Δημήτρης Καρατζάς **icsd1302**
Νίκος Κατσιώπης **icsd13076**

ΑΝΑΦΟΡΑ 2ΗΣ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ ΣΤΗ ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΕΡΩΤΗΜΑ 1: DECODER 4-σε-16

Απάντηση:
Ο Decoder 4-16 δέχεται ένα 4bit σήμα και δίνει ως έξοδο ένα 16bit αριθμό. Συγκεκριμένα δέχεται όλους τους συνδυασμούς των 4 στοιχείων, και για κάθε ένα από αυτούς δίνει μια συγκεκριμένη τιμή, πχ

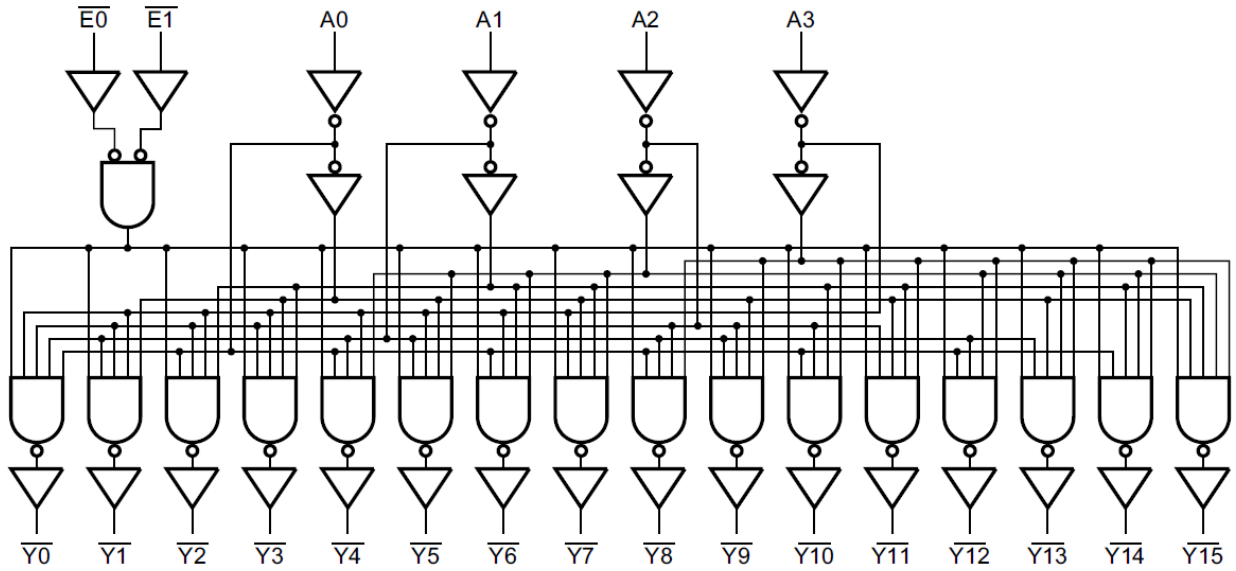
- 0000 → 0000000000000001
- 0001 → 0000000000000010
- 0010 → 0000000000000100
- ...(κτλ)...
- 1111 → 1000000000000000

Η υλοποίηση που κάναμε φαίνεται στο παρακάτω σχήμα

```
module decoder_4to16(
    input [0:3]in,
    output [0:15]out
);

wire not_in0;
wire not_in1;
wire not_in2;
wire not_in3;
not(not_in0, in[0]);
not(not_in1, in[1]);
not(not_in2, in[2]);
not(not_in3, in[3]);

and(out[15], not_in3, not_in2, not_in1, not_in0);
and(out[14], not_in1, not_in2, not_in3, in[0]);
and(out[13], not_in0, not_in2, not_in3, in[1]);
and(out[12], not_in2, not_in3, in[0], in[1]);
and(out[11], not_in0, not_in1, not_in3, in[2]);
and(out[10], not_in1, not_in3, in[0], in[2]);
and(out[9], not_in0, not_in3, in[1], in[2]);
and(out[8], not_in3, in[0], in[1], in[2]);
and(out[7], not_in0, not_in1, not_in2, in[3]);
and(out[6], not_in1, not_in2, in[0], in[3]);
and(out[5], not_in0, not_in2, in[1], in[3]);
and(out[4], not_in2, in[0], in[1], in[3]);
and(out[3], not_in0, not_in1, in[2], in[3]);
and(out[2], not_in1, in[0], in[2], in[3]);
and(out[1], not_in0, in[1], in[2], in[3]);
and(out[0], in[0], in[1], in[2], in[3]);

endmodule
```

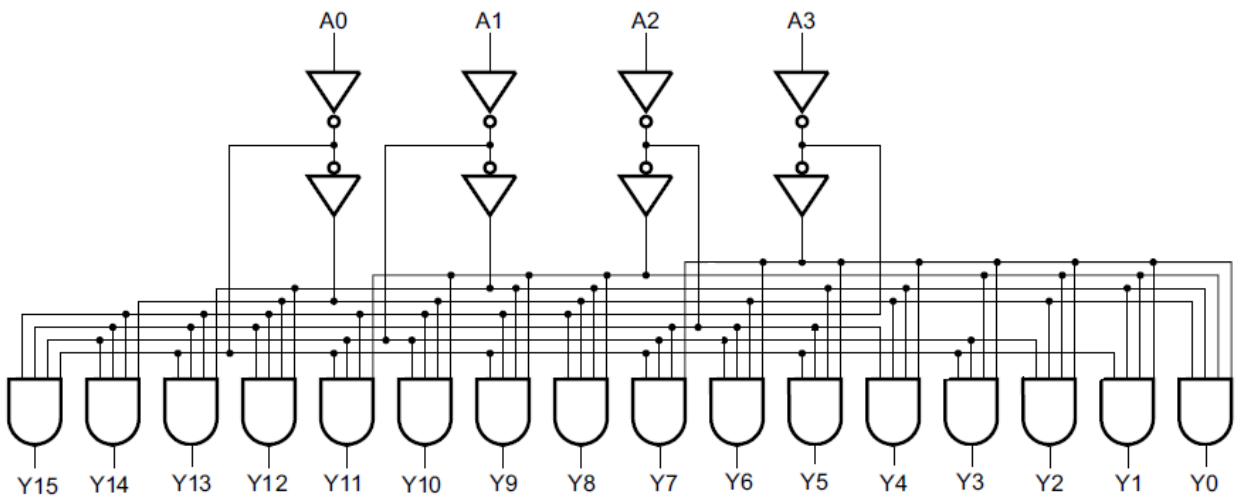
Σημείωση: οι εικόνες με κώδικα verilog που δείχνουμε εδώ δεν έχουν σχόλια για να μη πιάνουν χώρο, έχουμε όμως σχολιάσει τους κώδικες στα παραδοτέα ***.v*** *αρχεία.*

Το διάγραμμα του decoder μας βασίστηκε σε ένα ολοκληρωμένο το οποίο λειτουργεί ως decoder και demultiplexer. Εμείς τροποποιήσαμε το κύκλωμα ώστε να είναι μόνο decoder. Το αρχικό κύκλωμα του ολοκληρωμένου αυτού (που ονομάζεται **74HCT154**) φαίνεται παρακάτω



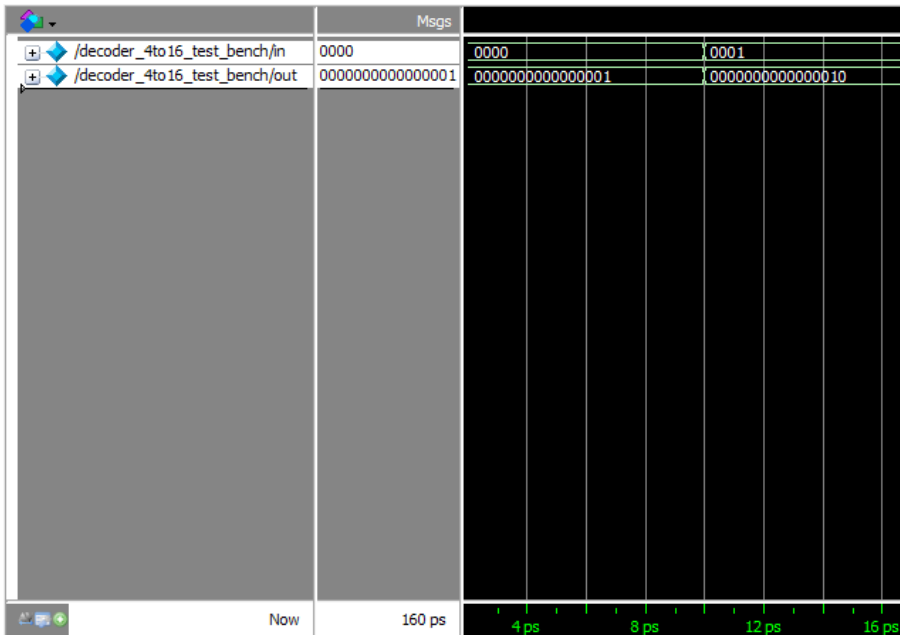
Για να τροποποιήσουμε το διάγραμμα αυτό, αρχικά αφαιρέσαμε το κομμάτι του κυκλώματος που λειτουργεί ως demultiplexer. Αυτό είναι το λογικό κύκλωμα που φαίνεται αριστερά (το οποίο δέχεται δυο εισόδους).
Στη συνέχεια αφαιρέσαμε τις γραμμές από κάθε πύλη NAND που συνδέεται σε αυτό, οπότε τώρα όλες οι πύλες NAND έχουν 4 εισόδους αντι για 5.
Για να βγουν σωστές οι έξοδοι, αντιστρέψαμε τα Y0…. Y15 σε Y15 …. Y0.
Για παράδειγμα, αν δώσουμε ως είσοδο το 0000, το παραπάνω κύκλωμα θα δώσει έξοδο 100000000000000 (βέβαια το κύκλωμα δίνει ως έξοδο το συμπλήρωμα της εξόδου οπότε στην ουσία δίνει το 0111111111111111 αλλά εμείς δε δίνουμε το συμπλήρωμα ώστε να μας βγει αυτό που θέλουμε).
Τέλος για δικιά μας ευκολία χρησιμοποιήσαμε AND αντί για NAND.
Όμως στο σενάριο μιας πραγματικής υλοποίησης, η χρήση πυλών NAND λόγω ότι είναι universal πύλες (μπορούμε μόνο με αυτές να υλοποιήσουμε όλες τις άλλες πύλες), προτιμώνται αντί για AND.

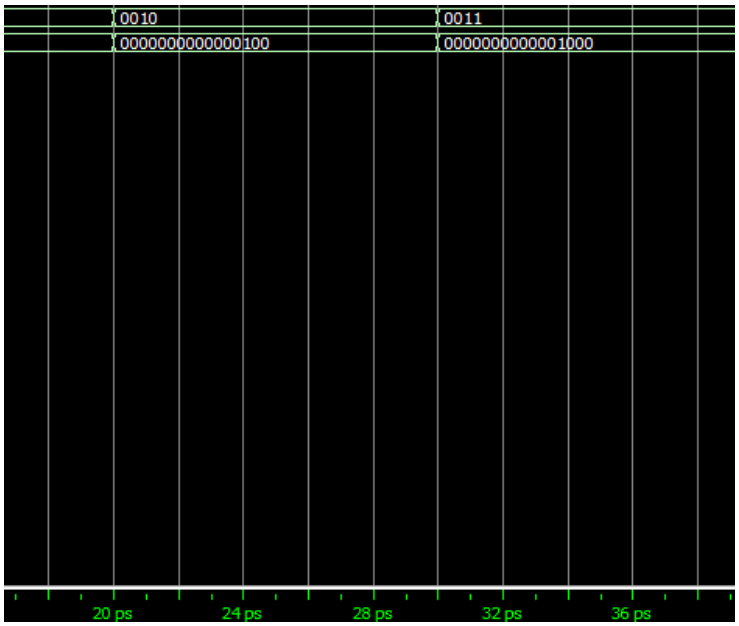
Παρακάτω είναι το τροποποιημένο κύκλωμα μας

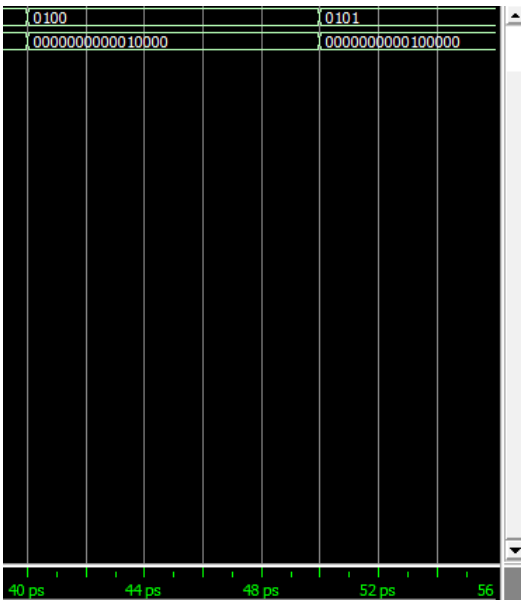


Ακολουθούν οι εικόνες υλοποίησης του test bench και το simulation.

```
module decoder_4to16_test_bench;

reg[0:3] in;
wire[0:15] out;

decoder_4to16 dec(in, out);
initial begin
  in = 4'b0000;
end
always begin
   #10 in = in + 1'b1;
end
endmodule
```







Στην πρώτη εικόνα έχουμε το test bench μας στο οποίο η είσοδος αλλάζει κάθε 10ps, οπότε για το simulation θέσαμε χρόνο εκτέλεσης = 10 * (2^4) ps = 160 ps για να εξομοιωθούν όλες οι πιθανές τιμές της εισόδου.
Στη συνέχεια βλέπουμε το simulation, στις παραπάνω εικόνες φαίνεται πως το κύκλωμα δουλεύει ως decoder.

ΕΡΩΤΗΜΑ 2: ΚΥΚΛΩΜΑ ΠΟΥ ΥΠΟΛΟΓΙΖΕΙ ΤΕΤΡΑΓΩΝΟ ΤΩΝ ΕΙΣΟΔΩΝ

Πρώτα ξεκινάμε δημιουργώντας των πίνακα αληθείας. Στη συνέχεια απλοποιούμε το κύκλωμα με χάρτες Karnaugh και τέλος σχεδιάζουμε το κύκλωμα.

Υπολογίζουμε σε δεκαδικό (για ευκολία) τα τετράγωνα των εισόδων από 0 (0000) μέχρι 15 (1111), δηλαδή:

```
 0 → 0,     1 → 1,     2 → 4,
 3 → 9,     4 → 16,    5 → 25,
 6 → 36,    7 → 49,    8 → 64,
 9 → 81,   10 → 100,  11 → 121,
12 → 144,  13 → 169,  14 → 196,
15 → 225
```

Τώρα απλώς βάζουμε τις τιμές αυτές στο πίνακα αληθείας (αφού πρώτα τις μετατρέψουμε στο δυαδικό σύστημα).

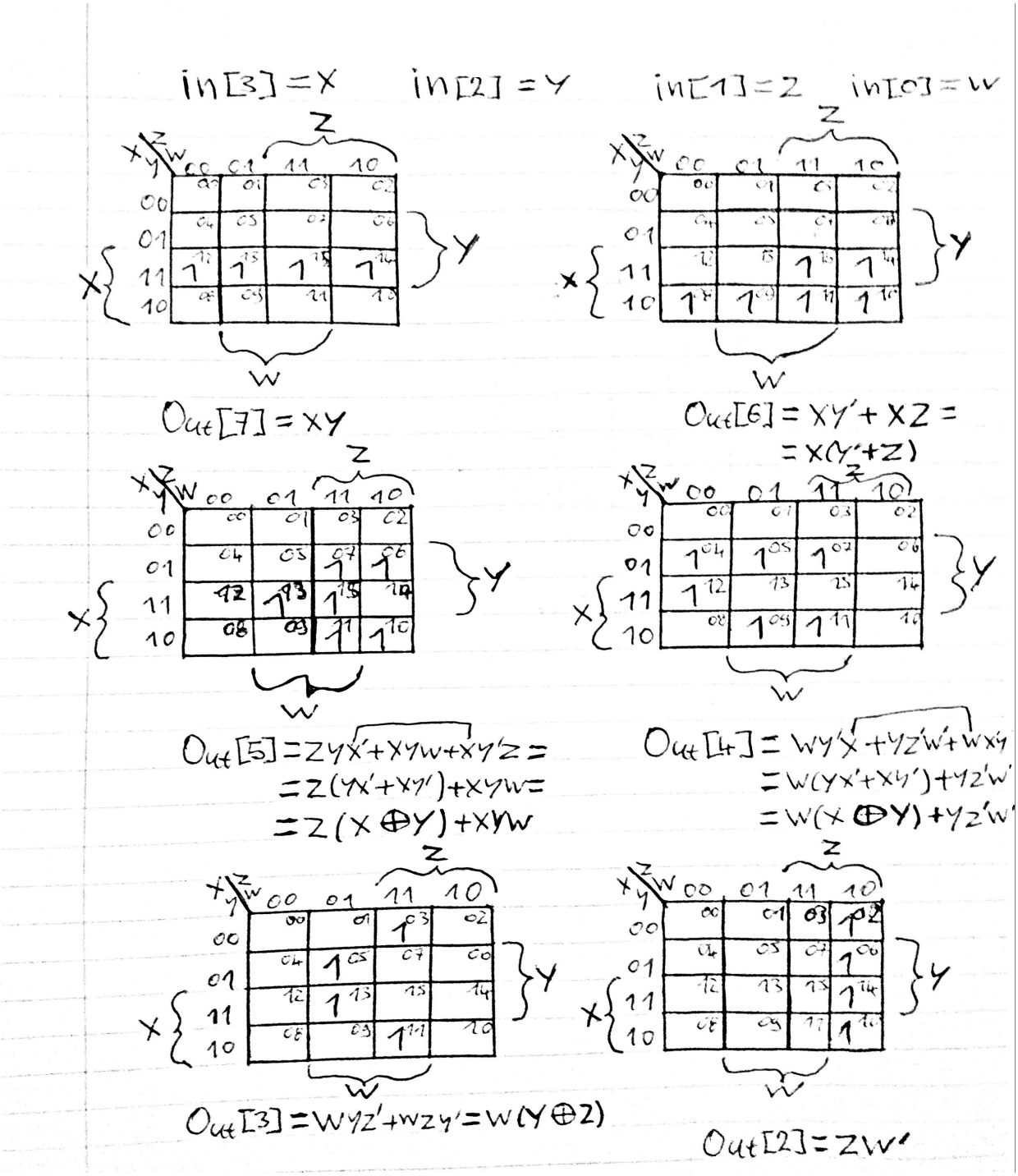
Πίνακας Αληθείας

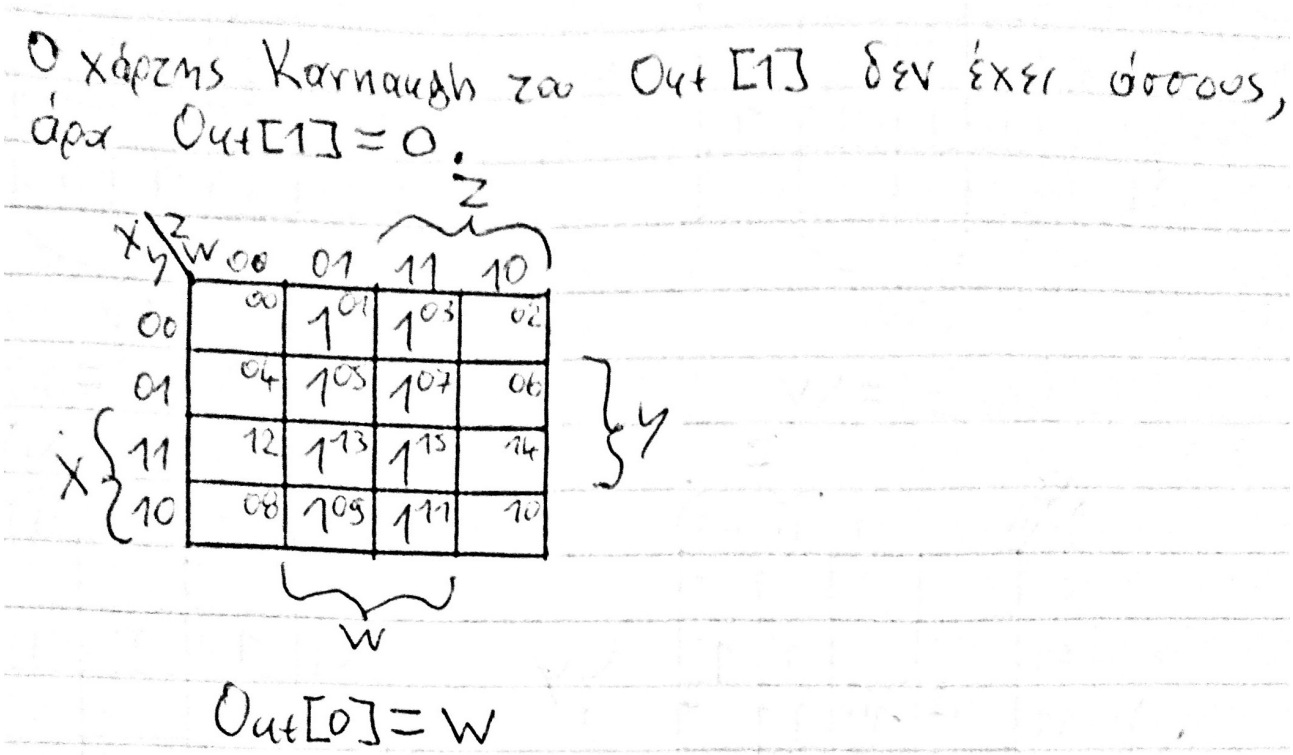
No.	in[3]	in[2]	in[1]	in[0]	out[7]	out[6]	out[5]	out[4]	out[3]	out[2]	out[1]	out[0]
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	1	0	0
3	0	0	1	1	0	0	0	0	1	0	0	1
4	0	1	0	0	0	0	0	1	0	0	0	0
5	0	1	0	1	0	0	0	1	1	0	0	1
6	0	1	1	0	0	0	1	0	0	1	0	0
7	0	1	1	1	0	0	1	1	0	0	0	1
8	1	0	0	0	0	1	0	0	0	0	0	0
9	1	0	0	1	0	1	0	1	0	0	0	1
10	1	0	1	0	0	1	1	0	0	1	0	0
11	1	0	1	1	0	1	1	1	1	0	0	1
12	1	1	0	0	1	0	0	1	0	0	0	0
13	1	1	0	1	1	0	1	0	1	0	0	1
14	1	1	1	0	1	1	0	0	0	1	0	0
15	1	1	1	1	1	1	1	0	0	0	0	1

Απλοποίηση του κυκλώματος με χάρτες Karnaugh

out[7] = Σ(12, 13, 14, 15)
out[6] = Σ(8, 9, 10, 11, 14 ,15)
out[5] = Σ(6, 7, 10, 11, 13 ,15)
out[4] = Σ(4, 5, 7, 9, 11 ,12)
out[3] = Σ(3, 5, 11, 13)
out[2] = Σ(2, 6, 10, 14)
out[1] = Σ( )
out[0] = Σ(1, 3, 5, 7, 9, 11, 13, 15)

Παρακάτω, δείχνουμε τους χάρτες **Karnaugh** και πως απλοποιούμε τις συναρτήσεις.





Ο Κώδικας για το **test bench**, καθώς και το **module** που υλοποιεί το ερώτημα, φαίνεται στις παρακάτω εικόνες

```
module square_module_test_bench;

reg[3:0] in;
wire[7:0] out;

square sq0(in, out);

initial begin
    in = 4'b0000;
end

always begin
    #10 in = in + 1'b1;
end
endmodule
```

```
module square(
    input [3:0]in,
    output [7:0]out
);

wire not_in3;
wire not_in2;
wire not_in1;
wire not_in0;

not(not_in3, in[3]);
not(not_in2, in[2]);
not(not_in1, in[1]);
not(not_in0, in[0]);
wire xor_in3_in2;
wire xor_in2_in1;
xor(xor_in3_in2, in[3], in[2]);
xor(xor_in2_in1, in[2], in[1]);

and(out[7], in[3], in[2]);

wire temp_6_or1;
or(temp_6_or1, not_in2, in[1]);
and(out[6], in[3], temp_6_or1);
```

```
wire temp_5_and1;
wire temp_5_and2;
and(temp_5_and1, in[1], xor_in3_in2);
and(temp_5_and2, in[3], in[2], in[0]);
or(out[5], temp_5_and1, temp_5_and2);

wire temp4_and1;
wire temp4_and2;
and(temp4_and1, in[0], xor_in3_in2);
and(temp4_and2, in[2], not_in1, not_in0);
or(out[4], temp4_and1, temp4_and2);

and(out[3], in[0], xor_in2_in1);

and(out[2], in[1], not_in0);

assign out[1] = 0;
assign out[0] = in[0];

endmodule
```

Το simulation φαίνεται παρακάτω, παρατηρούμε ότι η υλοποίηση του κυκλώματος είναι σωστή, εφόσον τα αποτελέσματα είναι τα επιθυμητά.