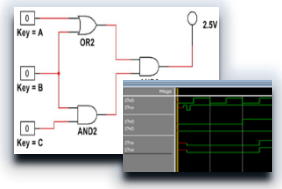
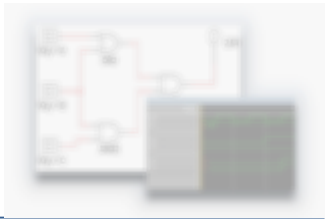


ECE119 – Ψηφιακή Σχεδίαση

Διδάσκοντες Εργαστηρίου: Δ. Καραμπερόπουλος
Δ. Γαρυφάλλου

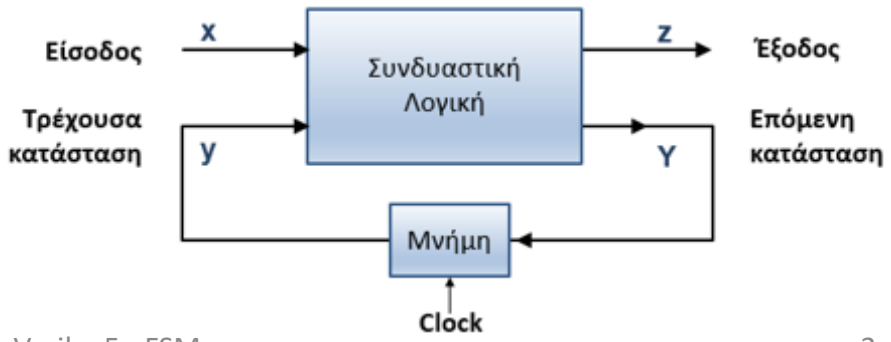
- Lab10: Verilog (Μέρος 5 - FSM)

Η Γλώσσα Verilog (Μέρος 5 - FSM)



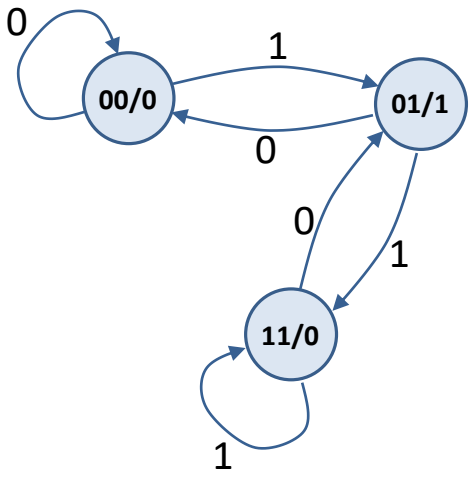
FSM (finite-state-machine)

- Ένα σύγχρονο ακολουθιακό λογικό κύκλωμα λέγεται και **Μηχανή Πεπερασμένων Καταστάσεων** (finite-state-machine, FSM)
- Ένα FSM περιγράφεται πλήρως από 2 λογικές συναρτήσεις
  - Η μία υπολογίζει την **επόμενη κατάσταση** ως συνάρτηση της εισόδου και τρέχουσας κατάστασης.
  - Η άλλη υπολογίζει την **έξοδό του** επίσης ως συνάρτηση της εισόδου? και τρέχουσας κατάστασης.
- Αυτές οι δύο συναρτήσεις περιγράφονται με έναν πίνακα καταστάσεων ή ένα διάγραμμα καταστάσεων.
- Άρα η σύνθεση μιας μηχανής πεπερασμένων καταστάσεων συνίσταται απλώς στη σύνθεση των λογικών συναρτήσεων **επόμενης κατάστασης** και **εξόδου**.



Υλοποίηση FSM

- Έχουμε 3 καταστάσεις, 1 είσοδο και 1 έξοδο
- Οι καταστάσεις κωδικοποιούνται με δύο D-type flip-flops, A και B.

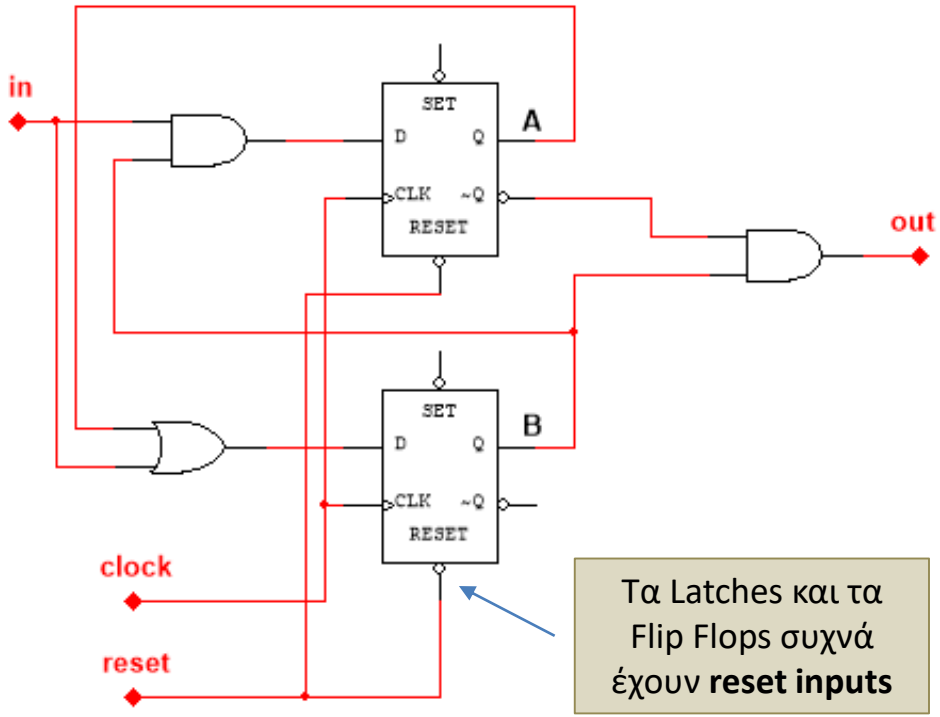


state encoding

AB/out

Παρούσα κατάσταση		Είσοδος	Επόμενη κατάσταση		Είσοδοι flip-flop		Έξοδος
A	B	in	A	B	DA	DB	out
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	1
1	0	X	X	X	X	X	X
1	0	X	X	X	X	X	X
1	1	0	0	1	0	1	0
1	1	1	1	1	1	1	0

DA = B & in
DB = A + in
out = A' & B



Τα Latches και τα Flip Flops συχνά έχουν **reset inputs**

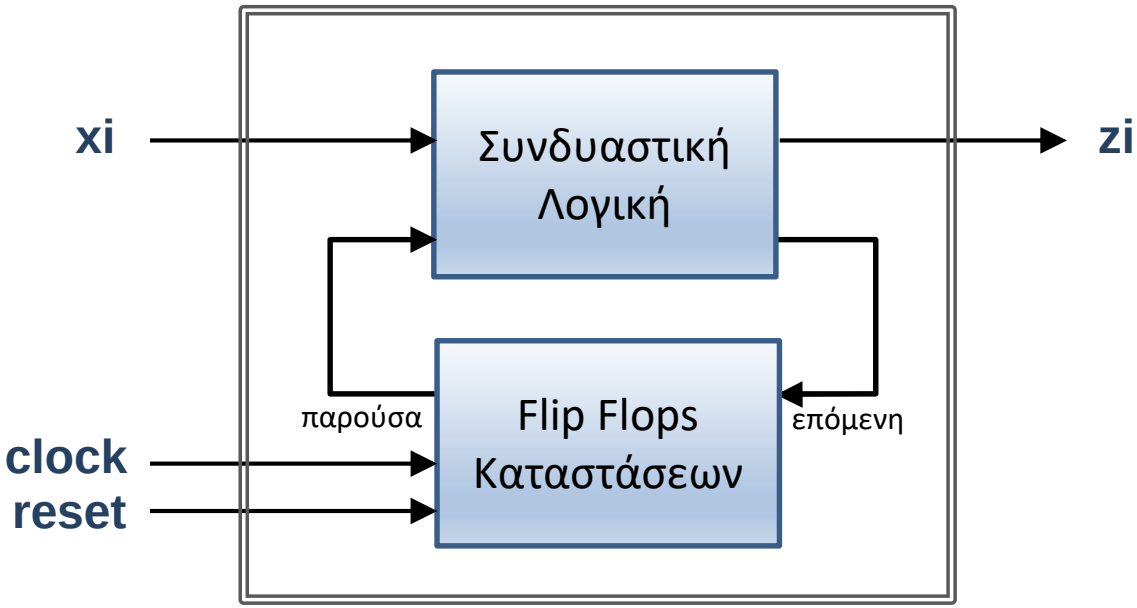
Υλοποίηση FSM σε Verilog - Γενικό Μοντέλο

Συνδυαστική λογική

- Η **επόμενη κατάσταση** είναι μία συνδυαστική συνάρτηση της παρούσης κατάστασης και των εισόδων
- Οι **έξοδοι** είναι μία συνδυαστική συνάρτηση της παρούσης κατάστασης [και των εισόδων]

Flip Flops καταστάσεων

- Η **έξοδος** τους είναι η παρούσα κατάσταση του FSM
- Η **είσοδός** τους είναι η επόμενη κατάσταση την οποία τα flip-flops θα «φορτώσουν στην έξοδό τους» έπειτα από την ακμή του ρολογιού



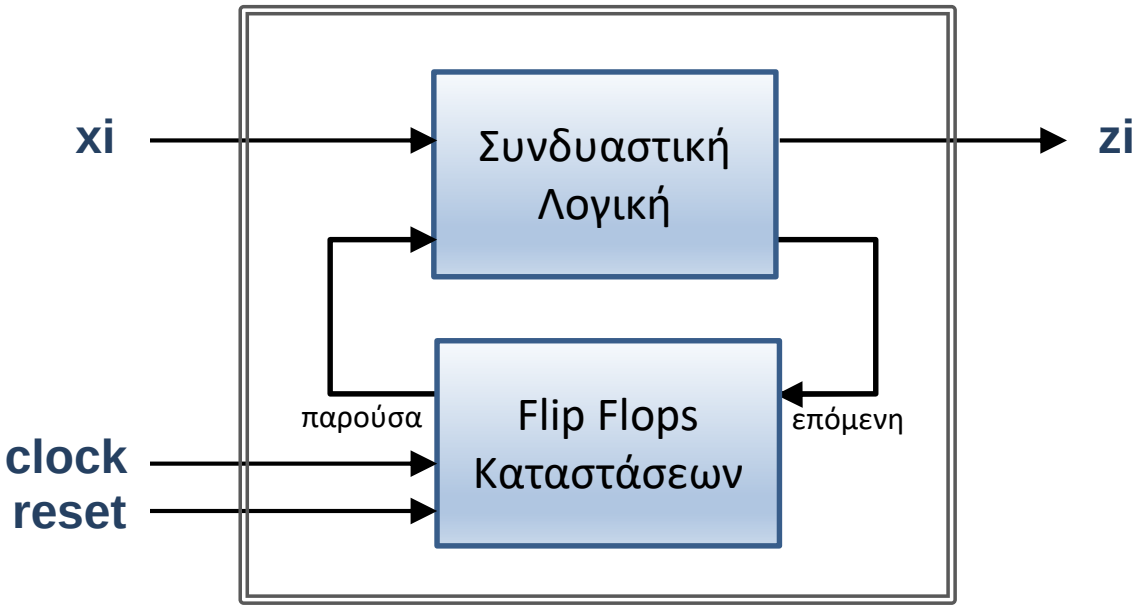
Standard Model of a Finite State Machine

Υλοποίηση FSM σε Verilog - Γενικό Μοντέλο

- Αυτή η δομή χρησιμοποιείται και στην περιγραφή **Verilog**

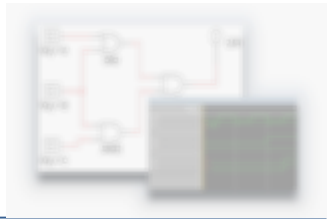
- Οι **Συνδυαστικές Λογικές Συναρτήσεις** της επόμενης κατάστασης και των εξόδων περιγράφονται με μοντελοποίηση συμπεριφοράς (behaviorally) σε ένα always block ακολουθώντας κάποιους κανόνες.

- Τα **Flip Flops Καταστάσεων** (ακολουθιακή λογική) περιγράφονται σε ένα ξεχωριστό always block ακολουθώντας ένα διαφορετικό set κανόνων.



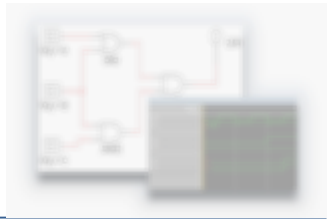
Standard Model of a Finite State Machine

Υλοποίηση FSM σε Verilog - Combinational Rules



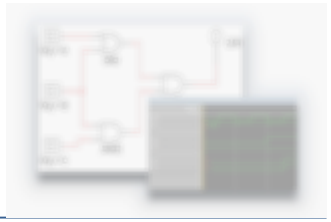
- **Κανόνες υλοποίησης Συνδυαστικών Κυκλωμάτων με always block**
  - Προσέχουμε να έχουμε συμπεριλάβει στην **sensitivity list** της **always** όλες τις εισόδους του συνδυαστικού μας κυκλώματος, έτσι ώστε εάν κάποια αλλάξει να επανυπολογιστούν η έξοδοι του κυκλώματος.
    - Θεμελιώδης κανόνας των συνδυαστικών κυκλωμάτων. Αλλάζει η είσοδος -> Αλλάζει η έξοδος.
  - Προσέχουμε ώστε να μην υπάρχει περίπτωση να εκτελεστεί ένα begin…end loop χωρίς να αναθέσουμε τιμή σε κάποια από τις συνδυαστικές εξόδους. Πρέπει **κάθε** συνδυαστική έξοδος σε **κάθε** begin…end loop να δέχεται μία νέα ανάθεση (μπορεί να έχει και την ίδια τιμή!)
    - Εάν δεν το κάνουμε τότε το κύκλωμα πρέπει να θυμάται την προηγούμενη τιμή. Οπότε οι έξοδοι του θα είναι συναρτήσεις των παρουσών εισόδων ΚΑΙ των προηγούμενων εξόδων. Αυτό είναι ένα θεμελιώδες χαρακτηριστικό των ακολουθιακών κυκλωμάτων! Το εργαλείο σύνθεσης θα τοποθετούσε μανδαλωτές (latches) για να υλοποιήσει αυτή την ακολουθιακή λογική και αυτό προφανώς δεν είναι επιθυμητό, δεδομένου ότι προσπαθούμε να υλοποιήσουμε ένα συνδυαστικό κύκλωμα.

Υλοποίηση FSM σε Verilog - Sequential Rules



- **Κανόνες υλοποίησης Ακολουθιακών Κυκλωμάτων με always block (1/2)**
  - Η sensitivity list του always block περιλαμβάνει μόνο τις ακμές των clock, reset.
    - Αυτές είναι οι μόνες είσοδοι που μπορούν να προκαλέσουν μία αλλαγή κατάστασης.
  - Μέσα στο always block η κατάσταση του reset καθορίζεται πρώτη.
    - Εάν αναμένεται negative edge του reset τότε η περιγραφή μπορεί να είναι "`if (~reset) …`"
    - Εάν αναμένεται positive edge του reset τότε η περιγραφή μπορεί να είναι "`if (reset) …`"
  - Η κατάσταση του clock δεν αναφέρεται-ελέγχεται μέσα στο begin…end block.

Υλοποίηση FSM σε Verilog - Sequential Rules



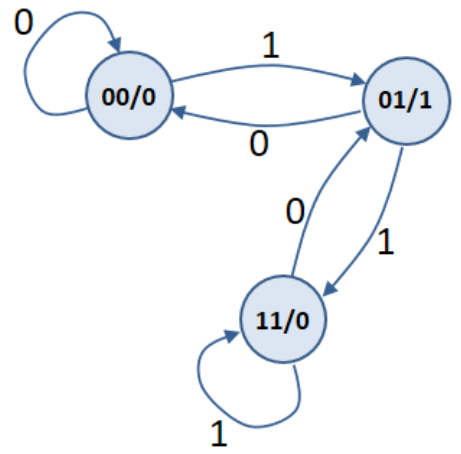
Κανόνες υλοποίησης Ακολουθιακών Κυκλωμάτων με always block (2/2)

- Κάθε register που του ανατίθεται τιμή σε ένα ακολουθιακό always block, θα υλοποιηθεί με flip flops στο τελικό κύκλωμα μετά τη σύνθεση.
  - Δεν μπορούμε να περιγράψουμε συνδυαστική λογική στο ίδιο always block που περιγράφουμε ακολουθιακή λογική.
  - Μπορούμε να γράψουμε συνδυαστική λογική, αλλά να γνωρίζουμε ότι το αποτέλεσμά της θα εκτιμηθεί στην ακμή του ρολογιού και θα “φορτωθεί” στον register.
- Non-blocking assignment (<=) είναι ο τελεστής ανάθεσης όταν περιγράφουμε την edge-sensitive συμπεριφορά ενός κυκλώματος.
  - Αυτό διότι όλες οι αναθέσεις, οι οποίες έχουμε καθορίσει να γίνουν στην ακμή του sensitivity list, πρέπει να γίνουν ταυτόχρονα!

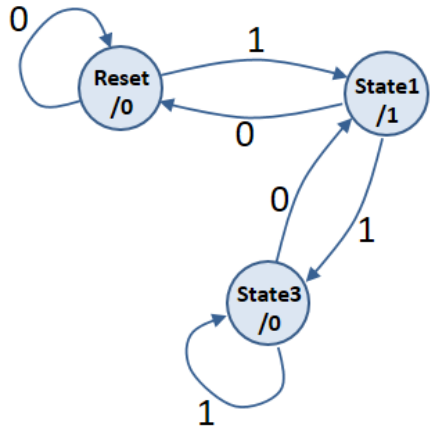
Υλοποίηση FSM σε Verilog

- **Έχουμε 3 καταστάσεις, 1 είσοδο και 1 έξοδο**
- Προσθέτουμε για εισόδους το **clock** και το **reset** (enable στο Low)
- Δηλώνουμε το **out ως register**.
- Δηλώνουμε δύο registers για τα states (**currentState**, **nextState**).
- Δίνουμε συμβολικές ονομασίες στα states (καταστάσεις)
- Δηλώνουμε ως **παραμέτρους** τις κωδικοποιημένες καταστάσεις, δίνοντας τις συμβολικές ονομασίες τους (π.χ. Reset, State1, State3)

```
module fsm (out, in, clock, reset);
    output          out;
    input           in, clock, reset;
    reg             out;
    reg [1:0]       currentState, nextState;

// State encoding //
parameter   Reset = 2'b00,
            State1 = 2'b01,
            State3 = 2'b11;
```



Μπορούμε να δώσουμε ονομασίες στα states

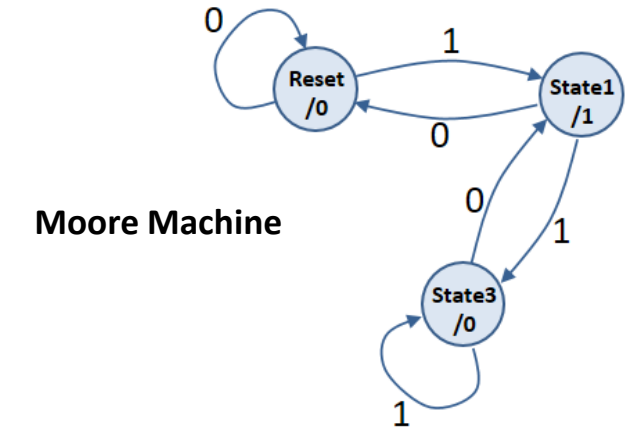


Υλοποίηση FSM σε Verilog

Συνδυαστική λογική

- Ενεργοποιείται στην αλλαγή της **εισόδου** ή της **παρούσης κατάστασης**
- Έχει εξόδους τα: **nextState** και **out**

```
// the Combinational portion //
always @(in or currentState) begin
    nextState = Reset;
    out = 1'b0;
    case (currentState)
        Reset: begin
            if (in) nextState = State1;
        end
        State1: begin
            out = 1'b1;
            if (in) nextState = State3;
        end
        State3: begin
            if (in) nextState = State3;
            else nextState = State1;
        end
        default: begin
            out = 1'bx;
            nextState = Reset;
        end
    endcase
end
```



Moore Machine

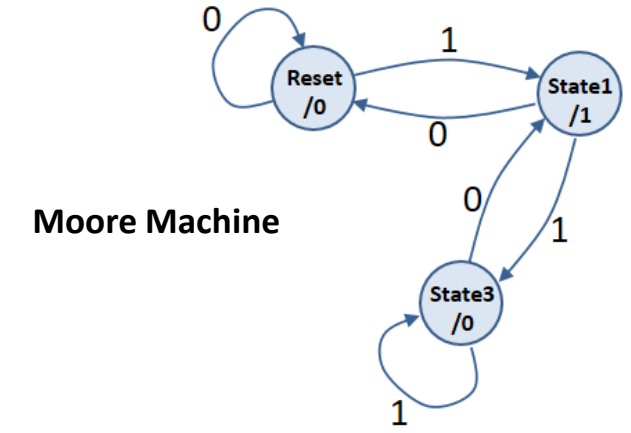
- Αρχικά **δίνουμε τιμές σε όλες τις εξόδους** του συνδυαστικού block (nextState, out) έτσι ώστε να είμαστε σίγουροι ότι σε κάθε εκτέλεση του block θα τους ανατεθούν τιμές.
- Στην δομή case περιλαμβάνουμε όλες τις καταστάσεις του κυκλώματος και αναθέτουμε στην **nextState**.
- Εάν σε κάποια κατάσταση η **έξοδος** δεν είναι αυτή που δώσαμε αρχικά, τότε της αναθέτουμε την σωστή τιμή.
- Εάν δεν έχουμε συμπεριλάβει όλες τις πιθανές περιπτώσεις στη δομή case, τότε χρησιμοποιούμε την περίπτωση **default**.

Υλοποίηση FSM σε Verilog

Flip Flops καταστάσεων - Ακολουθιακή λογική

- Η **έξοδος** τους είναι η παρούσα κατάσταση του FSM
- Η **είσοδός** τους είναι η επόμενη κατάσταση την οποία τα flip-flops θα «φορτώσουν στην έξοδό τους» έπειτα από την ακμή του ρολογιού
- Η **λίστα ευαισθησίας** αναμένει κάποιο γεγονός από τα δυο να συμβούν. Μία θετική ακμή (0 σε 1) του clock ή μία αρνητική ακμή (1 σε 0) του reset.
- Όταν ένα ή και τα δύο γεγονότα συμβούν τότε εκτελείται το begin…end block.

```
// the Sequential portion - State Registers //
always @(posedge clock or negedge reset) begin
    if (!reset)
        currentState <= Reset;
    else
        currentState <= nextState;
end
endmodule
```

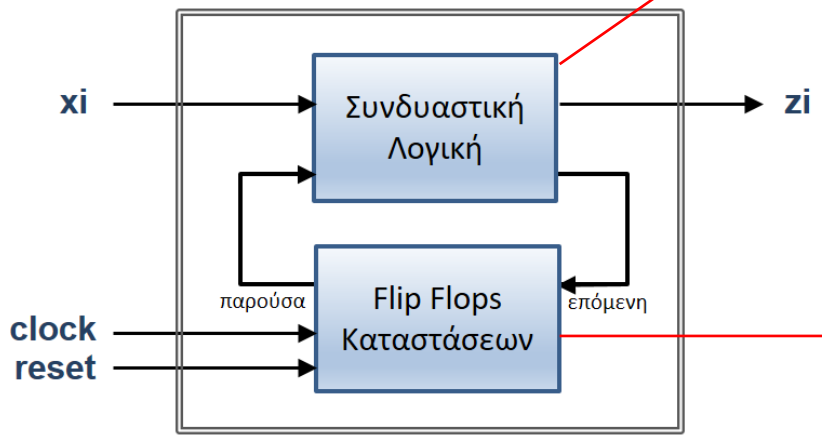


Moore Machine

- Εάν π.χ. έρθει μία **αρνητική ακμή στο reset**, τότε το currentState θα γίνει το Reset.
- Όσο το reset παραμένει στο “0”, ακόμα και εάν έρθει θετικός παλμός του clock, το currentState θα παραμένει στο Reset. (λειτουργία ασύγχρονου reset)
- ***Non-blocking*** assignments
- Εάν το reset είναι “1” και έρθει ένας **θετικός παλμός στο clock**, εκτελείται πάλι το block και “φορτώνει” στην παρούσα κατάσταση αυτή που είχε ετοιμάσει από πριν για την επόμενη.

Υλοποίηση FSM σε Verilog

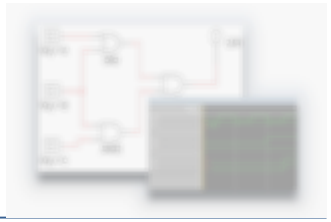
```
module fsm (out, in, clock, reset);
    output          out;
    input           in, clock, reset;
    reg             out;
    reg [1:0]       currentState, nextState;

// State encoding //
parameter   Reset = 2'b00,
            State1 = 2'b01,
            State3 = 2'b11;
```



Standard Model of a Finite State Machine

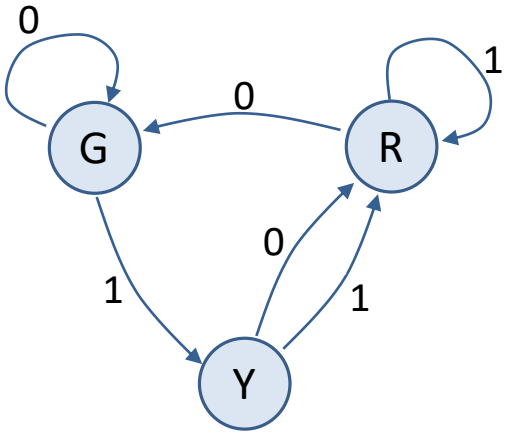
```
// the Combinational portion //
always @(in or currentState) begin
    nextState = Reset;
    out = 1'b0;
    case (currentState)
        Reset: begin
            if (in) nextState = State1;
        end
        State1: begin
            out = 1'b1;
            if (in) nextState = State3;
        end
        State3: begin
            if (in) nextState = State3;
            else nextState = State1;
        end
        default: begin
            out = 1'bx;
            nextState = Reset;
        end
    endcase
end

// the Sequential portion - State Registers //
always @(posedge clock or negedge reset) begin
    if (!reset)
        currentState <= Reset;
    else
        currentState <= nextState;
end
endmodule
```

Υλοποίηση FSM σε Verilog - Traffic Lights with Pedestrian Sensor



- Moore Machine
- Έχουμε 3 καταστάσεις
  - Green: 00
  - Yellow: 01
  - Red: 10

Παρούσα κατάσταση	Είσοδος	Επόμενη κατάσταση	Έξοδος				
	in		G	Y	R	Wait	Go
Green	0	Green	1	0	0	1	0
Green	1	Yellow	1	0	0	1	0
Yellow	0	Red	0	1	0	1	0
Yellow	1	Red	0	1	0	1	0
Red	0	Green	0	0	1	0	1
Red	1	Red	0	0	1	0	1



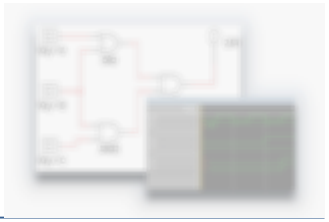
Traffic Lights with Pedestrian Sensor

```
module traffic (G, Y, R, Wait, Go, in, clock, reset);
    output          G, Y, R, Wait, Go;
    input           in, clock, reset;
    reg             G, Y, R, Wait, Go;
    reg [1:0]       currentState, nextState;

// State encoding //
parameter   Green = 2'b00,
            Yellow = 2'b01,
            Red = 2'b10;
```

Παρούσα κατάσταση	Είσοδος	Επόμενη κατάσταση	Έξοδος				
	in		G	Y	R	Wait	Go
Green	0	Green	1	0	0	1	0
Green	1	Yellow	1	0	0	1	0
Yellow	0	Red	0	1	0	1	0
Yellow	1	Red	0	1	0	1	0
Red	0	Green	0	0	1	0	1
Red	1	Red	0	0	1	0	1

```
// the Combinational portion //
always @(in or currentState) begin
    nextState = Green;
    G = 1'b0;
    Y = 1'b0;
    R = 1'b0;
    Wait = 1'b0;
    Go = 1'b0;
    case (currentState)
        Green: begin
            G = 1'b1;
            Wait = 1'b1;
            if (in) nextState = Yellow;
        end
        Yellow: begin
            Y = 1'b1;
            Wait = 1'b1;
            nextState = Red;
        end
        Red: begin
            R = 1'b1;
            Go = 1'b1;
            if (in) nextState = Red;
            else nextState = Green;  // Μπορεί να φύγει! γιατί?
        end
        default: begin
            nextState = 2'bxx;
            G = 1'bx;
            Y = 1'bx;
            R = 1'bx;
            Wait = 1'bx;
            Go = 1'bx;
        end
    endcase
end

// the Sequential portion - State Registers //
always @(posedge clock or negedge reset) begin
    if (!reset)
        currentState <= Green;
    else
        currentState <= nextState;
end
endmodule
```

Traffic Lights with Pedestrian Sensor

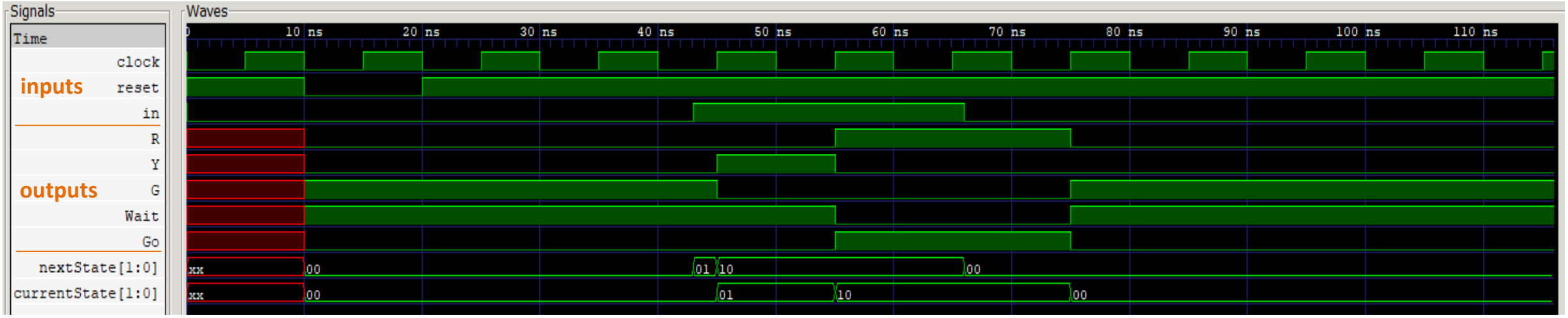


```
`timescale 1ns/100ps
module t_traffic;
wire    G, Y, R, Wait, Go;
reg     in, clock, reset;

traffic dut(G, Y, R, Wait, Go, in, clock, reset);

initial begin
    $dumpfile("traffic.vcd");
    $dumpvars(0,t_traffic);
end

initial begin
    in = 0;
    clock = 0;
    reset = 1;
    #10 reset = 0;
    #10 reset = 1;
    #23 in = 1;
    #23 in = 0;
    #50 $finish;
end

initial $monitor ("Time = %2d clock = %b reset = %b in = %b G = %b Y = %b R = %b Wait = %b Go = %b", $time, clock, reset, in, G, Y, R, Wait, Go);
always #5 clock = !clock;
endmodule
```

Traffic Lights with Pedestrian Sensor

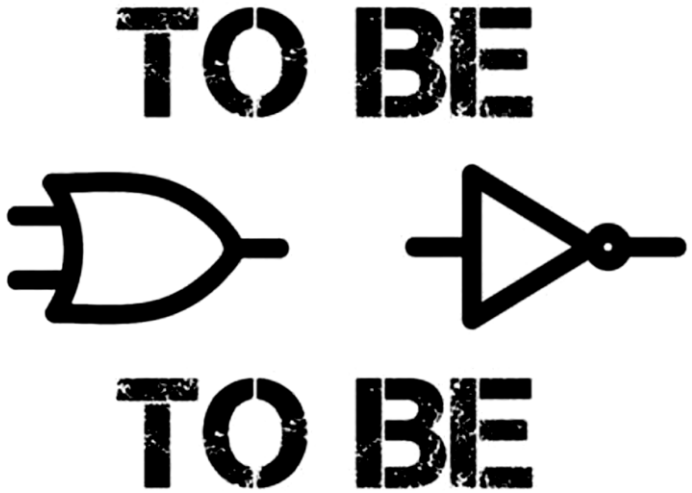
```
…
initial begin
    in = 0;
    clock = 0;
    reset = 1;
    #10 reset = 0;
    #10 reset = 1;
    #23 in = 1;
    #23 in = 0;
    #50 $finish;
end
…
```

```
Time =  0   clock = 0  reset = 1    in = 0    G = x  Y = x  R = x    Wait = x  Go = x
Time =  5   clock = 1  reset = 1    in = 0    G = x  Y = x  R = x    Wait = x  Go = x
Time = 10   clock = 0  reset = 0    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 15   clock = 1  reset = 0    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 20   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 25   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 30   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 35   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 40   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 43   clock = 0  reset = 1    in = 1    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 45   clock = 1  reset = 1    in = 1    G = 0  Y = 1  R = 0    Wait = 1  Go = 0
Time = 50   clock = 0  reset = 1    in = 1    G = 0  Y = 1  R = 0    Wait = 1  Go = 0
Time = 55   clock = 1  reset = 1    in = 1    G = 0  Y = 0  R = 1    Wait = 0  Go = 1
Time = 60   clock = 0  reset = 1    in = 1    G = 0  Y = 0  R = 1    Wait = 0  Go = 1
Time = 65   clock = 1  reset = 1    in = 1    G = 0  Y = 0  R = 1    Wait = 0  Go = 1
Time = 66   clock = 1  reset = 1    in = 0    G = 0  Y = 0  R = 1    Wait = 0  Go = 1
Time = 70   clock = 0  reset = 1    in = 0    G = 0  Y = 0  R = 1    Wait = 0  Go = 1
Time = 75   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 80   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 85   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 90   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 95   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 100   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 105   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 110   clock = 0  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
Time = 115   clock = 1  reset = 1    in = 0    G = 1  Y = 0  R = 0    Wait = 1  Go = 0
```



Ευχαριστώ για την προσοχή σας!

- **Ερωτήσεις / Απορίες ;**



Επικοινωνία: ece119.uth@gmail.com