Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής ΙΙ

Εργασία 2020

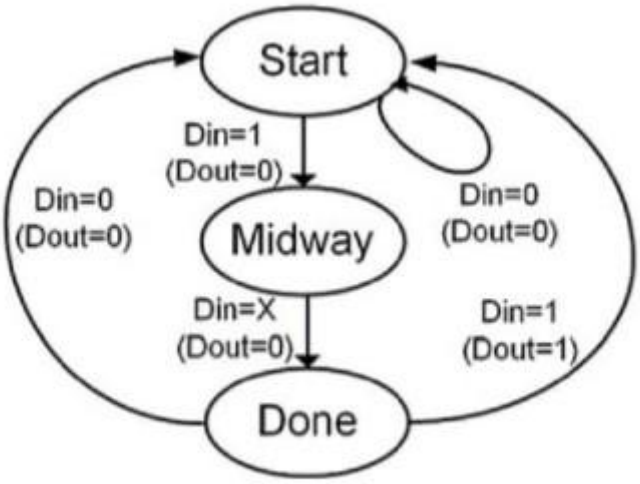
ΝΙΚΗΦΟΡΙΔΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ 9084

nikifori@ece.auth.gr

Θεσσαλονίκη, Ιούνιος 2020

Εργασία 1

Ερώτημα α)



Σχήμα 1. Διάγραμμα μεταβάσεων για την άσκηση 1.

Πίνακας Μεταβάσεων (Κατάστασης)

Present State	**Next State**	**Next State**	**Output**	**Output**
	με X=0	με X=1	με X=0	με X=1
Start	Start	Midway	0	0
Midway	Done	Done	0	0
Done	Start	Start	0	1

Κωδικοποίηση Καταστάσεων

Κατάσταση	**Κωδικοποίηση $D_{1:0}$**
Start	00
Midway	01
Done	10

Λογικές Εξισώσεις Καταστάσεων-Εξόδων

Present State D_1	Present State D_0	Input X	Next State $D_{1'}$	Next State $D_{0'}$	Output Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	0	1

- $D_{1,}' = D_0$
- $D_0' = \not{D}_1 \not{D}_0 X$, με $\not{D}_1$ = D_1_Bar και αντίστοιχα το $\not{D}_0$
- $Y = D_1 X$

Κώδικας Verilog

```
module fsm1_behavioral (output reg Dout,
                        input wire Clock, Reset, Din);

    reg [1:0] current_state, next_state;
    parameter Start = 2'b00,
              Midway = 2'b01,
              Done = 2'b10;

    // Apothikeush Katastashs---------------------------------
    always @ (posedge Clock or negedge Reset)
     begin: STATE_MEMORY
      if (!Reset)
            current_state <= Start;
      else
            current_state <= next_state;
    end

    // Gia Epomenh Katastash-----------------------------------
```

```
        always @ (current_state or Din)
         begin: NEXT_STATE_LOGIC
          case (current_state)
           Start : if (Din==1'b1) next_state = Midway;
                       else next_state = Start;
           Midway : next_state = Done;
           Done : next_state = Start;
           default : next_state = Start;
          endcase
        end

        // Exodos----------------------------------------------------------
        always @ (current_state or Din)
         begin: OUTPUT_LOGIC
          case (current_state)
           Start : Dout = 1'b0;
           Midway : Dout = 1'b0;
           Done : if (Din==1) Dout=1'b1;
                       else Dout = 1'b0;
          endcase
        end

endmodule
```

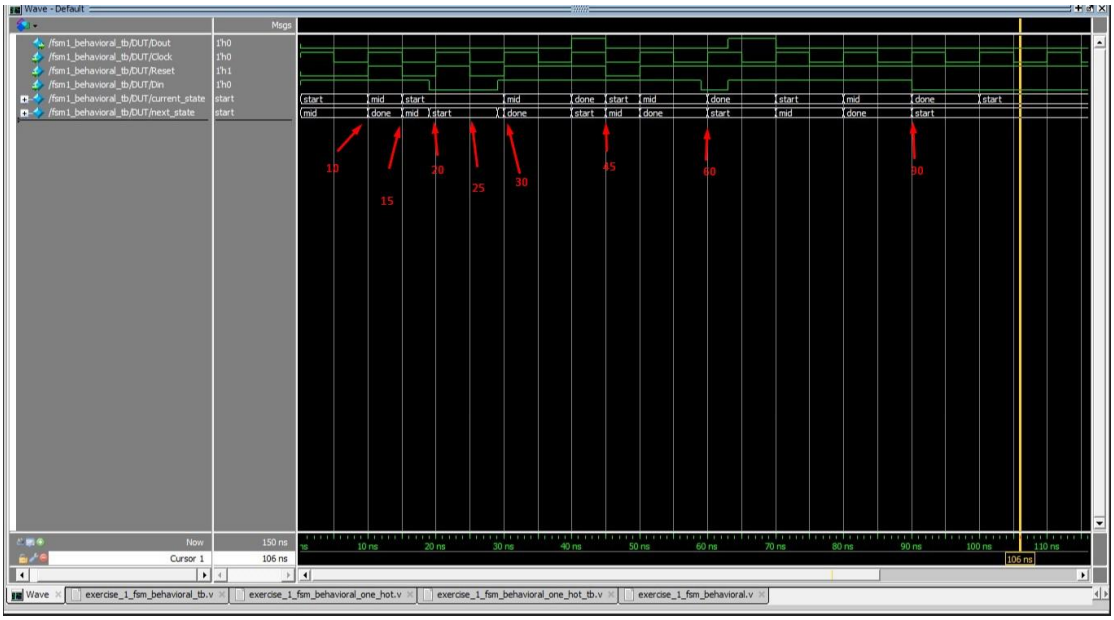
Test Bench

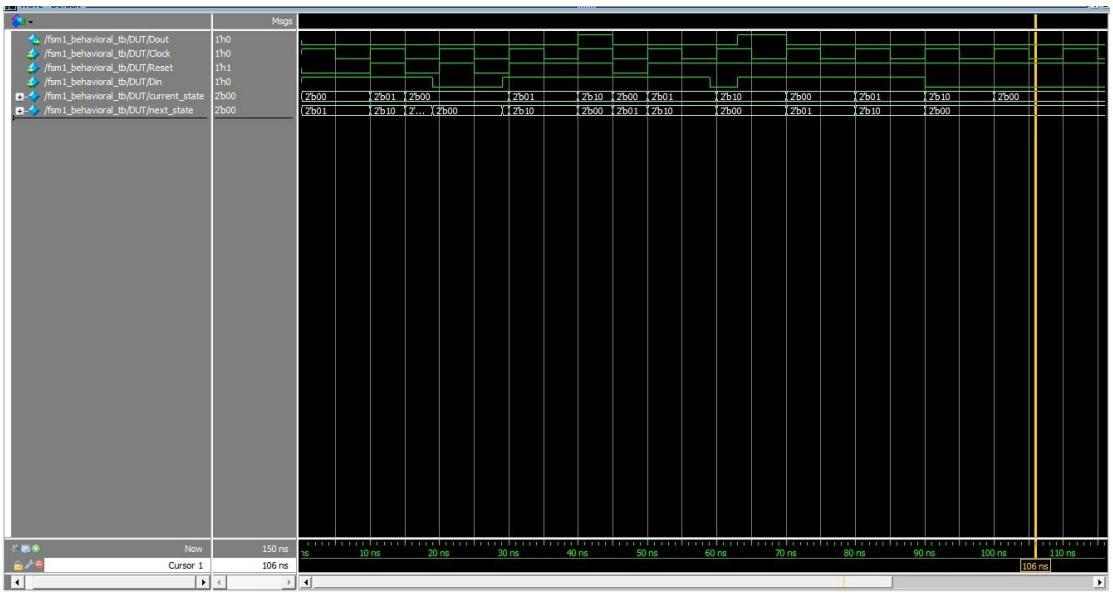
```
module fsm1_behavioral_tb;

        wire Dout_tb;
        reg Clock_tb, Reset_tb, Din_tb;
```

```
fsm1_behavioral DUT (Dout_tb, Clock_tb, Reset_tb, Din_tb);

// Reset-----------------------------------------------
initial
 begin
                    Reset_tb = 1'b0;
          #10       Reset_tb = 1'b1;
          #5        Reset_tb = 1'b0; // reset sto mid 15 ns
          #5        Reset_tb = 1'b1; // epanafora reset sto 1
          #5        Reset_tb = 1'b0; // reset sto start sto 25ns
          #5        Reset_tb = 1'b1; // epanafora reset sto 1
          #15       Reset_tb = 1'b0; // reset sto start sto 45ns
          #5        Reset_tb = 1'b1; // epanafora reset sto 1




 end
// Clock signal-----------------------------------------
initial
 begin
          Clock_tb = 1'b1;
 end

always
 begin
          #5 Clock_tb = ~Clock_tb; // Periodos rologiou 10ns
 end


// Shma Din-------------------------------------------------
initial
 begin
```

```
        #9      Din_tb = 1'b1; // start->mid->reset sto 10ns
        #10     Din_tb = 1'b0; // start->start->reset sto 20ns
        #10     Din_tb = 1'b1; // start->mid(Din=1)->done->reset sto 30ns
        #30     Din_tb = 1'b0; // start->mid(Din=0)->done
        #4      Din_tb = 1'b1; // start->mid(Din=0)->done(Dout=1)
        #27     Din_tb = 1'b0; // start->mid->done(Dout=0)


    end


endmodule
```

Φωτογραφίες





Σχόλια

Το FSM που υλοποιήθηκε είναι τύπου Mealy διότι η έξοδος εξαρτάται άμεσα από την είσοδο αλλά και από την τρέχουσα κατάσταση. Επίσης , στο μπλοκ για την επόμενη κατάσταση υλοποιείται το διάγραμμα μεταβάσεων που μας έχει δοθεί σε αυτή την άσκηση. Τέλος, στο μπλοκ εξόδου βλέπουμε ότι στο sensitivity list είναι και το current_state αλλά και το Din που όπως ήδη έχει αναφερθεί επηρεάζει άμεσα την έξοδο του FSM. Στην συνέχεια, στο test bench εξετάζεται αν το FSM λειτουργεί σωστά με τις προδιαγραφές που μας έχουν δοθεί και έτσι εξετάζονται οι εξής μεταβάσεις με το ρολόι να έχει 10 ns περίοδο:

- Αρχικά, εξετάζεται το reset
- Start -> Mid -> reset
- Start -> Start -> reset
- Start -> Mid -> Done -> reset
- Start -> Mid -> Done -> Start (Dout = 1)
- Start -> Mid -> Done -> Start (Dout = 0)

Πιο συγκεκριμένα δοκιμάζουμε τα εξής:

- Από 0-10 ns βάζουμε reset=0 και έτσι παρόλου που έχουμε Din = 1 το fsm παραμένει στην κατάσταση Start.
- Στα 10 ns με Din = 1 μεταβαίνουμε σωστά από Start -> Mid και στη συνέχεια στα 15 ns γίνεται reset και επιστρέφει το fsm στην κατάσταση Start.
- Στα 20 ns με Din = 0 και σε κατάσταση Start το fsm σωστά συνεχίζει να βρίσκεται στην κατάσταση Start και μετά με reset

στα 25 ns “επιστρέφει” στην κατάσταση Start όπως ήταν αναμενόμενο

- Στα 30 ns σε κατάσταση Start και με Din = 1 το fsm πάει στο Mid και πάλι με Din = 1 πάει στο Done και στα 45 ns κάνουμε reset και το fsm πάλι επιστρέφει σωστά στο Start.
- Στα 50 ns με Din = 1 και κατάσταση Start το fsm μεταβαίνει στο Mid και στα 60 ns με Din = 0 και κατάσταση Mid το fsm σωστά μεταβαίνει σε Done και με Din = 1 μας βγάζει έξοδο Dout = 1 και στα 70 ns επιστρέφει πάλι στο Start αφού με Din = 1 και κατάσταση Done το fsm επιστρέφει στην αρχική κατάσταση Start.
- Τέλος, στα 90 ns και σε κατάσταση Done με Din = 0 έχουμε και την αναμενόμενη έξοδο Dout = 0 με το fsm να επιστρέφει στην κατάσταση Start.

Ερώτημα β)

Πίνακας Μεταβάσεων (Κατάστασης) One-Hot

Present State	**Next State**	**Next State**	**Output**	**Output**
	με X=0	με X=1	με X=0	με X=1
Start	Start	Midway	0	0
Midway	Done	Done	0	0
Done	Start	Start	0	1

Κωδικοποίηση Καταστάσεων One-Hot

Κατάσταση	**Κωδικοποίηση $D_{1:0}$**
Start	001
Midway	010
Done	100

Λογικές Εξισώσεις Καταστάσεων-Εξόδων One-Hot

Present State D_2	Present State D_1	Present State D_0	Input X	Next State $D_{2'}$	Next State $D_{1'}$	Next State $D_{0'}$	Output Y
0	0	1	0	0	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	1	0	0	0
0	1	0	1	1	0	0	0
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	1

- $D_{2,}' = D_1$
- $D_{1,}' = D_0 X$
- $D_0' = \cancel{D_1}\ \cancel{X} + D_2 X$
- $Y = D_2 X$

Κώδικας Verilog

```
module fsm1_behavioral_one_hot (output reg Dout,
                                input wire Clock, Reset, Din);

    reg [2:0] current_state, next_state;
    parameter Start = 3'b001,
              Midway = 3'b010,
              Done = 3'b100;

    // Apothikeush Katastashs---------------------------------
    always @ (posedge Clock or negedge Reset)
     begin: STATE_MEMORY
      if (!Reset)
            current_state <= Start;
      else
            current_state <= next_state;
    end

    // Gia Epomenh Katastash-----------------------------------
    always @ (current_state or Din)
```

```
        begin: NEXT_STATE_LOGIC
         case (current_state)
          Start : if (Din==1'b1) next_state = Midway;
                    else next_state = Start;
          Midway : next_state = Done;
          Done : next_state = Start;
          default : next_state = Start;
         endcase
        end

        // Exodos----------------------------------------------------------
        always @ (current_state or Din)
         begin: OUTPUT_LOGIC
          case (current_state)
           Start : Dout = 1'b0;
           Midway : Dout = 1'b0;
           Done : if (Din==1) Dout=1'b1;
                    else Dout = 1'b0;
          endcase
        end

endmodule
```

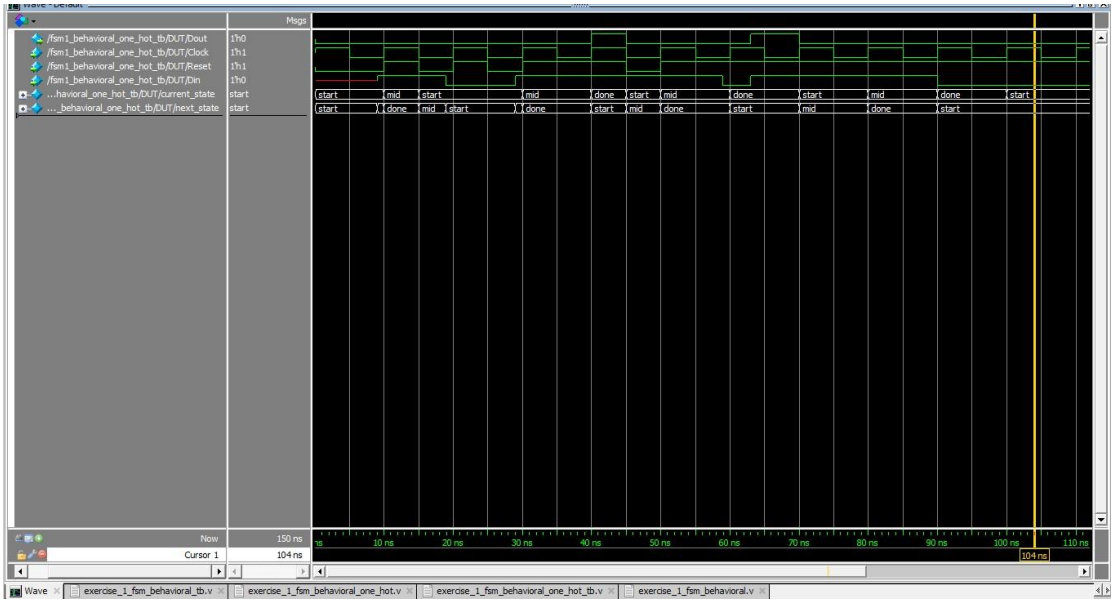
Test Bench

```
`timescale 1ns/1ns

module fsm1_behavioral_one_hot_tb;

        wire Dout_tb;
        reg Clock_tb, Reset_tb, Din_tb;
```

```
fsm1_behavioral_one_hot DUT (Dout_tb, Clock_tb, Reset_tb, Din_tb);

// Reset----------------------------------------------
initial
 begin
                    Reset_tb = 1'b0;
          #10       Reset_tb = 1'b1;
          #5        Reset_tb = 1'b0; // reset sto mid 15 ns
          #5        Reset_tb = 1'b1; // epanafora reset sto 1
          #5        Reset_tb = 1'b0; // reset sto start sto 25ns
          #5        Reset_tb = 1'b1; // epanafora reset sto 1
          #15       Reset_tb = 1'b0; // reset sto start sto 45ns
          #5        Reset_tb = 1'b1; // epanafora reset sto 1




 end
// Clock signal---------------------------------------------
initial
 begin
          Clock_tb = 1'b1;
 end

always
 begin
          #5 Clock_tb = ~Clock_tb; // Periodos rologiou 10ns
 end



// Shma Din-------------------------------------------------
initial
 begin
```

```
        #9      Din_tb = 1'b1; // start->mid->reset sto 10ns
        #10     Din_tb = 1'b0; // start->start->reset sto 20ns
        #10     Din_tb = 1'b1; // start->mid(Din=1)->done->reset sto 30ns
        #30     Din_tb = 1'b0; // start->mid(Din=0)->done
        #4      Din_tb = 1'b1; // start->mid(Din=0)->done(Dout=1)
        #27     Din_tb = 1'b0; // start->mid->done(Dout=0)


    end


endmodule
```

Φωτογραφίες

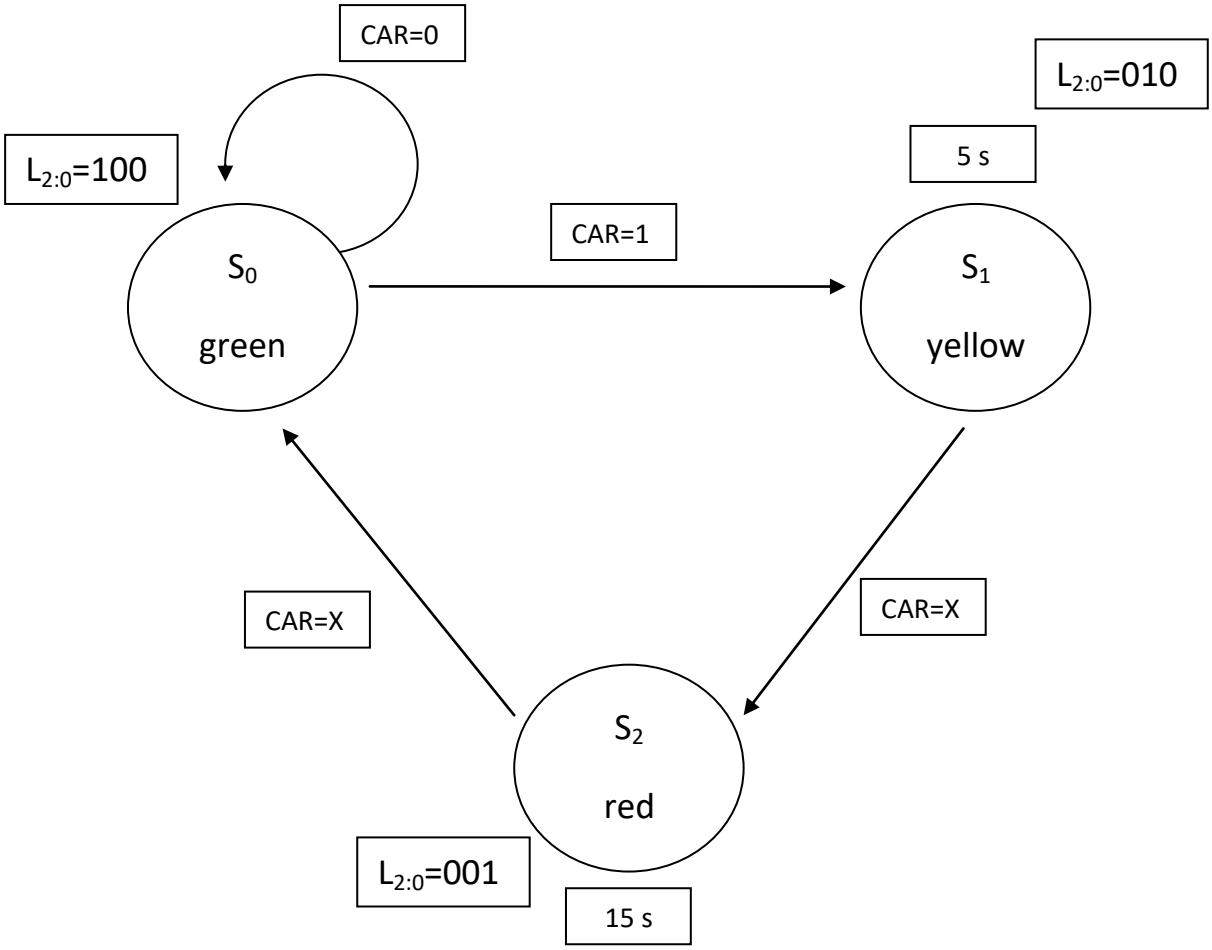


Σχόλια

Στο ερώτημα 2 της άσκησης 1 , το μόνο που αλλάζει είναι η κωδικοποίηση των καταστάσεων και αυτό φαίνεται στο ακριβώς παραπάνω screenshot όπου οι καταστάσεις κωδικοποιούνται με κωδικοποίηση One-Hot (001, 010, 100).

Εργασία 2

Διάγραμμα μεταβάσεων



Πίνακας Μεταβάσεων (Κατάστασης)

Present State	Next State CAR=0	Next State CAR=1
S_0	S_0	S_1
S_1	S_2	S_2
S_2	S_0	S_0

Κωδικοποίηση Καταστάσεων

Κατάσταση	Κωδικοποίηση $D_{1:0}$	Όνομα
S_0	00	green
S_1	01	yellow
S_2	10	red

Κωδικοποίηση Εξόδων

Έξοδος	Κωδικοποίηση $L_{2:0}$ $(_{GRN,YLW,RED})$
green	100
yellow	010
red	001

Λογικές Εξισώσεις Καταστάσεων

Present State D_1D_0	Next State CAR=0 $D_{1'}D_{0'}$	Next State CAR=1 $D_{1'}D_{0'}$
00	00	01
01	10	10
10	00	00

- $D_{1,}^{'} = D_0$
- $D_0^{'} = \cancel{D_1}\ \cancel{D_0}$ CAR

Λογικές Εξισώσεις Εξόδων

Present State **D_1D_0**	**Output** **(GRN,YLW,RED)** **$L_{2:0}$**
00	100
01	010
10	001

- $L_0 = D_1$
- $L_1 = D_0$
- $L_2 = \overline{D_1}\ \overline{D_0}$

Κώδικας Verilog

```
module fsm2_behavioral (output reg GRN, YLW, RED,
                        input wire Clock, Reset, CAR);

    reg [1:0] current_state, next_state;
    reg [4:0] CNT;
    reg TIMEOUT, YLWTIMEOUT, Reset_sy;

    parameter green = 2'b00,
              yellow = 2'b01,
              red = 2'b10;


    // Sichrono reset-----------------------------------------------------------
    always @ (posedge Clock)
     begin
          Reset_sy <= Reset;
     end


    // Apothikeush Katastashs---------------------------------------------------
```

```
always @ (posedge Clock or negedge Reset_sy)
 begin: STATE_MEMORY
  if (!Reset_sy)
            current_state <= yellow;
  else
            current_state <= next_state;
end


// Aparithmiths----------------------------------------------------------------
always @ (posedge Clock or negedge Reset_sy)
 begin: COUNTER
  if (!Reset_sy)
   begin
            CNT <= 13;
            TIMEOUT <= 0;
            YLWTIMEOUT <= 0;
   end
  else
   begin
            case (current_state)
             green: begin
                        CNT <= 14;
                        TIMEOUT <= 0;
                        YLWTIMEOUT <= 0;
                        end
             yellow: if (CNT==11)
                        begin
                        CNT <= 15;
                        YLWTIMEOUT <= 1;
                        end
                       else
                        begin
                        CNT <= CNT-1;
```

```
                    YLWTIMEOUT <= 0;
                    TIMEOUT <= 0;
                    end
            red: if (CNT==1)
                    begin
                    CNT <= 15;
                    TIMEOUT <= 1;
                    end
                   else
                    begin
                    CNT <= CNT-1;
                    TIMEOUT <= 0;
                    YLWTIMEOUT <= 0;
                    end
          endcase
  end
end



// Epomenh Katastash----------------------------------------------------------
always @ (current_state or CAR or YLWTIMEOUT or TIMEOUT)
 begin: NEXT_STATE_LOGIC
  case (current_state)
   green : if (CAR==1'b1) next_state = yellow;
          else next_state = green;

   yellow : if (YLWTIMEOUT==1) next_state = red;
          else next_state = yellow;

    red : if (TIMEOUT==1) next_state = green;
          else next_state = red;
  endcase
 end
```

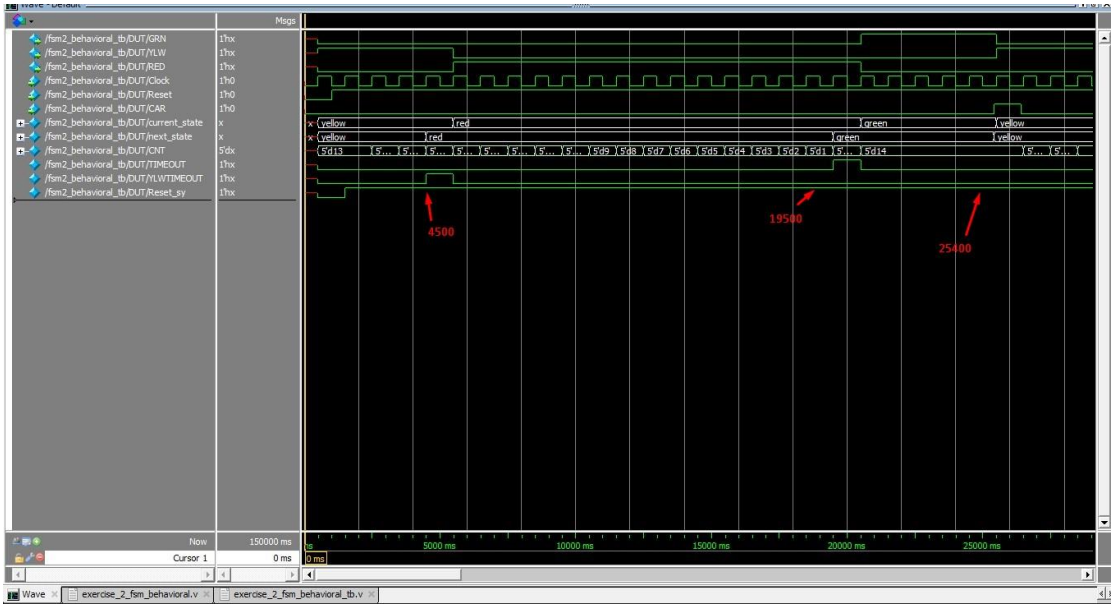
```
// Exodos----------------------------------------------------------------------
always @ (current_state or CAR)
 begin: OUTPUT_LOGIC
  case (current_state)
   green :
            begin
             GRN = 1'b1;
             YLW = 1'b0;
             RED = 1'b0;
            end
   yellow :
            begin
             GRN = 1'b0;
             YLW = 1'b1;
             RED = 1'b0;
            end
   red :
            begin
             GRN = 1'b0;
             YLW = 1'b0;
             RED = 1'b1;
            end

  endcase
 end

endmodule
```

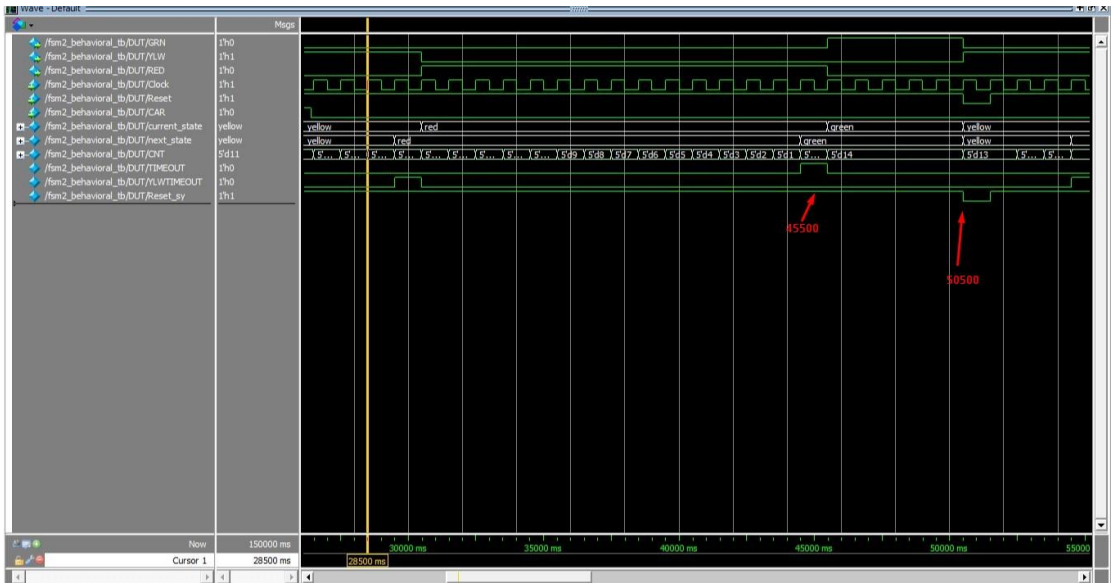
Test Bench

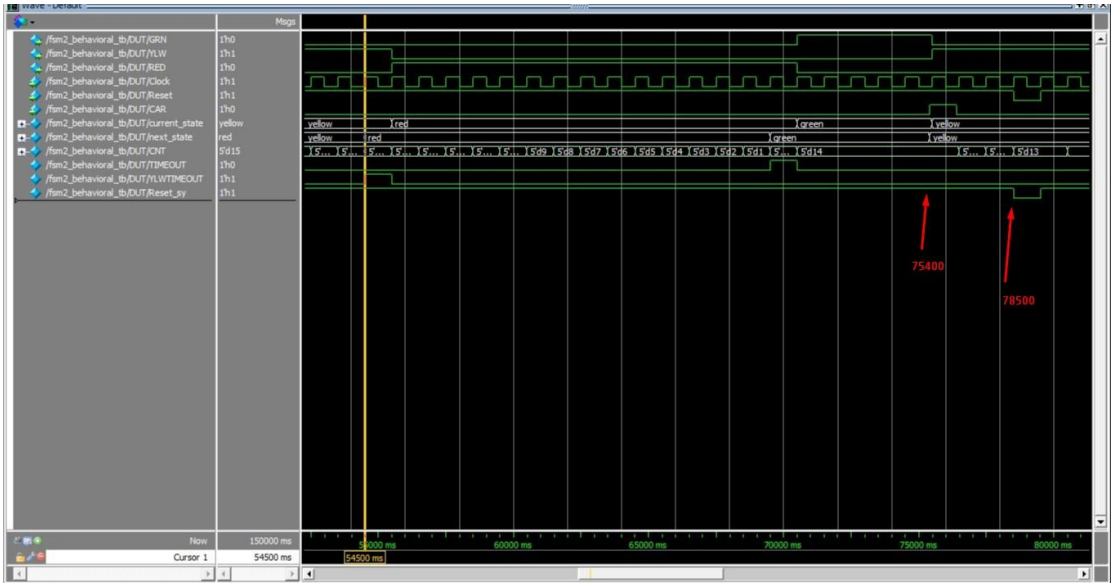
```
`timescale 1ms/1ms

module fsm2_behavioral_tb;

        wire GRN_tb, YLW_tb, RED_tb;
        reg Clock_tb, Reset_tb, CAR_tb;


        fsm2_behavioral DUT (.GRN(GRN_tb), .YLW(YLW_tb), .RED(RED_tb),
                                .Clock(Clock_tb), .Reset(Reset_tb), .CAR(CAR_tb));


        // Reset---------------------------------------------------
        initial
         begin
                Reset_tb = 1'b0; //Elegxw synchrono reset
        #1000   Reset_tb = 1'b1; // reset->1 , xekinaei o elegxos toy fsm
        #49500  Reset_tb = 1'b0; // reset->0 , elegxos metavashs green->yellow->red me to reset.
        #1000   Reset_tb = 1'b1; // epanafora reset se 1 meta apo mia periodo
        #27000  Reset_tb = 1'b0; // kanw reset otan exw yellow gia na paw yellow->yellow->red
        #1000   Reset_tb = 1'b1; // epanafora reset se 1 meta apo mia periodo
        #33000  Reset_tb = 1'b0; // reset sto red gia na paw red->yellow->red
        #1000   Reset_tb = 1'b1; // epanafora reset se 1 meta apo mia periodo



         end

        // Clock---------------------------------------------------
        initial
         begin
                Clock_tb = 1'b0;
         end

        always
```

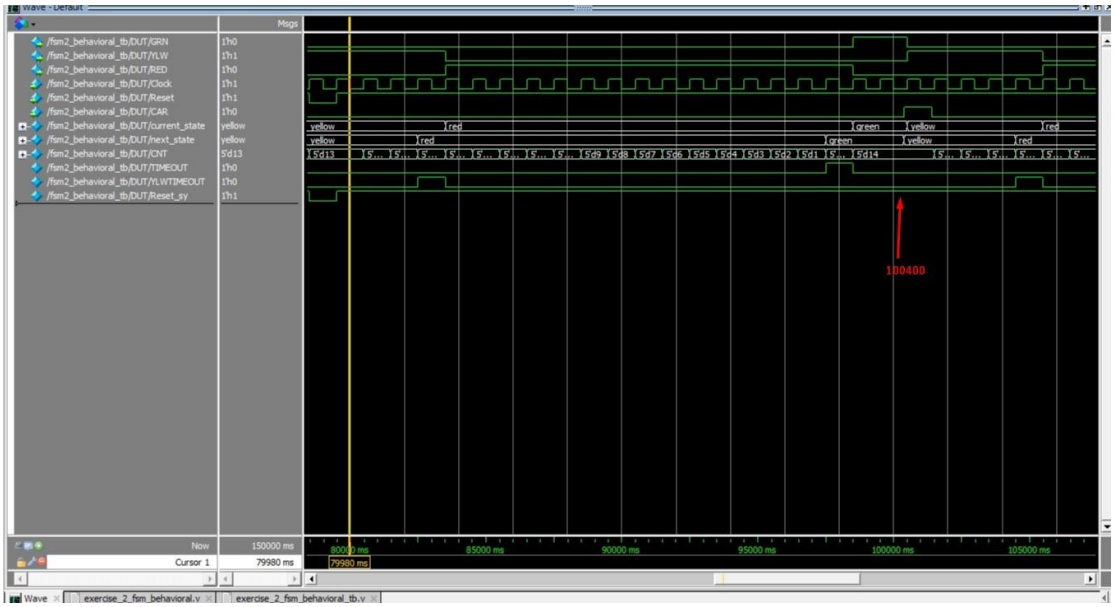
```
	begin
		#500 Clock_tb = ~Clock_tb;
	end


	// Shma CAR_tb ------------------------------------------------
	initial
	begin
			CAR_tb = 1'b0; //20500 me 25500 elegxw to green an sinexizei na einai green
		#25400	CAR_tb = 1'b1; // kanw CAR_tb=1 gia na elegxw green->yellow->red meta apo 20 second
		#1000	CAR_tb = 1'b0; // xanakanw CAR_tb=0 wste otan girisw sto green na meinw sto green
		#49000	CAR_tb = 1'b1; // CAR_tb=1 gia na paw yellow kai na testarw reset sto yellow
		#1000	CAR_tb = 1'b0; // xana CAR_tb=0
		#24000	CAR_tb = 1'b1; // CAR_tb=1 gia na paw red na kanw reset
		#1000	CAR_tb = 1'b0; // epanafora car wste otan paw green na meinw


	end

endmodule
```

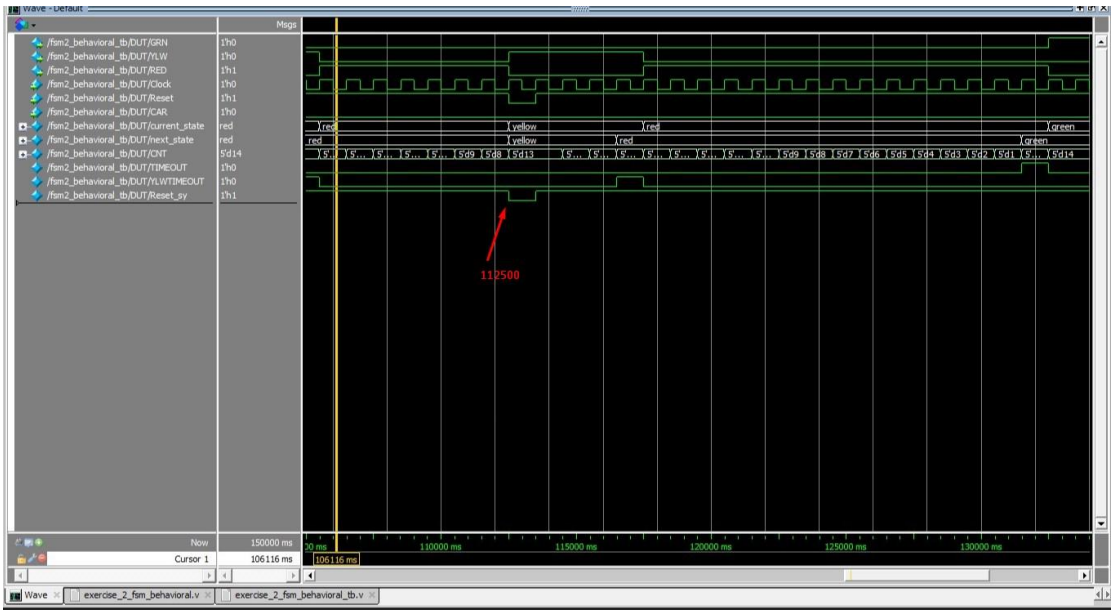
Φωτογραφίες






/fsm2_behavioral_tb/DUT/GRN
/fsm2_behavioral_tb/DUT/YLW
/fsm2_behavioral_tb/DUT/RED
/fsm2_behavioral_tb/DUT/Clock
/fsm2_behavioral_tb/DUT/Reset
/fsm2_behavioral_tb/DUT/CAR
/fsm2_behavioral_tb/DUT/current_state
/fsm2_behavioral_tb/DUT/next_state
/fsm2_behavioral_tb/DUT/CNT
/fsm2_behavioral_tb/DUT/TIMEOUT
/fsm2_behavioral_tb/DUT/YLWTIMEOUT
/fsm2_behavioral_tb/DUT/Reset_sy
75400
78500
Now
Cursor 1
54500 ms


/fsm2_behavioral_tb/DUT/GRN
/fsm2_behavioral_tb/DUT/YLW
/fsm2_behavioral_tb/DUT/RED
/fsm2_behavioral_tb/DUT/Clock
/fsm2_behavioral_tb/DUT/Reset
/fsm2_behavioral_tb/DUT/CAR
/fsm2_behavioral_tb/DUT/current_state
/fsm2_behavioral_tb/DUT/next_state
/fsm2_behavioral_tb/DUT/CNT
/fsm2_behavioral_tb/DUT/TIMEOUT
/fsm2_behavioral_tb/DUT/YLWTIMEOUT
/fsm2_behavioral_tb/DUT/Reset_sy
100400
Now
Cursor 1
79980 ms
Wave
exercise_2_fsm_behavioral.v
exercise_2_fsm_behavioral_tb.v


/fsm2_behavioral_tb/DUT/GRN
/fsm2_behavioral_tb/DUT/YLW
/fsm2_behavioral_tb/DUT/RED
/fsm2_behavioral_tb/DUT/Clock
/fsm2_behavioral_tb/DUT/Reset
/fsm2_behavioral_tb/DUT/CAR
/fsm2_behavioral_tb/DUT/current_state
/fsm2_behavioral_tb/DUT/next_state
/fsm2_behavioral_tb/DUT/CNT
/fsm2_behavioral_tb/DUT/TIMEOUT
/fsm2_behavioral_tb/DUT/YLWTIMEOUT
/fsm2_behavioral_tb/DUT/Reset_sy
112500
Now
Cursor 1
106116 ms
Wave
exercise_2_fsm_behavioral.v
exercise_2_fsm_behavioral_tb.v

Σχόλια

Αρχικά, χρησιμοποιήσαμε για αυτή την άσκηση ένα σύγχρονο σήμα Reset, το Reset_sy , ώστε να έχουμε ένα ολικά σύγχρονο fsm ώστε να έχουμε πιο ασφαλείς-ομαλές μεταβάσεις όταν ενεργοποιείται το Reset. Συνεπώς, για ένα σύγχρονο Reset, περνάμε το σήμα Reset από ένα flip-flop και παίρνουμε ως έξοδο το σήμα Reset_sy. Επίσης, το fsm που υλοποιήθηκε είναι τύπου Moore αφού η έξοδος εξαρτάται αποκλειστικά από την τρέχουσα κατάσταση. Στη συνέχεια, στο μπλοκ για την αποθήκευση των καταστάσεων χρησιμοποιήθηκε κατά την ενεργοποίηση του Reset_sy η αρχική κατάσταση του πορτοκαλί φαναριού όπου στη συνέχεια μετά από 5 δευτερόλεπτα θα καταλήξει σε κόκκινο και μετά από 15 δευτερόλεπτα σε πράσινο. Το πορτοκαλί φανάρι ως κατάσταση reset , έγινε για ένα πιο ασφαλές φανάρι, δηλαδή σε περίπτωση reset το φανάρι να οδηγείται σε πορτοκαλί->κόκκινο και όχι πχ σε πράσινο γιατί θα ήταν επικίνδυνο αν πατώντας το reset πηγαίναμε από πορτοκαλί ή κόκκινο κατευθείαν στο πράσινο , αλλά ούτε και σε κόκκινο γιατί πάλι θα ήταν επικίνδυνο να μεταβαίναμε από πράσινο σε κόκκινο κατευθείαν μετά την ενεργοποίηση του reset. Τέλος, τα 5 δευτερόλεπτα στο πορτοκαλί χρησιμοποιήθηκαν διότι είναι δρόμος ταχείας κυκλοφορίας συνεπώς για μεγαλύτερες ταχύτητες χρειαζόμαστε και περισσότερο χρόνο στο πορτοκαλί φανάρι. Επιπλέον, στο μπλοκ του απαριθμητή χρησιμοποιήθηκαν τα σήματα TIMEOUT και YLWTIMEOUT ώστε όταν περνάνε τα 15 δευτερόλεπτα στο κόκκινο και τα 5 δευτερόλεπτα αντίστοιχα στο πορτοκαλί να γίνονται το TIMEOUT=1 και το YLWTIMEOUT=1 πάλι αντίστοιχα . Μετά, το μπλοκ της επόμενης κατάστασης υλοποιεί το διάγραμμα μεταβάσεων με βάση τα σήματα TIMEOUT και YLWTIMEOUT αλλά και το σήμα CAR και το μπλοκ της εξόδου υλοποιεί την επιθυμητή έξοδο που θέλουμε να έχει το φανάρι στην κάθε κατάσταση. Επί πρόσθετα, μέσω του test bench εξετάζουμε τι εξής μεταβάσεις με το ρολόι να έχει περίοδο 1 s (1000 ns) :

- Πρώτα γίνεται ένα reset για να εξετάσουμε το σύγχρονο reset.
- green -> yellow -> red με CAR=1
- reset στο green
- green -> yellow με CAR=1 και reset στο yellow
- green -> yellow -> red με CAR=1 και reset στο red.

Πιο συγκεκριμένα έχουμε τα εξής στα screenshots

- από 0 μέχρι 20500 ms κάνουμε το αρχικό reset και περιμένουμε το φανάρι μετά από 20 δευτερόλεπτα (5 στο πορτοκαλί και 15 στο κόκκινο) να επανέλθει στην κατάσταση green. Επίσης, στα 4500 ms και στα 19500 ms βλέπουμε ότι ενεργοποιούνται τα σήματα YLWTIMEOUT και TIMEOUT για το yellow και το red αντίστοιχα.
- στα 25400 ms ενεργοποιούμε το CAR=1 ώστε να τεστάρουμε τις μεταβάσεις green -> yellow -> red. Και λειτουργεί σωστά αφού μετά από 20 δευτερόλεπτα δηλαδή στο 45500 το φανάρι

επανέρχεται ξανά στο green αφού πρώτα ήταν για 5 δευτερόλεπτα yellow και 20 δευτερόλεπτα red.

- Στα 50500 ms καθώς το φανάρι είναι green κάνουμε reset. Παρατηρείται η αναμενόμενη συμπεριφορά, δηλαδή το φανάρι γίνεται για 5 δευτερόλεπτα yellow και μετά για 15 δευτερόλεπτα red και τελικά επανέρχεται στο green.
- Στα 75400 ms κάνουμε το CAR=1 ώστε να μεταβεί το φανάρι σε yellow ώστε να κάνουμε reset στην κατάσταση yellow. Στα 78500 ms κάνουμε reset στο yellow και έχουμε τα επιθυμητά αποτελέσματα, δηλαδή +5 δευτερόλεπτα στο πορτοκαλί και μετά 15 δευτερόλεπτα στο red και τελικά το φανάρι γίνεται πάλι green.
- Στα 100400 ms κάνουμε το CAR=1 ώστε να μεταβεί το φανάρι σε red και εκεί να κάνουμε reset. Στα 112500 και ενώ το φανάρι είναι red κάνουμε reset και το φανάρι γίνεται yellow για 5 δευτερόλεπτα και μετά red για 15 δευτερόλεπτα και τελικά green.

Σε όλες τις παραπάνω μεταβάσεις φαίνονται και τα σήματα TIMEOUT και YLWTIMEOUT τα οποία δουλεύουν σωστά και μας δίνουν τις ανάλογες επιθυμητές μεταβάσεις. Επίσης, φαίνονται και οι έξοδοι GRN, YLW, RED που ακολουθούν σωστά την τρέχουσα κατάσταση σε κάθε χρονική στιγμή. Δηλαδή, όταν το φανάρι είναι σε κατάσταση green τότε η έξοδος είναι 100 (GRN,YLW,RED) , όταν το φανάρι είναι yellow τότε οι έξοδοι είναι 010 (GRN,YLW,RED) και όταν το φανάρι είναι red οι έξοδοι είναι 001 (GRN,YLW,RED).

Τέλος