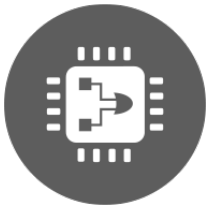


Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής II - Εργασία -

Αλέξανδρος Πετρίδης

Τελευταία ενημέρωση: 9 Ιουλίου 2021

Περιεχόμενα

1 Άσκηση 1

1.1 Πίνακας Αληθείας FSM

Τρέχουσα Κατάσταση	Είσοδος	Επόμενη Κατάσταση	Έξοδος
A	0	Δ	0
A	1	E	1
B	0	B	0
B	1	E	1
Γ	0	Γ	0
Γ	1	A	1
Δ	0	B	0
Δ	1	Γ	1
E	0	Γ	0
E	1	Δ	0

1.2 Πίνακας Κωδικοποίησης καταστάσεων

Κατάσταση	Κωδικοποίηση
A	000
B	001
Γ	010
Δ	011
E	100

1.3 Λογικές εξισώσεις για τις εξόδους των FSM

Μετασχηματίζω τον Πίνακα Αληθείας του FSM.

Τρέχουσα Κατάσταση	Είσοδος	Επόμενη Κατάσταση	Έξοδος
$D_2D_1D_0$	X	$D_2'D_1'D_0'$	Y
000	0	011	0
000	1	100	1
001	0	001	0
001	1	100	1
010	0	010	0
010	1	000	1
011	0	001	0
011	1	010	1
100	0	010	0
100	1	011	0

Ο οποίος μέσω πινάκων karnaugh καταλήγει στις παρακάτω εξισώσεις:

$$Y = \overline{D_2} \cdot X$$
$$D_2' = \overline{D_2} \cdot \overline{D_1} \cdot X$$
$$D_1' = (\overline{D_0} \cdot \overline{X}) + (D_1 \cdot D_0 \cdot X) + (D_2\overline{D_0})$$
$$D_0' = (\overline{D_2} \cdot \overline{D_1} \cdot \overline{X}) + (D_0 \cdot \overline{X}) + (D_2 \cdot X)$$

1.4 Πρώτο ερώτημα

Κώδικας Verilog για το ερώτημα (α):

1.4.1 Αρχείο a_FSM.v

```
module a_FSM (output reg y_out, input x_in, clk, reset);

    parameter
        A = 3'b000,
        B = 3'b001,
        C = 3'b010,
        D = 3'b011,
        E = 3'b100;

    reg [2:0] currentState, nextState;

    always @(posedge clk or posedge reset)
    begin: state_memory
        if(reset) currentState <= B;
        else       currentState <= nextState;
    end
    always @(x_in or currentState)
    begin: next_state_logic
        case(currentState)
            A: nextState = (x_in == 1'b1) ? E : D;
            B: nextState = (x_in == 1'b1) ? E : B;
            C: nextState = (x_in == 1'b1) ? A : C;
            D: nextState = (x_in == 1'b1) ? C : B;
            E: nextState = (x_in == 1'b1) ? D : C;
        endcase
        assign y_out = ((~currentState[2]) & x_in); // So it's a Mealy FSM
    end
endmodule
```

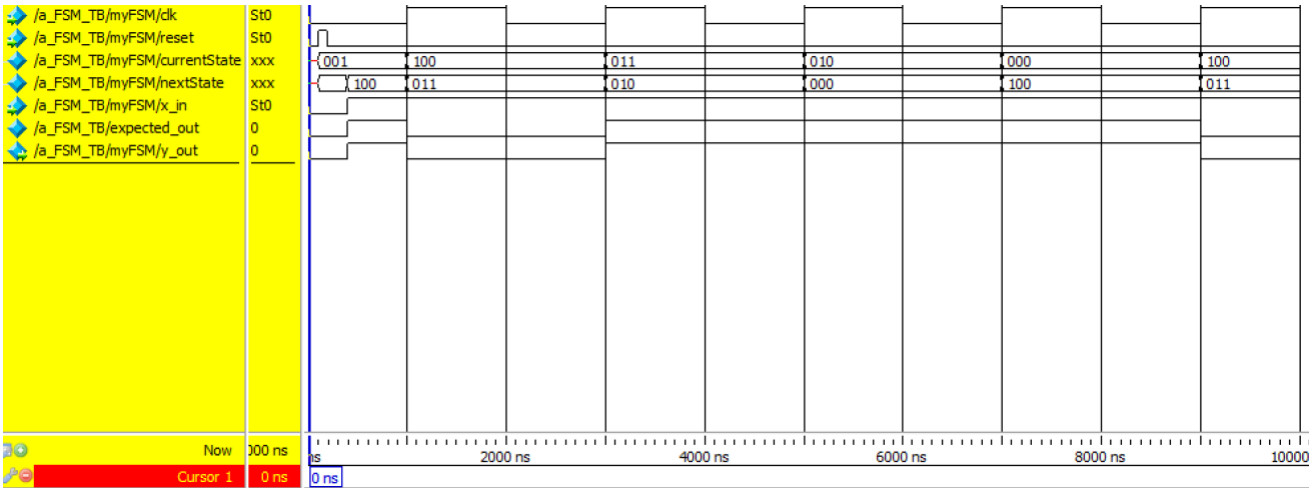
1.4.2 Αρχείο a_FSM_TB.v

```
`timescale 100ns/100ns      // For 1MHZ clock in #10
module a_FSM_TB;

    reg clk, reset, x_in, expected_out;
    wire y_out;

    a_FSM myFSM(y_out, x_in, clk, reset);

    initial
    begin
        clk = 1'b0;
        reset = 1'b0;
        x_in = 1'b0;
        expected_out = 1'b0;
        #1 reset = 1'b1;    // We need posedge of Reset to reset FSM
        #1 reset = 1'b0;
        // We are on State B(001)
        // with x_in = 0 we have y_out = 0 and nextState = B(001)
        #2 x_in = 1'b1;
        // with x_in = 1 we have y_out = 1 and nextState = E(100)
        expected_out = 1'b1;
        // at next posedge clock we have currentState = E(100)
        // and with x_in = 1 we have y_out = 0 and nextState = D(011)
        #6 expected_out = 1'b0;

        // at next posedge clock we have currentState = D(011)
        // and with x_in = 1 we have y_out = 1 and nextState = C(010)
        #20 expected_out = 1'b1;

        // at next posedge clock we have currentState = C(010)
        // and with x_in = 1 we have y_out = 1 and nextState = A(000)

        // at next posedge clock we have currentState = A(000)
        // and with x_in = 1 we have y_out = 1 and nextState = E(100)

```

```
35          // at next posedge clock we have currentState = E(100)
36          // and with x_in = 1 we have y_out = 0 and nextState = D(011)
37          #60 expected_out = 1'b0;
38
39
40      end
41
42      always
43      begin
44          #10 clk = ~clk;
45      end
46
47  endmodule
```

1.4.3 Αποτελέσματα του πρώτου FSM simulator

Παρατηρούμε πως τα αποτελέσματα είναι σωστά ελέγχοντας πρώτα τις καταστάσεις από τους πίνακες που δημιουργήθηκαν παραπάνω και ελέγχοντας ότι το expected_out είναι ίδιο με το y_out.



Σχήμα 1: Αποτελέσματα του πρώτου FSM simulator

1.5 Δεύτερο ερώτημα

Κώδικας Verilog για το ερώτημα (β):

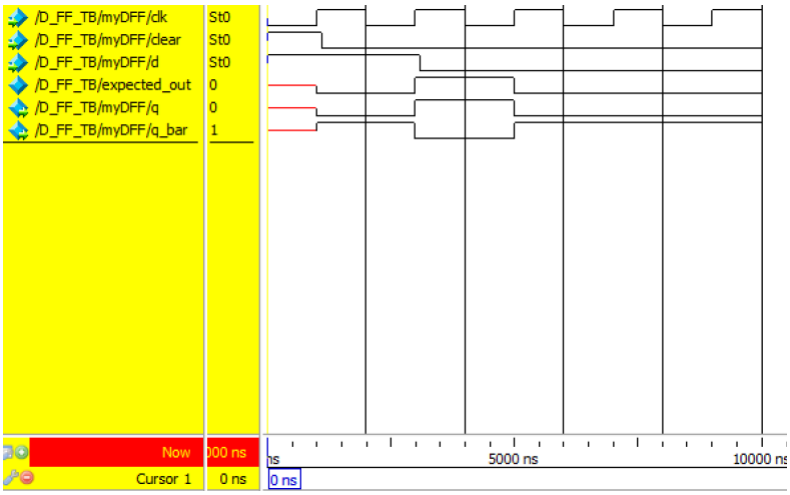
1.5.1 Αρχείο D_FF.v

```
0   module D_FF (output reg q, q_bar, input wire d, clk, clear);
1
2       always@(posedge clk)
3       begin
4           if(clear)
5           begin
6               q <= 0;
7               q_bar <= 1;
8           end
9           else
10          begin
11              q <= d;
12              q_bar <= ~d;
13          end
14      end
15
16  endmodule
```

1.5.2 Αρχείο D_FF_TB.v

```
0  `timescale 100ns/100ns      // For 1MHZ clock in #10
1  module D_FF_TB;
2
3      reg d, clk, clear, expected_out;
4      wire q, q_bar;
5
6      D_FF myDFF(q, q_bar, d, clk, clear);
7
8      initial
9      begin
10         clk = 1'b0;
11         clear = 1'b1;        // Set FF to q = 0, q_bar = 1 at first posedge clock
12         d = 1'b1;        // At second posedge clock q = d = 1, q_bar = 0
13         #10 expected_out = 1'b0;
14
15         #1 clear = 1'b0;
16         #19 expected_out = 1'b1;
17         #1 d = 1'b0;         // Set FF to q = d = 1, q_bar = 0 at next posedge clock
18         #19 expected_out = 1'b0;
19     end
20
21     always
22     begin
23         #10 clk = ~clk;
24     end
25
26 endmodule
```

1.5.3 Αποτελέσματα simulation του D-FF



Σχήμα 2: Αποτελέσματα simulation του D-FF

1.5.4 Αρχείο b_FSM.v

```
0  module b_FSM (output reg y_out, input x_in, clk, reset);
1
2      reg[2:0] nextState;
3      wire[2:0] currentState;
4
5      reg myclk, enable;
6      assign myclk = clk || enable;
7
8      D_FF myDFF [2:0](.q(currentState), .d(nextState), .clk(myclk));
9
10     parameter resetState = 3'b001;      // State B
```

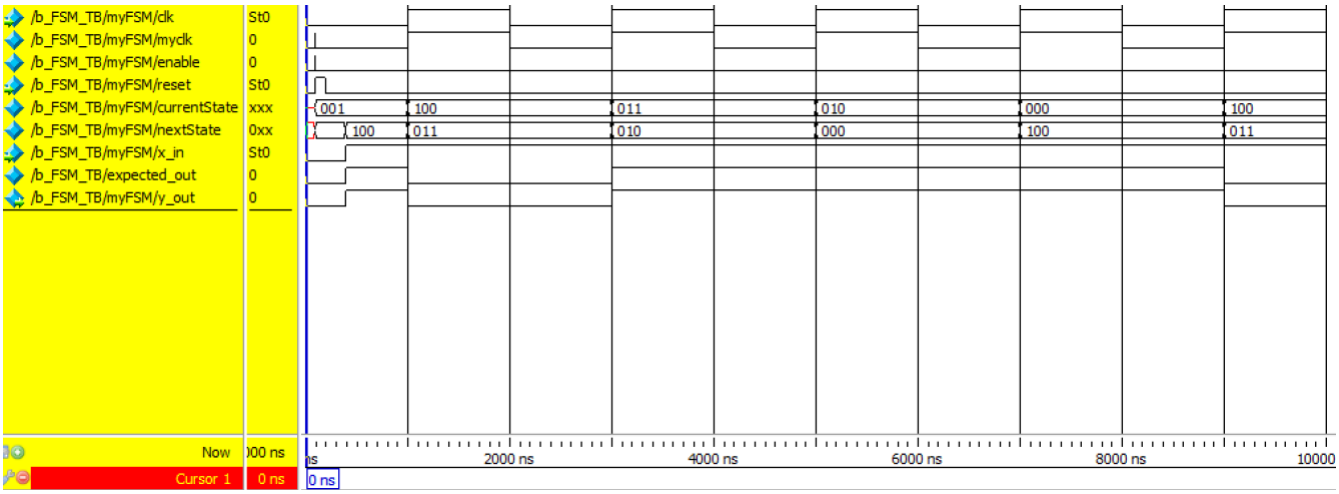
```
11
12      initial
13      begin
14          nextState = resetState;
15          enable = 1'b0;
16      end
17
18      assign nextState[2] = (reset) ? 0 : (~currentState[2] & ~currentState[1] & x_in);
19      assign nextState[1] = (reset) ? 0 : ((~currentState[0] & ~x_in) | (currentState[1] & currentState[0] & x_in) |
    (currentState[2] & ~currentState[0]));
20      assign nextState[0] = (reset) ? 1 : ((~currentState[2] & ~currentState[1] & ~x_in) | (currentState[0] & ~x_in) |
    (currentState[2] & x_in));
21      assign y_out = ~currentState[2] & x_in;
22
23      always@(posedge reset)
24      begin
25          nextState = resetState;
26          enable = 1;
27      end
28
29      always@(posedge enable)
30      begin
31          enable = 0;
32      end
33
34  endmodule
```

1.5.5 Αρχείο b_FSM_TB.v

```
0   `timescale 100ns/100ns      // For 1MHZ clock in #10
1   module b_FSM_TB;
2
3       reg clk, reset, x_in, expected_out;
4       wire y_out;
5
6       b_FSM myFSM(y_out, x_in, clk, reset);
7
8       initial
9       begin
10          clk = 1'b0;
11          reset = 1'b0;
12          x_in = 1'b0;
13          expected_out = 1'b0;
14          #1 reset = 1'b1;    // We need posedge of Reset to reset FSM
15          #1 reset = 1'b0;
16          // We are on State B(001)
17          // with x_in = 0 we have y_out = 0 and nextState = B(001)
18          #2 x_in = 1'b1;
19          // with x_in = 1 we have y_out = 1 and nextState = E(100)
20          expected_out = 1'b1;
21          // at next posedge clock we have currentState = E(100)
22          // and with x_in = 1 we have y_out = 0 and nextState = D(011)
23          #6 expected_out = 1'b0;
24
25          // at next posedge clock we have currentState = D(011)
26          // and with x_in = 1 we have y_out = 1 and nextState = C(010)
27          #20 expected_out = 1'b1;
28
29          // at next posedge clock we have currentState = C(010)
30          // and with x_in = 1 we have y_out = 1 and nextState = A(000)
31
32          // at next posedge clock we have currentState = A(000)
33          // and with x_in = 1 we have y_out = 1 and nextState = E(100)
34
35          // at next posedge clock we have currentState = E(100)
36          // and with x_in = 1 we have y_out = 0 and nextState = D(011)
37          #60 expected_out = 1'b0;
38
39      end
40
41      always
42      begin
43          #10 clk = ~clk;
44      end
45
46  endmodule
```

1.5.6 Αποτελέσματα του δεύτερου simulation του FSM simulator

Παρατηρούμε πως τα αποτελέσματα είναι σωστά ελέγχοντας πρώτα τις καταστάσεις από τους πίνακες που δημιουργήθηκαν παραπάνω και ελέγχοντας ότι το expected_out είναι ίδιο με το y_out.



Σχήμα 3: Αποτελέσματα του δεύτερου simulation του FSM simulator

1.6 Τρίτο ερώτημα

Κώδικας Verilog για το ερώτημα (γ):

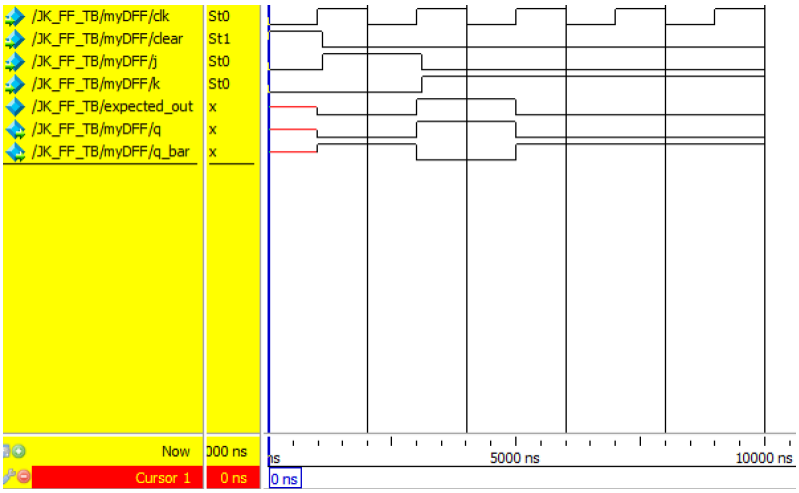
1.6.1 Αρχείο JK_FF.v

```
0  module JK_FF (output reg q, q_bar, input clear, clk, j, k);
1
2      always@(posedge clk)
3      begin
4          if(clear == 1)
5          begin
6              q <= 0;
7              q_bar <= 1;
8          end
9          if(j && k)
10         begin
11             q <= q;
12             q_bar <= q_bar;
13         end
14         else if(j)
15         begin
16             q <= 1;
17             q_bar <= 0;
18         end
19         else if(k)
20         begin
21             q <= 0;
22             q_bar <= 1;
23         end
24     end
25
26 endmodule
```

1.6.2 Αρχείο JK_FF_TB.v

```
0  `timescale 100ns/100ns      // For 1MHZ clock in #10
1  module JK_FF_TB;
2
3      reg j, k, clk, clear, expected_out;
4      wire q, q_bar;
5
6      JK_FF myDFF(q, q_bar, clear, clk, j, k);
7
8      initial
9      begin
10         clk = 1'b0;
11         clear = 1'b1;       // Set FF to q = 0, q_bar = 1 at first posedge clock
12         j = 1'b0;
13         k = 1'b0;
14         #10 expected_out = 1'b0;
15         #1 clear = 1'b0;
16         j = 1'b1;        // Set
17         #19 expected_out = 1'b1;// At second posedge clock q = 1, q_bar = 0
18         #1 j = 1'b0;
19         k = 1'b1;        // Reset
20         #19 expected_out = 1'b0;// At third posedge clock q = 0, q_bar = 1
21     end
22
23     always
24     begin
25         #10 clk = ~clk;
26     end
27 endmodule
```

1.6.3 Αποτελέσματα simulation του JK-FF



Σχήμα 4: Αποτελέσματα simulation του JK-FF

1.6.4 Αρχείο c_FSM.v

```
module c_FSM (output reg y_out, input x_in, clk, reset);

    reg[2:0] nextState;
    wire[2:0] currentState;

    reg myclk, enable;
    assign myclk = clk || enable;

    JK_FF myJKFF [2:0](.q(currentState), .j(nextState), .k(~nextState), .clk(myclk));

    parameter resetState = 3'b001;      // State B

    initial
    begin
        nextState = resetState;
        enable = 1'b0;
    end

    assign nextState[2] = (reset) ? 0 : (~currentState[2] & ~currentState[1] & x_in);
    assign nextState[1] = (reset) ? 0 : ((~currentState[0] & ~x_in) | (currentState[1] & currentState[0] & x_in) | (currentState[2] & ~currentState[0]));
    assign nextState[0] = (reset) ? 1 : ((~currentState[2] & ~currentState[1] & ~x_in) | (currentState[0] & ~x_in) | (currentState[2] & x_in));
    assign y_out = ~currentState[2] & x_in;

    always@(posedge reset)
    begin
        nextState = resetState;
        enable = 1;
    end

    always@(posedge enable)
    begin
        enable = 0;
    end

endmodule
```

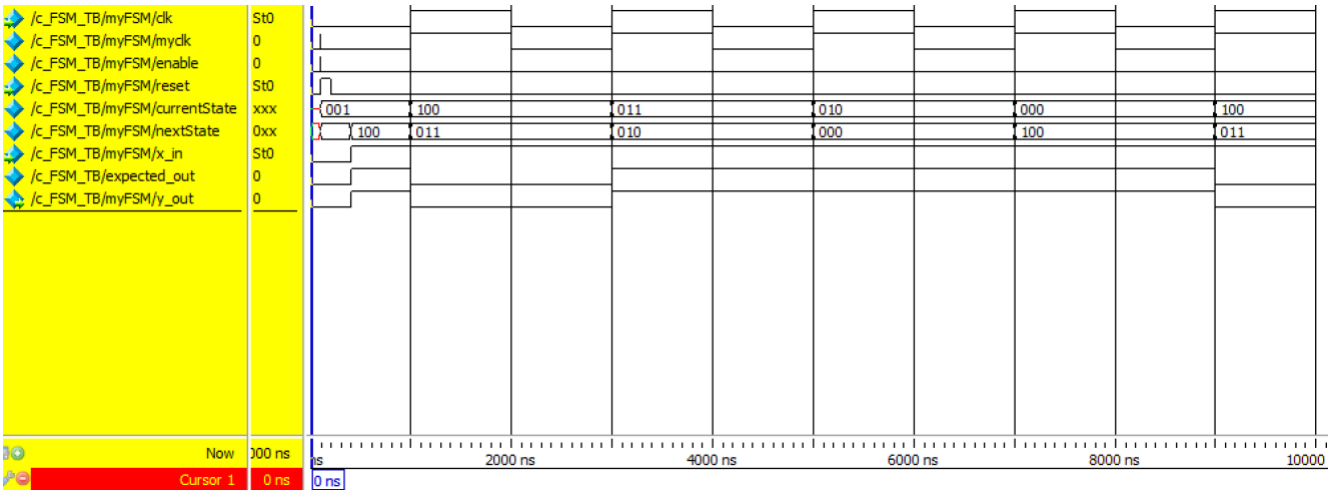
1.6.5 Αρχείο c_FSM_TB.v

```
`timescale 100ns/100ns      // For 1MHZ clock in #10
module c_FSM_TB;

    reg clk, reset, x_in, expected_out;
    wire y_out;

    c_FSM myFSM(y_out, x_in, clk, reset);

    initial
    begin
        clk = 1'b0;
        reset = 1'b0;
        x_in = 1'b0;
        expected_out = 1'b0;
        #1 reset = 1'b1;    // We need posedge of Reset to reset FSM
        #1 reset = 1'b0;
        // We are on State B(001)
        // with x_in = 0 we have y_out = 0 and nextState = B(001)
        #2 x_in = 1'b1;
        // with x_in = 1 we have y_out = 1 and nextState = E(100)
        expected_out = 1'b1;
        // at next posedge clock we have currentState = E(100)
        // and with x_in = 1 we have y_out = 0 and nextState = D(011)
        #6 expected_out = 1'b0;

        // at next posedge clock we have currentState = D(011)
        // and with x_in = 1 we have y_out = 1 and nextState = C(010)
        #20 expected_out = 1'b1;

        // at next posedge clock we have currentState = C(010)
        // and with x_in = 1 we have y_out = 1 and nextState = A(000)

        // at next posedge clock we have currentState = A(000)
```

```
33          // and with x_in = 1 we have y_out = 1 and nextState = E(100)
34
35          // at next posedge clock we have currentState = E(100)
36          // and with x_in = 1 we have y_out = 0 and nextState = D(011)
37          #60 expected_out = 1'b0;
38
39
40      end
41
42      always
43      begin
44          #10 clk = ~clk;
45      end
46
47  endmodule
```

1.6.6 Αποτελέσματα του τρίτου simulation του FSM simulator

Παρατηρούμε πως τα αποτελέσματα είναι σωστά ελέγχοντας πρώτα τις καταστάσεις από τους πίνακες που δημιουργήθηκαν παραπάνω και ελέγχοντας ότι το expected_out είναι ίδιο με το y_out.



Σχήμα 5: Αποτελέσματα του τρίτου simulation του FSM simulator

2 Άσκηση 2

2.1 T - Flip Flop

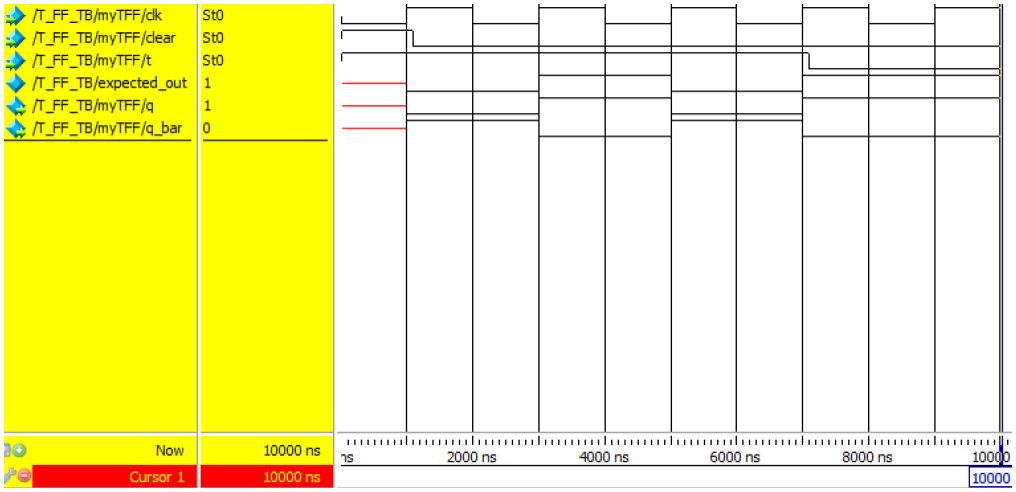
2.1.1 Αρχείο T_FF.v

```
0  module T_FF (output reg q, q_bar, input clk, clear, t);
1
2      always@(posedge clk)
3      begin
4          if(clear)
5          begin
6              q <= 0;
7              q_bar <= 1;
8          end
9          else
10         begin
11             if(t)
12             begin
13                 q <= ~q;
14                 q_bar <= ~q_bar;
15             end
16             else
17             begin
18                 q <= q;
19                 q_bar <= q_bar;
20             end
21         end
22     end
23 endmodule
```

2.1.2 Αρχείο T_FF_TB.v

```
0  `timescale 100ns/100ns      // For 1MHZ clock in #10
1  module T_FF_TB;
2
3      reg t, clk, clear, expected_out;
4      wire q, q_bar;
5
6      T_FF myTFF(q, q_bar, clk, clear, t);
7
8      initial
9      begin
10         clk = 1'b0;
11         clear = 1'b1;       // Set FF to q = 0, q_bar = 1 at first posedge clock
12     t = 1'b1;        // At second posedge clock q = ~q = 1, q_bar = 0
13     #10 expected_out = 1'b0;
14     #1 clear = 1'b0;
15     #19 expected_out = 1'b1;
16     #20 expected_out = 1'b0;// At third posedge clock q = ~q = 0, q_bar = 1
17     #20 expected_out = 1'b1;// At third posedge clock q = ~q = 1, q_bar = 0
18     #1 t = 1'b0;
19     // After we have no change on q and q_bar
20
21     end
22
23     always
24     begin
25         #10 clk = ~clk;
26     end
27
28 endmodule
```

2.1.3 Αποτελέσματα simulation του T-FF

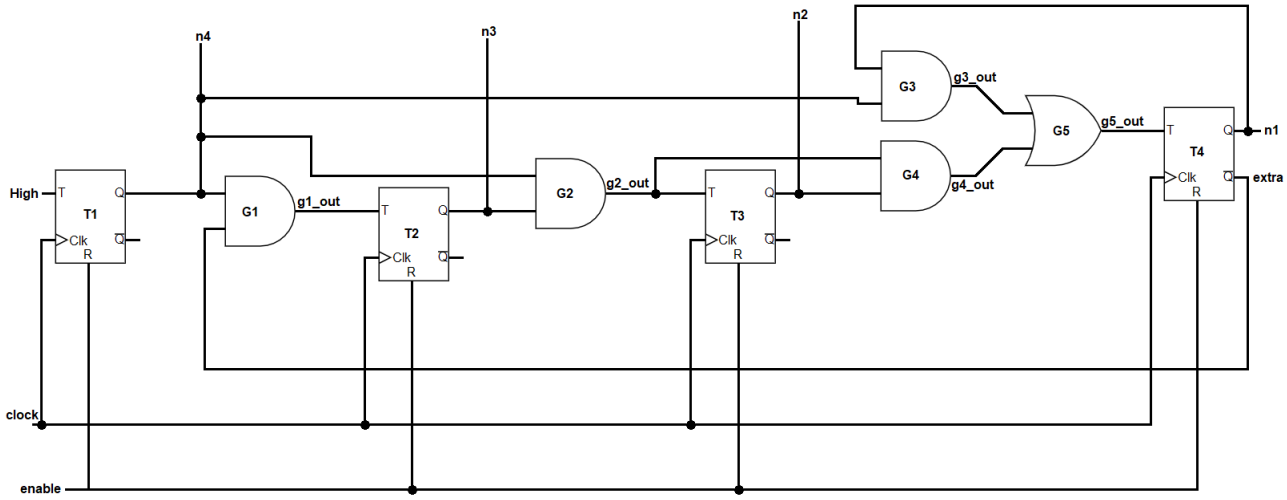


Σχήμα 6: Αποτελέσματα simulation του T -FF

2.2 Απαριθμητής BCD

2.2.1 Κύκλωμα BCD απαριθμητή 4-bit

Παρακάτω παραθέτω το κύκλωμα που υλοποιήθηκε στο αρχείο BCD_Counter.v



Σχήμα 7: Κύκλωμα BCD απαριθμητή 4-bit

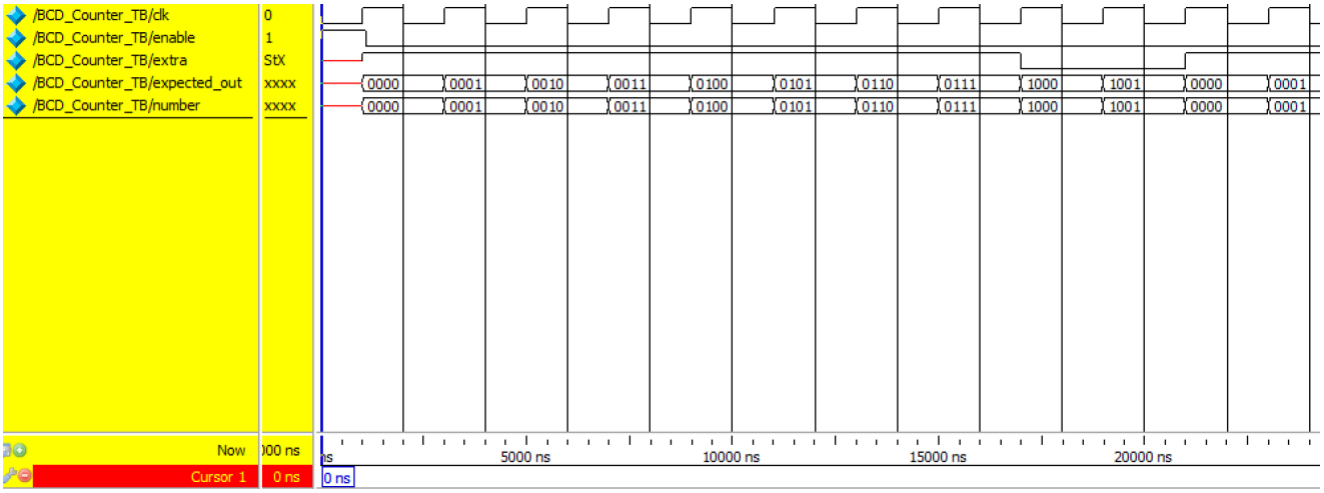
2.2.2 Αρχείο BCD_Counter.v

```
module BCD_Counter (output wire[3:0] number, wire extra, input clk, enable);

    wire n1, n2, n3, n4;
    assign {number[3:0]} = {n1,n2,n3,n4};

    and G1 (g_out1, n4, extra);
    and G2 (g_out2, n3, n4);
    and G3 (g_out3, n1, n4);
    and G4 (g_out4, n2, g_out2);

    or  G5 (g_out5, g_out3, g_out4);

    T_FF T1(.q(n4), .clk(clk), .clear(enable), .t(1'b1));
    T_FF T2(.q(n3), .clk(clk), .clear(enable), .t(g_out1));
    T_FF T3(.q(n2), .clk(clk), .clear(enable), .t(g_out2));
    T_FF T4(.q(n1), .q_bar(extra), .clk(clk), .clear(enable), .t(g_out5));

endmodule
```

2.2.3 Αρχείο BCD_Counter_TB.v

```
`timescale 100ns/100ns      // For 1MHZ clock in #10
module BCD_Counter_TB;

    wire [3:0] number;
    wire extra;
    reg enable, clk;
    reg[3:0] expected_out;

    BCD_Counter myCount(number, extra, clk, enable);

    initial
    begin
        clk = 1'b0;
        enable = 1'b1;
        #10 expected_out = 4'b0000;
        #1 enable = 1'b0;
        #19 expected_out = 4'b0001;

        #20 expected_out = 4'b0010;

        #20 expected_out = 4'b0011;

        #20 expected_out = 4'b0100;

        #20 expected_out = 4'b0101;

        #20 expected_out = 4'b0110;

        #20 expected_out = 4'b0111;

        #20 expected_out = 4'b1000;

        #20 expected_out = 4'b1001;

        #20 expected_out = 4'b0000;

        #20 expected_out = 4'b0001;

        #20 expected_out = 4'b0010;

    end

    always
    begin
        #10 clk = ~clk;
    end

endmodule
```

2.2.4 Αποτελέσματα simulation του BCD 4-bit απαριθμητή



Σχήμα 8: Αποτελέσματα simulation του BCD απαριθμητή

2.3 Αποκωδικοποιητής BCD 4-bit σε LED επτά τμημάτων

2.3.1 Πίνακας Αληθείας

Είσοδος αποκωδικοποιητή				Έξοδος αποκωδικοποιητή						
n_1	n_2	n_3	n_4	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Μέσω των πινάκων karnaugh καταλήγω στις παρακάτω εξισώσεις:

$A = (\overline{n_2} \cdot \overline{n_4}) + n_3 + (n_2 \cdot n_4) + n_1$

$B = (\overline{n_1} \cdot \overline{n_2}) + (\overline{n_3} \cdot \overline{n_4}) + (\overline{n_1} \cdot n_3 \cdot n_4) + (\overline{n_2} \cdot \overline{n_3})$

$C = \overline{n_3} + n_4 + n_2$

$D = (\overline{n_1} \cdot \overline{n_2} \cdot n_3) + (n_2 \cdot \overline{n_3} \cdot n_4) + (n_1 \cdot \overline{n_3}) + (\overline{n_1} \cdot \overline{n_2} \cdot \overline{n_4}) + (\overline{n_1} \cdot n_3 \cdot \overline{n_4})$

$E = (\overline{n_2} \cdot \overline{n_4}) + (n_3 \cdot \overline{n_4})$

$F = (\overline{n_3} \cdot \overline{n_4}) + (n_2 \cdot \overline{n_3}) + (n_2 \cdot \overline{n_4}) + n_1$

$G = (\overline{n_1} \cdot \overline{n_2} \cdot n_3) + (\overline{n_1} \cdot n_2 \cdot \overline{n_3}) + (n_1 \cdot \overline{n_2} \cdot \overline{n_3}) + (\overline{n_1} \cdot n_3 \cdot \overline{n_4})$

2.3.2 Αρχείο BCD_TO_LED.v

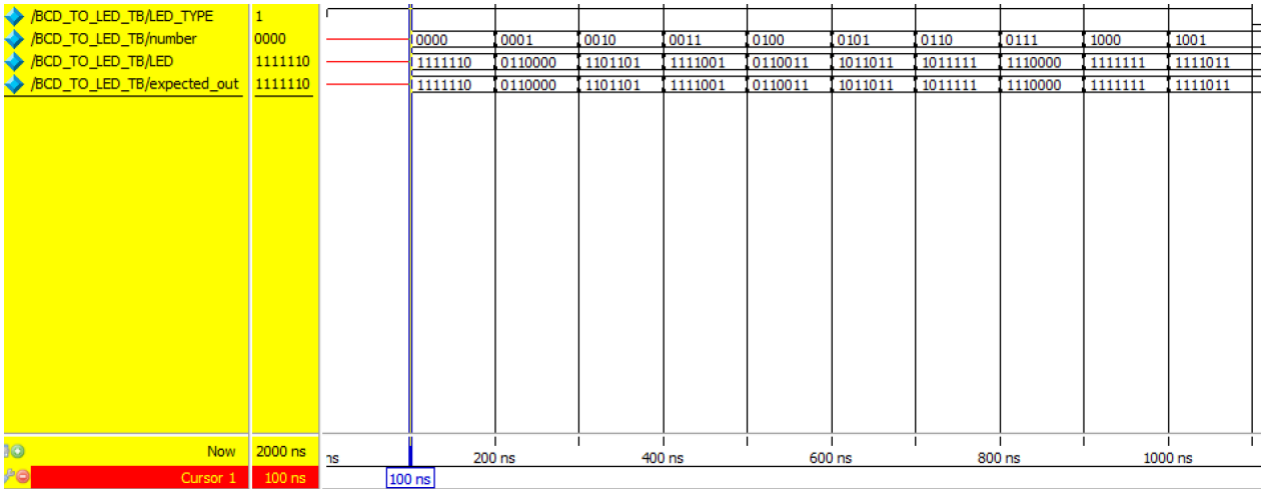
```
module BCD_TO_LED (output wire[6:0] LED, input wire[3:0] number, LED_TYPE);

    wire n1, n2, n3, n4;
    reg A, B, C, D, E, F, G;
    wire a, b, c, d, e, f, g;

    assign{LED[6:0]} = {A, B, C, D, E, F, G};
    assign{n1, n2, n3, n4} = {number[3:0]};

    assign a = (~n2 & ~n4) | (n3) | (n2 & n4) | (n1);
    assign b = (~n1 & ~n2) | (~n3 & ~n4) | (~n1 & n3 & n4) | (~n2 & ~n3);
    assign c = (~n3) | (n4) | (n2);
    assign d = (~n1 & ~n2 & n3) | (n2 & ~n3 & n4) | (n1 & ~n3) | (~n1 & ~n2 & ~n4) | (~n1 & n3 & ~n4);
    assign e = (~n2 & ~n4) | (n3 & ~n4);
    assign f = (~n3 & ~n4) | (n2 & ~n3) | (n2 & ~n4) | (n1);
    assign g = (~n1 & ~n2 & n3) | (~n1 & n2 & ~n3) | (n1 & ~n2 & ~n3) | (~n1 & n3 & ~n4);

    // LED_TYPE = 0 for common anode
    // LED_TYPE = 1 for common cathode
    always@(number)
    begin
        if (LED_TYPE)
        begin
            A = a;
            B = b;
            C = c;
            D = d;
            E = e;
            F = f;
            G = g;
        end
        else
        begin
            A = ~a;
            B = ~b;
            C = ~c;
            D = ~d;
            E = ~e;
            F = ~f;
            G = ~g;
        end
    end

endmodule
```

2.3.3 Αρχείο BCD_TO_LED_TB.v

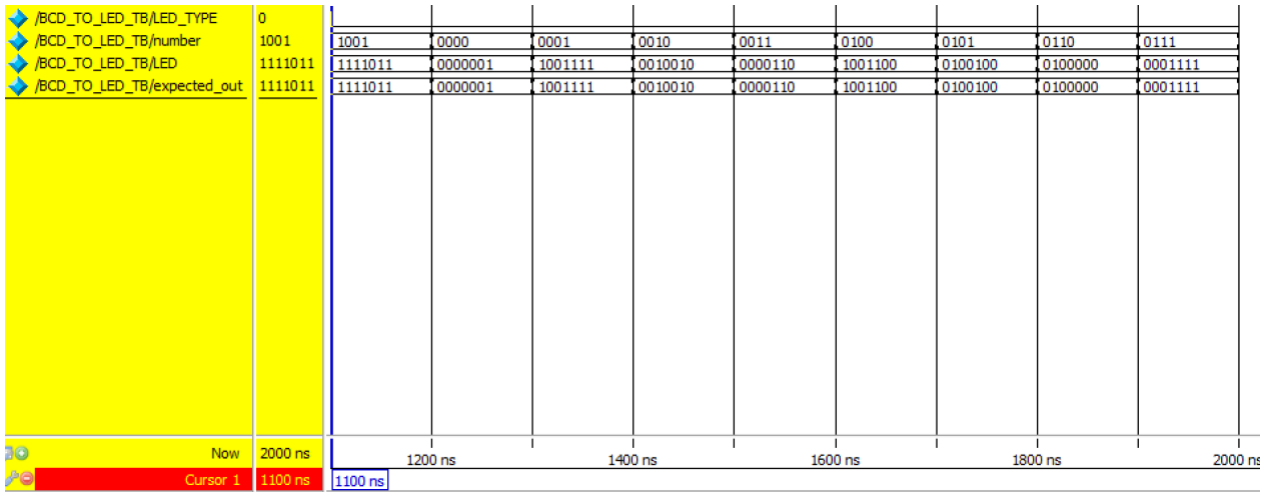
```
`timescale 100ns/100ns
module BCD_TO_LED_TB;

    wire[6:0] LED;
    reg[3:0] number;
    reg LED_TYPE;

    reg[6:0] expected_out;

    BCD_TO_LED myBCD_TO_LED(LED, number, LED_TYPE);


    initial
    begin
            // Common  node
            LED_TYPE = 1'b1;

            #1 number = 4'b_0000;   // Number 0
            expected_out = 7'b_1111110;

            #1 number = 4'b_0001;   // Number 1
            expected_out = 7'b_0110000;

            #1 number = 4'b_0010;   // Number 2
            expected_out = 7'b_1101101;

```

```
            #1 number = 4'b_0011;   //Number 3
            expected_out = 7'b_1111001;

            #1 number = 4'b_0100;   // Number 4
            expected_out = 7'b_0110011;

            #1 number = 4'b_0101;   // Number 5
            expected_out = 7'b_1011011;

            #1 number = 4'b_0110;   // Number 6
            expected_out = 7'b_1011111;

            #1 number = 4'b_0111;   // Number 7
            expected_out = 7'b_1110000;

            #1 number = 4'b_1000;   // Number 8
            expected_out = 7'b_1111111;

            #1 number = 4'b_1001;   // Number 9
            expected_out = 7'b_1111011;


            // Common cathode
            #1 LED_TYPE = 1'b0;

            #1 number = 4'b_0000;   // Number 0
            expected_out = ~(7'b_1111110);

            #1 number = 4'b_0001;   // Number 1
            expected_out = ~(7'b_0110000);

            #1 number = 4'b_0010;   // Number 2
            expected_out = ~(7'b_1101101);

            #1 number = 4'b_0011;   // Number 3
            expected_out = ~(7'b_1111001);

            #1 number = 4'b_0100;   // Number 4
            expected_out = ~(7'b_0110011);

            #1 number = 4'b_0101;   // Number 5
            expected_out = ~(7'b_1011011);

            #1 number = 4'b_0110;   // Number 6
            expected_out = ~(7'b_1011111);

            #1 number = 4'b_0111;   // Number 7
            expected_out = ~(7'b_1110000);

            #1 number = 4'b_1000;   // Number 8
            expected_out = ~(7'b_1111111);

            #1 number = 4'b_1001;   // Number 9
            expected_out = ~(7'b_1111011);

    end
endmodule
```

2.3.4 Αποτελέσματα simulation του BCD 4-bitσε LED επτά τμημάτων



Σχήμα 9: Αποτελέσματα κοινής ανόδου (με LED_TYPE = 1)



Σχήμα 10: Αποτελέσματα κοινής καθόδου (με LED_TYPE = 0)

2.4 Πλήρης απαριθμητής 4 by 4-bit

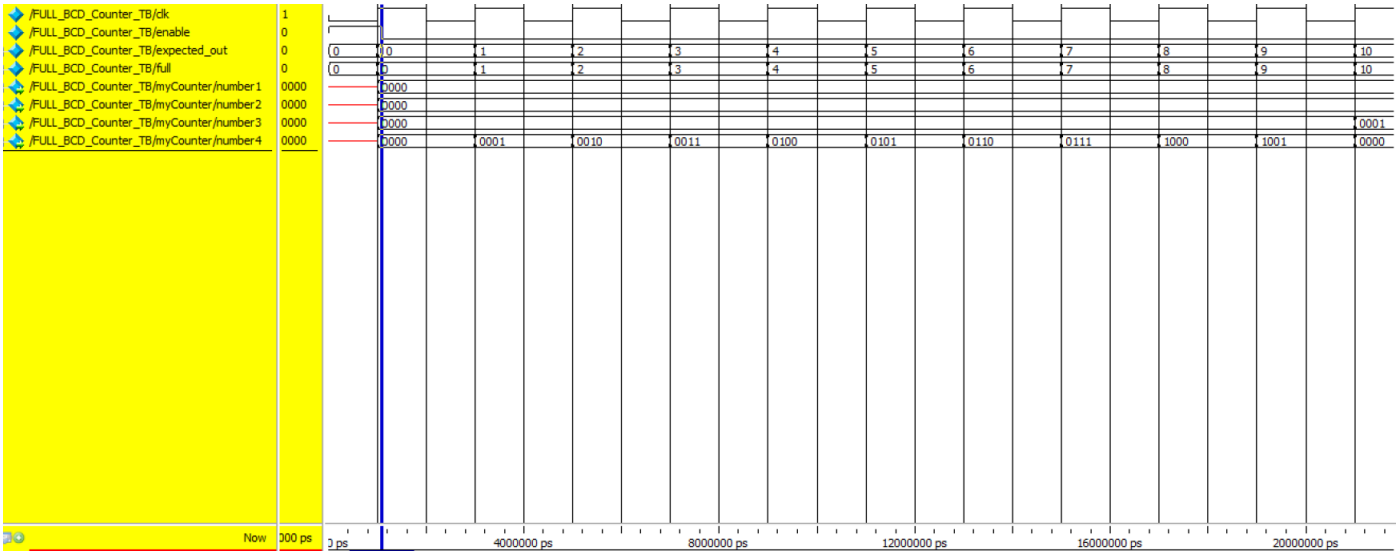
2.4.1 Αρχείο FULL_BCD_Counter.v

```
0  module FULL_BCD_Counter (output wire[3:0] number1, number2, number3, number4, input clk, enable);
1
2      wire[3:0] extra;
3
4      BCD_Counter C1(number1, extra[3], extra[2], enable);
5      BCD_Counter C2(number2, extra[2], extra[1], enable);
6      BCD_Counter C3(number3, extra[1], extra[0], enable);
7      BCD_Counter C4(number4, extra[0], clk, enable);
8
9  endmodule
```

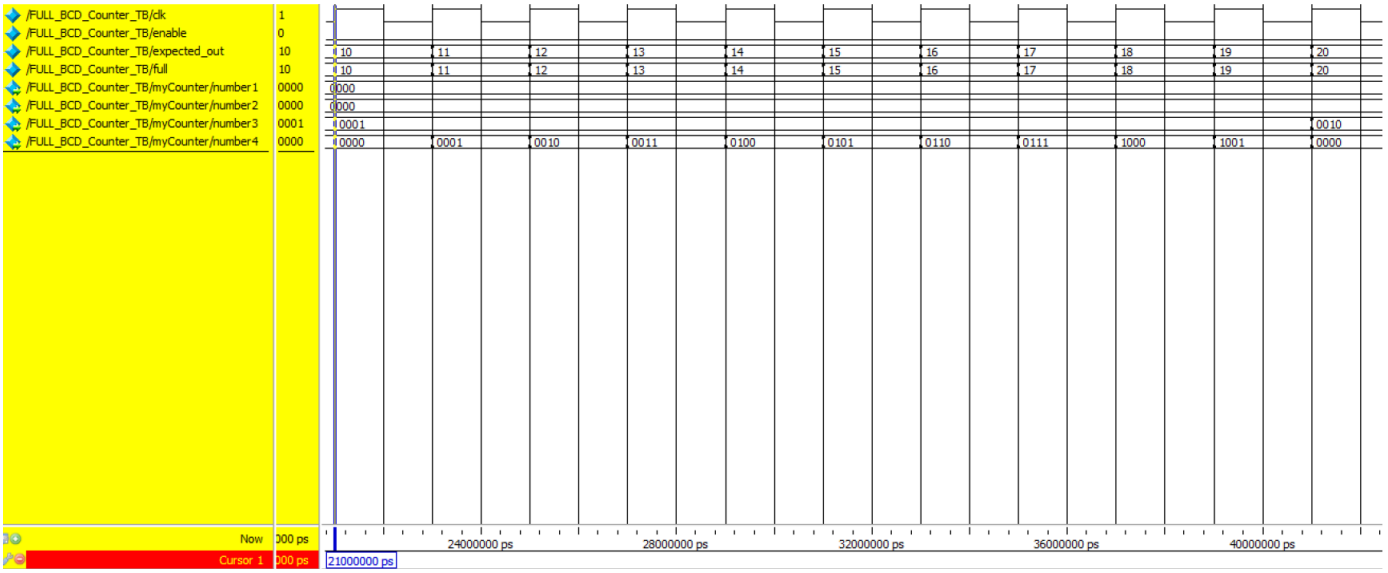
2.4.2 Αρχείο FULL_BCD_Counter_TB.v

```
0  `timescale 100ns/100ns      // For 1MHZ clock in #10
1  module FULL_BCD_Counter_TB;
2
3      reg clk, enable;
4
5      wire[3:0] number[3:0];
6      FULL_BCD_Counter myCounter(number[0], number[1], number[2], number[3], clk, enable);
7
8      integer full, n1, n2, n3, n4;
9      integer expected_out;
10
11     assign n1 = {number[0][3:0]};
12     assign n2 = {number[1][3:0]};
13     assign n3 = {number[2][3:0]};
14     assign n4 = {number[3][3:0]};
15     assign full = n4 + 10*n3 + 100*n2 + 1000*n1;
16
17     initial
18     begin
19         expected_out = 0;
20         full = 0;
21         enable = 1'b1;
22         clk  = 1'b0;
23
24         #11 enable = 1'b0;
25         expected_out = 0;
26
27     end
28
29     always
30     begin
31         #10 clk = ~clk;
32
33         if (clk)
34         begin
35             expected_out = expected_out + 1;
36         end
37         if(expected_out > 9999)
38         begin
39             expected_out = 0;
40         end
41
42     end
43
44 endmodule
```

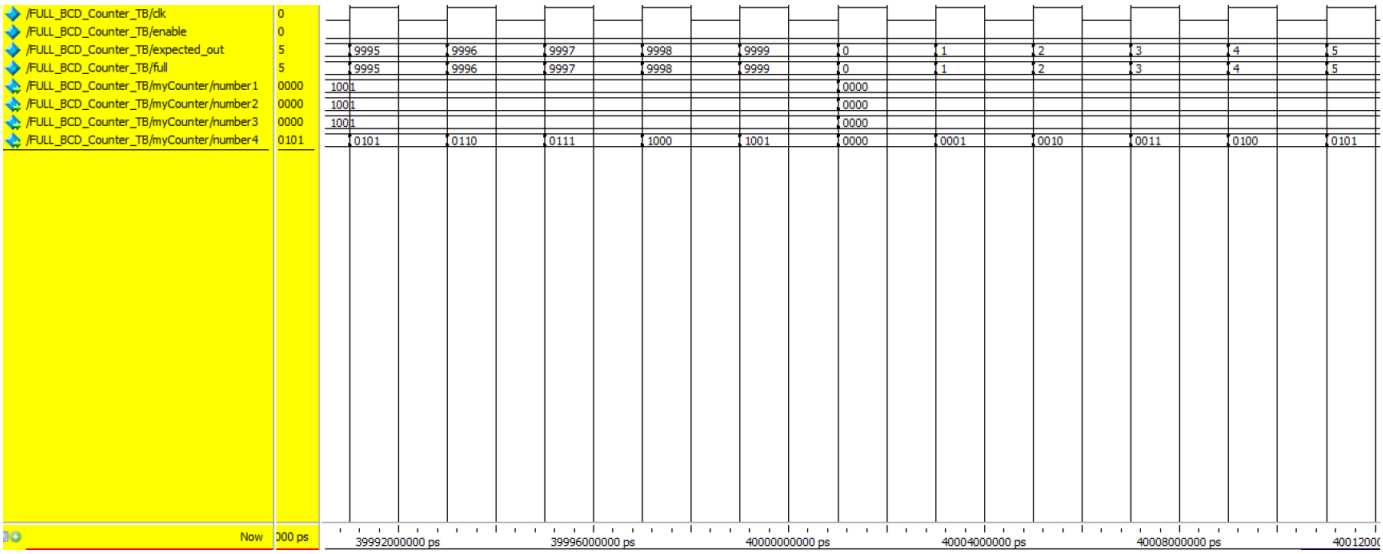
2.4.3 Αποτελέσματα simulation του BCD 4 by 4-bit



Σχήμα 11: Αποτελέσματα simulation του BCD 4 by 4-bit για τους αριθμούς 0 μέχρι 10



Σχήμα 12: Αποτελέσματα simulation του BCD 4 by 4-bit για τους αριθμούς 10 μέχρι 20



Σχήμα 13: Αποτελέσματα simulation του BCD 4 by 4-bit για τους αριθμούς 995 μέχρι 5

2.5 Αποκωδικοποιητής BCD 4 by 4-bit σε LED επτά τμημάτων

2.5.1 Αρχείο FULL_BCD_TO_LED.v

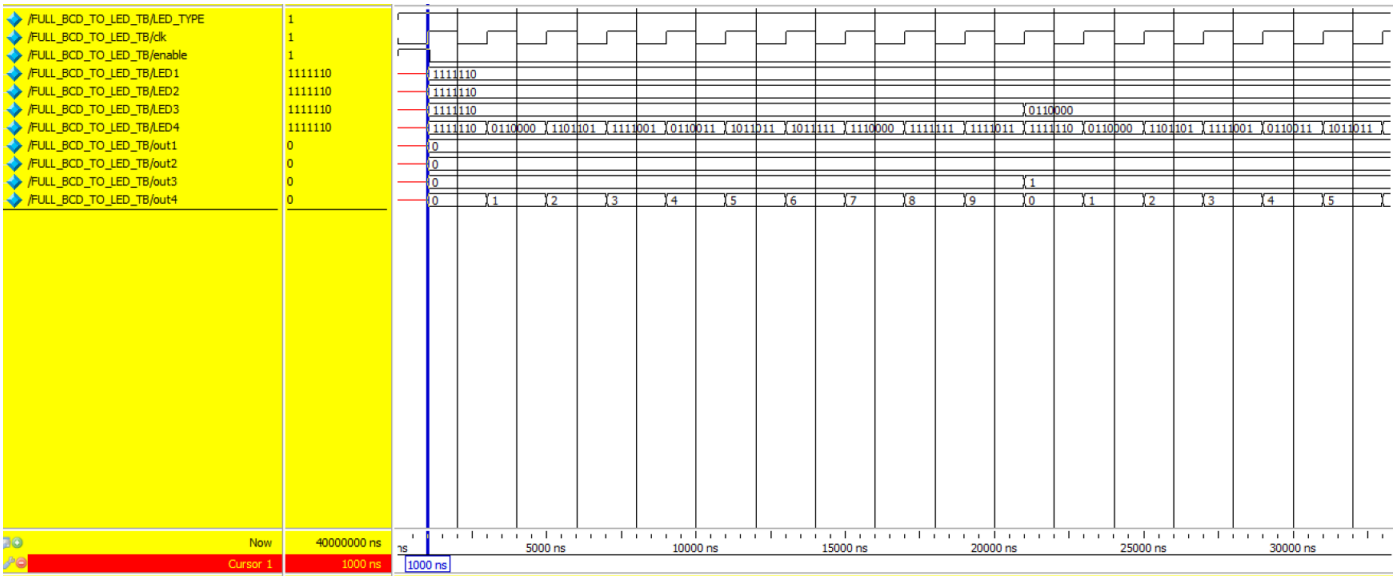
```
0  module FULL_BCD_TO_LED (output wire[6:0] LED1, LED2, LED3, LED4, input wire clk, enable, LED_TYPE);
1
2      wire[3:0] number1, number2, number3, number4;
3
4      FULL_BCD_Counter myCounter(number1, number2, number3, number4, clk, enable);
5
6      BCD_TO_LED myLEDs[3:0]({LED1, LED2, LED3, LED4},{number1, number2, number3, number4},
7                             {LED_TYPE, LED_TYPE, LED_TYPE, LED_TYPE});
8
9  endmodule
```

2.5.2 Αρχείο FULL_BCD_TO_LED_TB.v

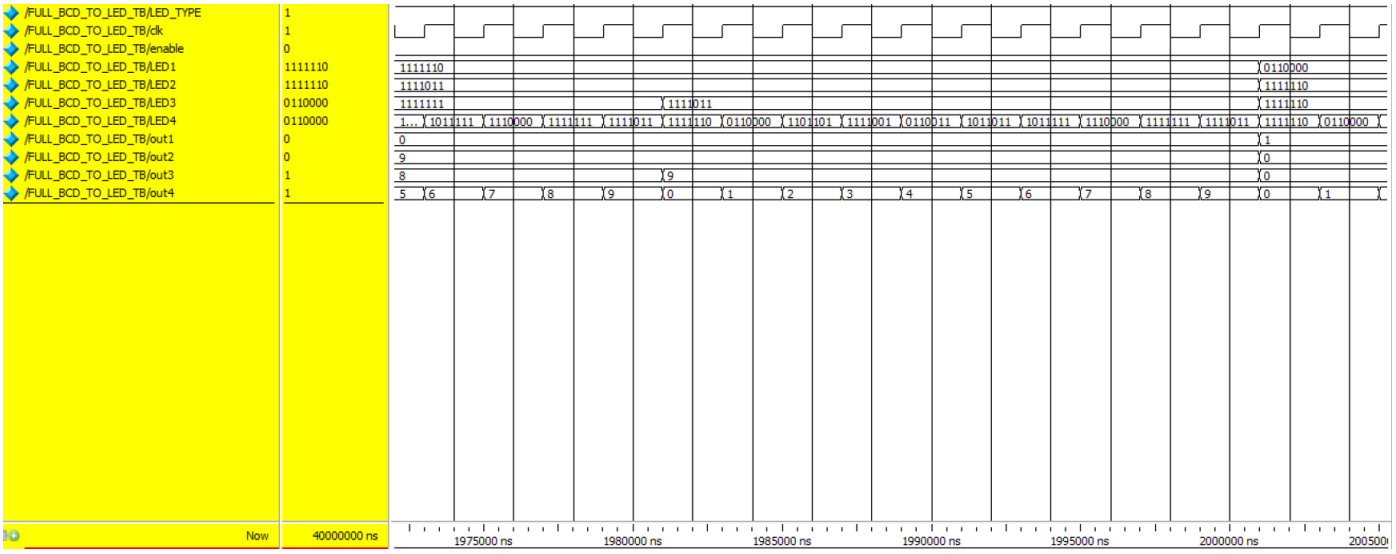
```
0  `timescale 100ns/100ns      // For 1MHZ clock in #10
1  module FULL_BCD_TO_LED_TB;
2
3      wire[6:0]     LED1, LED2, LED3, LED4;
4      reg clk, enable, LED_TYPE;
5
6      FULL_BCD_TO_LED myBCDtoLED(LED1, LED2, LED3, LED4, clk, enable, LED_TYPE);
7
8      reg[6:0] expected_out1, expected_out2, expected_out3, expected_out4;
9      integer out1, out2, out3, out4;
10
11     initial
12     begin
13         LED_TYPE = 1'b1;
14         clk = 1'b0;
15         enable = 1'b1;
16         #11 enable = 1'b0;
17     end
18
19     always @(LED1 or LED2 or LED3 or LED4)
20     begin
21         expected_out1 = (LED_TYPE) ? LED1 : ~LED1;
22             case(expected_out1)
23                     7'b_1111110:    out1 = 0;
24                     7'b_0110000:    out1 = 1;
25                     7'b_1101101:    out1 = 2;
26                     7'b_1111001:    out1 = 3;
```

```
                    7'b_0110011:    out1 = 4;
                    7'b_1011011:    out1 = 5;
                    7'b_1011111:    out1 = 6;
                    7'b_1110000:    out1 = 7;
                    7'b_1111111:    out1 = 8;
                    7'b_1111011:    out1 = 9;
                endcase
        expected_out2 = (LED_TYPE) ? LED2 : ~LED2;
            case(expected_out2)
                    7'b_1111110:    out2 = 0;
                    7'b_0110000:    out2 = 1;
                    7'b_1101101:    out2 = 2;
                    7'b_1111001:    out2 = 3;
                    7'b_0110011:    out2 = 4;
                    7'b_1011011:    out2 = 5;
                    7'b_1011111:    out2 = 6;
                    7'b_1110000:    out2 = 7;
                    7'b_1111111:    out2 = 8;
                    7'b_1111011:    out2 = 9;
                endcase
        expected_out3 = (LED_TYPE) ? LED3 : ~LED3;
            case(expected_out3)
                    7'b_1111110:    out3 = 0;
                    7'b_0110000:    out3 = 1;
                    7'b_1101101:    out3 = 2;
                    7'b_1111001:    out3 = 3;
                    7'b_0110011:    out3 = 4;
                    7'b_1011011:    out3 = 5;
                    7'b_1011111:    out3 = 6;
                    7'b_1110000:    out3 = 7;
                    7'b_1111111:    out3 = 8;
                    7'b_1111011:    out3 = 9;
                endcase
        expected_out4 = (LED_TYPE) ? LED4 : ~LED4;
            case(expected_out4)
                    7'b_1111110:    out4 = 0;
                    7'b_0110000:    out4 = 1;
                    7'b_1101101:    out4 = 2;
                    7'b_1111001:    out4 = 3;
                    7'b_0110011:    out4 = 4;
                    7'b_1011011:    out4 = 5;
                    7'b_1011111:    out4 = 6;
                    7'b_1110000:    out4 = 7;
                    7'b_1111111:    out4 = 8;
                    7'b_1111011:    out4 = 9;
                endcase
    end
    always
    begin
        #10 clk = ~clk;
    end

endmodule
```

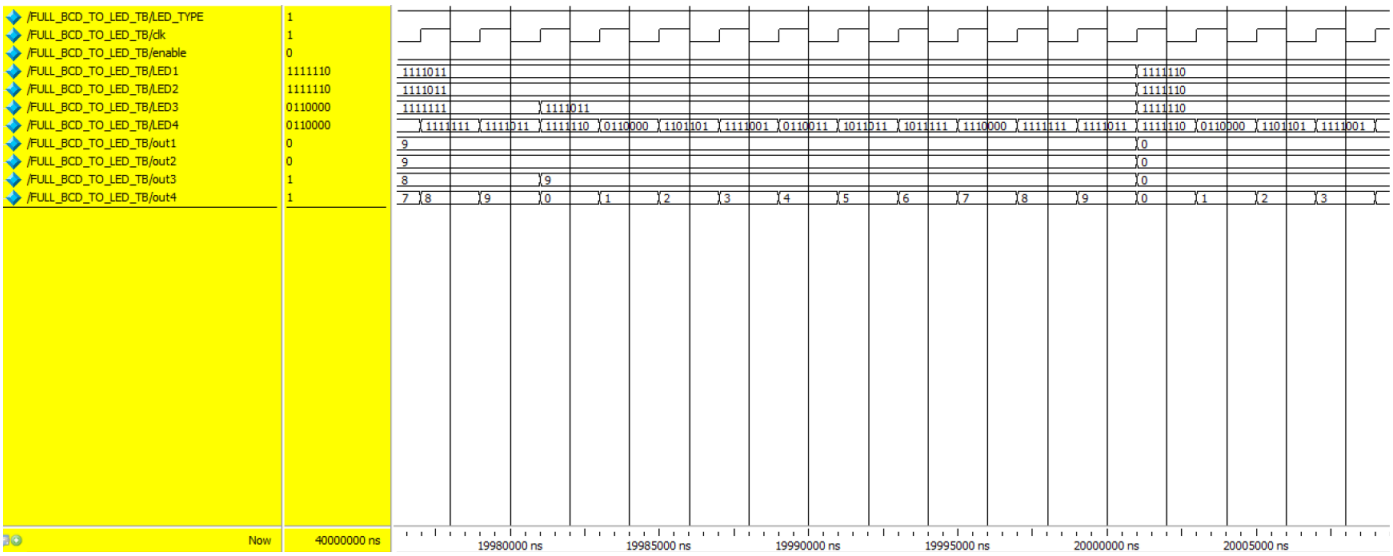
2.5.3 Αποτελέσματα simulation του BCD 4 by 4-bit σε LED κοινής ανόδου



Σχήμα 14: Αποτελέσματα simulation του BCD 4 by 4-bit σε LED κοινής ανόδου για τους αριθμούς 0 μέχρι 15



Σχήμα 15: Αποτελέσματα simulation του BCD 4 by 4-bit σε LED κοινής ανόδου για τους αριθμούς 985 μέχρι 1001



Σχήμα 16: Αποτελέσματα simulation του BCD 4 by 4-bit σε LED κοινής ανόδου για τους αριθμούς 9987 μέχρι 3

3 Άσκηση 3

3.1 Κωδικοποιητής Hamming (12,5)

3.1.1 Αρχείο hamming_encoder.v

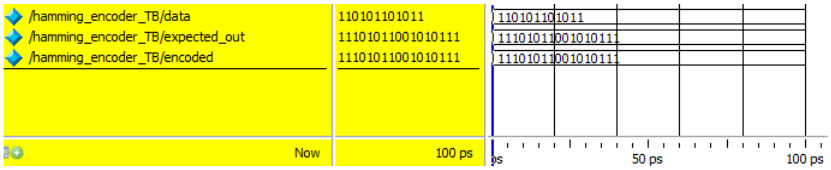
```
0  module hamming_encoder(output reg [17:1] encoded, input wire [11:0] data);
1
2      reg firstParityBit, secondParityBit, thirdParityBit, fourthParityBit, fifthParityBit;
3
4
5      assign firstParityBit = encoded[3] ^ encoded[5] ^ encoded[7] ^ encoded[9] ^ encoded[11] ^ encoded[13] ^ encoded[15] ^ encoded[17];
6      assign secondParityBit = encoded[3] ^ encoded[6] ^ encoded[7] ^ encoded[10] ^ encoded[11] ^ encoded[14] ^ encoded[15];
7      assign thirdParityBit = encoded[5] ^ encoded[6] ^ encoded[7] ^ encoded[12] ^ encoded[13] ^ encoded[14] ^ encoded[15];
8      assign fourthParityBit = encoded[9] ^ encoded[10] ^ encoded[11] ^ encoded[12];
9      assign fifthParityBit = encoded[17];
10
11
12     assign encoded[1] = firstParityBit;
13     assign encoded[2] = secondParityBit;
14     assign encoded[3] = data[0];
15     assign encoded[4] = thirdParityBit;
16     assign encoded[5] = data[1];
17     assign encoded[6] = data[2];
18     assign encoded[7] = data[3];
19     assign encoded[8] = fourthParityBit;
20     assign encoded[9] = data[4];
21     assign encoded[10] = data[5];
22     assign encoded[11] = data[6];
23     assign encoded[12] = data[7];
24     assign encoded[13] = data[8];
25     assign encoded[14] = data[9];
26     assign encoded[15] = data[10];
27     assign encoded[16] = fifthParityBit;
28     assign encoded[17] = data[11];
29
30 endmodule
```

3.1.2 Αρχείο hamming_encoder_TB.v

```
0  module hamming_encoder_TB;
1
```

```
    wire[17:1] encoded;
    reg[11:0] data;
    reg[17:1] expected_out;

    hamming_encoder myEncoder(encoded, data);

    initial
    begin
        data = 12'b110101101011;
        expected_out = 17'b11101011001010111;
    end

endmodule
```

3.1.3 Αποτελέσματα simulation του hamming_encoder



Σχήμα 17: Αποτελέσματα simulation του hamming_encoder

3.2 Αποκωδικοποιητής Hamming (12,5)

3.2.1 Αρχείο hamming_decoder.v

```
module hamming_decoder(output reg[11:0] decoded, input wire[16:0] data);

    wire[4:0] errorBits;
    reg[16:0] afterData;

    assign errorBits[0] = data[0] ^ data[2] ^ data[4] ^ data[6] ^ data[8] ^ data[10] ^ data[12] ^ data[14] ^ data[16];
    assign errorBits[1] = data[1] ^ data[2] ^ data[5] ^ data[6] ^ data[9] ^ data[10] ^ data[13] ^ data[14];
    assign errorBits[2] = data[3] ^ data[4] ^ data[5] ^ data[6] ^ data[11] ^ data[12] ^ data[13] ^ data[14];
    assign errorBits[3] = data[7] ^ data[8] ^ data[9] ^ data[10] ^ data[11];
    assign errorBits[4] = data[15] ^ data[16];

    assign decoded = {afterData[16], afterData[14:8], afterData[6:4], afterData[2]};

    always @(data) begin: ERROR_CORRECTION
        afterData = data;
        case (errorBits)
            1:  disable ERROR_CORRECTION;
            2:  disable ERROR_CORRECTION;
            4:  disable ERROR_CORRECTION;
            8:  disable ERROR_CORRECTION;
            16: disable ERROR_CORRECTION;
            default: {afterData[errorBits]} = ~{afterData[errorBits]};
        endcase
    end

endmodule
```

3.2.2 Αρχείο hamming_decoder_TB.v

```
module hamming_decoder_TB;

    wire[12:1] decoded;
    reg[17:1] data;
    reg[12:1] expected_out;

    hamming_decoder myDecoder(decoded, data);
```

```
7
8       initial
9       begin
10          data = 17'b11101011001010111;
11          expected_out = 12'b110101101011;
12      end
13
14  endmodule
```

3.2.3 Αποτελέσματα simulation του hamming_decoder

Σχήμα 18: Αποτελέσματα simulation του hamming_decoder

3.3 Επίδραση Θορύβου

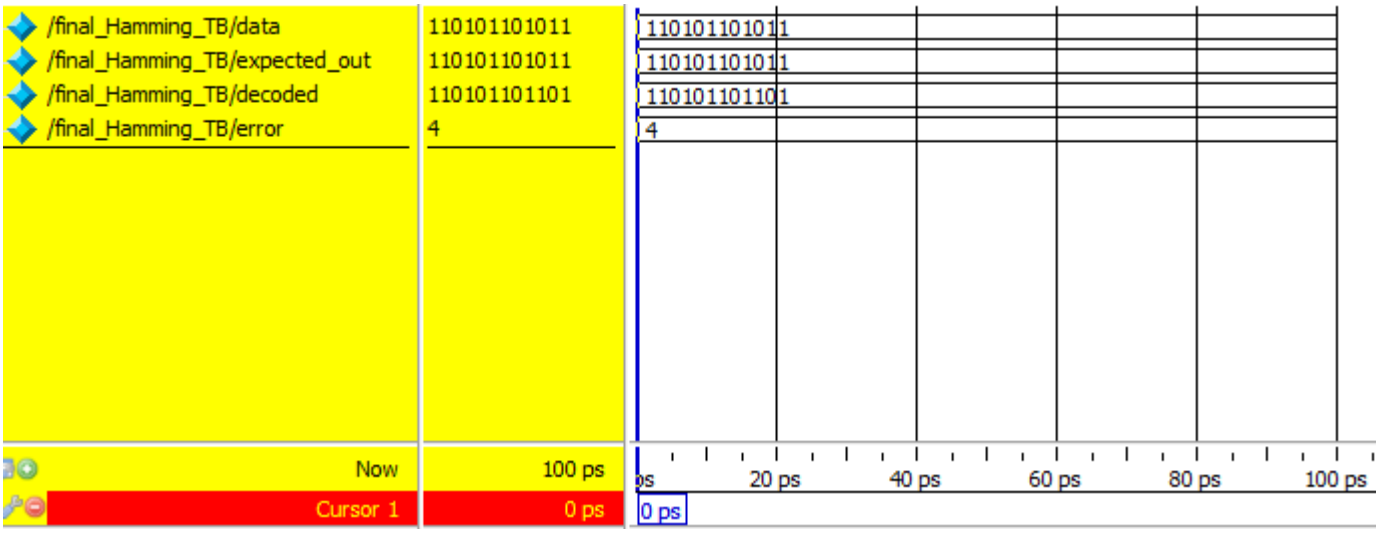
3.3.1 Αρχείο final_Hamming.v

```
0   module final_Hamming(output wire[11:0] decoded, input wire[11:0] data, wire[31:0] error);
1
2       reg[16:0] dataAfterNoise;
3       wire[16:0] encoded;
4
5       hamming_encoder myEncoder(encoded, data);
6
7       hamming_decoder myDecoder(decoded, dataAfterNoise);
8
9
10      always @(data)
11      begin
12          if(error > 11)
13          begin
14                  dataAfterNoise = encoded;
15              end
16          else
17          begin
18                  dataAfterNoise = encoded;
19                  dataAfterNoise[error] = ~dataAfterNoise[error];
20          end
21      end
22
23  endmodule
```

3.3.2 Αρχείο final_Hamming_TB.v

```
0  module final_Hamming_TB;
1
2      wire[11:0] decoded;
3      reg[11:0] data;
4      integer error;
5      reg[11:0] expected_out;
6
7      final_Hamming myFULL(decoded, data, error);
8
9      initial
10     begin
11             data = 12'b110101101011;
12             error = 4;
13             expected_out = 12'b110101101011;
14     end
15
16 endmodule
```

3.3.3 Αποτελέσματα simulation του Hamming



Σχήμα 19: Αποτελέσματα simulation του Hamming