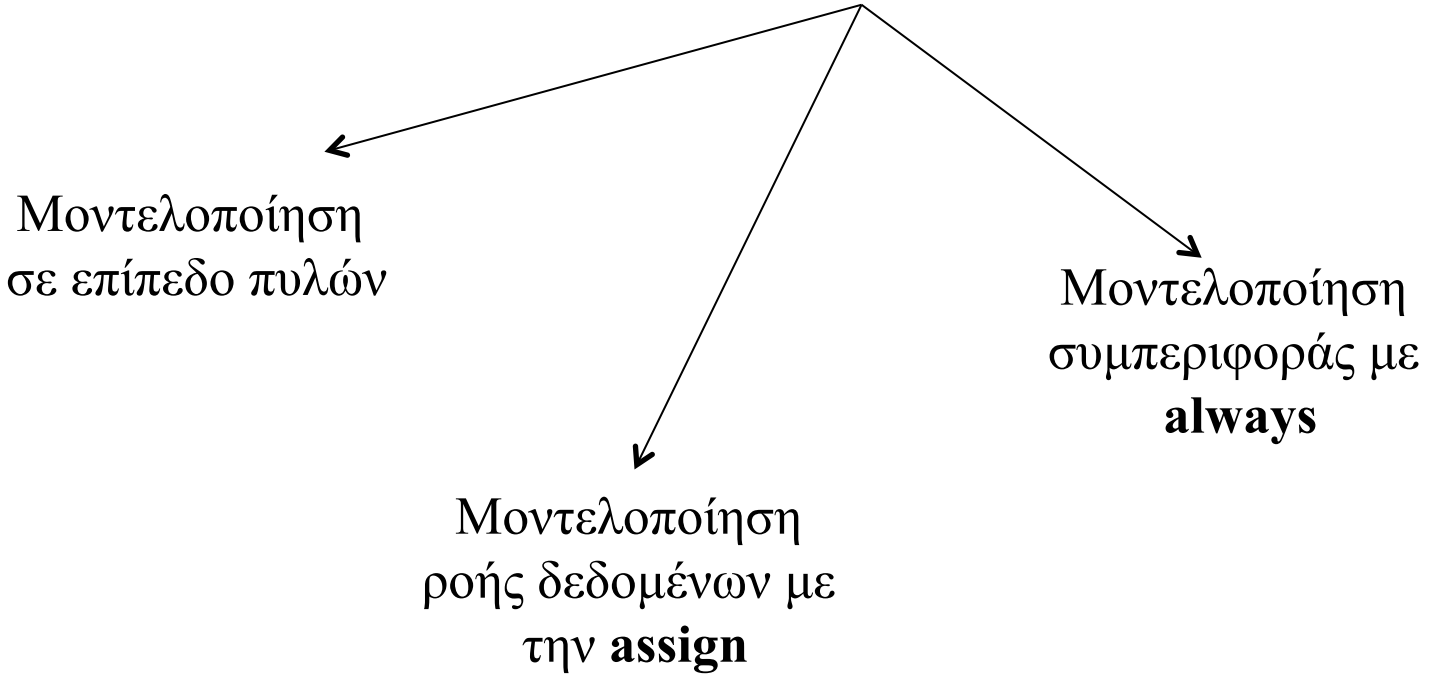
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΛΛΑΔΑ ΚΟΙΝΩΝΙΚΟ ΠΛΑΙΣΙΟ ΣΤΗΡΙΞΗΣ 2008
Ανάπτυξη παντού. Ανάπτυξη για όλους.
Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

8η Θεματική Ενότητα : Εισαγωγή στις Γλώσσες Περιγραφής Υλικού: Μοντέλα Συνδυαστικών Κυκλωμάτων

Εισαγωγή

Η λογική που περιγράφεται σε ένα module μπορεί να περιγραφεί με διάφορα στυλ



Μοντελοποίηση Πυλών

Το μοντέλο πυλών αποτελεί μία περιγραφή του δικτυώματος πυλών με κείμενο

Πρωταρχικές Πύλες (primitives)

- **and**, **nand**, **or**, **nor**, **xor**, **xnor** — Μονής εξόδου / πολλαπλών εισόδων
- **not**, **buf** — Μονής εισόδου / πολλαπλών εξόδων

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

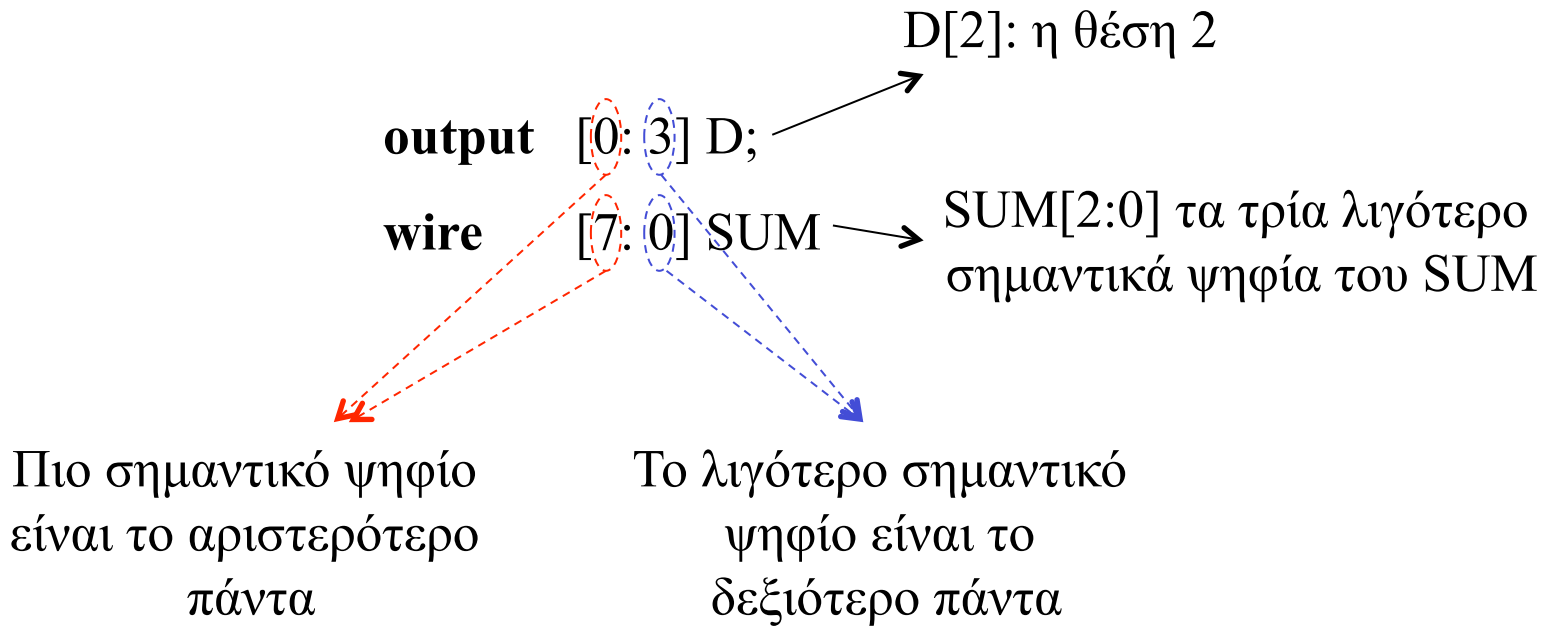
or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	input	output
	0	1
	1	0
	x	x
	z	x

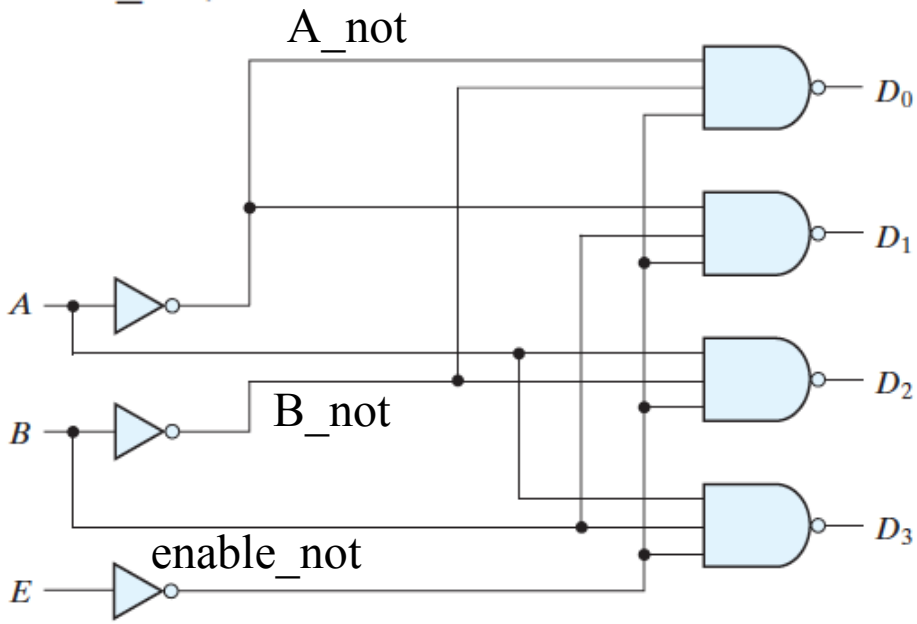
Μοντελοποίηση Πυλών

Σε όλους τους τύπους περιγραφών χειριζόμαστε διανύσματα (vectors)



Παράδειγμα: Αποκωδικοποιητής 2 σε 4

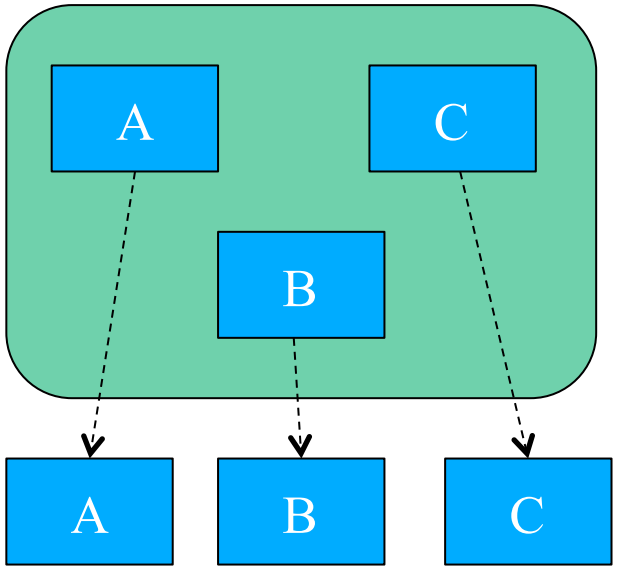
```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
  output      [0: 3]          D;
  input                       A, B;
  input                       enable;
  wire                        A_not,B_not, enable_not;

  not
    G1  (A_not, A),
    G2  (B_not, B),
    G3  (enable_not, enable);
  nand
    G4  (D[0], A_not, B_not, enable_not),
    G5  (D[1], A_not, B, enable_not),
    G6  (D[2], A, B_not, enable_not),
    G7  (D[3], A, B, enable_not);
endmodule
```

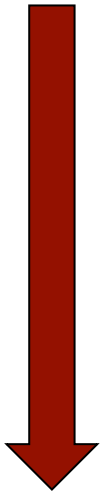


Ιεραρχική Σχεδίαση

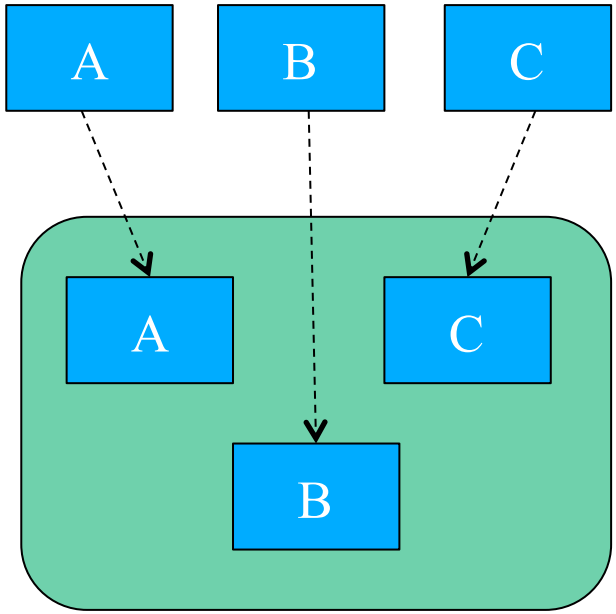
Πολλαπλά modules συνδυάζονται σε ιεραρχικές περιγραφές

Top - Down

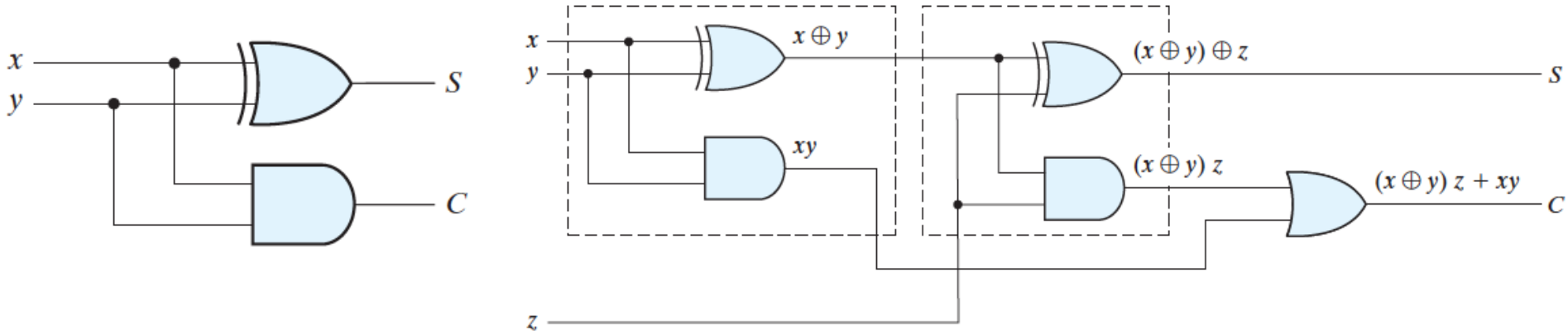




Bottom - Up



Ιεραρχική Σχεδίαση




```
module half_adder (S, C, x, y);
output   S, C;
input    x, y;

   xor (S, x, y);
   and (C, x, y);
endmodule
```

```
module full_adder (S, C, x, y, z);
output          S, C;
input           x, y, z;
 // Instantiate half adders
   half_adder HA1 (S1, C1, x, y);
   half_adder HA2 (S, C2, S1, z);
   or G1 (C, C2, C1);
endmodule
```

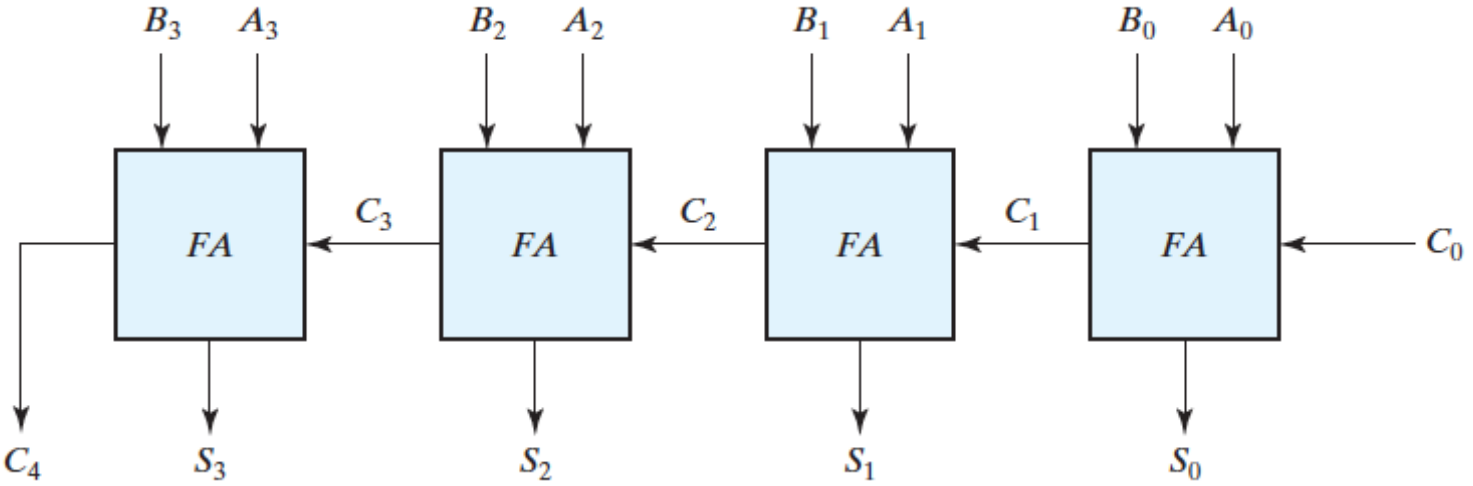
Ιεραρχική Σχεδίαση

```
module ripple_carry_4_bit_adder (Sum, C4, A, B, C0);
output [3: 0]    Sum;
output           C4;
input  [3: 0]    A, B;
input            C0;
 // Instantiate chain of full adders
 full_adder      FA0 (Sum[0], C1, A[0], B[0], C0),
                 FA1 (Sum[1], C2, A[1], B[1], C1),
                 FA2 (Sum[2], C3, A[2], B[2], C2),
                 FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
```

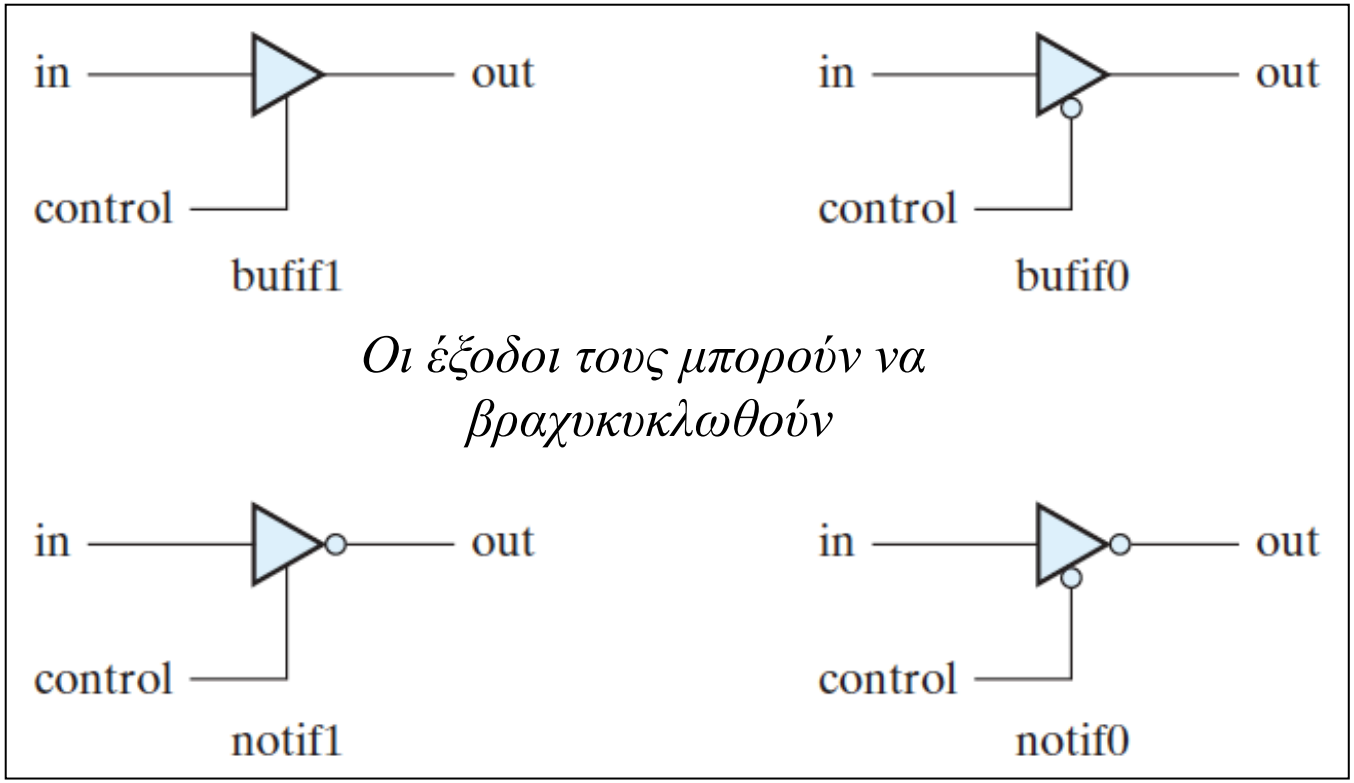
Δηλώνονται ως wires

Στιγμιότυπα

Ιεραρχική Σχεδίαση



Πύλες Τριών Καταστάσεων



bufif1, bufif0
notif1, notif0

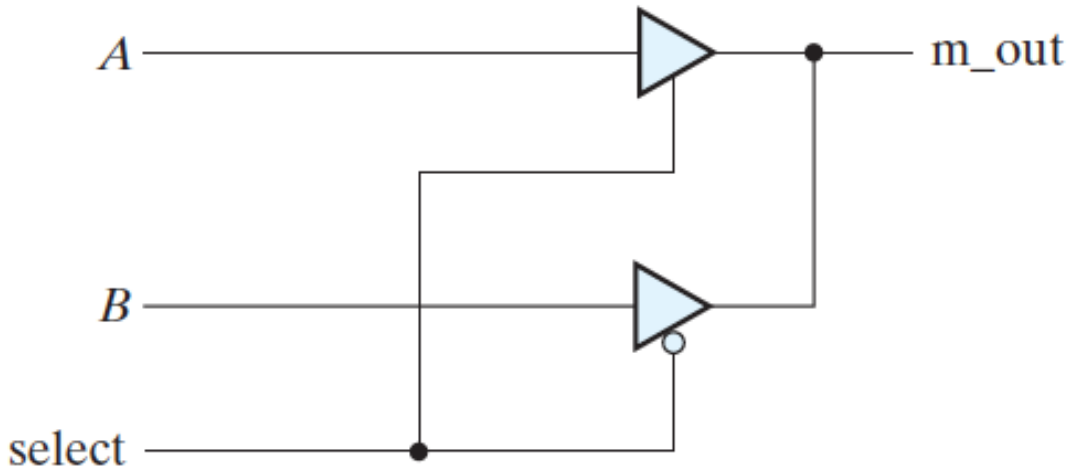
gate name (output, input, control)

Πύλες Τριών Καταστάσεων

```
// Mux with three-state output

module mux_tri (m_out, A, B, select);
  output  m_out;
  input   A, B, select;
  tri     m_out;

  bufif1 (m_out, A, select);
  bufif0 (m_out, B, select);
endmodule
```

Δήλωση πολλαπλών οδηγών

Βραχυκυκλωμένη έξοδος



Μοντελοποίηση Ροής Δεδομένων

Χρήση τελεστών σε δυαδικά δεδομένα

Symbol	Operation	Symbol	Operation
+	binary addition		
−	binary subtraction		
&	bitwise AND	&&	logical AND
|	bitwise OR	||	logical OR
^	bitwise XOR		
~	bitwise NOT	!	logical NOT
= =	equality		
>	greater than		
<	less than		
{}	concatenation		
?:	conditional		

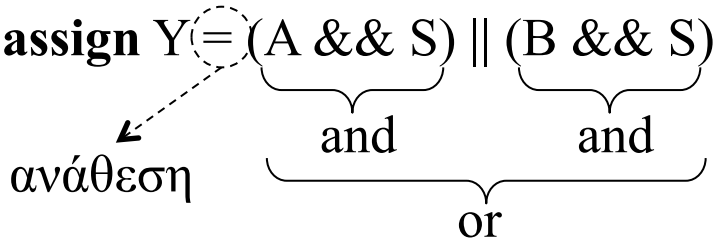
Σε διανύσματα δίνουν διαφορετικά αποτελέσματα

~(1010) = (0101), !(1010) = 0

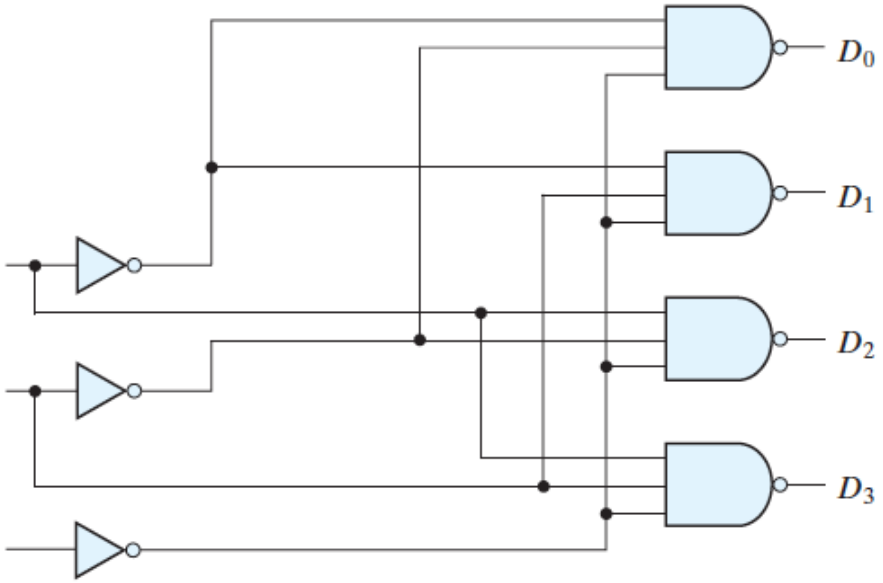
Χρησιμοποιούνται μαζί με το **assign** πάνω σε **net**

Τα **net** φέρουν λογικές τιμές

Μοντελοποίηση Ροής Δεδομένων




```
module decoder_2x4_df (
  output    [0: 3]        D,
  input                   A, B,
                          enable
);
  assign    D[0] = !((!A) && (!B) && (!enable)),
            D[1] = !(*!A) && B && (!enable)),
            D[2] = !(A &&!B && (!enable)
            D[3] = !(A && B && (!enable))
endmodule
```



Μοντελοποίηση Ροής Δεδομένων

Αθροιστής 4 ψηφίων

Περιγράφει την συνάρτηση και όχι την δομή

```
module binary_adder (
  output [3: 0]        Sum,
  output               C_out,
  input [3: 0]         A, B,
  input                C_in
);

  assign {C_out, Sum} = A + B + C_in;
endmodule
```

Τελεστής συννένωσης

Τελούμενα διαφορετικού μήκους

Μοντελοποίηση Ροής Δεδομένων

Συγκριτής 4 ψηφίων

Το σχηματικό προκύπτει με την διαδικασία της σύνθεσης

```
module mag_compare
( output          A_lt_B, A_eq_B, A_gt_B,
  input [3: 0]    A, B
);
  assign A_lt_B = (A < B);
  assign A_gt_B = (A > B);
  assign A_eq_B = (A = = B);
endmodule
```

Τελεστής σύγκρισης (ισότητας)

Μοντελοποίηση Ροής Δεδομένων

Πολυπλέκτης 2 σε 1

```
module mux_2x1_df(m_out, A, B, select);
  output      m_out;
  input       A, B;
  input       select;

  assign m_out = (select)? A : B;
endmodule
```

Τελεστής επιλογής ?:

if (select = 1) then m_out = A else m_out = B

Μοντελοποίηση Συμπεριφοράς

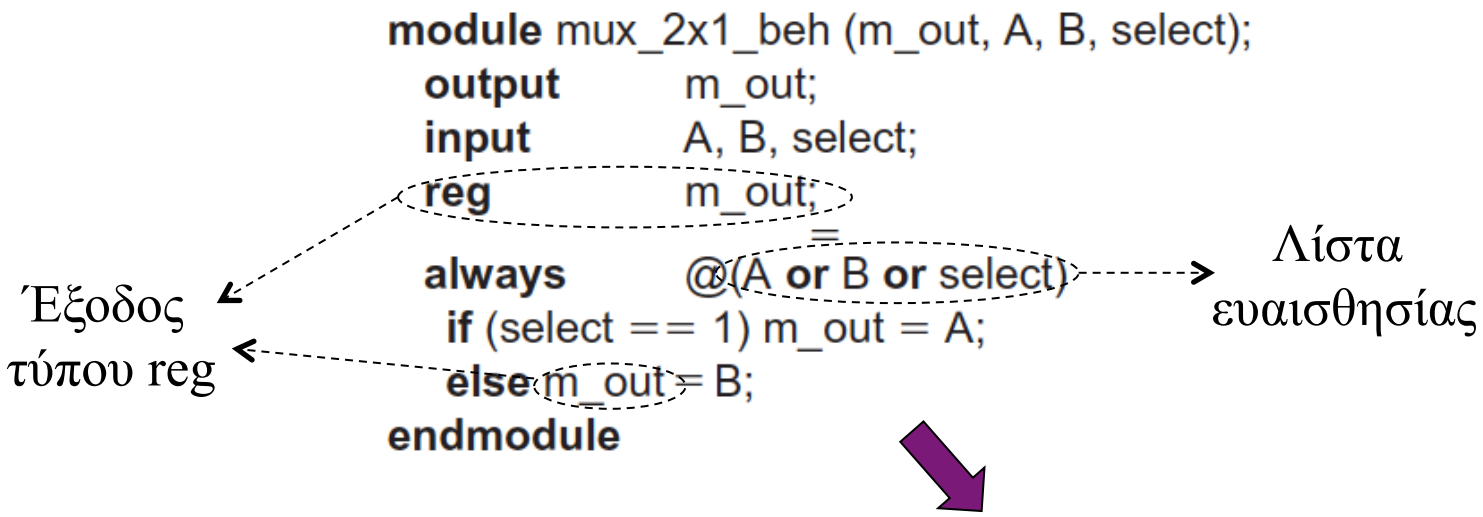
Περιγράφει κυκλώματα σε αλγοριθμικό επίπεδο → Αναπαριστάνει κυκλώματα σε λειτουργικό και αλγοριθμικό επίπεδο

always @ (event list)

- Η ανάθεση γίνεται υποχρεωτικά σε μεταβλητή τύπου **reg** → Δεν ανανεώνεται αυτόματα αλλά κρατά την τιμή του μέχρι να επανεκτελεστεί το **always**
- Καθορίζει πότε θα εκτελεστεί το always statement → Απαιτεί αλλαγή τιμής ενός στοιχείου της λίστας

Μοντελοποίηση Συμπεριφοράς

Πολυπλέκτης 2 σε 1



Μοντελοποίηση Συμπεριφοράς

Πολυπλέκτης 4 σε 1

(συντακτικό 2001-2005)

Δυαδικοί αριθμοί των 2 δυαδικών ψηφίων

d' : decimal,

o' : octal,

h' : hexadecimal

```
module mux_4x1_beh
( output reg m_out,
  input      in_0, in_1, in_2, in_3,
  input [1: 0] select
);
always @ (in_0, in_1, in_2, in_3, select)
  case (select)
    2'b00:    m_out = in_0;
    2'b01:    m_out = in_1;
    2'b10:    m_out = in_2;
    2'b11:    m_out = in_3;
  endcase
endmodule
```

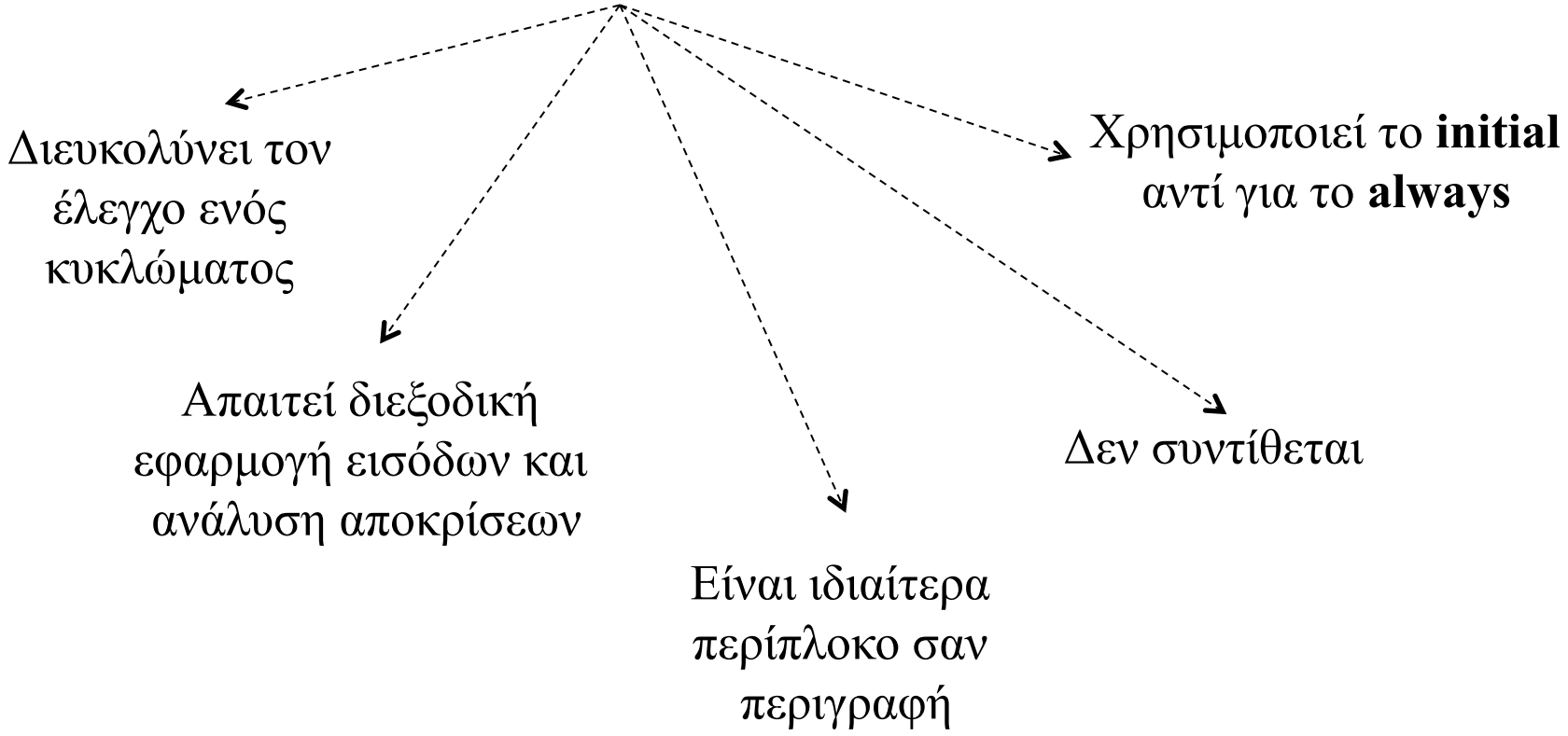
Ο έλεγχος γίνεται με την σειρά

Χωρίς καθορισμό μεγέθους θεωρείται αυτόματα ίσο με 32 δυαδικά ψηφία

Επιλογή default για τις υπόλοιπες τιμές

Φτιάχνοντας ένα Test Bench

Test Bench: μία περιγραφή HDL για την παραγωγή και εφαρμογή εισόδων σε ένα κύκλωμα, και την παρακολούθηση των αποκρίσεων



Εφαρμογή Εισόδων

10 units μετά το προηγούμενο γεγονός

Χρόνος : 30 units

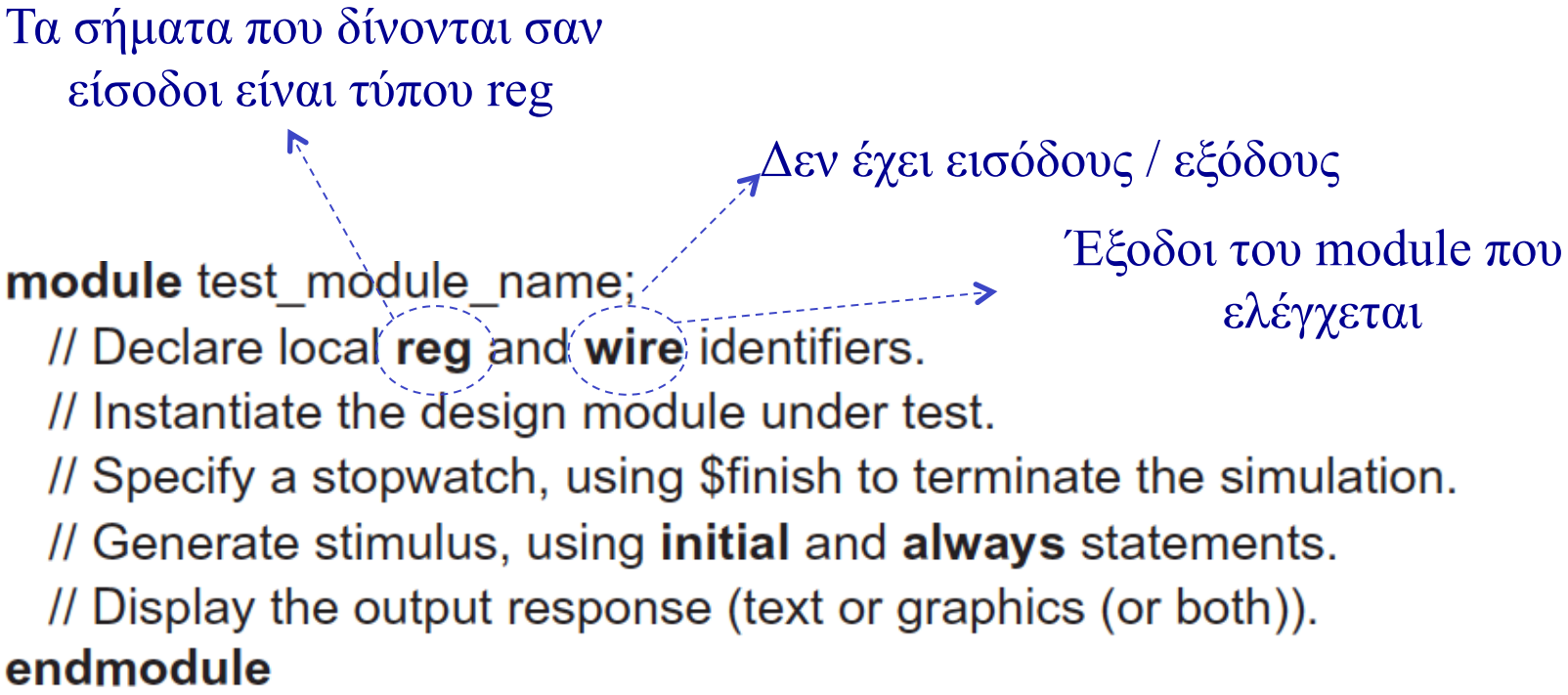
```
initial
  begin
      A = 0; B = 0;
  #10 A = 1;
  #20 A = 0; B = 1;
  end
```

Εκτελείται μόνο μία φορά

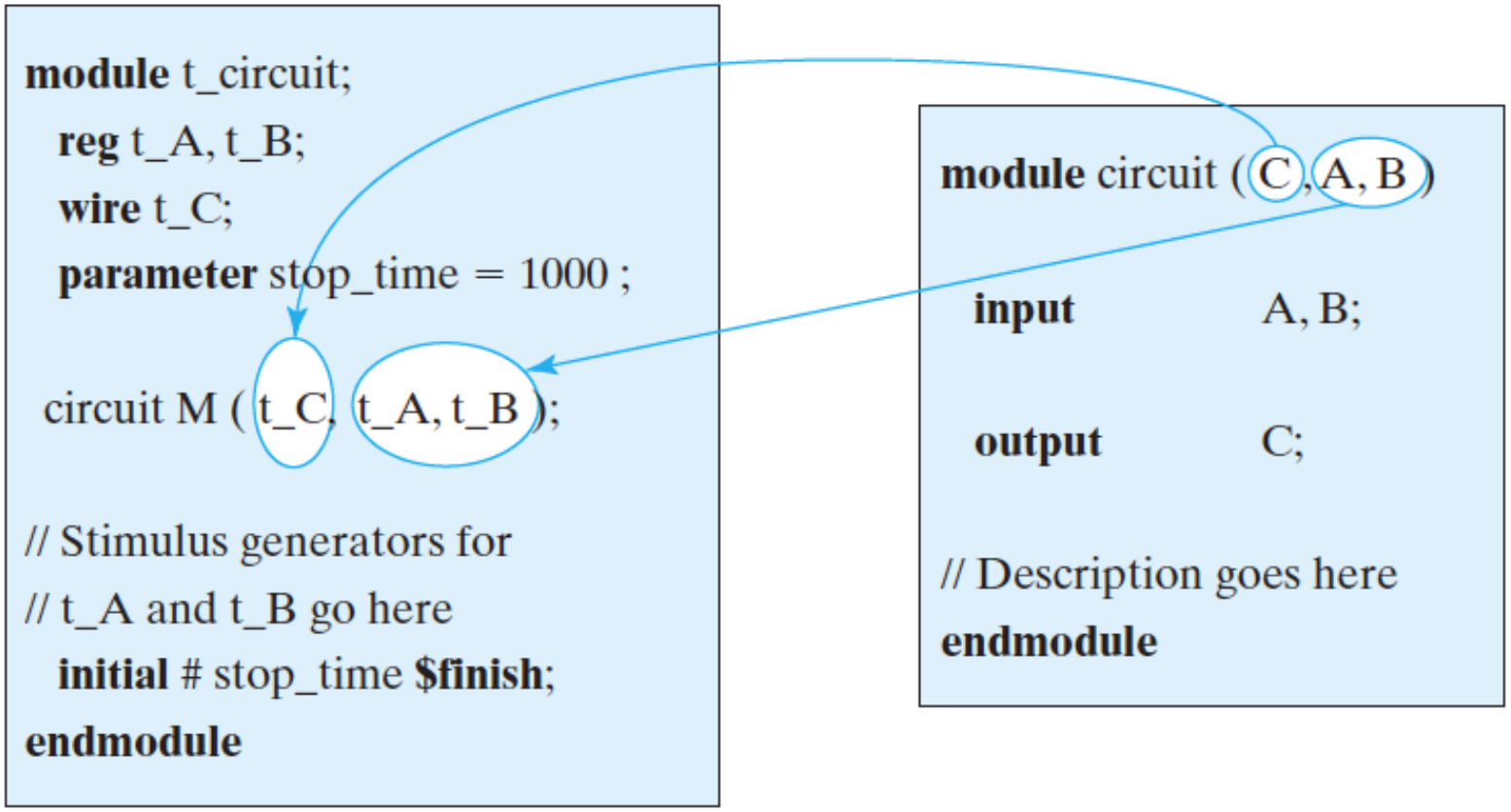
Το D λαμβάνει όλες τις δυνατές δυαδικές τιμές 3 ψηφίων ανά 10 μονάδες χρόνου

```
initial
  begin
    D = 3'b000;
    repeat (7)
    #10 D = D + 3'b001;
  end
```

Μοντέλο Test Bench



Μοντέλο Test Bench



Δυνατότητες Προβολής Αποτελεσμάτων

$display—display a one-time value of variables or strings with an end-of-line return,
$write**—same as **$display, but without going to next line,
$monitor—display variables whenever a value changes during a simulation run,
$time—display the simulation time,
$finish—terminate the simulation.

Σύνταξη: *Task-name (format specification, argumentlist)*

$display ("%d %b %b", C, A, B)

Παρακάμπτει τα αρχικά κενά

$display ("time = %0d A = %b B = %b", **$time**, A, B)

time = 3 A = 10 B = 1

Παράδειγμα

Έξοδοι προς παρακολούθηση

```
module mux_2x1_df (m_out, A, B, select);
  output    m_out;
  input     A, B;
  input     select;

  assign m_out = (select)? A : B;
endmodule
```

Είσοδοι δεδομένων

Πλήρης Έλεγχος:

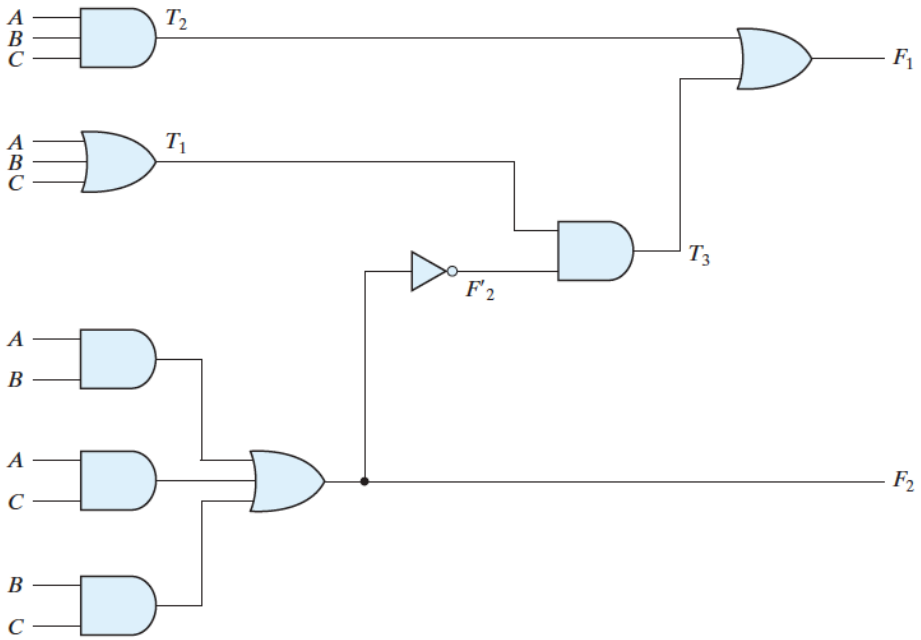
Select, A, B	
1 0 1	Επιλογή μίας εισόδου και θέση της άλλης στην συμπληρωματική της τιμή
1 1 0	
0 1 0	
0 0 1	

Παράδειγμα

```
module t_mux_2x1_df;
 wire          t_mux_out;
 reg           t_A, t_B;
 reg           t_select;
 parameter stop_time = 50;

mux_2x1_df M1 (t_mux_out, t_A, t_B, t_select);        // Instantiation of circuit to be tested

initial # stop_time $finish;

 initial begin                                        // Stimulus generator
        t_select = 1; t_A = 0; t_B = 1;
   #10 t_A = 1; t_B = 0;
   #10 t_select = 0;
   #10 t_A = 0; t_B = 1;
 end

 initial begin                                        // Response monitor
  // $display ("   time   Select   A   B   m_out");
  // $monitor ($time,, " %b   %b   %b   %b ", t_select, t_A, t_B, t_m_out);
    $monitor ("time =", $time,, "select = %b A = %b B = %b OUT = %b",
    t_select, t_A, t_B, t_mux_out);
  end
 endmodule
```

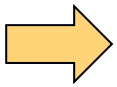
Παρακολουθεί τα σήματα και εκτυπώνει τις τιμές σε κάθε αλλαγή σήματος

Παράδειγμα 2




```
module Circuit_of_Fig_4_2 (A, B, C, F1, F2);
  input  A, B, C;
  output F1, F2;
  wire   T1, T2, T3, F2_b, E1, E2, E3;
  or   g1 (T1, A, B, C);
  and g2 (T2, A, B, C);
  and g3 (E1, A, B);
  and g4 (E2, A, C);
  and g5 (E3, B, C);
  or   g6 (F2, E1, E2, E3);
  not  g7 (F2_b, F2);
  and g8 (T3, T1, F2_b);
  or   g9 (F1, T2, T3);
endmodule
```

```
wire F1, F2;
Circuit_of_Fig_4_2 (D[2], D[1], D[0], F1, F2);
initial
 begin
   D = 3'b000;
   repeat (7) #10 D = D + 1'b1;
 end
initial
$monitor ("ABC = %b F1 = %b F2 =%b", D, F1, F2);
```



```
Simulation log:  ABC = 000 F1 = 0 F2 =0
ABC = 001 F1 = 1 F2 =0 ABC = 010 F1 = 1 F2 =0
ABC = 011 F1 = 0 F2 =1 ABC = 100 F1 = 1 F2 =0
ABC = 101 F1 = 0 F2 =1 ABC = 110 F1 = 0 F2 =1
ABC = 111 F1 = 1 F2 =1
```