LABORATOR 7

Elaborarea programelor de test (testbench) în VHDL

1. Introducere

Una din motivațiile procesului de simulare este posibilitatea de a testa proiectele înainte de construirea şi folosirea circuitului. Pentru a testa un bloc sau o componentă electronică trebuie **să furnizăm anumite semnale de intrare şi să observăm valorile care sunt furnizate la ieşire. In cazul în care acestea nu corespund cu valorile dorite, circuitul se modifică.** La fel stau lucrurile şi cu circuitele proiectate în VHDL. Trebuie să folosim un simulator care să ofere comenzi pentru aplicarea stimulilor la porturile de intrare ale circuitului proiectat. Vizualizând valorile care se obțin la porturile de ieşire, ne putem da seama dacă modelul funcționează corect.

2. Considerații teoretice

Structura bloc a unui program de test **(test bench)** VHDL este prezentată în figura 1. În această figură, un modul VHDL **(tester.vhd)** generează stimulii care trebuie aplicați. Un al doilea modul VHDL **(model.vhd)** este modelul de testat. Al treilea modul **(testbenchvhd)** este un model structural VHDL care descrie interconexiunile dintre tester şi modulul de testat. Acest model descrie modul în care porturile testerului sunt conectate la porturile modelului. Simularea evoluează prin aplicarea stimulilor la modelul de testat şi citirea şi înregistrarea răspunsurilor (valorile porturilor de ieşire). *Secvența de valori de intrare care trebuie să se aplice la porturile de intrare ale modelului de testat poate fi citită dintr-un fişier sau poate fi generată în VHDL*. De exemplu, semnalele de ceas, pulsurile reset, sau alte tipuri de forme de undă periodice pot fi generate urmând exemplele din acest laborator.

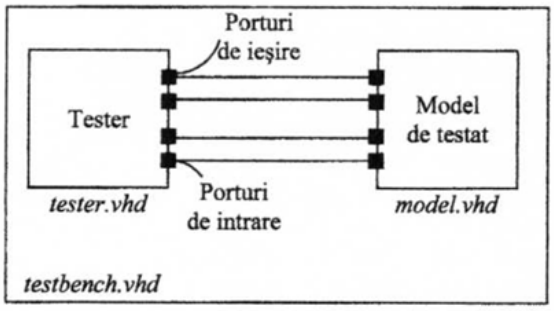

Fig. 1 Structura unui program de test.

Rezultatele returnate de modelul VHDL pot fi comparate de tester cu valorile corecte, cunoscute, şi pot fi semnalate erori în funcționare. În figura 2 este prezentată diagrama bloc a acțiunilor care pot avea loc în cazul unei operații de testare a funcționării.

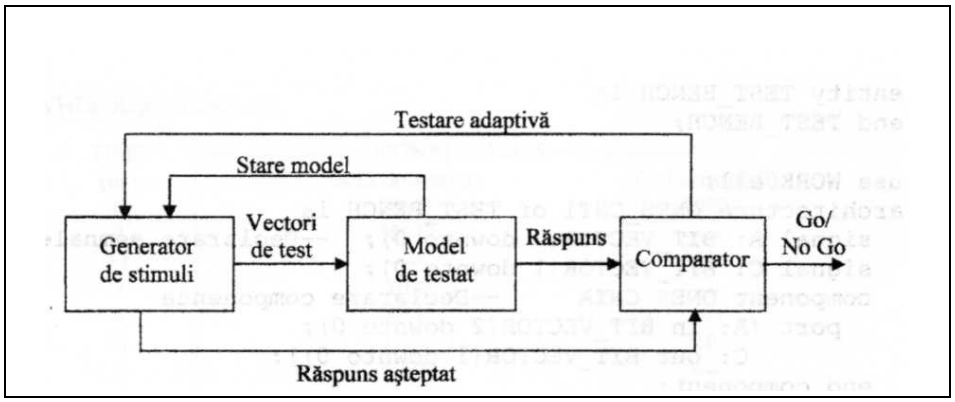

Fig. 2 Organizarea unui testbench.

3. Exemplu de descriere în VHDL a unui program de test

În figura 3 este prezentat modelul unui numărător de valori unu logic dintr-un vector binar. Test bench-ul pentru acest numărător este prezentat în figura 4. Este un exemplu simplu în care se folosesc asignările de semnal tipice.

```
entity ONES_CNT is
 port  (A:  in BIT_VECTOR  (2 downto 0) ;
  C:  out BIT_VECTOR  (1 downto 0));
 end ONES_CNT;
  architecture ALGORITHMIC of ONES_CNT is
  begin
  process(A)
  variable NUM:  INTEGER range 0 to 3;
  begin
    NUM := 0;
    for I in 0 to 2 loop
     if A{I)  = '1'    then
      NUM := NUM + 1;
     end if;
    end loop;
    case NUM is
      when   0 => C <= "00";
      when   1 => C <= "01";
      when   2 => C <= "10";
      when    3 => C <= "11";
    end case;
  end process;
 end ALGORITHMIC;
```

Fig. 3 Descriere algoritmica pentru un contor de valori 1.

```
entity TEST_BENCH is end
TEST_BENCH;
use WORK.all;
architecture ONES_CNTl of TEST_BENCH  is
  signal A: BIT_VECTOR (2 downto  0);    --Declarare semnale
  signal C: BIT_VECTOR (l  downto 0 ) ;
  component ONES__CNTA        --Declarare componenta
  port  (A:  in BIT_VECTOR (2 downto 0);
                C:  out BIT_VECTOR (1 downto 0)) ;
      end component;
for LI: ONES_CNTA use entity ONES_CNT(ALGORITHMIC);
    begin
  LI: ONES_CNTA
     port map(A, C);
    process
    begin
      A <= "000" after 100 ns,  "001" after 200 ns,
          "..." after 300 ns,  "…" after 400 ns,
          "..." after 500 ns,  "…" after 600 ns,
          "..." after 700 ns,  "…" after 800 ns;
      wait;
    end process;
   end 0NES_CNT1;
```

Fig. 4 Un test bench pentru entitatea ONES_CNT

Entitatea TEST_BENCH nu conține declarația de porturi pentru că semnalele de test sunt generate intern de programul de test. In interiorul arhitecturii ONES_CNT1, sunt declarate semnalele A - intrarea de test şi C - ieşirea de test pentru entitatea ONES_CNT. Se face apoi o declarație pentru componenta ONES_CNTA, care este apoi legată de entitatea ONES_CNT, arhitectura algorithmic, din biblioteca de proiect. După cuvântul cheie **begin,** se instanțiază componenta ONES_CNTA iar semnalele se la porturile acesteia sunt transformate în A şi C prin declarația **port map.** În continuare, declarația de proces conține o secvență de vectori de test care vor fi aplicați entității ONES_CNT. Procesul se execută o singură dată la începutul simulării apoi se suspendă datorită instrucțiunii **wait**. Toți cei opt vectori sunt programați să apară la intervale de 1 ns.

Rezultatul simulării este prezentat în figura 5. În coloana din stânga se specifică timpul (în ns) la care unele semnale îşi schimbă valoarea. Restul unei linii din raport specifică noile valori pentru fiecare semnal care îşi schimbă valoarea la acel moment. Semnele - dintr-o coloană înseamnă că semnalul nu îşi schimbă valoarea la momentul respectiv. Intrarea A este în coloana din mijloc iar răspunsul corespunzător C este în coloana cea mai din dreapata.

Răspunsul C prezintă întârziere **delta,** e.g., valoarea "001" este aplicată la A la momentul 2 ns. Răspunsul la C "01" apare la momentul 2ns + 1 **delta.**

TIME	SIGNAL NAMES	
(ns)	A (2 downto 0)	C (1 downto 0)
0	"000"	"00"
2	"001"	----
+1	-----	"01"
3	"010"	----
4	"011"	----
+1	-----	"10"
5	"100"	----
+1	-----	"01"
6	"101"	----
+1	-----	"10"
7	"110"	----
8	"111"	----
+1	-----	"11"

Fig. 5 Rezultatele simulării pentru entitatea ONES_CNT

Parcurgeți toate etapele menționate aici pentru a realiza simularea contorului de valori 1 cu ajutorul test bench-ului descris în figura 4. Completații spațiile libere conform tabelului T1. Prezentați rezultatele în scris.

4. Utilizarea fişierelor de text pentru descrierea unui program de test

În unele cazuri, pentru testare, se folosesc **fişiere text de I/O.** De exemplu, următorul fişier text poate fi folosit pentru a testa numărătorul de valori 1 de mai sus:

A	C
000	00
001	01
010	01
011	10
100	01
101	10
110	10
1 1 1	1 1

Tabel T1

Cei trei biți mai semnificativi reprezintă vectorul de intrare A; cei doi biți mai puțin semnificativi reprezintă ieşirea corectă C. În figura 6 este prezentat fişierul de test care citeşte acest fişier text I/O.

Datele din fişierele text VHDL sunt organizate în linii având un număr variabil de elemente pe fiecare linie. Pentru a citi datele dintr-un fişier text trebuie să se execute mai întîi o comandă READLINE pentru a se citii o linie întreagă, apoi un anumit număr de comenzi READ pentru elementele individuale din linia respectivă. Tipul LINE este declarat ca un tip

de acces STRING. Tipurile access sunt folosite pentru variabile care reprezintă pointeri la memorie. Aceşti pointeri pot fi creaţi şi eliminaţi pe durata simulării. Deci memoria nu este alocată în permanenţă pentru aceşti pointeri. Tipul TEXT este de asemenea un tip al cărui tip de bază este tipul STRING. Tipul STRING se foloseşte pentru a declara o gamă largă de date de intrare în mediul gazdă. Toate aceste funcţii sunt declarate în pachetul TEXTIO.

Când semnalul EN are o tranziţie din 0 în 1, începe citirea din fişier. La fiecare nanosecundă se citeşte o linie. Primul vector din linie (V1) se aplică intrării A. Al doilea vector din linie (V2) se compară cu ieşirea C în instrucţiunea **assert.**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use STD.TEXTIO.all;

entity TEST_BENCH
is end TEST_BENCH;
architecture ONES_CNT1 of
TEST_BENCH is
 signal EN:  BIT;
 signal A: BIT_VECTOR (2 downto 0);
 signal C: BIT_VECTOR(1 downto 0) ;
component ONES_CNT
port  (EN:  BIT;
      A:  in BIT_VECTOR(2 downto 0);
       C: out BIT_VECTOR(1 downto 0) ) ;
end component;
begin
LI:  ONES_CNT
 port map (EN, A, C);
process
 variable VLINE: LINE;
 variable V1:  BIT_VECTOR(2 downto 0);
 variable V2: BIT_VECTOR(1 downto 0);
 file INVECT:  TEXT is "TVECT.TEXT";
begin
 EN <= '0',   '1' after 200 ns;
 wait on EN until EN = '1';
      while not(ENDFILE(INVECT))  loop
      READLINE(INVECT, VLINE);
      READ(VLINE, V1);
      READ(VLINE, V2);
      A<= V1;
```

```
     wait for 100 ns;
     assert  (V2 = C)
     report "Atentie: C nu este egal cu (V2 sau C)"
      severity WARNING;
     end loop;
    end process;
end ONES_CNT1;
```

Fig. 6 Test bench de tip text I/O

Realizaţi simularea contorului de valori 1 cu ajutorul test bench-ului descris în figura 6. Proiectul va conţine două fişiere VHDL, cel din figura 4 şi cel din figura 5. Fişierului VHDL din figura 4 ce reprezintă contorul de valori trebuie să-i adăugăm o intrare de activare *EN* activă pe '1'. De asemenea trebuie creat cu ajutorul unui editor de text şi un fişier cu vectori de test conform tabelului T1, fişierul se va numii `TVECT.TEXT`. Verificaţi sintaxa, realizaţi sinteza şi simularea şi prezentaţi rezultatele în scris. Introduceţi valori eronate în coloana ieşirii C, pentru a verifica executarea instrucţiunii *ASSERT*.

5. Program de test pentru un circuit sumator

În figura 7 este prezentat codul VHDL al sumatorului pe n biţi. Dimensiunea variabilă a magistralelor de date se realizează utilizând instrucţiunea *generic*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity sum_test is
     generic (n: natural := 4);
     port ( A,B: in std_ulogic_vector (n-1 downto 0);
               Cin: in std_ulogic;
               Sum: out std_ulogic_vector (n-1 downto 0);
               Cout: out std_ulogic);
end sum_test;

architecture sum of sum_test is
     signal result : unsigned (n downto 0);
     signal carry : unsigned (n downto 0);
begin
     carry <= (0=> Cin, others => '0');
     result  <=  ('0'&  unsigned  (A))+  ('0'  &  unsigned(B))  +
carry;
```

```
     Sum <= std_ulogic_vector(result(n-1 downto 0));
     Cout<= result(n);
end sum;
```

Fig. 7 Cod VHDL pentru sumator pe n biți

În figura 8 este prezentat programul de test pentru sumatorul din figura 7. Pentru testare se utilizează două fişiere, unul pentru stimuli (vectori.txt) sau vectori de intrare iar celălalt stochează rezultatele simulării (rezultate.txt). Vectorii de test ce vor fi scrişi in fisierul de intrare vor fi de forma:

A B Cin
0000 0000 0
0000 0001 0
1111 1111 0
1111 1111 1
etc...

Fişierul de ieşire ce va rezulta în urma simulării va fi de forma:

Sum Cout
0000 0
0001 0
1110 1
1111 1
etc....

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE std.textio.all;

ENTITY Sum_test_tb_vhd IS
END Sum_test_tb_vhd;

ARCHITECTURE behavior OF Sum_test_tb_vhd IS
 file vectors: text; -- se defineste fisierul de intrare
 file results: text; -- se defineste fisierul de iesire
     -- declarare componenta UUT
     COMPONENT sum_test
     PORT(
          A : IN std_ulogic_vector(3 downto 0);
          B : IN std_ulogic_vector(3 downto 0);
          Cin : IN std_ulogic;
          Sum : OUT std_ulogic_vector(3 downto 0);
```

```
          Cout : OUT std_ulogic
          );
     END COMPONENT;

     --Inputs
     SIGNAL Cin :  std_logic := '0';
     SIGNAL A : std_ulogic_vector(3 downto 0) := (others=>'0');
     SIGNAL B : std_ulogic_vector(3 downto 0) := (others=>'0');

     --Outputs
     SIGNAL Sum :  std_ulogic_vector(3 downto 0);
     SIGNAL Cout :  std_ulogic;

BEGIN
     -- Intantiere Unit Under Test (UUT)
     uut: sum_test PORT MAP(
          A => A,
          B => B,
          Cin => Cin,
          Sum => Sum,
          Cout => Cout
     );
     tb : PROCESS is
     variable  ILINE,  OLINE:  Line;  --  declarare  variabile
pointer pententru ctire-scriere
     variable X_in, Y_in, Z_out: bit_vector(3 downto 0);
     variable ci_in, co_out : bit;
     variable ch : character;

     BEGIN
          file_open (vectors, "vectori.txt", read_mode);
-- deschidere in mod citire fisier cu vectori de test
          file_open (results, "rezultate.txt", write_mode);
--deschidere in mod scriere fisier cu rezultate simulare
          --ciclu de citire-scriere
          while not endfile(vectors) loop
               readline (vectors, ILINE);-- citeşte o linie din
fişier
               read(ILINE,  X_in);  --  citeşte  un  vector  din
linie
               read(ILINE, ch);
               read(ILINE, Y_in);
               read(ILINE, ch);
               read(ILINE, ci_in);
```

```
            --converteşte şi pune stimulii la intrare
            A <= to_stdulogicvector(X_in);
            B <= to_stdulogicvector(Y_in);
            cin <= to_stdulogic(ci_in);

            wait for 100 ns;
            --citeşte starea ieşirilor
            Z_out := to_bitvector (Sum);
            co_out := to_bit (cout);
          --scrie valorile convertite in fisierul de iesire
            write (OLINE, Z_out, right, 5);
            write (OLINE, co_out, right, 2);
            writeline (results, OLINE);
        end loop;
        --inchide fisierele
        file_close (vectors);
        file_close (results);
    END PROCESS;
END ARCHITECTURE;
```

Fig. 8 Program de test pentru sumator pe n biți ce permite citirea rezultatelor dintr-un fişier şi scrierea într-un alt fişier.

Creați un proiect care să conțină fişierele din figurile 7 şi 8. Creați fişierul cu stimuli de intrare (vectori.txt) conform exemplului din paragraful anterior. Simulați, analizați rezultatele simulării şi vizualizați conținutul fişierului *rezultate.txt* ce trebuie să conțină vectorii de ieşire. Extindeți numărul de vectori de test. Salvați într-un fişier formele de undă obținute.

Similar cu codul din figura 6, extindeți fişierul cu vectori de intrare cu coloanele corespunzătoare (ieşirile Sum şi Cout) şi utilizați instrucțiunile *assert* şi *report* pentru a verifica funcționarea sumatorului.