# Σχεδιασμός Ολοκληρωμένων Κυκλωμάτων VLSI – II

# Tree Adders

- Για τους αθροιστές μεγάλου μήκους (N >16 bits), η καθυστέρηση εξαρτάται από την καθυστέρηση του διάδοσης του κρατουμένου διαμέσου των σταδίων πρόβλεψης
- Μπορεί να κατασκευαστεί ένα πολυεπίπεδο δένδρο δομών πρόβλεψης => η καθυστέρηση διάδοσης του κρατούμενου να είναι $\log N$
- Τέτοιοι αθροιστές αναφέρονται συνήθως ως αθροιστές δένδρου
- Υπάρχουν πολλοί τρόποι κατασκευής του δένδρου πρόβλεψης με συμβιβασμούς ανάμεσα στο
  - πλήθος των επιπέδων λογικής
  - πλήθος των λογικών πυλών
  - μέγιστο βαθμό οδήγησης εξόδου κάθε πύλης και
  - ποσότητα καλωδίωσης μεταξύ των σταδίων
- Τρία θεμελιώδη δένδρα είναι οι αρχιτεκτονικές: Brent-Kung, Sklansky και Kogge-Stone

# Βασικά δομικά στοιχεία

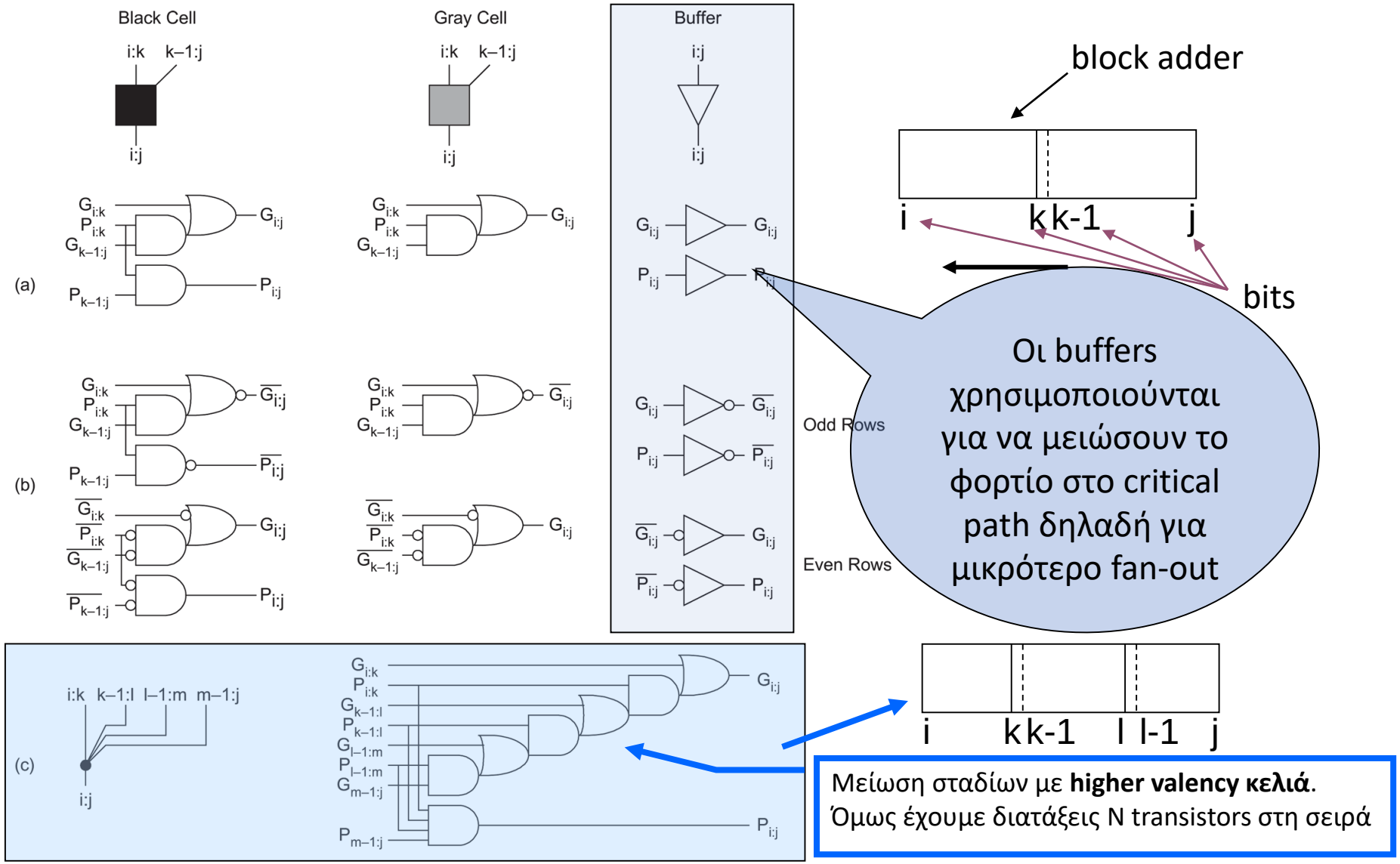
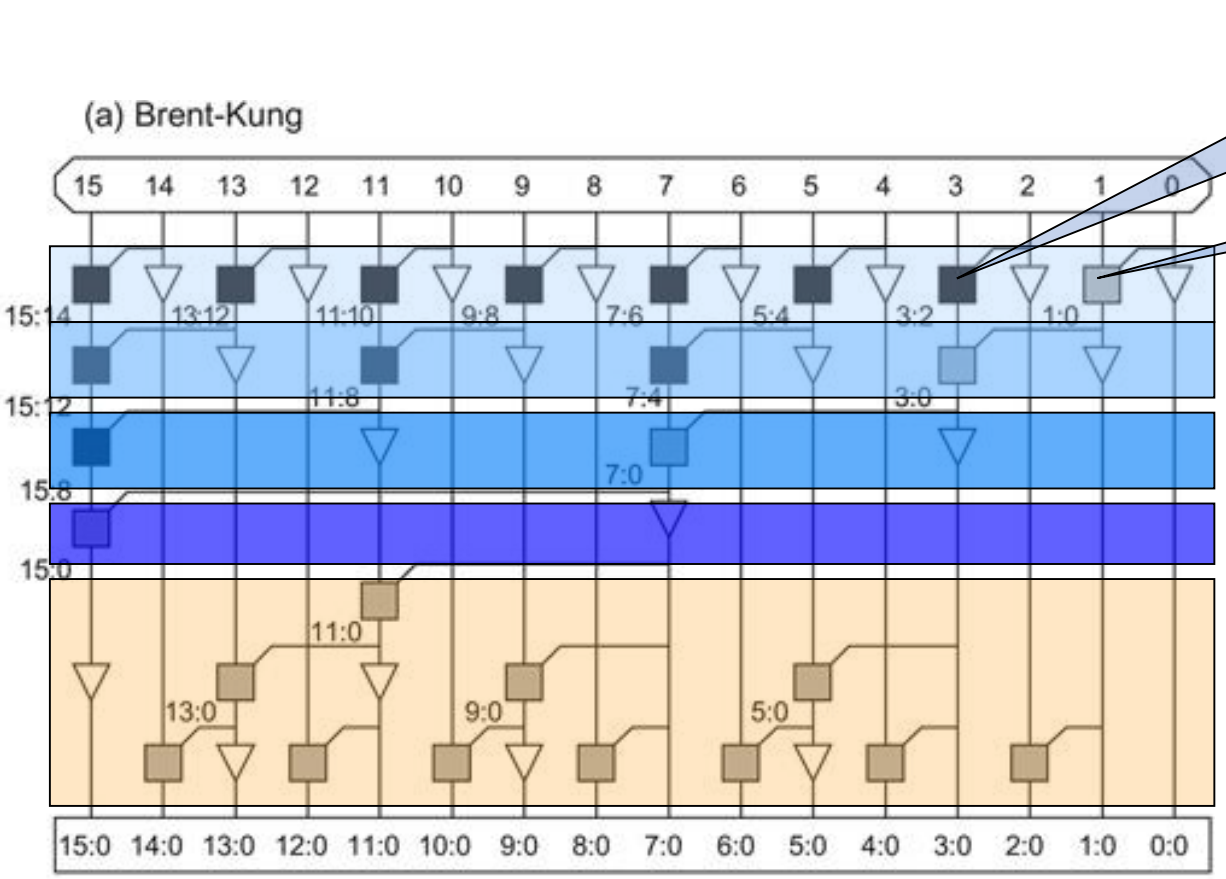



**FIG 10.17** Group PG cells

# Brent Kung (1/3)




Black cell

Grey cell

Υπολογίζονται αρχικά τα PG για κάθε block ανα 2 bits

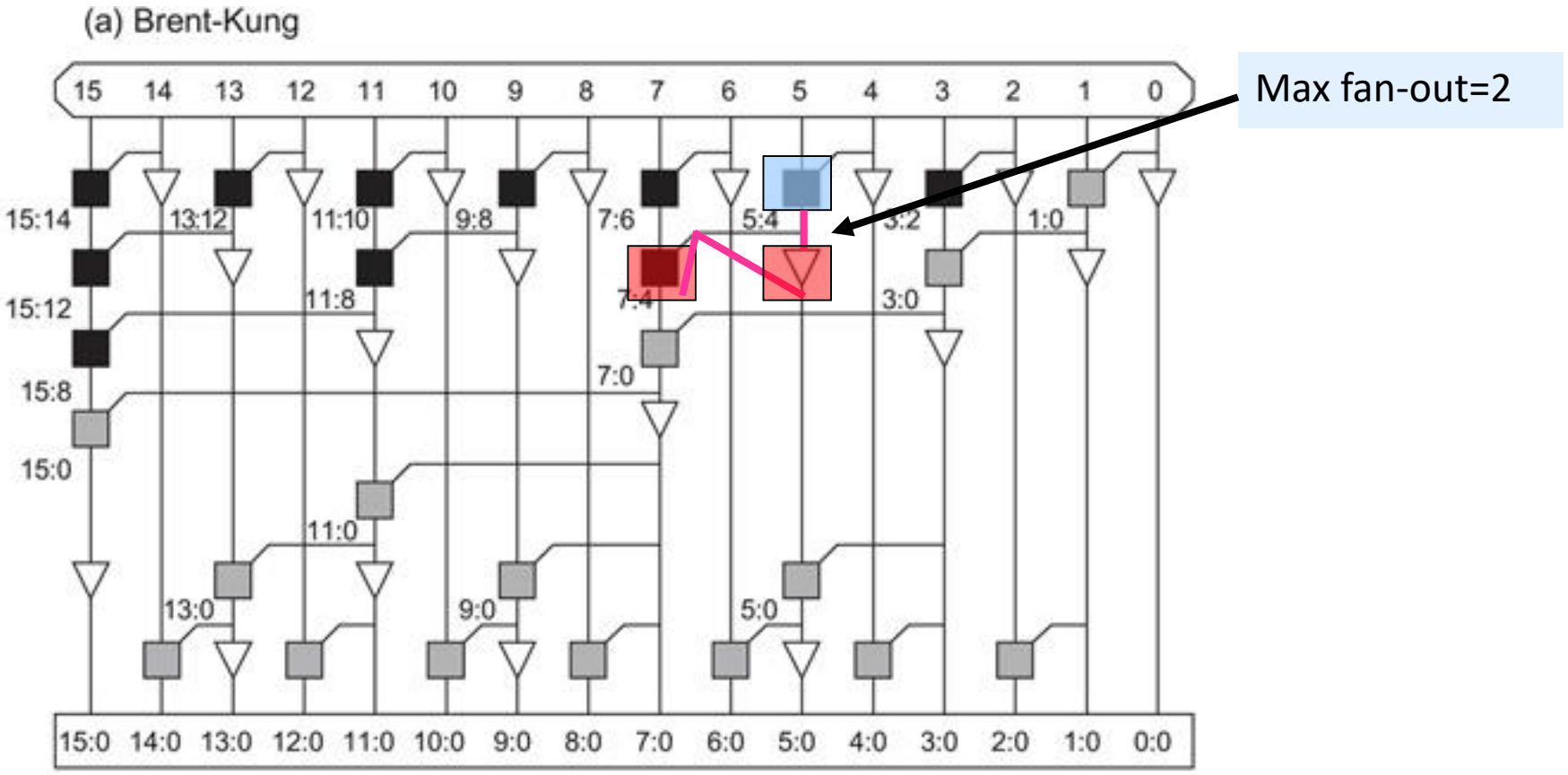
Υπολογισμός των PG για κάθε block ανα 4 bits

Υπολογισμός των PG για κάθε block ανα 8 bits

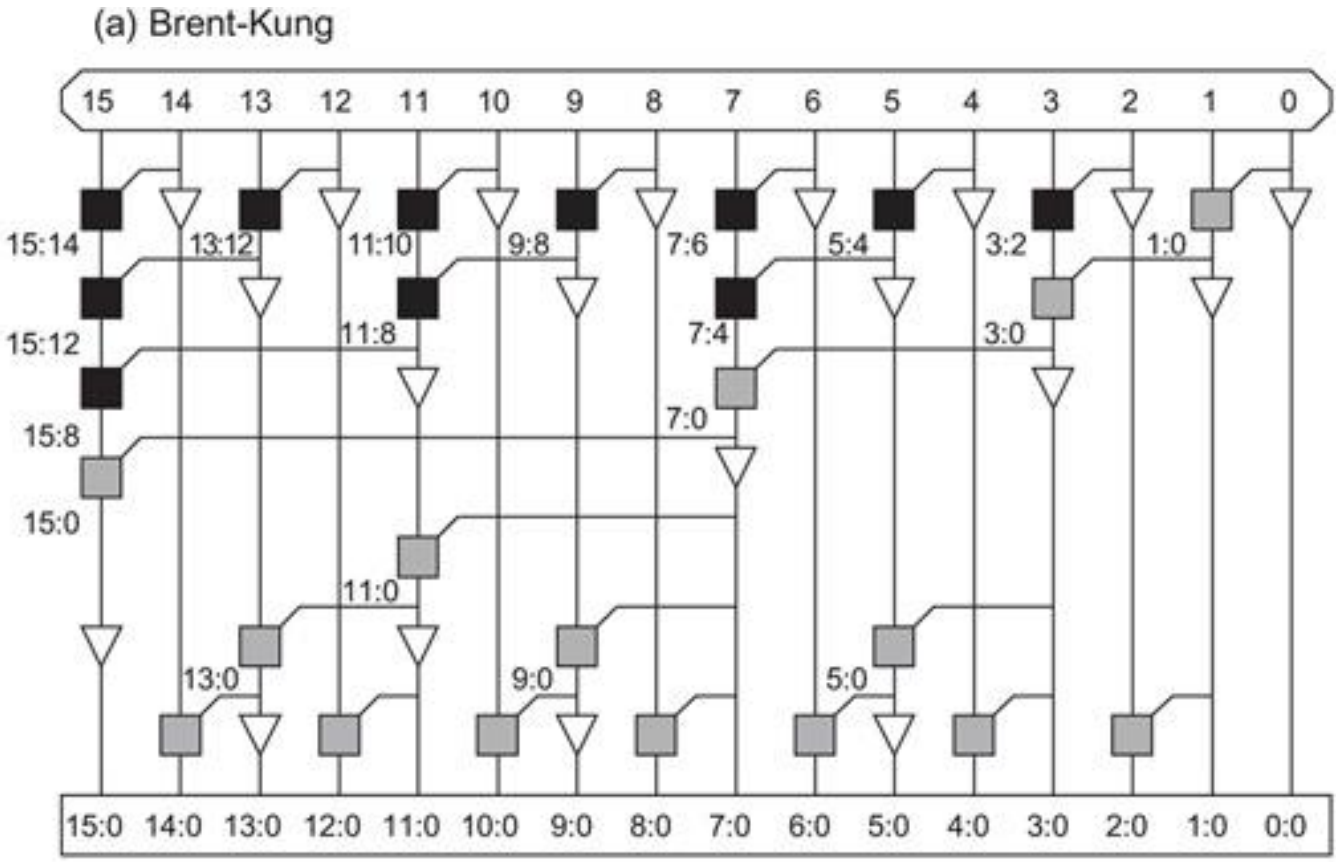
Υπολογισμός των PG για κάθε block ανα 16 bits

Υπολογισμός σε μορφή δέντρου των απαραίτητων G των ενδιάμεσων bit για το τελικό αποτέλεσμα

# Brent Kung (2/3)



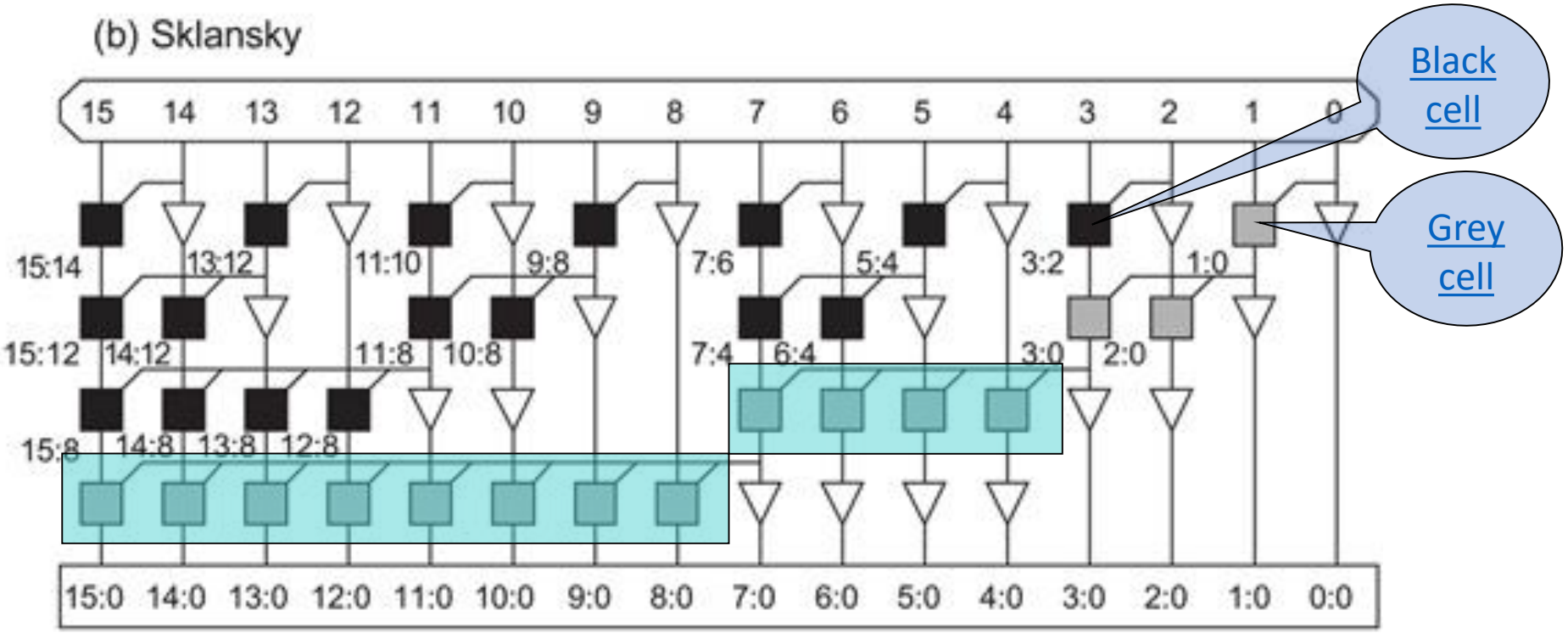
# Brent Kung (3/3)




- Δεν καταλαμβάνει μεγάλη επιφάνεια
- Δεν έχει πρόβλημα πυκνότητας καλωδίων

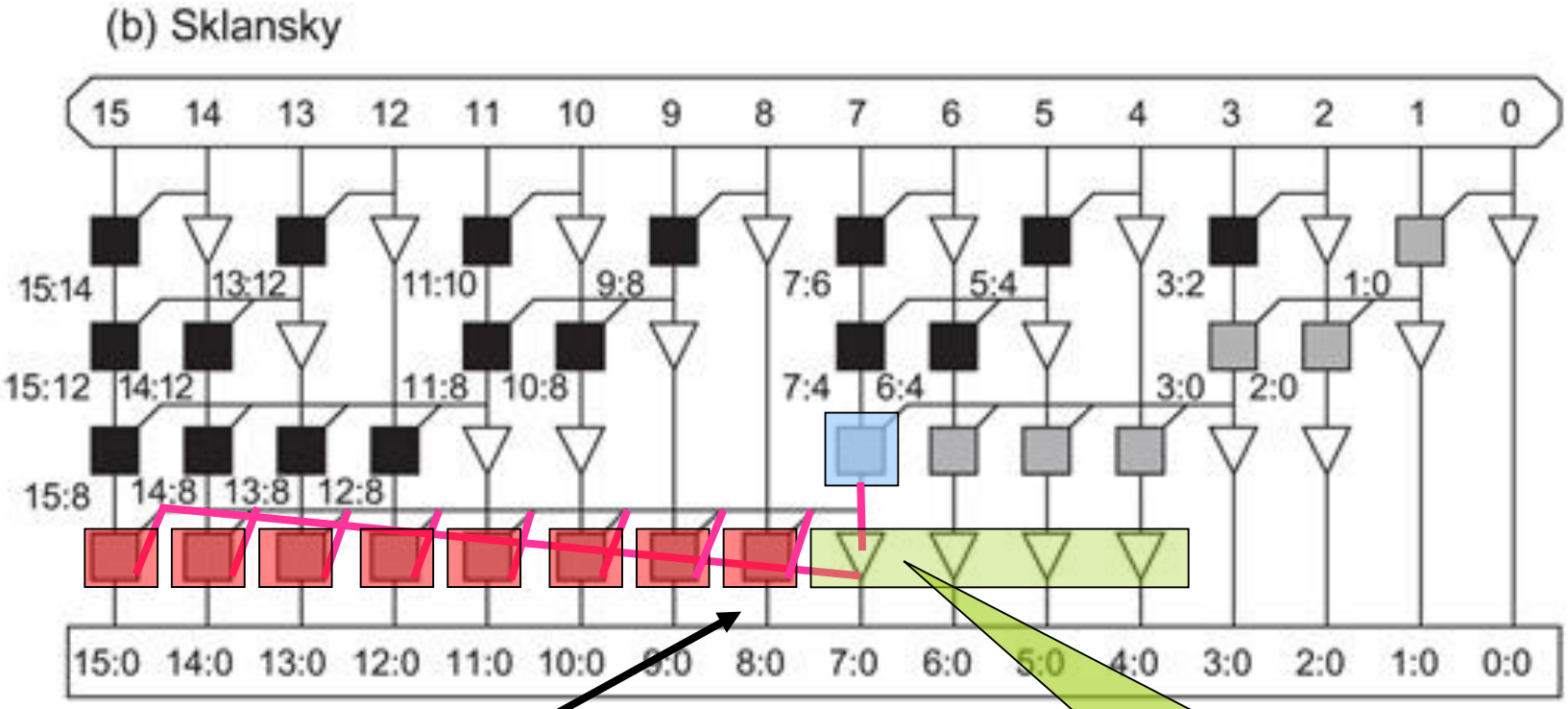
- Απαιτεί $2\log_2 N - 1$ επίπεδα για τον τελικό υπολογισμό – δεν είναι η καλύτερη επιλογή για την καθυστέρηση

# Sklansky (1/4)




➢ Υπολογίζει πολλά μαζί Generate μειώνοντας έτσι την καθυστέρηση σε σχέση με τον Brent-Kung tree adder

# Sklansky (2/4)

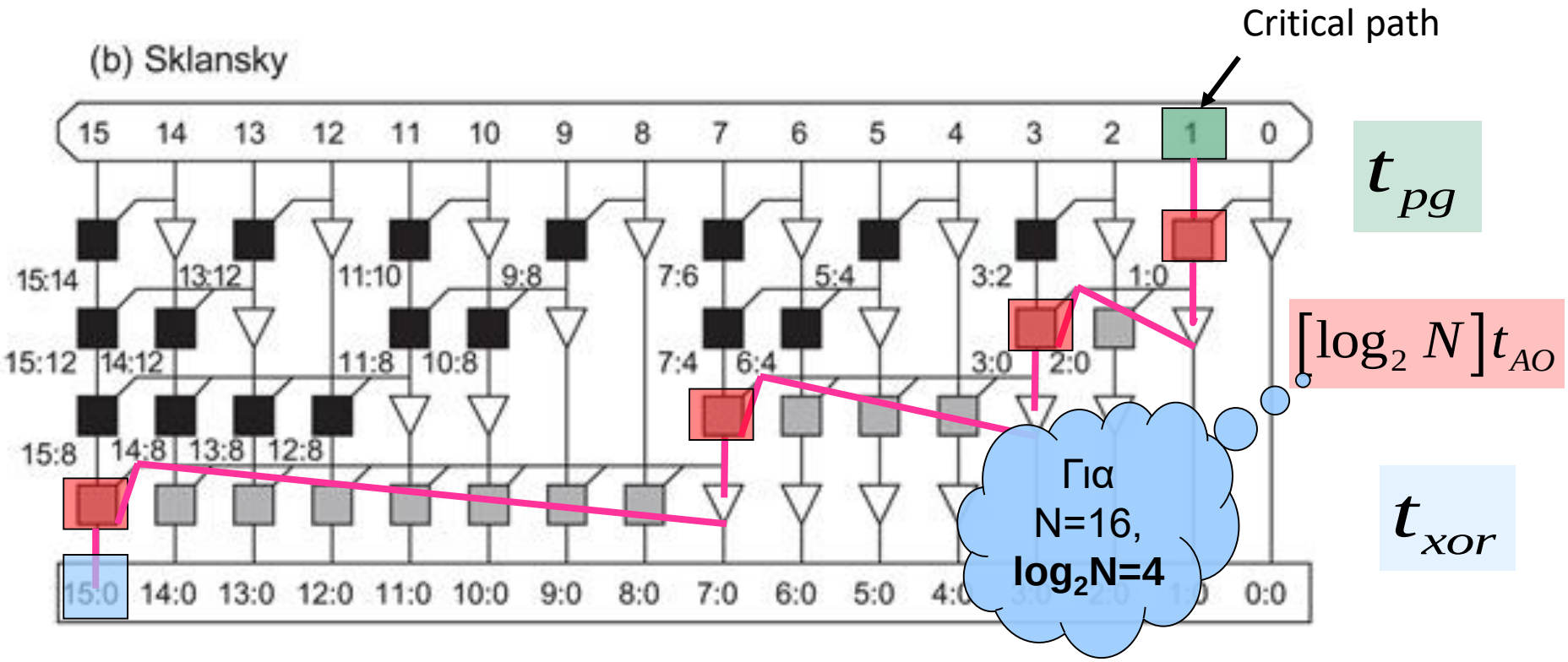



Max fan-out=8

Ο βαθμός οδήγησης εξόδου x2 σε κάθε στάδιο [8, 4, 2, 1]

Χρήση Buffers λόγω μεγάλου fan-out. Αλλιώς θα έχει μεγάλη καθυστέρηση

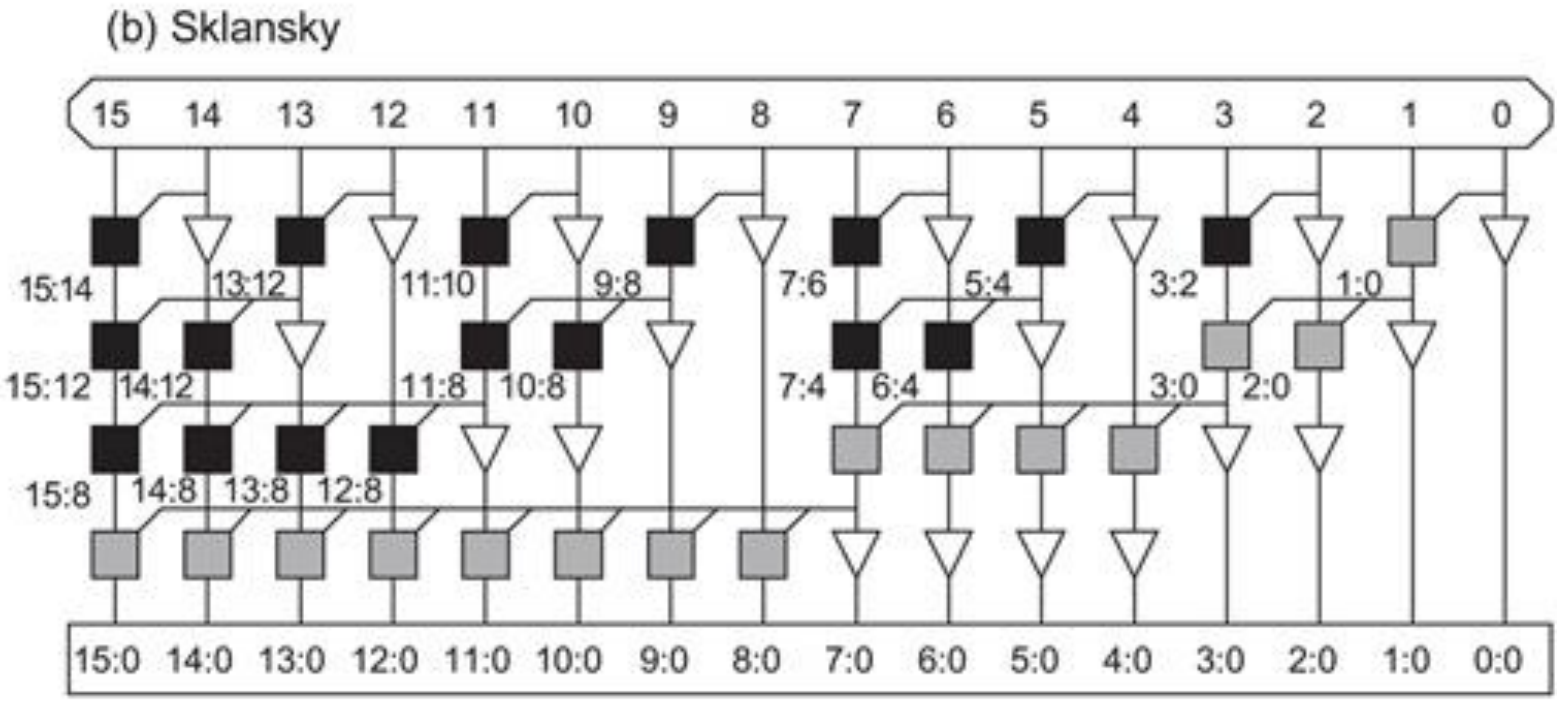
# Sklansky (3/4)




Critical path delay του Sklansky (ισχύει και για Kogge-Stone tree adder):

$$t_{tree} = t_{pg} + \lceil \log_2 N \rceil t_{AO} + t_{xor}$$

# Sklansky (4/4)

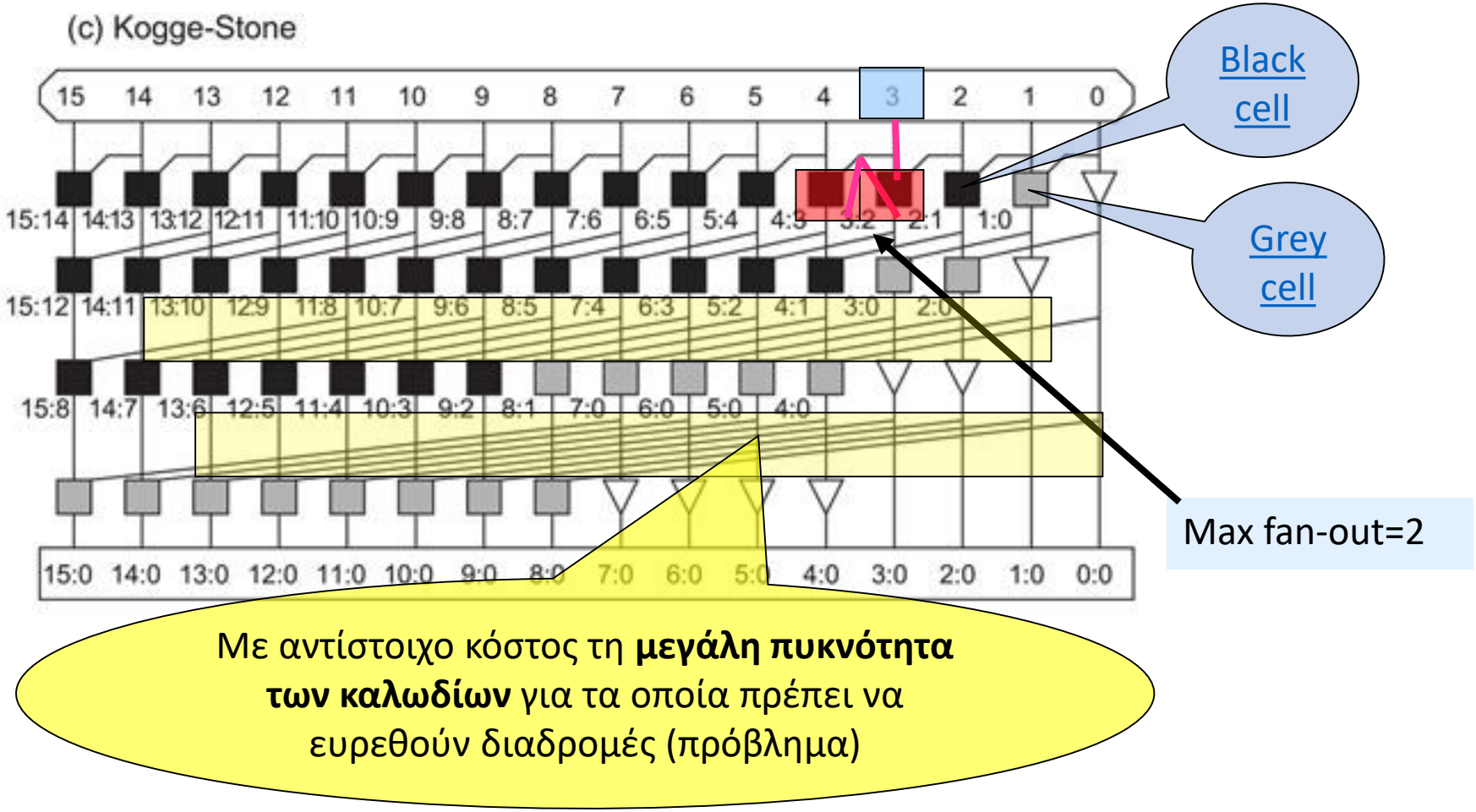



- Απαιτεί $\log_2 N$ επίπεδα για τον τελικό υπολογισμό (μικρότερο delay από τον Brent-Kung)

- Καταλαμβάνει μεγαλύτερη επιφάνεια σε σχέση με τον Brent-Kung

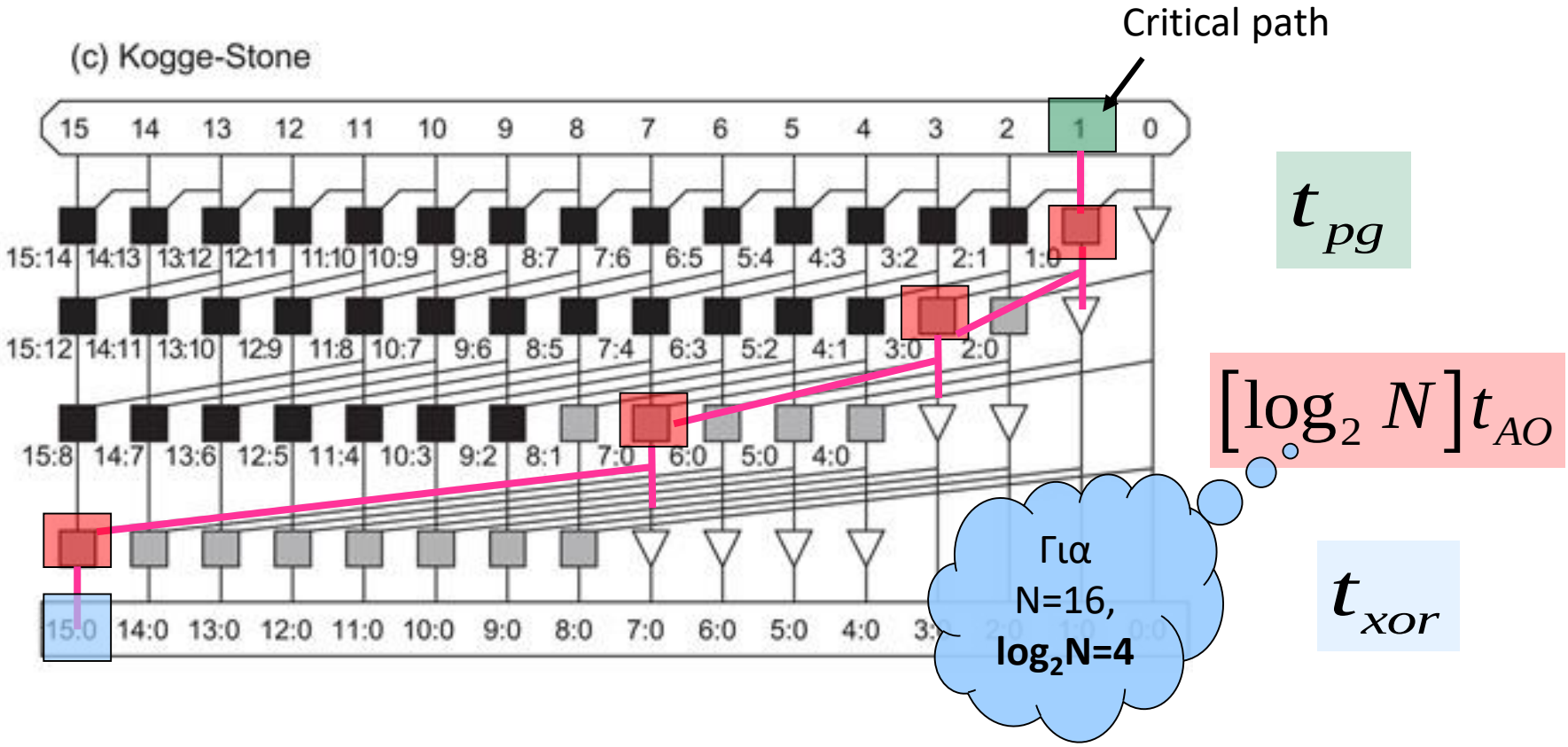
- Δεν έχει πρόβλημα πυκνότητας καλωδίων

# Kogge-Stone (1/3)




Μειώνει το max fan-out σε 2 (μία από τις διαδρομές με max fan-out)

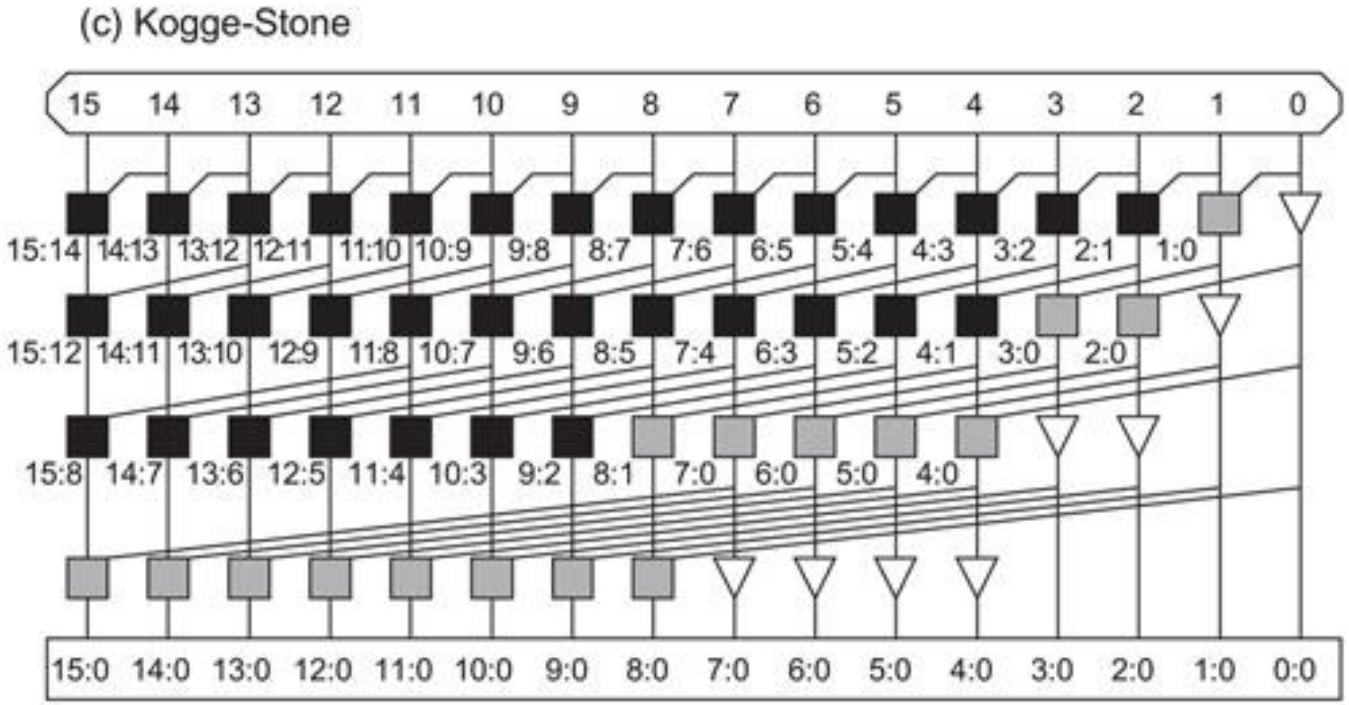
# Kogge-Stone (2/3)




Critical path delay του Kogge-Stone tree adder (ισχύει και για Slansky):

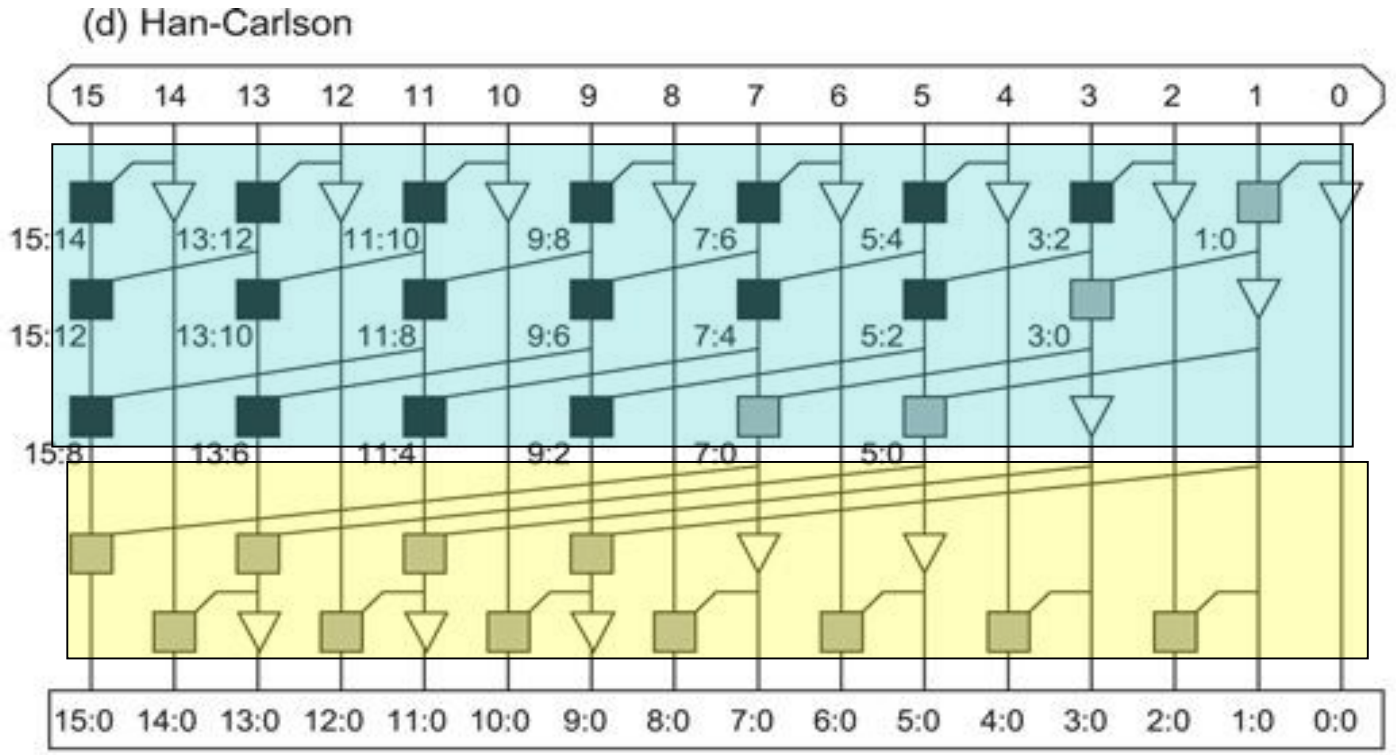
$$t_{tree} = t_{pg} + \lceil \log_2 N \rceil t_{AO} + t_{xor}$$

# Kogge-Stone (3/3)




- Απαιτεί $\log_2 N$ επίπεδα για τον τελικό υπολογισμό
- Καταλαμβάνει μεγάλη επιφάνεια (μεγάλος αριθμός από black cells)
- Έχει μεγάλο αριθμό καλωδίων για τα οποία πρέπει να ευρεθούν διαδρομές (πρόβλημα)

# Han – Carlson (1/4)

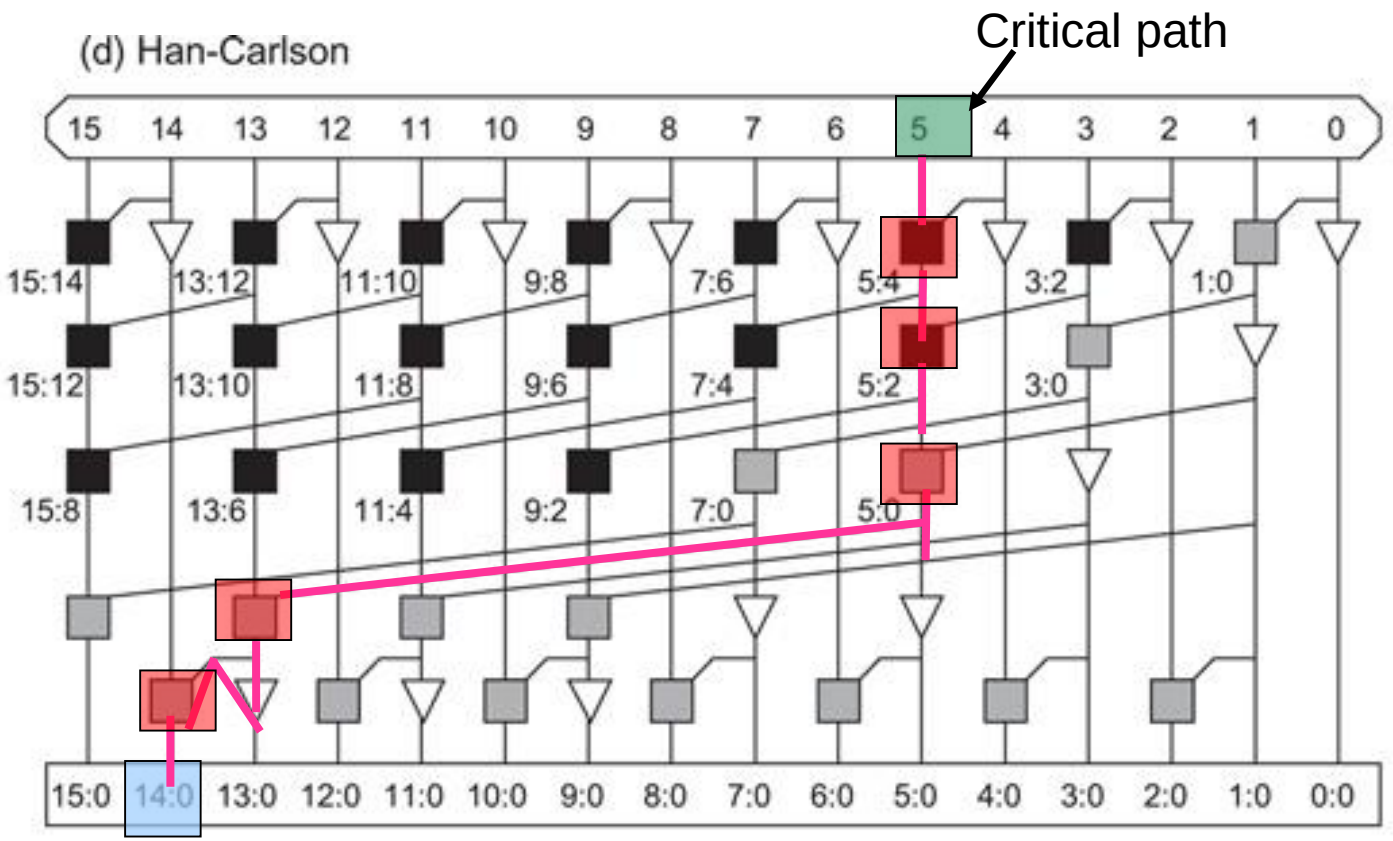



- Οι Han-Carlson trees είναι μία οικογένεια από δίκτυα μεταξύ Kogge-Stone και Brent-Kung

- Στο διάγραμμα χρησιμοποιεί την τεχνική Brent-Kung

- Μείωση επιπέδων με χρήση τεχνικής (μακριών καλωδίων) του Kogge-Stone

# Han – Carlson (2/4)

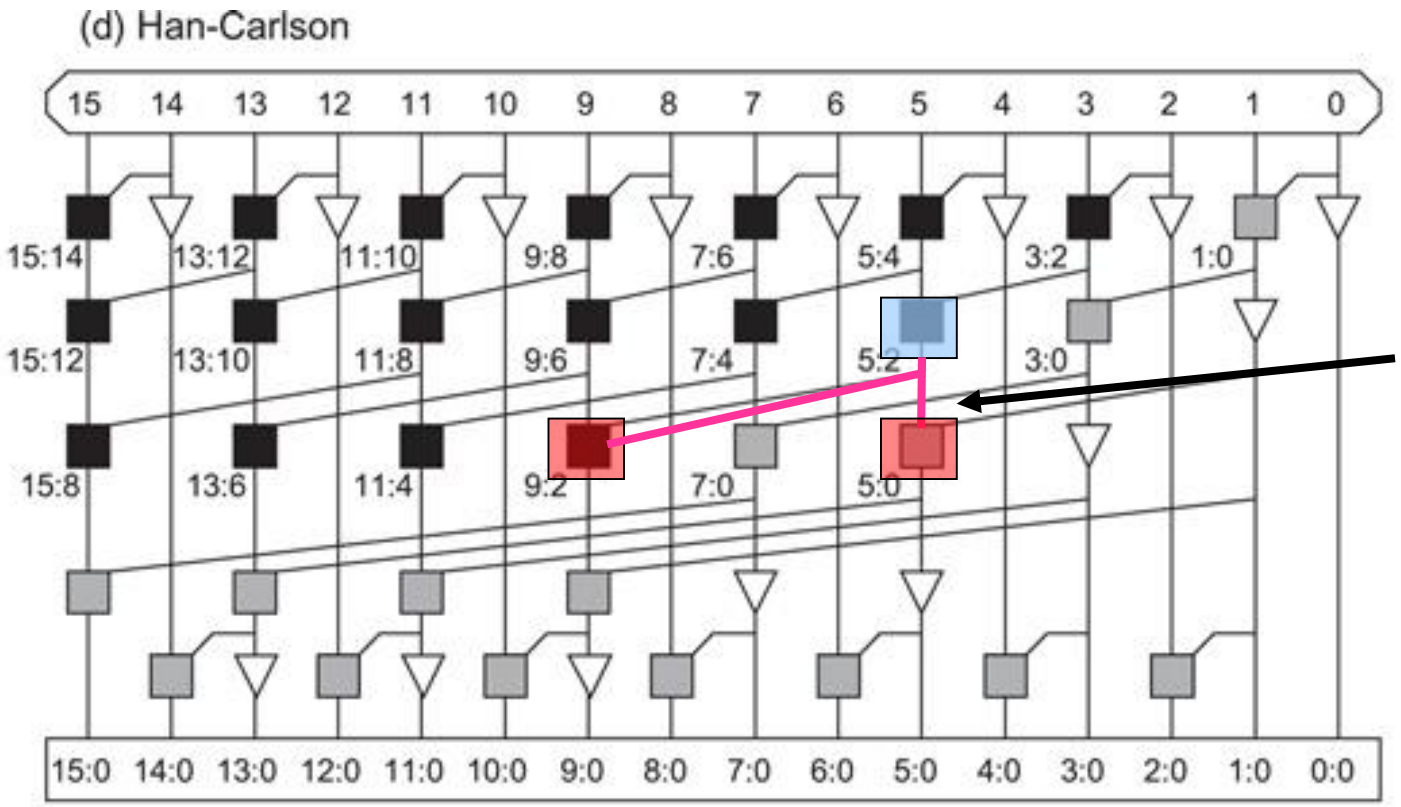



$t_{pg}$

$5t_{AO}$

$t_{xor}$

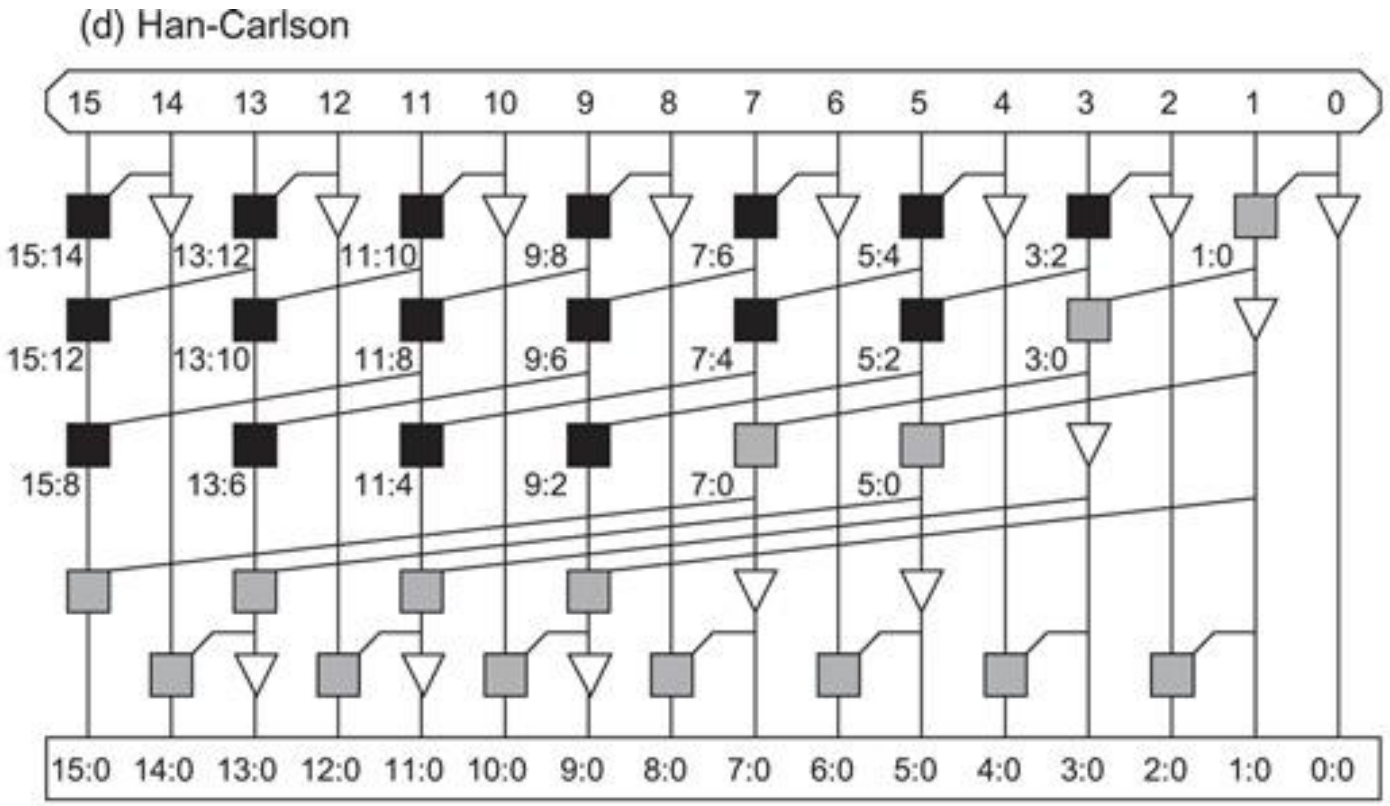
# Han – Carlson (3/4)




- Max fan-out=2

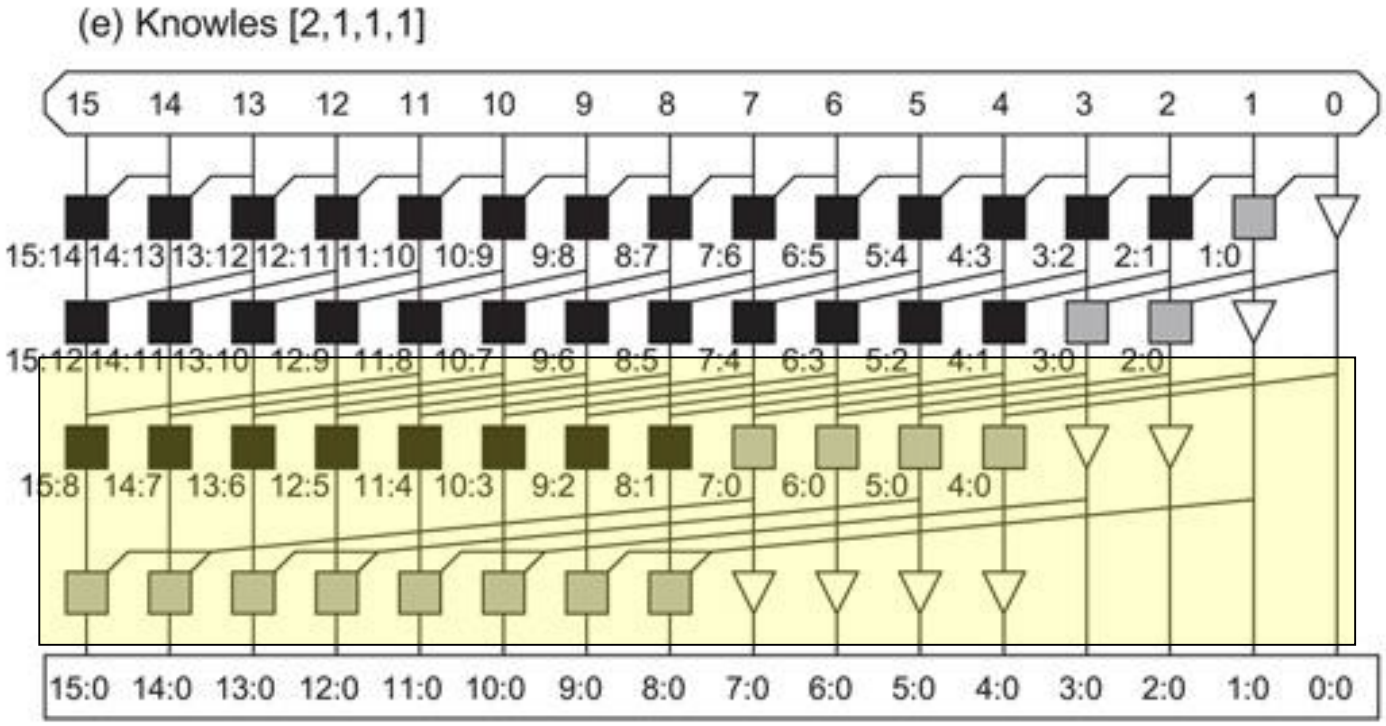
- Μία από τις διαδρομές με max fan-out

# Han – Carlson (4/4)




- Έχει ικανοποιητική καθυστέρηση
- Δεν καταλαμβάνει μεγάλη επιφάνεια (αριθμός πυλών)
- Δεν έχει μεγάλη πυκνότητα καλωδίων

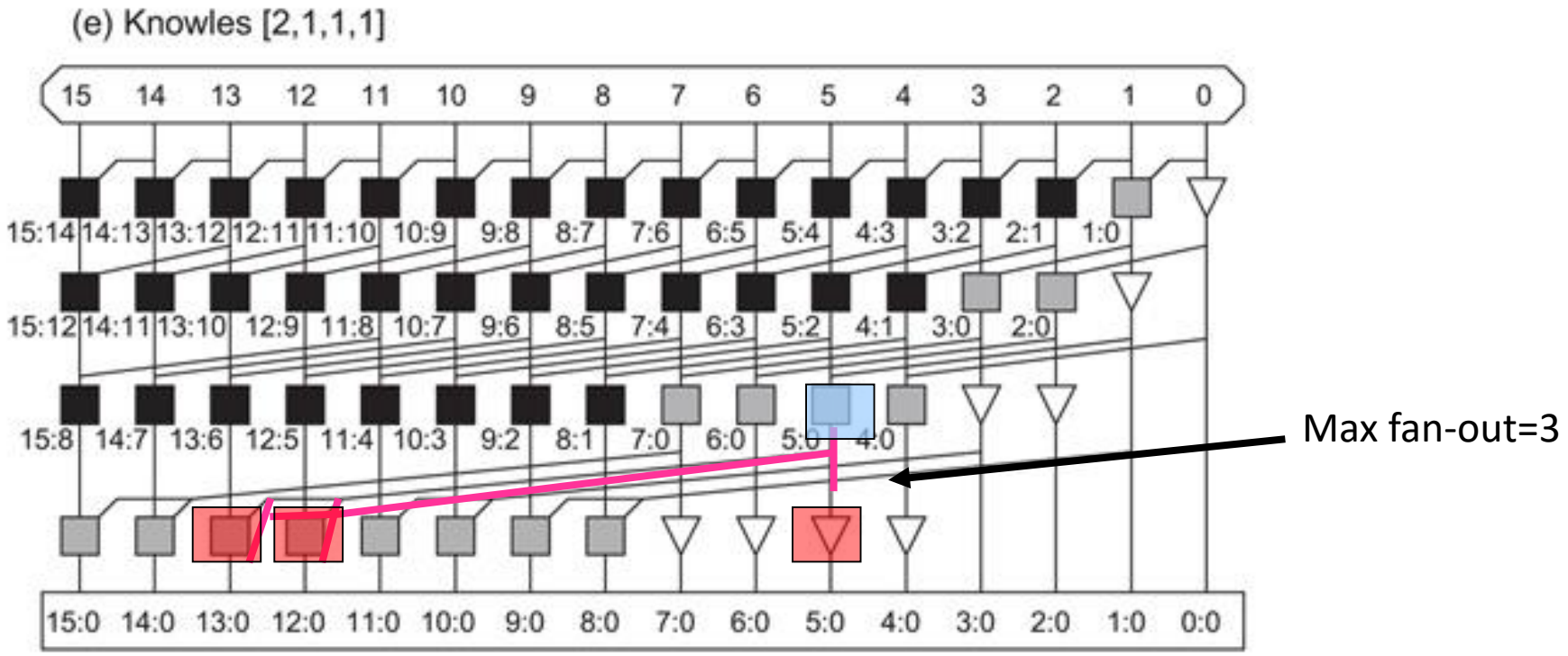
# Knowles (1/4)




- Είναι συνδυασμός Kogge-Stone και Sklansky.

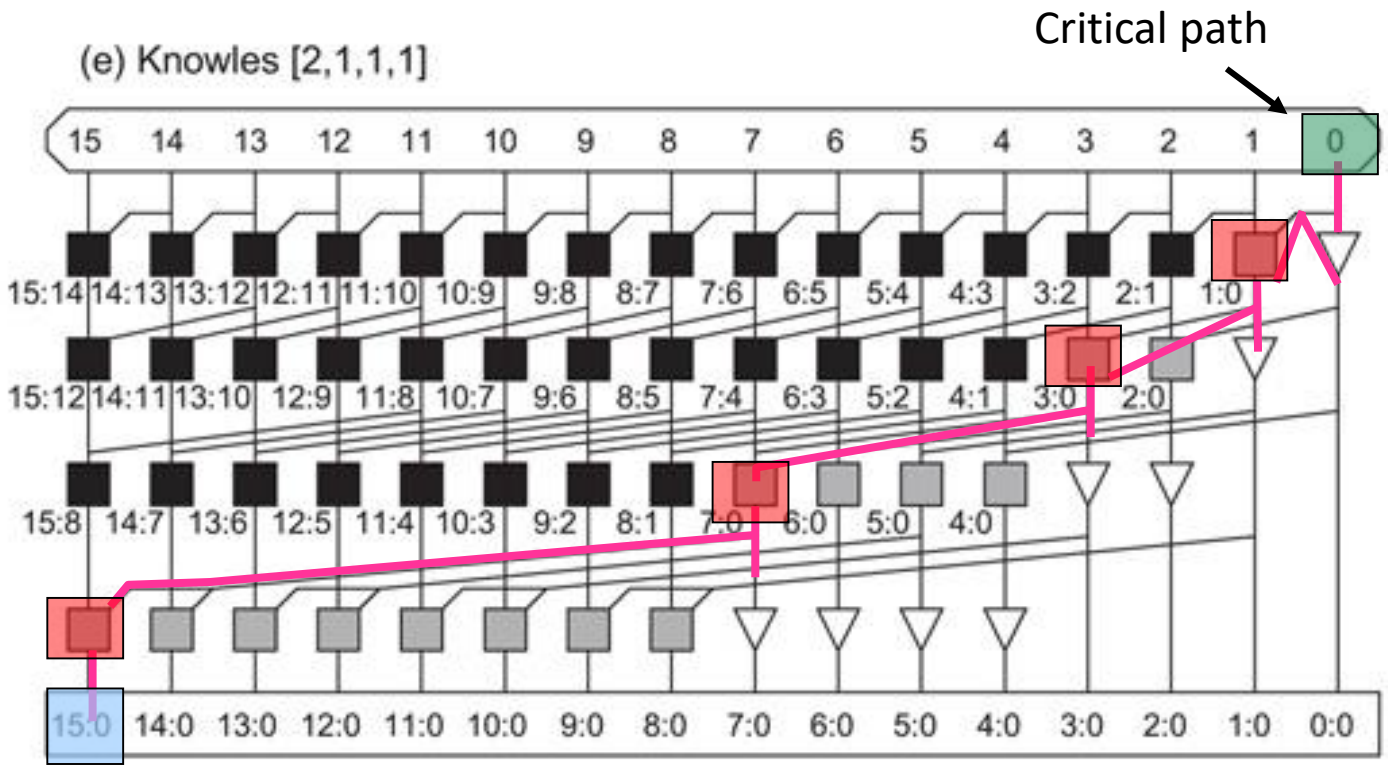
- Το πρόβλημα του μεγάλου fan-out που παρουσιάζει ο Sklansky επιλύεται με τη μέθοδο (μεγάλων καλωδίων) Kogge-Stone

# Knowles (2/4)




Μία από τις διαδρομές με max fan-out

# Knowles (3/4)



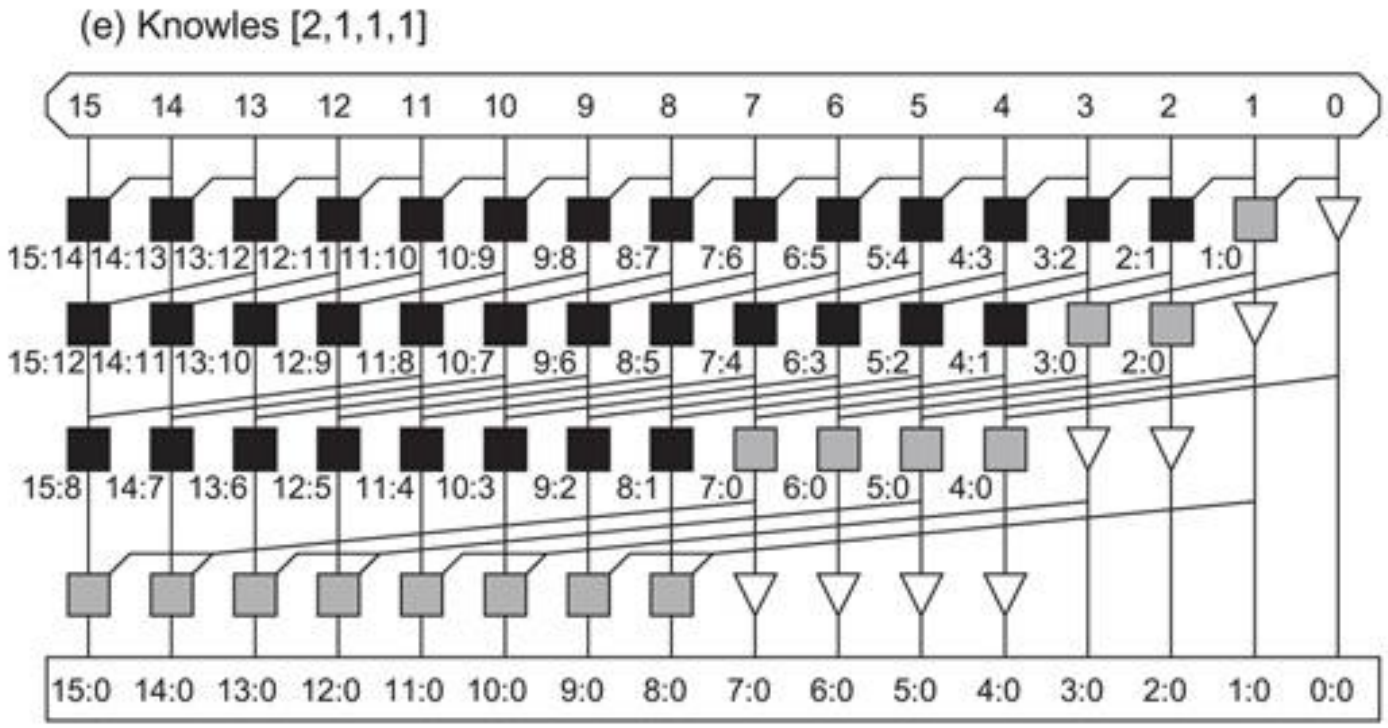

$t_{pg}$

$4t_{AO}$

$t_{xor}$

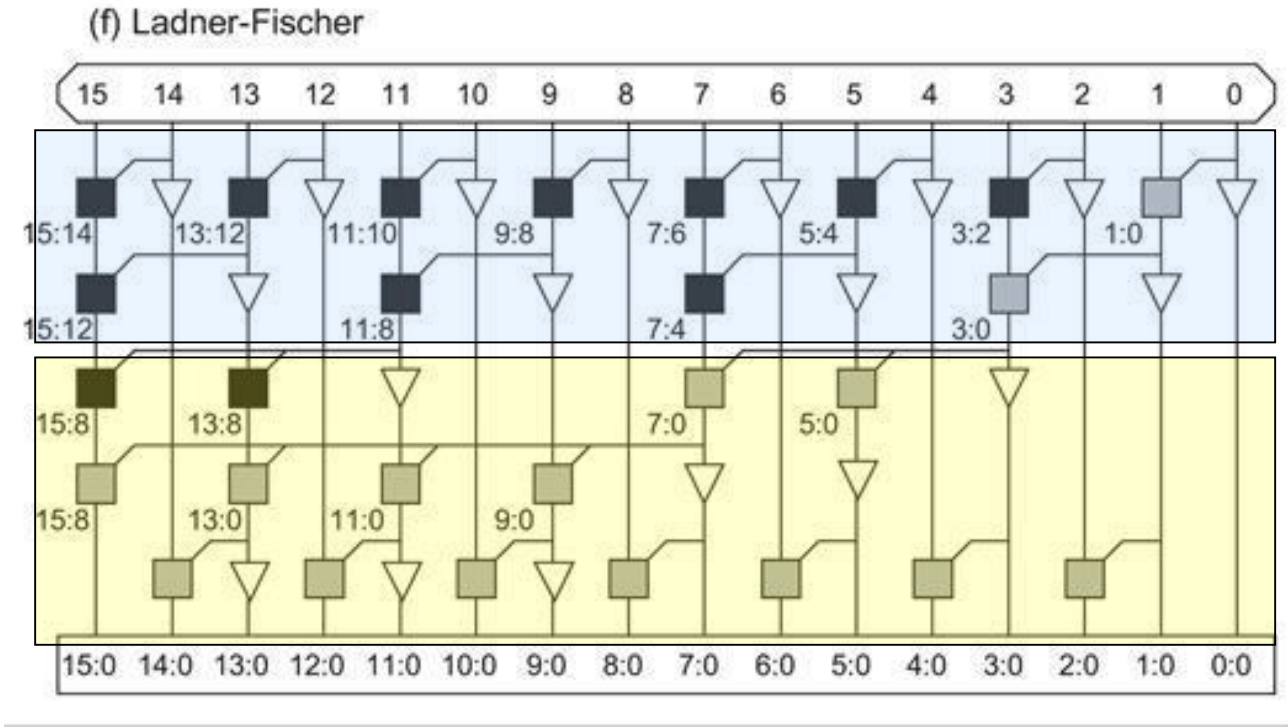
Μία από τις διαδρομές με τη μέγιστη καθυστέρηση

# Knowles (4/4)




- Απαιτεί **$\log_2 N$** επίπεδα για τον τελικό υπολογισμό (η μικρότερη καθυστέρση)
- Καταλαμβάνει μεγάλη επιφάνεια (αριθμός πυλών)
- Δεν έχει μεγάλο πρόβλημα πυκνότητας καλωδίων

# Lander – Fischer (1/4)

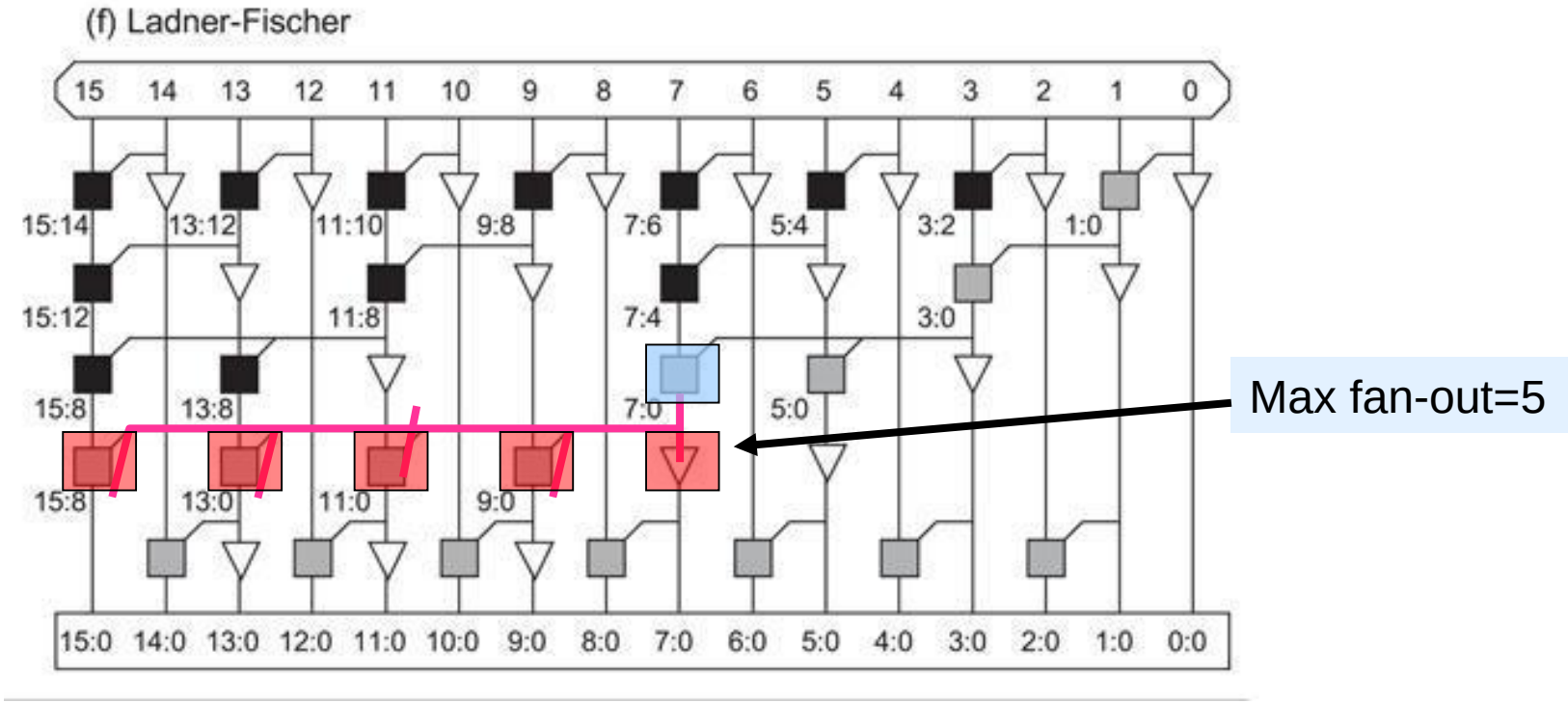



- Οι Ladner-Fischer trees είναι οικογένεια δικτύων μεταξύ Slansky και Brent-Kung

- Αρχικά γίνονται οι υπολογισμοί όπως στους Brent-Kung trees

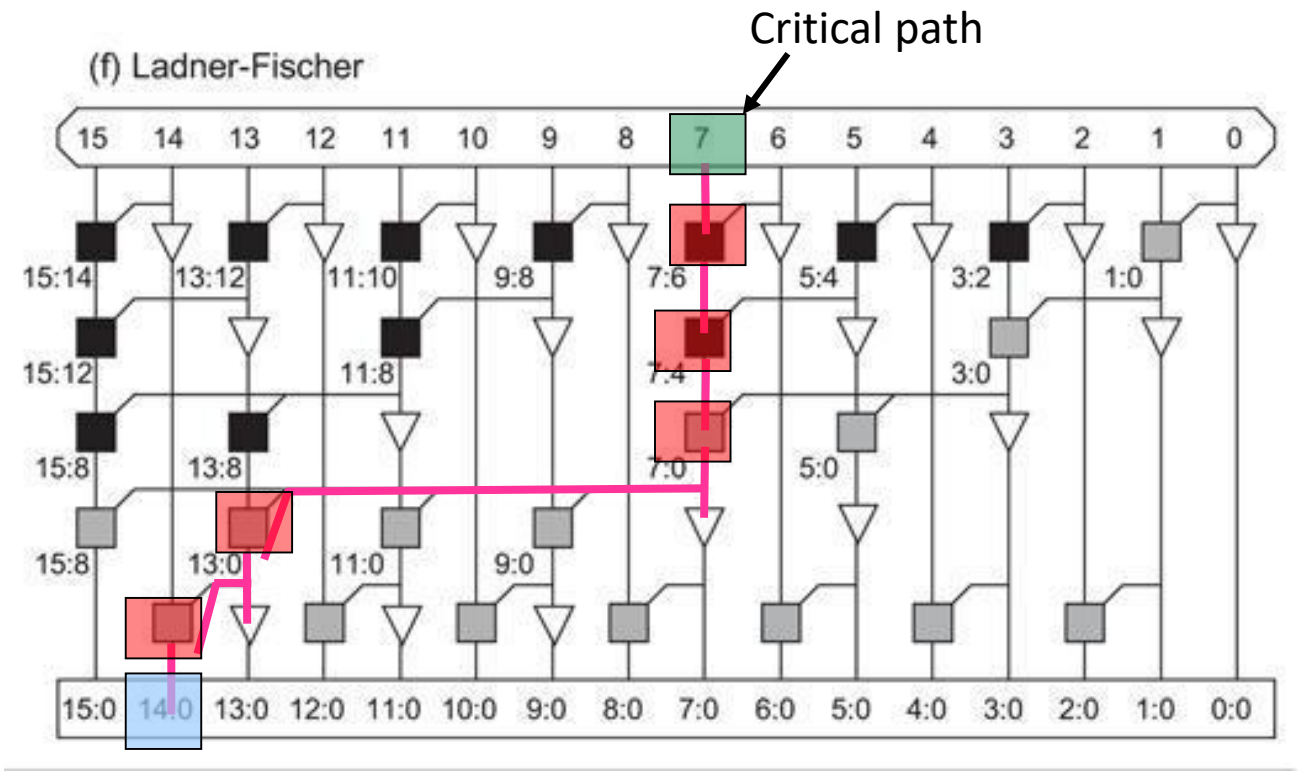
- Ενώ οι υπόλοιποι υπολογισμοί γίνονται με τη μέθοδο των Slansky trees

# Lander – Fischer (2/4)




Η διαδρομή με max fan-out

# Lander – Fischer (3/4)

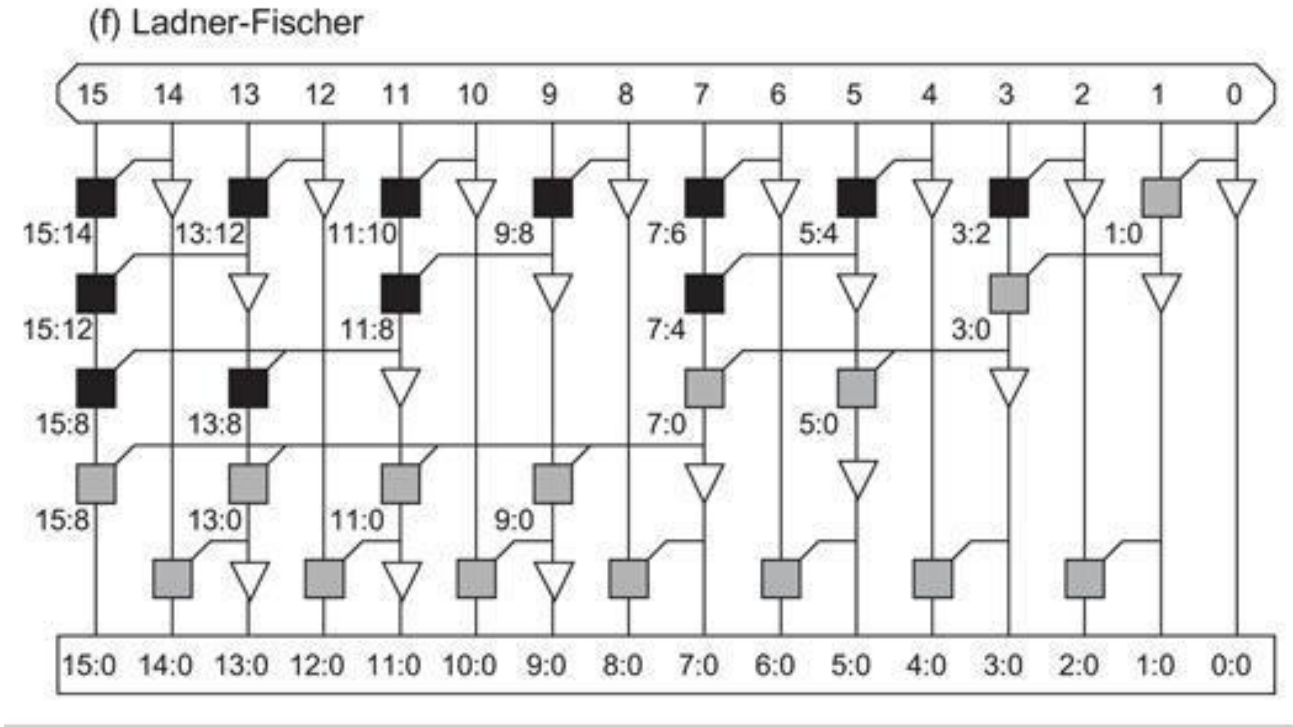



$t_{pg}$

$5t_{AO}$

$t_{xor}$

# Lander – Fischer (4/4)




- Απαιτεί **$2\log_2 N-1$** επίπεδα για τον τελικό υπολογισμό (όχι τόσο καλή καθυστέρηση)

- Καταλαμβάνει πολύ μικρή επιφάνεια
- Δεν έχει πρόβλημα πυκνότητας καλωδίων

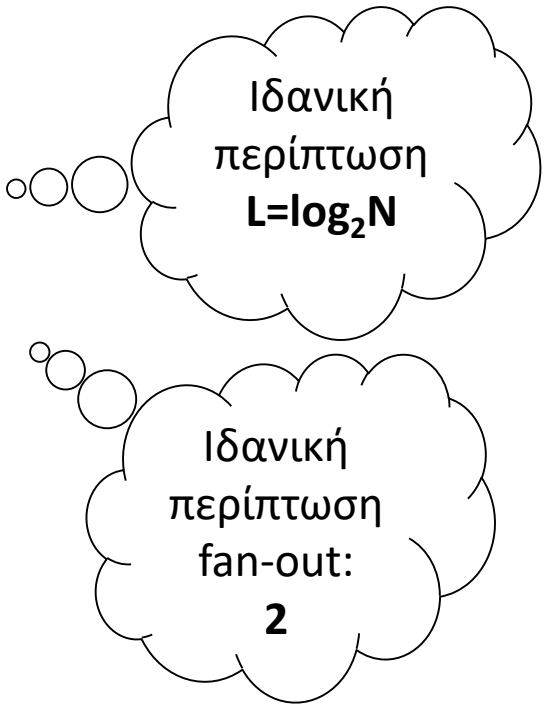
# Ομαδοποίηση tree adders (1/2)

Θέτοντας **$L=\log_2 N$** γίνεται η περιγραφή του κάθε tree με 3 μεταβλητές:

**(l,f,t)** στο σύνολο **[0,L-1]**

Αντιστοίχηση χαρακτηριστικών με μεταβλητές:

- Λογικά επίπεδα: **L + l**
- Fan-out: **$2^f + 1$**
- Διαδρομή Καλωδίων: **$2^t$**



# Ομαδοποίηση tree adders (2/2)

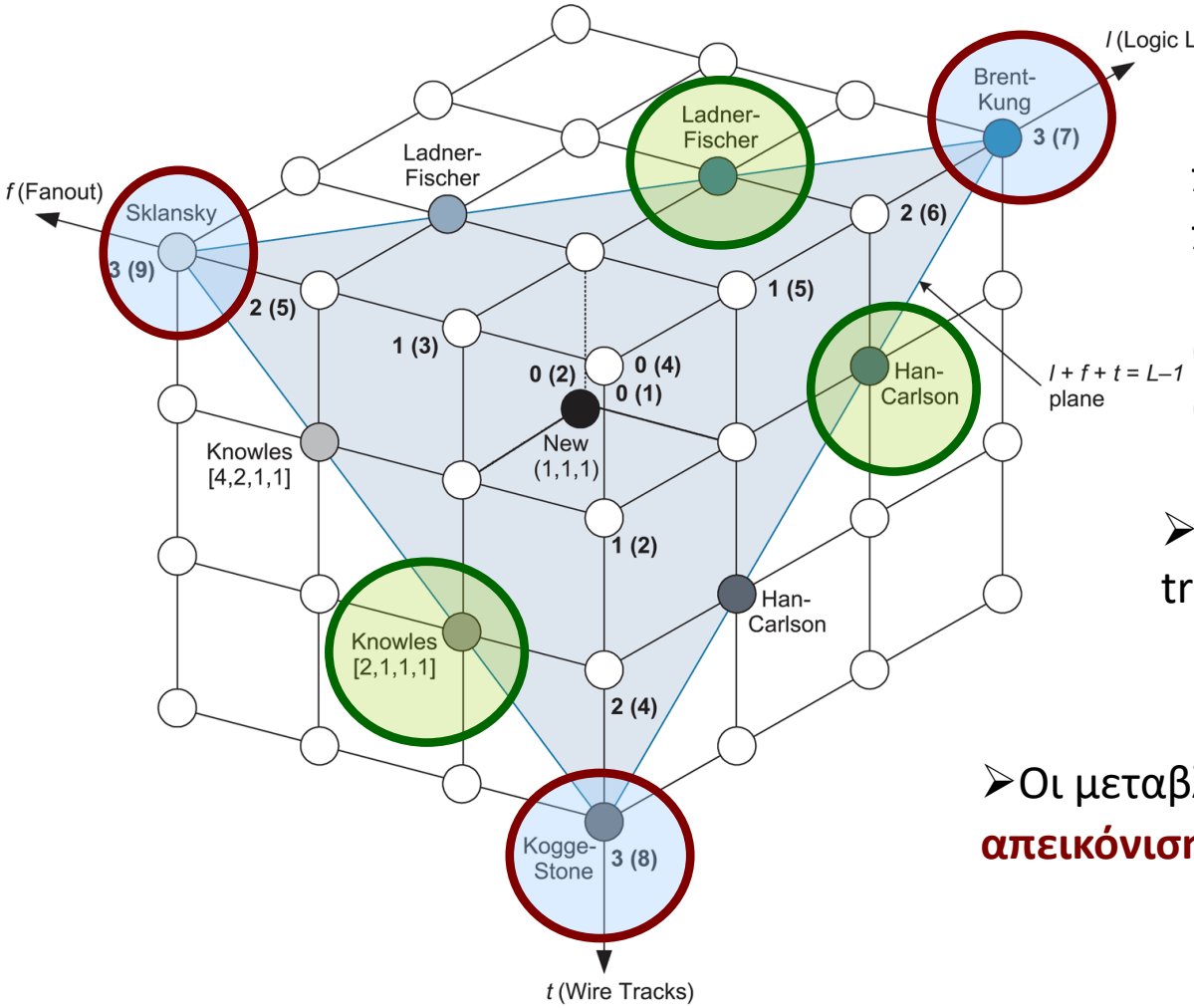


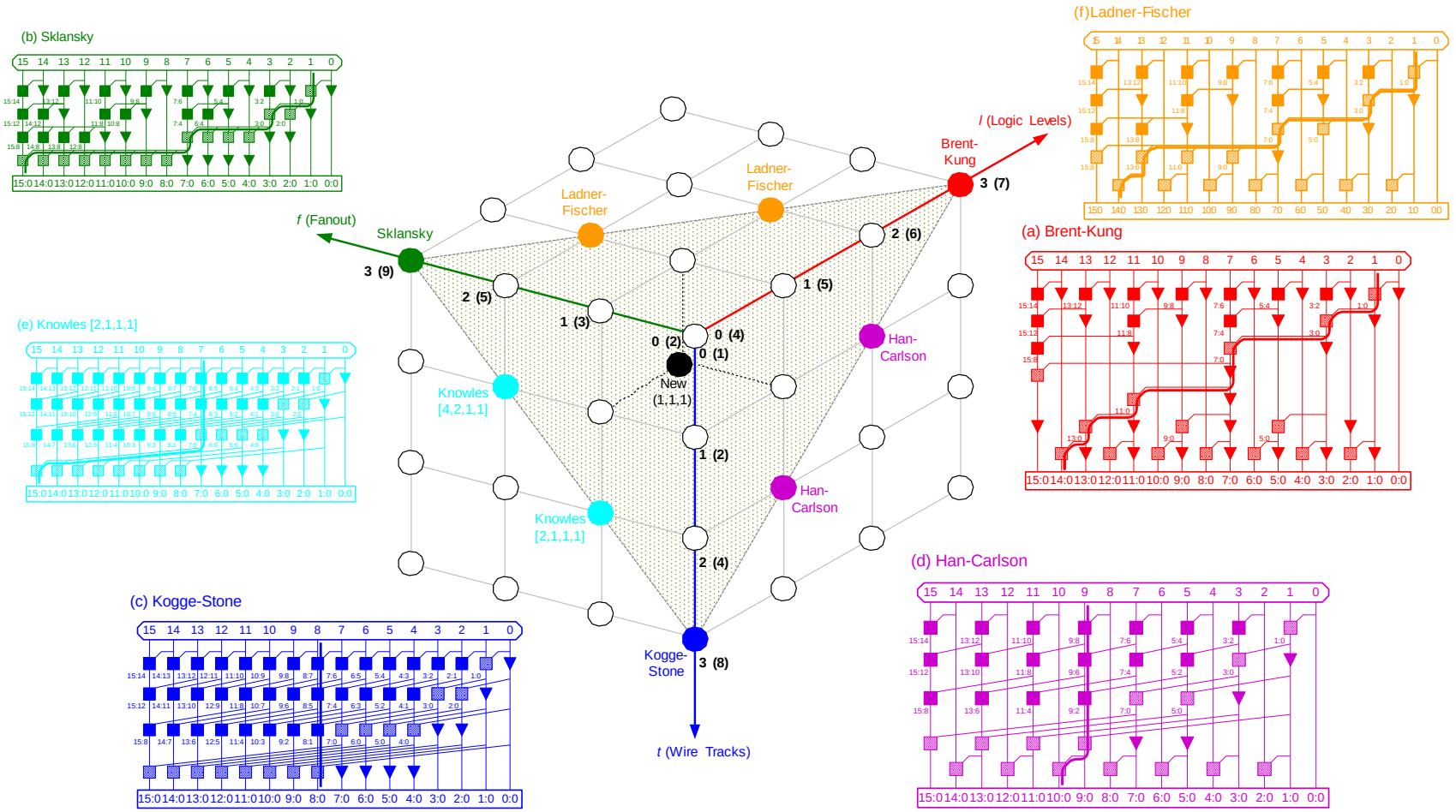

FIG 10.35 Taxonomy of prefix networks

- Στις 3 άκρες του κύβου παρουσιάζονται οι ακραίες περιπτώσεις των trees σε Logic levels (Brent-Kung), Fanout (Sklansky) και wire Tracks (Kogge-Stone)

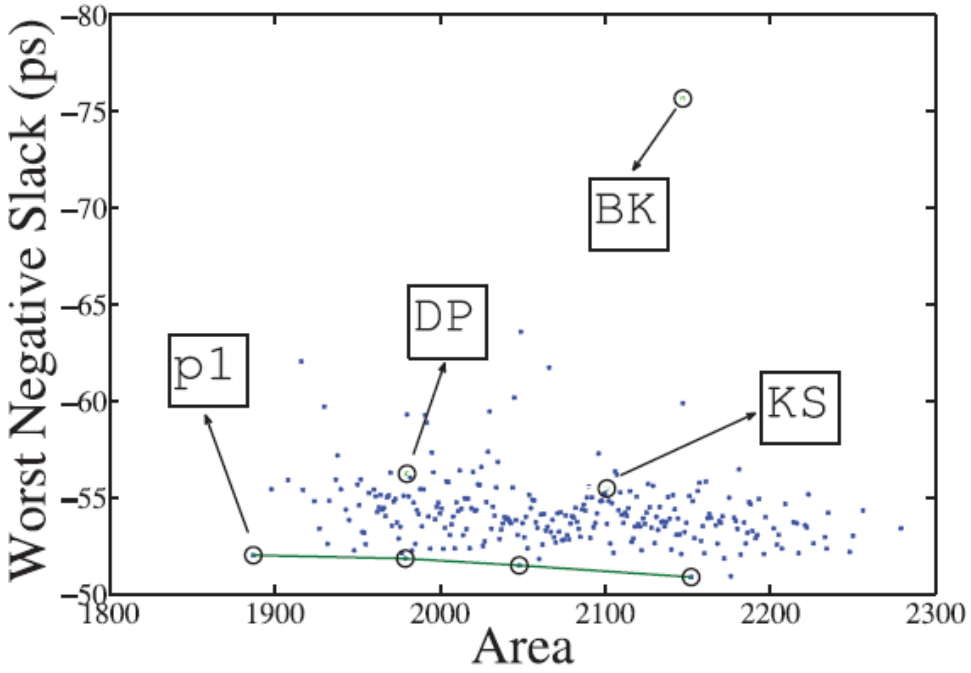
- Οι υπόλοιποι έχουν πιο ισορροπημένα tradeoffs

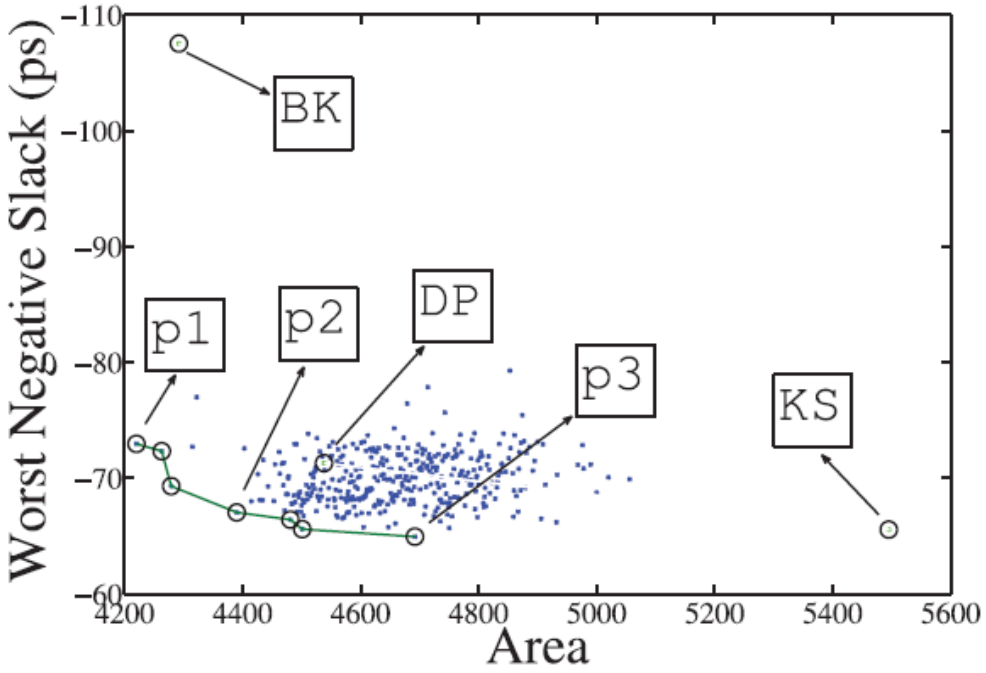
- Οι μεταβλητές **(l,f,t)** δίνουν μία **τρισδιάστατη απεικόνιση** των tree adders

# Taxonomy Revisited



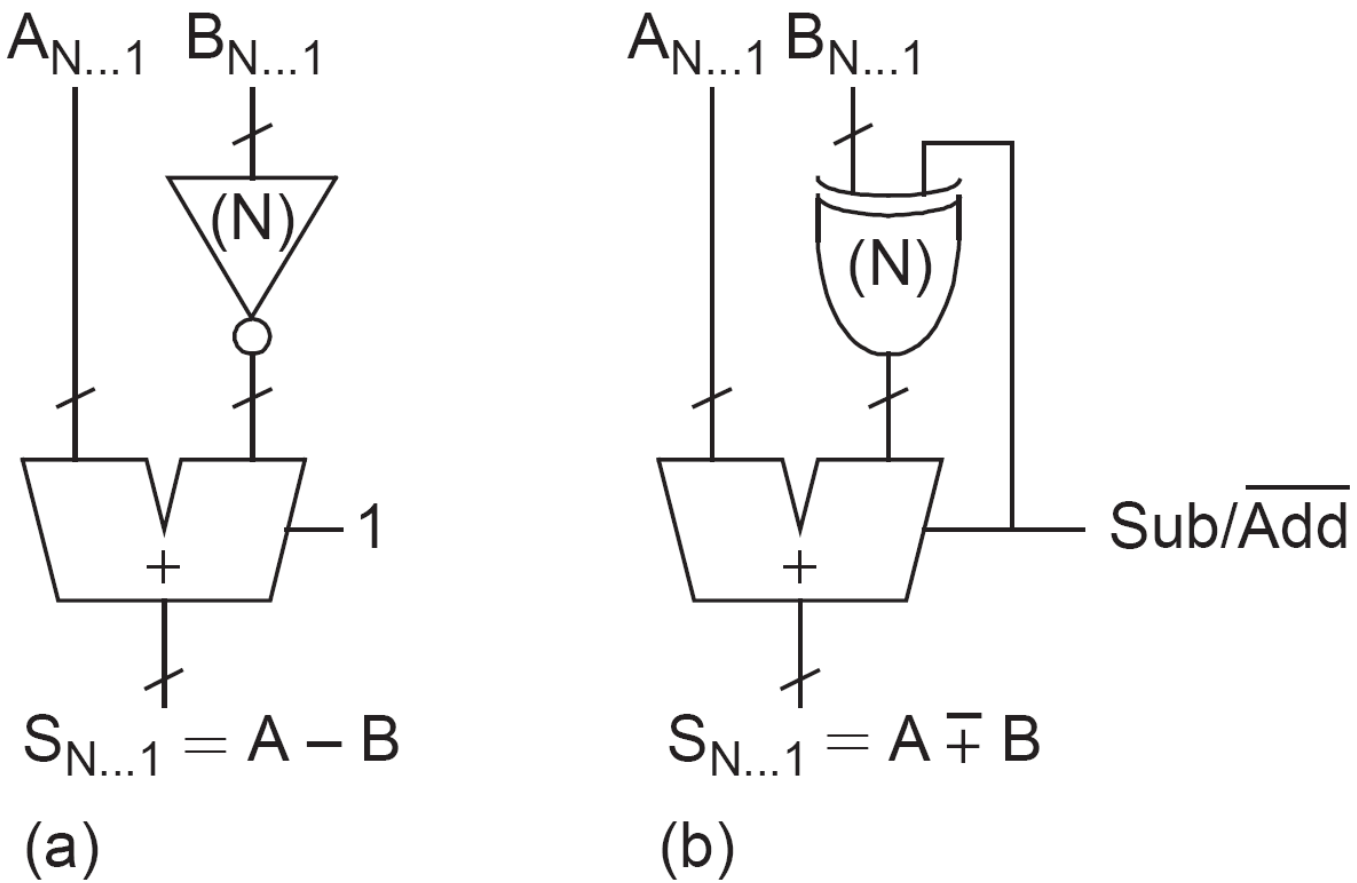
# Αλγοριθμική κατασκευή αθροιστών παράλληλου προθέματος



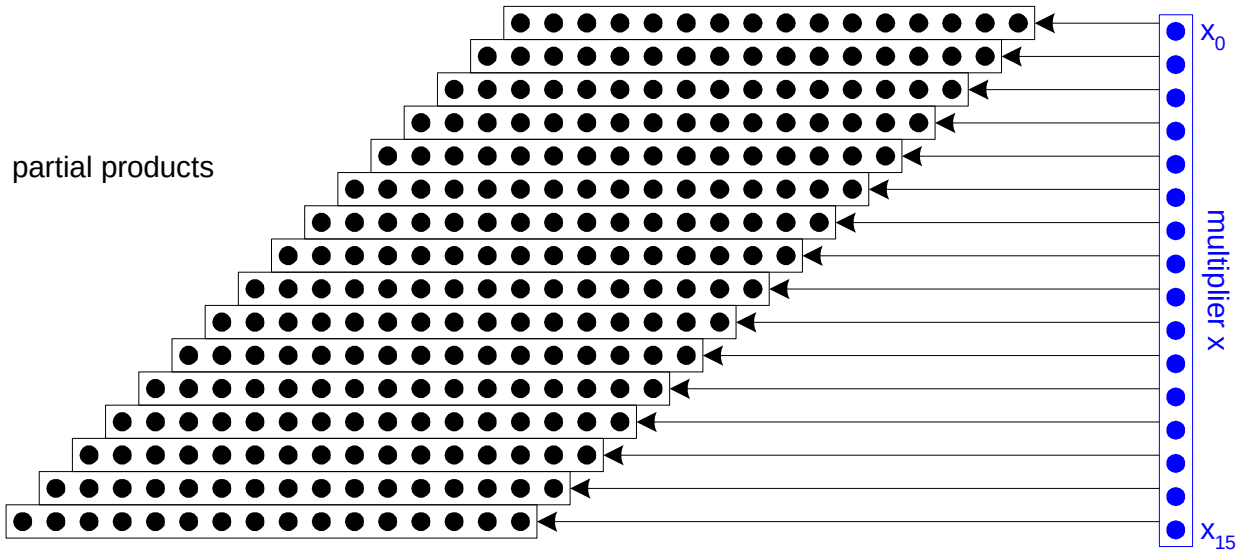




- S. Roy, M. Choudhury, R. Puri and D. Z. Pan, "Towards Optimal Performance-Area Trade-Off in Adders by Synthesis of Parallel Prefix Structures," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1517-1530, Oct. 2014, doi: 10.1109/TCAD.2014.2341926.

# Αφαίρεση



# Διάγραμμα κουκίδων (Dot Diagram)




- Πολλαπλασιασμοί μεγάλων αριθμών απεικονίζονται ευκολότερα με τα διαγράμματα κουκίδων
- Κάθε κουκίδα αντιπροσωπεύει μια θέση για ένα ψηφίο που μπορεί να είναι 0 ή 1
- Τα μερικά γινόμενα αναπαριστώνται από ένα οριζόντιο κουτί γραμμής κουκίδων, ολισθημένο σύμφωνα με το βάρος τους
- Τα δυαδικά ψηφία του πολλαπλασιαστή που χρησιμοποιούνται για την παραγωγή των μερικών γινομένων φαίνονται στα δεξιά

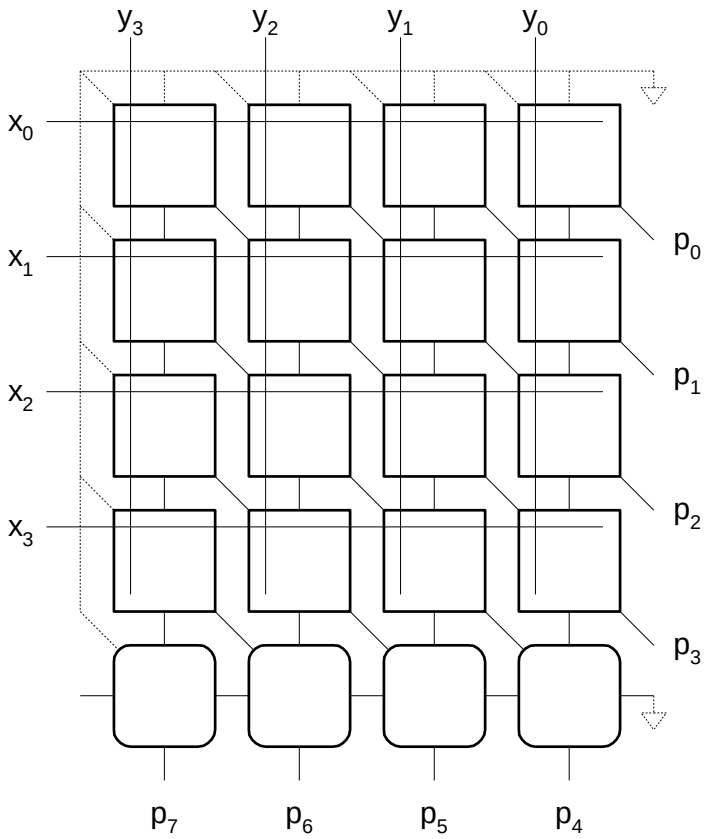
# Γενικές αρχές υλοποίησης πολλαπλασιασμού (1/2)

- Πλήθος τεχνικών για την εκτέλεση του πολλαπλασιασμού
- Η επιλογή βασίζεται πάνω σε μετρικές σχεδιασμού
  - η καθυστέρηση, ο ρυθμός λειτουργίας (throughput rate), επιφάνεια και πολυπλοκότητα
- Η προφανής λύση είναι η χρήση αθροιστή διάδοσης κρατουμένου (CPA) M+1 bits σε δομή αλυσίδας
  - Χρειάζεται N-1 CPAs και είναι αργή, ακόμα κι αν χρησιμοποιηθεί ένας γρήγορος CPA
- Αποδοτικότερες δομές με χρήση ορισμένου τύπου πίνακα ή δένδρων αθροιστών για την πρόσθεση των μερικών γινομένων

# Γενικές αρχές υλοποίησης πολλαπλασιασμού (2/2)

- Κλασσική δομή πίνακα για μη προσημασμένους αριθμούς
- Τροποποίηση πίνακα για προσημασμένους αριθμούς σε συμπλήρωμα ως προς 2 – αλγόριθμος Baugh-Wooley
- Κωδικοποίησης Booth για μείωση πλήθους μερικών γινομένων
- Δένδρα Wallace για μείωση λογικών επιπέδων πρόσθεσης
  - Τα δένδρα Wallace οδηγούν σε πολύπλοκα layouts και έχουν μεγάλου μήκους, μη κανονικές διασυνδέσεις
- Υβριδικές δομές πινάκων / δένδρων

# Rectangular Array




- Ίδιοι αθροιστές μετατοπισμένοι για να ταιριάζουν σε ένα ορθογώνιο σχήμα

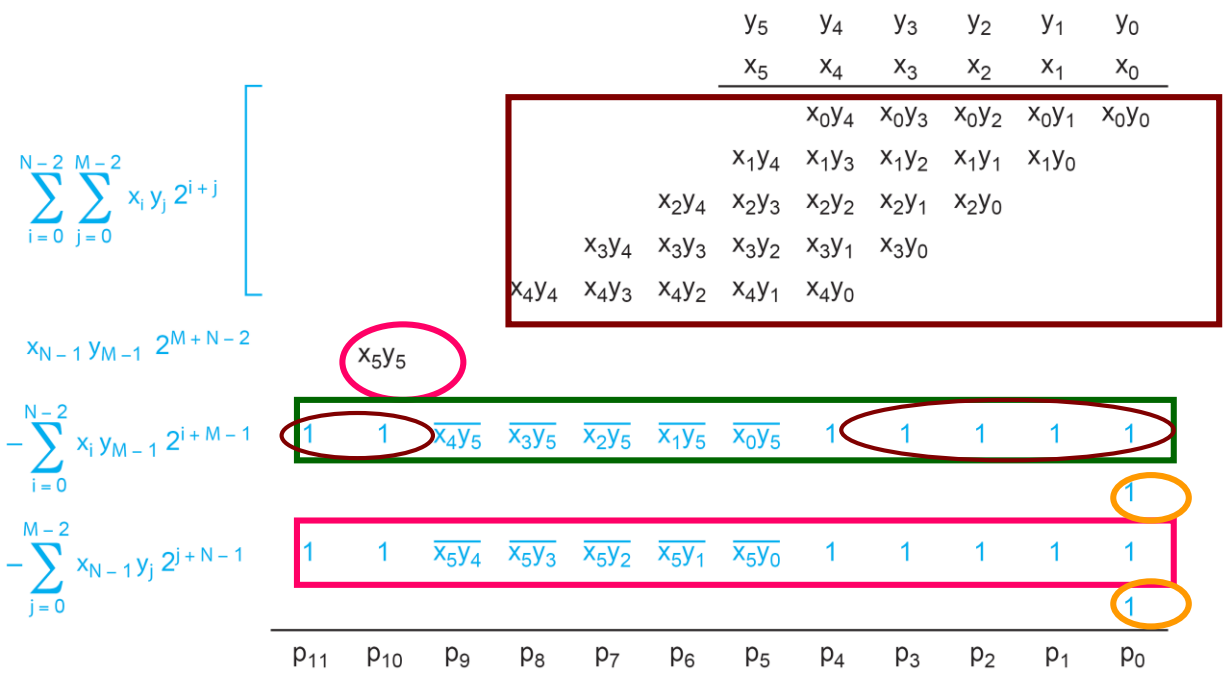
# Πολλαπλασιασμός σε αριθμών σε αναπαράσταση συμπληρώματος του δύο

$$x = -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i \qquad y = -y_{M-1}2^{M-1} + \sum_{j=0}^{M-2} y_j 2^j$$

$$x \cdot y = \left( -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i \right)\left( -y_{M-1}2^{M-1} + \sum_{j=0}^{M-2} y_j 2^j \right)$$

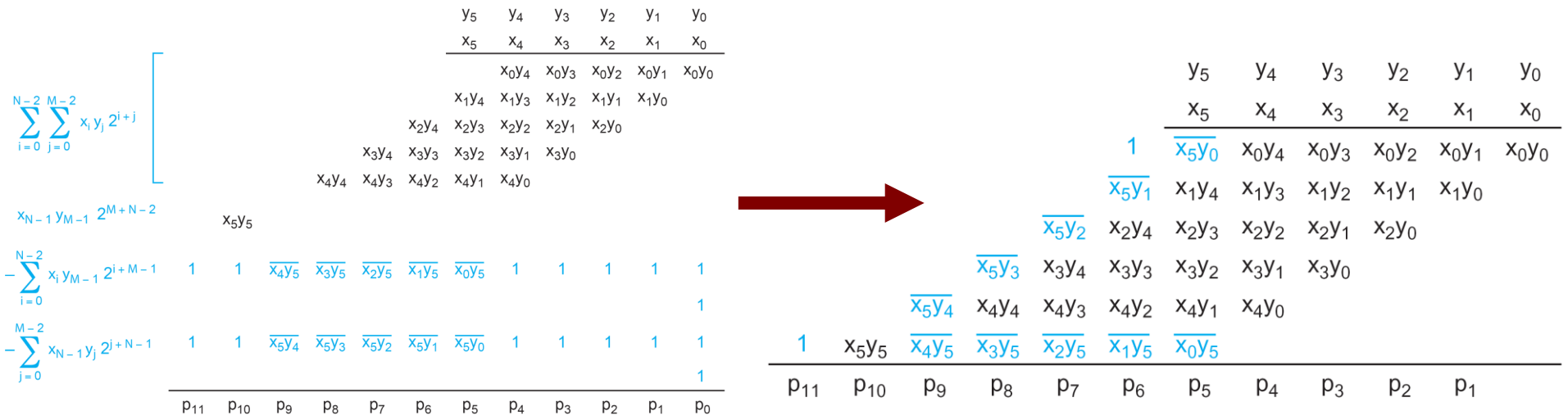
$$= x_{N-1}y_{M-1}2^{M+N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j} - x_{N-1}\sum_{j=0}^{M-2} y_j 2^j - y_{M-1}\sum_{i=0}^{N-2} x_i 2^i$$

# Πολλαπλασιασμός αριθμών σε αναπαράσταση συμπληρώματος ως προς 2 με Baugh – Wooley (2/4)



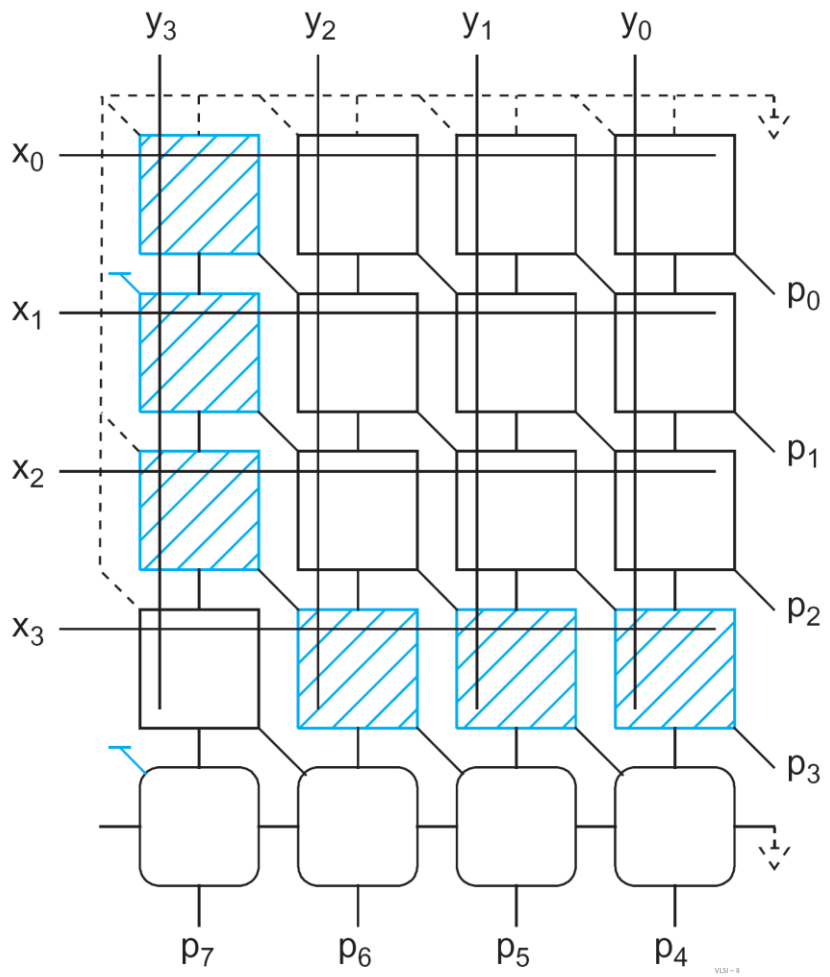

- Γινόμενων μη αρνητικών τμημάτων
- Γινόμενο των δύο MSBs
- **Υπό συνθήκη** αφαιρέσεις ως άθροισμα συμπληρώματος του δύο
  - Ανεστραμμένα bits
  - Πρόσθεση 1 στο λιγότερο σημαντικού ψηφίου
- Συνθήκες είναι οι τιμές των MSBs

# Πολλαπλασιασμός σε συμπλήρωμα ως προς 2 (3/4)




- Η καθυστέρηση εξαρτάται από τον αριθμό των γραμμών των μερικών γινομένων που θα προστεθούν
- Ο τροποποιημένος πολλαπλασιαστής Baugh-Wooley μειώνει τον αριθμό των μερικών γινομένων προ-υπολογίζοντας τα αθροίσματα των σταθερών 1 και μεταθέτοντας κάποιους προς τα πάνω σε επιπλέον στήλες

# Πολλαπλασιασμός σε συμπλήρωμα ως προς 2 (4/4)




| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $y_5$ | $y_4$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
| | | | | | | $x_5$ | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
| | | | | | 1 | $\overline{x_5y_0}$ | $x_0y_4$ | $x_0y_3$ | $x_0y_2$ | $x_0y_1$ | $x_0y_0$ |
| | | | | | $\overline{x_5y_1}$ | $x_1y_4$ | $x_1y_3$ | $x_1y_2$ | $x_1y_1$ | $x_1y_0$ | |
| | | | | $\overline{x_5y_2}$ | $x_2y_4$ | $x_2y_3$ | $x_2y_2$ | $x_2y_1$ | $x_2y_0$ | | |
| | | | $\overline{x_5y_3}$ | $x_3y_4$ | $x_3y_3$ | $x_3y_2$ | $x_3y_1$ | $x_3y_0$ | | | |
| | | $\overline{x_5y_4}$ | $x_4y_4$ | $x_4y_3$ | $x_4y_2$ | $x_4y_1$ | $x_4y_0$ | | | | |
| 1 | $x_5y_5$ | $\overline{x_4y_5}$ | $\overline{x_3y_5}$ | $\overline{x_2y_5}$ | $\overline{x_1y_5}$ | $\overline{x_0y_5}$ | | | | | |
| $p_{11}$ | $p_{10}$ | $p_9$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | |

# Κωδικοποίηση Booth

- Με τον κλασσικό αλγόριθμο κάθε ψηφίο του πολλαπλασιαστή παράγει ένα μερικό γινόμενο που πρέπει να προστεθεί =>
  - μεγάλο πλήθος προσθέσεων μερικών γινομένων (για μεγάλους πολλαπλασιαστές)
  - αύξηση της καθυστέρησης

- Ο αλγόριθμος του Booth κωδικοποιεί τον πολλαπλασιαστή ώστε να δημιουργήσει πολλές και μεγάλου μήκους ακολουθίες από “00…00”

- Τα παραγόμενα μερικά γινόμενα έχουν μηδενική τιμή (είναι “00…00”)

- Σημαντική μείωση των προσθέσεων και των χρησιμοποιούμενων αθροιστών

# Κωδικοποίηση Booth – Βασική ιδέα

- Έστω μια δυαδική ακολουθία

| Θέση | | $i+k$ | $i+k-1$ | $i+k-2$ | … | $i+1$ | $i$ | $i-1$ | … | … |
|---|---|---|---|---|---|---|---|---|---|---|
| Τιμή | | 0 | 1 | 1 | … | 1 | 1 | 0 | | |

k συνεχόμενοι “1”

➢Με βάση τη σχέση $2^{i+k} - 2^i = 2^{i+k-1} + 2^{i+k-2} + .... + 2^{i+1} + 2^i$

Αφαίρεση

| Θέση | | $i+k$ | $i+k-1$ | $i+k-2$ | … | $i+1$ | $i$ | $i-1$ | … | … |
|---|---|---|---|---|---|---|---|---|---|---|
| Τιμή | | 1 | 0 | 0 | … | 0 | −1 | 0 | | |

Πρόσθεση

k συνεχόμενα “0”

➢Απαιτούνται κατάλληλες ολισθήσεις και **1 πρόσθεση (+A) και 1 αφαίρεση (-A) αντί k προσθέσεις (A+A+…+A)** του πολλαπλασιαστέου Α

➢Εισάγονται 2 βοηθητικά (dummy) bits $b_n$, $b_{-1}$=0 και στο B=$b_{n-1}$, $b_{n-2}$, …, $b_1$, $b_0$

# Πίνακας κωδικοποίησης Booth (1/2)

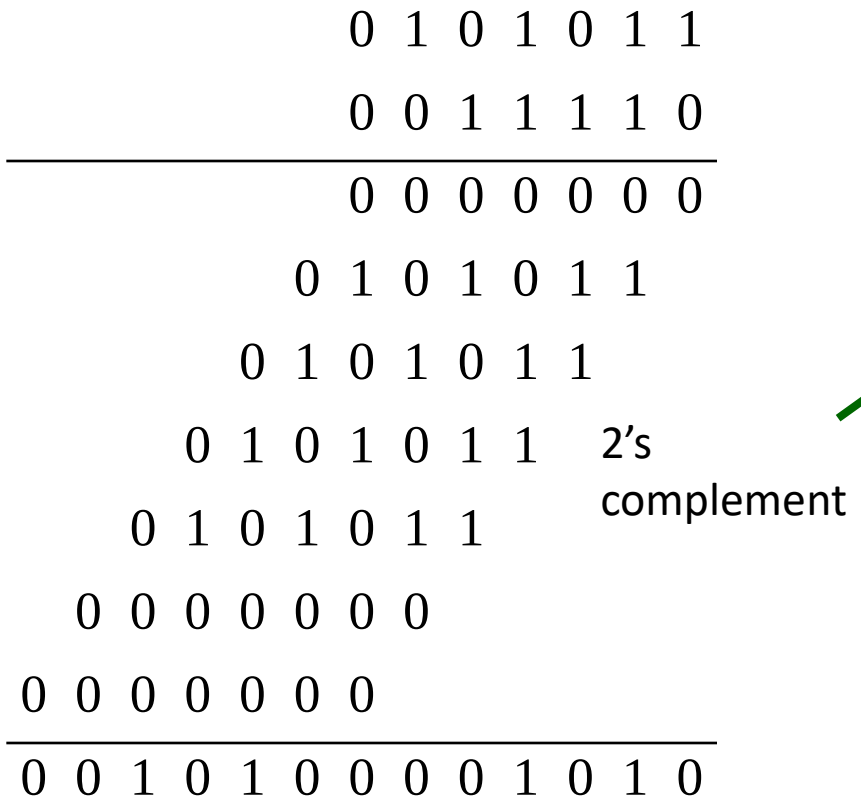
| Πολλαπλασιαστής | | |
|---|---|---|
| Bit $i$ | Bit $i + 1$ | Λειτουργία |
| 0 | 0 | $0 \times$ πολλαπλασιαστέος $(0 \times A)$ |
| 0 | 1 | $+1 \times$ πολλαπλασιαστέος $(+1 \times A)$ |
| 1 | 0 | $-1 \times$ πολλαπλασιαστέος $(-1 \times A)$ |
| 1 | 1 | $0 \times$ πολλαπλασιαστέος $(0 \times A)$ |

- Χρησιμοποιεί μόνο τους όρους 0, +A, −A και κατάλληλες ολισθήσεις
- Με βάση τον παραπάνω πίνακα ο αριθμός 0011110 (+30) κωδικοποιείται σε 0+1000-10 (32-2=30)

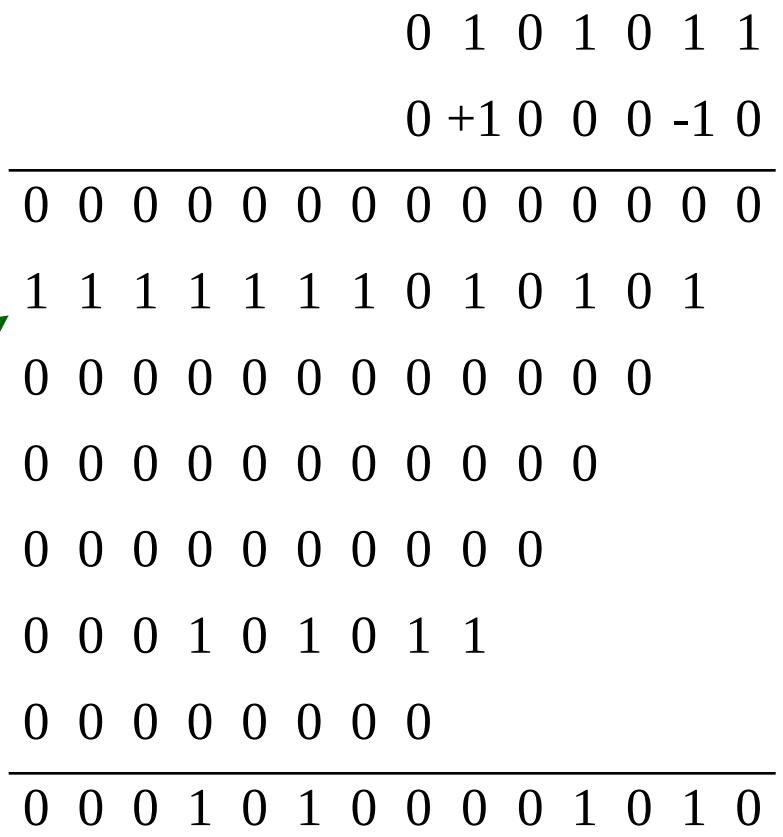
# Πίνακας κωδικοποίησης Booth (2/2)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Worst case | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| Κωδικοποίηση | +1 | -1 | +1 | -1 | +1 | -1 | +1 | -1 |
| Best case | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| Κωδικοποίηση | 0 | +1 | 0 | 0 | 0 | -1 | 0 | 0 |

# Πολλαπλασιασμός Booth – Παράδειγμα




Συμβατικός




Booth

# Κωδικοποίηση Booth – Radix 4 (1/3)

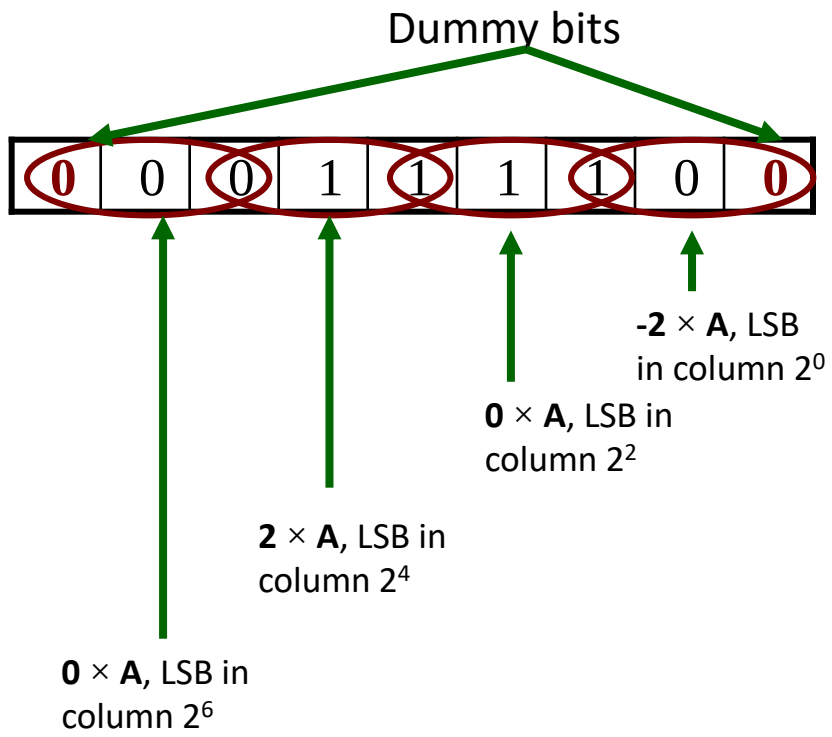
- Η προηγούμενη κωδικοποίηση επιταχύνει τον πολλαπλασιασμό υπερπηδώντας ακολουθίες από συνεχόμενους “1…..1”
- Η διαδικασία μπορεί να επιταχυνθεί ακόμη περισσότερο συνδυάζοντας 3-αδες ψηφίων του πολλαπλασιαστή
  - Στην ουσία εξετάζει ένα ζεύγος ψηφίων λαμβάνοντας υπόψη το αμέσως προηγούμενό ψηφίο δεξιά
- Οδηγεί στην παραγωγή το πολύ n/2 μερικών γινομένων για έναν n-bit πολλαπλασιαστή
- Όπως και η προηγούμενη κωδικοποίηση ισχύει για προσημασμένους και μη προσημασμένους αριθμούς

# Πίνακας κωδικοποίησης Booth radix 4 (2/3)

| Ζεύγος ψηφίων πολ/στή | | Ψηφίο δεξιά | Λειτουργία | Εξήγηση |
|---|---|---|---|---|
| $2^1$ | $2^0$ | | | |
| $i+1$ | $i$ | $i-1$ | | |
| 0 | 0 | 0 | 0×A | No string |
| 0 | 0 | 1 | +1×A | End of string |
| 0 | 1 | 0 | +1×A | Single 1 (+2−1) |
| 0 | 1 | 1 | +2×A | End of string |
| 1 | 0 | 0 | −2×A | Beginning of string |
| 1 | 0 | 1 | −1×A | End/ beginning of string |
| 1 | 1 | 0 | −1×A | Beginning of string |
| 1 | 1 | 1 | 0×A | String of 1s |

# Κωδικοποίηση Radix-4 (3/3)

| Ζεύγος ψηφίων πολλαπλασιαστή | | | Λειτουργία | Εξήγηση |
|---|---|---|---|---|
| $2^1$ | $2^0$ | | | |
| $i+1$ | $i$ | $i-1$ | | |
| 0 | 0 | 0 | 0×A | No string |
| 0 | 0 | 1 | +1×A | End of string |
| 0 | 1 | 0 | +1×A | Single 1 (+2−1) |
| 0 | 1 | 1 | +2×A | End of string |
| 1 | 0 | 0 | −2×A | Beginning of string |
| 1 | 0 | 1 | −1×A | End/ beginning of string |
| 1 | 1 | 0 | −1×A | Beginning of string |
| 1 | 1 | 1 | 0×A | String of 1s |



# Πολλαπλασιασμός Booth – Παράδειγμα

Συμβατικός

```
          0 0 0 0 1 1
          0 1 1 1 0 1
  ---------------------
          0 0 0 0 1 1
        0 0 0 0 0 0
      0 0 0 0 1 1
    0 0 0 0 1 1
  0 0 0 0 1 1
0 0 0 0 0 0
---------------------
0 0 0 0 1 0 1 0 1 1 1
```

Booth Radix2

```
                                  0  0  0  0  1  1
                                 +1  0  0 -1  0  1
  ------------------------------------------------
  0  0  0  0  0  0  0  0  0  0  1  1
  0  0  0  0  0  0  0  0  0  0  0
  1  1  1  1  1  1  1  1  0  1
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  1  1
  ------------------------------------------------
  0  0  0  0  0  1  0  1  0  1  1  1
```

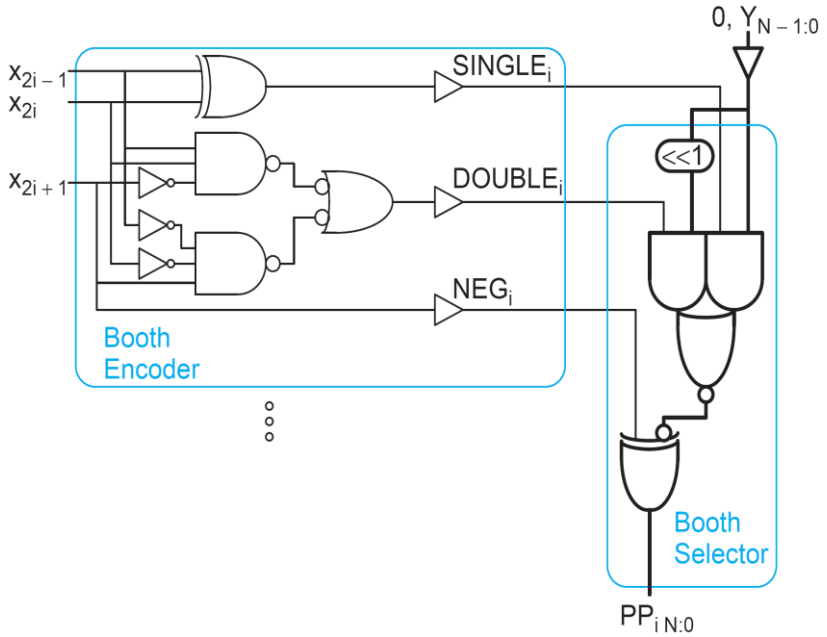
2's complement

Booth Radix4

```
          0   0   0   0   1   1
      0   1   1   1   0   1   0
        2×A    −1×A    1×A
  ------------------------------------------
  0   0   0   0   0   0   0   0   0   0   1   1
  1   1   1   1   1   1   1   1   0   1
  0   0   0   0   0   1   1   0
  ------------------------------------------
  0   0   0   0   0   0   1   0   1   0   1   1   1
```

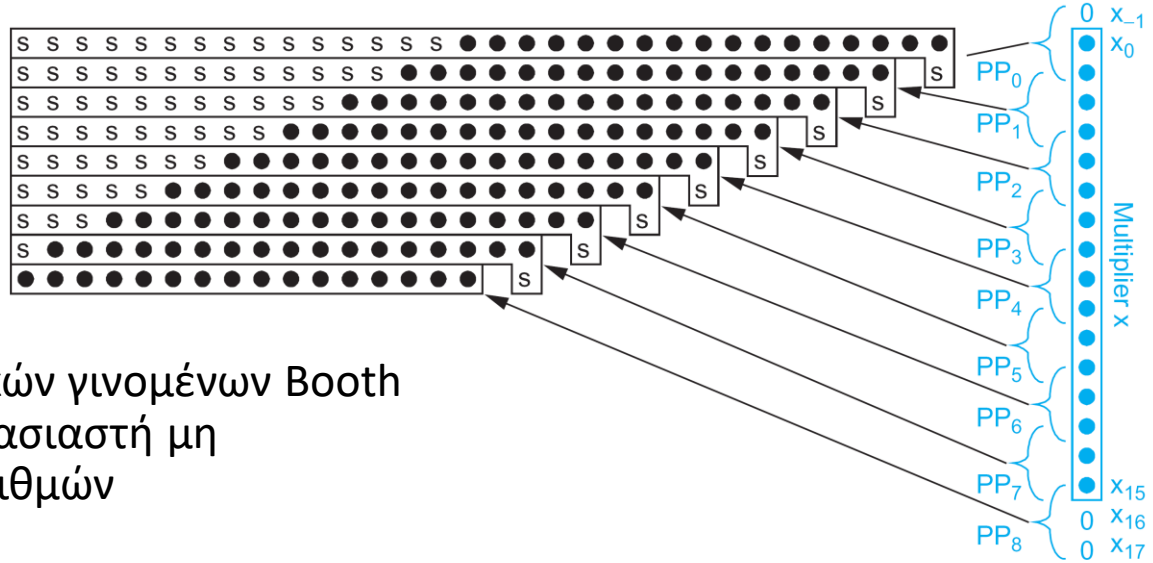
# Κυκλώματα Κωδικοποίησης & Επιλογής Booth

| Inputs | | | Partial Product | Booth Selects | | |
|---|---|---|---|---|---|---|
| $x_{2i+1}$ | $x_{2i}$ | $x_{2i-1}$ | $PP_i$ | $\text{SINGLE}_i$ | $\text{DOUBLE}_i$ | $\text{NEG}_i$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | $Y$ | 1 | 0 | 0 |
| 0 | 1 | 0 | $Y$ | 1 | 0 | 0 |
| 0 | 1 | 1 | $2Y$ | 0 | 1 | 0 |
| 1 | 0 | 0 | $-2Y$ | 0 | 1 | 1 |
| 1 | 0 | 1 | $-Y$ | 1 | 0 | 1 |
| 1 | 1 | 0 | $-Y$ | 1 | 0 | 1 |
| 1 | 1 | 1 | $-0\ (=0)$ | 0 | 0 | 1 |




- Το κύκλωμα κωδικοποίησης παράγει τα σήματα (single, double, neg)
- Το κύκλωμα επιλογής δέχεται τα σήματα (single, double, neg) και τον πολλαπλασιαστέο Y εκτεταμένο ως προς το μηδέν σε $N+1$ bits –έξοδος τιμές $0, Y, 2Y$
- Αν το μερικό γινόμενο είναι αρνητικό (neg=1) χρησιμοποιείται το συμπλήρωμα του δύο

# Επέκταση προσήμου

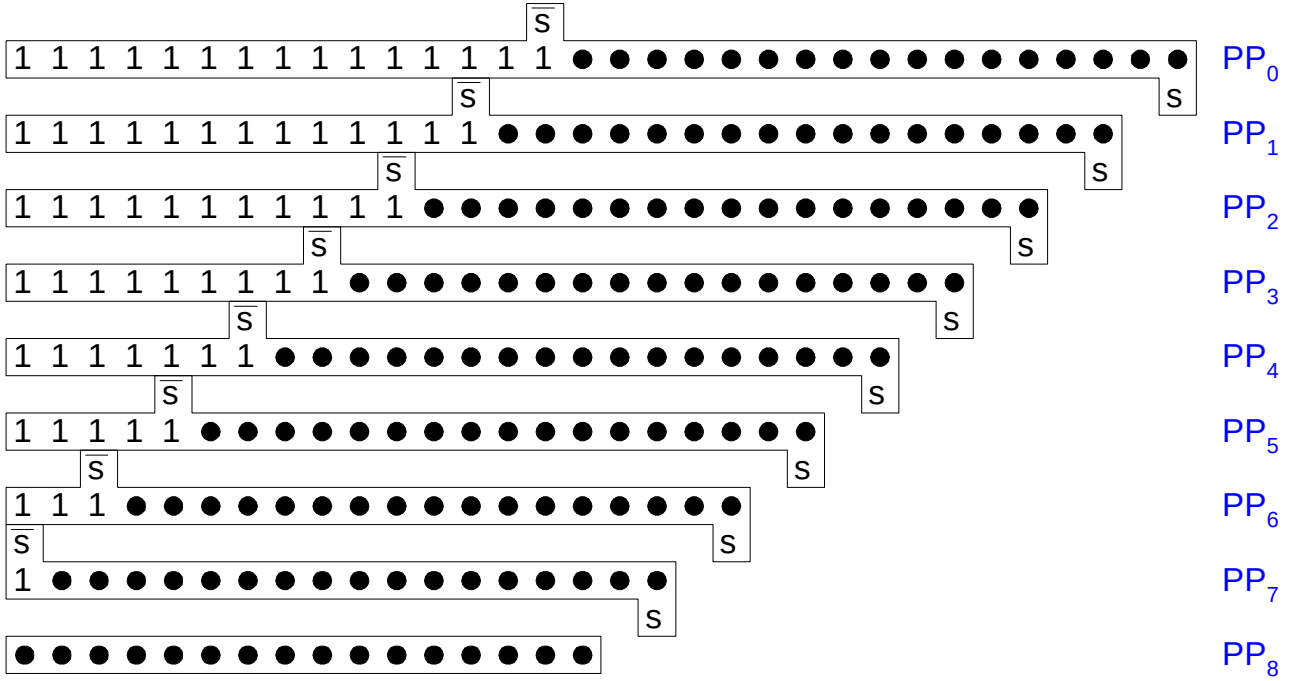



16-bit πίνακας μερικών γινομένων Booth radix-4 για πολλαπλασιαστή μη προσημασμένων αριθμών

- Ακόμα και σε μη προσημασμένους αριθμούς τα αρνητικά μερικά γινόμενα πρέπει να επεκταθούν ως προς το πρόσημο για να προστεθούν σωστά
- Κάθε μερικό γινόμενο επεκτείνεται ως προς το πρόσημο με βάση το σήμα $neg_i$
- Προστίθεται $s$ στο LSB στην επόμενη γραμμή (το συμπλήρωμα ως προς 2)
- Μεγάλες απαιτήσεις fanout για τα MSBs

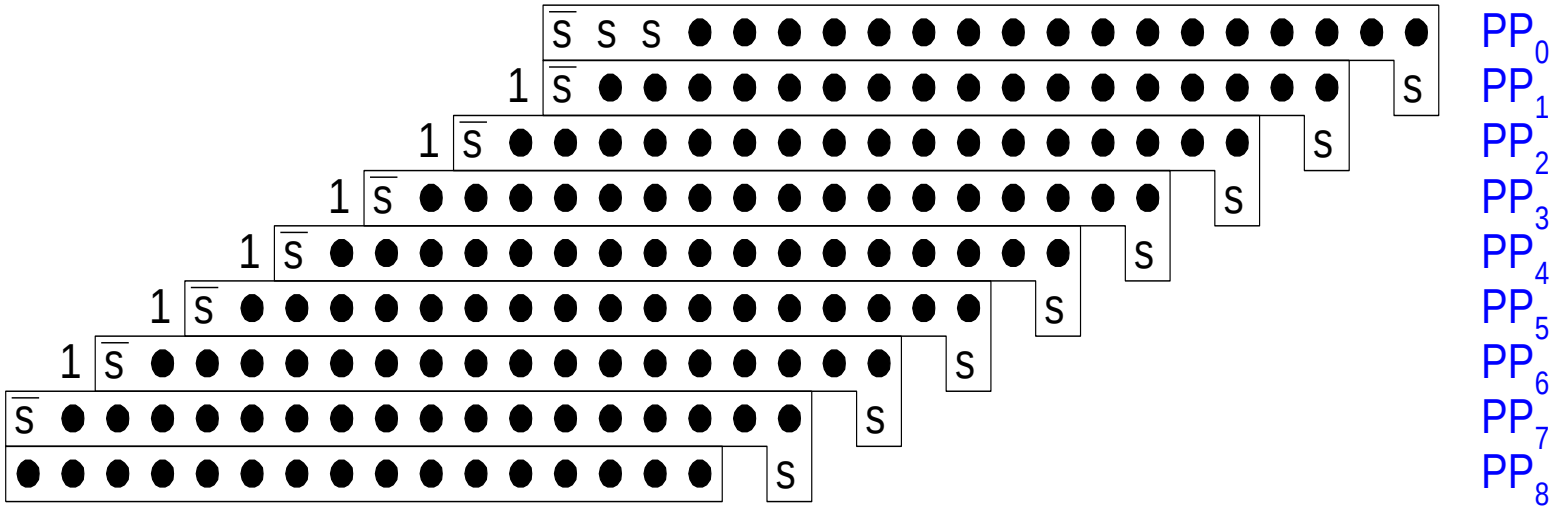
# Απλοποιημένη επέκταση προσήμου (1/2)

- Τα Sign bits είναι είτε όλα 0's είτε όλα 1's
  - Όμως το όλα 0's είναι ισοδύναμο με το όλα 1's + 1 στην κατάλληλη στήλη
  - Η ιδέα αυτή χρησιμοποιείται για να ελαττώσει το φορτίο του MSB



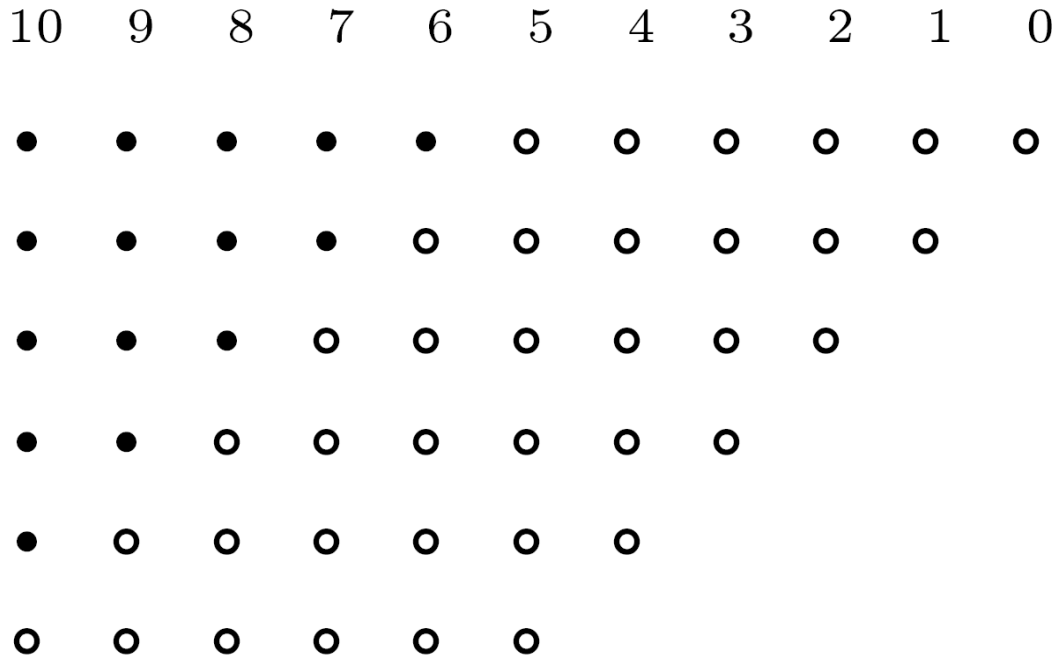
# Απλοποιημένη επέκταση προσήμου (2/2)

- Δε χρειάζεται να γίνονται όλες οι προσθέσεις των 1’s in hardware
  - Προϋπολογισμός έξω από τον πίνακα



# Τροποποιημένος πίνακας για αρνητικούς αριθμούς

- Τα ψηφία προσήμου πρέπει να επεκταθούν κατάλληλα
  - Στη 1η γραμμη έχουμε 11 αντί 6 ψηφία κοκ
- Αυξάνει την πολυπλοκότητα των multi-operand adder
- Αν χρησιμοποιηθεί 1’s complement και πρόσθεση 1 στο LSB => ακόμη μεγαλύτερη αύξηση των στηλών και πολυπλοκότητα των multi-operand adder



# Μείωση πολυπλοκότητας

- Two's complement αριθμός ssssss $z_4 z_3 z$ $z_1 z_0$ με τιμή

$$-s \cdot 2^{10} + s \cdot 2^9 + s \cdot 2^8 + s \cdot 2^7 + s \cdot 2^6 + s \cdot 2^5 + z_4 \cdot 2^4 + z_3 \cdot 2^3 + z_2 \cdot 2^2 + z_1 \cdot 2^1 + z_0$$

- Αντικαθίσταται από 00000 (-s) $z_4 z_3 z_2 z_1 z_0$ αφού

$$-s \cdot 2^{10} + s \cdot (2^9 + 2^8 + 2^7 + 2^6 + 2^5)$$

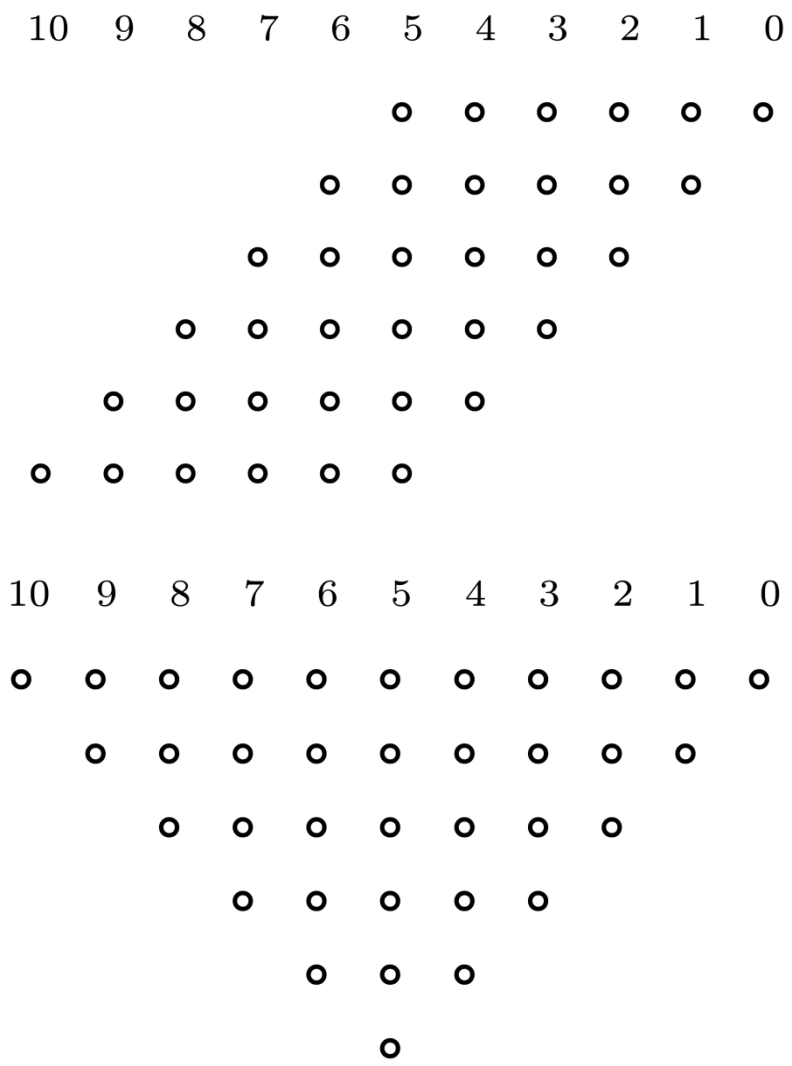
$$= -s \cdot 2^{10} + s \cdot (2^{10} - 2^5) = \quad -s \cdot 2^5$$

# Πρόσθεση μερικών γινομένων

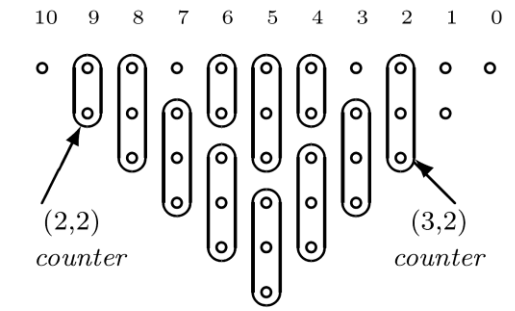
- Τα μερικά γινόμενα πρέπει να προστεθούν για την παραγωγή του τελικού αποτελέσματος
- Χρήση αθροιστών πολλαπλών ορισμάτων
  - Fast multi-operand adder
- Η δομή των μερικών γινομένων πρέπει να ληφθεί υπόψη ώστε να ελαττωθεί η πολυπλοκότητα
- Μερικά μερικά γινόμενα έχουν μικρότερο πλήθος ψηφίων από το μέγιστο
  - πρέπει να ευθυγραμμιστούν κατάλληλά
  - απαιτούν λιγότερους και απλούστερους αθροιστές / μετρητές

# Παράδειγμα - 6 Partial Products

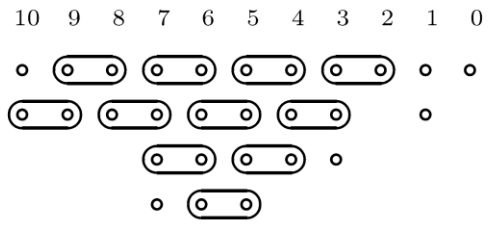
- Παράγονται όταν πολ/νται μη προσημασμένοι 6-bit αριθμοί
- 6 operands μπορούν να προστεθούν χρησιμοποιώντας 3 επίπεδα CSAs (Wallace tree)
- Το πλήθος των (3,2) μετρητών μπορεί να μειωθεί δραστικά εκμεταλλευόμενοι το γεγονός ότι μόνο μια στήλη έχει 6 ψηφία
- Επανασχεδίαση του διαγράμματος κουκίδων για την επιλογή των (3,2) μετρητών



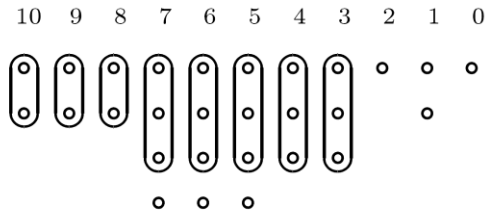
# Μείωση πολυπλοκότητας - Χρήση (2,2) Counters (HAs)



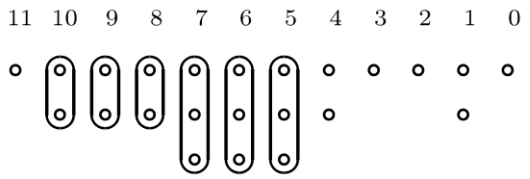

(a) Level 1 carry-save addition.




(b) Results of level 1.



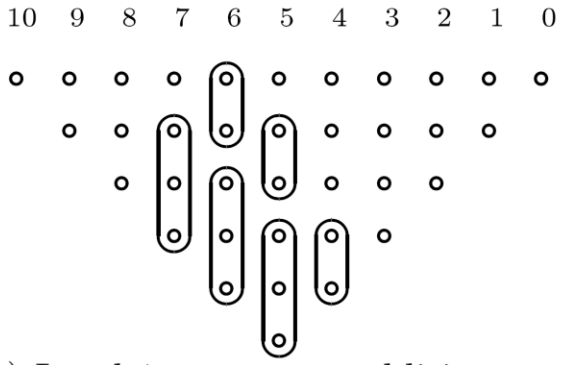

(c) Level 2 carry-save addition.



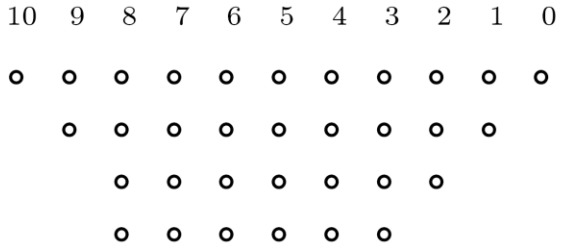

(d) Level 3 carry-save addition.

- Ο αριθμός των επιπέδων παραμένει 3 αλλά λιγότεροι counters

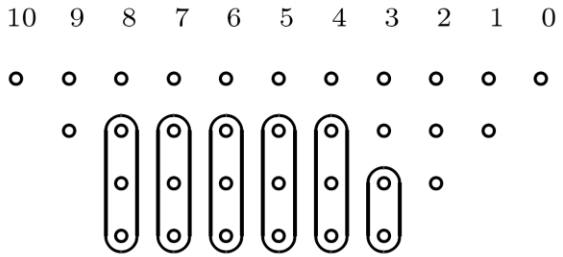
# Επιπλέον μείωση του πλήθους των μετρητών



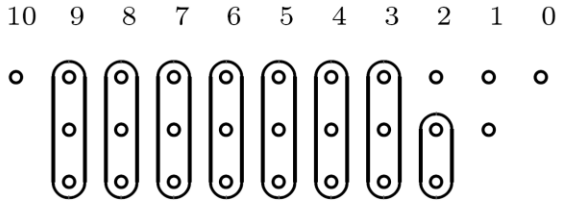

(a) Level 1 carry-save addition.




(b) Results of level 1.



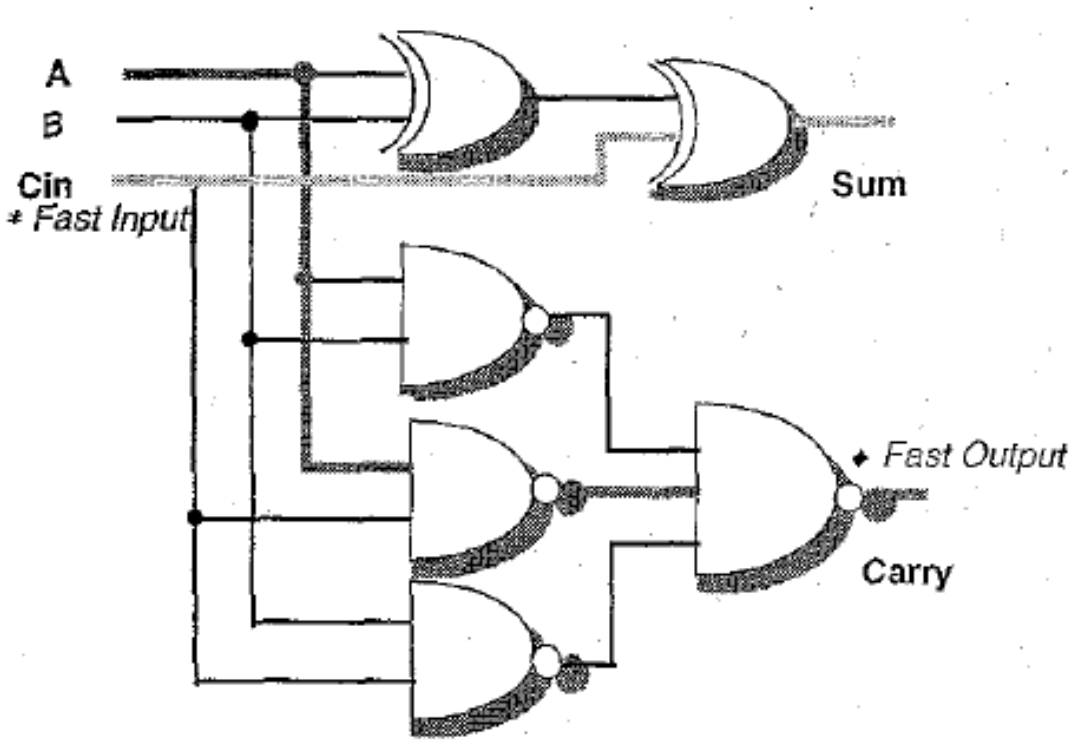

(c) Level 2 carry-save addition.



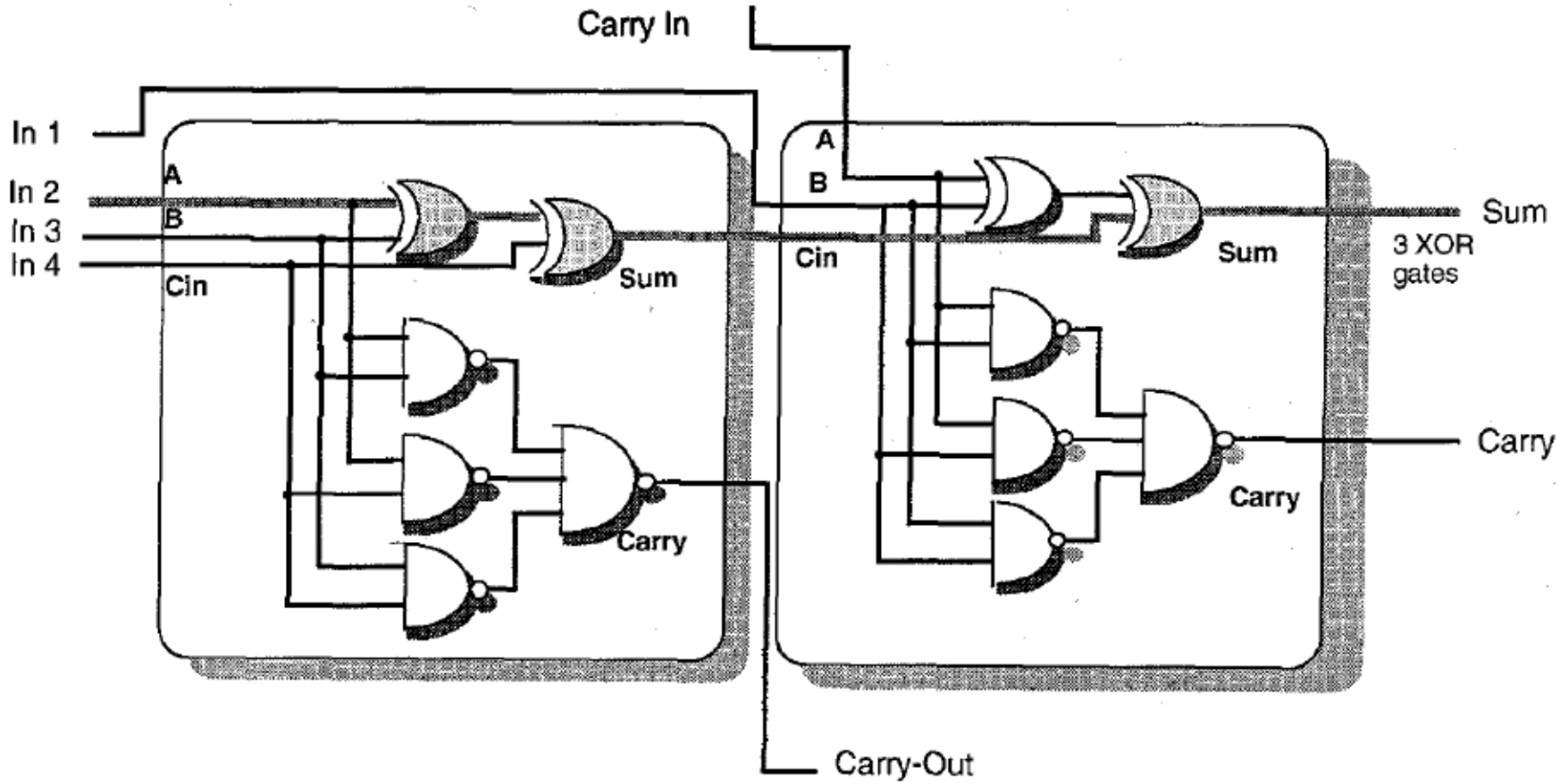

(d) Level 3 carry-save addition.

- Reduce # of bits to closest element of 3,4,6,9,13,19,…
- 15 (3,2) and 5 (2,2) vs. 16 (3,2) and 9 (2,2) counters

# Γρήγορες είσοδοι – γρήγορες έξοδοι

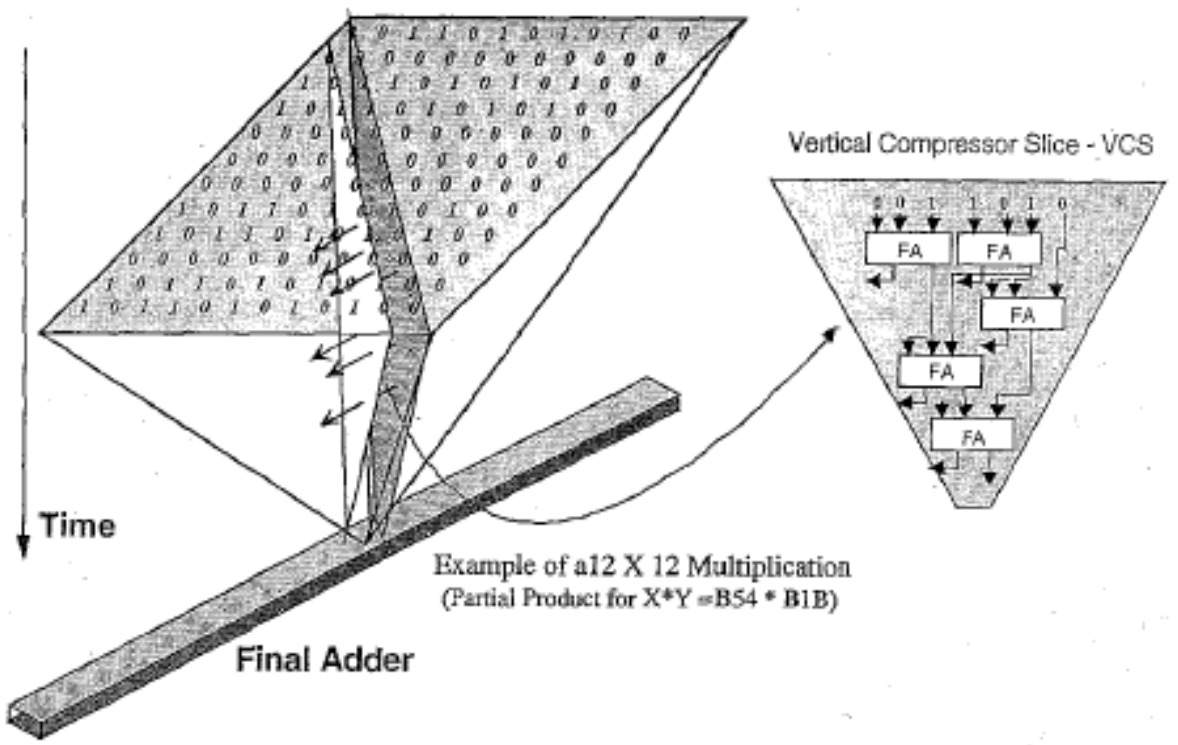


# Μια υλοποίηση του [4:2] compressor

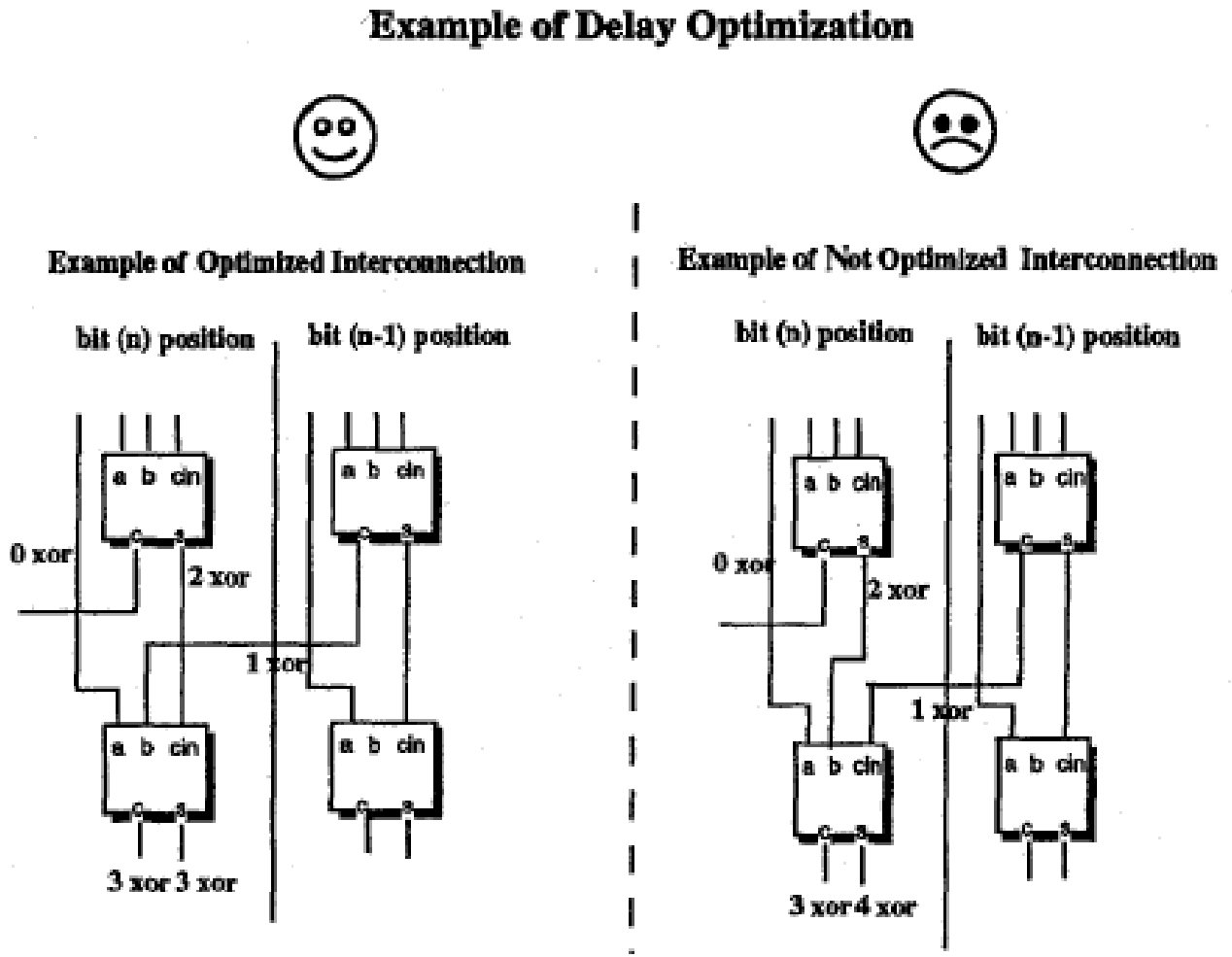



V.G. Oklobdzija, D.Villeger, and S.S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers*, pp. 294 – 306, March 1996.

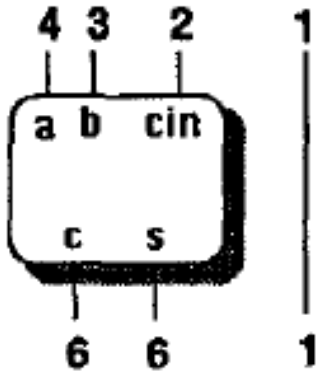
# 3D άποψη της αναγωγής μερικών γινομένων



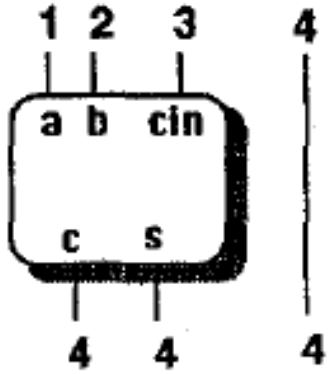
# Ελαχιστοποίηση καθυστέρησης με βέλτιστη ανάθεση εισόδων



# Μοντέλο καθυστέρησης πλήρους αθροιστή ενός ψηφίου




Worst Case




TDM Arrangement

$$\text{Delay}(S) = \text{MAX}\{\text{Delay}(A) + D_{A-S}, \text{Delay}(B) + D_{B-S}, \text{Delay}(C_{in}) + D_{Cin-S}\}$$

$$\text{Delay}(C) = \text{MAX}\{\text{Delay}(A) + D_{A-C}, \text{Delay}(B) + D_{B-C}, \text{Delay}(C_{in}) + D_{Cin-C}\}$$

# Αλγόριθμος κατασκευής δένδρου αναγωγής

**Αρχικοποίηση:**

Form 2N – 1 lists $L_i$ (i = 0,1, …,2N – 2) each consisting of $p_i$ elements where:

$p_i = i + 1$ for $i \leq N - 1$ and

$p_i = 2N - 1 - i$ for $i \geq N$

An element of a list $L_i$ (j = 0,1,..., $p_i$ – 1) is a pair: $\langle d_j, n_j \rangle_i$

where:

$n_j$ : is a unique node identifying name

$d_j$ : is a delay associated with that node representing a delay of a signal arriving to the node $n_j$ with respect to some reference point.

For i = 0, 1 and 2N - 2:

connect nodes from the corresponding lists Li directly to the CPA;

# Αλγόριθμος (2)

**Partial Product Array Generation:**
**For I = 2 to I = 2N - 3 Begin For**
  **if** length of $L_i$ is even **Then**
  **Begin If**
   Sort the elements of $L_i$ in ascending order by the values of delay d1;
  connect an HA to the first 2 elements of $L_i$ starting with the slowest input;
  Ds = max {$d_A$+$d_{A-s}$, $d_B$+$d_{B-s}$}
  Dc = max {dA+dA-c, dB+dB-c}
  remove 2 elements from $L_i$;
  insert the pair <Ds, NetName> into $L_i$;
  insert the pair <Dc, NetName> into $L_{i+1}$;
  decrement the length of $L_i$;
  increment the length of $L_{i+1}$;
  **End If;**

  **while** length of $L_i$ > 3
   **Begin While**
    sort the elements of $L_i$ in ascending order by the values of delay dl;
    connect an FA to the first 3 elements of $L_i$ starting with the slowest input of the FA:
    Ds = max { dcA + dcA-s,  dcB + dcB-s};
    Dc = max {dcA+dcA-c, dcB+dcB-c};
    remove 3 elements from $L_i$;
    insert the pair <Ds, NetName> into $L_i$;
    insert the pair <Dc, NetName> into $L_{i+1}$;
    subtract 2 from the length of $L_i$;
    increment the length of $L_{i+1}$;
   **End While;**
  sort the elements of $L_i$;
  connect an FA to the last 3 nodes of $L_i$;
  connect the S and C to the bit i and I + 1 of the CPA;
**End For;**
**End Method;**

# Παράδειγμα TDM

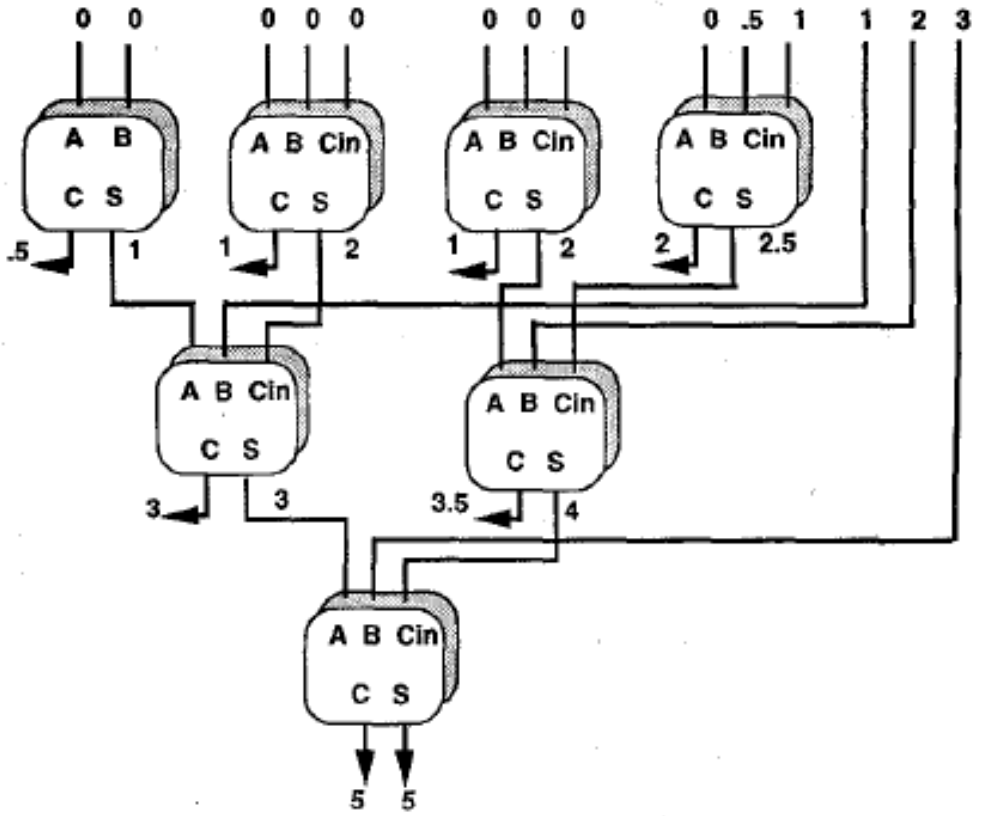


TABLE 1
COMPARISON BETWEEN TDM
AND OTHER REPRESENTATIVE SCHEMES,
IN XOR LEVELS USED IN THE PARTIAL PRODUCT ARRAY

| Multiplier Word-length | Wallace Tree [7] | 4:2 Tree [11]* | Fadavi-Ardekani [16] | TDM |
|---|---|---|---|---|
| 3 | 2 | 2 | 2 | 2 |
| 4 | 4 | 3 | 3 | 3 |
| 6 | 6 | 6 (5) | 5 | 5 |
| 8 | 8 | 6 | 7 | 5 |
| 9 | 8 | 8 | 7 | 6 |
| 11 | 10 | 9 (8) | 8 | 7 |
| 12 | 10 | 9 (8) | 8 | 7 |
| 16 | 12 | 9 | 10 | 8 |
| 19 | 12 | 12 (11) | 11 | 9 |
| 24 | 14 | 12 (11) | 12 | 10 |
| 32 | 16 | 12 | 13 | 11 |
| 42 | 16 | 15 (14) | 14 | 12 |
| 53 | 18 | 15 | 15 | 13 |
| 64 | 20 | 15 | 16 | 14 |
| 95 | 20 | 18 (17) | 17 | 15 |

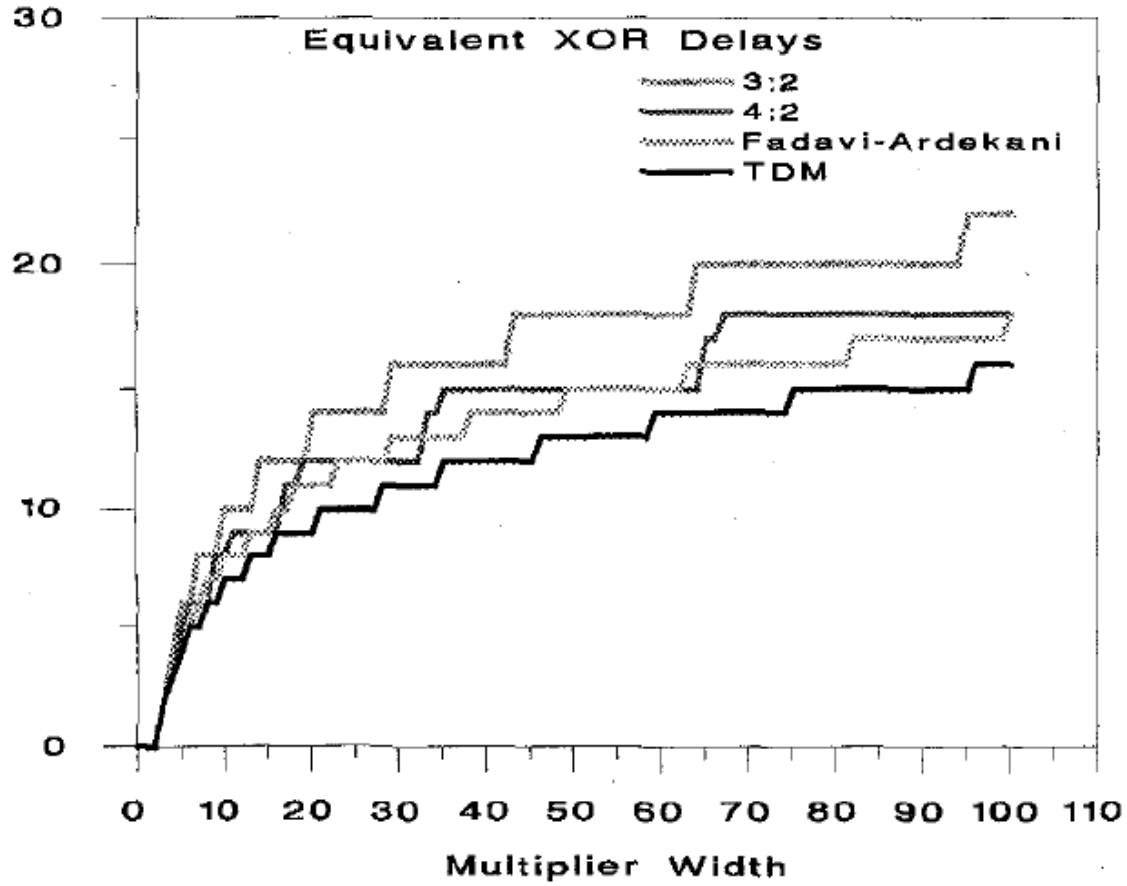
* *Number in parenthesis represent delays when a Full Adder is used (instead of 4:2 compressor) every time the column size is found to be three.*

Μέγιστη καθυστέρηση διαδρομής

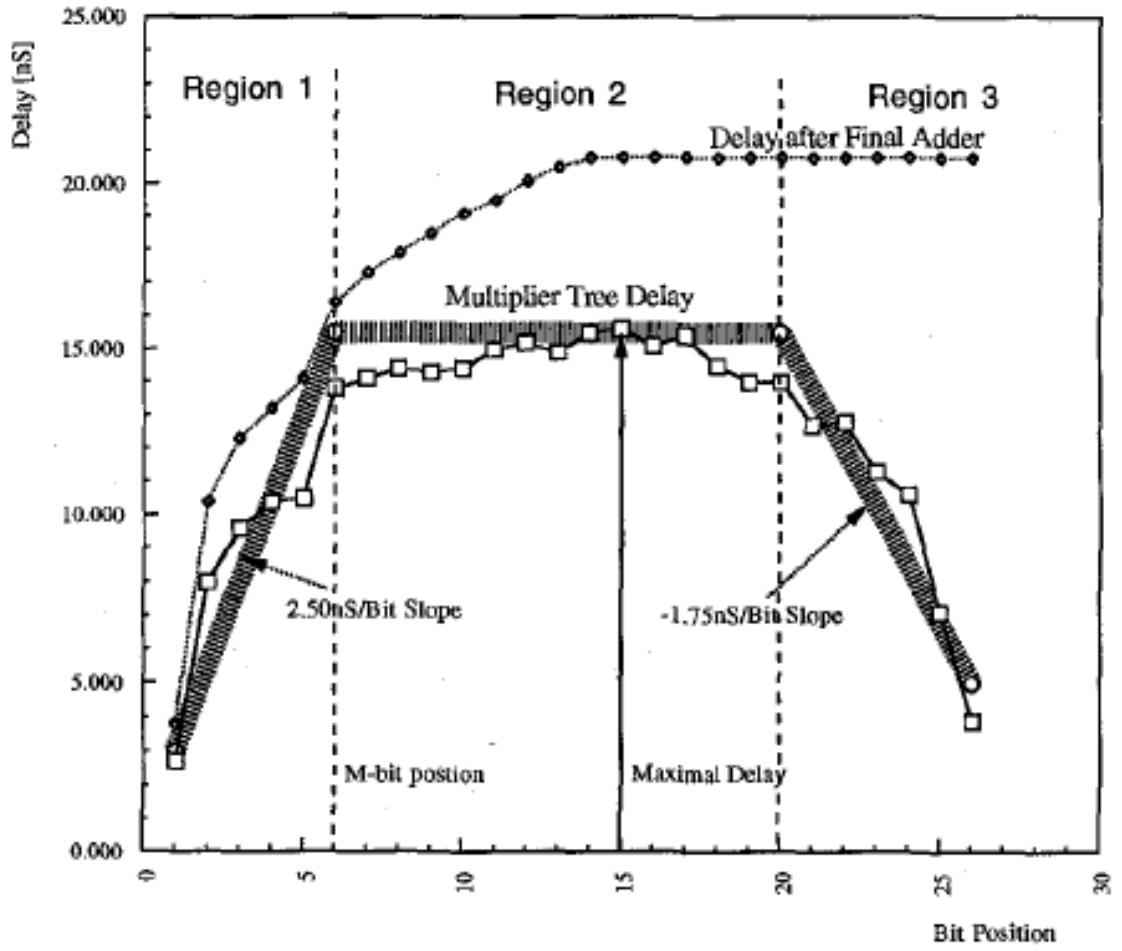
TABLE 2
CRITICAL PATH DELAY [CMOS: LEFF = 1μ, T = 25°C, $V_{cc}$ = 5V]

| N = 24 bits | 4:2 Design | 9:2 Design | Fadavi-Ardekani | TDM Design |
|---|---|---|---|---|
| Delay [nS] | 14.0 | 13.0 | 11.7 | 10.5 |

# Σύγκριση (3,2), [4:2], F-A, TDM

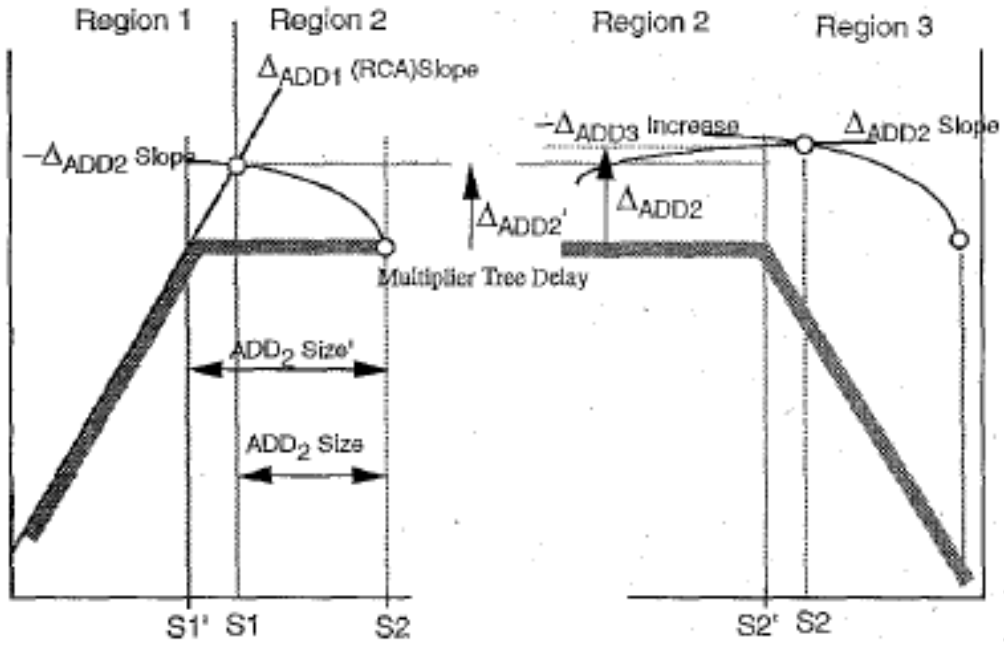


# Ο ρόλος του τελικού αθροιστή

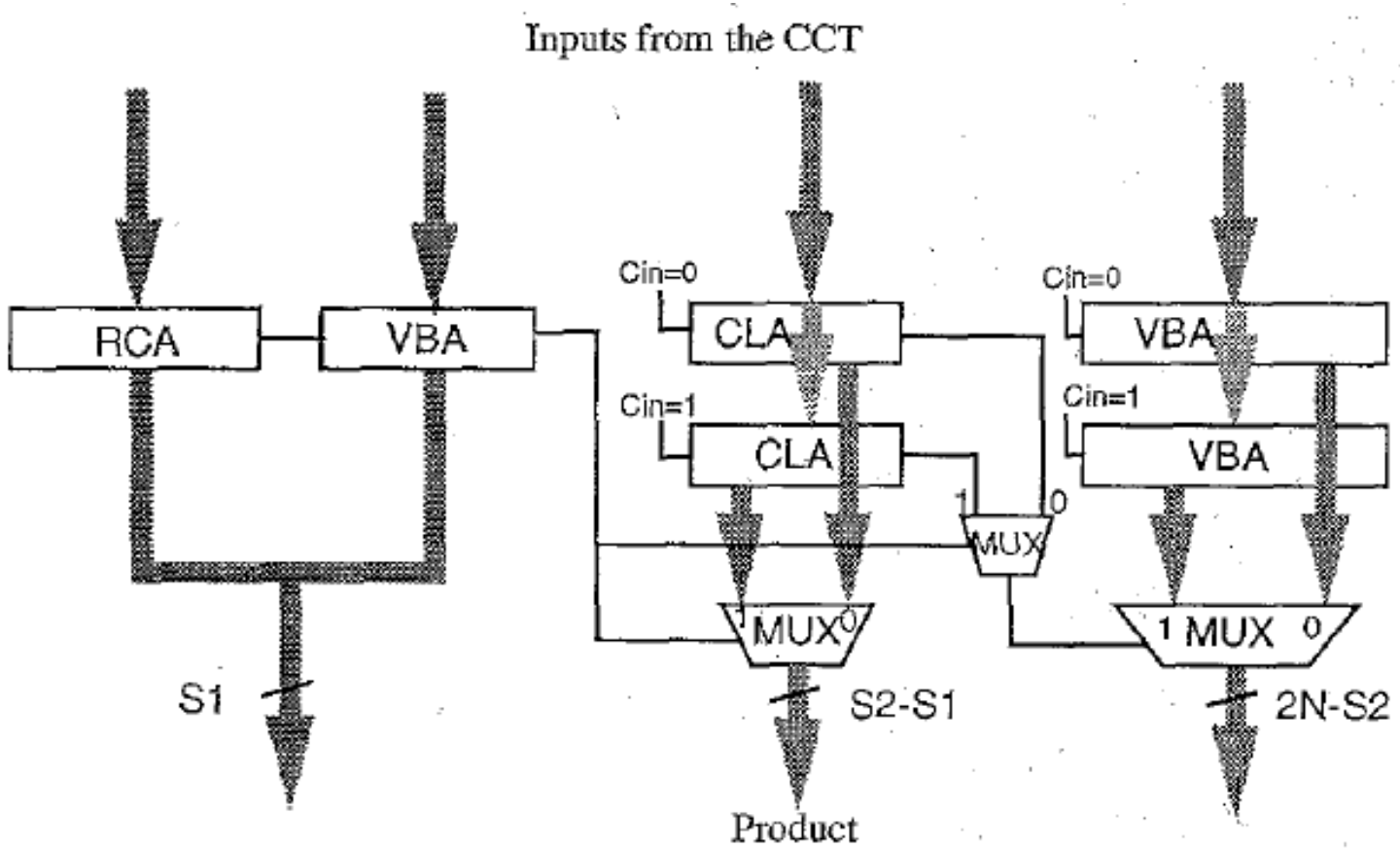


# Οργάνωση CPA: Καθορισμός περιοχών

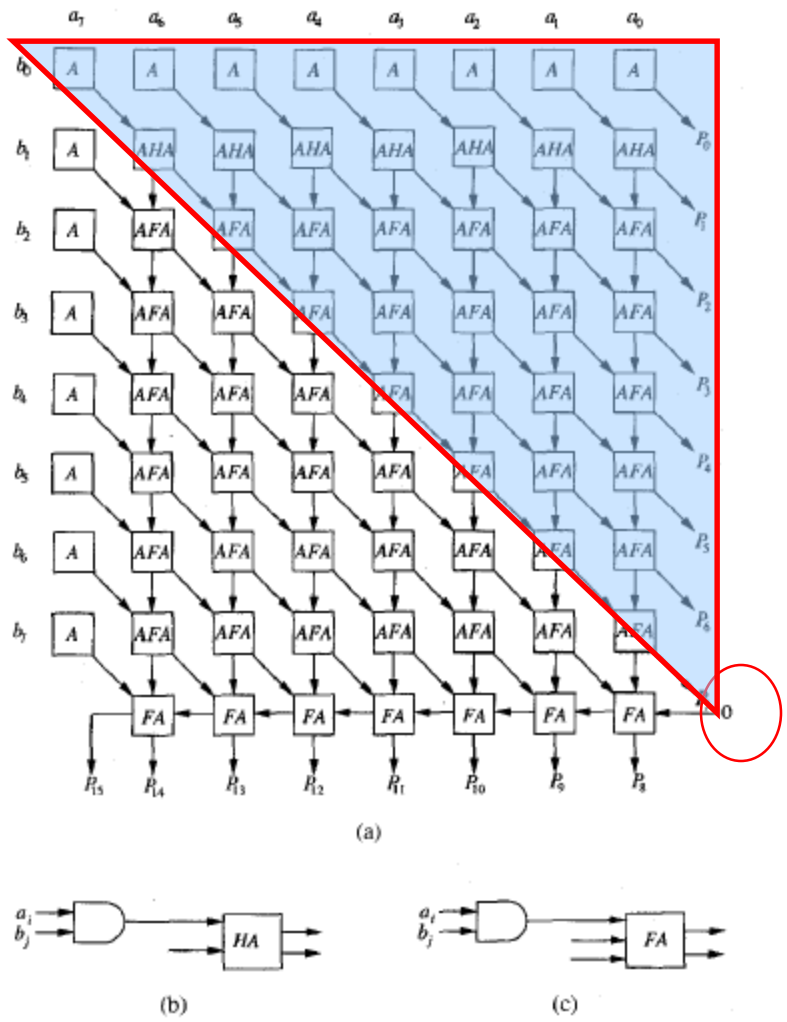
- <u>Περιοχή 1:</u> Γραμμική καθυστέρηση με τη θέση του δυαδικού ψηφίου.
- <u>Περιοχή 2:</u> Τα ψηφία καταφθάνουν περίπου ταυτόχρονα.
- <u>Περιοχή 3:</u> Τα περισσότερο σημαντικά ψηφία καταφθάνουν νωρίτερα.

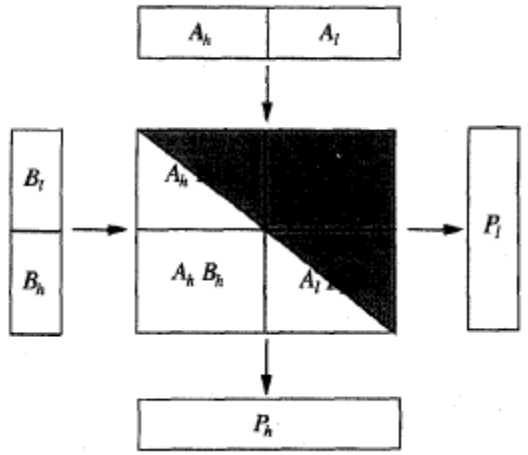


# Τελική οργάνωση CPA



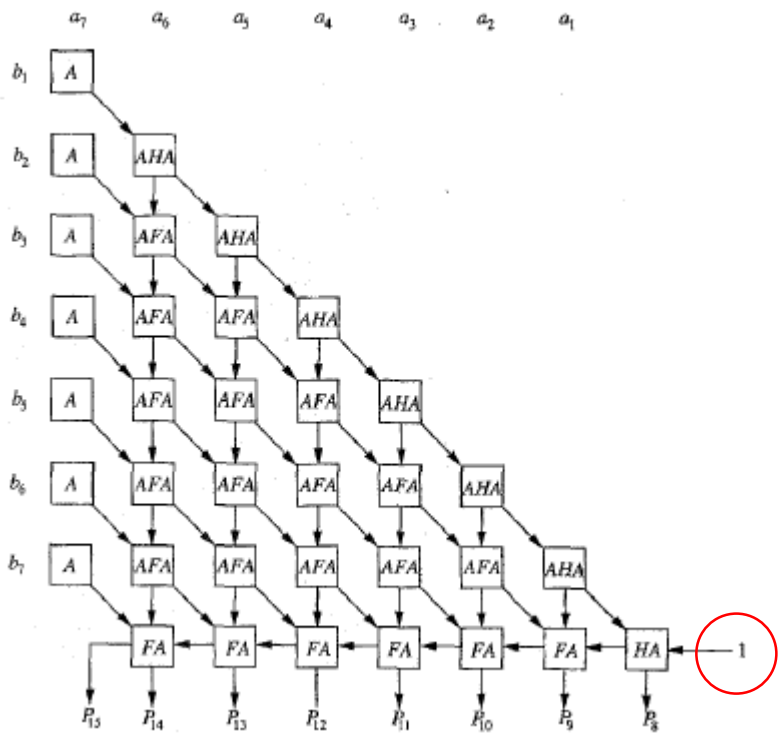
# Ο πολλαπλασιαστής των Kidambi, El-Guibaly και Antoniou[1]



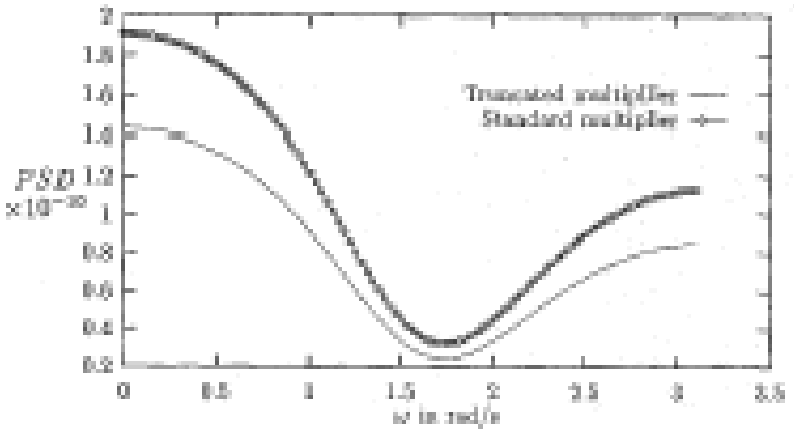



[1]"Area-Efficient Multipliers for Digital Signal Processing Applications," *IEEE Transactions on Circuits and Systems – Part II*, 43(2), σελ. 90 - 95, Feb. 96.

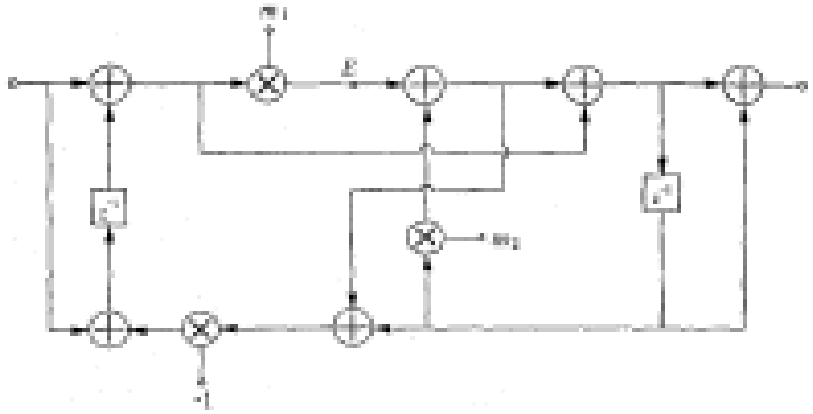
# Πολλαπλασιαστές Σταθερού Μήκους



# Επιπτώσεις σε φίλτρα



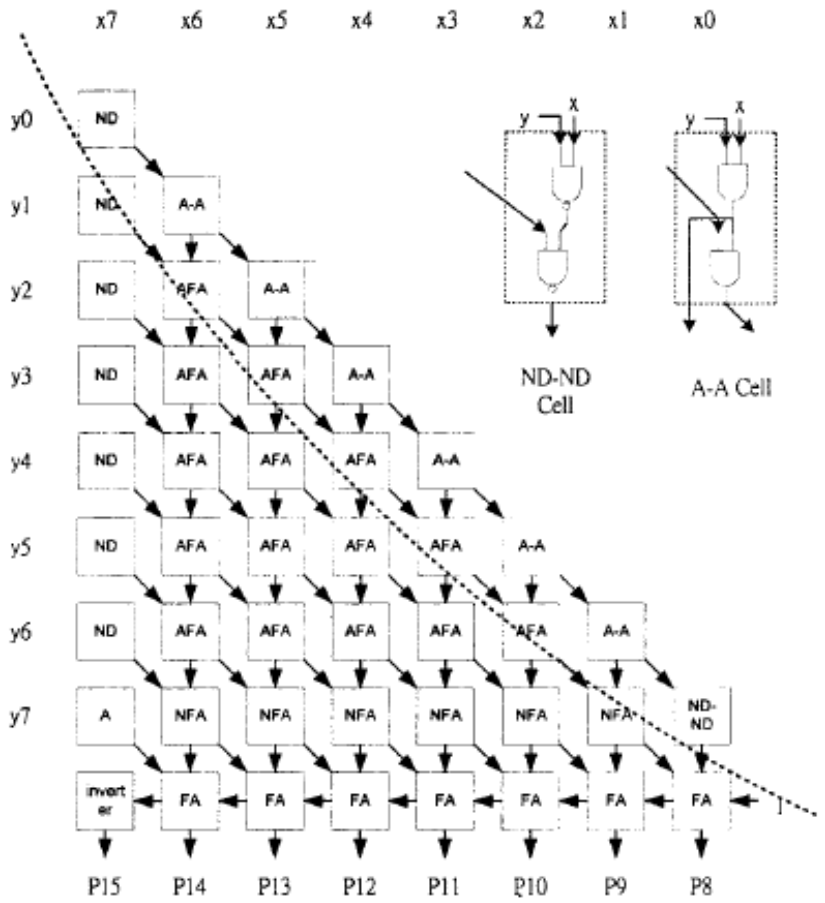

- Εφαρμογή του συγκεκριμένου πολλαπλασιαστή (16×16) σε κατωδιαβατό Butterworth wave φίλτρο δεύτερης τάξης και δομής GIC (generalized-immitance converter)



- A. Antoniou, *Digital filters analysis and design,* McGraw Hill, 1993.

# Το ίδιο πρόβλημα από άλλη πλευρά...[2]

- [2] Van, Wang, και Feng, “Design of the lower-error fixed-width multiplier and its application,” *IEEE Transactions on Circuits and Systems – Part II*, 47(10), Oct 2000.



# Επιδόσεις του δεύτερου πολλαπλασιαστή σε ψηφιακά φίλτρα φωνής

FIR φίλτρο

35 συντελεστών

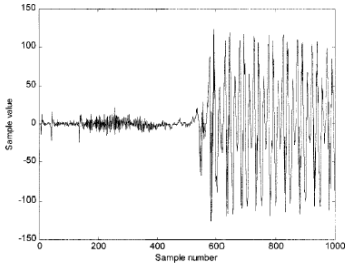



Fig. 7. Original input voice signal with 1000 samples.

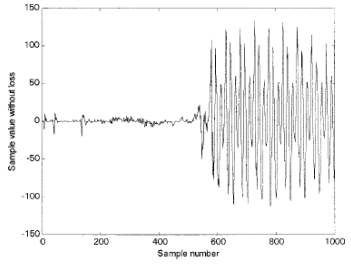



Fig. 8. Standard voice output signal without loss.

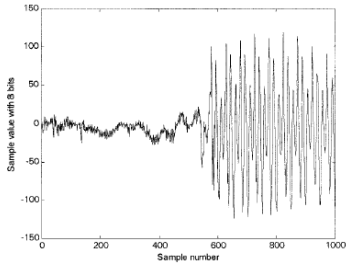



Fig. 9. Output signals using K-G-As' structure [6].

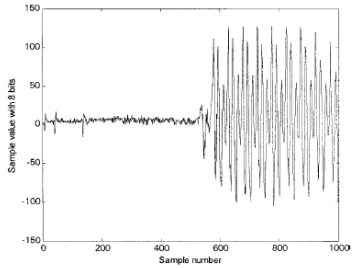
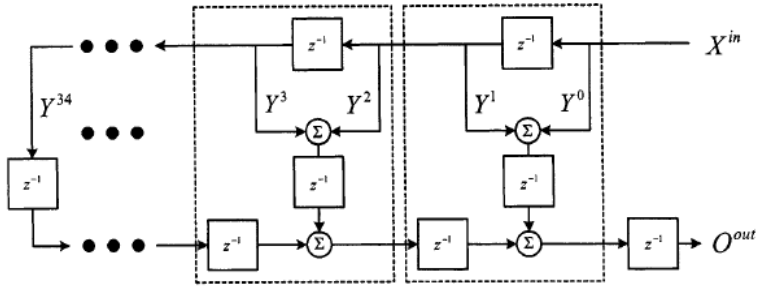



Fig. 11. Output signals using the proposed structure.



# Κατανεμημένη (Distributed) Αριθμητική

- Υπολογισμός εσωτερικού γινομένου σταθερού διανύσματος $C = [c_1 c_2 \dots c_N]$ με μεταβλητό διάνυσμα $X = [x_1\ x_2 \dots x_N]$.

$$Y = C \cdot X = \sum_{i=1}^{N-1} c_i x_i$$

- Εφαρμογές: φίλτρα σταθερών συντελεστών, διακριτοί μετασχηματισμοί όπως ο DCT…

# Παράδειγμα Κατανεμημένης Αριθμητικής

$$(001)_2 = 0\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$
$$= (0.125)_{10}$$

$$Y = c_0x_0 + c_1x_1 + c_2x_2 + c_3x_3$$

$$Y = c_0(001)_2 + c_1(010)_2 + c_2(110)_2 + c_3(101)_2$$

$$= 2^{-1}(c_0 0 + c_1 0 + c_2 1 + c_3 1) +$$
$$+2^{-2}(c_0 0 + c_1 1 + c_2 1 + c_3 0) +$$
$$+2^{-3}(c_0 1 + c_1 0 + c_2 0 + c_3 1)$$
$$= 2^{-1} f(0,0,1,1) + 2^{-2} f(0,1,1,0) + 2^{-3} f(1,0,0,1)$$

$$f(x_{0,j}, x_{1,j}, x_{2,j}, x_{3,j}) \triangleq c_0x_{0,j} + c_1x_{1,j} + c_2x_{2,j} + c_3x_{3,j}$$

Τρεις κύκλοι ρολογιού

Δεν απαιτούνται πολλαπλασιασμοί.

| $x_{0,j}$ | $x_{1,j}$ | $x_{2,j}$ | $x_{3,j}$ | Περιεχόμενα Θέσης Μνήμης |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | $c_3$ |
| 0 | 0 | 1 | 0 | $c_2$ |
| 0 | 0 | 1 | 1 | $c_2 + c_3$ |
| 0 | 1 | 0 | 0 | $c_1$ |
| 0 | 1 | 0 | 1 | $c_1 + c_3$ |
| 0 | 1 | 1 | 0 | $c_1 + c_2$ |
| 0 | 1 | 1 | 1 | $c_1 + c_2 + c_3$ |
| 1 | 0 | 0 | 0 | $c_0$ |
| 1 | 0 | 0 | 1 | $c_0 + c_3$ |
| 1 | 0 | 1 | 0 | $c_0 + c_2$ |
| 1 | 0 | 1 | 1 | $c_0 + c_2 + c_3$ |
| 1 | 1 | 0 | 0 | $c_0 + c_1$ |
| 1 | 1 | 0 | 1 | $c_0 + c_1 + c_3$ |
| 1 | 1 | 1 | 0 | $c_0 + c_1 + c_2$ |
| 1 | 1 | 1 | 1 | $c_0 + c_1 + c_2 + c_3$ |

# Αρχές Κατανεμημένης Αριθμητικής

- $c_i$: $M$-bit σταθερές
- $x_i$: $W$-bit δεδομένα συμπληρώματος δύο
- $x_{i,k}$ είναι 0 ή 1

$$\left.\begin{aligned} x_i &= -x_{i,W-1} + \sum_{j=1}^{W-1} x_{i,W-1} 2^{-j} \\ Y &= \sum_{i=0}^{N-1} c_i x_i \end{aligned}\right\} \Rightarrow Y = \sum_{i=0}^{N-1} c_i \left( -x_{i,W-1} + \sum_{j=1}^{W-1} x_{(i,W-1)}\, 2^{-j} \right) = -\sum_{i=0}^{N-1} c_i x_{i,W-1} + \sum_{i=0}^{N-1} \sum_{j=1}^{W-1} c_i x_{i,W-1} 2^{-j}$$

$$Y = \sum_{i=0}^{N-1} c_i x_i = \sum_{j=0}^{W-1} C_{W-1-j}\ 2^{-j},$$

$$C_{W-1-j} = \sum_{i=1}^{N-1} c_i x_{i,W-1-j},\ j \neq 0$$

$$C_{W-1} = -\sum_{i-0}^{N-1} c_i x_{i,W-1}$$

# Αλγόριθμος Εσωτερικού Γινομένου με Κατανεμημένη Αριθμητική

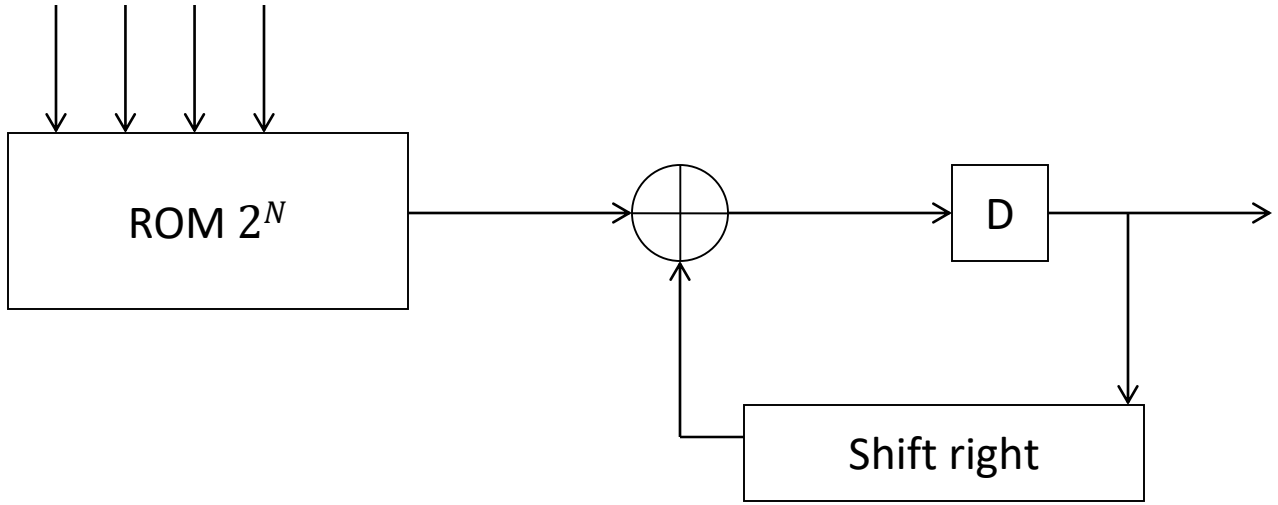
$$Y = \sum_{i=0}^{N-1} c_i x_i = \sum_{j=0}^{W-1} C_{W-1-j}\ 2^{-j},$$

$$C_{W-1-j} = \sum_{i=1}^{N-1} c_i x_{i,W-1-j},\ j \neq 0$$

$$C_{W-1} = -\sum_{i-0}^{N-1} c_i x_{i,W-1}$$

- **Παρατηρήσεις**
  - Οι $W$ σταθερές $C_j$ εξαρτώνται από τα $N$ bits $x_{i,j}$
  - Κάθε $C_j$ μπορεί να λάβει $2^N$ διαφορετικές τιμές.
  - Το εσωτερικό γινόμενο $Y$ προκύπτει ως άθροισμα ολισθημένων $C_j$.

# Τυπική Αρχιτεκτονική Κατανεμημένης Αριθμητικής




- Σε $W$ επαναλήψεις υπολογίζεται το εσωτερικό γινόμενο, *χωρίς πολλαπλασιασμούς.*

# Μείωση μεγέθους μνημών σε κατανεμημένη αριθμητική

$$F(a,b,c,d,e,f,g,h) = (a \cdot c_0 + b \cdot c_1 + c \cdot c_2 + d \cdot c_3) + (e \cdot c_4 + f \cdot c_5 + g \cdot c_6 + h \cdot c_7)$$
$$= F_1(a,b,c,d) + F_2(e,f,g,h)$$