

Τμήμα ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΑΣ

MAY 30, 2018

Αναδρομική Ling Αρχιτεκτονική
για αθροιστές υπολοίπου $2^n - 1$

Author

Βάσιλας Κωνσταντίνος

Supervisor

Βέργος Χαρίδημος

Περιεχόμενα

| | |
|--|----|
| List of Figures | 4 |
| List of Tables | 4 |
| 1 Εισαγωγή | 5 |
| 2 Πρόσθεση στο Δυαδικό | 6 |
| 2.1 Πρόσθεση δυαδικών αριθμών | 8 |
| 3 Οικογένειες Αθροιστών | 9 |
| 3.1 Διάδοσης Κρατουμένου | 9 |
| 3.2 Παράλειψής Κρατουμένου | 9 |
| 3.3 Επιλογής κρατουμένου | 10 |
| 3.4 Πρόβλεψης Κρατουμένου | 10 |
| 4 Προθεματική Αθροιστές | 12 |
| 4.1 Πρόβλημα προθέματος | 12 |
| 4.2 Παράλληλοι Προθεματικοί Αθροιστές | 15 |
| 4.2.1 Brent-Kung Adder | 15 |
| 4.2.2 Ladner-Fischer Adder | 15 |
| 4.3 Δέντρα-Δομές Προθεμάτων | 15 |
| 4.4 Αραίωση των Δέντρων | 15 |
| 4.5 Προθεματικός αθροιστής με κρατούμενο εισόδου | 16 |
| 5 Ling Αθροιστές | 18 |
| 5.1 Βασική Θεωρία | 18 |
| 5.2 Πλεονεκτήματα της Ling παραγοντοποίησης | 19 |
| 5.3 Αραίωση σε Ling Αρχιτεκτονικές | 21 |
| 6 Παραγοντοποίηση Jackson | 22 |
| 6.1 Βασικοί Όροι | 22 |
| 6.2 Αναδρομή του Jackson | 23 |
| 6.3 Εφαρμογές της αναδρομής | 25 |
| 6.4 Πρακτικές εφαρμογές | 25 |
| 6.5 Παράδειγμα υλοποίησης | 26 |
| 6.6 Αραίωση σε Jackson Αρχιτεκτονικές | 27 |
| 7 Αθροιστής υπολοίπου $2^n - 1$ | 28 |
| 7.1 Basic Operation | 28 |
| 7.2 Prefix αθροιστές $2^n - 1$ | 29 |
| 7.3 Architectures Improvements | 31 |
| 8 Ανάπτυξη Αθροιστών υπολοίπου $2^n - 1$ | 34 |

ΠΕΡΙΕΧΟΜΕΝΑ

| | | |
|----------|--|-----------|
| 8.1 | Prefix $2^n - 1$ | 34 |
| 8.2 | Ling $2^n - 1$ | 34 |
| 8.3 | Jackson $2^n - 1$ | 34 |
| 8.3.1 | $2^8 - 1$ | 34 |
| 9 | Αποτελέσματα | 36 |
| 9.1 | Μη-Τοπογραφικές Μετρήσεις | 36 |
| 9.2 | Μη-Τοπογραφικές Μετρήσεις sparse-2 | 36 |
| 9.3 | Μη-Τοπογραφικές Μετρήσεις sparse-4 | 37 |
| 9.4 | Τοπογραφικές Μετρήσεις | 37 |
| 9.5 | Γραφικές | 38 |
| | Βιβλιογραφία | 39 |

List of Figures

| | | |
|-----|---|----|
| 2.1 | Half-Adder schematic | 6 |
| 2.2 | Full-Adder schematic | 7 |
| 2.3 | Integer-Adder schematic | 8 |
| 4.1 | Serial-Prefix Tree Adder | 14 |
| 4.2 | Ladner-Fischer Prefix Tree Adder | 15 |
| 5.1 | Λογική υπολογισμού του αθροίσματος κατά Ling | 20 |
| 7.1 | Απλή δομή προθεματικού αθροιστή υπολοίπου $2^8 - 1$ | 30 |
| 7.2 | Παράλληλος προθεματικός αθροιστής υπολοίπου $2^8 - 1$ | 33 |
| 8.1 | Jackson 8-bit $2^n - 1$ Adder | 34 |

List of Tables

| | | |
|------|---|----|
| 2.1 | Half Adder Truth Table | 6 |
| 2.2 | Full Adder Truth Table | 7 |
| 6.1 | Σύγκριση προθεματικού αθροιστή Jackson | 26 |
| 9.1 | Μετρήσεις 8-bit | 36 |
| 9.2 | Μετρήσεις 16-bit | 36 |
| 9.3 | Μετρήσεις 32-bit | 36 |
| 9.4 | Μετρήσεις 64-bit | 36 |
| 9.5 | Μη-Τοπογραφικές sparse-2 Μετρήσεις 8-bit | 36 |
| 9.6 | Μη-Τοπογραφικές sparse-2 Μετρήσεις 16-bit | 36 |
| 9.7 | Μη-Τοπογραφικές sparse-2 Μετρήσεις 32-bit | 36 |
| 9.8 | Μη-Τοπογραφικές sparse-2 Μετρήσεις 64-bit | 36 |
| 9.9 | Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit | 37 |
| 9.10 | Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit | 37 |
| 9.11 | Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit | 37 |
| 9.12 | Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit | 37 |
| 9.13 | Τοπογραφικές Μετρήσεις 8-bit | 37 |
| 9.14 | Τοπογραφικές Μετρήσεις 16-bit | 37 |
| 9.15 | Τοπογραφικές Μετρήσεις 32-bit | 37 |
| 9.16 | Τοπογραφικές Μετρήσεις 64-bit | 37 |

List of Equations

| | | |
|-----|--|----|
| 4.4 | Αλγεβρικός ορισμός των (G_i, P_i) | 12 |
| 4.5 | Αλγεβρικός ορισμός των (G'_i, P'_i) | 16 |
| 4.6 | Ανάκτηση των (G'_i, P'_i) από (G_i, P_i) | 16 |
| 5.6 | άθροισμα κατά Ling | 19 |
| 5.7 | άθροισμα κατά Ling (2) | 20 |
| 7.1 | Ορισμός άθροισης υπολοίπου $2^n - 1$ | 28 |
| 7.3 | Εξίσωση κρατούμένων του $2^n - 1$ αθροιστή | 31 |

1 Εισαγωγή

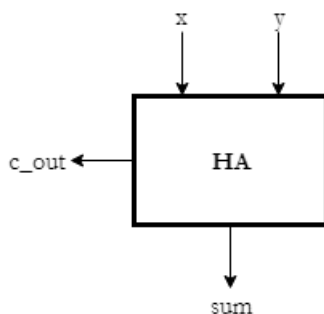
Οι αριθμητικές μονάδες είναι βασικά στοιχεία στο σύνολο του υλικού των ηλεκτρονικών υπολογιστών . Οι περισσότερες επεξεργαστικές μονάδες έχουν ως βασικό δομικό στοιχείο την Αριθμητική Λογική Μονάδα (ΑΛΜ) ή Arithmetic Logic Unit (ALU) η οποία είναι υπεύθυνη για την υλοποίηση βασικών αριθμητικών πράξεων όπως πρόσθεση και πολλαπλασιασμό αλλά και λογικών πράξεων όπως OR, AND και XOR . Αυτές οι μονάδες είναι κρίσιμες στην ορθή λειτουργία ενός συστήματος όχι μόνο ως προς την λειτουργικότητα αλλά και ως προς την ταχύτητα , κατανάλωση ενέργειας και χώρου. Στην παρούσα διπλωματική εργασία θα αναλυθούν οι αθροιστές και συγκεκριμένα μια υποομάδα αυτών , οι $2n-1$ αθροιστές . Η ομάδα αυτών των αθροιστών είναι χρήσιμη στα Συστήματα Αριθμού Υπολειμμάτων ή Residue Number Systems (RNS) , Συστήματα Ανθεκτικά σε Σφάλματα ή Fault-Tolerant Systems , στην ανίχνευση σφάλματος στα συστήματα δικτύου καθώς και στις αριθμητικές πράξεις κινητής υποδιαστολής [1] [2] [3].

2 Πρόσθεση στο Δυαδικό

Η πρόσθεση δυο δυαδικών ψηφίων x , y έχει αποτέλεσμα το άθροισμα των δυο αυτών ψηφίων sum καθώς και το κρατούμενο εξόδου c_{out} , όπως ακριβώς ισχύει και με την άθροιση στο δεκαδικό σύστημα, και ορίζεται ως :

| x | y | sum | c_{out} |
|-----|-----|-------|-----------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Πίνακας 2.1: Half Adder Truth Table



Εικόνα 2.1: Half-Adder schematic

Το παραπάνω σύστημα ονομάζεται ημιαθροιστής ή Half-Adder (HA), έχει δυο εισόδους τις x και y και δυο εξόδους τις c_{out} και sum και οι έξοδοι σύμφωνα με τον παραπάνω πίνακα αληθείας περιγράφονται από τις παρακάτω συναρτήσεις άλγεβρας Μπουλ :

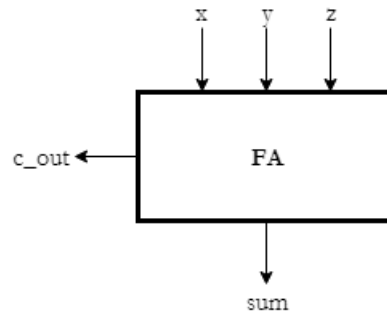
$$\begin{aligned} sum &= x \oplus y \\ c_{out} &= x * y \end{aligned} \tag{2.1}$$

Από τον ημιαθροιστής δομείται ο πλήρης αθροιστής ή Full-Adder (FA) με τρεις εισόδους x , y , z , δυο εξόδους sum και c_{out} και λειτουργία αντίστοιχη του FA με την διαφορά πως ο FA προσθέτει τρία δυαδικά ψηφιά

και οι εξισώσεις των εξόδων είναι :

| x | y | z | sum | c_out |
|---|---|---|-----|-------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Πίνακας 2.2: Full Adder Truth Table

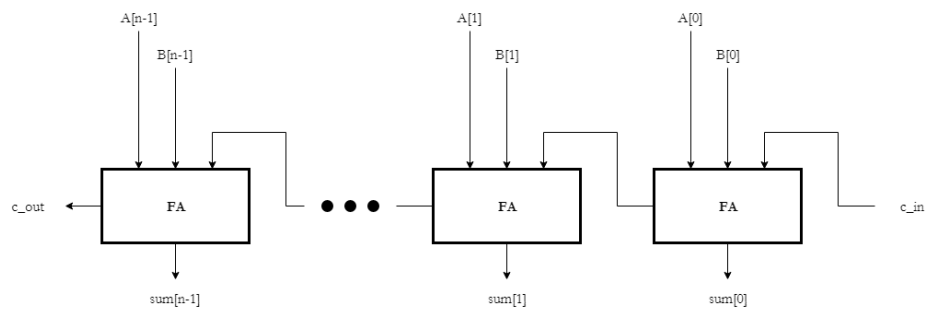


Εικόνα 2.2: Full-Adder schematic

$$\begin{aligned}
 sum &= x \oplus y \oplus z \\
 c_{out} &= (x * y) + (x * z) + (z * y)
 \end{aligned}
 \tag{2.2}$$

2.1 Πρόσθεση δυαδικών αριθμών

Η πρόσθεση δυο δυαδικών αριθμών A και B των n δυαδικών ψηφίων το κάθε ένα είναι μια επέκταση της πρόσθεσης μεταξύ ψηφίων που παρουσιάστηκε προηγουμένως τροφοδοτώντας το κρατούμενο εισόδου των προηγούμενων σημαντικών ψηφίων στην είσοδο του πλήρη αθροιστή των επομένων .



Εικόνα 2.3: Integer-Adder schematic

3 Οικογένειες Αθροιστών

Σε αυτή την ενότητα θα παρουσιαστούν κάποιες βασικές ομάδες αθροιστών με βάση την αρχιτεκτονική τους.

3.1 Διάδοσης Κρατουμένου

Αθροιστής Διάδοσης Κρατουμένου ή Ripple-Carry Adder (RCA) είναι ο πιο απλός αθροιστής και το σχηματικό του είναι στην εικόνα 2.3. Παρατηρούμε πως για να υπολογιστεί το $sum[1]$ δηλαδή το δεύτερο λιγότερο σημαντικό ψηφίο του αθροίσματος πρέπει πρώτα να υπολογιστεί το c_out του προηγούμενου πλήρη αθροιστή, αντίστοιχα για το τρίτο πρέπει να υπολογιστεί πρώτα το $c_out[0]$ έπειτα από τον δεύτερο πλήρη αθροιστή να υπολογιστεί το $c_out[1]$ με είσοδο στο c_in το $c_out[0]$ δηλαδή το c_out του προηγούμενου. Συμπεραίνουμε λοιπόν πως κάθε FA δεν λειτουργεί παράλληλα με τα υπόλοιπα αλλά υπάρχει μια χρονική καθυστέρηση για να λάβει την σωστή είσοδο c_in . Αποτέλεσμα αυτού είναι πως στην χειρότερη περίπτωση, δηλαδή στην περίπτωση που το c_in επηρεάζει άμεσα την τιμή του c_out του αθροιστή, υπάρχει γραμμική αύξηση της καθυστέρησης με το μήκος του. Για παράδειγμα έχοντας έναν δυαδικό αθροιστή 8 ψηφίων και εισάγοντας τους αριθμούς $A = 00000000$ και $B = 11111111$ και $c_in = 0$ θα πάρουμε έξοδο $sum = 11111111$ και $c_out = 0$. Αν κρατώντας όλες τις εισόδους σταθερές και αλλάζοντας μόνο την A σε 00000001 δηλαδή το $A[0]=1$ τότε το αποτέλεσμα θα είναι $sum=00000000$ και $c_out = 1$.

Η περίπτωση που περιγράφηκε ανήκει στις χειρότερες περιπτώσεις διότι

3.2 Παράλειψής Κρατουμένου

Για κάθε ψηφίο του αριθμού A και B ορίζονται δυο ακόμα στοιχεία, αυτά που παράγουν κρατούμενο ανεξάρτητα του κρατουμένου εισόδου και θα καλούνται generate και αυτά που διαδίδουν κρατούμενο και ονομάζουμε propagate. Οι εξισώσεις είναι αντίστοιχα:

$$\begin{aligned} g_i &= A_i * B_i \\ p_i &= A_i \oplus B_i \end{aligned} \tag{3.1}$$

ή

$$p_i = A_i + B_i$$

Όταν έχουμε $g_i = 1$ τότε γνωρίζουμε πως ο συγκεκριμένος πλήρης αθροιστής παράγει κρατούμενο εξόδου. Σε αντίθεση με την συνάρτηση εξόδου του c_out του Full-Adder η συνάρτηση του generate είναι πιο απλή καθώς αποτελείται από μόνο μια πύλη AND άλλα δεν εγγυάται αν θα υπάρχει κρατούμενο εισόδου ή όχι. Δηλαδή αν $g_i = 1$ τότε και $c_out_i = 1$ χωρίς όμως να ισχύει το αντίθετο.

Είναι μια υλοποίηση αθροιστή που βελτιώνει την καθυστέρηση του αθροιστή διάδοσης κρατουμένου με έναν απλό τρόπο αλλά όχι αρκετά αποτελεσματικό

σε σχέση με άλλες αρχιτεκτονικές. Η χειρίστη περίπτωση παρουσιάζεται όταν σε έναν ripple-carry όταν το propagate στοιχείο είναι αληθές για κάθε ζευγάρι ψηφίων ($A_i \oplus B_i$). Ορίζοντας τον propagate όρο ως το XOR των $A_i \oplus B_i$ Όταν όλοι οι όροι propagate είναι αληθείς τότε το κρατούμενο εισόδου προσδιορίζει το κρατούμενο εξόδου. Παίρνοντας κάθε όρο propagate και εισάγοντας τον σε μια n-εισόδων πύλη AND ορίζουμε τον όρο select το οποίο οδηγεί την είσοδο επιλογής ενός πολυπλέκτη 2-σε-1, όπου η έξοδος του είναι το c_{out} του αθροιστή, και όταν το select είναι εληθες τότε επιλέγεται το c_{in} αλλιώς το c_n .

$$\begin{aligned} p_i &= A_i \oplus B_i \\ select &= p_0 * p_1 * \dots * p_{n-1} \\ c_{out} &= select ? c_{in} : c_{n-1} \end{aligned} \quad (3.2)$$

Η βελτιστοποίηση της χειρίστης περίπτωσης επιτυγχάνεται με την χρήση πολλαπλών αθροιστών παράβλεψης κρατουμένου για να δομήσουν έναν block-carry-skip αθροιστή. Στην αντίθετη περίπτωση η καθυστέρησης είναι ίδιες με αυτές του ripple-carry. Ο αριθμός των εισόδων της πύλης AND για τον υπολογισμό του select είναι ίσος με τον μήκος του αθροιστή με αποτέλεσμα ένας αθροιστής μεγάλου μήκους να καθίσταται μη πρακτικός εφόσον οδηγεί σε επιπλέον καθυστερήσεις, διότι η πύλη AND πρέπει να κατασκευαστεί σαν ένα δέντρο.

3.3 Επιλογής κρατουμένου

Ο αθροιστής επιλογής κρατουμένου ή Carry-Select Adder υλοποιείται με τον εξής τρόπο :

- Έχουμε δυο αθροιστές ιδίου μήκους και εκτελούν τις ίδιες προσθέσεις με την διαφορά πως ο ένας έχει ως κρατούμενο εισόδου 0 και ο άλλος 1.
- Το κρατούμενο εισόδου του αθροιστή οδηγεί την είσοδο επιλογής ενός πολυπλέκτη με είσοδο τα αποτελέσματα των δυο αθροιστών που προαναφέρθηκαν.
- Ανάλογα με την κατάσταση του κρατουμένου εισόδου επιλέγεται και το σωστό αποτέλεσμα στην έξοδο.

3.4 Πρόβλεψης Κρατουμένου

Σε αντίθεση με τις προηγούμενες ταχτικές βελτίωσης της άθροισης ο Αθροιστής Πρόβλεψης Κρατουμένου ή Carry-Lookahead Adder (CLA) βελτιώνει την ταχύτητα μειώνοντας τον χρόνο υπολογισμού κάθε ενδιαμέσου κρατουμένου καθώς και του τελικού c_{out} . Αυτή η αρχιτεκτονική υλοποιείται με τον παρακάτω τρόπο :

- Υπολογίζονται, για κάθε ζεύγος (A_i, B_i) δύο σήματα, το ένα αληθεύει όταν το ζεύγος μπορεί να διαδώσει το κρατούμενο που εξάγει το προηγούμενο ζεύγος και το άλλο αληθεύει όταν το παρόν ζευγάρι παράγει κρατούμενο ανεξαρτήτως το αν θα έχει κρατούμενο εισόδου ή όχι. Τα σήματα αυτά θα ονομαστούν propagate ή p και generate ή g, αντίστοιχα.
- Συνδυάζοντας αυτά τα σήματα δίνεται η δυνατότητα να προσδιοριστεί ταχύτερα αν ένα τμήμα ζευγών ψηφίων πρόκειται να διαδώσει ή να παράξει ένα κρατούμενο .

Για παράδειγμα σε έναν αθροιστή των τεσσάρων bits η διάδοση του κρατουμένου εισόδου από το πρώτο έως το τελευταίο bit εξαρτάται από την ομάδα διάδοσης κρατουμένου ή Group Propagate (P). Επίσης η παραγωγή κρατουμένου εξαρτάται από την ομάδα παραγωγής κρατουμένου ή Group Generate (G) των τεσσάρων ζευγών. Για το P είναι εύκολο να βρούμε την συνάρτηση bool του εφόσον το έχουμε συναντήσει και στις παραπάνω ομάδες αθροιστών , αντίθετα το G είναι πιο περίπλοκο.

4 Προθεματική Αθροιστές

Στην ενότητα αυτή θα παρουσιαστεί μια νέα υλοποίηση αθροιστών, οι αθροιστές προθέματος, ο οποίοι έλαβαν σημαντικό ρόλο στην επιτάχυνση καθώς και μείωση του συνολικού εμβαδού των αθροιστών.

4.1 Πρόβλημα προθέματος

Ένα prefix problem ή πρόβλημα προθέματος ορίζεται από n εξόδους y ($y_{n-1}, y_{n-2}, \dots, y_0$) , n εισόδους x ($x_{n-1}, x_{n-1}, \dots, x_0$) και τον τελεστή \otimes . Κάθε έξοδος y υπολογίζεται με τον παρακάτω τρόπο :

$$\begin{aligned} y_0 &= x_0 \\ y_1 &= x_1 \otimes x_0 \\ y_2 &= x_2 \otimes x_1 \otimes x_0 \\ &\dots \\ y_{n-1} &= x_{n-1} \otimes x_{n-2} \otimes \dots \otimes x_1 \otimes x_0 \end{aligned} \tag{4.1}$$

Επίσης μπορούμε να το εκφράσουμε και αναδρομικά :

$$\begin{aligned} y_0 &= x_0 \\ y_i &= x_i \otimes y_{i-1} \end{aligned} \tag{4.2}$$

Ένα απλό παράδειγμα προβλημάτων που αντιμετωπίζονται ως προβλήματα προθέματος είναι η πρόσθεση πολλών αριθμών. Έστω πως έχουμε ένα σύνολο μεγέθους n από αριθμούς $(x_{n-1}, x_{n-2}, \dots, x_1, x_0)$, σύμφωνα με τον ορισμό που δόθηκε παραπάνω χρειαζόμαστε ένα ακόμα σύνολο ίδιου μεγέθους $(y_{n-1}, y_{n-2}, \dots, y_1, y_0)$ όπου κάθε στοιχείο του συνόλου αυτού υπολογίζεται αναδρομικά

$$\begin{aligned} y_0 &= x_0 \\ y_i &= x_i + y_{i-1} \end{aligned}$$

και το τελικό αποτέλεσμα είναι καταχωρημένο στο $y_n - 1$.

Το πρόβλημα υπολογισμού κρατούμενου μπορούμε να το μετατρέψουμε σε prefix problem δημιουργώντας το ζεύγος (G, P) και αναθέτοντας στον τελεστή \otimes την παρακάτω λειτουργία :

$$\begin{aligned} (g_i, p_i) \otimes (g_k, p_k) &= (g_i + p_i g_k, p_i p_k) \\ (G_i, P_i) \otimes (G_k, P_k) &= (G_i + P_i G_k, P_i P_k) \end{aligned} \tag{4.3}$$

Με αυτόν τον τρόπο μπορούμε να υπολογίσουμε κάθε ενδιάμεσο κρατούμενο c_i καθώς και το κρατούμενο εξόδου c_n για έναν αθροιστή των n -bits όπου $c_i = G_i$ και για $n \geq i \geq 0$ έχουμε

$$\begin{aligned} (G_0, P_0) &= (g_0, p_0) \\ (G_i, P_i) &= (g_i, p_i) \otimes (G_{i-1}, P_{i-1}) \end{aligned} \tag{4.4}$$

Η απόδειξη : Εφόσον δεν υπάρχει κρατούμενο εισόδου ($c_{in} = c_{-1} = 0$) έχουμε

$$c_0 = g_0 + p_0 c_{-1}$$

$$c_0 = g_0$$

$$c_0 = G_0$$

έστι το αποτέλεσμα ισχύει για $i - 1$

Αν $i > 0$ και $c_{i-1} = G_{i-1}$ τότε

$$(G_i, P_i) = (g_i, p_i) \otimes (G_{i-1}, P_{i-1})$$

$$= (g_i, p_i) \otimes (c_{i-1}, P_{i-1})$$

$$= (g_i + p_i c_{i-1}, p_i P_{i-1})$$

$$G_i = g_i + p_i c_{i-1}$$

$$G_i = c_i$$

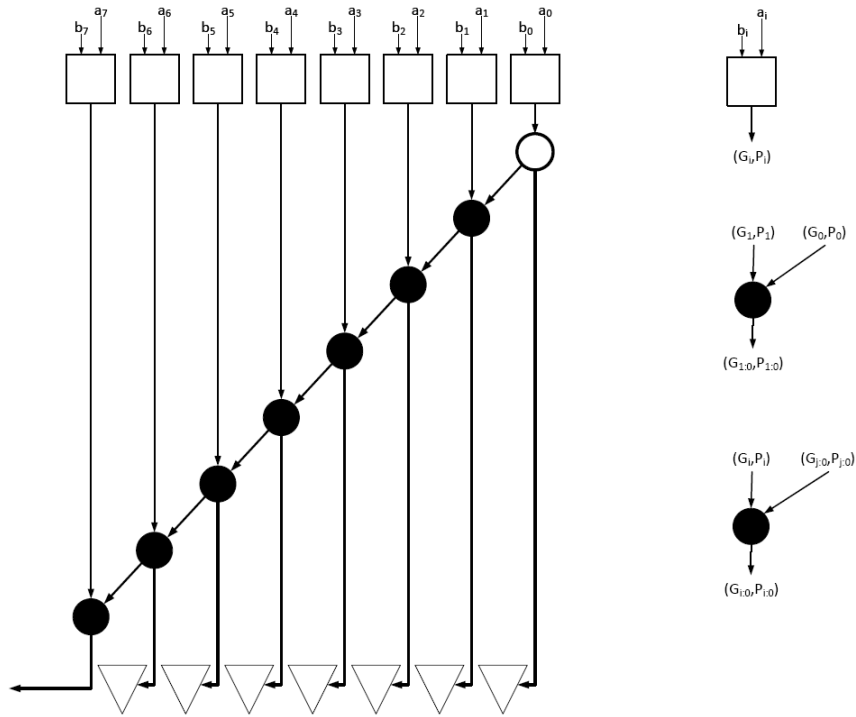
Επίσης ο τελεστής \otimes έχει προσεταιριστική ιδιότητα

$$(g_i, p_i) \otimes (g_j, p_j) \otimes (g_k, p_k) = [g_i + p_i g_j, p_i p_j] \otimes (g_k, p_k)$$

$$= (g_i, p_i) \otimes [g_j + p_j g_k, p_j p_k]$$

$$= (g_i + p_i g_j + p_i p_j g_k, p_i p_j p_k)$$

Παρακάτω παρουσιάζεται ένα γράφημα-δέντρο (Εικόνα 4.1) ενός απλού διαδότης κρατουμένου αθροιστή σε αναγόμενο σε πρόβλημα προθέματος.



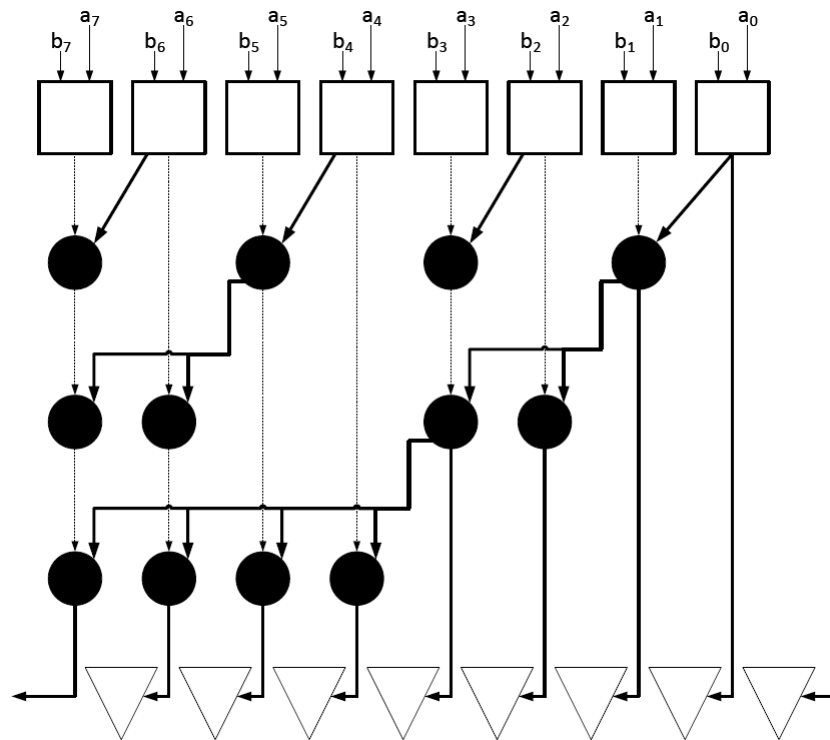
Εικόνα 4.1: Serial-Prefix Tree Adder

Σε κάθε μαύρο κόμβο ουσιαστικά υλοποιείται η λογική συνάρτηση του τελεστή \otimes που παρουσιάστηκε προηγουμένως. Ο παραπάνω αθροιστής υλοποιεί τον προθεματικό αλγόριθμο σειριακά με αποτέλεσμα να είναι πολύ αργό το μοντέλο αλλά να καταλαμβάνει μικρότερο εμβαδόν από άλλες τοπολογίες αθροιστών προθέματος που θα παρουσιαστούν στην συνέχεια.

4.2 Παράλληλοι Προθεματικοί Αθροιστές

4.2.1 Brent-Kung Adder

4.2.2 Ladner-Fischer Adder



Εικόνα 4.2: Ladner-Fischer Prefix Tree Adder

4.3 Δέντρα-Δομές Προθεμάτων

R.P. Brent and H.T. Kung, A Regular Layout for Parallel Adders,^o IEEE Trans. Computers, vol. 31, no. 3, pp. 260-264, Mar. 1982.

P.M. Kogge and H.S. Stone, A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, IEEE Trans. Computers, vol. 22, no. 8, pp. 783-791, Aug. 1973.

J. Sklansky, Conditional Sum Addition Logic, IRE Trans. Electronic Computers, vol. 9, no. 6, pp. 226-231, June 1960.

4.4 Αραίωση των Δέντρων

Οι δομές που περιγράφηκαν για την παραλληλοποίηση του υπολογισμού των κρατουμένων είναι αρκετά πυκνά όσο αφορά τους μαύρους κόμβους, το οποίο

συνεπάγει το ότι απαιτείται αρκετή ενέργεια και εμβαδόν. Με σκοπό την αραιώση (sparseness), είναι δυνατό να υπολογίζονται λιγότερα κρατούμενα από τα ενδιάμεσα στάδια, δηλαδή το δέντρο, και υπολογίζοντας τα υπόλοιπα με διάδοση.

Για παράδειγμα υπολογίζοντας το κρατούμενο που παράγεται από το τρίτο έως το μηδενικό δυαδικό ψηφίο, δηλαδή το $G_{3:0}$ και εισάγωντας στο κρατούμενο εισόδου ενός αθροιστή επιλογής κρατουμένου (CSA) των τεσσάρων δυαδικών ψηφίων με είσοδο τα δυαδικά ψηφία 7 : 0 [4].

Sparsness-2

$$\begin{aligned} sum_i &= x_i \oplus G_{i-1:0} \\ sum_{i+1} &= x_{i+1} \oplus G_{i:0} \\ &= x_{i+1} \oplus (g_i + p_i * G_{i-1:0}) \end{aligned}$$

Sparsness-4

$$\begin{aligned} sum_i &= x_i \oplus G_{i-1:0} \\ sum_{i+1} &= x_{i+1} \oplus G_{i:0} \\ &= x_{i+1} \oplus (g_i + p_i * G_{i-1:0}) \\ sum_{i+2} &= x_{i+2} \oplus G_{i+1:0} \\ &= x_{i+2} \oplus (g_{i+1} + p_{i+1}g_i + p_{i+1}p_i G_{i-1:0}) \\ sum_{i+3} &= x_{i+3} \oplus G_{i+2:0} \\ &= x_{i+3} \oplus (g_{i+2} + p_{i+2}g_{i+1} + p_{i+2}p_{i+1}g_i + p_{i+2}p_{i+1}p_i G_{i-1:0}) \end{aligned}$$

4.5 Προθεματικός αθροιστής με κρατούμενο εισόδου

Οι αθροιστές προθέματος που παρουσιάστηκαν παραπάνω αναπτύχθηκαν με την αρχική υπόθεση έλλειψης κρατουμένου εισόδου ή $c_{in} = 0$. Ενσωματώνοντας στους παραπάνω αθροιστές την επιλογή κρατουμένου εισόδου c_{in} , το οποίο ταυτίζεται με το σήμα c_{-1} , ορίζεται το αντίστοιχο ζεύγος σημάτων (G'_i, P'_i) , όπου είναι το Group Generate και Group Propagate αντίστοιχα με την διαφορά πως συμπεριλαμβάνουν και το κρατούμενο εισόδου. Ακολουθεί ο αλγεβρικός ορισμός των σημάτων G'_i και P'_i (εξίσωση 4.5)

$$(G'_i, P'_i) = \begin{cases} (g_0 + p_0 * c_{-1}, p_0), & i = 0 \\ (g_i, p_i) \oplus (G'_{i-1}, P'_{i-1}), & 1 \leq i \leq n-1 \end{cases} \quad (4.5)$$

Επίσης, στην περίπτωση που συνυπολογίζεται και το κρατούμενο εισόδου, προφανώς ισχύει και $c_i = G'_i$. Για την ανάκτηση των (G'_i, P'_i) χρησιμοποιείται ο παρακάτω τύπος:

$$(G'_i, P'_i) = (G_i + P_i * c_{-1}, P_i) \quad (4.6)$$

το οποίο αποδεικνύεται επίσης επαγωγικά στο i . Για $i = 0$ ισχύει :

$$(G'_0, P'_0) = (g_0 + p_0 * c_{-1}, p_0) = (G_0 + P_0 * c_{-1}, P_0)$$

Υποθέτοντας πως η σχέση ισχύει και για $i = k - 1$, δηλαδή

$$(G'_{k-1}, P'_{k-1}) = (G_{k-1} + P_{k-1} * c_{-1}, P_{k-1})$$

Τότε για $i = k$:

$$\begin{aligned}(G'_k, P'_k) &= (g_k, p_k) \otimes (G'_{k-1}, P'_{k-1}) \\ &= (g_k, p_k) \otimes (G_{k-1} + P_{k-1} * c_{-1}, P_{k-1}) \\ &= (g_k + p_k * (G_{k-1} + P_{k-1} * c_{-1}), p_k * P_{k-1}) \\ &= ((g_k + p_k G_{k-1}) + p_k P_{k-1} c_{-1}, P_k) \\ &= (G_k + P_k * c_{-1}, P_k)\end{aligned}$$

Όσο αφορά την τροποποίηση των γραφημάτων για τον συνυπολογισμό κρατούμενου εισόδου είναι δυνατό να κρατηθεί ανέπαφη η δομή υπολογίζοντας τα Group Generate και Group Propagate (G_i, P_i) χωρίς το κρατούμενο εισόδου και στο τέλος προστίθεται ένα ακόμα επίπεδο για κάθε δυαδικό ψηφίο i που υλοποιείται η λογική $G_i + P_i * c_{-1}$.

[Βάλε σχήμα που δείχνει το extra επίπεδο]

[Βάλε και σχήμα με Kogge-Stone δομή με c_{in}]

5 Ling Αθροιστές

Στα προηγούμενα κεφάλαια παρουσιάστηκαν διάφοροι δυαδικοί αθροιστές , ανάμεσα σε αυτούς και ο αθροιστής πρόβλεψης κρατουμένου. Επειτα γνωστοποιήθηκαν διάφοροι τρόποι υπολογισμού των κρατουμένων για την υλοποίηση του CLA. Όπως, λοιπον, έχει αποδειχθεί, οι δομές CLA είναι ιδανικές για την ελάττωση της καθυστέρησης υπολογισμού του αποτελέσματος. Παρ' όλ' αυτά στο παρόν κεφάλαιο θα παρουσιαστεί μια βελτίωση που προτάθηκε από τον Ling [5].

5.1 Βασική Θεωρία

Ξεκινώντας από την παρακάτω ισότητα

$$g_i = g_i * p_i \quad (5.1)$$

η οποία πηγάζει από

$$a_i * b_i = a_i * b_i * (a_i + b_i)$$

Η βασική βελτιστοποίηση που επέφερε η θεωρία του Ling είναι η παρακάτω τροποποίηση της συνάρτησης υπολογισμού του σήματος Group Carry Generate του συνόλου i έως j με $i > j$

$$\begin{aligned} G_{i:j} &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i p_{i-1} p_{i-2} \dots p_{j+1} p_j \\ &= p_i g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i p_{i-1} p_{i-2} \dots p_{j+1} p_j \\ &= p_i \left[g_i + g_{i-1} + p_{i-1} g_{i-2} + \dots + p_{i-1} p_{i-2} \dots p_{j+1} p_j \right] \end{aligned} \quad (5.2)$$

Η τροποποίηση της συνάρτησης G δημιουργεί έναν όρο μέσα στις αγκύλες ο οποίος ονομάζεται H κατά Ling και έχει τον παρακάτω ορισμό

$$H_{i:j} = g_i + G_{i-1:j} \quad (5.3)$$

Επιπλέον για την ανάκτηση του σήματος G , δηλαδή του κρατούμενου που παράγει ένα σύνολο, που είναι και ο αρχικός σκοπός των CLA αθροιστών γίνεται με την παρακάτω συνάρτηση :

$$G_{i:j} = p_i * H_{i:j} \quad (5.4)$$

Σημαντικό, επίσης, αυτής της παραγοντοποίησης αυτής είναι πως υποστηρίζεται ο τελεστής \oplus και η αναγωγή της σε πρόβλημα προθέματος. Έχοντας το σύνολο i έως j με $i > k > j$ αποδεικνύεται πως

$$\begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k} * G_{k-1:j} \\ p_i * H_{i:j} &= p_i * H_{i:k} + P_{i:k} * (p_{k-1} * H_{k-1:j}) \\ &= p_i [H_{i:k} + P_{i-1:k-1} * H_{k-1:j}] \\ H_{i:j} &= H_{i:k} + P_{i-1:k-1} * H_{k-1:j} \end{aligned} \quad (5.5)$$

Ακολουθεί ένα απλό παράδειγμα για καλύτερη εμπέδωση αλλά και επαλήθευση

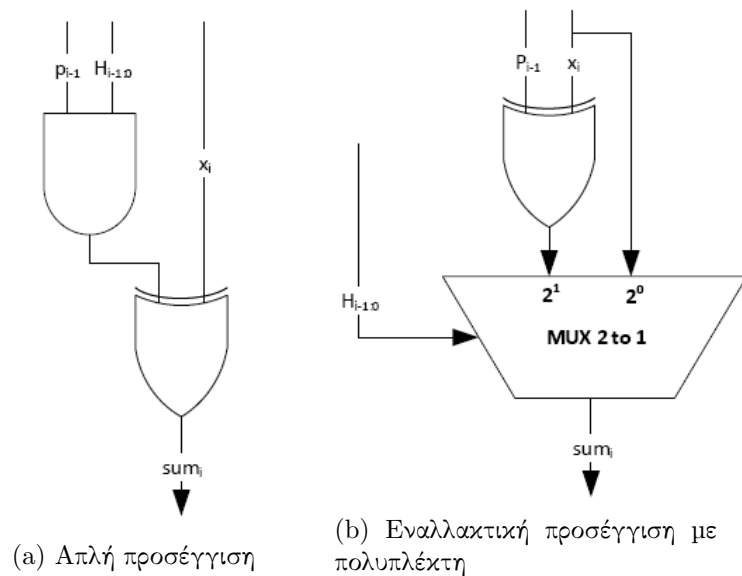
$$\begin{aligned}
H_{7:2} &= H_{7:5} + P_{6:4} * H_{4:2} \\
H_{7:5} &= g_7 + g_6 + p_6 g_5 \\
H_{4:2} &= g_4 + g_3 + p_3 g_2 \\
P_{6:4} &= p_6 p_5 p_4 \\
H_{7:2} &= g_7 + g_6 + p_6 g_5 + p_6 p_5 p_4 * (g_4 + g_3 + p_3 g_2) \\
H_{7:2} &= g_7 + g_6 + p_6 g_5 + p_6 p_5 g_4 + p_6 p_5 p_4 g_3 + p_6 p_5 p_4 p_3 g_2
\end{aligned}$$

Εφόσον στην παραγοντοποίηση που σύστησε ο Ling έχουν κληρονομηθεί όλα τα προνόμια των προθεματικών αθροιστών μένει η έκφραση του αποτελέσματος συναρτήσει του σήματος H . Ως γνωστόν, κάθε ένα δυαδικό ψηφίο του αθροίσματος με την τεχνική πρόβλεψης κρατουμένου υπολογιζόταν με την συνάρτηση $sum_i = G_{i-1:0} \oplus x_i$ όπου $x_i = a_i \oplus b_i$. Τελικά, το άθροισμα με την παραγοντοποίηση του Ling υπολογίζεται με :

$$\begin{aligned}
sum_i &= G_{i-1:0} \oplus x_i \\
&= (p_{i-1} * H_{i-1:0}) \oplus x_i
\end{aligned} \tag{5.6}$$

5.2 Πλεονεκτήματα της Ling παραγοντοποίησης

Ένας αθροιστής Ling βελτιστοποιεί την επίδοση του CLA μειώνοντας κατά ένα το πλήθος εισόδων των λογικών πυλών (fan-in). Όμως αφαιρώντας από κάθε όρο του Group Generate σήματος ένα propagate έχει ως αποτέλεσμα να αυξάνεται η πολυπλοκότητα του τελευταίου σταδίου όπου υπολογίζεται το άθροισμα (εξίσωση [5.6]). Παρατηρείται στην εξίσωση 5.6 πως πρέπει να υπολογιστεί το H στην συνέχεια να πραγματοποιηθεί η λογική του πράξης AND με το σήμα p_i και τέλος η έξοδος της πύλης AND να οδηγήσει την είσοδο της πύλης XOR σε συνδυασμό με το σήμα x_i (Εικόνα 5.1a).



Εικόνα 5.1: Λογική υπολογισμού του αθροίσματος κατά Ling

Η εξίσωση 5.6 αυτή μπορεί να εκφραστεί με μεγαλύτερη πολυπλοκότητα αλλά ταυτόχρονα μειώνοντας τον συνολικό χρόνο υπολογισμού του αθροίσματος.

$$sum_i = H_{i-1,0}?(p_{i-1} \oplus x_i) : x_i \quad (5.7)$$

Στην εναλλακτική τροποποίηση που εκφράζεται στην εξίσωση 5.7 και απεικονίζεται στην εικόνα 5.1b με πρώτη ματιά φαίνεται να είναι υπάρχει αρνητική απόδοση σε σχέση με την αρχική και πιο απλή υλοποίηση. Στην πραγματικότητα όμως η απόδοση αφορά την καθυστέρησή και όχι το εμβαδόν. Το σήμα H είναι αυτό που υπολογίζεται τελευταίο χρονικά και στην πρώτη αρχιτεκτονική είναι στο κρίσιμο μονοπάτι¹. Αντίθετα στην δεύτερη υλοποίηση η λογική πύλη XOR έχει τελική τιμή στην έξοδο της πριν οριστικοποιηθεί η τιμή του σήματος H , παράλληλα με τον υπολογισμό της.

[Ίσως να ήταν ενδιαφέρον να φτιαχτούν αυτά τα δύο modules και να μετρηθεί το εμβαδόν και το delay τους στο Design Compiler]

Παρατηρούμε πως η μέθοδος του Ling μειώνει την πολυπλοκότητα μόνο στο πρώτο επίπεδο του αθροιστή

¹Κρίσιμο μονοπάτι (critical path) Το μονοπάτι με αρχί την είσοδο και τέλος την έξοδο του κυκλώματος που έχει την μεγαλύτερη καθυστέρηση

5.3 Αραίωση σε Ling Αρχιτεκτονικές

Sparsness-2

$$\begin{aligned}sum_i &= x_i \oplus G_{i-1:0} \\sum_i &= x_i \oplus p_{i-1} * H_{i-1:0} \\sum_i &= H_{i-1:0} ? x_i \oplus p_{i-1} : x_i \\sum_{i+1} &= x_{i+1} \oplus G_{i:0} \\sum_{i+1} &= x_{i+1} \oplus (g_i + p_i * G_{i-1:0}) \\sum_{i+1} &= x_{i+1} \oplus (g_i + p_i * p_{i-1} * H_{i-1:0}) \\sum_{i+1} &= H_{i-1:0} ? x_{i+1} \oplus (g_i + p_i * p_{i-1}) : x_{i+1} \oplus g_i\end{aligned}$$

Sparsness-4

$$\begin{aligned}sum_i &= x_i \oplus G_{i-1:0} \\&= x_i \oplus p_{i-1} * H_{i-1:0} \\&= H_{i-1:0} ? x_i \oplus p_{i-1} : x_i \\sum_{i+1} &= x_{i+1} \oplus G_{i:0} \\&= x_{i+1} \oplus (g_i + p_i * G_{i-1:0}) \\&= x_{i+1} \oplus (g_i + p_i * p_{i-1} * H_{i-1:0}) \\&= H_{i-1:0} ? x_{i+1} \oplus (g_i + p_i * p_{i-1}) : x_{i+1} \oplus g_i \\sum_{i+2} &= x_{i+2} \oplus G_{i+1:0} \\&= x_{i+2} \oplus (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i G_{i-1:0}) \\&= x_{i+2} \oplus (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i p_{i-1} * H_{i-1:0}) \\&= H_{i-1:0} ? x_{i+2} \oplus (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i p_{i-1}) : x_{i+2} \oplus (g_{i+1} + p_{i+1} g_i) \\sum_{i+3} &= x_{i+3} \oplus G_{i+2:0} \\&= x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i G_{i-1:0}) \\&= x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i p_{i-1} * H_{i-1:0}) \\&= H_{i-1:0} ? x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i p_{i-1}) \\&\quad : x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i)\end{aligned}$$

6 Παραγοντοποίηση Jackson

Στο παρόν κεφάλαιο θα παρουσιάσουμε μια μέθοδο επιπλέον παραγοντοποίησης για τον υπολογισμό του κρατουμένου, η οποία προτάθηκε από τους Jackson και Tawlar [6]. Με την συγκεκριμένη τεχνική δεν μειώνεται μόνο η πολυπλοκότητα του πρώτου δέντρου προθεμάτων αλλά σε όλα τα επίπεδα. Ουσιαστικά στο κεφάλαιο αυτό θα παρουσιάσουμε την γενίκευση της παραγοντοποίησης που προτάθηκε παραπάνω.

Στις παρακάτω εξισώσεις παραγοντοποιούμε την εξίσωση υπολογισμού κρατουμένου. Στην πρώτη παραγοντοποίηση αναφερθήκαμε στην προηγούμενη ενότητα (Ling), οι επόμενες δύο επιφέρουν επιπλέον παραγοντοποίηση της αρ-
χικής συνάρτησης.

$$\begin{aligned} G_{4:0} &= g_4 + p_4g_3 + p_4p_3g_2 + p_4p_3p_2g_1 + p_4p_3p_2p_1g_0 \\ &= \begin{bmatrix} p_4 \end{bmatrix} \begin{bmatrix} g_4 + g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 \end{bmatrix} \\ &= \begin{bmatrix} g_4 + p_4p_3 \end{bmatrix} \begin{bmatrix} g_4 + g_3 + g_2 + p_2g_1 + p_2p_1g_0 \end{bmatrix} \\ &= \begin{bmatrix} g_4 + p_4g_3 + p_4p_3p_2 \end{bmatrix} \begin{bmatrix} g_4 + g_3 + g_2 + g_1 + p_1g_0 \end{bmatrix} \end{aligned} \quad (6.1)$$

Στην συνέχεια θα γενικεύσουμε αυτές τις παραγοντοποιήσεις και επιπλέον θα δείξουμε τεχνικές μείωσης πολυπλοκότητας και στα επιμέρους τμήματα τις παραπάνω εξίσωσης μέσα στις αγκύλες. Θα ονομάσουμε αυτή την γενίκευση ως Jackson παραγοντοποίηση.

6.1 Βασικοί Όροι

Αρχικά ο Jackson ορίζει δυο βοηθητικά σήματα τα D και B, όπου το σήμα $D_{i:j}$ είναι αληθές όταν η ομάδα δυαδικών ψηφίων των σημμάτων εισόδου από τα ψηφία a_i, b_i έως τα a_j, b_j παράγουν κρατούμενο εξόδου ή διαδίδουν το κρατούμενο εισόδου,

$$\begin{aligned} D_{i:k} &= G_{j:k} + P_{j:k} \\ &= G_{j:k+1} + P_{j:k} \end{aligned} \quad (6.2)$$

ενώ το σήμα $B_{i:j}$ αληθεύει όταν έστω ένα ζευγάρι (a_k, b_k) παράγει κρατούμενο

$$B_{i:j} = g_i + g_{i-1} + \dots + g_j \quad (6.3)$$

Γνωρίζοντας τα παραπάνω σήματα μπορούμε να εκφράσουμε πλέον την βασική παραγοντοποίηση του υπολογισμού του κρατουμένου εξόδου που συστήνεται από τον Jackson

$$G_{j:i} = D_{j:k} \begin{bmatrix} B_{j:k} + G_{k-1:i} \end{bmatrix} \quad (6.4)$$

Από την παραπάνω εξίσωση το τμήμα μέσα στις αγκύλες ορίζεται ως $R_{j:i}^{j-k+1}$

$$\begin{aligned} R_{j:i}^{j-k+1} &= B_{j:k} + G_{k-1:i} \\ R_{n-1:j}^{n-m} &= B_{n-1:m} + G_{m-1:j} \\ R_{i:j}^p &= B_{i:i-p+1} + G_{i-p:j} \end{aligned} \quad (6.5)$$

οπού η υπογεγραμμένη αναφέρεται στην ομάδα των δυαδικών ψηφίων εισόδου και το υπερκείμενο στο πλήθος των ψηφίων που δέχεται σαν είσοδο το σήμα B . Έτσι η παραπάνω εξίσωση παίρνει την παρακάτω μορφή

$$G_{j:i} = D_{j:k} R_{j:i}^{j-k+1} \quad (6.6)$$

Πρέπει να σημειώσουμε πως στην περίπτωση που $j = k$ στην παραπάνω περίπτωση τότε περιγράφουμε την παραγοντοποίηση που παρουσιάσαμε στο προηγούμενο κεφάλαιο όπως είχε οριστεί από τον Ling.

Το τελικό άθροισμά υπολογίζεται παρόμοια με τον Ling

$$\begin{aligned} sum_i &= x_i \oplus G_{i:0} \\ sum_i &= R?(x \oplus D) : x \end{aligned} \quad (6.7)$$

6.2 Αναδρομή του Jackson

Βασικό πλεονέκτημα των αθροιστών προθέματος είναι η ανάδραση, όπου όπως έχει προαναφερθεί το κρατούμενο εξόδου μίας ομάδας δυαδικών ψηφίων μπορεί να παραχθεί υπολογίζοντας επιμέρους μικρότερες υποομάδες αυτού του συνόλου. Παρακάτω θα αποδειχθεί πως και το σήμα R που ορίστηκε παραπάνω έχει επίσης αυτή την ιδιότητα.

Θα κατασκευάσουμε το σήμα $R_{n-1:0}$ με είσοδο n δυαδικά ψηφία συνδυάζοντας υποομάδες αυτού του συνόλου χωρίζοντας το σε τρία, ίσου μεγέθους διαμερίσματα, τις υποομάδες bits εισόδου $n-1 : k$, $k-1 : j$ και $j-1 : 0$. Επίσης θα επιλέξουμε και τα m και v ως τα μεσαία στοιχεία των δύο τελευταίων διαστημάτων.

Αρχίζοντας με την εξίσωση του R που θέλουμε να παράξουμε

$$R_{n-1:0}^{n-m} = B_{n-1:m} + G_{m-1:0}$$

χρησιμοποιώντας την εξίσωση αναδρομής του G έχουμε

$$R_{n-1:0}^{n-m} = B_{n-1:m} + G_{m-1:j} + P_{m-1:j} G_{j-1:0}$$

επίσης σπάμε το B σε δυο σήματα και έχουμε

$$R_{n-1:0}^{n-m} = [B_{n-1:k}] + [B_{k-1:m} + G_{m-1:j}] + P_{m-1:j} G_{j-1:0}$$

Ήδη οι πρώτοι δυο όροι που έχουν εμφανιστεί μέσα στις αγκύλες σχηματίζουν τα σήματα R από υποομάδες, κατασκευάζοντας και τον τελευταίο όρο, εφαρμόζοντας την εξίσωση του G που παρουσιάσαμε παραπάνω, έχουμε

$$R_{n-1:0}^{n-m} = [B_{n-1:k}] + [B_{k-1:m} + G_{m-1:j}] + [P_{m-1:j}D_{j-1:v}][B_{j-1:v} + G_{v-1:0}]$$

Οπότε η τελική εξίσωση είναι

$$R_{n-1:0}^{n-m} = R_{n-1:k}^{n-k} + R_{k-1:j}^{k-m} + [P_{m-1:j}D_{j-1:v}]R_{j-1:0}^{j-v} \quad (6.8)$$

Για περισσότερη ευκολία στις παρακάτω εξισώσεις θα ορίσουμε και το σήμα Q το οποίο είναι το σήμα μέσα στις αγκύλες στην εξίσωση παραπάνω

$$Q_{n-1:k}^{n-m} = P_{n-1:m}D_{m-1:k} \quad (6.9)$$

Μεχρι στιγμής έχουμε αποδειξει πως το σήμα R ενός συνόλου δυαδικών ψηφίων μπορεί να υπολογιστεί από τα σήματα R μικρότερων υποομάδων. Πρέπει να τονίσουμε πως υπάρχουν πολλοί διαφορετικοί συνδιασμοί και τύποι, που υλοποιούν τον αναδρομικό υπολογισμό, οι οποίοι θα παρουσιαστούν σε παρακάτω κεφάλαιο. Παρακάτω θα δείξουμε πως και το σήμα Q υπολογίζεται αναδρομικά, αφού πρώτα παρουσιάσουμε τον αναδρομικό υπολογισμό του σήματος D με τους παρακάτω δύο τύπους

$$\begin{aligned} D_{j:i} &= D_{j:k}[B_{j:k} + D_{k-1:i}] \\ D_{j:i} &= G_{j:k} + P_{j:k}D_{k-1:i} \end{aligned} \quad (6.10)$$

Με την ίδια λογική που ακολουθήσαμε προηγουμένως θα υποδείξουμε πώς υπολογίζεται αναδρομικά το σήμα Q με είσοδο το σύνολο των bits $n - 1 : 0$. Χωρίζουμε το σύνολο σε τρία διαμερίσματα $n - 1 : k$, $k - 1 : j$ και $j - 1 : 0$, ομοίως επιλέγουμε τα μεσαία στοιχεία των τελευταίων δύο υποομάδων m και n αντίστοιχα. Ξεκινώντας από την εξίσωση του σήματος Q που παρουσιάσαμε παραπάνω, αρχικά κάνουμε χρήση της πρώτης εξίσωσης από τις παραπάνω δυο αναδρομικές συναρτήσεις του D, ενώ παράλληλα σπάμε το σήμα P, και έχουμε

$$\begin{aligned} Q_{n-1:0}^{n-m} &= P_{n-1:m}D_{m-1:0} \\ Q_{n-1:0}^{n-m} &= P_{n-1:m}D_{m-1:j}[B_{m-1:j} + D_{j-1:0}] \\ Q_{n-1:0}^{n-m} &= P_{n-1:k}[P_{k-1:m}D_{m-1:j}][B_{m-1:j} + D_{j-1:0}] \end{aligned}$$

Σε αυτό το σημείο μέσα στις δύο πρώτες αγκύλες έχουμε ήδη χτίσει τα Q των πρώτων δύο υποσυνόλων, για να παράξουμε και το τρίτο στην τελευταία αγκύλη θα κάνουμε χρήση της δεύτερης εξίσωσης από τις από τις δυο αναδρομικές συναρτήσεις του D

$$Q_{n-1:0}^{n-m} = P_{n-1:k}[P_{k-1:m}D_{m-1:j}][B_{m-1:j} + G_{j-1:v} + P_{j-1:v}D_{v-1:0}]$$

Αντικαθιστώντας στην παραπάνω εξίσωση αντίστοιχα σήματα R και Q έχουμε την τελική μορφή της αναδρομικής συνάρτησης

$$Q_{n-1:0}^{n-m} = Q_{n-1:k}^{n-k} Q_{k-1:j}^{k-m} [R_{m-1:v}^{m-j} + Q_{j-1:0}^{j-v}] \quad (6.11)$$

6.3 Εφαρμογές της αναδρομής

Όπως προαναφέρθηκε παραπάνω υπάρχουν διάφορες υλοποιήσεις της αναδρομής ακόμα και με ίδιο αριθμό υποομάδων [7] σε αντίθεση με τις απλές αναδρομές των αθροιστών προθέματος. Παρακάτω θα παρουσιαστούν όλες οι πιθανές μορφές των σημάτων R και Q με πλήθος υποομάδων δύο έως τέσσερα.

[Implementation of Recursive Ling Adders in CMOS VLSI
by Neil Burgess]

Παρατηρείται πως όταν απλοποιείται η υλοποίηση του σήματος R τότε επιβαρύνεται η υλοποίηση του Q

6.4 Πρακτικές εφαρμογές

Στην προηγούμενη παράγραφο παρουσιάστηκε ένα μεγάλο πλήθος επιτρεπτών υλοποιήσεων για τον αναδρομικό υπολογισμό των σημάτων R και Q. Ασχέτως, όμως, αν όλες αυτές οι συναρτήσεις βγάζουν σωστό αποτέλεσμα, το κύριο ζητούμενο είναι το πιο αποδοτικό. Ένα πιθανό σενάριο είναι η δοκιμή κάθε συνδυασμού των παραπάνω συναρτήσεων και συγκρίνοντας τα τελικά αποτελέσματα, λαμβάνοντας ως παράγοντες σύγκρισης την καθυστέρηση, το εμβαδόν και την κατανάλωση, να χρησιμοποιείται ο κατάλληλος για την εφαρμογή. Αυτή η τεχνική είναι αρκετά χρονοβόρα και πολύπλοκη, οι συνδυασμοί είναι πολλοί και τριπλασιάζονται εάν συμπεριληφθούν και περιπτώσεις αραιών υλοποιήσεων (Ενότητα 6.6).

Το κλειδί στον αποδοτικό σχεδιασμό αναδρομικών Ling αθροιστών βρίσκεται στην σωστή ισορροπία πολυπλοκότητας μεταξύ των σημάτων R, D και Q. Στην παραγοντοποίηση του Ling αφαιρείται ένας propagate p παράγοντας από την πολυπλοκότητα του κρατουμένου και στο τελευταίο στάδιο συνυπολογίζεται. Αντίστοιχα, στις περιπτώσεις των αθροιστών Jackson, που στην ουσία είναι μια γενικευμένη έκφραση του Ling, αφαιρούνται n propagate σήματα τα οποία μειώνουν την πολυπλοκότητα του σήματος R που ελέγχει τον πολυπλέκτη του τελευταίου σταδίου, αντίστοιχο του H στον Ling, άλλα επιβαρύνεται το σήμα D. Γνωρίζοντας πως το σήμα D εισάγεται μαζί με το x σε πύλη XOR πριν τον πολυπλέκτη (εξίσωση 6.7), το D πρέπει να έχει την τελική του τιμή τουλάχιστον δυο FO4² πριν από το σήμα R. Στον παρακάτω πίνακα 6.1 συγκρίνονται οι συναρτήσεις ενός απλού παράλληλου προθεματικού αθροιστή και ενός Jackson.

²FO4 : Fan-out of 4 is a process-dependent delay metric used in digital CMOS technologies

| Radix | Parallel Prefix | Jackson |
|-------|---|--|
| 3 | $G_2 + P_2G_1 + P_2P_1G_0$ $P_2P_1P_0$ | $R_2 + R_1 + Q_1R_0$ $Q_2Q_1[R_0 + Q_0]$ |
| 4 | $G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$ $P_3P_2P_1P_0$ | $R_3 + R_2 + Q_2R_1 + Q_2Q_1R_0$ $Q_3Q_2Q_1[R_0 + Q_0]$ |

Πίνακας 6.1: Σύγκριση προθεματικού αθροιστή Jackson

Είναι εμφανές το ζύγισμα που αναφέρθηκε. Όσο απλοποιείται το σήμα R (η πάνω εξίσωση σε κάθε κελί του πίνακα στην στήλη των Jackson συναρτήσεων) τόσο πιο περίπλοκο γίνεται το σήμα Q (η κάτω εξίσωση). Επίσης παρατηρείται στην γραμμή Radix-4 πως ενώ ο Q παράγοντας είναι πιο περίπλοκος από τον P, ο όρος R εξισώνει αυτή την διαφορά και στις περισσότερες περιπτώσεις συνολικά είναι πιο αποδοτική η υλοποίηση κατά Jackson.

Παρακάτω δίνεται ένας αρχικός προσανατολισμός σε μορφή κανόνων σχεδίασης τω αθροιστών Jackson, χωρίς όμως την εγγύησης του αποδοτικότερου αποτελέσματος. Ανάλογα την εφαρμογή και τους πόρους είναι πιθανό να παραβιαστούν με σκοπό το καλύτερο δυνατό αποτέλεσμα. Οι κανόνες αυτοί προτείνουν :

1. Όπως προαναφέρθηκε, η κατασκευή του όρου D πρέπει να είναι τουλάχιστον δυο με τρία FO4 επίπεδα απο τον αντίστοιχο όρο R.
2. Δεν πρέπει να αφαιρούμε κανένα propagate p όρο μέσω της παραγοντοποίησης του τελευταίου σταδίου. Ο λόγος για τον ισχυρισμό αυτό είναι απόρροια της παραπάνω ζύγισης.
3. Στο πρώτο επίπεδο προτείνεται η Ling παραγοντοποίηση, δηλαδή η αφαίρεση ενός propagate. Αυτή η πρόταση έχει επίσης να κάνει με την ποικιλία των CMOS κελιών που παρέχονται από τις βιβλιοθήκες των κατασκευαστών. Με Ling παραγοντοποίηση στην πρώτη βαθμίδα δίνεται το προνόμιο χρήσης ενός μόνο κελιού το οποίο στις περισσότερες βιβλιοθήκες συμπεριλαμβάνεται (AOI και OAI³).
4. Οι όροι D μπορούν να κατασκευαστούν από R και Q, οπότε είναι προτιμότερο, όσο αφορά το εμβαδόν, το σήμα αυτό να οδηγείται από ήδη υπάρχοντα σήματα R και Q.

6.5 Παράδειγμα υλοποίησης

Με σκοπό την πλήρη κατανόηση της δομής που περιγράφηκε για τον υπολογισμό του κρατουμένου, σε αυτή την παράγραφο θα αναπτυχθεί ένας αθροιστής των 18 δυαδικών ψηφίων

[Δημιουργία Jackson αθροιστή]

³AND-OR-INVERTER και OR-AND-INVERTER

6.6 Αραίωση σε Jackson Αρχιτεκτονικές

Sparsness-2

$$\begin{aligned}
sum_i &= x_i \oplus G_{i-1:0} \\
sum_i &= x_i \oplus D_{i-1:k} R_{i-1:0}^{i-k} \\
sum_i &= R_{i-1:0}^{i-k} ? x_i \oplus D_{i-1:k} : x_i \\
sum_{i+1} &= x_{i+1} \oplus G_{i:0} \\
sum_{i+1} &= x_{i+1} \oplus (g_i + p_i * G_{i-1:0}) \\
sum_{i+1} &= x_{i+1} \oplus (g_i + p_i * D_{i-1:k} R_{i-1:0}^{i-k}) \\
sum_{i+1} &= R_{i-1:0}^{i-k} ? x_{i+1} \oplus (g_i + p_i * D_{i-1:k}) : x_{i+1} \oplus g_i
\end{aligned}$$

Sparsness-4

$$\begin{aligned}
sum_i &= x_i \oplus G_{i-1:0} \\
&= x_i \oplus D_{i-1:k} R_{i-1:0}^{i-k} \\
&= R_{i-1:0}^{i-k} ? x_i \oplus D_{i-1:k} : x_i \\
sum_{i+1} &= x_{i+1} \oplus G_{i:0} \\
&= x_{i+1} \oplus (g_i + p_i * G_{i-1:0}) \\
&= x_{i+1} \oplus (g_i + p_i * D_{i-1:k} R_{i-1:0}^{i-k}) \\
&= R_{i-1:0}^{i-k} ? x_{i+1} \oplus (g_i + p_i * D_{i-1:k}) : x_{i+1} \oplus g_i \\
sum_{i+2} &= x_{i+2} \oplus G_{i+1:0} \\
&= x_{i+2} \oplus (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i G_{i-1:0}) \\
&= x_{i+2} \oplus (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i D_{i-1:k} R_{i-1:0}^{i-k}) \\
&= R_{i-1:0}^{i-k} ? x_{i+1} \oplus (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i D_{i-1:k}) \\
&\quad : x_{i+1} \oplus (g_{i+1} + p_{i+1} g_i) \\
sum_{i+3} &= x_{i+3} \oplus G_{i+2:0} \\
&= x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i G_{i-1:0}) \\
&= x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i D_{i-1:k} R_{i-1:0}^{i-k}) \\
&= R_{i-1:0}^{i-k} ? x_{i+1} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i D_{i-1:k}) \\
&\quad : x_{i+3} \oplus (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i)
\end{aligned}$$

7 Αθροιστής υπολοίπου $2^n - 1$

Η αριθμητική υπολοίπου αφορά το υπόλοιπο ενός αριθμού X διαιρεμένου με έναν Y , εναλλακτικά αφαιρείτε από το X το Y μέχρι $X < Y$. Την πράξη αυτή την συμβολίζουμε ως

$$X \bmod Y$$

Αριθμητικές υπολοίπου έχουν εφαρμογές σε ένα μεγάλο πλήθος εφαρμογών εφόσον αποτελούν και την βάση για τα Residue Number Systems (RNS). Επίσης αποτελούν μέρος της ψηφιακής επεξεργασίας σημάτων και ψηφιακών φίλτρων, κρυπτογραφίας, σε τεχνικές ανίχνευσης και διόρθωσης σφάλματος καθώς και σε υψηλών ταχυτήτων δίκτυα. Η διαδική άθροιση είναι σε αριθμητική υπολοίπου και γράφεται $(A + B) \bmod n$ όπου n είναι το πλήθος των δυαδικών ψηφίων των A και B . Στην παρούσα ενότητα θα μελετηθούν οι αθροιστές υπολοίπου $2^n - 1$ όπου η επιτάχυνση τους είναι ο σκοπός μας.

7.1 Basic Operation

Ο μαθηματικός υπολογισμός του αθροίσματος υπολοίπου $2^n - 1$ στην πραγματικότητα είναι ένας υπο-συνθήκη υπολογισμός με συνθήκη $A + B < 2^n$ και ορίζεται ως

$$(A + B) \bmod (2^n - 1) = \begin{cases} (A + B) \bmod 2^n, & A + B < 2^n \\ (A + B) \bmod 2^n + 1, & A + B \geq 2^n \end{cases} \quad (7.1)$$

Ο ορισμός αυτός μαθηματικά έχει ένα λάθος το οποίο όμως, λαμβάνοντας μία παραδοχή, θα εξαλειφθεί.

[Γιατί είναι αυτός ο ορισμός ?

Γράψε την εξήγηση]

Για παράδειγμα έστω πως $n = 3$ άρα και $2^n - 1 = 7$, όπου n το πλήθος των δυαδικών ψηφίων που έχει κάθε αριθμός εισόδου. Για τον υπολογισμό του υπολοίπου, όπως προαναφέρθηκε, διαιρείται ο αριθμός εισόδου με το 7 και το υπόλοιπο είναι το αποτέλεσμα. Με είσοδο τον αριθμό 3 υπολογίζεται $3/7 = 0$ και υπόλοιπο 3. Παρακάτω παρουσιάζονται μια σειρά από παραδείγματα.

$$8 \bmod 7 = 1$$

$$7 \bmod 7 = 0$$

$$14 \bmod 7 = 0$$

$$6 \bmod 7 = 6$$

$$13 \bmod 7 = 6$$

Όπως είναι εμφανές και στα παραδείγματα ο μαθηματικός υπολογισμός που ορίστηκε φαίνεται να μην ισχύει για την περίπτωση $7 \bmod 7$ διότι ο αριθμός 7 είναι μικρότερος του $2^n = 2^3 = 8$, άρα ανήκει στην πρώτη περίπτωση της

εξίσωσης 7.1 όπου σύμφωνα με αυτή το αποτέλεσμα θα έπρεπε να ήταν επτά και όχι μηδέν. Σε αυτό το σημείο λοιπόν είναι σημαντικό να τονιστεί πως χρησιμοποιείται διπλή αναπαράσταση του μηδέν. Η μία αναπαράσταση είναι η προφανής όπου όλα τα ψηφία είναι 0 και η δεύτερη είναι η $2^n - 1$, δηλαδή όλα τα ψηφία να είναι στο 1.

[Διπλή Αναπαράσταση του μηδέν, πως μπορούμε να την αντιμετωπίσουμε]

Υπάρχουν διάφοροι τρόποι για να υπολογιστεί στο υλικό το αποτέλεσμα ενός αθροιστή υπολοίπου $2^n - 1$.

Η πιο απλή ιδέα είναι αποτελείται από δύο αθροιστές όπου ο πρώτος δεν έχει κρατούμενο εισόδου, παίρνει ως είσοδο τα A και B και η έξοδος του τροφοδοτεί την είσοδο του δεύτερου αθροιστή με δεύτερο όρισμα τον μηδενικό αριθμό και κρατούμενο εισόδου το κρατούμενο εξόδου του πρώτου αθροιστή. Το άθροισμα του δεύτερου αθροιστή είναι και το ζητούμενο. Στο παρακάτω σχηματικό αποτυπώνεται αυτή η απλή αρχιτεκτονική που περιγράφηκε.

[Βάλε εικόνα του απλού αθροιστή $2^n - 1$]

Η παραπάνω τεχνική έχει πολύ μεγάλη χρονική καθυστέρηση διότι υπάρχουν δύο επίπεδα αθροιστών. Για να μειωθεί ο χρόνος που απαιτείται για να οδηγηθεί η έξοδος με το σωστό αποτέλεσμα μπορούμε να εκτελέσουμε παράλληλα δυο προσθέσεις του A και B με τον ένα αθροιστή να έχει κρατούμενο εισόδου και τον άλλο να μην έχει. Τα αποτελέσματα των δύο αθροιστών θα οδηγούνται σε έναν πολυπλέκτη με είσοδο επιλογής το κρατούμενο εισόδου του αθροιστή χωρίς κρατούμενο εισόδου. Αν η είσοδος επιλογής είναι ενεργή τότε θα επιλέγεται η έξοδος του αθροιστή με κρατούμενο εισόδου όπως φαίνεται στην παρακάτω εικόνα.

[Βάλε εικόνα του Επιλογής κρατουμένου αθροιστή $2^n - 1$]

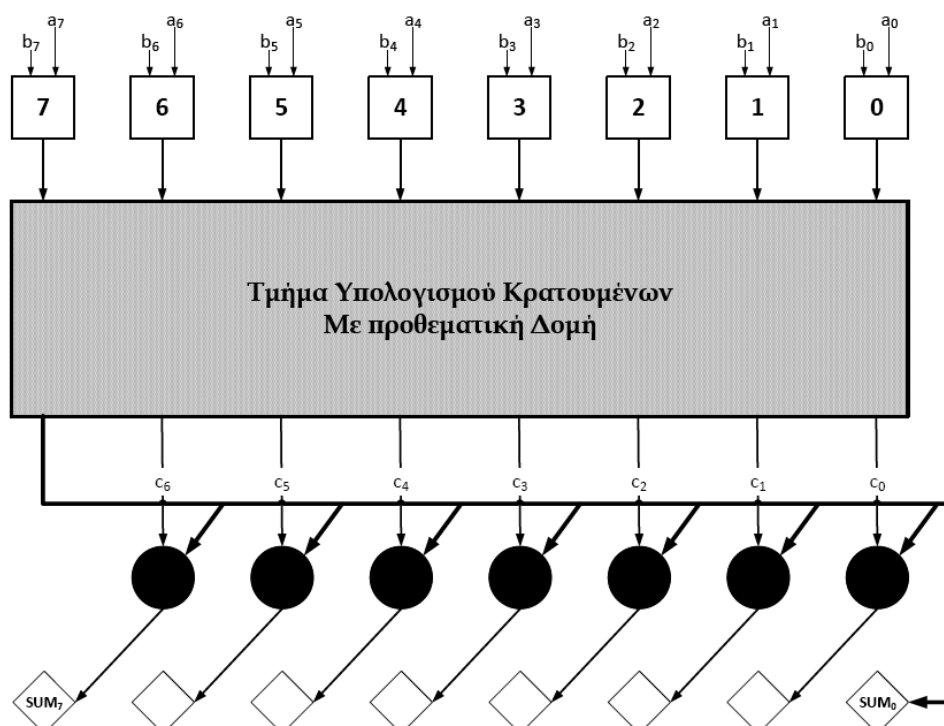
7.2 Prefix αθροιστές $2^n - 1$

Στην περίπτωση ενός αθροιστή που αποτελείται από δύο στάδια, το πρώτο χωρίς κρατούμενο εισόδου και το δεύτερο στάδιο έχει κρατούμενο εισόδου το κρατούμενο εξόδου του πρώτου. Οπότε στο πρώτο στάδιο $c_{-1} = 0$. Από την εξίσωση 4.6 συμπεραίνετε πως $(G'_{n-1}, P'_{n-1}) = (G_{n-1}, P_{n-1})$. Στο δεύτερο επίπεδο ισχύει $c_{-1} = G'_{n-1} = G_{n-1}$ εφόσον στο πρώτο στάδιο δεν το κρατούμενο εισόδου είναι μηδέν. Για $c_{-1} = G_{n-1}$ η εξίσωση 4.6 παίρνει την μορφή

$$\begin{aligned}(G'_{n-1}, P'_{n-1}) &= (G_{n-1} + P_{n-1} * c_{-1}, P_{n-1}) \\ &= (G_{n-1} + P_{n-1} * G_{n-1}, P_{n-1}) \\ &= (G_{n-1}, P_{n-1})\end{aligned}$$

Αποτέλεσμα της παραπάνω εξίσωσης είναι πως το κρατούμενο εξόδου του δεύτερου επιπέδου $c'_{n-1} = G'_{n-1}$ (αν και δεν αφορά άμεσα την υλοποίηση εφόσον δεν υπάρχει λόγος υπολογισμού του στους αθροιστές υπολοίπου $2^n - 1$) είναι σταθερό από το πρώτο επίπεδο και παραμένει και στο δεύτερο. Επίσης από την εξίσωση 4.6 ισχύει

$$(G'_i, P'_i) = (G_i + P_i G_{n-1}, P_i) \quad (7.2)$$



Εικόνα 7.1: Απλή δομή προθεματικού αθροιστή υπολοίπου $2^8 - 1$

Ένας σχεδιασμός σαν τον παραπάνω, εκτός από το γεγονός του επιπλέον επιπέδου, έχει και το μειονέκτημα στο ότι το κρατούμενο εξόδου του πρώτου επιπέδου οδηγεί η κόμβους του τελευταίου επιπέδου, όπως εκφράζεται και την εξίσωση 7.2.

J.J. Shedletsky, ^aComment on the Sequential and Indeterminate Behavior of an End-Around-Carry Adder,^o IEEE Trans. Computers, vol. 26, pp. 271-272, Mar. 1977.

J.F. Wakerly, ^aOne's Complement Adder Eliminates Unwanted Zero,^o Electronics, pp. 103-105, Feb. 1976.

7.3 Architectures Improvements

Χρησιμοποιώντας τον ειδικό τελεστή που παρουσιάστηκε στο κεφάλαιο 4 " \otimes " ο αθροιστής υπολοίπου $2^n - 1$ μπορεί να υλοποιηθεί με ένα ακόμα παράλληλο τμήμα με αποτέλεσμα να μειωθεί κατά ένα επίπεδο ο υπολογισμός του [8]. Όπως εξηγήθηκε προηγουμένως το κρατούμενο εισόδου, στο δεύτερο επίπεδο, του αθροιστή υπολοίπου η δυαδικών ψηφίων είναι ίσο με το κρατούμενο εξόδου του, στο πρώτο επίπεδο που είναι χωρίς κρατούμενο εισόδου, $c_{-1} = G_{n-1}$, άρα ισχύει και $(G_{-1}^*, P_{-1}^*) = (G_{n-1}, P_{n-1})$, συμβολίζοντας G^* τα κρατούμενα που υπολογίζονται στο δεύτερο επίπεδο.

Με επαγωγικό τρόπο θα αποδειχθεί πως

$$(G_i^*, P_i^*) = \begin{cases} (G_{n-1}, P_{n-1}), & i = -1 \\ (g_i, p_i) \otimes (G_{i-1}^*, P_{i-1}^*), & 0 \leq i \leq n-2 \end{cases} \quad (7.3)$$

Απόδειξη :

1. Για $i = -1$ ισχύει $(G_{-1}^*, P_{-1}^*) = (G_{n-1}, P_{n-1})$. Όπως προαναφέρθηκε προηγουμένως, $c_{-1} = G_{n-1}$, $c_{-1}^* = G_{-1}^*$. Άρα $c_{-1}^* = G_{-1}^*$.
2. Αρχική υπόθεση πως η εξίσωση 7.3 ισχύει και για $i = k-1$ με $k \geq 0$. Άρα ισχύει και $c_{k-1}^* = G_{k-1}^*$. Θα αποδειχθεί πως ισχύει και για $i = k$. Ξεκινώντας με τον ορισμό και έχοντας την παραπάνω υπόθεση καταλήγουμε

$$\begin{aligned} (G_k^*, P_k^*) &= (g_k, p_k) \otimes (G_{k-1}^*, P_{k-1}^*) \\ &= (g_k, p_k) \otimes (c_{k-1}^*, P_{k-1}^*) \\ &= (g_k + p_k c_{k-1}^*, p_k P_{k-1}^*) \\ &= (c_k', P_k) \end{aligned}$$

με τις παραπάνω ισότητες πως ισχύει $c_i^* = G_i^*$ για $-1 \leq i \leq n-2$.

Πριν την παρουσίαση της νέας δομής, η οποία έχει ένα λιγότερο επίπεδο από την δομή που περιγράφηκε παραπάνω, πρέπει να γίνει μία απόδειξη ενός ισχυρισμού που θα χρειαστεί στην συνέχεια. Ο ισχυρισμός αυτός είναι

$$(G_i, P_i) \otimes (g, p) \otimes (G_i, P_i) = (G_i, P_i) \otimes (g, p) \quad (7.4)$$

Απόδειξη

$$\begin{aligned} (G_i, P_i) \otimes (g, p) \otimes (G_i, P_i) &= (G_i + P_i * g, P_i * p) \otimes (G_i, P_i) \\ &= (G_i + P_i * g + P_i * p * G_i, P_i * p * P_i) \\ &= (G_i * (1 + P_i * p) + P_i * g, P_i * p) \\ &= (G_i + P_i * g, P_i * p) \\ &= (G_i, P_i) \otimes (g, p) \end{aligned}$$

Από την εξίσωση 7.3 αποδεικνύεται

$$\begin{aligned}
 (G_i^*, P_i^*) &= (g_i, p_i) \otimes (G_{i-1}^*, P_{i-1}^*) \\
 &= (g_i, p_i) \otimes (g_{i-1}, p_{i-1}) \otimes \dots \otimes (g_0, p_0) \otimes (G_{-1}^*, P_{-1}^*) \\
 &= (g_i, p_i) \otimes \dots \otimes (g_0, p_0) \otimes (G_{n-1}, P_{n-1}) \\
 &= (g_i, p_i) \otimes \dots \otimes (g_0, p_0) \otimes (g_{n-1}, p_{n-1}) \otimes (G_{n-2}, P_{n-2}) \\
 &= (g_i, p_i) \otimes \dots \otimes (g_0, p_0) \otimes (g_{n-1}, p_{n-1}) \otimes \dots \otimes (g_i, p_i) \otimes \dots \otimes (g_0, p_0) \\
 &= (G_i, P_i) \otimes (g_{n-1}, p_{n-1}) \otimes \dots \otimes (g_{i+1}, p_{i+1}) \otimes (G_i, P_i) \\
 (G_i^*, P_i^*) &= (G_i, P_i) \otimes (g_{n-1}, p_{n-1}) \otimes \dots \otimes (g_{i+1}, p_{i+1})
 \end{aligned} \tag{7.5}$$

Στην τελευταία ισότητα της παραπάνω διαδικασίας εφαρμόζεται ο ισχυρισμός της εξίσωσης 7.4 Η σχέση αυτή (εξίσωση 7.5) είναι και η αρχιτεκτονική βελτίωση των αθροιστών υπολοίπου $2^n - 1$.

Σε αυτό το σημείο είναι αναγκαίο να δηλωθεί μια αναπαράσταση με σκοπό την ευκολία στην έκφραση συναρτήσεων και όρων. Το σήμα G_i αντιπροσωπεύει το $G_{i:0}$, δηλαδή έχει κάθε ζευγάρι (g_k, p_k) , με $i \geq k \geq 0$, με τον τελεστή \otimes . Είναι δυνατό, όπως έχει προαναφερθεί, να έχουμε τον όρο $G_{i:j}$, με $i \geq j$ με την περίπτωση της ισότητας τότε το $G_{i:i} = g_i$. Θα οριστεί μια νέα έκφραση, η οποία προβλέπει την περίπτωση του $i < j$ στην έκφραση $G_{i:j}$. Στην περίπτωση, λοιπόν, που το $i < j$, τότε ορίζουμε :

$$(G_{i:j}, P_{i:j}) = \begin{cases} (g_i, p_i) \otimes (g_{i-1}, p_{i-1}) \otimes \dots \otimes (g_j, p_j), & i > j \\ (g_i, p_i), & i = j \\ (G_{i:0}, P_{i:0}) \otimes (G_{n-1:j}, P_{n-1:j}), & i < j \end{cases} \tag{7.6}$$

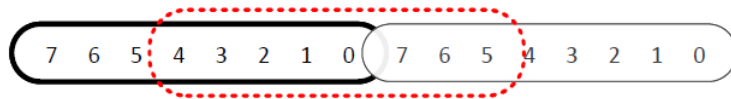
Έχοντας ορίσει την παραπάνω έκφραση, είναι αποδεκτή και μια εναλλακτική διατύπωση των σημάτων (G_i^*, P_i^*)

$$(G_i^*, P_i^*) = (G_{i:i+1}, P_{i:i+1}) \tag{7.7}$$

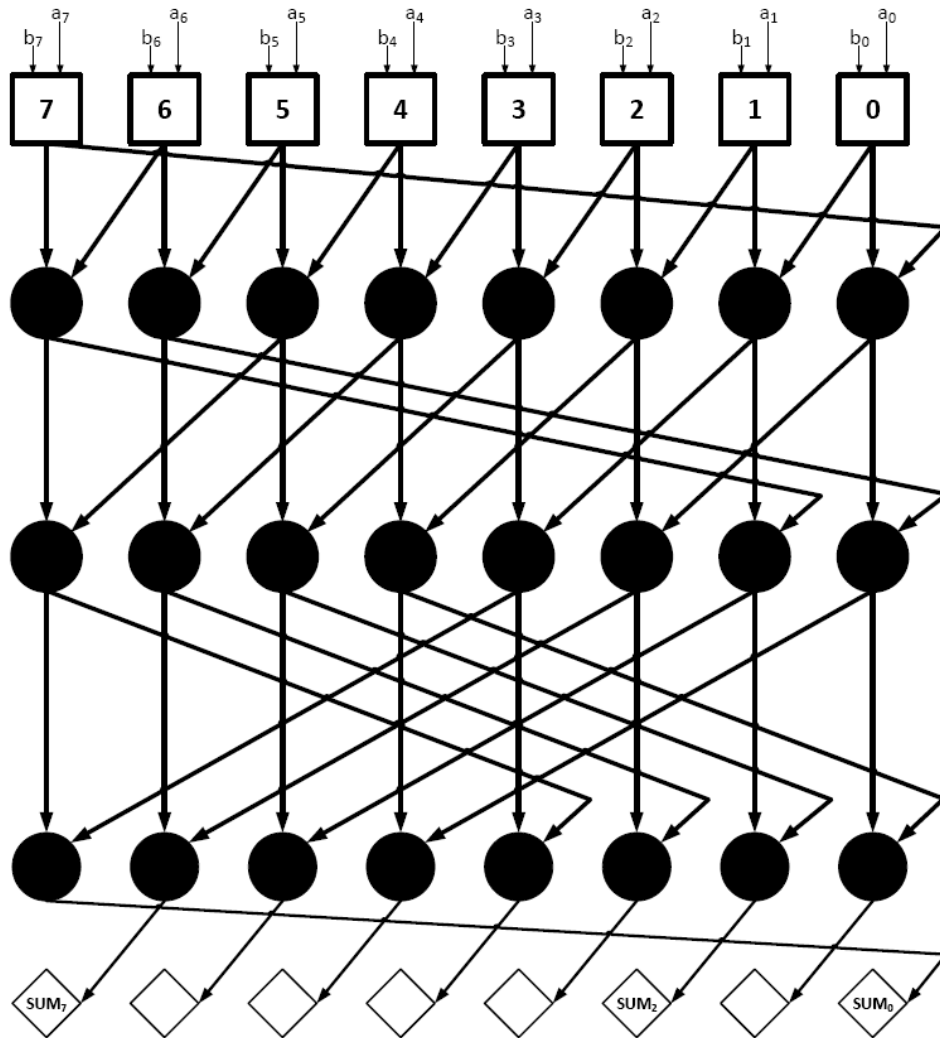
Το ίδιο ισχύει και για τους υπόλοιπους όρους (P,H,Q). Για παράδειγμα σε ένα αθροιστή των οκτώ δυαδικών ψηφίων $n = 8$, για τον υπολογισμό του $G_{1:6}$ υπονοείται η παρακάτω έκφραση :

$$G_{1:7} = g_1 + p_1 g_0 + p_1 p_0 g_7 + p_1 p_0 p_7 g_6$$

Στην παρακάτω εικόνα φαίνεται η διαδικασία αυτή για το $G_{4:5} = G_{4:5}^*$. Είναι εμφανές πως το προηγούμενο στοιχείο του μηδέν είναι το $n - 1$, δηλαδή το επτά στην περίπτωση του 8-bit αθροιστή.



[Εξήγησε το παρακάτω παράδειγμα]



Εικόνα 7.2: Παράλληλος προθεματικός αθροιστής υπολοίπου $2^8 - 1$

8 Ανάπτυξη Αθροιστών υπολοίπου $2^n - 1$

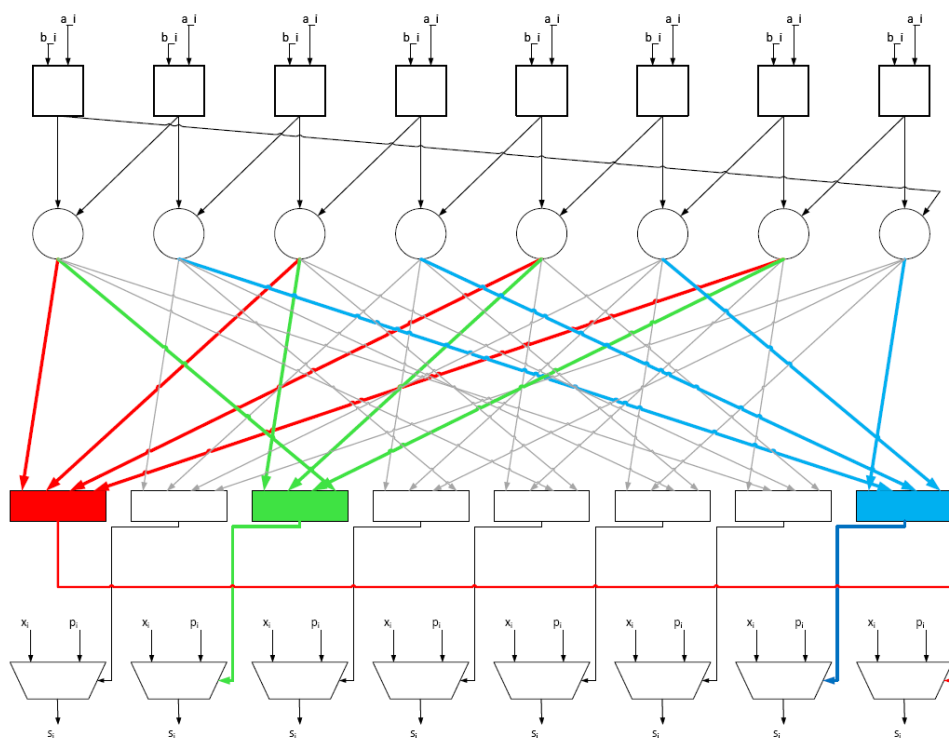
Σε αυτό το κεφάλαιο θα αναπτυχθούν συνολικά δώδεκα αθροιστές υπολοίπου $2^n - 1$ ακολουθώντας την αρχιτεκτονική που παρουσιάστηκε στο προηγούμενο κεφάλαιο με τα ελάχιστα επίπεδα. Ανάλογα με την παραγοντοποίηση που τους εφαρμόζεται οι αθροιστές ομαδοποιούνται σε τρεις ομάδες, Prefix, Ling και Jackson, και σε κάθε ομάδα θα αναπτυχθεί ένας 8-bit, ένας 16-bit, ένας 32-bit και ένας 64-bit αθροιστής. Για κάθε ομάδα θα αναλύεται και σχηματικά ο 8-bit αθροιστής λόγω του ευδιάκριτου σχήματος που τον περιγράφει, ενώ για τους υπόλοιπους θα δοθεί η συναρτησιακή λογική πλήρως σε άλγεβρα Μπουλ.

8.1 Prefix $2^n - 1$

8.2 Ling $2^n - 1$

8.3 Jackson $2^n - 1$

8.3.1 $2^8 - 1$



Εικόνα 8.1: Jackson 8-bit $2^n - 1$ Adder

Επίπεδο 1:

$$\begin{aligned} p_i &= a_i + b_i \\ g_i &= a_i * b_i \\ x_i &= a_i \oplus b_i \end{aligned} \tag{8.1}$$

Επίπεδο 2:

$$\begin{aligned} R_{i:i-1}^1 &= g_i + g_{i-1} \\ Q_{i:i-1}^1 &= p_i * p_{i-1} \end{aligned} \tag{8.2}$$

Επίπεδο 3:

$$\begin{aligned} R_{i:i-7}^3 &= R_{i:i-1}^1 + R_{i-2:i-3}^1 + Q_{i-3:i-4}^1 R_{i-4:i-5}^1 \\ &\quad + Q_{i-3:i-4}^1 Q_{i-5:i-6}^1 R_{i-6:i-7}^1 \end{aligned} \tag{8.3}$$

Group Generate:

$$G_{i:i-7} = D_{i:i-2} R_{i:i-7}^3 \tag{8.4}$$

Όπου :

$$\begin{aligned} D_{i:i-2} &= G_{i:i-1} + P_{i:i-2} \\ D_{i:i-2} &= g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2} \end{aligned} \tag{8.5}$$

Επίπεδο 5 - Sum computation:

$$sum_i = R_{i-1:i-8}^3 (x_i \oplus D_{i-1:i-3}) : x_i \tag{8.6}$$

Για παράδειγμα:

$$\begin{aligned} p_7 &= a_7 + b_7 \\ g_7 &= a_7 * b_7 \\ R_{7:6}^1 &= g_7 + g_6 \\ Q_{7:6}^1 &= p_7 * p_6 \\ R_{7:0}^3 &= R_{7:6}^1 + R_{5:4}^1 + Q_{4:3}^1 R_{3:2}^1 + Q_{4:3}^1 Q_{2:1}^1 R_{1:0}^1 \\ D_{7:5} &= g_7 + p_7 g_6 + p_7 p_6 p_5 \\ sum_7 &= R_{6:7}^3 * (a_7 \oplus b_7) + R_{6:7}^3 * (a_7 \oplus b_7 \oplus D_{6:4}) \end{aligned}$$

9 Αποτελέσματα

9.1 Μη-Τοπογραφικές Μετρήσεις

| | Delay | Area | Power |
|---------|-------|--------|-------|
| Prefix | 0.39 | 216.70 | 50.25 |
| Ling | 0.39 | 237.66 | 49.91 |
| Jackson | 0.35 | 281.51 | 58.69 |

Πίνακας 9.1: Μετρήσεις 8-bit

| | Delay | Area | Power |
|---------|-------|---------|--------|
| Prefix | 0.55 | 1319.29 | 265.71 |
| Ling | 0.58 | 1354.47 | 275.13 |
| Jackson | 0.48 | 1679.46 | 336.28 |

Πίνακας 9.3: Μετρήσεις 32-bit

| | Delay | Area | Power |
|---------|-------|--------|--------|
| Prefix | 0.50 | 573.10 | 116.68 |
| Ling | 0.48 | 590.69 | 119.84 |
| Jackson | 0.43 | 634.68 | 129.00 |

Πίνακας 9.2: Μετρήσεις 16-bit

| | Delay | Area | Power |
|---------|-------|---------|--------|
| Prefix | 0.67 | 3100.09 | 596.56 |
| Ling | 0.65 | 3170.45 | 612.25 |
| Jackson | 0.57 | 4049.84 | 766.75 |

Πίνακας 9.4: Μετρήσεις 64-bit

9.2 Μη-Τοπογραφικές Μετρήσεις sparse-2

| | Delay | Area | Power |
|---------|-------|--------|-------|
| Prefix | 0.44 | 173.73 | 40.2 |
| Ling | 0.47 | 186.01 | 43.6 |
| Jackson | 0.42 | 206.22 | 48.2 |

Πίνακας 9.5: Μη-Τοπογραφικές sparse-2 Μετρήσεις 8-bit

| | Delay | Area | Power |
|---------|-------|---------|--------|
| Prefix | 0.60 | 932.93 | 191.75 |
| Ling | 0.64 | 983.48 | 205.91 |
| Jackson | 0.53 | 1143.91 | 236.01 |

Πίνακας 9.7: Μη-Τοπογραφικές sparse-2 Μετρήσεις 32-bit

| | Delay | Area | Power |
|---------|-------|--------|--------|
| Prefix | 0.54 | 421.26 | 87.90 |
| Ling | 0.53 | 446.91 | 94.20 |
| Jackson | 0.48 | 496.23 | 103.34 |

Πίνακας 9.6: Μη-Τοπογραφικές sparse-2 Μετρήσεις 16-bit

| | Delay | Area | Power |
|---------|-------|---------|--------|
| Prefix | 0.72 | 2088.89 | 416.18 |
| Ling | 0.71 | 2191.47 | 442.85 |
| Jackson | 0.61 | 2619.22 | 518.93 |

Πίνακας 9.8: Μη-Τοπογραφικές sparse-2 Μετρήσεις 64-bit

9.3 Μη-Τοπογραφικές Μετρήσεις sparse-4

| | Delay | Area | Power |
|---------|-------|--------|-------|
| Prefix | 0.57 | 162.32 | 37.43 |
| Ling | 0.47 | 191.83 | 43.01 |
| Jackson | 0.40 | 228.68 | 47.46 |

Πίνακας 9.9: Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit

| | Delay | Area | Power |
|---------|-------|--------|--------|
| Prefix | 0.73 | 779.54 | 163.35 |
| Ling | 0.63 | 902.82 | 184.92 |
| Jackson | 0.52 | 983.48 | 198.59 |

Πίνακας 9.11: Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit

| | Delay | Area | Power |
|---------|-------|--------|-------|
| Prefix | 0.67 | 346.41 | 73.69 |
| Ling | 0.52 | 411.05 | 84.11 |
| Jackson | 0.48 | 434.99 | 88.50 |

Πίνακας 9.10: Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit

| | Delay | Area | Power |
|---------|-------|---------|--------|
| Prefix | 0.85 | 1587.56 | 326.94 |
| Ling | 0.70 | 1846.12 | 369.31 |
| Jackson | 0.62 | 2202.05 | 426.12 |

Πίνακας 9.12: Μη-Τοπογραφικές sparse-4 Μετρήσεις 64-bit

9.4 Τοπογραφικές Μετρήσεις

| | Delay | Area | Power |
|---------|-------|-------|-------|
| Prefix | 0.393 | 150.7 | 38 |
| Ling | 0.433 | 152.5 | 39 |
| Jackson | 0.427 | 156 | 37 |

Πίνακας 9.13: Τοπογραφικές Μετρήσεις 8-bit

| | Delay | Area | Power |
|---------|-------|-------|-------|
| Prefix | 0.667 | 949.8 | 252.9 |
| Ling | 0.747 | 917.2 | 245 |
| Jackson | 0.652 | 1109 | 299.6 |

Πίνακας 9.15: Τοπογραφικές Μετρήσεις 32-bit

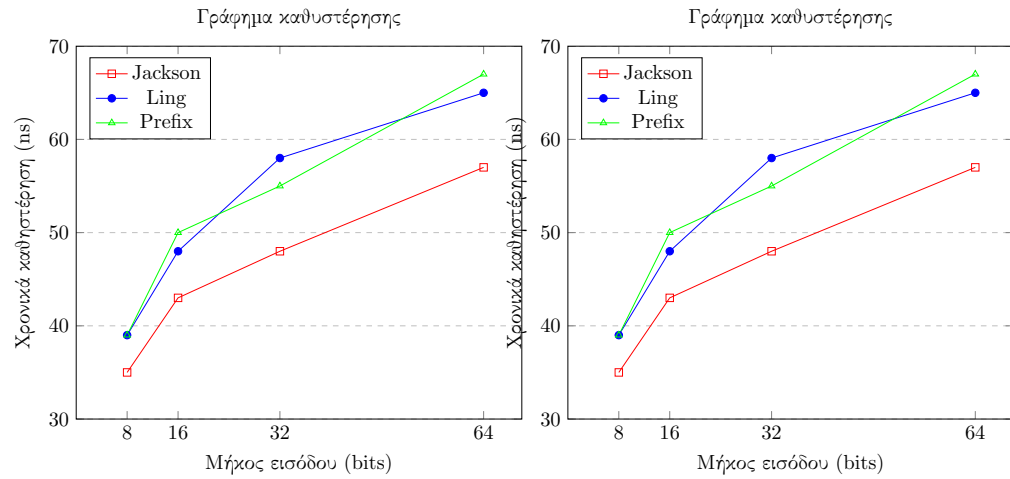
| | Delay | Area | Power |
|---------|-------|------|-------|
| Prefix | 0.589 | 384 | 97 |
| Ling | 0.597 | 397 | 103 |
| Jackson | 0.542 | 449 | 120 |

Πίνακας 9.14: Τοπογραφικές Μετρήσεις 16-bit

| | Delay | Area | Power |
|---------|-------|------|-------|
| Prefix | 0.726 | 2725 | 736 |
| Ling | 0.824 | 2349 | 625 |
| Jackson | 0.863 | 2161 | 550 |

Πίνακας 9.16: Τοπογραφικές Μετρήσεις 64-bit

9.5 Γραφικές



[Τις μονάδες μέτρησής πρέπει να διορθώσεις]

Βιβλιογραφία

- [1] Matthew Keeter, David Money Harris, Andrew Macrae, Rebecca Glick, Madaleine Ong, and Justin Schauer. Implementation of 32-bit ling and jackson adders. 2011.
- [2] Tynan McAuley, William Koven, Andrew Carter, Paul Ning, and David Money Harris. Implementation of 64-bit jackson adders. 2013.
- [3] Lampros Kalampoukas, Dimitris Nikolos, Costas Efstathiou, Haridimos T. Vergos, and John Kalamatianos. High-speed parallel-prefix modulo $2^n - 1$ adders. 2000.
- [4] Giorgos Dimitrakopoulos and Dimitris Nikolos. High-speed parallel-prefix vlsi ling adders. 2005.
- [5] H. Ling. High speed binary adder. 1981.
- [6] Robert Jackson and Sunil Talwar. High speed binary addition. 2004.
- [7] Neil Burgess. Implementation of recursive ling adders in cmos vlsi. 2009.
- [8] G.Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos, and C. Efstathiou. New architectures for module $2^n - 1$ adders.