```
    if (SH)
      DO <= {DO[2:0],1'b0};

endmodule
```

Schema acestui registru, sintetizat cu *Leonardo* este prezentată în figura 6.2.

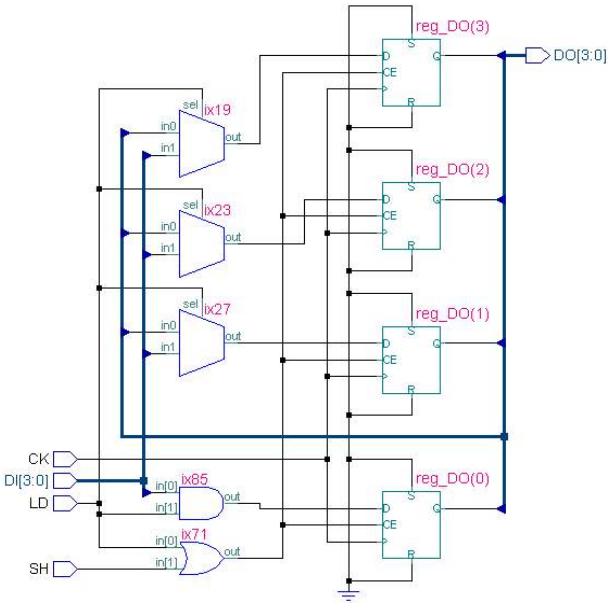

Figura 6.2: Schema după sinteza cu *Leonardo* a unui registru de deplasare cu încărcare şi ieşiri paralele.

Observaţii:

- Semnalul de ceas nu este prezentat în schema bloc a registrului. Acesta este un semnal implicit. În modelul prezentat registrul este senzitiv pe frontul crescător al semnalului de ceas.
- Intrarea de memorare este prioritară celei de deplasare.
- Deplasarea la stânga s-a realizat prin concatenarea celor mai puţin semnificativ biţi ai registrului cu un bit zero.

Exerciţii:

- Modificaţi modelul prezentat pentru a fi senzitiv pe frontul descrescător al semnalului de ceas.
- Modificaţi modelul prezentat pentru ca semnalul SH să fie prioritar.
- Modificaţi modelul prezentat pentru ca deplasarea să se facă spre dreapta.
- Cărei operaţii aritmetice îi corespunde deplasarea spre stânga/dreapta?
- Realizaţi un modul pentru testarea acestui tip de registru.

6.3 Registrul serie

Schema bloc a unui registru serie este prezentată în figura 6.3.

Figura 6.3: Schema bloc a unui registru serie.

Un registru serie funcţionează astfel:

- Dacă semalul de deplasare (Shift - SH) este activ, atunci conţinutul registrului este deplasat spre dreapta cu o poziţie. Pe poziţia cea mai semnificativă se va memora semnalul de intrare serial (SI).
- Iesirea este reprezentată de bitul cel mai puţin semnificativ.

Descrierea Verilog a unui astfel de registru este prezentată în continuare:

```
module serial_reg (SI, SH, CK, SO);

parameter width = 4;

input SI;
input SH;
input CK;
output  SO;

reg  [width-1:0] state;

always @(posedge CK)
  if (SH)
    state <= {SI, state[width-1:1]};

assign SO = state [0];

endmodule
```

Schema acestui registru, sintetizat cu *Leonardo* este prezentată în figura 6.4. **Observaţii:**

- Numărul de tacte necesar pentru ca semnalul de intrare să fie prezent la ieşire este egal cu lătimea registrului. Aceasta este dată de parametrul "width".
- Registrele de deplasare sunt folosite pe larg în domeniul comunicaţiilor, când este necesar ca datele să se transmită serial.

Exerciţii:

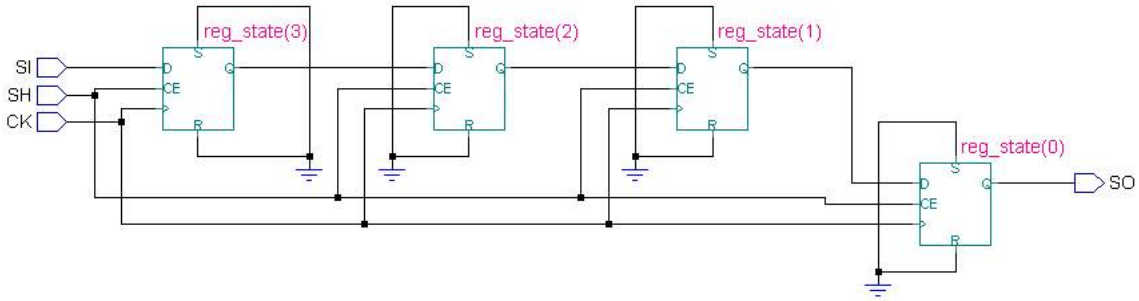

Figura 6.4: Schema după sinteza cu *Leonardo* a unui registru serie.

- Modificaţi numărul de biţi ai registrului. Cum se modifică funcţionarea sistemului?
- Modificaţi modelul prezentat pentru ca deplasarea să se facă spre drepta. Cum se modifică funcţionarea sistemului?
- Constuiţi un registru cu încărcare serie şi paralelă şi cu ieşire serie şi paralelă.

6.4 LFSR (Linear feed-back shift register)

Un LFSR (*Linear feed-back shift register*) este un registru de deplasare care are în plus o poartă XOR. Acest tip de construcţie poate genera o secvenţă de stări pseudoaleatoare. Numărul de stări generate de un LFSR cu lăţime de N biţi este de $2^N - 1$, pe când a unui numărător este de 2^N stări.

La intrarea porţii XOR se introduc biţii corespunzători polinomului care caracterizează circuitul. Acest polinom este dependent de lăţimea registrului. De exemplu, pentru o lăţime de 8 biţi, polinomul caracteristic este: $f(x) = x^8 + x^6 + x^5 + x + 1$.

Schema bloc a unui LFSR de 8 biţi este prezentată în figura 6.5.

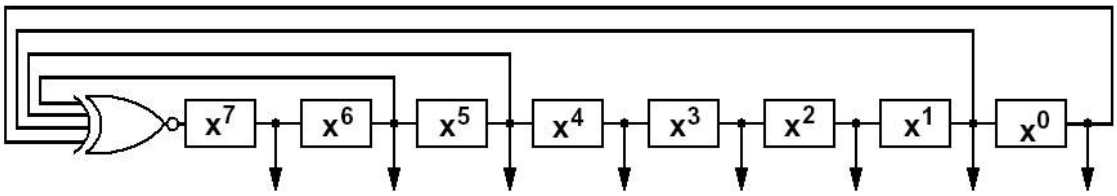

Figura 6.5: Schema bloc a unui LFSR de 8 biţi.

Descrierea Verilog a unui astfel de registru este prezentată în continuare:

```
module LFSR_8 (RST, CK, DO);

input           RST;  // semnal de reset asincron
input           CK;
output  [7:0]   DO;

reg     [7:0]   DO;

always @(posedge CK or posedge RST)
  if (RST)
    DO <= 8'b0;
  else
    DO <= {~(DO[6]^DO[5]^DO[1]^DO[0]),DO[7:1]};

endmodule
```

Observaţii:

- Polinomul corespunzător fiecărei lăţimi a registrului de deplasare poate fi găsit în *"DesignWare Foundation Library Databook, Volume 1"* editat de *Synopsys Inc.*, secţiunea *"Sequential Overview"*. Pentru mai multe detalii consultaţi *www.synopsys.com.*
- Starea "11...11" este o stare stabilă. Dacă registrul este adus în această stare, el nu mai comută. De aceea, trebuie avut grijă la valoarea cu care se iniţializează registrul.
- Registrele de tip LFSR pot fi folosite pentru verificarea corectitudinii unei secvenţe de biţi. Astfel, dacă poarta XOR are o intrare suplimentară pentru data de intrare (de verificat),

atunci succesiunea stărilor registrului este dirijată şi de această intrare. Dacă la un moment dat, semnalul de intrare este diferit de cel aşteptat, atunci evoluţia automatului este cu totul diferită decât cea dorită. După terminarea setului de date, este foarte probabil ca starea LFRS-ului să fie diferită de cea aşteptată (şi determinată anterior, pentru un semnal de intrare corect).

Exerciţii:

- Adăugaţi circuitului prezentat facilitatea de LOAD.
- Încărcaţi starea 8'hFF în LFSR. Observaţi funcţionarea acestuia.
- Construiţi un LFSR cu lătime de 4 biţi. Polinonul caracteristic este: $f(x) = x^4 + x + 1$.
- Sintetizaţi un LFSR de 8 biţi cu *Leonardo* şi determinaţi aria ocupată pe un circuit Xilinx XC2V1000.
- Construiţi un numărător de 8 biţi. Comparaţi aria ocupată de aceasta cu cea ocupată de LFSR cu acelaşi număr de biţi.

6.5 Registru care implementează algoritmul CRC32

Algoritmul CRC32 (*32-bit Cyclyc Redundancy Check*) este folosit pe scară largă pentru detectarea integrităţii datelor. Acest algoritm detectează cu o bună aproximaţie dacă datele transmise sunt eronate sau nu, însă nu poate detecta care dintre cuvintele transmise au erori.

Funcţionarea algoritmului este descrisă de polinomul caracteristic. Pentru CRC-32 acesta este: $P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

Pentru studiul funcţionării unui registru care implementează algoritmul CRC32 se va apela la sursele disponibile pe Internet.

Compania EASICS pune în domeniul public (la adresa *http://www.easics.com*) un utilitar pentru generarea automată a funcţiilor VHDL şi Verilog pentru CRC32.

Exerciţii:

1. Deschideţi browser-ul la adresa www.easics.com.
2. Selectaţi fereastra "Web tools", rubrica "CRC Tools". Va apărea o fereastră similară cu cea din figura 6.6.
3. Setaţi polinomul caracteristic "CRC32 Ethernet / AAL5". Setaţi lăţimea bus-ului de 8 biţi.
4. Generaţi modelul Verilog al funcţiei de calcul al CRC.
5. Completaţi modelul pentru a avea funcţia de registru (senzitiv pe frontul crescător al ceasului) cu reset.
6. Studiaţi funcţionarea circuitului rezultat dacă la intrare datele au valoarea 8'b0. Cum diferă comportarea circuitului dacă iniţializarea acestuia se face cu 32'h00000000 sau cu 32'hFFFFFFFF?

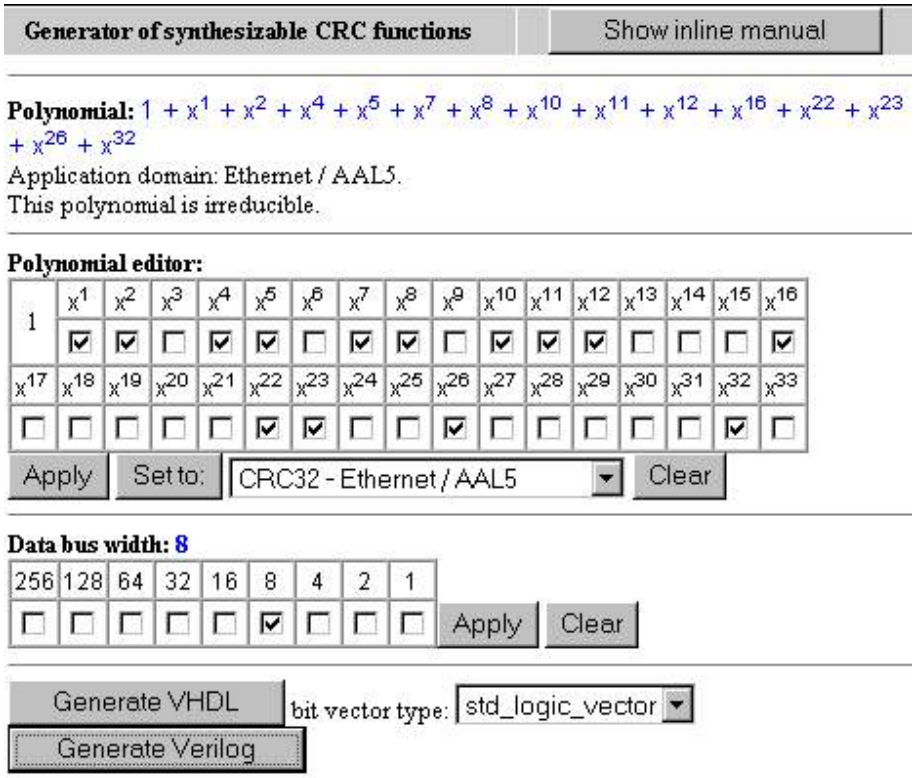

Figura 6.6: Fereastra de generare a funcţiei CRC-32.

6.6 Mediu de simulare cu stimuli aleatori şi detector de erori

În multe cazuri generarea de vectori de test pentru simularea modelelor HDL este un un proces complicat datorită faptului ca este dificil de a testa toate combinaţiile care pot apărea în practică.

De aceea, uneori este utilă realizarea unui generator de stimuli aleatori sau pseudoaleatori.

De asemenea, un mediu de testare complet trebuie să conţină un bloc de detectare a erorilor care să analizeze stimulii primiţi de DUT precum şi ieşirile cestuia şi să semnalizeze apariţia oricărui tip de erori.

Dacă pentru circuite simple verificarea funcţionalităţii se poate face prin analiza formelor de undă, pentru circuite complexe un bloc de detecţie a erorilor este absolut necesar.

Schema bloc al unui mediu de testare care conţine un generator de stimuli aleatori şi un detector de erori este prezentată în figura 6.7.

Generatorul de stimuli aleatori poate fi realizat folosind în Verilog funcţia *$random*.

Functţia *$random* generează un număr aleator la fiecare apel (de dimensiunea unei variabile de tip integer - 32 biţi). Sintaxa funcţiei *$random* este:

```
$random;
$random (<seed>);
```

Parametrul <seed> este de tip *inout* si este folosit pentru a controla numerele generate de

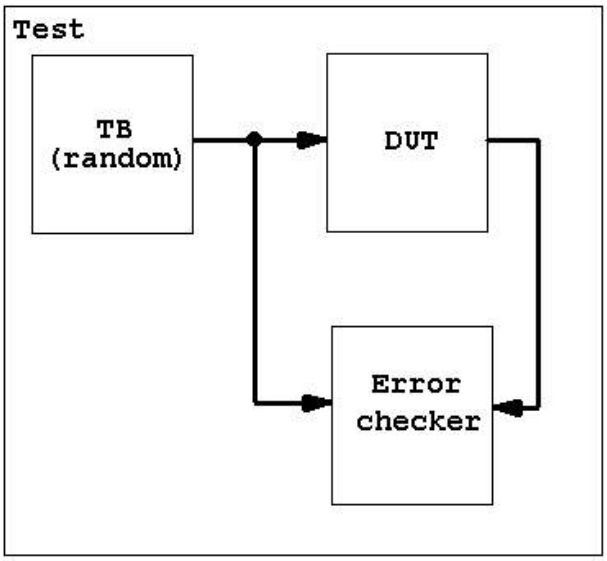

Figura 6.7: Mediu de simulare cu generator de stimuli aleatori şi detector de erori.

funcţia *$random*. Un argument pentru <seed> poate fi un registru, întreg sau variabilă de timp si trebuie sa primească o valoare înaintea apelului funcţiei *$random*.

Un astfel de modul este prezentat în continuare:

```
module TB_random (ck,out);

parameter width = 8;

output              ck;
output [width-1:0] out;
reg                 ck;
reg    [width-1:0] out;

initial ck <= 0;
always #5 ck <= ~ck;

always @(posedge ck)
  out <= $random;

endmodule
```

Observaţii:

- Modelul prezentat este parametrizabil, deci poate fi folosit pentru orice număr de biţi necesari la intrarea circuitului de testat.
- Circuitul generează şi stimuli de ceas, pentru testarea circuitelor secvenţiale.

Exerciţii:

- Care este frecvenţa semnalului de ceas? Introduceţi această frecvenţă ca parametru în codul Verilog.

- Modificaţi circuitul prezentat astfel încât să tipărească pe ecran şi într-un fişier secvenţele de date generate. Pentru tipărire se folosesc funcţiile *$display* şi *$fdisplay*.

Detectarea erorilor pentru un modul Verilog se realizează cu ajutorul unui model care primeşte ca intrare atât intrarea DUT cât şi ieşirea acestuia. Detectorul de erori simulează funcţionarea circuitului de testat şi semnalează diferenţele exstente între funcţionarea DUT şi cea ideală.

Pentru registrul paralel prezentat anterior, un astfel de detector de erori poate fi descris astfel:

```
module error_checker (DI, LD, SH, CK, DO);

input [3:0] DI;
input LD;
input SH;
input CK;
input [3:0] DO;

reg [3:0] temp_DO;
reg [3:0] temp_DI;

always @(posedge CK) begin
  temp_DI = DI;
  temp_DO = DO;

//verifică funcţionarea cnd LD este activ
  if (LD & ~SH) begin
    #1 if (temp_DI != DO)
          $display ("Erorare LOAD la momentul: %t", $time);
  end
  else
//verifică funcţionarea cnd LD şi SH este activ
  if (LD & SH) begin
    #1 if (temp_DI != DO)
          $display ("Erorare LOAD şi SHIFT la momentul: %t", $time);
  end
  else
//verifică funcţionarea cnd SH este activ
  if (~LD & SH) begin
    #1 if ((temp_DO[2:0] != DO[3:1]) || (DO[0] != 1'b0))
          $display ("Erorare SHIFT la momentul: %t", $time);
  end
  else
// Verifică funcţionarea cnd nici un semnal
// de control nu este activ
  if (~LD & ~SH) begin
    #1 if (temp_DO != DO)
       $display ("Erorare (nici un semnal de control) la momentul: %t", $time);
  end
end

endmodule
```

Observaţii:

- Întârzierea (#1) este necesară pentru ca circuitul să-şi schimbe starea.
- Variabila *$time* întoarce valoarea efectivă a timpului de simulare.
- Funcţia *$display* afişează pe ecran un mesaj.

- Deşi codul scris pentru detectorul de erori este mai voluminos decât cel al circuitului de testat, acesta are sens, pentru că oferă o comparaţie a două descrieri diferite ale aceluiaşi circuit. În plus, detectorul de erori poate conţine construcţii nesintetizabile sau care ar necesita o arie mare sau frecvenţă prea mică dacă ar fi realizate fizic.

Exerciţii:

- Realizaţi modulul de test care să includă registrul paralel, generatorul de stimuli si detectorul de erori.
- Realizaţi un detector de erori pentru registrul serie şi pentru numărător.
- Realizaţi un registru RS de N biţi şi verificaţi funcţionarea acestuia.
- Folosiţi în locul generatorului de stimului aleatori un LFSR. Poate fi testat un circuit folosind acest generator de stimuli?

Lucrarea 7

Modelarea memoriilor

Această lucrare prezintă modul de descriere a memoriilor în Verilog

7.1 Scopul lucrării

- Modelarea memoriilor cu un port.
- Modelarea memoriilor cu două porturi.
- Modelarea memoriilor FIFO, cu un semnal de ceas.

7.2 Modelul unei memorii cu un singur port

Memoriile cu un singur port permit, la un moment dat, accesul la o singură locaţie de memorie fie pentru citire, fie pentru scriere. Structura şi semnificaţia porturilor unei astfel de memorii este prezentată în tabelul 7.1.

Port	*Dir*	*Dimensiune*	*Semnificaţie*
clk	in	1	Ceasul memoriei activ pe frontul pozitiv
addr	in	8	Adresa memoriei
di	in	16	Datele de intrare în memorie
ce	in	1	Semnal de selecţie ă memoriei
we	in	1	Semnal de validare a scrierii
oe	in	1	Semnal de activarea ieşirilor
dout	out	16	Datele de ieşire din memorie

Tabelul 7.1: Semnificaţiile porturilor unei memorii cu un singur port.

Operaţiile de scriere şi citire a memoriei sunt caracterizate prin formele de undă din figurile 7.1 respectiv 7.2:

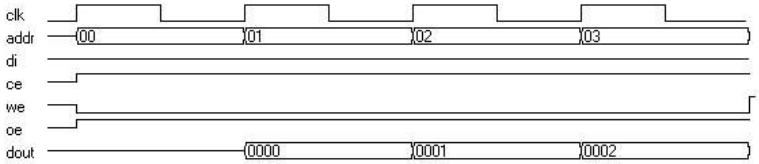

Figura 7.1: Operaţia de citire

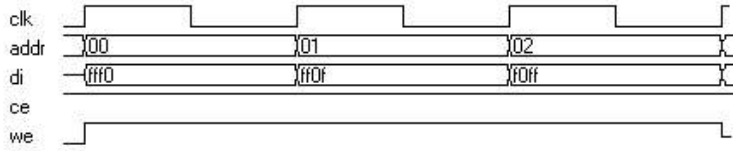

Figura 7.2: Operaţia de scriere.

Modelul Verilog al unei memorii cu un port este prezentat în continuare:

```
module single_port_mem (clk, addr, di, ce, we, oe, dout );
// declaraţia porturilor
input               clk;   // ceasul memoriei
input  [7:0]        addr;  // bus de adrese
input  [15:0]       di;    // bus date de intrare
input               ce;    // semnal de selecţie a memoriei
input               we;    // semnal de validarea scrierii
input               oe;    // semnal de activarea ieşirii
output [15:0]       dout;  // bus date de ieşire
reg [15:0]       dout;

// declaraţia de memorie
reg [15:0] mem [0:255];    // memorie de 256 de locaţii a câte 16 biţi

// memorie sincronă
always @(posedge clk)
  if(ce) begin
    if(we)
      mem[addr] <= di;     // scrierea în memorie a datelor de pe
    else                   // bus-ul de intrare di
      if(oe)
        dout <= mem[addr]; // citirea din memorie
  end
  else
   dout <= 16'bz;         // când memoria nu este selectată
                           //portul de date de ieşire este în High Z
// iniţializarea memoriei
$readmemb ("mem_content.dat", mem);
endmodule
```

Pentru testarea unei memorii cu un singur port se realizează un mediu de testare în care se