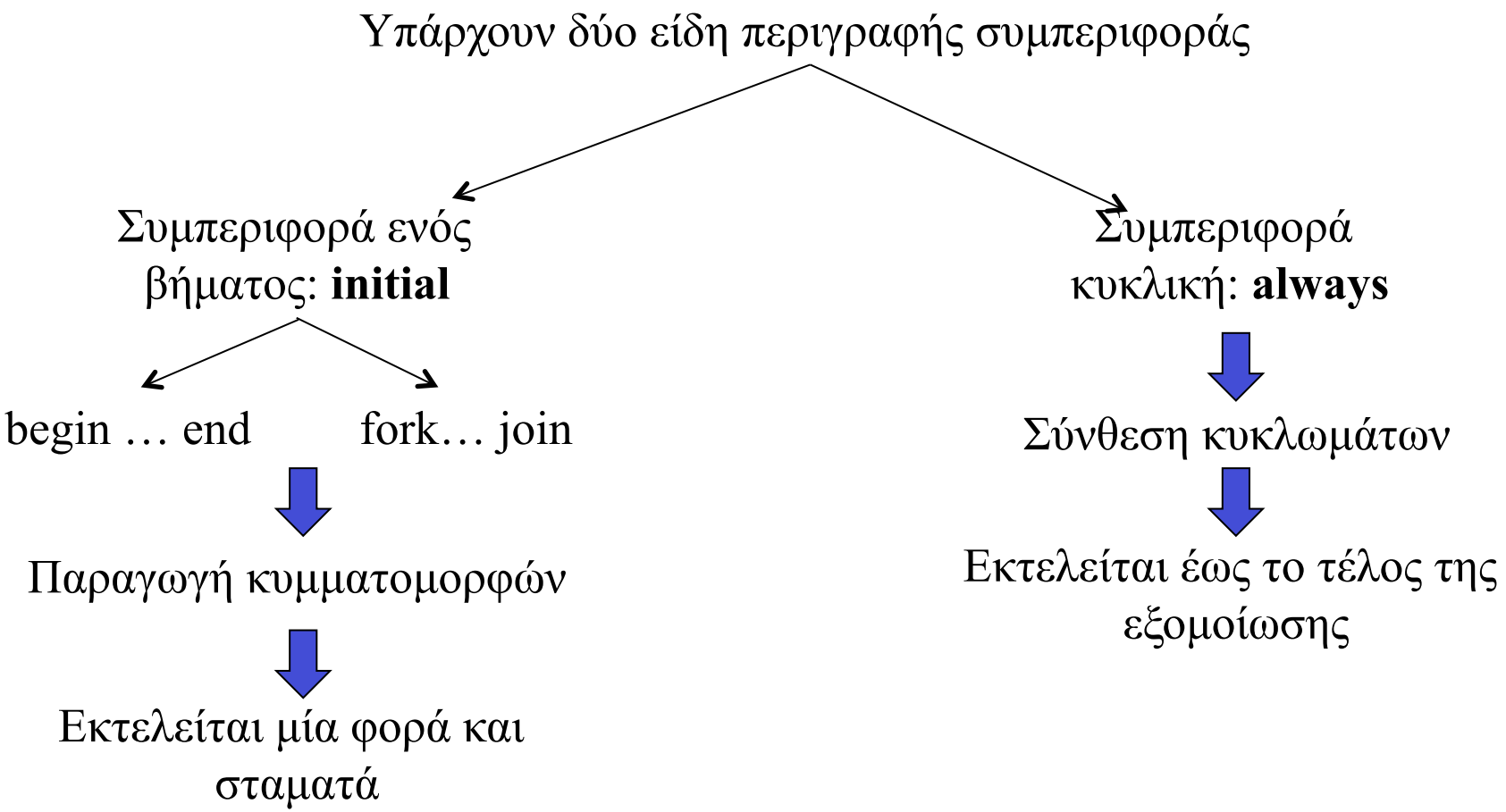
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΛΛΑΔΑ ΚΟΙΝΩΝΙΚΟ ΠΛΑΙΣΙΟ ΣΤΗΡΙΞΗΣ 2008
Ανάπτυξη παντού. Ανάπτυξη για όλους.
Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

9η Θεματική Ενότητα : Εισαγωγή στις Γλώσσες Περιγραφής Υλικού: Μοντέλα Ακολουθιακών Κυκλωμάτων, Καταχωρητών, Μετρητών, Μνημών

Εισαγωγή


Υπάρχουν δύο είδη περιγραφής συμπεριφοράς
Συμπεριφορά ενός βήματος: **initial**
Συμπεριφορά κυκλική: **always**
begin … end
fork… join
Παραγωγή κυμματομορφών
Εκτελείται μία φορά και σταματά
Σύνθεση κυκλωμάτων
Εκτελείται έως το τέλος της εξομοίωσης

Παράδειγμα: ρολόι με initial

Μπορούμε να ορίσουμε ένα περιοδικό σήμα όπως το ρολόι με πολλούς τρόπους

```
initial
  begin
    clock = 1'b0;
    repeat (30)
      #10 clock = ~clock;
  end
```

```
initial
  begin
    clock = 1'b0;
  end
```

Αρχικοποίηση

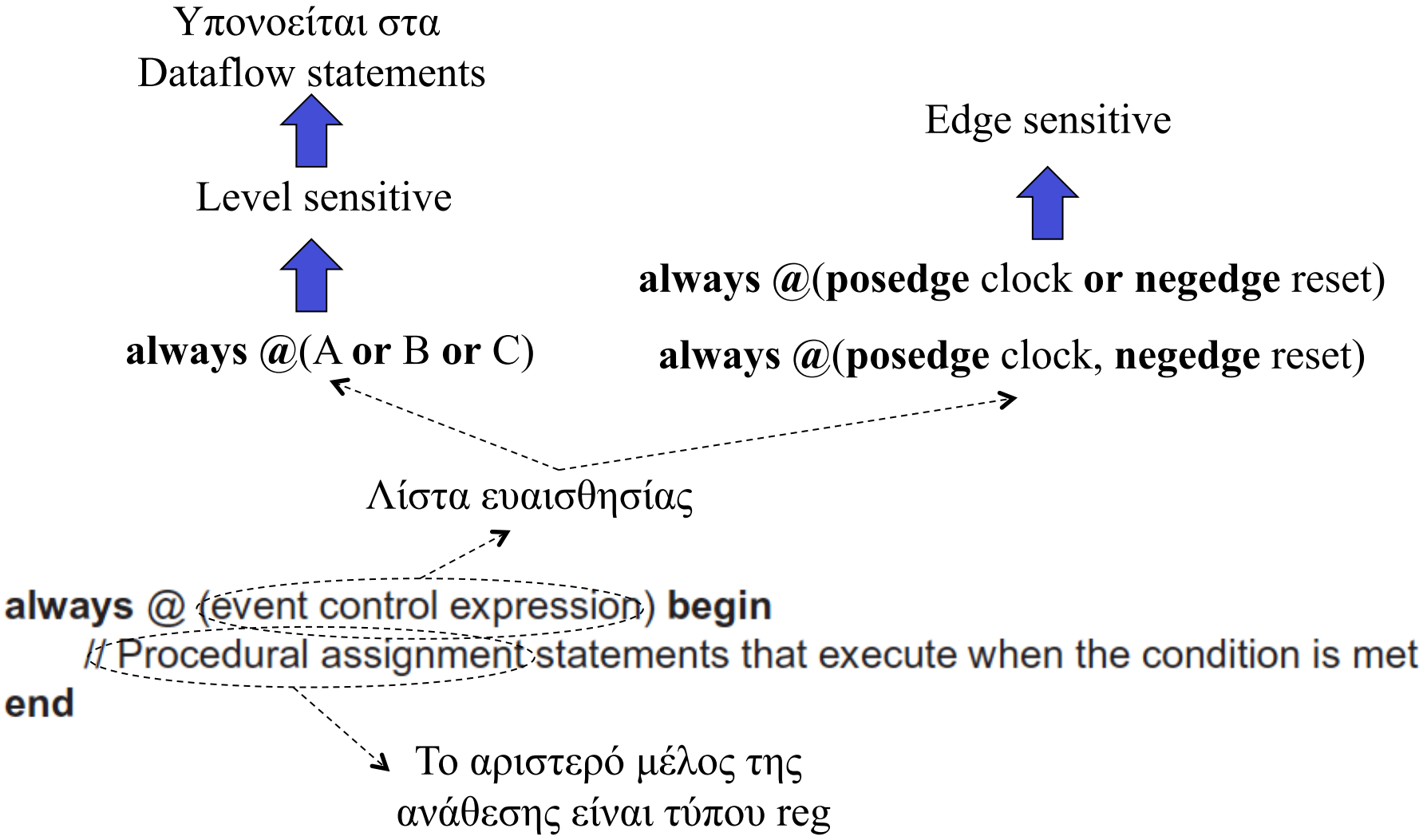
```
initial 300 $finish;
always #10 clock = ~clock;
```

Ορίζονται 15 περίοδοι ρολογιού

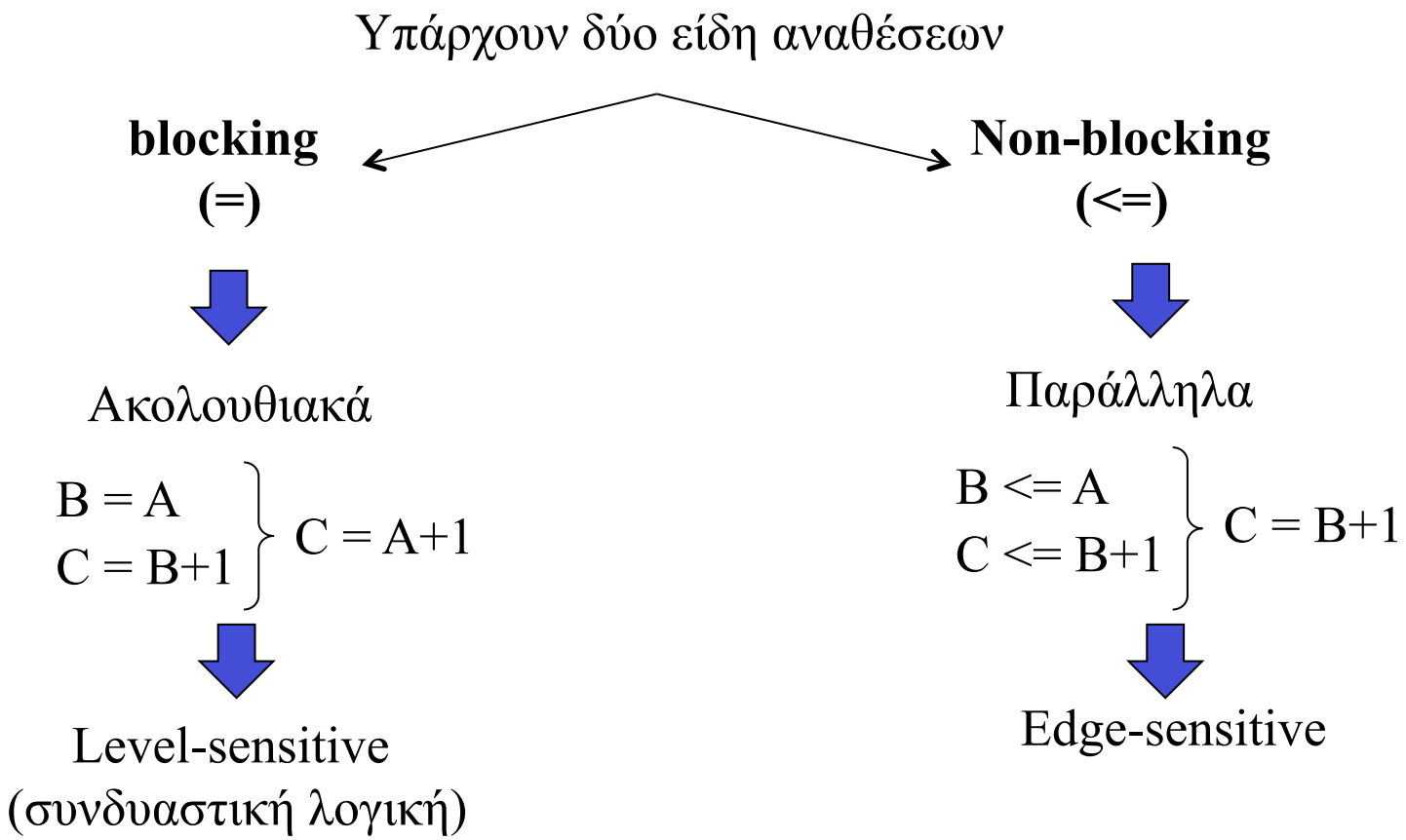
Τελεστής καθυστέρησης

```
initial begin clock = 0; forever #10 clock = ~clock; end
```

Τελεστής Ελέγχου Γεγονότων



Procedural assignments



D-Latch

```
// Description of D latch (See Fig. 5.6)
module D_latch (Q, D, enable);
  output Q;
  input  D, enable;
  reg    Q;
  always @ (enable or D)
    if (enable) Q <= D;                  // Same as: if (enable == 1)
endmodule

// Alternative syntax (Verilog 2001, 2005)
module D_latch (output reg Q, input enable, D);
  always @ (enable, D)
    if (enable) Q <= D;                  // No action if enable not asserted
endmodule
```

Η ανάθεση στο **always** block υποχρεώνει την χρήση τύπου **reg**

D-Flip Flop

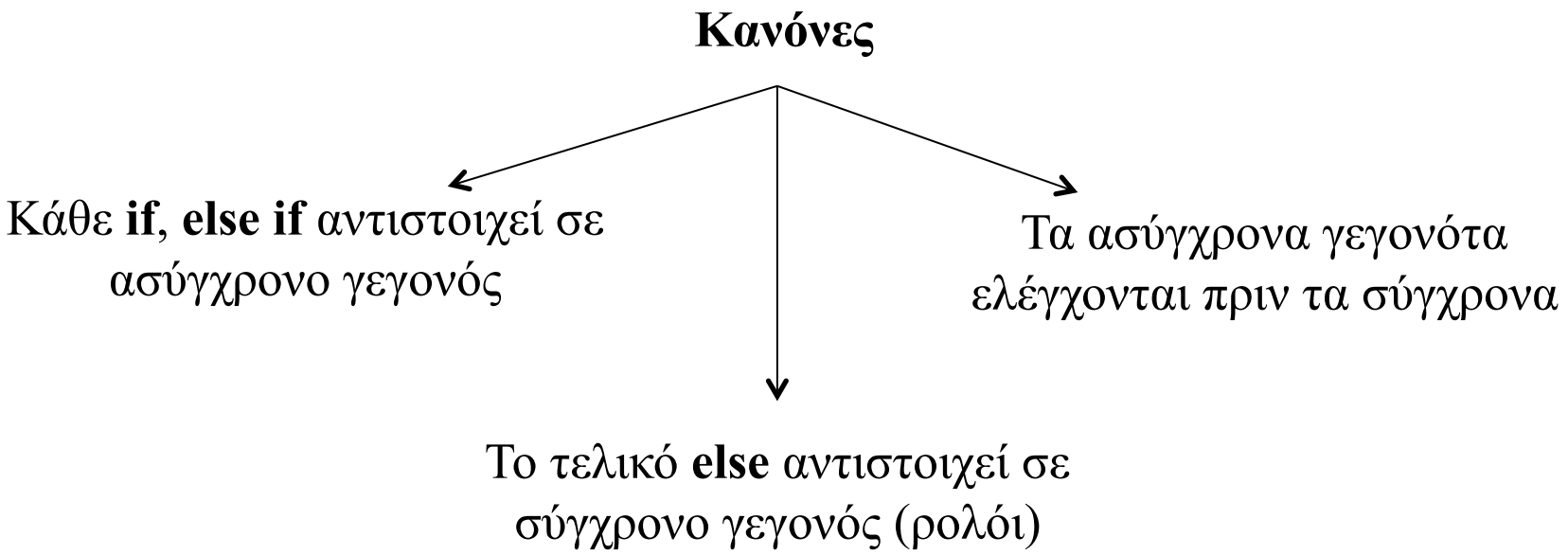
```
// D flip-flop without reset
module D_FF (Q, D, Clk);
  output Q;
  input  D, Clk;
  reg    Q;
  always @ (posedge Clk)
     Q <= D;
  endmodule

// D flip-flop with asynchronous reset (V2001, V2005)
module DFF (output reg Q, input D, Clk, rst);
  always @ (posedge Clk, negedge rst)
  if (!rst) Q <= 1'b0;          // Same as: if (rst == 0)
  else Q <= D;
endmodule
```

Η αποθήκευση γίνεται στην θετική ακμή του ρολογιού

Ασύγχρονη μηδένιση

D-Flip Flop

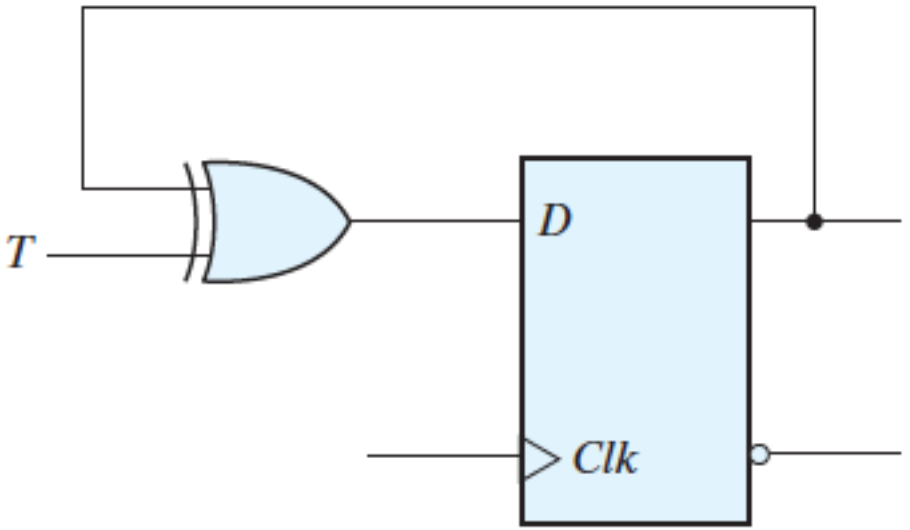
Η αναγνώριση του σήματος ρολογιού γίνεται μέσω της κατάλληλης περιγραφής



T-Flip Flop

```
module DFF (output reg Q, input D, Clk, rst);
  always @ (posedge Clk, negedge rst)
  if (!rst) Q <= 1'b0;            // Same as: if (rst == 0)
  else Q <= D;
endmodule
```

```
module TFF (Q, T, Clk, rst);
  output Q;
  input   T, Clk, rst;
  wire    DT;
  assign DT = Q ^ T ;
// Instantiate the D flip-flop
  DFF TF1 (Q, DT, Clk, rst);
endmodule
```

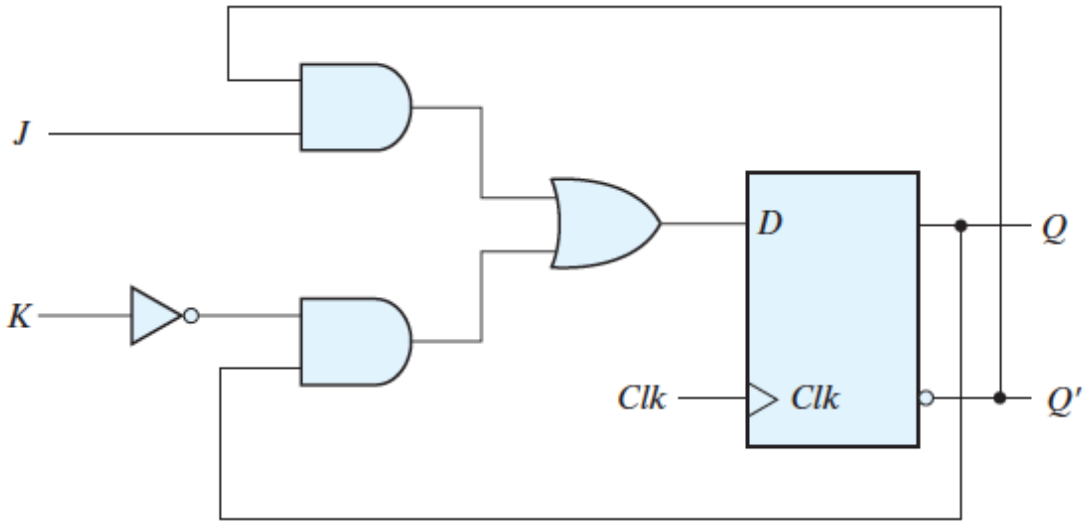


(b) From *D* flip-flop

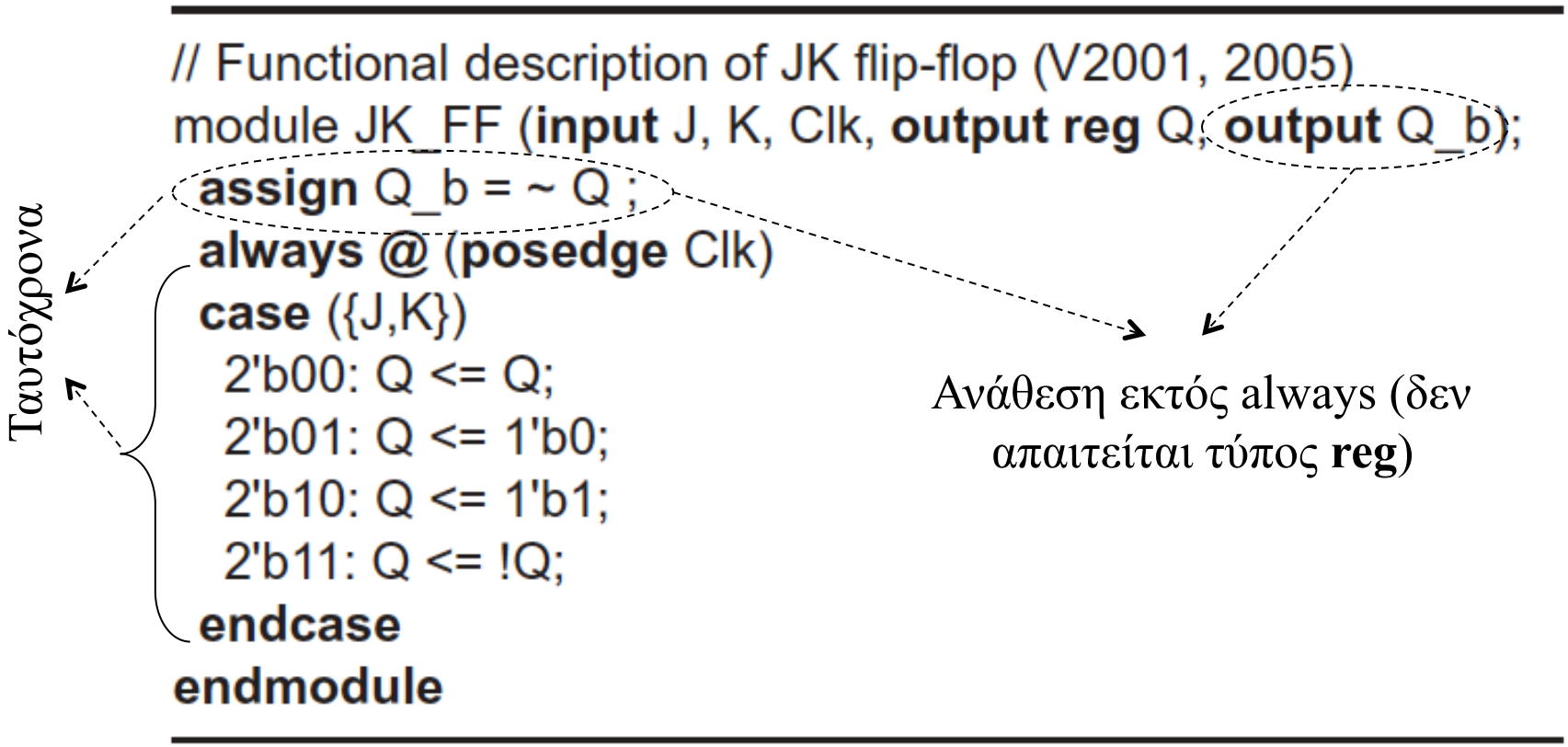
JK-Flip Flop

```
module DFF (output reg Q, input D, Clk, rst);
  always @ (posedge Clk, negedge rst)
  if (!rst) Q <= 1'b0;          // Same as: if (rst == 0)
  else Q <= D;
endmodule
```

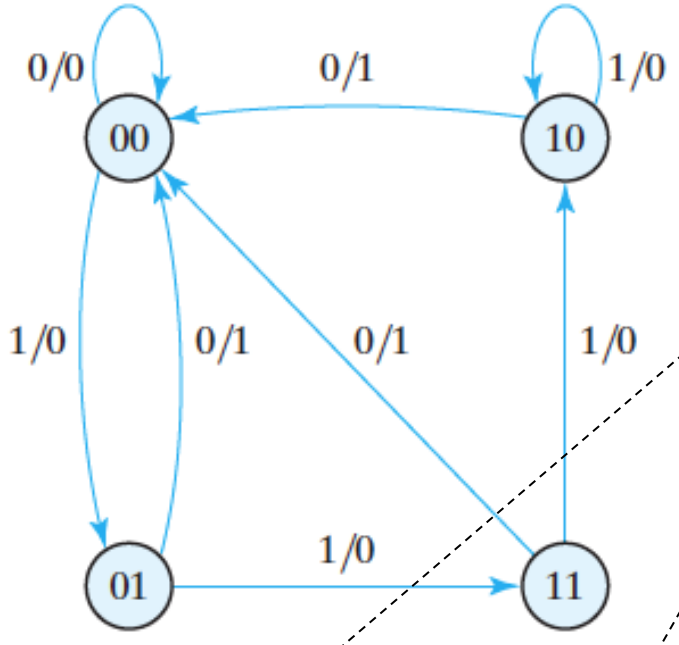
```
module JKFF (output reg Q, input J, K, Clk, rst);
  wire JK;
  assign JK = (J & ~Q) | (~K & Q);
// Instantiate D flip-flop
  DFF JK1 (Q, JK, Clk, rst);
endmodule
```



JK-Flip Flop (2)



Διαγράμματα Κατάστασης σε HDL



Αλλαγή κατάστασης στην ακμή του ρολογιού

Επόμενη κατάσταση (level sensitive)

```
module Mealy_Zero_Detector (
 output reg y_out,
 input   x_in, clock, reset
);
 reg [1: 0]      state, next_state;
 parameter       S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
always @ (posedge clock, negedge reset)
  if (reset == 0) state <= S0;
  else state <= next_state;

always @ (state, x_in)                          // Form the next state
  case (state)
     S0:       if (x_in)  next_state = S1; else next_state = S0;
     S1:       if (x_in)  next_state = S3; else next_state = S0;
     S2:       if (~x_in)  next_state = S0; else next_state = S2;
     S3:       if (x_in)  next_state = S2; else next_state = S0;
  endcase

always @ (state, x_in)                          // Form the Mealy output
  case (state)
     S0:          y_out = 0;
     S1, S2, S3: y_out = ~x_in;
  endcase
endmodule
```

Δήλωση σταθερών

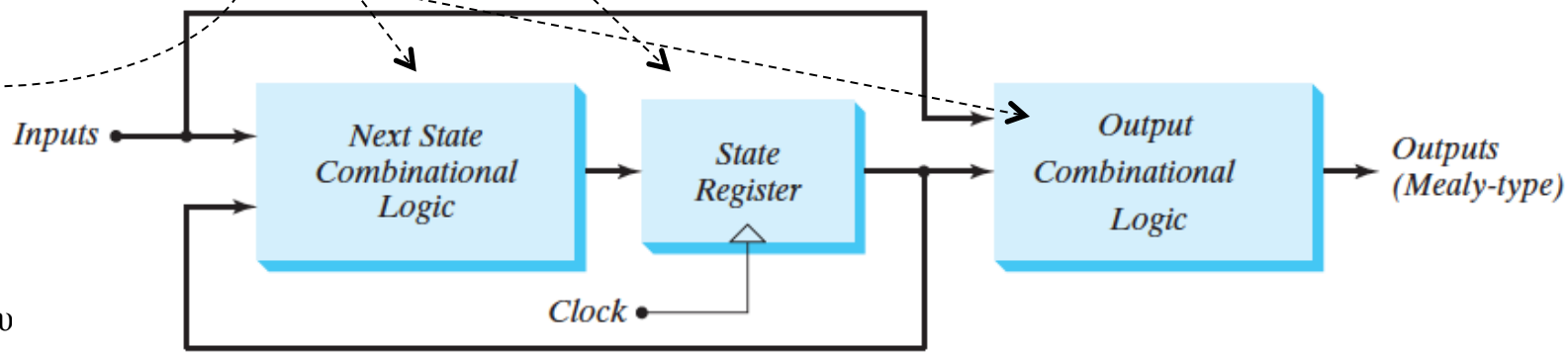
Πλάτος 2 bit

Verilog 2001, 2005 syntax

Παραγωγή εξόδων

Αντιστοίχιση σε υλικό

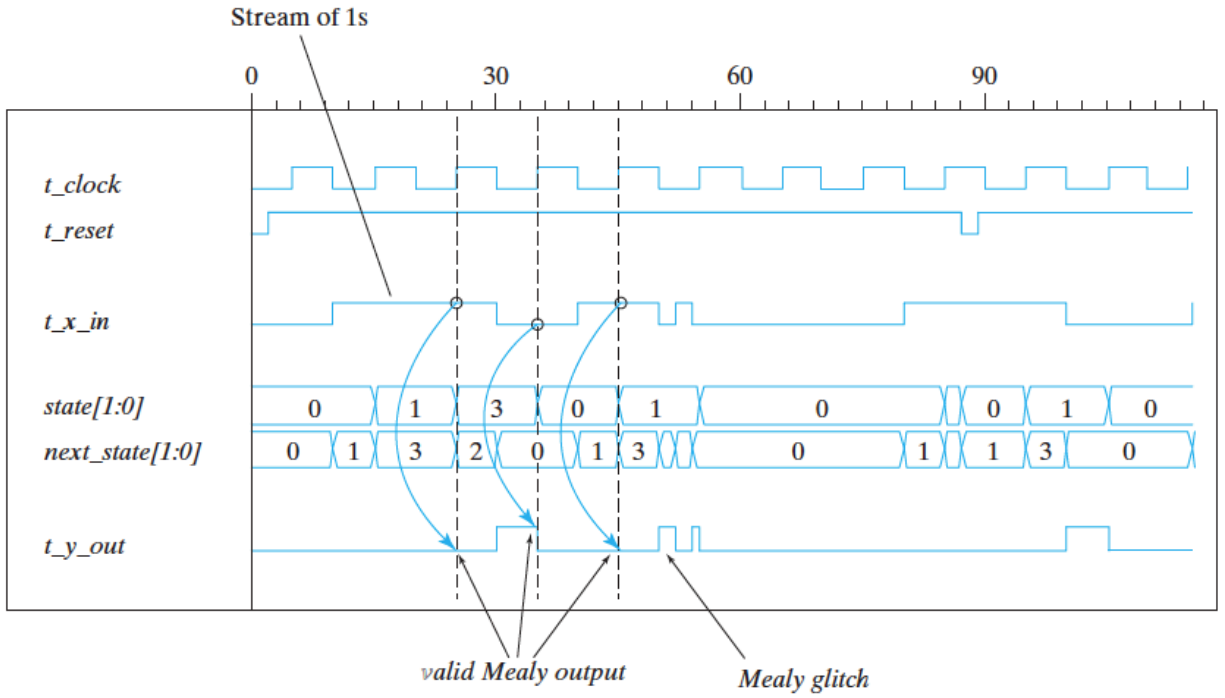
```
module Mealy_Zero_Detector (
  output reg y_out,
  input    x_in, clock, reset
);
  reg [1: 0]          state, next_state;
  parameter           S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
  always @ (posedge clock, negedge reset)    Verilog 2001, 2005 syntax
    if (reset == 0) state <= S0;
    else state <= next_state;

  always @ (state, x_in)                     // Form the next state
    case (state)
       S0:       if (x_in)  next_state = S1; else next_state = S0;
       S1:       if (x_in)  next_state = S3; else next_state = S0;
       S2:       if (~x_in)  next_state = S0; else next_state = S2;
       S3:       if (x_in)  next_state = S2; else next_state = S0;
    endcase

  always @ (state, x_in)                     // Form the Mealy output
    case (state)
       S0:          y_out = 0;
       S1, S2, S3: y_out = ~x_in;
    endcase
endmodule
```



Test Bench

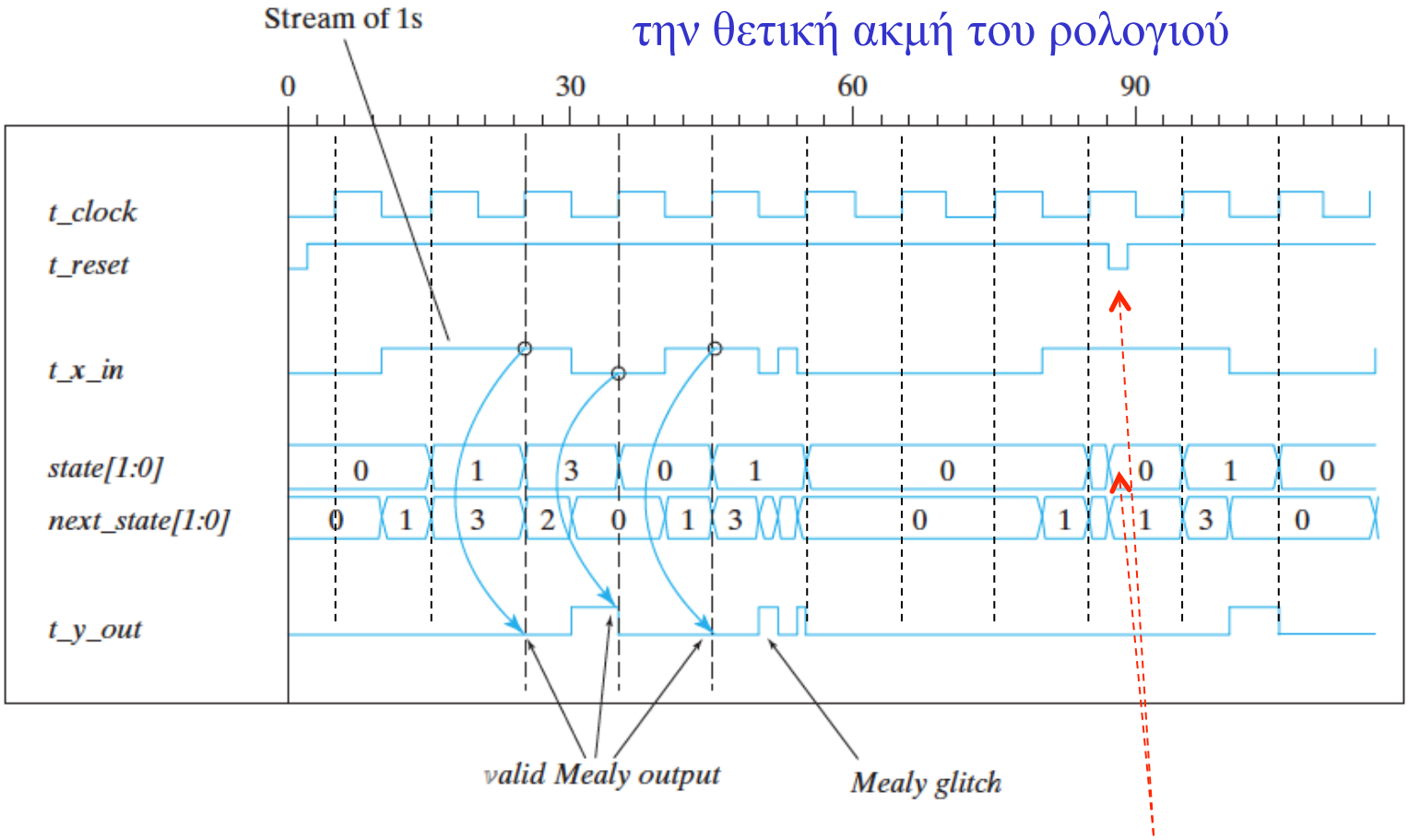
```
module t_Mealy_Zero_Detector;
  wire      t_y_out;
  reg       t_x_in, t_clock, t_reset;

Mealy_Zero_Detector M0 (t_y_out, t_x_in, t_clock, t_reset);
initial #200 $finish;
initial begin t_clock = 0; forever #5 t_clock = ~t_clock; end

initial fork
      t_reset = 0;
  #2  t_reset = 1;
  #87 t_reset = 0;
  #89 t_reset = 1;

  #10 t_x_in = 1;
  #30 t_x_in = 0;
  #40 t_x_in = 1;
  #50 t_x_in = 0;
  #52 t_x_in = 1;
  #54 t_x_in = 0;
  #70 t_x_in = 1;
  #80 t_x_in = 1;
  #70 t_x_in = 0;
  #90 t_x_in = 1;
  #100 t_x_in = 0;
  #120 t_x_in = 1;
  #160 t_x_in = 0;
  #170 t_x_in = 1;
join
endmodule
```

Εκτελούνται παράλληλα από t=0 οι εσωτερικές αναθέσεις



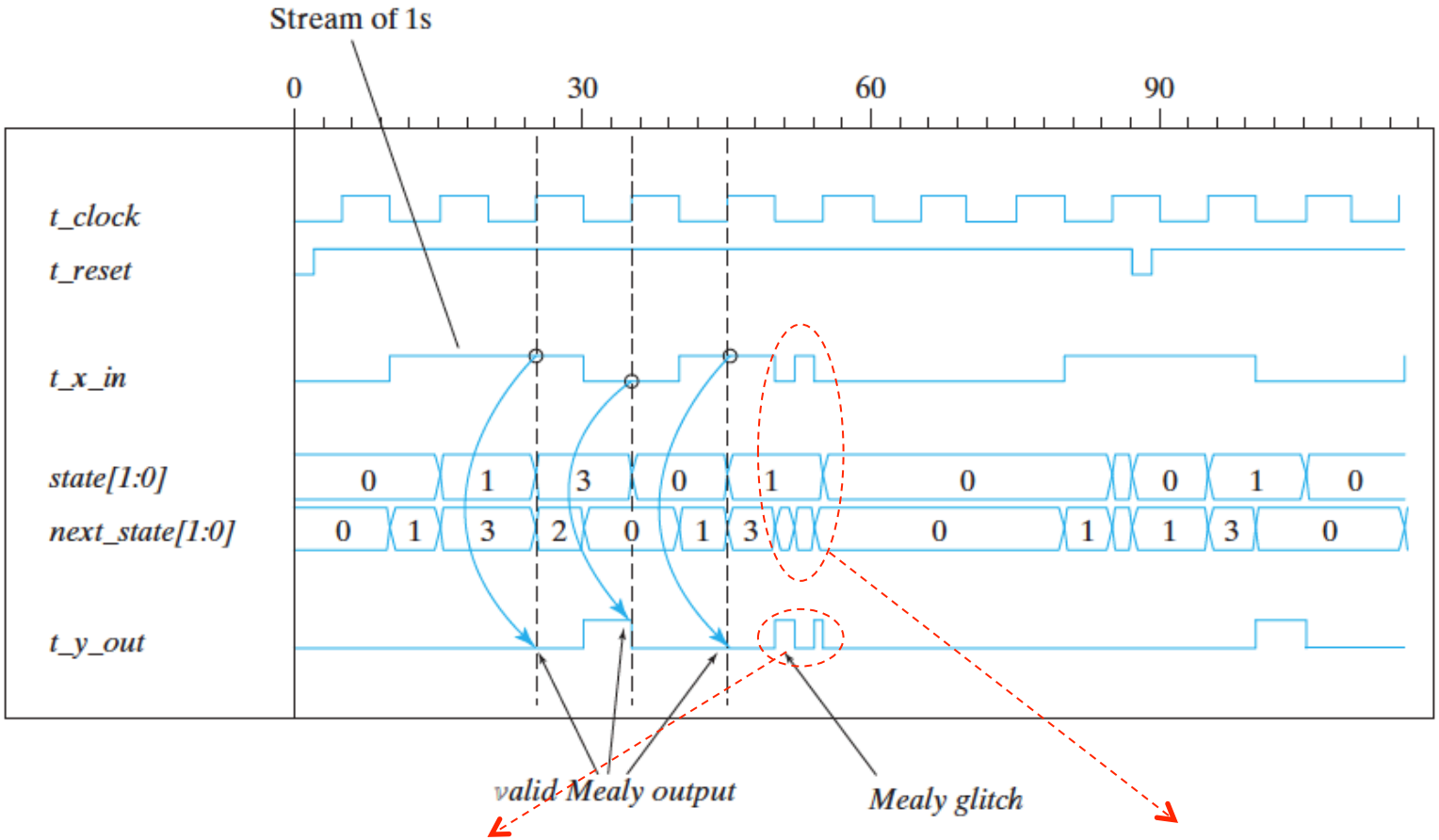
Test Bench

Η επόμενη κατάσταση αλλάζει πάντα με την θετική ακμή του ρολογιού



Εξαίρεση: ασύγχρονος μηδενισμός

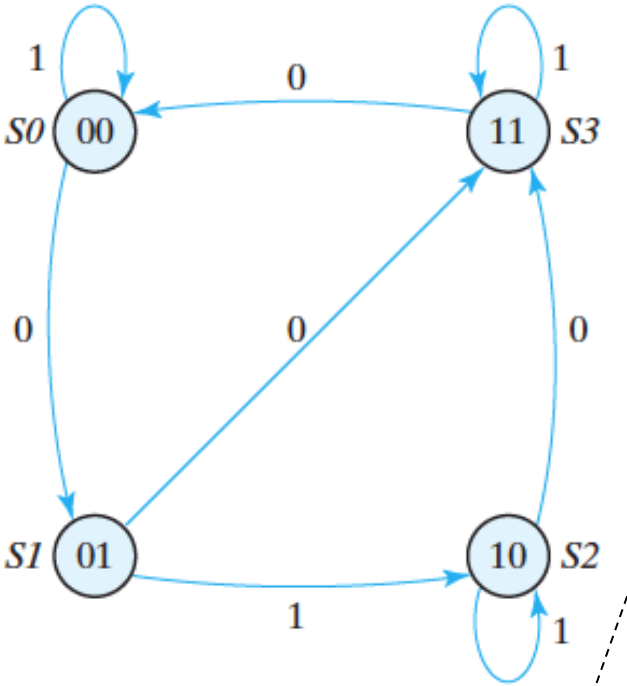
Test Bench



Η έξοδος είναι level sensitive (συνδυαστικό κύκλωμα)

Η επόμενη κατάσταση είναι level sensitive (συνδυαστικό κύκλωμα)

Παράδειγμα



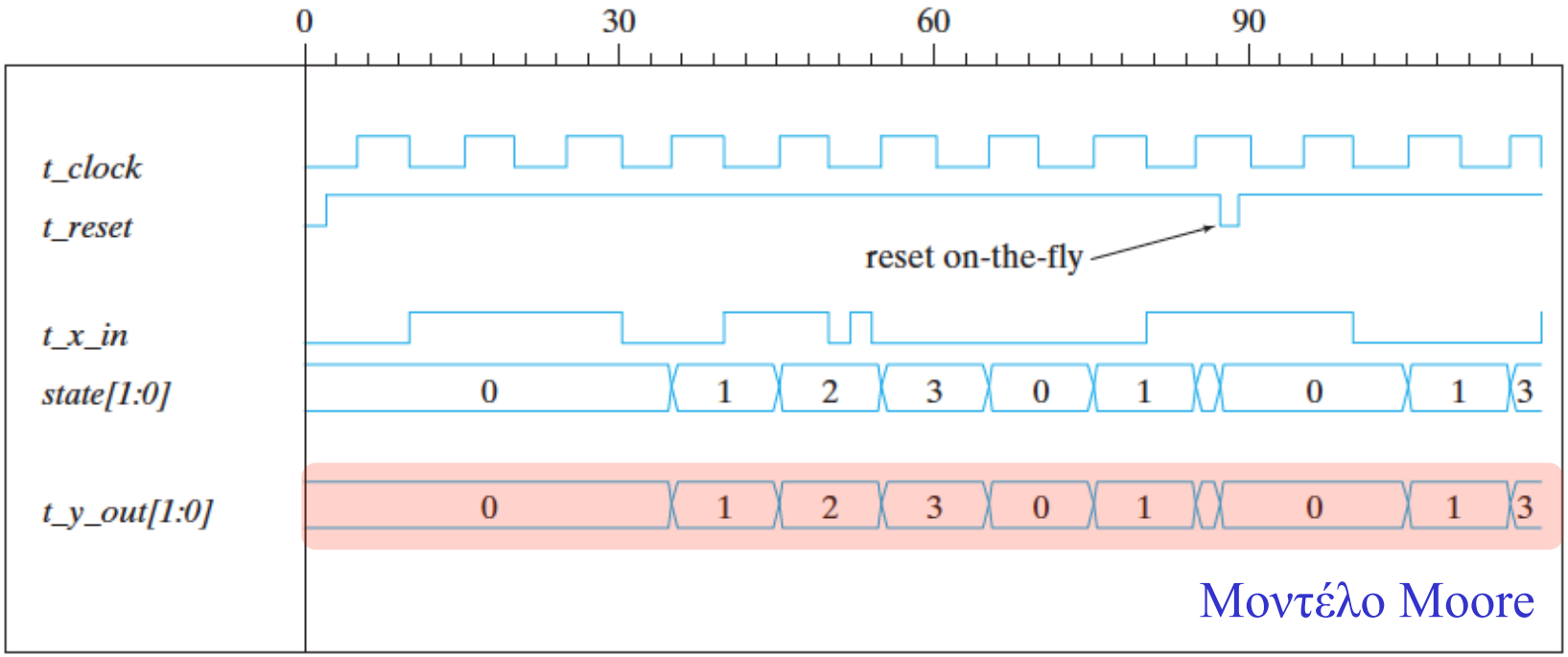

```
// Moore model FSM (see Fig. 5.19)                    Verilog 2001, 2005 syntax
module Moore_Model_Fig_5_19 (
 output [1: 0]          y_out,
 input                  x_in, clock, reset
);
 reg [1: 0]             state;
 parameter              S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
 always @ (posedge clock, negedge reset)
  if (reset == 0) state <= S0;                          // Initialize to state S0
  else case (state)
   S0:      if (~x_in) state <= S1; else state <= S0;
   S1:      if (x_in)  state <= S2; else state <= S3;
   S2:      if (~x_in) state <= S3; else state <= S2;
   S3:      if (~x_in) state <= S0; else state <= S3;
  endcase
 assign y_out = state;          // Output of flip-flops
endmodule
```

Το else καθορίζει ευασθησία στην ακμή ρολογιού

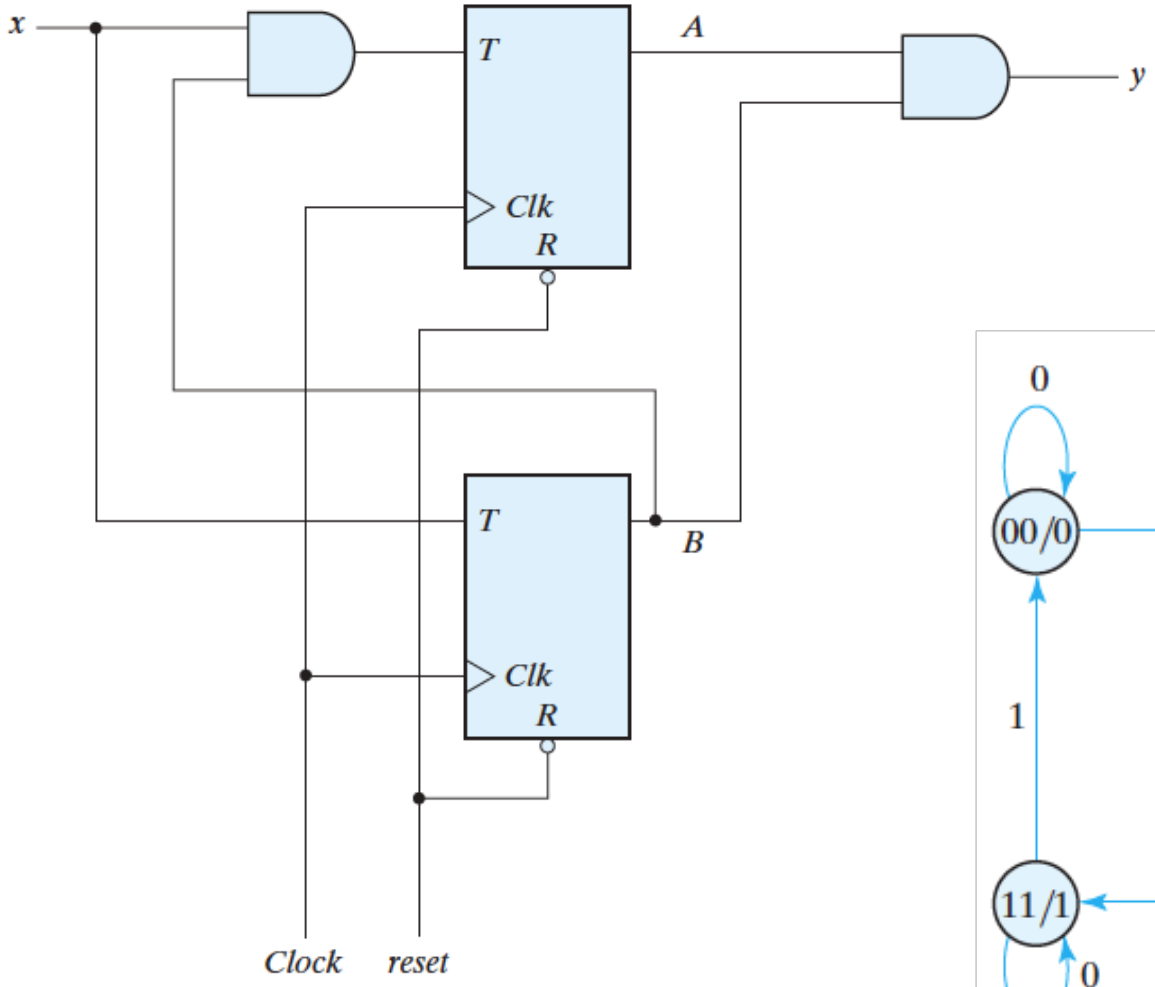
Συνδυαστική Λογική

non-blocking assignment

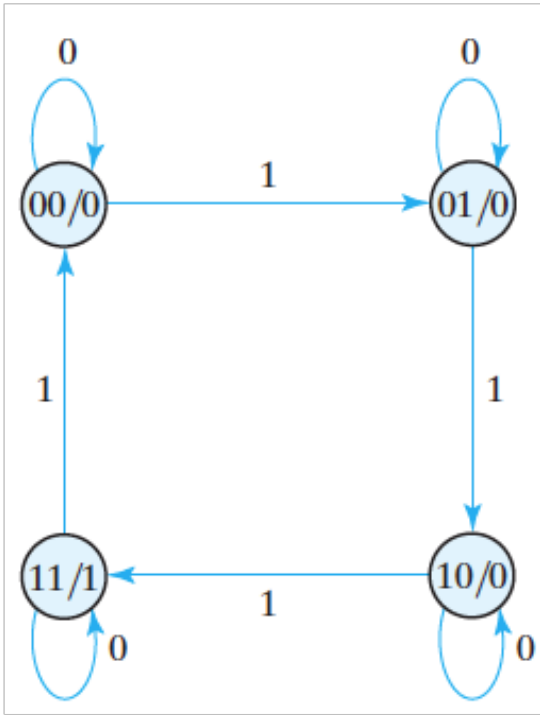
Παράδειγμα (test bench)



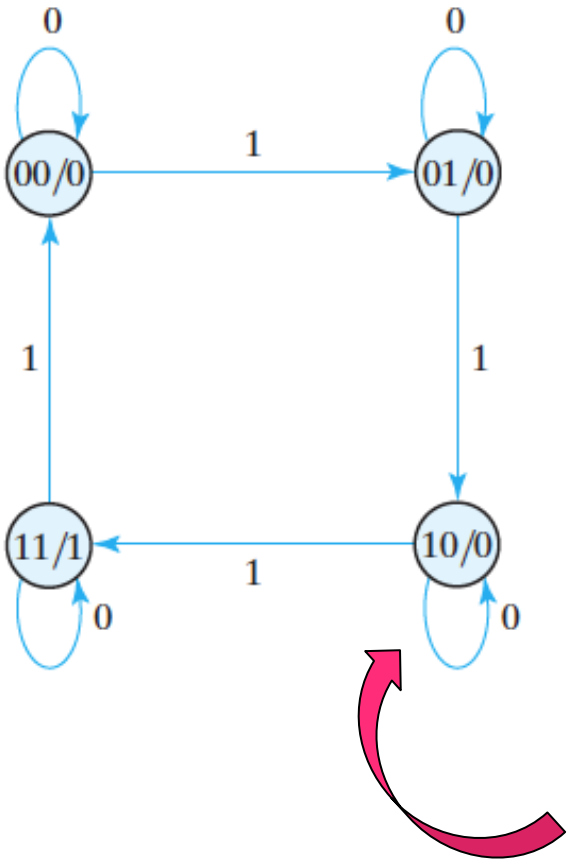
Παράδειγμα (2): Δυαδικός Μετρητής



(a) Circuit diagram



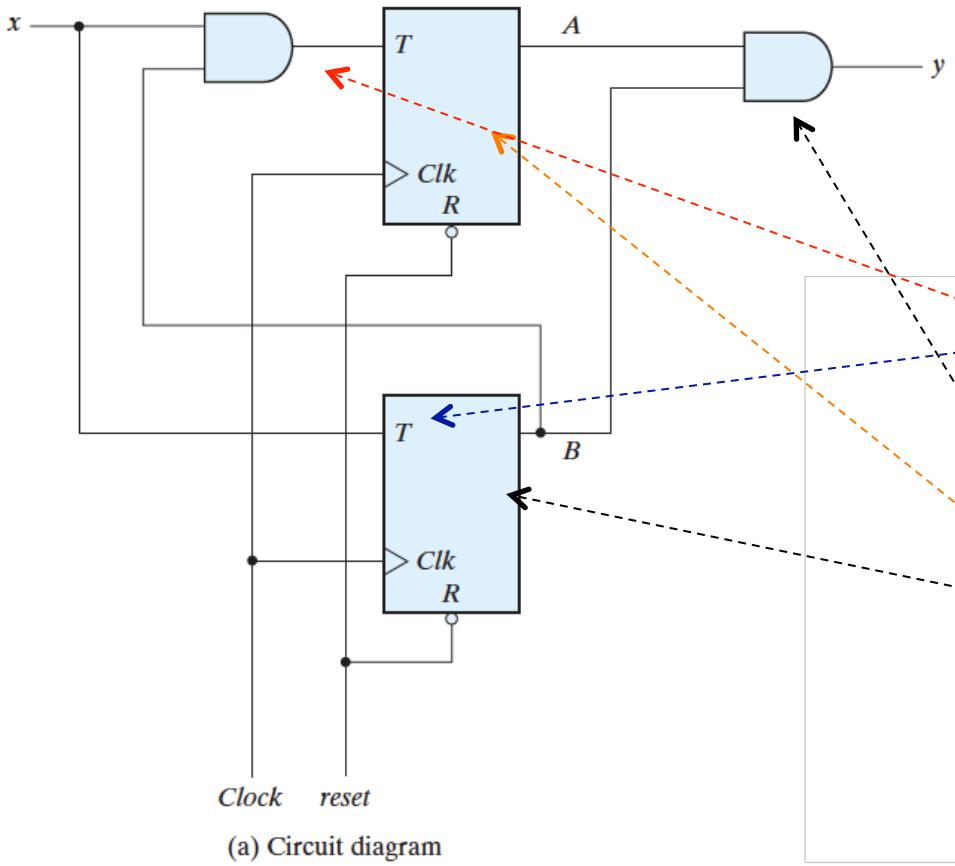
(α) Με Μοντέλο Διαγράμματος Καταστάσεων




```
module Moore_Model_Fig_5_20 (
 output  y_out,
 input   x_in, clock, reset
);
 reg [1: 0]              state;
 parameter               S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
 always @ (posedge clock, negedge reset)
  if (reset == 0) state <= S0;                  // Initialize to state S0
  else case (state)
    S0:     if (x_in)  state <= S1; else state <= S0;
    S1:     if (x_in)  state <= S2; else state <= S1;
    S2:     if (x_in)  state <= S3; else state <= S2;
    S3:     if (x_in)  state <= S0; else state <= S3;
  endcase
 assign y_out = (state == S3);          // Output of flip-flops
endmodule
```

Η αντιστοιχία
είναι προφανής

(β) Με Μοντέλο Δομής



(a) Circuit diagram

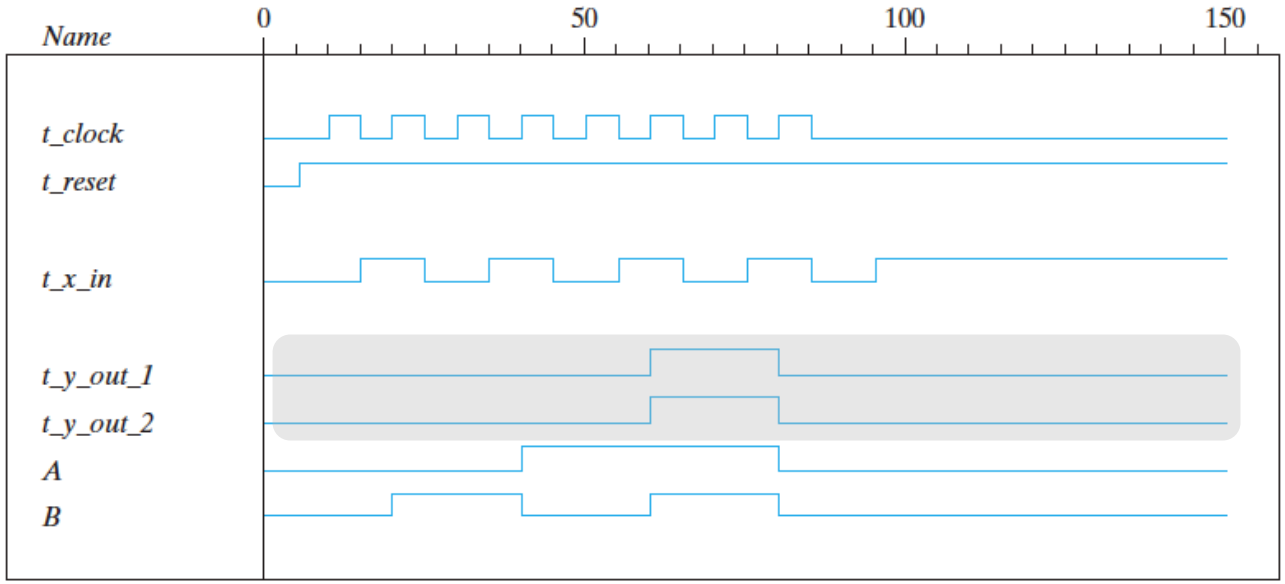
Η αντιστοιχία είναι προφανής

```
module Moore_Model_STR_Fig_5_20 (
  output      y_out, A, B,
  input       x_in, clock, reset
);
  wire        TA, TB;

// Flip-flop input equations
  assign TA = x_in & B;
  assign TB = x_in;
// Output equation
  assign y_out = A & B;

// Instantiate Toggle flip-flops
  Toggle_flip_flop_3 M_A (A, TA, clock, reset);
  Toggle_flip_flop_3 M_B (B, TB, clock, reset);
endmodule

module Toggle_flip_flop (Q, T, CLK, RST_b);
  output      Q;
  input       T, CLK, RST_b;
  reg         Q;

  always @ (posedge CLK, negedge RST_b)
    if (RST_b == 0) Q <= 1'b0;
    else if (T) Q <= ~Q;
endmodule
```

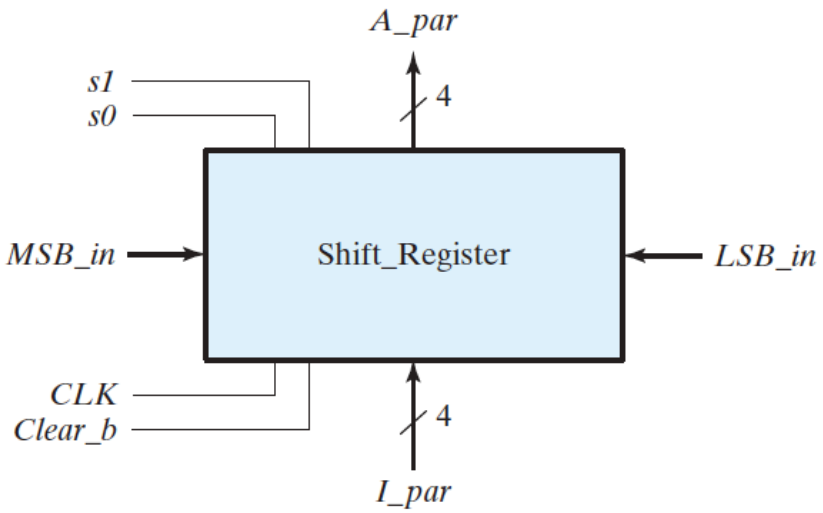
Test Bench για (α), (β)

```
module t_Moore_Fig_5_20;
 wire        t_y_out_2, t_y_out_1;
 reg         t_x_in, t_clock, t_reset;

Moore_Model_Fig_5_20        M1(t_y_out_1, t_x_in, t_clock, t_reset);
Moore_Model_STR_Fig_5_20    M2 (t_y_out_2, A, B, t_x_in, t_clock, t_reset);

initial #200 $finish;
initial begin
     t_reset = 0;
     t_clock = 0;
     #5 t_reset = 1;
 repeat (16)
     #5 t_clock = ~t_clock;
end
initial begin
         t_x_in = 0;
 #15 t_x_in = 1;
 repeat (8)
   #10 t_x_in = ~t_x_in;
 end
endmodule
```

Έλεγχος και των δύο περιγραφών



Καταχωρητής Ολίσθησης (behavioral)



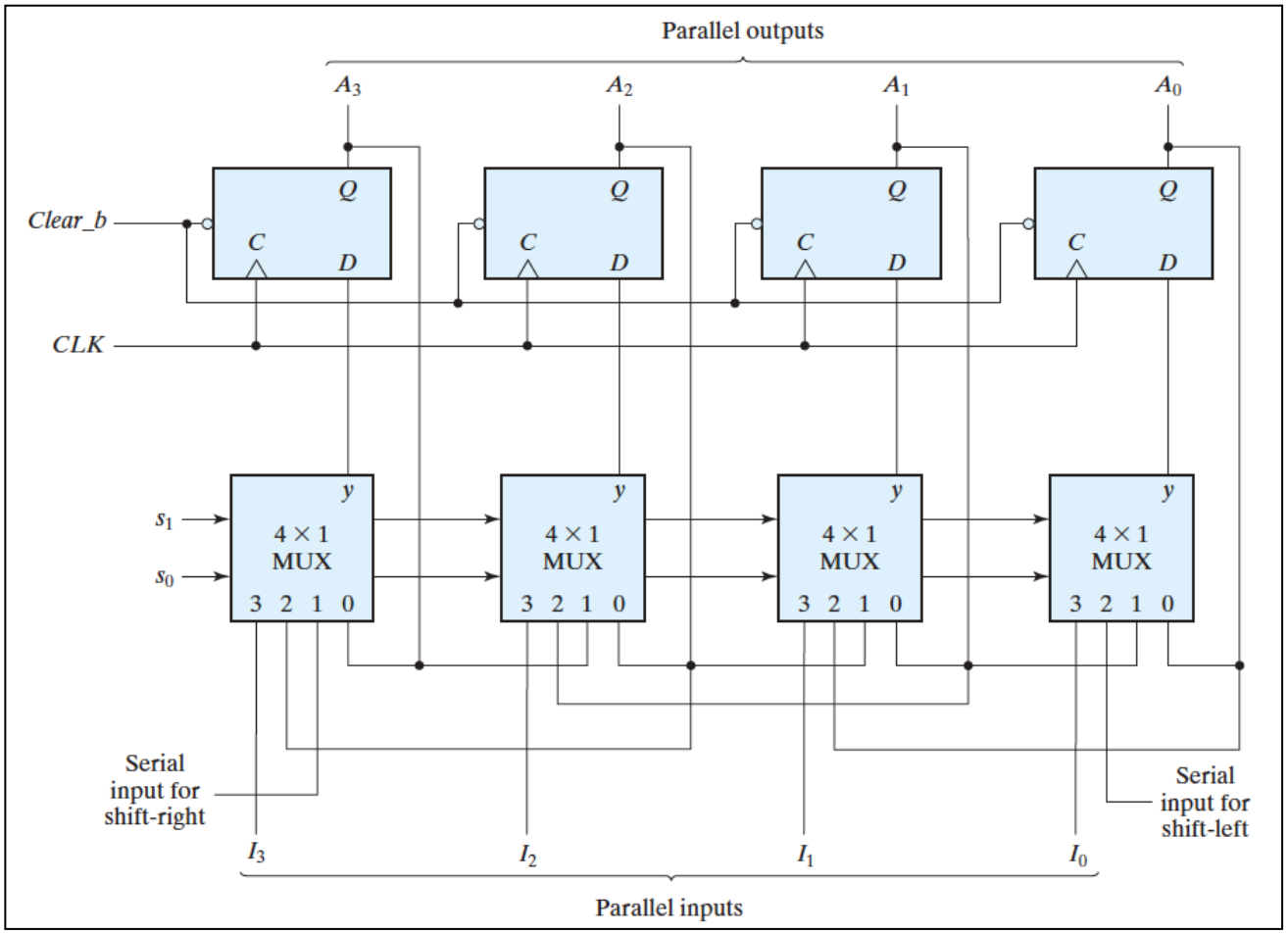
Mode Control		
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

```
module Shift_Register_4_beh (
  output reg      [3: 0]    A_par,
  input           [3: 0]    I_par,
  input           s1, s0,
                  MSB_in, LSB_in,
                  CLK, Clear_b
);
```

Μπορεί και να παραληφθεί

```
always @ (posedge CLK, negedge Clear_b)
  if (Clear_b == 0) A_par <= 4'b0000;
  else
    case ({s1, s0})
      2'b00: A_par <= A_par;
      2'b01: A_par <= {MSB_in, A_par[3: 1]};
      2'b10: A_par <= {A_par[2: 0], LSB_in};
      2'b11: A_par <= I_par;
    endcase
endmodule
```

Καταχωρητής Ολίσθησης (structural)



Καταχωρητής Ολίσθησης

```
// 4x1 multiplexer                  // behavioral model
module Mux_4_x_1 (mux_out, i0, i1, i2, i3, select);
  output        mux_out;
  input         i0, i1, i2, i3;
  input [1: 0]  select;
  reg           mux_out;
  always @ (select, i0, i1, i2, i3)
    case (select)
      2'b00:    mux_out = i0;
      2'b01:    mux_out = i1;
      2'b10:    mux_out = i2;
      2'b11:    mux_out = i3;
    endcase
endmodule
```

```
module D_flip_flop (Q, D, CLK, Clr_b);
  output        Q;
  input         D, CLK, Clr;
  reg           Q;

  always @ (posedge CLK, negedge Clr_b)
    if (!Clr_b) Q <= 1'b0; else Q <= D;
endmodule
```

Καταχωρητής Ολίσθησης

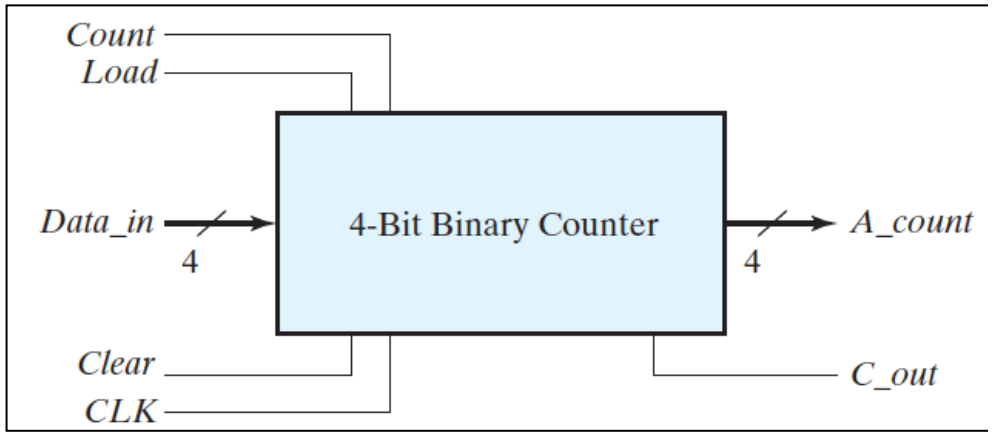
```
// One stage of shift register
module stage (i0, i1, i2, i3, Q, select, CLK, Clr_b);
  input          i0,          // circulation bit selection
                 i1,          // data from left neighbor or serial input for shift-right
                 i2,          // data from right neighbor or serial input for shift-left
                 i3;          // data from parallel input
  output         Q;
  input [1: 0]   select;      // stage mode control bus
  input          CLK, Clr_b;  // Clock, Clear for flip-flops
  wire           mux_out;

// instantiate mux and flip-flop
  Mux_4_x_1   M0      (mux_out, i0, i1, i2, i3, select);
  D_flip_flop M1      (Q, mux_out, CLK, Clr_b);
endmodule
```

Καταχωρητής Ολίσθησης

```
module Shift_Register_4_str (
  output [3: 0] A_par,
  input [3: 0]   I_par,
  input          s1, s0,
  input          MSB_in, LSB_in, CLK, Clear_b
);

// bus for mode control
  assign   [1:0]   select = {s1, s0};
// Instantiate the four stages
  stage ST0 (A_par[0], A_par[1], LSB_in, I_par[0], A_par[0], select, CLK, Clear_b);
  stage ST1 (A_par[1], A_par[2], A_par[0], I_par[1], A_par[1], select, CLK, Clear_b);
  stage ST2 (A_par[2], A_par[3], A_par[1], I_par[2], A_par[2], select, CLK, Clear_b);
  stage ST3 (A_par[3], MSB_in, A_par[2], I_par[3], A_par[3], select, CLK, Clear_b);
endmodule
```

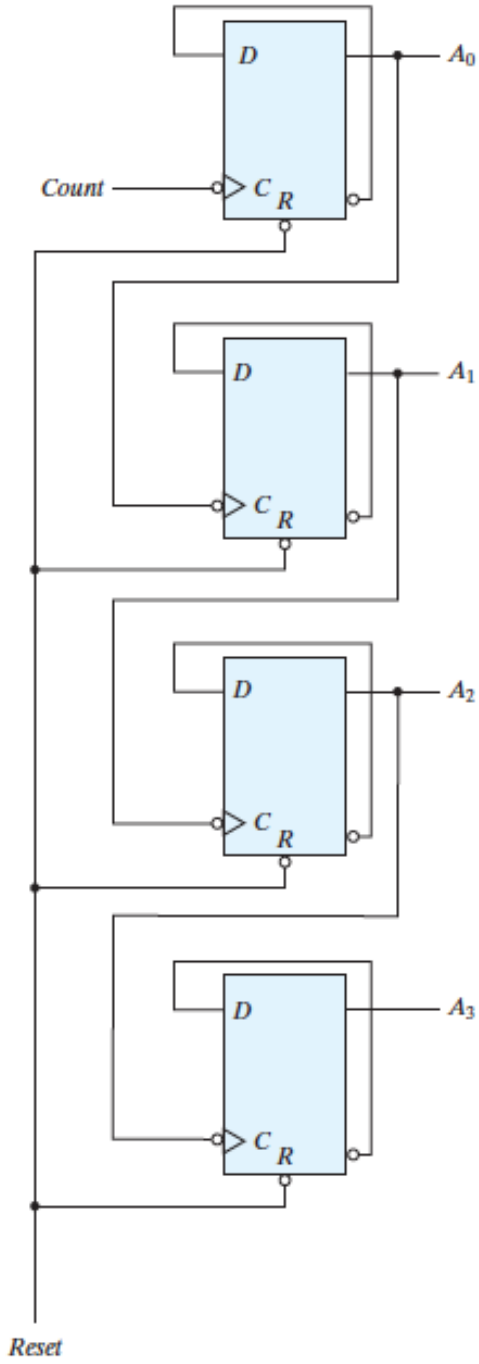
Σύγχρονος Μετρητής

```
module Binary_Counter_4_Par_Load (
  output reg [3: 0]                A_count,
  output                           C_out,
  input [3: 0]                     Data_in,
  input                            Count,
                                   Load,
                                   CLK,
                                   Clear_b
);
assign C_out = Count && (~Load) && (A_count == 4'b1111);
always @ (posedge CLK, negedge Clear_b)
    if (~Clear_b)                  A_count <= 4'b0000;
    else if (Load)                 A_count <= Data_in;
    else if (Count)                A_count <= A_count + 1'b1;
    else                           A_count <= A_count;  // redundant statement
endmodule
```



Μετρητής Ριπής

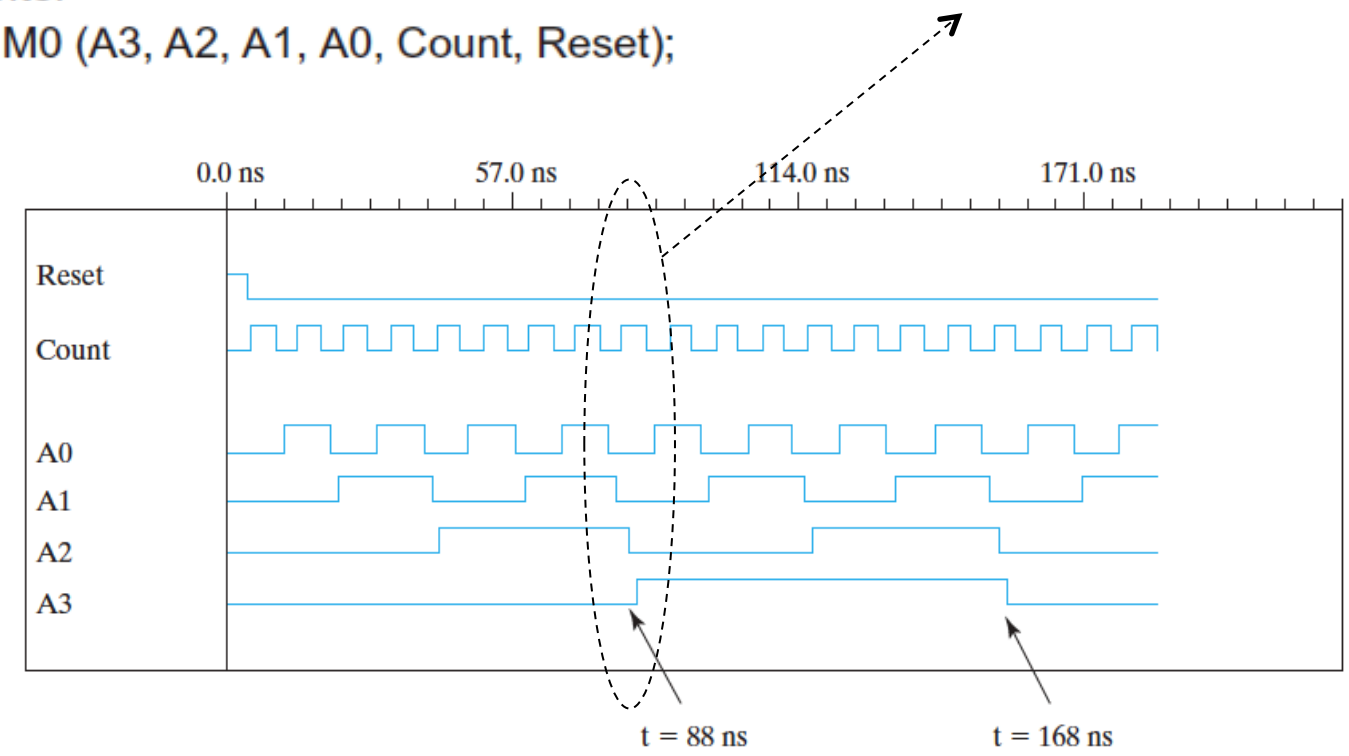
```
module Ripple_Counter_4bit (A3, A2, A1, A0, Count, Reset);
   output A3, A2, A1, A0;
   input Count, Reset;
// Instantiate complementing flip-flop
   Comp_D_flip_flop F0 (A0, Count, Reset);
   Comp_D_flip_flop F1 (A1, A0, Reset);
   Comp_D_flip_flop F2 (A2, A1, Reset);
   Comp_D_flip_flop F3 (A3, A2, Reset);
endmodule
// Complementing flip-flop with delay
// Input to D flip-flop = Q'
module Comp_D_flip_flop (Q, CLK, Reset);
   output   Q;
   input    CLK, Reset;
   reg      Q;
   always @ (negedge CLK, posedge Reset)
   if (Reset) Q <= 1'b0;
   else Q <= #2 ~Q;              // intra-assignment delay
endmodule
```



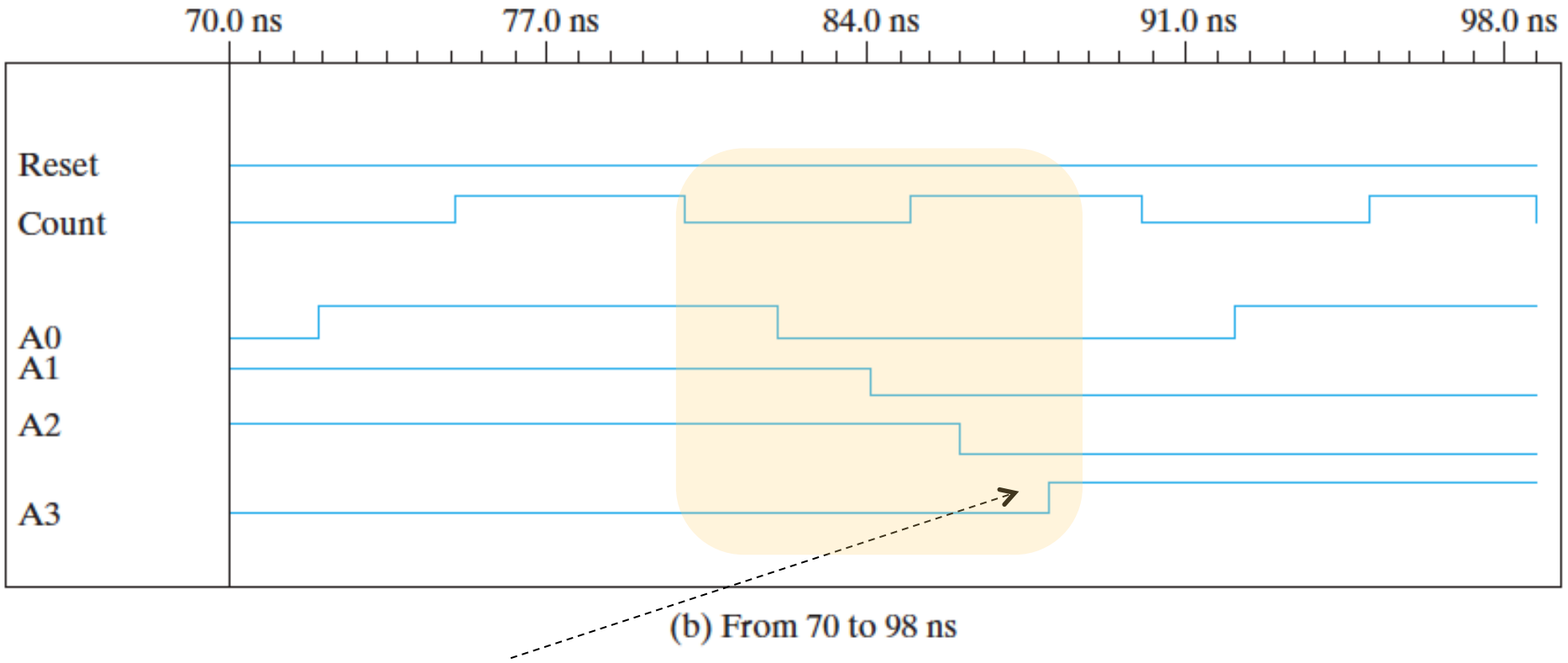
Μετρητής Ριπής

```
// Stimulus for testing ripple counter
module t_Ripple_Counter_4bit;
   reg      Count;
   reg      Reset;
   wire     A0, A1, A2, A3;
// Instantiate ripple counter
   Ripple_Counter_4bit M0 (A3, A2, A1, A0, Count, Reset);
    always
   #5 Count = ~Count;
    initial
      begin
   Count = 1'b0;
   Reset = 1'b1;
   #4 Reset = 1'b0;
     end

    initial #170 $finish;

endmodule
```

Συσσώρευση καθυστέρησης



Μετρητής Ριπής



(b) From 70 to 98 ns

Συσσώρευση καθυστέρησης → Μη αποδοτικό

Μνήμη

Η μνήμη μοντελοποιείται με διάταξη πίνακα:

Μέγεθος Λέξης <---------- **reg**[15:0] memword [0: 1023] ----> Μέγεθος Μνήμης

```
module memory (Enable, ReadWrite, Address, DataIn, DataOut);
 input   Enable, ReadWrite;
 input   [3: 0] DataIn;
 input   [5: 0] Address;
 output [3: 0] DataOut;
 reg [3: 0]      DataOut;
 reg [3: 0]      Mem [0: 63];                    // 64 x 4 memory
 always @ (Enable or ReadWrite)
  if (Enable)
    if (ReadWrite) DataOut = Mem [Address];     // Read
   else Mem [Address] = DataIn;                 // Write
  else DataOut = 4'bz;                          // High impedance state
endmodule
```