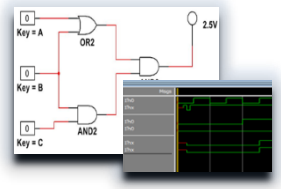
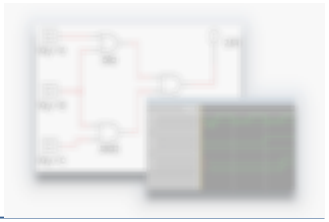


ECE119 – Ψηφιακή Σχεδίαση

Διδάσκοντες Εργαστηρίου: Δ. Καραμπερόπουλος
Δ. Γαρυφάλλου

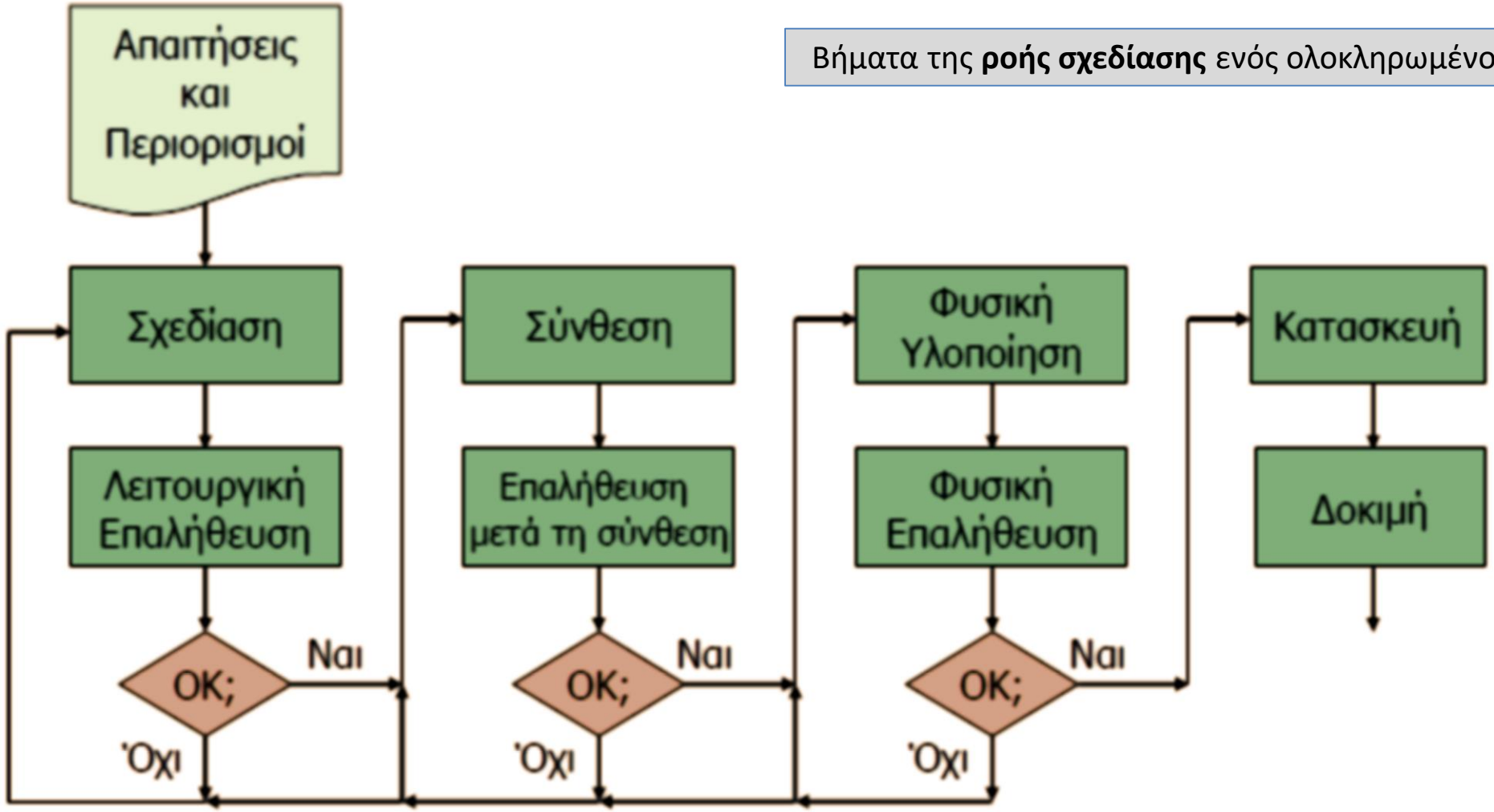
- Lab 2: Verilog (Μέρος 1)

Η Γλώσσα Verilog (Μέρος 1)



Σχεδιασμός Ψηφιακών Κυκλωμάτων: Σχεδιαστική Ροή

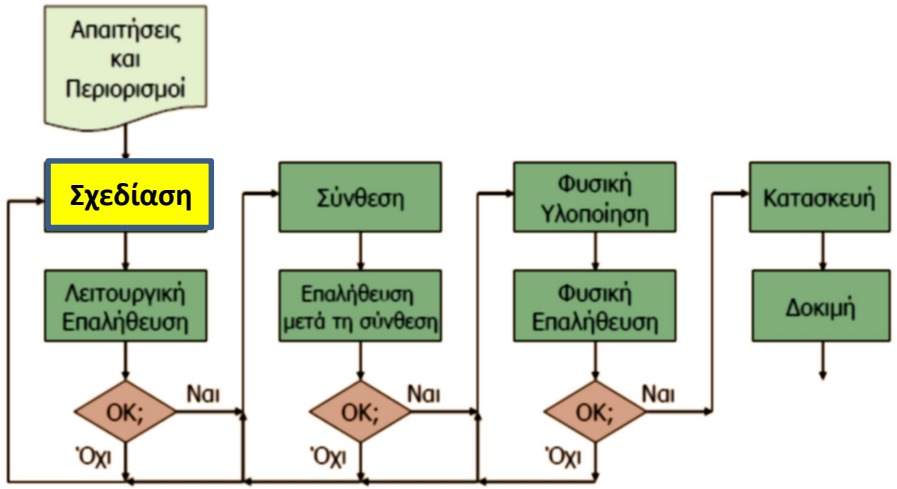
Βήματα της **ροής σχεδίασης** ενός ολοκληρωμένου κυκλώματος



Σχεδίαση

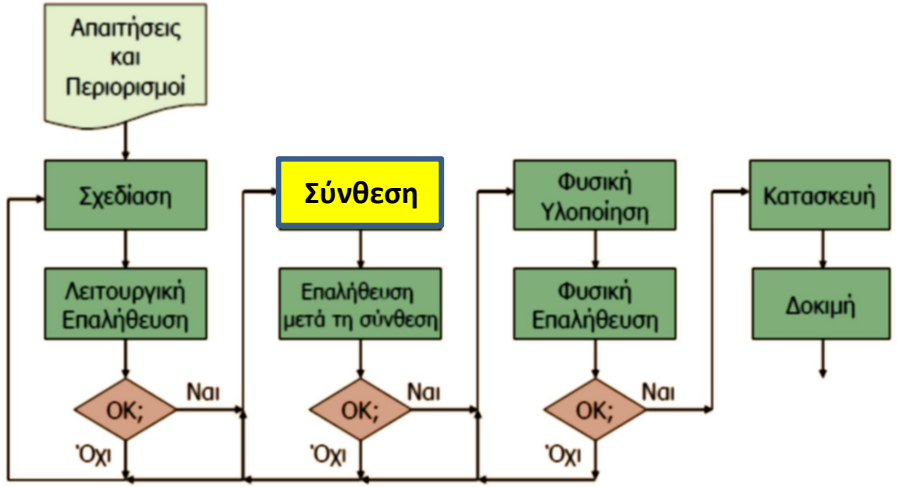
- Συνήθως, η σχεδίαση των συστημάτων γίνεται σε επίπεδο μεταφοράς καταχωρητή (**register-transfer-level – RTL**)
  - Υψηλότερο επίπεδο αφαίρεσης σε σχέση με τη σχεδίαση με πύλες

- Λειτουργική επαλήθευση: Πρόβλεψη της λειτουργικής συμπεριφοράς – Με μία μονάδα δοκιμαστικής εισόδου (test bench)



Σύνθεση (synthesis)

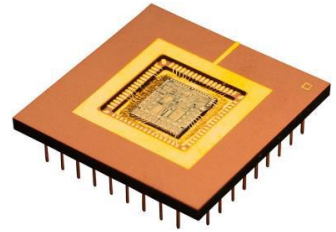
- Τα εργαλεία σύνθεσης μεταφράζουν τη σχεδίαση RTL σε ένα **κύκλωμα με πύλες** που εκτελεί την ίδια λειτουργία
- Διαδικασία εύρεσης μια λίστας φυσικών στοιχείων και των διασυνδέσεών τους
- Στο εργαλείο σύνθεσης πρέπει να καθορίσουμε:
  - Την τεχνολογία υλοποίησης
  - Περιορισμούς σε χρόνο, επιφάνεια, κτλ. (αν υπάρχουν)
- Επαλήθευση μετά τη σύνθεση (post-synthesis verification):
  - Ότι το κύκλωμα που έχει προκύψει από τη σύνθεση ικανοποιεί τους περιορισμούς



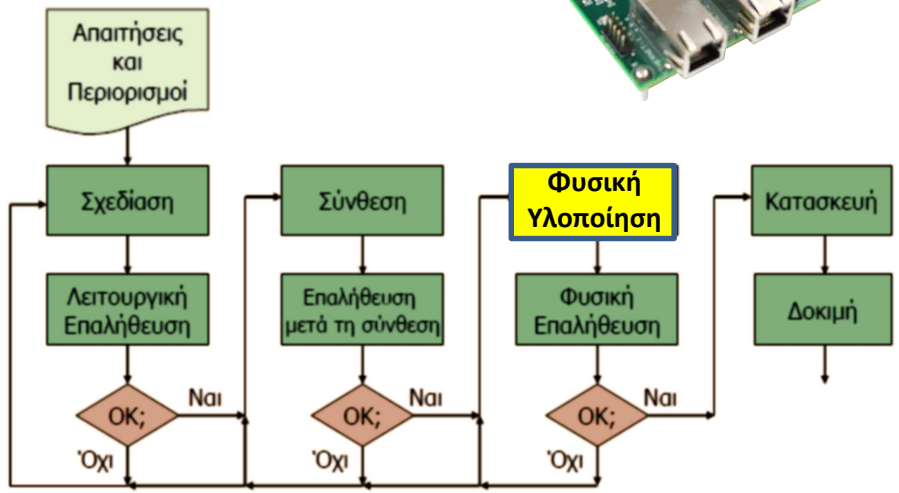
Φυσική υλοποίηση (physical implementation)

- **Δομές υλοποίησης:**
  - **A**pplication-**S**pecific **I**ntegrated **C**ircuit (ASIC)
  - **F**ield-**P**rogrammable **G**ate **A**rray (FPGA)
- **Χωροθέτηση (floor-planning)**
  - Τοποθετεί τα υποσυστήματα
- **Τοποθέτηση (placement)**
  - Τοποθετεί τις πύλες μέσα στα υποσυστήματα
- **Δρομολόγηση (routing)**
  - Συνδέει τις πύλες με αγωγούς
- **Φυσική επαλήθευση (physical verification)**
  - Το φυσικό κύκλωμα ικανοποιεί ακόμα τους περιορισμούς
  - Καλύτερη εκτίμηση των χρονικών προδιαγραφών

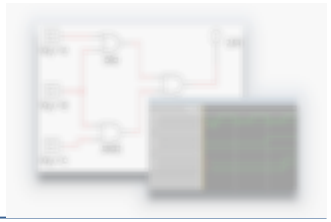
Ολοκληρωμένο Κύκλωμα Συγκεκριμένης Εφαρμογής



Πίνακας Πυλών, Προγραμματισμός στο Πεδίο



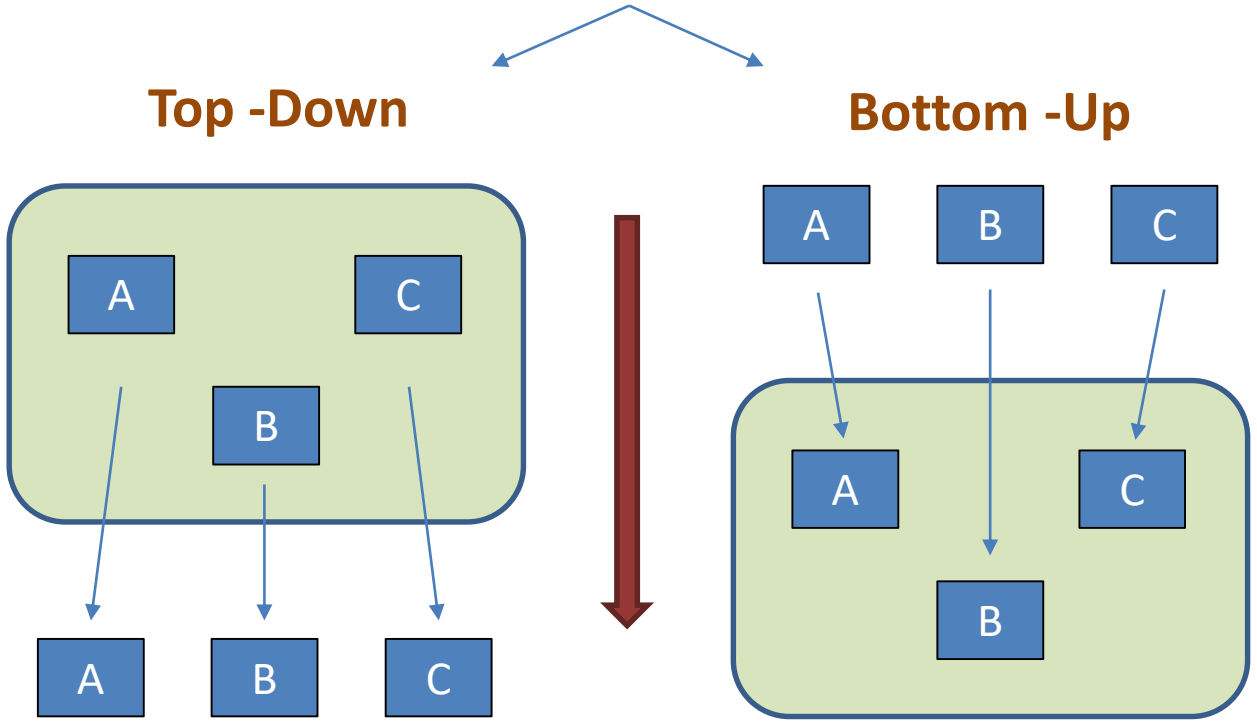
Ιεραρχική Σχεδίαση (1/2)



- Τα κυκλώματα είναι αρκετά πολύπλοκα για να σχεδιάσουμε όλες τις λεπτομέρειες σε ένα επίπεδο
- Σχεδιάζουμε **υποσυστήματα** για απλές λειτουργίες
- Συνθέτουμε υποσυστήματα για να σχηματίσουμε το σύστημα
  - Αντιμετωπίζουμε τα υποκυκλώματα ως «μαύρα κουτιά»
  - Επαληθεύουμε ανεξάρτητα, και έπειτα επαληθεύουμε την ολοκλήρωσή τους
- Σχεδίαση **top-down** (από πάνω προς τα κάτω) ή **bottom-up** (από κάτω προς τα πάνω)

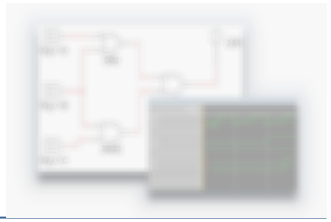
Ιεραρχική Σχεδίαση (2/2)

Πολλαπλά modules συνδυάζονται σε ιεραρχικές περιγραφές



Σχεδιάζουμε υποσυστήματα τα οποία αποτελούν μέρος ενός μεγαλύτερο συστήματος

Γλώσσες περιγραφής υλικού

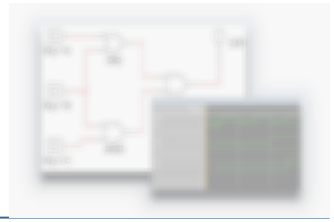


- **Hardware Description Language (HDL)**
  - Μια γλώσσα για την μοντελοποίηση της συμπεριφοράς και της δομής των ψηφιακών συστημάτων
- **Electronic Design Automation (EDA)** **using HDL** -Αυτοματοποίηση ηλεκτρονικής σχεδίασης: σχεδίαση ηλεκτρονικών κυκλωμάτων με χρήση εργαλείων CAD (computer-aided design)
  - Εισαγωγή σχεδίασης (design entry)

a category of software tools for designing electronic systems

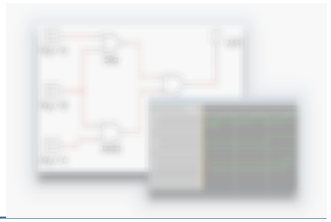
- **Κώδικας αντί για σχηματικά διαγράμματα**
  - Επαλήθευση (verification)
- **Προσομοίωση του κώδικα**
  - Σύνθεση (synthesis)
- **Αυτόματη παραγωγή των κυκλωμάτων**

Πλεονεκτήματα HDL



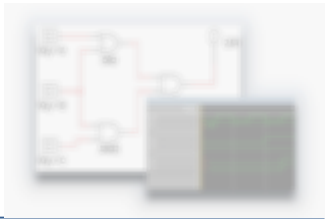
- **Υπερτερούν από τα σχηματικά διαγράμματα:**
  - Η μοντελοποίηση του συστήματος μπορεί να γίνει σε όλα τα επίπεδα (από τα υψηλότερα ως τα χαμηλότερα)
  - Η περιγραφή σε HDL είναι συνήθως πιο κατανοητή από ένα σχηματικό διάγραμμα
  - Η περιγραφή σε HDL είναι ανεξάρτητη από τις βιβλιοθήκες σχεδίασης (design libraries) και τα CAD εργαλεία
- Επιτρέπει στους σχεδιαστές να μιλάνε για το τι πρέπει να κάνει το υλικό χωρίς κάποιος να σχεδιάζει πραγματικά το ίδιο το υλικό ή με άλλα λόγια οι γλώσσες HDL επιτρέπουν στους σχεδιαστές να **διαχωρίζουν τη συμπεριφορά από την υλοποίηση** σε διάφορα επίπεδα αφαίρεσης.
- Οι σχεδιαστές μπορούν να αναπτύξουν μια εκτελέσιμη λειτουργική προδιαγραφή που να τεκμηριώνει την **ακριβή συμπεριφορά** όλων των στοιχείων και των διεπαφών τους.
- Οι σχεδιαστές μπορούν να πάρουν αποφάσεις σχετικά με το **κόστος**, τις **επιδόσεις**, την **ισχύ** και την **περιοχή** νωρίτερα στη διαδικασία σχεδιασμού.
- Οι σχεδιαστές μπορούν να **δημιουργήσουν εργαλεία** που χειρίζονται αυτόματα τον σχεδιασμό για επαλήθευση, σύνθεση, βελτιστοποίηση κλπ.

Γλώσσες περιγραφής υλικού: VHDL



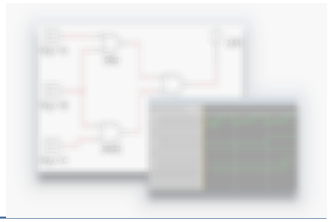
- **VHDL: V**HSIC **H**ardware **D**escription **L**anguage
  - VHSIC: Very High-Speed Integrated Circuits
- Ιστορική αναδρομή:
  - Ξεκίνησε το 1981 από το Υπουργείο Άμυνας των ΗΠΑ ως γλώσσα περιγραφής ολοκληρωμένων κυκλωμάτων
  - Οι εταιρείες IBM, Texas Instruments, Intermetrics ανάπτυξαν και κυκλοφόρησαν την 1η έκδοση το 1985
- Πρότυπο από τον οργανισμό IEEE → **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
  - IEEE Standard 1076-1987 (VHDL-87)
  - IEEE Standard 1076-1993 (VHDL-93)
  - IEEE Standard 1076-2000 (VHDL-2000)
  - IEEE Standard 1076-2002 (VHDL-2002)
  - IEEE Standard 1076-2008 (VHDL-2008)
- Πιο διαδεδομένη στην Ευρώπη

Γλώσσες περιγραφής υλικού: Verilog



- **Verilog** - Ιστορική αναδρομή:
  - Αναπτύχθηκε ως γλώσσα μοντελοποίησης υλικού από την εταιρεία Gateway Design Automation το 1984 για ιδιωτική χρήση
  - Η εταιρεία Cadence Design Systems αγόρασε την Gateway το 1990
  - Η εταιρεία Cadence είναι υπεύθυνη για την προώθηση της Verilog ως γλώσσα μοντελοποίησης & προσομοίωσης
  - Η εταιρεία Synopsys είναι υπεύθυνη για την προώθηση της Verilog ως γλώσσα σύνθεσης
- Πρότυπο από τον οργανισμό IEEE
  - IEEE Standard 1364-1995 (Verilog-95)
  - IEEE Standard 1364-2001 (Verilog-2001)
  - IEEE Standard 1364-2005 (Verilog-2005)
- Πιο διαδεδομένη στην Αμερική

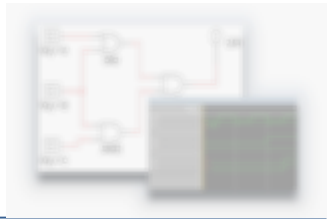
HDL: μοντελοποίηση & προσομοίωση



- Αρχικά οι γλώσσες περιγραφής υλικού (HDL) σχεδιάστηκαν για τη **μοντελοποίηση** και τη **προσομοίωση** των συστημάτων
  - Η ιδέα ήταν να εισάγουν δομές στην γλώσσα που να επιτρέπουν τη μοντελοποίηση και τη προσομοίωση του υλικού στα υψηλότερα επίπεδα αφαίρεσης

- Χαρακτηριστικά μοντελοποίησης των HDLs:
  - Παράλληλη εκτέλεση
  - Ιεραρχική σχεδίαση
  - Περιγραφή χρονισμών
  - Περιγραφή ακολουθίας γεγονότων
  - Περιγραφή σύγχρονης/ασύγχρονης συμπεριφοράς

Πώς να **ΜΗΝ** γράφετε κώδικα HDL?!

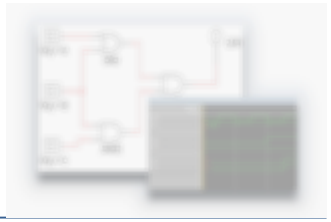


- Επειδή οι HDLs έχουν τις ρίζες τους σε γλώσσες προγραμματισμού (η VHDL στην Ada και η Verilog στην C) είναι εύκολες στην εκμάθηση αλλά δύσκολες στη σωστή χρήση τους.

- Οι αρχάριοι σχεδιαστές τείνουν να γράφουν κώδικα HDL που μοιάζει με τα προγράμματα υπολογιστών (πολλές μεταβλητές και πολλούς βρόχους)

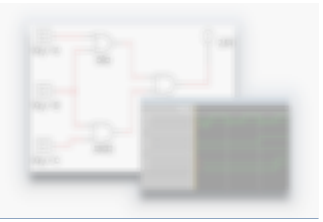
- Για αυτό:
  - Μη γράφετε κώδικα HDL όπως θα γράφατε ένα λογισμικό πρόγραμμα
  - Θυμηθείτε τις δυνατότητες που σας δίνει η HDL (π.χ. **παράλληλη εκτέλεση**, **περιγραφή χρονισμών**, **περιγραφή ακολουθίας γεγονότων**)
  - Να έχετε πάντα στο μυαλό σας τι κύκλωμα αντιστοιχεί στον κώδικα HDL που γράφετε

Η Γλώσσα Verilog



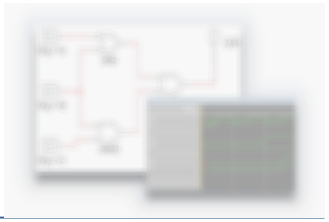
- **Γλώσσα Περιγραφής Υλικού (HDL)**
  - Γλώσσα προγραμματισμού με υποδομές για υλοποίηση υλικού
    - Έννοια του χρόνου, έννοια του σήματος
- **Δυνατότητες**
  - να αναπαριστά (σε διάφορα επίπεδα) και
  - να προσομοιώνει ψηφιακά κυκλώματα.
  - ένα υποσύνολο της είναι συνθέσιμο (HDL → κυκλωματική δομή)
- **Υποστηρίζει**
  - Παράλληλη εκτέλεση τμημάτων υλικού και παράλληλες διαδικασίες
  - Σημασιολογία (semantics) για χρόνο και τιμές σημάτων
- **Παραδείγματα σχεδίασης με Verilog HDL**
  - Intel Pentium, AMD K5, K6, Athlon, ARM7, etc
  - Thousands of ASIC designs using Verilog HDL
- **Άλλες HDL** : VHDL, SystemC, SystemVerilog

Αναπαράσταση και Υλοποίηση σε Verilog



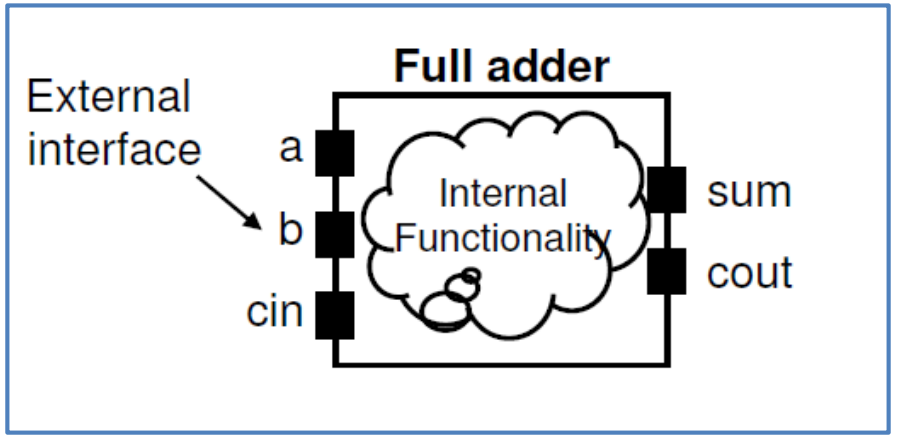
- Η Verilog μπορεί να χρησιμοποιηθεί σε διάφορα στάδια για την υλοποίηση ενός συστήματος από ιδέα σε κύκλωμα
- **Δυνατότητες**:
  - Ορισμός Απαιτήσεων (Requirements Specification)
  - Έγγραφη Τεκμηρίωση (Documentation)
  - Έλεγχος μέσω Προσομοίωσης (Simulation)
  - Λειτουργικός Έλεγχος και Επαλήθευση (Functional Test - Formal Verification)
  - Συνθεσιμότητα σε Σχηματικό, δηλ. σύνολο από πύλες
- **Στόχοι**
  - Αξιόπιστη διεργασία σχεδίασης με χαμηλές απαιτήσεις κόστους και χρόνου
  - Αποφυγή και πρόληψη λαθών σχεδίασης

Έννοιες μοντελοποίησης της HDL



- **Διασύνδεση** (interface)
- **Συμπεριφορά** (behavior)
- **Δομή** (structure)
- **Μοντέλα δοκιμής** (test benches)

HDL: μοντελοποίηση & προσομοίωση



Module

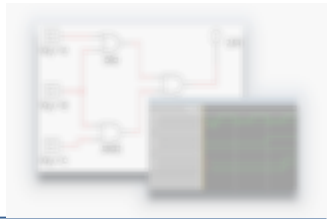
Η λογική που περιγράφεται σε ένα module μπορεί να περιγραφεί με διάφορα στυλ

Μοντελοποίηση σε επίπεδο πυλών
Structural model

Μοντελοποίηση **ροής δεδομένων** με την **assign**

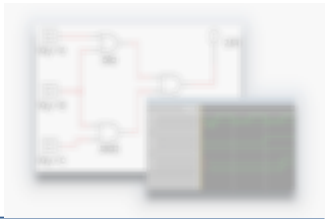
Μοντελοποίηση συμπεριφοράς με **always**
Behavioral model

Συμβάσεις στην γλώσσα Verilog



- **Η Verilog είναι case sensitive**.
  - Λέξεις κλειδιά είναι σε μικρά.
- **Σχόλια**
  - Για μία γραμμή είναι //
  - Για πολλές /* */
- **Βασικές τιμές 1-bit σημάτων**
  - 0: λογική τιμή 0.
  - 1: λογική τιμή 1
  - x: άγνωστη τιμή
  - z: ασύνδετο σήμα. High impedance

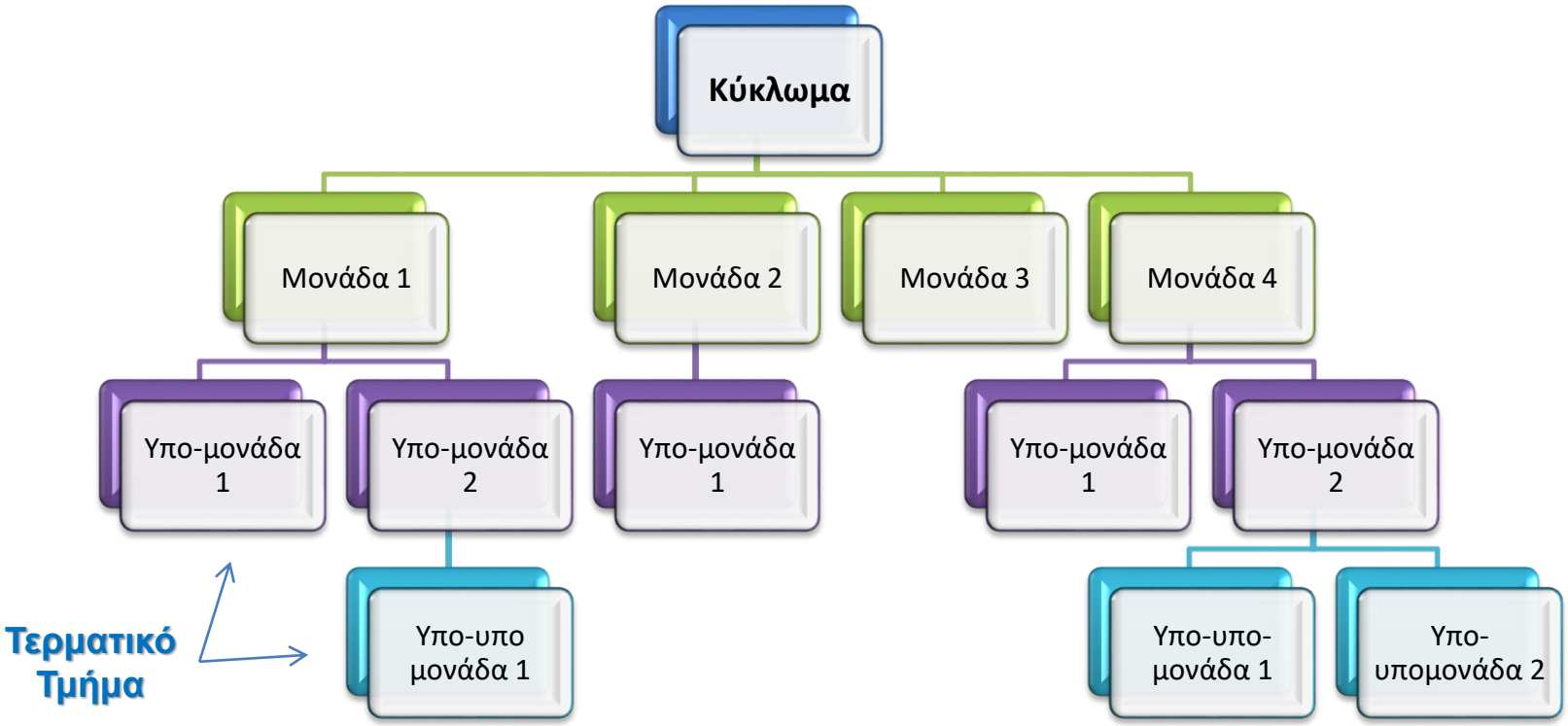
Verilog: Αριθμοί



Αναπαράσταση αριθμών

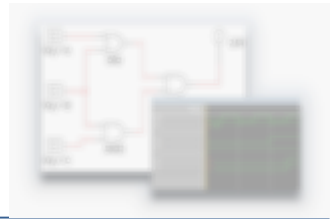
- **<size>' <base_format> <number>**
  - < size > δείχνει τον αριθμό από bits
  - < base_format > μπορεί να είναι : d, h, b, o
- Όταν το <size> λείπει το μέγεθος καθορίζεται από τον compiler
- Παραδείγματα:
  - `4'd3` // 3, 4 bits (0011)
  - `4'b1111` // 15, 4bits
  - `6'h3a` // 58, 6bits
  - `6'b111010` // 58, 6bits

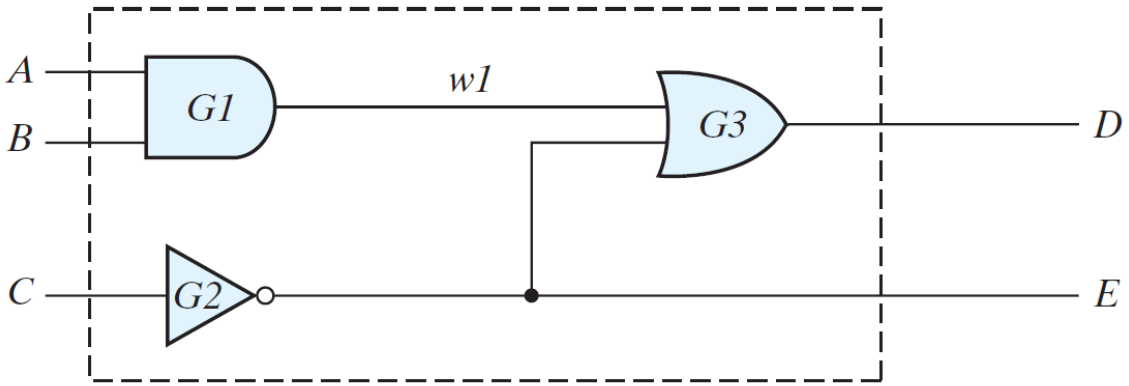
Μεθοδολογία Σχεδίασης (ιεραρχία)



Όταν έχουμε μεγάλο design: Το τελικό σύστημα αποτελείται από τα Τερματικά Τμήματα ή φύλλα (Leaf blocks) που τρέχουν όλα παράλληλα.

Verilog: Βασικό Block: module (1/7)



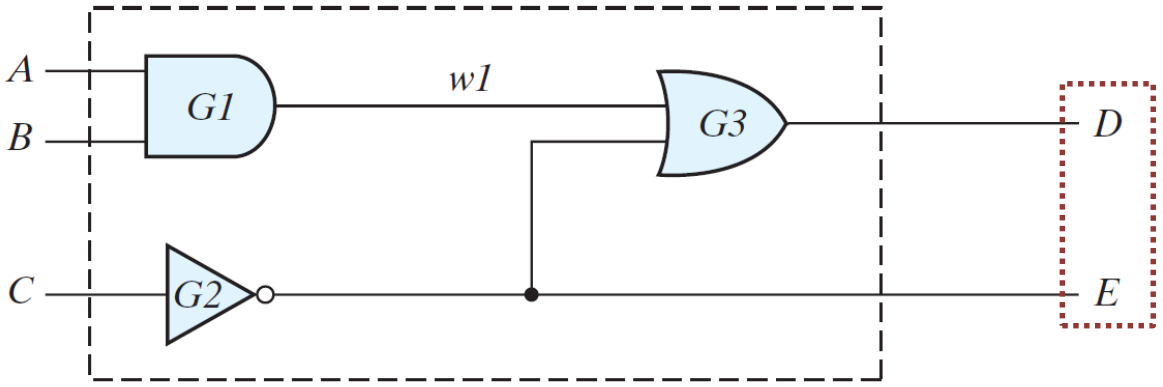


Η υπομονάδα **module** είναι η θεμελιώδης περιγραφική μονάδα

```
module  Simple_Circuit (A, B, C, D, E);

endmodule
```

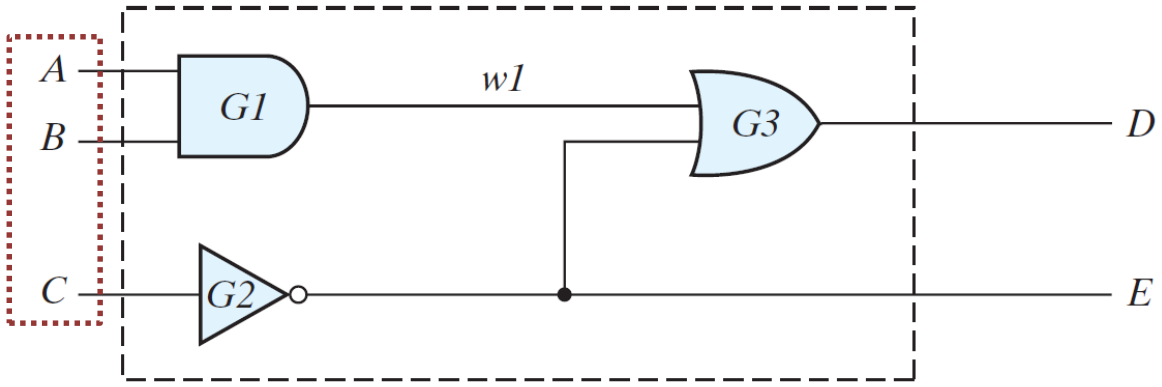
Verilog: Βασικό Block: module (2/7)




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

```
endmodule
```

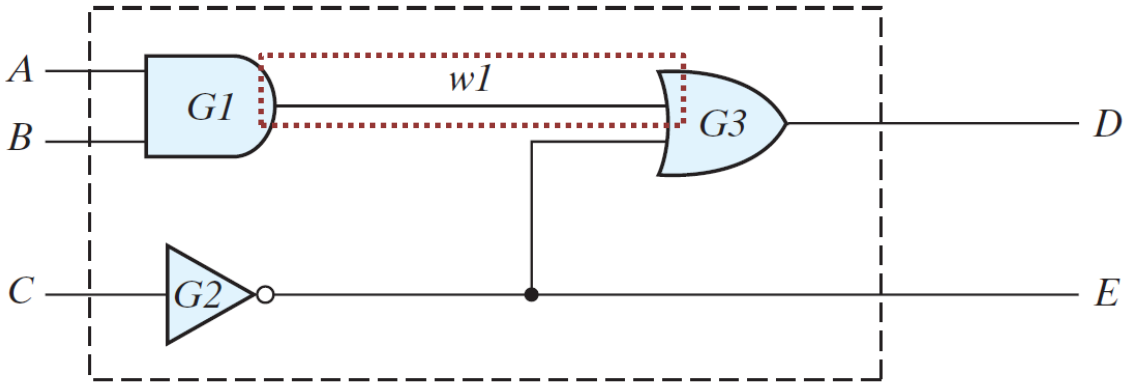
Verilog: Βασικό Block: module (3/7)




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

endmodule
```

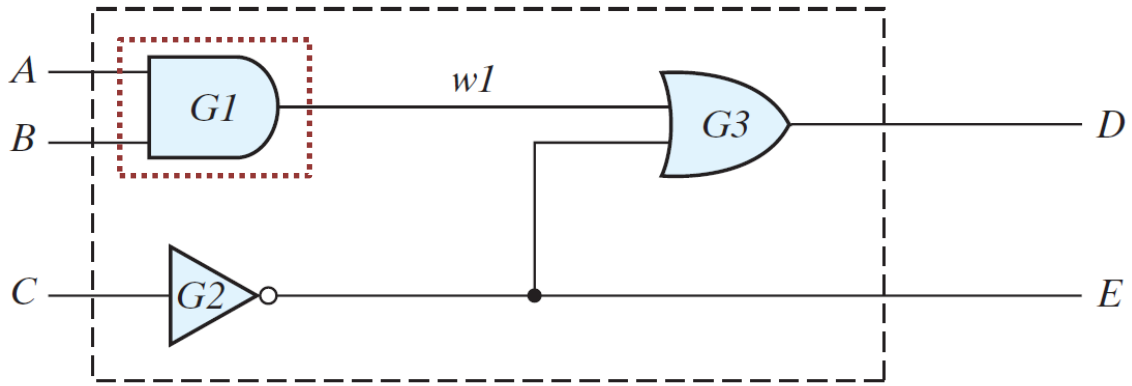
Verilog: Βασικό Block: module (4/7)




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


endmodule
```

Verilog: Βασικό Block: module (5/7)

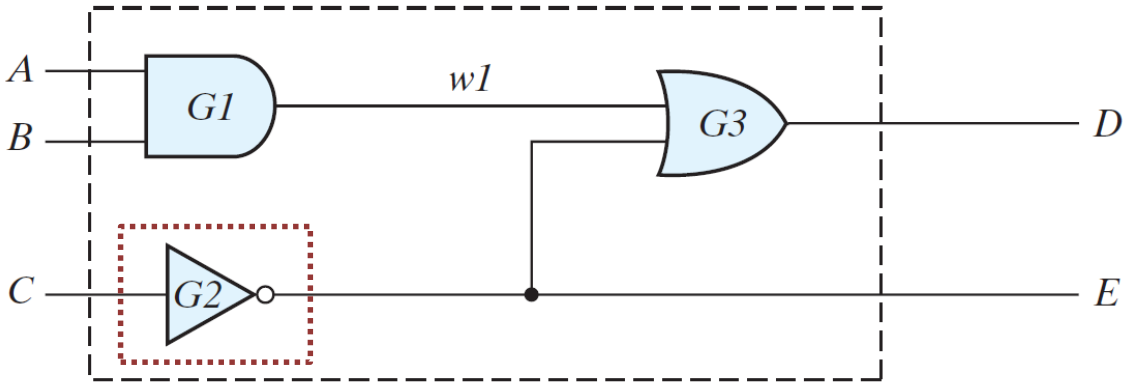



```
module  Simple_Circuit (A, B, C, D, E);
 output          D, E;
 input           A, B, C;
 wire            w1;

 and             G1 (w1, A, B); // Optional gate instance name


endmodule
```

Verilog: Βασικό Block: module (6/7)

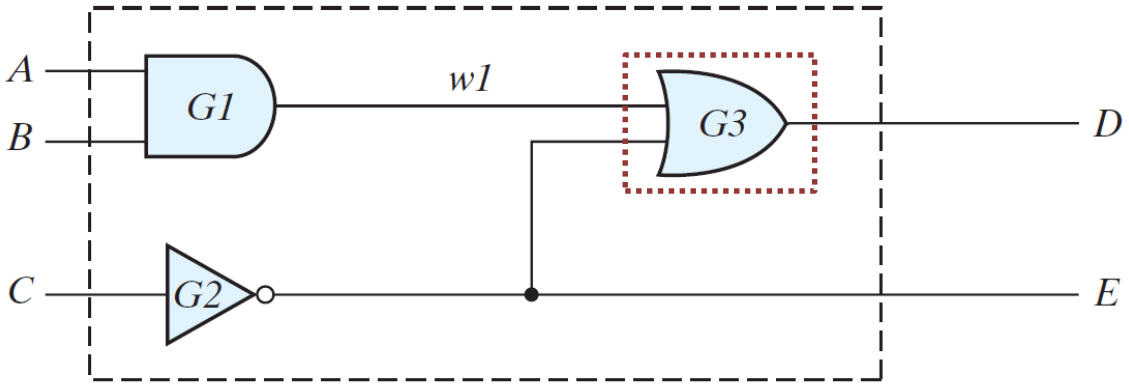



```
module  Simple_Circuit (A, B, C, D, E);
 output          D, E;
 input           A, B, C;
 wire            w1;

 and             G1 (w1, A, B); // Optional gate instance name
 not             G2 (E, C);

endmodule
```

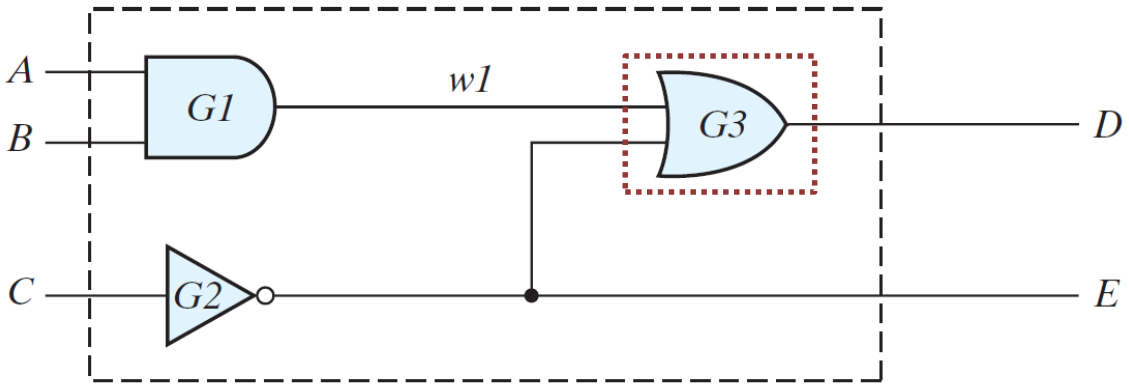
Verilog: Βασικό Block: module (7/7)




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

 and             G1 (w1, A, B); // Optional gate instance name
 not             G2 (E, C);
 or              G3 (D, w1, E);
endmodule
```

Verilog: Βασικό Block: module



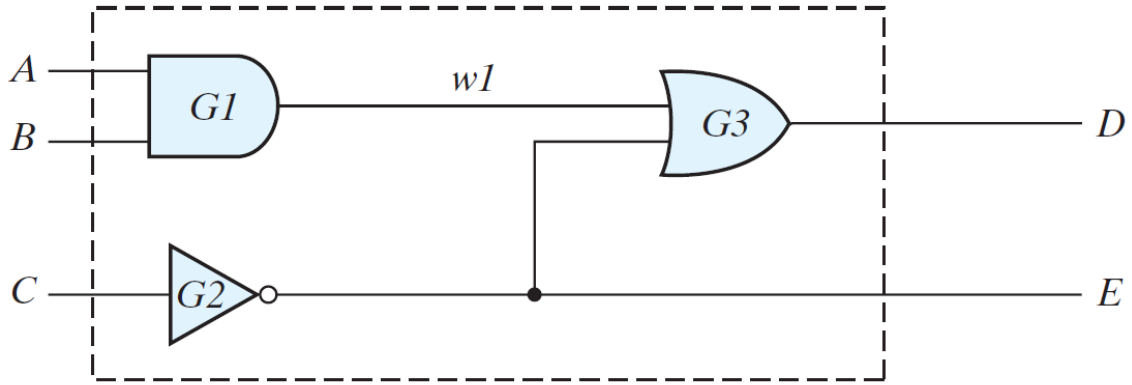

```
module  Simple_Circuit (A, B, C, D, E);
  output        D, E;
  input         A, B, C;
  wire          w1;

  and           G1 (w1, A, B); // Optional gate instance name
  not           G2 (E, C);
  or            G3 (D, w1, E);
endmodule
```

Διαφορά Δήλωσης και Στιγμιότυπου.

- Η **υπομονάδα δηλώνεται**. Η δήλωση καθορίζει την συμπεριφορά των εξόδων σε σχέση με τις τιμές που δίνονται στην είσοδο.
- Οι **προκαθορισμένες βασικές πύλες δεν δηλώνονται** γιατί ο ορισμός τους είναι ενσωματωμένος στη γλώσσα και δεν μπορεί να αλλάξει από τον χρήστη
- Χρησιμοποιούνται ως **στιγμιότυπα** ήδη προ-ορισμένων πυλών,
- Μπορούμε να δημιουργήσουμε δικές μας πύλες.
- Όταν δηλωθεί ένα module μπορούμε να χρησιμοποιήσουμε στιγμιότυπά του.
- Το παραπάνω κύκλωμα δεν είναι ένα υπολογιστικό μοντέλο (π.χ. C)
- Είναι ένα περιγραφικό μοντέλο (π.χ. εδώ δεν παίζει ρόλο η σειρά των στιγμιοτύπων πυλών)

Verilog: Βασικό Block: module

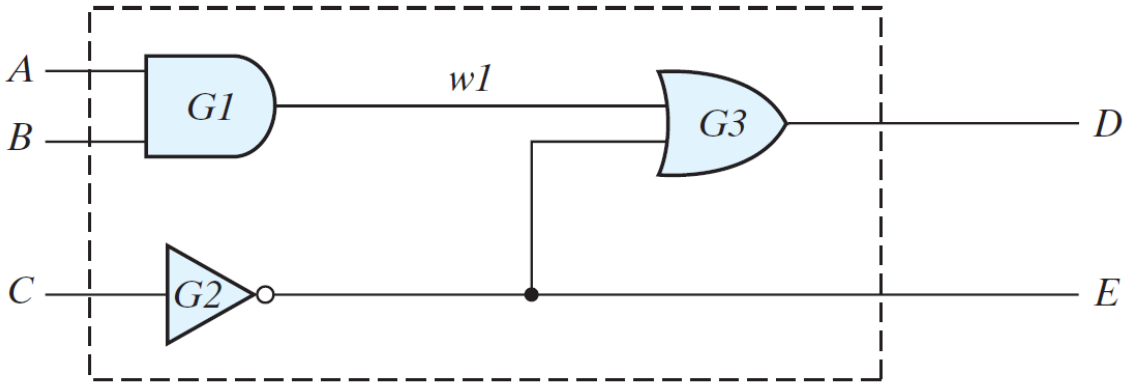


Μπορούν να γραφούν με οποιαδήποτε σειρά

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

and         G1 (w1, A, B); // Optional gate instance name
not         G2 (E, C);
or          G3 (D, w1, E);
endmodule
```

Verilog: Βασικό Block: module

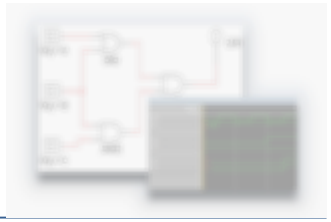


Μπορούν να γραφούν με οποιαδήποτε σειρά

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

  and           G1 (w1, A, B); // Optional gate instance name
  not           G2 (E, C);
  or            G3 (D, w1, E);
endmodule
```

Verilog: Modules, Files



- Η **υπομονάδα (module)** είναι η βασική δομή για τη μοντελοποίηση κυκλωμάτων με χρήση της Verilog
  - Ένα module μπορεί να περιγράφει ένα ολόκληρο κύκλωμα (Σπάνια, για πολύ μικρά κυκλώματα!)
  - Ένα module μπορεί να περιγράφει ένα μέρος του κυκλώματος (υπο-κύκλωμα), οπότε πολλά modules μπορούν να συνδυαστούν για να περιγράψουν συνολικά ένα κύκλωμα
  - Το κάθε module έχει ένα μοναδικό όνομα
- Το **αρχείο (file)** είναι ο χώρος που συντάσσονται και αποθηκεύονται τα modules
  - Σε ένα αρχείο (file) μπορούν να συμπεριληφθούν ένα ή περισσότερα modules
  - Ένα κύκλωμα μπορεί να περιγράφεται από πολλά modules, τα οποία βρίσκονται σε διαφορετικά αρχεία
  - Το αρχείο ουσιαστικά είναι ένα απλό έγγραφο κειμένου (text document)
  - Για να μπορέσουν τα διάφορα εργαλεία - προγράμματα να αναγνωρίσουν ότι το αρχείο αυτό περιέχει modules της Verilog, δίνουμε στο αρχείο την κατάληξη “**.v** ”
- Σημείωση:

  Όταν δημιουργούμε ένα νέο text document στα windows τότε το νέο αυτό αρχείο έχει κατάληξη “.txt” την οποία πρέπει να τροποποιήσουμε σε “.v”

Exercise: Γλώσσα Περιγραφής Υλικού HDL - Verilog

Γράψτε την περιγραφή HDL σε επίπεδο πυλών του κυκλώματος “Figure 2.8”.

Ονομάστε το αρχείο “Figure2_8.v” και τη μονάδα: “Figure2_8”

(βλέπε Παράδειγμα HDL 3.1 – Morris Mano)

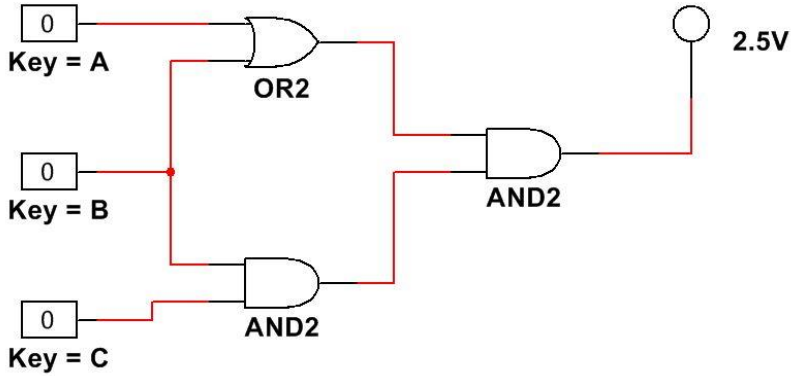
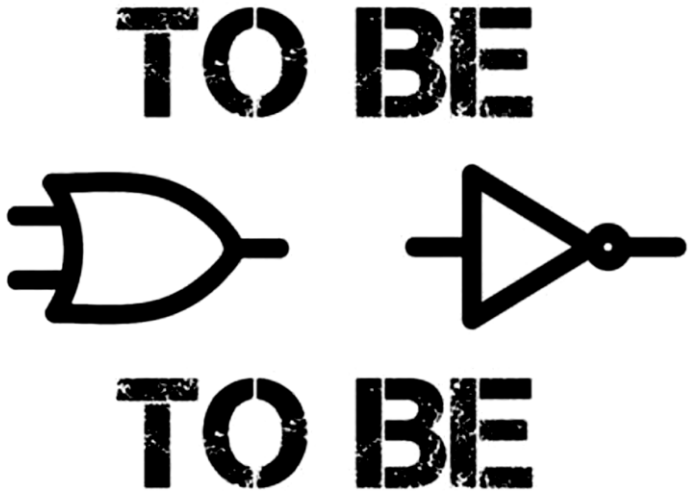


Figure 2-8 Circuit with AND OR Logic gates

Ευχαριστώ για την προσοχή σας!

- **Ερωτήσεις / Απορίες ;**



Επικοινωνία: ece119.uth@gmail.com