LUCRARE LABORATOR NR. 7
SIMULAREA ŞI VERIFICAREA MODELELOR VHDL ÎNTR-UN TESTBENCH.

1. Scopul lucrării

Însuşirea principiilor de simulare şi verificare a unui model VHDL cu ajutorul entităților de test (testbench-uri). Tehnici de implementare a unui testbench.

2. Rolul şi structura unui testbench

După descrierea (modelarea) unui proiect în VHDL, acesta trebuie verificat pentru a constata dacă are comportarea în concordanță cu specificațiile dorite pentru acel proiect.

Cea mai simplă metodă de verificare a unui model descris în VHDL constă în efectuarea unei simulări în cadrul căreia se aplică semnale adecvate la intrările modelului iar constatarea valabilității acestuia se face prin vizualizarea şi inspecția semnalelor de la ieşire.

O astfel de metodă a fost prezentată şi utilizată în lucrarea de laborator anterioară în care, pentru simularea şi verificarea modelelor VHDL ale circuitelor, semnalele de la intrări s-au specificat şi generat din linia de comandă cu ajutorul comenzii *force*. Această metodă de simulare poate fi utilizată pentru verificarea modelelor unor circuite simple dar este ineficientă în cazul verificării modelelor circuitelor complexe, din două motive:

– pe de o parte comanda *force* permite generarea doar a unor forme de undă simple pentru semnalele de la intrări prin specificarea tranzițiilor acestora prin perechi *valoare-timp*. Comanda *force* nu permite, de exemplu, corelarea tranzițiilor unui semnal în funcție de cele ale altui semnal.

– pe de altă parte, în cazul circuitelor complexe având multe de intrări şi ieşiri, simularea trebuie efectuată pentru un număr mare de combinații de valori la intrări sau pentru multe perioade ale semnalelor de clock. În aceste condiții este extrem de dificil de efectuat inspecția vizuală a tuturor combinațiilor de valori (stări) ale semnalelor de la intrări şi ieşiri pentru a aprecia dacă modelul se comportă în concordanță cu specificațiile dorite.

O altă metodă de simulare şi verificare a unui model VHDL, mult mai răspândită în practică, este cea a includerii modelului VHDL în cadrul unui *model de test* descris tot în VHDL. *Modelul de test* (*testbench*) este, în fapt, o entitate fără porturi care conține în arhitectura sa, pe de o parte, instanțierea modelului suspus testării şi, pe de altă parte, instrucțiuni, procese sau componente care generează valori pentru semnalele conectate la intrările modelului supus testării şi, eventual, pot monitoriza valorile semnalelor de la ieşirile acestuia.

Testbench-ul VHDL are următoarele **avantaje**:

– poate fi implementat sub diverse forme prin care pot fi testate atât modele simple cît şi modele ale circuitelor complexe;

– nu depinde de tipul simulatorului utilizat;

– permite simularea şi testarea rapidă a modelelor VHDL.

Pe de altă parte un testbench are **dezavantajul** că poate conține erori logice în aceeaşi măsură ca şi proiectul testat. De exemplu, proiectantul crede că testul este corect când, de fapt, e posibil să fi fost conceput greşit şi, astfel, se ajunge ca un model VHDL bun să fie declarat în mod eronat ca având erori! De aceea, la construcția unui testbench trebuie acordată o atenție deosebită, în special în cazul celor complexe.

Din punct de vedere structural un testbench conţine două blocuri (Figura 1):

– un prim bloc îl reprezintă *modelul VHDL suspus testării* ale cărui terminale (intrările şi ieşirile) sunt conectate la semnalele interne din testbench;

– al doilea bloc *generează valori* pentru semnalele conectate la intrările modelului testat şi constă, de exemplu, din instrucţiuni concurente de atribuire pentru semnale sau procese de generare de forme de undă. Acest bloc poate conţine, de asemenea, o serie de instrucţiuni pentru *monitorizarea şi testarea* automată a semnalelor de la ieşire.

Exemplu de instrucţiuni concurente pentru generarea de valori la semnalele de la intrarea modelului:

```
in_model <= '1', '0' after  5 ns, '1' after 20 ns,  '0' after  30 ns;
clk_in <= not clk_in after 10 ns;
```

Prima instrucţiune de atribuire determină ca semnalul `in_model` să ia valoarea '1' la momentul de timp 0, apoi valori inversate la momentele de timp 5ns, 20ns şi, respectiv, 30ns. A doua instrucţiune determină ca semnalul `clk_in` să aibă o formă de undă cu semiperioade egale cu 10ns.

2.1 Clasificarea testbench-urilor

După modul în care sunt verificate semnalele de la ieşirea modelului testat, testbench-urile pot fi împărţite în două categorii:

– testbench **fără monitorizarea automată** a semnalelor de la ieşire (Figura 1). În acest caz, în urma simulării, semnalele de la ieşire sunt verificate manual prin inspecţie vizuală. Acest tip de testbench poate fi utilizat în practică pentru testarea nodelelor VHDL ale unor circuite simple.

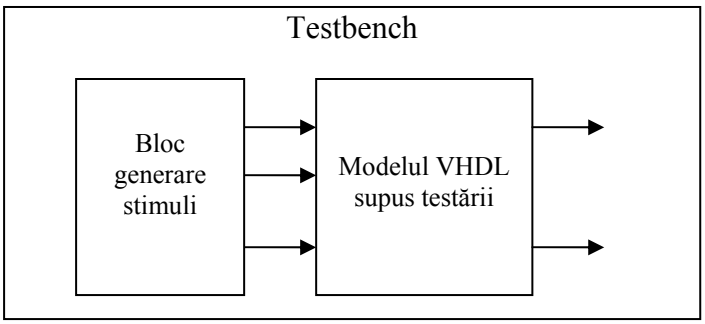

Figura 1 – *Structura unui testbench fără monitorizarea semnalelor de la ieşire*

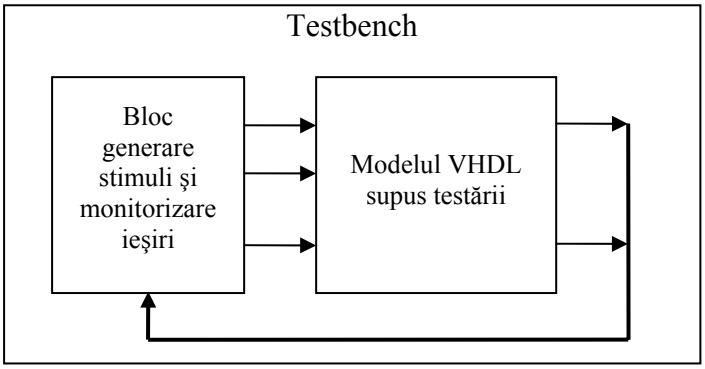

Figura 2 – *Structura unui testbench cu monitorizarea semnalelor de la ieşire*

– testbench **cu monitorizarea automată** a semnalelor de la ieşire (Figura 2). În acest caz, în timp ce se efectuează simularea sunt verificate valorile semnalelor de la ieşire dacă sunt în concordanță cu specificațiile pe care trebuie să le îndeplinească modelul testat. De exemplu, se compară valorile semnalelor de la ieşire cu cele din tabelul de adevăr al circuitului pentru care s-a descris modelul VHDL.

Pentru verificarea şi testarea semnalelor de la ieşirea unui model se utilizează uzual instrucțiunea **assert**. Instrucțiunea **assert** se execută dacă expresia booleană din cadrul său are valoarea false.

Exemplu de utilizare a instrucțiunii assert într-un process dintr-un testbench:

```
process(out_model, in_model)
begin
  assert  (out_model = '1' nand in_model = '1')
      report "Semnalele out si in au valoarea 1 in acelasi timp"
      severity Error;
end process;
```

În exemplul de mai sus se presupune că în specificațiile modelului semnalele `in_model` şi `out_model` de la intrare şi ieşire nu trebuie să se afle simultan în '1'. Dacă în timpul simulării această condiție nu este îndeplinită atunci se va afişa mesajul specificat cu instrucțiunea **report** însoțit de nivelul de severitate (Error).

În cazul **testbench-urilor cu monitorizarea automată** a semnalelor de la ieşire, pe lângă testarea valorilor semnalelor se pot testa şi probleme de timing. De exemplu, se poate verifica dacă un semnal este stabil (nu are nici o tranziție) un anumit interval de timp. Un posibil cod VHDL care include o astfel de verificare este următorul:

```
process
begin
  wait for 20 ns;
  if out_model /='0' or not out_model'stable(5 ns) then
    test_ok<='0';
  end if;
end process;
```

În acest exemplu, instrucțiunea **if** se execută la momentul de timp 20 ns şi semnalul `test_ok` va lua valoarea '0' dacă semnalul `out_model` este diferit de '0' sau dacă nu a fost stabil timp de 5 ns (în intervalul 15-20ns).

3. Testarea unui model VHDL într-un testbench fără monitorizarea automată semnalelor de la ieşire

În continuare se prezintă un posibil testbench pentru simularea şi verificarea modelului sumatorului complet pe un bit, model definit de entitatea *FullAdder* şi arhitectura *ecuatii* în lucrarea de laborator anterioară. Reamintim aici, codul VHDL al modelului sumatorului:

```
entity FullAdder is
port ( X,Y : in Bit;
       Cin:   in Bit;
       Cout: out Bit;
       Sum: out Bit);
end FullAdder ;
```

```
architecture Ecuatii of FullAdder is
begin
      Sum<= X xor Y xor Cin;
      Cout<= (X and Y) or (X and Cin) or (Y and Cin);
end;
```

Codul VHDL şi o primă arhitectură pentru testbench-ul de simulare şi verificare a modelului sumatorului FullAdder este următorul:

(**Fişier Test1_FullAdder.vhd**)

```
1.      entity Test_FullAdder is
2.      end;

3.      architecture Test1 of Test_FullAdder is
4.        component FullAdder
5.          port (X,Y: in Bit; Cin: in Bit;
6.                Cout: out Bit; Sum: out Bit ) ;
7.        end component;
8.        signal  SX, SY, SCin, SCout, SSum: Bit;
9.        for all: FullAdder use entity work.fulladder(ecuatii);

10.     begin
11.       UUT: FullAdder port map (SX,SY,SCin,SCout,SSum);

12.       SX <= not SX after 10 ns;
13.       SY <= not SY after 20 ns;
14.       SCin <= not SCin after 40 ns;
15.     end;
```

Declarația de entitate (liniile 1-2) corespund entității de test (nu conține porturi pentru că nu interacționează cu exteriorul).

În arhitectură, în partea declarativă (liniile 4-9) se declară componenta *FullAdder*. Această componentă reprezintă interfața pentru entitatea *FullAdder* şi arhitectura *ecuatii*. Pentru a asocia în mod explicit componentei *FullAdder* perechea entitate-arhitectură *Fulladder-ecuatii* din biblioteca **work** se utilizează declarația de configurație din linia 9.

Observație: declarația de configurație din linia 9 poate lipsi deoarece unei componente i se asociază în mod automat perechea entitate-arhitectură dacă entitatea şi componenta au acelaşi nume, aceleaşi porturi şi aceeaşi ordine de definire a acestora.

În linia 8 se declară semnalele interne din testbench la care se vor conecta porturile componentei. Pentru semnale se poate utiliza acelaşi nume cu al porturilor (X, Z, Cin, etc). Totuşi, pentru a evidenția diferența dintre porturile componentei şi semnalele din entitatea de test, denumirile semnalelor conțin prefixul S (SX, SY, etc).

După cuvântul cheie **begin** urmează zona de instrucțiuni concurente (liniile 11-14). În această zonă, linia 11 corespunde instrucțiunii de instanțiere în testbench a componentei *FullAdder* ale cărei porturi (X, Y, Cin, Cout şi Sum) se conectează la semnalele SX, SY, şamd.

Liniile 12-14 reprezintă înstrucțiuni de atribuire pentru semnale prin care se generează forme de undă periodice pentru semnalele SX, SY şi Scin de la intrările componentei suspuse testului.

3.1 Aplicație: simularea testbench-ului

1. Editați codul VHDL al testbench-ului de mai sus în fişierul **Test1_FullAdder.vhd.** Fişierul trebuie salvat în directorul de lucru.
2. Compilați fişierul şi corectați eventualele erori.

3. Încărcați în simulator entitatea `Test_FullAdder` împreună cu arhitectura `Test1`. folosind comanda *vsim* sau, din meniul ModelSim, **Simulate => Simulate.**
4. Deschideți ferestrele *Signals* şi *Wave*
5. Adăugați în ferestra Wave toate semnalele de interes (SX, SY, etc)
6. Efectuați simularea pe durata a 100 ns. Remarcați faptul că în cazul simulării unui testbench pentru verificarea modelului sumatorului nu mai este necesar, în prealabil, să se genereze forme de undă la intrări cu comanda *force*.
7. În fereastra Wave urmăriți vizual formele de undă de la intrări şi decideți dacă sunt în concordanță cu specificațiile (tabelul de adevăr al sumatorului)

8. În cadrul fişierului **Test1_FullAdder.vhd** editați o nouă arhitectură denumită *Test2* pentru aceeaşi entitate de test, *Test_FullAdder*. Această arhitectură este identică cu arhitectura *Test1* cu excepția faptului că liniile 12-14 sunt înlocuite cu instrucțiuni **process** pentru a genera aceleaşi forme de undă pentru semnalele de la intrări. Forma arhitecturii *Test2* este următoarea:

```
architecture Test2 of Test_FullAdder is
  ….

begin
  …
Stim_X: process
  begin
    SX <= '0'; wait for 10 ns;
    SX <= '1'; wait for 10 ns;
  end process;
Stim_Y: process
  begin
  SY <= '0', '1' after 20 ns;
  wait for 40 ns;
  end process;

Stim_Cin: process
  begin
  …
  end process;

end;
```

Completați instrucțiunile din procesul cu eticheta `Stim_Cin`, similar cu unul din procesele precedente, pentru a genera forma de undă pentru semnalul SCin. Remarcați diferența dintre procesele Stim_X şi Stim_Y.

9. Recompilați fişierul **Test1_FullAdder.vhd** şi simulați entitatea de test împreună cu arhitectura *Test2*. Vizualizați formele de undă ale semnalelor şi remarcați dacă acestea sunt identice cu cele rezultate în urma simulării cu arhitectura *Test1*.
10. Dacă toate porturile sumatorului şi semnalele din testbench ar fi declarate de tip *std_logic* cu 9 stări, ce diferențe ar apare între formele de undă generate pentru cele două arhitecturi *Test1* şi *Test2*. Se va ține la formularea răspunsului că prima stare a tipului std_logic este 'U' – neinițializat. Cum poate fi corectată deficiența?

4. **Testarea unui model VHDL într-un testbench cu monitorizarea automată semnalelor de la ieşire**

La acest punct se va considera din nou testarea sumatorului pe un bit (FullAdder), dar de această dată circuitul de test (testbench-ul) va fi construit astfel încât să monitorizeze şi să verifice automat în timpul procesului de simulare dacă valorile rezultate la ieşire sunt în concondanţă cu tabelul de adevăr al sumatorului. Codul VHDL al circuitul de test este descris mai jos în fisierul **Test2_Fulladder.vhd**:

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

21 entity Test_FullAdder is end;

25 architecture Driver of Test_FullAdder is
27    component FullAdder
28    port (X,Y: in Bit;
29            Cin: in Bit;
30             Cout: out Bit;
31             Sum: out Bit ) ;
32 end component;
34 signal  X,Y,Cin,Cout,Sum:Bit;

36 begin
38 UUT: FullAdder port map (X,Y,Cin,Cout,Sum);

41 Stimulus: process
42    type Entry is record
43          X,Y,Cin : Bit;
44          Cout,Sum: Bit;
45    end record;
46    type Table is array ( 0 to 7) of Entry;
47    constant TruthTable: Table :=
48    (
--------------X----Y---Cin---Cout---Sum
49          ('0' ,'0', '0',  '0',    '0'),
50          ('0', '0', '1',  '0',    '1' ),
51          ('0', '1', '0',  '0',    '1' ),
52          ('0', '1', '1',  '1',    '0' ),
53          ('1', '0', '0',  '0',    '1' ),
54          ('1', '0', '1',  '1',    '0' ),
55          ('1', '1', '0',  '1',    '0' ),
56          ('1', '1', '1',  '1',    '1' )
57      );
58   begin
59    for i in TruthTable'range loop
60    X <= TruthTable(i).X;
61    Y <= TruthTable(i).Y;
62    Cin <= TruthTable(i).Cin;
63    wait for 1 ns;
64    assert
65          Cout = TruthTable(i).Cout and
66          Sum = TruthTable(i).Sum;
67   end loop;
68  wait;
69  end process ;
71  end;
```

În acest caz corpul arhitecturii circuitului de test (liniile 25-71) corespunde cu cel din Figura 2 şi este alcătuit din unitatea supusă testului (linia 38) şi blocul de generare forme de undă şi testare automată (liniile 41-69). Unitatea supusă testului constă în instanţierea componentei *FullAdder*.

Astfel, componenta *FullAdder*, a cărei interfaţă corespunde cu cea a modelului, este instanţiată în cadrul testbench-ului şi are porturile conectate prin semnale la blocul de generare şi testare forme de undă.. Componenta este declarată la liniile 27-32, semnalele interne prin care instanţa este conectată sînt declarate la linia 34. Sintaxa **port map** conectează porturile componentei la semnalele interne. În acest testbench semnalele interne au fost denumite identic cu porturile componentei.

Blocul de generare şi testare automată a semnalelor de ieşire este modelat cu ajutorul construcţiei *process*. Procesul conţine declaraţia unui tablou (liniile 47-57) care descrie stimulii (semnalele de test aplicabile la intrări) şi semnalele răspuns. Tipul tabloului (*Table*) este declarat la linia 46 ca un tablou cu elemente ale tipului record *Entry*, elementele tipului *Entry* fiind descrise ca tip bit (liniile 43-45)

Blocul de generare şi testare forme de undă aplică stimulii şi verifică semnalele răspuns în secţiunea cu instrucţiuni secvenţiale a procesului (59-68). Instrucţiunile secvenţiale ale procesului sînt executate una după alta, în ordinea apariţiei lor. Bucla *for* (liniile 59-67) aplică succesiv fiecare linie a tabelului semnalelor de intrare şi verifică semnalele de ieşire.

Instrucţiunile de atribuire de semnale (liniile 60-62) aplică intrarea curentă semnalului intern corespunzător, care la rîndul său este conectat la intrarea corespunzătoare unităţii supusă testării.

Specificaţia *wait* (linia 63) suspendă o nanosecundă din timpul simulat, dând astfel timp sumatorului pentru calculul valorilor semnalelor de ieşire.

Specificaţia *assert* verifică îndeplinirea condiţiei (expresia booleana) ce urmează cuvântului rezervat *assert*, în acest caz semnalele de ieşire curente sînt comparate cu răspunsurile aşteptate.

După epuizarea domeniului tabloului, bucla *for* îsi incheie execuţia iar instrucţiunea *wait* (linia 68) determină suspendarea execuţiei procesului. Deoarece *wait* nu conţine nici o clauză de timp procesul este suspendat definitiv.

Datorită faptului că testbench-ul prezentat mai sus efectuează testarea automată a valorilor de la ieşire, în urma simulării nu mai este necesară vizualizarea formelor de undă pentru a decide dacă modelul suspus testului este corect. Este suficient să se urmărească în fereastra principală a *ModeSim* dacă sunt afişate eventuale mesaje de eroare.

4.1 Aplicaţie: simularea testbench-ului cu testare automată a semnalelor de la ieşire

Efectuaţi următoarele:

1. Copiaţi fişierul Test_FullAdder.vhd în directorul de lucru.
2. Compilaţi fişierul, încărcaţi în simulator entitatea FullAdder împreună cu arhitectura driver şi efectuaţi simularea.
3. Urmăriţi în fereastra prncipală *ModelSim* dacă sunt afişate eventuale mesaje de eroare. Întrucât în arhitectura testbenchului instrucţiuea **assert** nu este insoţită de instrucţiunea **report**, eventualele mesaje de eroare sunt afişate sub forma: *Assert violation* însoţit de momentul de timp la care condiţia nu a fost îndeplinită.
4. Efectuaţi cu bună ştiinţă o modificare a unei valori pentru Sum sau Cout din tabelul de adevăr definit în testbench astfel încât, în cursul simulării, să rezulte eroare.
5. Recompilaţi fişierul, repetaţi simularea testbench-ului şi observaţi modul de afişare a erorii în fereastra principală.

6. Pentru circuitul combinațional din Figura 3, care este un multiplexor pentru semnalele *b* şi *c*, iar *a* este intrare de selecție, descrieți modelul acestuia în VHDL.
7. Definiți tabelul de adevăr al multiplexorului
8. Pentru verificarea modelului multiplexorului implementați în VHDL un circuit de testare cu verificarea automată a semnalului de la ieşire.

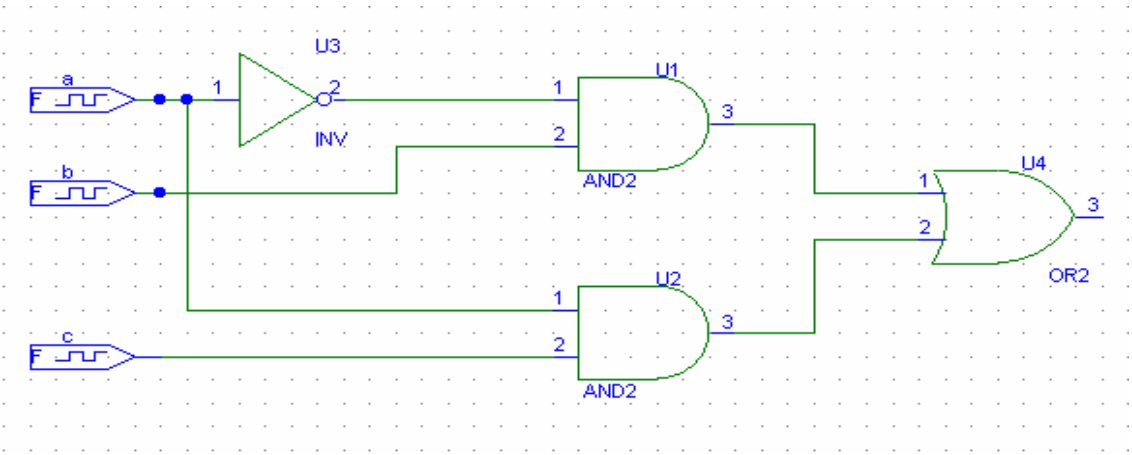

Figura 3. Multiplexor