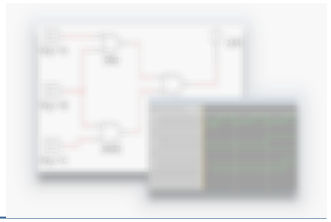


ECE119 – Ψηφιακή Σχεδίαση

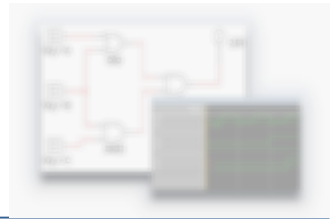
Διδάσκοντες Εργαστηρίου: Δ. Καραμπερόπουλος
Α. Φεύγας

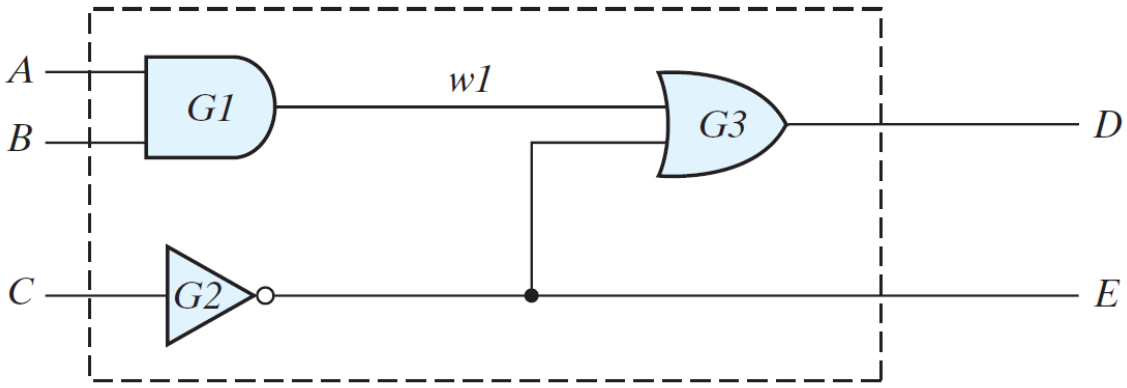
- Lab 3: Verilog (Μέρος 2)

Η Γλώσσα Verilog (Μέρος 2)



Η Γλώσσα Verilog

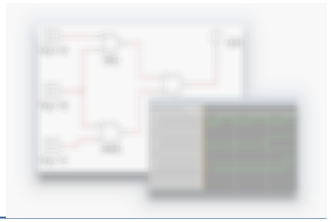





```
module Simple_Circuit (A, B, C, D, E);
  output      D, E;
  input       A, B, C;
  wire        w1;

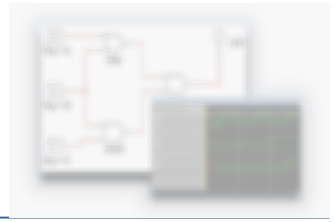
  and         (w1, A, B);
  not         (E, C);
  or          (D, w1, E);
endmodule
```

Καθυστερήσεις διάδοσης πυλών

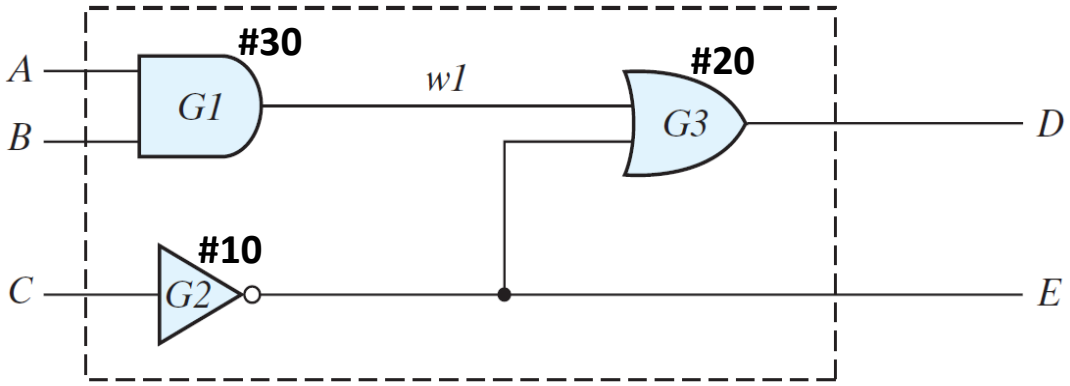


- Όλα τα φυσικά κυκλώματα εμφανίζουν **καθυστέρηση διάδοσης** σημάτων (propagation delay)
- Στη Verilog η καθυστέρηση διάδοσης μιας πύλης δίνεται σε χρονικές μονάδες (time units) και δηλώνεται με το #
- Οι τιμές των χρονικών καθυστερήσεων στη Verilog είναι αδιάστατες.
- Η απόδοση φυσικής τιμής σε μία τέτοια μονάδα χρόνου γίνεται με την οδηγία (directive) **`timescale** πριν την δήλωση του module.
- Π.χ. **`timescale 1ns/100ps** (μονάδα μέτρησης / ακρίβεια)
- Αν δεν καθοριστεί τότε default = 1ns

Καθυστερήσεις διάδοσης πυλών



- Αν οι πύλες **and**, **or** και **not** έχουν καθυστέρηση διάδοσης **30ns**, **20ns** και **10ns** αντίστοιχα τότε:




```
module Simple_Circuit_prop_delay (A, B, C, D, E);
  output    D, E;
  input     A, B, C;
  wire      w1;

  and       #(30) (w1, A, B);
  not       #(10) (E, C);
  or        #(20) (D, w1, E);
endmodule
```

Output of Gates after Delay

	Time Units (ns)	Input ABC	Output E w1 D
Initial	—	0 0 0	1 0 1
Change	—	1 1 1	1 0 1
	10	1 1 1	0 0 1
	20	1 1 1	0 0 1
	30	1 1 1	0 1 0
	40	1 1 1	0 1 0
	50	1 1 1	0 1 1

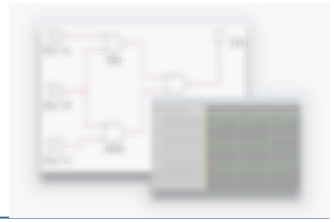
20ns

Αρχ. Κατ. ABC=000

ABC=111

Τελ. Κατ. ABC=111

Καθυστερήσεις διάδοσης πυλών



- Εμφάνιση στην έξοδο του κυκλώματος (D) ένας παρασιτικός αρνητικός σπινθήρας (spike)

Output of Gates after Delay

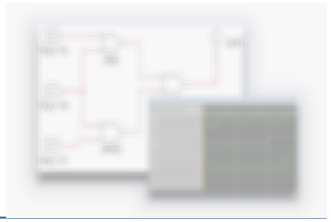
	Time Units (ns)	Input ABC	Output E w1 D
Initial	—	0 0 0	1 0 1
Change	—	1 1 1	1 0 1
	10	1 1 1	0 0 1
	20	1 1 1	0 0 1
	30	1 1 1	0 1 0
	40	1 1 1	0 1 0
	50	1 1 1	0 1 1

Αρχική κατάσταση για ABC=000

Ανεπιθύμητος σπινθήρας

Τελική κατάσταση για ABC=111

Μονάδα δοκιμής (Test Bench)



- Μια ρουτίνα κώδικα HDL, ειδικά δημιουργημένη **για την παραγωγή εισόδων** κατάλληλων για την προσομοίωση ενός κυκλώματος
- Στην απλούστερη μορφή της περιέχει μια **γεννήτρια σήματος** και ένα **στιγμιότυπο του μοντέλου**, του οποίου η λειτουργία πρέπει να επαληθευτεί.

```
// Test bench for Simple_Circuit_prop_delay

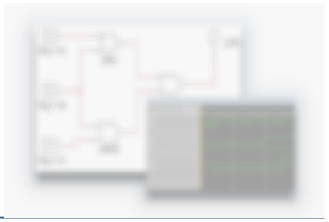
module  t_Simple_Circuit_prop_delay;
  wire     D, E;
  reg      A, B, C;

Simple_Circuit_prop_delay M1 (A, B, C, D, E); // Instance name required

initial
  begin
    A = 1'b0; B = 1'b0; C = 1'b0;
    #100 A = 1'b1; B = 1'b1; C = 1'b1;
  end

  initial #200 $finish;
endmodule
```

Μονάδα δοκιμής (Test Bench)



- Δεν έχει θύρες εισόδου - εξόδου, επειδή δεν αλληλεπιδρά με το (κυκλωματικό) περιβάλλον της.

- **reg**: Οι είσοδοι του κυκλώματος
- **wire**: Οι έξοδοι του κυκλώματος
- Χρησιμοποιούμε ένα **στιγμιότυπο της υπομονάδας** που θέλουμε να ελέγξουμε.

```
// Test bench for Simple_Circuit_prop_delay

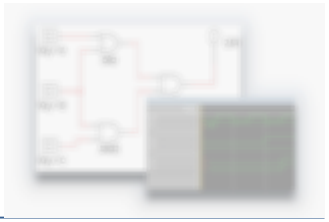
module  t_Simple_Circuit_prop_delay;
  wire     D, E;
  reg      A, B, C;

Simple_Circuit_prop_delay M1 (A, B, C, D, E); // Instance name required

initial
  begin
    A = 1'b0; B = 1'b0; C = 1'b0;
    #100 A = 1'b1; B = 1'b1; C = 1'b1;
  end

  initial #200 $finish;
endmodule
```

Μονάδα δοκιμής (Test Bench)



- **initial**: ένα μπλοκ εντολών που οριοθετούνται μεταξύ των δεσμευμένων λέξεων **begin** και **end**.
  - Εκτελείται μια φορά, στην εκκίνηση της προσομοίωσης.
  - Οι εντολές αυτές εκτελούνται **σειριακά** από τον προσομοιωτή από πάνω προς τα κάτω.
  - Δημιουργούν τις **δοκιμαστικές εισόδους** του υπό σχεδίαση κυκλώματος.
- Μια δεύτερη initial χρησιμοποιεί την λειτουργία συστήματος (system task) **$finish** για να ορίσει τον τερματισμό της προσομοίωσης.

```
// Test bench for Simple_Circuit_prop_delay

module  t_Simple_Circuit_prop_delay;
 wire    D, E;
 reg     A, B, C;

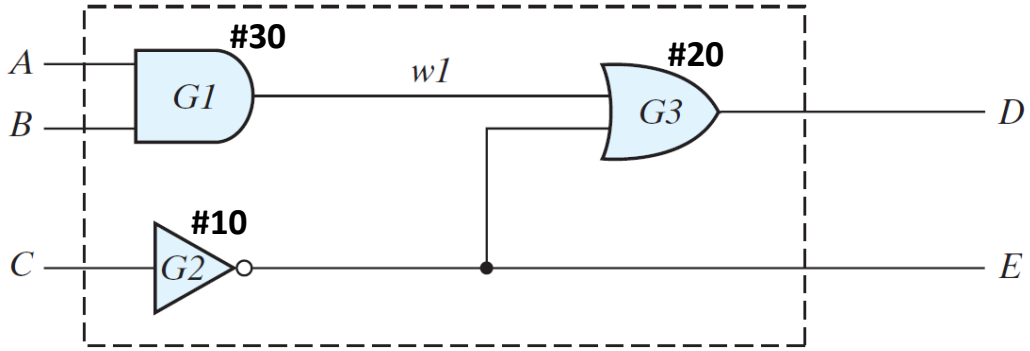
Simple_Circuit_prop_delay M1 (A, B, C, D, E); // Instance name required

initial
  begin
    A = 1'b0; B = 1'b0; C = 1'b0;
    #100 A = 1'b1; B = 1'b1; C = 1'b1;
  end

 initial #200 $finish;
endmodule
```

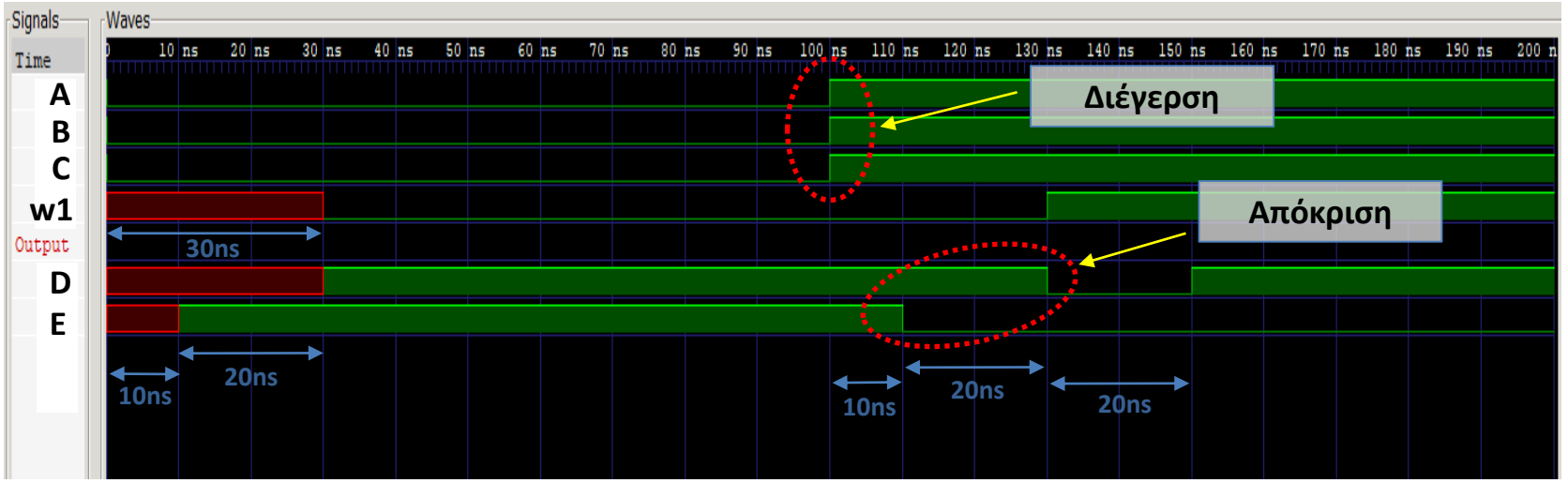
Μονάδα δοκιμής (Test Bench)

- Το διάγραμμα χρονισμού των κυματομορφών, όπως προκύπτει από την προσομοίωση:




```
module Simple_Circuit_prop_delay (A, B, C, D, E);
 output D, E;
 input  A, B, C;
 wire   w1;

 and                #(30) G1 (w1, A, B);
 not                #(10) G2 (E, C);
 or                 #(20) G3 (D, w1, E);
endmodule
```




```
module t_Simple_Circuit_prop_delay;
 wire   D, E;
 reg    A, B, C;

Simple_Circuit_prop_delay M1 (A, B, C, D, E);

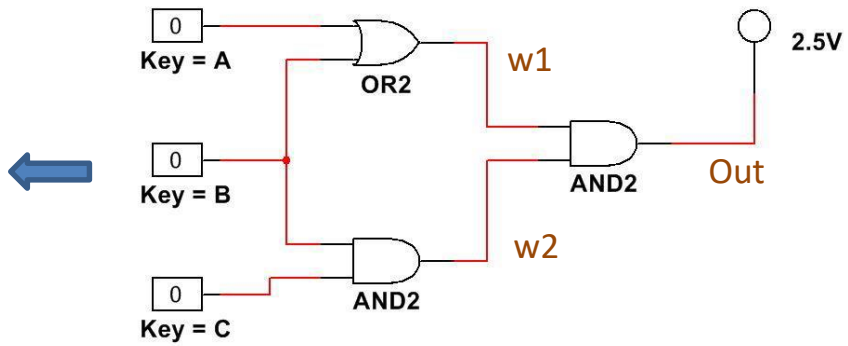
initial
 begin
   A = 1'b0; B = 1'b0; C = 1'b0;
   #100 A = 1'b1; B = 1'b1; C = 1'b1;
 end

 initial #200 $finish;
endmodule
```

Μονάδα δοκιμής (Test Bench)

```
module Figure_2_10 (A, B, C, Out);
output     Out;
input      A, B, C;
wire       w1, w2;

or         G1(w1, A, B);
and        G2(w2, B, C);
and        G3(Out, w1, w2);
endmodule
```



Όνομα αρχείου: Figure2_10.v

```
iverilog -o figure2_10 figure2_10.v
vvp figure2_10
```

```
input= 0 0 0 Out= 0
input= 0 0 1 Out= 0
input= 0 1 0 Out= 0
input= 0 1 1 Out= 1
input= 1 0 0 Out= 0
input= 1 0 1 Out= 0
input= 1 1 0 Out= 0
input= 1 1 1 Out= 1
```

Μονάδα δοκιμής (Test Bench)

```
module tb;
reg A, B, C;
wire Out;
Figure_2_10 dut(A, B, C, Out); // Instance name Υποχρεωτικό

initial begin
        A = 0; B = 0; C = 0;
    #10 A = 0; B = 0; C = 1;
    #10 A = 0; B = 1; C = 0;
    #10 A = 0; B = 1; C = 1;
    #10 A = 1; B = 0; C = 0;
    #10 A = 1; B = 0; C = 1;
    #10 A = 1; B = 1; C = 0;
    #10 A = 1; B = 1; C = 1;
end
initial $monitor("input= %b %b %b Out= %b",A, B, C, Out);
endmodule
```

Χρησιμοποιούμε ένα στιγμιότυπο της υπομονάδας που θέλουμε να ελέγξουμε

Παραγωγή κατάλληλων εισόδων για την προσομοίωση της λειτουργίας του κυκλώματος

Παρακολουθεί τα σήματα και τα τυπώνει όταν αποκτήσουν νέα τιμή – ανάλογη της printf

Μονάδα δοκιμής (Test Bench)

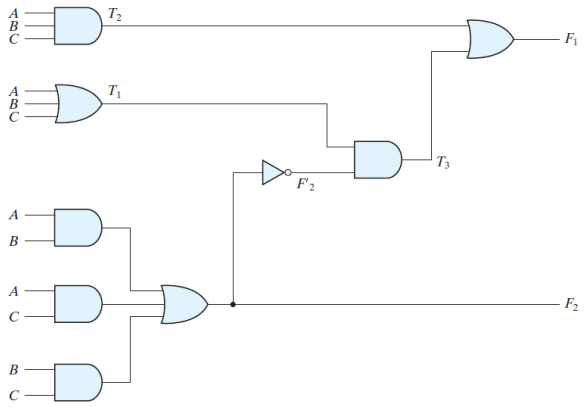


FIGURE 4.2
Logic diagram for analysis example

```
module circuit1 (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
```

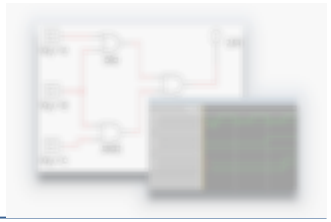
```
module test_circuit;
reg     [2:0]    D;
wire    F1, F2;

//Δημιουργία στιγμιότυπου για το κύκλωμα
circuit1    dut(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);
endmodule
```

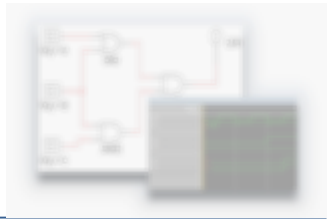
```
ABC = 000  F1 = 0  F2 = 0
ABC = 001  F1 = 1  F2 = 0
ABC = 010  F1 = 1  F2 = 0
ABC = 011  F1 = 0  F2 = 1
ABC = 100  F1 = 1  F2 = 0
ABC = 101  F1 = 0  F2 = 1
ABC = 110  F1 = 0  F2 = 1
ABC = 111  F1 = 1  F2 = 1
Execution finished with exit code 0
```

Διαδικασίες Συστήματος



Όνομα Διαδικασίας	Λειτουργία
`$time`	Επιστρέφει τον χρόνο της προσομοίωσης (με την μορφή ενός 64-bit vector)
`$display`	Τυπώνει τιμές σημάτων – ανάλογη της printf `$display("format-string", expr1, …, exprn);` `%d` (decimal), `%h` (hex), `%b` (binary), `%t` (time)
`$monitor`	Παρακολουθεί σήματα ως γεγονότα, και τα τυπώνει όταν αποκτήσουν νέα τιμή – έχει ανάλογα ορίσματα όπως η `$display`
`$stop`	Διακόπτει την προσομοίωση (παύση)
`$finish`	Ολοκληρώνει την προσομοίωση & έξοδος από το περιβάλλον προσομοίωσης

Διαδικασίες Συστήματος



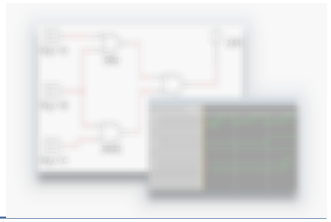
Παραδείγματα

```
$display("Error at time %t: value is %h, expected %h", $time, actual_value, expected_value);
```

```
$monitor ("Time=%2d  x=%b y=%b out=%b", $time, x, y, out)
```

```
Time= 0  x=0 y=0 out=0
Time=10  x=0 y=1 out=1
Time=20  x=1 y=0 out=1
Time=30  x=1 y=1 out=0
```

Λογικοί Τελεστές

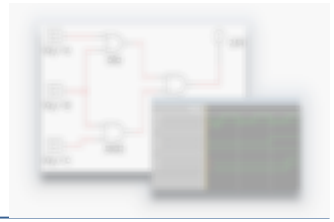


		Λογικοί
!	!m	**NOT**: Είναι το m ψευδές; (1-bit αποτέλεσμα)
&&	m && n	**AND**: Είναι το m και το n αληθή; (1-bit αποτέλεσμα)
||	m || n	**OR**: Είναι το m ή το n αληθές; (1-bit αποτέλεσμα)

- Οι τελεστέοι αξιολογούνται σαν 1-bit: 0, 1, x
- Το αποτέλεσμα της πράξης είναι 1-bit: 0, 1, x

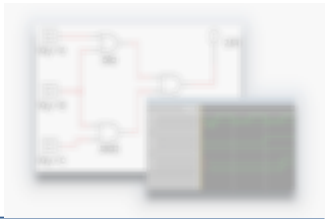
```
A = 5;     A && B     → 1 && 0     → 0
B = 0;     A || !B    → 1 || 1     → 1
C = x;     C || B     → x || 0     → x
```

Εκφράσεις Boole



- Οι εκφράσεις Boole που περιγράφουν **συνδυαστική λογική** γράφονται με μία εντολή διαρκούς ανάθεσης, την **assign**.
  - π.χ. `assign D =(A&&B)||(!C);`
  - Ο προσομοιωτής εντοπίζει τις χρονικές στιγμές αλλαγής τιμής μίας ή περισσοτέρων εισόδων (A, B, C) και κάθε φορά που συμβαίνει αυτό, ενημερώνει την τιμή της D.
- Προσδιορίζει μία **μόνιμη σχέση** μεταξύ των μεταβλητών (A, B, C) και της ανατιθέμενης τιμής (D).
- Θεωρούμε ότι έχει ένα ισοδύναμο λογικό κύκλωμα σε επίπεδο πυλών. Αυτό ονομάζεται **υποκρυπτόμενη συνδυαστική λογική** (implicit combinatorial logic).

Εκφράσεις Boole



Παράδειγμα:

Αν έχουμε να περιγράψουμε ένα κύκλωμα που ορίζεται από τις δύο εκφράσεις Boole.

E = A + BC + B'D

F = B'C + BC'D'

```
//Verilog model: Circuit with Boolean expressions

module Circuit_Boolean(E,F,A,B,C,D);
output  E,F;
input   A,B,C,D;

assign  E = A||(B&&C)||((!B)&&D);
assign  F = ((!B)&&C)||(B&&(!C)&&(!D));

endmodule
```

Μονάδες και Εμφανίσεις (Instances)

- Η διαδικασία που επικαλούμαστε μια μονάδα και την τοποθετούμε στο κύκλωμα ονομάζεται **εμφάνιση (instantiation)**

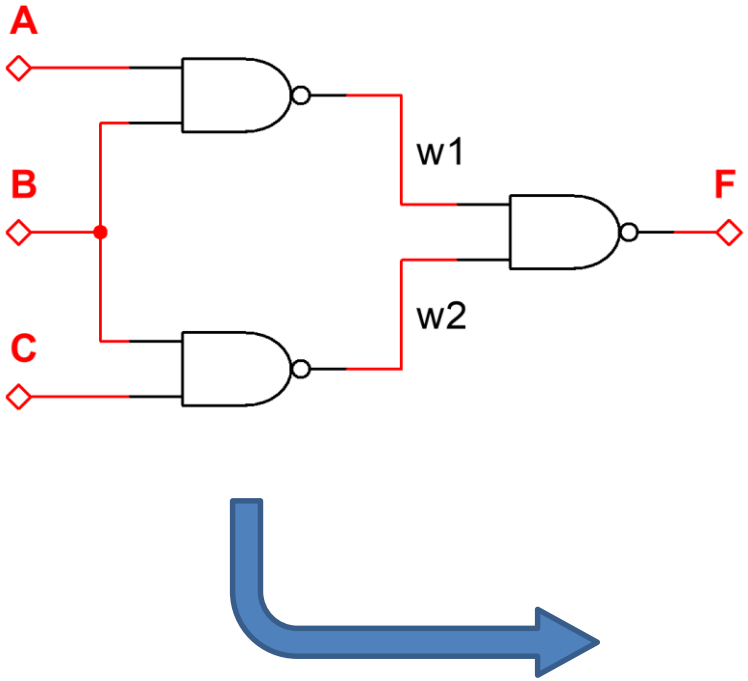
System Primitive NAND

(Είναι ενσωματωμένη στη Verilog)

```
module nand(out,a,b);
input  a,b;
output out;

assign out= !(a && b);

endmodule
```



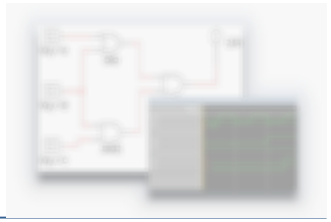
3 Instances of NAND

```
module SOP(F,A,B,C);
input  A,B,C;
output F;

// instantiation
nand (w1,A,B);
nand (w2,B,C);
nand (F,w1,w2);

endmodule
```

Στοιχειώδη Κυκλώματα του Συστήματος (system primitives)



- Οι λογικές πύλες που χρησιμοποιούνται σε περιγραφές της Verilog με τις δεσμευμένες λέξεις **and**, **or** κλπ. είναι ήδη ορισμένες από το σύστημα και αποκαλούνται στοιχειώδη κυκλώματα του συστήματος (**system primitives**).
- Αυτές οι μονάδες-στοιχεία μπορούν να χρησιμοποιηθούν ως εμφανίσεις (**instances**) σε δικά μας modules με το εξής τρόπο:
- **gate_type #(delay) instance_name [array of gate instances] (terminal, terminal, …)**

Τύπος Πύλης		Σειρά Συνδέσεων

`and`
`nand`
`xor`
	`or`
	`nor`
	`xnor`
		1 έξοδος,
		2 ή περισσότερες είσοδοι

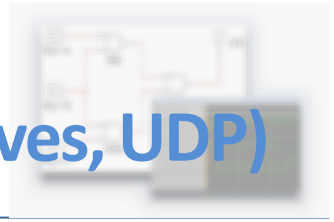
`buf`	`not`	1 έξοδος, 1 είσοδος

- Παραδείγματα:

```
and g1(out,in1,in2);
and #2 U100(out,a,b);
```

```
and #5 (o,i1,i2,i3,i4);
not g2[31:0] (y,a);
```

Στοιχειώδη κυκλώματα ορισμένα από τον χρήστη (user-defined primitives, UDP)



- Έχουμε την δυνατότητα να δημιουργήσουμε επιπλέον στοιχειώδη κυκλώματα, ορίζοντάς τα με μια μορφή πίνακα, όπως είναι π.χ. ο πίνακας αληθείας.
- Αυτοί οι τύποι κυκλωμάτων αναφέρονται ως **στοιχειώδη κυκλώματα ορισμένα από τον χρήστη** (**user-defined primitives, UDP**)
- Για τις περιγραφές των UDP χρησιμοποιείται το ζεύγος δεσμευμένων λέξεων **primitive … endprimitive**
- **Επιτρέπεται να υπάρχει μόνο μία έξοδος του κυκλώματος η οποία μπαίνει πρώτη στον κατάλογο των θυρών.**
- Μπορούν να υπάρχουν **οσεσδήποτε είσοδοι**.
- Ο πίνακας αληθείας περικλείεται από τις δεσμευμένες λέξεις **table** και **endtable**.

Στοιχειώδη κυκλώματα ορισμένα από τον χρήστη (user-defined primitives, UDP)

```
primitive UDP_1 (W,X,Y,Z);
output   W;
input    X,Y,Z;

// Truth table for W = f(X,Y,Z) = Σ(0,2,4,6,7)

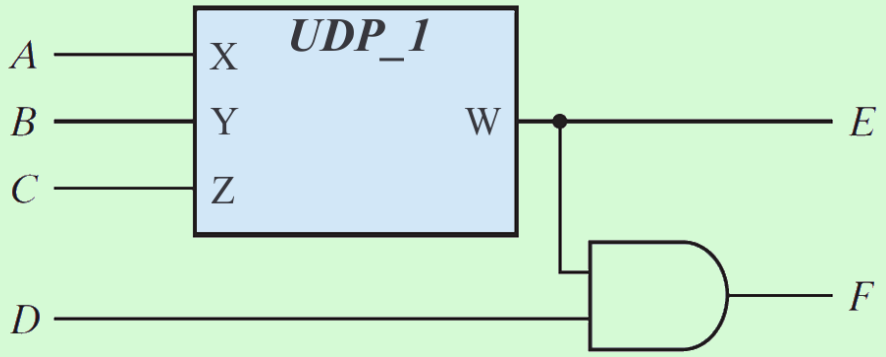
   table
      0 0 0 : 1;
      0 0 1 : 0;
      0 1 0 : 1;
      0 1 1 : 0;
      1 0 0 : 1;
      1 0 1 : 0;
      1 1 0 : 1;
      1 1 1 : 1;
   endtable

endprimitive
```

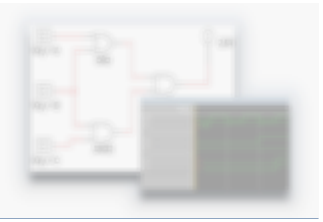
```
module example_circuit_with_UDP (E,F,A,B,C,D);
output  E,F;
input   A,B,C,D;

UDP_1    M1(E,A,B,C);  //Instance name Υποχρεωτικό
and      M2(F,E,D);    //Instance name Προαιρετικό

endmodule
```



Exercise: HDL - Verilog, Product-of Sums (POS)



- Γράψτε την περιγραφή HDL του κυκλώματος O_2 της παραγράφου 3.1 με τρεις τρόπους:
- Δημιουργήστε 3 αρχεία Verilog με τα αντίστοιχα modules:

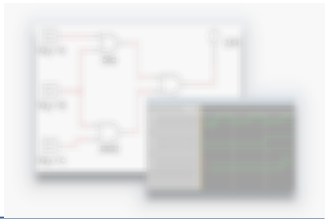
1) Με τη δημιουργία στοιχειώδους κυκλώματος (**user-defined primitive, UDP**), κάνοντας χρήση του πίνακα αληθείας του κυκλώματος.
"POS_3_1_UDP.v"

2) Με **εκφράσεις Boole**.
"POS_3_1_ Boole.v"

3) Περιγραφή **σε επίπεδο πυλών**.
"POS_3_1_ Gates.v"

A	B	C	O_1	O_2
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

Exercise: HDL - Verilog - Test Circuit



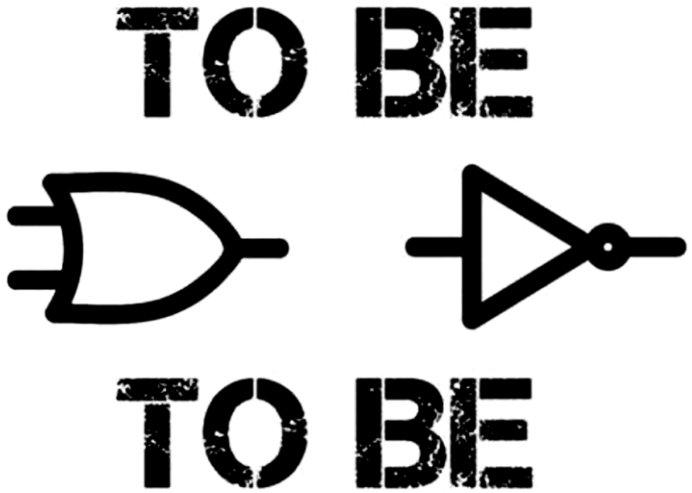
- Δημιουργήστε μια **υπομονάδα διέγερσης (test bench)** για την ανάλυση του κυκλώματος “POS_3_1_ Gates.v” που κατασκευάσατε στην προηγούμενη άσκηση.

- Δημιουργήστε την στο αρχείο που κατασκευάσατε:
  “ POS_3_1_ Gates.v ”

- Συμπληρώστε τον Πίνακα Αληθείας

Ευχαριστώ για την προσοχή σας!

- **Ερωτήσεις / Απορίες ;**



Επικοινωνία: ece119.uth@gmail.com