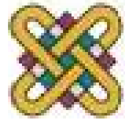


Πανεπιστήμιο Δυτικής Μακεδονίας
Τμήμα Μηχανικών Πληροφορικής & Τηλεπικοινωνιών

Ενσωματωμένα Συστήματα

Ενότητα 3: Η γλώσσα περιγραφής υλικού VHDL

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

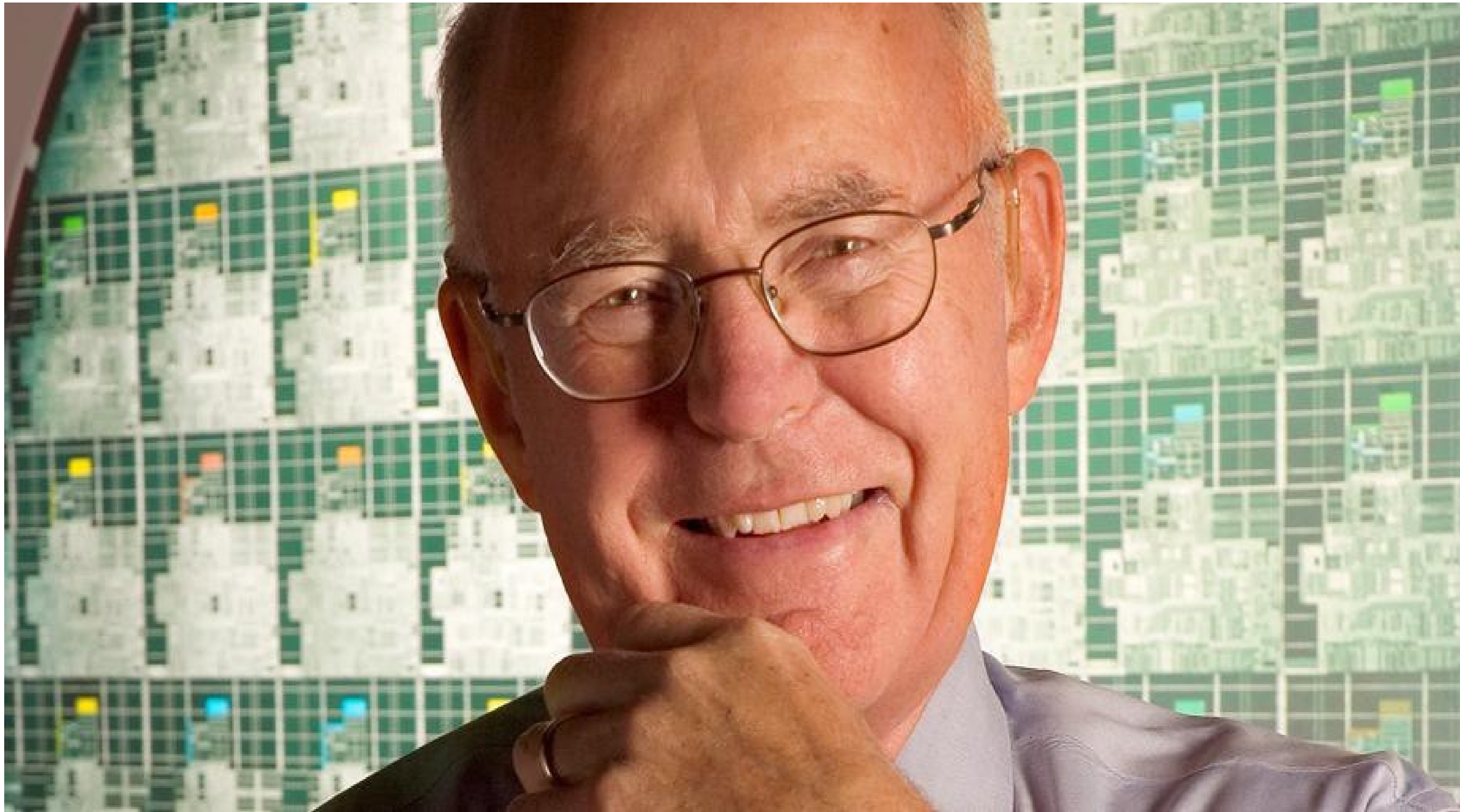
Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός ενότητας

- Η κατανόηση της αύξησης της σχεδιαστικής πολυπλοκότητας, και η ανάγκη για τη χρήση μιας γλώσσας HDL.
- Η εκμάθηση της VHDL γλώσσας περιγραφής, για την αύξηση της σχεδιαστικής παραγωγικότητας.

Ο νόμος του Moore (1/2)



Ο νόμος του Moore (2/2)

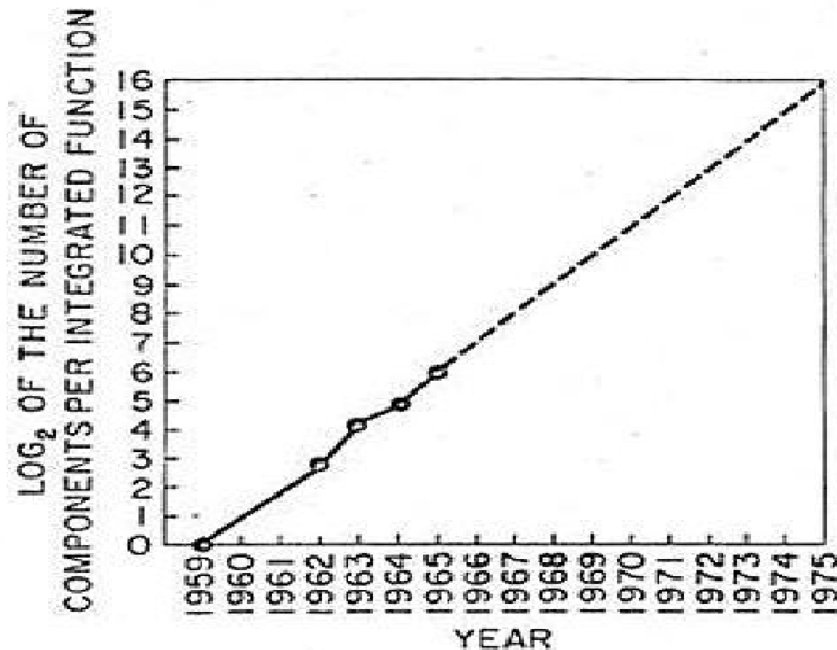
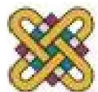


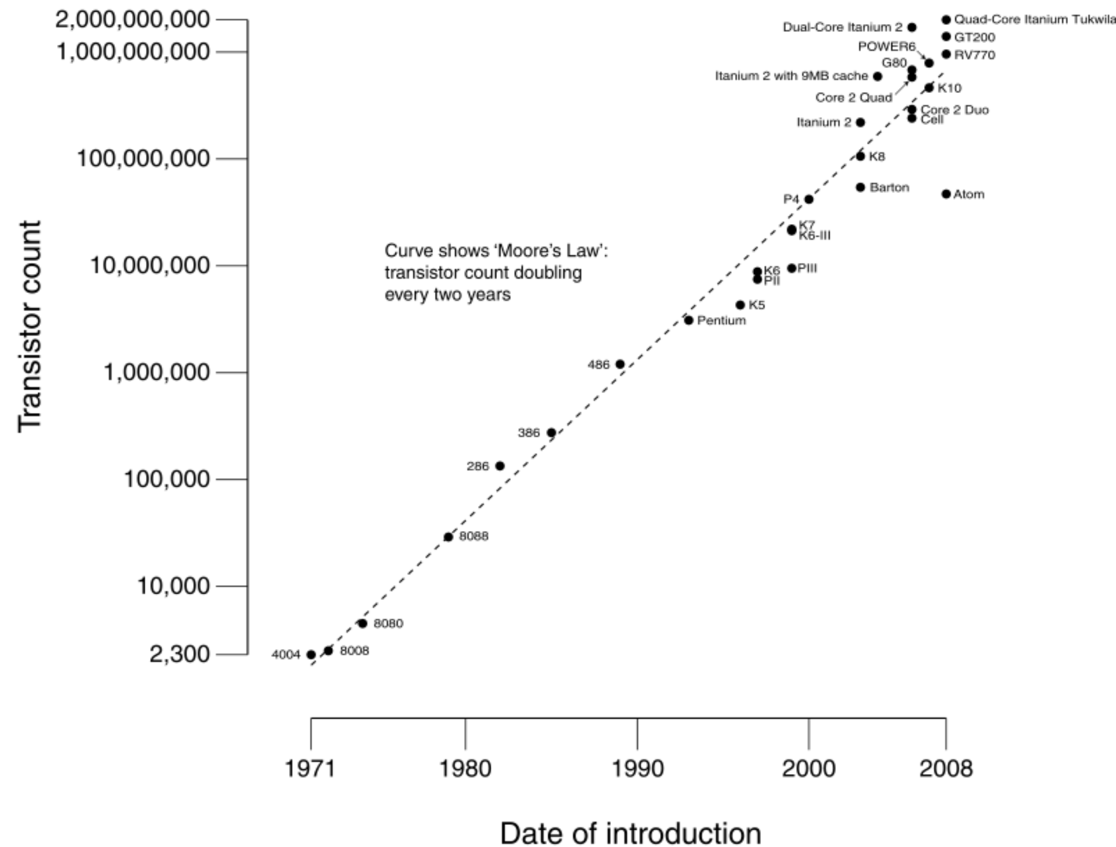
Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

«ο αριθμός των τρανζίστορ σε ένα μικροεπεξεργαστή θα διπλασιάζεται κάθε 24 μήνες»

--Gordon Moore, Intel Co-Founder



Ο νόμος του Moore σε μΕ Domain

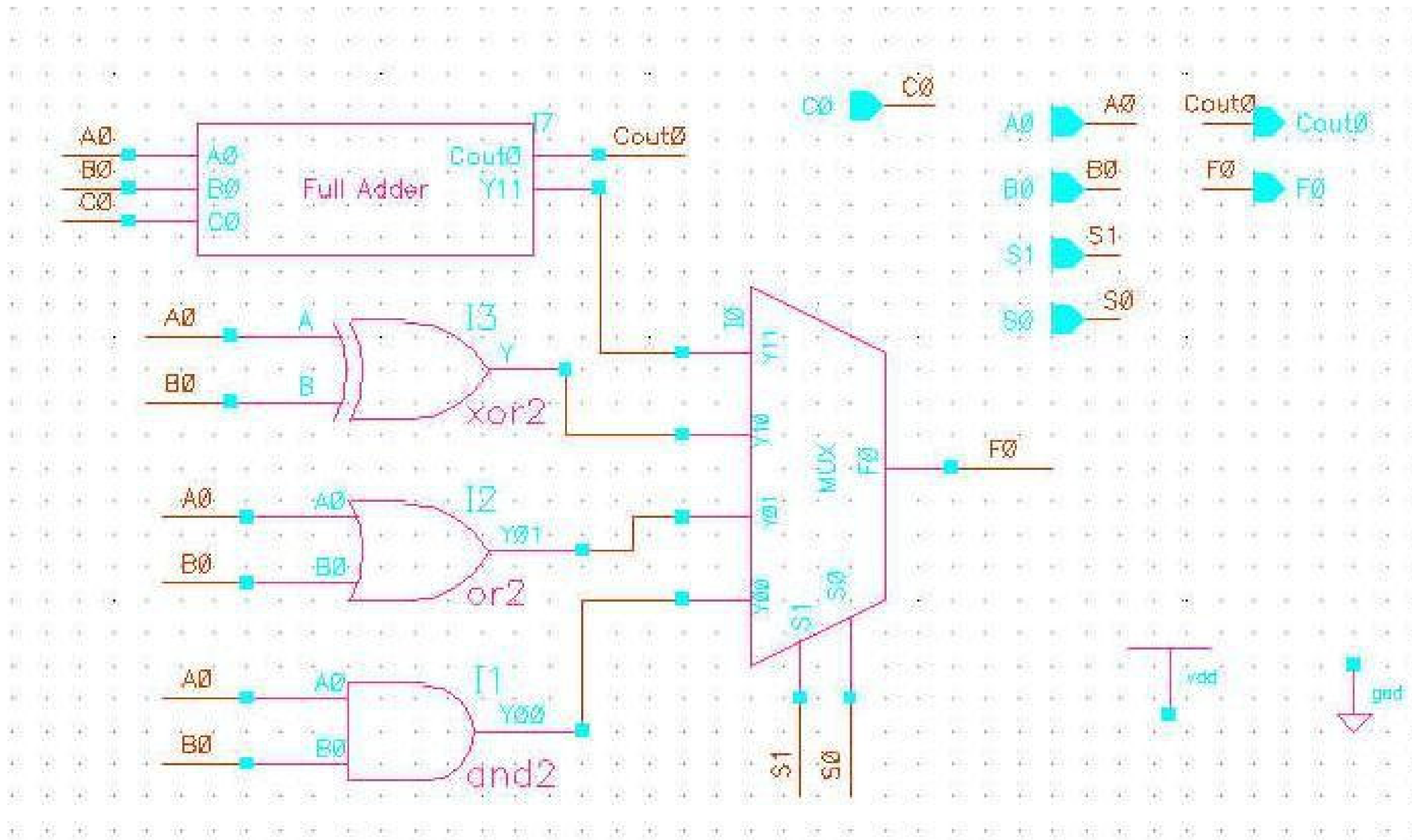


Ο νόμος του Moore: CPUs & FPGAs

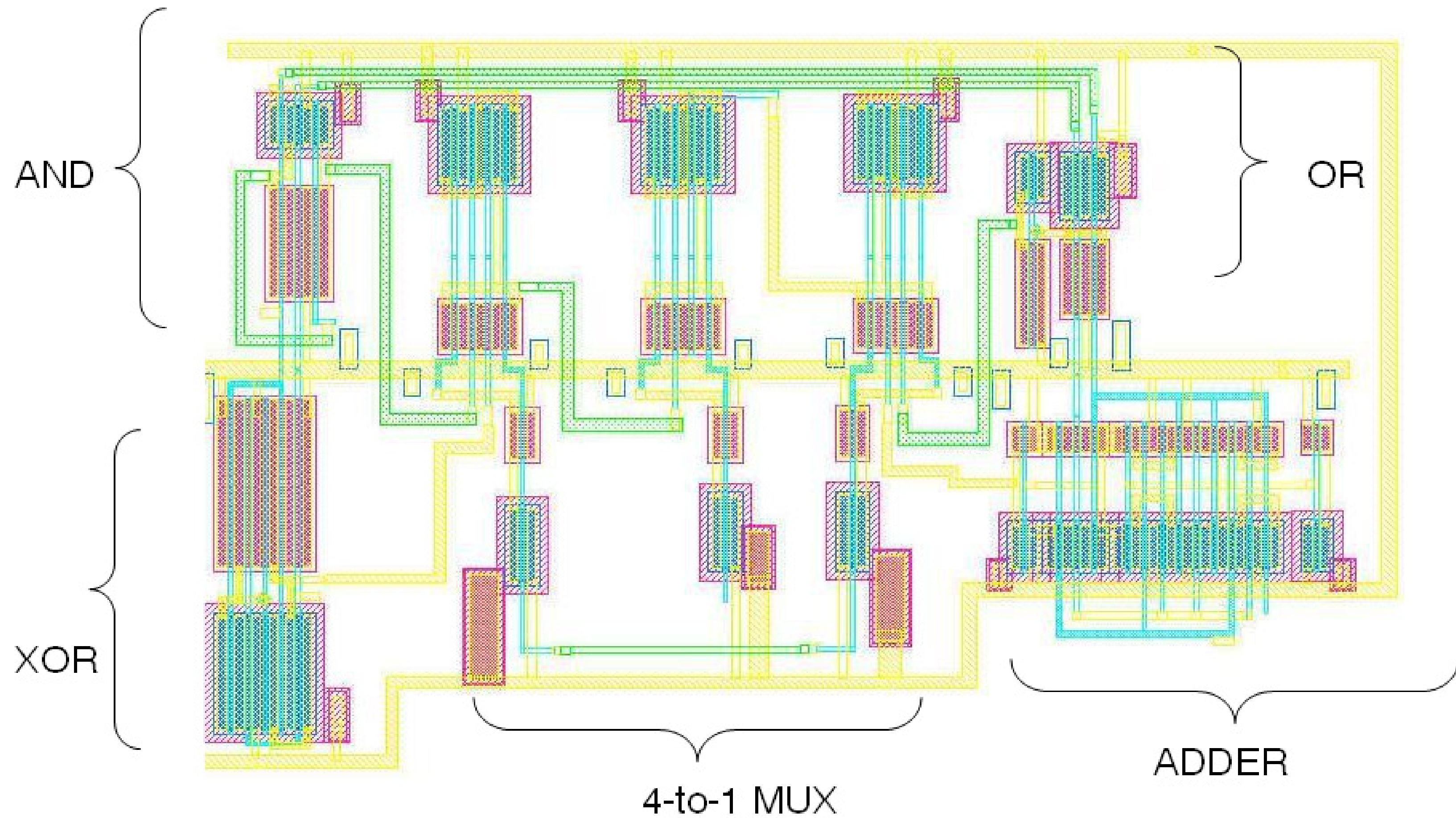
CPU	Transistor count	Process
Pentium 4	42,000,000	180 nm
Atom	47,000,000	45 nm
AMD K8	105,900,000	130 nm
Itanium 2	220,000,000	130 nm
Cell	241,000,000	90 nm
Core 2 Duo	291,000,000	65 nm
Core i7 (Quad)	731,000,000	45 nm
Six-Core Xeon 7400	1,900,000,000	45 nm
POWER6	789,000,000	65 nm
16-Core SPARC T3	1,000,000,000	40 nm
8-core POWER7	1,200,000,000	45 nm
Quad-core z196	1,400,000,000	45 nm
Dual-Core Itanium 2	1,700,000,000	90 nm

FPGA	Transistor count	Process
Virtex	~70,000,000	
Virtex-E	~200,000,000	
Virtex-II	~350,000,000	130 nm
Virtex-II PRO	~430,000,000	
Virtex-4	1,000,000,000	90 nm
Virtex-5	1,100,000,000	65 nm
Stratix IV	2,500,000,000	40 nm
Stratix V	3,800,000,000	28 nm
Virtex-7	6,800,000,000	28 nm

1-bit ALU (schematic-Σχηματικό)

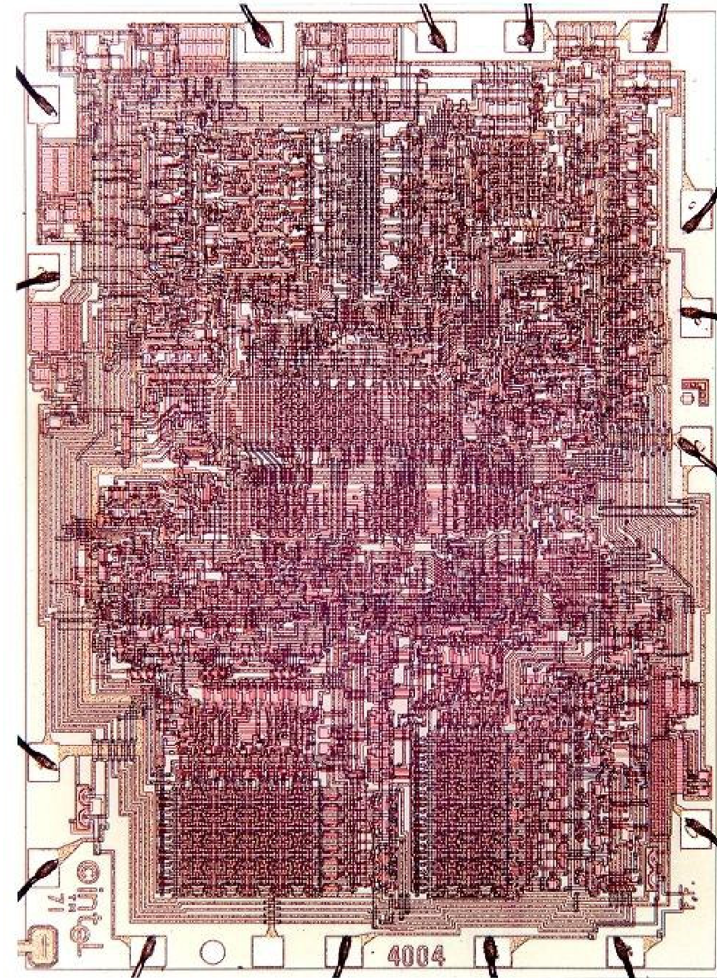


1-bit ALU (layout)

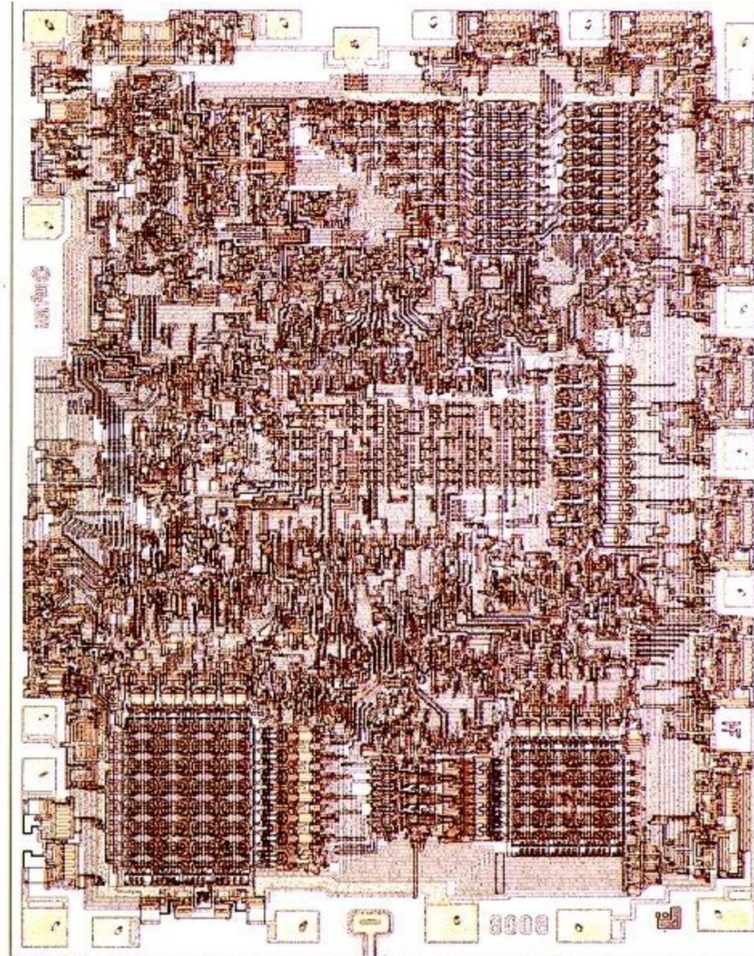


4004 μΕ *(1971)*

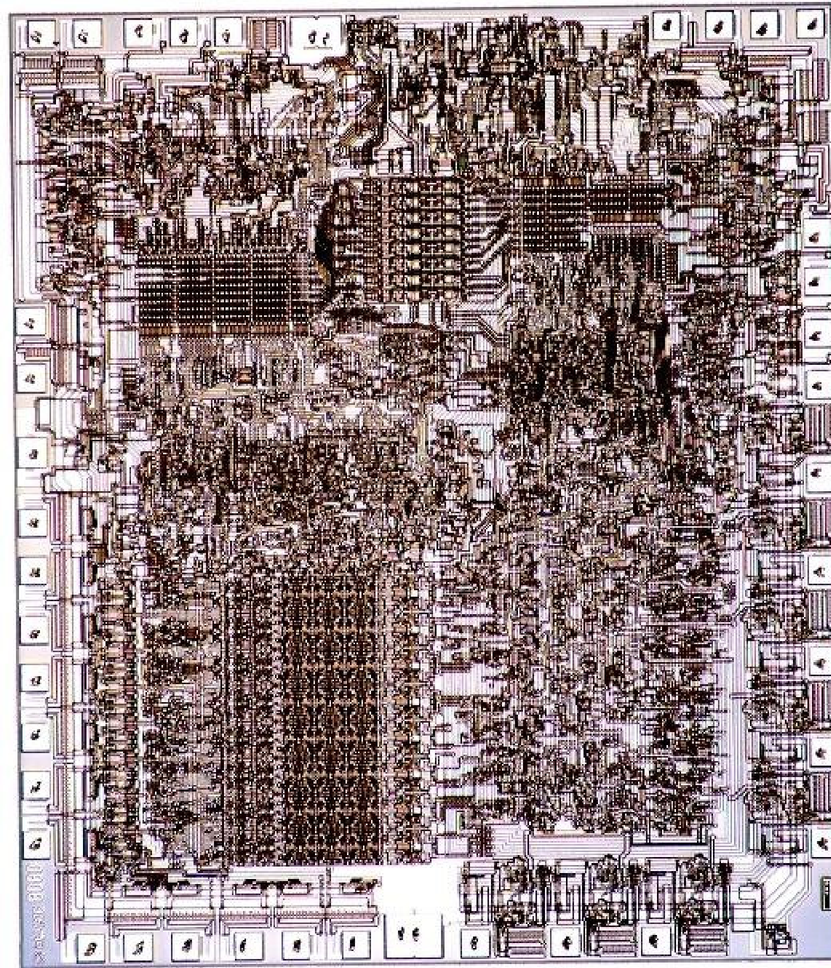
- 4-bit CPU.
- Πρώτη Ολοκληρωμένη CPU σε chip.
- Πρώτη εμπορικά διαθέσιμη CPU.
- 2,300 transistors.
- MIPS: 0.092.
- Ταχύτητα ρολογιού Clock speed: 1MHz.



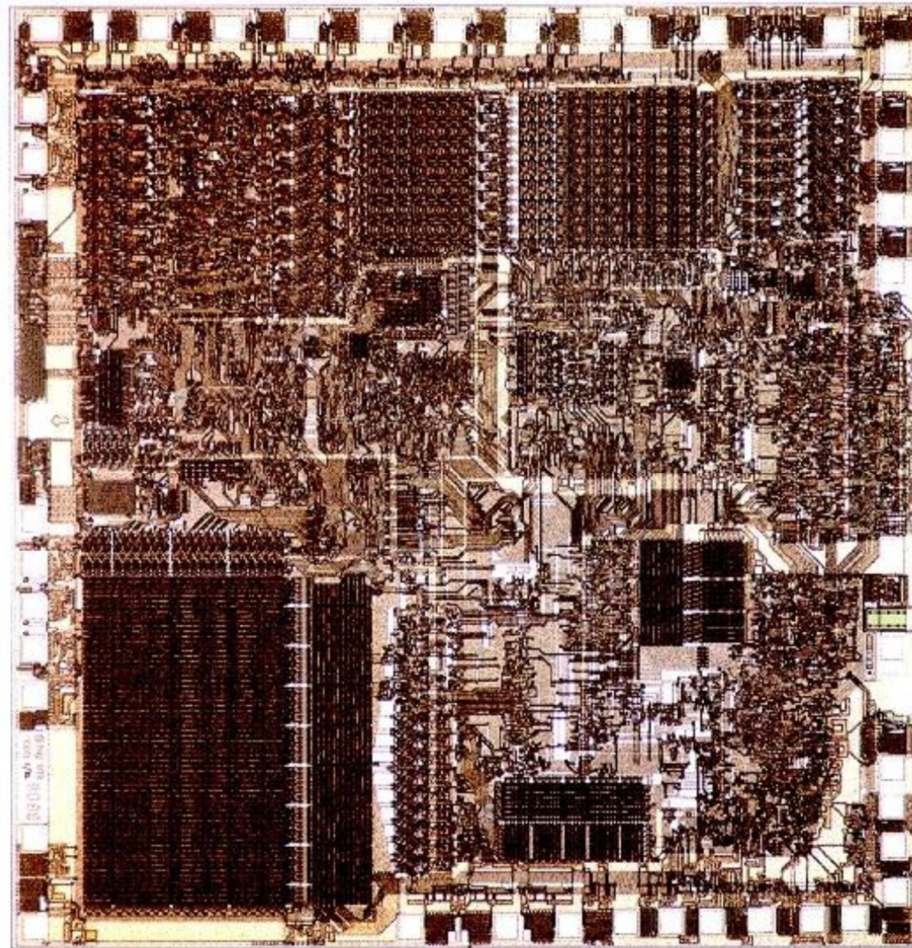
8008 μΕ *(1972)*



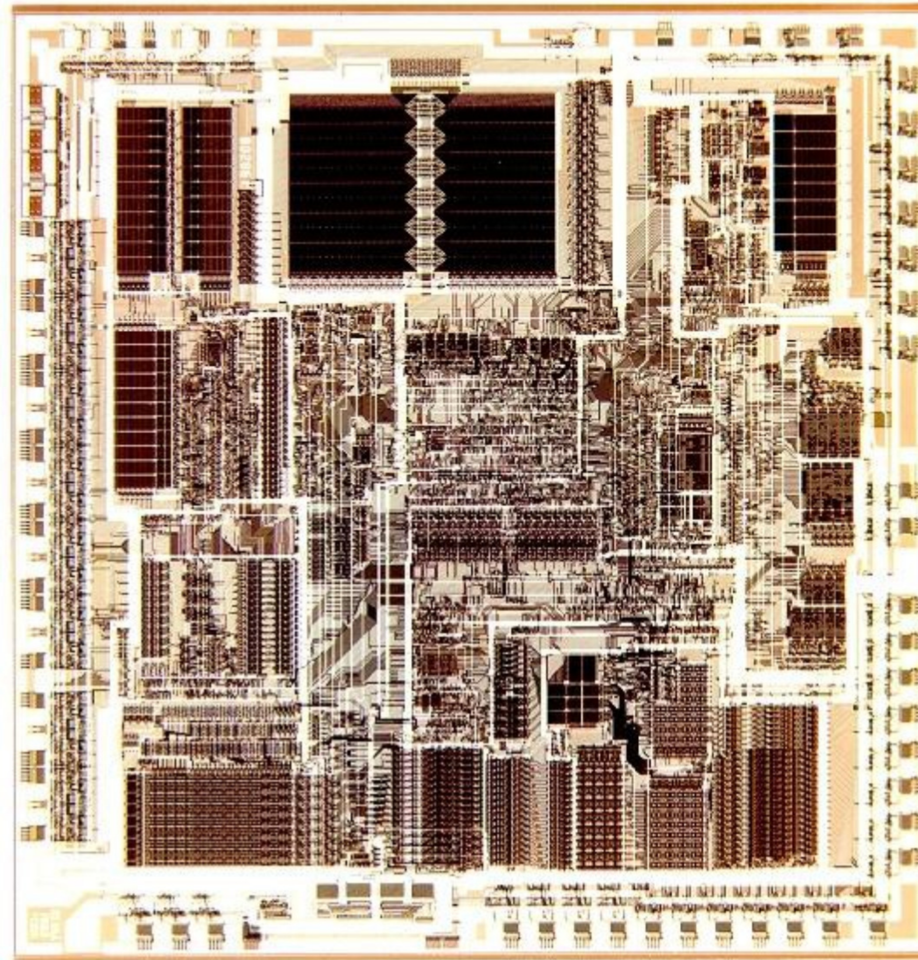
8080 μΕ *(1974)*



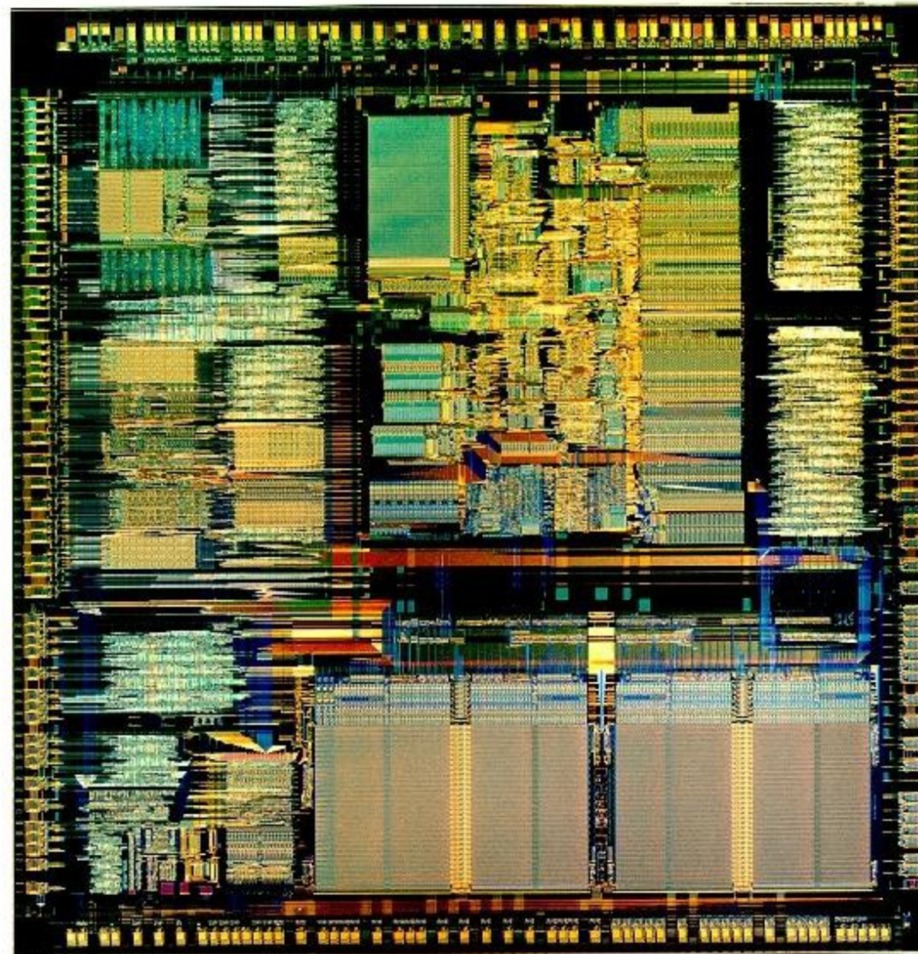
8088 μΕ *(1978)*



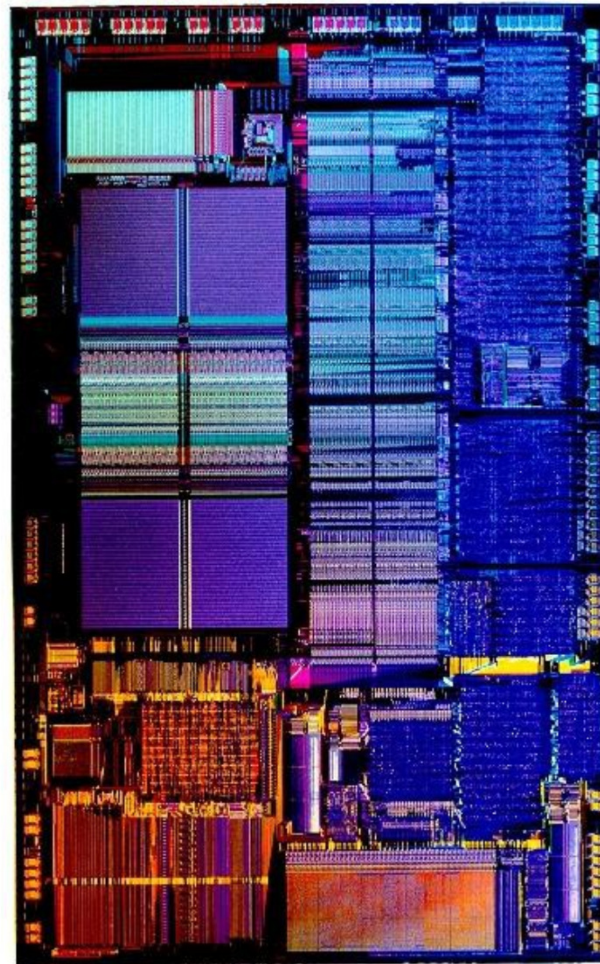
8286 μΕ *(1982)*



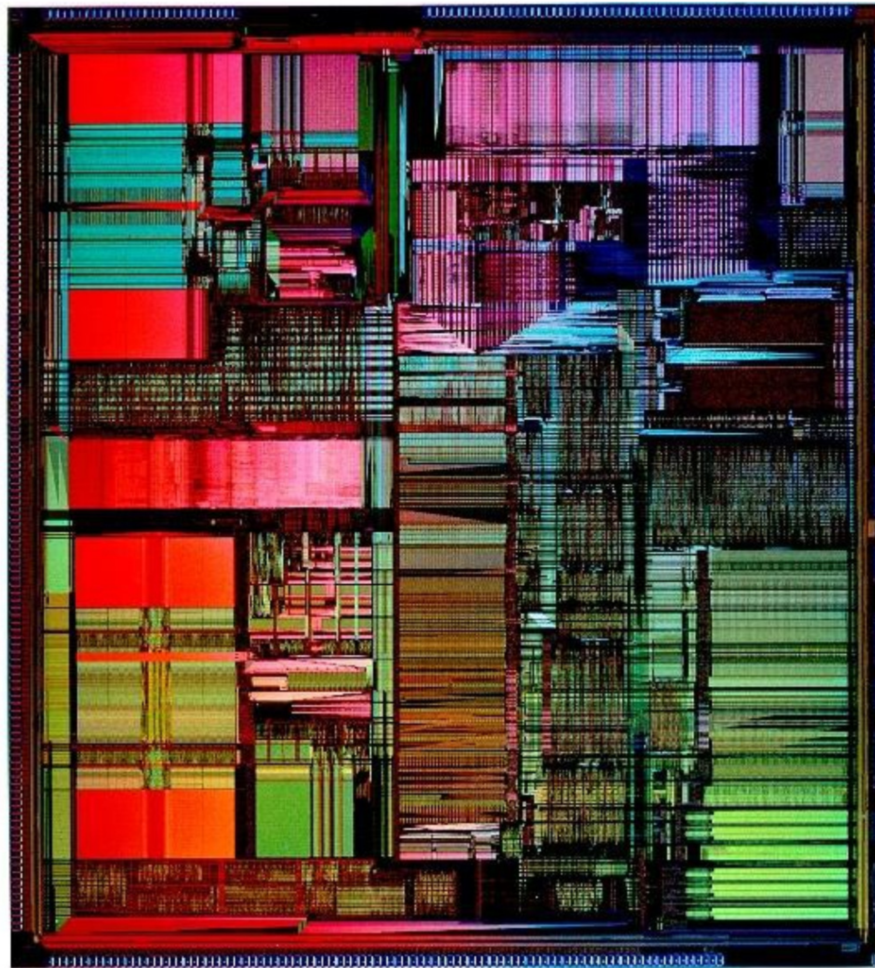
8386 μΕ *(1985)*



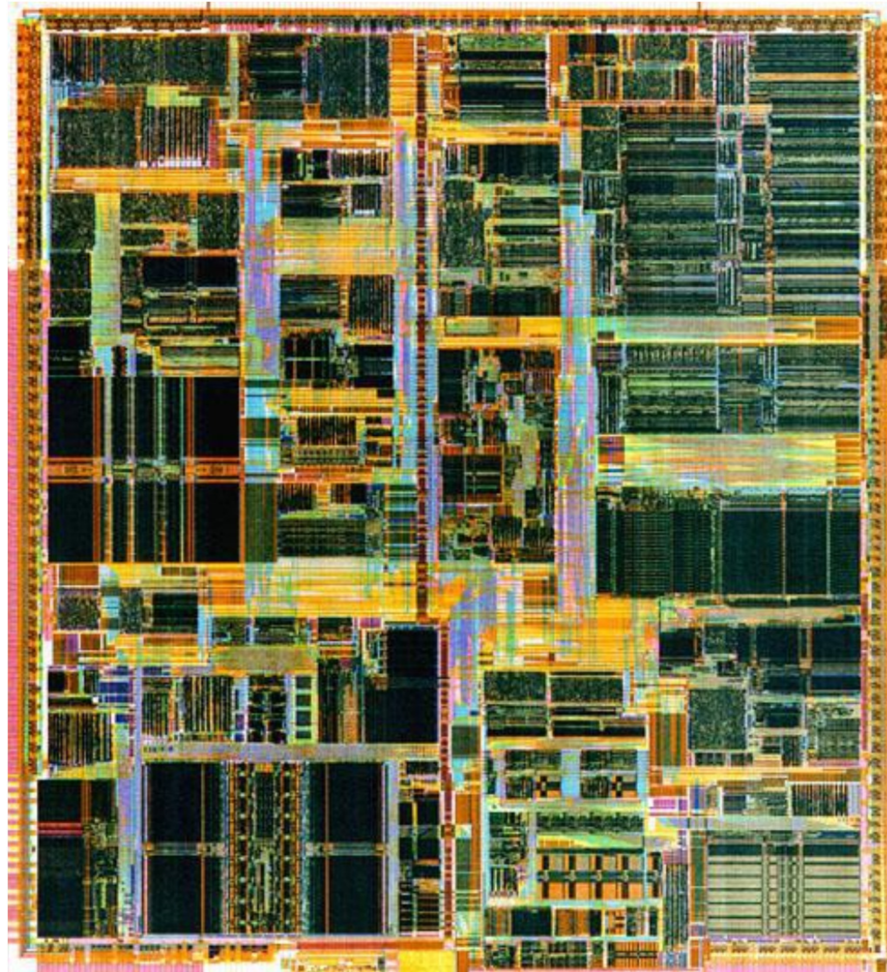
8486 μΕ *(1989)*



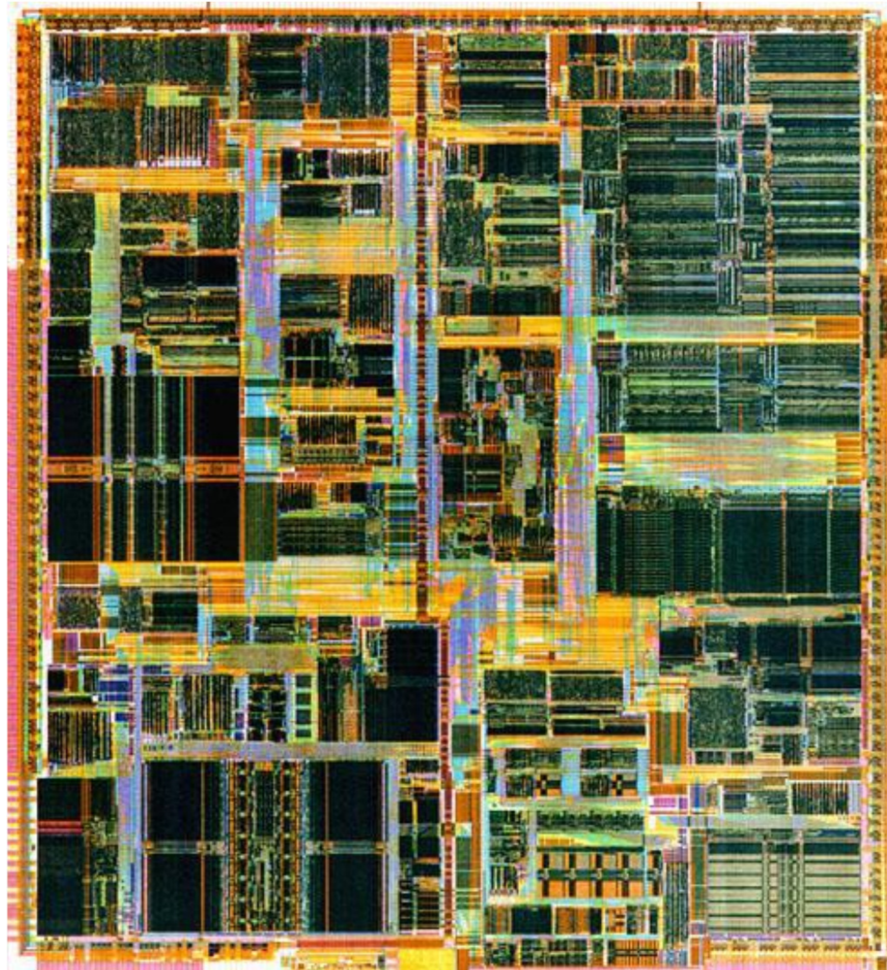
Pentium μE *(1993)*



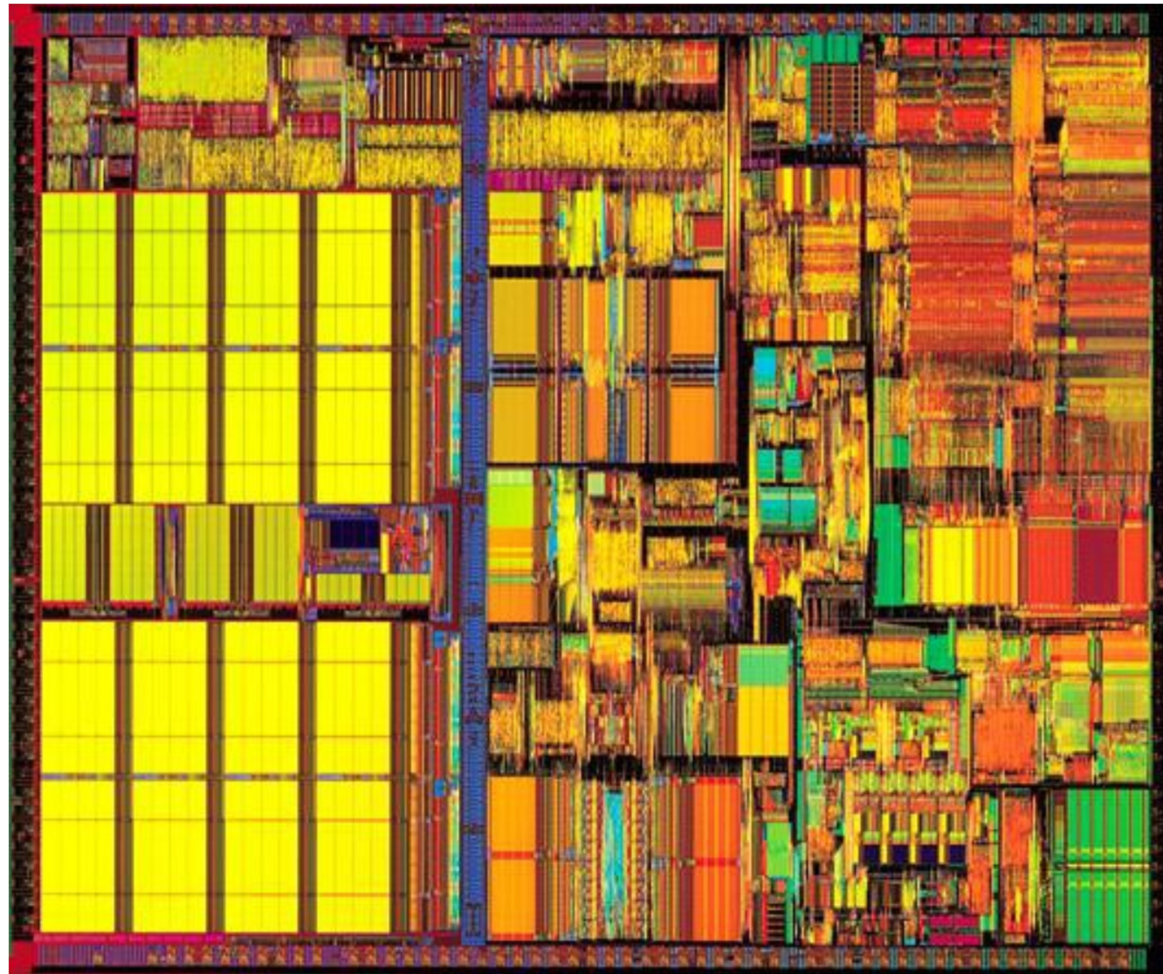
Pentium II μE *(1997)*



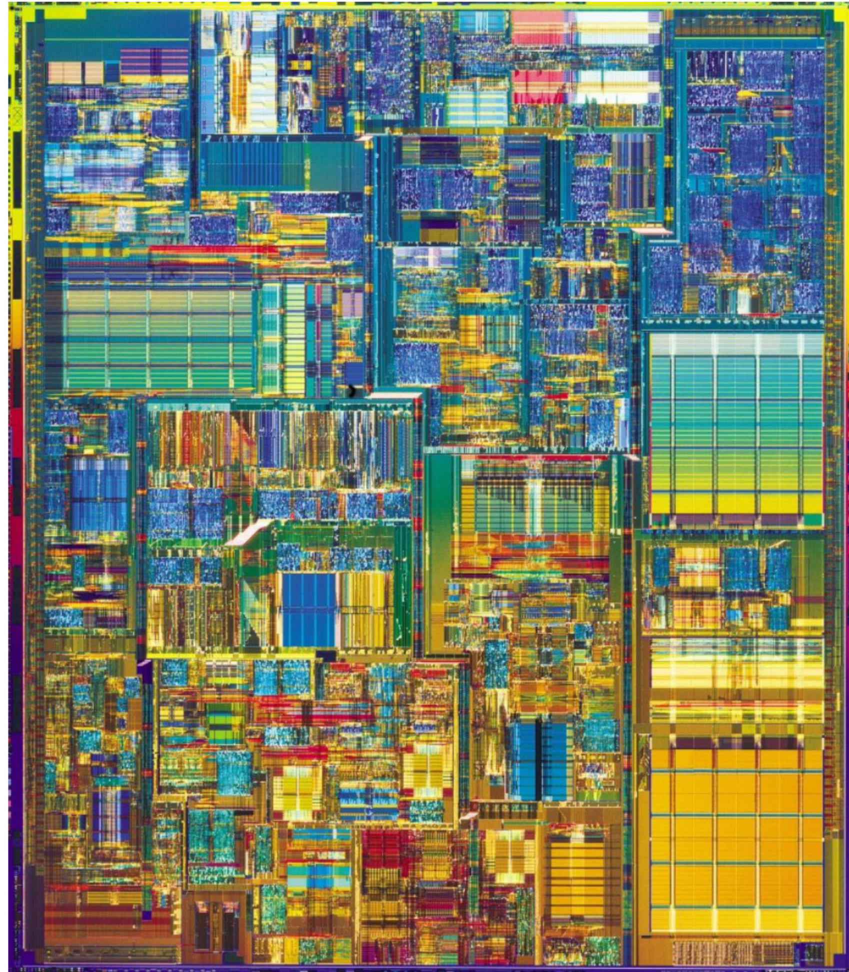
Pentium II XEON μE *(1998)*



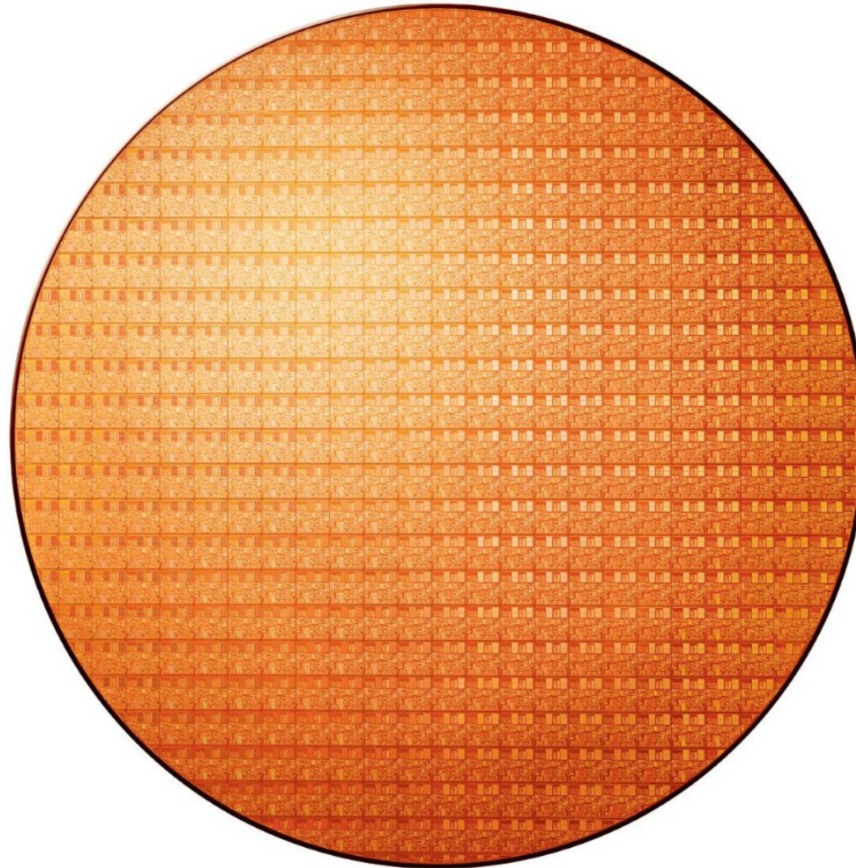
Pentium III μΕ *(1999)*



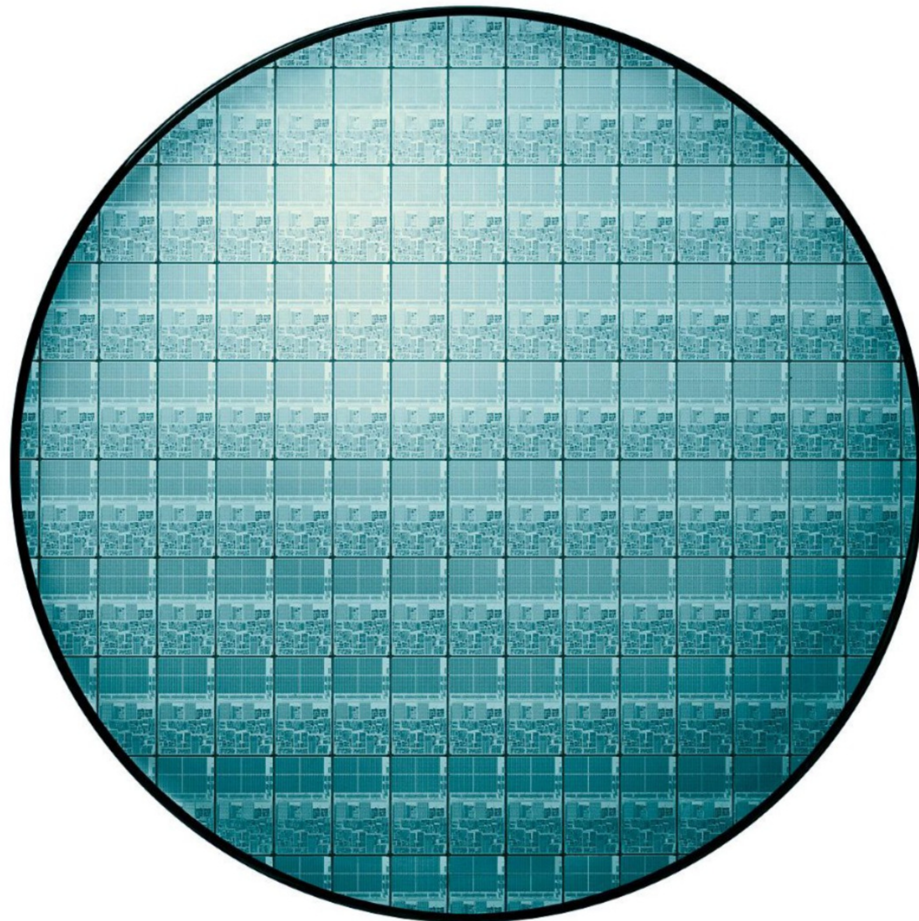
Pentium 4 μE *(2000)*



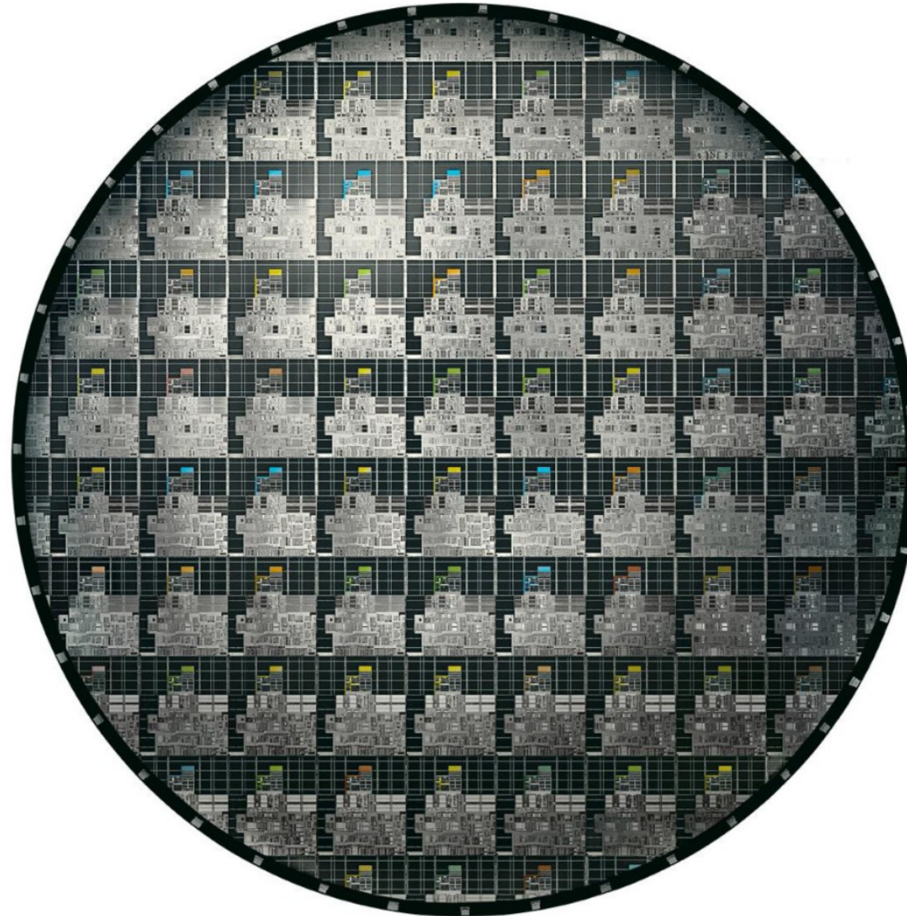
Wafer από Pentium® 4 Επεξεργαστές



Wafer από Intel® Xeon™ Επεξεργαστές



Wafer από Itanium® Επεξεργαστές



VHDL (1/2)

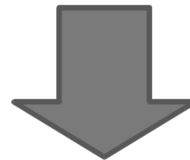
Very Hard Difficult Language

Πολύ δύσκολη γλώσσα προγραμματισμού

VHDL (2/2)

Very Hard Difficult Language

Πολύ δύσκολη γλώσσα προγραμματισμού

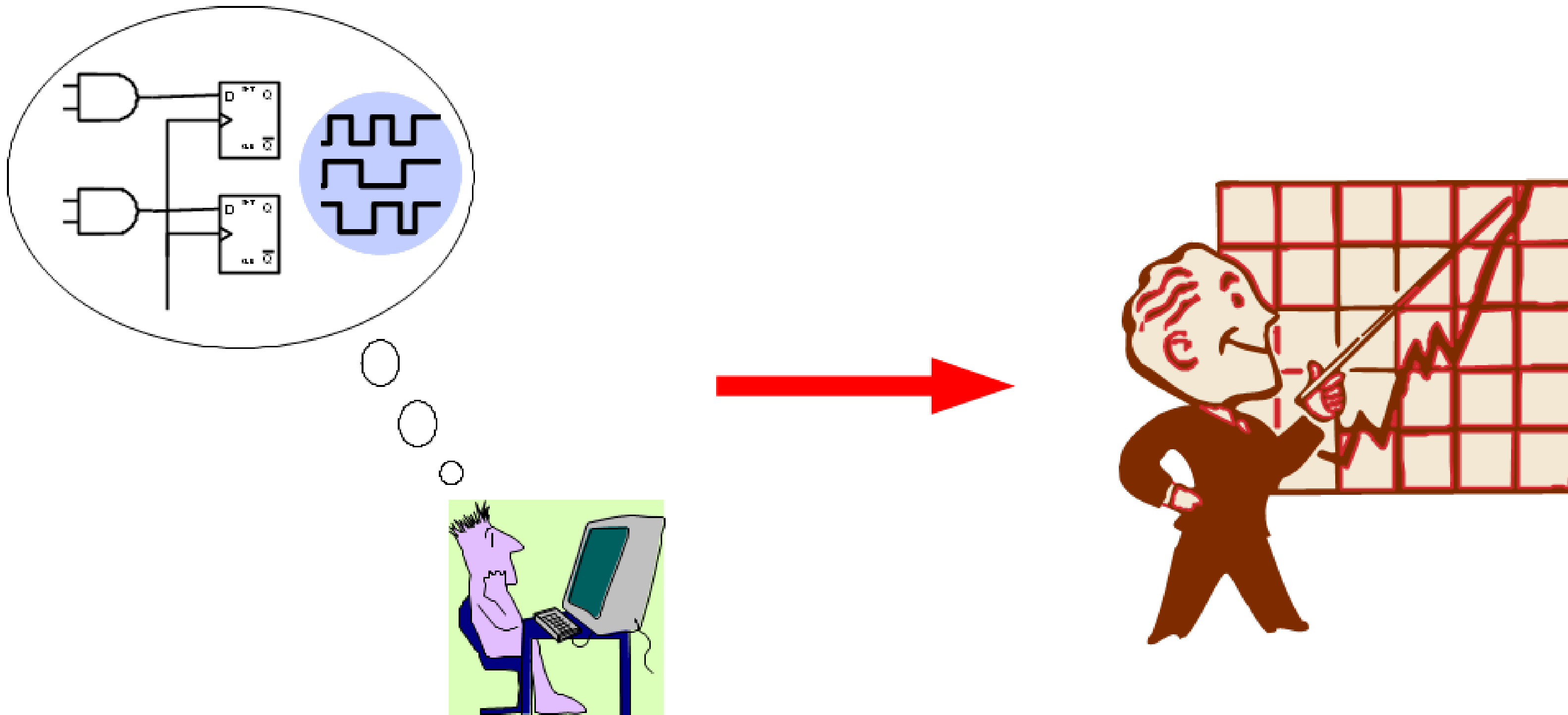


Very High Speed Integrated Circuit Hardware Description Language

Περιγραφική Γλώσσα Επιπέδου Hardware για Πολύ Υψηλής Ταχύτητας Ολοκληρωμένα Κυκλώματα

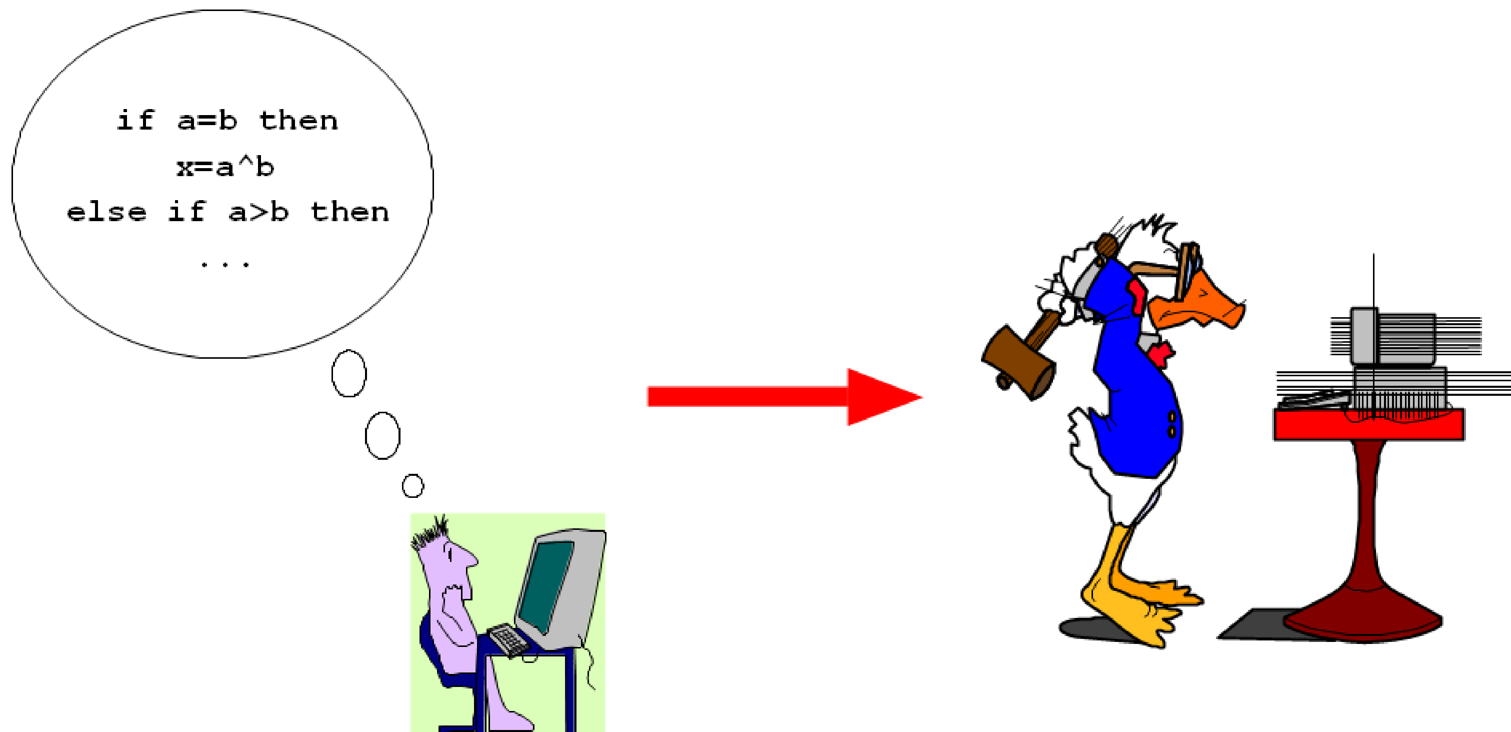
Βασικοί Κανόνες (1/2)

✓ **Πως να γράψεις HDL κώδικα:**



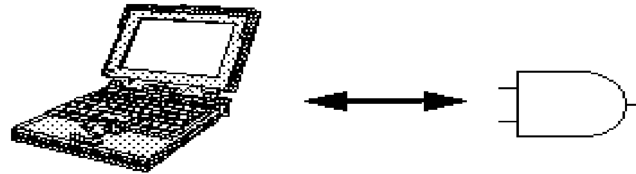
Βασικοί Κανόνες (2/2)

✓ **Πως να <u>ΜΗΝ</u> γράψεις HDL κώδικα:**

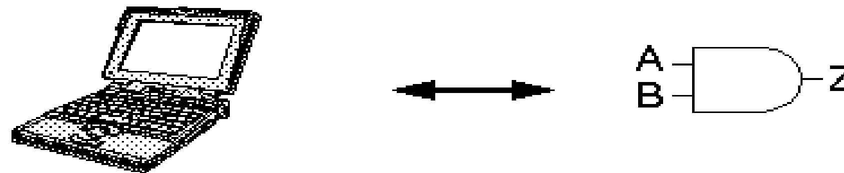


VHDL – Επισκόπηση και Πεδίο Εφαρμογής

✓ Τι είναι hardware;



✓ Τι είδος περιγραφή;



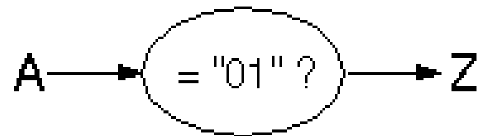

```
If PRINTREQUEST then            Z<=A and B ;
   --print protokoll
End if ;
```

Hardware Description Language (HDL) =

«Γλώσσα προγραμματισμού για μοντελοποίηση *(ψηφιακού)* Hardware»

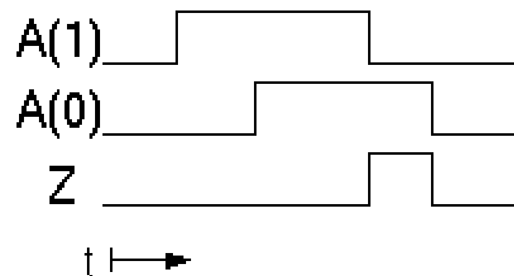
Εφαρμογές των HDLs (1/2)

Modelling

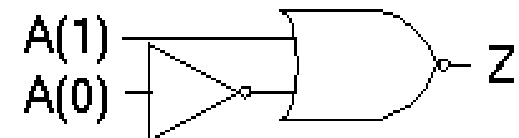



```
Z <= '1' when A="01"
else '0' ;
```

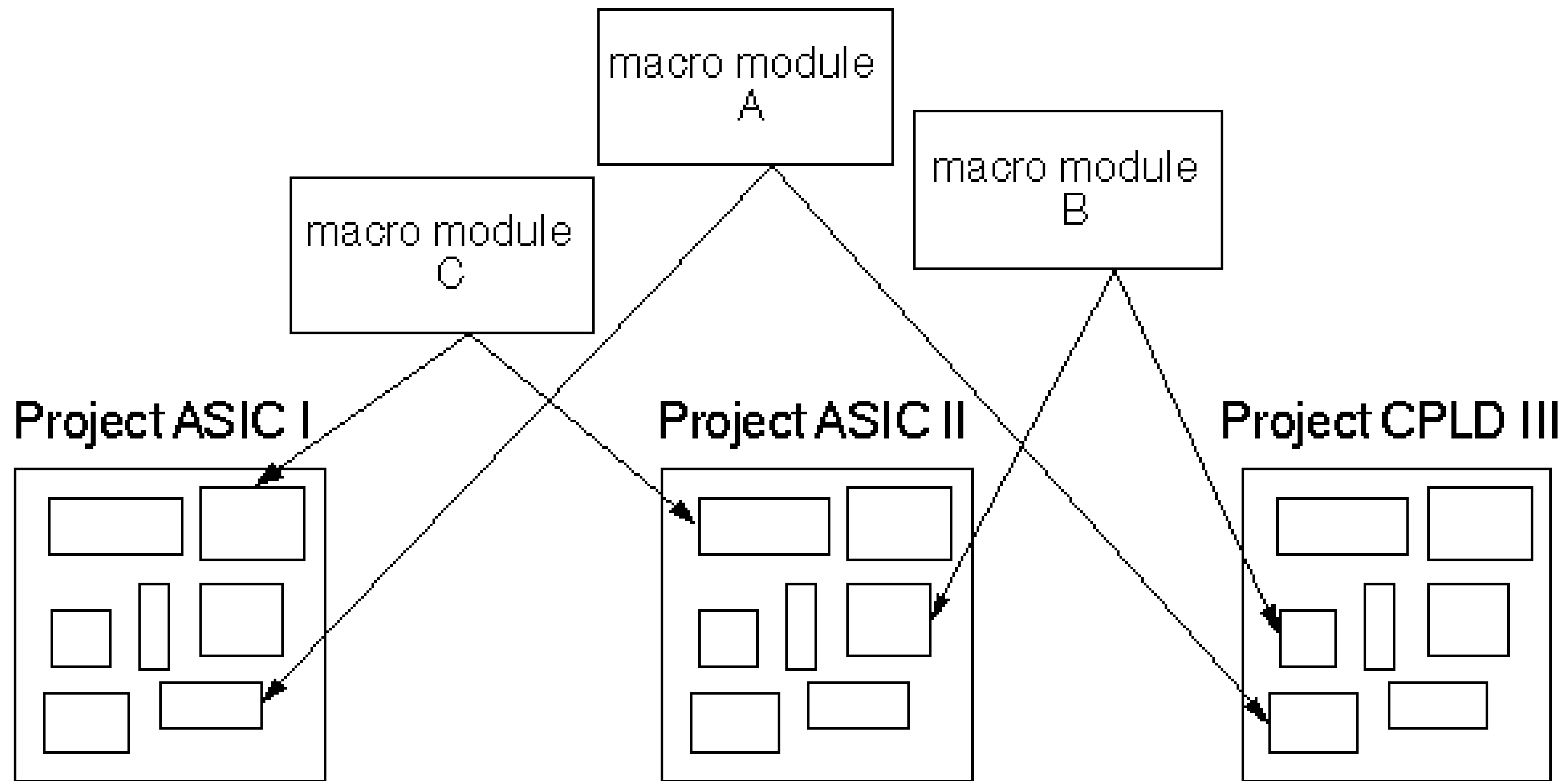
Simulation



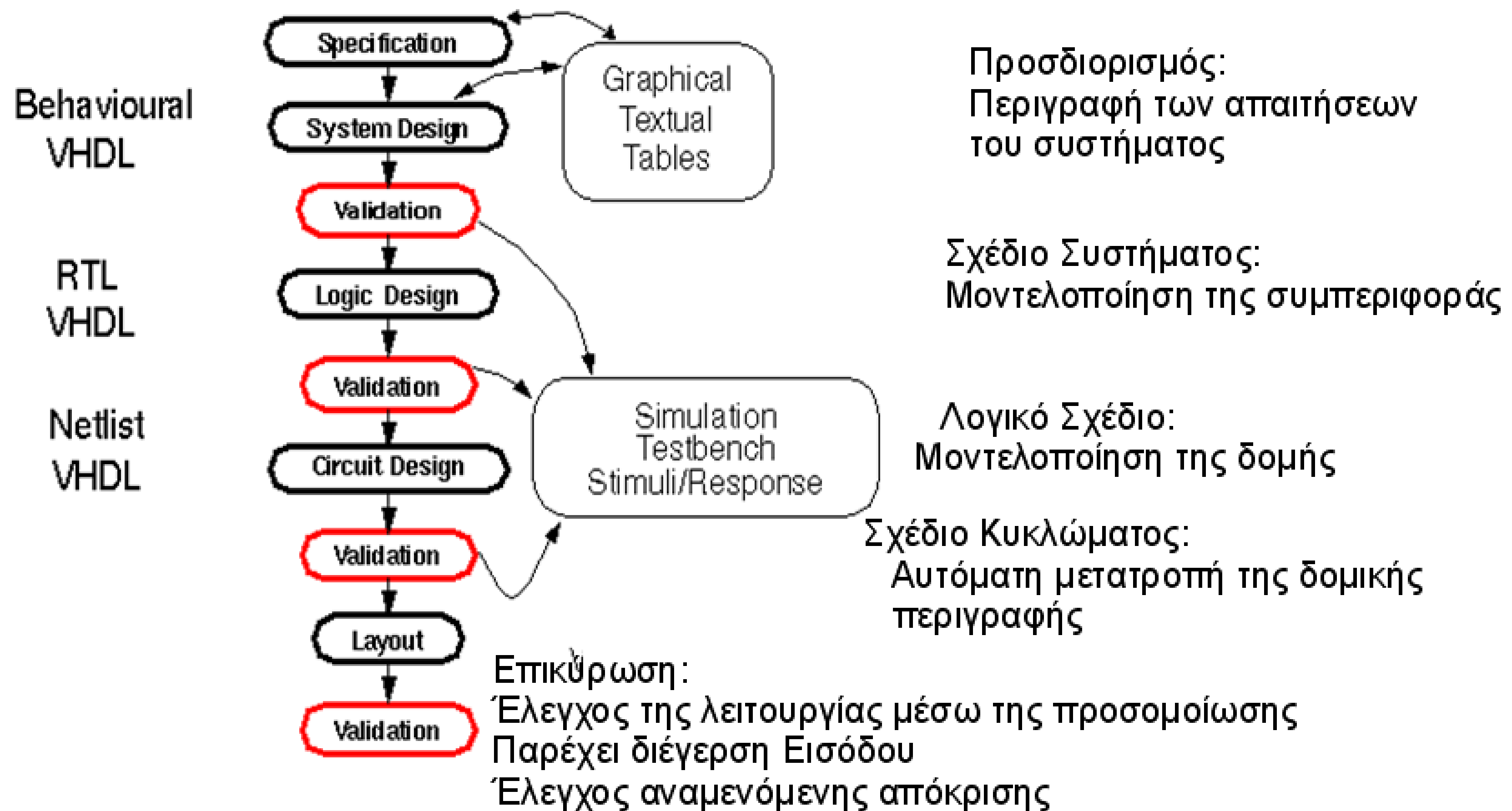
Synthesis



Εφαρμογές των HDLs (2/2)



Εμβέλεια Χρήσης



VHDL - Επισκόπηση

- Very High Speed Integrated Circuit Hardware Description Language.
  - Μοντελοποίηση Ψηφιακών Συστημάτων.
  - Ταυτόχρονες και διαδοχικές δηλώσεις.
  - Προσδιορισμός αναγνώσιμος από τη μηχανή.
  - Χρόνος ζωής Σχεδίου > Χρόνος ζωής του σχεδιαστή.
  - Τεκμηρίωση *(Κείμενο)* αναγνώσιμη από τον άνθρωπο και από τη μηχανή.
- Διεθνή Πρότυπα:
  - IEEE Std 1076-1987.
  - IEEE Std 1076-1993.
- Αναλογική και Μικτού σήματος Επέκταση: VHDL-AMS.
  - IEEE Std 1076.1-1999.
- Καθαρός Ορισμός της Γλώσσας στο LRM *(Language Reference Manual- Εγχειρίδιο αναφοράς της Γλώσσας).*
  - Χωρίς πρότυπα για εφαρμογές και μεθοδολογία.

VHDL – Ιστορία (1/2)

- **early `70s:** Αρχικές Συζητήσεις.
- **late `70s:** Ορισμός των απαιτήσεων.
- **mid -`82:** Συμβόλαιο ανάπτυξης με την IBM, Intermetrics and TI.
- **mid -`84:** Version 7.2.
- **mid -`86:** IEEE- Πρότυπο.

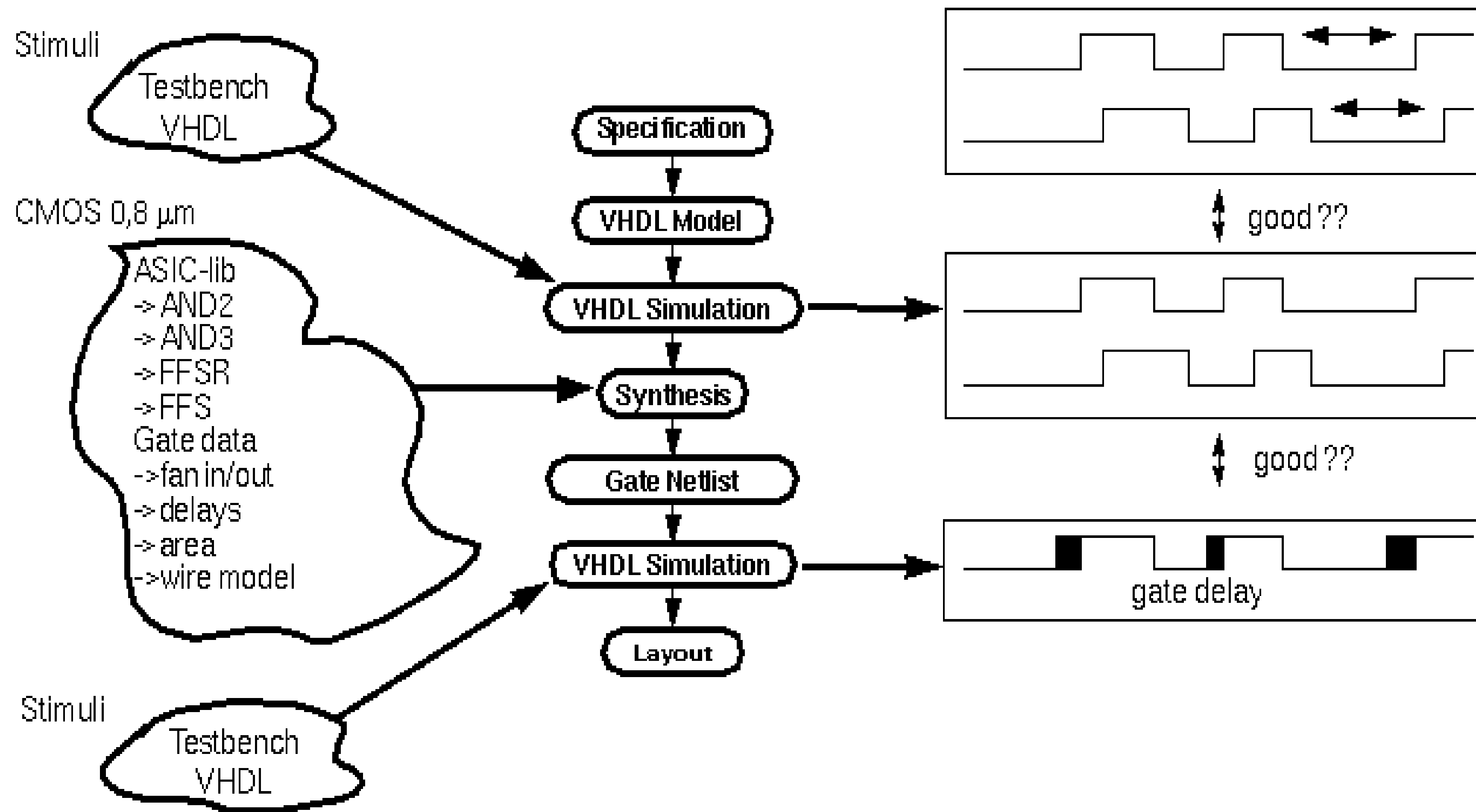
VHDL – Ιστορία (2/2)

- **1987:** Η DoD υιοθετεί το πρότυπο→IEEE.1076.
- **mid -`88:** Ενίσχυση Υποστήριξης από κατασκευαστές CAE.
- **late `91:** Αναθεώρηση.
- **1993:** Νέα πρότυπα.
- **1999:** VHDL-AMS επέκταση.

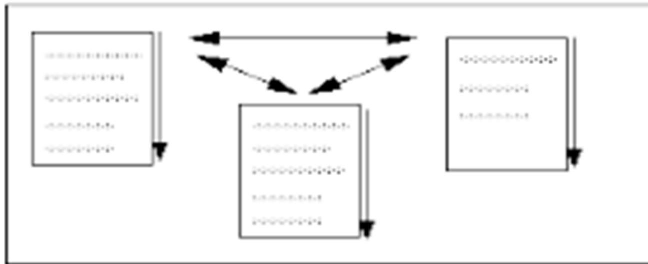
VHDL – Πεδίο Εφαρμογής

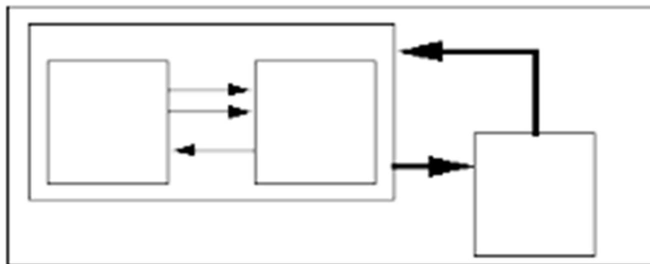
- Hardware σχεδίαση:
  - ASIC: χαρτογράφηση της τεχνολογίας.
  - FPGA: CLB χαρτογράφηση.
  - PLD: μικρότερες δομές, σπάνια οποιαδήποτε χρήση της VHDL.
  - Πρότυπες λύσεις, μοντέλα, περιγραφή συμπεριφοράς, ...
- Software σχεδίαση:
  - VHDL - C interface (tool-specific).
  - Κύριο σημείο έρευνας (hardware/software συ-σχεδίαση).

ASIC Ανάπτυξη



Έννοιες της VHDL





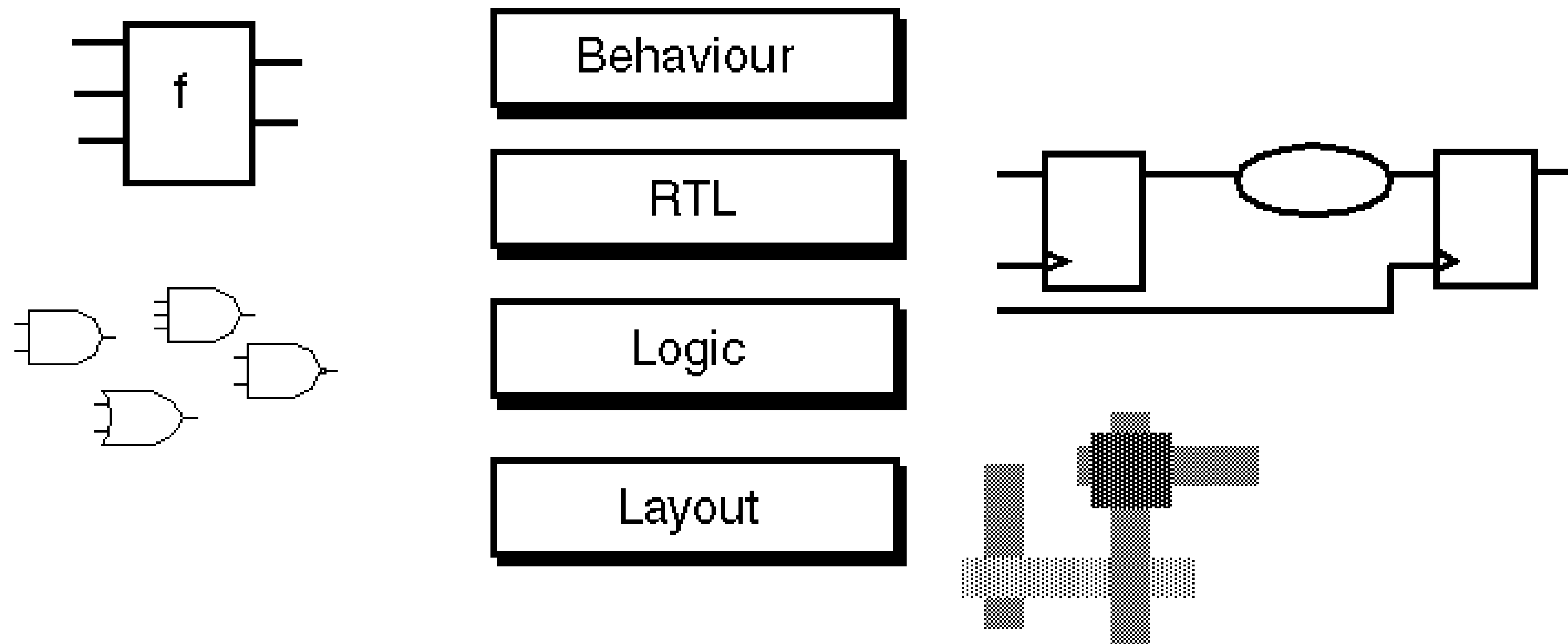
- Εκτέλεση εργασιών:
  - Συνεχόμενες.
  - Ταυτόχρονες.

- Μεθοδολογίες:
  - Αφαίρεση.
  - Συναρμολογούμενα δομοστοιχεία.
  - Ιεραρχία.

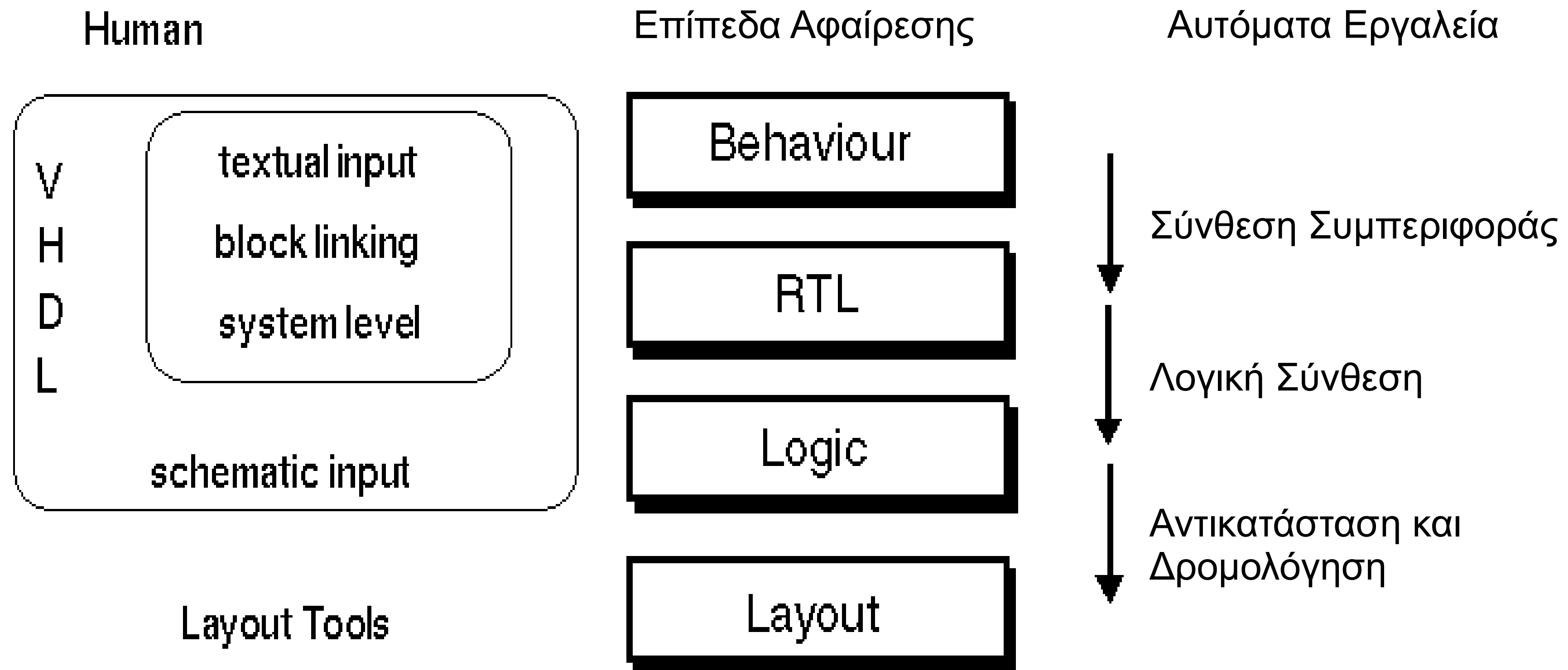
Αφαίρεση

- Η Αφαίρεση κρύβεται από λεπτομέρειες: διαφοροποίηση μεταξύ ουσιώδης και μη πληροφορίας.
- Δημιουργία αφαιρετικών επιπέδων: σε κάθε αφαιρετικό επίπεδο μόνο η ουσιώδης πληροφορία λαμβάνεται υπόψη, η μη ουσιώδης αγνοείται.
- Ισομετρία της αφαίρεσης: Όλη η πληροφορία ενός μοντέλου σε ένα αφαιρετικό επίπεδο περιέχει τον ίδιο βαθμό αφαίρεσης.

Αφαιρετικά επίπεδα στην σχεδίαση των IC



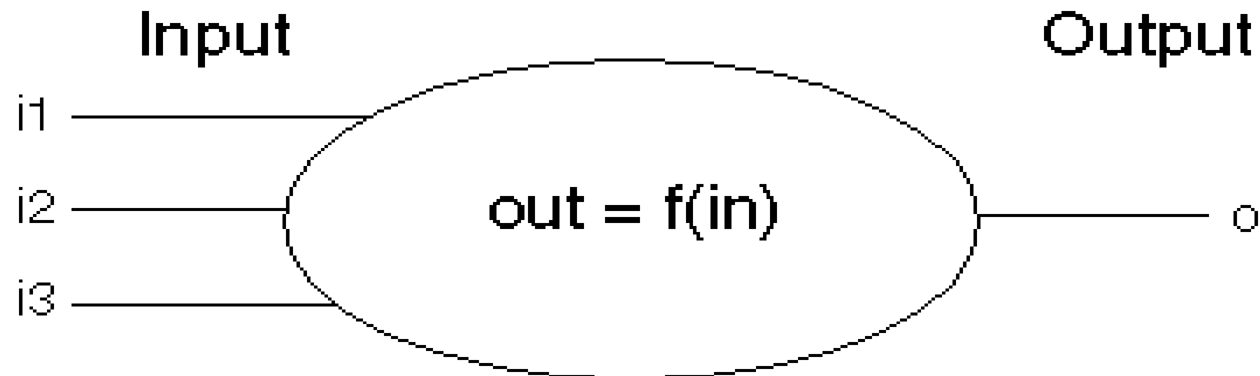
Αφαιρετικά επίπεδα και VHDL

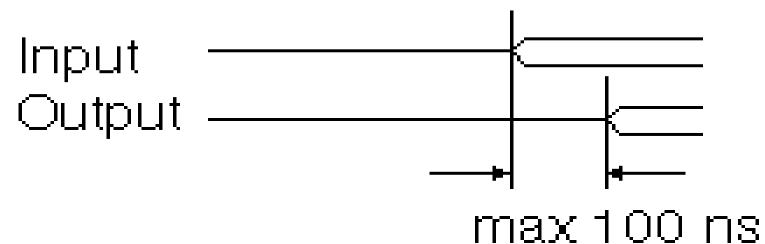


Περιγραφή των αφαιρετικών επιπέδων

Επεξήγηση Συστήματος, μοντέλα πρότυπων συνδεσμολογιών	Behaviour	Αλγοριθμικό Επίπεδο, μοντελοποίηση συστήματος bus, ερεθίσματα εισόδων
ASIC/FPGA σύνθεση, μοντέλα σύνθεσης	RTL	Ανεξάρτητο της Μηχανής, περιγραφή, Καταχωρητές, λογική, ρολόι
Επίπεδο πυλών, ανάπτυξη PLD	Logic	netlists, δομή πυλών
Πλήρες σύνηθες σχέδιο	Layout	Περιγραφή Τεχνολογίας (π.χ CMOS 0.35μm)

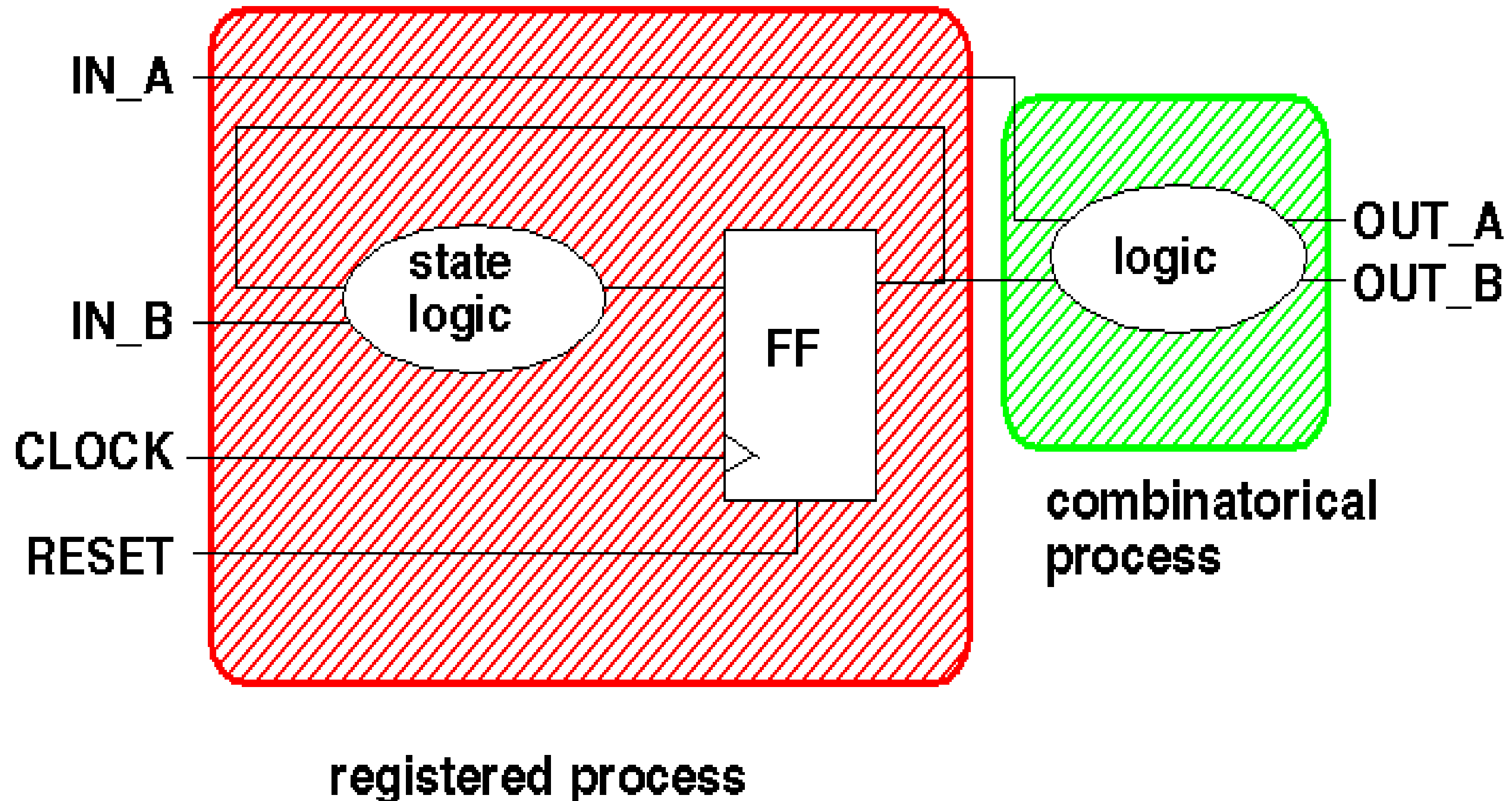
Περιγραφή συμπεριφοράς στην VHDL



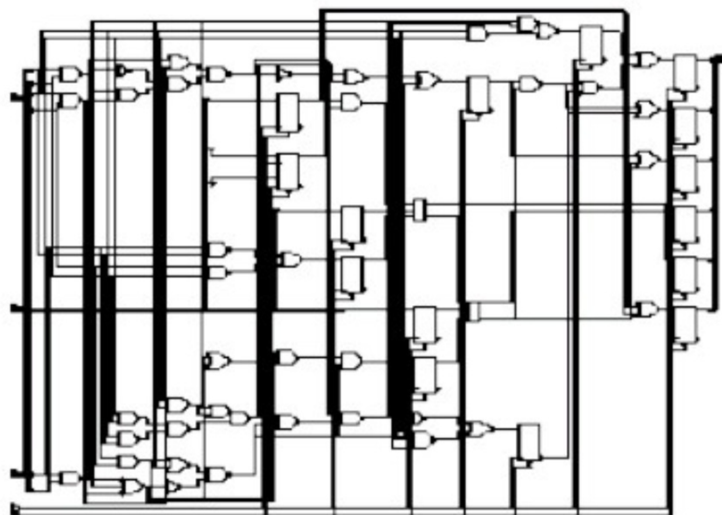


o <= μεταφορά i1 + i2 * i3 μετά από 100 ns;

RT Επίπεδο στην VHDL

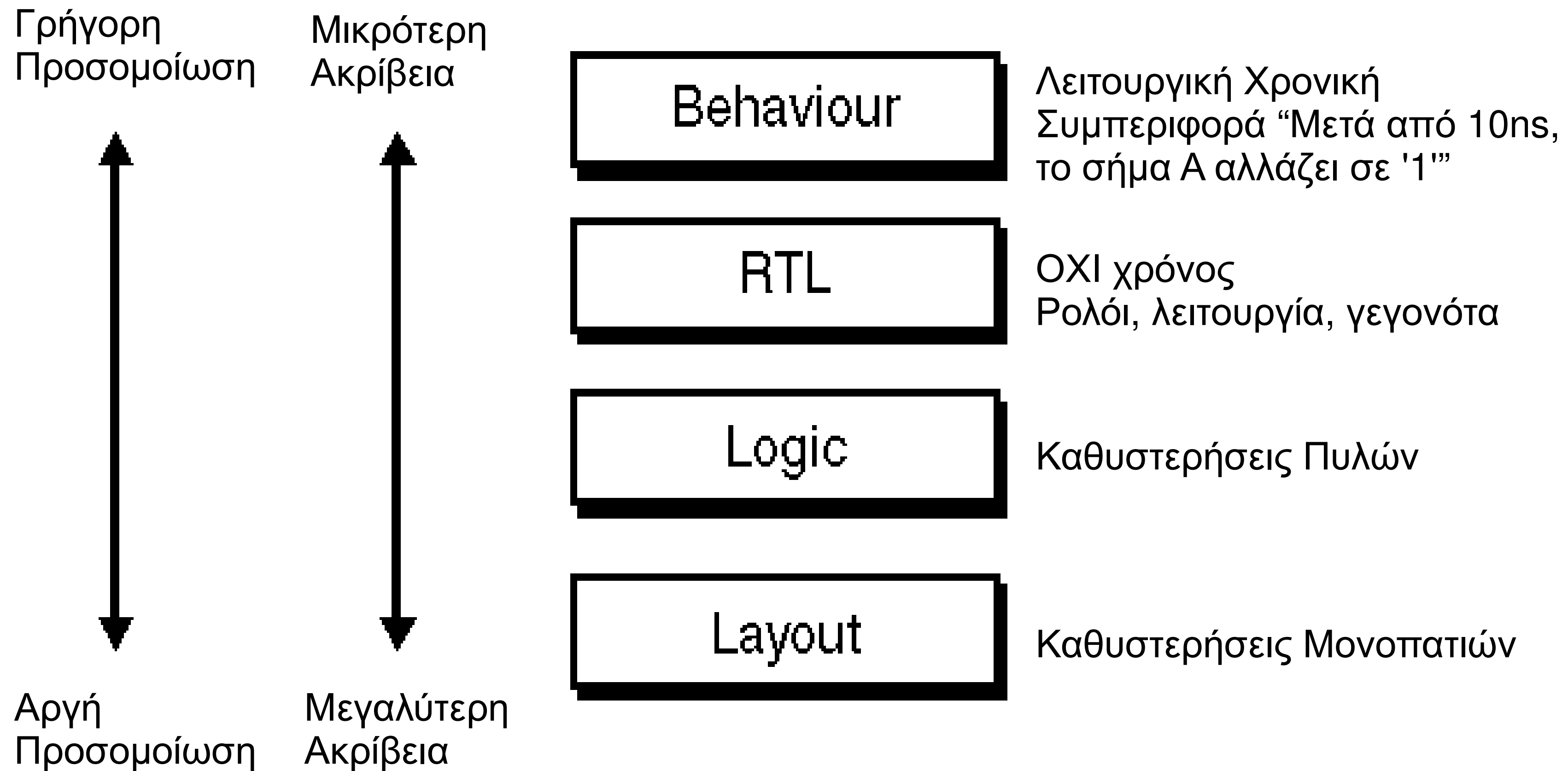


Επίπεδο πυλών στη VHDL



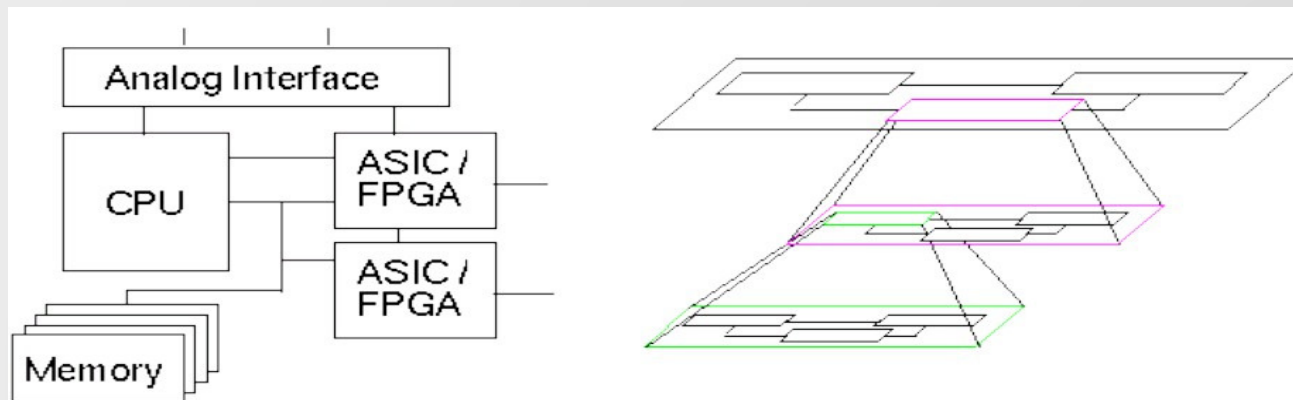
```
U86 : ND2 port map( A => n192, B => n191, Z => n188);
U87 : ND2 port map( A => I3_2, B => I2_0, Z => n175);
U88 : ND2 port map( A => I2_2, B => I3_0, Z => n173);
U89 : NR2 port map( A => mul_36_PROD_not_0,
        B => n174, Z => n185);
U90 : EN port map( A => n181, B => n182, Z => n180);
U91 : ND2 port map( A => I3_2, B => I2_1, Z => n181);
U92 : ND2 port map( A => I2_2, B => I3_1, Z => n182);
U93 : IVP port map( A => n180, Z => n192);
U94 : AO6 port map( A => n173, B => n174, C => n175,
        Z => n172);
U95 : NR2 port map( A => n174, B => n173, Z => n176);
U96 : ND2 port map( A => I3_1, B => I2_1, Z => n174);
U97 : EN port map( A => n183, B => n178,
        Z => product64_4);
U98 : ND3 port map( A => I2_2, B => I3_2, C => n174,
        Z => n183);
```

Πληροφοριακό περιεχόμενο των αφαιρετικών επιπέδων



Σπονδύλωση (modularity) και Ιεραρχία

- Συμμετοχή σε πολλά τμηματικά σχέδια:
  - Περιορίζει την πολυπλοκότητα.
  - Δίνει τη δυνατότητα για ομαδική δουλειά.
  - Μελέτη εναλλακτικών υλοποιήσεων.
  - Μακροεντολές.
  - Μοντέλα προσομοίωσης.



Σχεδιάζοντας με HDL

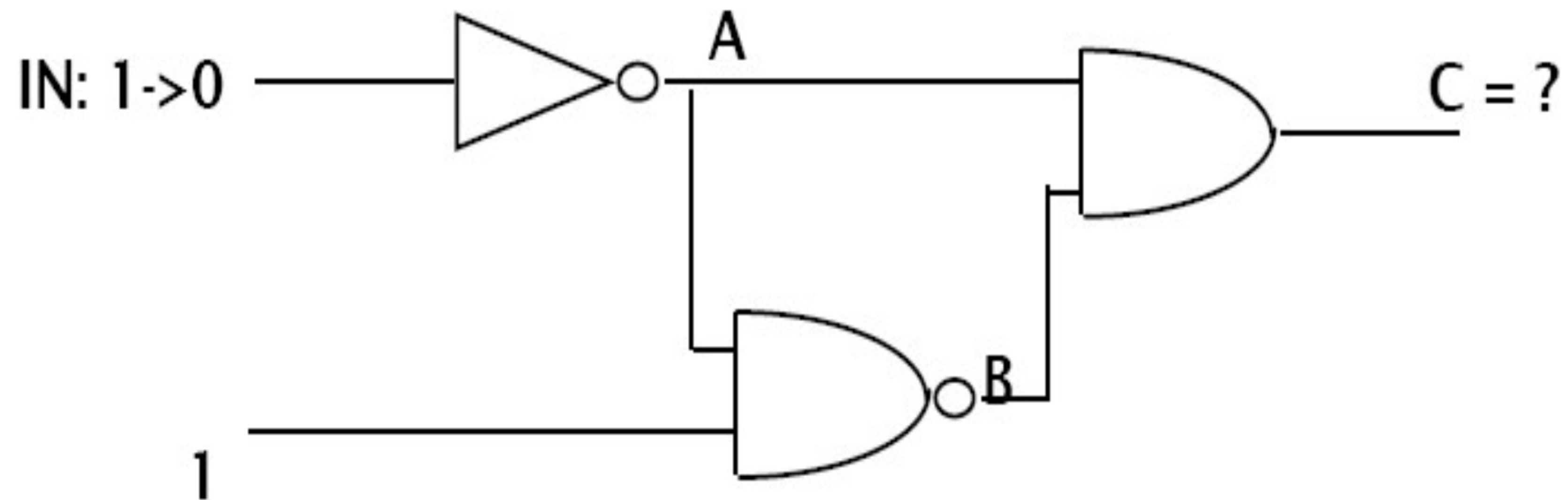
Ερώτηση:

- Πως γνωρίζουμε ότι δεν έχουμε κάνει λάθος όταν χειροκίνητα σχεδιάζουμε ένα σχηματικό και συνδέουμε εξαρτήματα για να υλοποιήσουμε μία λειτουργία;

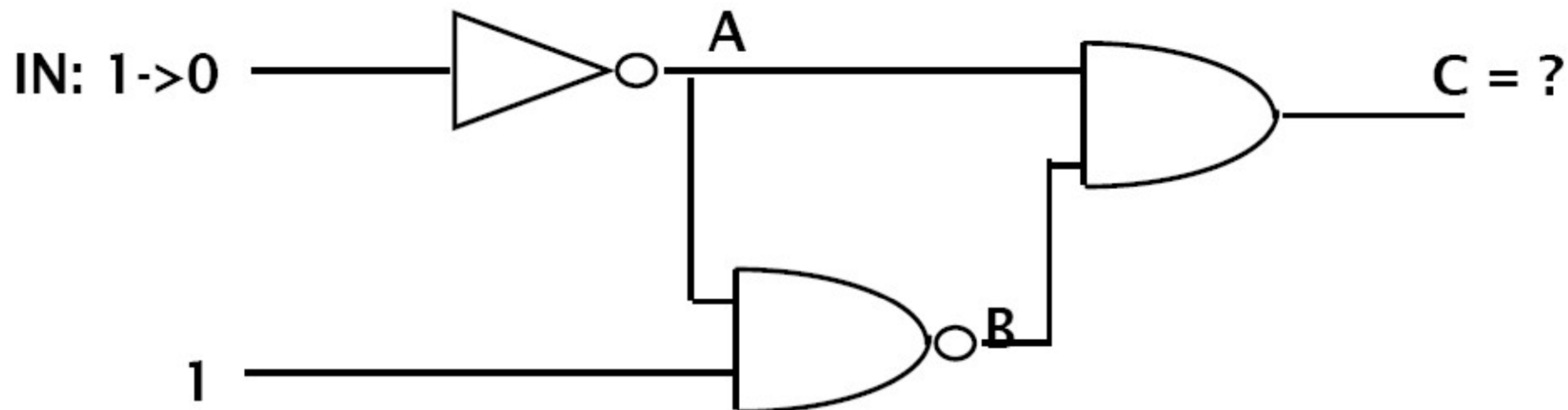
Απάντηση:

- Περιγράφοντας το σχέδιο σε μια υψηλού επιπέδου (=κατανοητή) γλώσσα, μπορούμε να προσομοιώσουμε το σχέδιο μας πριν το κατασκευάσουμε.
  - Αυτό μας επιτρέπει να εντοπίζουμε λάθη σχεδίασης, π.χ., το σχέδιο δεν λειτουργεί με τον τρόπο που πιστεύαμε ότι θα δούλευε.
- Η προσομοίωση εγγυάται ότι το σχέδιο συμπεριφέρεται όπως θα έπρεπε.

Πως λειτουργεί η προσομοίωση;



Ποια είναι η έξοδος της C;



NAND gate evaluated first:

IN: 1->0

A: 0->1

B: 1->0

C: 0->0

AND gate evaluated first:

IN: 1->0

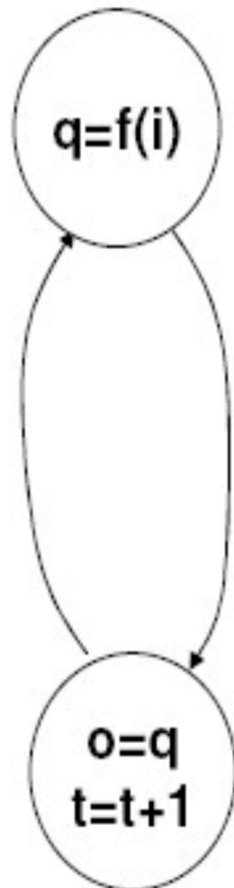
A: 0->1

C: 0->1

B: 1->0

C: 1->0

Προσομοιωτές Κυκλική Ροής



- Περνάνε από όλες τις συναρτήσεις χρησιμοποιώντας τις τρέχουσες εισόδους και υπολογίζουν τις επόμενες εξόδους.

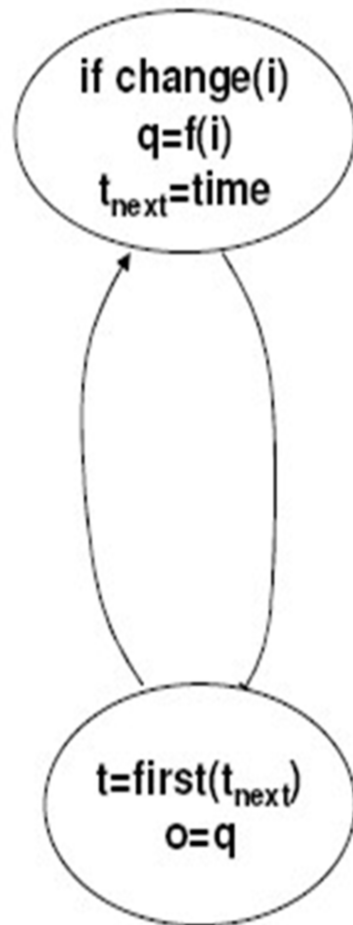
- Ενημερώνουν τις εξόδους και αυξάνουν το χρόνο κατά μία μονάδα καθυστέρησης.

Προσομοιωτές Γεγονότων

- Περνάνε από όλες τις συναρτήσεις των οποίων οι είσοδοι έχουν αλλάξει και υπολογίζουν την επόμενη έξοδο.

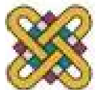
- Ενημερώνουν τις εξόδους και αυξάνουν το χρόνο κατά μία μονάδα καθυστέρησης.

Προσομοιωτές Γεγονότων με Ουρές Γεγονότων



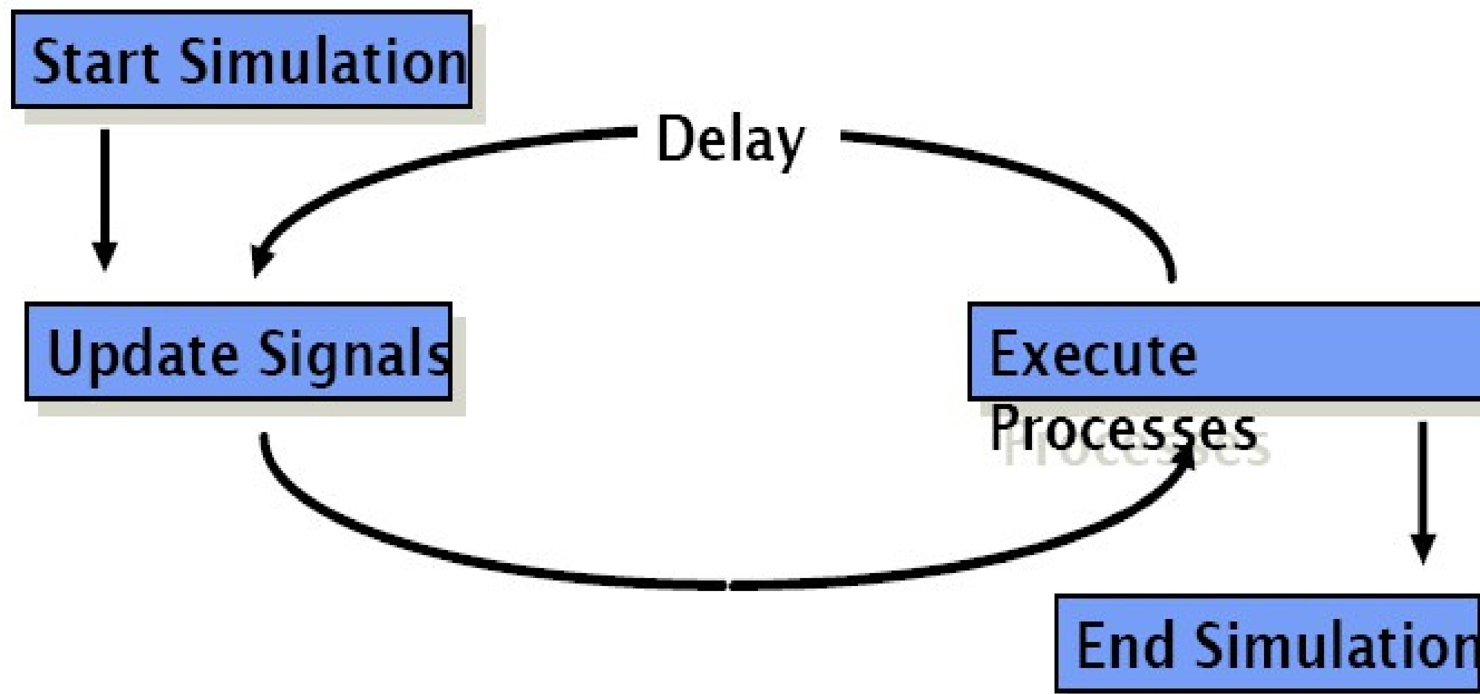
- Περνάνε από όλες τις συναρτήσεις των οποίων οι είσοδοι Έχουν αλλάξει και υπολογίζουν την τιμή και τον χρόνο για την επόμενη αλλαγή της εξόδου.

- Αυξάνουν τον χρόνο ώστε πρώτα να προγραμματιστεί το γεγονός & ενημερώνουν τα σήματα.



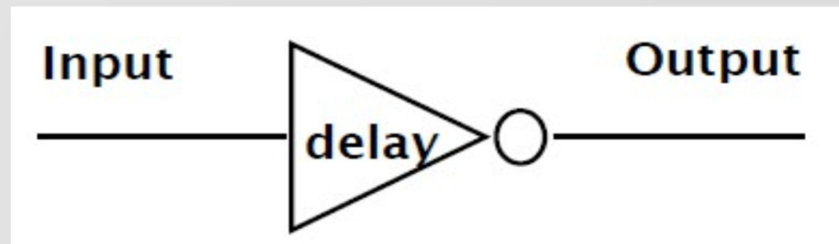
Κύκλος Προσομοίωσης της VHDL

Η VHDL χρησιμοποιεί έναν κύκλο προσομοίωσης για να μοντελοποιήσει τις διεγέρσεις και την φυσική αντίδραση του ψηφιακού hardware.



Μοντέλα Καθυστέρησης της VHDL

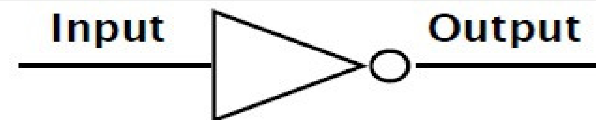
- Η καθυστέρηση δημιουργείται προγραμματίζοντας μία εκχώρηση σήματος για μια μελλοντική χρονική στιγμή.
- Η καθυστέρηση σε έναν VHDL κύκλο μπορεί να είναι διαφορετικών τύπων:
  - Αδρανειακή.
  - Μεταφοράς.
  - Δέλτα.

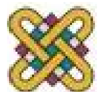


Αδρανειακή Καθυστέρηση

- Default τύπος καθυστέρησης.
- Επιτρέπει στο χρήστη καθορισμένη καθυστέρηση.
- Απορροφά παλμούς μικρότερης διάρκειας από την καθορισμένη καθυστέρηση.

```
-- Inertial is the default
Output <= NOT Input AFTER 10 ns;
```

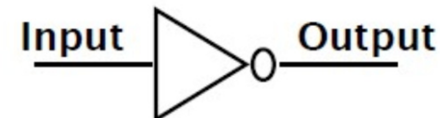


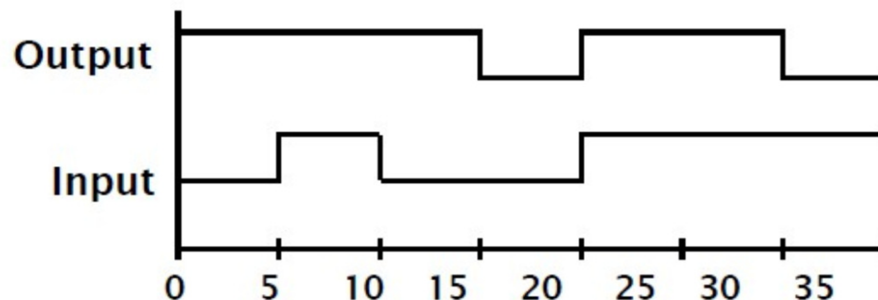


Καθυστέρηση Μεταφοράς

- Πρέπει να είναι ρητά καθορισμένη από το χρήστη.
- Επιτρέπει στο χρήστη καθορισμένη καθυστέρηση.
- Περνάει όλες τις μεταβάσεις της εισόδου με καθυστέρηση.

```
-- TRANSPORT must be specified
Output <= TRANSPORT NOT Input AFTER 10 ns;
```





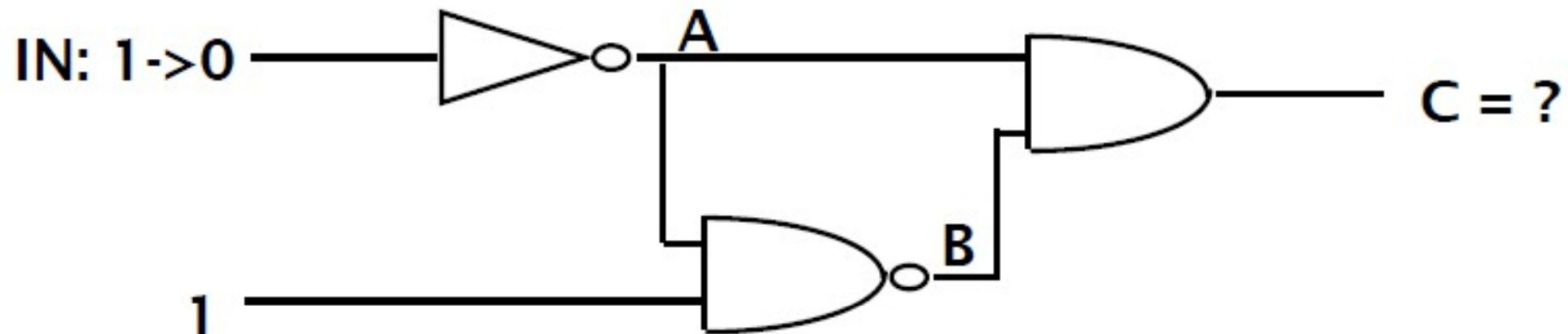
Καθυστέρηση Δέλτα

Η καθυστέρηση Δέλτα απαιτείται για να παρέχει υποστήριξη στις τρέχουσες λειτουργίες με μηδενική καθυστέρηση.

- Η σειρά εκτέλεσης για εξαρτήματα με μηδενική καθυστέρηση δεν είναι ξεκάθαρη.

- **Ο προγραμματισμός συσκευών μηδενικής καθυστέρησης απαιτεί την Δέλτα καθυστέρηση.**
  - Μια καθυστέρηση Δέλτα είναι απαραίτητη αν δεν καθορίζεται κάποια άλλη καθυστέρηση.
  - Μια καθυστέρηση Δέλτα δεν προωθεί τον χρόνο του προσομοιωτή.
  - Μία καθυστέρηση Δέλτα είναι μια απειροελάχιστη ποσότητα χρόνου.
  - Η Δέλτα είναι ένας προγραμματιζόμενος μηχανισμός, ώστε να εξασφαλίσει επαναληψιμότητα.

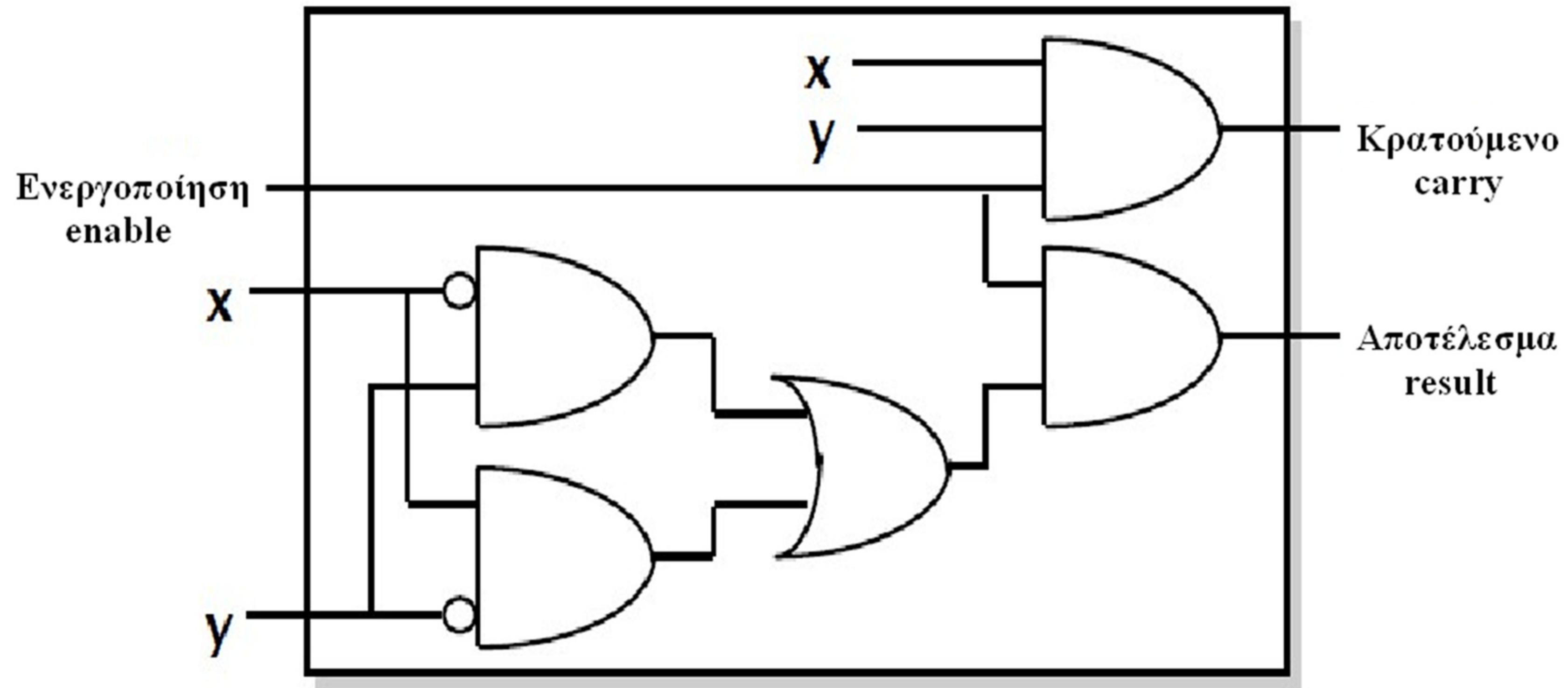
Παράδειγμα – Καθυστέρηση Δέλτα



Using delta delay scheduling

Time	Delta	Event
0 ns	1	IN: 1->0
		eval inverter
	2	A: 0->1
		eval NAND, AND
	3	B: 1->0
		C: 0->1
		eval AND
	4	C: 1->0
1 ns		

Πως γράφουμε κώδικα;



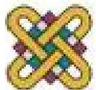
Βασική μορφή VHDL κώδικα

- Κάθε σχέδιο περιγραφής VHDL αποτελείται από τουλάχιστον μία οντότητα/ζευγάρι αρχιτεκτονικής, ή μία οντότητα με πολλές αρχιτεκτονικές.
  - Το τμήμα της οντότητας χρησιμοποιείται για να ορίσει τις θύρες Ε/Ε του κυκλώματος .
  - Το τμήμα της αρχιτεκτονικής περιγράφει την συμπεριφορά του κυκλώματος.
- Ένα μοντέλο συμπεριφοράς είναι παρόμοιο με ένα "μαύρο κουτί".
- Πρότυπες βιβλιοθήκες σχεδίασης περιλαμβάνονται πριν από τον ορισμό της οντότητας.

Πρότυπες Βιβλιοθήκες

- Περιλαμβάνει βιβλιοθήκη της ieee; πριν τον ορισμό της οντότητας.
- ieee.std_logic_1164 ορίζει ένα πρότυπο για τους σχεδιαστές για να χρησιμοποιηθεί στην περιγραφή των τύπων δεδομένων διασύνδεσης στην VHDL μοντελοποίηση.
- ieee.std_logic_arith παρέχει ένα σύνολο συναρτήσεων αριθμητικής, μετατροπής, σύγκρισης για τύπους προσημασμένους- signed, μη- unsigned, std_ulogic, std_logic, std_logic_vector.
- Ieee.std_logic_unsigned παρέχει ένα σύνολο συναρτήσεων μη προσημασμένης αριθμητικής,μετατροπής και σύγκρισης για std_logic_vector.
- Βλέπε όλα τα διαθέσιμα πακέτα στο

  csee.umbc.edu/portal/help/VHDL/stdpkg



Ονοματολογία και Ετικέτες

Γενικοί κανόνες διαχείρισης (σύμφωνα με το VHDL-87):

- Όλα τα ονόματα πρέπει να αρχίζουν με έναν αλφαβητικό χαρακτήρα (a-z ή A-Z).
- Χρησιμοποιείστε μόνο αλφαβητικούς χαρακτήρες (a-z ή A-Z) ψηφία (0-9) και underscore (_).
- Μην χρησιμοποιείται οποιοδήποτε δεσμευμένο χαρακτήρα ή τονισμού σε ένα όνομα (!, ?, ., &, +, -, etc.)
- Μην χρησιμοποιείται δύο ή περισσότερα διαδοχικά underscore (__) μέσα σε ένα όνομα (e.g., Sel__A is invalid).
- Όλα τα ονόματα και οι ετικέτες που δίνονται σε μία οντότητα ή αρχιτεκτονική πρέπει να είναι μοναδικά.

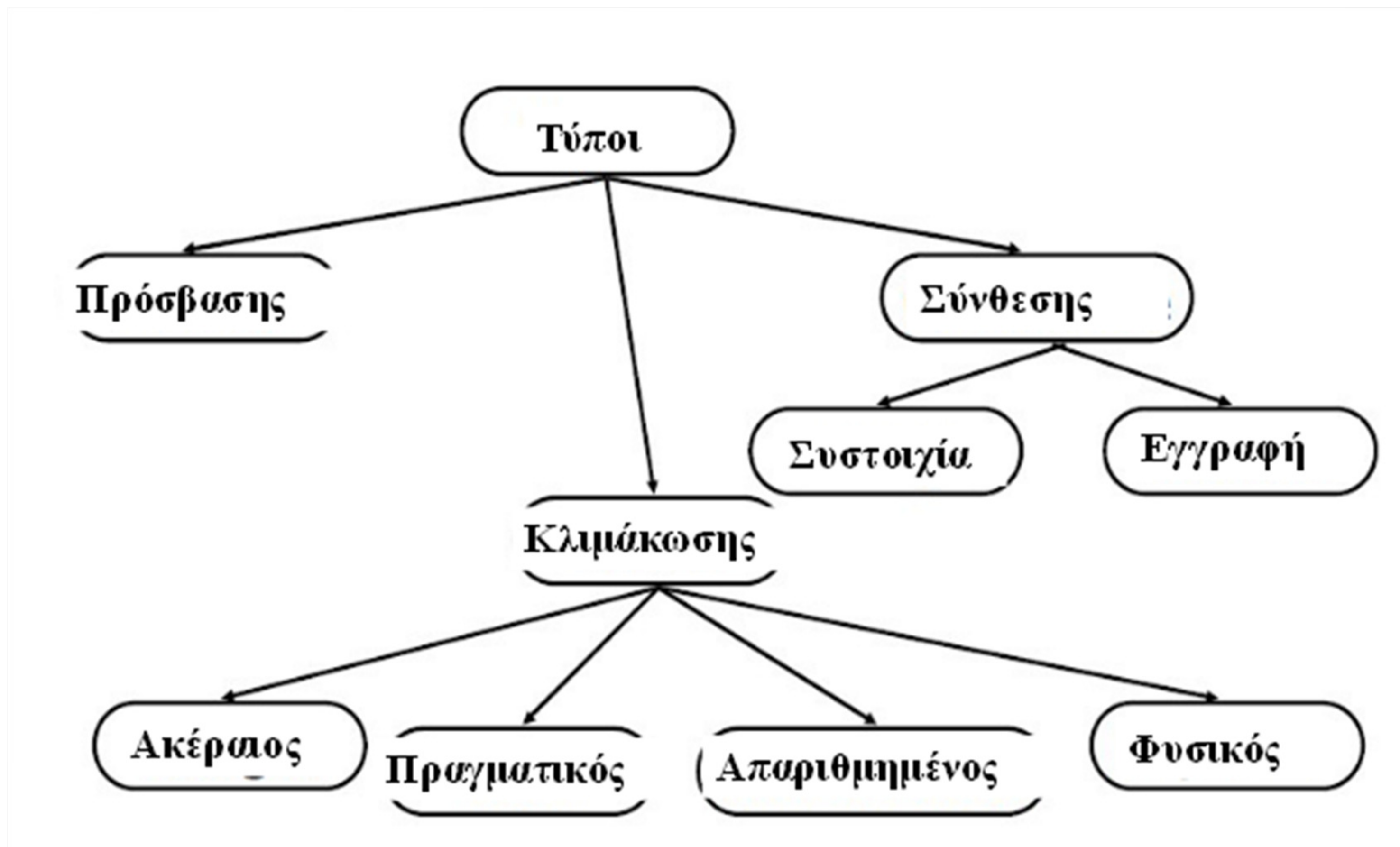
Ελεύθερη μορφή

- Η VHDL είναι μια γλώσσα ελεύθερης μορφής.
- Δεν γίνονται μετατροπές μορφής, όπως κενά ή εσοχές που επιβάλλονται από τους compilers της VHDL.
- Κενά και μεταφορές αντιμετωπίζονται με τον ίδιο τρόπο.

Παράδειγμα:

```
 If (a=b) then
Or
    If (a=b)        then
Or
    If (a =
    b) then
Είναι όλα ισοδύναμα
```

Τύποι Δεδομένων στην VHDL



Προκαθορισμένοι τύποι δεδομένων

- bit (‘0’ or ‘1’).
- Διάνυσμα bit (πίνακας από bits).
- Ακέραιος.
- Πραγματικός.
- Χρόνος (φυσικός τύπος δεδομένων).

Ακέραιος

- Ακέραιος:
  - Ελάχιστο εύρος για οποιαδήποτε υλοποίηση όπως ορίζεται από το πρότυπο:
    - -2,147,483,647 to 2,147,483,647
  - Παράδειγμα ανάθεσης ακεραίου.

```
ARCHITECTURE test_int OF  test IS
BEGIN
        PROCESS (X)
            VARIABLE a: INTEGER;
        BEGIN
            a := 1; -- ok
            a := -1; -- ok
            a := 1.0; -- bad
        END PROCESS;
END TEST;
```

Πραγματικός

- Πραγματικός:
  - Ελάχιστο εύρος για οποιαδήποτε υλοποίηση όπως ορίζεται από το πρότυπο:
    - -1.0E38 to 1.0E38
  - Παράδειγμα ανάθεσης πραγματικού τύπου:

```
ARCHITECTURE test_real OF  test
IS
BEGIN
        PROCESS (X)
             VARIABLE a: REAL;
        BEGIN
            a := 1.3; -- ok
            a := -7.5; -- ok
            a := 1; -- bad
            a := 1.7E13; -- ok
            a := 5.3 ns; -- bad
         END PROCESS;
END TEST;
```

Απαρηθμιμένα

```
type BIT is ( '0', '1' );                    -- the BIT is a predefined
                                             -- enumeration type in VHDL

type BOOLEAN is (FALSE, TRUE);               -- BOOLEAN is a predefined
                                             -- enumeration type in VHDL

type SEVERITY_LEVEL is ( NOTE, WARNING, ERROR, FAILURE );
                                             -- SEVERITY_LEVEL is predefined
                                             -- enumeration type in VHDL

type FILE_OPEN_KIND is ( READ_MODE, WRITE_MODE, APPEND_MODE );
                                             -- FILE_OPEN_KIND is predefined
                                             -- enumeration type in VHDL

type FILE_OPEN_STATUS is ( OPEN_OK, STATUS_ERROR, NAME_ERROR, MODE_ERROR );
                                             -- FILE_OPEN_STATUS predefined
                                             -- enumeration type in VHDL

type THREE_STATE is ('U','0','1','Z');       -- set up THREE_STATE as
                                             -- variable that can have
                    -- bit values plus Z for tristate and U for unknown

type STATE is ( IDLE, SEND, RECEIVE, PAUSE, SIGNAL );
                                             -- STATE is one of five values
                                             -- set up for state machine

type HEIGHT is  (SHORT, AVERAGE, TALL);      -- HEIGHT is set up to classify
                                             -- objects into three groups
                    -- In this type definition, SHORT is assigned to
                    -- position 0, AVERAGE to 1, and TALL to 2, so that
                    -- TALL > AVERAGE > SHORT.
```

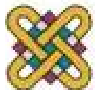
Φυσικός

- Φυσικός:
  - Μπορεί να οριστεί το εύρος από τον χρήστη.
  - Παράδειγμα φυσικού τύπου.

```
TYPE resistance IS RANGE 0 to 1000000

UNITS
     ohm; --ohm
     kohm = 1000 ohm; -- 1 KΩ
     mohm = 1000 kohm; -- 1 MΩ
END UNITS;
```

```
type DISTANCE is range 0 to 1E5
units
   um; --micrometer
   mm=1000 um; -- millimeter
   in_a =25400 um; -- inch
end units DISTANCE;
variable Dis1,Dis2 : DISTANCE;
Dis1 := 28 mm;
Dis2 :=2 in_a -1 mm;
If Dis1 < Dis2 then ...
```

Οι μονάδες χρόνου είναι οι μοναδικοί προκαθορισμένοι φυσικοί τύποι στην VHDL.



Πίνακας

- Πίνακας:
  - Χρησιμοποιείται για να συλλέξει ένα ή περισσότερα στοιχεία ίδιου τύπου σε μία μόνο δομή.
  - Τα στοιχεία μπορεί να είναι οποιουδήποτε τύπου δεδομένων VHDL.

```
TYPE data_bus IS ARRAY (0 TO 31) OF BIT;
```

0...element numbers... 31

0	...array values...	1

```
VARIABLE X:   data_bus;
VARIABLE Y:   BIT

Y  := X(12)  ;   -- Y gets value of 12th element
```

```
TYPE register IS ARRAY  (15 DOWNTO 0) OF BIT;
```

Εγγραφές

- Εγγραφές:
  - Χρησιμοποιούνται για να συλλέξουν ένα ή περισσότερα στοιχεία διαφορετικού τύπου σε μία μόνο δομή.
  - Τα στοιχεία μπορεί να είναι οποιουδήποτε τύπου της VHDL.
  - Τα στοιχεία προσπελάζονται μέσω του ονόματος του πεδίου.

```
TYPE binary IS ( ON, OFF) ;
TYPE switch_info IS
      RECORD
         status : binary;
         IDnumber : integer;
      END RECORD;
VARIABLE switch : switch_info;

switch.status := on;  -- status of the switch
switch.Idnumber :=30; --number of the switch
```

Υπό-τύπος

- Υπό-τύπος:
  - Επιτρέπει στο χρήστη να καθορίσει περιορισμούς σε έναν τύπο δεδομένων.
  - Μπορεί να περιλαμβάνει όλο το εύρος ενός βασικού τύπου.
  - Αναθέσεις εκτός του εύρους του υπό-τύπου καταλήγουν σε σφάλμα.
  - Παράδειγμα υπό-τύπου:

```
SUBTYPE name IS  base_type  RANGE <user range>;

SUBTYPE first_ten IS INTEGER RANGE 0 to 9;
```

Φυσικοί και θετικοί ακέραιοι

- Ακέραιος υπό-τύπος:
  - Subtype Natural is integer range 0 to integer'high;
  - Subtype Positive is integer range 1 to integer”high;

Boolean, Bit και Bit_διάνυσμα

- type Boolean is (false, true);
- Type Bit is (‘0’, ‘1’);
- type Bit_vector is array (integer range <>) of bit;

Char και String

- Type Char is (NUL, SOH, …, DEL:
  - 128 chars in VHDL’87.
  - 256 chars in VHDL’93.

- type String is array (positive range <>) of Char;

```
Value set is array of characters
        TYPE string IS ARRAY (POSITIVE RANGE <>) OF character;

        SIGNAL stringName : STRING (4 downto 0) :=”START”;
        stringName <=” VALUE”;
```

STD_LOGIC

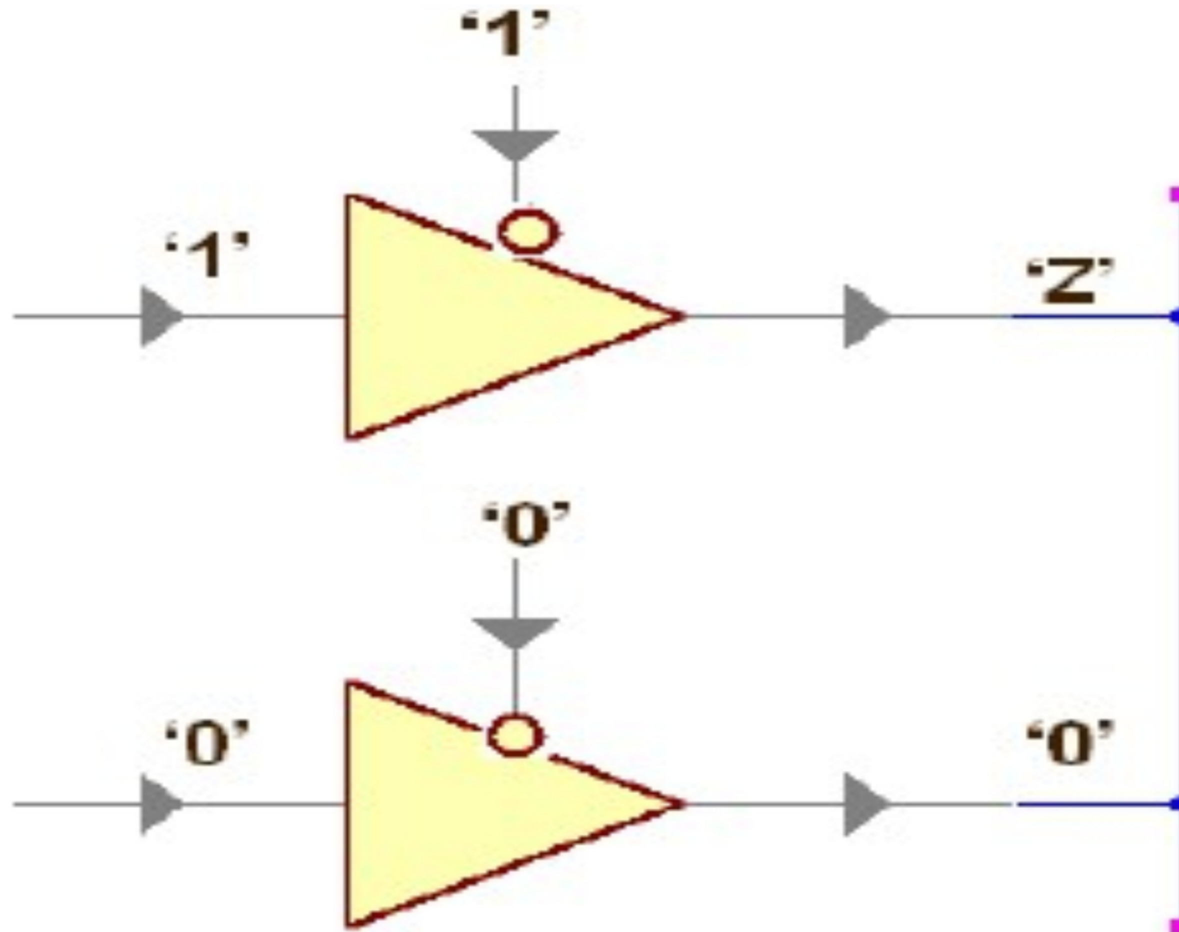
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
	PORT(
		a    : IN STD_LOGIC;
		b    :IN STD_LOGIC;
		z    :OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
	Z <= a NAND b;
END model
```

Τι είναι STD_LOGIC;

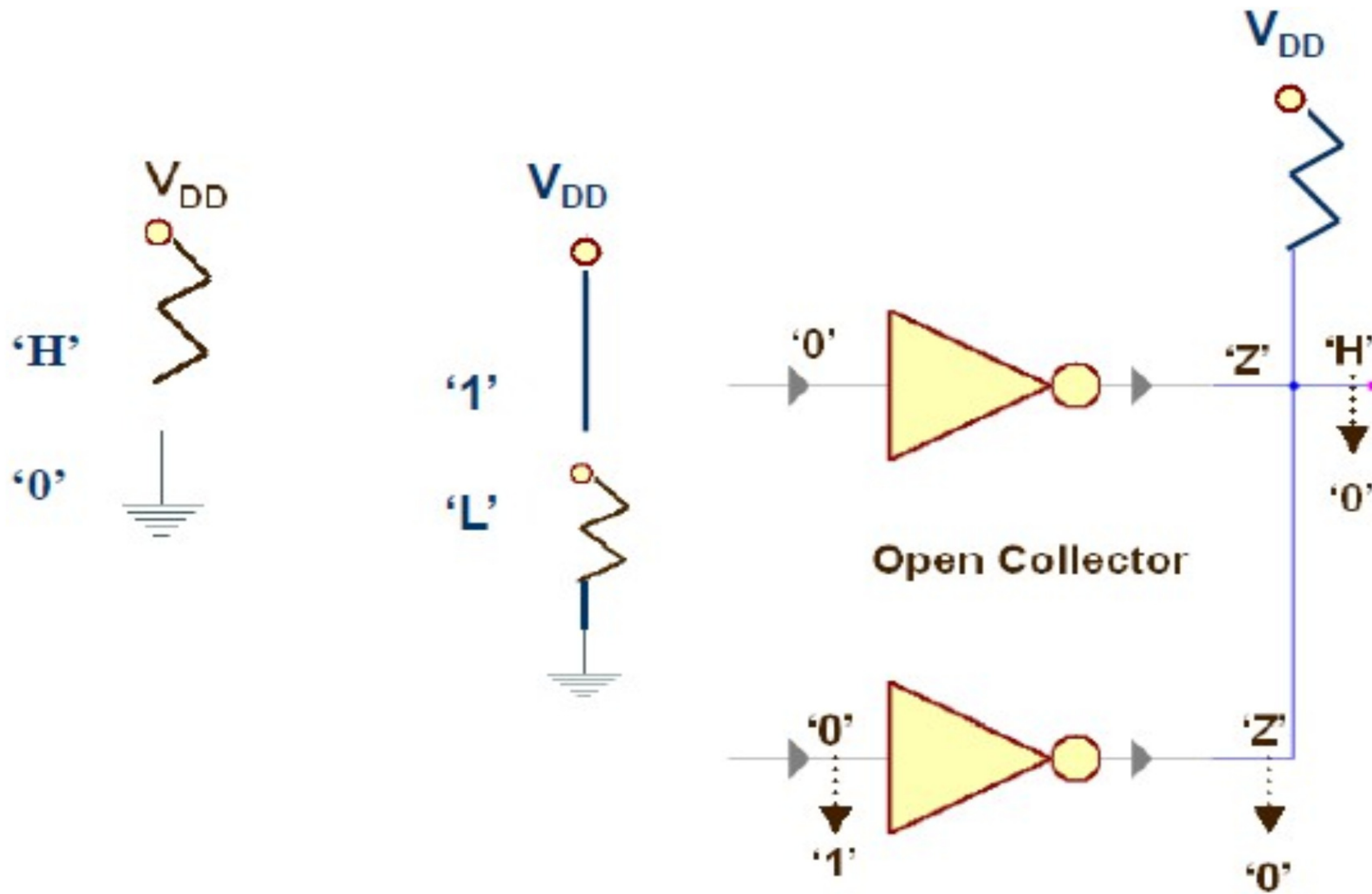
STD_LOGIC τύπος

Value	Meaning
'X'	Forcing (Strong driven) Unknown
'0'	Forcing (Strong driven) 0
'1'	Forcing (Strong driven) 1
'Z'	High Impedance
'W'	Weak (Weakly driven) Unknown
'L'	Weak (Weakly driven) 0. Models a pull down.
'H'	Weak (Weakly driven) 1. Models a pull up.
'-'	Don't Care

STD_LOGIC εξήγηση (1/3)



STD_LOGIC εξήγηση (2/3)



STD_LOGIC εξήγηση (3/3)

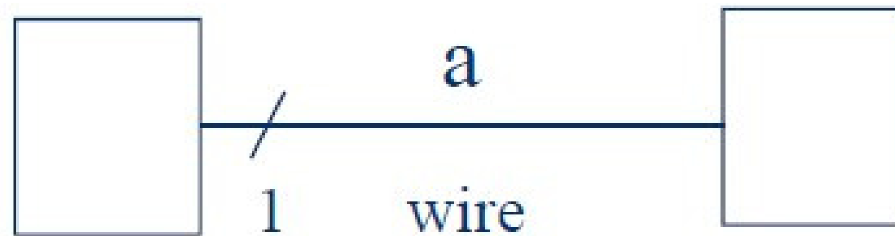
- Δεν ενδιαφέρει.
- Μπορεί να ανατεθεί σε εξόδους για περιπτώσεις μη έγκυρων εισόδων (μπορεί να παράγει σημαντική βελτίωση στην ουσιώδη χρήση μετά τη σύνθεση).
- Να χρησιμοποιείται με προσοχή. Το '1'='-' προκαλεί σφάλμα.

Τα λογικά επίπεδα της επίλυσης

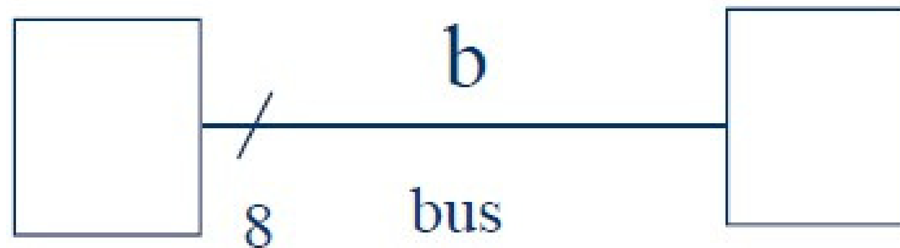
	X	0	1	Z	W	L	H	-
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	W	W	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
-	X	X	X	X	X	X	X	X

Wires και Buses

```
SIGNAL a: STD_LOGIC;
```




```
SIGNAL b: STD_LOGIC_VECTOR(7 DOWNTO 0);
```



Πρότυπα Λογικά Διανύσματα

```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL f:  STD_LOGIC_VECTOR(8 DOWNTO 0);
                    ..........
a <='1';
b <=”0000”              --Binary base assumed by default
c <=B”0000”;            --Binary base explicitly specified
d <=”0110 0111”;        --You can use '_' to increase readability
e <=X”AF67”;            --Hexadecimal base
f <=O”723”;             --Octal base
```

Αναθέσεις

- constant a: integer := 523;
- signal b: bit_vector(11 downto 0);

```
b <= "000000010010";
b <= B"000000010010";
b <= B"0000_0001_0010";
b <= X"012";
b <= O"0022";
```

Σύνδεση (concatenate)

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

a <=”0000”;
b <=”1111”;
c <= a & b;                          -- c = “00001111”

d <= '0' & “0001111”;                -- d <=“00001111”
e <= '0' & '0' & '0' & '0' & '1' & '1' & '1' & '1';
                                     --e <= “00001111”
```

Αναθέσεις (Παραδείγματα)

```
signal a: std_logic;
signal b: std_logic_vector(6 downto 0);
signal c: std_logic_vector(3 downto 0);
signal d: std_logic_vector(2 downto 0);
```

Correct	Incorrect
`a <= ‘1’;`	`a <= “01”;`
`b <= “010001”;`	`b <= ‘0’; b(1) <= ‘0’;`
`c <= (others => ‘0’);`	`c <= ‘0000’;`
`d <= (1 => ‘0’, others => ’1’);`	`d <= b & c;`
`b <= c & d;`	`b(3 downto 1) <= d(1 downto 0);`
	`b(5 downto 3) <= d;`

Δήλωση Alias

- Signal instruction: bit_vector(31 downto 0);
- Alias op1: bit_vector(3 downto 0) is instruction(23 downto 20);
- Alias op2: bit_vector(3 downto 0) is instruction(19 downto 16);
- Alias op3: bit_vector(3 downto 0) is instruction(15 downto 12);

    Op1 <= “0000”;

    Op2 <= “0001”;

    Op3 <= “0010”;

- Regs(bit2int(op3)) <= regs(bit2int(op1)) + regs(bit2int(op2));

Μετατροπή Τύπου (Παρόμοια Βάση)

Παρόμοια αλλά όχι η ίδια βάση τύπου:

- signal i: integer;
- signal r: real;

- i <= integer(r);
- r <= real(i);

Μετατροπή Τύπου (Ίδια Βάση)

- Ίδια βάση τύπου:

```
type a_type is array(0 to 4) of bit;
signal a:a_type;
signal s:bit_vector(0 to 4);
```

```
a<="00101"          -- Error, είναι το RHS ένα διάνυσμα bit_vector ή ένας  a_type?
a<=a_type  '("00101");    -- αρμοδιότητα τύπου
a<=a_type(s); -- μετατροπή τύπου
```

Μετατροπή Τύπου (Διαφορετική Βάση)

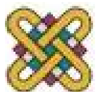
```
Function int2bits(value:integer;ret_size:integer) return bit_vector;
Function bits2int(value:bit_vector) return integer:

        signal i:integer;
        signal b:bit_vector (3 downto 0)

        i<=bits2int(b);
        b<=int2bits(i,4);
```

Τελεστές Built-In

- Λογικοί τελεστές:
  - AND OR, NAND, NOR, XOR, XNOR XNOR στην VHDL'93 μόνο.
- Τελεστές συσχέτισης:
  - =, /=, <, <=, >, >=
- Τελεστές πρόσθεσης:
  - +, -, &
- Τελεστές πολλαπλασιασμού:
  - *, /, mod, rem
- Διάφοροι άλλοι τελεστές:
  - **, abs, not

Η Συνθήκη “after”

- Χρησιμοποιείται για να εκχωρήσει σήματα με καθυστέρηση, κατά τη μοντελοποίηση της συμπεριφοράς του κυκλώματος.
  - a <= d after 5 ns; -- 5 ns wire delay
  - b <= a and c after 20 ns; -- 20 ns gate delay
- Δεν συντίθεται, αγνοείται από τα εργαλεία σύνθεσης.
- Χρήσιμο σε testbenches για τη δημιουργία κυματομορφών των σημάτων εισόδου.
  - clk <= not clk after 20 ns; -- 40 ns clock period
  - rst_n <= ‘0’, ‘1’ after 210 ns



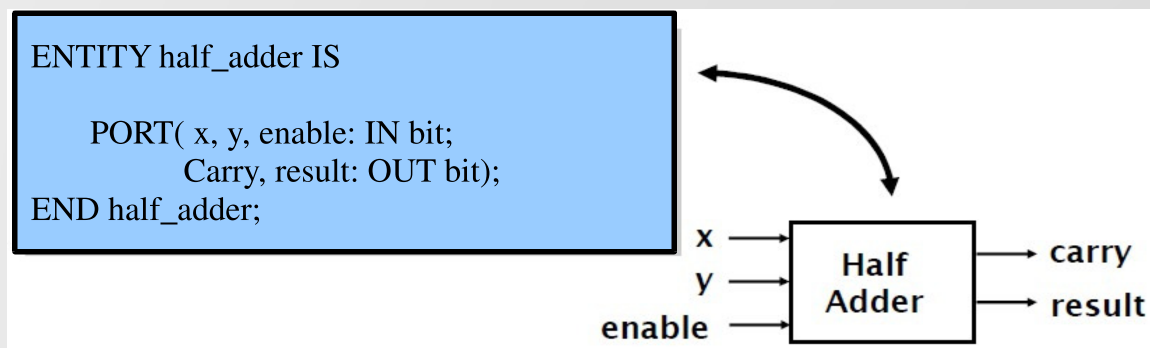
Port Καταστάσεις

Η Port κατάσταση της διεπαφής περιγράφει την κατεύθυνση μέσω της οποία τα δεδομένα περνάνε με σχέση με τα εξαρτήματα:

- **In:** Το δεδομένο περνάει από αυτή τη θύρα και μπορεί να διαβαστεί μόνο εντός της οντότητας. Μπορεί να εμφανιστεί μόνο στην αριστερή πλευρά μια ανάθεσης σήματος ή μεταβλητής.
- **Out:** Η τιμή μιας θύρας εξόδου μπορεί να ενημερωθεί μόνο εντός της οντότητας. Δεν μπορεί να διαβαστεί. Μπορεί να εμφανιστεί μόνο στη δεξιά πλευρά μιας ανάθεσης σήματος.
- **Inout:** Η τιμή μιας διπλής κατεύθυνσης θύρας μπορεί να διαβαστεί και να ενημερωθεί εντός της οντότητας. Μπορεί να εμφανιστεί και στις δύο πλευρές ανάθεσης σήματος.
- **Buffer:** U χρησιμοποιείται για ένα σήμα το οποίο είναι έξοδος μιας οντότητας. Η τιμή του σήματος μπορεί να χρησιμοποιηθεί εντός της οντότητας, που σημαίνει ότι σε μία δήλωση ανάθεσης το σήμα μπορεί να εμφανιστεί στην αριστερή και δεξιά πλευρά του τελεστή <= .

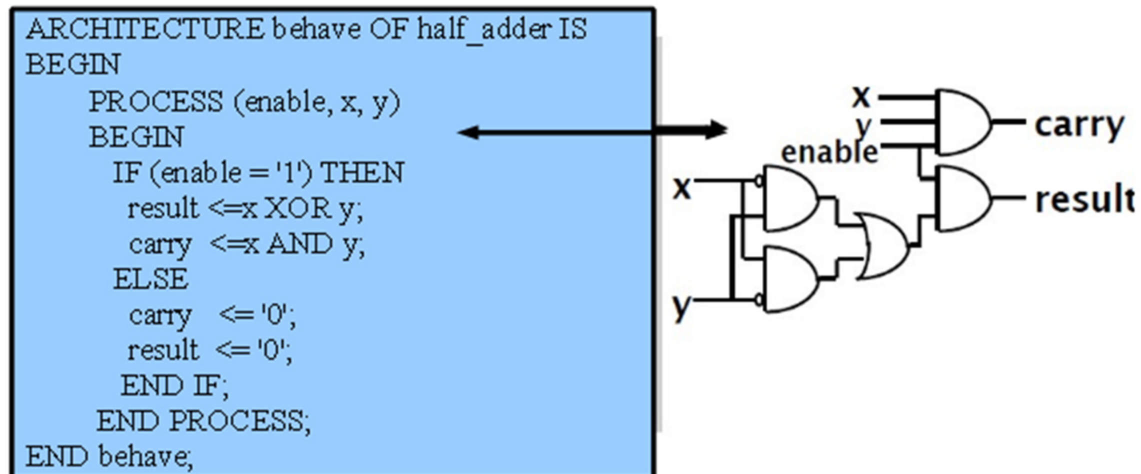
Δήλωση Οντότητας

- Μία δήλωση οντότητας περιγράφει τη διεπαφή με τα εξαρτήματα.
- Η συνθήκη PORT υποδεικνύει τις θύρες εισόδου και εξόδου.
- Μία οντότητα μπορεί να θεωρηθεί ως ένα σύμβολο για ένα εξάρτημα.

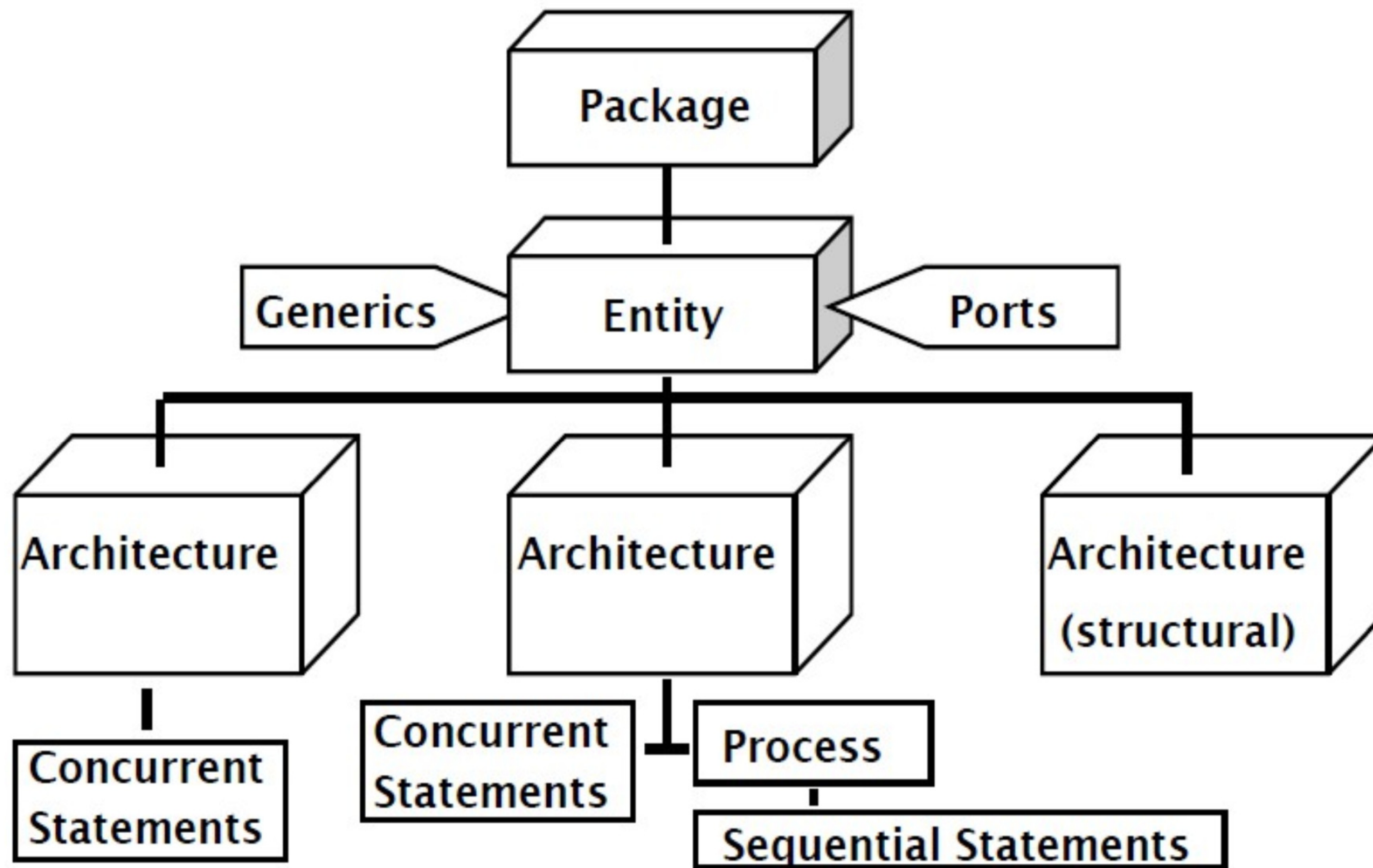


Δήλωση Αρχιτεκτονικής

- Η δήλωση της αρχιτεκτονικής περιγράφει τη λειτουργία του εξαρτήματος.
- Μπορούν να υπάρχουν πολλές αρχιτεκτονικές για μια οντότητα, αλλά μόνο μία μπορεί να είναι ενεργή κάθε φορά.
- Μία αρχιτεκτονική είναι παρόμοια με το σχηματικό ενός εξαρτήματος.



Ιεραρχία της VHDL



Τρόποι Μοντελοποίησης

- Υπάρχουν τρεις τρόποι μοντελοποίησης:
  - Συμπεριφοράς (Διαδοχικός).
  - Ροής δεδομένων.
  - Διαρθρωτική.

Συμπεριφορά (1/5)

- Η προσέγγιση συμπεριφοράς για τη μοντελοποίηση εξαρτημάτων hardware δεν επηρεάζει απαραίτητα το πως υλοποιείται το σχέδιο.
  - Αποτελεί την προσέγγιση 'Μαύρο Κουτί' για την μοντελοποίηση.
- Μοντελοποιεί με ακρίβεια το τι συμβαίνει στις εισόδους και εξόδους του μαύρου κουτιού, αλλά το τι γίνεται μέσα στο κουτί (πως αυτό δουλεύει) είναι αδιάφορο.

Συμπεριφορά (2/5)

- Η περιγραφή με συμπεριφορά συνήθως γίνεται με δύο τρόπους στην VHDL.
  - Μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει σύνθετα εξαρτήματα που θα ήταν κουραστικό να γίνει με κάποια άλλη μέθοδο.
    - Αυτό θα μπορούσε να είναι η περίπτωση για παράδειγμα, αν ήθελες να προσομοιώσεις την λειτουργία του σύνηθες σχεδίου σου που συνδέεται με κάποιο εμπορικό μέρος όπως με ένα μικροεπεξεργαστή.
  - Οι ικανότητες συμπεριφοράς της VHDL μπορεί να είναι πιο ισχυρές και είναι πιο εύχρηστες για κάποια σχέδια.
    - Οι περιγραφές με συμπεριφορά υποστηρίζονται με τις διαδικαστικές δηλώσεις.
    - Το περιεχόμενο των διαδικαστικών δηλώσεων μπορεί να είναι διαδοχικές δηλώσεις σαν αυτές που συναντώνται σε γλώσσες προγραμματισμού software.
    - Αυτές οι δηλώσεις χρησιμοποιούνται για να υπολογιστούν οι έξοδοι της διαδικασίας από τις εισόδους.
    - Οι διαδοχικές δηλώσεις είναι συχνά πιο ισχυρές, αλλά πολλές φορές δεν έχουν άμεση ανταπόκριση σε μία υλοποίηση hardware.

Συμπεριφορά (3/5)

- Το πρώτο μας παράδειγμα δήλωσης διαδικασίας είναι ασήμαντο και κανονικά δεν θα χρησιμοποιούταν σε μια δήλωση.

```
compute_xor: process (b,c)
begin
    a<=>b xor c;
end process;
```

- Η λίστα ευαισθησίας σήματος χρησιμοποιείται για να ορίσει ποια σήματα θα πρέπει να προκαλέσουν τη διαδικασία να αξιολογηθεί ξανά.
  - Όποτε κάποιο γεγονός συμβαίνει σε κάποιο σήμα της λίστας ευαισθησίας, η διαδικασία αξιολογείται ξανά.
- Σε αντίθεση με τις αναθέσεις σήματος που εμφανίζονται έξω από τη δήλωση της διαδικασίας, αυτή η ανάθεση σήματος αξιολογείται μόνο όταν γεγονότα συμβαίνουν στα σήματα της λίστας ευαισθησίας, άσχετα με το ποιο σήμα εμφανίζεται στην αριστερή πλευρά του τελεστή <= .

Συμπεριφορά (4/5)

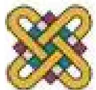
- Το παρακάτω παράδειγμα δείχνει πως μια μεταβλητή χρησιμοποιείται σε μια διαδικασία.

```
Count:process (x)
   variable cnt : integer := -1;
begin
    cnt:=cnt+1;
End process;
```

- Η δήλωση των μεταβλητών γίνεται πριν από τη λέξη begin στη δήλωση της διαδικασίας.
  – Η δήλωση αρχικών τιμών είναι προαιρετική (επηρεάζει την αρχή της προσομοίωσης).

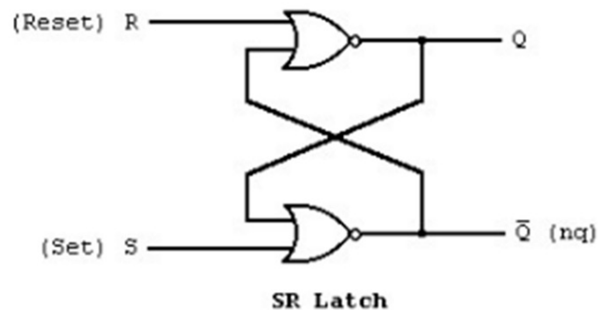
Συμπεριφορά (5/5)

- Η ανάθεση στο παράδειγμα υπολογίζει την τιμή του cnt συν ένα και αμέσως αποθηκεύει την νέα τιμή στη μεταβλητή **cnt**.
  - Άρα, ο cnt θα αυξάνεται κατά ένα κάθε φορά που η διαδικασία εκτελείται.
- Εφόσον η τιμή αρχικοποιείται στο -1, και η διαδικασία εκτελείται μία φορά πριν ξεκινήσει η προσομοίωση, ο cnt θα έχει τιμή 0 μόλις ξεκινήσει η προσομοίωση. Μετά, ο cnt θα αυξάνεται κατά ένα κάθε φορά που το σήμα x αλλάζει, διότι αυτό βρίσκεται στη λίστα ευαισθησίας.
  - Αν x είναι σήμα τύπου bit, τότε αυτή η διαδικασία θα μετράει τον αριθμό των ανυψώσεων και πτώσεων που συμβαίνουν στο σήμα x.



Ροή Δεδομένων (1/3)

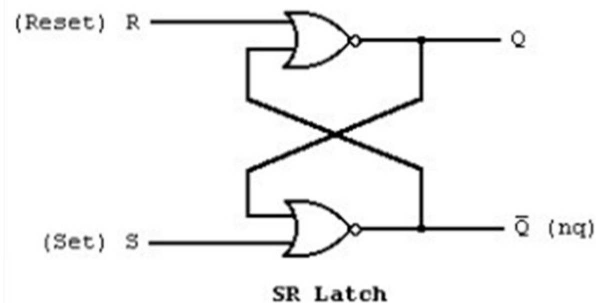
- Στην προσέγγιση ροής δεδομένων, τα κυκλώματα περιγράφονται υποδεικνύοντας το πως οι είσοδοι και έξοδοι των αρχικών εξαρτημάτων (πχ μια πύλη and) συνδέονται μεταξύ τους.
- Με άλλα λόγια περιγράφουμε πως τα σήματα (δεδομένα) ρέουν δια μέσου του κυκλώματος.
- Υποθέτουμε ότι θέλουμε να περιγράψουμε το ακόλουθο SR μάνταλο χρησιμοποιώντας VHDL όπως στο παρακάτω σχηματικό.




```
entity latch is
   port (s,r : in bit;
          q,nq : out bit);
end latch;

architecture dataflow of latch is
Begin
      q<=r nor nq;
     nq<=s nor q;
End dataflow;
```

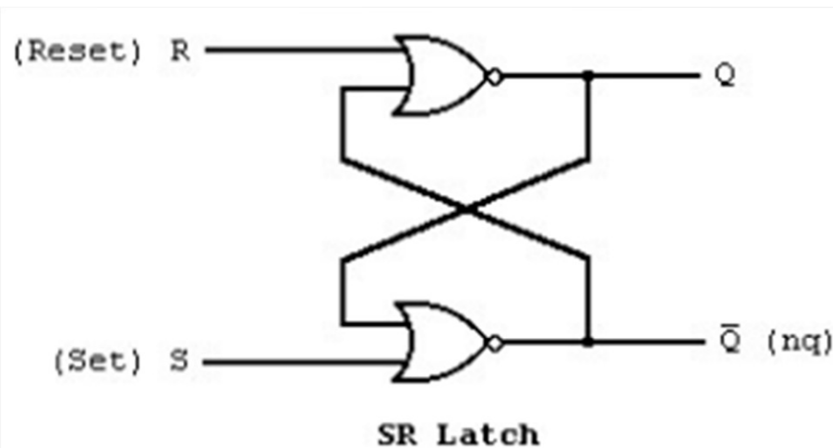
Ροή Δεδομένων (2/3)

- Προσομοίωση:
  - Αρχή : r='0',s='0',q='1',nq='0'
  - Επανάληψη 1: r='1',s='0',q='1',nq='0', The value '0' is scheduled on q.
  - Επανάληψη 2: r='1',s='0',q='0',nq='0', The value '1' is scheduled on nq.
  - Επανάληψη 3: r='1',s='0',q='0',nq='1', No new events are scheduled.
  - Επανάληψη 4: r='0',s='0',q='0',nq='1', No new events are scheduled.




```
entity latch is
    port (s,r : in bit;
            q,nq : out bit);
End latch;

Architecture dataflow of latch is
Begin
    q<=r nor nq;
    nq<=s nor q;
End dataflow;
```

Ροή Δεδομένων (3/3)

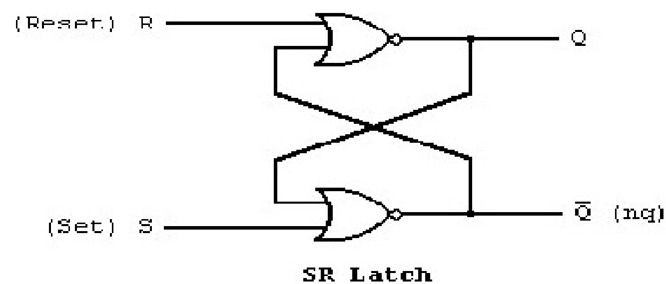


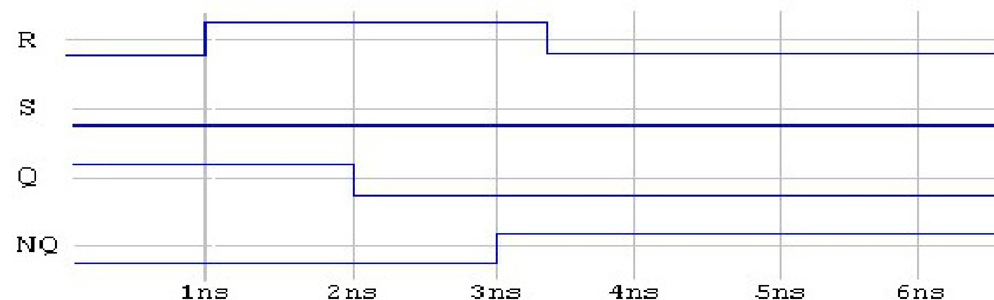

```
entity latch is
    port (s,r : in bit;
          q,nq : out bit);
End latch;

Architecture dataflow of latch is
Begin
    q<=r nor nq;
    nq<=s nor q;
End dataflow;
```

Το Μοντέλο Καθυστέρησης (1/6)

- Θα συζητήσουμε δύο μοντέλα καθυστέρησης που χρησιμοποιούνται στην VHDL.
- Το μοντέλο αδρανειακής καθυστέρησης ορίζεται δηλώνοντας μία συνθήκη **after** στη δήλωση ανάθεσης των σημάτων.

```
q<=r nor nq after 1ns;
nq<=s nor q after 1ns;
```





Το Μοντέλο Καθυστέρησης (2/6)

- Κατά τη διάρκεια της προσομοίωσης, υποθέτοντας ότι το σήμα r αλλάζει και θα προκαλέσει το σήμα q να αλλάξει, από το να προγραμματίσεις το γεγονός στο q να συμβεί κατά τη διάρκεια του επόμενου γύρου, αυτό είναι προγραμματισμένο να συμβεί 1ns από τον τρέχοντα χρόνο.

- Παρατήρησε ότι το γεγονός δεν συνέβη στο q έως 1ns μετά την αλλαγή στο r. Όμοια η αλλαγή στο nq δεν συνέβη έως 1ns μετά την αλλαγή στο q. Άρα, το "after 1ns" μοντελοποιεί μαι εσωτερική καθυστέρηση της nor πύλη.

Το Μοντέλο Καθυστέρησης (3/6)

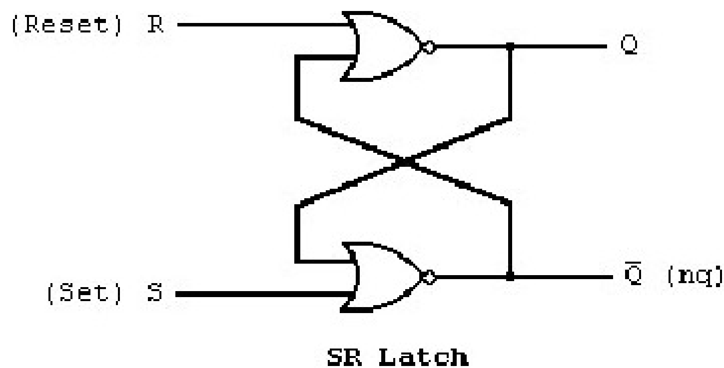
- Το μοντέλο αδρανειακής καθυστέρησης ορίζεται προσθέτοντας μία **after** συνθήκη στη δήλωση ανάθεσης των σημάτων.

```
q<=r nor nq after 1ns;
nq<=s nor q after 1ns;
```

- Όταν ένα εξάρτημα έχει κάποια αδρανειακή καθυστέρηση και μία έξοδος αλλάζει για χρόνο μικρότερο αυτής της καθυστέρησης, τότε δεν συμβαίνει καμιά αλλαγή στην έξοδο.

Το Μοντέλο Καθυστέρησης (4/6)

- Το ακόλουθο χρονικό διάγραμμα θα παραγόταν κατά τη χρήση αδρανειακής καθυστέρησης, αν ο παλμός '1' στο σήμα r ήταν μικρότερος (οτιδήποτε μικρότερο από 1ns) από το προηγούμενο παράδειγμα.



Το Μοντέλο Καθυστέρησης (5/6)

- Παρόλο που τις περισσότερες φορές η αδρανειακή καθυστέρηση είναι επιθυμητή, μερικές φορές όλες οι αλλαγές στην είσοδο πρέπει να επηρεάζουν την έξοδο.
  - *Για παράδειγμα, κάποιο bus αντιμετωπίζει μια χρονική καθυστέρηση, αλλά δεν θα απορροφήσει μικρούς παλμούς όπως γίνεται με το μοντέλο αδρανειακής καθυστέρησης.*

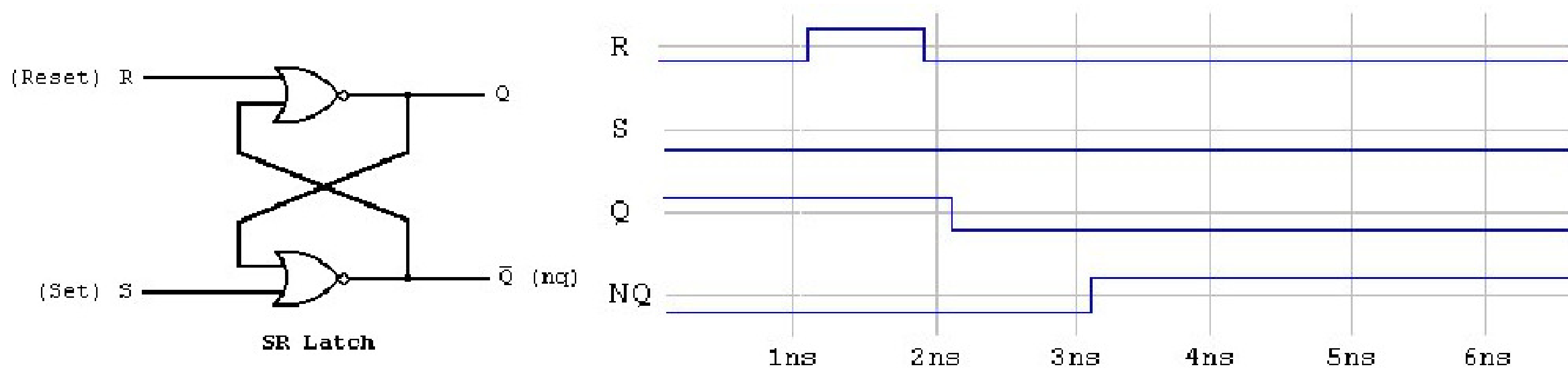
- Ως εκ τούτου, η VHDL παρέχει **το μοντέλο καθυστέρησης μεταφοράς**.
- Το μοντέλο αυτό απλά καθυστερεί την αλλαγή στην έξοδο μέχρι το χρόνο που καθορίζεται στη συνθήκη after.
- Μπορείς να επιλέξεις να χρησιμοποιήσεις το μοντέλο καθυστέρησης μεταφοράς αντί του μοντέλου αδρανειακής καθυστέρησης προσθέτοντας τη λέξη **transport** στη δήλωση ανάθεσης των σημάτων.

Το Μοντέλο Καθυστέρησης (6/6)

- Το παράδειγμα του SR μανδάλου μπορεί να διαμορφωθεί ώστε να χρησιμοποιεί το μοντέλο καθυστέρησης μεταφοράς αντικαθιστώντας την ανάθεση των σημάτων με τις παρακάτω δύο δηλώσεις:

```
q<=transport r nor nq after 1ns;
nq<=transport s nor q after 1ns;
```

Αν χρησιμοποιούταν το μοντέλο καθυστέρησης μεταφοράς, το αποτέλεσμα της ίδιας προσομοίωσης που φαίνεται στο τελευταίο διάγραμμα θα κατέληγε στο παρακάτω χρονικό διάγραμμα.



Διαρθρωτική (1/3)

- Τυπικά αναλύεται σε πολλά τμήματα-blocks:
  - Κάθε παρτίδα/τμήμα ενός σχεδίου VHDL θεωρείται ως ένα block.
  - Ονομάζεται οντότητα.
  - Η οντότητα περιγράφει τη διεπαφή σε αυτό το block και ένα ξεχωριστό μέρος που σχετίζεται με την οντότητα περιγράφει πως λειτουργεί το block.
  - Η περιγραφή της διεπαφής είναι σαν μια περιγραφή για pin σε ένα βιβλίο δεδομένων, ορίζοντας τις εισόδους και εξόδους στο block.
- Αυτά τα blocks συνδέονται μεταξύ τους για να σχηματίσουν ένα ολοκληρωμένο σχέδιο.
- Ένα σχέδιο VHDL μπορεί να περιγράφεται εξολοκλήρου σε ένα μόνο block, ή μπορεί να αναλύεται σε πολλά blocks.

Διαρθρωτική (2/3)

```
entity latch is
   port (s,r: in bit;
            q,nq: out bit);
end latch;
```

- Οι γραμμές στο ενδιάμεσο, που ονομάζονται port συνθήκες, περιγράφουν την διεπαφή με το σχέδιο.
- Μία port συνθήκη περιέχει μία λίστα από ορισμούς διεπαφών.
  - Κάθε ορισμός διεπαφής ορίζει ένα ή περισσότερα σήματα που είναι είσοδοι ή έξοδοι στο σχέδιο.
- Η κατάσταση-ανάθεση ορίζει αν αυτή είναι είσοδος (in), έξοδος (out), η και τα δύο (inout).
- Ο τύπος καθορίζει τι τύπου τιμές μπορεί να παίρνει το σήμα.
  - Τα σήματα s και r είναι κατάστασης in (είσοδοι) και τύπου bit.
  - Τα σήματα q και nq ορίζονται να είναι κατάστασης out (έξοδοι) και τύπου bit (δυαδικά).

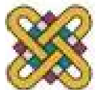
Διαρθρωτική (3/3)

- Το δεύτερο μέρος της περιγραφής του σχεδίου του μανδάλου είναι μια περιγραφή του πως λειτουργεί το σχέδιο.
- Αυτό καθορίζεται από τη δήλωση της αρχιτεκτονικής.
  - Το παρακάτω είναι ένα παράδειγμα της δήλωσης της αρχιτεκτονικής για την οντότητα του μανδάλου.

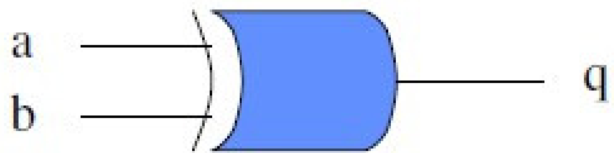
```
Architecture dataflow of latch is
   Signal q0 : bit := '0';
   Signal nq0 :bit := '1';
Begin
   q0<=r nor nq0;
   nq0<=s nor q0;

nq<=nq0;
q<=q0;
end dataflow;
```

Διαδοχικές vs Ταυτόχρονες Δηλώσεις

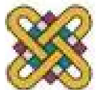
- Η VHDL παρέχει δύο διαφορετικούς τύπους εκτέλεσης: διαδοχική και ταυτόχρονη.
- Οι διαφορετικοί τύποι εκτέλεσης είναι χρήσιμοι για μοντελοποίηση πραγματικού hardware.
  - Υποστηρίζει πολλά επίπεδα αφαίρεσης.
- Οι διαδοχικές καταστάσεις βλέπουν το hardware από την προσέγγιση του προγραμματιστή.
- Οι ταυτόχρονες δηλώσεις είναι ανεξάρτητες της τάξεως και ασύγχρονες.



Διαδοχικός Τρόπος




```
process(a,b)
begin
        if(a/=b) then
                q <= '1';
        else
                q<='0';
        end if;
end process;
```



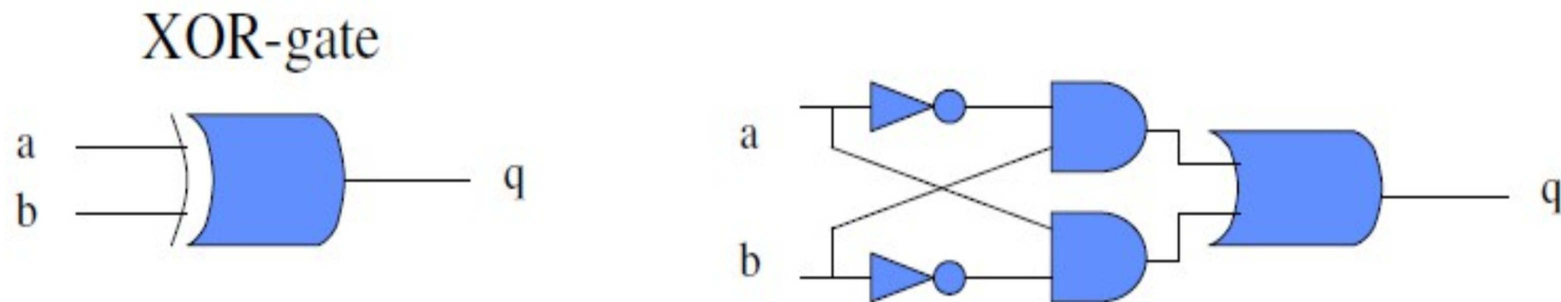
Τρόπος Ροής Δεδομένων

```
q <= a xor b;
```

Or in behavioral data flow style:

```
q <= ‘1’ when a/=b else ‘0’;
```

Διαρθρωτικός Τρόπος

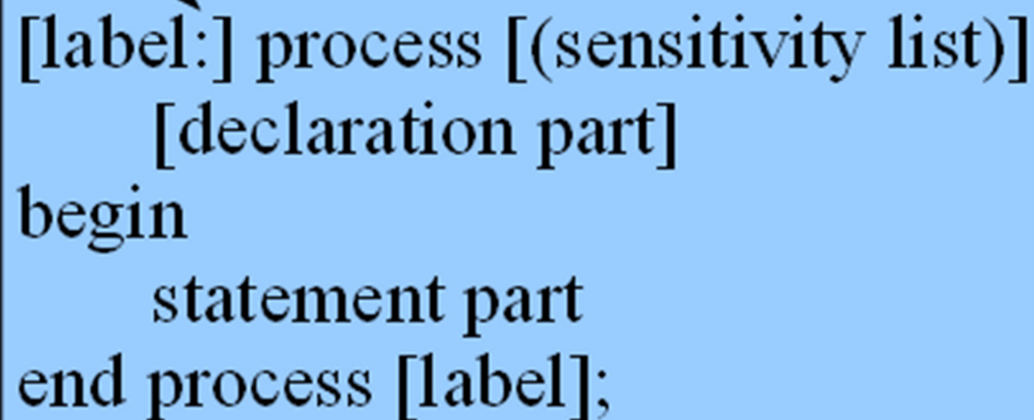



```
u1: inverter port map (a,ai);

u2: inverter port map (b,bi);

u3: and_gate port map (ai,b,t3);

u4: and_gate port map (bi,a,t4);

u5: or_gate port map (t3,t4,q);
```

Διαδικασία: Σύνταξη Διαδοχικού τρόπου (1/2)

Οι αναθέσεις-εργασίες εκτελούνται διαδοχικά μέσα στη διαδικασία..

OPTIONAL



Διαδικασία: Σύνταξη Διαδοχικού τρόπου (2/2)

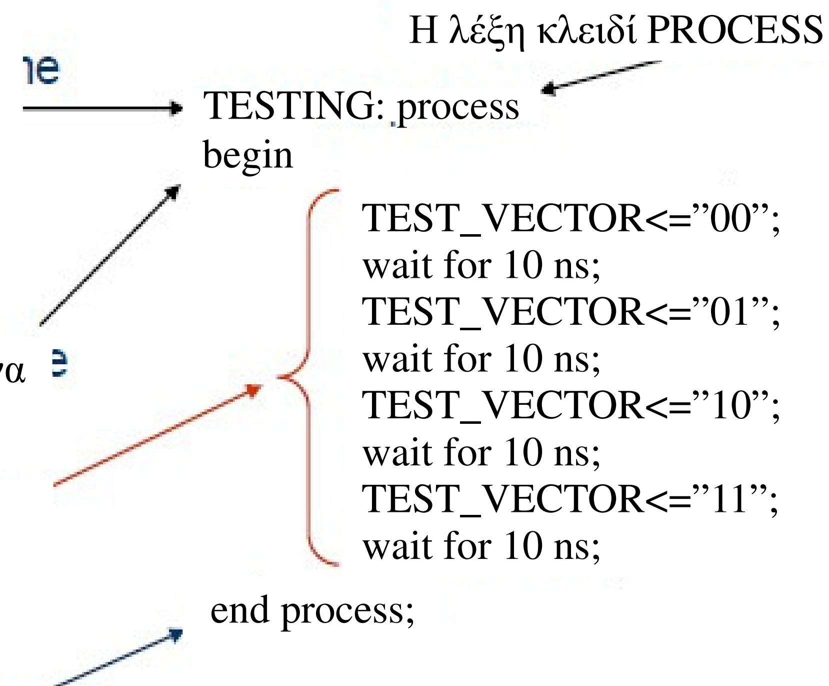
- Περιέχει διαδοχικές δηλώσεις για να εκτελεστούν κάθε φορά που η διαδικασία εκτελείται.

- Ανάλογο με τους συμβατικούς τρόπους προγραμματισμού.

Τι είναι μια διαδικασία

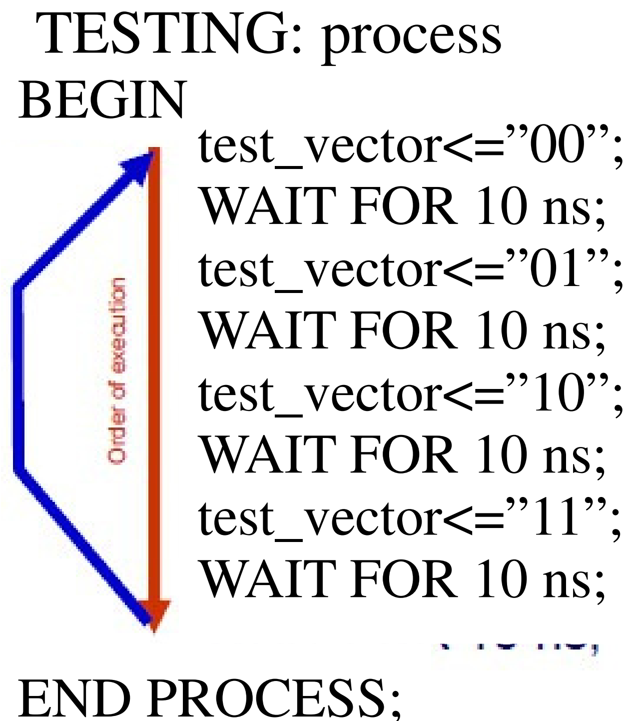
- Μία διεργασία είναι μια ακολουθία εντολών αναφερόμενων ως διαδοχικές δηλώσεις

- Σε μια διεργασία μπορεί να δωθεί μοναδικό όνομα χρησιμοποιώντας μια προαιρετική ετικέτα-Label
- Ακολουθείται από την λέξη κλειδί PROCESS
- Η λέξη κλειδί BEGIN χρησιμοποιείται για να υποδείξει την αρχή της διεργασίας
- Όλες οι δηλώσεις μέσα σε μια διεργασία εκτελούνται διαδοχικά. Συνεπώς, η σειρά των δηλώσεων είναι σημαντική.
- Μία διεργασία πρέπει να τελειώνει με τις λέξεις κλειδιά END PROCESS

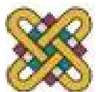


Εκτέλεση Δηλώσεων

- Η εκτέλεση των δηλώσεων συνεχίζει διαδοχικά μέχρι την τελευταία δήλωση στη διεργασία
- Μετά την εκτέλεση της τελευταίας δήλωσης, ο έλεγχος περνάει ξανά στην αρχή της διεργασίας



Program control is passed to the first statement after BEGIN

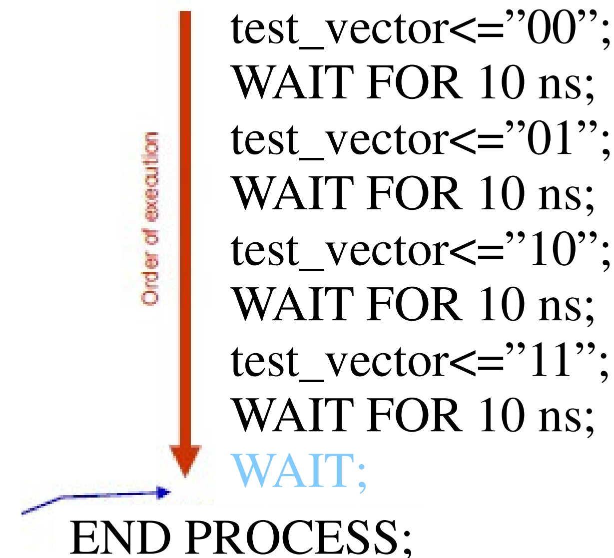


Η Δήλωση WAIT

- Η τελευταία δήλωση στην διεργασία είναι η **WAIT** αντί της WAIT FOR10ns
- Αυτό θα προκαλέσει τη διαδικασία να ανασταλεί για απεριόριστο χρόνο όσο η WAIT εκτελείται.
- Αυτή η μορφή της WAITμπορεί να χρησιμοποιηθεί σε μια διεργασία που περιλαμβάνεται σε ένα testbench όταν όλες οι πιθανοί συνδυασμοί εισόδων έχουν δοκιμαστεί ή όταν ένα μη περιοδικό σήμα πρέπει να δημιουργηθεί.

Η εκτέλεση του προγράμματος σταματά εδώ.

Testing: PROCESS
BEGIN



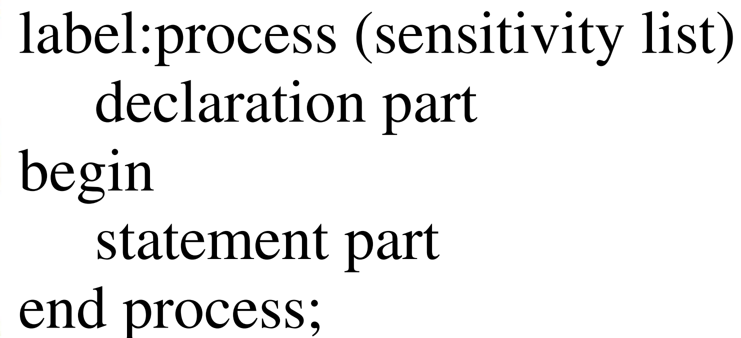
WAIT FOR vs. WAIT

WAIT FOR: Η κυματομορφή θα συνεχίσει να επαναλαμβάνεται απεριορίστου

WAIT: Η κυματομορφή θα συνεχίσει την κατάσταση της μετά την τελευταία wait εντολή.

Λίστα Ευαισθησίας

- Λίστα σημάτων στα οποία η διεργασία εμφανίζει ευαισθησία
- Όποτε υπάρχει ένα γεγονός σε οποιοδήποτε σήμα της λίστας, η διεργασία ανάπτει
- Κάθε φορά που η διεργασία ανάπτει, τρέχει-εκτελείται ολόκληρη
- **Οι δηλώσεις WAIT ΔΕΝ ΕΠΙΤΡΕΠΟΝΤΑΙ σε διεργασίες που έχουν λίστα ευαισθησίας.**



Καταλληλότητα της Διαδικασίας

- Οι διεργασίες περιγράφουν διαδοχική συμπεριφορά.
- Οι διεργασίες στην VHDL είναι πολύ ισχυρές δηλώσεις
  - Επιτρέπουν να οριστεί μια αυθαίρετη συμπεριφορά που μπορεί να είναι δύσκολο να παρουσιαστεί σε ένα πραγματικό κύκλωμα.
  - Όχι όλες οι διεργασίες δεν είναι δυνατόν να συντεθούν
- Χρήση των διεργασιών με προσοχή σε κώδικα που πρόκειται να συντεθεί
- Χρήση των διεργασιών ελεύθερα σε testbenches.

Component Equivalent of a Process

```
Priority:PROCESS (clk)
BEGIN
        IF w(3) ='1' THEN
                y<="11";
        ELSEIF w(2)='1' THEN
                y<="10";
        ELSEIF w(1)= c THEN
                y<=a and b;
        ELSE
                z<="00" ;
        END IF;
END PROCESS;
```

Όλα τα σήματα που εμφανίζονται στην αριστερή πλευρά της δήλωσης ανάθεσης σημάτων (<=) αποτελούν εξόδους π.χ y,z.

Όλα τα σήματα που εμφανίζονται στην δεξιά πλευρά της δήλωσης ανάθεσης σημάτων (<=) ή σε λογικές εκφράσεις αποτελούν εισόδους π.χ w, a, b, c.

Όλα τα σήματα που εμφανίζονται στην λίστα ευαισθησίας είναι είσοδοι π.χ ckl.

Προσέξτε ότι δεν είναι απαραίτητο όλες οι είσοδοι να βρίσκονται στη λίστα ευαισθησίας.

Διαδοχικές Δηλώσεις

Αναθέσεις {σήματα, μεταβλητές}

- Έλεγχος Ροής
  - if <συνθήκη> then <δηλώσεις>
    [elsif <συνθήκη> then <δηλώσεις>]
    else <δηλώσεις>
    end if;
  - for <εύρος> loop <δηλώσεις> end loop;
  - while <συνθήκη> loop <δηλώσεις> end loop;
  - case <συνθήκη> is
    when <τιμή> => <δηλώσεις>;
    when <τιμή> => <δηλώσεις>;
    when <τιμή> => <δηλώσεις>;
- Wait on <σήμα> until <έκφραση> for <χρόνος>;

IF Statement - Syntax

```
If boolean expression then
        Statements
elseif boolean expression then
        Statements
else boolean expression then
       Statements
end if;
```

- Τα else και elseif είναι προαιρετικά.

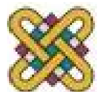
IF Statement: Example

```
SELECTOR: process
Begin
        WAIT UNTIL Clock'EVENT AND Clock='1';
        IF Sel = “00” THEN
                f <=X1;
        ELSEIF Sel=”10” THEN
                f <=x2;
        ELSE
                f <=x3;
        END IF;
End process;
```

Αντικείμενα Δεδομένων

Υπάρχουν τρεις τρόποι για αντικείμενα-δεδομένα:

- **Σήματα:**
  - Μπορεί να θεωρηθεί σαν τα καλώδια στο σχηματικό.
  - Μπορεί να έχει τρέχουσα τιμή αλλά και μελλοντικές.
- **Μεταβλητές και Σταθερές:**
  - Χρησιμοποιούνται για να μοντελοποιήσουν τη συμπεριφορά κυκλωμάτων.
  - Χρησιμοποιούνται σε διεργασίες, διαδικασίες και συναρτήσεις.



Constant Declaration

Μια σταθερά έχει μοναδική τιμή ενός προκαθορισμένου τύπου.

- Η τιμή μιας σταθεράς δεν μπορεί να αλλάξει στη διάρκεια της προσομοίωσης.
- Οι σταθερές που ορίζονται στην αρχή μιας αρχιτεκτονικής μπορεί να χρησιμοποιηθεί οπουδήποτε στην αρχιτεκτονική.
- Οι σταθερές που ορίζονται σε μία διεργασία μπορούν να χρησιμοποιηθούν μόνο μέσα σε συγκεκριμένες διεργασίες.

```
CONSTANT constant_name : type_name [:= value];

CONSTANT rise_fall_time : TIME := 2 ns;
CONSTANT data_bus : INTEGER := 16;
```

Variable Declaration

- Οι μεταβλητές χρησιμοποιούνται για τοπική αποθήκευση τιμών.
- Οι μεταβλητές γενικά δεν είναι διαθέσιμες σε πολλά εξαρτήματα ή διεργασίες.
- Όλες οι αναθέσεις των μεταβλητών συμβαίνουν αμέσως.
- Οι μεταβλητές είναι πιο εύχρηστες από τα σήματα για την αποθήκευση προσωρινών δεδομένων.

```
VARIABLE variable_name : type_name [:=value];

VARIABLE opcode : BIT_VECTOR(3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

Ορισμός Σημάτων

- Τα σήματα χρησιμοποιούνται για την επικοινωνία μεταξύ των εξαρτημάτων.
- Τα σήματα δηλώνονται έξω από τη διεργασία.
- Τα σήματα μπορούν να θεωρηθούν ως πραγματικά, φυσικά σήματα.
- Κάποια καθυστέρηση πρέπει να συμβεί κατά την εκχώρηση των σημάτων.

```
SIGNAL signal_name : type_name  [:=value];

SIGNAL brdy : BIT;
SIGNAL output : INTEGER :=  2;
```

Εκχώρηση Σημάτων

- Μια σημαντική διαφορά μεταξύ των μεταβλητών και των σημάτων είναι η εκχώρηση καθυστέρησης.

```
ARCHITECTURE signals OF test IS
	SIGNAL a, b, c, out_1, out_2: BIT;
BEGIN
	Out_1 <= a NAND b;
	Out_2 <= out_1 XOR c;
END signals;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

Εκχώρηση Μεταβλητών

```
ARCHITECTURE variables OF test IS
BEGIN
    PROCESS (a, b, c)
    VARIABLE  a,b,c,out_3,out_4: BIT;
    BEGIN
        out_3 := a NAND b;
        out_4 := out_3 XOR c;
    END PROCESS;
END example;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	1

Σύνταξη των IF – vs CASE – Δηλώσεων

```
If (a='1') then
        q <= '1';
elsif (b='1') then
        q<='1';
else
        q <='0';
end if;
```

```
case (a&b) is
when “00” =>
  q <= '0';
      when others=>
           q <= '1';
end case;
```

Δήλωση Παραγωγής-Generate

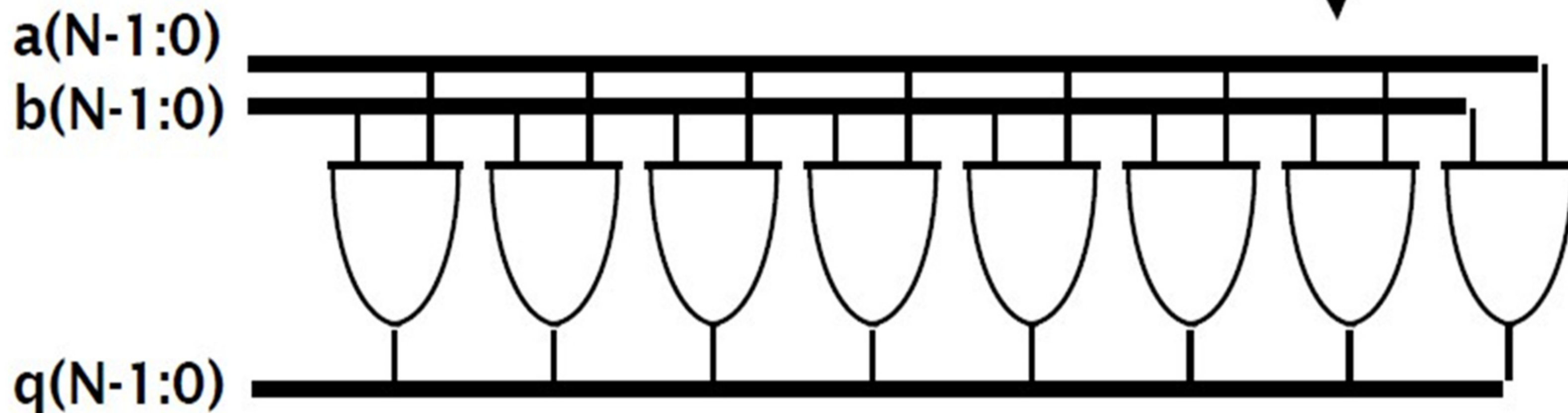
- Διαρθρωτικά for-loops: Η δήλωση Παραγωγής:
  - Μερικές δομές στο ψηφιακό hardware είναι επαναλαμβανόμενες. (RAM, ROM, registers, adders, multipliers, …)
  - Η VHDL παρέχει τη δήλωση GENERATE ώστε αυτόματα να δημιουργηθεί κανονικό hardware.
  - Οποιαδήποτε VHDL ταυτόχρονη δήλωση μπορεί να περιέχεται σε μια GENERATE δήλωση, περιλαμβάνοντας και άλλη GENERATE δήλωση.

```
Label:  FOR identifier IN range GENERATE
              BEGIN
                {Concurrent Statements}
              END GENERATE;
```

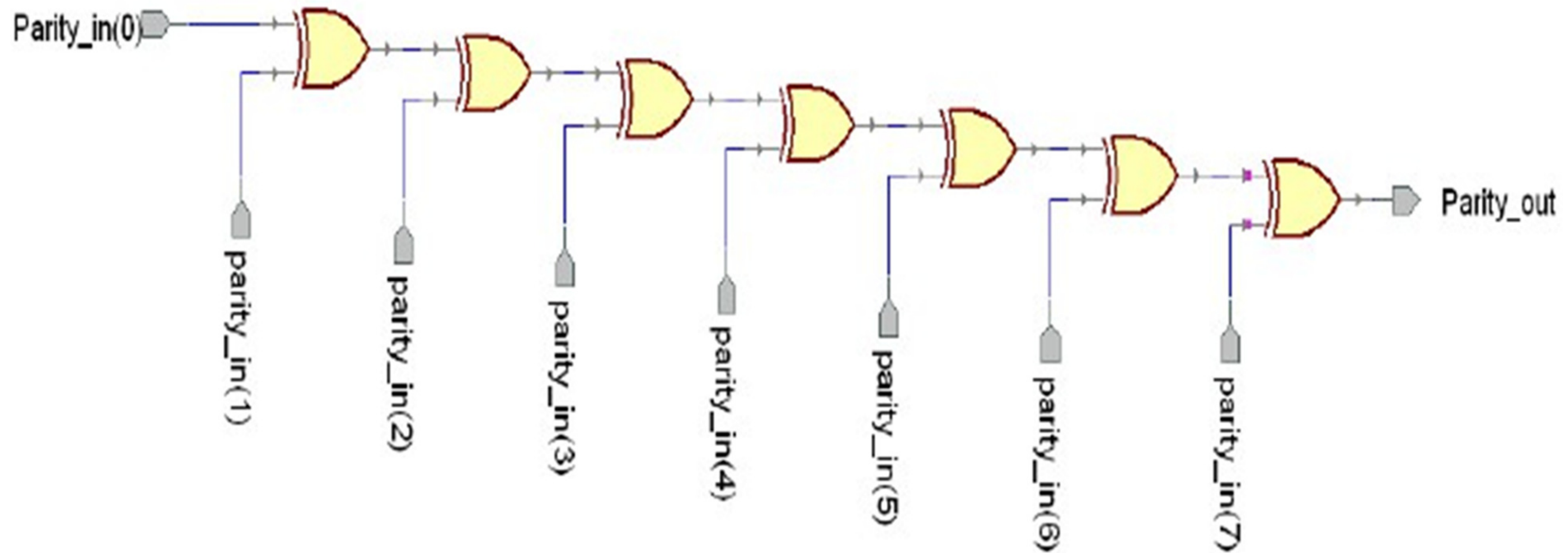
- Όλα τα αντικείμενα που δημιουργούνται είναι παρόμοια.
- Η παράμετρος GENERATE πρέπει να είναι διακριτή και έξω από τη δήλωση GENERATE.

Παράδειγμα: συστοιχία από πύλες AND

```
USE work.my_gates.all;
ARCHITECTURE structural OF and_bit_vector IS
BEGIN
    G1 : FOR i IN N-1 DOWNTO 0 GENERATE
      and_array : and_gate
        GENERIC MAP (2 ns, 3 ns)
        PORT MAP (i1=>a(i),i2=>b(i),q=>q(i));
    END GENERATE G1;
END structural;
```



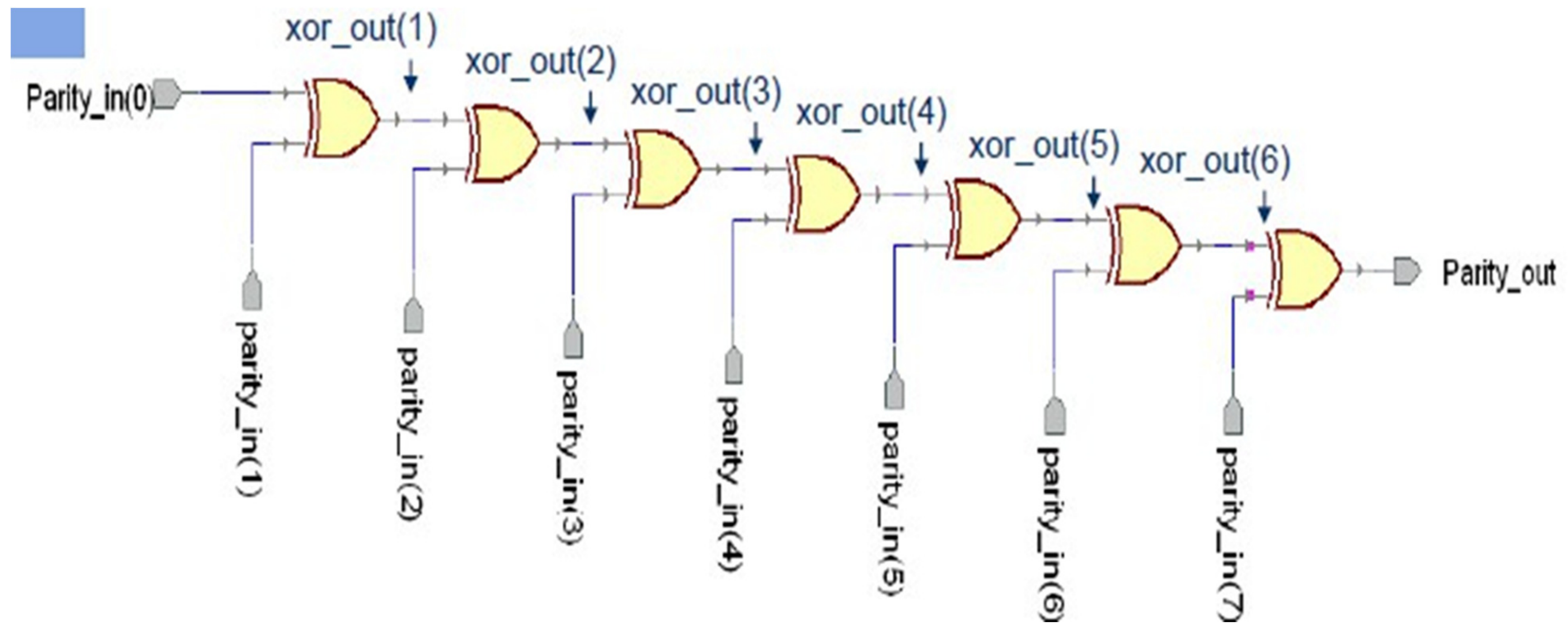
Παράδειγμα Ισοτιμίας



Ισοτιμία: Δήλωση Οντοτήτων

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY parity IS
      PORT(
            parity_in   :IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            parity_out :OUT STD_LOGIC;
         );
END parity;
```

Ισοτιμία: Block Διάγραμμα



Ισοτιμία: Αρχιτεκτονική

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: std_logic_vector (6 downto 1);

BEGIN
    xor_out(1) <= parity_in(0) XOR parity_in(1);
    xor_out(2) <= xor_out(1) XOR parity_in(2);
    xor_out(3) <= xor_out(2) XOR parity_in(3);
    xor_out(4) <= xor_out(3) XOR parity_in(4);
    xor_out(5) <= xor_out(4) XOR parity_in(5);
    xor_out(6) <= xor_out(5) XOR parity_in(6);
    parity_out  <= xor_out(6) XOR parity_in(7);
END parity_dataflow;
```

Ισοτιμία: Block Διάγραμμα 2

xor_out(0)
xor_out(1)
xor_out(2)
xor_out(3)
xor_out(4)
xor_out(5)
xor_out(6)
xor_out(7)
Parity_in(0)
parity_in(1)
parity_in(2)
parity_in(3)
parity_in(4)
parity_in(5)
parity_in(6)
parity_in(7)
Parity_out

Ισοτιμία: Αρχιτεκτονική 2

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR(7 downto 0);
BEGIN
      xor_out(0) <= parity_in(0);
      xor_out(1) <= xor_out(0) XOR parity_in(1);
      xor_out(2) <= xor_out(1) XOR parity_in(2);
      xor_out(3) <= xor_out(2) XOR parity_in(3);
      xor_out(4) <= xor_out(3) XOR parity_in(4);
      xor_out(5) <= xor_out(4) XOR parity_in(5);
      xor_out(6) <= xor_out(5) XOR parity_in(6);
      xor_out(7) <= xor_out(6) XOR parity_in(7);
      Parity_out <= xor_out(7);
END parity_dataflow;
```

Ισοτιμία: Αρχιτεκτονική 3

```
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    xor_out(0) <= parity_in(0);

    G2:FOR i IN 1 TO 7 GENERATE
            xor_out(i) <= xor_out(i-1) XOR parity_in(i);
    end generate G2;

    parity_out <=xor_out(7);
END parity_dataflow;
```

Σύνταξη των FOR – vs WHILE – Δηλώσεων

```
for i in 0 to 9 loop
        q(i) <= a(i) and b(i);
end loop
```

Το For θεωρείται ως ένα συνδυαστικό κύκλωμα από μερικά εργαλεία σύνθεσης

Άρα, δεν μπορεί να έχει μία wait δήλωση για την σύνθεση του

```
I:=0;
while (i<9) loop
        q <= a(i) and b(i);
        WAIT ON clk UNTIL clk='1';
end loop
```

Η While θεωρείται να είναι FSM από εργαλεία σύνθεσης για την σύνθεση της

Δήλωση - WAIT – Σύνταξη

- **Η δήλωση wait προκαλεί την παύση μιας δήλωσης διεργασίας ή μιας διαδικασίας**
- **wait [περιορισμός ευαισθησίας] [συνθήκη] [timeout_clause];**
  - Sensitivity_clause ::= on signal_name
    ```
    wait on CLOCK;
    ```
  - Condition_clause ::= until boolean_expression
    ```
    wait until Clock = '1';
    ```
  - Timeout_clause ::= for time_expression
    ```
    wait for 150 ns;
    ```

Sensitivity-lists vs Wait-on - statement

```
Summation:
  PROCESS( A, B, Cin)
  BEGIN
      Sum <= A xor B xor Cin;
END PROCESS Summation;
```

=

```
Summation: PROCESS
   BEGIN
      Sum <= A xor B xor Cin;
         WAIT ON A, B, Cin;
END PROCESS Summation;
```

Αν τοποθετήσεις μια λίστα ευαισθησίας σε μια διεργασία, δεν μπορείς να έχεις δήλωση wait

Αν τοποθετήσεις μια δήλωση wait σε μια διεργασία, δεν μπορείς να έχεις λίστα ευαισθησίας

Ταυτόχρονες Ισοδύναμες Διεργασίες

- **Όλες οι ταυτόχρονες δηλώσεις αντιστοιχούν σε μία ισοδύναμη διεργασία**

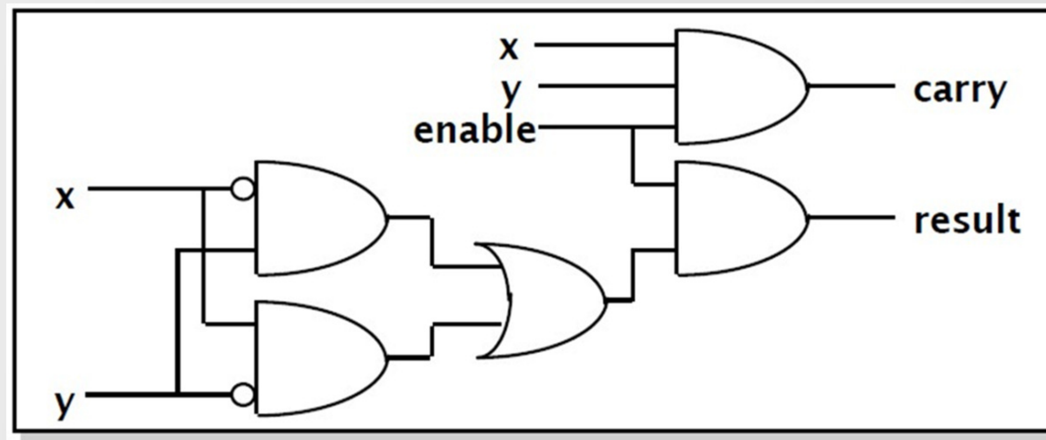
```
U0: q <= a xor b after 5 ns;
```

Αποτελεί συντομογραφία για το:

```
U0: process
begin
        q <= a xor b after 5 ns;
        wait on a, b;
end process;
```

Διαρθρωτικός Τρόπος

- Τα κυκλώματα μπορούν να περιγραφούν ως netlist.
- Τα εξαρτήματα μπορούν να κανονικοπουηθούν.
- Μεγάλα, κανονικά κυκλώματα μπορούν να δημιουργηθούν.

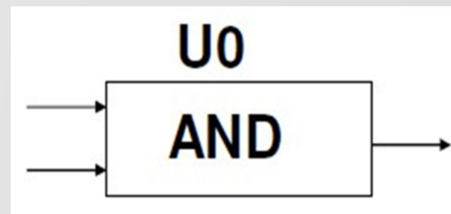


Διαρθρωτικές Δηλώσεις (1/5)

- Η διαρθρωτική VHDL περιγράφει τη διευθέτηση και τη διασύνδεση των εξαρτημάτων.
  - ✓ Οι περιγραφές συμπεριφοράς, από την άλλη πλευρά, ορίζουν τις ανταποκρίσεις στα σήματα.
- Οι διαρθρωτικές δηλώσεις μπορούν να δείξουν μια πιο συμπαγή σχέση μεταξύ κώδικα και φυσικού hardware.
- Οι διαρθρωτικές περιγραφές δείχνουν τις διασυνδέσεις σε οποιοδήποτε επίπεδο αφαίρεσης.

Διαρθρωτικές Δηλώσεις (2/5)

- The component instantiation είναι ένα από τα κατασκευασμένα blocks των διαρθρωτικών περιγραφών.
- The component instantiation process απαιτεί δηλώσεις των εξαρτημάτων και and component instantiation statements.
- Component instantiation ορίζει τη διεπαφή των εξαρτημάτων που χρησιμοποιείται στην αρχιτεκτονική.
- At instantiation, μόνο η διεπαφή είναι ορατή.
  - ✓ Τα εσωτερικά των εξαρτημάτων είναι κρυμμένα.



Ορισμός Εξαρτήματος-Component

- Ο ορισμός εξαρτήματος ορίζει τη διεπαφή του εξαρτήματος στην αρχιτεκτονική.
- Απαραίτητα, αν η διεπαφή του εξαρτήματος δεν είναι ορισμένη κάπου αλλού (package, library).

```
ARCHITECTURE test OF test_entity
      COMPONENT and_gate
         PORT ( in1, in2 : IN BIT;
                   Out1 : OUT BIT0;
      END COMPONENT;
```

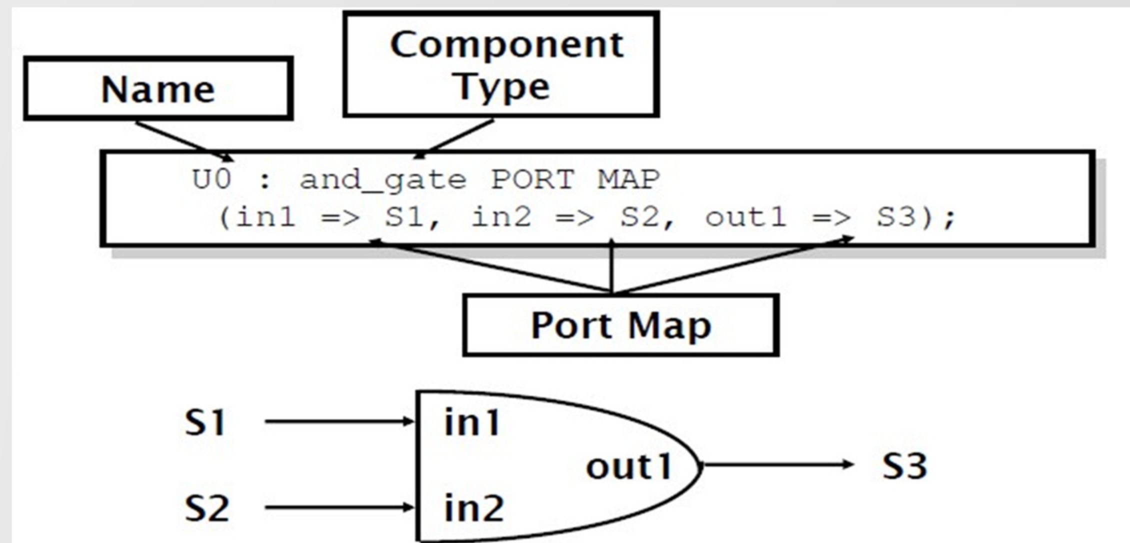
Συγκεκριμενοποίηση Εξαρτήματος

- Η δήλωση συγκεκριμενοποίσης χαρτογραφεί τη διεπαφή του εξαρτήματος σε άλλα αντικείμενα στην αρχιτεκτονική.

```
ARCHITECTURE test OF test_entity
       COMPONENT and_gate
            PORT ( in1, in2  : IN BIT;
                         out1 : OUT BIT);
       END COMPONENT;
       SIGNAL S1, S2, S3 : BIT;
BEGIN
        U0: and_gate PORT MAP (in1 => S1,
           In2 => s2, out1 => s3);
END test;
```

Σύνταξη Συγκεκριμενοποίησης Εξαρτήματος

- Η συγκεκριμενοποίηση έχει τρία βασικά σημεία:
  - ✓ Όνομα.
  - ✓ Τύπο εξαρτήματος-component type.
  - ✓ Αντιστοίχηση των θυρών-Port map.



Βιβλιοθήκες Στοιχείων/Υπομονάδων

Οι δηλώσεις των στοιχείων μπορούν να γίνει εντός πακέτων-packages.

- ✓ **Τα στοιχεία δεν απαιτείται να ορίζονται μέσα στο σώμα της αρχιτεκτονικής.**

```
PACKAGE my_stuff  IS
    COMPONENT and_gate
      PORT ( in1, in2 : IN BIT;
                 out1  : OUT BIT);
      END COMPONENT;
END my_stuff;

USE Work.my_stuff.ALL;

ARCHITECTURE test OF test_entity
        SIGNAL S1, S2, S3 : BIT;
BEGIN
        Gate1 : and_gate
          PORT MAP(S1,  S2, S3);
END test;
```

Generics

- Τα Generics επιτρέπουν το δομοστοιχείο να προσαρμοστεί κατά την συγκεκριμενοποίηση.
- Τα Generics μεταφέρουν την πληροφορία από την οντότητα στην αρχιτεκτονική.
- Κοινές χρήσεις των generics:
  - Προσαρμόζουν το χρονοδιάγραμμα.
  - Μεταβάλουν το εύρος των υπό-τύπων- subtypes.
  - Αλλάζουν το μέγεθος των συστοιχιών.

```
ENTITY adder IS
GENERIC (n: natural  :=2);
PORT(
        A: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        B: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        C: OUT STD_LOGIC;
        SUM: OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0)
);
END adder;
```

Μοντελοποίηση Τεχνολογίας

- Μία χρήση των generics είναι να εναλλάσσει το χρόνο ενός εξαρτήματος.
- Είναι πιθανό να υποδεικνύεται μία generic χρονική καθυστέρηση και μετά να καθορίζεται η ακριβής καθυστέρηση στην συγκεκριμενοποίση.

```
COMPONENT inv IS
    PORT ( in1 : IN BIT;
           out1 : OUT BIT);
    GENERIC (tplh, tphl : TIME);
END COMPONENT;
```

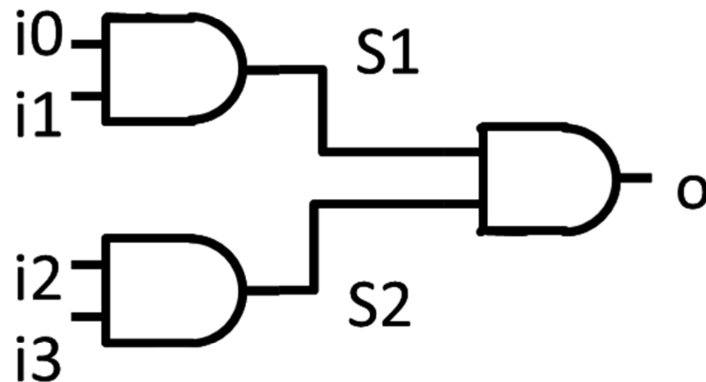
- Το παραπάνω παράδειγμα καθορίζει την ορίζει τη διεπαφή σε ένα εξάρτημα που ονομάζεται inv.
- Η χρονική διάδοση σε μεταβάσεις από high-σε-low και από low-σε-high μπορούν να προσαρμοστούν αργότερα.

Διαρθρωτικές Δηλώσεις (3/5)

Το GENERIC MAP είναι παρόμοιο με το PORT MAP μιας και χαρτογραφεί συγκεκριμένες τιμές στα generics που ορίζονται στο εξάρτημα.

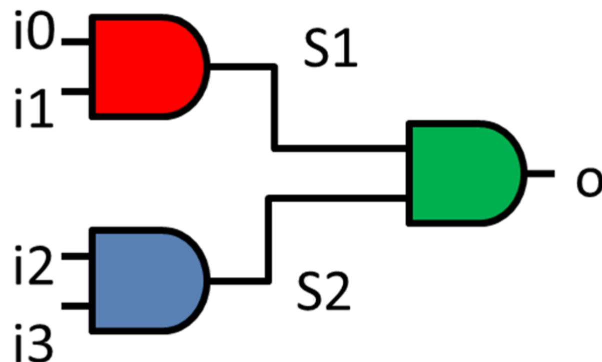
```
PACKAGE my_stuff IS
      COMPONENT and_gate
         GENERIC ( tplh, tphl : time);
         PORT ( in2, in2 : IN BIT; out1 : OUT BIT);
       END COMPONENT;
END my_stuff;

USE Work.my_stuff.ALL;
ARCHITECTURE test OF test_entity
       SIGNAL S1, S2, S3 : BIT;
BEGIN
        Gate1 : my_stuff.and_gate
           GENERIC MAP (2 ns, 3 ns)
           PORT MAP (S1, S2, S3);
END test;
```

Διαρθρωτικές Δηλώσεις (4/5)




```
ENTITY gates IS
  PORT (i0,i1,i2,i3: std_logic;
        o: out std_logic);
END gates;
ARCHITECTURE str of gates IS
COMPONENT and_gate
PORT (i1,i2: in std_logic;
        o: out std_logic);
END COMPONENT;
SIGNAL s1,s2: std_logic;
BEGIN
……………
```

Διαρθρωτικές Δηλώσεις (5/5)




```
U_and1: and_gate
PORT MAP(i1 => i0,
         i2 => i1,
          o => s1);
U_and2: and_gate
PORT MAP(i1 => i2,
         i2 => i3,
          o => s2);
U_and3: and_gate
PORT MAP(i1 => s1,
         i2 => s2,
          o => o);
END;
```

Απλοί Κανόνες (1/3)

Για συνδυαστική λογική, να χρησιμοποιείτε μόνο ταυτόχρονες δηλώσεις:

- Ταυτόχρονη εκχώρηση σημάτων (<=).
- Υπό συνθήκη ταυτόχρονη εκχώρηση σημάτων (when-else).
- Επιλεγμένη ταυτόχρονη εκχώρηση σημάτων (with-select-when).
- Παραγωγή σχεδίου για εξομοιώσεις (for-generate).

Απλοί Κανόνες (2/3)

Για κυκλώματα που αποτελούνται από:

- απλές λογικές λειτουργίες (λογικές πύλες),
- απλές αριθμητικές λειτουργίες (πρόσθεση, αφαίρεση, πολλαπλασιασμό),
- μετατοπίσεις, περιστροφές κατά μία σταθερά.

Χρησιμοποιείστε:

- ταυτόχρονη εκχώρηση σήματος (<=).

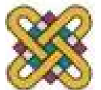
Απλοί Κανόνες (3/3)

Για κυκλώματα που αποτελούνται από:

- πολυπλέκτες.
- αποκωδικοποιητές, κωδικοποιητές.
- τριών καταστάσεων buffers.

Χρησιμοποιείστε:

- Υπό συνθήκη ταυτόχρονη εκχώρηση σημάτων (when-else).
- Επιλεγμένη ταυτόχρονη εκχώρηση σημάτων (with-select-when).



Αριστερή vs Δεξιά Πλευρά Εκχώρησης

Αριστερή πλευρά <= Δεξιά πλευρά

<= when-else

with-select <=

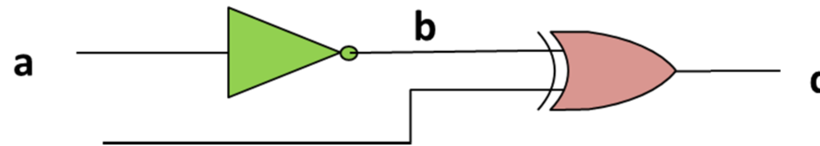
- Εσωτερικά σήματα (καθορισμένα σε μια δοσμένη αρχιτεκτονική)
- Ports καταστάσεων
  - **- out**
  - - inout
  - - buffer

Περιλαμβανόμενες Εκφράσεις:

- Εσωτερικά σήματα (καθορισμένα σε μια δοσμένη αρχιτεκτονική)
- Ports καταστάσεων
  - **- in**
  - - inout
  - - buffer

Περιγραφή Συνδυαστικών Κυκλωμάτων

```
ENTITY gates is
  port (a: in std_logic;
          d: out std_logic);
end gates;

Architecture rtl of gates is
signal b: std_logic;
begin
  b <= not a;
  d <= c xor b; --d<=c xor (not a);
end rtl;
```



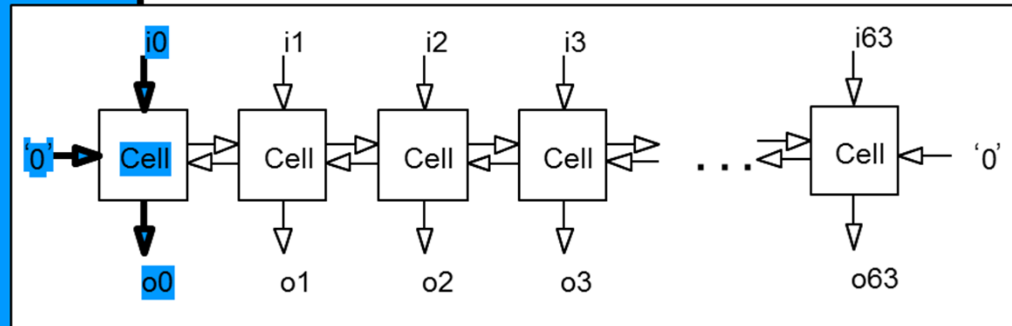
Δηλώσεις GENERATE (ταυτόχρονες μόνο)

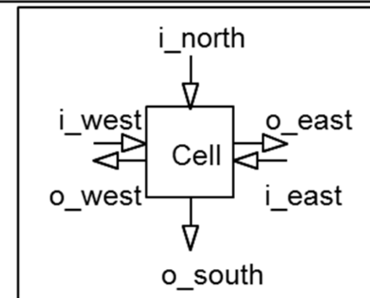
- Χρησιμοποιείται για να παράγει πολλαπλά παραδείγματα δομοστοιχείων σε μία ομογενή αρχιτεκτονική.

```
Z_Gen: For i in 0 to 7 generate

 z(i) <= x(i) AND y(i+8);

end generate;
```

Generate Παράδειγμα (1/2)

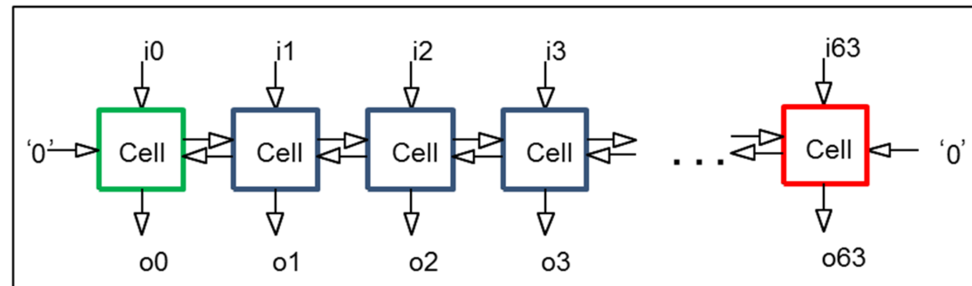
```
ENTITY cell_array IS
  PORT
    (i: in std_logic_vector(63 downto 0);
     o: out std_logic_vector(63 downto 0)
    );
END ENTITY cell_array;

ARCHITECTURE str OF cell_array IS
COMPONENT cell
  PORT
    (i_north: in std_logic;
     i_west: in std_logic;
     i_east: in std_logic;
     o_east: out std_logic;
     o_west: out std_logic;
     o_south: out std_logic
     );
end component;
  signal west: std_logic_vector(62 downto 0);
  signal east: std_logic_vector(62 downto 0);
BEGIN
```

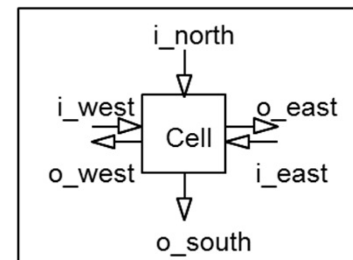




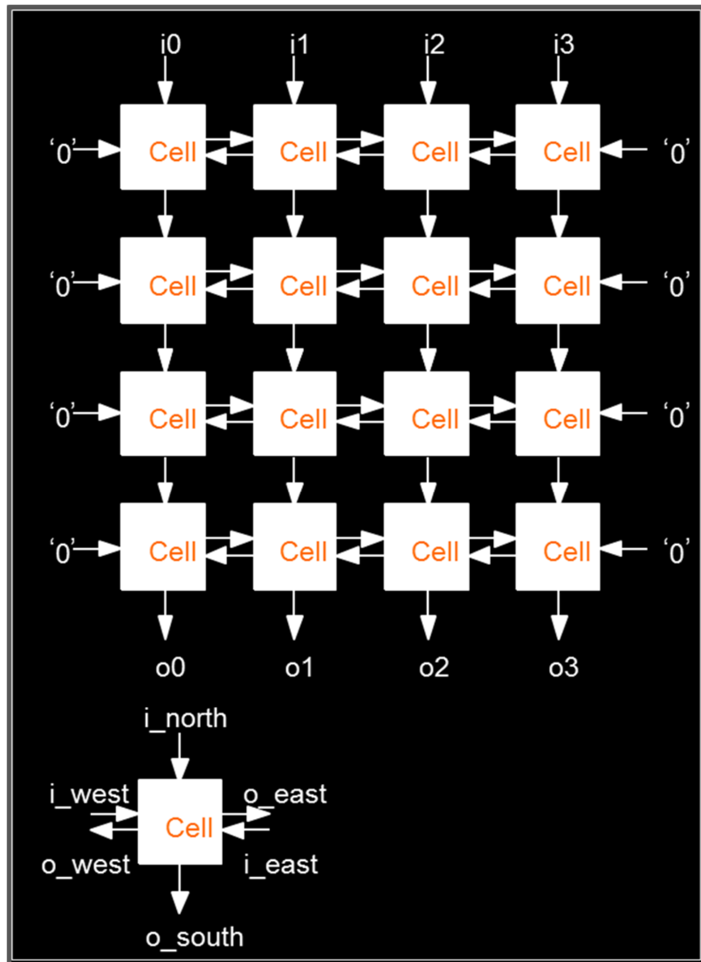
Generate Παράδειγμα (2/2)

```
  PORT MAP
    (i_north => i(0),
     i_west => '0',
     i_east => west(0),
     o_east => east(0),
     o_west => open,
     o_south => o(0)
     );
U_cell_63: cell
  PORT MAP
    (i_north => i(63),
     i_west => east(62),
     i_east => '0',
     o_east => open,
     o_west => west(62),
     o_south => o(63)
     );
U_top_gen: for i in 1 to 62
generate
  U_cell_i: cell
    PORT MAP
      (i_north => i(i),
       i_west => east(i-1),
       i_east => west(i),
       o_east => east(i),
       o_west => west(i-1),
       o_south => o(i)
       );
end generate;
```



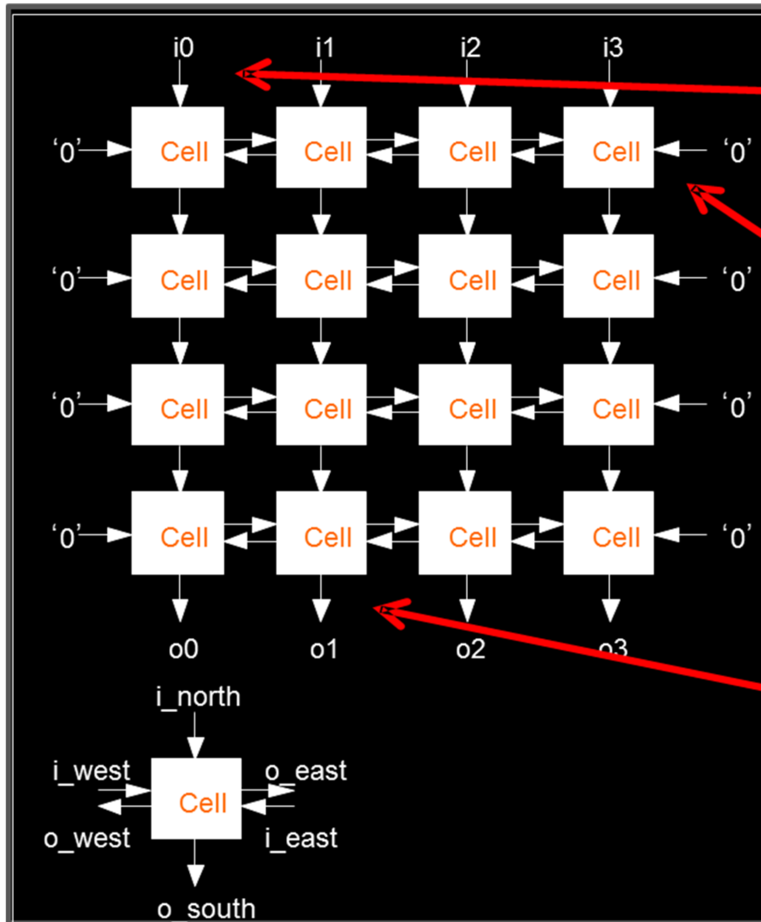


Generate Παράδειγμα 2 (1/5)



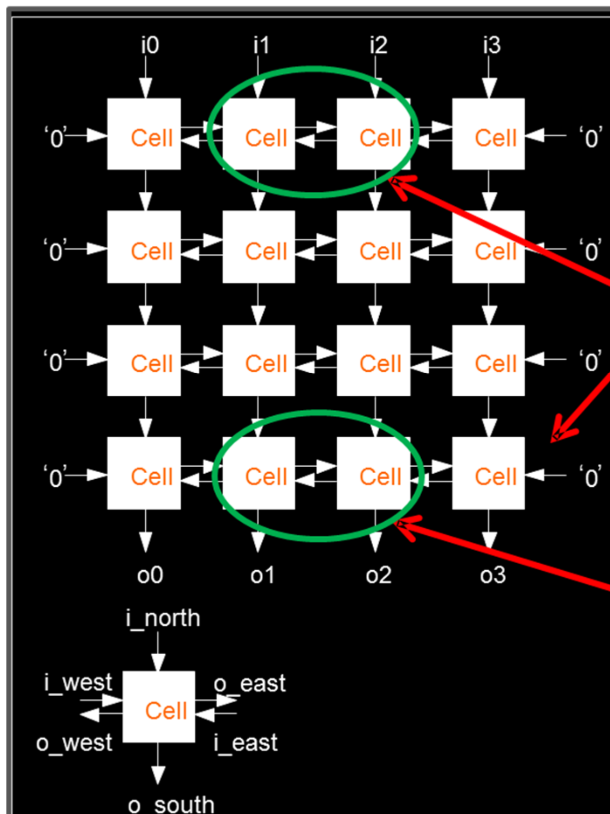

```
ENTITY cell_array IS
  PORT



    );
END ENTITY cell_array;

--
ARCHITECTURE str OF cell_array IS
COMPONENT cell
  PORT
    (i_north: in std_logic;
     i_west: in std_logic;
     i_east: in std_logic;
     o_east: out std_logic;
     o_west: out std_logic;
     o_south: out std_logic
     );
end component;
  type sig_array is array (3 downto 0)
of std_logic_vector(3 downto 0);
  type sig_array2 is array (2 downto 0)
of std_logic_vector(3 downto 0);
  signal south: sig_array2;
  signal west: sig_array;
  signal east: sig_array;
```

Generate Παράδειγμα 2 (2/5)



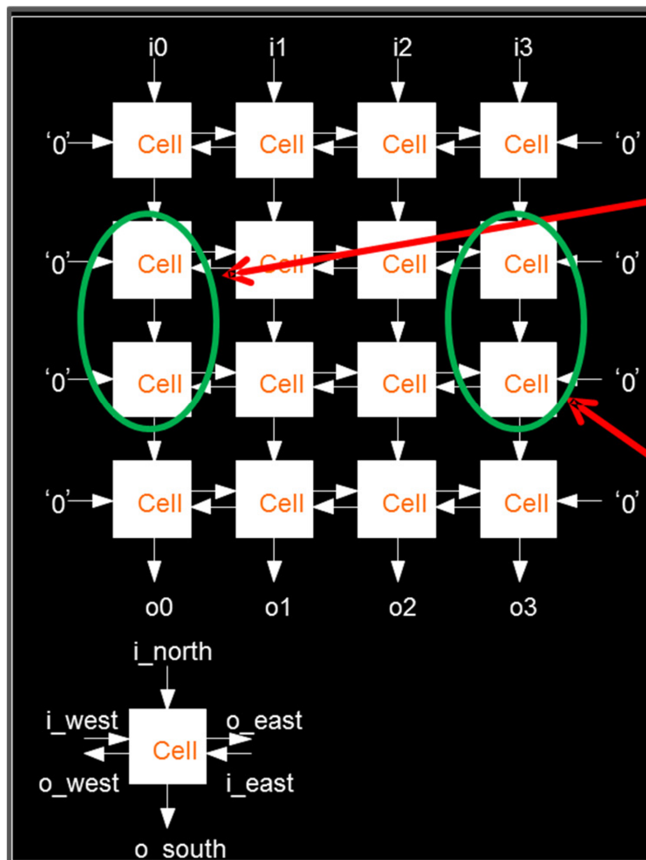

```
BEGIN
U cell 0 0: cell
   PORT MAP
     (i north => i(0),
      i west => '0',
      i east => west(0)(0),
      o east => east(0)(0),
      o west => open,
      o south => south(0)(0)
      );

U cell 0 3: cell
   PORT MAP
     (i north => i(3),
      i west => west(0)(3),
      i east => '0',
      o east => east(0)(3),
      o west => west(0)(3),
      o south => south(0)(3)
      );

U cell 3 0: cell
   PORT MAP
     (i north => north(2)(0),
      i west => '0',
      i east => west(3)(0),
      o east => east(3)(0),
      o west => open,
      o south => o(0)
      );
```

Generate Παράδειγμα 2 (3/5)



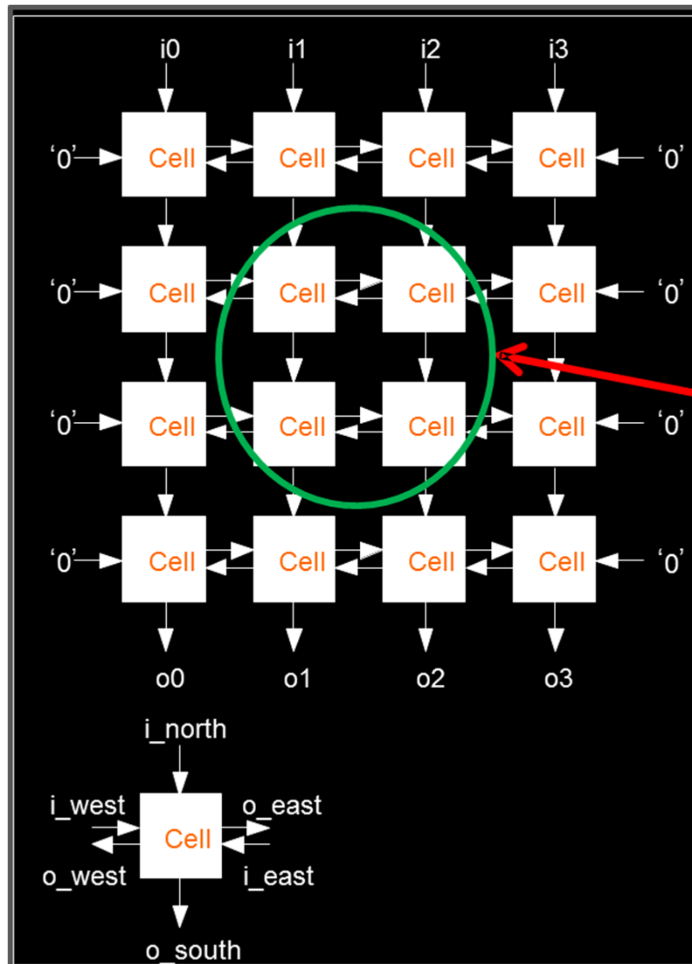

```
U_cell_3_3: cell
  PORT MAP
    (i_north => north(2)(3),
     i_west => west(3)(3),
     i_east => '0',
     o_east => open,
     o_west => west(3)(3),
     o_south => o(3)
     );
U_top_gen: for i in 1 to 2 generate
  U_cell_i_0: cell
    PORT MAP
      (i_north => i(i),
       i_west => east(i-1)(0),
       i_east => west(i)(0),
       o_east => east(i)(0),
       o_west => west(i-1)(0),
       o_south => south(i)(0)
       );
end generate;

  U_cell_i_3: cell
    PORT MAP
      (i_north => south(i)(2),
       i_west => east(i-1)(3),
       i_east => west(i)(3),
       o_east => east(i)(3),
       o_west => west(i-1)(3),
       o_south => o(i)
       );
```

Generate Παράδειγμα 2 (4/5)




```
U_left_gen: for i in 1 to 2
generate
  U_cell_0_i: cell
    PORT MAP
      (i_north => south(0)(i),
       i_west => '0',
       i_east => west(0)(i),
       o_east => east(0)(i),
       o_west => open,
       o_south => south(0)(i)
       );
end generate;

U_right_gen: for i in 1 to 2
generate
  U_cell_3_i: cell
    PORT MAP
      (i_north => south(2)(i),
       i_west => east(3)(i-1),
       i_east => '0',
       o_east => east(3)(i),
       o_west => west(3)(i-1),
       o_south => south(3)(i)
       );
end generate;
```

Generate Παράδειγμα 2 (5/5)




```
U_inner_gen_x: for i in 1 to 2
generate
  U_inner_gen_y: for j in 1 to 2
generate
    U_cell_i_j: cell
      PORT MAP

         i_west => east(i)(j-1),
         i_east => west(i)(j),
         o_east => east(i)(j),
         o_west => west(i)(j-1),
         o_south => south(i)(j)
         );
  end generate;
end generate;
END ARCHITECTURE str;
```

Περιγραφή Αριθμητικής Μονάδας

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

ENTITY add1 is
   port (a, b: in std_logic;
             cin: in std_logic;
             sum: out std_logic;
             cout: out std_logic);
  end add1;

ARCHITECTURE rtl of add1 is
Signal s: std_logic_vector(1 downto 0);
begin
  s <= ('0' & a) + b + cin;
  sum <= s(0);
  cout <= s(1);
end;
```

Χρονολογημένες Διεργασίες (Μάνταλο-Latch με ασύγχρονο reset) (1/2)

```
PROCESS(clk, rst_n)
BEGIN
  IF rst_n = '0' THEN
    q <= (others => '0');
  ELSIF clk= '1' THEN
    q <= d;
  END IF;
END PROCESS;
```

Χρονολογημένες Διεργασίες (Μάνταλο-Latch με ασύγχρονο reset) (2/2)

```
PROCESS(clk)
BEGIN
  IF clk = '1' THEN
     if rst_n = '0' then
       q <= (others => '0');
     else q <= d;
     end if;
  END IF;
END PROCESS;
```

Χρονολογημένες Διεργασίες (Flip-Flop με ασύγχρονο reset)

```
PROCESS(clk, rst_n)
BEGIN
  IF rst_n = '0' THEN
    q <= (others => '0');
  ELSIF clk'event and clk= '1' THEN
    q <= d;
  END IF;
END PROCESS;
```

Χρονολογημένες Διεργασίες (Flip-Flop με σύγχρονο reset)

```
PROCESS(clk)
BEGIN
   IF clk'event and clk= '1' THEN
      IF rst_n = '0' THEN
         q <= (others => '0');
      else q <= d;
     end if;
   END IF;
END PROCESS;
```

Τέλος Ενότητας