Lucrarea 3

Modelarea circuitelor combinaţionale

Această lucrare prezintă modul de descriere a circuitelor combinaţionale în Verilog/VHDL, testarea cu *ModelSim*, şi sinteza acestora utilizând sintetizatorul *Leonardo*.

3.1 Scopul lucrării

- Modelarea circuitelor combinaţionale. Exemplificarea circuitelor multiplexor, codificator, sumator.
- Prezentarea şi utilizarea bibliotecii IEEE (VHDL).
- Compilarea şi simularea circuitelor combinaţionale cu *Modelsim*.
- Monitorizarea rezultatelor simulării.
- Evidenţierea hazardului combinaţional.
- Sinteza circuitelor combinaţionale cu *Leonardo*.

3.2 Modelele unor circuite combinaţionale

Circuitul multiplexor realizează selecţia unei linii de intrare în funcţie de semnalele de selecţie active.

Varianta VHDL:

```
-- Multiplexor de 2 biti

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY MUX2x1 IS
  PORT (IN0, IN1, SEL : IN std_logic;
        MUX_OUT       : OUT std_logic);
END MUX2x1;
ARCHITECTURE Behave OF MUX2x1 IS
BEGIN
MUX_OUT = IN1 WHEN (SEL = '1') ELSE IN0;
END Behave;
```

Varianta Verilog:

```
//Multiplexor de 2 biti

module  mux2x1(out, in0, in1, sel);
output out;
input  in0;
input  in1;
input  sel;

assign out = (sel) ? in1 : in0;

endmodule
```

Pentru a testa circuitul multiplexor se va folosi un generator de stimuli parametrizabil care are ca parametru numărul de intrări al dispozitivului de testat, în acest caz multiplexor. Modulul unui generator de stimuli parametrizabil este descris în continuare:

```
module  test_bench (stimulus);
parameter width = 3;
parameter per   = 5;
output [width-1 : 0] stimulus;
reg clk;
reg [width-1 : 0] stimulus;
initial
begin
  clk      <= 0;
  stimulus <= 0;
end
// generator de ceas
always #per clk  <= ~clk;

always @(posedge clk)
  stimulus <= stimulus + 1;
endmodule
```

Observaţii:

- Generatorul de stimuli este de fapt un numărător pe **width** biţi, unde **width** este numărul de intrări ale modulului de testat.
- În figura 3.1 este prezentată structura implementată in interiorul test−bench-ului.
- Prin intermediul specificaţiei *parameter* se transmite pe câţi biţi este numărătorul, şi perioada (**per**), de comutare a semnalelor generate.
- Modificarea parametrilor **width** şi **per** se realizează în modulul **test** prin suprascrierea acestor parametri. În cazul în care aceşti parametrii nu sînt specificaţi ei ramân cu valoarea specificată în **test−bench**.

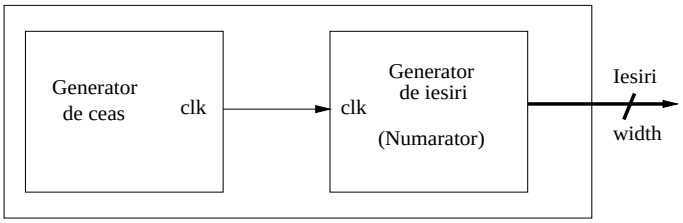

Figura 3.1: Structura unui test bench parametrizabil.

Circuitul codificator generează la ieşire un cuvânt de cod când este activată o linie la intrare. În exemplul următor s-a descris un codificator binar şi unul prioritar.

Codificator binar:

Varianta VHDL:

```
ARCHITECTURE Behave OF ENC IS
BEGIN
ENC: PROCESS (ENC_IN)
  BEGIN
    CASE ENC_IN IS
      WHEN "0000001" => ENC_OUT <= "001";
      WHEN "0000010" => ENC_OUT <= "010";
      WHEN "0000100" => ENC_OUT <= "011";
      WHEN "0001000" => ENC_OUT <= "100";
      WHEN "0010000" => ENC_OUT <= "101";
      WHEN "0100000" => ENC_OUT <= "110";
      WHEN "1000000" => ENC_OUT <= "111";
      WHEN OTHERS => ENC_OUT <= "000";
    END CASE;
END PROCESS;
```

Varianta Verilog:

```
always @(enc_in)
begin
 case (enc_inc)
  7'b0000001 : enc_out <= 3'b001;
  7'b0000010 : enc_out <= 3'b010;
  7'b0000100 : enc_out <= 3'b011;
  7'b0001000 : enc_out <= 3'b100;
  7'b0010000 : enc_out <= 3'b101;
  7'b0100000 : enc_out <= 3'b110;
  7'b1000000 : enc_out <= 3'b111;
  default : enc_out <= 3'b000;
  endcase
 end
```

Pentru testarea codificatorului binar se utilizează acelaşi generator de stimuli ca şi la multiplexor. Deoarece acesta este parametrizabil, singura diferenţă este valoarea parametrului care desemnează numărul de intrari în modulul testat.

Modelarea unui codificator prioritar este diferită de cea a unui codificator binar. În continuare este prezentat un exemplu de codificator prioritar:

Varianta VHDL :

```
ARCHITECTURE Behave OF PRI_ENC IS
BEGIN
PRI_ENC: PROCESS (PRI_ENC_IN)
  BEGIN
  IF PRI_ENC_IN(6) = '1' THEN
    PRI_ENC_OUT <= "111";
  ELSIF PRI_ENC_IN(5) = '1' THEN
    PRI_ENC_OUT <= "110";
  ELSIF PRI_ENC_IN(4) = '1' THEN
    PRI_ENC_OUT <= "101";
  ELSIF PRI_ENC_IN(3) = '1' THEN
    PRI_ENC_OUT <= "100";
  ELSIF PRI_ENC_IN(2) = '1' THEN
    PRI_ENC_OUT <= "011";
  ELSIF PRI_ENC_IN(1) = '1' THEN
    PRI_ENC_OUT <= "010";
  ELSIF PRI_ENC_IN(0) = '1' THEN
    PRI_ENC_OUT <= "001";
  ELSE
    PRI_ENC_OUT <= "000";
 END IF;
END PROCESS;
END Behave;
```

Varianta Verilog:

```
casex (PRI_ENC_IN)
  7'b1xxxxxx : PRI_ENC_OUT = 3'b111;
  7'b01xxxxx : PRI_ENC_OUT = 3'b110;
  7'b001xxxx : PRI_ENC_OUT = 3'b101;
  7'b0001xxx : PRI_ENC_OUT = 3'b100;
  7'b00001xx : PRI_ENC_OUT = 3'b011;
  7'b000001x : PRI_ENC_OUT = 3'b010;
  7'b0000001 : PRI_ENC_OUT = 3'b001;
  default : PRI_ENC_OUT = 3'b000;
endcase
```

Circuitul decodificator poate fi privit ca un identificator de cod. Un cod aplicat la intrare este identificat de către o singură ieşire.În continuare este prezentat un fragment de cod Verilog care modeleaza un decodificator:

```
output [6:0] dcd_out;
input  [2:0] dcd_in;
assign dcd_out = 1 << dcd_in;
```

Decodificarea este de fapt deplasarea unui "1" cu *dcd_in* poziţii spre stânga.

3.3 Prezentarea şi utilizarea bibliotecii IEEE (VHDL)

Circuitele aritmetice se descriu comportamental cu ajutorul operatorilor aritmetici. În VHDL, funcţiile operatorilor aritmetici sunt descrise in biblioteci specifice. În Verilog, operatorii sunt predefiniţi în limbaj.

În cazul VHDL se folosesc tipuri de date cu valori multiple. Pachetul **std_logic_1164** conţine definiţia tipului de date **std_ulogic** care are 9 valori logice. O parte din codul sursă al pachetului **std_logic_1164** este prezentată în continuare.

```
PACKAGE std_logic_1164 IS

    ------------------------------------------------------------------
    -- logic state system  (unresolved)
    ------------------------------------------------------------------
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing  Unknown
                         '0',  -- Forcing  0
                         '1',  -- Forcing  1
                         'Z',  -- High Impedance
                         'W',  -- Weak     Unknown
                         'L',  -- Weak     0
                         'H',  -- Weak     1
                         '-'   -- Don't care
                       );
    ------------------------------------------------------------------
    -- unconstrained array of std_ulogic for use with
    -- the resolution function
    ------------------------------------------------------------------
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> )
    OF std_ulogic;

    ------------------------------------------------------------------
    -- *** industry standard logic type ***
    ------------------------------------------------------------------
    SUBTYPE std_logic IS resolved std_ulogic;

    ------------------------------------------------------------------
    -- unconstrained array of std_logic for use in declaring
    -- signal arrays
    ------------------------------------------------------------------
    TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>)
    OF std_logic;

    ..............................................................

END std_logic_1164;
```

Pachetul **std_logic_1164** nu conţine definirea operatorilor aritmetici. Definirea acestor operatori se găseşte în pachetul **std_logic_unsigned**. În continuare se prezintă un fragment din partea declarativă a acestui pachet.

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

package STD_LOGIC_UNSIGNED is

    attribute builtin_subprogram: string;
```

```
    function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR)
    return STD_LOGIC_VECTOR;
    attribute builtin_subprogram of
        "+"[STD_LOGIC_VECTOR, STD_LOGIC_VECTOR
    return STD_LOGIC_VECTOR]:
    function is "stdarith_plus_uuu";
    ..........................................................

    function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR)
    return STD_LOGIC_VECTOR;
    attribute builtin_subprogram of
        "-"[STD_LOGIC_VECTOR, STD_LOGIC_VECTOR
    return STD_LOGIC_VECTOR]:
    function is "stdarith_minus_uuu";
    ..........................................................

end STD_LOGIC_UNSIGNED;
```

Modelele VHDL şi Verilog ale unui sumator pe n biţi se prezintă în continuare.

Fişierul adder.vhd

```
-- Sumator pe n biţi
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY adder IS
GENERIC (n : integer := 8);
PORT  (s  : OUT std_logic_vector(n-1 DOWNTO 0);
       a  : IN  std_logic_vector(n-1 DOWNTO 0);
       b  : IN  std_logic_vector(n-1 DOWNTO 0));
END adder;

ARCHITECTURE adder OF adder IS
BEGIN
  s <= a + b;
END adder;
```

Fişierul adder.v

```
// Sumator pe n biţi
module adder (s, a, b);
parameter n = 8;
output [n-1:0] s;
input  [n-1:0] a;
input  [n-1:0] b;

assign s = a + b;
```

```
endmodule
```

Observatii:

- De remarcat simplitatea codului Verilog faţă de cel VHDL. Modelul VHDL trebuie sa fie obligatoriu însoţit şi de pachetele în care se declară tipurile de date şi funcţiile operatorilor aritmetici.

3.4 Compilarea şi simularea circuitelor combinaţionale

1. Verificaţi existenţa directorului *C:\Home* şi a dreptului de scriere în acesta. Ştergeţi eventualele fişiere şi subdirectoare aflate în acest director. Aceste acţiuni se pot executa din sistemul de operare, folosind Explorer.
2. Copiaţi fişierele Verilog (fişiere cu extensia ".v") din directorul *\asd_hdl\exemple\l3* în directorul *C:\home*. Acest director va fi director de proiect în *ModelSim*. Lansaţi în execuţie *ModelSim*. Din fereastra principală selectaţi:

   Fereastra principală: File > Change Directory...

   şi selectaţi directorul *C:\home* ca director curent de lucru.
3. Înainte de a putea compila un proiect, trebuie să creati biblioteca de proiect (denumită **work**) în directorul curent. Aceasta se poate face în două variante:

   Fereastra principală: Design > Create a New Library...

   sau

   Prompter: vlib work
4. Ca proiect exemplu se alege multiplexorul. Pentru a compila fişierele Verilog selectaţi butonul  . Va apare o casetă de dialog de unde selectaţi următoarele fişiere pentru a fi compilate:
   (a) **mux.v** care reprezintă descrierea comportamentală a multiplexorului;
   (b) **mux_tb.v** care reprezintă generatorul de vectori de test;
   (c) **mux_test.v** care reprezintă modulul de test în care este instanţiat multiplexorul (numele instanţei este **DUT** - *Device Under Test*) şi generatorul de stimuli (numele instanţei este **TB** - *Test−Bench*).

   Urmăriţi în fereastra principală eventualele mesaje de eroare sau atenţionare rezultate în urma compilării.
5. Lansaţi simulatorul selectând butonul *Load Design*  . Va apare o casetă de dialog de unde selectaţi **mux_test**. Confirmaţi selecţia prin apăsarea butonului *Load*. Se va simula în continuare modulul **mux_test** care este prezentat în figura 3.2.

   După simulare, verificaţi dacă în fereastra principală nu sunt mesaje de eroare sau atenţionare.
6. Deschideţi fereastra *Signals*:

   Fereastra principală: View > Signals

   selectaţi *View > Wave > Signals in Region*

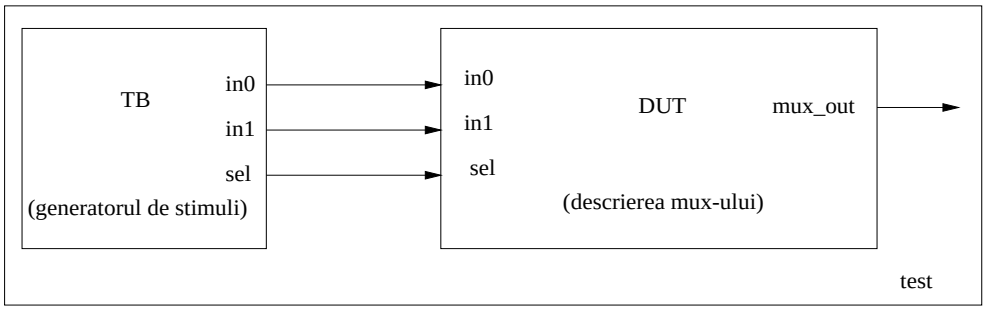

Figura 3.2: Structura modului *test* care va fi simulat.

7. Activaţi fereastra *Wave* pentru a vizualiza formele de undă ale semnalelor. Apăsaţi butonul *RUN*  pentru a rula simularea pentru un timp selectat (implicit 100 ns).
   Prompter: run
   Fereastra principală: Run > Run 100 ns
8. Studiaţi funcţionalitatea modelului descris, pe baza formelor de undă a semnalelor de ieşire, în funcţie de configuraţia semnalelor de intrare.

3.5 Evidenţierea hazardului combinaţional

Pentru a pune în evidenţă hazardul combinaţional se consideră circuitul din figura 3.3. La intrarea unei porţi AND se aplică semnalul **a** şi semnalul complementar **an** care a fost întârziat printr-o poartă inversoare.

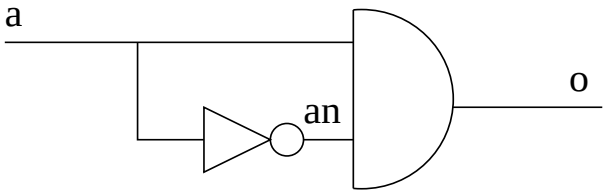

Figura 3.3: Circuit combinaţional care generează hazard combinaţional.

Modelul VHDL al circuitului din figura 3.3 este prezentat în continuare:

Fişierul hazard.vhd

```
-- Circuit combinaţional care generează hazard combinaţional
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;


ENTITY hazard IS
PORT ( o : OUT std_logic;
       a : IN  std_logic );
END hazard;

ARCHITECTURE hazard OF hazard IS
  SIGNAL an : std_logic;
```

```
BEGIN
 an <= NOT a;
 o  <= a AND an;
END hazard;
```

Modulul de test, descris în fişierul **hazard_test.vhd** conţine două instanţe:

- circuitul supus testării descris în fişierul **hazard.vhd**;
- generatorul de vectori de test descris în **hazard_tb.vhd**.

Pentru vizualizarea hazardului combinaţional se deschide fereastra *List*:

Fereastra principală: View > Signals, View > List > Signals in Region

În continuare se va simula "pas cu pas" folosind butonul *Step* din fereastra principală (figura 3.4).

Figura 3.4: Butonul *Step* din fereastra principală a *ModelSim*.

Este de aşteptat ca ieşirea să fie permanent '0'. Din cauza hazardului combinaţional, în momentul în care intrarea **a** comută în '1' ieşirea **o** este diferită de '0', aşa cum rezultă din figura 3.4.

De remarcat faptul că în fereastra *wave* se poate observa pulsul dde hazard combinaţional ca o linie verticală. Această linie nu are lăţime (în timp) indiferent de factorul de zoom.

3.6 Monitorizarea rezultatelor simularii

În cazul circuitelor foarte complexe, este aproape imposibil ca testarea să se facă urmărind formele de undă. Testarea acestor circuite se face prin monitorizarea rezultatelor. Aceasta presupune compararea rezultatelor furnizate de circuitul supus testării (DUT) cu rezultatele unui monitor (circuit martor). Dacă descrierea DUT trebuie să fie sintetizabilă, monitorul este un model comportamental care descrie funcţionarea circuitului la nivelul porturilor de intrare-ieşire (prin intermediul funcţiilor de transfer).

Schema bloc a unui modul de test care permite monitorizarea rezultatelor este prezentaă în figura 3.6.

Funcţiile modulului de monitorizare:

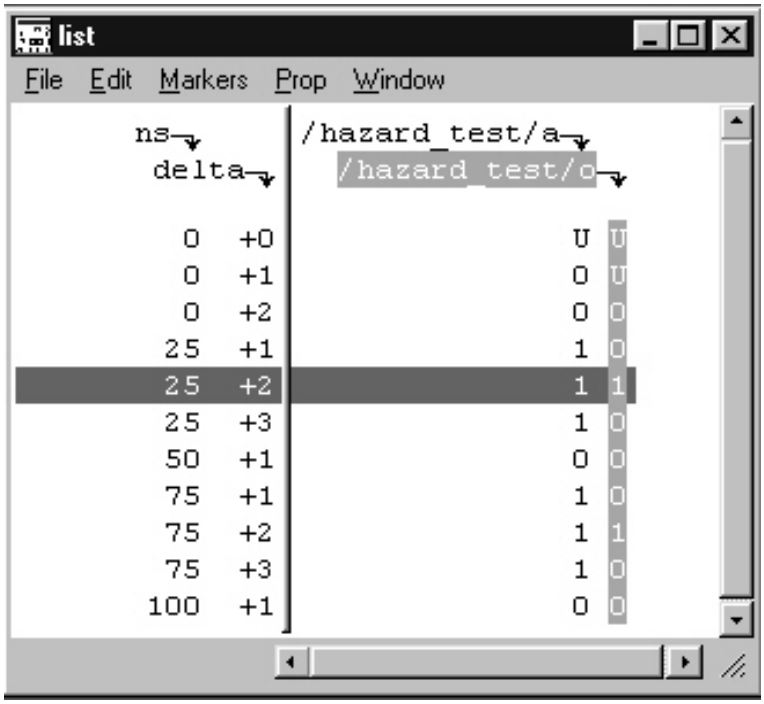

Figura 3.5: Punerea în evidenţa a hazardului combinaţional în fereastra *List*.

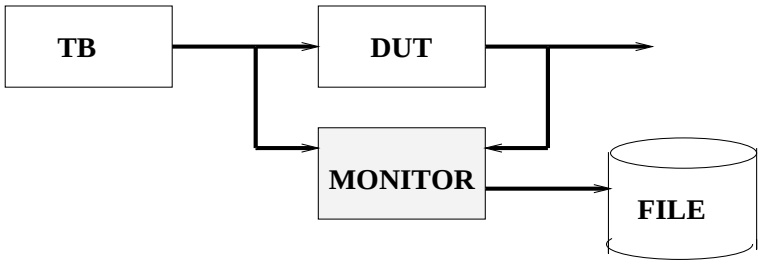

Figura 3.6: Modul de test pentru monitorizarea rezultatelor.

- verificarea funcţiei de transfer a circuitului supus testării;
- emiterea de mesaje de eroare sau atenţionare referitoare la condiţiile de intrare (ex. îndeplinirea condiţiilor de set-up sau hold la o memorie) sau de ieşire (ex. terminarea cu eroare sau succes a monitorizării);
- interfaţarea cu fişiere (ex. scrierea în mod text a rezultatelor).

Monitorul primeşte la intrare semnalele generate de generatorul de semnale de test (TB) şi ieşirile circuitului supus testării (DUT). Pe baza semnalelor generate de TB se calculează ieşirile dorite, după care se compară aceste rezultate cu rezultatele furnizate de DUT, într-un fişier text se inscriu rezultatele şi se semnalizează prezenţa erorilor de funcţionare.

Fişierul adder_monitor.v

```
// Monitor pentru sumatorul de n biţi
module adder_monitor ( s, a, b, end_stim );
parameter n          = 8;
parameter out_file   = "adder.txt";
parameter eval_delay = 1;
input  [n-1:0] s;
input  [n-1:0] a;
input  [n-1:0] b;
input          end_stim;
reg            error;
wire   [n-1:0] s_m;
wire           co_m;
integer        out_file_id;

// scriere header fişier de ieşire
initial
begin
  error <= 0;
  out_file_id = $fopen(out_file);
  $fdisplay(out_file_id, "   a  b     s      error time");
  $fdisplay(out_file_id, "============================");
end


// descrierea comportamentală a DUT
assign s_m = a + b;

// compararea rezultatelor şi inscrierea în fişier
always @( s )
begin
  #eval_delay
  if((s_m !== s) && (~error))
    error <= 1;
  if(~end_stim)
   $fdisplay(out_file_id, "   %h %h    %h     %0b     %0t",
             a, b, s, error, $time);
end

// semnalizare terminare monitorizare cu eroare sau succes
always @(end_stim)
  if (end_stim)
    if(error)
      begin
        #eval_delay
        $fdisplay(out_file_id, "Simularea s-a terminat la momentul %0t
                  cu EROARE.", $time);
        $fclose(out_file_id);
```

```
            $display("%M : Simularea s-a terminat la momentul %0t
                     cu EROARE.", $time);
          end
        else
          begin
            #eval_delay
            $fdisplay(out_file_id, "Simularea s-a terminat la momentul %0t
                      cu SUCCES.", $time);
            $fclose(out_file_id);
            $display("%M : Simularea s-a terminat la momentul %0t
                     cu SUCCES.", $time);
          end
    endmodule
```

Pentru a simula sumatorul se parcurg următorii paşi:

- compilaţi fişierele
  - **adder.v**
  - **adder_tb.v**
  - **adder_monitor.v**
  - **adder_test.v**
- simulaţi modulul **adder_test**:
  Prompter: vsim adder_test
- deschideţi fereastra cu formele de undă:
  Fereastra principală: View > Signals, Wiev > Vawe > Signals in Region
- rulaţi **3000 us** prin comanda:
  Prompter: run 3000 us

Rezultatele monitorizării se înscriu în fişierul **adder.txt**. În fereastra principală din *Modelsim* se va semnaliza încheierea monitorizării cu succes sau eroare, ca în figura 3.7, acelaşi mesaj fiind scris şi în fişierul cu rezultate.

Fişierul adder.txt

```
   a  b     s      error time
============================
   00 00    00     0     2
   01 00    01     0     12
   02 00    02     0     32
   03 00    03     0     52
   04 00    04     0     72
   05 00    05     0     92
   ........................
   fd ff    fc     0     1310652
   fe ff    fd     0     1310672
   ff ff    fe     0     1310692
   00 00    00     0     1310712
Simularea s-a terminat la momentul 1310732 cu SUCCES.
```

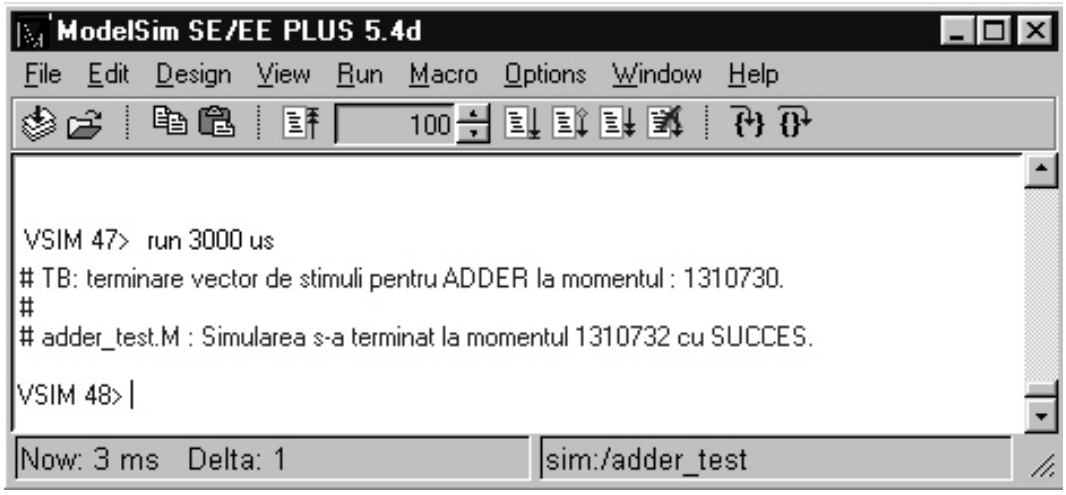

Figura 3.7: Afişarea rezultatului monitorizării în fereastra principală a *ModelSim*.

3.7 Sinteza circuitelor combinaţionale cu *Leonardo*

- **Sinteza** este procesul prin care modelul comportamental descris în Verilog sau VHDL este convertit la o structură dintr-o anumită tehnologie, rezultând un fişier de tip netlist în care este descrisă reprezentarea la nivel de porţi logice a modelului comportamental. Altfel spus, prin sinteză se face trecerea de la reprezentarea **RTL** (**R**egister **T**ransfer **L**evel) la reprezentarea **G**ate **L**evel. Sintetizatorul prezentat în continuare se numeşte *Leonardo Spectrum* şi aparţine firmei **Exemplar Logic Mentor Graphics**.

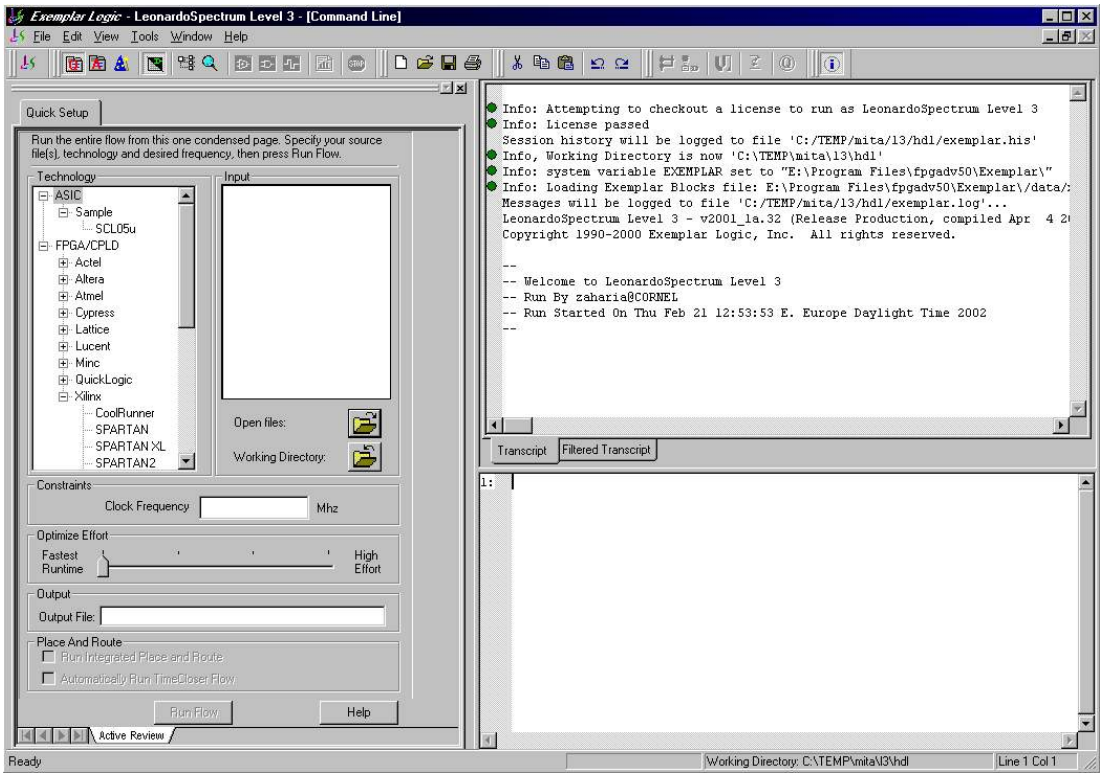

Figura 3.8: Fereastra principală *Leonardo*.

- **Etapele necesare pentru o sinteza cu** *Leonardo.* Pentru înţelegerea mai uşoră a acestui sintetizator s-a optat pentru parcurgerea acestorl etape pe un proiect. Ca exemplu se consideră multiplexorul. Acest circuit va fi implementat pe un circuit programabil de tip **FPGA VirtexII Xilinx**.
- Lansaţi în execuţie programul *Leonardo.* Va apare fereastra principală prezentată în figura 3.8.
- Setati tehnologia pe care se va face implementarea proiectului:
  Fereastra principala Technology > FPGA/CPLD > Xilinx > VirtexII
- Setati directorul de lucru:
  Fereastra principală Working Directory
  Va apare o fereastră de dialog în care se selectează directorul de lucru.
- Pentru a introduce fişierele cu descrierea HDL a proiectului selectaţi:
  Fereastra principală Open Files
  În fereastra *Input*, va apare lista cu fişierele proiectului care urmează a fi sintetizat.
- În cazul în care setările anterioare au fost realizate corect se va activa butonul *Run Flow.*
- Lansaţi sinteza selectând butonul *Run Flow* 

. Urmăriţi eventualele mesaje de eroare sau atentionare în fereastra *Transcript.* În această fereastră se vor afişa informaţii legate de design şi rapoarte privind performaţele circuitului implementat (aria ocupată şi frecvenţa maximă).
- Pentru a vizualiza structura rezultată în urma sintezei selectaţi butonul *RTL*  din *Toolbar.* În urma acestei acţiuni va apare o fereastră ca cea din figura 3.9.

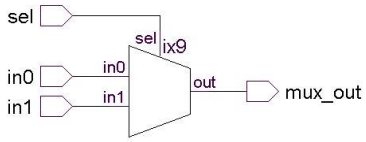

Figura 3.9: Reprezentarea RTL a circuitului sintetizat.

- Pentru a vizualiza implementarea pe circuitul VirtexII selectaţi butonul *Gate*  .
- Pentru a realiza o sinteză optimală selectaţi butonul *Advanced*  . Va aparea o fereastră ca cea din figura 3.10. În această fereastră există o colectie de sub-ferestre în care se pot seta mai multe opţiuni de optimizare a implementării proiectului. Aceste optiuni sunt împărţite în două categorii:
  - opţiuni independente de tehnologie care actionează în etapa de sinteză logică;
  - opţiuni dependente de tehnologie, care acţionează în etapa de sinteză la nivel gate level.
- Rezultatul sintezei este un fişier netlist care este reprezentarea la nivel de porti logice a proiectuluiului. Formatul implicit în care se salvează netlistul este EDIF (**E**lectronic **D**esign **I**nterchange **F**ormat).

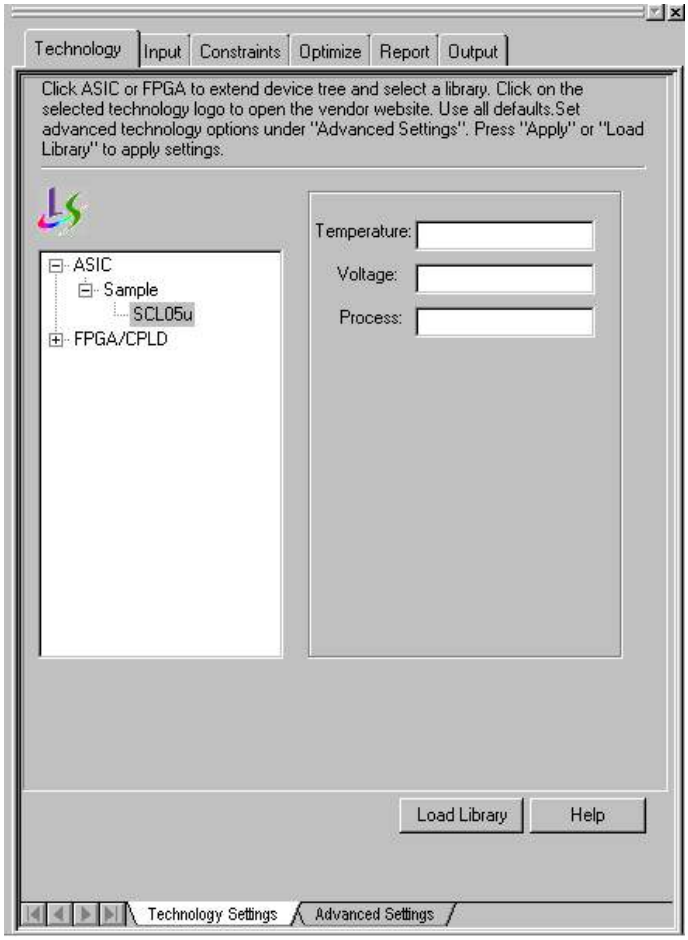

Figura 3.10: Fereastra de setări avansate pentru optimizarea sintezei.

3.8 Teme de laborator

1. Pe baza celor prezentate în această lucrare de laborator, modelaţi un circuit comparator care sa genereze următoarea ieşire: în cazul în care cele două intrări sunt egale semnalul de ieşire să fie activ în "1" logic, 0 în rest. Intrările să fie pe 4 biţi iar testarea circuitului să se realizeze utilizînd un generator parametrizabil.
2. Scrieţi un model structural al unui mux 4:1 folosind multiplexoare 2:1. Realizaţi un mux parametrizabil folosind specificaţia *parameter*.
3. Explicaţi următoarea specificaţie Verilog:

```
always @ (a or b or c) begin
  e = (a|b)&(c|d)
end
```

4. Explicaţi ce este greşit în codul Verilog următor:

```
if (i > 0)
  if (i<2)
    $display ("i este 1");
else
  $display ("i este mai mic ca 0");
```