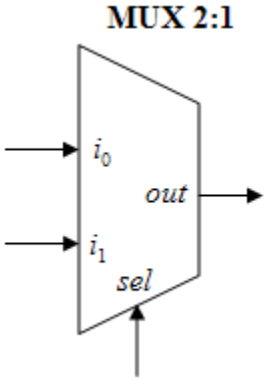
Modelarea/simularea circuitelor logice combinaţionale MUX şi DCD

1. Modelarea unui MUX 2:1 (multiplexor cu 2 itrări de date şi una de selecţie)

În multe situaţii intervine necesitatea efectuării mai multor transmiteri de informaţii pe un acelaşi canal. Cum acest lucru nu se poate face simultan, se recurge la o partajare în timp a acelui canal. Această acţiune de partajare este denumită ***multiplexare***. Ea are ca operaţie inversă ***demultiplexarea***, în cadrul căreia un semnal sosit pe o cale unică este dispersat pe mai multe căi derivaţie.

Multiplexorul oferă posibilitatea de selecţie a uneia dintre intrările de date ce va fi canalizată spre ieşire. Această selecţie se realizează cu un număr de intrări de selecţie care să asigure toate variantele de selecţie egale în număr cu numărul intrărilor de date.

a) Schema bloc a unui multiplexor cu 2 linii intrări de date (i_0, i_1), cu o linie de ieşire date (*out*) şi cu o linie intrare de selecţie (*sel*) este prezentată mai jos.



b) Tabelul de adevăr

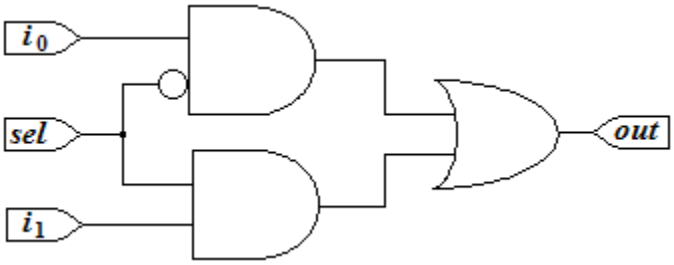
Circuitul este astrfel construit încât, când semnalul de selecţie ***sel*= 0**, la ieşire se va afla valoarea semnalului i_0, iar când ***sel*=1**, la ieşire se va afla valoarea semnalului i_1. Deci ***sel,*** în acest caz, este o variabilă de 1 bit ce poate cunoaşte două stări logice care sunt totodată stări de selecţie pentru cele două intrări de date.

Input	*Output*
sel	*out*
0	i_0
1	i_1

c) Funcţia logică executată de către MUX:

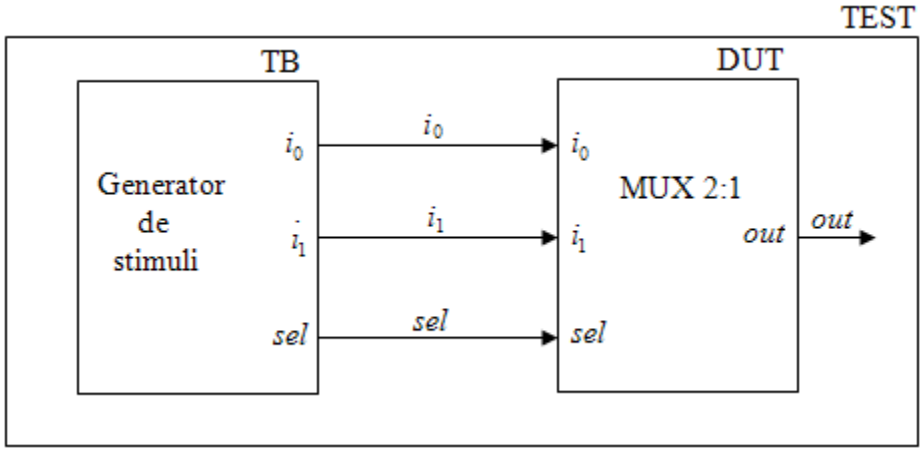
$$out = i_0 \cdot \overline{sel} + i_1 \cdot sel$$

d) Structura multiplexorului poate fi realizată cu două porţi logice AND, un inversor şi o poartă OR, conform schemei logice de mai jos.



e) Schema bloc a dispozitivului de simulare

Obiectul simulării de faţă este multiplexorul MUX 2:1, botezat DUT (*Device Under Test*). Simularea se face cu ajutorul semnalelor de test (stimuli) generate de un generator de stimuli botezat TB (*Test Bench*). Reprezentarea bloc a celor două dispozitive logice este dată mai jos. Legăturile dintre aceste două blocuri sunt asigurate de blocul TEST cu rol de părinte. În desen sunt prezentate denumirile porturilor de intrare şi de ieşire, precum şi denumirile firelor de legătură. Sensul circulaţiei stimulilor pe fire este de la blocul TB spre blocul DUT.



f) Codul Verilog scris pentru descrierea şi simularea multiplexorului MUX 2:1

Pentru modelarea comportamentală a multiplexorului s-a folosit instrucţiunea de atribuire continuă ***assign***, în două variante:

- în var.1, cu utilizarea operatorilor "***&***", "**~**" şi "**|**" (denumiţi AND, INV şi respectiv OR) şi
- în var.2, prin utilizarea operatorului condiţional *"**?**",* prin care variabilei ***out*** i se atribuie valoarea lui i_1 dacă *sel=true* = 1, respectiv a lui i_0 dacă *sel=false* = 0).

- **Descrierea comportamentală a multiplexorului MUX 2:1**. Descrierea este conţinută în modulul de mai jos botezat **mux**. Acest modul se va salva în fişierul denumit **mux.v**, în directorul proiectului**.**

Varianta 1 de modelare, cu utilizarea operatorilor "&" "~" şi "|".

```
1  //Descrierea strunctcurala a multiplexorului de 2 biti. Versiunea 1
2  module mux1 (i0, i1, sel, out);
3
4  input i0, i1, sel;
5  output out;
6
7  assign out = (i0 & (~sel)) | (i1 & sel);
8
9  endmodule
```

Varianta 2 de modelare, cu utilizarea operatorului condiţional "?"

```
1  //Descrierea strunctcurala a multiplexorului de 2 biti,versiunea 2
2  module mux2 (i0, i1, sel, out);
3
4  input i0, i1, sel;
5  output out;
6
7  assign out = (sel) ? i1 : i0;// s-a folosit operatorul conditional
8
9  endmodule
```

- **Descrierea comportamentală a Test Bencher-ului TB**. Descrierea este conţinută în modulul botezat **test_bench_mux** de mai jos. Acest modul se va salva în fişierul denumit **test_bench_mux.v**, în directorul proiectului**.**

```
 1  module test_bench_mux (i0, i1, sel);
 2
 3  parameter per = 5; // semi-perioada de ceas pentru gen. stimuli
 4  output i0, i1, sel;
 5
 6  reg [2:0] counter;
 7  reg clk;
 8
 9  initial
10  begin
11    clk = 0;
12    counter = 0;
13  end
14
15  always @( posedge clk )
16  begin
17    if (counter == {3{1'b1}})
18    begin
19       $stop;
20    end
21    else
22    begin
23       counter = counter + 1;
24    end
25  end
26
27  assign i0 = counter[0];
28  assign i1 = counter[1];
29  assign sel = counter[2];
30
31  always
32    #per clk = ~clk;
33
34  endmodule
```

- Modulul **test_mux1** reprezintă descrierea structurală în cod Verilog, de nivel superior, a conexiunilor dintre blocurile DUT şi TB. Fişierul modulului se salvează sub numele de **test_mux1.v** în directorul proiectului.

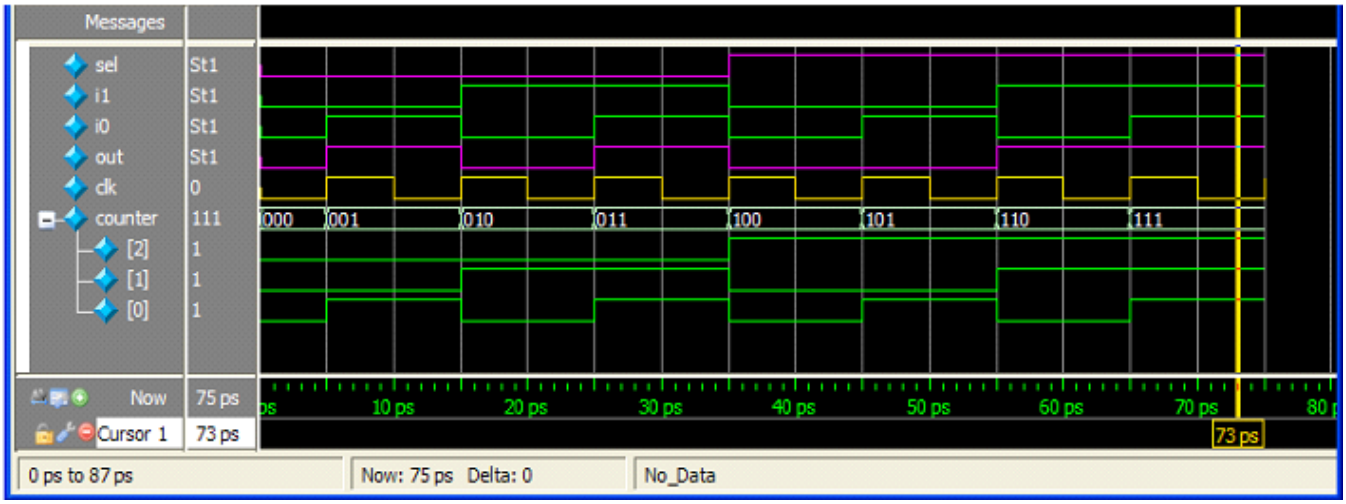
  În acest modul părinte, modulele blocurilor DUT şi TB apar instanţiate (citate) sub denumirile *mux1 DUT* şi respectiv *test_bench_mux BT*.

  În cazul simulării variantei 2 de MUX (varianta cu utilizarea operatorului condiţional ***"?"***) se va utiliza un modul părinte *test_mux2*, în care modulul blocului DUT trebuie să apară instanţiat (citat) sub denumirea *mux2 DUT*.

Observaţie

În parantezele înscrise după instanţe sunt reprezentat conexiunile în stilul mapării după nume (*named mapping*). De exemplu, în cazul instanţei *mux DUT*, înscrisul ***.a(a)*** reprezintă o conexiune prin care portul ***a*** al ***mux***–ului este legat la firul ***a*** (numele ***a*** al firului e înscris între paranteze). Pentru uşurinţa consemnării conexiunilor şi evitarea unor adresări greşite, se recomandă ca scrierea liniilor lor de cod să se facă cu consultarea schemei bloc.

```
1   /* Instantierea modulelor test_bench_mux
2      si mux1 in modulul test_mux1 */
3   module test_mux1();
4
5   parameter per = 5;
6
7   //wire i0, i1, sel, out;
8
9   test_bench_mux #(per) TB (
10      .i0(i0),
11      .i1(i1),
12      .sel(sel)
13      );
14
15  mux1 DUT (
16      .i0(i0),
17      .i1(i1),
18      .sel(sel),
19      .out(out)
20      );
21
22  endmodule
```



Pe diagramă, se observă că semnalul ***sel*** ia valoarea bitului *counter* [2] în vreme ce biţii *counter* [1] şi *counter* [0] sunt atribuiţi intrărilor i_1 şi respectiv i_0 de date.

Modelarea unui decodificator DCD 3:8

Decodificatoarele sunt circuite logice combinaţionale, cu n intrări şi 2^n ieşiri, la care fiecărei combinaţii distincte a celor n semnale binare de intrare îi corespunde aducerea în stare activă (1 logic) a uneia dintre cele 2^n ieşiri. Conform tabelei de adevăr de mai jos corspunzătoare decodificatorului DCD 3:8, valoarea 1 este generată doar pe ieşirea (Output) O_i al cărei indice zecimal este echivalentul secvenţei binare a celor trei biţi de intrare, pe celelalte ieşiri fiind generată starea 0 (zero) logic.

a) Tabelul de adevăr

Input			Output							
i_2	i_1	i_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b) Funcţia logică:

$$out = \bar{i}_0\bar{i}_1\bar{i}_2 + i_0\bar{i}_1\bar{i}_2 + \bar{i}_0 i_1\bar{i}_2 + i_0 i_1\bar{i}_2 + \bar{i}_0\bar{i}_1 i_2 + i_0\bar{i}_1 i_2 + \bar{i}_0 i_1 i_2 + i_0 i_1 i_2$$

c) Codul verilog

- Descrierea comportamentală în cod Verilog a decodificatorului DCD 3:8 este cuprinsă în corpul modulului botezat **dec**, salvat sub numele de fişier **dec.v**.

```
1  //Descrierea comportamentala a unui decodificator 3:8
2  module dec (in, enable, out);
3   input [2:0] in;  // dimensiunea liniei de adresare
4   input enable;
5   output[7:0] out; // dimensiunea liniei de cuvant
6
7   reg[7:0] out;    // intrucat out trebuie memorat, e ales de tip reg
8
9   always @(in or enable)            // <in> si <enable> sunt variabilele senzitive
10   if  (enable) out <= (1'b1 << in); /*cand enable=1, out e asignat cu un numar ce are
11                                       un singur bit 1, pozitionat la stanga de <in> ori */
12   else    out <= 8'b0;              //daca enable=0, cei 8 biti ai lui out sunt pusi pe 0
13 endmodule
```

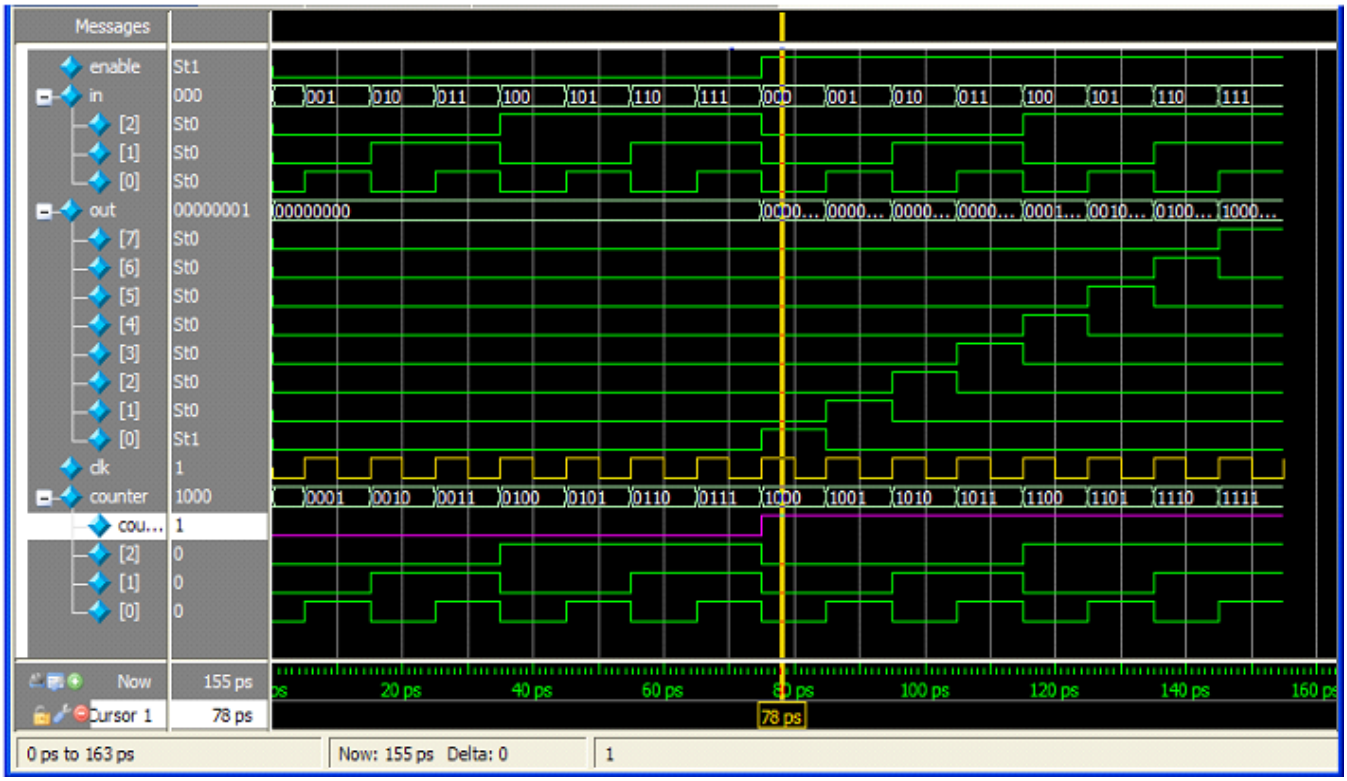
- Descrierea comportamentală în cod Verilog a generatorului TestBench (TB) de 3 stimuli este pezentată în modulul **tb_dec**, salvat în fişierul **tb_dec.v**.

```
1   module tb_dec (in, enable); // identificatorul (lista porturilor)
2
3   parameter per = 5;  // 5 untitati de timp ptr. semi-perioada de clock
4
5   output[2:0] in;     // cei trei biti stimuli trimisi decodificatorului
6   output enable;      // intrarea de autorizare decodificator
7
8   reg [3:0] counter;  // trebuie sa fie de tip reg
9   reg clk;            // trebuie sa fie de tip reg
10
11  initial            // blocul begin/end dupa initial se executa o singura data
12    begin
13      clk = 0;
14      counter = 0;
15    end
16
17  always @( posedge clk )
18    begin
19      if (counter == {4{1'b1}}) // cand cei 4 biti ai lui counter sunt 1111
20        begin
21         $stop;   // e rulat taskul $stop care stopeaza procesul de simulare
22        end
23      else
24        begin
25         counter = counter + 1; // altfel se incrementeaza cu 1 vectorul counter
26        end
27    end
28
29  assign in[0] = counter[0]; //bitul LSB al lui counter e atribuit bitului LSB al lui in
30  assign in[1] = counter[1];
31  assign in[2] = counter[2];
32  assign enable = counter[3]; //enable = bitul MSB counter
33
34  always
35    #per clk = ~clk; /* comutare clock in starea complementara,
36                        la intervale de 5 unitati de timp */
37
38  endmodule
```

- Codul Verilog al modulului părinte **test_dec**, care realizează structura conexiunilor dintre instanţele modulelor copii/fii **dec** si **tb_dec**, este prezentat mai jos. Modulul se va salva într-un fişier sub numele **test_dec.v.** Acest modul, pentru calitatea lui de părinte, este cel ce va trebui selectat în procedura de simulare în ModelSim.

```
1   module test_dec();     // Modulul de instantiere. Nu are parametri.
2     parameter per = 5;  // per e declarata parametru si atribuita cu valoarea 5
3
4     wire[2:0] in;    // semnalele de adresare DCD
5     wire enable;     // semnalul enable de autorizare DCD
6     wire[7:0] out;  // dimensiunea de 8 biti a liniilor de cuvant
7
8     tb_dec #(per) TB (.in(in),.enable(enable)); // portul < .in > se lega la firul < (in) >
9
10    dec DUT (.in(in),.enable(enable),.out(out));
11
12  endmodule
```

La simulare în ModelSim s-a obţinut diagrama ***wave*** de mai jos. Se poate observa că, abia după ce ***counte***r devine egal cu secvenţa binară 1000, variabila ***enable*** devine 1 şi autorizează decodificatorul să lucreze. Din acel moment (75 ps), secvenţa celor trei biţi ***in[2], in[1], in[0]*** activează pe rând liniile de cuvânt reprezentate de variabila vector ***out***.



În continuare, este prezentat modulul unei variante de decodificator 3:8 care conţine în corpul său şi generarea stimulilor necesari. Modulul a fost botezat ***decoder***.

Prin task-urile ***$display*** şi ***$monitor*** înscrise în liniile de cod 8 şi 9, se prevede afişarea, în panoul ***Transcript*** din fereastra aplicaţiei ModelSim, a valorilor variabilelor de intrare şi de ieşire. Momentele de afişare sunt stabilite cu întârzieri între ele de 1 ps.

```
 1  module decoder(); // decodificator 3:8 (3 intrari : 8 iesiri)
 2
 3    reg [2:0] in;  // cele 3 intrari ale DCD
 4    reg [7:0] out; // cele 8 iesiri ale DCD
 5
 6    initial
 7      begin
 8       $display("Time\t out[b]\t\t in[b]\t  in[d]");    // afiseaza capul de tabel
 9       $monitor("%g\t %b\t %b\t %d", $time,out,in,in);  // afiseaza valorile semnalelor
10       #1 in = 3'b000;
11       #1 in = 3'b001;
12       #1 in = 3'b010;
13       #1 in = 3'b011;
14       #1 in = 3'b100;
15       #1 in = 3'b101;
16       #1 in = 3'b110;
17       #1 in = 3'b111;
18      end
19
20      always @(in) out = (1'b1 << in);
21  endmodule
```

Transcript

# Time	out[b]	in[b]	in[d]
# 0	xxxxxxxx	xxx	x
# 1	00000001	000	0
# 2	00000010	001	1
# 3	00000100	010	2
# 4	00001000	011	3
# 5	00010000	100	4
# 6	00100000	101	5
# 7	01000000	110	6
# 8	10000000	111	7

Întocmit,
Îndrumător lucr. lab. ing. Hurubeanu Ştefan Valeriu
Braşov, la 27 febr. 2012.