# HY220
# Εργαστήριο Ψηφιακών Κυκλωμάτων

## Εαρινό Εξάμηνο
## 2025

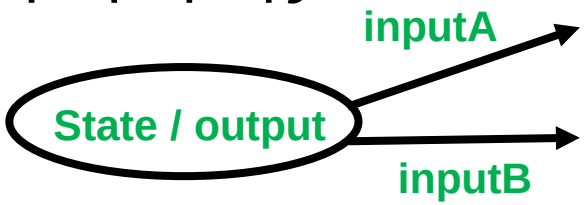
## Μηχανές Πεπερασμένων Καταστάσεων

# FSMs

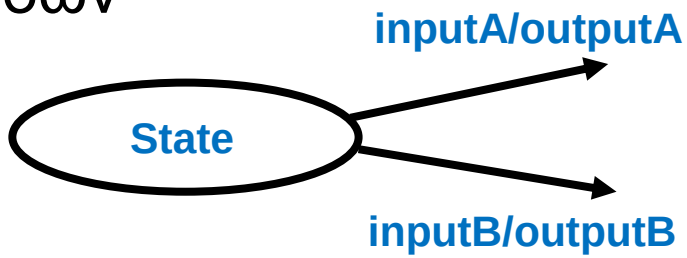
- Οι μηχανές πεπερασμένων καταστάσεων Finite State Machines (FSMs)
  - πιο αφηρημένος τρόπος να εξετάζουμε ακολουθιακά κυκλώματα
  - είσοδοι, έξοδοι, τρέχουσα κατάσταση, επόμενη κατάσταση
  - σε κάθε ακμή του ρολογιού συνδυαστική λογική παράγει τις εξόδους και την επόμενη κατάσταση σαν συναρτήσεις των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

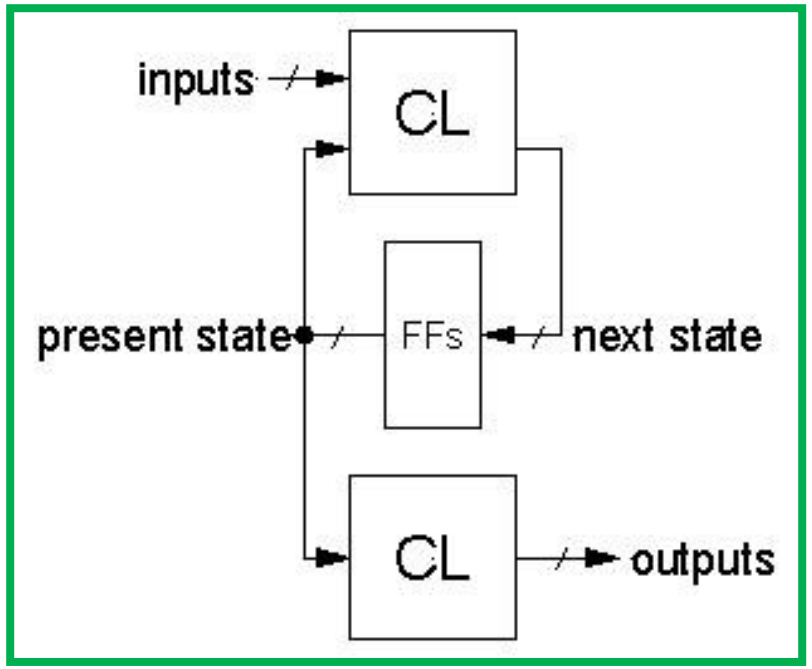
# Χαρακτηριστικά των FSM

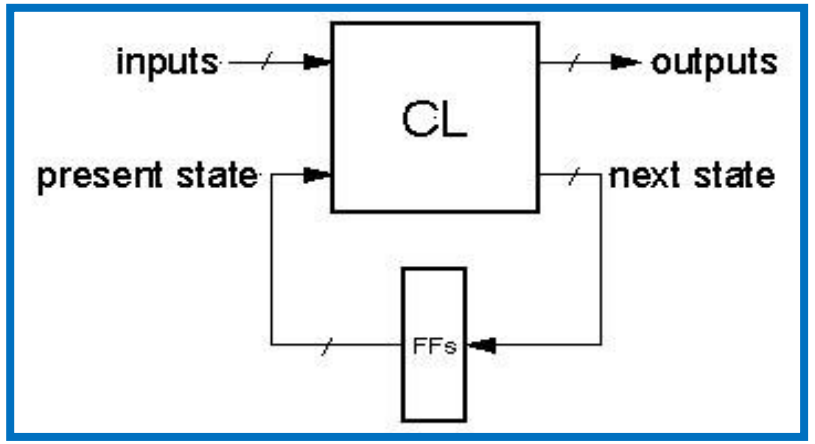
- Η επόμενη κατάσταση είναι συνάρτηση της τρέχουσας κατάστασης και των εισόδων
- Moore Machine: Οι έξοδοι είναι συνάρτηση της κατάστασης




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



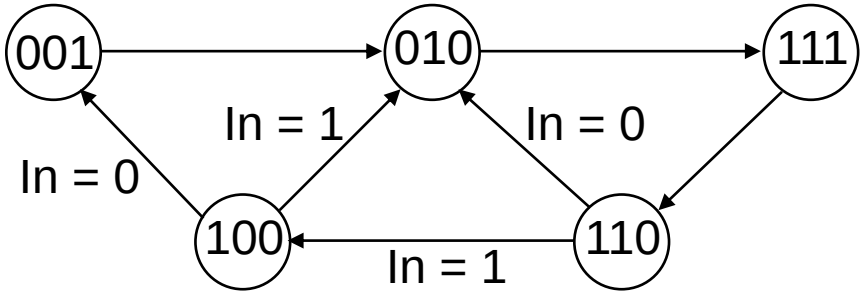



# Βήματα Σχεδίασης

- 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

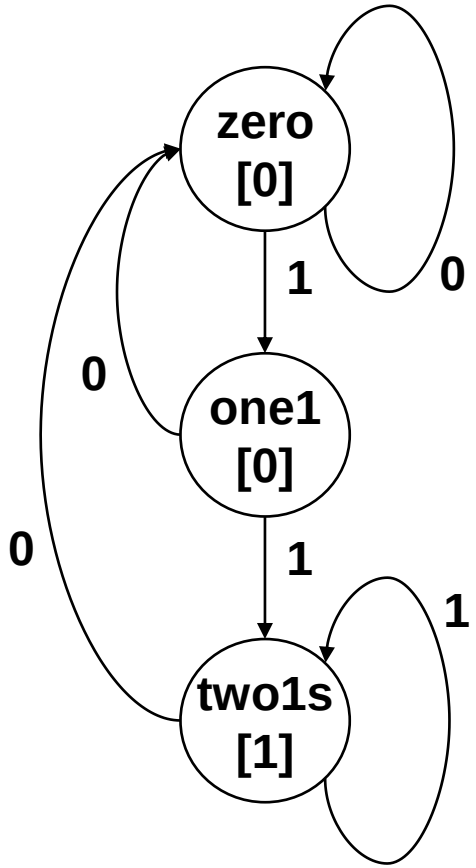
- Καταστάσεις: όλες οι πιθανές τιμές στα ακολουθιακά στοιχεία μνήμης (FFs)
- Μεταβάσεις: αλλαγή κατάστασης
- Αλλαγή τις κατάστασης με το ρολόι αφού ελέγχει την φόρτωση τιμής στα στοιχεία μνήμης (FFs)




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

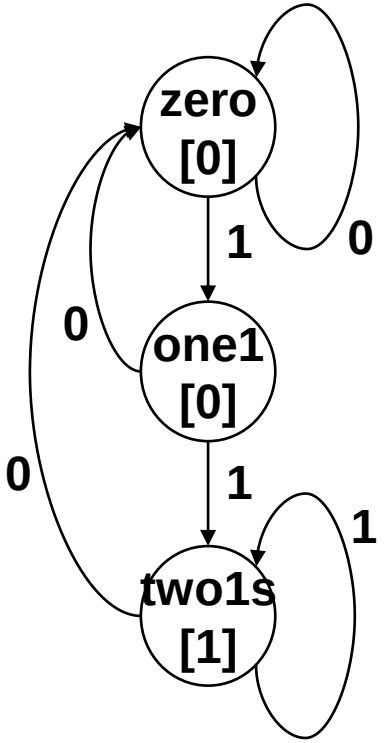
# Παράδειγμα FSM - Reduce 1s

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



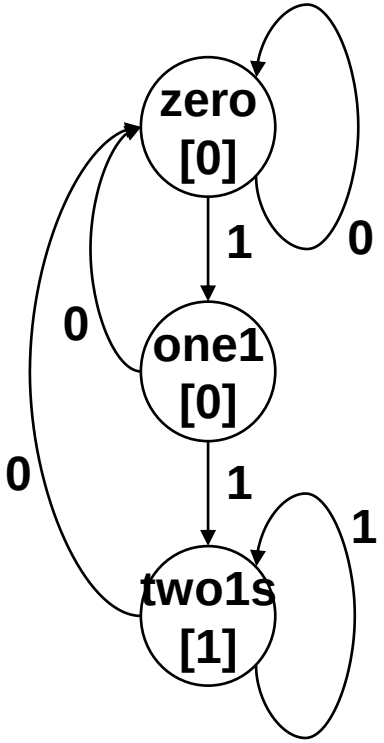
# Moore FSM: General & State

```
module ReduceMoore(
  input logic  clk,
  input logic  rst,
  input logic  in,
  output logic out
);
logic [1:0]    current_state; // state register
logic [1:0]    next_state;

// State declarations
parameter int  STATE_Zero  =  2'h0,
               STATE_One1  =  2'h1,
               STATE_Two1s =  2'h2,
               STATE_X     =  2'hX;
// Implement the state register
always_ff @( posedge clk) begin
  if (rst) current_state <= STATE_Zero;
  else     current_state <= next_state;
end
```



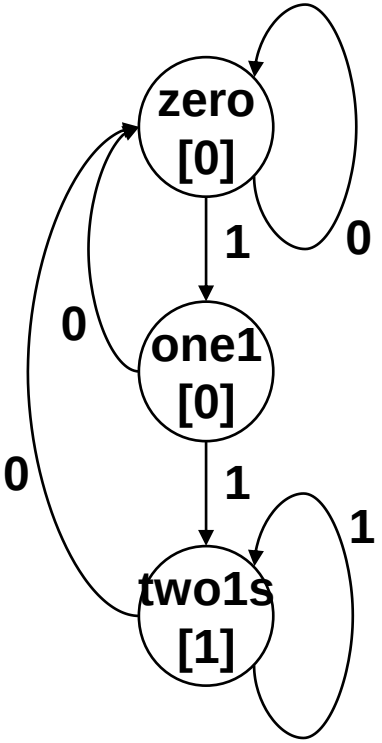
# Moore FSM: Combinatorial

```
always_comb begin
  next_state = current_state; // acts as the default/else
  out = 1'b0;                 // default output
  case (current_state)
    STATE_Zero: begin          // last input was a zero
      if (in)  next_state = STATE_One1;
    end
    STATE_One1: begin          // we've seen one 1
      if (in)  next_state = STATE_Two1s;
      else     next_state = STATE_Zero;
    end
    STATE_Two1s: begin         // we've seen at least 2 ones
      out = 1;
      if (~in) next_state = STATE_Zero;
    end
    default: begin             // in case we reach a bad state
      out = 1'bx;
      next_state = STATE_Zero;
    end
  endcase
end
```



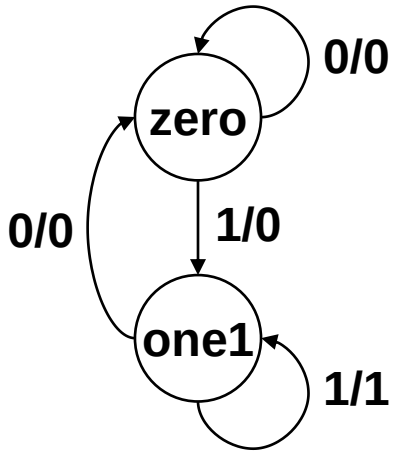
# Moore FSM: SystemVerilog Enums

```
module ReduceMooreEnums(
  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 } current_state, next_state;
// alternative:
// typedef enum logic [1:0] {
// STATE_Zero, STATE_One1, STATE_Two1s } FSM_State_e;
// FSM_State_e current_state, next_state;

// Implement the state register
always_ff @( posedge clk) begin
  if (rst) current_state <= STATE_Zero;
  else     current_state <= next_state;
end
```

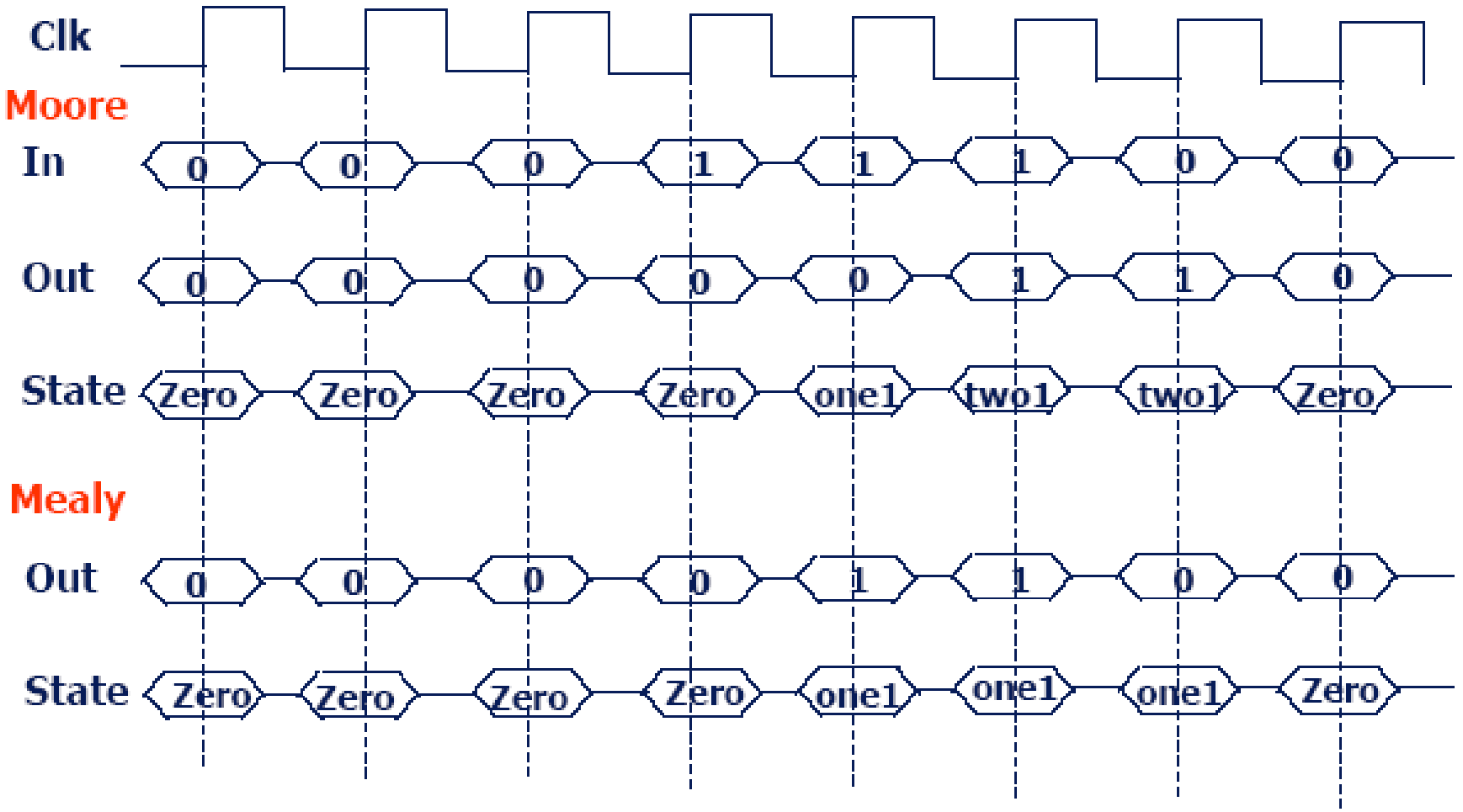


# Mealy FSM

```
module ReduceMealy( input logic clk, rst, in, output logic out);

logic   current_state; // state register
logic   next_state;

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

always_ff @(posedge clk) begin
  if (rst) current_state <= STATE_Zero;
  else     current_state <= next_state;
end

always_comb begin
    next_state = current_state;
    out = 1'b0;
    case (current_state)
        STATE_Zero: if (in) next_state = STATE_One;
        STATE_One1: begin   // we've seen one 1
          if (in)           next_state = STATE_One;
          else              next_state = STATE_Zero;
          Out = In;
        end
    endcase
end

endmodule
```



# Moore vs Mealy

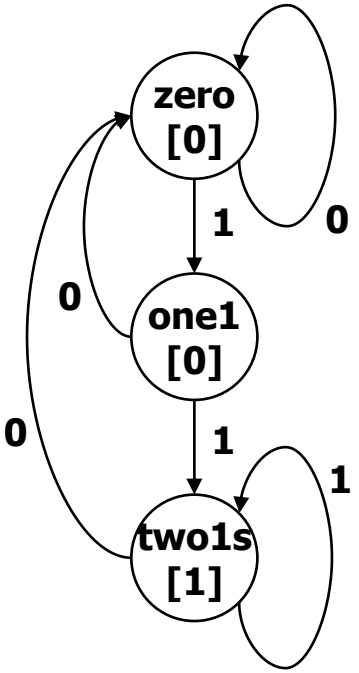


# Moore vs Mealy Συμπεριφορά

- 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  logic  clk,
  input  logic  rst,
  input  logic  in,
  output logic  out
);

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



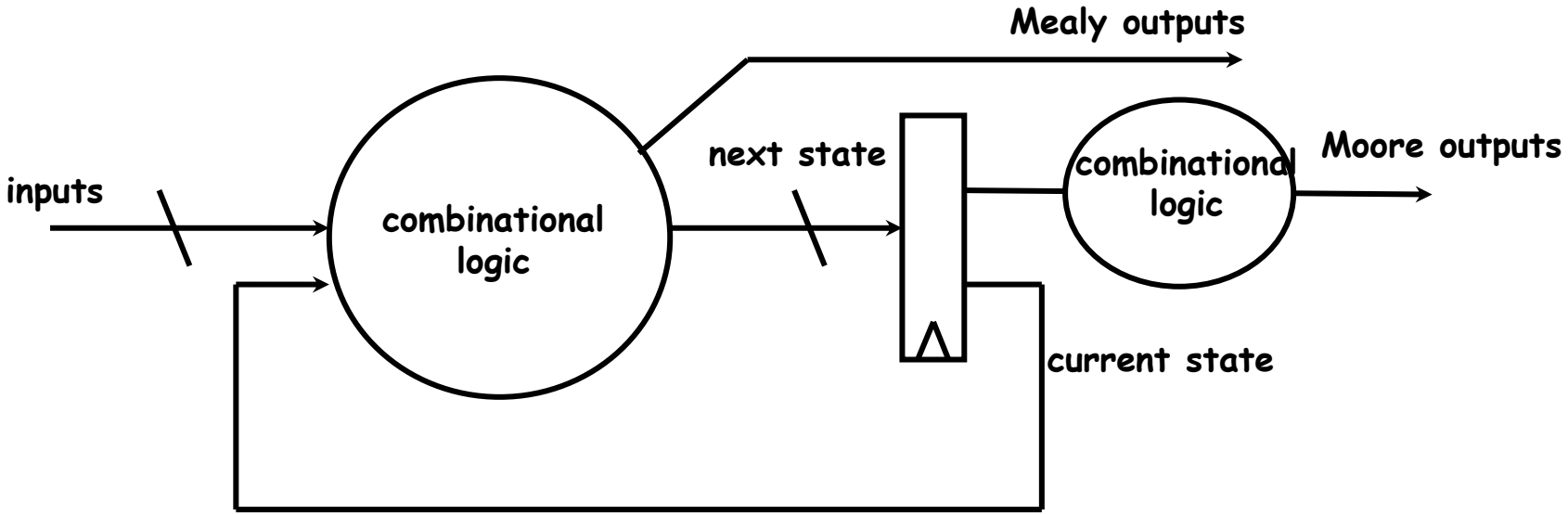
# Moore Machine σε 1 always block (Bad Idea)

```
always_ff @(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**

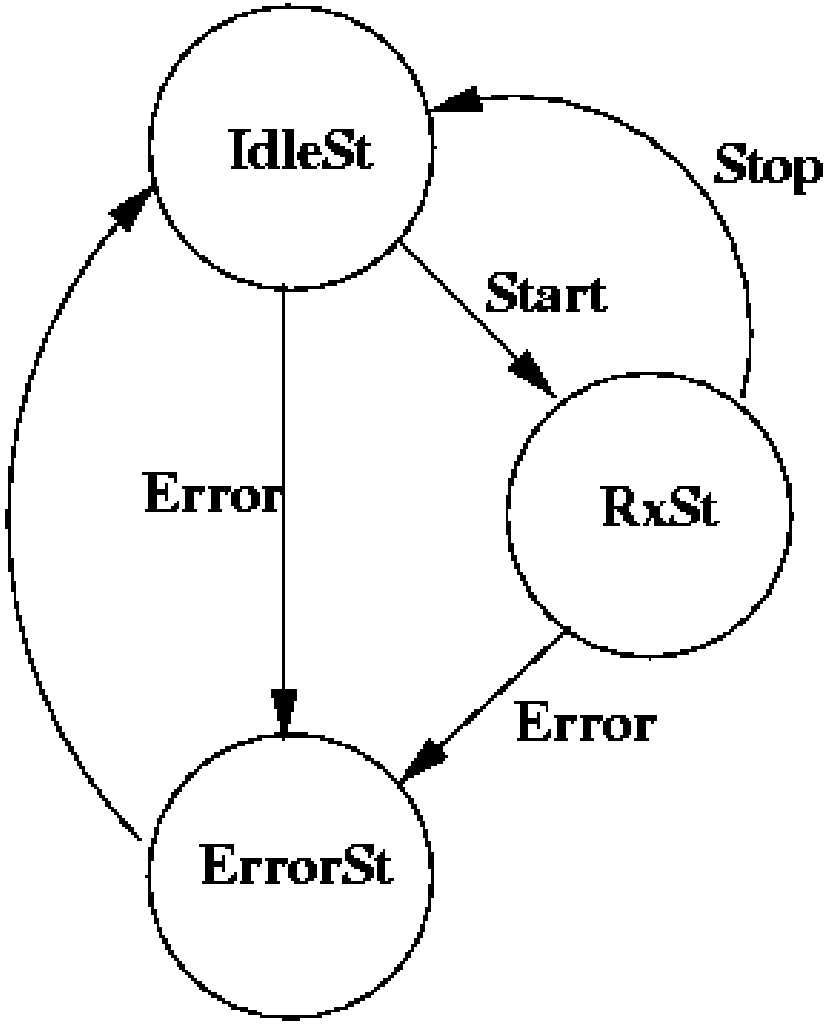
**Μπερδεμένο!!!**
**Η έξοδος αλλάζει στον επόμενο κύκλο**

# Υλοποίηση FSMs




- Προτεινόμενο στυλ υλοποίησης FSM
  - Ο καταχωρητής κατάστασης σε ένα ξεχωριστό always_ff
    - clocked – πάντα reset – χρήση μόνο non-blocking assignment
  - Η συνδυαστική λογική αλλαγής καταστάσεων σε always_comb
    - πάντα default – χρήση μόνο blocking assignments
  - Έξοδοι είτε από always_comb της CL είτε από wires
    - χρήση μόνο blocking assignments *(και σιγουρευτείτε για τις default τιμές)*

# Απλή FSM

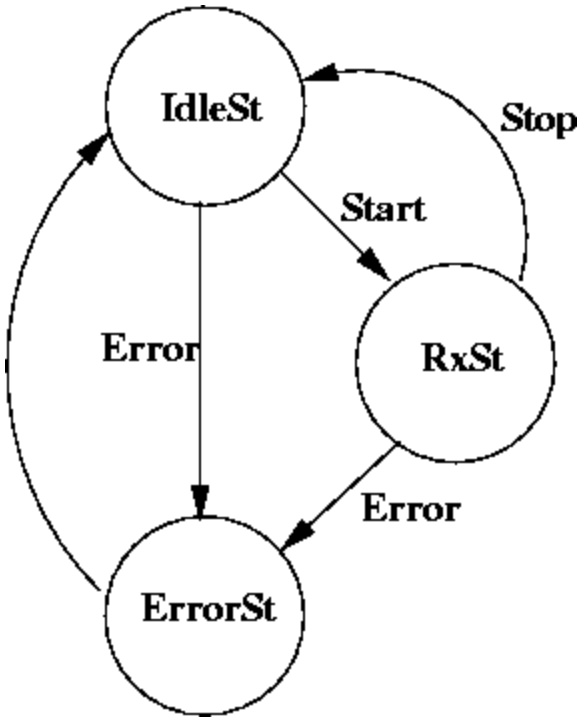


# Απλή FSM (1/3)

```
module fsm( receive, start, stop,
            error, clk, reset_n);
//
parameter int C2Q = 1;
//
input  logic start, stop, error, clk, reset_n;
output logic receive;
//
parameter logic [1:0] IdleState    = 0,
                      ReceiveState = 1,
                      ErrorState   = 2;
//
logic [1:0] fsm_state, fsm_nxtstate;
//
always_ff @(posedge clk) begin
   if (~reset_n) fsm_state <= #C2Q IdleState;
   else          fsm_state <= #C2Q fsm_nxtstate;
end
```

# Απλή FSM (2/3)

```
always_comb begin
 case(fsm_state)
   IdleState:
   begin
     if(error)    fsm_nxtstate = ErrorState;
     else begin
       if(start)  fsm_nxtstate = ReceiveState;
       else       fsm_nxtstate = IdleState;
     end
   end
   ReceiveState:
   begin
     if(error)    fsm_nxtstate = ErrorState;
     else begin
       if(stop)   fsm_nxtstate = IdleState;
       else       fsm_nxtstate = ReceiveState;
     end
   end
   ErrorState:    fsm_nxtstate = IdleState;
   default:       fsm_nxtstate = IdleState;
  endcase
end
```



# Απλή FSM (3/3) – Οι έξοδοι

- The Moore Output

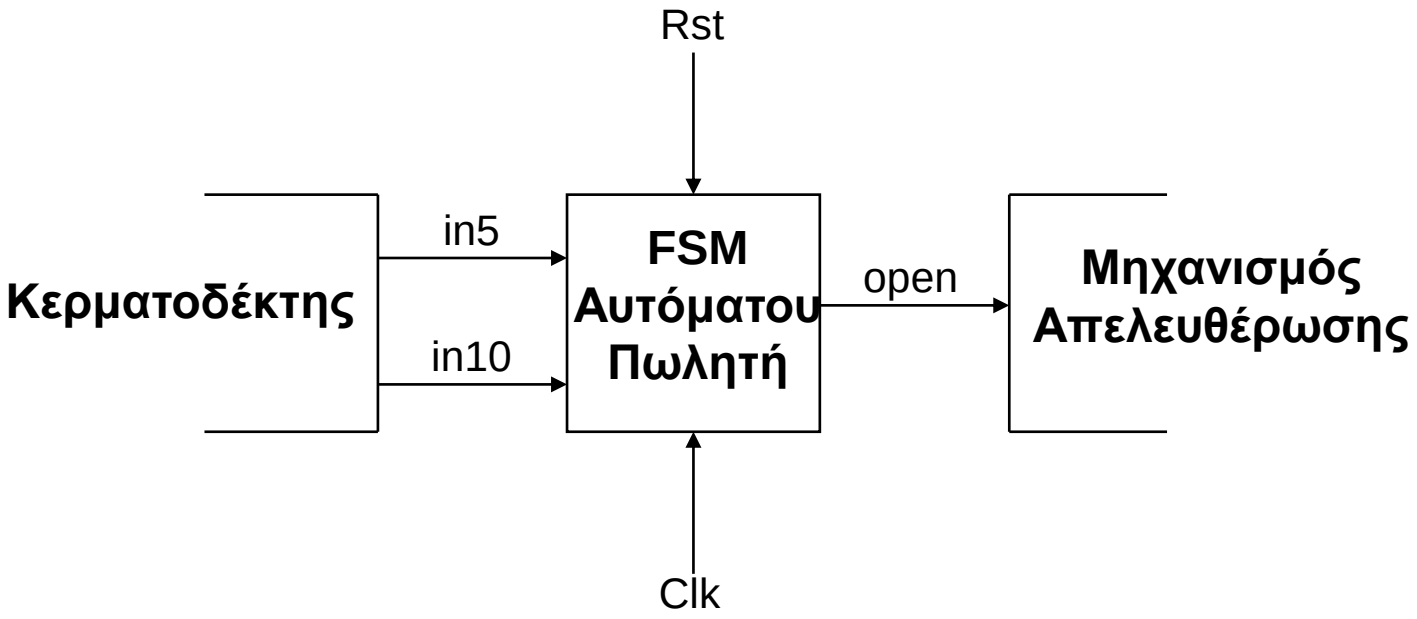
```
assign receive = fsm_state[0];
// alternative:
// assign receive = (fsm_state == ReceiveState);
```

- The Mealy Output

```
assign receive = ((fsm_state == IdleState )    & start) |
                 ((fsm_state == ReceiveState) & ~error & ~stop );

// alternative: with blocking assignments
// always_comb begin
//    receive = 0;  // default value for output
//    if ((fsm_state == IdleState )    & start)             receive = 1;
//    if ((fsm_state == ReceiveState) & ~error & ~stop ) receive = 1;
// end
```

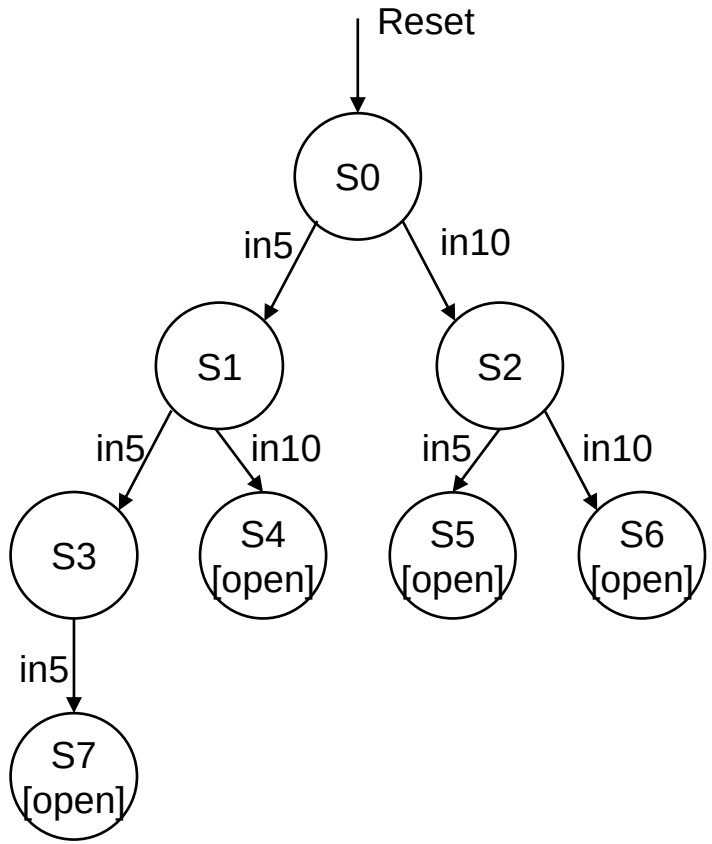
# Παράδειγμα: «Αυτόματος Πωλητής» (1/5)

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



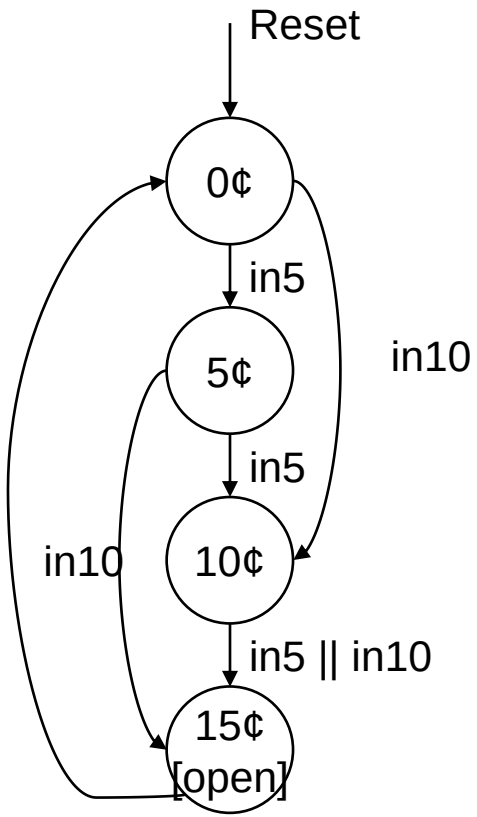
# Παράδειγμα: «Αυτόματος Πωλητής» (2/5)

- 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)

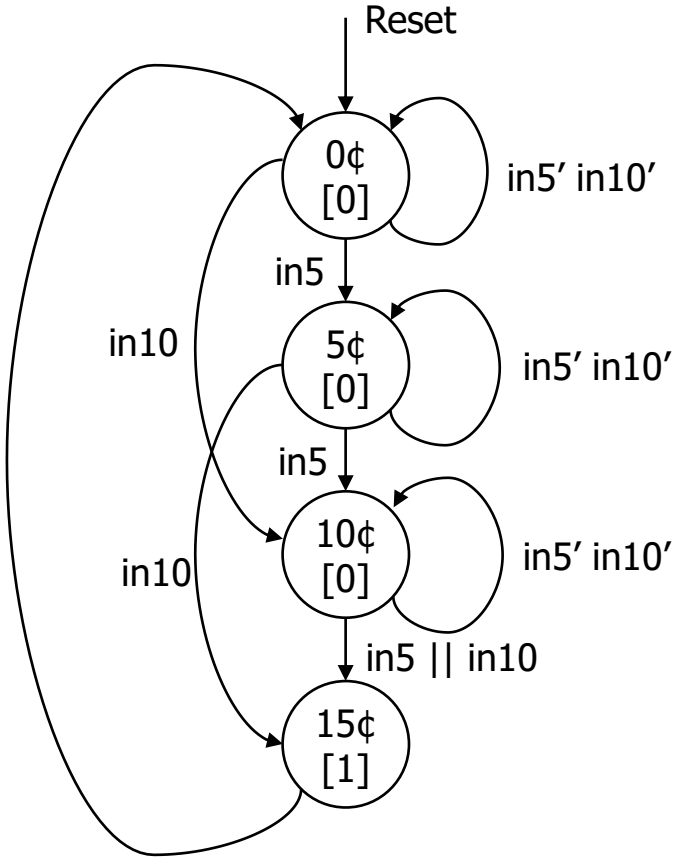
- Κωδικοποίηση Καταστάσεων – One-hot

| 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

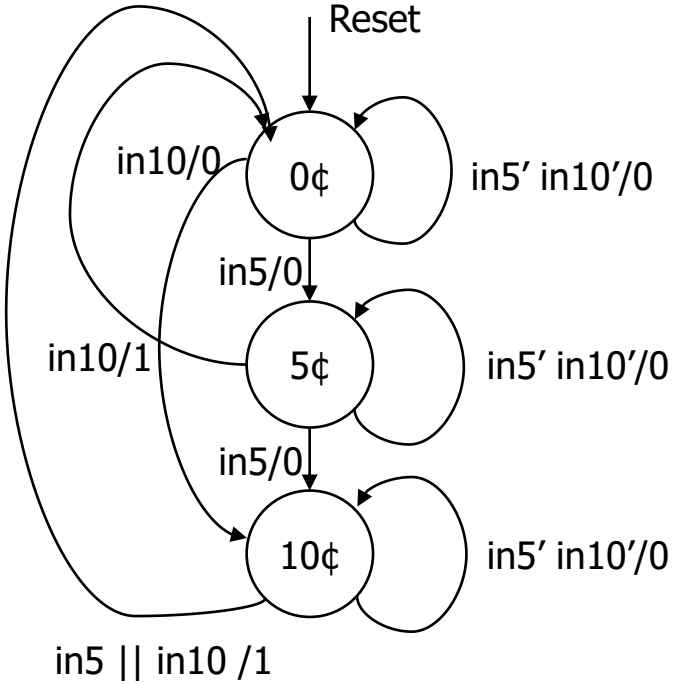
## Moore machine

## Έξοδοι από κατάσταση



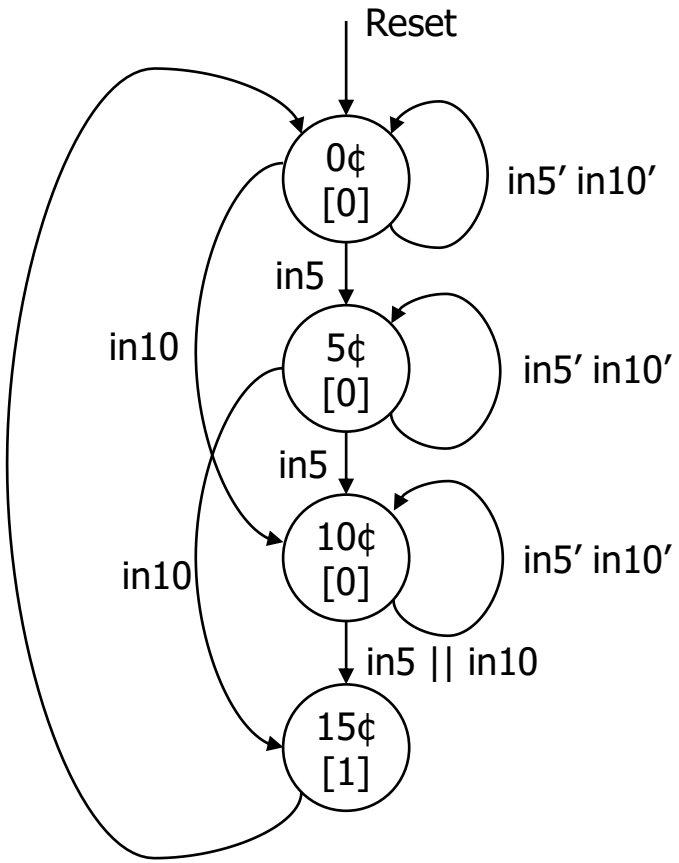

## Mealy machine

## Έξοδοι στις μεταβάσεις



# Moore Verilog FSM

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

  always_comb begin
    open = 0; // default output value
    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_ff @(posedge clk)
    if (rst) state <= zero;
    else     state <= next_state;
endmodule
```



# Mealy Verilog FSM

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

  always_comb begin
    open = 0; // default output value
    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_ff @(posedge clk)
    if (rst) state <= zero;
    else     state <= next_state;
endmodule
```