

## Hazardul combinațional și simularea acestuia în ModelSim

Circuitele logice combinaționale (CLC) sunt circuite logice fără memorie, adică independente de propriile lor stări de ieșire anterioare. Ele sunt caracterizate de faptul că semnalele de ieșire sunt rezultatul doar al operațiilor logice între semnalele lor de intrare.

Independența față de timp a relației care se stabilește între ieșirea unui circuit logic combinațional și intrările sale poate fi dedusă din răspunsul, independent față de timp, produs la ieșirea circuitului logic ca urmare a modificării stărilor logice de pe intrările acestuia.

Un CLC poate fi constituit dintr-un ansamblu de porți logice elementare interconectate între ele în diverse moduri. Datele de intrare, în timpul procesării lor, pot parcurge în drumul lor către ieșire mai multe căi, ce pot avea un număr diferit de porți. Din acest motiv, efectul modificării valorii logice a intrărilor se propagă către ieșire pe durate de timp diferite. O astfel de durată este egală cu timpul  $t$  de propagare printr-o poartă (pentru simplificare, aproximat egal pentru toate porțile unei căi) multiplicat cu numărul  $m$  de porți parcurse de semnal pe calea respectivă.

Presupunând că cea mai scurtă cale intrare-ieșire parcurge  $p$  porți, iar cea mai lungă  $P$  porți, înseamnă că semnalul la ieșire va începe să se modifice după un interval de timp  $p \cdot t$  dela momentul modificării semnalului pe intrare și se va stabili în intervalul de timp  $P \cdot t$ .

În intervalul  $(P - p) \cdot t$ , semnalul de ieșire va înregistra valori neconforme cu rezultatul așteptat. Acest efect a fost denumit **hazard combinațional** sau **hazard logic**.

Eliminarea hazardului logic poate fi posibilă fie printr-o proiectare riguroasă care să asigure întârzieri egale pe toate căile dintre intrare și ieșire, fie prin citirea informațiilor la ieșirea circuitului abia după încheierea duratei  $(P - p) \cdot t$  de desfășurare a efectului de hazard.

Pe durata unui efect de hazard combinațional, forma undelor semnalelor afișate la simularea cu **ModelSim** se aseamănă cu niște țepi (spikes). Mai jos, în figura 1, este prezentat un exemplu de hazard oferit de circuitul realizat cu o poartă de tipul AND cu două intrări și un inversor:

### c) Schema logică

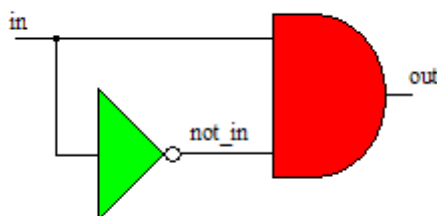


Fig. 1 Circuit combinațional capabil de hazard

### c) Codul Verilog

Pentru a simula funcționarea circuitului de mai sus, se creează fișierele text **hazard.v**, **test\_bench.v** și **test.v** ce conțin câte un modul cu același nume.

➤ fișierul **hazard.v**, conține codul Verilog ce modelează circuitul logic din figura 1.

```

1  module hazard (in, out);
2      input in;
3      output out;
4      reg out, not_in;
5
6      always @(in or not_in) //lista senzitiva supravegheaza semnalele in si not_in
7      begin
8          out <= in & not_in; //out este evaluat la fiecare nou in si la fiecare nou not_in.
9          not_in <= ~ in; //aici not_in este ulterior lui not_in utilizat in atribuirea lui out.
10     end
11 endmodule

```

**Observație:** la schimbarea de stare a intrării *in*, ieșirea *not\_in* a inversorului cunoaște două stări: una prezentă (în momentul modificării lui *in*) și alta zisă următoare (după propagarea prin inversor a modificării lui *in*).

➤ fișierul *test\_bench.v*, destinat generării stimulului de testare, botezat *in*

```

1  module test_bench (in);
2      output in;
3      reg in;
4      initial
5          begin
6              in = 1'b0; // la momentul de start, semnalul <in> este asignat cu 0 binar
7              repeat (4)
8                  in = #5 ~ in; // schimbarea starii lui <in> se repeta de 4 de ori, din 5 in 5 u.t.
9          end
10     endmodule

```

➤ fișierul *test.v*, care realizează interconexiunile între instanțele celor două module:

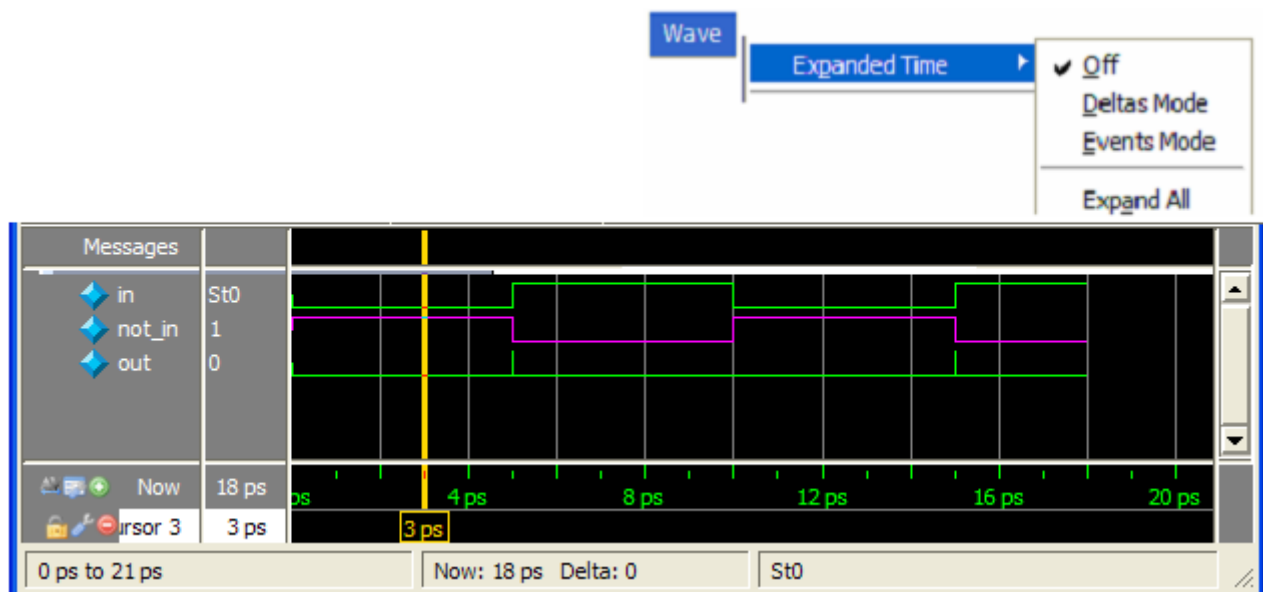
```

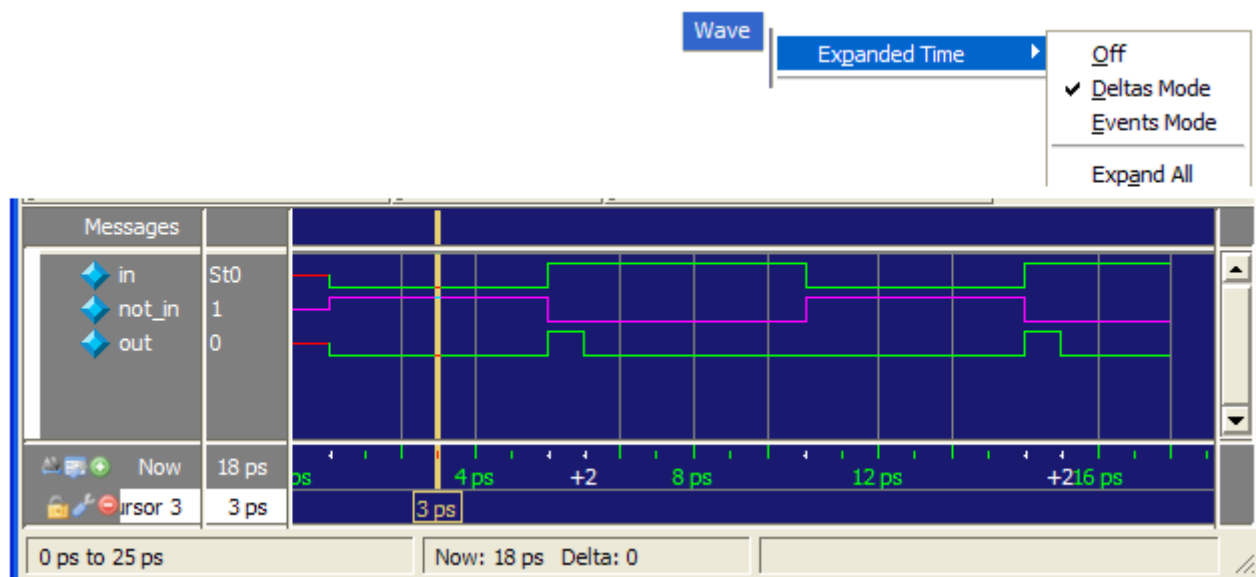
1  module test();
2      wire in, out;
3      test_bench TB (.in(in));
4      hazard DUT (.in(in), .out(out));
5  endmodule

```

Conform principiului contradicției din logica booleană, ieșirea **out** a porții AND ar trebui să fie mereu 0, întrucât întotdeauna expresia  $in \cdot \overline{in} = 0$ . Datorită însă efectului de hazard, în momentul în care *in* trece pe 1, *not\_in* nu devine imediat 0 ci păstrează încă valoarea 1 anterioară și deci ieșirea **out** nu este 0 ci este egală cu  $1 \cdot 1 = 1$  pentru o perioadă scurtă de timp (perioada de *glitch*).

### c) Diagrama obținută la simulare în ModelSim





Cele două diagrame wave de mai sus corespund variantei de modul **hazard.v** lipsit de o întârziere introdusă în linia de cod care execută atribuirea semnalului **not\_in**.

În prima diagramă, se observă **spikes**-urile din momentele producerii **glitch**-urilor de hazard. În cea de-a doua diagramă, în care opțiunea **Expand Time > Deltas Mode** din meniul **Wave** este activată (opțiune existentă în ModelSim vers. 6.4a), se văd evidențiate glitch-urile (în locurile de apariție a spikes-urilor) datorită expandării artificiale de către program a timpilor de afișare.

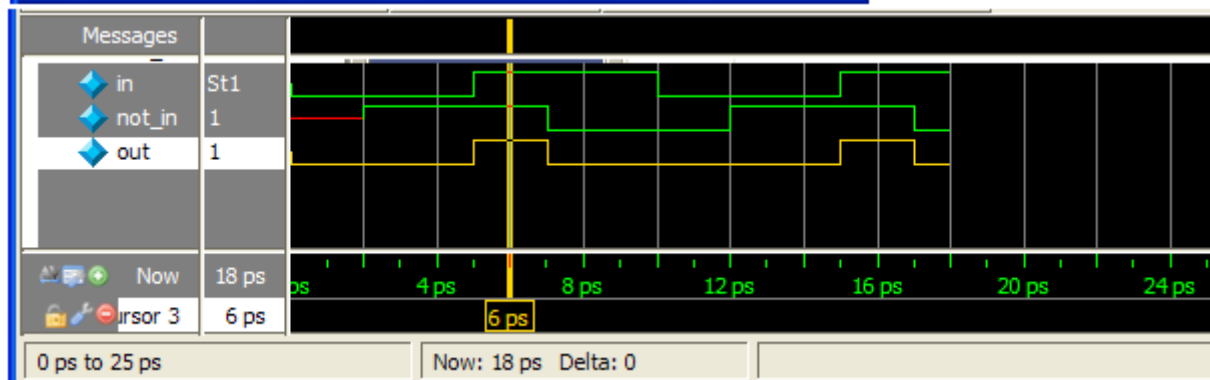
Diagrama wave următoare corespunde variantei de modul **hazard.v** în care s-a prevăzut un **delay #1** în linia de cod respectivă, egal cu valoarea apreciată pentru timpul de propagare prin inversor. Durata glitch-urilor este egală cu timpul de propagare al semnalului prin inversor.

➤ fișierul **hazard.v**, variantă de modul ce introduce delay-ul 1# în evaluarea lui not\_in.

```

1  module hazard (in, out);
2      input in;
3      output out;
4      reg out, not_in;
5
6      always @(in or not_in)
7          begin
8              out <= in & not_in;
9              not_in <= #2 ~in;
10         end
11     endmodule

```



ps	delta	in	not_in	out
0	+0	StX	x	x
0	+1	St0	x	0
2	+0	St0	1	0
5	+1	St1	1	1
7	+0	St1	0	0
10	+1	St0	0	0
12	+0	St0	1	0
15	+1	St1	1	1
17	+0	St1	0	0

## Modelarea registrelor și simularea în ModelSim

### 1. Registrul serie

Registrul serie, prezentat în figura 2, este format din 4 bistabili de tip D, cu deplasare la dreapta.

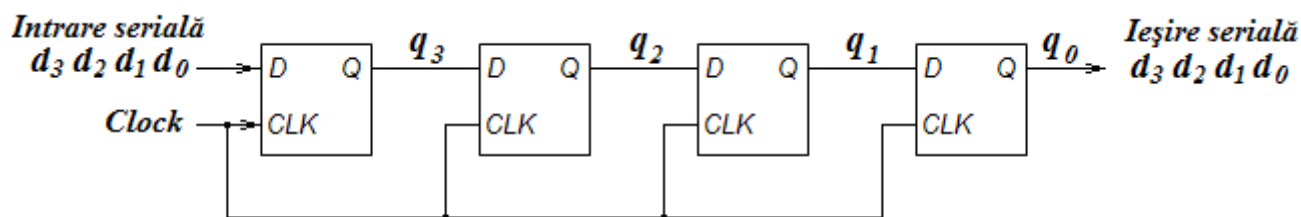


Figura 2. Schema bloc a unui registru serie cu deplasare stânga-dreapta

În timpul funcționării, bistabilele tip D ale registrului se încarcă fiecare, pe frontul crescător (pozitiv) al tactului de ceas, cu informația prezentă pe intrarea lor D. La primul tact, bitul  $d_0$  prezent la intrarea D a primului bistabil din stânga este încărcat la ieșirea Q a acestuia. La următorul tact, bitul  $d_0$  prezent acum pe firul  $q_3$  se încarcă în bistabilul următor fiind deci prezent pe firul  $q_2$  în vreme ce pe firul  $q_3$  apare bitul  $d_1$  ș.a.m.d. Întregul cuvânt este încărcat în registru la al 4-lea front de tact. La următoarele tacte, biții cuvântului se deplasează în continuare, pas cu pas (bistabil cu bistabil) spre dreapta, astfel că la al 8-lea tact registru este golit de cuvânt, aflat acum cu toate bistabilele trecute pe 0. Deplasarea informației în registru a dat acestor registre serie și denumirea de registre cu deplasare (*shift registers*). Deplasarea poate fi posibilă (funcție de construcția registrului) fie doar într-o direcție, fie în ambele direcții.

Registrele de deplasare serie sunt utilizate drept memorii cu acces serial (*SAR - Serial Acces Register*). Ele sunt construite pentru un număr mare de biți; cu toate acestea, numărul mare de celule de memorie nu prezintă nici un fel de implicații asupra numărului de conexiuni externe ale circuitului integrat.

Ca exemplu, se prezintă modelarea unui **registru serie de 4 biți, cu deplasare la dreapta** (v. schemele bloc din fig. 3 și 4).

Registrul este alcătuit din 4 bistabile tip D legate serial între ele prin firele notate cu  $q_3, q_2, q_1$ . Firul de intrare **shiftin** în registru e legat la pinul **d** al primului bistabil ( $G_3$ ) iar firul **shiftout** de ieșire din registru este legat la firul  $q_0$  de ieșire al ultimului bistabil ( $G_0$ ). Cuvântul ales pentru încărcare în registru a fost denumit **wordin** = 1011 iar cuvântul descărcat din registru este denumit **wordout**.

a) Schema bloc (v. fig. 3)

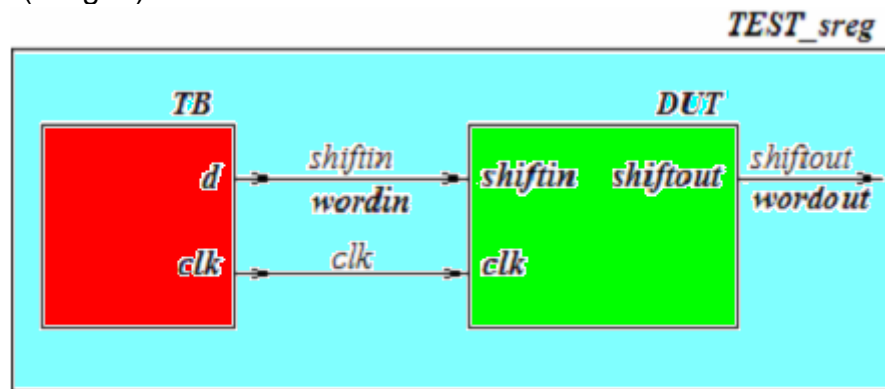


Figura 3 - Schema conexiunilor dintre blocul TB (Test Bench- generatorul de stimuli) și blocul DUT (Device Under Test) al registrului serie. Conexiunile interioare ale blocului DUT sunt structurate în corpul modului test\_sreg.

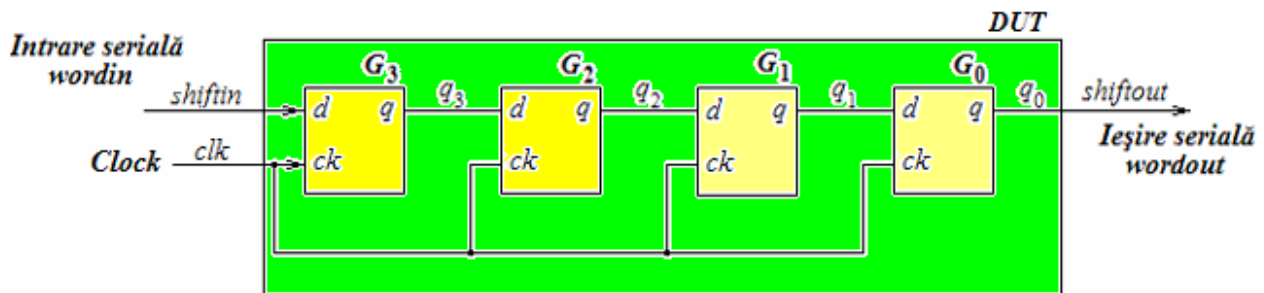


Figura 4. Schema blocului denumit DUT al registrului serie de 4 biți, cu deplasare la dreapta, cu cele 4 bistabile tip D notate  $G_3$ ,  $G_2$ ,  $G_1$ ,  $G_0$ ,

b) Codul Verilog

➤ Modelarea bistabilului tip D (ce face parte din componența registrului), fisierul *dff.v*

```
module dff (q, d, ck); // device flip-flop, de tip d
    input d, ck; // porturi de intrare, de 1 bit
    output q; // port de iesire, de 1 bit
    reg q; // iesirea q declarata de tip registru
    always @(posedge ck)
        q = d; // atribuirea se efectueaza pe frontul poz. de tact
endmodule
```

➤ Modelarea registrului serial compus din 4 bistabile tip D, fisierul *sreg4b.v*

```
module sreg4b (shiftout, shiftin, clk);
    input shiftin, clk;
    output shiftout;

    wire q3, q2, q1, q0;
    reg[3:0] wordout;
    reg[3:0] counter;

    dff G3 (.d(shiftin), .ck(clk), .q(q3));
    dff G2 (.d(q3), .ck(clk), .q(q2));
    dff G1 (.d(q2), .ck(clk), .q(q1));
    dff G0 (.d(q1), .ck(clk), .q(shiftout));
```

```
assign q0 = shiftout; //pentru a face posibila afisarea q0
initial counter = 0;
always @(posedge clk)
begin
    if (counter == 4'b1000)
        $stop; //se opreste simularea la al 8-lea tact clock
    else
    begin
        if (counter == 4'b0100) wordout[0] = shiftout;
        if (counter == 4'b0101) wordout[1] = shiftout;
        if (counter == 4'b0110) wordout[2] = shiftout;
        if (counter == 4'b0111) wordout[3] = shiftout;
        counter = counter + 1; //la fiecare front pozitiv clock
    end
end
endmodule
```

➤ Modelarea test bench-ului generator de stimuli, fișierul *test\_bench\_sreg.v* :

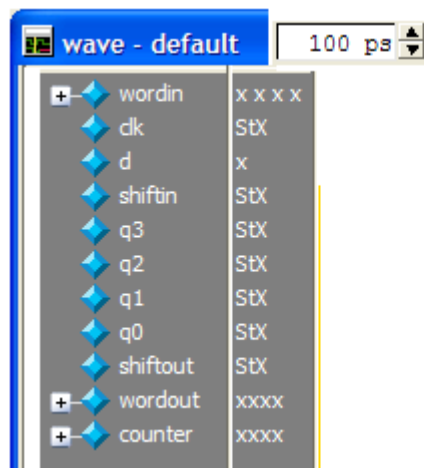
```
module test_bench_sreg (d, clk); //d,clk sunt stimulii generati
output d, clk;
reg d, clk;
reg wordin[3:0]; //wordin este cuvantul ce se incarca in registru

initial //se seteaza compozitia cuvantului wordin
begin
    wordin[3]=1;//assignere bit wordin[3] destinat incarcarii in registru
    wordin[2]=0;//assignere bit wordin[2] destinat incarcarii in registru
    wordin[1]=1;//assignere bit wordin[1] destinat incarcarii in registru
    wordin[0]=1;//assignere bit wordin[0] destinat incarcarii in registru
    clk = 1'b0; // se pozitioneaza pe zero semnalul clock
end

always #5 clk <= ~clk; //clock-ul este comutat tot la 5 u.t.
initial //sunt expediate succesiv bitii cuvantului spre registru
begin
    d <= #4 wordin[0]; // se aplica wordin[0]=1 la intrarea registrului
    d <= #14 wordin[1]; // se aplica wordin[1]=1 la intrarea registrului
    d <= #24 wordin[2]; // se aplica wordin[2]=0 la intrarea registrului
    d <= #34 wordin[3]; // se aplica wordin[3]=1 la intrarea registrului
    d <= #44 1'bx; /* se aplica X la intrarea registrului,in vederea incarcarii lui x in bistabile la
        descarcarea acestora */
end
endmodule
```

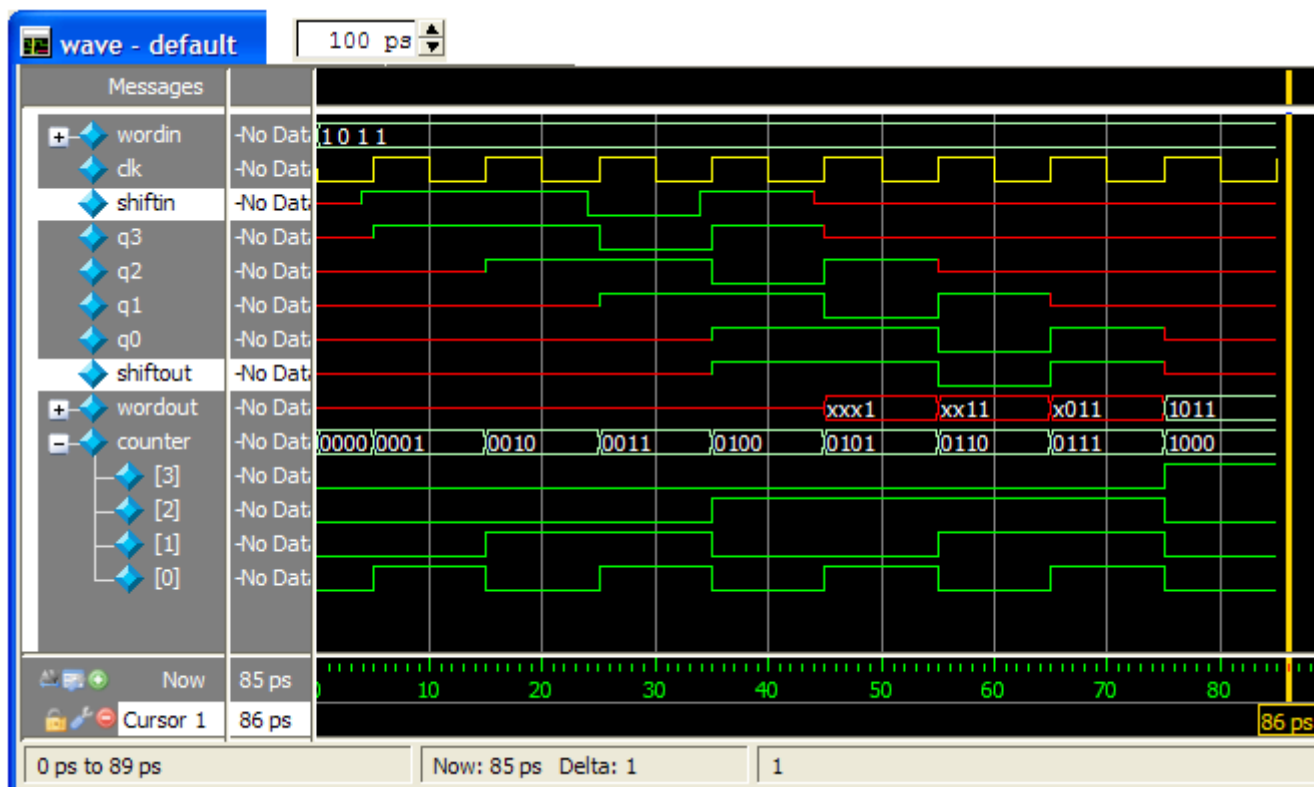
➤ Modulul *test\_sreg* de interfațare conexiuni instanțe, fișierul *test\_sreg.v*:

```
module test_sreg (); //modulul parinte
    wire shiftout, shiftin, clk; //cele 3 fire de legare cu exteriorul
    test_bench_sreg TB (.d(shiftin), .clk(clk));
    sreg4b DUT (.shiftin(shiftin), .clk(clk), .shiftout(shiftout));
endmodule
```

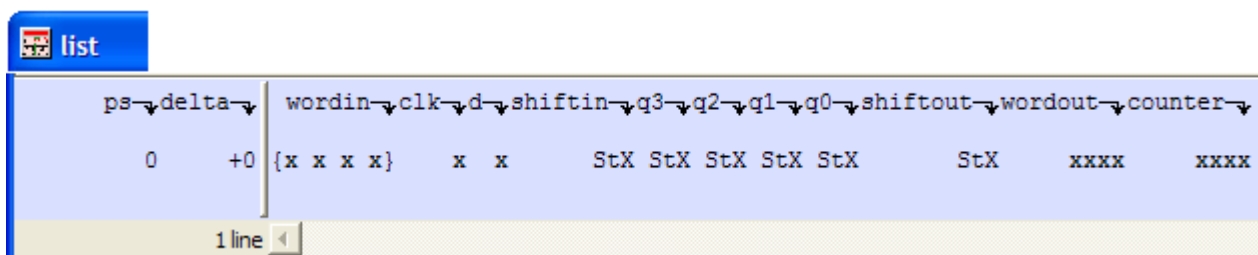


La simularea cu ModelSim, se recomandă ordinea de mai sus a semnalelor în panoul **wave**. În panoul prezentat se pot vedea, totodată, valorile semnalelor înainte de startul simulării.

### c) Diagrama obținută la simularea în ModelSim a registrului serie



În operația de citire a cuvântului la ieșirea registrului s-a folosit un contor (**counter**) pentru stabilirea momentelor de evaluare a biților lui **wordout** precum și pentru oprirea procesului de simulare în momentul terminării operației de descărcare a registrului de cuvântul **wordin** (când semnalele *q* au toate din nou valoarea X).





Mai sus, este prezentat panoul **list** cu valorile semnalelor dinaintea startului simulării.

Mai jos, este prezentată evoluția valorilor semnalelor pe parcursul simulării.

list											
File Edit View Add Tools Window											
ps delta wordin clk d shiftin q3 q2 q1 q0 shiftout wordout counter											
0	+0	{1 0 1 1}	0	x	StX	StX	StX	StX	StX	xxxx	0000
4	+0	{1 0 1 1}	0	1	StX	StX	StX	StX	StX	xxxx	0000
4	+1	{1 0 1 1}	0	1	St1	StX	StX	StX	StX	xxxx	0000
5	+0	{1 0 1 1}	1	1	St1	StX	StX	StX	StX	xxxx	0000
5	+1	{1 0 1 1}	1	1	St1	StX	StX	StX	StX	xxxx	0001
5	+2	{1 0 1 1}	1	1	St1	St1	StX	StX	StX	xxxx	0001
10	+0	{1 0 1 1}	0	1	St1	St1	StX	StX	StX	xxxx	0001
15	+0	{1 0 1 1}	1	1	St1	St1	StX	StX	StX	xxxx	0001
15	+1	{1 0 1 1}	1	1	St1	St1	StX	StX	StX	xxxx	0010
15	+2	{1 0 1 1}	1	1	St1	St1	St1	StX	StX	xxxx	0010
20	+0	{1 0 1 1}	0	1	St1	St1	St1	StX	StX	xxxx	0010
24	+0	{1 0 1 1}	0	0	St1	St1	St1	StX	StX	xxxx	0010
24	+1	{1 0 1 1}	0	0	St0	St1	St1	StX	StX	xxxx	0010
25	+0	{1 0 1 1}	1	0	St0	St1	St1	StX	StX	xxxx	0010
25	+1	{1 0 1 1}	1	0	St0	St1	St1	StX	StX	xxxx	0011
25	+2	{1 0 1 1}	1	0	St0	St0	St1	St1	StX	xxxx	0011
30	+0	{1 0 1 1}	0	0	St0	St0	St1	St1	StX	xxxx	0011
34	+0	{1 0 1 1}	0	1	St0	St0	St1	St1	StX	xxxx	0011
34	+1	{1 0 1 1}	0	1	St1	St0	St1	St1	StX	xxxx	0011
35	+0	{1 0 1 1}	1	1	St1	St0	St1	St1	StX	xxxx	0011
35	+1	{1 0 1 1}	1	1	St1	St0	St1	St1	StX	xxxx	0100
35	+2	{1 0 1 1}	1	1	St1	St1	St0	St1	StX	xxxx	0100
35	+3	{1 0 1 1}	1	1	St1	St1	St0	St1	St1	xxxx	0100
40	+0	{1 0 1 1}	0	1	St1	St1	St0	St1	St1	xxxx	0100
44	+0	{1 0 1 1}	0	x	St1	St1	St0	St1	St1	xxxx	0100
44	+1	{1 0 1 1}	0	x	StX	St1	St0	St1	St1	xxxx	0100
45	+0	{1 0 1 1}	1	x	StX	St1	St0	St1	St1	xxxx	0100
45	+1	{1 0 1 1}	1	x	StX	St1	St0	St1	St1	xxx1	0101
45	+2	{1 0 1 1}	1	x	StX	StX	St1	St0	St1	xxx1	0101
50	+0	{1 0 1 1}	0	x	StX	StX	St1	St0	St1	xxx1	0101
55	+0	{1 0 1 1}	1	x	StX	StX	St1	St0	St1	xxx1	0101
55	+1	{1 0 1 1}	1	x	StX	StX	St1	St0	St1	xx11	0110
55	+2	{1 0 1 1}	1	x	StX	StX	StX	St1	St1	xx11	0110
55	+3	{1 0 1 1}	1	x	StX	StX	StX	St1	St0	xx11	0110
60	+0	{1 0 1 1}	0	x	StX	StX	StX	St1	St0	xx11	0110
65	+0	{1 0 1 1}	1	x	StX	StX	StX	St1	St0	xx11	0110
65	+1	{1 0 1 1}	1	x	StX	StX	StX	St1	St0	x011	0111
65	+2	{1 0 1 1}	1	x	StX	StX	StX	StX	St0	x011	0111
65	+3	{1 0 1 1}	1	x	StX	StX	StX	StX	St1	x011	0111
70	+0	{1 0 1 1}	0	x	StX	StX	StX	StX	St1	x011	0111
75	+0	{1 0 1 1}	1	x	StX	StX	StX	StX	St1	x011	0111
75	+1	{1 0 1 1}	1	x	StX	StX	StX	StX	St1	1011	1000
75	+2	{1 0 1 1}	1	x	StX	StX	StX	StX	St1	1011	1000
75	+3	{1 0 1 1}	1	x	StX	StX	StX	StX	StX	1011	1000
80	+0	{1 0 1 1}	0	x	StX	StX	StX	StX	StX	1011	1000
85	+0	{1 0 1 1}	1	x	StX	StX	StX	StX	StX	1011	1000



## 2. Registrul paralel

Registrul paralel este format din 4 bistabile de tip D care se încarcă în derivație (concomitent) la semnalul de tact al unui ceas/clock aplicat sincron (v. fig. 6). Cuvântul destinat încărcării trebuie aplicat în prealabil la intrările bistabilelor.

În momentul aplicării tactului, cuvântul binar **wordin** prezent cu cei 4 biți ai săi pe firele de intrare  $d_3, d_2, d_1, d_0$  ale registrului este înscris (încărcat) în cele 4 bistabile ce îndeplinesc rolul de celule de memorie. Biții cuvântului astfel memorat pot fi citiți pe firele de ieșire  $q_3, q_2, q_1, q_0$  ale bistabilelor, respectiv la ieșirile  $Q_3 Q_2 Q_1 Q_0$  ale registrului.

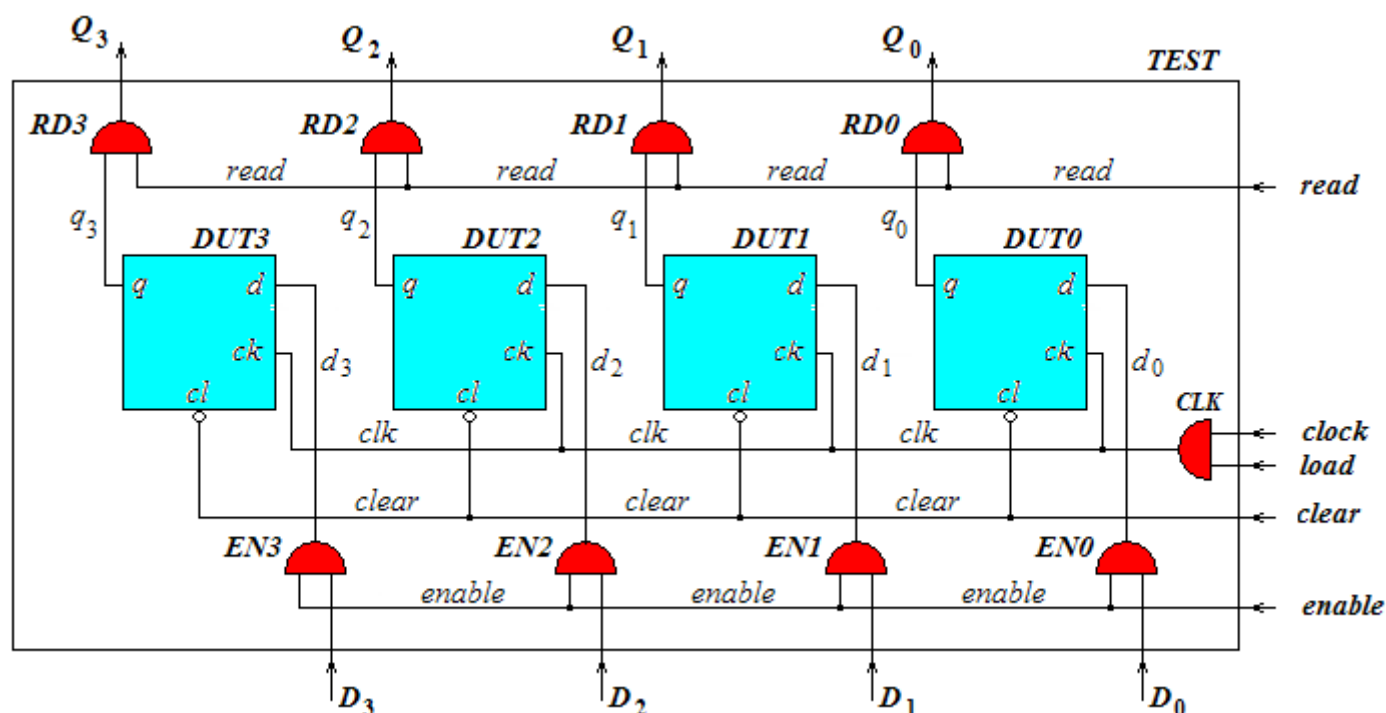


Figura 6 Schema logică a unui registru paralel de 4 biți, cu bistabile tip D

Dintre memoriile utilizate în sistemele digitale de prelucrare a datelor, registrul paralel este memoria cu accesarea cea mai rapidă.

S-a luat, ca exemplu, pentru modelarea în Verilog și testarea cu ModelSim registrul din figura 6 de mai sus, a cărei schemă bloc este prezentată în figura 7 de mai jos.

### a) Schema bloc

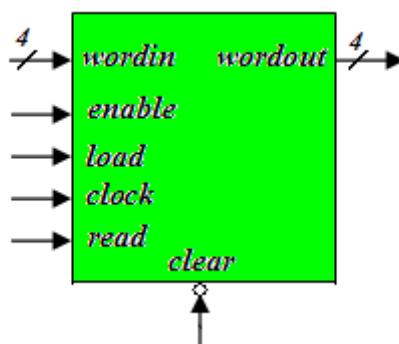




Figura 7. Schema bloc a registrului cu încărcare/descărcare paralelă

## b) Semnificația semnalelor

enable	$d_3d_2d_1d_0$	Semnificație
0	0000	Se inhibă aplicarea datelor la intrările $d$ ale bistabilelor
1	$D_3D_2D_1D_0$	Este permisă aplicarea datelor la intrările $d$ ale bistabilelor

clear	$q_3q_2q_1q_0$	Semnificație
0	0000	Se resetează (inițializare pe 0) bistabilele registrului
1	$q_3q_2q_1q_0$	Se inactivează resetarea asincronă

clk	$q_3q_2q_1q_0$	Semnificație
1,  , 0	$q_3q_2q_1q_0$	Se păstrează setările $q$ existente la ieșirile bistabilelor
	$d_3d_2d_1d_0$	Se încarcă bistabilele pe frontul pozitiv al ceasului

read	$Q_3Q_2Q_1Q_0$	Semnificație
0	0000	Se inactivează (pe zero) citirea la ieșirile registrului
1	$q_3q_2q_1q_0$	Se activează citirea datelor $q$ la ieșirile $Q$ din registru

Tabelele evidențiază acțiunile registrului pentru semnalele **enable**, **clear**, **clk** și **read**. Dacă semnalul **enable** este 1 (activ), atunci datele prezente la intrările  $D$  ale registrului sunt aplicate pe firele de intrare  $d$  ale bistabilele. Datele de la intrarea bistabilelor se vor încărca în registru doar pe frontul pozitiv al tactului **clk** de clock. În absența unui front pozitiv de clock, registrul își păstrează starea deja memorată (starea de inițializare sau datele deja memorate). Încărcarea biților cuvântului în registru se face concomitent (în paralel), la fel și descărcarea biților cuvântului din registru.

## c) Codul Verilog

Codul Verilog pentru **registru de 4 biți, cu bistabile tip D, cu încărcare/descărcare derivație** pe frontul pozitiv de **clock**, cu inițializare pe 0 (semnal asincron de **clear**, activ pe 0) și cu semnal **enable** de autorizare intrare date (activ pe 1). În starea de aplicare date ne-autorizată, firele de intrare ale bistabilelor s-au considerat în starea 0.

➤ Modelarea bistabilului tip D, fișierul *dff.v* :

```

module dff (cl, d, ck, q); // device flip-flop, de tip d
input cl, d, ck; // porturi de intrare, de 1 bit
output q; // port de iesire, de 1 bit
reg q; // iesirea q declarata de tip registru

always @(posedge ck, negedge cl)
begin
if (cl == 0) //daca pe intrarea cl a cbb este aplicat 0
q = 0;
else
q = d;
end

```

endmodule

➤ Modelarea porții AND cu 2 intrări, fișierul *AND.v* :

```
module AND (in1, in2, out); // poarta SI cu 2 intrari
  input in1, in2;          // porturi de intrare, de 1 bit
  output out;              // port de iesire, de 1 bit
  wire in1, in2, out;      //intrari si iesire tip fir (declaratie inutila)

  assign out = in1 & in2; //assign implica oricum declararea wire
endmodule
```

➤ Modelarea generatorului de stimuli, fișierul *test\_bench.v* :

```
module test_bench (enable, clear, clk, load, read, D3, D2, D1, D0);
  output enable, clear, clk, load, read, D3, D2, D1, D0;
  reg enable, clear, clk, load, read;
  reg[3:0] wordin;

  initial
  begin
    wordin <= 4'b1011;

    clk = 1'b0; // se pozitioneaza pe zero semnalul clk de ceas
    enable = 0; // sunt trecute toate firele d de intrare date pe 0 (zero)
    clear = 1; // este dezactivata stergerea clear
    clear <= #1 0; // se activeaza stergerea clear
    clear <= #2 1; // se dezactiveaza din nou stergerea clear
    clear <= #18 0; // se activeaza din nou stergerea clear
    clear <= #19 1; // se dezactiveaza stergerea clear
    load = 0; // se blocheaza aplicarea tactelor de clock
    read = 0; // se inhiba citirea datelor la iesirile Q din registru
    enable <= #8 1; // se permite aplicarea datelor D registrului
    load <= #9 1; // se permite aplicarea tactelor de clock
    read <= #13 1; // se permite citirea datelor la iesirile Q ale registrului

    repeat(4) // se repeta de 4 ori blocul begin-end
    begin
      #5 clk <= ~clk; //comutare ceas din 5 in 5 unitati timp
    end
  end

  assign D3 = wordin[3]; //atribuire lui D3 valoarea bitului wordin[3]
  assign D2 = wordin[2]; // idem bit D2
  assign D1 = wordin[1]; // idem bit D1
  assign D0 = wordin[0]; // idem bit D0

endmodule
```

➤ Modelul test de interfațare conexiuni instanțe, fișierul *test.v*:

```
module test ();
  wire enable, clear, load, clock, read ;
  wire D3,D2,D1,D0,d3,d2,d1,d0,q3,q2,q1,q0,Q3,Q2,Q1,Q0;
  wire wordout[3:0];

  AND EN3 (.in1(D3), .in2(enable),.out(d3));
  AND EN2 (.in1(D2), .in2(enable),.out(d2));
  AND EN1 (.in1(D1), .in2(enable),.out(d1));
  AND EN0 (.in1(D0), .in2(enable),.out(d0));

  dff DUT3 (.d(d3), .cl(clear), .ck(clk), .q(q3));
  dff DUT2 (.d(d2), .cl(clear), .ck(clk), .q(q2));
  dff DUT1 (.d(d1), .cl(clear), .ck(clk), .q(q1));
  dff DUT0 (.d(d0), .cl(clear), .ck(clk), .q(q0));

  AND RD3 (.in1(q3), .in2(read),.out(Q3));
  AND RD2 (.in1(q2), .in2(read),.out(Q2));
  AND RD1 (.in1(q1), .in2(read),.out(Q1));
  AND RD0 (.in1(q0), .in2(read),.out(Q0));

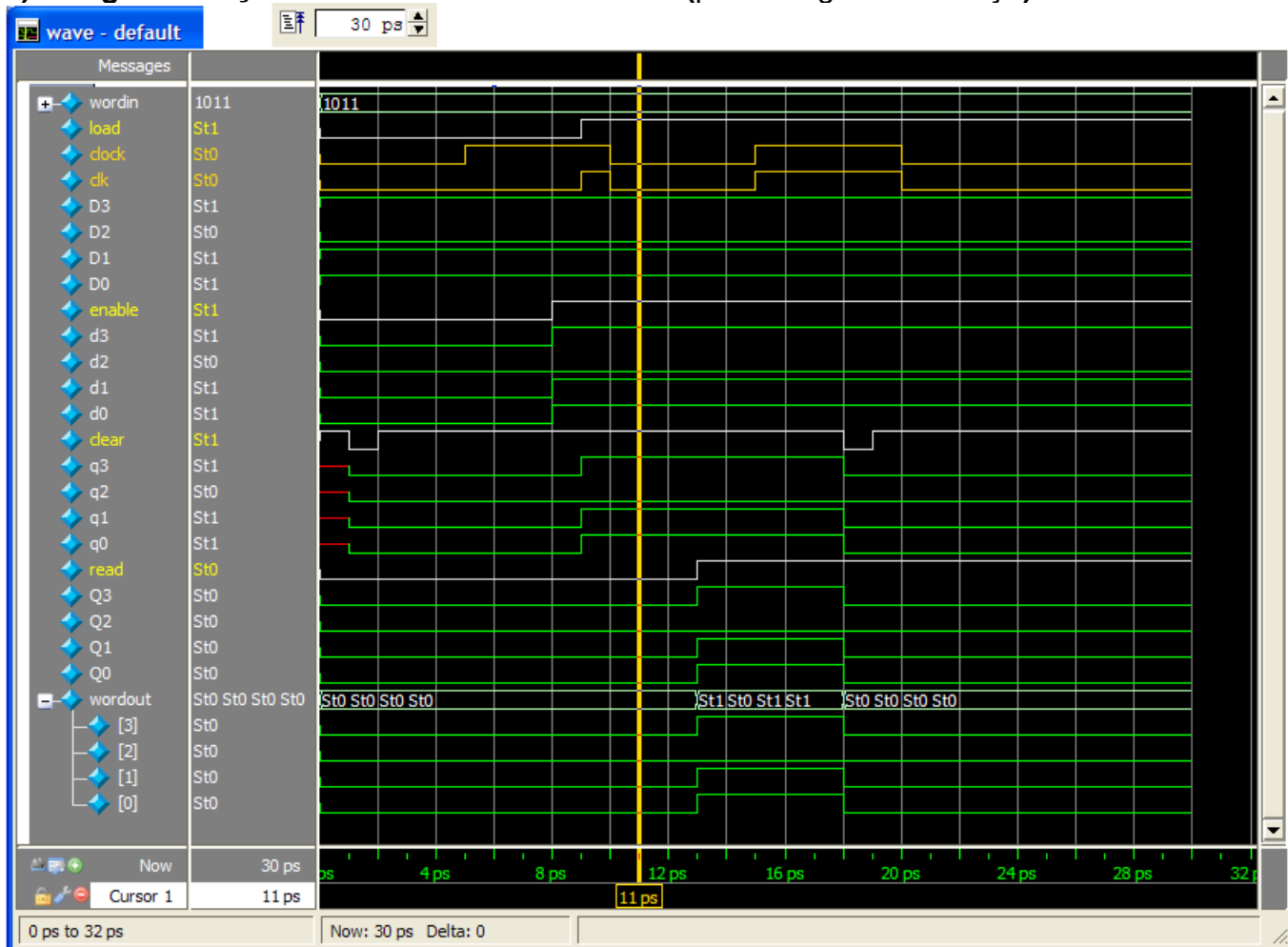
  AND CLK (.in1(clock), .in2(load),.out(clk));

  test_bench TB (
    .enable(enable),
    .clear(clear),
    .clk(clock),
    .load(load),
    .D3(D3), .D2(D2), .D1(D1), .D0(D0),
    .read(read)
  );

  // constituirea cuvântului wordout citit la iesirea din registru
  assign wordout[3] = Q3;
  assign wordout[2] = Q2;
  assign wordout[1] = Q1;
  assign wordout[0] = Q0;

endmodule
```

## d) Diagrama obținută la simulare în ModelSim (pentru registrul derivație)



## Observație:

Cuvântului **wordin** ales pentru încărcare în registru i s-a alocat secvența binară 1011 (respectiv 11 zecimal). După 4 comutări succesive ale ceasului, la intervale de 5 ns (deci după 20 ns), procesul simulării se oprește, moment în care cuvântul **wordout** citit/descărcat din registru are deja componența binară a lui **wordin**.

Semnalul **ck** de ceas devine valid abia la momentul 9 ns când **load** devine activ (pe 1).

Firele **d<sub>3</sub>d<sub>2</sub>d<sub>1</sub>d<sub>0</sub>** primesc valorile lui **D<sub>3</sub>D<sub>2</sub>D<sub>1</sub>D<sub>0</sub>** la momentul 8 ns când **enable** devine activ (pe 1).

Firele **q<sub>3</sub>q<sub>2</sub>q<sub>1</sub>q<sub>0</sub>** își transmit valorile ieșirilor **Q<sub>3</sub>Q<sub>2</sub>Q<sub>1</sub>Q<sub>0</sub>** ale registrului la momentul 13 ns când **read** devine activ (1).

e) Panoul **list** afișat la simularea în ModelSim

list																								
ps	wordin	load	clock	clk	D3	D2	D1	D0	enable	d3	d2	d1	d0	clear	q3	q2	q1	q0	read	Q3	Q2	Q1	Q0	
0	1011	St0	St0	St0	St1	St0	St1	St1	0	St0	St0	St0	St0	St1	StX	StX	StX	StX	St0	St0	St0	St0	St0	
1	1011	St0	St0	St0	St1	St0	St1	St1	0	St0	St0	St0	St0	St0	St0	St0	St0	St0	St0	St0	St0	St0	St0	
2	1011	St0	St0	St0	St1	St0	St1	St1	0	St0	St0	St0	St0	St1	St0	St0	St0	St0	St0	St0	St0	St0	St0	
5	1011	St0	St1	St0	St1	St0	St1	St1	0	St0	St0	St0	St0	St1	St0	St0	St0	St0	St0	St0	St0	St0	St0	
8	1011	St0	St1	St0	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St0	St0	St0	St0	St0	St0	St0	St0	St0	
9	1011	St1	St1	St1	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St1	St0	St1	St1	St0	St0	St0	St0	St0	
10	1011	St1	St0	St0	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St1	St0	St1	St1	St0	St0	St0	St0	St0	
13	1011	St1	St0	St0	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St1	St0	St1	St1	St0	St1	St1	St0	St1	
15	1011	St1	St1	St1	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St1	St0	St1	St1	St0	St1	St1	St0	St1	
18	1011	St1	St1	St1	St1	St0	St1	St1	1	St1	St0	St1	St1	St0	St0	St0	St0	St0	St0	St1	St0	St0	St0	
19	1011	St1	St1	St1	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St0	St0	St0	St0	St0	St1	St0	St0	St0	
20	1011	St1	St0	St0	St1	St0	St1	St1	1	St1	St0	St1	St1	St1	St0	St0	St0	St0	St0	St1	St0	St0	St0	