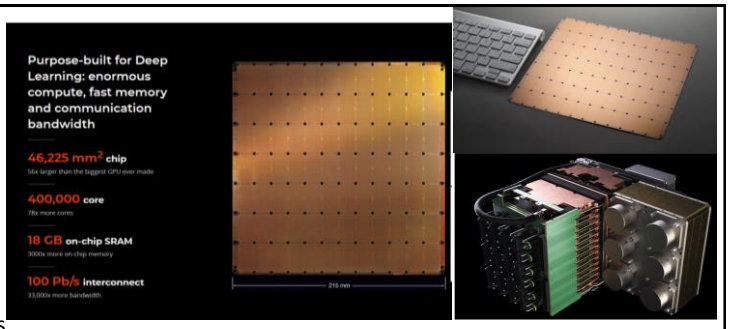
# Σχεδιασμός Ολοκληρωμένων Κυκλωμάτων
# VLSI – II

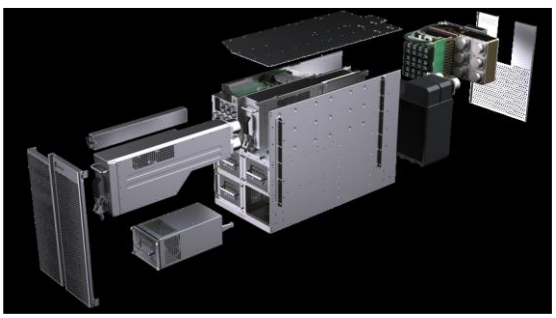


1

## Hardware
## Τεχνολογικές προκλήσεις




- Cerebras → WSE 1.2 τρισεκατομμύρια transistors
  - 86 δις νευρώνες στον ανθρώπινο εγκέφαλο
  - 500 δις άστρα στον γαλαξία
  - ~$1.2 trillion η χρηματιστηριακή αξία της Apple@2018!
- Κόστος, επιδόσεις, πολυπλοκότητα, ενέργεια/ισχύς,…

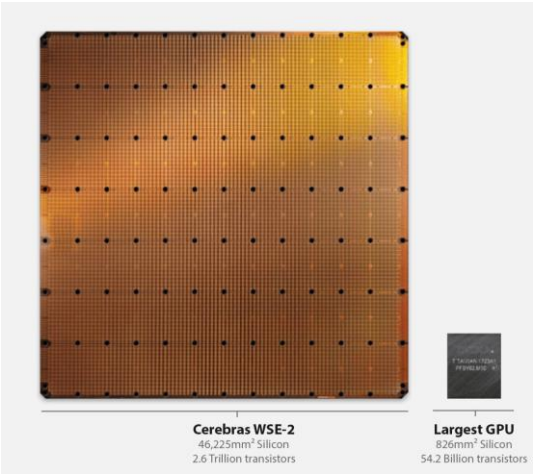


http://en.wikipedia.org/wiki/Transistor_count

Πηγή: https://www.cerebras.net/

2

# Εξέλιξη @2022




**Cerebras Wafer-Scale Engine**

*Fabrication process*
**7nm**

*Silicon area*
**46,225mm²**

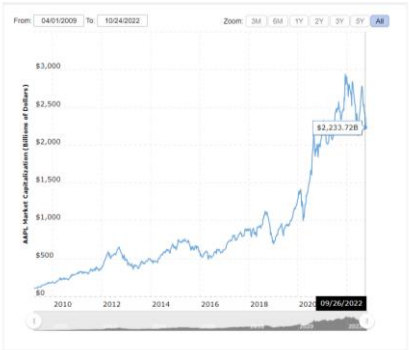
*Transistors*
**2.6 Trillion**

*AI-optimized cores*
**850,000**

*Memory (on-chip)*
**40GB**

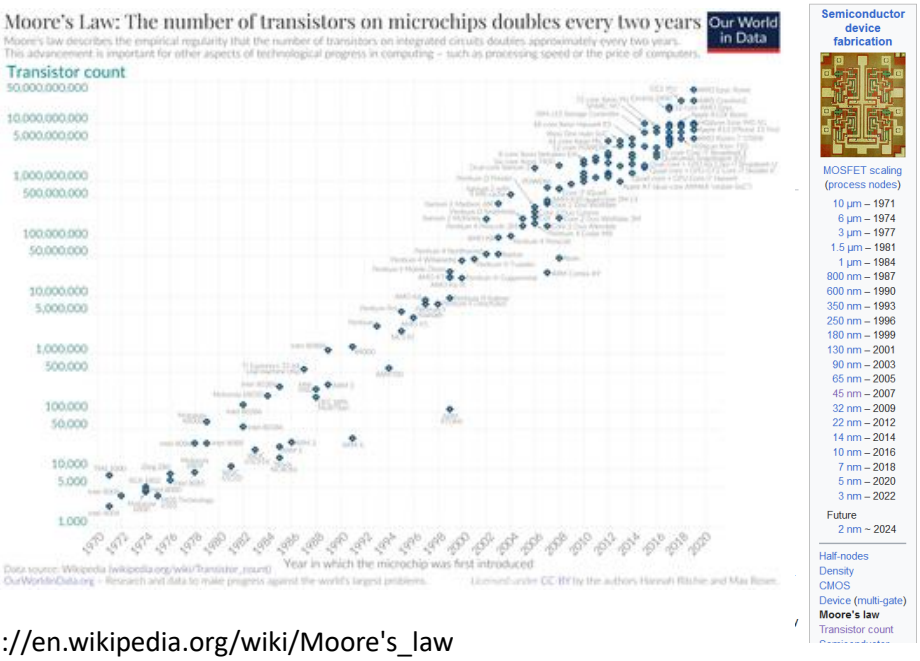
*Memory bandwidth*
**20PB/s**

*Fabric bandwidth*
**220Pb/s**




3

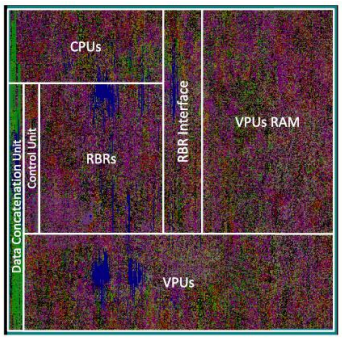
# Εξέλιξη chips

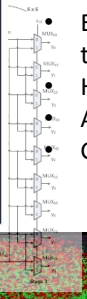


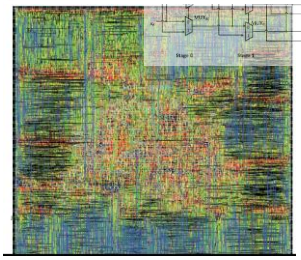

Wikipedia: https://en.wikipedia.org/wiki/Moore's_law

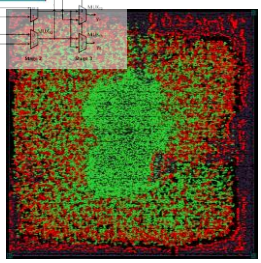
4

## Ψηφιακά συστήματα μεγάλης κλίμακας ολοκλήρωσης

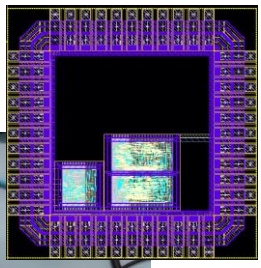


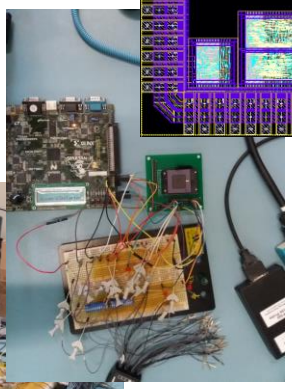



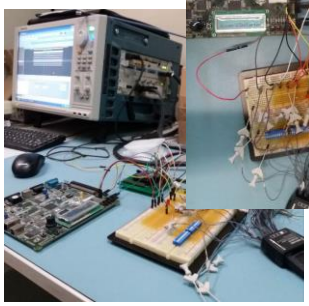




- Ολοκληρωμένα ψηφιακά συστήματα VLSI σε τεχνολογίες 90 nm, 65 nm, 45 nm, 28 nm, 22 nm
- Επεξεργαστές ειδικού σκοπού, telecom, video, secure/crypto,…
- Hardware για AI και machine learning ASIC, MPSoC, Ultrascale FPGAs,…
- GPUs, Jetson Xavier,…

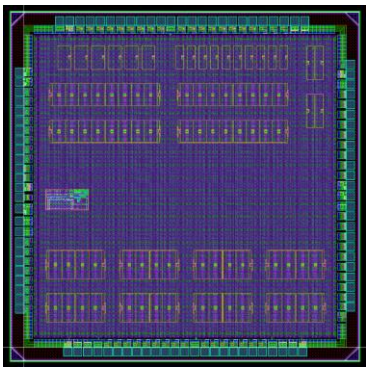


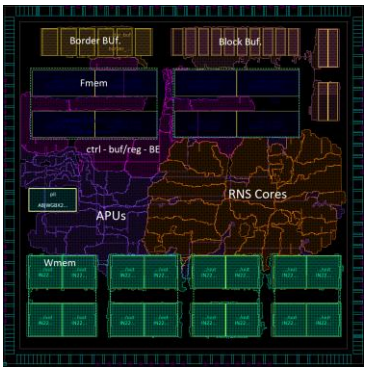


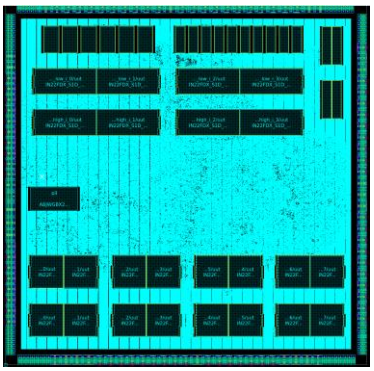


5

## Ένας επιταχυντής μηχανικής μάθησης



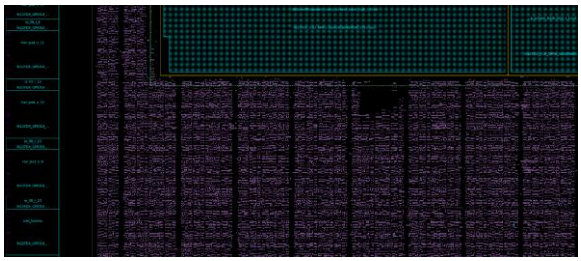


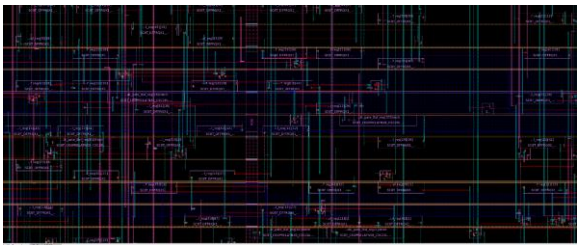



GF 22nm

6

# Υλοποίηση με standard-cell library

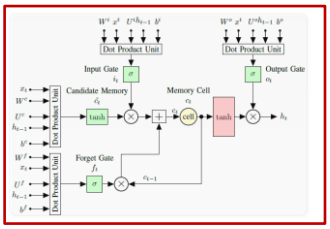


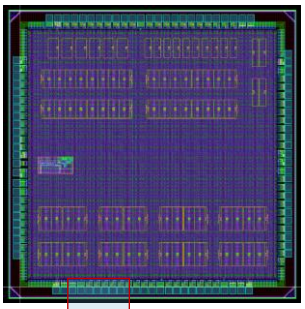


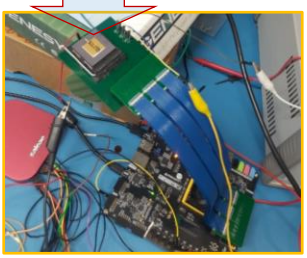
7

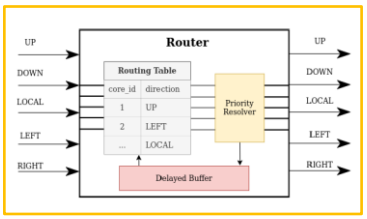
# Research on hardware for machine learning

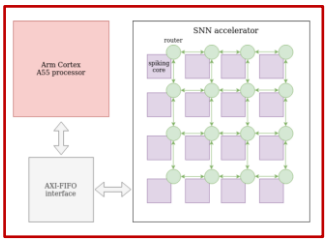
- Hardware accelerators for machine learning
  - ASIC designs @ 22 nm
  - Collaboration with KU, UAE
  - FPGA – MPSoC designs









8

# European Chips Act


https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-chips-act_en

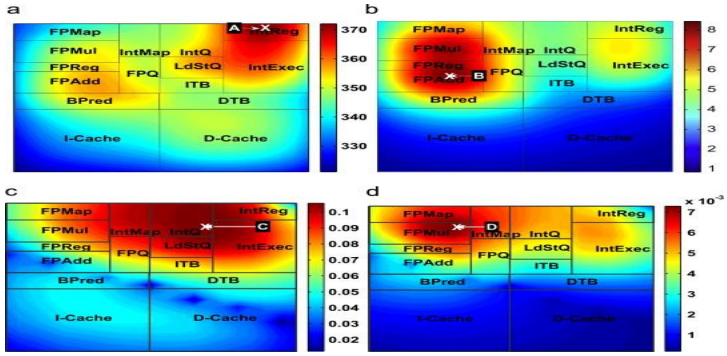
9

# Υποσυστήματα χειρισμού Δεδομένων

- Περίγραμμα Διάλεξης
  - Κυκλώματα για πρόσθεση/αφαίρεση
  - Ανιχνευτές 1/0
  - Συγκριτές
  - Μετρητές
  - Κωδικοποίηση
  - Ολισθητές
  - Πολλαπλασιασμός



10



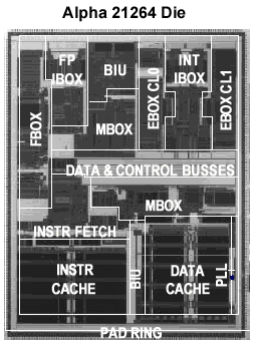




Fig.?5 Statistical profile of the temperature and voltage drop of a 16,642 node power grid across an Alpha 21<ce:hsp sp="0.25"/> 364 CPU core: (a) temperature expected value, (b) standard deviation of the temperature, (c) expected value of the vo...

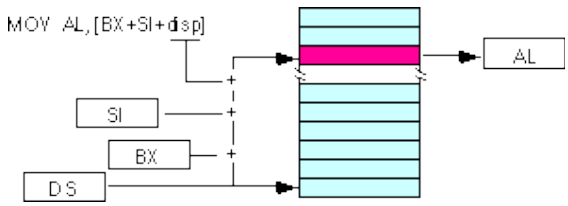
Kian Haghdad , Javid Jaffari , Mohab Anis

**Power grid analysis and verification considering temperature variations**

Microelectronics Journal Volume 43, Issue 3 2012 189 - 197

http://dx.doi.org/10.1016/j.mejo.2011.12.008

11




mov al, disp[bx][si]

ADD AL, -3



12

# Χρήσεις - Αθροιστές

## Γενικού σκοπού

- ALUs
- Παραγωγή διευθύνσεων σε συστήματα μνήμης
- Ακέραιες μονάδες και
- Μονάδες πράξεων κινητής υποδιαστολής

## Ειδικού σκοπού

13

13

# Βάρη ψηφίων

| | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|
| | 0 | 0 | 1 | 0 | 1 | 0 |
| + | 0 | 1 | 0 | 1 | 1 | 1 |
| | 1 | 0 | 0 | 0 | 0 | 1 |

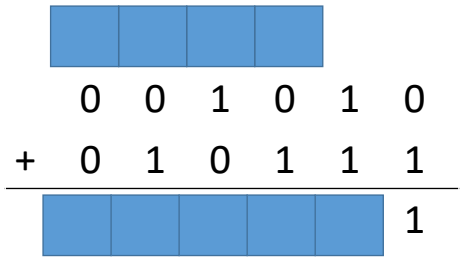
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | $2^3$ | 0 | $2^1$ | 0 |
| 0 | $2^4$ | 0 | $2^2$ | $2^1$ | $2^0$ |
| $2^5$ | 0 | 0 | 0 | 0 | $2^0$ |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 8 | 0 | 2 | 0 |
| 0 | 16 | 0 | 4 | 2 | 1 |
| 32 | 0 | 0 | 0 | 0 | 1 |

| | |
|---|---|
| | 10 |
| + | 23 |
| | 33 |

14

# Άθροιση δυαδικών




ακέραιοι χωρίς πρόσημο

15

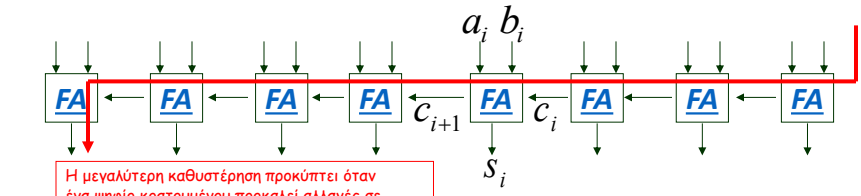
# Δυαδικός πολλαπλασιασμός

```
          101100    Πολλαπλασιαστέος
        ×   1011    Πολλαπλασιαστής
          101100
         101100          Μερικά γινόμενα
        000000
    +  101100
       111100100    Γινόμενο
```

$(101100)_2 \times (1011)_2 = 44 \times 11 = 484 = (111100100)_2$

16

# Αθροιστής Κυματισμού Κρατουμένου (Ripple Carry)




Η μεγαλύτερη καθυστέρηση προκύπτει όταν ένα ψηφίο κρατουμένου προκαλεί αλλαγές σε όλα τα ψηφία μέχρις και το περισσότερο σημαντικό του αποτελέσματος.

Καθυστέρηση ενός FA

$$A = nA_{\text{FA}} \qquad T_d \approx nT_{\text{fa}}$$

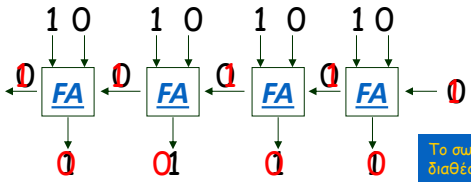
Εμβαδό ενός FA

Η πολυπλοκότητα της υλοποίησης εξαρτάται γραμμικά από το μήκος λέξης $n$ των όρων.

Πλήθος FAs

20

# Μεγαλύτερη Καθυστέρηση σε Αθροιστή Κυματισμού 4 Δυαδικών Ψηφίων




Το σωστό αποτέλεσμα είναι διαθέσιμο μετά από ($T_{FA}$)

Θεωρήστε ότι θέλουμε να προσθέσουμε τους δυαδικούς 1111 και 0000 με κρατούμενο εισόδου 0.

Μετά από ικανό χρονικό διάστημα τα ενδιάμεσα κρατούμενα γίνονται 0 και η έξοδος 1111.

Έστω τώρα ότι το κρατούμενο εισόδου γίνεται 1.

Μετά από χρόνο ίσο με την χρονική καθυστέρηση ενός FA ($T_{FA}$), το κρατούμενο εξόδου του λιγότερο σημαντικού αθροιστή γίνεται 1, και το ψηφίο αθροίσματος 0. Το συνολικό αποτέλεσμα είναι τώρα 1110

Μετά από $2T_{FA}$, το κρατούμενο εξόδου του δεύτερου λιγότερο σημαντικού αθροιστή γίνεται 1, και το ψηφίο αθροίσματος 0. Το συνολικό αποτέλεσμα είναι τώρα 1100

Μετά από $3T_{FA}$, το κρατούμενο εξόδου του τρίτου λιγότερο σημαντικού αθροιστή γίνεται 1, και το ψηφίο αθροίσματος 0. Το συνολικό αποτέλεσμα είναι τώρα 1000

Μετά από $4T_{FA}$, το κρατούμενο εξόδου του περισσότερο σημαντικού αθροιστή γίνεται 1, και το ψηφίο αθροίσματος 0. Το συνολικό αποτέλεσμα είναι τώρα 0000

21

# Επιτάχυνση Διάδοσης του Κρατουμένου

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$
$$= a_i b_i + (a_i + b_i) c_i$$

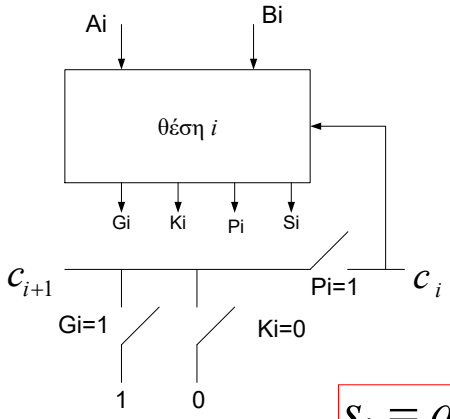
Το ψηφίο κρατουμένου εκφράζεται συναρτήσει νέων σημάτων ελέγχου.

$$\left.\begin{array}{l} p_i \triangleq a_i + b_i \\ g_i \triangleq a_i b_i \end{array}\right\} \Rightarrow c_{i+1} = g_i + p_i c_i$$

$$k_i \triangleq \overline{a_i}\,\overline{b_i}$$

22

# Η Τεχνική Διάδοσης Κρατουμένου Manchester Carry Chain




$$s_i = a_i \oplus b_i \oplus c_i$$

23

# Η Τεχνική Πρόβλεψης Κρατουμένου (carry look ahead)

Ξεκινώντας από τον αναδρομικό ορισμό του κρατούμενου, εκτελούμε *unrolling*

$$c_{i+1} = g_i + p_i c_i$$

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 c_1 = g_1 + p_1 \left( g_0 + p_0 c_0 \right) = g_1 + p_1 g_0 + p_1 p_0 c_0$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 \left( g_1 + p_1 g_0 + p_1 p_0 c_0 \right)$$

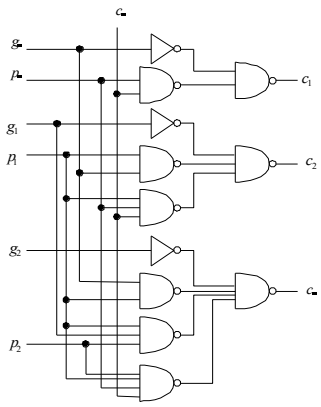
$$= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$\vdots$$

$$c_{i+1} = g_i + \sum_{j=0}^{i-1} \left( \prod_{k=j+1}^{i} p_k \right) g_j + \prod_{k=0}^{i} p_k c_0$$



# Αθροιστής Πρόβλεψης Κρατουμένου




- $p_{n/2} \rightarrow n(n+2)/4$ πύλες.
- Αριθμός transistors

  $\left( n^3 + 9n^2 + 20n \right) / 3$

  [(*i*+2)(*i*+5) tr ανά ψηφιακή θέση]
- Καθυστέρηση (με fanin)

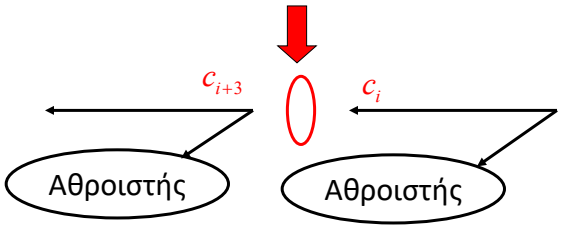
  $T_D = t_L \left( 0.25n^2 + 4.4n + 33.1 \right)$

  $t_L$, καθυστέρηση αντιστροφέα

# Group Carry Look-Ahead

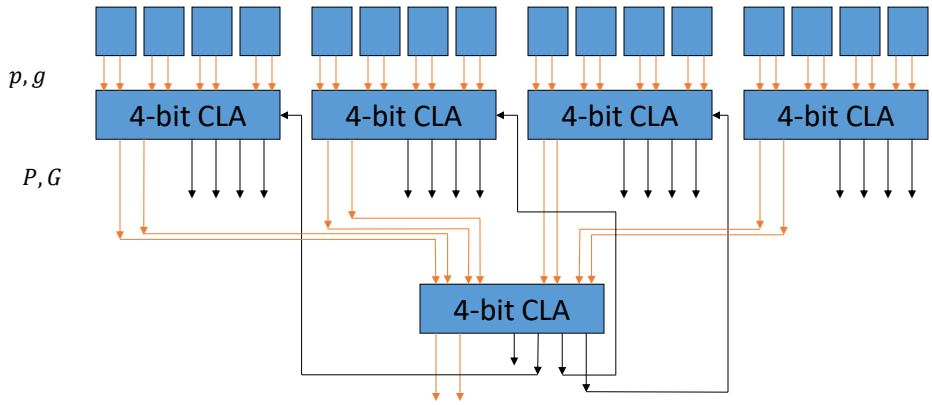
$$G_{i+3:i} = g_{i+3} + g_{i+2}p_{i+3} + g_{i+1}p_{i+2}p_{i+3} + g_i p_{i+1}p_{i+2}p_{i+3}$$
$$P_{i+3:i} = p_i p_{i+1}p_{i+2}p_{i+3}$$

$$c_{i+3} = G_{i+3:i} + P_{i+3:i}c_i$$




26

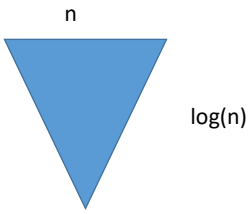
## Πρόβλεψη Κρατουμένου Πολλών Επιπέδων




$$T = 4\log_4 n + 1$$

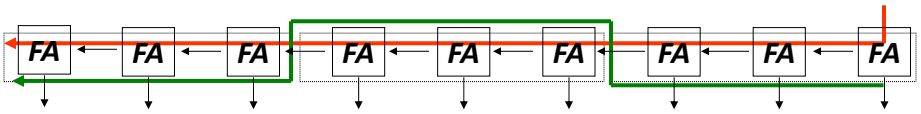
27

- p, g-> 1bit
- P, G-> 4 bit = 4* 1  level = 1 => 4^1 bits
- P’,G’->4 * 4  = 16, level = 2 => 4^2 bits
- P’’,G’’-> 4 * 4 * 4 = 64, l=3 => 4^3 bits
- P’’’,G’’’-> 4 * 4 * 4 * 4 = 256, l=4 => 4^4 bits




28

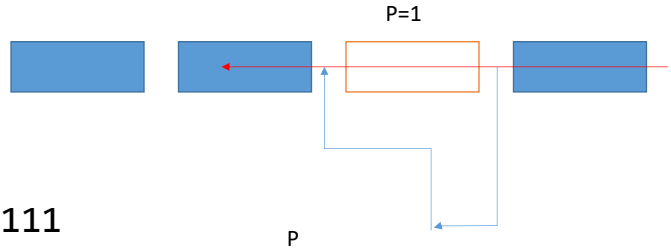
# Ο Αθροιστής Carry-skip




$$T_d = 2T_{add} + \left(\left\lceil \frac{n}{r} \right\rceil - 2\right) T_{skip} = 4(r-1) + 2\left(\left\lceil \frac{n}{r} \right\rceil - 2\right)$$
$$= 4(r-2) + 2\left\lceil \frac{n}{r} \right\rceil$$

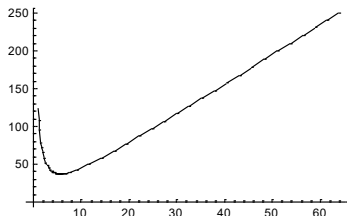
*n*, μήκος λέξης αθροιστή
*r*, μήκος μπλοκ

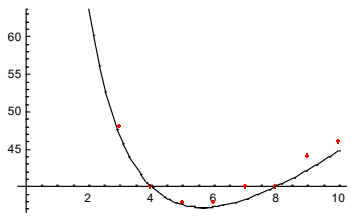
29




- 11111
- A=101011
- B=010101
- 0

30




$$T_d = 4(r-2) + 2\frac{n}{r}$$

31

## Βέλτιστος Carry-Skip Σταθερού Μήκους Μπλοκ

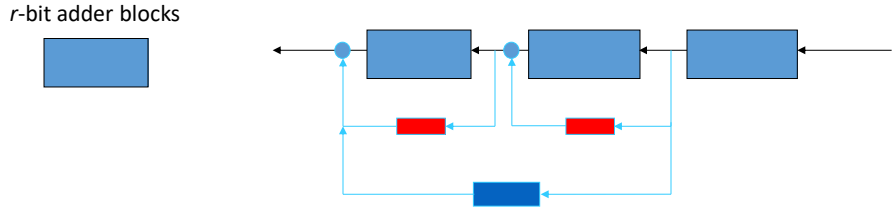
$$T(r) = 4(r-2) + 2\frac{n}{r}$$

$$\frac{dT}{dr} = 4 - 2\frac{n}{r^2} \qquad \frac{dT}{dr} = 0 \Leftrightarrow r_{\text{opt}} = \sqrt{\frac{n}{2}}$$

$$T_{\text{opt}} = 4\left(\sqrt{\frac{n}{2}} - 2\right) + 2\frac{n}{\sqrt{\frac{n}{2}}} = 4\sqrt{2n} - 8$$

32

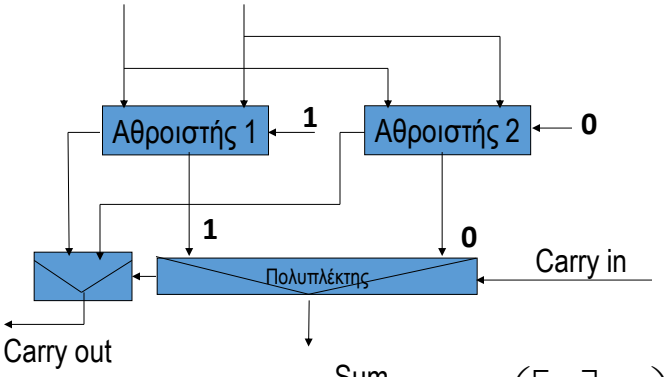
## Αθροιστής Carry-Skip Πολλών Επιπέδων




Καταργεί συνεχόμενες παρακάμψεις.

33

# Ο Αθροιστής Επιλογής Κρατουμένου (carry-select )

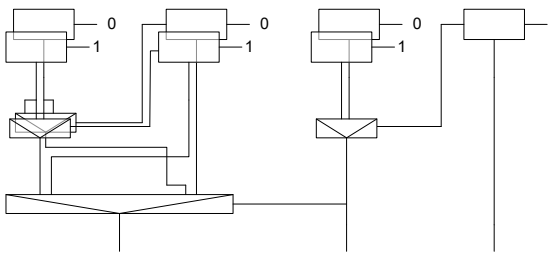



σειριακή εξάρτηση
carry-out από carry-in

$$T_d = 2\left(\left\lceil \frac{n}{r} \right\rceil - 1\right) + T_{add}(r)$$



# Αθροιστής Επιλογής Κρατουμένου Δύο Επιπέδων

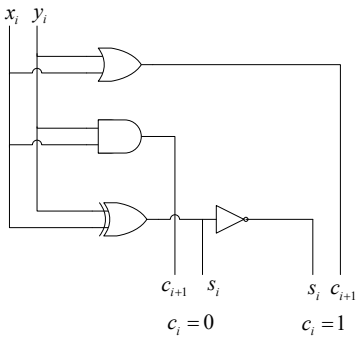



$$T_d = K + 2\left\lceil \log_2\left(\left\lceil \frac{n}{r} \right\rceil - 1\right)\right\rceil$$

Η επιλογή των αποτελεσμάτων γίνεται με δυαδικό δένδρο.
Ο κάθε αθροιστής έχει μήκος
*r = n / 4*

# Ο Αθροιστής Conditional-Sum

- Ο **carry-select** με αθροιστές μήκους *r* =1 λέγεται αθροιστής **conditional sum.**
- Το κύκλωμα άθροισης χρησιμοποιεί κοινό hardware για τις δύο περιπτώσεις.




36

- S= a xor b xor c

- S(c=1)= a xor b xor 1 = (a xor b)’
- S(c=0) =a xor b xor 0 = (a xor b)

- Cout = a b + bc + ac
- Cout(c=1) = a b + b + a = (a+1)b + a = a + b
- Cout(c=0) = ab

37

# Ο Αθροιστής Ling

- Τροποποίηση των βοηθητικών σημάτων του Carry Look Ahead.

$$h_i = c_i + c_{i-1}$$

$$t_i = p_i + g_i \qquad \text{(σήμα Transmit)}$$

38

# Αρχή Λειτουργίας Αθροιστή Ling

$$\begin{aligned} c_{i-1}p_{i-1} &= c_{i-1}p_{i-1} + g_{i-1}p_{i-1} + c_{i-1}p_{i-1}p_{i-1} \\ &= c_{i-1}p_{i-1} + (g_{i-1} + c_{i-1}p_{i-1})p_{i-1} \\ &= c_{i-1}p_{i-1} + c_i p_{i-1} = (c_{i-1} + c_i)p_{i-1} = h_i p_{i-1} \end{aligned}$$

$$p_i = a_i \oplus b_i$$

$$\begin{aligned} c_i &= g_{i-1} + c_{i-1}p_{i-1} \\ &= h_i g_{i-1} + h_i p_{i-1} = h_i t_{i-1} \end{aligned}$$

$$\begin{aligned} h_i &= c_i + c_{i-1} = (g_{i-1} + p_{i-1}c_{i-1}) + c_{i-1} \\ &= g_{i-1} + c_{i-1} = g_{i-1} + h_{i-1}t_{i-2} \end{aligned}$$

$$s_i = p_i \oplus c_i = p_i \oplus h_i t_{i-1} = (t_i \oplus h_{i+1}) + h_i g_i t_{i-1}$$

39

## Το Πλεονέκτημα του Αθροιστή Ling

- Unrolling του διαδιδόμενου σήματος για τέσσερις όρους (Ling):

$$h_i = g_{i-1} + g_{i-2} + g_{i-3}t_{i-2} + g_{i-4}t_{i-3}t_{i-2} + h_{i-4}t_{i-4}t_{i-3}t_{i-2}$$

- Unrolling του διαδιδόμενου σήματος για τέσσερις όρους (CLA):

$$c_i = g_{i-1} + g_{i-2}t_{i-1} + g_{i-3}t_{i-2}t_{i-1} + g_{i-4}t_{i-3}t_{i-2}t_{i-1} + c_{i-4}t_{i-4}t_{i-3}t_{i-2}t_{i-1}$$

- Ένας παράγοντας λιγότερος στους όρους πολλών παραγόντων $\Rightarrow$ μείωση fan-in

40

## Συνδυασμένοι (Hybrid) Αθροιστές

Οι αθροιστές των επεξεργαστών συνδυάζουν τεχνικές.

Συνήθως κάποια διασταύρωση carry look ahead και carry select είναι αποδοτικότερη.

| Τεχνολογία | Μήκος λέξης |
|---|---|

41

# Ασυμπτωτικές Πολυπλοκότητες Αθροιστών

| Είδος Αθροιστή | Σύντμηση | Χρόνος | Εμβαδό |
|---|---|---|---|
| Κυματισμού (ripple) | RCA | $O(n)$ | $O(n)$ |
| Manchester | MCC | $O(n)$ | $O(n)$ |
| Constant width Carry Skip | CSK | $O(\sqrt{n})$ | $O(n)$ |
| Variable width Carry Skip | VSK | $O(\sqrt{n})$ | $O(n)$ |
| Carry Select | CSL | $O(\sqrt{n})$ | $O(n)$ |
| Carry look ahead | CLA | $O(\log n)$ | $O(n \log n)$ |
| ELM | ELM | $O(\log n)$ | $O(n \log n)$ |
| Brent-Kung | B&K | $O(\log n)$ | $O(n \log n)$ |
| Signed Digit (βάση $r$) | SD-$r$ | $O(b)$ | $O(n)$ |
| Carry-save | CSA | $O(1)$ | $O(n)$ |

42

# Πολυπλοκότητες Αθροιστών

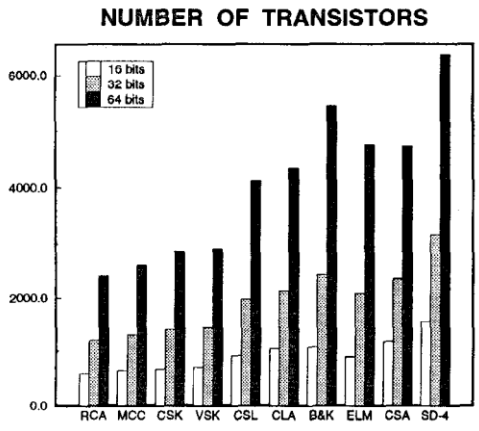
ADDER CIRCUIT DESCRIPTION

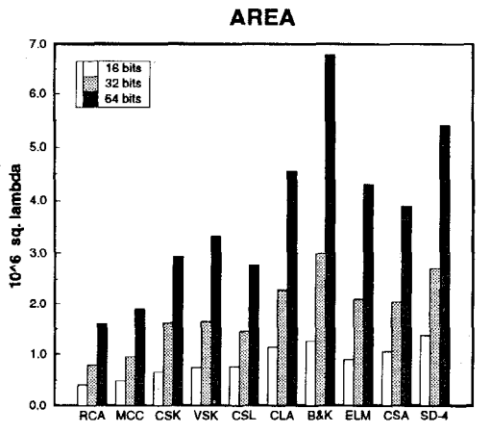
| *Adder Type* | *Area* ($\times 10^6$ $\lambda^2$) | | | *No. of transistors* | | | *Max transistor size* ($n/p$) | | |
|---|---|---|---|---|---|---|---|---|---|
| | *16-bit* | *32-bit* | *64-bit* | *16-bit* | *32-bit* | *64-bit* | *16-bit* | *32-bit* | *64-bit* |
| RCA | 0.40 | 0.80 | 1.60 | 596 | 1204 | 2420 | 3/3 | 3/3 | 3/3 |
| MCC | 0.48 | 0.96 | 1.90 | 642 | 1298 | 2610 | 4/4 | 4/4 | 4/4 |
| CSK | 0.82 | 1.62 | 3.22 | 682 | 1410 | 2866 | 4/4 | 4/4 | 4/4 |
| VSK | 0.81 | 1.76 | 3.44 | 706 | 1440 | 2900 | 3/3 | 3/3 | 3/3 |
| CSL | 0.76 | 1.45 | 2.75 | 914 | 1982 | 4128 | 5/7 | 6/10 | 8/13 |
| CLA | 1.14 | 2.27 | 4.55 | 1038 | 2132 | 4348 | 4/4 | 4/4 | 4/4 |
| B&K | 1.25 | 3.00 | 6.76 | 1072 | 2442 | 5444 | 3/3 | 3/3 | 3/3 |
| ELM | 1.08 | 2.36 | 5.38 | 892 | 2078 | 4752 | 3/5 | 5/7 | 8/12 |
| CSA | 1.05 | 2.03 | 3.90 | 1176 | 2360 | 4728 | 3/3 | 3/3 | 3/3 |
| SD-4 | 1.36 | 2.71 | 5.41 | 1550 | 3166 | 6398 | 3/5 | 3/5 | 3/5 |
| SD-8 | 1.11 | 2.42 | 4.61 | 1228 | 2812 | 5452 | 3/5 | 3/5 | 3/5 |
| SD-16 | 1.09 | 2.17 | 4.32 | 1186 | 2490 | 5098 | 3/5 | 3/5 | 3/5 |
| SD-32 | 0.97 | 2.25 | 4.16 | 1020 | 2572 | 4900 | 3/5 | 3/5 | 3/5 |
| SD-64 | 1.12 | 2.23 | 4.07 | 1180 | 2530 | 4780 | 3/5 | 3/5 | 3/5 |
| SD-128 | 1.27 | 2.11 | 3.78 | 1340 | 2364 | 4412 | 3/5 | 3/5 | 3/5 |

- C. Nagendra, M. J. Irwin and R. M. Owens, "Area-time-power tradeoffs in parallel adders," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 689-702, Oct. 1996, doi: 10.1109/82.539001.

43

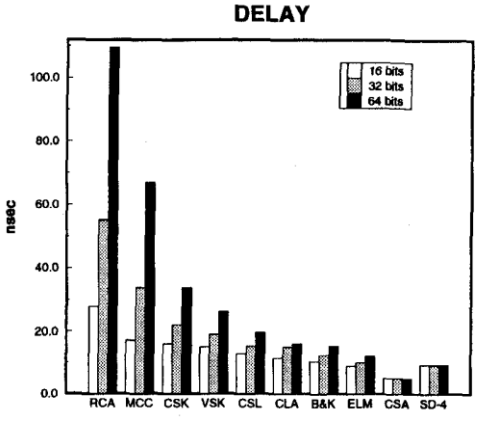
# Απαιτήσεις εμβαδού αθροιστών

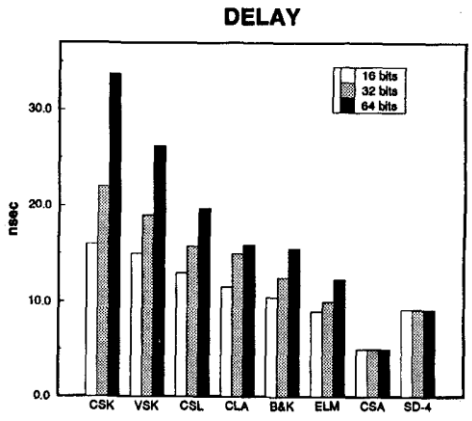



44

# Καθυστερήσεις αθροιστών

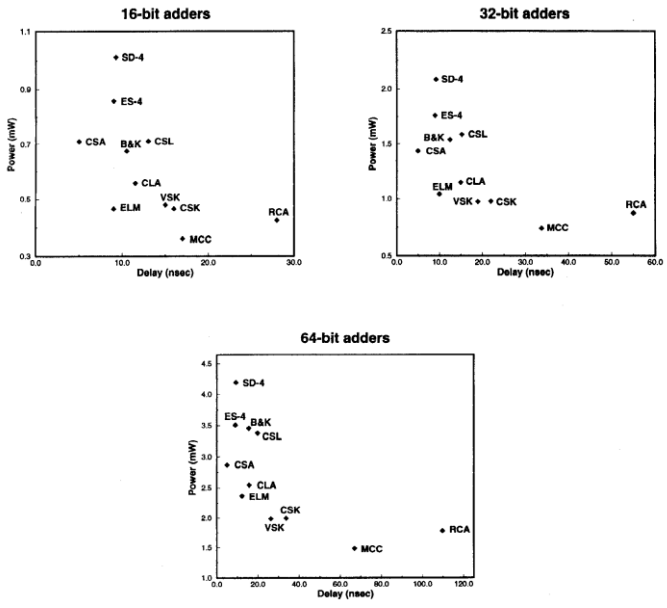



Ταχύτεροι αθροιστές

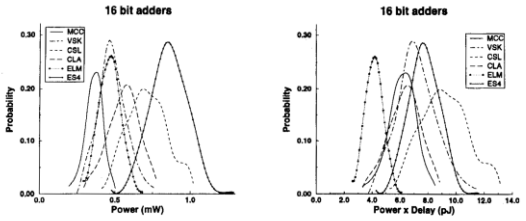
45

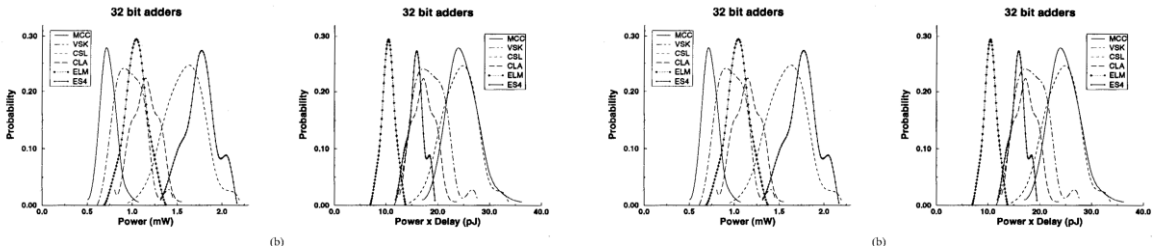
# Power vs delay




46

# Στοιχεία Απαιτήσεων Ισχύος Αθροιστών




47

# Γιατί Υπολογισμός Προθέματος (Prefix)

Το πρόβλημα υπολογισμού του κρατουμένου με τη χρήση βοηθητικών σημάτων μετασχηματίζεται σε ***γνωστό πρόβλημα.***

Αντιμετωπίζει το σειριακό υπολογισμό του κρατουμένου ⇔ CLA

Αντιμετωπίζει το πρόβλημα του fan-out στον carry look-ahead.

48

# Κυκλώματα Παράλληλου Προθέματος (Parallel Prefix)

- Υπολογίζεται η παράσταση

$$a_1 \circ a_2 \circ \cdots \circ a_n$$

όπου $a_i$ γνωστά

προσεταιριστικός αλλά όχι αντιμεταθετικός τελεστής

**Παράδειγμα:**

$$(a_1 \circ a_2) \circ (a_3 \circ a_4) \circ (a_5 \circ a_6) = ((a_1 \circ a_2) \circ a_3) \circ ((a_4 \circ a_5) \circ a_6)$$

49

# Ο Τελεστής Υπολογισμού Κρατουμένου

$$(P,G)\circ(\tilde{P},\tilde{G}) \triangleq (P\cdot\tilde{P}, G+P\cdot\tilde{G})$$

Εφαρμόζεται σε ζεύγη σημάτων propagate, generate

$$P_{i:j} \triangleq \begin{cases} P_i, & i=j \\ P_i \cdot P_{i-1:j}, & i>j \end{cases}$$

$$G_{i:j} \triangleq \begin{cases} G_i, & i=j \\ G_i + P_i \cdot G_{i-1:j}, & i>j \end{cases}$$

Ορισμός σημάτων group propagate, generate

$$(P_{i:j},G_{i:j}) = (P_{i:m},G_{i:m})\circ(P_{m-1:j},G_{m-1:j}),\ i\geq m\geq j+1$$

Μη επικαλυπτόμενα groups

$$(P_{i:j},G_{i:j}) = (P_{i:m},G_{i:m})\circ(P_{v:j},G_{v:j}),\ i\geq m,\quad v\geq j,\quad v\geq m-1$$

Επικαλυπτόμενα groups

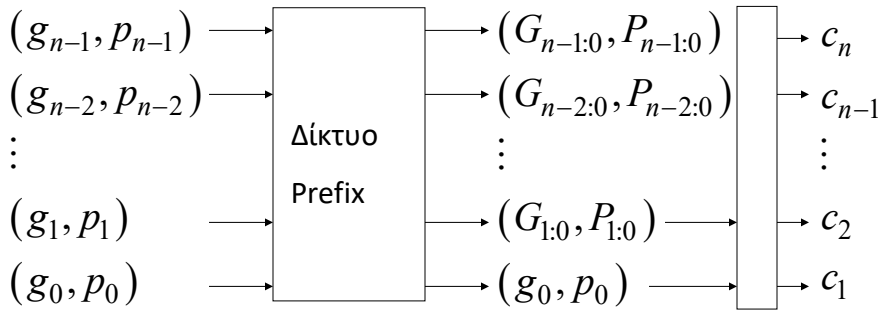
50

# Αρχή Λειτουργίας Αθροιστών Parallel Prefix

1. Πρόβλημα το fanout των σημάτων propagate ανά bit σε CLA
2. Λύση ⇒Χρήση propagate και generate σημάτων από επικαλυπτόμενα groups
3. Αποδοτικός υπολογισμός με parallel prefix δίκτυα τελεστή κρατουμένου

51

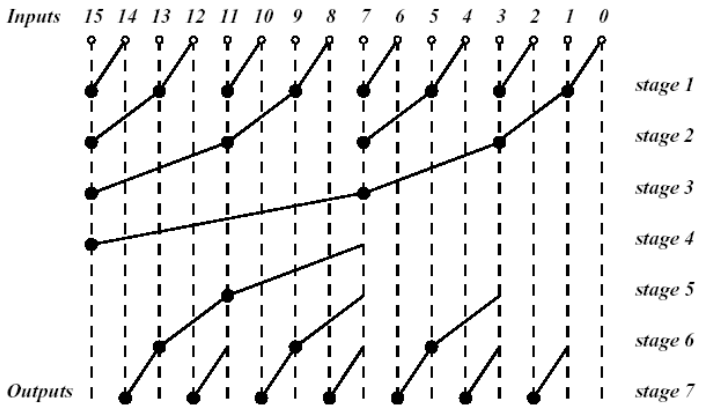
# Οργάνωση Αθροιστών Parallel Prefix




$$c_m = G_{m:0} + P_{m:0}c_0, \quad m > 0$$

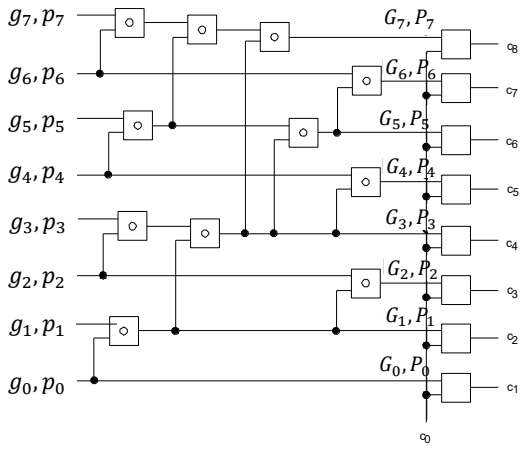
52

# Αθροιστής Παράλληλου Προθέματος Brent-Kung




53

# Κρατούμενα σε Brent-Kung

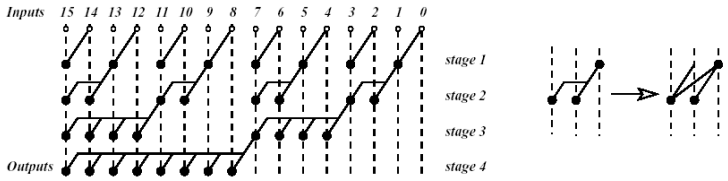



$$N_{tr} = 64n - 16\log_2 n - 32 \quad n = 2^k$$

$$T_{D,ADD} = t_L\left(2\log_2^2 n + 16\log_2 n + 17.8\right)$$

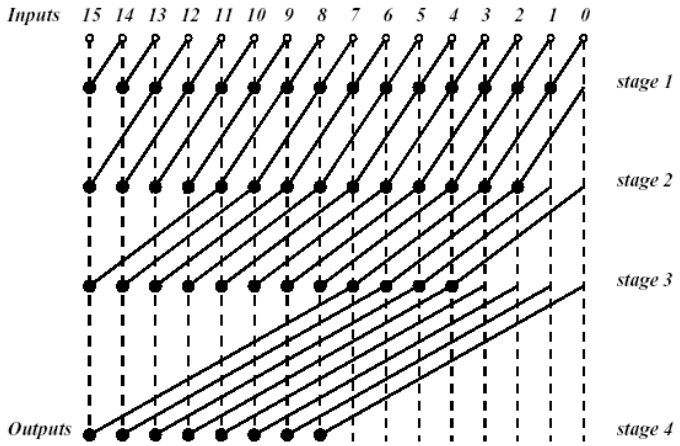
54

# Αθροιστής Ladner-Fischer



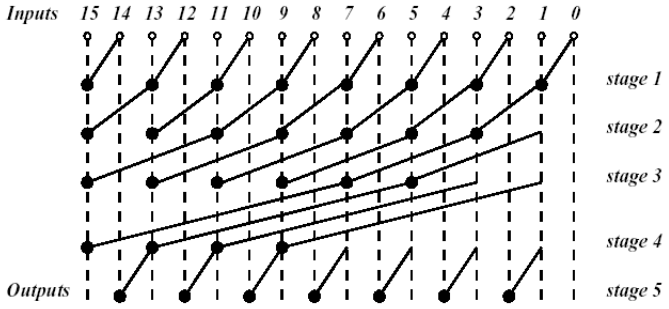

55

# Αθροιστής Kogge-Stone



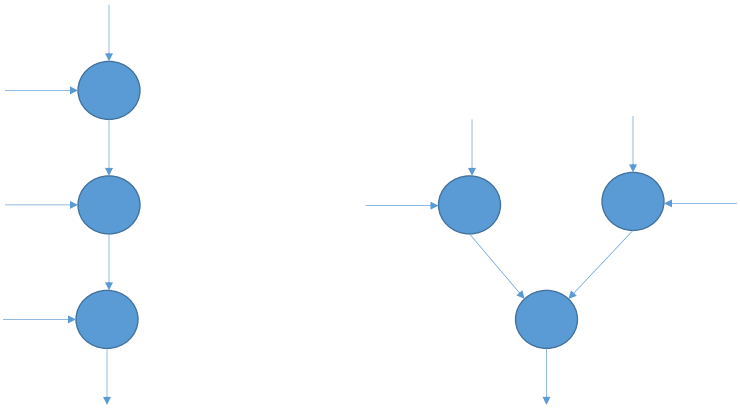

56

# Αθροιστής Han-Carlson



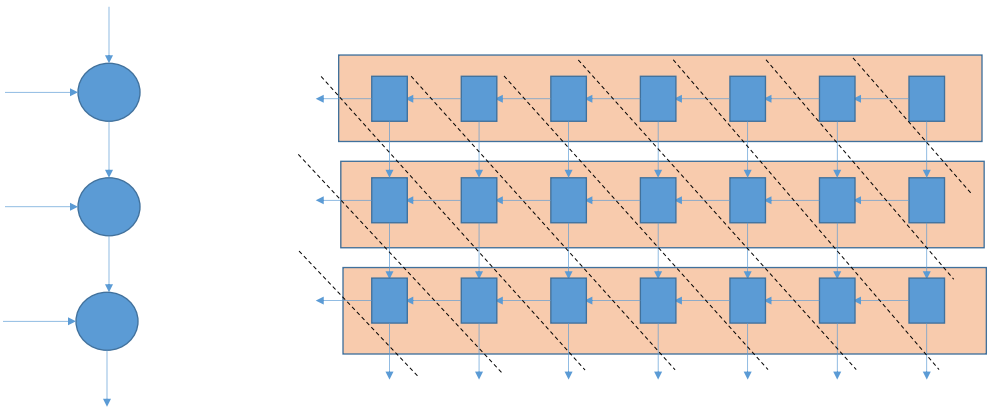

57

# Άθροιση πολλών όρων



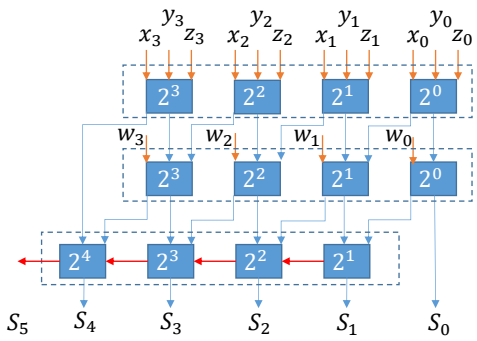
58

# Άθροιση πολλών όρων



59

## Άθροιση Διατήρησης Κρατουμένου (carry save)




Carry Save Adder (CSA): delay 1

Carry Save Adder (CSA): delay 1

Carry propagate adder (CPA): delay depends on $n$
(can be implemented as any two-operand adder)

$k$ is the number of operands

$n$ is the wordlength

$S = X + Y + Z + W$

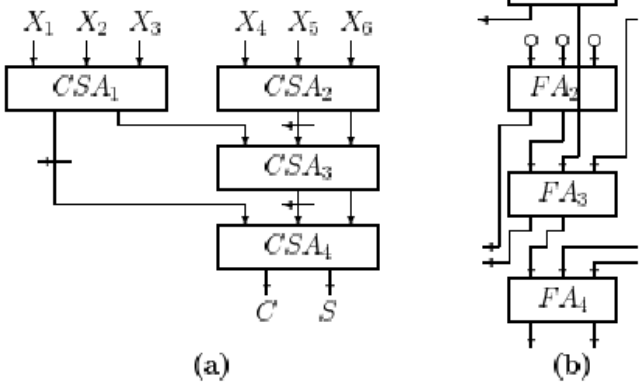
$$T = (k-2)T_{CSA} + T_{CPA}$$

linear with $k$

depends on $n$

60

# Δένδρα Carry Save Αθροιστών

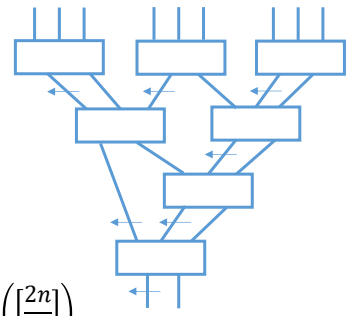



61

# Αριθμός επιπέδων σε δένδρα carry save

| Αριθμός Όρων | Αριθμός Επιπέδων |
|---|---|
| 3 | 1 |
| 4 | 2 |
| $5 \le k \le 6$ | 3 |
| $7 \le k \le 9$ | 4 |
| $10 \le k \le 13$ | 5 |
| $14 \le k \le 19$ | 6 |
| $20 \le k \le 28$ | 7 |
| $29 \le k \le 42$ | 8 |
| $43 \le k \le 63$ | 9 |

$$n \rightarrow \left\lceil \frac{2n}{3} \right\rceil$$



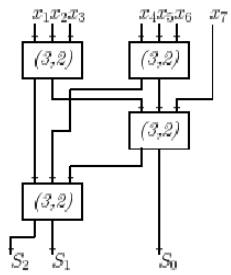
$$\text{levels} \approx \frac{\log \frac{k}{2}}{\log \frac{3}{2}}$$

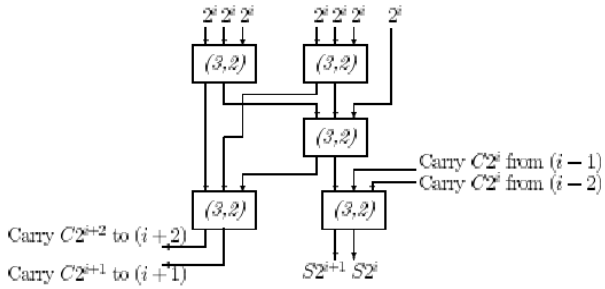
$$h(n) = 1 + h\left(\left\lceil \frac{2n}{3} \right\rceil\right)$$

$$h(3) = 1$$

62

# Counters και Compressors




63

## Γενικευμένοι Counters

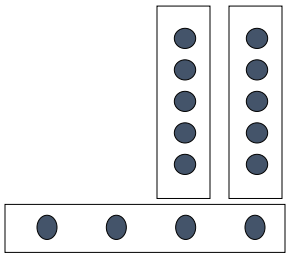
$(k_{l-1}, k_{l-2}, \cdots, k_0, m)$

$$2^m - 1 \geq \sum_{i=0}^{l-1} k_i 2^i$$

$$2^m - 1 \geq k\left(2^l - 1\right)$$

Παράδειγμα:

Ένας (5,5,4) counter



64

## Πρόσθεση 1 bit

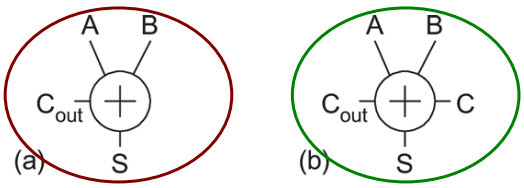



**FIG 10.1** Half and full adders

- Ημιαθροιστής (Half adder) προσθέτει 2 εισόδους (A, B).
  - Απαιτούνται 2 bits για την αναπαράσταση του αποτελέσματος,
    - το άθροισμα S (Sum) και
    - το κρατούμενο Cout (Carry-out)

- Ο πλήρης αθροιστής (Full-adder) έχει τρεις εισόδους.
  - γιατί χρειάζεται αυτό;



65

## Πρόσθεση 1 bit – Πίνακες Αληθείας

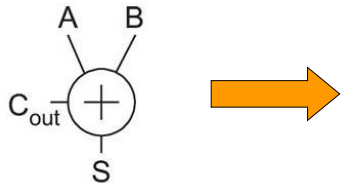



**Table 10.1** Truth table for half adder

| A | B | $C_{out}$ | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

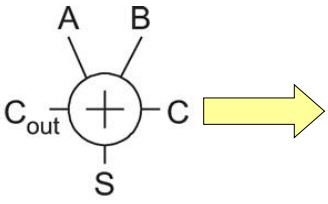



**Table 10.2** Truth table for full adder

| A | B | C | G | P | K | $C_{out}$ | S |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | 1 | | | | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| | | 1 | | | | 1 | 1 |



66

## Συνάρτηση Generate

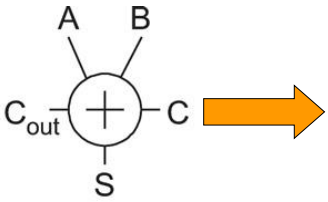



**Table 10.2** Truth table for full adder

| A | B | C | G | P | K | $C_{out}$ | S |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | 1 | | | | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| | | 1 | | | | 1 | 1 |

Στον full adder είναι χρήσιμος ο ορισμός των Generate (G) , Propagate (P) και των Kill (K) σημάτων.

Ο αθροιστής δημιουργεί (generates) κρατούμενο όταν το Cout είναι αληθές ανεξάρτητα από το Cin, οπότε: G=A • B



67

## Συναρτήσεις Kill & Propagate

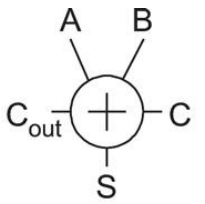



**Table 10.2** Truth table for full adder

| A | B | C | G | P | K | $C_{out}$ | S |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | 1 | | | | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| | | 1 | | | | 1 | 1 |

Ο αθροιστής «σκοτώνει» (kills) το carry όταν A=B=0 ανεξάρτητα από το Cin, οπότε: $K = \overline{A} \bullet \overline{B} = \overline{A + B}$

Ο αθροιστής μεταδίδει (propagates) το carry-in όταν μία είσοδος είναι 1, δηλαδή: $P = A \oplus B$

68

68

## Συναρτήσεις Ημιαθροιστή 1 bit

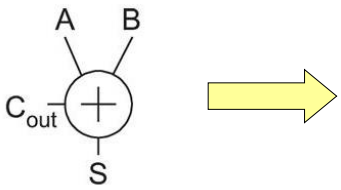



**Table 10.1** Truth table for half adder

| A | B | $C_{out}$ | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Από τον πίνακα αληθείας, η λογική του half adder είναι:

$$S = A \oplus B$$
$$C_{out} = A \bullet B$$

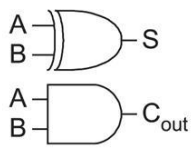



**FIG 10.2** Half adder design

69

69

## Συναρτήσεις Πλήρη Αθροιστή 1 bit

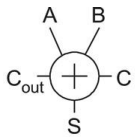
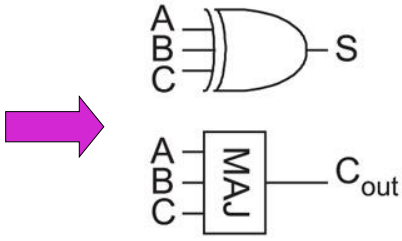



**Table 10.2** Truth table for full adder

| A | B | C | G | P | K | $C_{out}$ | S |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | 1 | | | | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| | | 1 | | | | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| | | 1 | | | | 1 | 1 |

Ομοίως από τον πίνακα αληθείας, η λογική του full adder είναι:

$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \overline{AB}C + ABC =$$
$$= (A \oplus B) \oplus C = P \oplus C$$
$$C_{out} = AB + AC + BC =$$
$$= AB + C(A + B) =$$
$$= \overline{\bar{A}\bar{B} + \bar{C}(\bar{A} + \bar{B})} = MAJ(A,B,C)$$




Δηλαδή το carry παίρνει την τιμή της πλειοψηφίας των A, B, C π.χ. αν A=1, B=0, C=1 πλειοψηφούν οι '1'.

70

70

## Κυκλωματικός Σχεδιασμός Αθροιστή 1 bit

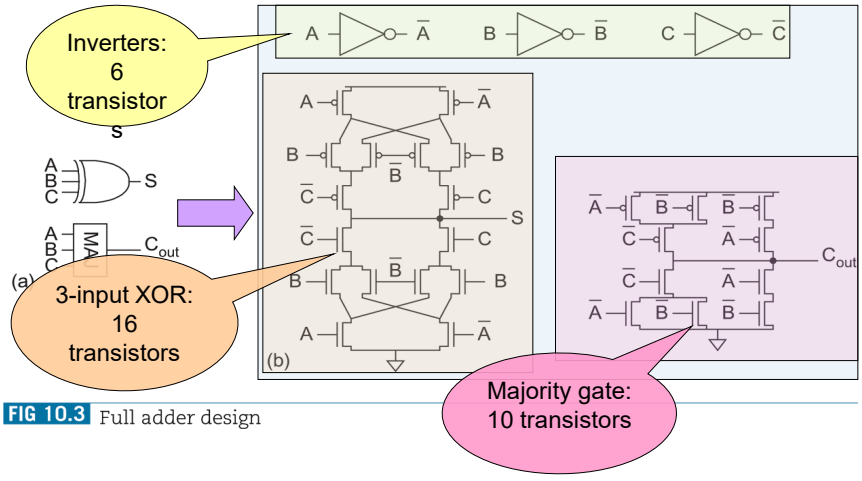


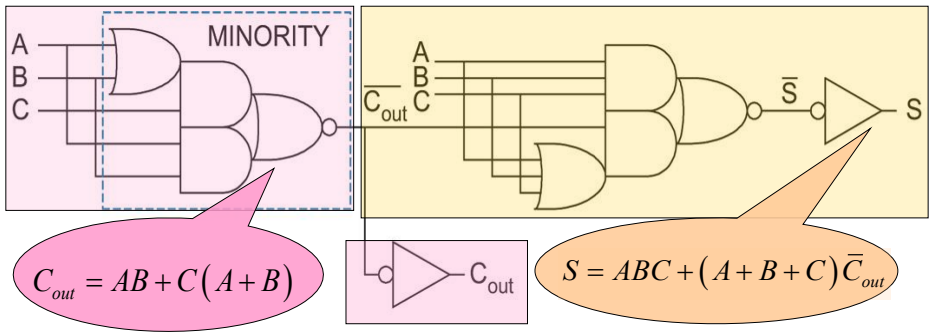

FIG 10.3 Full adder design

Πλήρης αθροιστής με συνολικά **32 transistors**

71

71

## Εναλλακτικός Κυκλωματικός Σχεδιασμός Αθροιστή 1 bit (1/)




Μία πιο συμπαγής μορφή του adder βασίζεται στο γεγονός ότι το S μπορεί να παραγοντοποιηθεί έτσι ώστε να συμπεριλάβει και το Cout:

$$S = ABC + (A + B + C)\overline{C}_{out}$$

72

72

## Εναλλακτικός Κυκλωματικός Σχεδιασμός Αθροιστή 1 bit (1/)

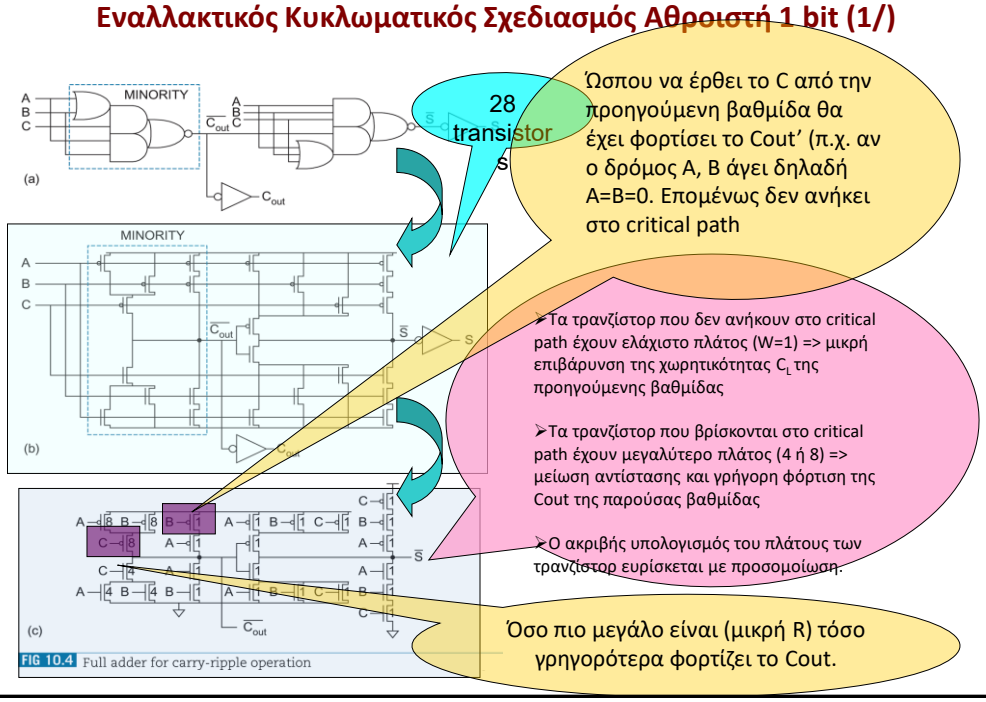


FIG 10.4 Full adder for carry-ripple operation

73

## Εναλλακτικός Κυκλωματικός Σχεδιασμός Αθροιστή 1 bit (1/)

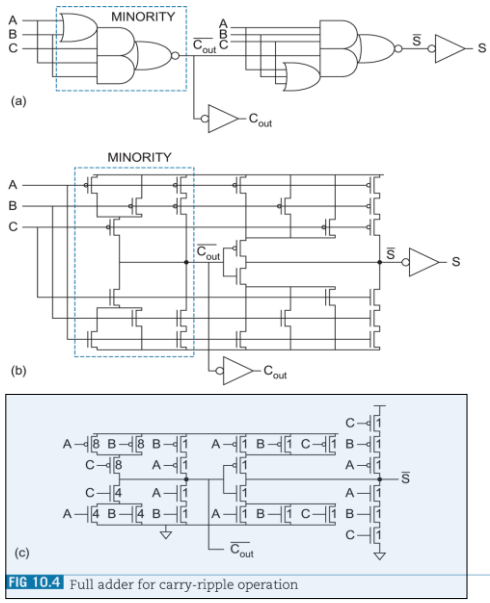



FIG 10.4 Full adder for carry-ripple operation

- Τοποθέτηση του πιο αργού σήματος carry-in στις εσωτερικές εισόδους
  - οι εσωτερικές χωρητικότητες να έχουν εκφορτιστεί/φορτιστεί όταν έρχεται το πιο αργό σήμα ( carry)

- Εξάλειψη των αντιστροφέων στην έξοδο C. Απαιτείται αντιστροφή των A και B στις ζυγές βαθμίδες. Συνολικά έχουμε λιγότερο υλικό δηλ, υλοποίηση με 24 αντί 28 transistors

74

## Εναλλακτικός Κυκλωματικός Σχεδιασμός Αθροιστή 1 bit (1/)

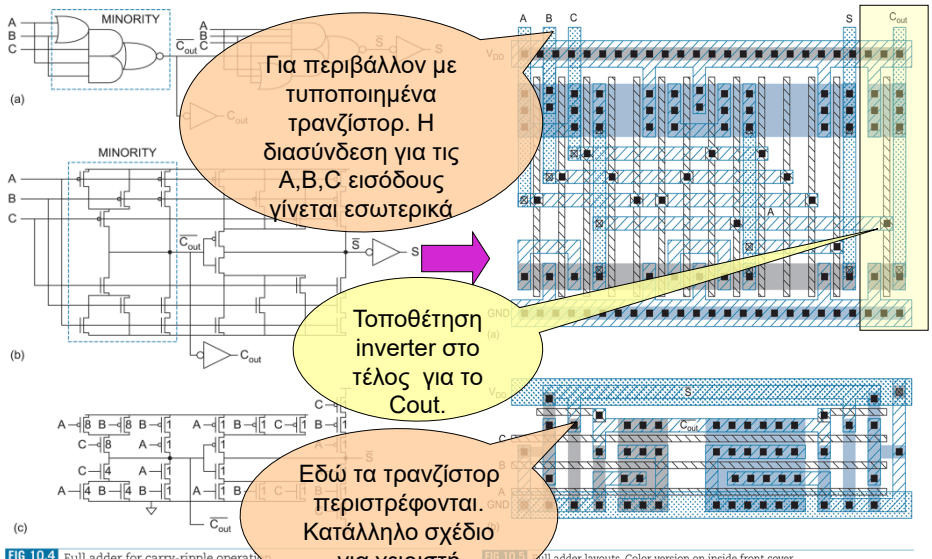



FIG 10.4 Full adder for carry-ripple operation

FIG 10.5 Full adder layouts. Color version on inside front cover.

75

## Κυκλωματικός Σχεδιασμός πύλες μετάδοσης και XOR (1/2)

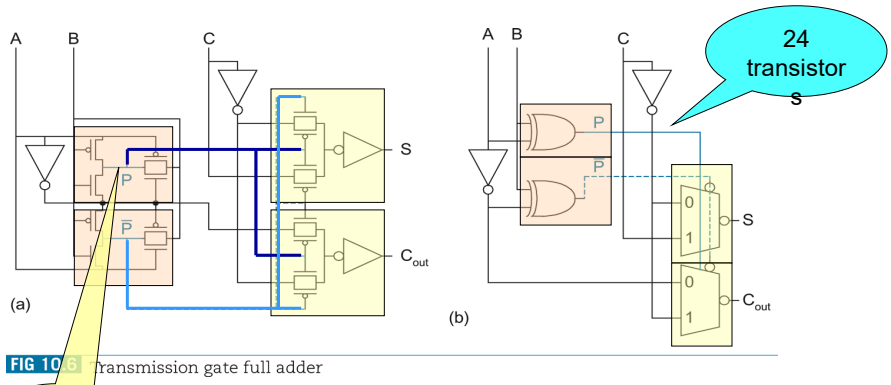



FIG 10.6 Transmission gate full adder

Δημιουργία XOR: Όταν A=1 ο inverter δεν λειτουργεί

Χρήση των $P, \overline{P}$ για έλεγχο των 4 πυλών μεταφοράς

Λαμβάνουμε τα $P, \overline{P}$ αυτόχρονα. Με inverter αντί για XOR για παραγωγή του $\overline{P}$ θα υπήρχε καθυστέρηση

76

76

## Κυκλωματικός Σχεδιασμός πύλες μετάδοσης και XOR (2/2)

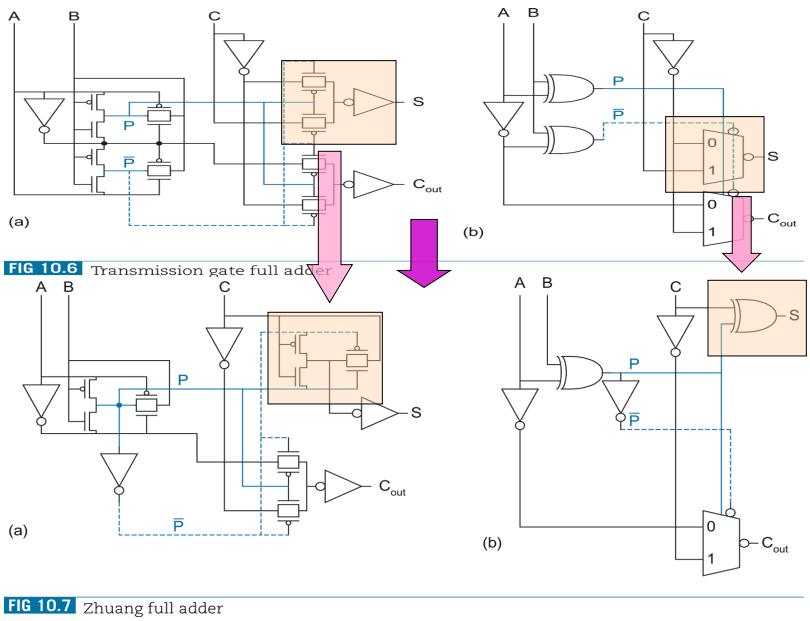



FIG 10.7 Zhuang full adder

77

## Κυκλωματικός Σχεδιασμός με CPL λογική

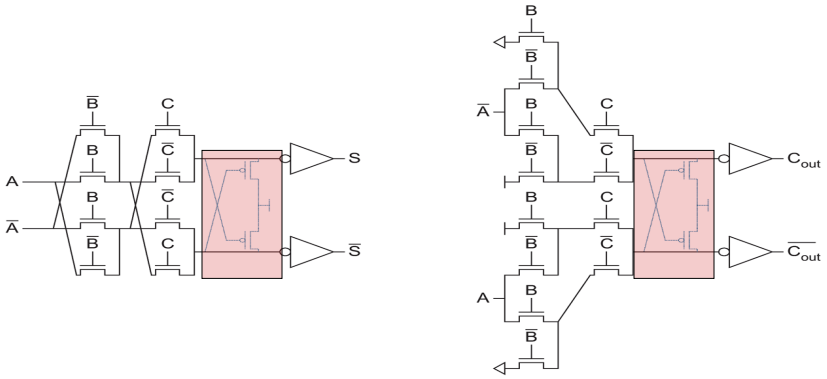



FIG 10.8 CPL full adder

- Τα αμυδρά p-τύπου τρανζίστορ επαναφέρουν την τάση στο $V_{dd}$
- Συγκρινόμενη με προσεκτική υλοποίηση του Σχ. 10.4c είναι ελαφρώς γρηγορότερη, συγκρίσιμη κατανάλωση και με μεγαλύτερη επιφάνεια

78

78

## Κυκλωματικός Σχεδιασμός με dual rail domino λογική

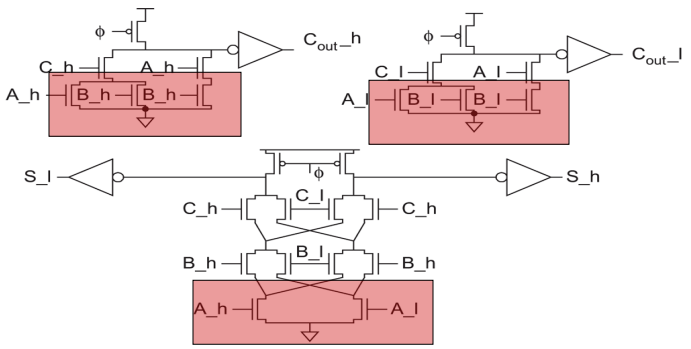



FIG 10.9 Dual-rail domino full adder

Οι δυναμικοί full adders χρησιμοποιούνται στους γρήγορους multipliers

- Χρήση footless λογικής – λείπει το n transistor για το ϕ
- Όταν στην προφόρτιση δεν δημιουργείται μονοπάτι προς την γη μπορεί να παραλειφθεί το transistor
  - Σε πολλές περιπτώσεις αυτό επιτυγχάνεται με πρόσθετο κύκλωμα.

79

79

## Carry – Propagate Addition

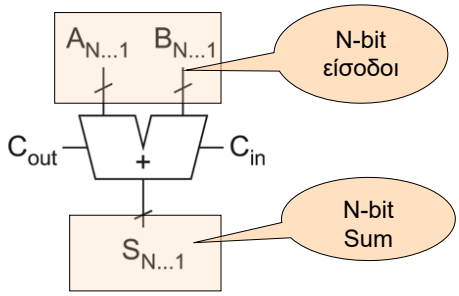



**FIG 10.10** Carry-propagate adder

80

80

## Μετάδοση Κρατουμένου

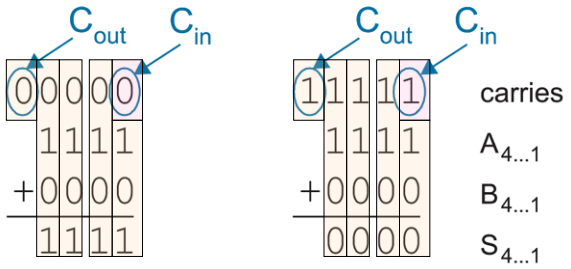



**FIG 10.11** Example of carry propagation

Δύο παραδείγματα όπου τα carry και sum bits επηρεάζονται από το $C_{in}$

Η απλούστερη σχεδίαση είναι ο carry–ripple adder (RCA) όπου το carry–out ενός bit είναι συνδεδεμένο στο carry–in του επόμενου bit

81

81

## Ripple Carry Adder

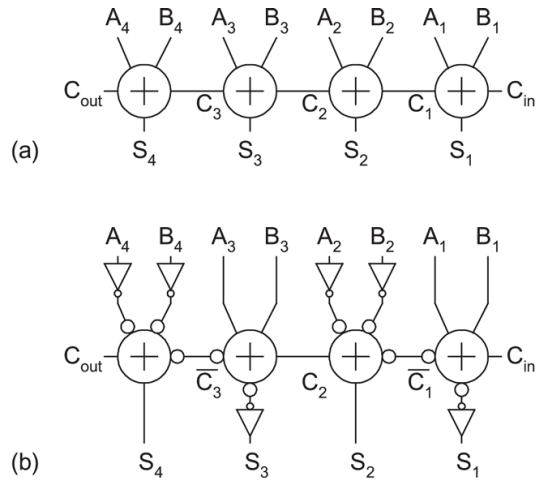



FIG 10.12 4-bit carry-ripple adder

- Η καθυστέρηση είναι ο χρόνος για τη μετάδοση του carry μέσα από τα N στάδια
- Η πρόσθεση είναι αυτοδυϊκή πράξη (η συνάρτηση των συμπληρωματικών εισόδων είναι το συμπλήρωμα της συνάρτησης)
- **Ένας αθροιστής που δέχεται συμπληρ. εισόδους παράγει μη συμπληρ. εξόδους (C, S)**
- **Η παράληψη των αντιστροφέων μικραίνει το critical path**
- Η χρήση NOT στις εισόδους και sum δεν είναι στο critical path

82

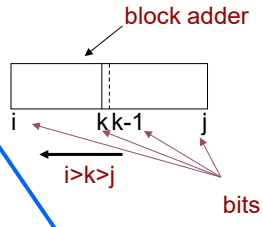
82

## Carry Generation & Propagation

$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$
$$P_{i:j} = P_{i:k}\, P_{k:j}$$

To block (i:j) παράγει carry όταν το block (i:k) παράγει carry ή όταν το block [(k-1):j] παράγει carry και το block (i:k) το μεταδίδει

$$G_{i:i} = G_i = A_i B_i$$
$$P_{i:i} = P_i = A_i \oplus B_i$$




To group μεταδίδει carry αν και το πάνω και το κάτω μέρος μεταδίδουν carry

Για το bit 0 έχουμε:

$$C_0 = C_{in}$$
$$C_n = C_{out}$$

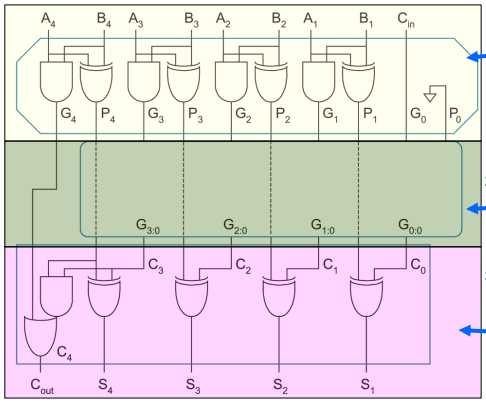
$$G_{0:0} = C_{in}$$
$$P_{0:0} = 0$$

Το sum είναι: $S_i = P_i \oplus G_{i-1:0}$

83

83

## Πρόσθεση με Λογική Γέννησης & Διάδοσης Κρατουμένου (1/2)




$$G_{i:i} \equiv G_i = A_i \cdot B_i$$
$$P_{i:i} \equiv P_i = A_i \oplus B_i$$

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

$$S_i = P_i \oplus G_{i-1:0}$$

FIG 10.13 Addition with generate and propagate logic

➢Η πρόσθεση μπορεί να απλοποιηθεί στη διαδικασία των παραπάνω τριών βημάτων

84

84

## Πρόσθεση με Λογική Γέννησης & Διάδοσης Κρατουμένου (2/2)

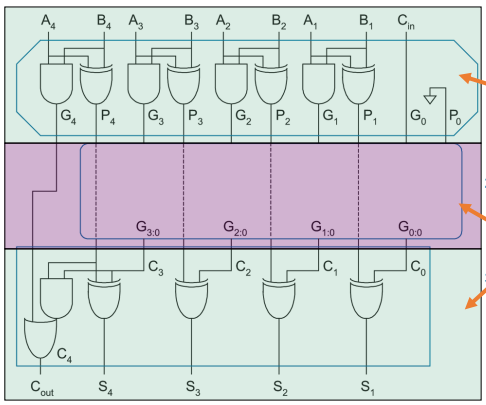



FIG 10.13 Addition with generate and propagate logic

Η λογική υπολογισμού του 1ου και 3ου επιπέδου δεν αλλάζει στις διάφορες μεθόδους υπολογισμού του $C_i = G_{i:0}$ , i=1…N

Υπάρχουν εναλλακτικές μέθοδοι υλοποίησης για το 2ο επίπεδο με πλεονεκτήματα σε ταχύτητα, επιφάνεια και πολυπλοκότητα

85

85

## Διαμερισμός λογικής PG

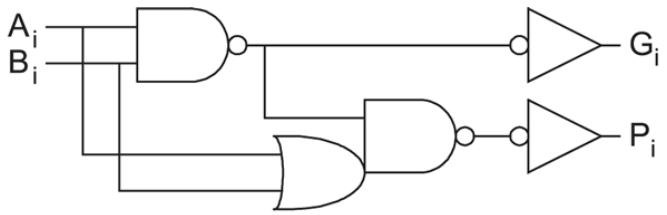



FIG 10.14 Shared bitwise PG logic

➢ Ένα μέρος από hardware του Group PG logic μπορεί να μοιραστεί στην bitwise PG logic για απλοποίηση του Group PG logic

86

86

## Carry-Ripple Addition (1/7)

➢ Το critical path του carry–ripple adder περνάει από carry–in σε carry–out κατά μήκος της αλυσίδας όλων των βαθμίδων

➢ Επειδή τα σήματα P, G θα έχουν ήδη υπολογιστεί τη στιγμή που θα έρθει το carry μπορούν να χρησιμοποιηθούν για την απλούστευση της συνάρτησης του Cout σε μία AND-OR πύλη

$$\begin{aligned} C_i &= A_i B_i + (A_i + B_i) C_{i-1} \\ &= A_i B_i + (A_i \oplus B_i) C_{i-1} \\ &= G_i + P_i C_{i-1} \end{aligned}$$

$C_i = G_{i:0}$

$$G_{i:0} = G_i + P_i G_{i-1:0}$$

➢ Η πρόσθεση με κύμα μπορεί να παρουσιασθεί σαν ακραία μιας λογικής PG ομάδας όπου

– Μια ομάδα 1-bit (εδώ το $G_i$) group συνδυάζεται με ένα (i)-bit ($G_{(i-1)0}=C_{i-1}$) group για να σχηματισθεί ένα (i+1)-bit group

87

87

## Carry-Ripple Addition (2/7)

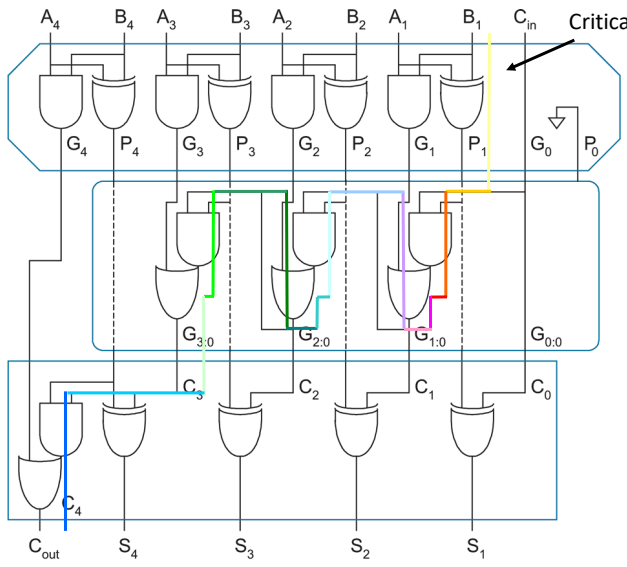



FIG 10.15 4-bit carry-ripple adder using PG logic

Το critical path περνάει μέσα από μια αλυσίδα από AND-OR πύλες και όχι από αλυσίδα από majority πύλες που έχουν μεγαλύτερη καθυστέρηση και επιφάνεια

88

## Carry-Ripple Addition (3/7)

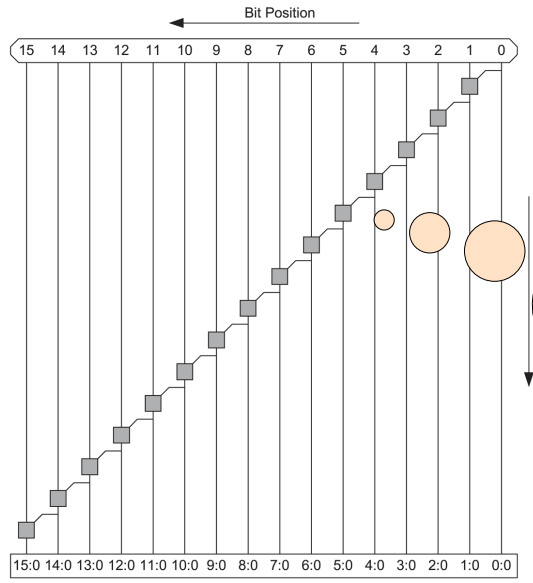


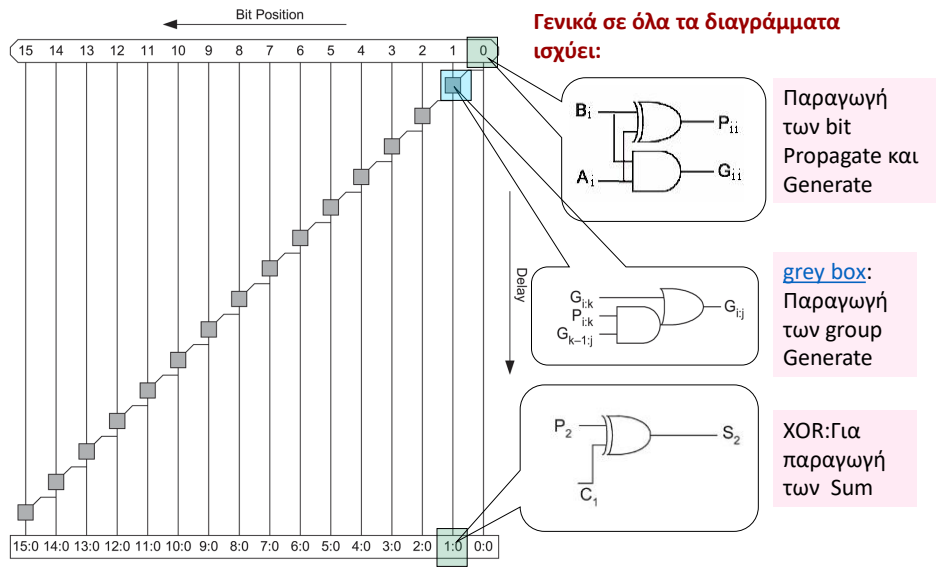

FIG 10.16 Carry-ripple adder group PG network

PG Logic για 16-bit adder

Χρήση διαγραμμάτων αυτού του είδους για σύγκριση διαφορετικών υλοποιήσεων αθροιστών. Δείχνει την καθυστέρηση της κάθε λογικής

89

## Carry-Ripple Addition (4/7)




**Γενικά σε όλα τα διαγράμματα ισχύει:**

Παραγωγή των bit Propagate και Generate

grey box: Παραγωγή των group Generate

XOR:Για παραγωγή των Sum

FIG 10.16 Carry-ripple adder group PG network





## Carry-Ripple Addition (5/7)

Σε τέτοιου είδους διαγράμματα θα ισχύει αυτή η αντιστοίχηση

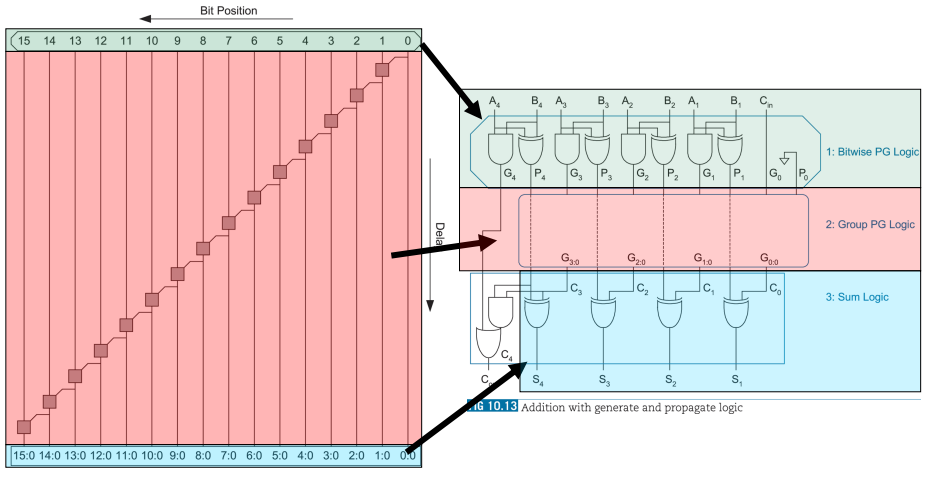



FIG 10.13 Addition with generate and propagate logic

FIG 10.16 Carry-ripple adder group PG network

## Carry-Ripple Addition (6/7)

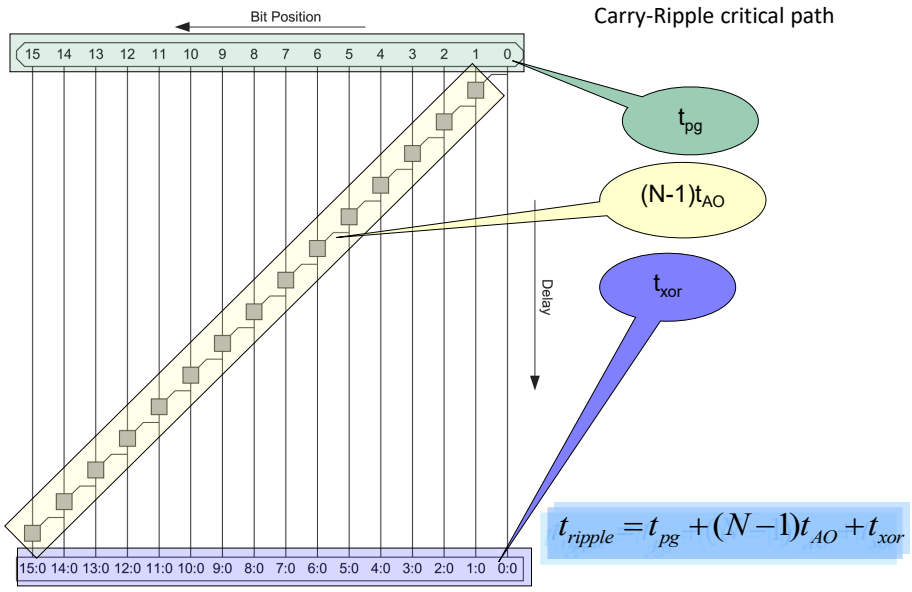



FIG 10.16 Carry-ripple adder group PG network

92

## Carry-Ripple Addition (7/7)

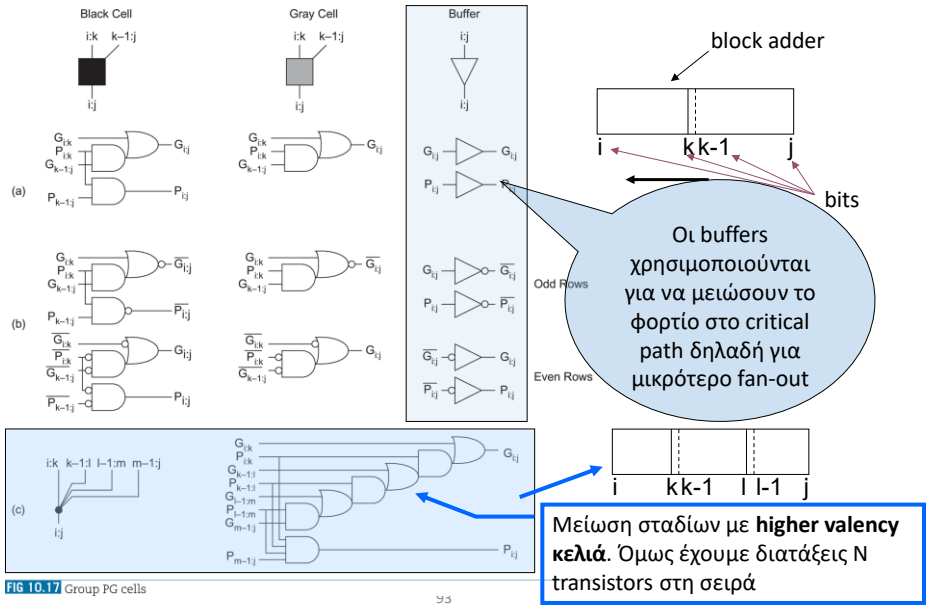



FIG 10.17 Group PG cells

93

## Manchester Carry Chain Adder (1/5)

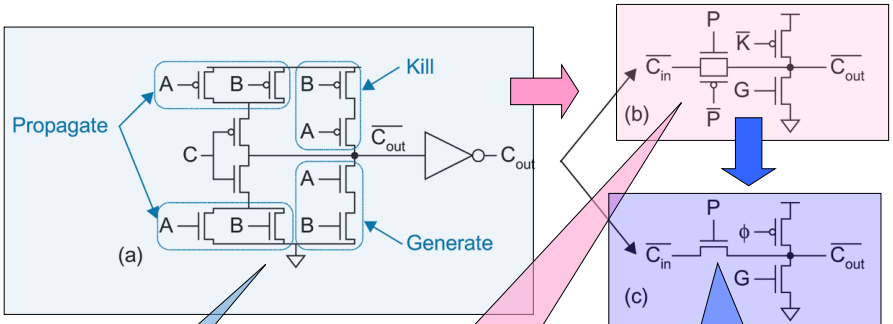



FIG 10.18 Carry chain designs

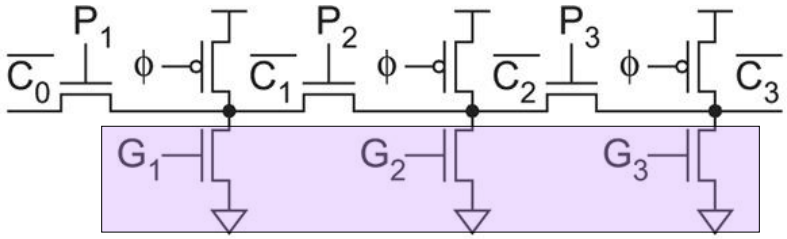
Majority gate

**Στατική Βαθμίδα:** Το $\overline{C}_{in}$ μεταδίδεται μέσω της πύλης μεταφοράς. G (Generate Carry) δημιουργεί το Nmos transistor όταν άγει και Kill (K') το Pmos

**Δυναμική Βαθμίδα:** Είναι γρηγορότερη και απαιτεί λιγότερη επιφάνεια

94

## Manchester Carry Chain Adder (2/5)




➢ Πολλά επίπεδα της δυναμικής βαθμίδας συνδέονται και δημιουργούν την Manchester carry chain για έναν multi-bit Manchester adder

Η διάταξη αυτή επειδή έχει N transistors στη σειρά εισάγει **μεγάλη καθυστέρηση** γι αυτό το N είναι συνήθως μικρό (π.χ. N=4)

95

95

## Manchester Carry Chain Adder (3/5)

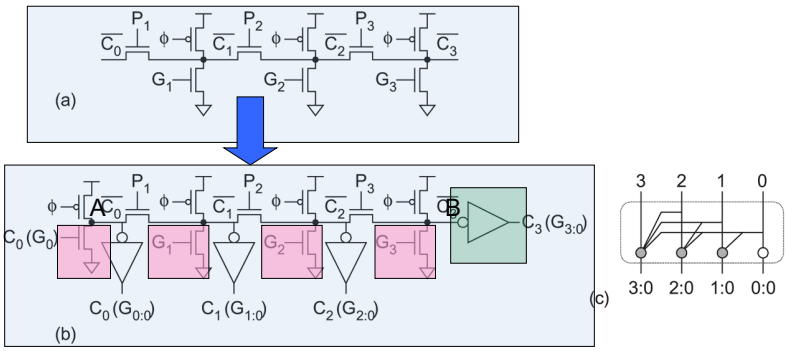



FIG 10.19 Manchester carry chains

- Επίλυση του προηγούμενου προβλήματος με εισαγωγή inverter

- **Προφόρτιση:** Άγουν όλα τα φ οπότε η χωρητικότητα του κόμβου (AB) φορτίζεται από 4 τρανζίστορς στα οποία εφαρμόζεται το φ

- **Εκφόρτιση:** Η κατανεμημένη χωρητικότητα εκφορτίζεται μέσω 4 τρανζίστορ (3 του κόμβου AB και το n τρανζίστορ του αντιστροφέα)





## Manchester Carry Chain Adder (4/5)

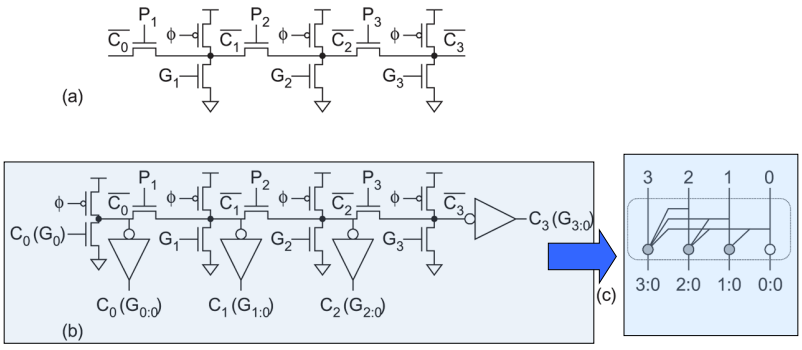



FIG 10.19 Manchester carry chains

$$C_0 = G_{0:0} = C_0$$
$$C_1 = G_{1:0} = G_1 + P_1 C_0$$
$$C_2 = G_{2:0} = G_2 + P_2 \left( G_1 + P_1 C_0 \right)$$
$$C_3 = G_{3:0} = G_3 + P_3 \left( G_2 + P_2 \left( G_1 + P_1 C_0 \right) \right)$$

Ο Manchester carry chain μπορεί να αναπαρασταθεί σαν ένας buffer με 4 gray cells (συνήθως βέλτιστη επιλογή στο μέγεθος της αλυσίδας)

## Manchester Carry Chain Adder (5/5)

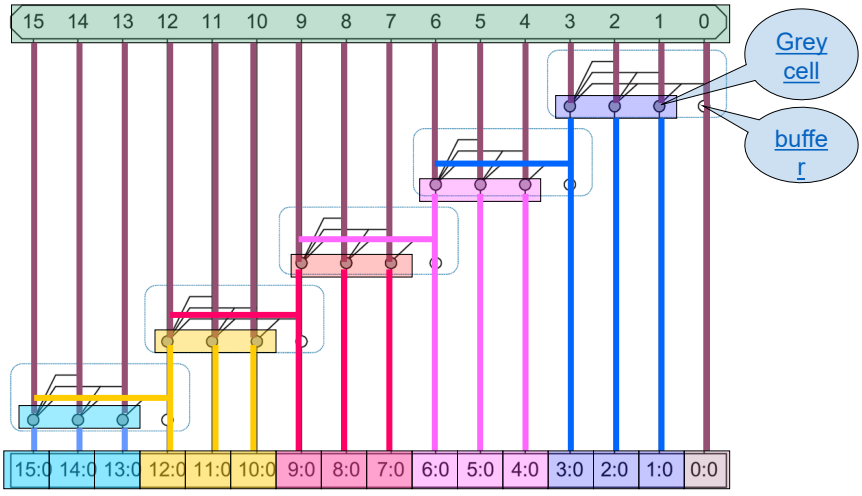



**FIG 10.21** Manchester carry chain adder group PG network

Καλύτερος από τον carry-ripple (N/3 slots) – όμως **αργός για μεγάλους adders**





## Carry-Skip Adder (1/12)

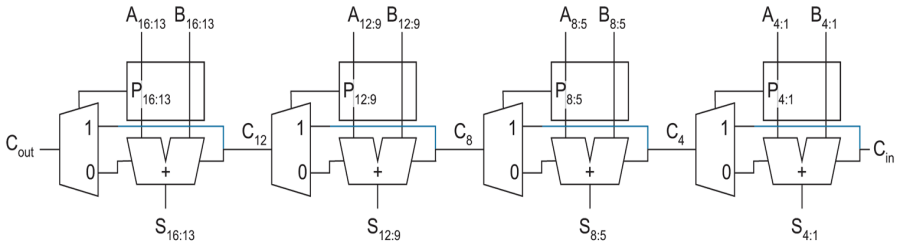



**FIG 10.22** Carry-skip adder

➢ Υπολογίζει το group propagate (P4:1, P8:5, P12:9, P16:13) για κάθε αλυσίδα από carry (τεσσάρων bit εδώ) και το χρησιμοποιεί για να παρακάμψει μεγάλους carry ripple adders => μείωση critical path

## Carry-Skip Adder (2/12)

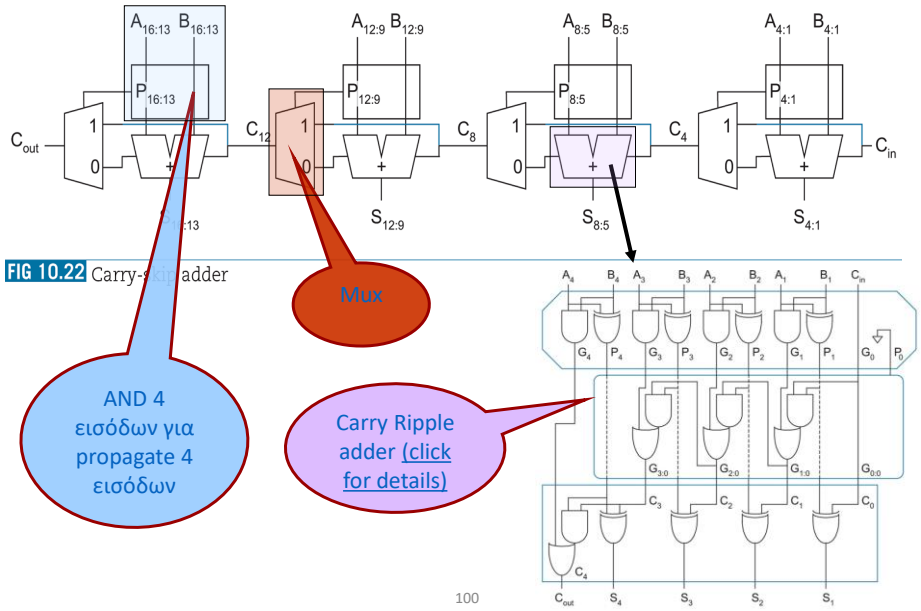



FIG 10.22 Carry-skip adder

100

100

## Carry-Skip Adder (3/12)

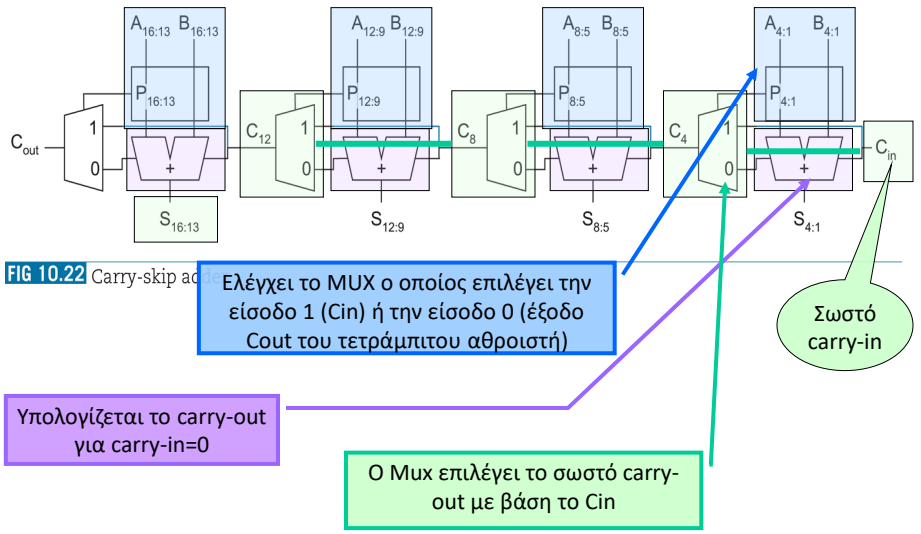



FIG 10.22 Carry-skip ad

101

101

# Carry-Skip Adder (4/12)

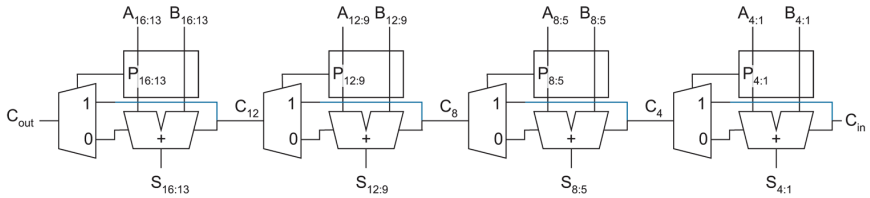



FIG 10.22 Carry-skip adder

- Αρχικά τα 4 group propagate blocks ταυτόχρονα υπολογίζουν το P4:1, P8:5, P12:9, P16:13
- Παράλληλα ο carry ripple που αντιστοιχεί στα ψηφία 4:1 και του είναι γνωστό το Cin υπολογίζει τα S4:1 και το C4
- Όταν P4:1=1 δηλαδή ο block αθροιστής (4:1) κάνει propagate τότε ο MUX επιλέγει το Cin, δηλαδή το Cin διαδίδεται στην έξοδο του block.
- Όταν το P4:1=0 τότε ο MUX επιλέγει την έξοδο του ripple carry

102

102

# Carry-Skip Adder (5/12)

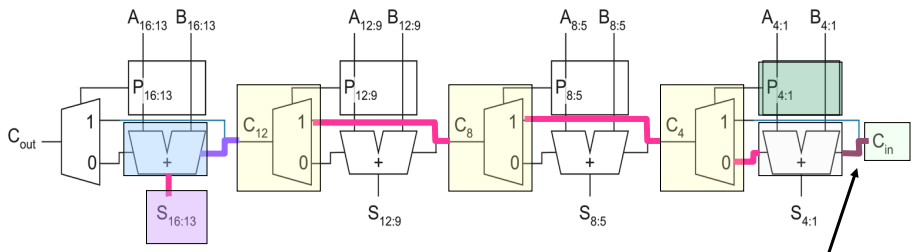



FIG 10.22 Carry-skip adder

➢ Το critical path (συνολική καθυστέρηση) μπορεί να οριστεί με βάση την χειρότερη περίπτωση για την εύρεση του carry-out (το πιο αργό σήμα)

➢ Με βάση τα προηγούμενα η χειρότερη περίπτωση είναι να χρειαστεί να περάσει το carry-out από όλους τους multiplexers δηλαδή να έχουμε συνεχώς Propagate εκτός του πρώτου (4:1)

103

103

## Carry-Skip Adder (6/12)

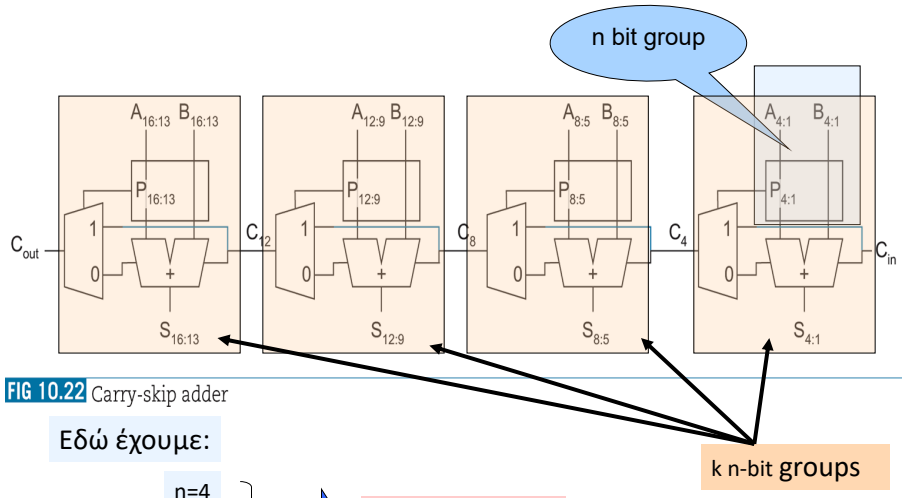



FIG 10.22 Carry-skip adder

Εδώ έχουμε:

n=4

k=4

N=n·k=16 bits

k n-bit groups

104

104

## Carry-Skip Adder (7/12)

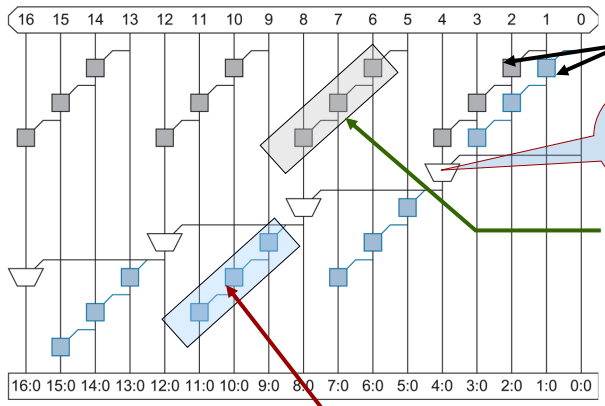



FIG 10.23 Carry-skip adder PG network

Grey cell

Mux. Delay ίσο με μία AND22-OR

Καθορίζουν αν κάθε ομάδα γεννά carry
Με γκρί αναπαρίσταται ο υπολογισμός του sum και carry-out αγνοώντας το carry-in στο bit 5

Με γαλάζιο αναπαρίσταται ο υπολογισμός του σωστού carry-out
Αθροιστές παράγουν carry-out όταν το carry-in στέλνεται μέσω παράκαμψης

105

105

## Carry-Skip Adder (8/12)

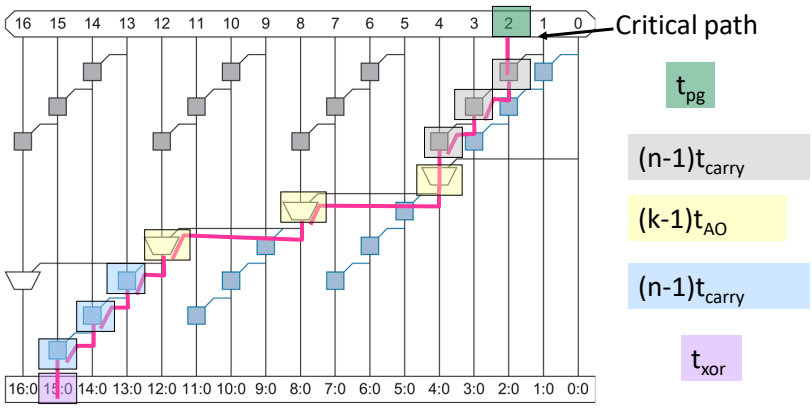



FIG 10.23 Carry-skip adder PG network

Συνολικά η καθυστέρηση είναι:

$$t_{skip} = t_{pg} + (n-1)t_{carry} + (n-1)t_{carry} + (k-1)t_{AO} + t_{xor} \Rightarrow$$

$$t_{skip} = t_{pg} + 2(n-1)t_{carry} + (k-1)t_{AO} + t_{xor}$$

106

106

## Carry-Skip Adder (10/12)

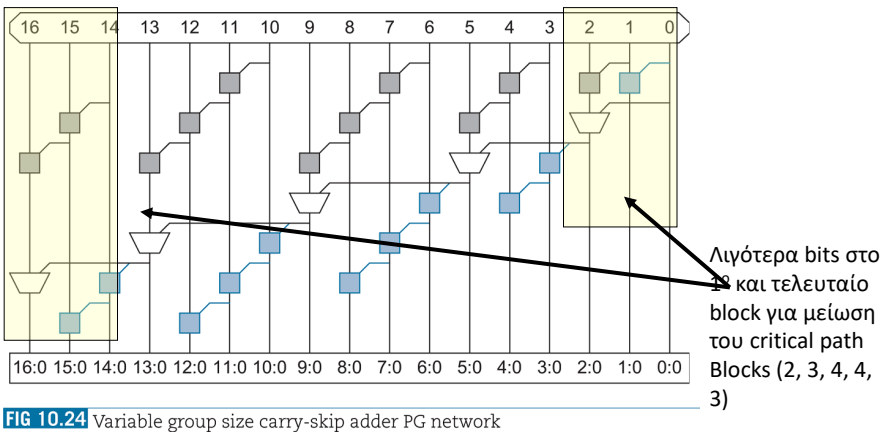



FIG 10.24 Variable group size carry-skip adder PG network

➢ Το critical path εξαρτάται από το μέγεθος του 1ου και τελευταίου block. Μία βελτίωσή του είναι η προσθήκη λιγότερων bits στο 1ο και τελευταίο block και περισσότερα bits στα ενδιάμεσα blocks

108

108

## Carry-Skip Adder (11/12)

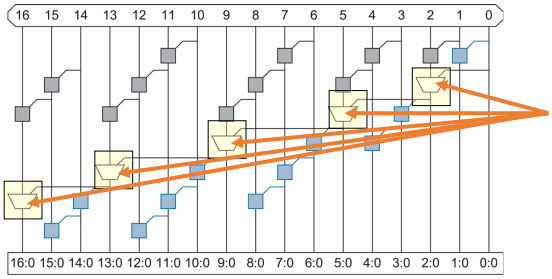



FIG 10.24 Variable group size carry-skip adder PG network

Ο Mux παρουσιάζει περίπου την ίδια καθυστέρηση με μία AND-OR πύλη => μπορούσαν να αντικατασταθούν γιατί αντιστοιχούν εδώ στην ίδια λογική

- Αύξηση του critical path και προβλήματα στον adder
  - Στην πρόσθεση 111…111+000…000+Cin και Cin=0 δεν παρουσιάζεται πρόβλημα
  - Αν όμως Cin=1 κάθε block παράγει carry. Αν το Cin=0 θα πρέπει να μεταδοθεί μέσω των N βαθμίδων

- Το πρόβλημα αυτό δεν υπάρχει στους domino skip adders, CLA και carry select adders

109

109

## Carry-Skip Adder (12/12)

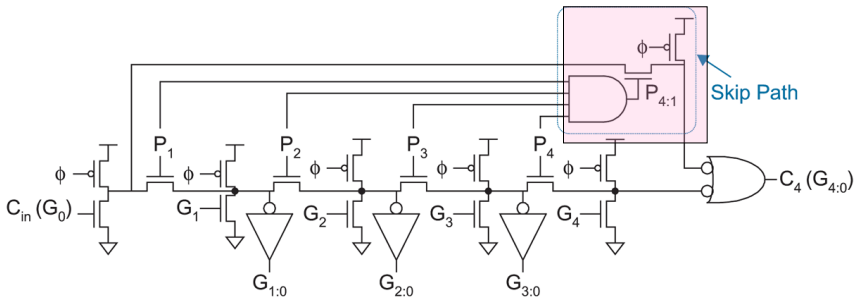



FIG 10.25 Carry-skip adder Manchester stage

- Το πρόβλημα αυτό λύνεται με την χρήση domino λογικής επειδή όλα τα κρατούμενα μηδενίζονται κατά τη διάρκεια της προφόρτισης

- Ο carry-skip adder με τη βαθμίδα Manchester επιλύει το πρόβλημα που δημιουργούν οι AND-OR πύλες

110

110

## Carry-Lookahead Adder (1/7)

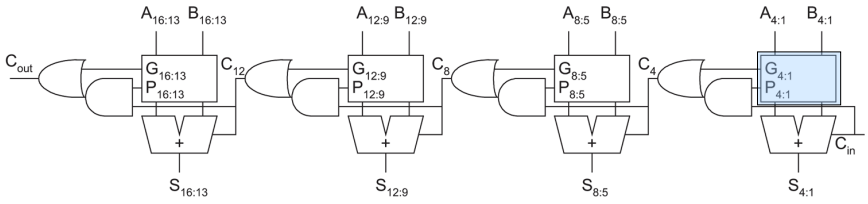



FIG 10.26 Carry-lookahead adder

➢ Ο Carry-lookahead Adder είναι παρόμοιος με τον carry-skip

➢ Υπολογίζονται στα ορθογώνια **group generate και τα group propagate σήματα** (γρηγορότερο από το ripple carry) ώστε να μην χρειαστεί να περάσει από τον ripple-carry για να προσδιορίσει αν δημιουργεί carry





## Carry-Lookahead Adder (2/7)

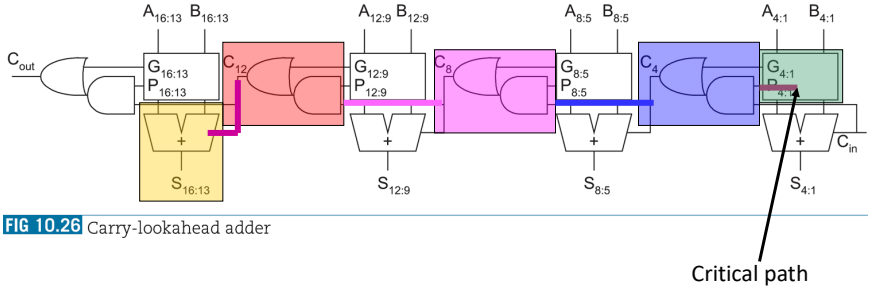



FIG 10.26 Carry-lookahead adder

➢ Ο Carry-lookahead Adder παρουσιάζει καλύτερο critical path από τον carry-skip επειδή δε χρειάζεται να περάσει από όλους τους ripple carry

## Carry-Lookahead Adder (3/7)

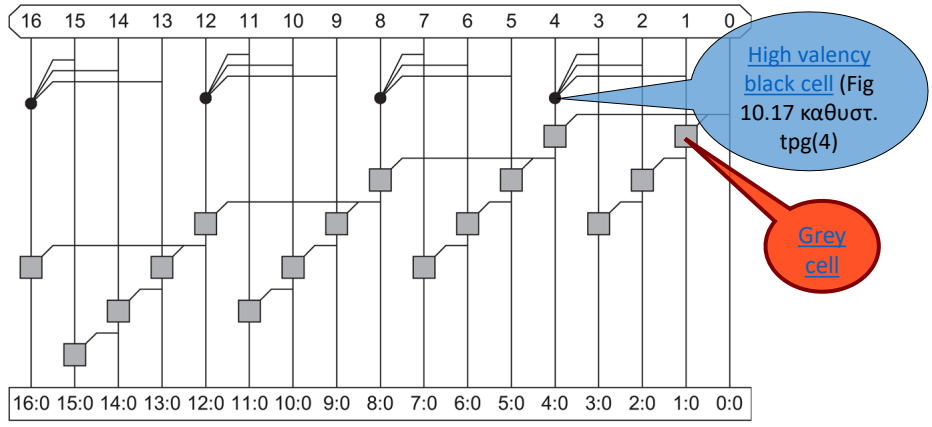



**FIG 10.27** Carry-lookahead adder group PG network



113

## Carry-Lookahead Adder (4/7)

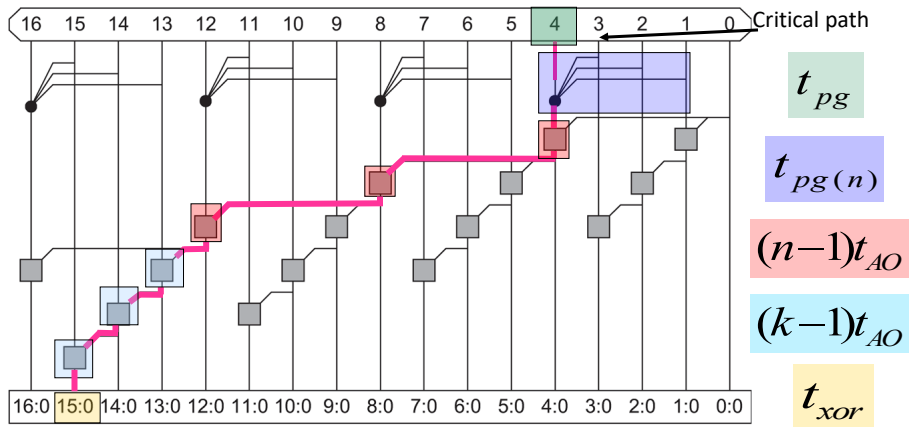



**FIG 10.27** Carry-lookahead adder group PG network

Critical path του Carry-Lookahead Adder

$$t_{cla} = t_{pg} + t_{pg(n)} + \left[(n-1)+(k-1)\right]t_{AO} + t_{xor}$$



114

## Carry-Lookahead Adder (5/7)

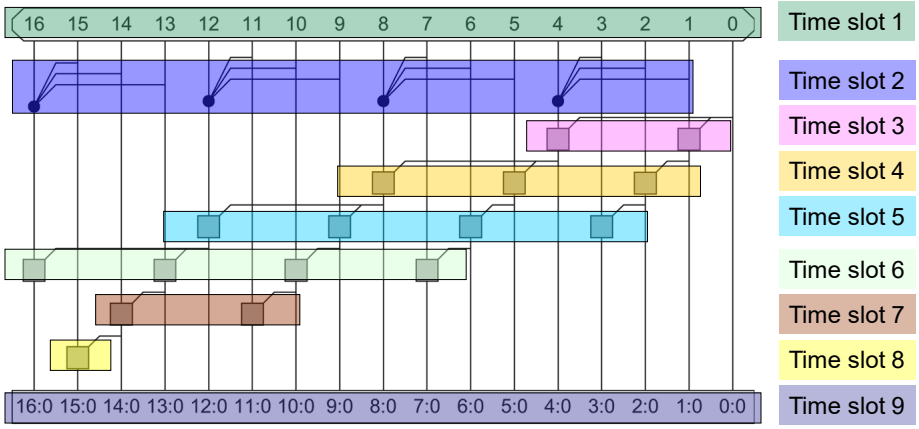



**FIG 10.27** Carry-lookahead adder group PG network

➢ Οι χρονικές στιγμές δε διαρκούν όλες το ίδιο



115

## Carry-Lookahead Adder (6/7)

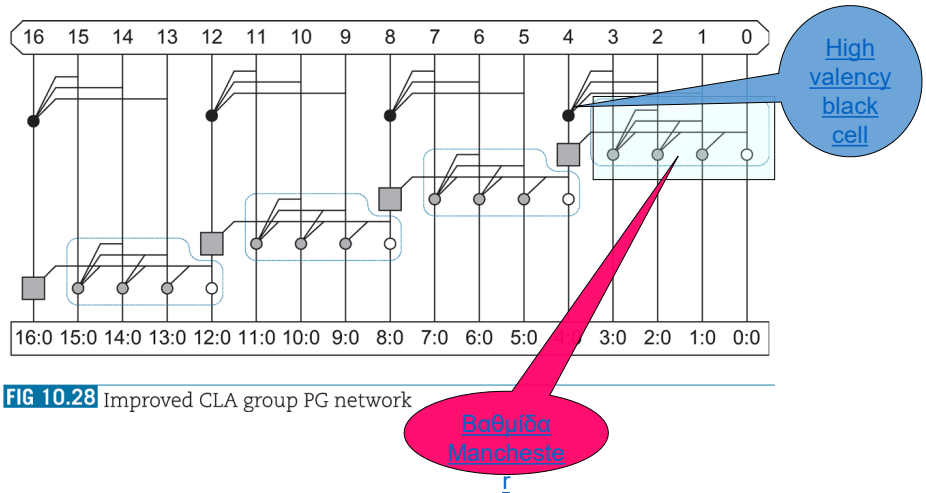


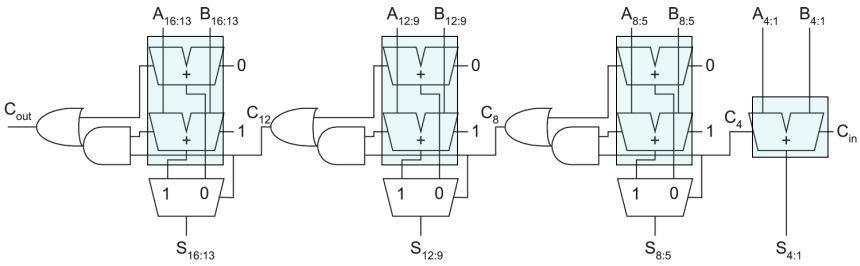

**FIG 10.28** Improved CLA group PG network

➢ Για βελτίωση του critical path γίνεται συνδυασμός των high valency κελιών και της βαθμίδας Manchester



116

## Carry-Select Adder (1/12)

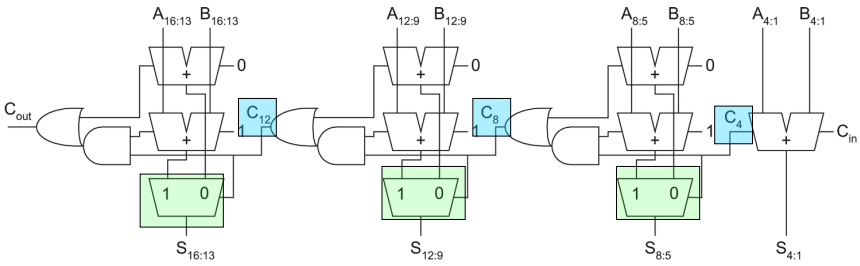



- Γίνεται προϋπολογισμός των εξόδων για τις πιθανές εισόδους



118

## Carry-Select Adder (2/12)

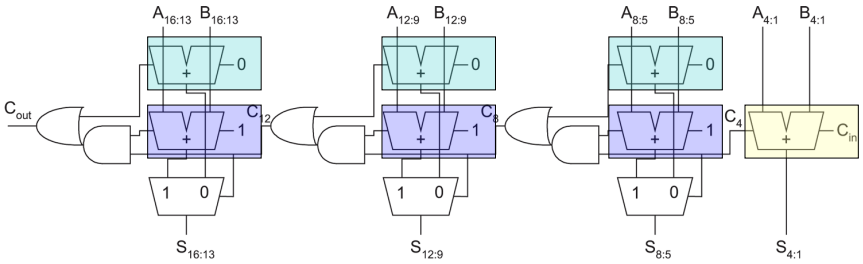



ύστερα με ένα MUX γίνεται η επιλογή της σωστής εξόδου S ανάλογα με το Carry in



119

## Carry-Select Adder (3/12)

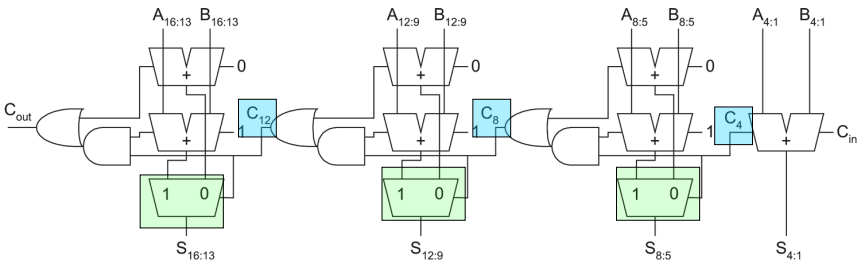



- Ένας αθροιστής υπολογίζει το άθροισμα σε περίπτωση που το carry-in είναι 0
- Ένας δεύτερος αθροιστής υπολογίζει το άθροισμα σε περίπτωση που το carry-in είναι 1
- Εκτός από το πρώτο group που έχουμε έναν αθροιστή επειδή το Carry-in είναι ήδη γνωστό, συνήθως 0



120

## Carry-Select Adder (4/12)

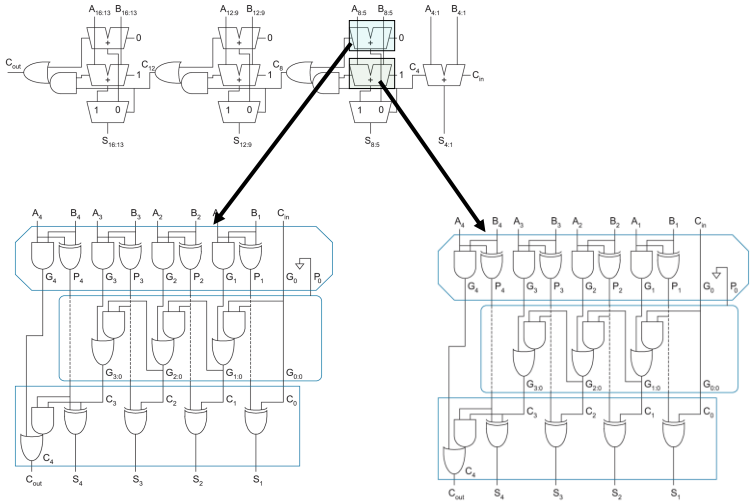



- Με βάση το πραγματικό carry που έρχεται από την προηγούμενη βαθμίδα ο Multiplexer επιλέγει το κατάλληλο άθροισμα που έχει προϋπολογιστεί



121

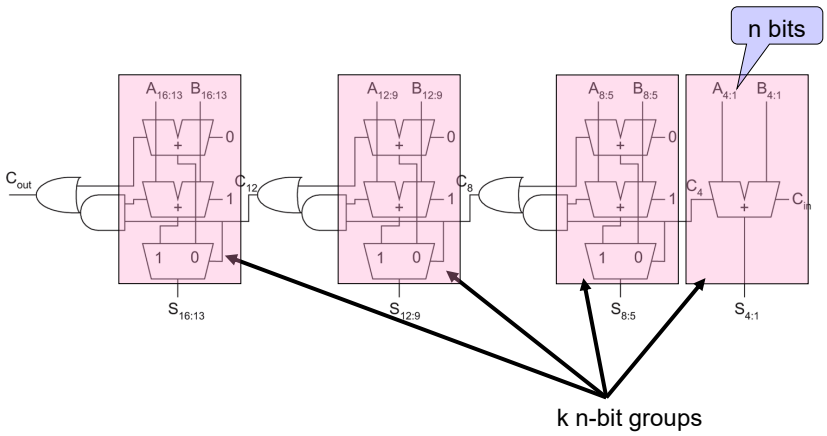
## Carry-Select Adder (5/12)





122

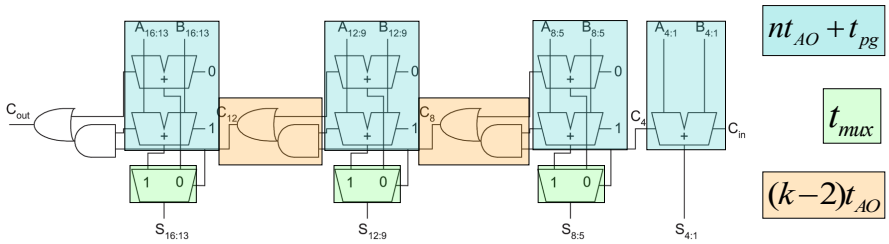
## Carry-Select Adder (6/12)






123

## Carry-Select Adder (7/12)




Critical path delay of the carry-select adder

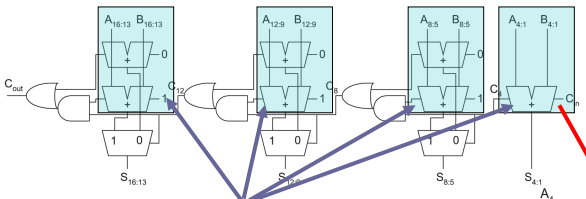
$$t_{select} = t_{pg} + [n + (k-2)]t_{AO} + t_{mux}$$



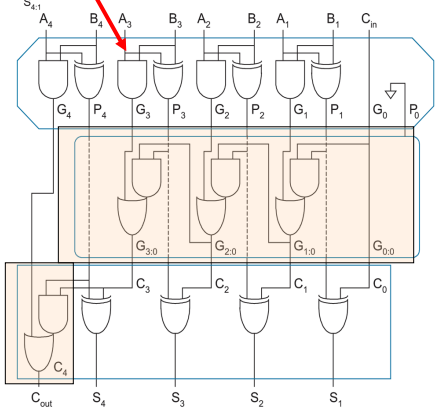
124

## Carry-Select Adder (8/12)

Critical path του carry-select



$$t_{select} = t_{pg} + nt_{AO} + (k-2)t_{AO} + t_{mux}$$

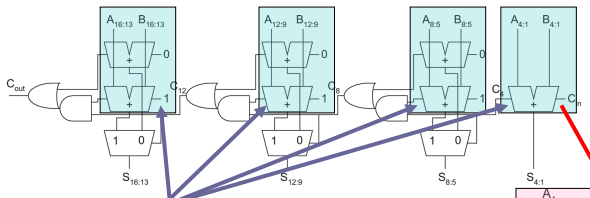



- Ο χρόνος για τον υπολογισμό μόνο των Carries για κάθε block
- Επειδή το Carry in για κάθε block είναι γνωστό (0 ή 1) ο υπολογισμός των Carries για όλα τα blocks γίνεται ταυτόχρονα => η συνολική καθυστέρηση είναι ίση με την καθυστέρηση ενός block



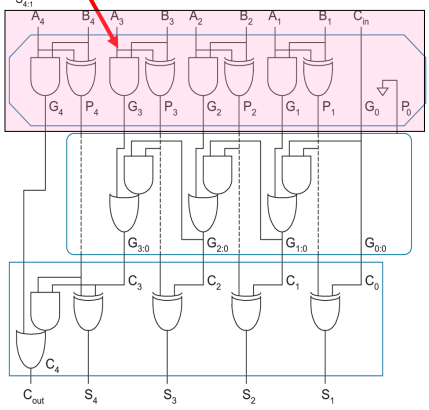
125

# Carry-Select Adder (9/12)



Critical path delay of the carry-select adder

$$t_{select} = t_{pg} + nt_{AO} + (k-2)t_{AO} + t_{mux}$$

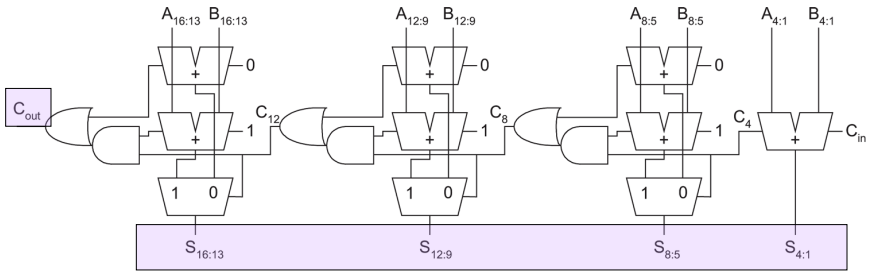


- Ο χρόνος για τον υπολογισμό των Propagate και Generate για κάθε block
- Ο υπολογισμός των Propagate και Generate γίνεται σε ένα slot
- Επειδή τα P,G και των k blocks υπολογίζονται ταυτόχρονα => η συνολική καθυστέρηση είναι ίση με αυτή του ενός block, δηλαδή $t_{pg}$





# Carry-Select Adder (10/12)



Στον συγκεκριμένο 16-bit Full Adder έχουμε

**n=4**

**k=4**

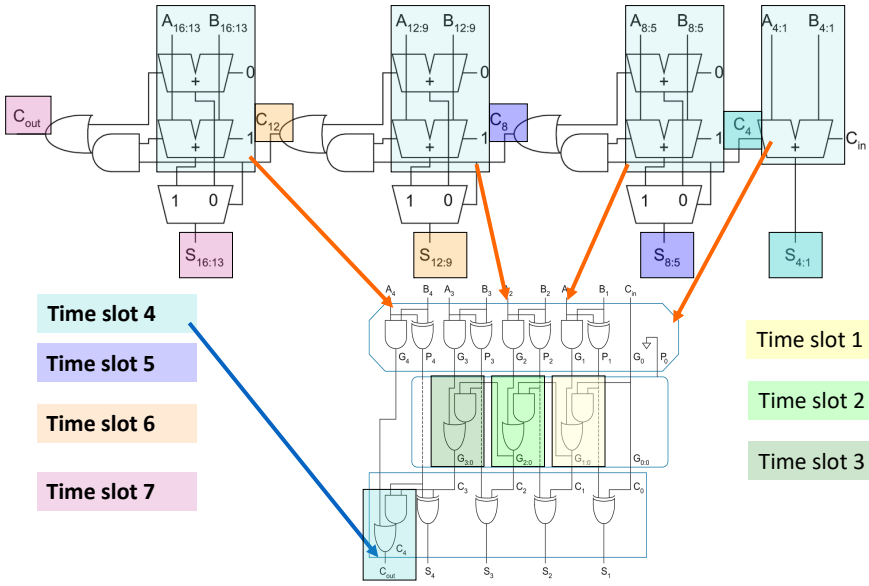
N=16 bits (k·n)

**Critical path του carry-select**

$$t_{select} = t_{pg} + nt_{AO} + (k-2)t_{AO} + t_{mux}$$

Click for critical path delay details

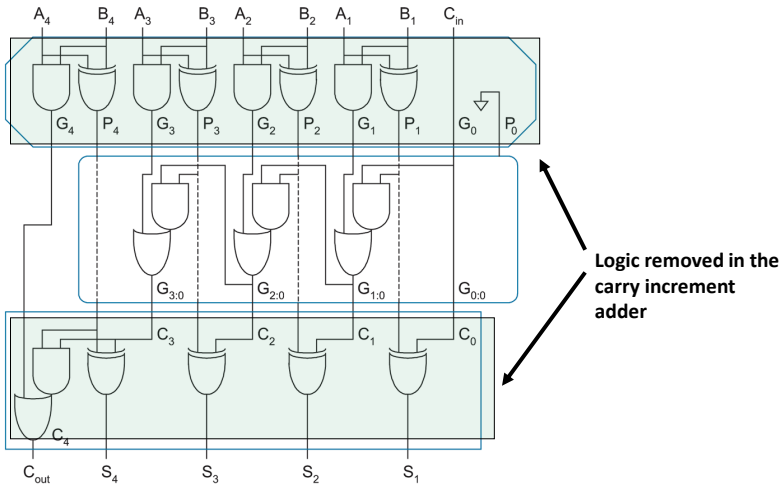
## Carry-Select Adder (11/12)






128

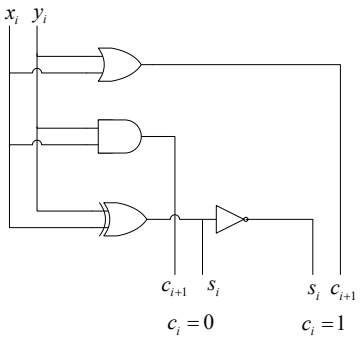
## Carry-Select Adder (12/12)






129

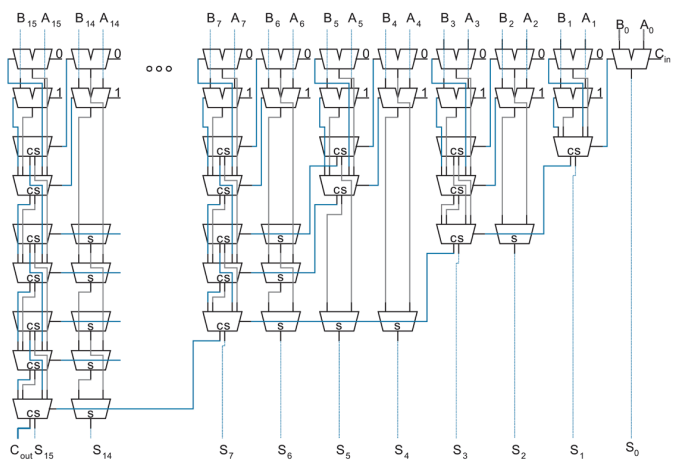
- Το κύκλωμα άθροισης χρησιμοποιεί κοινό hardware για τις δύο περιπτώσεις.




130

## Conditional-Sum Adder (1/7)

| Block Width | Block Carry In |
|---|---|
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| 4 | 0 |
| | 1 |
| 8 | 0 |
| | 1 |
| 16 | 0 |



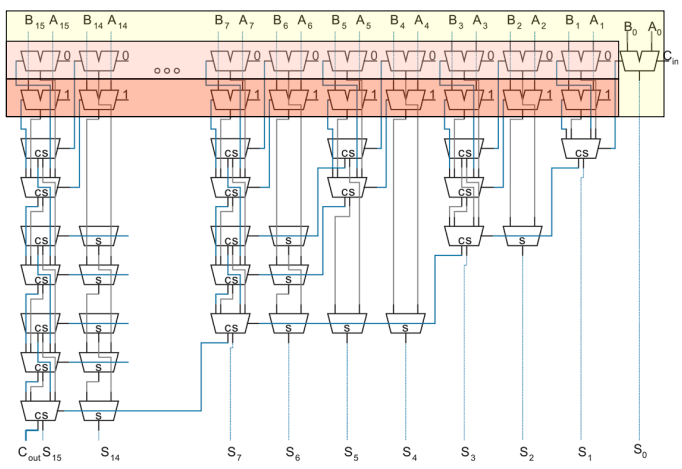

➢ Η λειτουργία του στηρίζεται στην λειτουργία του carry select adder



136

## Conditional-Sum Adder (2/7)

| Block Width | Block Carry In |
|---|---|
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| 4 | 0 |
| | 1 |
| 8 | 0 |
| | 1 |
| 16 | 0 |



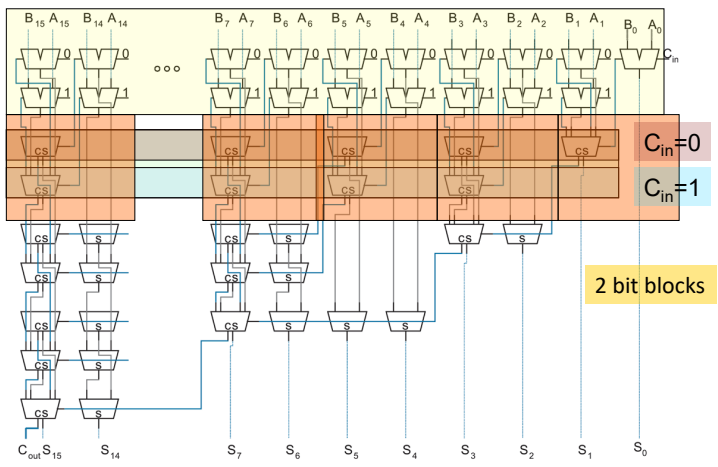

➢ Στις 2 πρώτες γραμμές οι full adders υπολογίζουν το sum και carry-out για κάθε bit υποθέτοντας για carry-in 0 και 1αντίστοιχα



137

## Conditional-Sum Adder (3/7)

| Block Width | Block Carry In |
|---|---|
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| 4 | 0 |
| | 1 |
| 8 | 0 |
| | 1 |
| 16 | 0 |

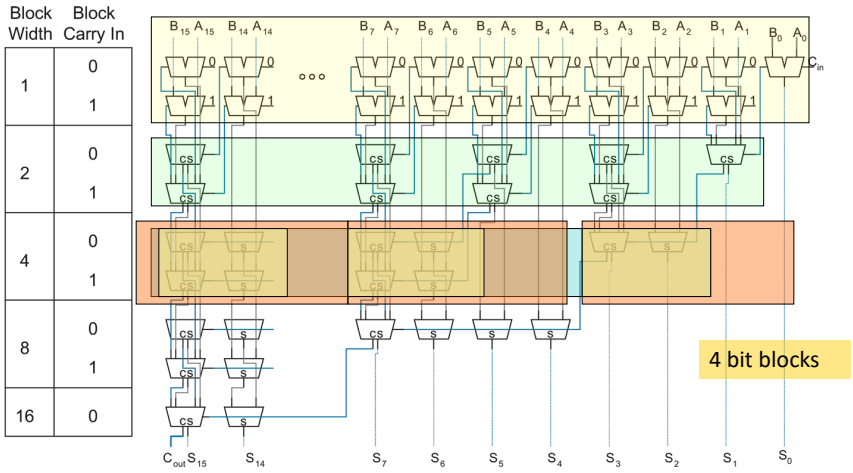



➢ Στις επόμενες 2 γραμμές τα ζεύγη από Multiplexers επιλέγουν το sum και carry-out από το ανώτερο επίπεδο για κάθε block που αποτελείται από 2 bit, υποθέτοντας πάλι για carry-in 0 και 1.



138

## Conditional-Sum Adder (4/7)

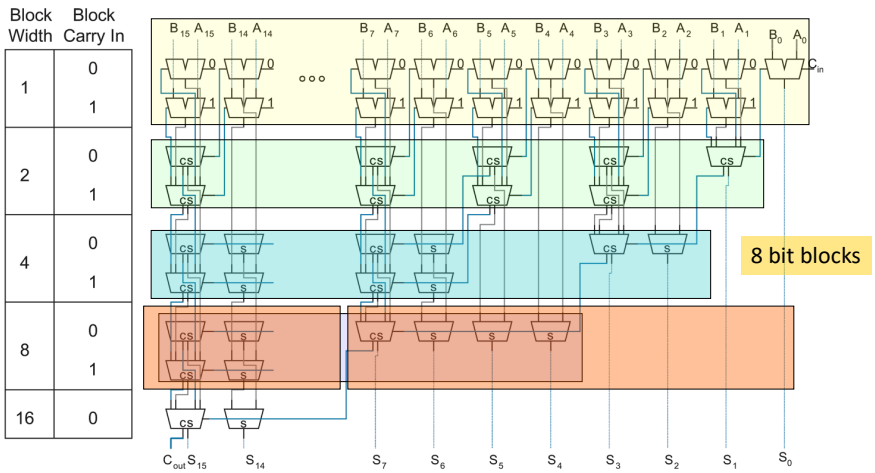



➢ Στις επόμενες 2 γραμμές τα ζεύγη από Multiplexers επιλέγουν το sum και carry-out 2-bits από το προηγούμενο επίπεδο για κάθε 4-bit block



139

## Conditional-Sum Adder (5/7)



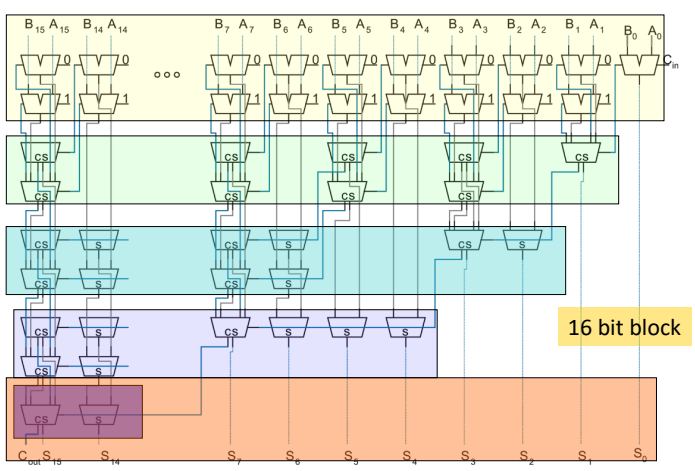

➢ Ομοίως ισχύει και για τα επόμενες 2 γραμμές οι οποίες επιλεγούν sum και carry για block 8-bits



140

## Conditional-Sum Adder (6/7)

| Block Width | Block Carry In |
|---|---|
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| 4 | 0 |
| | 1 |
| 8 | 0 |
| | 1 |
| 16 | 0 |



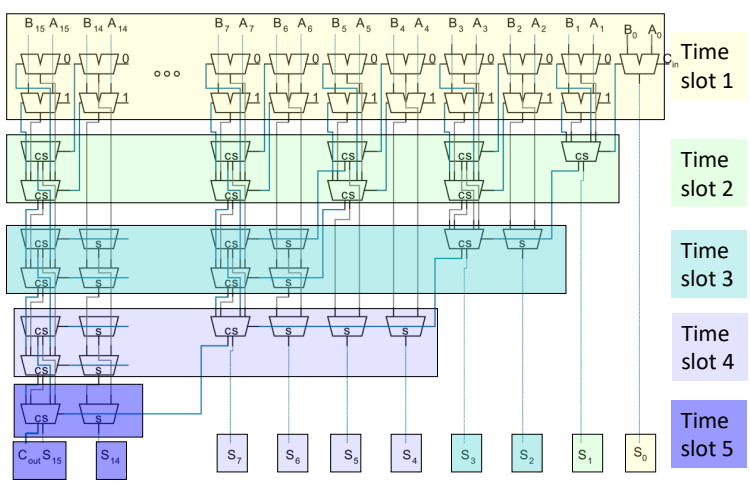

➢ Τέλος η τελευταία γραμμή επιλέγει το τελικό Carry-out και το sum για block 16-bit.



141

## Conditional-Sum Adder (7/7)

| Block Width | Block Carry In |
|---|---|
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| 4 | 0 |
| | 1 |
| 8 | 0 |
| | 1 |
| 16 | 0 |



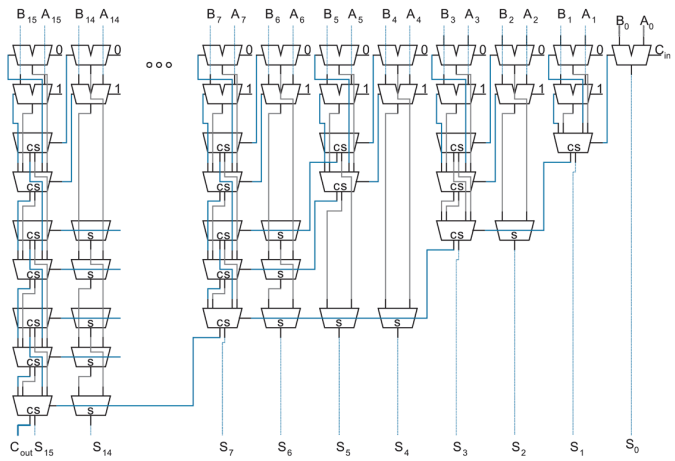

Ο **carry select adder** υπολογίζει το άθροισμα 2 16-bit αριθμών σε **7 times slots** (click for details) ενώ ο **conditional sum adder** σε **5 times slots**



142

## Conditional-Sum Adder

| Block Width | Block Carry In |
|---|---|
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| 4 | 0 |
| | 1 |
| 8 | 0 |
| | 1 |
| 16 | 0 |

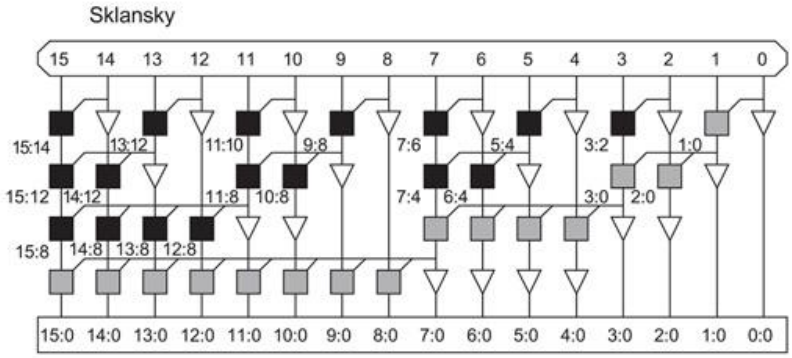



Ο conditional sum adder περιέχει ***2N* full adders** και ***2Nlog$_2$N* multiplexters**



152

## Conditional-Sum Adder- Βελτιώσεις




- Όπως με τον carry-select adder έτσι και ο Conditional Sum Adder μπορεί να βελτιωθεί υπολογίζοντας το sum με XORs και αντικαθιστώντας τους multiplexers με AND-OR πύλες.

- Αυτό μας οδηγεί σε Sklansky tree adder (αναλύεται παρακάτω)



153

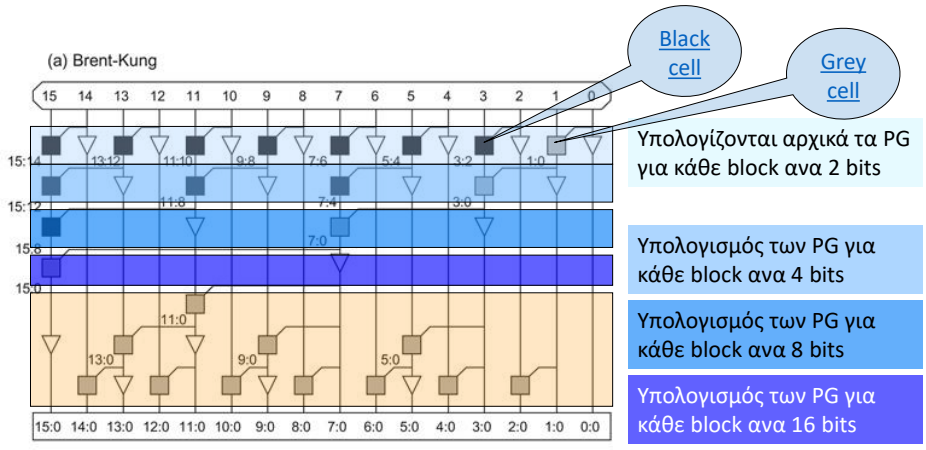
## Tree Adders

- 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



154

## Brent Kung (1/3)

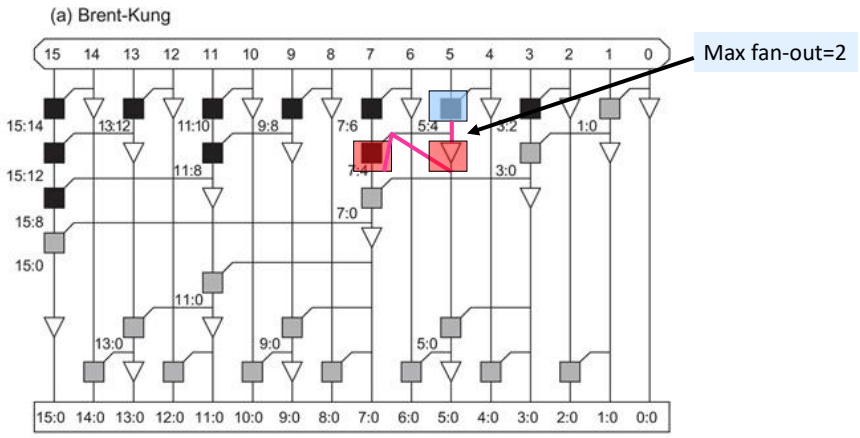



Υπολογισμός σε μορφή δέντρου των απαραίτητων G των ενδιάμεσων bit για το τελικό αποτέλεσμα



155

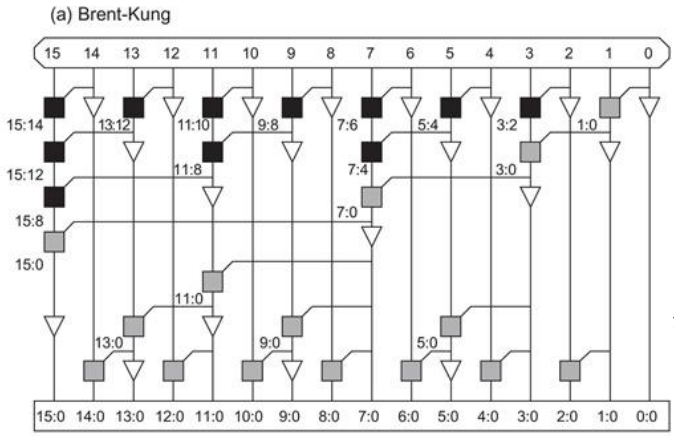
## Brent Kung (2/3)






156

## Brent Kung (3/3)



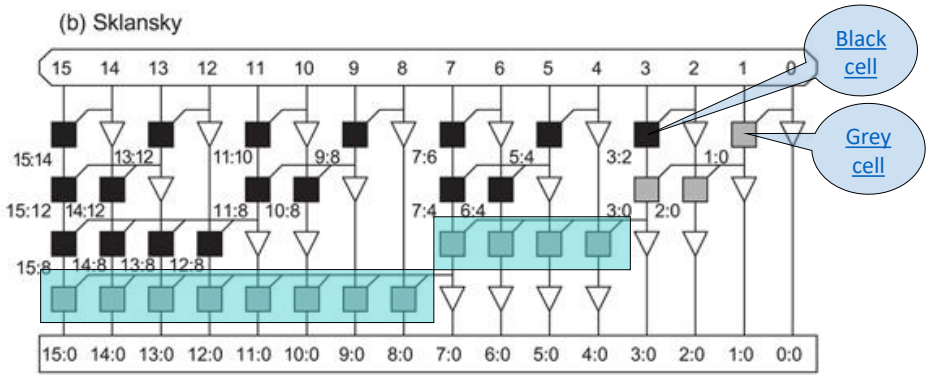

- Δεν καταλαμβάνει μεγάλη επιφάνεια
- Δεν έχει πρόβλημα πυκνότητας καλωδίων

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



157

## Sklansky (1/4)

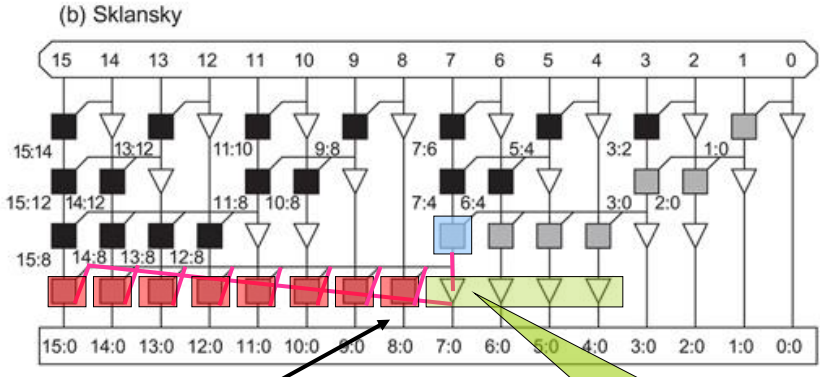



➢ Υπολογίζει πολλά μαζί Generate μειώνοντας έτσι την καθυστέρηση σε σχέση με τον Brent-Kung tree adder



158

## Sklansky (2/4)




Max fan-out=8

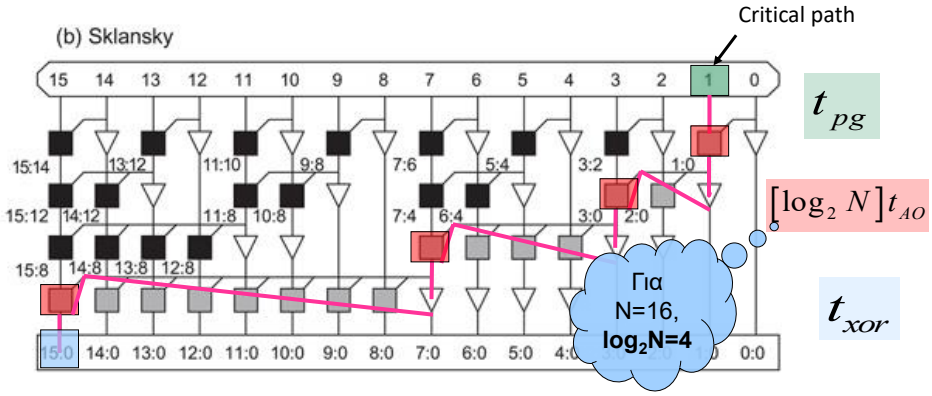
Ο βαθμός οδήγησης εξόδου x2 σε κάθε στάδιο [8, 4, 2, 1]

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



159

## Sklansky (3/4)



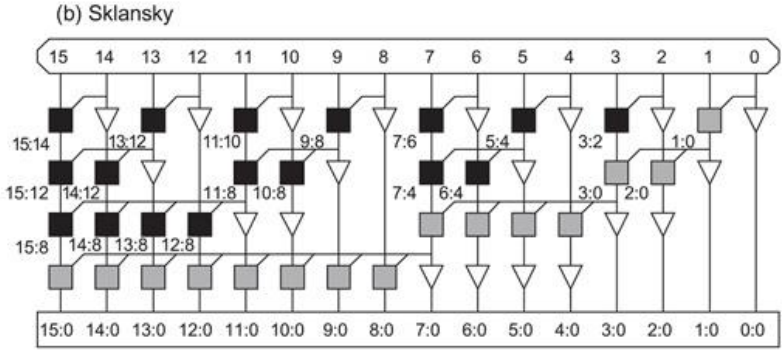

Critical path delay του Slansky (ισχύει και για Kogge-Stone tree adder):

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



160

## Sklansky (4/4)

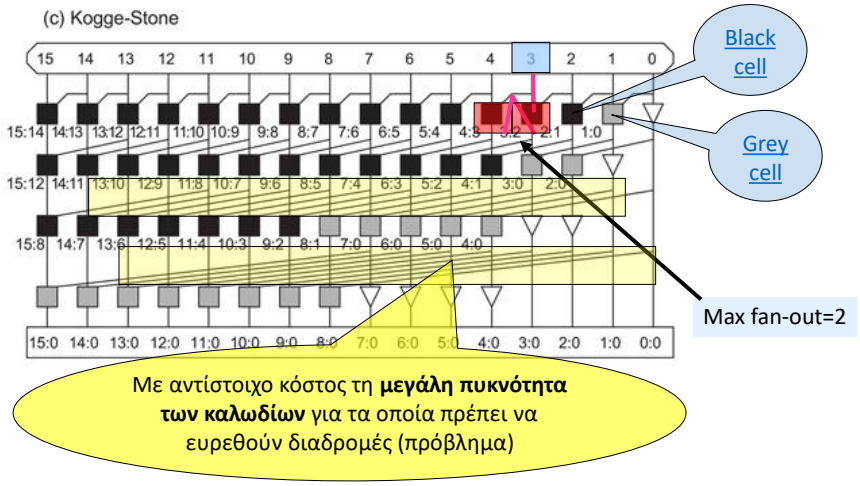



- Απαιτεί **$\log_2 N$** επίπεδα για τον τελικό υπολογισμό (μικρότερο delay από τον Brent-Kung)
- Καταλαμβάνει μεγαλύτερη επιφάνεια σε σχέση με τον Brent-Kung
- Δεν έχει πρόβλημα πυκνότητας καλωδίων



161

## Kogge-Stone (1/3)

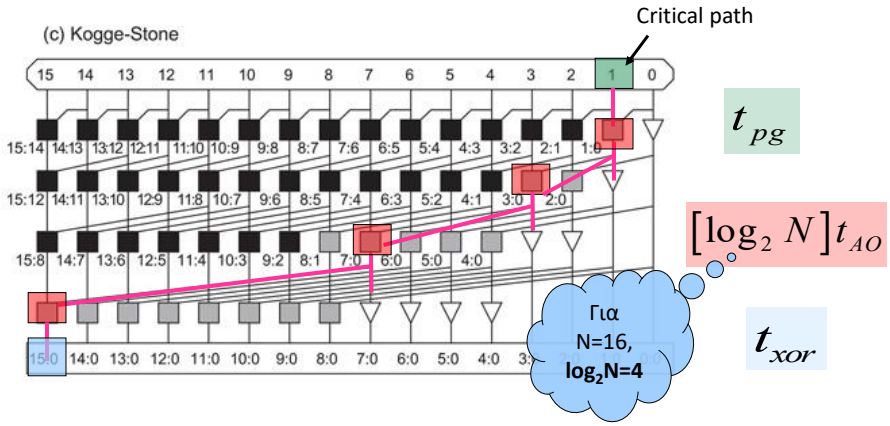



Μειώνει το max fan-out σε 2 (μία από τις διαδρομές με max fan-out)





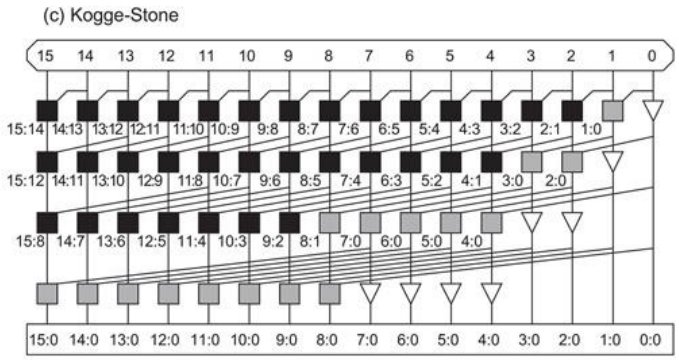
## Kogge-Stone (2/3)




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

$$t_{tree} = t_{pg} + [\log_2 N]t_{AO} + t_{xor}$$

## Kogge-Stone (3/3)

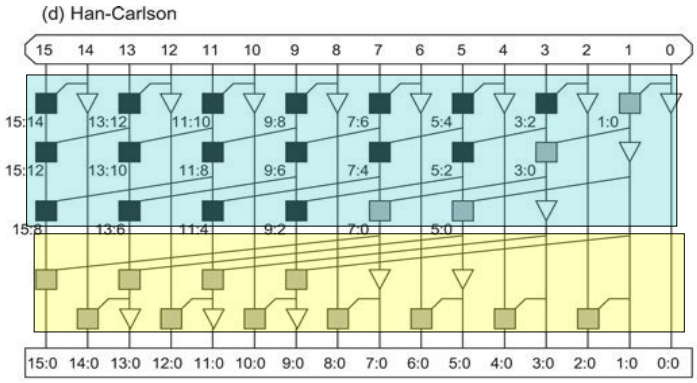



- Απαιτεί **$\log_2 N$** επίπεδα για τον τελικό υπολογισμό
- Καταλαμβάνει μεγάλη επιφάνεια (μεγάλος αριθμός από black cells)
- Έχει μεγάλο αριθμό καλωδίων για τα οποία πρέπει να ευρεθούν διαδρομές (πρόβλημα)



164

## Han – Carlson (1/4)

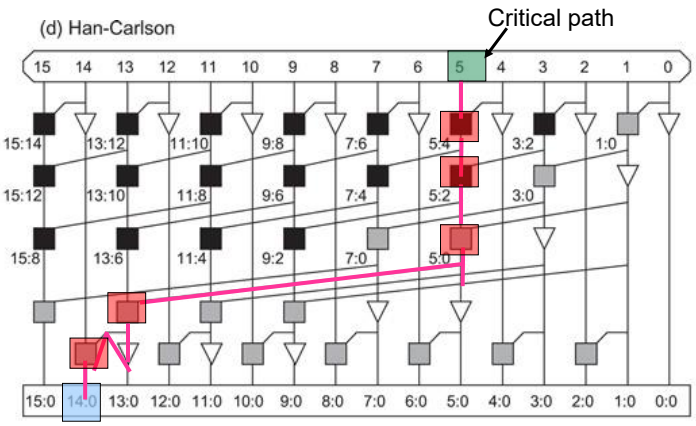



- Οι Han-Carlson trees είναι μία οικογένεια από δίκτυα μεταξύ Kogge-Stone και Brent-Kung
- Στο διάγραμμα χρησιμοποιεί την τεχνική Brent-Kung
- Μείωση επιπέδων με χρήση τεχνικής (μακριών καλωδίων) του Kogge-Stone



165

## Han – Carlson (2/4)




$t_{pg}$

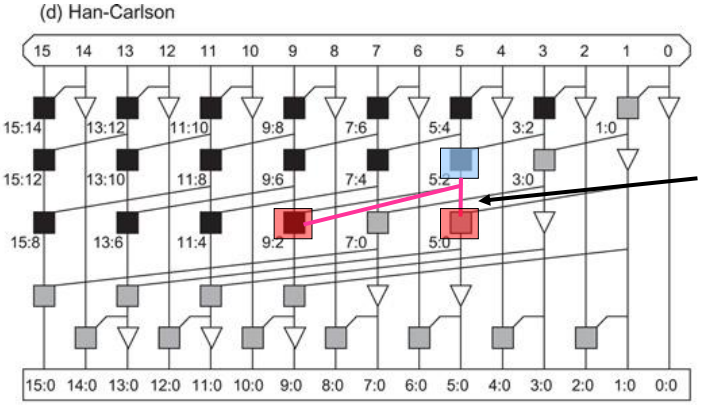
$5t_{AO}$

$t_{xor}$



166

## Han – Carlson (3/4)



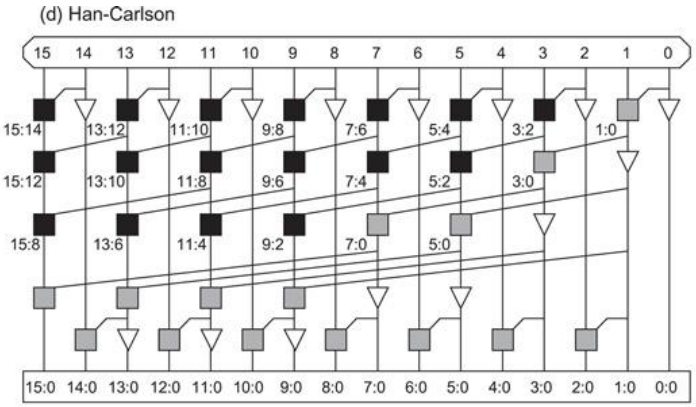

➢ Max fan-out=2

➢Μία από τις διαδρομές με max fan-out



167

## Han – Carlson (4/4)

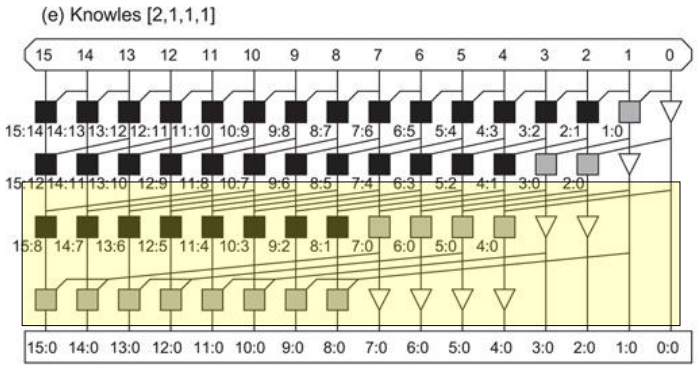



- Έχει ικανοποιητική καθυστέρηση
- Δεν καταλαμβάνει μεγάλη επιφάνεια (αριθμός πυλών)
- Δεν έχει μεγάλη πυκνότητα καλωδίων



168

## Knowles (1/4)

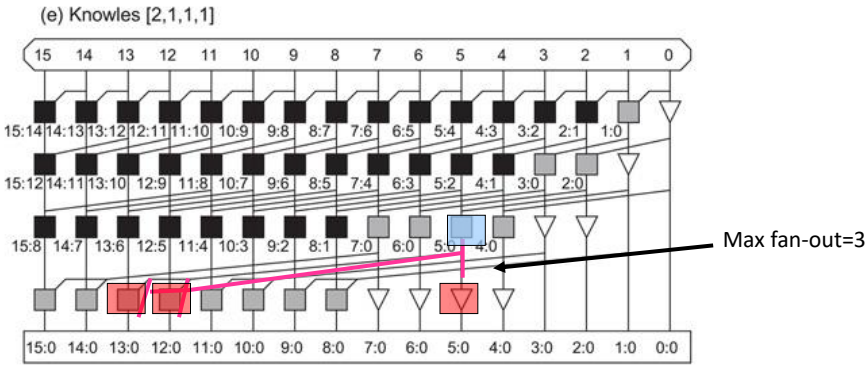



- Είναι συνδυασμός Kogge-Stone και Slansky.
- Το πρόβλημα του μεγάλου fan-out που παρουσιάζει ο Slansky επιλύεται με τη μέθοδο (μεγάλων καλωδίων) Kogge-Stone



169

## Knowles (2/4)

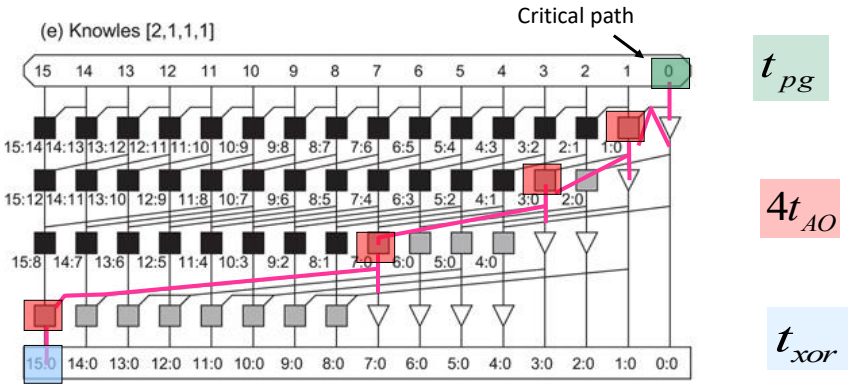



Μία από τις διαδρομές με max fan-out



170

## Knowles (3/4)




$t_{pg}$

$4t_{AO}$

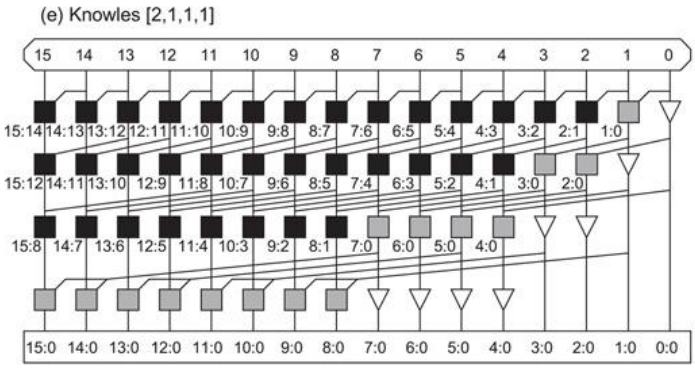
$t_{xor}$

Μία από τις διαδρομές με την μέγιστη καθυστέρηση



171

## Knowles (4/4)

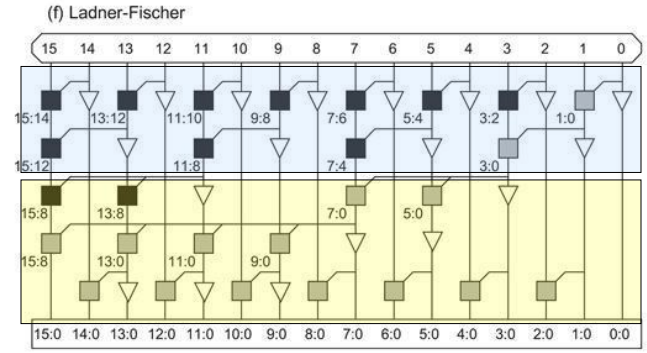



- Απαιτεί **$\log_2 N$** επίπεδα για τον τελικό υπολογισμό (η μικρότερη καθυστέρση)
- Καταλαμβάνει μεγάλη επιφάνεια (αριθμός πυλών)
- Δεν έχει μεγάλο πρόβλημα πυκνότητας καλωδίων



172

## Lander – Fischer (1/4)



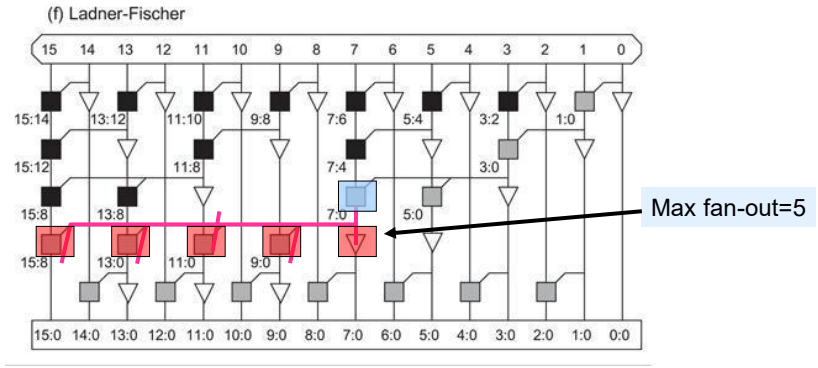

- Οι Ladner-Fischer trees είναι οικογένεια δικτύων μεταξύ Slansky και Brent-Kung

- Αρχικά γίνονται οι υπολογισμοί όπως στους Brent-Kung trees
- Ενώ οι υπόλοιποι υπολογισμοί γίνονται με τη μέθοδο των Slansky trees



173

## Lander – Fischer (2/4)

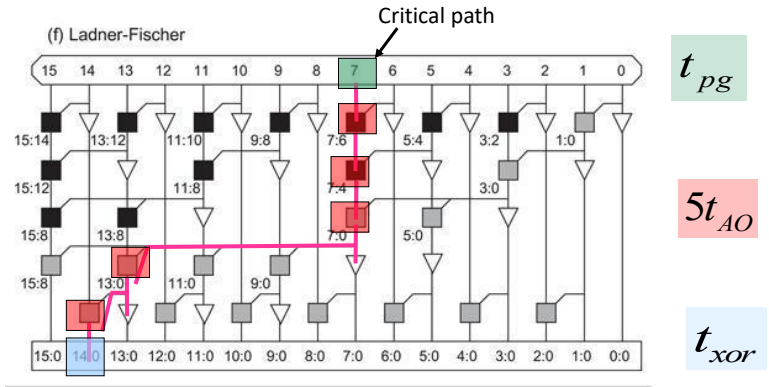



Η διαδρομή με max fan-out



174

## Lander – Fischer (3/4)




$t_{pg}$

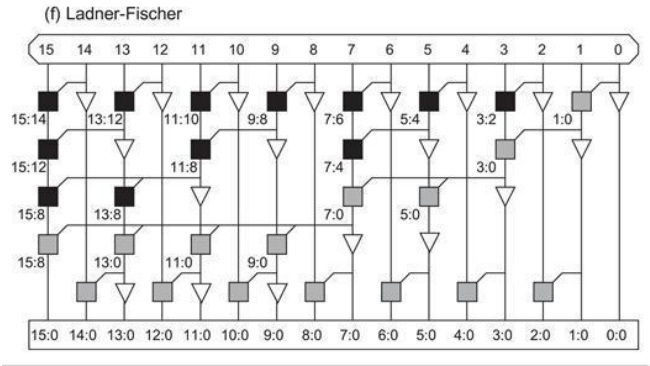
$5t_{AO}$

$t_{xor}$



175

## 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



176

## 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]**

Αντιστοίχιση χαρακτηριστικών με μεταβλητές:

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

Ιδανική περίπτωση **$L=\log_2 N$**

Ιδανική περίπτωση fan-out: **2**



177

## Ομαδοποίηση tree adders (2/2)

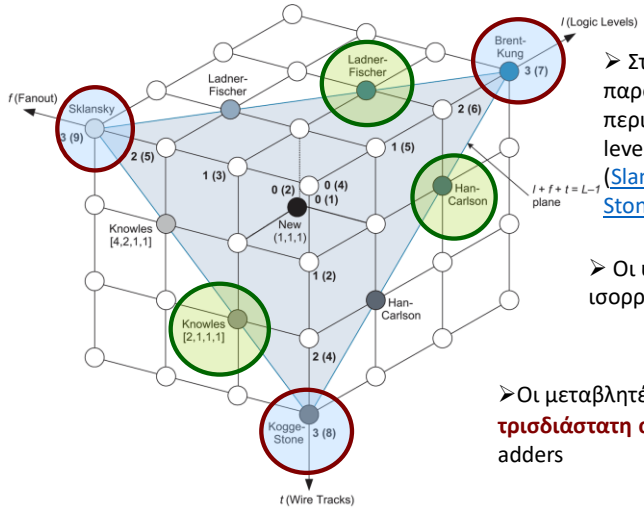


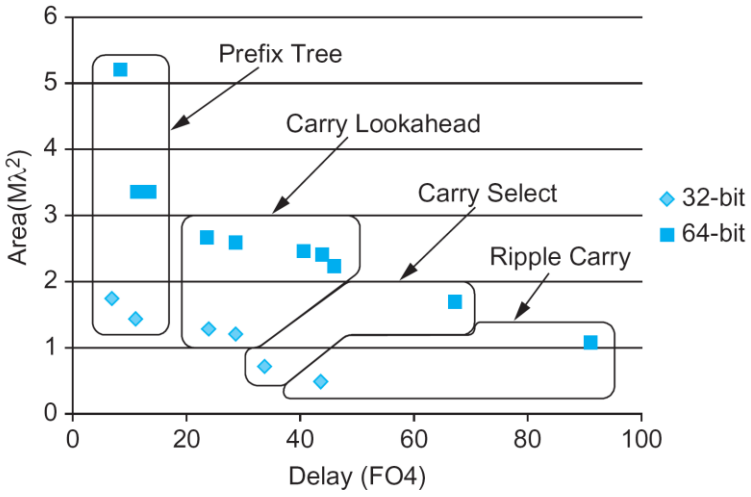

FIG 10.35 Taxonomy of prefix networks

- Στις 3 άκρες του κύβου παρουσιάζονται οι ακραίες περιπτώσεις των trees σε Logic levels (Brent-Kung), Fanout (Slansky) και wire Tracks (Kogge-Stone)
- Οι υπόλοιποι έχουν πιο ισορροπημένα tradeoffs
- Οι μεταβλητές **(l,f,t)** δίνουν μία **τρισδιάστατη απεικόνιση** των tree adders



178

Σύνοψη – Αθροιστών (Επιφάνεια – Καθυστέρηση μετά από σύνθεση)

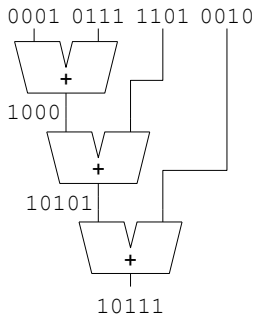





192

# Πρόσθεση Πολλών Εισόδων (Multi-input Adders)

- Υποθέστε ότι θέλουμε να προσθέσουμε k N-bit λέξεις
  - Παράδειγμα: 0001 + 0111 + 1101 + 0010 = 10111
- Άμεση λύση: k-1 N-input CPAs
  - Μεγάλη επιφάνεια
  - Μεγάλη καθυστέρηση

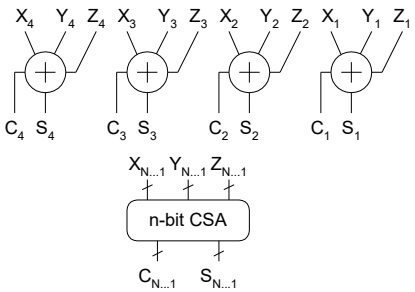






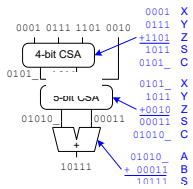

# Carry Save Addition

- Ένας A full adder αθροίζει 3 εισόδους και παράγει 2 εξόδους (άθροισμα & κρατούμενο)
  - Το κρατούμενο έχει διπλάσιο βάρος από το άθροισμα
  - X+Y+Z = S +2C
- Χρήση παράλληλων N full adders – η δομή καλείται *carry save adder* ή *[3:2] adders*
  - Παράγει N sums και N carry εξόδους



# Εφαρμογή Carry Save Addition (CSA)

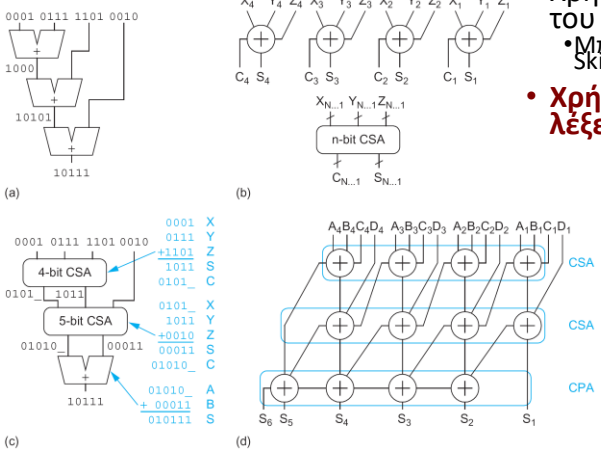
- Χρήση k-2 βαθμίδων από CSAs
  - Δεν υπάρχει μετάδοση του κρατουμένου
- Χρήση ενός τελικού CPA για τον υπολογισμό του αποτελέσματος
  - Μπορεί να είναι οτιδήποτε: RCA, Carry select, Carry Skip ….
- **Χρήση σε πολ/στές για άθροιση πολλαπλών λέξεων μερικών γινομένων !!!**





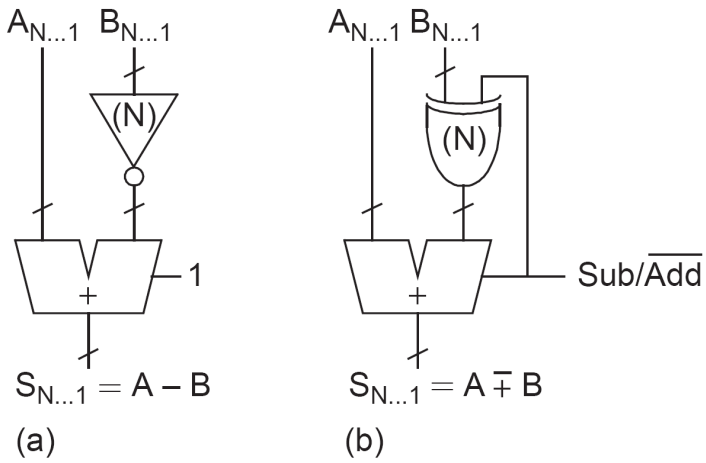



## CSA – Κυκλωματική Δομή




- Χρήση k-2 βαθμίδων από CSAs
  - Δεν υπάρχει μετάδοση του κρατουμένου
- Χρήση ενός τελικού CPA για τον υπολογισμό του αποτελέσματος
  - Μπορεί να είναι οτιδήποτε: RCA, Carry select, Carry Skip ….
- **Χρήση σε πολ/στές για άθροιση πολλαπλών λέξεων μερικών γινομένων !!!**

Αφαίρεση








# Περίγραμμα Διάλεξης

- Πρόσθεση / Αφαίρεση
- Ανιχνευτές 1/0
- Συγκριτές
- Μετρητές
- Κωδικοποίηση
- Ολισθητές
- Πολλαπλασιασμός

# Συγκριτές

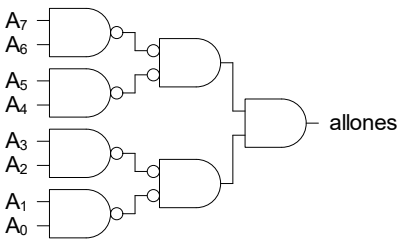
- 0's ανιχνευτής : A = 00…000
- 1's ανιχνευτής : A = 11…111
- Συγκριτής ισότητας: A = B
- Συγκριτής μέτρου: A < B

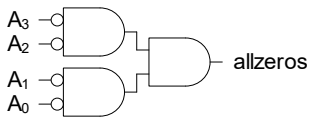




# 1's & 0's Ανιχνευτές (1/3)

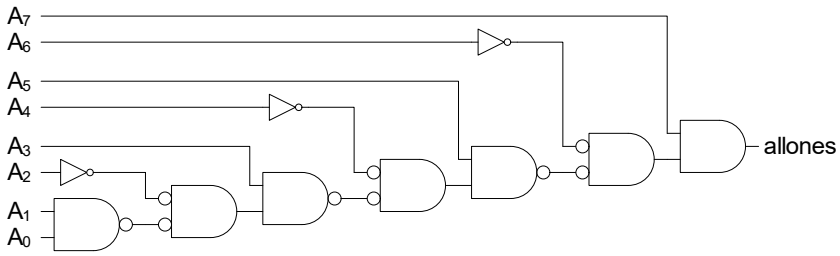
- 1's detector: N-input AND gate
- 0's detector: NOTs + 1's detector (N-input NOR)



# 1's & 0's Ανιχνευτές (2/3)

- Αν η λέξη που ελέγχεται έχει μια απόκλιση ως προς το χρόνο άφιξης των εξόδων θα μπορούσε να χρησιμοποιηθεί μία ασύμμετρη σχεδίαση
- Εδώ, η καθυστέρηση από την τελευταία έξοδο που αλλάζει, *A7*, είναι μια καθυστέρηση ενός κύκλου

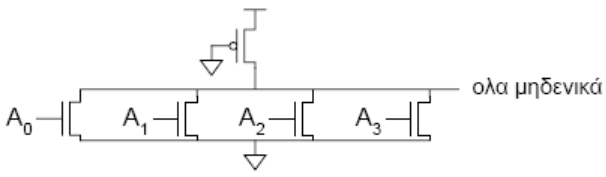



201

201

# 1's & 0's Ανιχνευτές (3/3)

- Χρήση δομής ψευδο-nMOS ή δυναμική NOR για υλοποίηση του καλωδιωμένο-OR
- Ικανοποιητικό για λέξεις ως και 16 bit
- Για μεγαλύτερου μεγέθους λέξεις, οι πύλες μπορούν να χωριστούν σε δομικές μονάδες των 8 με 16 bit
  - μείωση καθυστέρησης παρασιτικών στοιχείων
  - αποφυγή προβλημάτων με το ρεύμα διαρροής της περιοχής υποκατωφλίου.

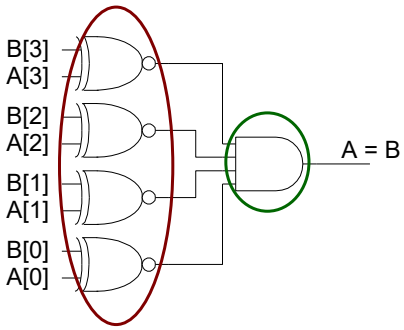



202

202

# Συγκριτής ισότητας

- Έλεγχος ισότητας ανά ψηφίο (XNOR, aka equality gate)
- 1's ανιχνευτής

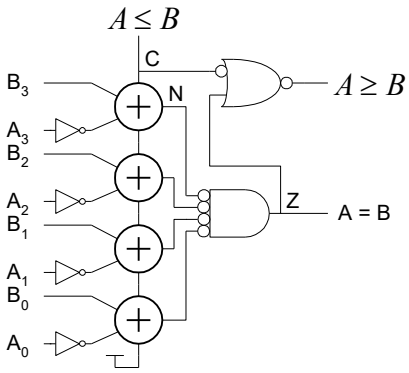





203

# Συγκριτής μέτρου (1/3)

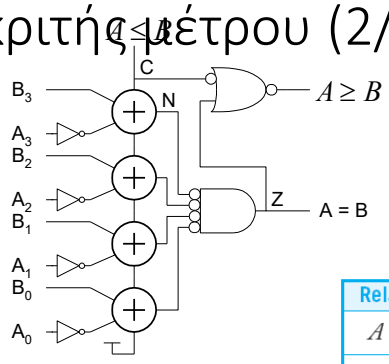
- Για μη προσημασμένους αριθμούς A και B, υπολογίζουμε το *B*-*A* = *B*+ *A'* +1
  - Αν το κρατούμενο εξόδου δεν είναι μηδέν, τότε A ≤ B
  - . Ένας ανιχνευτής του 0 δείχνει ότι οι αριθμοί είναι ίσοι. Το






204

# Συγκριτής μέτρου (2/3)




| Relation | Unsigned Comparison | Signed Comparison |
|---|---|---|
| $A = B$ | $Z$ | $Z$ |
| $A \neq B$ | $\overline{Z}$ | $\overline{Z}$ |
| $A < B$ | $C \cdot \overline{Z}$ | $\overline{S} \cdot \overline{Z}$ |
| $A > B$ | $C$ | $S$ |
| $A \leq B$ | $C$ | $\overline{S}$ |
| $A \geq B$ | $\overline{C} + Z$ | $S + Z$ |



# Συγκριτής μέτρου (3/3)

- Η σύγκριση προσημασμένων αριθμών σε συμπλήρωμα ως προς 2 είναι πιο σύνθετη λόγω της εμφάνισης υπερχείλισης
- Αντί να ελέγχεται το κρατούμενο εξόδου, πρέπει να καθοριστεί αν το αποτέλεσμα είναι αρνητικό (Negative –N)
  - πιο σημαντικό δυαδικό ψηφίο του αποτελέσματος
- και αν υπάρχει υπερχείλιση (Overflow –V)
  - V = 1, αν οι είσοδοι έχουν διαφορετικά πρόσημα (τα MSBs είναι διαφορετικά) και το πρόσημο εξόδου είναι διαφορετικό από το πρόσημο του B
- Το πραγματικό πρόσημο της διαφοράς *B-A* είναι το (*N XOR V*)
  - η υπερχείλιση αναστρέφει το πρόσημο
- Αν το πρόσημο είναι αρνητικό γνωρίζουμε ότι *A* > *B*
  - Οι άλλες σχέσεις μεταξύ των *A* και *B* μπορούν να εξαχθούν από το σήμα που δίνει το πρόσημο και το σήμα Z.

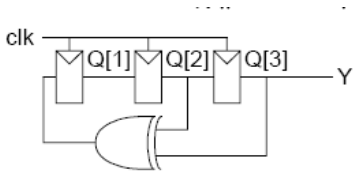
# Signed vs. Unsigned

| Relation | Unsigned Comparison | Signed Comparison |
|---|---|---|
| $A = B$ | $Z$ | $Z$ |
| $A \neq B$ | $\overline{Z}$ | $\overline{Z}$ |
| $A < B$ | $C \cdot \overline{Z}$ | $\overline{S} \cdot \overline{Z}$ |
| $A > B$ | $C$ | $S$ |
| $A \leq B$ | $C$ | $\overline{S}$ |
| $A \geq B$ | $\overline{C} + Z$ | $S + Z$ |





Καταχωρητές ολίσθησης γραμμικής ανατροφοδότησης (LFSR)




Πίνακας 10.6 Ακολουθία LFSR

| Κύκλος | Q[1] | Q[2] | Q[3] / Y |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |
| επαναλαμβάνεται χωρίς τέλος | | | |

- Ένας (LFSR –Linear Feedback Shift Register) αποτελείται από N καταχωρητές συνδεδεμένους σε σειρά
- Η είσοδος έρχεται από μια πύλη XOR συγκεκριμένων ψηφίων
- Αυτός ο LFSR είναι ένας μέγιστου μήκους καταχωρητή ολίσθησης
  - η έξοδός του διέρχεται διαδοχικά από όλους τους 2n-1 συνδυασμούς
- Οι είσοδοι της πύλης XOR ονομάζονται ακολουθία λήψεων και συχνά προσδιορίζονται από ένα χαρακτηριστικό πολυώνυμο
  - χαρακτηριστικό πολυώνυμο $1+x^2+x^3$ (οι είσοδοι από το $2^o$ και $3^o$ καταχωρητή)

# Καταχωρητές ολίσθησης γραμμικής ανατροφοδότησης

- Η έξοδος Y προκύπτει από την ακολουθία 7 bit [1110010]
  - παράδειγμα μιας ψευδοτυχαίας ακολουθίας
- Οι LFSRs χρησιμοποιούνται ως:
  - μετρητές υψηλής ταχύτητας
  - γεννήτριες ψευδοτυχαίων αριθμών
- Οι ψευδοτυχαίες ακολουθίες είναι βολικές για ενσωματωμένη αυτοδοκιμή και έλεγχο του ρυθμού μετάδοσης λαθών σε τηλεπικοινωνιακές συνδέσεις





# Περίγραμμα Διάλεξης

- Πρόσθεση / Αφαίρεση
- Ανιχνευτές 1/0
- Συγκριτές
- Μετρητές
- Κωδικοποίηση
- Ολισθητές
- Πολλαπλασιασμός

# Ολισθητές (Shifters)

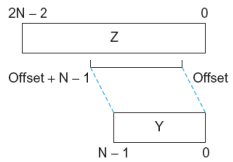
- Λογική ολίσθηση (Logical Shift):
  - Shifts number left or right and fills with 0's
    - 1011 LSR 1 = 0101  1011 LSL
- Αριθμητική Ολίσθηση (Arithmetic Shift):
  - Shifts number left or right. Rt shift sign extends
    - 1011 ASR1 = 1101  1011 ASL1 = 0110
- Περιστροφή (Rotate):
  - Shifts number left or right and fills with lost bits
    - 1011 ROR1 = 1101  1011 ROL1 = 0111



# Ολισθητής Χοάνης (Funnel Shifter)

**Πίνακας 10.11 Απλοποιημένος ολισθητής συγκερασμού**

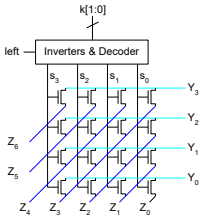
| Shift Type | Z | Offset |
|---|---|---|
| Logical Right | $0..0, A_{N-1}...A_0$ | $k$ |
| Logical Left | $A_{N-1}...A_0, 0..0$ | $\overline{k}$ |
| Arithmetic Right | $A_{N-1}...A_{N-1}, A_{N-1}...A_0$ | $k$ |
| Arithmetic Left | $A_{N-1}...A_0, 0..0$ | $\overline{k}$ |
| Rotate Right | $A_{N-2}...A_0, A_{N-1}...A_0$ | $k$ |
| Rotate Left | $A_{N-1}...A_0, A_{N-1}...A_1$ | $\overline{k}$ |




- Δημιουργεί μία λέξη 2N-1 bits και επιλέγει ένα υπό-πεδίο Y των N bits
  - Ο πίνακας δείχνει τη χρήση των εισόδων για ολίσθηση μιας λέξης A των *N* bit κατά *k* bit.

# Array Funnel Shifter

- N N-input multiplexers
  - Use 1-of-N hot select signals for shift amount
  - nMOS pass transistor design ($V_t$ drops!)

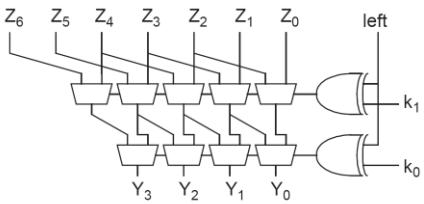





213

# Logarithmic Funnel Shifter

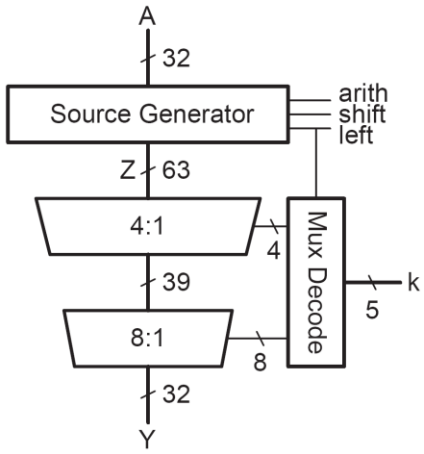
- Η προηγούμενη διάταξη λειτουργεί καλά για μεσαίου μεγέθους ολισθητές αλλά έχει υψηλή παρασιτική χωρητικότητα σε μεγαλύτερους ολισθητές
- Λογαριθμικός ολισθητής χοάνης – Βασίζεται σε πολλαπλά επίπεδα μικρότερων πολυπλεκτών
  - το πρώτο επίπεδο ολισθαίνει κατά N/2, το δεύτερο κατάN/4 κοκ, μέχρι το τελευταίο επίπεδο να ολισθήσει κατά 1 – δε χρειάζεται κωδικοποιητής
- Οι πύλες XOR στις εισόδους ελέγχου αναστρέφουν υπό συνθήκη το μέγεθος που υφίσταται ολίσθηση, όταν πρόκειται για αριστερές ολισθήσεις






214

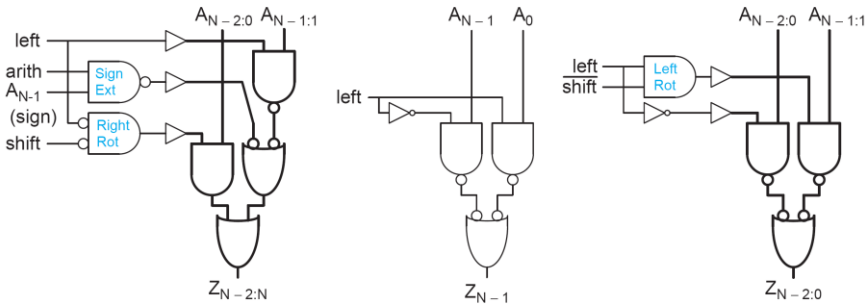
# 32-bit Logarithmic Funnel Shifter






215

# Γεννήτρια λέξεων

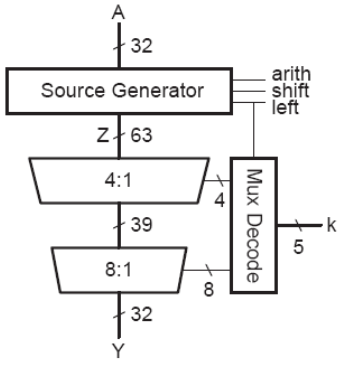


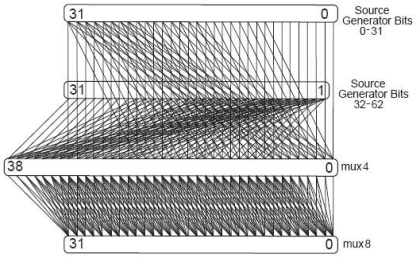



216

# 32-bit Logarithmic Funnel

- Μεγάλοι πολυπλέκτες για μείωση καθυστέρησης και κατανάλωσης
- Λέξεις με εί ένειες



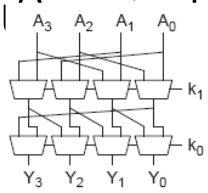




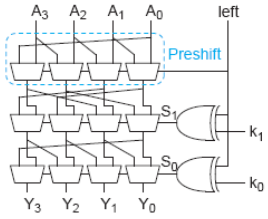

# Περιστροφικός Ολισθητής – Barrel Shifter

- Εκτελεί δεξιές περιστροφές
- Χειρίζεται τις αριστερές περιστροφές χρησιμοποιώντας το συμπληρωματικό του ποσού ολίσθησης
- Οι ολισθήσεις γίνονται με περιστροφές όταν έχουν το κατάλληλο κύκλωμα για masking
- Υλοποιούνται σε δομή πίνακα αλλά και λογαριθμική
- Η λογαριθμική είναι κατάλληλη για μεγάλες ολισθήσεις

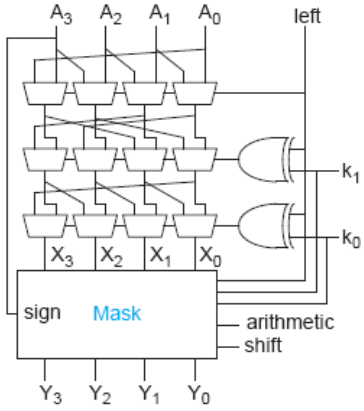
# Λογαριθμικός Barrel Shifter




Right shift only




Right/Left shift

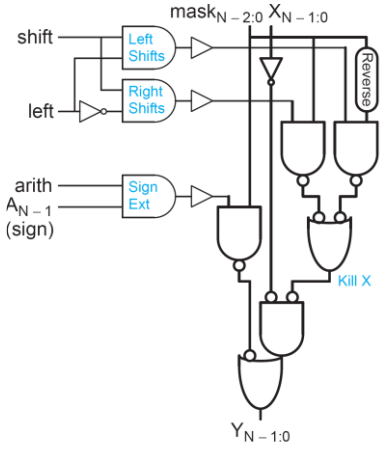



Right/Left Shift & Rotate



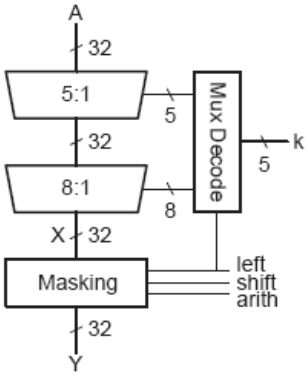


# Κύκλωμα μάσκας Barrel Shifter



# 32-bit Logarithmic Barrel

- Datapath never wider than 32 bits
- First stage preshifts by 1 to handle left shifts








# Περίγραμμα Διάλεξης

- Πρόσθεση / Αφαίρεση
- Ανιχνευτές 1/0
- Συγκριτές
- Μετρητές
- Κωδικοποίηση
- Ολισθητές
- Πολλαπλασιασμός

## Πολλαπλασιασμός (1/2)

- Ο πολλαπλασιασμός είναι μια λιγότερο συνηθισμένη πράξη απ' ότι η πρόσθεση
  - είναι όμως βασική για μP, ψηφιακούς επεξεργαστές σήματος και γραφικά
- Η βασικότερη μορφή αφορά στο σχηματισμό του γινομένου δύο μη προσημασμένων (θετικών) δυαδικών αριθμών
  - μπορεί να γίνει με την παραδοσιακή τεχνική στη βάση του 2
  - παράδειγμα, ο πολ/σμός δύο θετικών αριθμών των 4 bit

```
   011001 : 25_10      multiplicand
 × 100111 : 39_10      multiplier
   011001
  011001
 011001                 partial
000000                  products
 000000
+011001
001111001111 : 975_10   product
```

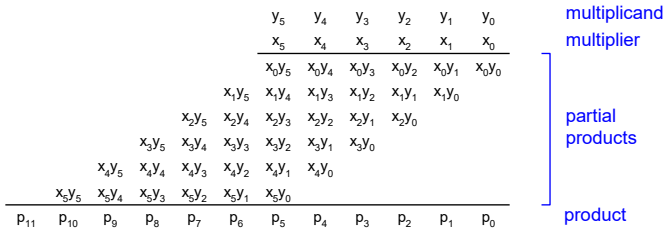


## Πολλαπλασιασμός (2/2)

- Ο πολλαπλασιασμός *M*×*N* bit μπορεί να θεωρηθεί ως:
  - Σχηματισμός N μερικών γινομένων M bits το καθένα
  - Κατάλληλη ολίσθηση και μερικών γινόμενων
  - Παραγωγή αποτελέσματος P ***M+N* bits**
- Ο δυαδικός πολλαπλασιασμός ισοδυναμεί με τη λογική πράξη AND
  - τα μερικά γινόμενα είναι το AND των ψηφίων του πολ/στή και του πολ/στέου
- Κάθε στήλη μερικών γινομένων πρέπει να προστεθεί και κάθε κρατούμενο περνά στην επόμενη στήλη
- Ορίζουμε τον πολλαπλασιαστέο ως *Y* = (*y*M-1, *y*M-2, …., *y*1, *y*0) και τον πολλαπλασιαστή ως *X* = (*x*N-1, *x*N-2 , …, *x*1, *x*0)
- Για πολλαπλασιασμό μη προσημασμένων αριθμών ισχύει: $\left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j}$

# Πολλαπλασιασμός –Γενική μορφή

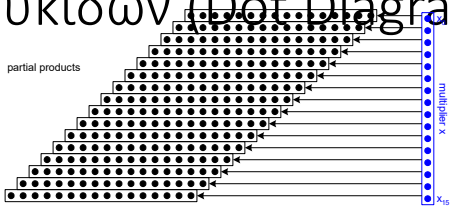
- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product: $$P = \left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$






225

# Διάγραμμα κουκίδων (Dot Diagram)




- Πολ/σμοί 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 πολ/στή που χρησιμοποιούνται για την παραγωγή των μερικών γινομένων φαίνονται στα δεξιά



226

## Γενικές αρχές υλοποίησης πολ/σμού (1/2)

- Πλήθος τεχνικών για την εκτέλεση του πολλαπλασιασμού
- Η επιλογή βασίζεται πάνω σε μετρικές σχεδιασμού
  - η καθυστέρηση, ο ρυθμός λειτουργίας (throughput), 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 πίνακες ή δένδρα αθροιστών για την πρόσθεση των μερικών γινομένων

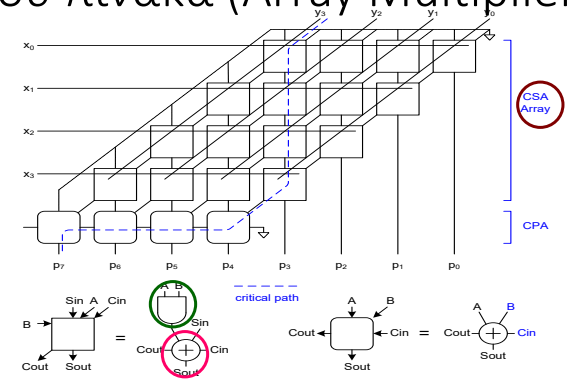




## Γενικές αρχές υλοποίησης πολ/σμού (2/2)

- 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

# Πολ/στής τύπου πίνακα (Array Multiplier) (1/3)

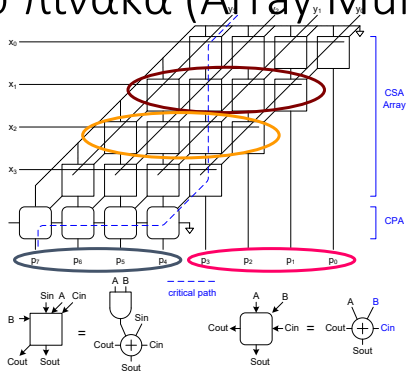



- Χρήση CSAs για την πρόσθεση των μερικών γινομένων
- Κάθε κύτταρο περιέχει μια πύλη AND δύο εισόδων
  - σχηματίζει ένα μερικό γινόμενο,
- και έναν αθροιστή CSA – πρόσθεση μερικού γινόμενου στο τρέχον άθροισμα



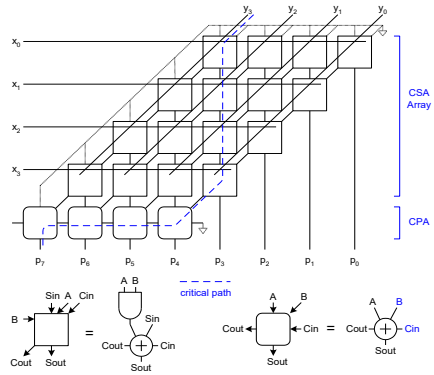


# Πολ/στής τύπου πίνακα (Array Multiplier) (2/3)




- Κάθε γραμμή χρησιμοποιεί CSAs για πρόσθεση μερικού γινομένου στο αποτέλεσμα της προηγούμενης και παραγωγή ένα τρέχοντος αθροίσματος-κρατουμένου
- Τα *N* LSBs είναι διαθέσιμα ως έξοδοι αθροίσματος κατευθείαν από τους CSA
- Τα MSBs παράγονται σε πλεονάζουσα μορφή αθροίσματος-κρατουμένου
  - χρήση *M*-bit CPA για μετατροπή σε κανονική δυαδική μορφή

# Πολ/στής τύπου πίνακα (Array Multiplier) (3/3)

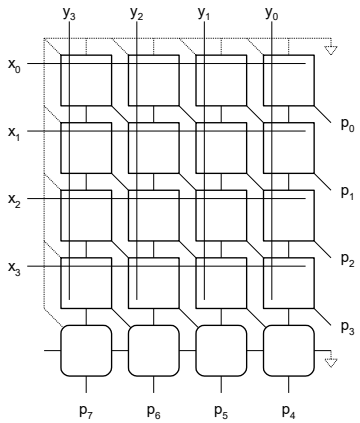



- Υποθέτοντας ότι σε έναν CSA η έξοδος κρατουμένου είναι γρηγορότερη από αυτή του αθροίσματος, το κρίσιμο μονοπάτι σημειώνεται με τη διακεκομμένη γραμμή
- Μπορεί εύκολα να εφαρμοστεί pipeline
  - καταχωρητές ανάμεσα στις γραμμές
- Χρήση ενός μόνο τύπο κυττάρου => εύκολη σχεδίαση & παραγωγή layout

231

231

# Rectangular Array



- Ίδιοι αθροιστές μετατοπισμένοι για να ταιριάζουν σε ένα ορθογώνιο σχήμα

232

232

# Πολ/σμός σε συμπλήρωμα ως προς 2 (1/4)

- Στον πολ/σμό σε συμπλήρωμα ως προς 2 κάποια μερικά γινόμενα είναι αρνητικά και πρέπει να αφαιρεθούν
  - Το πιο MSB ενός αριθμού σε συμπλήρωμα ως προς 2 έχει αρνητική αξία

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

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

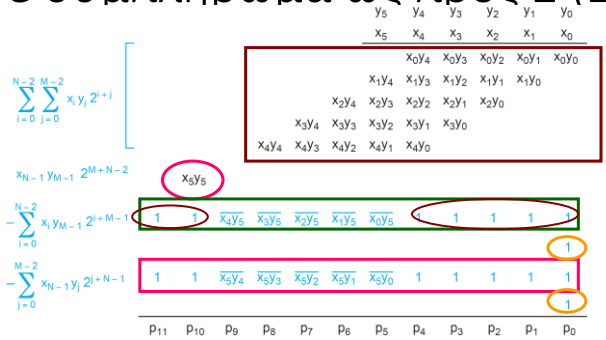
- Δύο από τα μερικά γινόμενα έχουν αρνητικό βάρος => πρέπει να αφαιρεθούν και όχι να προστεθούν

- Ο αλγόριθμος Baugh-Wooley χειρίζεται την αφαίρεση λαμβάνοντας το συμπλήρωμα ως προς 2 των όρων που πρόκειται να αφαιρεθούν
  - Αναστρέφοντας τα δυαδικά ψηφία και προσθέτοντας το 1



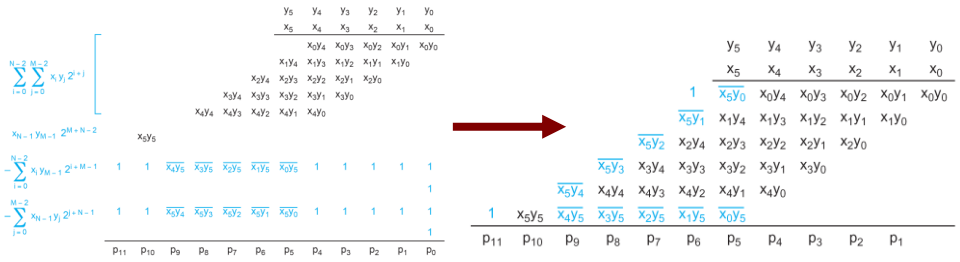


# Πολ/σμός σε συμπλήρωμα ως προς 2 (2/4)




- Το επάνω παραλληλόγραμμο αντιστοιχεί στο μη προσημασμένο πολλαπλασιασμό των δυαδικών ψηφίων, εκτός από τα MSBs
- Η επόμενη γραμμή έχει ένα μόνο ψηφίο, που αντιστοιχεί στο γινόμενο MSBs
- Οι δύο επόμενες γραμμές είναι οι ανεστραμμένοι όροι που πρόκειται να αφαιρεθούν
  - Κάθε όρος έχει μηδενικά που βρίσκονται στην αρχή και στο τέλος του που με την αναστροφή γίνονται μονάδες
- Μια επιπλέον μονάδα προστίθεται LSB για λήψη συμπληρώματος ως προς 2

# Πολ/σμός σε συμπλήρωμα ως προς 2 (3/4)

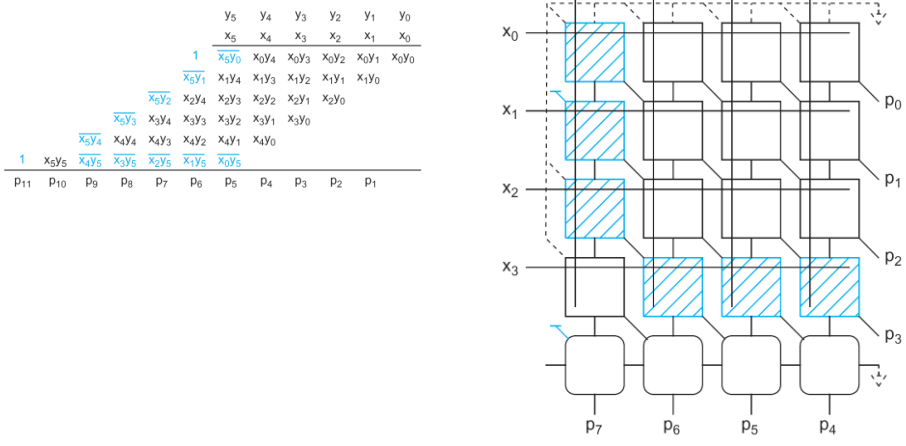


- Η καθυστέρηση εξαρτάται από τον αριθμό των γραμμών των μερικών γινομένων που θα προστεθούν
- Ο τροποποιημένος πολλαπλασιαστής Baugh-Wooley μειώνει τον αριθμό των μερικών γινομένων προ-υπολογίζοντας τα αθροίσματα των σταθερών 1 και μεταθέτοντας κάποιους προς τα πάνω σε επιπλέον στήλες

235



# Πολ/σμός σε συμπλήρωμα ως προς 2 (4/4)



236

# Κωδικοποίηση Booth

- Με τον κλασσικό αλγόριθμο κάθε ψηφίο του πολ/στή παράγει ένα μερικό γινόμενο που πρέπει να προστεθεί =>
  - μεγάλο πλήθος προσθέσεων μερικών γινομένων (για μεγάλους ποσ/στές)
  - αύξηση της καθυστέρησης
- Ο αλγόριθμος του Booth κωδικοποιεί τον πολ/στή 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 & 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 $2^{i+k} - 2^i = 2^{i+k-1} + 2^{i+k-2} + \ldots + 2^{i+1} + 2^i$

Αφαίρεση

| Θέση | | $i+k$ | $i+k-1$ | $i+k-2$ | … | $i+1$ | $i$ | $i-1$ | … | … |
|---|---|---|---|---|---|---|---|---|---|---|
| Τιμή | | 1 | 0 | 0 | 0 | 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 πολ/στέου A

➢Η διαδικασία είναι πιο κατανοητή αν 2 dummy bits $b_n=b_{-1}=0$ εισαχθούν στο $B=b_{n-1}, b_{n-2}, \ldots, b_1, b_0$

# Πίνακας κωδικοποίησης Booth (1/2)

| Multiplier | | |
|---|---|---|
| Bit i | Bit i+1 | Λειτουργία |
| 0 | 0 | 0 x multiplicand (0xA) |
| 0 | 1 | +1 x multiplicand (+1xA) |
| 1 | 0 | -1 x multiplicand (-1xA) |
| 1 | 1 | 0 x multiplicand (0xA) |

- 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)

239

239

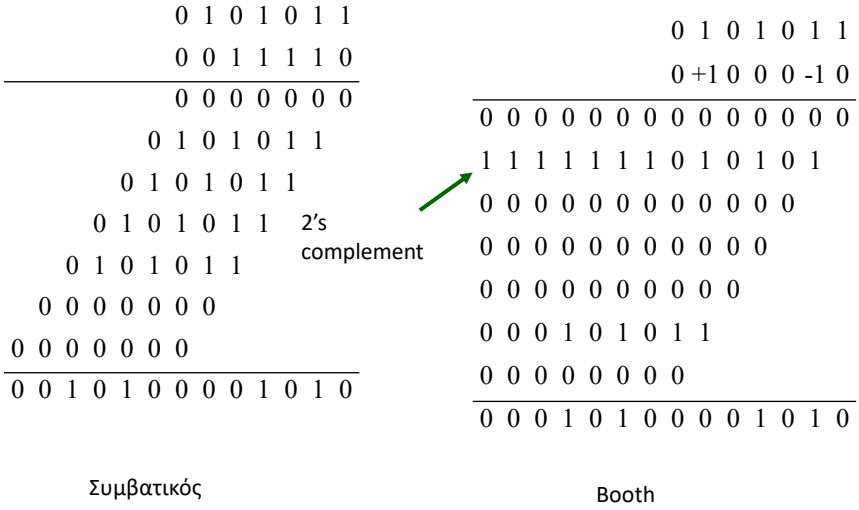
# Πίνακας κωδικοποίησης Booth (2/2)

| 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 |

240

240

# Πολλαπλασιασμός Booth – Παράδειγμα




Συμβατικός

Booth

241

241

# 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 πολ/στή
  - 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 πολ/στή
- Όπως και η προηγούμενη κωδικοποίηση ισχύει για προσημασμένους και μη προσημασμένους αριθμούς

242

242

# Πίνακας κωδικοποίησης Booth radix 4 (2/3)

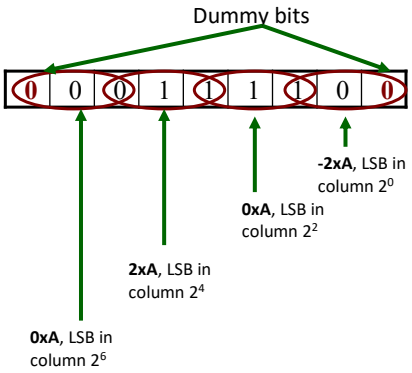
| Ζεύγος ψηφίων πολ/στή | | Ψηφίο δεξιά | Λειτουργία | Εξήγηση |
|---|---|---|---|---|
| $2^1$ | $2^0$ | | | |
| i+1 | i | i-1 | | |
| 0 | 0 | 0 | 0xA | No string |
| 0 | 0 | 1 | +1xA | End of string |
| 0 | 1 | 0 | +1xA | Single 1 (+2-1) |
| 0 | 1 | 1 | +2xA | End of string |
| 1 | 0 | 0 | -2xA | Beginning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





# Κωδικοποίηση Radix-4 (3/3)

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



# Πολλαπλασιασμός Booth – Παράδειγμα

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

```
          0 0 0 0 1 1
          0 1 1 1 0 1
              2xA  -1xA  1xA
-------------------------------
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
```

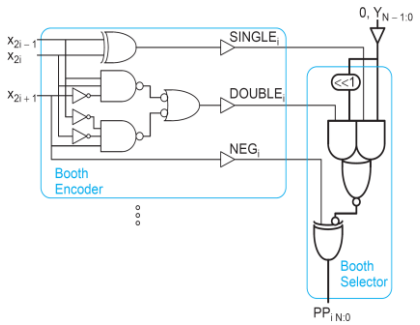
Booth Radix4





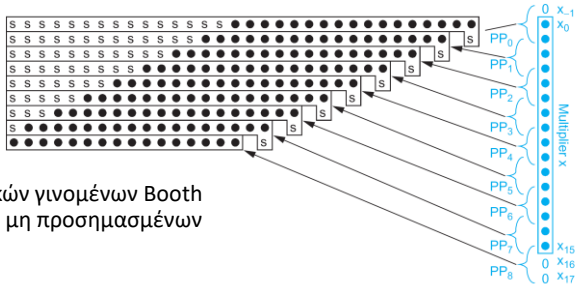
## Κυκλώματα Κωδικοποίησης & Επιλογής Booth

| Inputs | | | Partial Product | Booth Selects | | |
|---|---|---|---|---|---|---|
| $x_{2i+1}$ | $x_{2i}$ | $x_{2i-1}$ | $PP_i$ | $SINGLE_i$ | $DOUBLE_i$ | $NEG_i$ |
| 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 πολ/στεο Y επεκταμένο ως προς το μηδέν σε N+1 bits –έξοδος τιμές 0, Y, 2Y
- Αν το μερικό γινόμενο είναι αρνητικό ( neg=1) χρησι ιι complement

## Επέκταση προσήμου




16-bit πίνακας μερικών γινομένων Booth τάξης-4 για πολ/στή μη προσημασμένων αριθμών

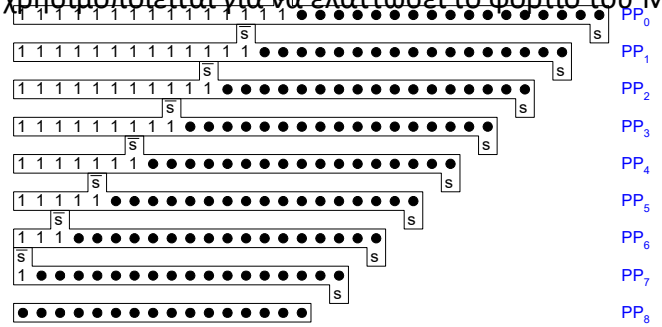
- Ακόμα και σε μη προσημασμένους αριθμούς τα αρνητικά μερικά γινόμενα πρέπει να επεκταθούν ως προς το πρόσημο για να προστεθούν σωστά
- Κάθε μερικό γινόμενο επεκτείνεται ως προς το πρόσημο με βάση το σήμα $neg_i$
- Ένας 1 προστίθεται στο LSB στην επόμενη γραμμή (το συμπλήρ ως προς 2)
- Μεγάλες απαιτήσεις fanout για τα MSBs

247

247

# 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

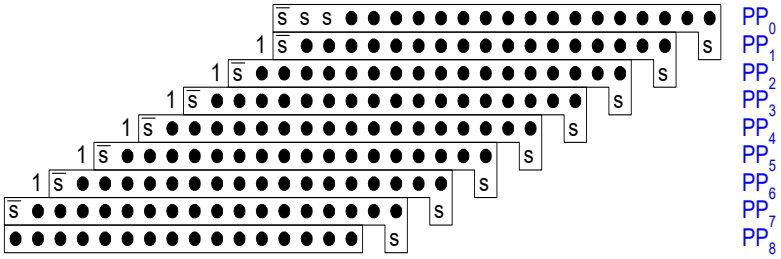



248

248

# Απλοποιημένη επέκταση προσήμου (2/2)

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

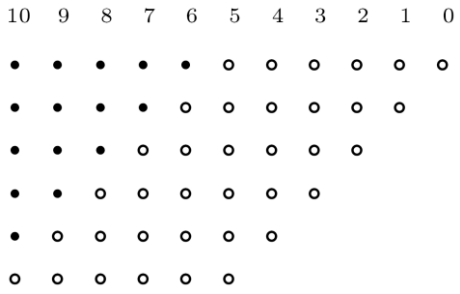



249

249

# 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




250

250

# Μείωση πολυπλοκότητας

- Two's complement αριθμός ssssss $z_4z_3z_2z_1z_0$ 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) $z_4z_3z_2z_1z_0$ αφού

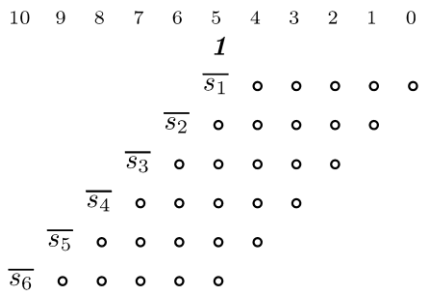
$$-s \cdot 2^{10} + s \cdot (2^9 + 2^8 + 2^7 + 2^6 + 2^5)$$
$$= -s \cdot 2^{10} + s \cdot (2^{10} - 2^5) = \quad -s \cdot 2^5$$

251

251

# Νέος πίνακας ψηφίων

- Για την παραγωγή του -s στη στήλη 5, συμπλήρωμα του αρχικού s σε (1-s) και πρόσθεση 1
  - Κρατούμενο 1 στη στήλη 6 λειτουργεί ως το επιπλέον 1 που χρειάζεται για ψηφίο προσήμου στο δεύτερο μερικό γινόμενο κ.ο.κ.
- Ο νέος πίνακας έχει λιγότερα ψηφία αλλά έχει στήλες με μεγαλύτερο ύψος (7 αντί 6)

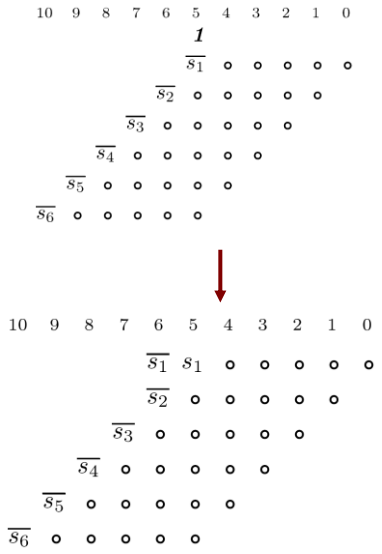



252

252

# Εξάλειψη του επιπλέον 1 στη στήλη 5

- Τοποθέτηση των δύο sign bits $s_1$, $s_2$ στην ίδια στήλη
- $(1-s_1)+(1-s_2) = 2 - s_1 - s_2$
  - Το 2 είναι το carry out για την επόμενη στήλη
- Επιτυγχάνεται επεκτείνοντας αρχικά το sign bit $s_1$
- Το μέγιστο ύψος της στήλης είναι πάλι 6 αντί 7

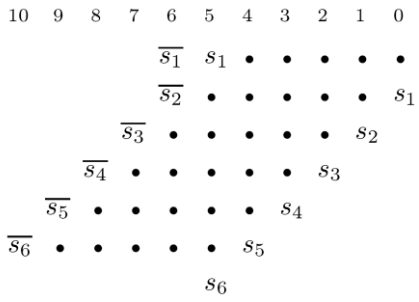



253

253

# Χρήση One's Complement & Carry

- Αν το αρνητικό μερικό γινόμενο προκύπτει από 1's complement +1 τα επιπλέον carries εισέρχονται όπως φαίνεται στον πίνακα
- Οι κύκλοι δηλώνουν ότι τα συμπληρώματα των αντίστοιχων ψηφίων λαμβάνονται όταν si=1
- Το επιπλέον s6 στη στήλη 5 αυξάνει το ύψος σε 7
- Αν το τελευταίο μερικό γινόμενο είναι θετικό (ο πλο/στής είναι θετικός) s6 μπορεί να απαληφθεί




254

254

# Πρόσθεση μερικών γινομένων

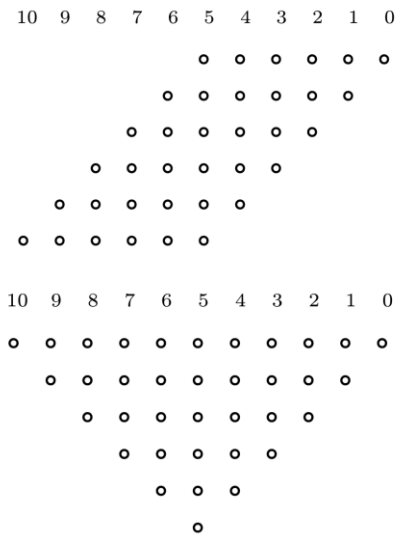
- Τα μερικά γινόμενα πρέπει να προστεθούν για την παραγωγή του τελικού αποτελέσματος
- Χρήση αθροιστών πολλαπλών ορισμάτων
  - Fast multi-operand adder
- Η δομή των μερικών γινομένων πρέπει να ληφθεί υπόψη ώστε να ελαττωθεί η πολυπλοκότητα
- Μερικά μερικά γινόμενα έχουν μικρότερο πλήθος ψηφίων από το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



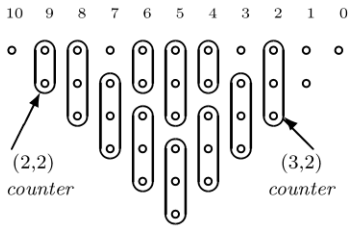


# Παράδειγμα - 6 Partial Products

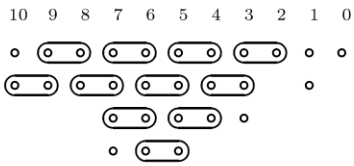
- Παράγονται όταν πολ/νται μη προσημασμένοι 6-bit αριθμοί
- 6 operands μπορούν να προστεθούν χρησιμοποιώντας 3 επίπεδα CSAs (Wallace tree)
- Το πλήθος των (3,2) μετρητών μπορεί να μειωθεί δραστικά εκμεταλλευόμενοι το γεγονός ότι μόνο μια στήλη έχει 6 ψηφία
- Επανασχεδίαση του διαγράμματος κουκίδων για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



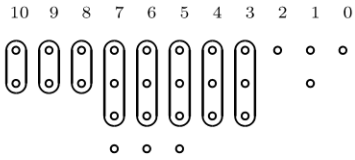
# Μείωση πολυπλοκότητας - Χρήση (2,2) Counters (HAs)



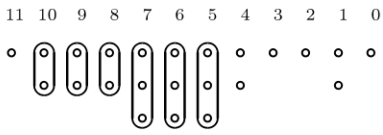

(a) Level 1 carry-save addition.




(b) Results of level 1.




(c) Level 2 carry-save addition.



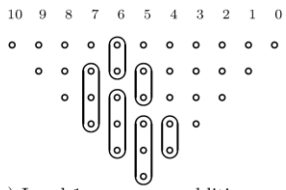

(d) Level 3 carry-save addition.

- Ο αριθμός των επιπέδων παραμένει 3 αλλά λιγότεροι counters

262

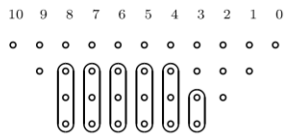
262

# Επιπλέον μείωση του πλήθους των μετρητών

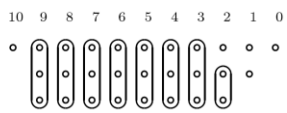



(a) Level 1 carry-save addition.


(b) Results of level 1.




(c) Level 2 carry-save addition.



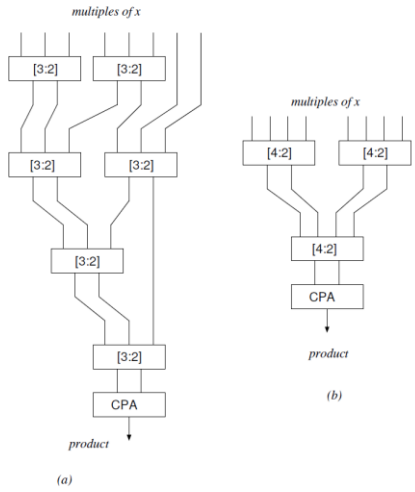

(d) Level 3 carry-save addition.

- 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

263

263

# Reduction by Rows – Adders Trees

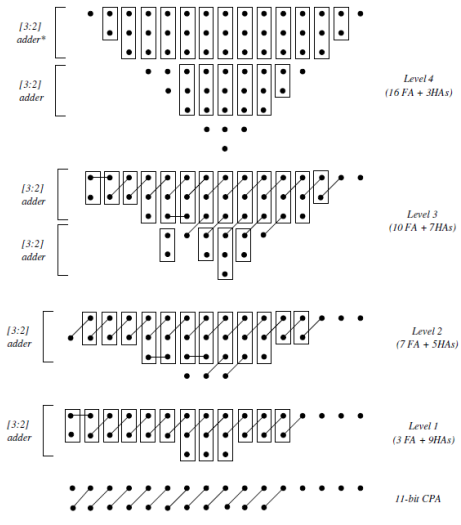



TREE ARRAYS OF ADDERS: a) with [3:2] adders. b) with [4:2] adders





# Reduction by Rows – Adders Trees




REDUCTION BY ROWS USING FAs AND HAs ($n = 8$): Cost 36 FAs, 24 HAs, 11-bit CPA

# Υπολογισμοί παράλληλου προθέματος (1/2)

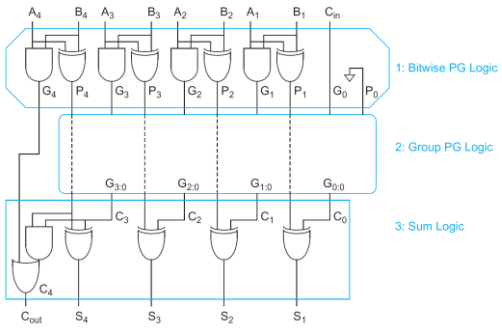
- Πολλές λειτουργίες χειριστών δεδομένων περιλαμβάνουν υπολογισμό μιας ομάδας εξόδων από μια ομάδα εισόδων
  - κάθε ψηφίο εξόδου εξαρτάται από όλα τα προηγούμενα ψηφία εισόδου
- Η πρόσθεση δύο εισόδων N bit είναι ένα κλασικό παράδειγμα
  - Κάθε έξοδος *Y*i εξαρτάται από ένα κρατούμενο εισόδου $c_{i-1}$ από το προηγούμενο δυαδικό ψηφίο, το οποίο με τη σειρά του εξαρτάται από ένα κρατούμενο εισόδου $c_{i-2}$ από το δυαδικό ψηφίο που προηγείται αυτού κοκ
- Με μονάδες πρόβλεψης κρατουμένου αυξανόμενου μεγέθους μπορούμε να κατασκευάσουμε αθροιστές που έχουν log*N* επίπεδα

268

268

# Υπολογισμοί παράλληλου προθέματος (2/2)

- Η πρόσθεση είναι ένας προθεματικός υπολογισμός που περιλαμβάνει
  - Έναν προ-υπολογισμό σε επίπεδο bit,
  - Ένα δένδρο λογικής ομάδας για το σχηματισμό των προθεμάτων και
  - Ένα τελικό επίπεδο εξόδου, το οποίο
- Επέκταση τεχνικής σε άλλους προθεματικούς υπολογισμούς με σχετιζόμενες λογικές λειτουργίες ομάδας




269

269

# Κωδικοποιητής προτεραιότητας (1/4)

- Η συνήθης εφαρμογή είναι να διαιτητεύει ανάμεσα σε N μονάδες που ζητούν πρόσβαση σε ένα κοινό πόρο
- Κάθε μονάδα *i* στέλνει ένα δυαδικό ψηφίο $A_i$ (δηλώνει αίτηση) και λαμβάνει ένα δυαδικό ψηφίο $Y_i$, (δηλώνει του δόθηκε πρόσβαση)
- Σε πολλαπλή αίτηση ακολουθείται σειρά προτεραιότητας
- Αν το λιγότερο σημαντικό δυαδικό ψηφίο της εισόδου αντιστοιχεί στη υψηλότερη προτεραιότητα, η λογική μπορεί να εκφρασθεί ως εξής

$$Y_1 = A_1$$
$$Y_2 = A_2 \cdot \overline{A_1}$$
$$Y_3 = A_3 \cdot \overline{A_2} \cdot \overline{A_1}$$
$$\ldots$$
$$Y_N = A_N \cdot \overline{A_{N-1}} \cdot \ldots \cdot \overline{A_1}$$





# Κωδικοποιητής προτεραιότητας (2/4)

- Μπορούμε να εκφράσουμε την κωδικοποίηση προτεραιότητας ως μια λειτουργία προθέματος, καθορίζοντας ένα πρόθεμα *X*i:j που δηλώνει ότι καμία από τις εισόδους *A*i…*A*j δεν έχουν τεθε.
- Η η κωδικοποίηση προτεραιότητας καθορίζεται με προ-υπολογισμούς σε επίπεδο ψηφίου, λογικής ομάδας και λογικής εξόδου με $i \geq k \geq j$ :

$$X_{i:i} = \overline{A_i}$$
$$X_{i:j} = X_{i:k} \cdot X_{k-1:j}$$
$$Y_i = A_i \cdot X_{i-1:1}$$

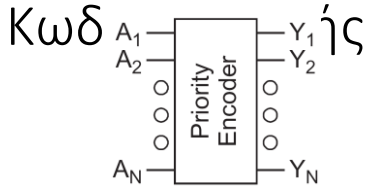
# Κωδικοποιητής προτεραιότητας (3/4)

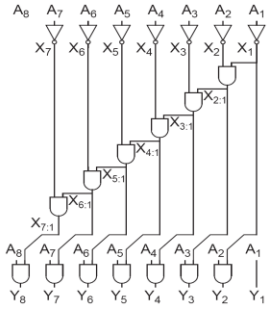
- Οποιοδήποτε από τα δίκτυα ομάδας (π.χ, κύματος, παράκαμψης, πρόβλεψης, επιλογής, αύξησης, δένδρου) της πρόσθεσης μπορεί να χρησιμοποιηθεί για την υλοποίηση της λογικής ομάδας υπολογισμού του προθέματος *X*i:0
- Οι κωδικ. μικρού μήκους κωδικοποιητές χρησιμοποιούν δομή κύματος, οι μεσαίου μήκους μπορούν να χρησιμοποιούν μια δομή παράκαμψης, πρόβλεψης, επιλογής ή αύξησης
- Οι μεγάλου μήκους κωδικοποιητές χρησιμοποιούν δένδρα για να πετυχαίνουν καθυστέρηση log*N*

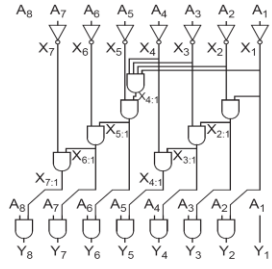


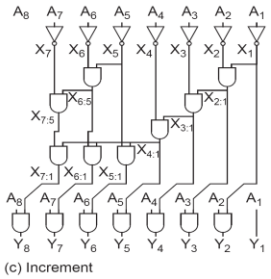


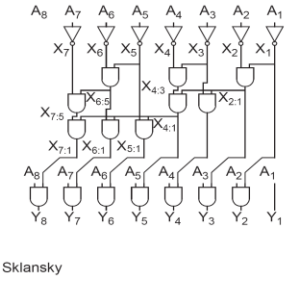
# Κωδικοποιητής





(a) Ripple


(b) Lookahead


(c) Increment


(d) Sklansky