Stiluri de proiectare şi descriere a circuitelor

O arhitectură este formată dintr-o serie de instrucţiuni concurente. Fiecare instrucţiune se execută simultan cu celelalte instrucţiuni din arhitectură. Instrucţiunile concurente sunt necesare pentru a putea simula comportamentul circuitelor modelate, în care componentele interacţionează unele cu altele.

Într-o arhitectură *VHDL* dacă ultima instrucţiune modifică starea unui semnal care este folosit de prima instrucţiune din arhitectură atunci simulatorul se va întoarce la prima instrucţiune şi o va reactualiza cu noua valoare, cu alte cuvinte simulatorul va continua să propage schimbările şi să reactualizeze valorile semnalelor până când circuitul simulat se stabilizează.

Sintaxa *VHDL* conţine câteva instrucţiuni concurente şi de asemenea un mecanism de grupare a instrucţiunilor secvenţiale astfel încât ele să funcţioneze ca şi o sigură instrucţiune concurentă. Utilizate în diferite modalităţi, aceste instrucţiuni generează trei stiluri diferite de proiectare şi descriere a circuitelor:

- proiectare structurală;
- proiectare ca flux de date;
- proiectare comportamentală.

Proiectare structurală

În proiectarea structurală se definesc exact elementele şi interconexiunile dintre elementele circuitului. O descriere structurală pură este echivalentă cu o descriere schematică sau cu un netlist (listă de conexiuni între elementele unui circuit).

Cea mai elementară instrucţiune concurentă, folosită în descrierea structurală este instrucţiunea *component:*

etichetă: nume-componentă **port-map** (semnal1, semnal2, ..., semnal n);
etichetă: nume-componentă **port-map** (port1=> semnal1, port2=>semnal2, …portn=>semnaln);

Numele componentei este numele unei entităţi definite anterior care trebuie utilizată sau instanţiată (multiplicată).

Fiecare instrucţiune *component* care invocă numele entităţii creează o "clonă" (instanţă) respectivei entităţi care se va distinge printr-un nume unic dat de o etichetă (*label*).

Cuvântul cheie *port map* introduce o listă prin care porturile entităţii sunt asociate unor semnale din arhitectura curentă.

Această listă se poate scrie în două moduri: poziţional şi asociativ.

În modul poziţional de atribuire semnalele din listă sunt asociate cu porturile entităţii în aceeaşi ordine în care apar în entitate.

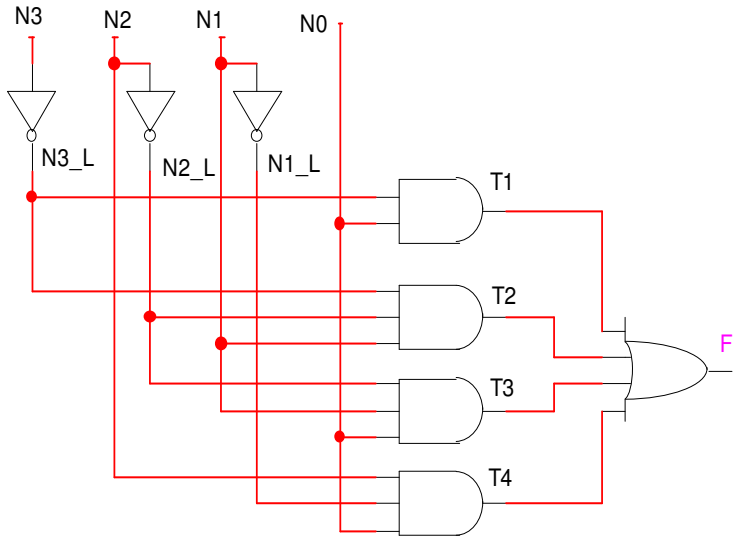
În modul de atribuire asociativ fiecare port al entităţii se asociază unui semnal folosind operatorul "=>", iar asocierea poate fi făcută în orice ordine.

Înainte de a fi instanţiată într-o arhitectură, o componentă trebuie să fie declarată în cadrul arhitecturii, sintaxa de declarare este:

component nume-componentă
port (nume-semnale: mod tip-semnale;
nume-semnale: mod tip-semnale;
...
nume-semnale: mod tip-semnale);
end component;

Componentele folosite în arhitectură pot să fie de două tipuri:
a) componente descrise anterior care sunt specifice proiectului,
b) componente care sunt apelate din bibliotecile standard.

Exemplu. Codul VHDL pentru un detector de numere prime pe 4 biţi.
Numerele prime sunt: 1, 2, 3, 5, 7, 11, 13.
Funcţia minimizată: $F = \bar{N}_3 N_0 + \bar{N}_3 \bar{N}_2 N_1 + \bar{N}_2 N_1 N_0 + N_2 \bar{N}_1 N_0$
Circuitul logic:



$$T_1 = \bar{N}_3 N_0$$
$$T_2 = \bar{N}_3 \bar{N}_2 N_1$$
$$T_3 = \bar{N}_2 N_1 N_0$$
$$T_4 = N_2 \bar{N}_1 N_0$$

Programul structural VHDL pentru detectorul de numere prime:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime is
```

```
        port ( N: in STD_LOGIC_VECTOR (3 downto 0);
                F: out STD_LOGIC ) ;
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L  :  STD_LOGIC;
signal T1, T2, T3, T4  :  STD_LOGIC;
component  G_INV  port  (x :  in   STD_LOGIC; Q:  out   STD_LOGIC) ;
end component ;
component  G_AND2  port  (x, y : in STD_LOGIC; Q:  out   STD_LOGIC) ;
end component ;
component  G_AND3  port  (x, y, z : in STD_LOGIC; Q:  out   STD_LOGIC) ;
end component ;
component G_OR4  port (x, y, z, w : in STD_LOGIC; Q:  out   STD_LOGIC ;
end component ;

begin
        U1 : G_INV  port map  (N(3) , N3_L) ;
        U2 : G_INV  port map  (N(2) , N2_L) ;
        U3 : G_INV  port map  (N(1) , N1_L) ;
        U4 : G_AND2  port map (N3_L,  N(0), T1) ;
        U5 : G_AND3  port map (N3_L, N2_L,  N(1), T2) ;
        U6 : G_AND3  port map (N2_L, N(1), N(0),   T3) ;
        U7 : G_AND3  port map (N(2), N1_L, N(0), T4) ;
        U8 : G_OR4  port map (T1, T2, T3, T4, F) ;
end prime1;
```

Prin declararea entităţii se declară intrările şi ieşirile circuitului. În cadrul arhitecurii sunt declarate toate semnalele care vor fi folosite, de asemenea şi numele componentelor **(G_INV, G_AND2, G_AND3, G_OR4)** care trebuie definite în acelaşi proiect în fişiere VHDL de nivel ierarhic mai jos.

Exemplu de descriere a porţii logice AND cu două intrări:

```
library ieee;
use ieee.std_logic_1164.all;
entity G_And2 is
port (    x: IN std_logic;
          y: IN std_logic;
          Q: OUT std_logic);
end G_And2;
architecture G_And2_beh of G_And2 is
```

```
begin
  Q <= x and y;
end G_And2_beh;
```

Deoarece descrierea circuitului este concurentă în orice ordine am introduce componentele se va sintetiza acelaşi circuit, iar funcţionarea va fi aceeaşi.

Există proiecte în care este necesară crearea mai multor copii a unui element în cadrul unei arhitecturi. De exemplu, un sumator pe *n* biţi se poate crea din *n* sumatoare pe un bit.

Limbajul *VHDL* include o instrucţiune (*generate*) care permite crearea unei structuri repetitive folosind un fel de buclă, fără a fi necesar să instanţiem separat fiecare element.

Sintaxa unei bucle *for- generate* :

```
label: for identificator  in domeniu  generate
instrucţiune concurentă
end generate;
```

Identificatorul este implicit declarat ca variabilă compatibilă cu domeniul (*range*). Instrucţiunile concurente se execută câte o dată pentru fiecare valoare posibilă a identificatorului inclusă în domeniu. Identificatorul trebuie folosit în cadrul instrucţiunii concurente.

Exemplu. Crearea unei porţi inversoare pe 8-biţi.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity inv8 is
        port ( X: in STD_LOGIC_VECTOR  (1  to  8);
                Y: out  STD_LOGIC_VECTOR  (1  to  8) );
end inv8;

architecture inv8_arch  of inv8  is
component G_INV  port (I: in  STD_LOGIC;
                        Q: out STD_LOGIC);
end component;
begin:
        g1: for b  in  1 to 8  generate
```

```
        U1: G_INV  port map (X(b) , Y(b)) ;
    end generate;
end inv8_arch;
```

Programul pentru inversorul pe un bit trebuie inclus în proiectul curent:

```
library ieee;
use ieee.std_logic_1164.all;
entity G_INV is
  port( I  : in std_logic;
        Q : out std_logic);
end G_INV;
architecture func of G_INV is
begin
  Q <= not I;
end func;
```

Valoarea constantelor trebuie să fie cunoscută în momentul compilării unui program VHDL. În unele aplicaţii este util să se proiecteze şi să se compileze o entitate împreună cu arhitectura corespunzătoare care să conţină unii parametrii (cum ar fi lăţimea magistralei) care nu sunt specificaţi. Această facilitate este introdusă de constanta *generic.*

Genericele nu se modifică în timpul simulării. Datele conţinute în genericele transmise entităţii sau componentei se pot utiliza pentru a modifica rezultatele simulării, dar rezultatele nu pot modifica genericele.

```
entity nume-entitate  is
generic ( nume constante : tip-constante;
            …
            nume-constante:  tip-constante);
port ( nume-semnale: mod tip-semnale;
        …
        nume-semnale: mod tip-semnale);
end nume-entitate;
```

Una sau mai multe constante generice pot fi definite într-o declaraţie de entitate înaintea declaraţiei de porturi. O valoare îi va fi atribuită numai când entitatea va fi instanţiată în cadrul altei arhitecturi. În cadrul definirii componentelor (*component*), constantelor *generic* li se atribuie valori folosind instrucţiunea *generic map,* asemănătoare cu instrucţiunea *port map.*

Exemplu. Definirea unui inversor de magistrală cu o lăţime impusă de utilizator.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity businv is
        generic ( WIDTH: positive);
        port ( X: in STD_LOGIC_VECTOR  (WIDTH -1  downto  0);
                Y: out  STD_LOGIC_VECTOR  (WIDTH -1  downto  0) );
end businv;

architecture businv_arch  of businv  is
component G_INV  port (I: in  STD_LOGIC;
                        Q: out STD_LOGIC);
end component;
begin
        g1: for b  in  WIDTH -1  downto  0 generate
            U1: G_INV  port map (X(b) , Y(b)) ;
        end generate;
end businv_arch;
```

În următorul exemplu sunt instanţiate mai multe exemplare ale acestui inversor, fiecare pentru o altă lăţime.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv1 is
        port ( IN8: in STD_LOGIC_VECTOR  (7  downto  0);
                OUT8: out  STD_LOGIC_VECTOR  (7  downto  0);
                IN16: in STD_LOGIC_VECTOR  (15  downto  0);
                OUT16: out  STD_LOGIC_VECTOR  (15  downto  0);
                IN32: in STD_LOGIC_VECTOR  (31  downto  0);
                OUT32: out  STD_LOGIC_VECTOR  (31  downto  0) );
end businv1;

architecture businv1_arch  of businv1  is
component businv
         generic ( WIDTH: positive);
         port ( X: in STD_LOGIC_VECTOR  (WIDTH -1  downto  0);
```

```
                Y: out  STD_LOGIC_VECTOR  (WIDTH -1  downto  0) );
end component;

begin:
        U1: businv  generic map (WIDTH=>8)  port map (IN8, OUT8);
        U2: businv  generic map (WIDTH=>16)  port map (IN16, OUT16);
        U3: businv  generic map (WIDTH=>32)  port map (IN32, OUT32);
end businv1_arch;
```