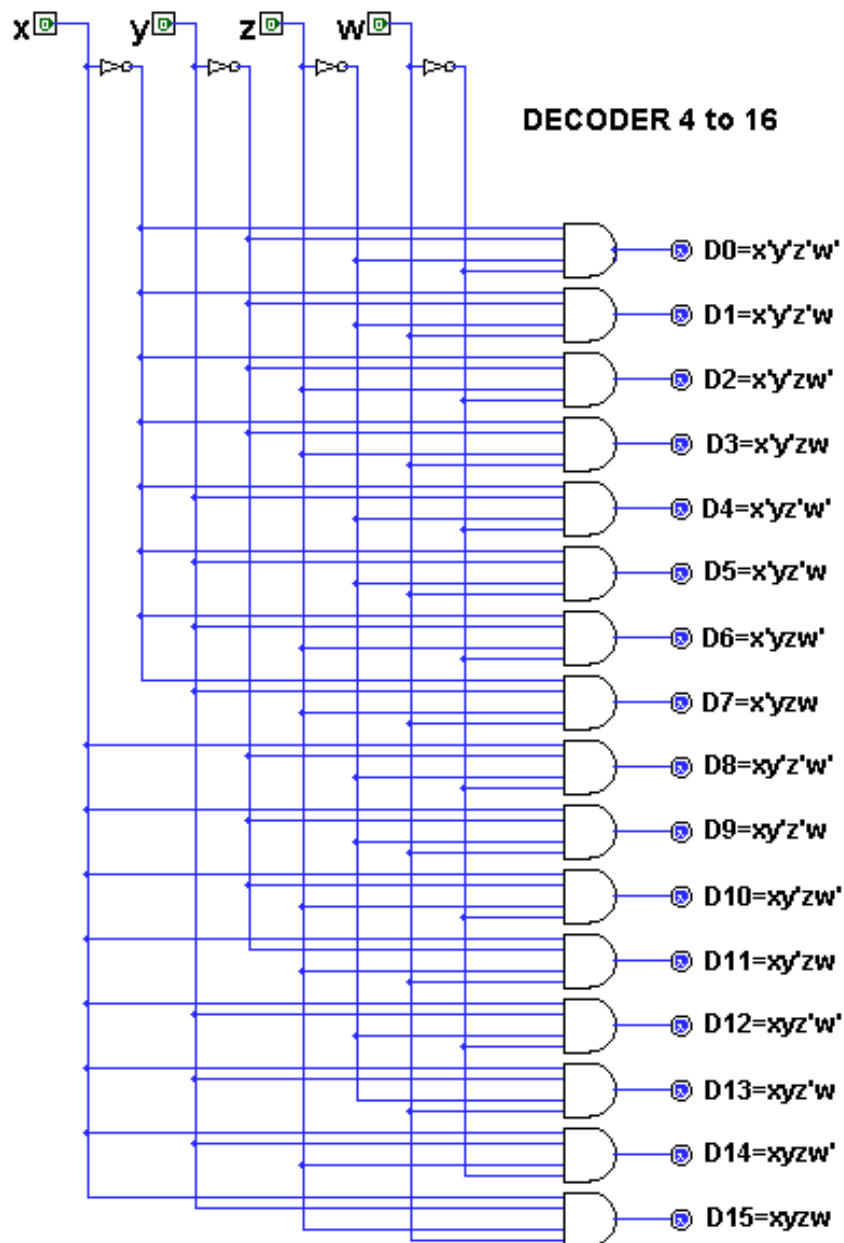


Ερωτήματα:

1. Υλοποίηση decoder 4 σε 16. Όπως αναφέρεται και στην εκφώνηση για κάθε είσοδο στον αποκωδικοποιητή ενεργοποιείται και μια διαφορετική έξοδος. Επομένως κάθε μια έξοδος από τις 16 αντιστοιχίζεται με έναν ελαχιστόρο στον αντίστοιχο χάρτη Karnaugh. Παρακάτω φαίνεται το διάγραμμα του κυκλώματος που σχεδιάστηκε αφού πρώτα γράφτηκε ο πίνακας αλήθειας και ο αντίστοιχος χάρτης. Για την υλοποίηση χρειαστήκαμε 16 πύλες and 4-άρων εισόδων και 4 αντιστροφείς.



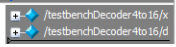
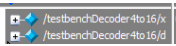
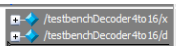

Κώδικας Verilog για το module decoder4to16:

```
module decoder4to16(in,d);  
    input [3:0] in;  
    output [15:0] d;  
    wire notx,noty,notz,notw;  
    not(notx,in[3]);  
    not(noty,in[2]);  
    not(notz,in[1]);  
    not(notw,in[0]);  
    and(d[15],notx,noty,notz,notw);  
    and(d[14],notx,noty,notz,in[0]);  
    and(d[13],notx,noty,in[1],notw);  
    and(d[12],notx,noty,in[1],in[0]);  
    and(d[11],notx,in[2],notz,notw);  
    and(d[10],notx,in[2],notz,in[0]);  
    and(d[9],notx,in[2],in[1],notw);  
    and(d[8],notx,in[2],in[1],in[0]);  
    and(d[7],in[3],noty,notz,notw);  
    and(d[6],in[3],noty,notz,in[0]);  
    and(d[5],in[3],noty,in[1],notw);  
    and(d[4],in[3],noty,in[1],in[0]);  
    and(d[3],in[3],in[2],notz,notw);  
    and(d[2],in[3],in[2],notz,in[0]);  
    and(d[1],in[3],in[2],in[1],notw);  
    and(d[0],in[3],in[2],in[1],in[0]);  
endmodule
```

Κώδικας Verilog για το module testbenchDecoder4to16 το οποίο εφαρμόζει όλους τους δυνατούς συνδυασμούς στο παραπάνω module:

```
module testbenchDecoder4to16;  
    reg [3:0] x;  
    wire [15:0] d;  
    decoder4to16 dec0(x,d);  
    initial  
        x=1'b0;  
    always  
        #10 x<=x+1;  
endmodule
```

Παρακάτω φαίνονται ενδεικτικά 3 screenshots από την εξομοίωση:

	<code>/testbench/Decoder4to16/x</code>	0000	0000	0001	0010	0011	0100	0101
	<code>/testbench/Decoder4to16/y</code>	1000000000000000	1000000000000000	0100000000000000	0010000000000000	0001000000000000	0000100000000000	0000010000000000
	<code>/testbench/Decoder4to16/z</code>	0000	0110	0111	1000	1001	1010	1011
	<code>/testbench/Decoder4to16/w</code>	1000000000000000	0000000100000000	0000000010000000	0000000001000000	0000000000100000	0000000000010000	0000000000001000

2. Για να μπορέσουμε να σχεδιάσουμε το κύκλωμα 4 σε 8 υπολογισμού του τετραγώνου των εισόδων αρχικά υπολογίζουμε τον πίνακα αλήθειας, ο οποίος φαίνεται παρακάτω. Οι στήλες x, y, z, w αποτελούν τα bit εισόδου του κυκλώματος, οι στήλες D0, D1, D2, D3, D4, D5, D6, D7 αποτελούν τις εξόδους του κυκλώματος, ενώ οι στήλες in, out δεν αποτελούν μέρος του κυκλώματος και έχουν προστεθεί για να μας δείξουν τις αντίστοιχες εισόδους και εξόδους στο δεκαδικό.

Πίνακας αλήθειας

x	y	z	w	D0	D1	D2	D3	D4	D5	D6	D7	in(dec)	out(dec)
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	0	0	1	0	0	2	4
0	0	1	1	0	0	0	0	1	0	0	1	3	9
0	1	0	0	0	0	0	1	0	0	0	0	4	16
0	1	0	1	0	0	0	1	1	0	0	1	5	25
0	1	1	0	0	0	1	0	0	1	0	0	6	36
0	1	1	1	0	0	1	1	0	0	0	1	7	49
1	0	0	0	0	1	0	0	0	0	0	0	8	64
1	0	0	1	0	1	0	1	0	0	0	1	9	81
1	0	1	0	0	1	1	0	0	1	0	0	10	100
1	0	1	1	0	1	1	1	1	0	0	1	11	121
1	1	0	0	1	0	0	1	0	0	0	0	12	144
1	1	0	1	1	0	1	0	1	0	0	1	13	169
1	1	1	0	1	1	0	0	0	1	0	0	14	196
1	1	1	1	1	1	1	0	0	0	0	1	15	225

Στη συνέχεια για κάθε έξοδο D0, D1, D2, D3, D4, D5, D6, D7 πρέπει να φτιάξουμε τον αντίστοιχο χάρτη Karnaugh. Άρα θέλουμε να φτιάξουμε 8 χάρτες συνολικά. Αφού φτιάξουμε τον χάρτη για την αντίστοιχη έξοδο συμπληρώνουμε τις αντίστοιχες θέσεις χρησιμοποιώντας τον πίνακα αλήθειας και ομαδοποιούμε όσο το δυνατόν περισσότερους άσσους, χωρίς να αφήσουμε κανέναν έξω. Στο τέλος εξάγουμε την συνάρτηση της εξόδου και αυτό ήταν.

Χάρτες Karnaugh

i. Για την έξοδο **D0**:

zw \ xy	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	0	0

$$D0=xy$$

ii. Για την έξοδο **D1**:

zw \ xy	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	1
10	1	1	1	1

$$D1=xy'+xz=x(y'+z)$$

iii. Για την έξοδο **D2**:

zw \ xy	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	1

$$D2=x'yz+xyw+xy'z=x'yz+x(yw+y'z)$$

iv. Για την έξοδο **D3**:

zw \ xy	00	01	11	10
00	0	0	0	0
01	1	1	1	0
11	1	0	0	0
10	0	1	1	0

$$D3=x'yw+yz'w'+xy'w=y(x'w+z'w')+xy'w$$

v. Για την έξοδο **D4**:

zw \ xy	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	0	1	0	0
10	0	0	1	0

$$D4=yz'w+y'zw=w(z'y+zy')=w(z\oplus y)$$

vi. Για την έξοδο **D5**:

zw \ xy	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	0	0	1
10	0	0	0	1

$$D5=w'z$$

vii. Για την έξοδο **D6**:

xy \ zw	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

D6=0

viii. Για την έξοδο **D7**:

xy \ zw	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

D7=w

Εφόσον τώρα έχουμε τις συναρτήσεις των εξόδων μπορούμε εύκολα να προχωρήσουμε στην υλοποίηση μέσω Verilog.

Κώδικας Verilog για το module sqr:

```

module sqr(in,d);
  input [3:0] in;
  output [7:0] d;
  wire notx,noty,notz,notw,v1,v2,v3,v4,v5,v6,v7,v8;
  not(notx,in[3]);
  not(noty,in[2]);
  not(notz,in[1]);
  not(notw,in[0]);
  //ipologismos tou d[7]
  and(d[7],in[3],in[2]);
  //ipologismos tou d[6]
  or(v1,noty,in[1]);
  and(d[6],in[3],v1);
  //ipologismos tou d[5]
  and(v2,notx,in[2],in[1]);
  and(v3,in[3],in[2],in[0]);
  and(v4,in[3],noty,in[1]);
  or(d[5],v2,v3,v4);
  //ipologismos tou d[4]
  and(v5,notx,in[2],in[0]);
  and(v6,in[2],notz,notw);
  and(v7,in[3],noty,in[0]);
  or(d[4],v5,v6,v7);

```

```

//ipologismos tou d[3]
xor(v8,in[1],in[2]);
and(d[3],in[0],v8);
//ipologismos tou d[2]
and(d[2],notw,in[1]);
//ipologismos tou d[1]
buf(d[1],1'b0);
//ipologismos tou d[0]
buf(d[0],in[0]);
endmodule

```

Κώδικας Verilog για το module testbenchSqr:

```

module testbenchSqr;
    reg [3:0] x;
    wire [7:0] d;
    sqr sqr0(x,d);
    initial
        x=4'b0;
    always
        #10 x<=x+1;
endmodule

```

Παρακάτω φαίνονται οι δυνάμεις όλων των δυνατών εισόδων:

/testbenchSqr/x	0	0	1	2	3	4	5	6	7	8	9	
/testbenchSqr/d	0	1	4	9	16	25	36	49	64	81		

/testbenchSqr/x	15	6	7	8	9	10	11	12	13	14	15	
/testbenchSqr/d	225	36	49	64	81	100	121	144	169	196	225	