# HY220
# Εργαστήριο Ψηφιακών Κυκλωμάτων

## Χειμερινό Εξάμηνο
## 2019-2020

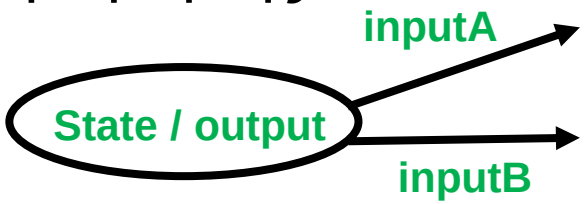
## Μηχανές Πεπερασμένων Καταστάσεων

# FSMs

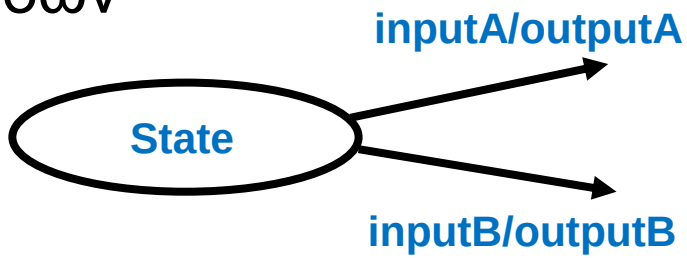
- Οι μηχανές πεπερασμένων καταστάσεων Finite State Machines (FSMs)
  - πιο αφηρημένος τρόπος να εξετάζουμε ακολουθιακά κυκλώματα
  - είσοδοι, έξοδοι, τρέχουσα κατάσταση, επόμενη κατάσταση
  - σε κάθε ακμή του ρολογιού συνδυαστική λογική παράγει τις εξόδους και την επόμενη κατάσταση σαν συναρτήσεις των εισόδων και της τρέχουσας κατάστασης.

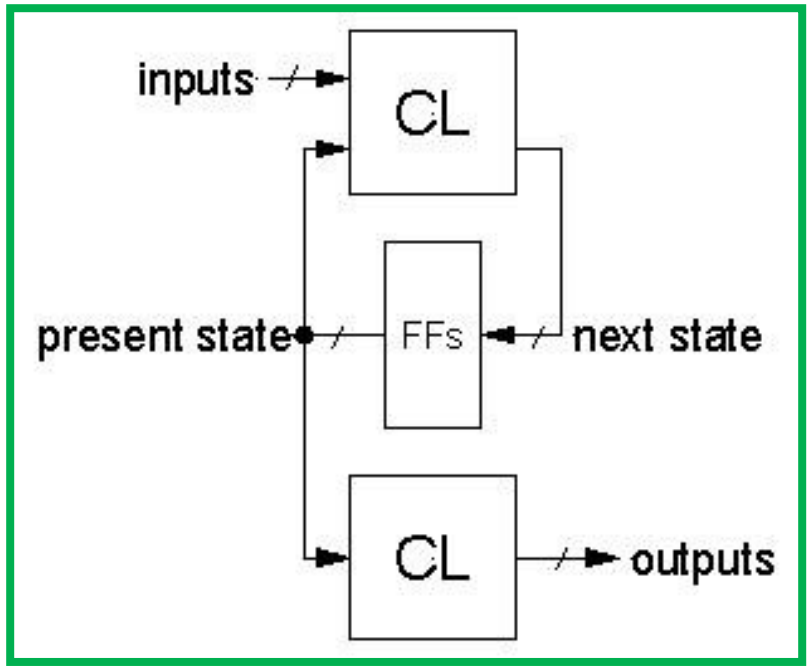
# Χαρακτηριστικά των FSM

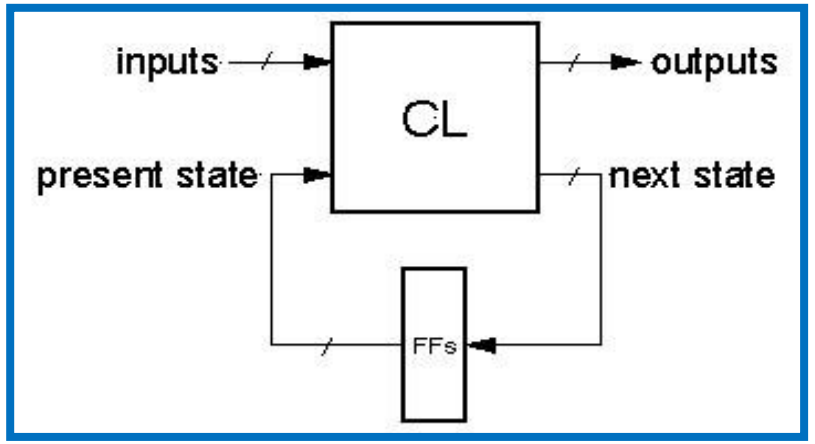
- Η επόμενη κατάσταση είναι συνάρτηση της τρέχουσας κατάστασης και των εισόδων
- Moore Machine: Οι έξοδοι είναι συνάρτηση της κατάστασης




- Mealy Machine: Οι έξοδοι είναι συνάρτηση της κατάστασης και των εισόδων



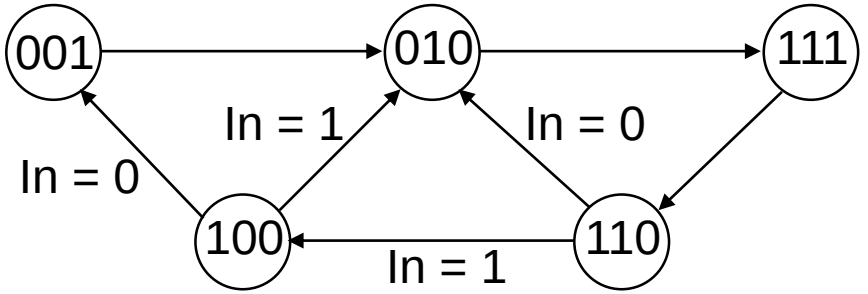



# Βήματα Σχεδίασης

- Περιγραφή λειτουργία του κυκλώματος (functional specification)
- Διάγραμμα μετάβασης καταστάσεων (state transition diagram)
- Πίνακας καταστάσεων και μεταβάσεων με συμβολικά ονόματα (symbolic state transition table)
- Κωδικοποίηση καταστάσεων (state encoding)
- Εξαγωγή λογικών συναρτήσεων
- Διάγραμμα κυκλώματος
  - FFs για την κατάσταση
  - ΣΛ για την επόμενη κατάσταση και τις εξόδους

# Αναπαράσταση FSM

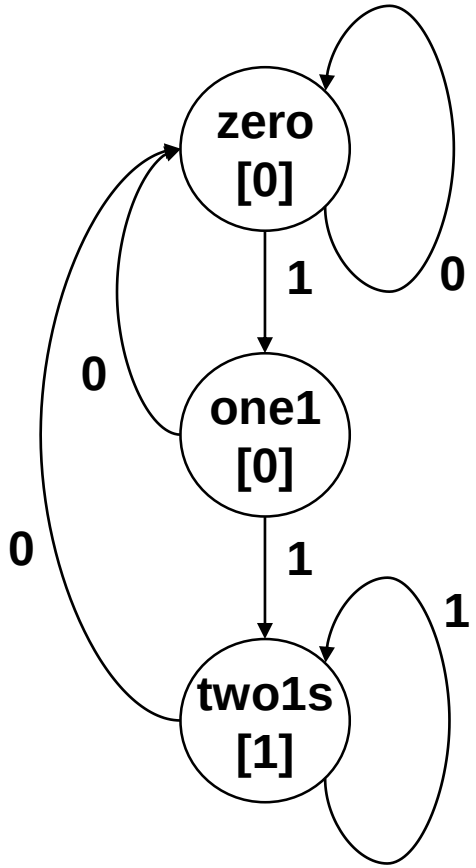
- Καταστάσεις: όλες οι πιθανές τιμές στα ακολουθιακά στοιχεία μνήμης (FFs)
- Μεταβάσεις: αλλαγή κατάστασης
- Αλλαγή τις κατάστασης με το ρολόι αφού ελέγχει την φόρτωση τιμής στα στοιχεία μνήμης (FFs)




- Ακολουθιακή λογική
  - Ακολουθία μέσω μιας σειράς καταστάσεων
  - Βασίζεται στην ακολουθία των τιμών στις εισόδους

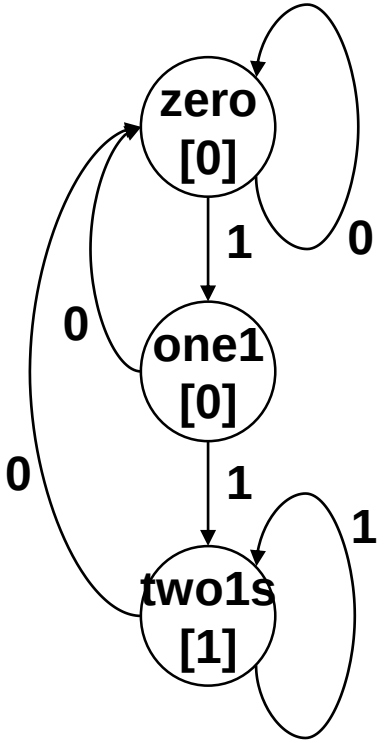
# Παράδειγμα FSM - Reduce 1s

- Αλλαγή του πρώτου 1 σε 0 σε μια σειρά από 1
  - Moore FSM



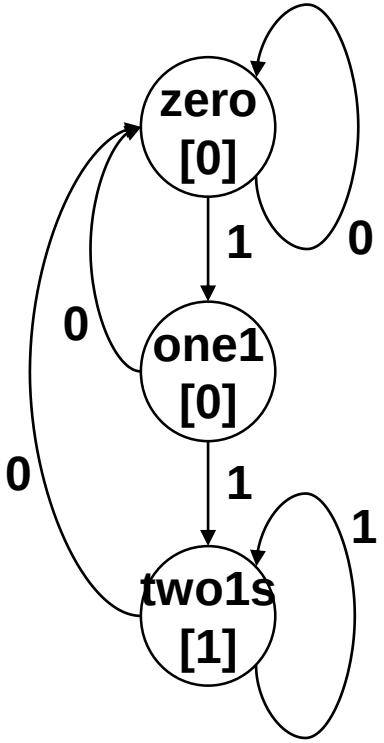
# Moore FSM: General & State

```
module ReduceMoore(
  input       Clk,
  input       Rst,
  input       In,
  output reg Out
);
reg [1:0]  CurrentState; // state reg
reg [1:0]  NextState;

// State assignment
parameter  STATE_Zero  =      2'h0,
           STATE_One1  =      2'h1,
           STATE_Two1s =      2'h2,
           STATE_X     =      2'hX;
// Implement the state register
always @( posedge Clk) begin
  if (Rst) CurrentState <= STATE_Zero;
  else     CurrentState <= NextState;
end
```



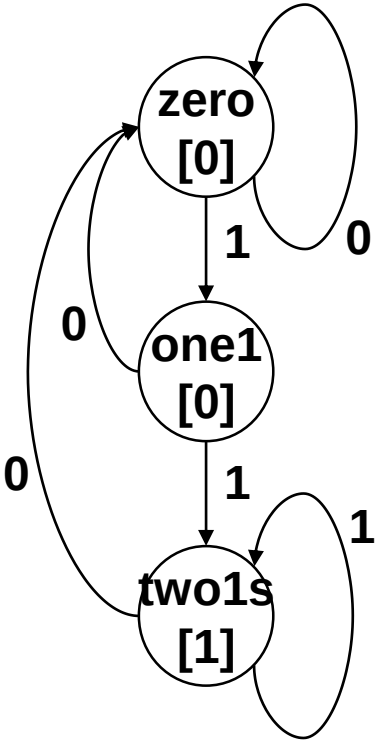
# Moore FSM: Combinatorial

```
always @(In or CurrentState) begin
  NextState =  CurrentState;
  Out = 1'b0;
  case (CurrentState)
    STATE_Zero: begin // last input was a zero
      if (In)  NextState =    STATE_One1;
    end
    STATE_One1: begin // we've seen one 1
      if (In)  NextState =    STATE_Two1s;
      else     NextState =    STATE_Zero;
    end
    STATE_Two1s: begin // we've seen at least 2 ones
      Out = 1;
      if (~In) NextState =    STATE_Zero;
    end
    default: begin // in case we reach a bad state
      Out = 1'bx;
      NextState =      STATE_Zero;
    end
  endcase
end
```



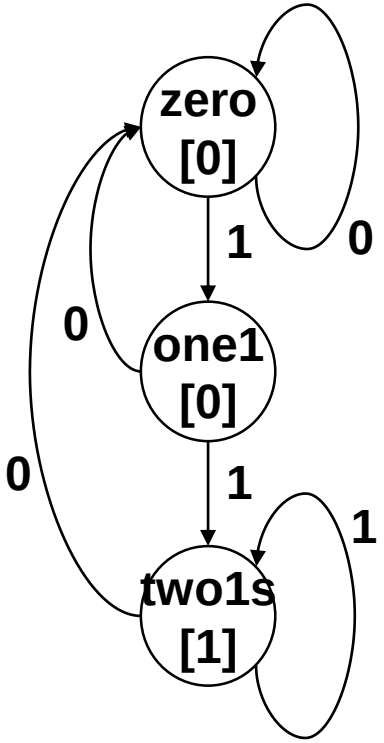
# Moore FSM: SystemVerilog Enums

```
module ReduceMooreSV(
  input  logic Clk,
  input  logic Rst,
  input  logic In,
  output logic Out
);
enum logic [1:0] {
  STATE_Zero  = 2'h0,
  STATE_One1  = 2'h1,
  STATE_Two1s = 2'h2 } CurrentState, NextState;
// alternative:
// typedef enum logic [1:0] {
// STATE_Zero, STATE_One1, STATE_Two1s } FSM_State_t;
// FSM_State_t CurrentState, NextState;

// Implement the state register
always_ff @( posedge Clk) begin
  if (Rst) CurrentState <= STATE_Zero;
  else     CurrentState <= NextState;
end
```



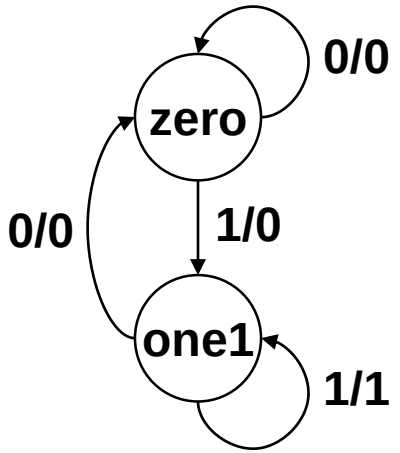
# Moore FSM: SystemVerilog Combinatorial

```
always_comb begin
  NextState =  CurrentState;
  Out = 1'b0;
  case (CurrentState)
    STATE_Zero: begin // last input was a zero
      if (In)  NextState = STATE_One1;
    end
    STATE_One1: begin // we've seen one 1
      if (In)  NextState = STATE_Two1s;
      else     NextState = STATE_Zero;
    end
    STATE_Two1s: begin // we've seen at least 2 ones
      Out = 1;
      if (~In) NextState = STATE_Zero;
    end
    default: begin // in case we reach a bad state
      Out = 1'bx;
      NextState = STATE_Zero;
    end
  endcase
end
```

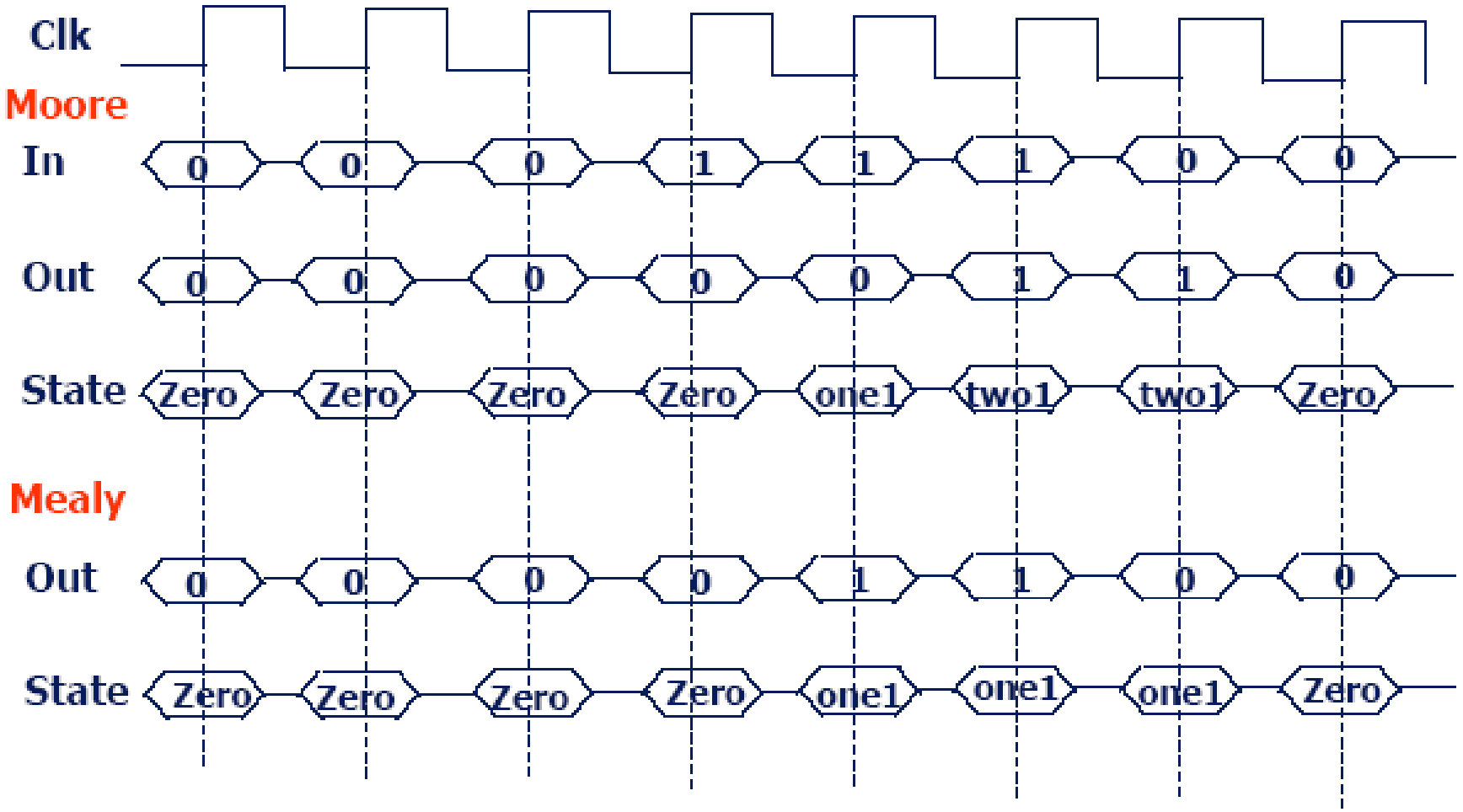


# Mealy FSM

```
module ReduceMealy( input Clk, input Rst, input In, output reg Out);

reg     CurrentState;// state register
reg     NextState;

parameter  STATE_Zero =       1'b0,
           STATE_One1 =       1'b1;

always @(posedge Clk) begin
  if (Rst) CurrentState <= STATE_Zero;
  else     CurrentState <= NextState;
end

always @ (In or CurrentState) begin
    NextState = CurrentState;
    Out = 1'b0;
    case (CurrentState)
        STATE_Zero: if (In)  NextState = STATE_One;
        STATE_One1: begin  // we've seen one 1
          if (In) NextState = STATE_One;
          else    NextState = STATE_Zero;
          Out = In;
        end
    endcase
end
endmodule
```



# Moore vs Mealy

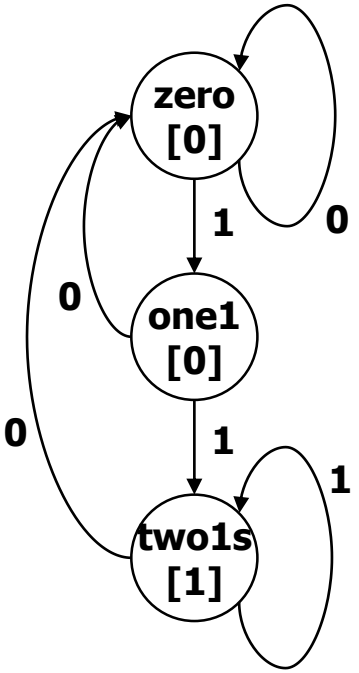


# Moore vs Mealy Συμπεριφορά

- Moore
  - απλοποιούν τη σχεδίαση
  - αδυναμία αντίδρασης στις εισόδους στον ίδιο κύκλο - έξοδοι ένα κύκλο μετά
  - διαφορετικές καταστάσεις για κάθε αντίδραση
- Mealy
  - συνήθως λιγότερες καταστάσεις
  - άμεση αντίδραση στις εισόδους – έξοδοι στον ίδιο κύκλο
  - δυσκολότερη σχεδίαση αφού καθυστερημένη είσοδος παράγει καθυστερημένη έξοδο (μεγάλα μονοπάτια)
- Η Mealy γίνεται Moore αν βάλουμε καταχωρητές στις εξόδους

# Moore Machine σε 1 always block (Bad Idea)

```
module ReduceMoore(
   input          Clk,
   input          Rst,
   input          In,
   output reg     Out
);

   reg [1:0] state;  // state register
   parameter zero = 0,
             one1 = 1,
             two1s = 2;
```



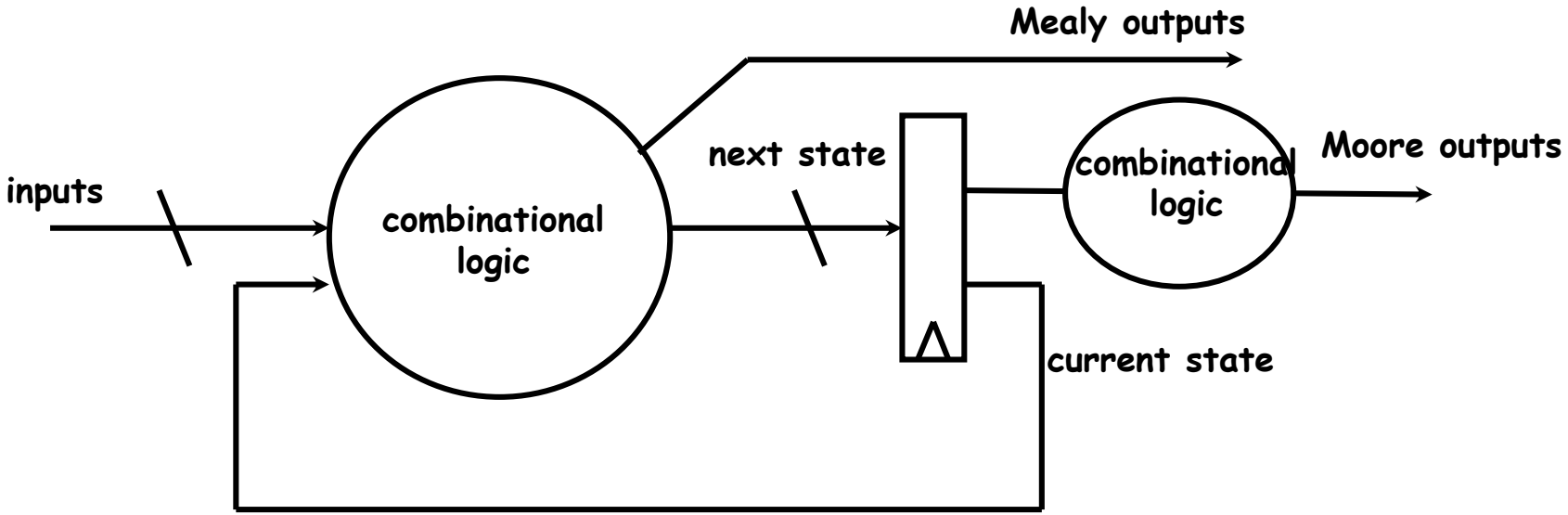
# Moore Machine σε 1 always block (Bad Idea)

```
always @(posedge clk)
  case (state)
   zero: begin
      out <= 0;
      if (in) state <= one1;
      else    state <= zero;
    end
    one1:
     if (in) begin
        state <= two1s;
        out <= 1;
     end else begin
        state <= zero;
        out <= 0;
     end
    two1s:
     if (in) begin
        state <= two1s;
        out <= 1;
     end else begin
        state <= zero;
        out <= 0;
     end
    default: begin
       state <= zero;
       out <= 0;
    end
  endcase
endmodule
```

**Οι έξοδοι είναι καταχωρητές**

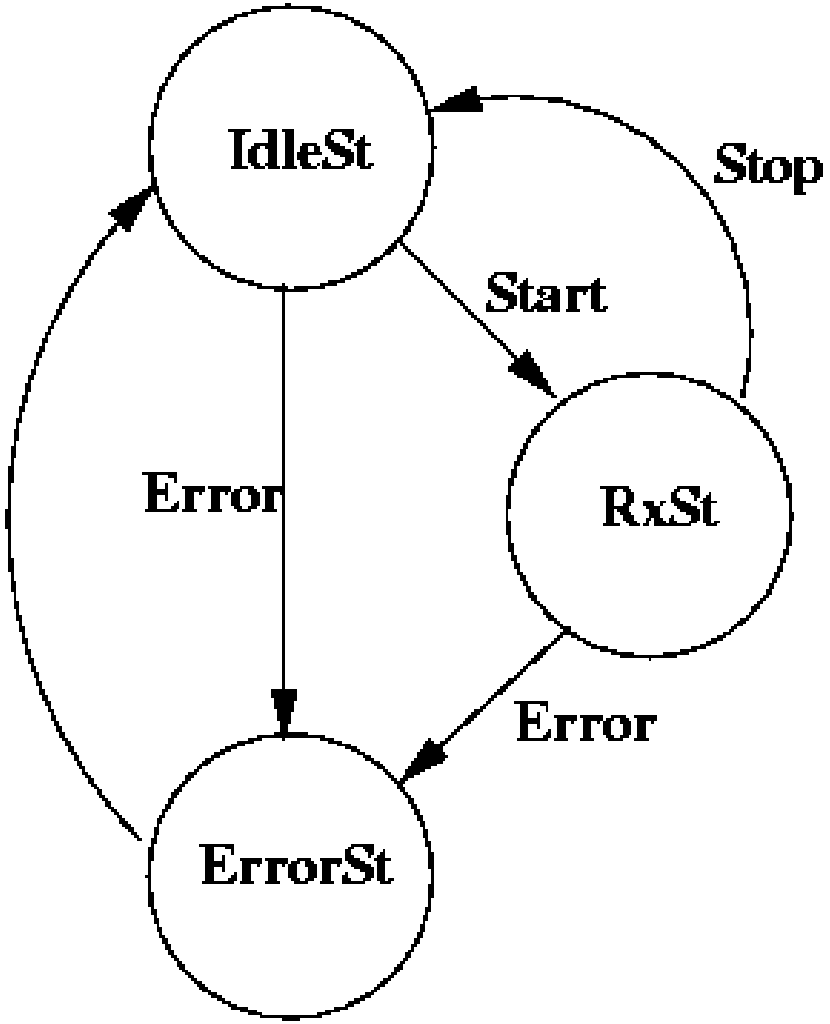
**Μπερδεμένο!!!**
**Η έξοδος αλλάζει στον επόμενο κύκλο**

# Υλοποίηση FSMs




- Προτεινόμενο στυλ υλοποίησης FSM
  - Η συνδυαστική λογική καταστάσεων σε always block (πάντα default)
  - Ο καταχωρητής κατάστασης σε ένα ξεχωριστό always block (clocked – πάντα reset)
  - Έξοδοι είτε από το always της CL είτε από wires

# Απλή FSM

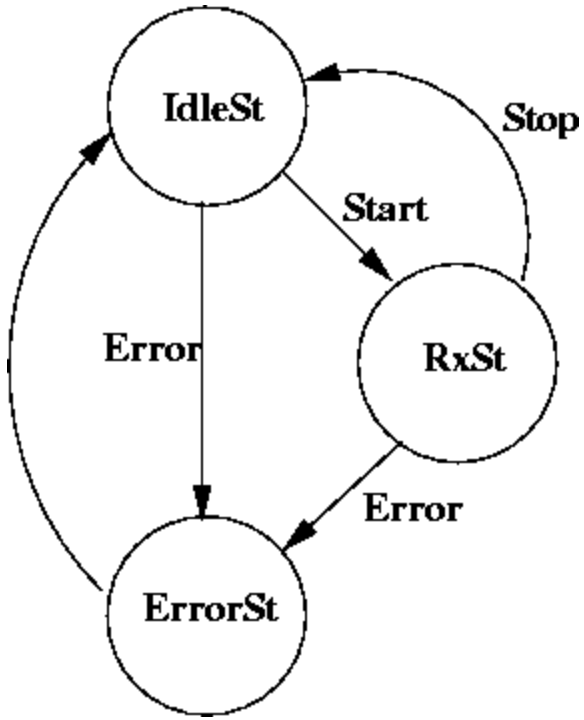


# Απλή FSM (1/3)

```
module fsm( Receive, Start, Stop,
            Error, Clk, Reset_);
//
input  Start, Stop, Error, Clk, Reset_n;
output Receive;
//
parameter [1:0] IdleState    = 0,
                ReceiveState = 1,
                ErrorState   = 2;
//
reg [1:0] FSMstate, nxtFSMstate;
//
always @(posedge Clk) begin
   if (~Reset_n) FSMstate <= #`dh IdleState;
   else          FSMstate <= #`dh nxtFSMstate;
end

//
always @(FSMstate or Start or Stop or Error) begin
//
   case(FSMstate)
```

# Απλή FSM (2/3)

```
    IdleState:
    begin
      if(Error)    nxtFSMstate <= ErrorState;
      else begin
        if(Start)  nxtFSMstate <= ReceiveState;
        else       nxtFSMstate <= IdleState;
      end
    end
//
    ReceiveState:
    begin
      if(Error)    nxtFSMstate <= ErrorState;
      else begin
        if(Stop)   nxtFSMstate <= IdleState;
        else       nxtFSMstate <= ReceiveState;
      end
    end
//
    ErrorState :   nxtFSMstate <= IdleState;
//
    default    :   nxtFSMstate <= IdleState;
//
  endcase
end
```



# Απλή FSM (3/3) – Οι έξοδοι

- The Moore Output

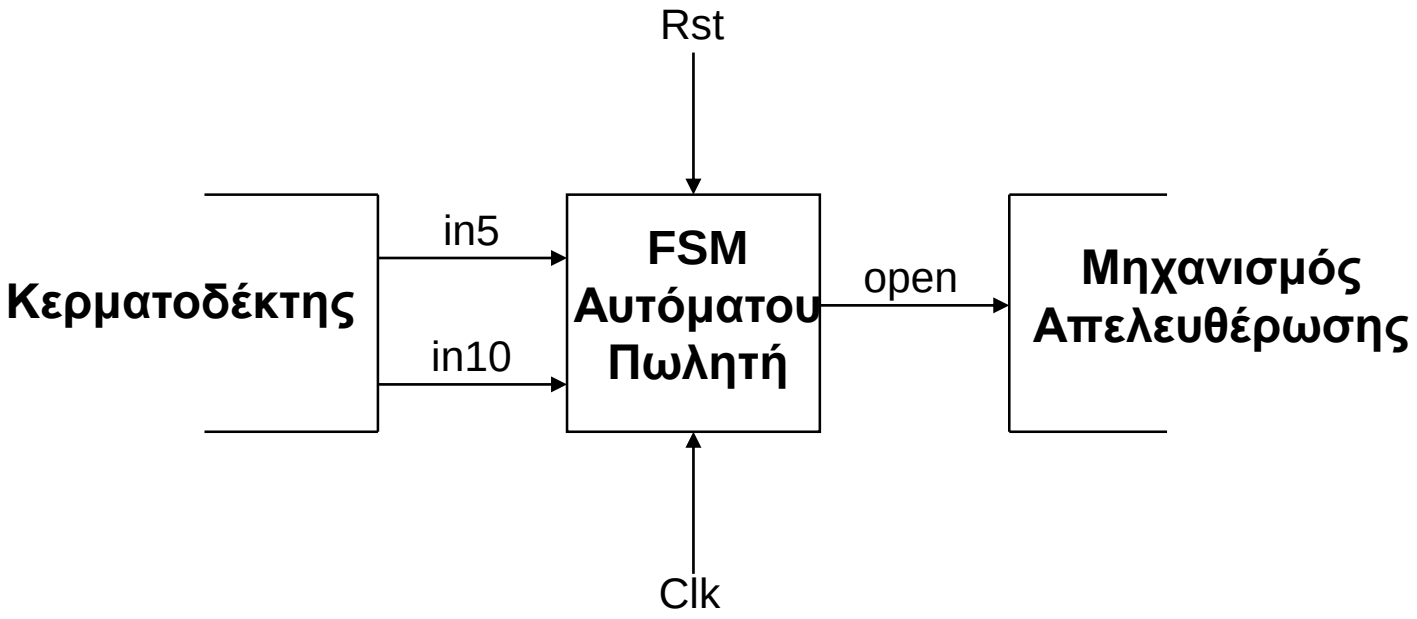
```
wire Receive = FSMstate[0];
```

- The Mealy Output

```
wire Receive =
       ((FSMstate == IdleState ) & Start) |
       ((FSMstate == ReceiveState) & ~Error & ~Stop );
```

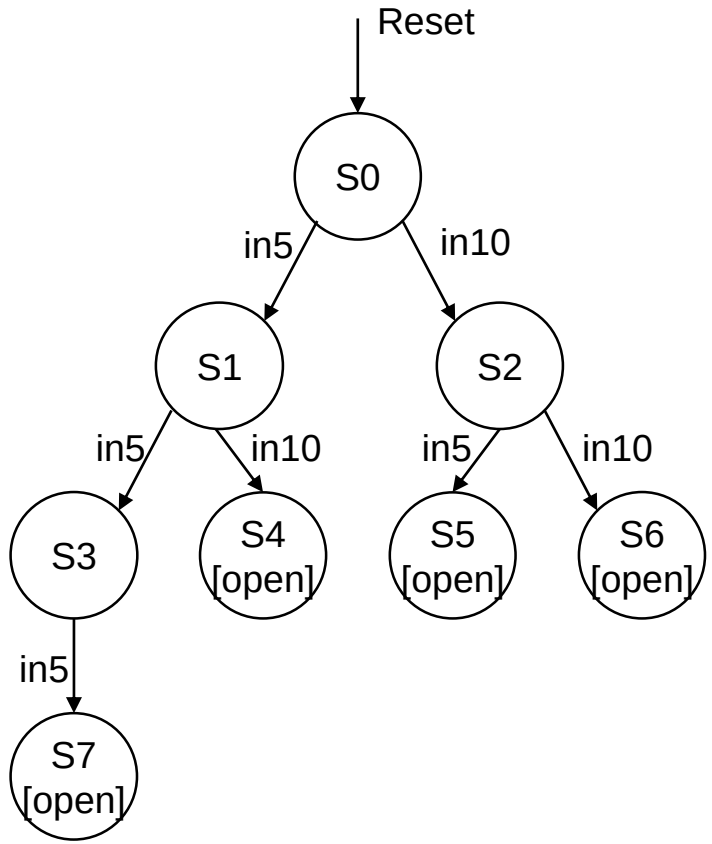
# Παράδειγμα: «Αυτόματος Πωλητής» (1/5)

- Βγάζει αναψυκτικό όταν βάλουμε 15 λεπτά του €
- Κερματοδέκτης για νομίσματα των 5 και 10 λεπτών του €
- Δεν δίνει ρέστα!



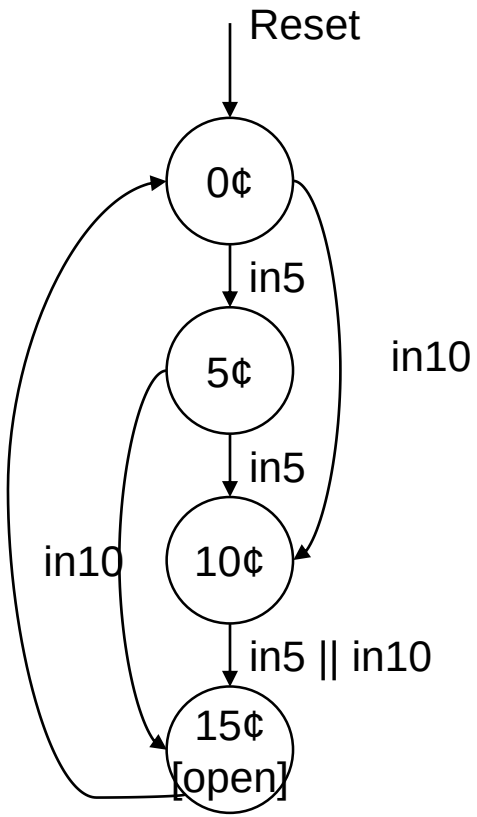
# Παράδειγμα: «Αυτόματος Πωλητής» (2/5)

- Αναπαράσταση
  - Τυπικές είσοδοι:
    - 3 των 5¢
    - 5¢, 10¢
    - 10¢, 5¢
    - 2 των 10¢
  - Διάγραμμα Καταστάσεων:
    - Είσοδοι: in5, in10, reset, clock
    - Έξοδοι: open
  - Assumptions:
    - in5 και in10 εμφανίζονται για 1 κύκλο
    - Μένουμε στην ίδια κατάσταση αν δεν έρθει είσοδος
    - Όταν έρθει reset πάμε στην αρχική κατάσταση



# Παράδειγμα: «Αυτόματος Πωλητής» (3/5)

- Ελαχιστοποίηση καταστάσεων - επαναχρησιμοποίηση




| present state | inputs in10 | inputs in5 | next state | output open |
|---|---|---|---|---|
| 0¢ | 0 | 0 | 0¢ | 0 |
| | 0 | 1 | 5¢ | 0 |
| | 1 | 0 | 10¢ | 0 |
| | 1 | 1 | – | – |
| 5¢ | 0 | 0 | 5¢ | 0 |
| | 0 | 1 | 10¢ | 0 |
| | 1 | 0 | 15¢ | 0 |
| | 1 | 1 | – | – |
| 10¢ | 0 | 0 | 10¢ | 0 |
| | 0 | 1 | 15¢ | 0 |
| | 1 | 0 | 15¢ | 0 |
| | 1 | 1 | – | – |
| 15¢ | – | – | 0¢ | 1 |

symbolic state table

# Παράδειγμα: «Αυτόματος Πωλητής» (4/5)

- Κωδικοποίηση Καταστάσεων – Τυπική

| pres. state | | inputs | | next state | | output |
|---|---|---|---|---|---|---|
| Q1 | Q0 | in10 | in5 | D1 | D0 | open |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 0 | 1 | 0 | 1 | 0 |
| | | 1 | 0 | 1 | 0 | 0 |
| | | 1 | 1 | – | – | – |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| | | 0 | 1 | 1 | 0 | 0 |
| | | 1 | 0 | 1 | 1 | 0 |
| | | 1 | 1 | – | – | – |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | 0 | 1 | 1 | 1 | 0 |
| | | 1 | 0 | 1 | 1 | 0 |
| | | 1 | 1 | – | – | – |
| 1 | 1 | – | – | 0 | 0 | 1 |

# Παράδειγμα: «Αυτόματος Πωλητής» (5/5)

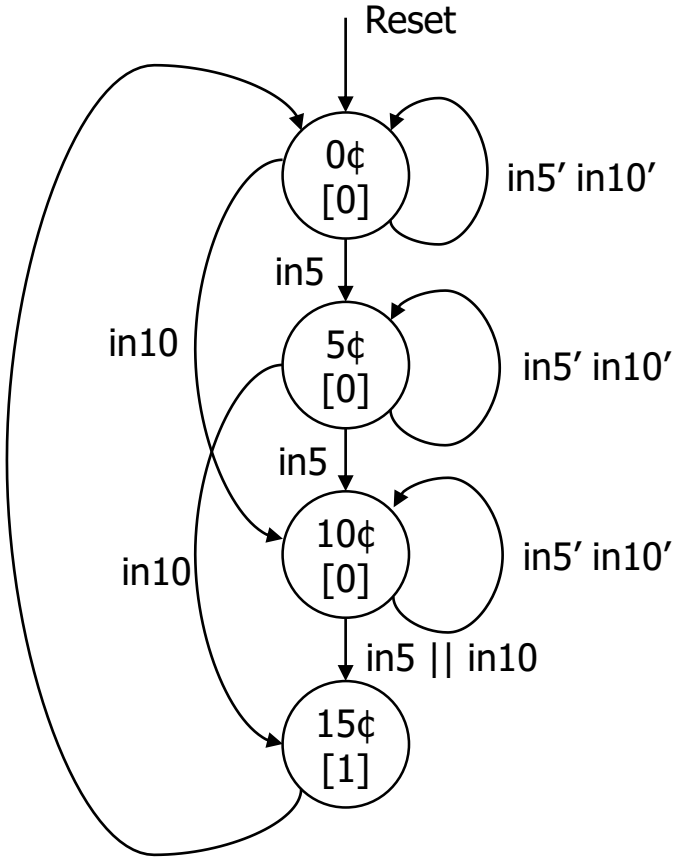
- Κωδικοποίηση Καταστάσεων – One-hot

| present state | | | | inputs | | next state | | | | output |
|---|---|---|---|---|---|---|---|---|---|---|
| Q3 | Q2 | Q1 | Q0 | in10 | in5 | D3 | D2 | D1 | D0 | open |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | | | | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| | | | | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| | | | | 1 | 1 | - | - | - | - | - |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | | | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| | | | | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| | | | | 1 | 1 | - | - | - | - | - |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| | | | | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| | | | | 1 | 1 | - | - | - | - | - |
| 1 | 0 | 0 | 0 | - | - | 0 | 0 | 0 | 1 | 1 |

# Διαγράμματα καταστάσεων – Moore and Mealy

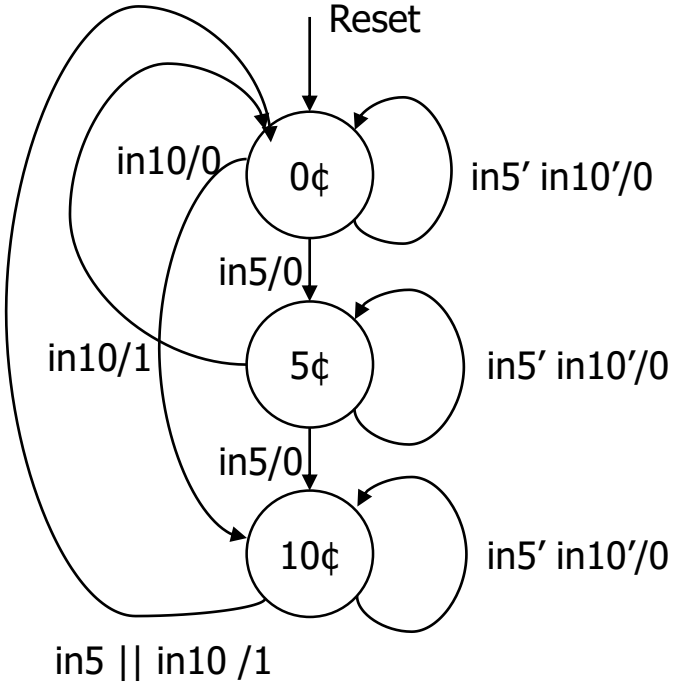
## Moore machine

Έξοδοι από κατάσταση



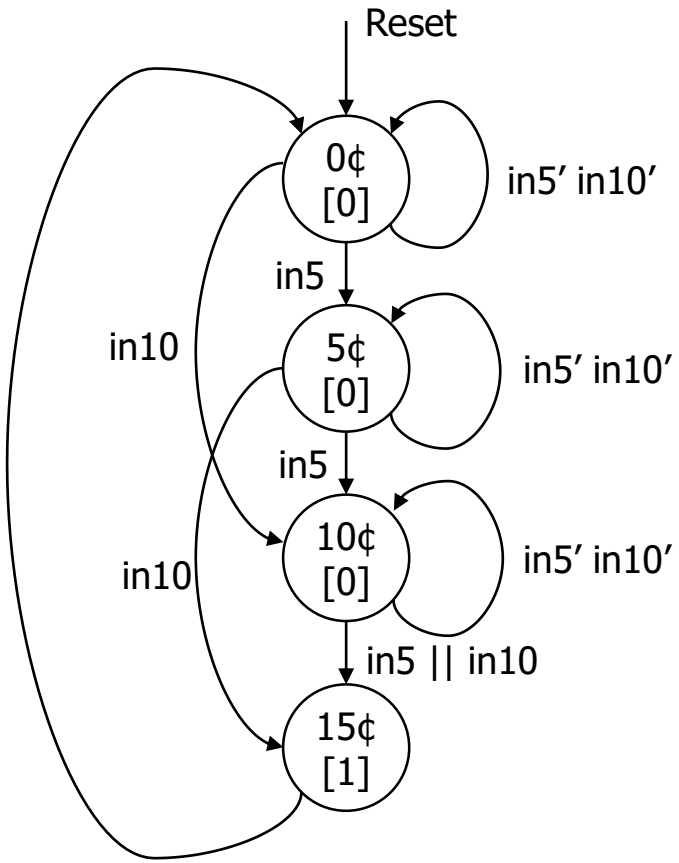

## Mealy machine

Έξοδοι στις μεταβάσεις



# Moore Verilog FSM

```
module vending (open, clk, Rst, in5, in10);
  input clk, Rst, in5, in10;
  output open;
  reg open; reg [1:0] state;      // state register
  reg [1:0] next_state;
  parameter zero = 0, five = 1, ten = 2, fifteen = 3;

  always @(in5 or in10 or state)
    case (state)
      zero: begin
              if (in5)       next_state = five;
              else if (in10) next_state = ten;
              else           next_state = zero;
              open = 0;
      end
      …
      fifteen: begin
                 next_state = zero;
                 open = 1;
      end
      default: begin
                 next_state = zero;
                 open = 0;
      end
    endcase

  always @(posedge clk)
    if (Rst) state <= zero;
    else     state <= next_state;

endmodule
```



# Mealy Verilog FSM

```
module vending (open, Clk, Rst, in5, in10);
  input Clk, Rst, in5, in10;
  output open;
  reg open; reg [1:0] state;       // state register
  reg [1:0] next_state;
  parameter zero = 0, five = 1, ten = 2, fifteen = 3;

  always @(in5 or in10 or state)
    case (state)
      zero: begin
              open = 0;
              if (in10)       next_state = ten;
              else if (in5) next_state = five;
              else             next_state = zero;
      end
      five: begin
             if (in5) begin
               next_state = ten;
               open = 0;
             end
             else if (in10 ) begin
               next_state = zero;
               open = 1´;
             end
             else begin
               next_state = five;
               open = 0;
             end
      end
      …
    endcase

  always @(posedge clk)
    if (Rst) state <= zero;
    else     state <= next_state;
endmodule
```