ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ

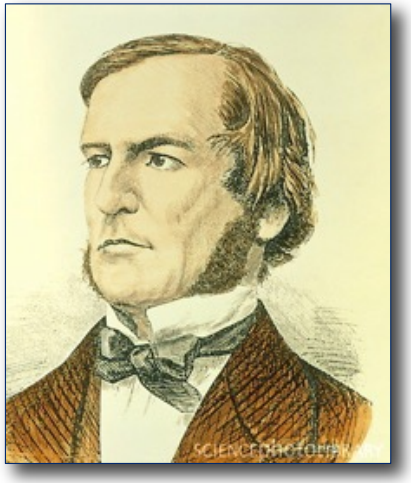
ΑΚΑΔΗΜΑΙΚΟ ΕΤΟΣ 2023-2024

ΔΙΔΑΣΚΩΝ: ΝΙΚΟΛΑΟΣ ΖΟΜΠΑΚΗΣ

Άλγεβρα Boole
και Λογικές Πύλες

BOOLEAN ALGEBRA: BIG PICTURE

- An algebra on 1's and 0's
  - With AND, OR, NOT operations



- What you start with
  - Axioms: basic things about objects and operations you just assume to be true at the start

- What you derive first
  - Laws and theorems: allow you to manipulate boolean expressions
  - …Also allow us to do some simplification on boolean expressions

- What you derive later
  - More "sophisticated" properties useful for manipulating digital designs represented in the form of boolean equations

George Boole, "The Mathematical Analysis of Logic," 1847.

Μαθηματικά Αξιώματα και Ιδιότητες

Κάθε μαθηματικό επαγωγικό σύστημα ορίζεται αν έχουμε:

- ✓ Ένα σύνολο στοιχείων Ι.
- ✓ Ένα σύνολο τελεστών Τ, δηλαδή συναρτήσεων που έχουν πεδίο ορισμού και τιμών στο σύνολο των στοιχείων Ι.
- ✓ Ένα σύνολο αξιωμάτων πάνω στα οποία βασιζόμαστε για να συνάγουμε τους κανόνες, θεωρήματα και τις ιδιότητες του συστήματος.

Βασικοί Ορισμοί

Οι παρακάτω ιδιότητες είναι χρήσιμες για το καθορισμό των χαρακτηριστικών των διάφορων αλγεβρικών δομών:

- ✓ *Κλειστότητα* (Closure): Ένα σύνολο S είναι κλειστό προς έναν δυαδικό τελεστή T αν για κάθε $x,y \in S$, έχουμε και $T(x,y) \in S$.
- ✓ *Προσεταιριστικός Νόμος* (Associative Law). Ένας δυαδικός τελεστής είναι προσεταιριστικός αν $T(x,T(y,z))=T(T(x,y),z)$ για κάθε $x,y,z \in S$
- ✓ *Αντιμεταθετικός Νόμος* (Commutative Law). Ένας δυαδικός τελεστής είναι αντιμεταθετικός αν $T(x,y)=T(y,x)$, για κάθε $x,y \in S$.
- ✓ *Ουδέτερο Στοιχείο*, ονομάζεται ένα στοιχείο $e \in S$ το οποίο έχει την ιδιότητα $T(e,x)=T(x,e)=x$, για κάθε $x,y \in S$.
- ✓ *Αντίστροφο στοιχείο*. Το x είναι το αντίστροφο του y αν και μόνο αν $T(x,y)=T(y,x)=e$
- ✓ *Επιμεριστική Ιδιότητα*. Ο τελεστής T είναι επιμεριστικός ως προς τον J αν $T(x,J(y,z))=J(T(x,y),T(x,z))$.

Παραδείγματα...

- Η πρόσθεση $T(x,y)=x+y$ και ο πολλαπλασιασμός $J(x,y)=x\cdot y$ είναι δύο παραδείγματα *δυαδικών τελεστών* που μπορούν να οριστούν στο σύνολο των ακεραίων αριθμών $I=\{...,-2,-1,0,1,2,...\}$

- Το I είναι *κλειστό* ως προς την πρόσθεση, τον πολλαπλασιασμό και την αφαίρεση. Αντίθετα το σύνολο των φυσικών αριθμών $N=\{1,2,...\}$ δεν είναι κλειστό ως προς την αφαίρεση.

- Το I έχει ως *ουδέτερο στοιχείο* το 0 ως προς την πρόσθεση και το 1 ως προς τον πολλαπλασιασμό.
- Κάθε στοιχείο του I έχει *αντίθετο* ως προς την πρόσθεση. Π.χ. Ο αντίθετος του 2 είναι ο -2 επειδή $2+(-2)=(-2)+2=0$. Ωστόσο δεν έχει αντίθετο ως προς τον πολλαπλασιασμό.

- Η πρόσθεση και ο πολλαπλασιασμός είναι αντιμεταθετικοί τελεστές σε αντίθεση με την αφαίρεση.

- Ο πολλαπλασιασμός είναι επιμεριστικός ως προς την πρόσθεση: $(x+y)\cdot z= x\cdot z+y\cdot z$ στο I και στο N.
- Ο πολλαπλασιασμός και η πρόσθεση είναι αντιμεταθετικοί τελεστές: $x+y=y+x$ και $x\cdot y=y\cdot x$.

Ορισμός της Άλγεβρας Boole.

Ένα σύνολο στοιχείων *B* μαζί με δύο τελεστές + και · ονομάζεται *άλγεβρα* Boole *αρκεί να* ικανοποιούνται τα *παρακάτω αξιώματα* (αξιώματα Huntington):

1. Το B είναι κλειστό ως προς τους δύο τελεστές.
2. Υπάρχει ουδέτερο στοιχείο ως προς τον +, το οποίο συμβολίζεται με 0.
3. Υπάρχει ουδέτερο στοιχείο ως προς τον ·, το οποίο συμβολίζεται με 1.
4. Οι τελεστές + και · είναι αντιμεταθετικοί.
5. Ο + είναι επιμεριστικός ως προς τον · και ταυτόχρονα ο · είναι επιμεριστικός ως προς τον +.
6. Για κάθε $x \in B$ υπάρχει ένα στοιχείο x' το οποίο ονομάζεται συμπλήρωμα του x και για το οποίο ισχύει $x+x'=1$ και $x \cdot x'=0$.
7. Υπάρχουν τουλάχιστον δύο στοιχεία του *x* και *y* του B για τα οποία έχουμε $x \neq y$.

Το σύνολο των ακεραίων I δεν είναι άλγεβρα Boole ως προς τον πολλαπλασιασμό και την πρόσθεση!!!!!

BOOLEAN ALGEBRA: ΑΞΙΩΜΑΤΑ

Formal version	*English version*
1. *B* contains at least two elements, *0* and *1*, *such that* $0 \neq 1$	Math formality…
2. *Closure* $a,b \in B$, (i) $a + b \in B$ (ii) $a \bullet b \in B$	Result of AND, OR stays in set you start with
3. *Commutative Laws*: $a,b \in B$, (i) $a + b = b + a$ (ii) $a \bullet b = b \bullet a$	For primitive AND, OR of 2 inputs, order doesn't matter
4. *Identities*: $0, 1 \in B$ (i) $a + 0 = a$ (ii) $a \bullet 1 = a$	There are identity elements for AND, OR, that give you back what you started with
5. *Distributive Laws*: (i) $a + (b \bullet c) = (a + b) \bullet (a + c)$ (ii) $a \bullet (b + c) = a \bullet b + a \bullet c$	• distributes over +, just like algebra …but + distributes over •, also (‼)
6. *Complement*: (i) $a + \bar{a} = 1$ (ii) $a \bullet \bar{a} = 0$	There is a complement element; AND/ORing with it gives the identity elm.

Μερικές Παρατηρήσεις...

- ✓ Τα αξιώματα του Huntington δεν συμπεριλαμβάνουνε τον προσεταιριστικό νόμο, ο οποίος όμως αποδεικνύεται εύκολα για την δίτιμη άλγεβρα Boole που θα εξετάσουμε παρακάτω.
- ✓ Σε αντίθεση με την συνηθισμένη άλγεβρα, το + είναι επιμεριστικό ως προς το ·, δηλαδή $x+(y\cdot z)=(x+y)\cdot(x+z)$.
- ✓ Η άλγεβρα Boole δεν έχει πολλαπλασιαστικά ή προσθετικά αντίστροφα και επομένως δεν υπάρχουν οι πράξεις της αφαίρεσης και της διαίρεσης.
- ✓ Το συμπλήρωμα είναι ένας τελεστής που δεν υπάρχει στην συνηθισμένη άλγεβρα.
- ✓ Το σύνολο B μπορεί να έχει και πεπερασμένο αριθμό στοιχείων ενώ η συνηθισμένη άλγεβρα είναι ορισμένη στο απειροσύνολο των πραγματικών ή των μιγαδικών αριθμών.

Ορισμός της Δίτιμης Άλγεβρας Boole.

- ✓ Θεωρούμε πως το B περιέχει 2 στοιχεία το 0 και το 1, δηλαδή B={0,1}.
- ✓ Ορίζουμε τους τελεστές + και · ώστε να ανταποκρίνονται στο λογικό OR και το λογικό AND αντίστοιχα:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

- ✓ Για να αποδείξουμε πως το σύνολο B είναι άλγεβρα Boole υπό τους δύο τελεστές θα πρέπει να επαληθεύσουμε τα αξιώματα του Huntington.
- ✓ Καταρχήν το B είναι κλειστό ως προς τους δύο τελεστές αφού για κάθε x και y που ανήκουν στο B, έχουμε και $x \cdot y \in B$, $x+y \in B$. Επίσης το 0 είναι το ουδέτερο στοιχείο του +, αφού $x+0=0+x=x$ ενώ το 1 είναι το ουδέτερο στοιχείο του · αφού $x \cdot 1=1 \cdot x=x$.
- ✓ Το συμπλήρωμα του 1 είναι το 0, αφού 0+1=1+0=1 και 0·1=1·0=0. Επίσης 0≠1.
- ✓ Για κάθε x,y ∈B ισχύει, $x \cdot y = y \cdot x$ και $x+y = y+x$.

Επιμεριστικότητα Τελεστών

- ✓ Για να αποδείξουμε πως ο + είναι επιμεριστικός ως προς τον · και ταυτόχρονα ο · είναι επιμεριστικός ως προς τον + δεν έχουμε παρά να θεωρήσουμε όλους τους δυνατούς συνδυασμούς τριών στοιχείων του Β. Για παράδειγμα για να αποδείξουμε πως $x\cdot(y+z)=(x\cdot y)+(x\cdot z)$ φτιάχνουμε τον παρακάτω πίνακα:

x	y	z	y+z	x·(y+z)	x·y	x·z	(x·y)+(x·z)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

- ✓ Επομένως προκύπτει πως $x\cdot(y+z)=(x\cdot y)+(x\cdot z)$ ενώ με παρόμοιο τρόπο αποδεικνύουμε πως $x+y\cdot z=(x+y)\cdot(x+z)$.

ΔΥΙΣΜΟΣ (DUALITY)

- Observation
  - All the axioms come in "dual" form
  - Anything true for an expression also true for its dual
  - So any derivation you could make that is true, can be flipped into dual form, and it stays true
- Duality — more formally
  - A dual of a boolean expression is derived by replacing
    - Every AND operation with... An OR operation
    - Every OR operation with... An AND
    - Every constant 1 with... A constant 0
    - Every constant 0 with... A constant 1
    - But don't change any of the literals or play with the complements!

Example

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

$$\rightarrow \; a + (b \cdot c) = (a + b) \cdot (a + c)$$

BOOLEAN ALGEBRA: ΘΕΩΡΗΜΑΤΑ (1/2)

	Dual	
Operations with 0 and 1:		
1. $X + 0 = X$	1D. $X \cdot 1 = X$	AND, OR with identities gives you back the original variable or the identity
2. $X + 1 = 1$	2D. $X \cdot 0 = 0$	
Idempotent Law:		
3. $X + X = X$	3D. $X \cdot X = X$	AND, OR with self = self
Involution Law:		
4. $\overline{(\overline{X})} = X$		double complement = no complement
Laws of Complementarity:		
5. $X + \overline{X} = 1$	5D. $X \cdot \overline{X} = 0$	AND, OR with complement gives you an identity
Commutative Law:		
6. $X + Y = Y + X$	6D. $X \cdot Y = Y \cdot X$	Just an axiom…

BOOLEAN ALGEBRA: ΘΕΩΡΗΜΑΤΑ (2/2)

Associative Laws:

7. $(X + Y) + Z = X + (Y + Z) = X + Y + Z$

7D. $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$

Parenthesis order does not matter

Distributive Laws:

8. $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$

8D. $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$

Axiom

Simplification Theorems:

9. 9D.

10. 10D.

11. 11D.

Useful for simplifying expressions

Actually worth remembering — they show up a lot in real designs…

BOOLEAN ALGEBRA: PROVING THINGS

Proving theorems via axioms of Boolean Algebra:

EX: Prove the theorem: $X \cdot Y + X \cdot \overline{Y} = X$

Distributive (5)

Complement (6)

Identity (4)

EX2: Prove the theorem: $X + X \cdot Y = X$

Identity (4)

Distributive (5)

Identity (2)

Identity (4)

Προτεραιότητα Τελεστών

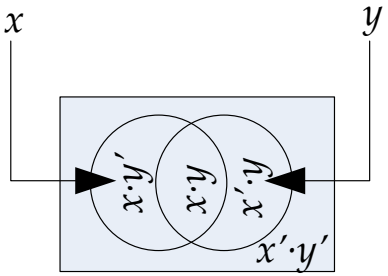
- ✓ Όπως και στην κανονική αριθμητική θα πρέπει να ορίσουμε μία σειρά προτεραιότητας για τους τελεστές της άλγεβρας Boole.
- ✓ Η προτεραιότητα που χρησιμοποιούμε ακολουθεί την εξής σειρά: πρώτες έρχονται οι παρενθέσεις, μετά τα συμπληρώματα (λογικό NOT), μετά το · (λογικό AND) και μετά το + (λογικό OR).
- ✓ Για παράδειγμα αν θέλαμε να υπολογίσουμε το $(x+y)'$ για διάφορους συνδυασμούς x και y θα πρέπει πρώτα να κάνουμε την πράξη OR μέσα στην παρένθεση και μετά να πάρουμε το συμπλήρωμα του αποτελέσματος.
- ✓ Στην $x'\cdot y'$ πρώτα παίρνουμε τα συμπληρώματα των x και y και μετά πραγματοποιούμε το λογικό AND.
- ✓ Στην $x+x\cdot y$ πρώτα πρέπει να πραγματοποιήσουμε το λογικό AND $x\cdot y$ και μετά το λογικό OR.
- ✓ Η προτεραιότητα των τελεστών θυμίζει την προτεραιότητα των πράξεων στην κανονική άλγεβρα αν αντιστοιχήσουμε το · στον πολλαπλασιασμό και το + στην πρόσθεση.

Διαγράμματα Venn

Συχνά είναι χρήσιμο να αναπαριστούμε σχέσεις μεταξύ διαφόρων μεταβλητών της Άλγεβρας Boole με τη βοήθεια των διαγραμμάτων Venn.

Στα διαγράμματα αυτά, κάθε μεταβλητή παριστάνεται με ένα κύκλο και οι κύκλοι των διαφόρων μεταβλητών αλληλοκαλύπτονται.

Το λογικό OR δύο μεταβλητών είναι τα σημεία που ανήκουν τουλάχιστον σε έναν από τους δύο κύκλους, το λογικό AND είναι τα σημεία που ανήκουν και στους δύο κύκλους ενώ το συμπλήρωμα είναι τα σημεία που δεν ανήκουν στον αντίστοιχο κύκλο.



Βασικά Θεωρήματα της Άλγεβρας Boole.

- ✓ $((x)')'=x$. Δηλαδή δύο αρνήσεις κάνουν μία κατάφαση. Προκύπτει από το γεγονός ότι $x'+x=x+x'=1$ ενώ $x\cdot x'=x'\cdot x=0$, επομένως το συμπλήρωμα του x' είναι το x οπότε, $((x)')'=x$.

- ✓ $x+x=x$ και $x\cdot x=x$ (για την απόδειξη δείτε την προηγούμενη διαφάνεια).

- ✓ $x+1=1$ και $x\cdot 0=0$. Η πρώτη σχέση προκύπτει εύκολα αν χρησιμοποιήσουμε το γεγονός πως $x+1=1\cdot(x+1)=(x+x')\cdot(x+1)=x+x'\cdot 1=x+x'=1$. Η δεύτερη σχέση προκύπτει από τον δυϊσμό.

- ✓ $x+(y+z)=(x+y)+z$ και $x\cdot(y\cdot z)=(x\cdot y)\cdot z$, δηλαδή οι τελεστές + και · είναι προσεταιριστικοί.

- ✓ $(x+y)'=x'\cdot y'$ και $(x\cdot y)'=x'+y'$ (Θεώρημα του DeMorgan).

- ✓ $x+xy=x$ και $x\cdot(x+y)=x$ (Ιδιότητα της απορρόφησης).

- ✓ Φυσικά θα μπορούσαμε να αποδείξουμε όλα τα θεωρήματα με την βοήθεια πινάκων σαν αυτών που χρησιμοποιήσαμε για να αποδείξουμε τον επιμερισμό.

Συναρτήσεις Boole

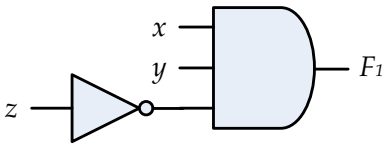
- ✓ Μια συνάρτηση Boole ονομάζεται οποιαδήποτε έκφραση μεταβλητών που ορίζονται μέσα στο B.
- ✓ Μπορεί να σχηματίζεται από παρενθέσεις, τους τελεστές +, · και συμπληρώματα.
- ✓ Παραδείγματα συναρτήσεων είναι η $F_1=xyz'$, η $F_2=x+y'\cdot z$ και η $F_3=x'\cdot y'\cdot z + x'\cdot y\cdot z+x\cdot y'$ και η $F_4=xy+x'z$.
- ✓ Για κάθε συνάρτηση Boole μπορούμε να φτιάξουμε τον πίνακα αλήθειας της που σε κάθε συνδυασμό τιμών των μεταβλητών της αντιστοιχεί μία τιμή στη συνάρτηση. Αν μία συνάρτηση είναι n μεταβλητών τότε έπεται πως ο πίνακας αλήθειας της έχει 2^n καταχωρήσεις.

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

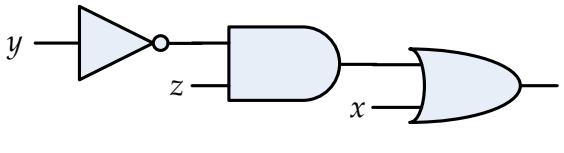
- ✓ Παρατηρούμε πως $F_3=F_4$ και επομένως μία συνάρτηση Boole δεν έχει μοναδικό τρόπο γραφής!

Αναπαράσταση Συναρτήσεων Boole με Πύλες

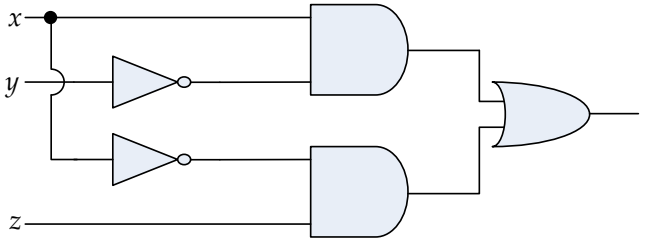
- ✓ Δεδομένης της έκφρασης της συνάρτησης Boole είναι σχετικά απλό να την αναπαραστήσουμε με λογικές πύλες.
- ✓ Ωστόσο θα πρέπει να προσέχουμε την προτεραιότητα των τελεστών.



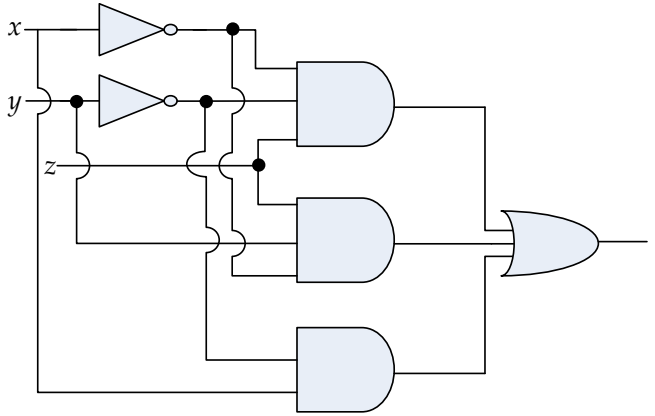
$F_1=xyz'$



$F_2=x+y'z$



$F_4=xy'+x'z$



$F_3=x'y'z+x'yz+xy'$

- ✓ **Παρατηρούμε πόσο πιο απλό είναι το κύκλωμα της F_4 από το κύκλωμα της F_3. Επομένως ο τρόπος έκφρασης μιας συνάρτησης έχει άμεση σχέση με την πολυπλοκότητα του αντίστοιχου ψηφιακού κυκλώματος!**

Συμπληρώματα Συναρτήσεων

- ✓ Εξ' ορισμού **το συμπλήρωμα *F'* μιας συνάρτησης *F* είναι η συνάρτηση εκείνη η οποία ισούται με 1 όταν *F*=0 ενώ είναι ίση με 0 όταν *F*=1.**
- ✓ Χρησιμοποιώντας το **θεώρημα του DeMorgan** μπορούμε να υπολογίσουμε το συμπλήρωμα μιας συνάρτησης.
- ✓ Για παράδειγμα αν $F=x+y+z$, έχουμε $F'=(x+y+z)'=x'y'z'$
- ✓ Γενικά οποιοδήποτε άθροισμα της μορφής $a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_N + b_1 \cdot \ldots \cdot b_N + \ldots$ έχει ως συμπλήρωμα το $(a_1'+a_2'+a_3'+\ldots+a_N') \cdot (b_1'+\ldots+b_N') \cdot \ldots$
- ✓ **Επομένως κάθε μεταβλητή εμφανίζεται με το συμπλήρωμα της, τα λογικά OR μετατρέπονται σε λογικά AND και το αντίστροφο**.

Κανονικές και Πρότυπες Μορφές

- ✓ Για ένα σύνολο n μεταβλητών $\{a_1,\ldots,a_N\}$ μπορούμε να σχηματίσουμε 2^n διαφορετικά γινόμενα της μορφής $a_1' \cdot \ldots \cdot a_N$ όπου κάθε μεταβλητή μπορεί να συμμετέχει είτε αυτούσια, είτε με το συμπλήρωμα της.

- ✓ Κάθε τέτοιο γινόμενο ονομάζεται **ελαχιστόρος** ή πρότυπο γινόμενο. Κάθε ελαχιστόρος ισούται με 1 για έναν και μόνο συνδυασμό τιμών των μεταβλητών $a_1,\ldots, a_N$

- ✓ Ο παρακάτω πίνακας δείχνει τους ελαχιστόρους που παράγονται από 3 μεταβλητές (x,y,z). Κάθε όρος ονομάζεται m_i όπου i είναι το δεκαδικό ισοδύναμο του δυαδικού συνδυασμού για τον οποίο ο ελαχιστόρος είναι ίσος με 1.

x	y	z	Ελαχιστόρος	Ονομασία
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

Κανονικές και Πρότυπες Μορφές

- ✓ Ομοίως, για ένα σύνολο n μεταβλητών $\{a_1,\ldots,a_N\}$ μπορούμε να σχηματίσουμε 2^n διαφορετικά *αθροίσματα* της μορφής $a_1'+\ldots+a_N$ όπου κάθε μεταβλητή μπορεί να συμμετέχει είτε αυτούσια, είτε με το συμπλήρωμα της.

- ✓ Κάθε τέτοιο άθροισμα ονομάζεται **μεγιστόρος** ή πρότυπο άθροισμα. Κάθε μεγιστόρος **ισούται με 0 για έναν και μόνο συνδυασμό τιμών** των μεταβλητών $a_1,\ldots, a_N$

- ✓ Ο παρακάτω πίνακας δείχνει τους μεγιστόρους που παράγονται από 3 μεταβλητές (x,y,z). Κάθε όρος ονομάζεται M_i όπου i είναι το δεκαδικό ισοδύναμο του δυαδικού συνδυασμού για τον οποίο ο μεγιστόρος είναι ίσος με 0.

x	y	z	Μεγιστόρος	Ονομασία
0	0	0	$x+y+z$	M_0
0	0	1	$x+y+z'$	M_1
0	1	0	$x+y'+z$	M_2
0	1	1	$x+y'+z'$	M_3
1	0	0	$x'+y+z$	M_4
1	0	1	$x'+y+z'$	M_5
1	1	0	$x'+y'+z$	M_6
1	1	1	$x'+y'+z'$	M_7

Πίνακες Αλήθειας και Πρότυπες Μορφές

- ✓ Δεδομένου του πίνακα αλήθειας μίας συνάρτησης μπορούμε να την γράψουμε ως γινόμενο μεγιστόρων ή άθροισμα ελαχιστόρων.
- ✓ Για να την γράψουμε ως γινόμενο μεγιστόρων, απλά πολλαπλασιάζουμε τους μεγιστόρους για τους οποίους η συνάρτηση είναι ίση με 0.
- ✓ Για να την γράψουμε ως άθροισμα ελαχιστόρων, απλά αθροίζουμε τους ελαχιστόρους για τους οποίους η συνάρτηση είναι ίση με 1.
- ✓ Για παράδειγμα έστω μία συνάρτηση F τριών μεταβλητών (x,y,z) με τον παρακάτω πίνακα αλήθειας:

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- ✓ Μπορούμε να γράψουμε την F, ως $F=m_1+m_4+m_7$ ή ως $F=M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$.

- ✓ Για συντομία, μπορούμε να γράψουμε την F και στην εξής μορφή:

  $F=\Sigma(1,4,7)$ και $F=\Pi(0,2,3,5,6)$. Παρατηρείστε πως αν ο ελαχιστόρος m_j ανήκει στο άθροισμα ελαχιστόρων της F, τότε ο M_j δεν ανήκει στο γινόμενο μεγιστόρων της F και αντίστροφα.

Άλλες Λογικές Πράξεις

- ✓ Η πράξεις του λογικού AND και του λογικού OR είναι στην ουσία λογικές συναρτήσεις δύο μεταβλητών στις οποίες αντιστοιχούν μία τιμή 0 ή 1 σε κάθε δυνατή δυάδα των μεταβλητών (*x,y*). Ωστόσο υπάρχουν άλλες 14 πιθανές συναρτήσεις που μπορούν να κατασκευαστούν, όπως δείχνει και ο παρακάτω πίνακας:

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Σύμβολο Τελεστή			·	/		/		⊕	+	↓	⊙	‘	⊂	‘	⊃	↑	

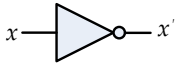
Άλλες Λογικές Πράξεις

Συναρτήσεις Boole	Σύμβολο Τελεστή	Όνομα	Σχόλια
$F_0=0$		Ουδέτερη	Δυαδική Σταθερά 0
$F_1=x\cdot y$	$x\cdot y$	Λογικό AND	x AND y
$F_2=x\cdot y'$	x/y	Αποτροπή	x αλλά όχι y
$F_3=x$		Μεταφορά	x
$F_4=x'y$	y/x	Αποτροπή	y αλλά όχι x
$F_5=y$		Μεταφορά	y
$F_6=xy'+x'y$	$\oplus$	Αποκλειστικό OR	είτε x είτε y
$F_7=x+y$	+	Λογικό OR	x OR y
$F_8=(x+y)'$	$x\downarrow y$	Λογικό NOR	x NOR y
$F_9=x'y'+xy$	$x \odot y$	Ισοδυναμία	$x = y$
$F_{10}=y'$	y'	Συμπλήρωμα	NOT y
$F_{11}=x+y'$	$x \subset y$	Συνεπαγωγή	Αν ισχύει το y τότε x
$F_{12}=x'$	x'	Συμπλήρωμα	NOT x
$F_{13}=x'+y$	$x \supset y$	Συνεπαγωγή	Αν ισχύει το x τότε y
$F_{14}=(xy)'$	$x\uparrow y$	Λογικό NAND	x NAND y
$F_{15}=1$		Ταυτότητα	Δυαδική Σταθερά 1

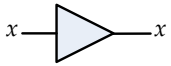
Άλλες Λογικές Πύλες

AND

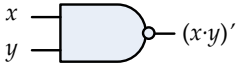
OR



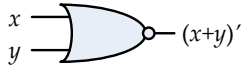
NOT



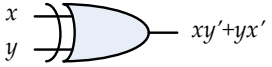
Απομονωτής



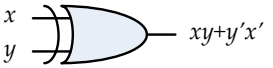
NAND



NOR



XOR



XNOR

- ✓ Πέρα από τις 3 βασικές λογικές πύλες (AND, OR, NOT), ορίζονται και άλλες 3 λογικές πύλες.
- ✓ Ο Απομονωτής συνήθως χρησιμοποιείται μόνο για ενίσχυση του σήματος ώστε να μπορεί να οδηγηθεί και σε άλλες λογικές πύλες.
- ✓ Οι πύλες NAND και NOR είναι στην ουσία τα συμπληρώματα των AND και OR.
- ✓ Οι NAND και NOR χρησιμοποιούνται ευρύτατα επειδή είναι εύκολο να υλοποιηθούν με τρανζίστορ και επειδή οι πιο πολύπλοκες συναρτήσεις Boole μπορούν εύκολα να υλοποιηθούνε με αυτές.
- ✓ Οι πύλες XOR χρησιμοποιούνται και αυτές αρκετά συχνά (π.χ. Σε αθροιστές, κτλ).
- ✓ Εκτός από την πύλη NOT και τον απομονωτή, οι υπόλοιπες πύλες μπορούνε εύκολα να επεκταθούνε για παραπάνω από δύο εισόδους.
- ✓ Στις πύλες AND και OR δεν έχει σημασία η σειρά με την οποία θεωρούμε τις μεταβλητές εισόδου αφού ισχύει η προσαιτεριστική και η αντιμεταθετική ιδιότητα, οπότε για παράδειγμα (x+y)+z=x+(y+z)=x+y+z=x+z+y, κτλ κτλ.
- ✓ Για τις πύλες NAND και NOR χρειάζεται λίγο προσοχή αφού ναι μεν είναι αντιμεταθετικές αλλά δεν είναι προσαιτεριστικές. Επομένως ορίζουμε απευθείας την NOR τριων μεταβλητών ως (x+y+z)' και την NAND ως (xyz)'
- ✓ Οι πύλες XOR και XNOR πολλαπλών εισόδων μπορούν να θεωρηθούν ως κυκλώματα ισοτιμίας!

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

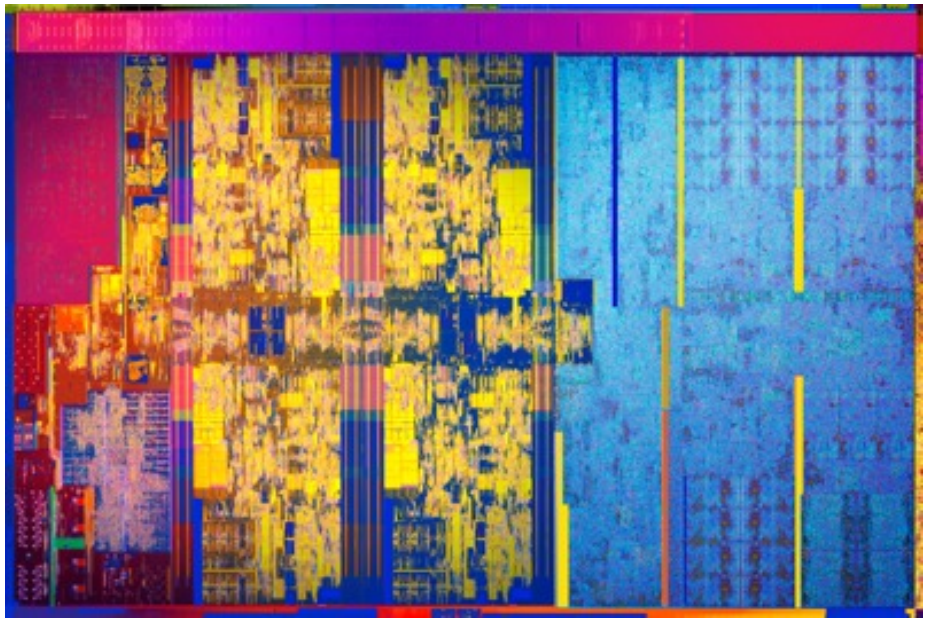
ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ

ΑΚΑΔΗΜΑΙΚΟ ΕΤΟΣ 2023-2024

VERILOG INTRO I

ΔΙΔΑΣΚΩΝ: ΝΙΚΟΛΑΟΣ ΖΟΜΠΑΚΗΣ ΞΥΔΗΣ

2017: INTEL KABY LAKE



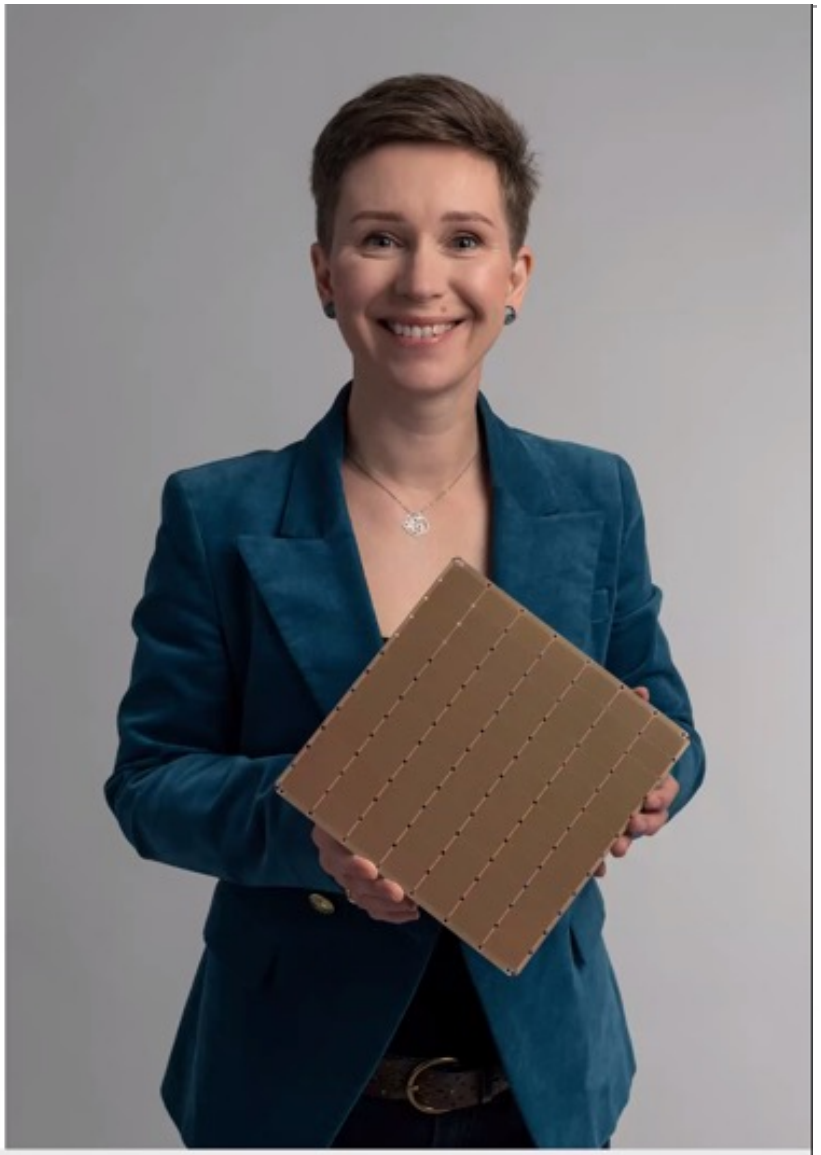
https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake

- **64-bit processor**
- **4 cores, 8 threads**
- **14-19 stage pipeline**
- **3.9 GHz clock**
- **1.75B transistors**

Cerebras WSE-2
56x Larger than Traditional Chip

The Largest Chip Ever Built

46,225	mm^2 silicon
2.6	Trillion transistors
850,000	AI optimized cores
40	Gigabytes on-chip memory
20	Petabyte/s memory bandwidth
220	Petabit/s fabric bandwidth
7nm	Process technology at TSMC



cerebras

Πώς να αντιμετωπίσουμε αυτήν την πολυπλοκότητα;

- Τι χρειαζόμαστε στη σχεδίαση ψηφιακών συστημάτων
  - Να μπορούμε να προδιαγράφουμε πολύπλοκα συστήματα και σχεδιαστικές λύσεις
    - επικοινωνήστε με άλλους στην ομάδα σχεδιασμού σας
  - … Και να προσομοιώνουμε τη συμπεριφορά τους
    - Είναι τελικά αυτό που θέλω να φτιάξω
  - … Και για να συνθέτουμε (αυτόματα σχεδίαση) τμημάτα τους
    - Κυκλώματα χωρίς σφάλματα για εφαρμογή

- Γλώσσες περιγραφής υλικού (Hardware description languages-- HDLs)

Hardware description languages

- **Verilog**
  - Developed in 1984 by gateway design automation
  - Became an IEEE standard (1364) in 1995
  - More popular in US

- **Vhdl (vhsic hardware description language)**
  - Developed in 1981 by the US department of defense
  - Became an IEEE standard (1076) in 1987
  - More popular in europe

Verilog HDL (Hardware Description Language)

Η γλώσσα **Verilog HDL** (**H**ardware **D**escription **L**anguage), όπως και η **VHDL**, ονομάζονται ***Γλώσσες Περιγραφής Υλικού*** (*ΓΠΥ*) και χρησιμοποιούνται ευρέως στη βιομηχανία για την περιγραφή ψηφιακών συστημάτων.

Συγκεκριμένα, οι περιγραφές με *ΓΠΥ* μπορεί να δίνουν διάφορες λεπτομέρειες σχετικά με τη λειτουργία και τη δομή ενός κυκλώματος, ανάλογα με το *αφαιρετικό επίπεδο* στο οποίο περιγράφεται.

- Η πρώτη HDL γλώσσα (1983)
- Αρχικά μόνο για προσομοίωση Κατόπιν και για σύνθεση
- Αρχικά ιδιοκτησία της Cadence Design Systems
- Verilog-95 ανοιχτό πρότυπο
- Verilog 2001 (IEEE Standard 1364-2001)
- Verilog 2005 (IEEE Standard 1364-2005)
- SystemVerilog

An HDL is NOT a Software Programming Language

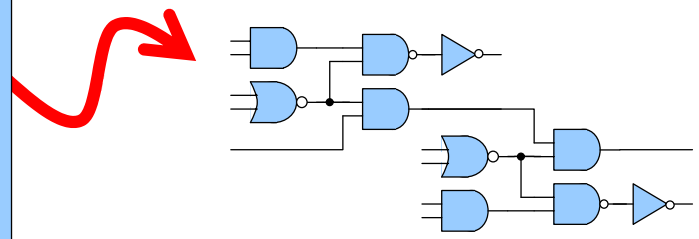
Software Programming Language

- Language which can be translated into machine instructions and then executed on a computer

Hardware Description Language

- Language with syntactic and semantic support for modeling the temporal behavior and spatial structure of hardware

```
module foo(clk,xi,yi,done);
  input [15:0] xi,yi;
  output done;

  always @(posedge clk)
    begin:
      if (!done) begin
        if (x == y) cd <= x;
        else (x > y) x <= x - y;
      end
    end
endmodule
```



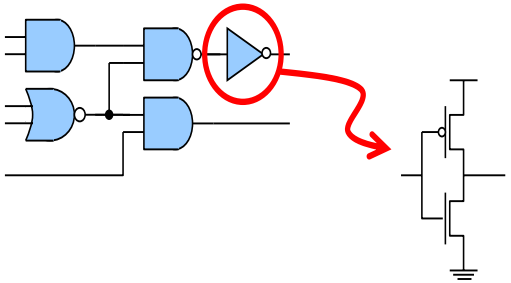
Ιεραρχική σχεδίαση

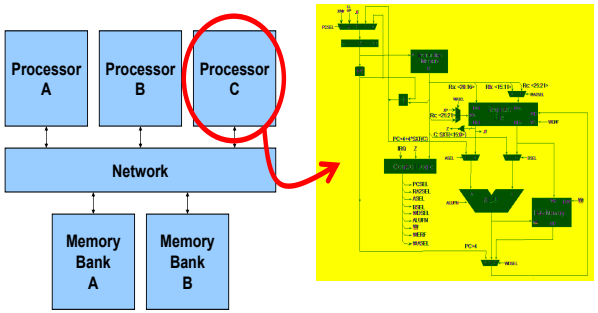
Hierarchy controls complexity

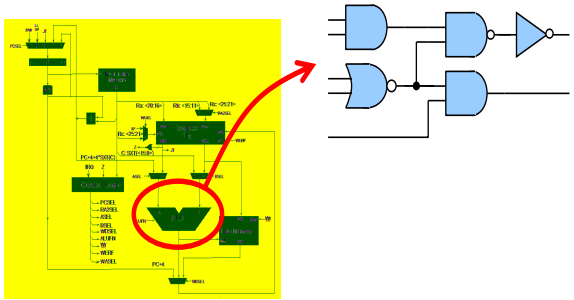
Analogous to the use of function abstraction in SW

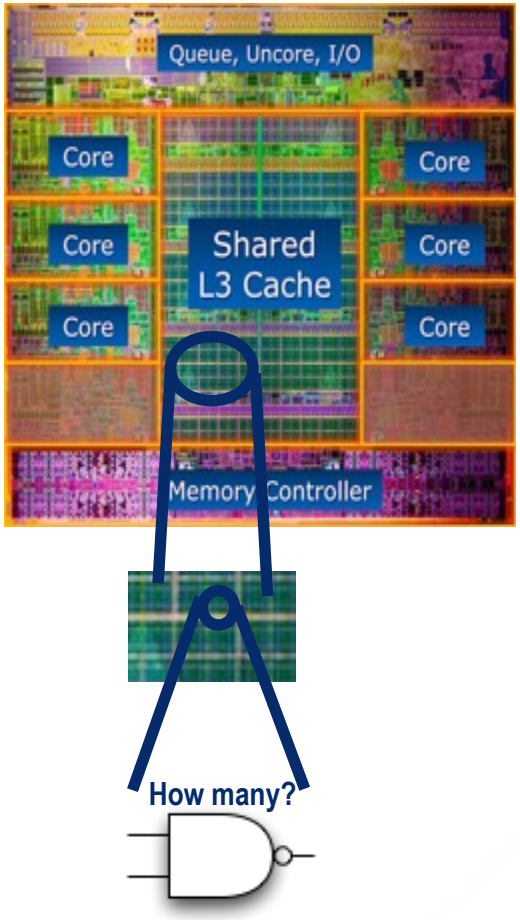
Advantages of HDLs

Allows designers to talk about what the hardware should do without actually designing the hardware itself, or in other words HDLs allow designers to separate behavior from implementation at various levels of abstraction



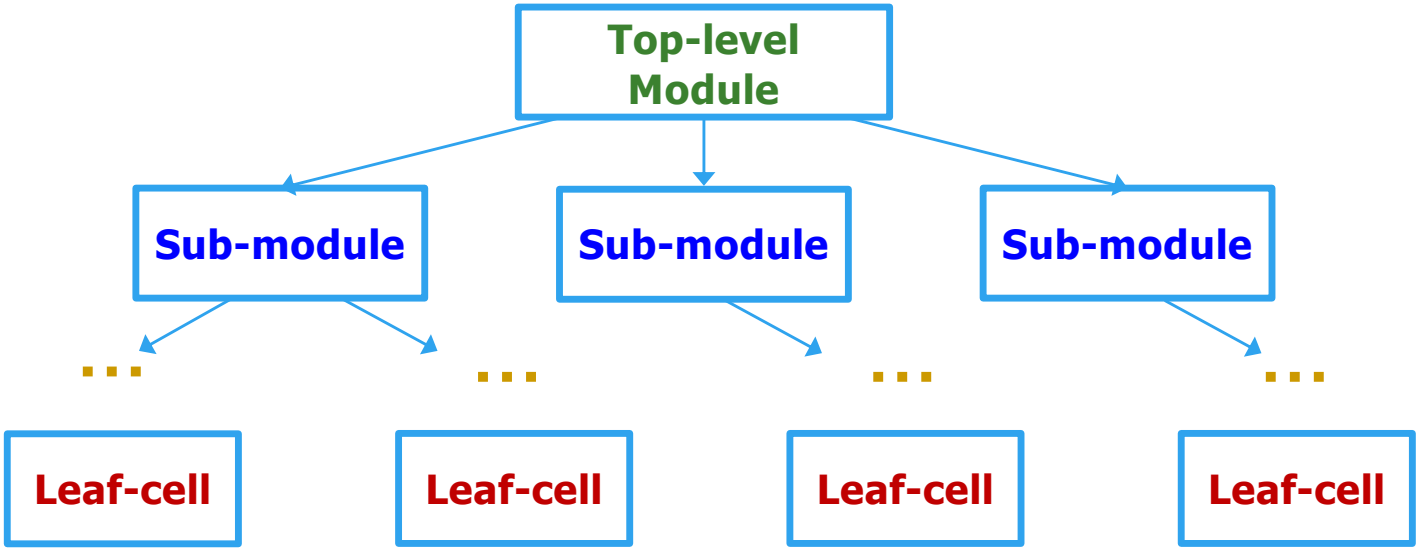






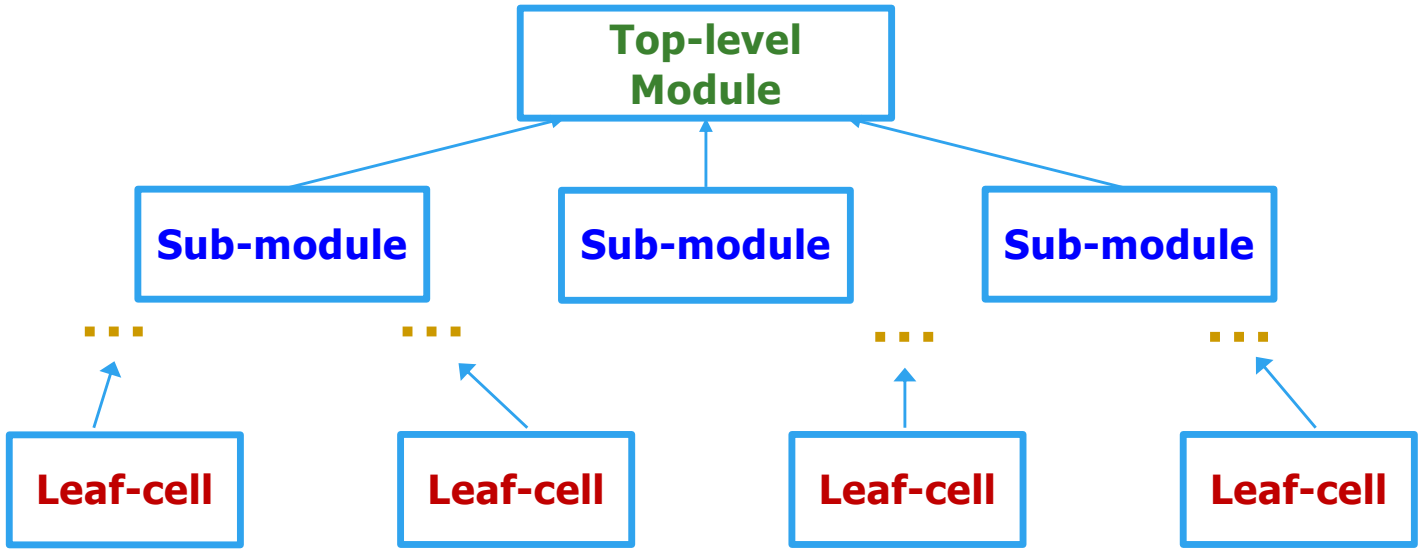
Top-down μεθοδολογία σχεδίασης

- Καθορίζουμε το top-level module και αναγνωρίζουμε τα sub-modules που χρειάζονται για να «χτίσουμε/κατασκευάσουμε» το top-level module
- Υποδιαιρούμε το κάθε sub-module μέχρι να φτάσουμε σε leaf cells
  - Leaf cell: κυκλωματικά στοιχεία που τα οποία δεν μπορούν να υποδιαιρεθούν περαιτέρω (π.χ, *λογικές πύλες, ή πρωτογενή FPGA στοιχεία (LUTs, FFs κτλ)*)



Bottom-up μεθοδολογία σχεδίασης

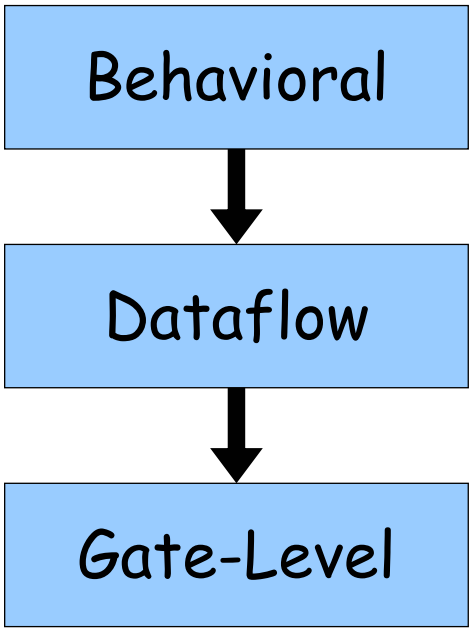
- Αρχικά προσδιορίζουμε τα δομικά στοιχεία που είναι διαθέσιμα σε εμάς
- Δημιουργούμε μεγαλύτερες ενότητες, χρησιμοποιώντας αυτά τα δομικά στοιχεία
- Αυτές οι ενότητες στη συνέχεια χρησιμοποιούνται για ενότητες υψηλότερου επιπέδου έως ότου δημιουργήσουμε τη μονάδα ανώτατου επιπέδου στο σχεδιασμό



Αφαιρετικά επίπεδα περιγραφής

- Το λιγότερο αφαιρετικό επίπεδο είναι το επίπεδο δομής, που περιγράφει ιεραρχικά τη δομή του κυκλώματος χρησιμοποιώντας δομικές μονάδες και ορίζοντας σήματα (καλώδια) για τις συνδέσεις μεταξύ τους.
- Το δεύτερο σε βαθμό αφαιρετικό επίπεδο είναι το επίπεδο ροής δεδομένων, που περιγράφει συνήθως πιο απλές σχέσεις (λογικές πράξεις) μεταξύ σημάτων, που θεωρούνται ότι εκτελούνται ταυτόχρονα (όπως και πρέπει να είναι η λειτουργία ενός κυκλώματος).
- Το περισσότερο αφαιρετικό επίπεδο είναι το επίπεδο συμπεριφοράς, που περιγράφει τη λειτουργία ενός κυκλώματος με τη μορφή πολύπλοκων τελεστών, κυρίως αριθμητικών και λογικών, που τοποθετούνται σε εντολές της μορφής if-else, for-loop κ.λπ. όπως σε μια γλώσσα προγραμματισμού.

3 Common Abstraction Levels



Module's high-level algorithm is implemented with little concern for the actual hardware

Module is implemented by specifying how data flows between registers

Module is implemented in terms of concrete logic gates (AND, OR, NOT) and their interconnections

DESIGN FLOW - 1

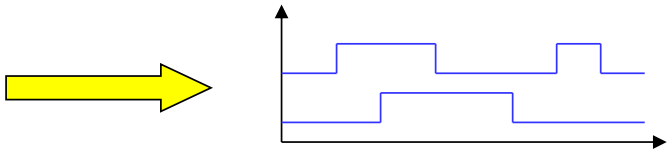
Σχεδιάστε και εφαρμόστε μια απλή μονάδα που επιτρέπει την επιτάχυνση της κρυπτογράφησης με παρόμοιο RC5 κρυπτογράφηση με σταθερό κλειδί σε 8031 μικροελεγκτή. Αυτή η μονάδα πρέπει να είναι σε θέση να εκτελέσει έναν αλγόριθμο κρυπτογράφησης από μόνη της, εκτελώντας 32 γύρους

Specification

HDL description

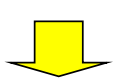
```
Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

module RC5_core ( input clock, input  reset, input [31:0] key_input,
data_input, input [31:0], output [31:0] data_output);

….

endmodule;
```

Functional simulation

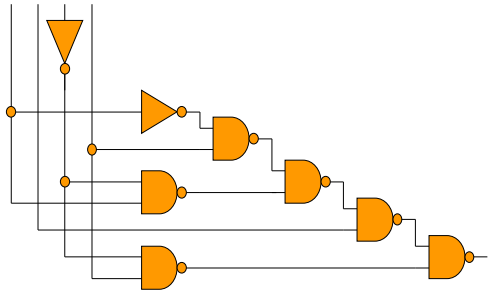


Synthesis

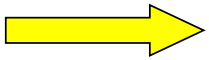
DESIGN FLOW - 2

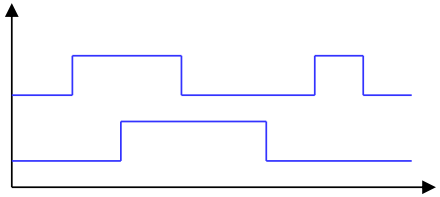


Synthesis

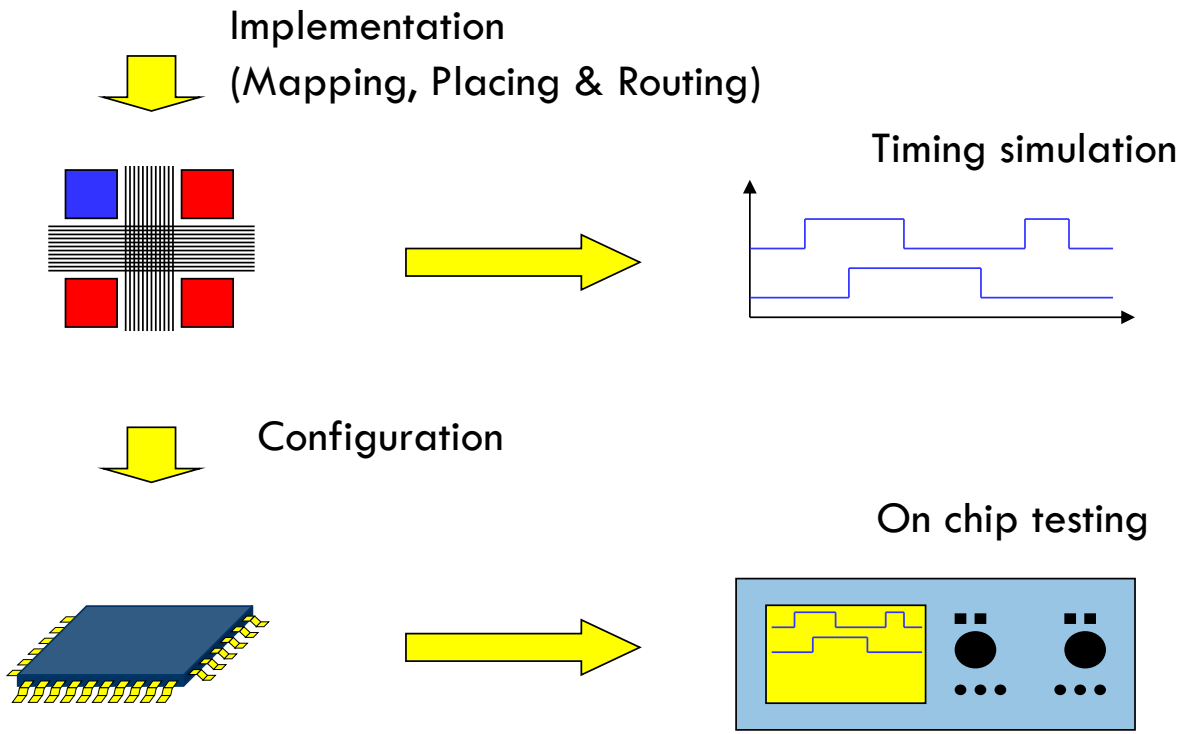


Post-synthesis simulation



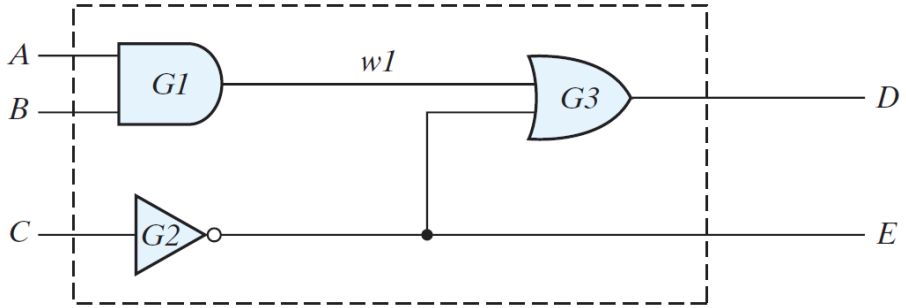


DESIGN FLOW - 3



ΠΕΡΙΓΡΑΦΗ VERILOG ΑΠΛΟΥ ΚΥΚΛΩΜΑΤΟΣ

```
// Verilog model of circuit of Figure 3.35.
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 (G1)
    not G2 (E, C);
    or G3 (D, w1, E);
endmodule
```



Λέξεις κλειδιά της VERILOG

Η verilog HDL έχει κωδικές λέξεις που ορίζουν τις δομές

Της γλώσσας. Οι βασικές λέξεις κλειδιά, που είναι πεζά γράμματα, είναι οι εξής:

- **Module:** *ονομασία μονάδας και (κατάλογος θυρών)*
- **Output, input, tri:** *ορίζουν τι είναι κάθε θύρα*
- **Wire:** *ορίζουν τα καλώδια διασύνδεσης*
- Βασικές πύλες**: and, or, not, nand, nor, xor, xnor, buf** κ.λπ. **:** *Η περιγραφή του κυκλώματος δίνει τη διασύνδεση των πυλών.*
- **Endmodule:** *τερματισμός περιγραφής μονάδας*

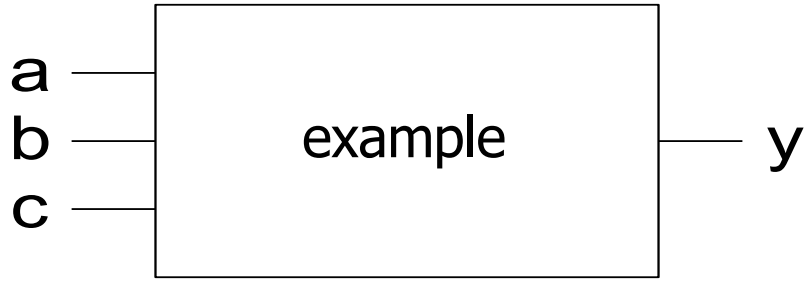
- Επίσης κείμενο μετά από ‘ // ’ αποτελεί *σχόλιο*.
- Κάθε εντολή να τερματίζεται με ‘ **;** ’

Defining a module in Verilog

- A module is the main building block in verilog

- We first need to define:
  - Name of the module
  - Directions of its ports (e.G., Input, output)
  - Names of its ports
- Then:
  - Describe the functionality of the module

Implementing a module in verilog



name of module

Port list (inputs and outputs)

```
module example (a, b, c, y);
        input a;
        input b;
        input c;
        output y;

// here comes the circuit description

endmodule
```

ports have a declared type

a module definition

Module interface styles in Verilog

- **THE FOLLOWING TWO CODES ARE FUNCTIONALLY IDENTICAL**

```
module test ( a, b, y );
        input a;
        input b;
        output y;

endmodule
```

```
module test ( input a,
              input b,
              output y );

endmodule
```

port name and direction declaration
can be combined

Multi-bit input/output?

- **You can also define multi-bit input/output (bus)**
  - [range_end : range_start]
  - **Number of bits:** range_end – range_start + 1
- **Example:**

```
input  [31:0] a;   // a[31], a[30] .. a[0]
output [15:8] b1;  // b1[15], b1[14] .. b1[8]
output [7:0]  b2;  // b2[7], b2[6] .. b2[0]
input         c;   // single signal
```

- **A** represents a 32-bit value, so we prefer to define it as: `[31:0] a`
- It is preferred over `[0:31] a` which resembles *array* definition
- It is good practice to be consistent with the representation of multi-bit signals, i.E., Always [31:0] or always [0:31]

Manipulating bits

- BIT SLICING
- CONCATENATION
- DUPLICATION

Basic syntax

- Verilog is case sensitive
  - `Somename` and `somename` are not the same!
- Names cannot start with numbers:
  - `2good` is not a valid name
- Whitespaces are ignored

```
// Single line comments start with a //

/* Multiline comments
   are defined like this */
```

Two main styles of HDL implementation

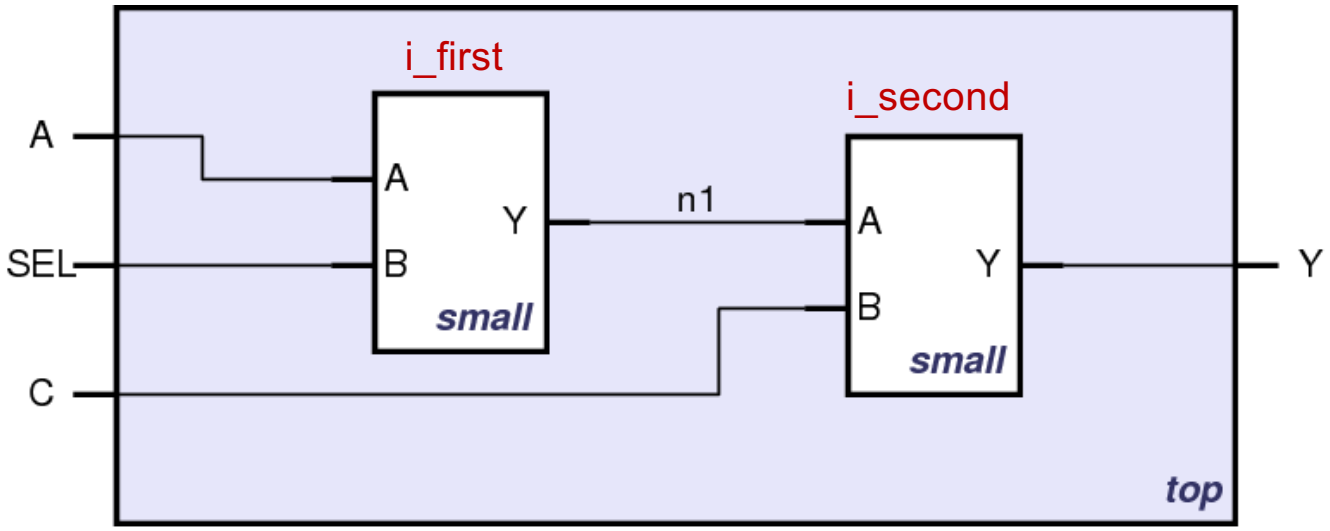
- **Structural (gate-level)**
  - The module body contains gate-level description of the circuit
  - Describe how modules are interconnected
  - Each module contains other modules (instances)
  - … And interconnections between these modules
  - Describes a hierarchy

- **Behavioral**
  - The module body contains functional description of the circuit
  - Contains logical and mathematical operators
  - **Level of abstraction is higher than gate-level**
    - Many possible gate-level realizations of a behavioral description

- **Practical circuits use a combination of both**

STRUCTURAL (GATE-LEVEL) HDL

Structural HDL: instantiating a module

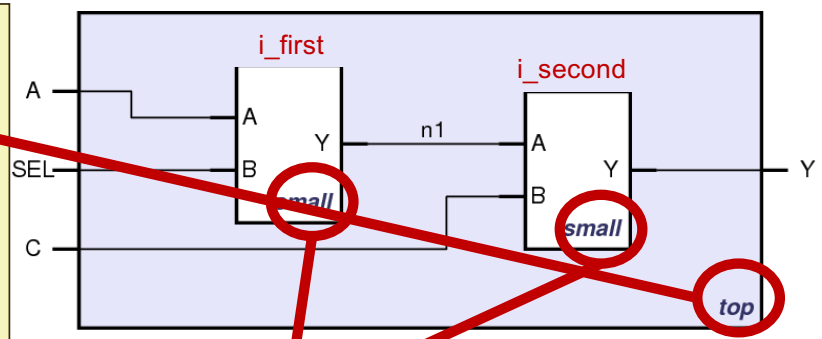


Schematic of module "top" that is built from two instances of module "small"

Structural HDL example

- **Module definitions in verilog**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;









endmodule
```

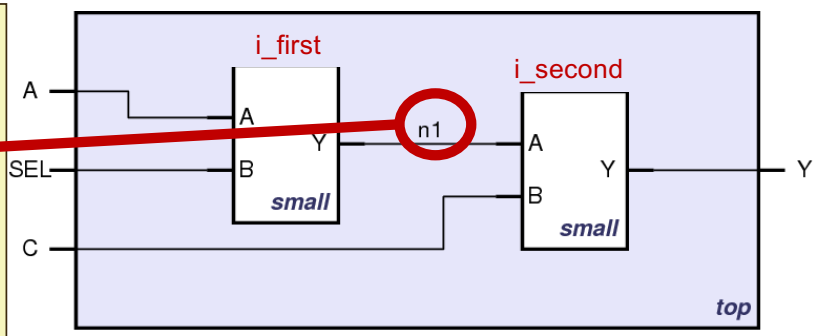



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL example

- **Defining wires (module interconnections)**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;




endmodule
```

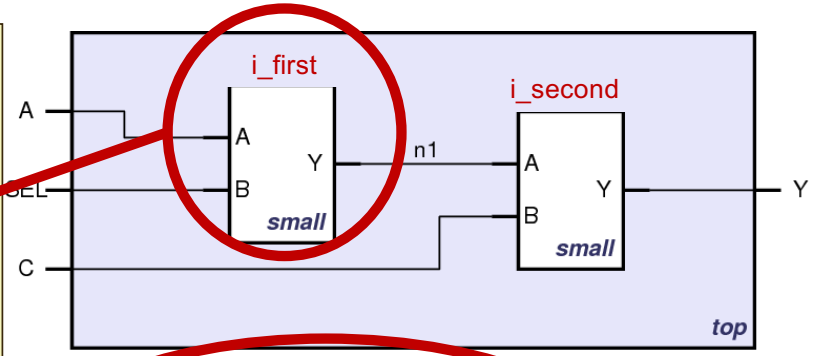



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL example

- **The first instantiation of the "small" module**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );




endmodule
```

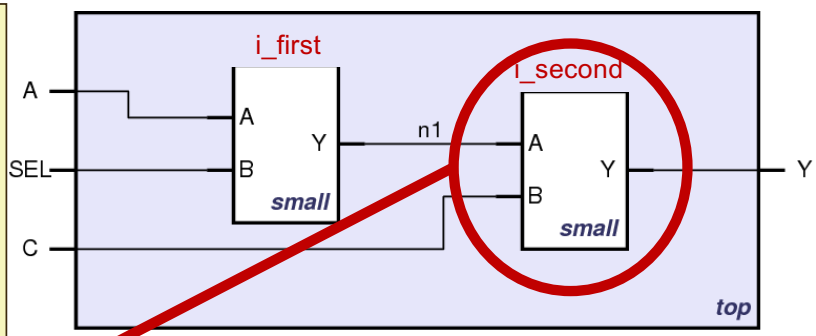



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL example

- **The second instantiation of the "small" module**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );

// instantiate small second time
small i_second ( .A(n1),
           .B(C),
           .Y(Y) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL example

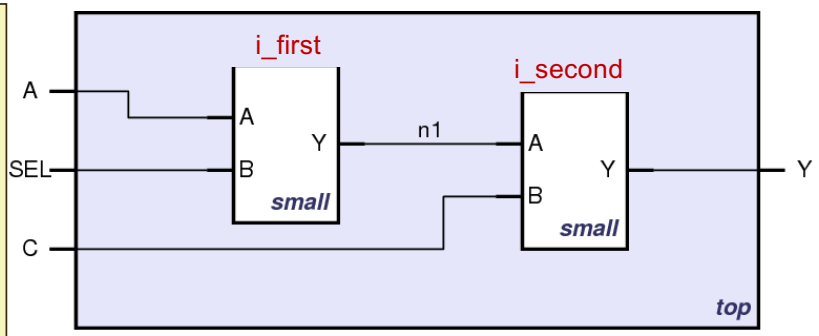
- **Short form of module instantiation**

Port Connection by ordered list

v.s

Port Connection by name

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// alternative
small i_first ( A, SEL, n1 );

/* Shorter instantiation,
   pin order very important */

// any pin order, safer choice
small i_second ( .B(C),
                 .Y(Y),
                 .A(n1) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Short form is not good practice as it reduces code maintainability

Primitive logic gates in verilog

- Η Verilog ορίζει μερικές βασικές πύλες λογικής ως μέρος της γλώσσας. Οι πρωτογενείς (primitive gates) πύλες που χρησιμοποιούνται στη verilog είναι οι NOT, AND, OR, NAND, NOR, XOR, XNOR.
- Η Verilog περιλαμβάνει προκαθορισμένες μονάδες (modules) που εφαρμόζουν βασικές πύλες λογικής. Αυτές οι πύλες επιτρέπουν την περιγραφή της δομής ενός κυκλώματος χρησιμοποιώντας δηλώσεις instantiation πύλης της φόρμας:

Gate_name [instance_name] (output_port, input_port {, input_port});

- Προσοχή! Το output port δηλώνεται πρώτο στην port first και ακολουθείται από τα input ports.

Gate-level primitives

- Verilog provides the following:

And	nand		logical and/nand
Or		nor	logical OR/NOR
Xor	xnor		logical XOR/XNOR
Buf	not		buffer/inverter
Bufif0	notif0		tristate with low enable
Bifif1	notif1		tristate with high enable


That's all Folks!