Lucrarea 5

Modelarea automatelor sincrone

Această lucrare prezintă modurile de descriere a automatelor Mealy şi Moore în Verilog.

5.1 Scopul lucrării

- Modelarea automatelor Mealy şi Moore.
- Folosirea automatelor pentru controlul căii de date a unui sistem digital.
- Circuite formatoare de impulsuri.
- Circuite pentru recunoaştere de pattern.

5.2 Modelarea automatelor Mealy şi Moore

Automatele sincrone pot fi modelate în Verilog folosind instrucţiunea *case* inclusă în corpul unei instrucţiuni *always*. Informaţia de stare va fi stocată într-un registru. În cadrul fiecărei ramuri a instrucţiunii *case* va fi descris comportamentul automatului pentru starea respectivă. Ca exemplu va fi descris un circuit de recunoaştere de pattern care semnalează pe ieşire dacă pe intrare s-au primit trei sau mai multe tacte de ceas succesive semnal în starea logică "1".

5.2.1 Modelarea unui automat de tip Mealy

Într-un automat Mealy ieşirile depind atât de starea curentă a automatului cât şi de intrări. Graful de tranziţie a stărilor pentru automatul de tip Mealy pentru problema considerată este prezentat în figura 5.1.

- Modelul Verilog asociat automatului Mealy:

```
//Descrierea unui automat Mealy cu întârziere
module mealy (out, ck, reset, in);

output out;
```

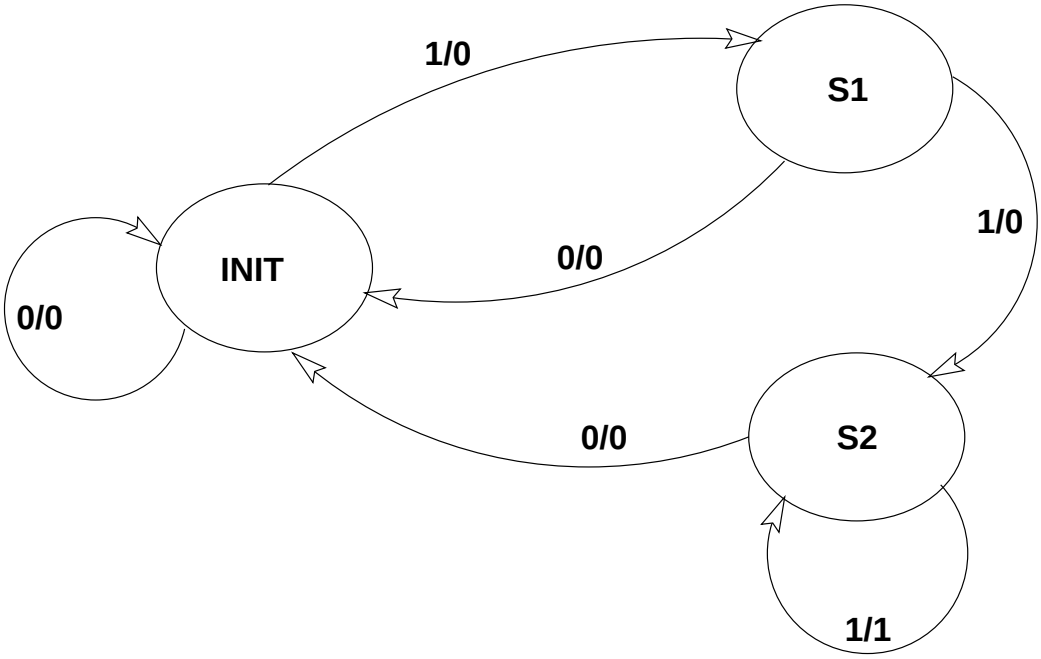

Figura 5.1: Graful de tranziţie al automatului de tip Mealy.

```
input ck;
input reset;
input in;

//definirea stărilor automatului
parameter INIT= 0;
parameter S0  = 1;
parameter S1  = 2;

//registru de stare
reg[1:0] stare;

//modelarea tranziţilor
always @(posedge ck or negedge reset)
  if (~reset)
    stare <= INIT;
  else
    case (stare)
      INIT:
       if (in)
          stare <= S0;
      S0:
        if (in)
          stare <= S1;
        else
          stare <= INIT;
      S1: stare <= in ? S1 : INIT;
    endcase
```

```
//modelarea ieşirii
reg out;
always @(posedge ck)
  out <= (stare == S1) & in;

endmodule
```

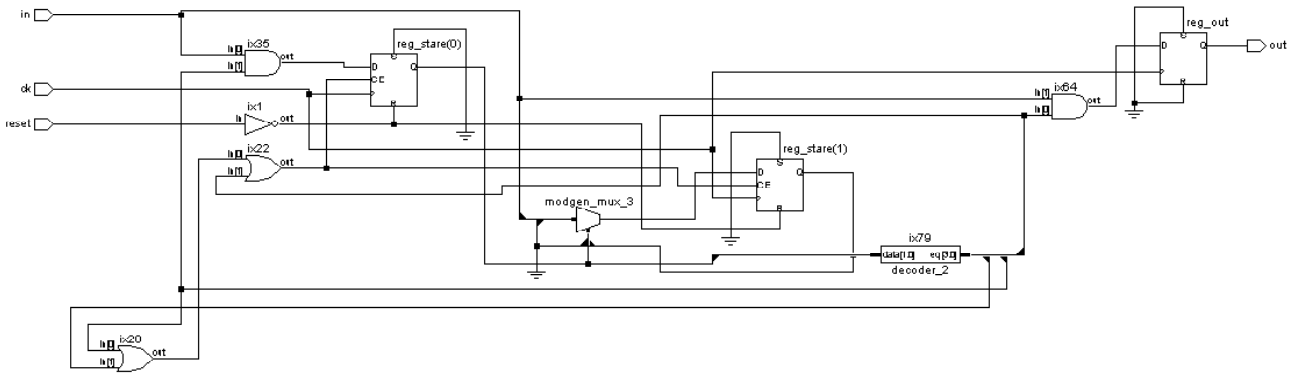

Figura 5.2: Automatul tip Mealy sintetizat cu *Leonardo*.

5.2.2 Modelarea unui automat de tip Moore

La modelarea unui automat de tip Moore în Verilog trebuie ţinut cont de faptul că ieşirile automatului depind doar de starea prezentă şi nu depind de intrări. Diagrama de tranziţie a stărilor pentru automatul de tip Moore este prezentată în figura 5.3. După cum se observă, pentru descrierea aceluiaşi automat ca fiind de tip Moore este necesar un număr mai mare de stări.

- Modelul Verilog asociat automatului Moore:

```
//Descrierea unui automat Moore imediat
module moore (ck, reset, in, out);

output out;
input ck;
input reset;
input in;

//definirea stărilor automatului
parameter INIT= 0;
parameter S0  = 1;
parameter S1  = 2;
parameter S2  = 3;

//registru de stare
reg[1:0] stare;
```

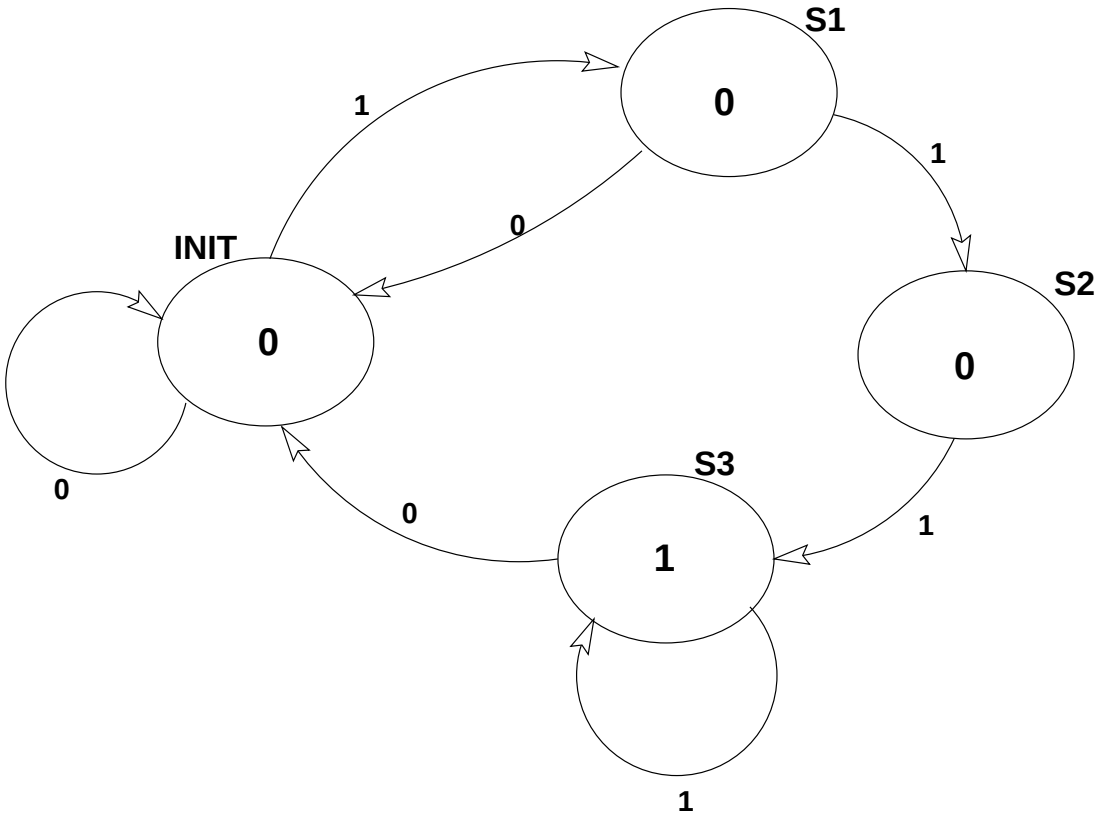

Figura 5.3: Graful de tranziţie al automatului de tip Moore.

```
//modelarea tranziţilor
always @(posedge ck or negedge reset)
  if (~reset)
    stare <= INIT;
  else
    case (stare)
      INIT:
       if (in)
          stare <= S0;
        else
          stare <= INIT;
      S0:
        if (in)
          stare <= S1;
        else
          stare <= INIT;
      S1:
        if (in)
          stare <= S2;
        else
          stare <= INIT;
      S2: stare <= in ? S2 : INIT;
    endcase
```

```
//modelarea ieşirii
wire out;
assign out = (stare == S2);

endmodule
```

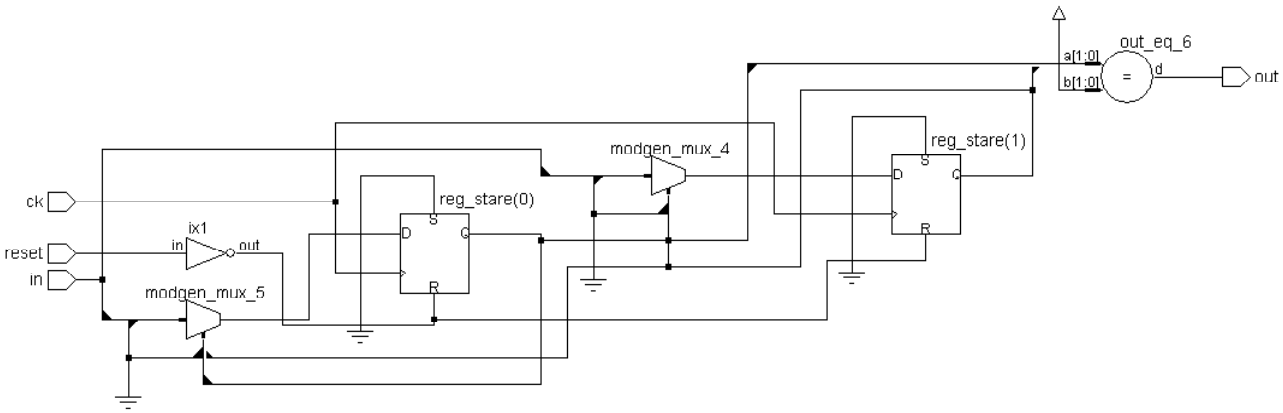

Figura 5.4: Automatul tip Moore sintetizat cu *Leonardo*.

Observaţie:

- În ambele exemple s-a folosit atribuirea neblocantă, cu operatorul de atribuire "<=", pentru semnalele. Se recomandă folosirea atribuirii neblocante a semnalelor în cadrul blocurilor de tip procedural *(always)*. Dacă nu este respectată această regulă pot apare diferenţe în simularea modelului pre şi post sinteză datorită modului diferit de interpretare de către simulatoare a ordinii de atribuire a semnalelor.

Exemplu: Diferenţa între cei doi operatori este ilustrată în următoarele exemple de atribuiri procedurale cu întârziere:

- Atribuirea procedurală blocantă:

```
initial
  begin
    rst = #5 0;
    rst = #4 1;
    rst = #10 0;
end
```

  Formele de undă rezultate la simulare sunt prezentate în figura 5.5.

- Atribuirea procedurală neblocantă:

```
initial
  begin
    rst <= #5 0;
    rst <= #4 1;
    rst <= #10 0;
end
```

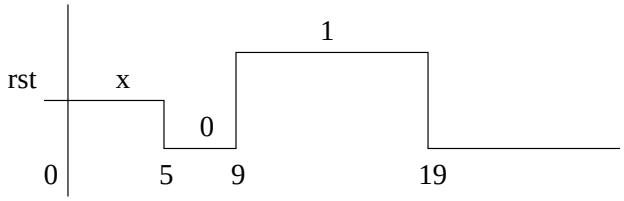

Figura 5.5: Atribuire blocantă cu întârziere în instrucţiune.

Formele de undă rezultate la simulare sunt prezentate în figura 5.6.

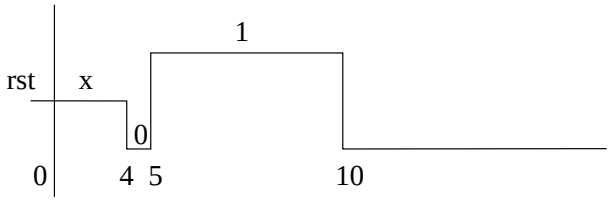

Figura 5.6: Atribuire neblocantă cu întârziere în instrucţiune.

5.3 Utilizarea automatelor

Un sistem digital complex poate fi structurat în două componente principale:

- *cale de date*
- *cale de control*

Automatele constituie *calea de control* în cadrul circuitelor digitale asigurând semnale de comandă şi control *căii de date*. În figura 5.7 este prezentată schema bloc a unui multiplicator secvenţial ce foloseşte metoda adunărilor repetate. Pentru simplificarea schemei nu au fost reprezentate semnalele de ceas şi reset prezente la intrarea tuturor registrelor.

- Modelul Verilog asociat multiplicatorului:

```
module mult (ready, rez, ck, reset, start, a, b);

output ready;
output [15:0] rez;
input ck;
input reset;
input start;
input [7:0] a, b;

//stăriile automatului
parameter INIT   = 0;
```

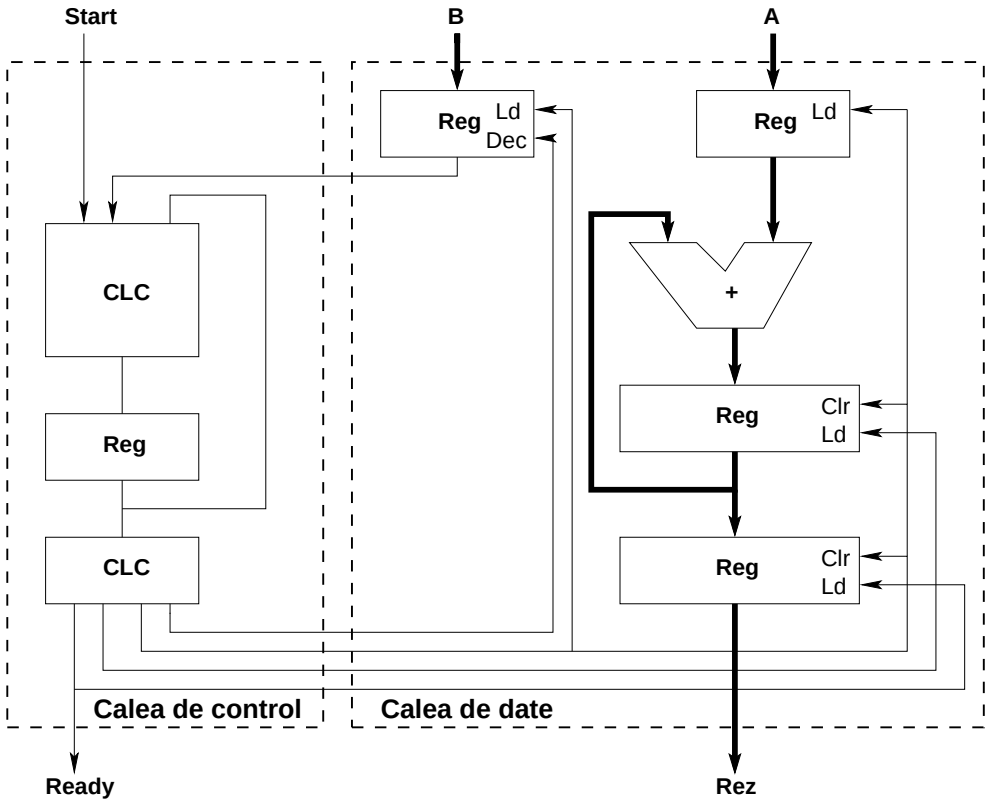

Figura 5.7: Schema bloc a multiplicatorului.

```
parameter SET_UP = 1;
parameter DEC    = 2;
parameter ADD    = 3;
parameter READY  = 4;

//descrierea căii de date
reg [7:0] a_d;
wire ld_op;
always @(posedge ck or negedge reset) if (~reset) a_d <= 0; else
                                      if (ld_op)  a_d <= a;
reg [7:0] b_d;
wire dec_b;
always @(posedge ck or negedge reset) if (~reset) b_d <= 0; else
                                      if (ld_op)  b_d <= b; else
                                      if (dec_b)  b_d <= b_d - 1;
wire bzero = (b_d == 0);
reg [15:0] acc;
wire ld_acc;
always @(posedge ck or negedge reset) if (~reset) acc <= 0; else
                                      if (ld_op)  acc <= 0; else
                                      if (ld_acc) acc <= acc + a_d;
reg [15:0] rez;
wire ld_rez;
always @(posedge ck or negedge reset) if (~reset) rez <= 0; else
                                      if (ld_op)  rez <= 0; else
```

```
                                        if (ld_rez) rez <= acc;
reg  ready;
always @(posedge ck or negedge reset) if (~reset) ready <= 0; else
                                        if (ld_rez) ready <= 1; else
                                                    ready <= 0;

//descrierea căii de control
//registru de stare
reg[2:0] state;

//modelarea tranziţilor
always @(posedge ck or negedge reset)
  if (~reset)  //reset asincron
    state <= INIT;
  else
    case (state)
      INIT:
       if (start)
          state <= SET_UP;
      SET_UP:
          state <= DEC;
      DEC:
        if (bzero)
          state <= READY;
        else
          state <= ADD;
      ADD:
        if (bzero)
          state <= READY;
        else
          state <= DEC;
      READY:
        if (start)
          state <= SET_UP;
      default:
          state <= INIT;
    endcase

//modelarea ieşirilor automatului
assign ld_rez   = (state == READY);
assign ld_op    = (state == SET_UP);
assign dec_b    = (state == DEC);
assign ld_acc   = (state == ADD);
endmodule
```

Schema la nivel RTL rezultată în urma sintezei cu *Leonardo* este prezentă în figura 5.8.

Observaţii

- Delimitarea în "cale de control" şi "cale de date" nu a fost făcută la nivel structural ci

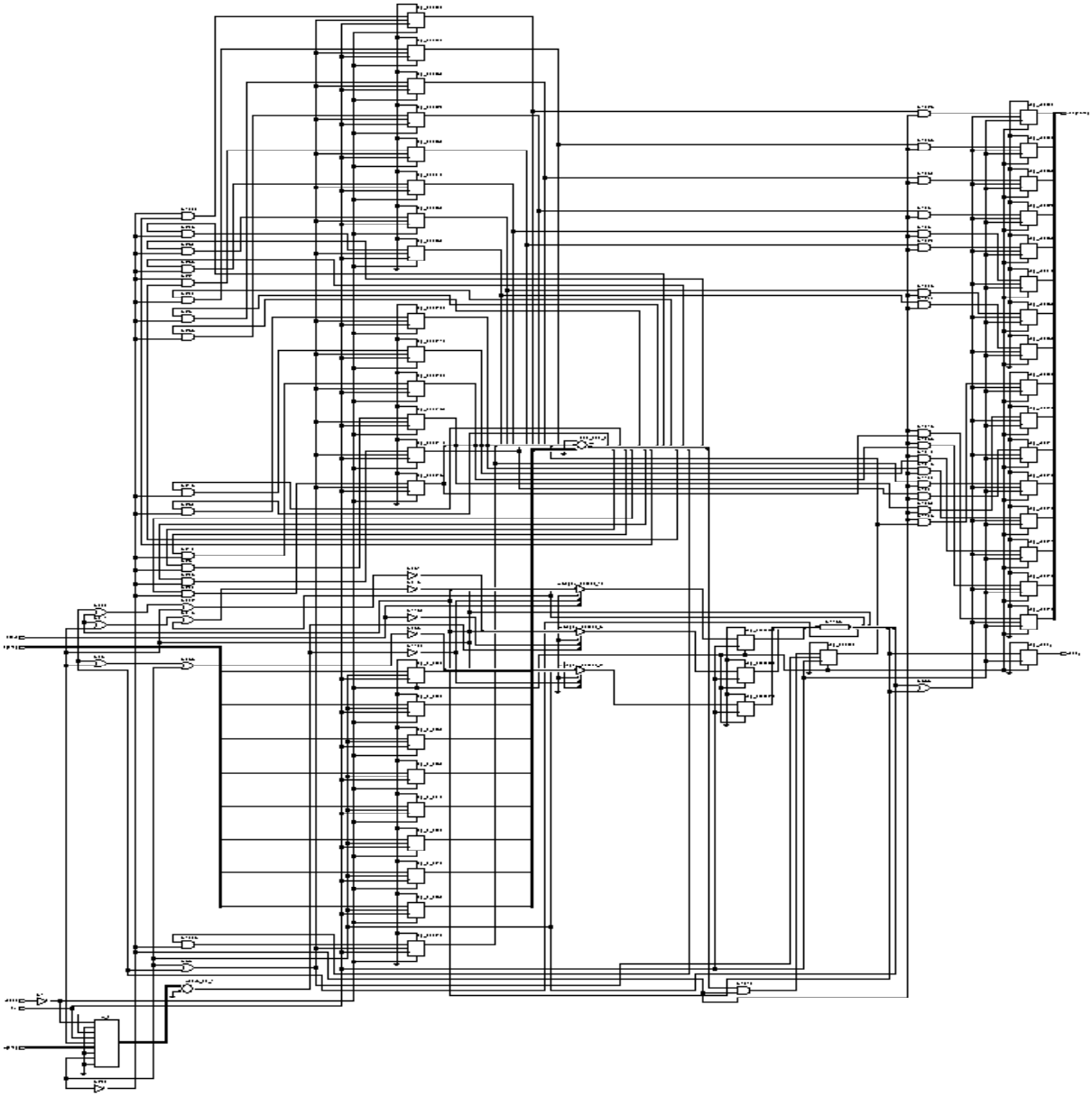

Figura 5.8: Multiplicatorul sintetizat cu *Leonardo*.

"dataflow". Din acest motiv, după sinteză nu există o graniţă netă între cele două blocuri. Totuşi, la o analiză atentă a schemei se poate pune în evidenţă apartenenţa fiecărui circuit la calea de date sau de control.

- Exemplul prezentat în continuare dovedeşte că nu totdeauna un sistem se descrie cel mai simplu sub forma unui automat, pornind de la graful de tranziţii sau organigrama acestuia.

5.4 Circuite formatoare de impulsuri

Automatul prezentat este un circuit care trebuie să genereze un număr par de impulsuri. Se cere proiectarea unui sistem secvenţial cu o intrare şi o ieşire. Intrarea este generată de un sistem sincron şi este activă în starea "1" un număr oarecare **Ni** de perioade de ceas. Ieşirea

este totdeauna activă un număr par **No** de perioade de ceas, astfel:

- dacă **Ni** este par atunci **No** = **Ni**
- dacă **Ni** este impar atunci **No** = **Ni** + **1**

Pentru exemplificare, figura 5.9 prezintă descrierea sistemului prin forme de undă:

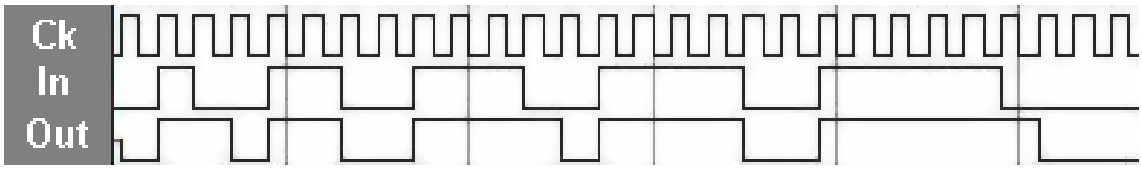

Figura 5.9: Formele de undă pentru generatorul de impulsuri.

Graful de tranziţii al unui automat Mealy imediat este prezentat în figura 5.10.

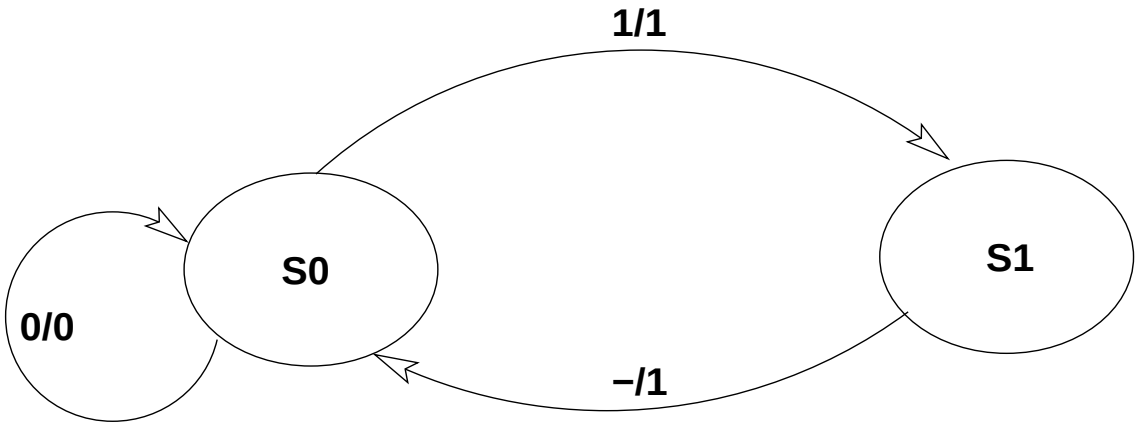

Figura 5.10: Graful de tranziţie al automatului.

Se observă că automatul are numai două stări între care comută. Din acest motiv se sugerează generarea unui semnal intermediar *(toggle)* care să comande comutarea stării automatului.

Semnalul *toggle* comută în starea complementară dacă *In = 1*, altfel *toggle* este în starea 0. Ieşirea *Out* este obţinută printr-o poartă SAU între *In* şi *toggle*. Figura 5.11 prezintă forma de undă corespunzatoare semnalului *toggle*.

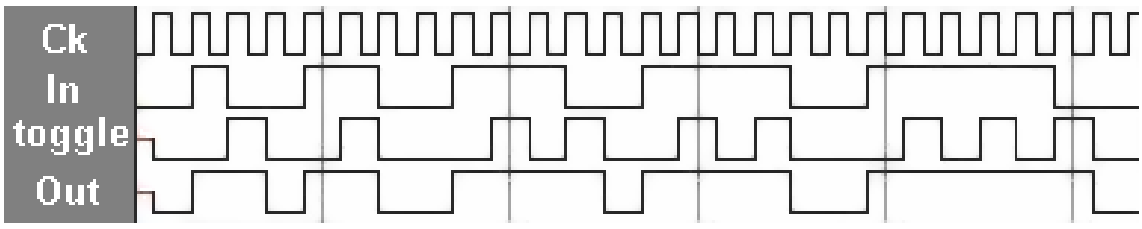

Figura 5.11: Formele de undă pentru generatorul de impulsuri şi bistabilul toggle

- Codul Verilog care descrie sistemul:

```
reg       toggle;
assign    Out = In | toggle;

always @(posedge Ck)
  toggle <= In & ~toggle;
```

Schema rezultată dupa sinteza cu *Leonardo* este cea din figura 5.12.

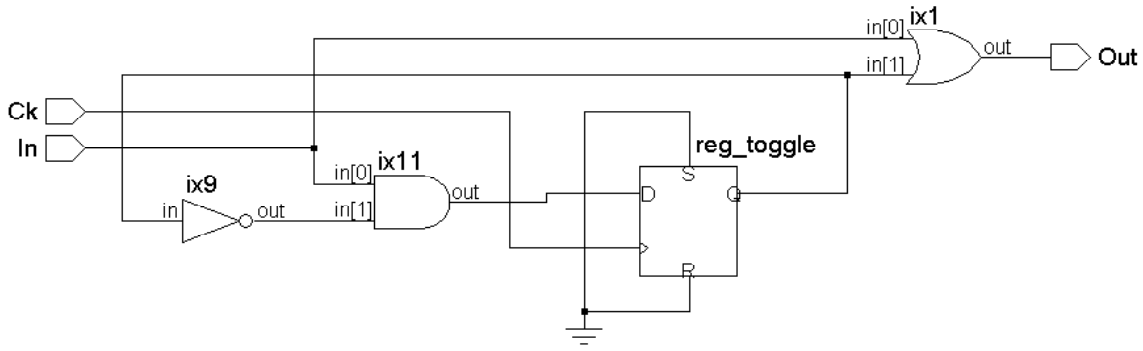

Figura 5.12: Generatorul de impulsuri sintetizat cu *Leonardo*.

- Dacă se doreşte ieşirea Out direct din registru, atunci codul trebuie modificat astfel:

```
reg       toggle;
reg       Out;

always @(posedge Ck)
  begin
    toggle <= In & ~toggle;
    Out <= In | toggle;
  end
```

În acest caz, schema rezultată după sinteza cu *Leonardo* este cea din figura 5.13. Ieşirea Out va fi întârziată cu un tact faţă de cea precedantă, dar hazardul combinaţional va fi eliminat.

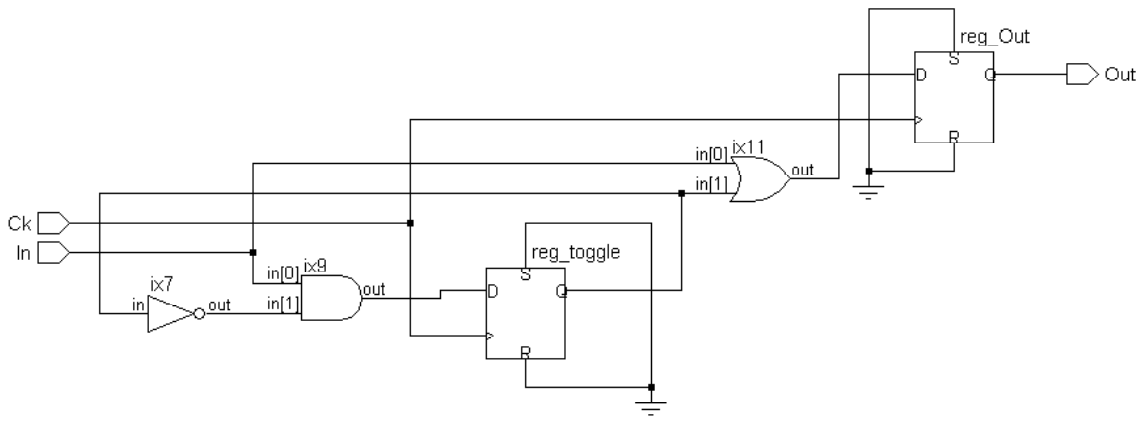

Figura 5.13: Generatorul de impulsuri cu ieşire din registru sintetizat cu *Leonardo*.

Observaţii:

- Deşi schemele rezultate (fig 5.12,5.13) prezintă structura unor automate, modelul Verilog nu este similar celui propus pentru descrierea automatelor în general.

- Pentru automate cu puţine stări, este uneori preferată scrierea modelului rezultat din transcrierea în HDL a ecuaţiilor de tranziţie a ieşirilor.

5.5 Simularea automatelor.

1. Verificaţi existenţa directorului *C:\Home* şi a dreptului de scriere în acesta. Copiaţi fişierele Verilog (fişiere cu extensia ".v") din directorul *\asd_hdl\exemple\l5* în directorul *C:\home*. Acest director va fi director de proiect în *ModelSim*. Lansaţi în execuţie *ModelSim*.

2. Simulaţi automatele Mealy şi Moore. Pentru simulare folosiţi un modul de testbench, precum şi un modulul de test aferent automatelor. În modulul de test instanţiaţi atât automatul Mealy cât şi automatul tip Moore. Schema modulului de test este prezentată în figura 5.14.
   Descrierile automatelor se găsesc în fişierele **mealy.v** şi **moore.v**. Descrierile generatorului de vectori de test şi a modulului de test se găsesc în fişierele **auto_tb.v** şi **auto_test.v**.

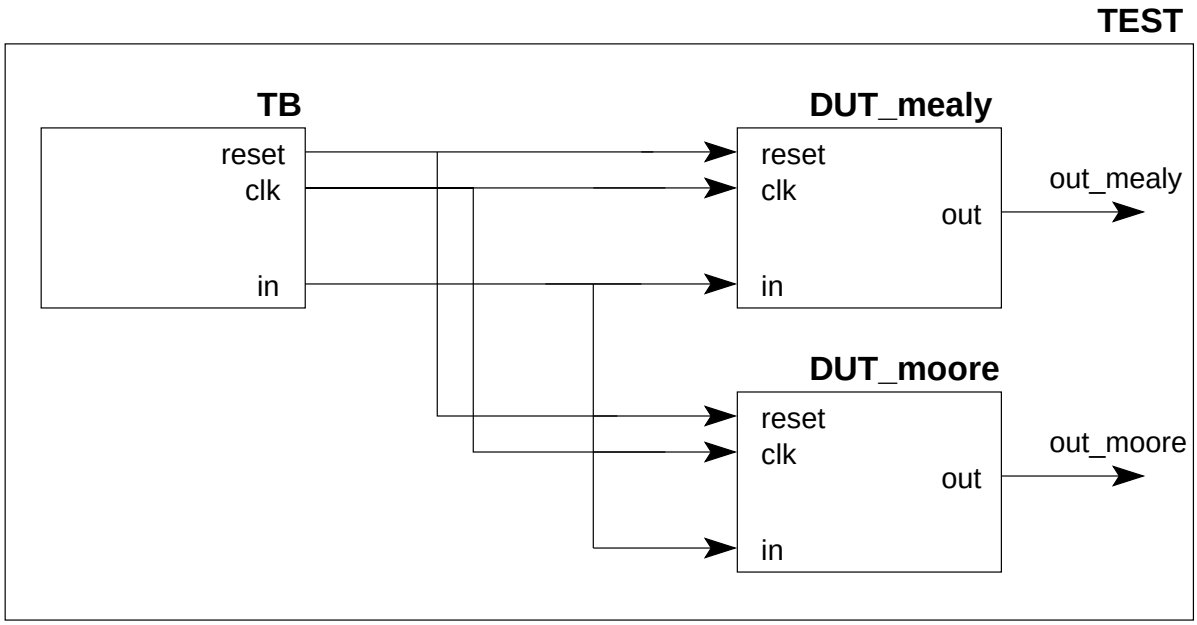

Figura 5.14: Structura modului *test* pentru automate.

3. Îmbunătăţiţi modelele de automate prezentate anterior pentru ca acestea să semnaleze pe ieşire dacă pe intrare s-au primit 4 sau mai multe tacte de ceas succesive semnal în starea logică "1".

4. Simulaţi circuitul multiplicator. Pentru simulare folosiţi modelul de multiplicator din fişierul **mult.v**, modelul de generator de vectori de test din fişierul **mult_tb.v** şi modulul de test din fişierul **mult_test.v**.

5. Modificaţi automatul de control al multiplicatorului din automat de tip Moore în automat de tip Mealy.

5.6 Exerciţii

1. Proiectaţi un circuit care numără apariţiile unui anumit pattern la intrare.

```
module(seg1, seg2, ov, reset, clk, data);
input reset;              //semnal de reset sincron activ High
input clk;                //semnal de ceas activ pe frontul crescător
input data;               //un bit de date de intrare
output [6:0] seg1, seg2; //două ieşiri de c\^ate 7 biţi corespunzătoare a
                  //două afişoare 7 segmente pentru două cifre zecimale
output ov;                //semnalizarea depăşirii 99-00
reg [6:0] seg1, seg2;

//Inseraţi modelul Verilog aici

endmodule
```

Descrierea sistemului este:

- Recunoaşte patternul 11100.
- Contorizează numărul de apariţii ale patternului în timp.
- La apariţia unui pattern, se actualizează numărul afişat, aşa ca în figura 5.15.

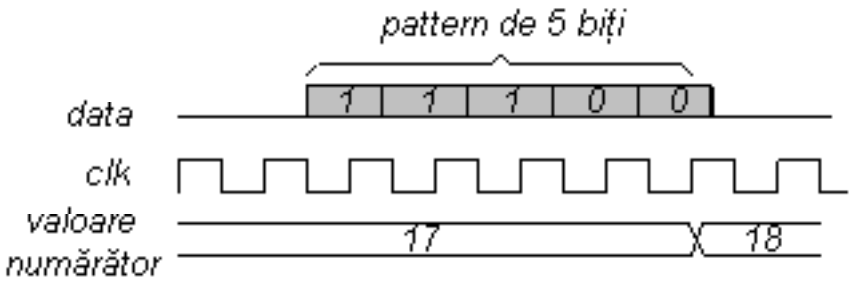

Figura 5.15: Actualizarea numărului de apariţii ale patternului.

- Pentru afişarea numărului se utilizează două afişoare cu câte şapte segmente.
- La depăşirea numărului de 99 de apariţii ale patternului se activează semnalul *ov* (overflow - depăşire).
- După reset, numărătorul reîncepe de la zero. Dacă a fost recunoscută doar o parte de pattern, ea este ignorată.
- Înainte de primul reset circuitul are un comportament imprevizibil.

Indicaţie: Ţineţi cont că la ieşirea unui bistabil D este valoarea aflată la intrarea acestuia cu un tact în urmă. Prin întârzierea datei de intrare printr-o succesiune de bistabile D se poate avea acces la un moment dat la valorile intrării la momente de timp anterioare.

2. Modelaţi în Verilog un automat numărător în cod Gray pe 3 biţi.

3. Modelaţi un sumator serial pe 8 biţi.

4. Puneţi în evidenţa prin simulare diferenţele de comportament din următoarele modele şi explicaţi motivele diferenţelor:

```
always(a or b) begin
    b <= a;
    c <= b;
end
```

```
always(a or b) begin
    b = a;
    c = b;
end
```

5. Scrieţi codul HDL care modelează un filtru digital cu următoarea caracteristică de transfer:

```
y0 <= c(0)*x(0)+c(1)*x(1)+c(2)*x(2) ;
```

Folosiţi c(0) =-4, c(1) =+5, c(0) = -3, dar scrieţi codul în aşa fel încât coeficienţii să poată fi uşor modificaţi. Simulaţi, testaţi şi sintetizaţi modelul.

6. Completaţi următoarele fragmente de cod şi sintetizaţi-le:

```
always @(posedge ck) begin
    #0  phase <= 0;
    #10 phase <= 1;
end

always @(posedge ck) begin
    case (phase)
        0:        phase <= 1;
        default:  phase <= 0;
    endcase
end
```

Care sunt mesajele de eroare? Explicaţi rezultatele obţinute în urma sintezei.

Lucrarea 6

Modelarea şi testarea registrelor

Această lucrare prezintă modul de realizare a diferitelor modele de registre, precum şi modalităţi de testare a acestora.

6.1 Scopul lucrării

- Construcţia unui model de registru paralel.
- Construcţia unui model de registru serie.
- Construcţia unui model de LFSR (Linear feed-back shift register).
- Construcţia unui model de registru care implementează algoritmul CRC32.
- Realizarea unui mediu de simulare cu stimuli aleatori şi detector de erori.

6.2 Registrul paralel

Schema bloc a unui registru de deplasare, cu încărcare şi ieşire paralele, este prezentată în figura 6.1.

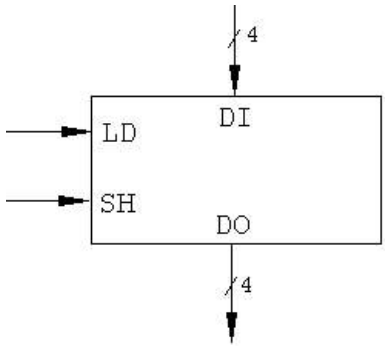

Figura 6.1: Schema bloc a unui registru de deplasare.

Registrul funcţionează astfel:

- Dacă semnalul de încărcare (Load - LD) este activ, atunci datele de intrare (DI) sunt stocate în registru.
- Dacă semnalul de deplasare (Shift - SH) este activ, atunci cuvântul memorat de registru este deplasat spre stânga cu un bit.

Tabelul 6.1 prezintă acţiunile registrului: Descrierea Verilog a unui astfel de registru este prezen-

LD	*SH*	*CK*	*Q*	*Semnificaţie*
1	X	↑	Di	Încărcare
0	1	↑	Q << 1	Deplasare stânga
0	0	X	Q	Păstrează starea

Tabelul 6.1: Tabelul de adevăr al regustrului de deplasare.

tată în continuare:

```
module paralel_reg (CLK, LD, SH, DI, DO);
input           CK;
input           LD;
input           SH;
input   [3:0]   DI;
output  [3:0]   DO;
reg     [3:0]   DO;

always @(posedge CK)
  if (LD)
    DO <= DI;
  else
```