ <= '1';
elsif rising_edge(Clk) then
--  eslif (clk'event and clk = '1') then  Q <= D;  nQ <= not D; acelaşi efect
state := D;
end if;
Q <= state;
```

```
nQ <= not state;
end process p0;
end architecture D_comp;
```

Procesul utilizat pentru descrierea bistabilului este sensibil numai la modificările semnalului de ceas clk. Tranziţia semnalului de intrare ***D*** nu determină activarea procesului. În acest model intrarea asincronă de ştergere CLR este dominantă faţă de orice comportare determinată de semnalul de tact CLK, deci este prima evaluată în clauza „if”. Când CLR este negat, se evaluează clauza „elsif”.

Descrierea unui bistabil de tip JK cu resetare asincronă

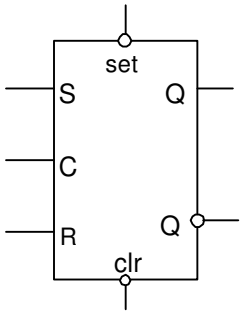


Program comportamental

```
library ieee;
use ieee.std_logic_1164.all;
entity JK_FF is
port (   clk, J, K, clr:     in std_logic;
         Q, Qn:  out std_logic );
end JK_FF;
architecture behv of JK_FF is
   signal s: std_logic;
   signal x: std_logic_vector(1 downto 0);
begin
    x  <= J & K; -- vector JK concatenat;
process (clk, clr) is
   begin
     if (clr = '1') then s <= '0';
    elsif (rising_edge(clk)) then
       case (x) is
when "11" => s <= not s;
when "10" => s <= '1';
when "01" => s <= '0';
when others =>    null;
           end case;
           end if;
   end process;
   Q <= s; --instrucţiuni concurente
   Qn <= not s;
end behv;
```

Funcţia rising_edge este definită în pachetul std_logic_1164, având rolul de a detecta frontul crescător al unui semnal. Această funcţie se poate utiliza în locul expresiei (clk'event and clk = '1'), dacă semnalul clk este de tip std_logic. În acelaşi pachet este definită şi funcţia falling_edge, care detectează fronturile descrescătoare ale semnalelor. Aceste funcţii sunt preferate de către unii proiectanţi deoarece la simulare funcţia rising_edge, de exemplu, va asigura că tranziţia este de la '0' la '1' şi nu va ţine cont de alte tranziţii, cum este cea de la 'U' la '1'.

Descrierea unui bistabil de tip SR cu resetare şi setatare asincronă.



Program comportamental

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;
entity SR-FF is
  port ( S,R,CLK,clr,set: in std_logic;
         Q, nQ: out std_logic);
end SR-FF;
architecture SR_arch of SR-FF is
  begin
    process (CLK,clr, set)
      variable x: std_logic;
        begin
          if (clr = '0')  then
            x:='0';
          elsif (set = '0')  then
             x:='1';
          elsif (CLK='1' and CLK'EVENT) then
            if( S='0' and R='0') then
               x:=x;
            elsif (S='1' and R='1') then
               x:='Z';
            elsif (S='0' and R='1')  then
               x:='0';
            else
               x:='1';
            end if;
          end if;

  Q<=x;
```

```
    nQ<=not x;
end process;
end SR_arch;
```

Registre

Descrierea unui registru de 8 biţi. D şi Q sunt vectori. Registrul are un semnal de validare a ceasului (ce).

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8 is
  port (clk, ce: in std_logic;
        D: in std_logic_vector (7 downto 0);
        Q: out std_logic_vector (7 downto 0));
end reg8;
architecture reg8_arch of reg8 is
   begin
     process (clk)
       begin
        if (clk'event and clk = '1') then
           if (ce = '1') then Q <= D;
           end if;
        end if;
      end process;
end reg8_arch;
```

Registre de deplasare

Un registru de deplasare este un circuit secvenţial care deplasează la stânga sau la dreapta conţinutul registrului cu o poziţie în fiecare ciclu de ceas. De obicei, intrările unui registru de deplasare sunt reprezentate de semnalul de ceas, o intrare serială de date, un semnal de setare/resetare sincronă sau asincronă şi un semnal de validare a ceasului. În plus, un registru de deplasare poate avea semnale de control şi de date pentru încărcarea paralelă sincronă sau asincronă. Datele de ieşire ale unui registru de deplasare pot fi accesate fie serial, atunci când este accesibil numai conţinutul ultimului bistabil pentru restul circuitului, fie în paralel, atunci când este accesibil conţinutul mai multor bistabile.

Utilitarele de sinteză ale diferitor firme conţin resurse dedicate (ex. primitivele SRL16 şi SRL32, XILINX) care permit o implementare eficientă a registrelor de

deplasare fără utilizarea unor bistabile suplimentare. Totuşi, aceste resurse permit numai operaţii de deplasare la stânga şi au un număr limitat de semnale de intrare/ieşire: ceas, validarea ceasului, intrare serială de date şi ieşirea sincronă.

Există mai multe posibilităţi pentru descrierea registrelor de deplasare în limbajul VHDL:

- Utilizarea operatorului de concatenare: reg <= reg (6 downto 0) & SI;
- Utilizarea construcţiilor for loop;
- Utilizarea operatorilor de deplasare predefiniţi (sll, srl, sla, sra).

Descrierea unui registru de deplasare la stânga de 8 biţi cu semnale de validare a ceasului, intrare serială şi ieşire serială. Pentru descrierea registrului de deplasare se utilizează o construcţie for loop.

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8_depl is
  port (clk, ce, si: in std_logic;
        so:          out std_logic);
end reg8_depl;
architecture reg_depl of reg8_depl is
   signal tmp: std_logic_vector (7 downto 0);
    begin
     process (clk)
        begin
         if (clk'event and clk = '1') then
           if (ce = '1') then
           for i in 0 to 6 loop
             tmp(i+1) <= tmp(i);
           end loop;
             tmp(0) <= si;
           end if;
        end if;
     end process;
  so <= tmp(7);
end reg_depl;
```

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

```
entity test is
port(clk,il,ir :in bit;
 s :in bit_vector(1 downto 0);
 i : in bit_vector(3 downto 0);
 q : out bit_vector(3 downto 0));
end test;
architecture beh2 of test is
signal qtmp: bit_vector(5 downto 0);
begin
 process(clk)
   begin
     if (clk = '1' and clk'event) then
          case s is
       when "00" => qtmp <= qtmp;
 when "01" =>   qtmp <=il&i&ir;
 when "10" =>   qtmp <= (il&qtmp(4 downto 1)&ir) sll 1;
 when "11" =>   qtmp<= (il&qtmp(4 downto 1)&ir) srl 1 ;
 when others =>   null;
     end case;
       end if;
 end process;
 q<=qtmp(4 downto 1);
end beh2;
```

Numărătoare

Descriea unui numărător de 3 biţi.

```
library ieee;
use ieee.std_logic_1164.all;
entity num3 is
   port (clk: in std_logic;
           num: out integer range 0 to 7);
end num3;
architecture num3_integer of num3 is
   signal tmp: integer range 0 to 7;
     begin
       cnt: process (clk)
          begin
            if (clk'event and clk = '1') then
```

```
            tmp <= tmp + 1;
          end if;
        end process cnt;
        num <= tmp;
end num3_integer;
```

În exemplul anterior, pentru semnalul num, care este de tip integer, se utilizează operatorul de adunare. Majoritatea utilitarelor de sinteză permit această utilizare, convertind tipul integer la tipul bit_vector sau std_logic_vector.

Utilizarea tipului integer pentru porturi pune însă unele probleme:

1) Pentru a utiliza valoarea num într-o altă porţiune a proiectului pentru care interfaţa are porturi de tip std_logic, trebuie efectuată o conversie de tip.

2) Vectorii aplicaţi în timpul simulării codului sursă nu pot fi utilizaţi pentru simularea modelului generat în urma sintezei. Pentru codul sursă, vectorii trebuie să fie valori întregi. Modelul generat în urma sintezei necesită vectori de tip std_logic.

Deoarece operatorul nativ + al limbajului VHDL nu este definit pentru tipurile bit sau std_logic, acest operator trebuie redefinit înainte de adunarea operanzilor care au aceste tipuri. Standardul IEEE 1076.3 defineşte funcţii pentru redefinirea operatorului + pentru următoarele perechi de operanzi: (unsigned, unsigned), (unsigned, integer), (signed, signed) şi (signed, integer). Aceste funcţii sunt definite în pachetul numeric_std al standardului 1076.3.

În următorul exemplu se utilizează tipul unsigned pentru ieşirea numărătorului.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity num3 is
  port (clk: in std_logic;
        num: out unsigned (2 downto 0));
end num3;
architecture num3_unsigned of num3 is
  signal tmp: unsigned (2 downto 0);
    begin
      cnt: process (clk)
       begin
          if (clk'event and clk = '1') then
            tmp <= tmp + 1;
          end if;
      end process cnt;
      num <= tmp;
```

```
end num3_unsigned;
```

De obicei, utilitarele de sinteză pun la dispoziţie pachete suplimentare care redefinesc operatorii pentru tipul std_logic. Deşi acestea nu sunt pachete standard, ele se utilizează adesea de către proiectanţi, deoarece permit operaţii aritmetice şi relaţionale cu tipul std_logic, din acest punct de vedere fiind chiar mai utile decât pachetul numeric_std. De asemenea, aceste pachete nu necesită utilizarea a două tipuri suplimentare (signed, unsigned) în plus faţă de tipul std_logic_vector şi nici a funcţiilor necesare conversiei între aceste tipuri. La utilizarea unuia din aceste pachete pentru operaţiile aritmetice, utilitarul de sinteză va utiliza pentru tipul std_logic_vector o reprezentare fără semn sau una cu semn (în complement faţă de 2) şi va genera componentele aritmetice corespunzătoare.
În următorul exemplu se prezintă descrierea modificată a numărătorului din exemplele precedente pentru a se utiliza pachetul std_logic_unsigned şi tipul std_logic_vector pentru ieşirea numărătorului.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity num3 is
  port (clk: in std_logic;
        num: out std_logic_vector (2 downto 0));
end num3;
architecture num3_std_logic of num3 is
  signal tmp: std_logic_vector (2 downto 0);
    begin
     cnt: process (clk)
       begin
         if (clk'event and clk = '1') then
            tmp <= tmp + 1;
         end if;
      end process cnt;
          num <= tmp;
end num3_std_logic;
```

Numărător de 8 biţi cu semnale asincrone de resetare şi setare.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity num8 is
```

```
   port (clk: in std_logic;
         rst, set: in std_logic;
         en, load: in std_logic;
         data: in std_logic_vector (7 downto 0);
          num: out std_logic_vector (7 downto 0));
end num8;
architecture arh_num8 of num8 is
   signal tmp: std_logic_vector (7 downto 0);
     begin
      cnt: process (rst, set, clk)
           begin
              if (rst = '1') then
                 tmp <= (others => '0');
              elsif (set = '1') then
                 tmp <= (others => '1');
              elsif (clk'event and clk = '1') then
                 if (load = '1') then
                    tmp <= data;
              elsif (en = '1') then
                    tmp <= tmp + 1;
              end if;
            end if;
       end process cnt;
         num <= tmp;
end arh_num8;
```

Majoritatea circuitelor programabile dispun de ieşiri cu trei stări sau semnale bidirecţionale de I/E. În plus, anumite circuite dispun de buffere interne cu trei stări. Un semnal cu trei stări poate avea valorile '0', '1' şi 'Z', toate acestea fiind permise de tipul std_logic.

Descrierea modificată a numărătorului din exemplul precedent pentru a utiliza ieşiri cu trei stări. Acest numărător nu dispune de un semnal de setare asincronă.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity num8 is
   port (clk, rst: in std_logic;
         en, load: in std_logic;
         oe: in std_logic;
```

```
        data: in std_logic_vector (7 downto 0);
        num: out std_logic_vector (7 downto 0));
end num8;
architecture arh_num8 of num8 is
    signal tmp: std_logic_vector (7 downto 0);
      begin
   cnt: process (rst, clk)
       begin
         if (rst = '1') then
           tmp <= (others => '0');
         elsif rising_edge (clk) then
           if (load = '1') then
             tmp <= data;
         elsif (en = '1') then
            tmp <= tmp + 1;
           end if;
         end if;
     end process cnt;
     oep: process (oe, tmp)
       begin
         if (oe = '0') then
          num <= (others => 'Z');
        else
          num <= tmp;
         end if;
     end process oep;
end arh_num8;
```

În această descriere se utilizează un semnal suplimentar oe pentru controlul ieşirilor cu trei stări. Procesul etichetat cu oep descrie ieşirile cu trei stări ale numărătorului. Dacă semnalul oe nu este activat, ieşirile sunt trecute în starea de înaltă impedanţă. Descrierea procesului oep este în concordanţă cu funcţionarea unui buffer cu trei stări: