Τμημα Μηχανικων Ηλεκτρονικων Υπολογιστων
και Πληροφορικης

Πανεπιστημιο Πατρας

June 5, 2018

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

Author
Βάσιλας Κωνσταντίνος

Supervisor
Βέργος Χαρίδημος

Περιεχόμενα

List of Figures

List of Tables

List of Equations

1 Εισαγωγη

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.

2 Πρόσθεση στο Δυαδικό

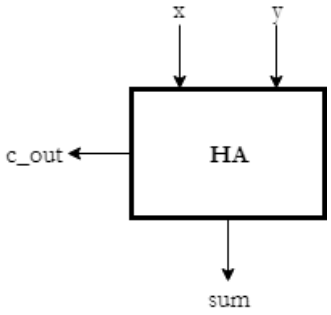
Οι αριθμητικές πράξεις σε αριθμούς εκφρασμένου σε μία συγκεκριμένη βάση ακολουθούν τους ίδιους κανόνες με αυτούς του δεκαδικού συστήματος, χρησιμοποιώντας σε κάθε περίπτωση μόνο τα διαθέσιμα ψηφία του κάθε συστήματος. Στα πλαίσια αυτής της διπλωματικής χρησιμοποιείται το δυαδικό σύστημα εφόσον κύριο θέμα της είναι η αρχιτεκτονική μελέτη με σκοπό την επιτάχυνση των δυαδικών αθροιστών υπολοίπου.

2.1 Πρόσθεση δύο ψηφίων

Στην πρόσθεση δύο διάδικων ψηφίων $x + y$ υπάρχουν τέσσερις πιθανές περιπτώσεις σύμφωνα με το συνδιασμό των τιμών που παίρνει το κάθε ψηφίο (0 ή 1). Στην περίπτωση $0 + 0$ το αποτέλεσμα είναι προφανώς μηδέν και ίδιο με εκείνο του δεκαδικού συστήματος. Οι περιπτώσεις $0 + 1$ και $1 + 0$ έχουν κοινό αποτέλεσμα, ίσο με ένα και αυτό οφείλεται στην προσεταιριστική ιδιότητα της πρόσθεσης, η οποία ισχύει σε όλα τα αριθμητικά συστήματα. Τελευταία περίπτωση είναι και η πιο περίπλοκη, όπου $1 + 1$ έχει άθροισμα μηδέν και ένα κρατούμενο. Στον πίνακα 2.1 παρουσιάζεται ο πίνακας αληθείας, δηλαδή ο πίνακας που περιγράφει την συμπεριφορά των εξόδων, στην περίπτωση αυτή το άθροισμα (sum) και το κρατούμενο (c_out), σύμφωνα με κάθε πιθανό συνδυασμό των εισόδων x και y.

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

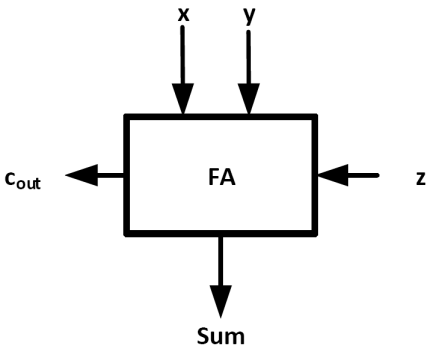
Το παραπάνω σύστημα ονομάζεται ημιαθροιστής ή Half-Adder (HA) , έχει δυο εισόδους τις x και y και δυο εξόδους τις c_{out} και sum και οι έξοδοι σύμφωνα με τον πίνακα αληθείας περιγράφονται από τις παρακάτω συναρτήσεις άλγεβρας Μπουλ :

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

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

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

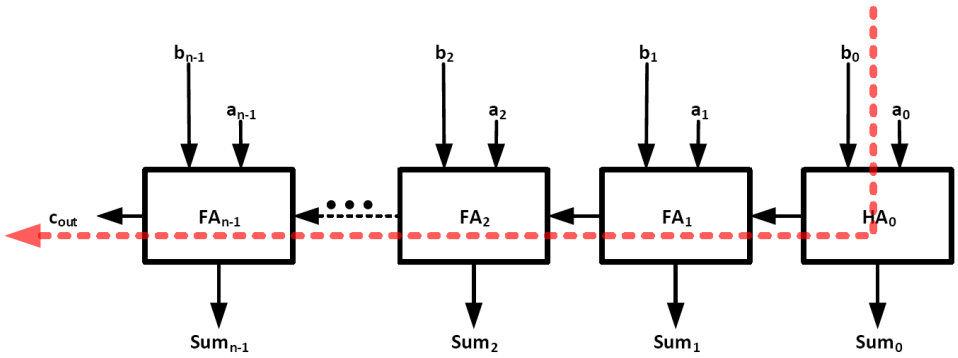
και οι εξισώσεις των εξόδων είναι :

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

2.2 Πρόσθεση δυαδικών αριθμών

Η πρόσθεση δυο δυαδικών αριθμών A και B των n δυαδικών ψηφίων είναι μια επέκταση της πρόσθεσης μεταξύ ψηφίων που παρουσιάστηκε προηγουμένως τροφοδοτώντας το κρατούμενο εξόδου των προηγούμενων σημαντικών ψηφίων στην είσοδο του πλήρη αθροιστή των επομένων. Συμβολίζοντας c_i το κρατούμενο που παράγεται από την πρόσθεση των ψηφίων $a_i + b_i$. Με αποτέλεσμα το κάθε ψηφίο αθροίσματος, εκτός του πρώτου, να υπολογίζεται από την λογική συνάρτηση:

$$\begin{aligned} sum_i &= a_i \oplus b_i \oplus c_{i-1} \\ sum_0 &= a_0 \oplus b_0 \end{aligned} \tag{2.3}$$



Εικόνα 2.3: Integer-Adder schematic

Ο αθροιστής στην εικόνα 2.3 αποτελεί τον πιο κλασικό σχεδιασμό με αρκετά μικρή επιφάνεια και κατανάλωση ενέργειάς αλλά πολύ υψηλούς χρόνους καθυστέρησης για τον υπολογισμό του αθροίσματος. Με κόκκινη γραμμή συμβολίζεται το κρίσιμο μονοπάτι.

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 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 χείριστη περίπτωση παρουσιάζεται όταν σε έναν ripple-carry όταν το propagate στοιχείο είναι αληθές για κάθε ζευγάρι ψηφίων (A_i B_i) . Ορίζοντας τον propagate όρο ως το XOR των A_i 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 .

$$\begin{aligned} p_i &= A_i \oplus B_i \\ select &= p_0 * p_1 * ... * p_{n-1} \\ c\_out &= select\ ?\ c\_in\ :\ c_{n-1} \end{aligned} \tag{3.2}$$

Η βελτιστοποίηση της χείριστης περίπτωσης επιτυγχάνεται με την χρήση πολλαπλών αθροιστών παράβλεψης κρατουμένου για να δομήσουν έναν block-carry-skip αθροιστή. Στην αντίθετή περίπτωση η καθυστερήσεις 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-Sellect 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 τακτικές βελτίωσης της άθροισης ο Αθροιστής Πρόβλεψης Κρατουμένου ή Carry-Lookahead Adder (CLA) βελτιώνει την ταχύτητα μειώνοντας τον χρόνο υπολογισμού κάθε ενδιάμεσου 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 ($y_{n-1}, y_{n-2}, ..., y_0$) , n εισόδους x ($x_{n-1}, x_{n-1}, ..., x_0$) και τον τελεστή $\circledast$. Κάθε έξοδος y υπολογίζεται με τον παρακάτω τρόπο :

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

Επίσης μπορούμε να το εκφράσουμε και αναδρομικά :

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

Ένα απλό παράδειγμα προβλημάτων που αντιμετωπίζονται ως προβλήματα προθέματος είναι η πρόσθεση πολλών αριθμών. Έστω πως έχουμε ένα σύνολο μεγέθους n από αριθμούς $(x_{n-1}, x_{n-2}, ..., x_1, x_0)$, σύμφωνα με τον ορισμό που δόθηκε παραπάνω χρειαζόμαστε ένα ακόμα σύνολο ίδιου μεγέθους $(y_{n-1}, y_{n-2}, ..., y_1, y_0)$ 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 κρατουμένου μπορούμε να το μετατρέψουμε σε prefix problem δημιουργώντας το ζεύγος (G, P) και αναθέτοντας στον τελεστή $\circledast$ την παρακάτω λειτουργιά :

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

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

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

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

$$\begin{aligned} c_0 &= g_0 + p_0 c_{-1} \\ c_0 &= g_0 \\ c_0 &= G_0 \end{aligned}$$

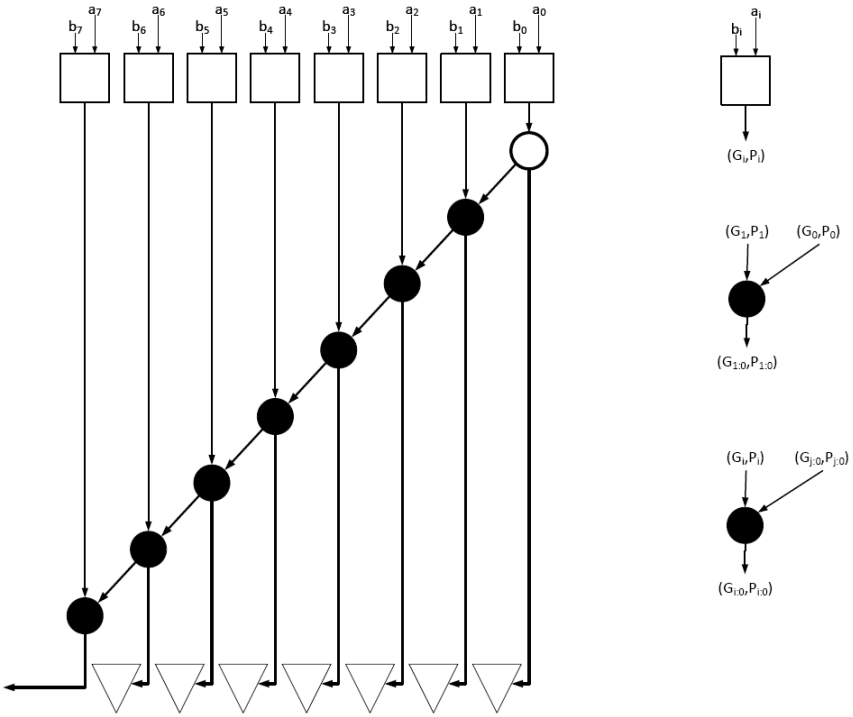
έστι το αποτέλεσμα ισχύει για $i - 1$
Αν $i > 0$ και $c_{i-1} = G_{i-1}$ τοτε

$$\begin{aligned} (G_i, P_i) &= (g_i, p_i) \circledast (G_{i-1}, P_{i-1}) \\ &= (g_i, p_i) \circledast (c_{i-1}, P_{i-1}) \\ &= (g_i + p_i c_{i-1}, p_i P_{i-1}) \\ G_i &= g_i + p_i c_{i-1} \\ G_i &= c_i \end{aligned}$$

Επίσης ο τελεστής $\circledast$ έχει προσεταιριστική ιδιότητα

$$\begin{aligned} (g_i, p_i) \circledast (g_j, p_j) \circledast (g_k, p_k) &= [g_i + p_i g_j, p_i p_j] \circledast (g_k, p_k) \\ &= (g_i, p_i) \circledast [g_j + p_j g_k, p_j p_k] \\ &= (g_i + p_i g_j + p_i p_j g_k, p_i p_j p_k) \end{aligned}$$

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



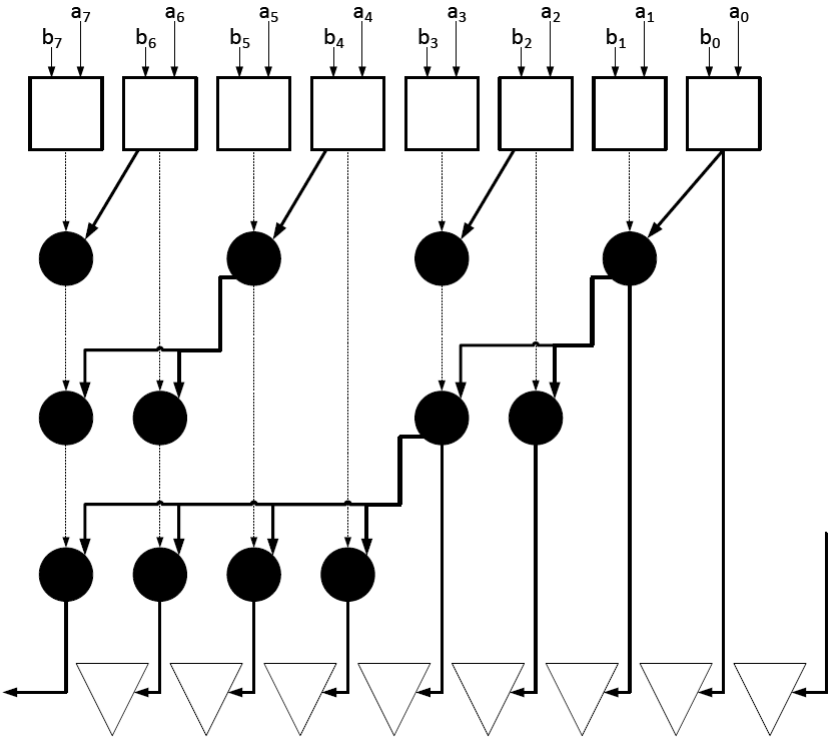
Εικόνα 4.1: Serial-Prefix Tree Adder

Σε κάθε μαύρο κόμβο ουσιαστικά υλοποιείται η λογική συνάρτηση του τελεστή $\circledast$ 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 Ladner-Fischer Adder

Ανελυσε Ladner-Fischer



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 κόμβου, το οποίο συνεπάγει το ότι απαιτείται αρκετή ενέργεια και εμβαδόν. Με σκοπό την αραίωση (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 στο κρατούμενο εισόδου ενός αθροιστή επιλογής κρατουμένου (CSA) των τεσσάρων δυαδικών ψηφίων με είσοδο τα δυαδικά ψηφία $7:0$ [DN05].

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

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

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 Grou 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)

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

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:

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

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 $i = k$:

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

Όσο αφορά την τροποποίηση των γραφημάτων για τον συνυπολογισμό κρατουμένου 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 $G_\iota + P_\iota * c_{-1}$.

[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 [Lin81] .

5.1 Βασική Θεωρία

Ξεκινώντας από την παρακάτω ισότητα

$$g_i = g_i * p_i \tag{5.1}$$

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 εως j με $i > j$

$$\begin{aligned} G_{i:j} &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + ... + p_i p_{i-1} p_{i-1} ... p_{j+1} p_j \\ &= p_i g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + ... + p_i p_{i-1} p_{i-1} ... p_{j+1} p_j \\ &= p_i \Big[g_i + g_{i-1} + p_{i-1} g_{i-2} + ... + p_{i-1} p_{i-1} ... p_{j+1} p_j \Big] \end{aligned} \tag{5.2}$$

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

$$H_{i:j} = g_i + G_{i-1:j} \tag{5.3}$$

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 :

$$G_{i:j} = p_i * H_{i:j} \tag{5.4}$$

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

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

Ακολουθεί ένα απλό παράδειγμα για καλύτερη εμπέδωση αλλά και επαλήθευση

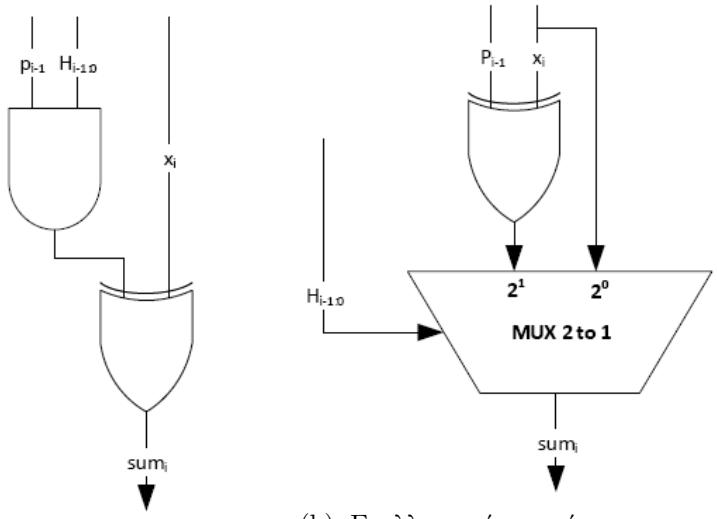
$$
\begin{aligned}
H_{7:2} &= H_{7:5} + P_{6:4} * H_{4:2} \\
H_{7:5} &= g_7 + g_6 + p_6 g_5 \\
H_{4:2} &= g_4 + g_3 + p_3 g_2 \\
P_{6:4} &= p_6 p_5 p_4 \\
H_{7:2} &= g_7 + g_6 + p_6 g_5 + p_6 p_5 p_4 * \big(g_4 + g_3 + p_3 g_2\big) \\
H_{7:2} &= g_7 + g_6 + p_6 g_5 + p_6 p_5 g_4 + p_6 p_5 p_4 g_3 + p_6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

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 $sum_i = G_{i-1:0} \oplus x_i$ οπου $x_i = a_i \oplus b_i$. 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 ενα 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 πράξη 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).



(a) Απλή προσέγγιση

(b) Εναλλακτική προσέγγιση με πολυπλέκτη

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.

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

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 μονοπάτι[1]. 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.

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

[1]Κρίσιμο 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

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

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 [JT04]. 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 αρχικής συνάρτησης.

$$\begin{aligned} G_{4:0} &= g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0 \\ &= \Big[p_4\Big]\Big[g_4 + g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0\Big] \\ &= \Big[g_4 + p_4 p_3\Big]\Big[g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0\Big] \\ &= \Big[g_4 + p_4 g_3 + p_4 p_3 p_2\Big]\Big[g_4 + g_3 + g_2 + g_1 + p_1 g_0\Big] \end{aligned} \tag{6.1}$$

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 σημάτων εισόδου από τα ψηφία a_i, b_i έως τα a_j, b_j παράγουν κρατούμενο εξόδου ή διαδίδουν το κρατούμενο εισόδου,

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

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

$$B_{i:j} = g_i + g_{i-1} + ... + g_j \tag{6.3}$$

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

$$G_{j:i} = D_{j:k}\Big[B_{j:k} + G_{k-1:i}\Big] \tag{6.4}$$

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

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

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

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

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

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

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

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

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

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

Οπότε η τελική εξίσωση είναι

$$R^{n-m}_{n-1:0} = R^{n-k}_{n-1:k} + R^{k-m}_{k-1:j} + [P_{m-1:j}D_{j-1:v}]R^{j-v}_{j-1:0} \tag{6.8}$$

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

$$Q^{n-m}_{n-1:k} = P_{n-1:m}D_{m-1:k} \tag{6.9}$$

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

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

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 v 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

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

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

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

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

$$Q^{n-m}_{n-1:0} = Q^{n-k}_{n-1:k}Q^{k-m}_{k-1:j}[R^{m-j}_{m-1:v} + Q^{j-v}_{j-1:0}] \tag{6.11}$$

6.3 Εφαρμογές της αναδρομής

Όπως προαναφέρθηκε παραπάνω υπάρχουν διάφορες υλοποιήσεις της αναδρομής ακόμα και με ίδιο αριθμό υποομάδων [Bur09] 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 (σθένος-2) και τέσσερα (σθένος-4). Υπάρχουν προφανώς και αντίστοιχες συναρτήσεις και για τρεις, πέντε και ούτω καθεξής, αλλά θα διατυπωθούν επιλεγμένα αυτά τα δύο σύνολα γιατί θα χρησιμοποιηθούν σε επόμενες ενότητες.

Λογικές συναρτήσεις Radix-2:

$$\begin{aligned} R^m_{i:0} &= R^m_{i:j+1} + Q^{i-m-j}_{i-m:j+1-m} * R^m_{j:0} \\ R^{i-j+m}_{i:0} &= R^m_{i:j+1} + R^m_{j:0} \end{aligned} \tag{6.12}$$

$$\begin{aligned} Q^m_{i:0} &= Q^m_{i:j+1} * (R^{i-m-j}_{i-m:j+1-m} + Q^m_{j:0}) \\ Q^{i-j+m} &= Q^m_{i:j+1} * Q^m_{j:0} \end{aligned} \tag{6.13}$$

Λογικές συναρτήσεις Radix-4:

$$\begin{aligned} R^m_{i:0} =& R^m_{i:j+1} + Q^{i-m-j}_{i-m:j+1-m} R^m_{j:k+1} + \\ & Q^{i-m-j}_{i-m:j+1-m} Q^{j-m-k}_{j-m:k+1-m} R^m_{k:l+1} \\ & Q^{i-m-j}_{i-m:j+1-m} Q^{j-m-k}_{j-m:k+1-m} Q^{k-m-l}_{k-m:l+1-m} R^m_{l:0} \\ R^{i-j+m}_{i:0} =& R^m_{i:j+1} + R^m_{j:k+1} + Q^{j-m-k}_{j-m:k+1-m} R^m_{k:l+1} \\ & Q^{j-m-k}_{j-m:k+1-m} Q^{k-m-l}_{k-m:l+1-m} R^m_{l:0} \\ R^{i-k+m}_{i:0} =& R^m_{i:j+1} + R^m_{j:k+1} + R^m_{k:l+1} + Q^{k-m-l}_{k-m:l+1-m} R^m_{l:0} \\ R^{i-l+m}_{i:0} =& R^m_{i:j+1} + R^m_{j:k+1} + R^m_{k:l+1} + R^m_{l:0} \end{aligned} \tag{6.14}$$

$$\begin{aligned} Q^m_{i:0} =& Q^m_{i:j+1}(R^{i-m-j}_{i-m:j+1-m} + Q^m_{j:k+1}(R^{j-m-k}_{j-m:k+1-m} + \\ & Q^m_{k:l+1}(R^{k-m-l}_{k-m:l+1-m} + Q^m_{l:0}))) \\ Q^{i-j+m}_{i:0} =& Q^m_{i:j+1} Q^m_{j:k+1}(R^{j-m-k}_{j-m:k+1-m} + \\ & Q^m_{k:l+1}(R^{k-m-l}_{k-m:l+1-m} + Q^m_{l:0})) \\ Q^{i-k+m}_{i:0} =& Q^m_{i:j+1} Q^m_{j:k+1} Q^m_{k:l+1}(R^{k-m-l}_{k-m:l+1-m} + Q^m_{l:0}) \\ Q^{i-l+m}_{i:0} =& Q^m_{i:j+1} Q^m_{j:k+1} Q^m_{k:l+1} Q^m_{l:0} \end{aligned} \tag{6.15}$$

Παρατηρείται πως στην περίπτωση των Radix-2 δεν υπάρχει διαφορά σε σχέση με τις αντίστοιχες συναρτήσεις Prefix και Ling. Η χρονική καθυστέρηση και

στις τρεις περιπτώσεις είναι ίδια, εκτός των συναρτήσεων του πρώτου επιπέδου όπου ο Ling και ο Jackson υπερτερούν. Επίσης, 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 βρίσκετε 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 FO4 [2] πριν από το σήμα R. Στον παρακάτω πίνακα 6.1 συγκρίνονται οι συναρτήσεις ενός απλού παράλληλου προθεματικού αθροιστή και ενός Jackson.

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

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

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 OAI[3]).

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]

[3] AND-OR-INVERTER και OR-AND-INVERTER

6.6 Αραίωση σε Jackson Αρχιτεκτονικές

Sparsness-2

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

Sparsness-4

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

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

$$X \ mod \ Y$$

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 δυαδική άθροιση είναι σε αριθμητική υπολοίπου και γράφεται $(A + B) \ mod \ n$ 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

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

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

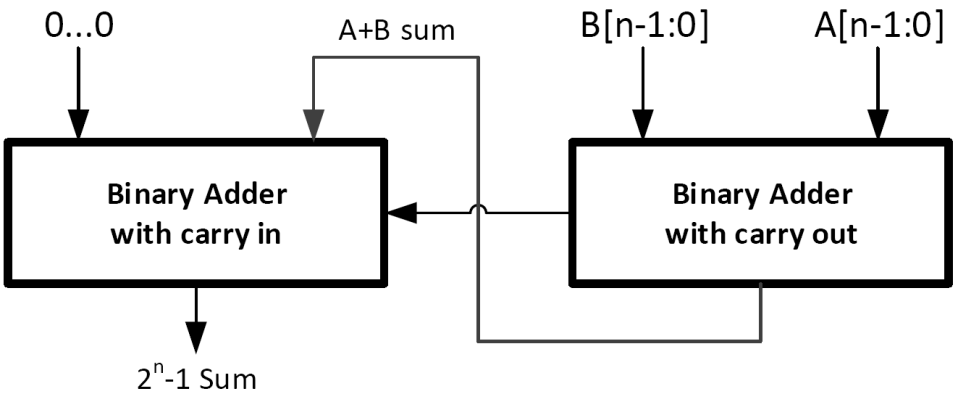
$$8 \ mod \ 7 = 1$$
$$7 \ mod \ 7 = 0$$
$$14 \ mod \ 7 = 0$$
$$6 \ mod \ 7 = 6$$
$$13 \ mod \ 7 = 6$$

Όπως είναι εμφανές και στα παραδείγματα ο μαθηματικός υπολογισμός που ορίστηκε φαίνεται να μην ισχύει για την περίπτωση $7 \ mod \ 7$ 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.

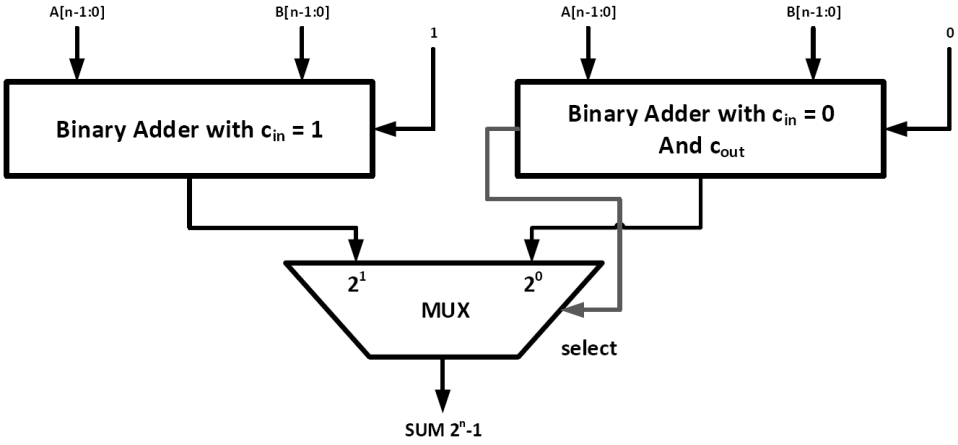
[Διπλή Αναπαράσταση του μηδέν, πως μπορούμε να την αντιμετωπίσουμε]

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.



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

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.



Εικόνα 7.2: Αρχιτεκτονική αθροιστή επιλογής κρατουμένου $2^n - 1$

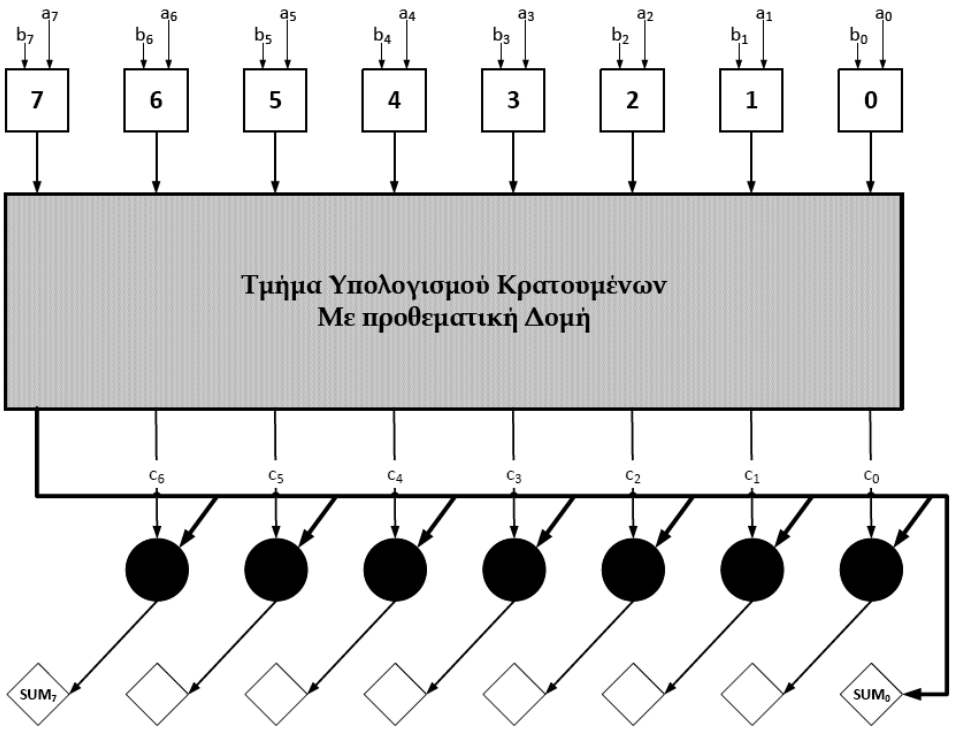
7.2 Prefix αθροιστές $2^n - 1$

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 επίπεδου $c'_{n-1} = G'_{n-1}$ (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

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



Εικόνα 7.3: 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 n 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 " $\circledast$ " ο αθροιστής υπολοίπου $2^n - 1$ μπορεί να υλοποιηθεί με ένα ακόμα παράλληλο τμήμα με αποτέλεσμα να μειωθεί κατά ένα επίπεδο ο υπολογισμός του [Kal+00]. Όπως εξηγήθηκε προηγουμένως το κρατούμενο εισόδου, στο δεύτερο επίπεδο, του αθροιστή υπολοίπου n 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

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

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

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

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

$$(G_i, P_i) \circledast (g, p) \circledast (G_i, P_i) = (G_i, P_i) \circledast (g, p) \tag{7.4}$$

Απόδειξη

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

Από την εξίσωση 7.3 αποδεικνύεται

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

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 $\circledast$. 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 :

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

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

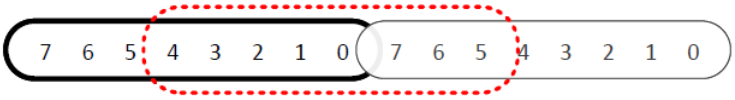
$$(G_i^*, P_i^*) = (G_{i:i+1}, P_{i:i+1}) \tag{7.7}$$

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 :

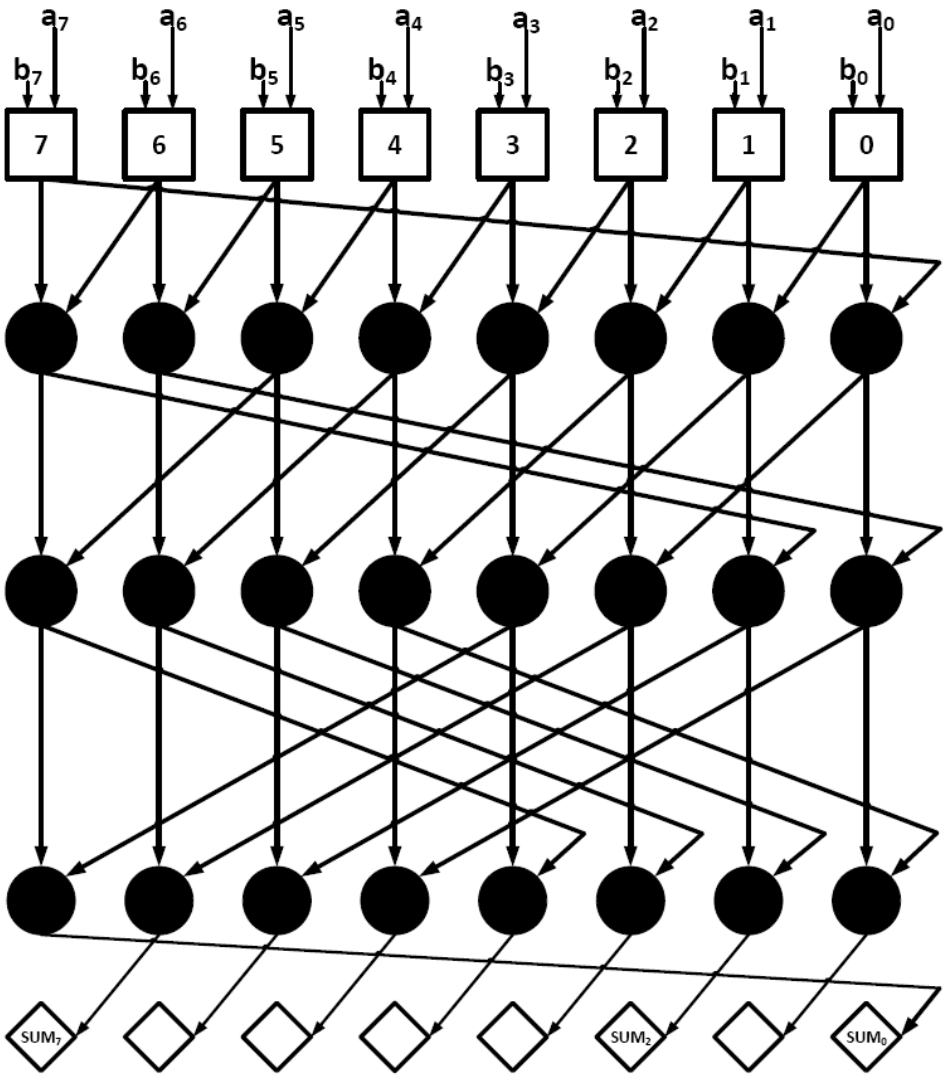
$$G_{1:7} = g_1 + p_1 g_0 + p_1 p_0 g_7 + p_1 p_0 p_7 g_6$$

Είναι σημαντικό να ξεκαθαριστεί πως αυτός ο τρόπος διατύπωσης είναι χρήσιμος στην περιγραφή μόνο αθροιστών υπολοίπου $2^n - 1$. Στις περιπτώσεις άλλων αθροιστών καθίσταται απαγορευμένη έκφραση.Για παράδειγμα στους αθροιστές ακεραίων με κρατούμενο εισόδου το προηγούμενο ζεύγος του (g_0, p_0) δεν είναι το (g_{n-1}, p_{n-1}) αλλά, όπως έχει προαναφερθεί και στα προηγούμενα κεφάλαια. το c_{-1}

Στην παρακάτω εικόνα εφαρμόζεται 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 $2^8 - 1$ αθροιστή υπολοίπου.



[Εξήγησε το παρακάτω παράδειγμα]



Εικόνα 7.4: 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 το είδος της παραγοντοποίησης 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. Κάθε αθροιστής που αναπτύσσεται περιγράφεται πλήρως από τις λογικές συναρτήσεις κάθε επιπέδου του.

8.1 Βασική δομή

Για την περιγραφή των κυκλωμάτων χρησιμοποιήθηκε η γλώσσα περιγραφής υλικού Verilog [IEE06], ενώ η προσομοίωση και ο έλεγχος ορθής περιγραφής και λειτουργίας υλοποιείται με την χρήση του εργαλείου VCS της Synopsys [Syn15]. Επίσης, για την καλύτερη βελτιστοποίηση επιλέχθηκε η περιγραφή των κυκλωμάτων να γινεί σε αρκετά χαμηλό επίπεδο. Κάθε κόμβος που αντιπροσωπεύει μια λογική συνάρτηση περιγράφηκε σε ξεχωριστή μονάδα. Όμως στις περιπτώσεις όπου η είσοδος των αθροιστών είναι μεγάλη θα έπρεπε να γίνουν αρκετές κλήσεις των verilog-modules στον κώδικα. Για παράδειγμα, με μήκος εισόδου 64 δυαδικά ψηφία, μόνο και μόνο οι μονάδες που υπολογίζουν τα σήματα generate και propagate καλούνται 64 φορές και δεν έχει ακόμα αρχίσει η περιγραφή του δέντρου, πράγμα που καθιστά την γραφή του κώδικα δύσκολη. Για την αποφυγή ανθρώπινων λαθών, τα οποία θα καθυστερούσαν πολύ την διαδικασία του ελέγχου, αναπτύχθηκε κώδικας σε python για την εκτύπωση της Verilog περιγραφής του κάθε αθροιστή.

Στην ενότητα 6 έγινε μια αναφορά σε σχεδιαστικούς κανόνες όσο αφορά τους αθροιστές Jackson. Για την ανάπτυξη ενός Jacson αθροιστή υπάρχει ένα μεγάλο πλήθος πιθανών υλοποιήσεων, όχι μόνο στην επιλογή του προθεματικού δέντρου και το πλήθος των όρων της παραγοντοποίησης (Radix-2, Radix-3, Radix-4 ...), κάτι που αφορά και τους αθροιστές Prefix και Ling, αλλά και στην επιλογή της συνάρτησης παραγοντοποίησης σε κάθε επίπεδο.

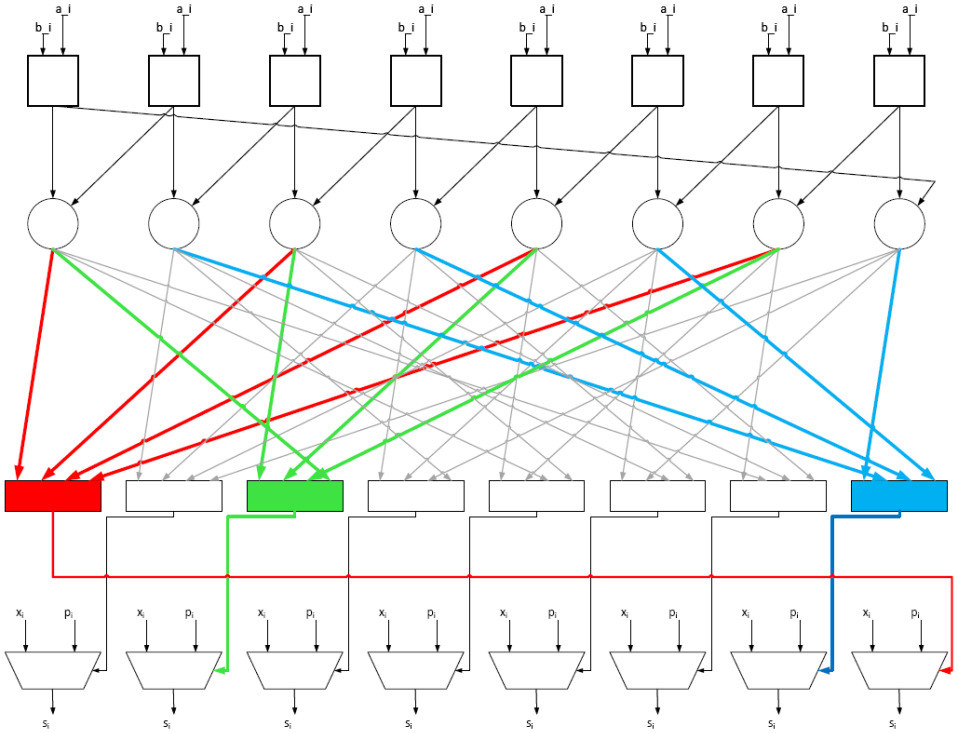
Όσο αφορά την επιλογή του πλήθους εισόδων στους κόμβους του κάθε επίπεδου αποφεύγεται η επιλογή του Radix-2 διότι, επαναδιατυπώνοντας, οι συναρτήσεις κατά Jacskon δεν διαφέρουν με αυτές ενός απλού Prefix. Θα ήταν αρκετά αποδοτική η υλοποίηση ορισμένων επιπέδων με Radix-3 , όπως και στους αθροιστές που αναπτύχθηκαν στα [Kee+11] και [McA+13], διότι στις περιπτώσεις περισσότερων των δύο εισόδων έχει εφαρμογή η παραγοντοποίηση και επιπλέον, εφόσον οι κόμβοι θα είναι των τριών εισόδων, αρά και οι συναρτήσεις πιο απλές, θα υπάρχουν μεγαλύτερες αντιστοιχίες των υλοποιημένων λογικών συναρτήσεων από τις CMOS βιβλιοθήκες. Στην περίπτωση των αθροιστών 2^n-1, με $n = 8, 16, 32$ και 64, δεν υπάρχει πολλαπλάσιο του τρία που δίνει τουλάχιστον έναν από αυτούς τους αριθμούς. Στην πραγματικότητα υπάρχει τρόπος αλλά το prefix-tree θα διαφέρει αρκετά με αυτό του Kogge-stone

και δεν θα υπάρχει το προνόμιο επαναχρησιμοποίησης κόμβων, με αποτέλεσμα η πολυπλοκότητα των αθροιστών καθώς η επιφάνεια και κατανάλωση να είναι αρκετά μεγαλύτερες. Σύμφωνα με αυτά που προειπώθηκαν στον πίνακα 8.1 παρουσιάζονται οι επιλογές που έγιναν, οι οποίες είναι κοινές για κάθε είδος αθροιστή ώστε τα αποτελέσματα των μετρήσεων να είναι έγκυρα.

Μήκος εισόδου	Αρχιτεκτονική
8-bit	2x4
16-bit	4x4
32-bit	2x4x4
64-bit	4x4x4

Πίνακας 8.1: Αρχιτεκτονική δομή των αθροιστών $2^n - 1$ προς υλοποίηση

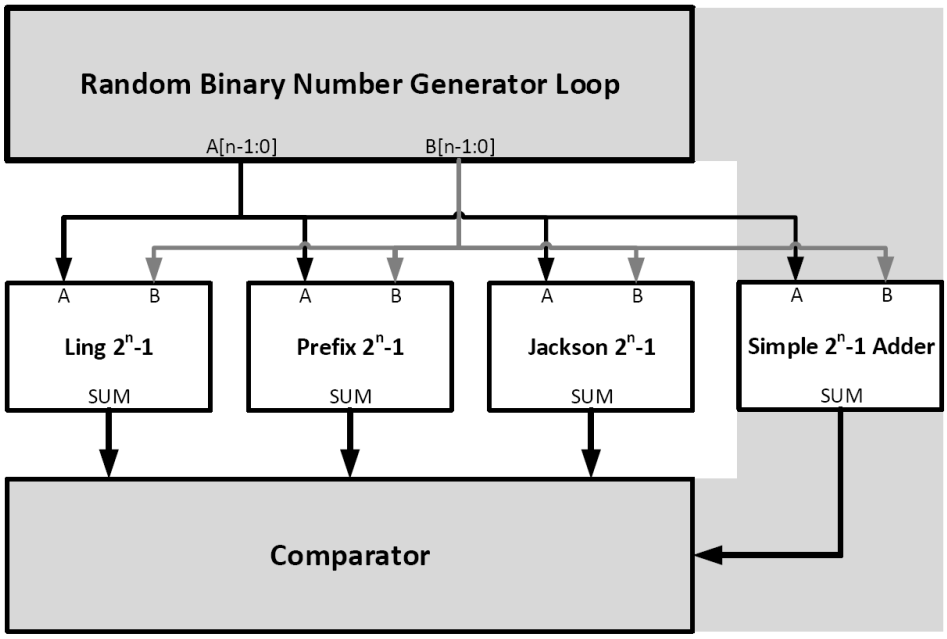
Στην εικόνα 8.1 παρουσιάζεται προσεγγιστικά η δομή των αθροιστών υπολοίπου $2^8 - 1$ που θα χρησιμοποιηθεί παρακάτω. Ενώ η δομή είναι κοινή και για τα τρία είδη αθροιστών το κάθε σχήμα αντιπροσωπεύει διαφορετική λογική συνάρτηση. Επίσης στο δέντρο αυτό ακολουθείται η κατασκευή μόνο των σημάτων που οδηγούν τον τελευταίο πολυπλέκτη, δηλαδή στην περίπτωση του Prefix το σήμα G, στου Ling το σήμα H και στο Jackson το R.



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

8.2 Διαδικασία ελέγχου ορθής λειτουργίας

Για τον έλεγχο ορθής λειτουργίας των αθροιστών που αναπτύχθηκαν κατασκευάστηκε ένας αθροιστής υπολοίπου $2^n - 1$ με την απλούστερη αρχιτεκτονική. Επίσης υλοποιήθηκε μια διαδικασία ελέγχου, η οποία τροφοδοτεί παράλληλα όλους τους αθροιστές με τυχαίους δυαδικούς αριθμούς και συγκρίνει την έξοδο τους με αυτήν των αθροιστών απλής δομής. Αυτός ο έλεγχος επαναλαμβάνεται αρκετές φορές, στην περίπτωση αυτή πραγματοποιήθηκαν 10000 επαναλήψεις, με τυχαίο σύνολο αριθμών κάθε φορά, για ισχυρότερη πιστοποίηση της ισχυριζόμενης λειτουργίας.



Εικόνα 8.2: Test-Bench

Στην εικόνα 8.2 παρουσιάζεται η αρχιτεκτονική του συστήματος ελέγχου. Στην μονάδα σύγκρισης (Comparator) υπάρχει ένας δείκτης PASS αρχικοποιημένος με την τιμή μηδέν. Σε κάθε επανάληψη αυξάνεται κατά μία μονάδα αν τα αποτελέσματα προς σύγκριση είναι ίδια. Στο τέλος των επαναλήψεων το ποσοστό επιτυχίας είναι ίσο με

$$SUCCESS = \frac{PASS}{\text{Number of Test-cycles}}$$

Όπως και σε κάθε περίπτωση σχεδιασμού υλικού πρέπει ο έλεγχος σε επίπεδο προσομοίωσης να έχει πλήρη επιτυχία, έτσι και σε αυτήν την περίπτωση ένας αθροιστής λειτουργεί σωστά όταν στο 100% των πιθανών διανυσμάτων εισόδου η έξοδος είναι η αναμενόμενη σύμφωνα με τον σχεδιασμό.

8.3 Αλγεβρική περιγραφή των αθροιστών

Η περιγραφή των απλών Prefix καθώς και των Ling αθροιστών υπολοίπου 2^n-1 εφαρμόζεται αλγεβρικά σε ένα πίνακα για το κάθε είδος παραγοντοποίησης εκτός των αθροιστών Jackson όπου παρουσιάζεται μία πιο αναλυτική, αλγεβρική επίσης, προσέγγιση. Στους παρακάτω πίνακες, στην δεξιά στήλη υποδεικνύεται το βάθος του επιπέδου στο προθεματικό δέντρο και συνοδεύεται από το πλήθος των εισόδων της παραγοντοποίησης που εφαρμόζεται στο επίπεδο αυτό (σε παρένθεση).

Με σκοπό την ευ-ανάγνωση των συναρτήσεων που θα διατυπωθούν παρακάτω, αναιρείται η παρακάτω συμβολική αναπαράσταση που δηλώθηκε σε προηγούμενα κεφάλαια.

$$X_i = X_{i:0}$$

όπου X ενα απο τα σήματα G,P,H,R και Q. Με σκοπό την χρήση του υπογεγραμμένου ως δείκτη, δίνοντάς την παρακάτω νέα ερμηνεία

$$X_i = X_{i:i-k+1}$$

όπου ο όρος k συμβολίζει το πλήθος των εισόδων που συμπεριλαμβάνει το σήμα X. Ο όρος k είναι ίσος με το γινόμενο των στοιχείων, στις παρενθέσεις, τις αριστερής στήλης, του επιπέδου αναφοράς και των προηγούμενων του. Για παράδειγμα στον πίνακα 8.2 το σήμα $G3_i$ του 64-bit αθροιστή, έχει $k = 4*4*4$, δηλαδή συμβολίζει το σήμα $G3_{i:i-k+1} = G_{i:i-63}$.

8.3.1 Prefix $2^n - 1$

level	P-G Equations
0	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
	8-bit
1 (x2)	$G1_i = g_i + p_i g_{i-1}$ $P1_i = p_i * p_{i-1}$
2 (x4)	$G2_i = G1_i + P1_i G1_{i-2} + P1_i P1_{i-2} G1_{i-4} +$ $P1_i P1_{i-2} P1_{i-4} G1_{i-6}$
	16-bit
1 (x4)	$G1_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-1} + p_i p_{i-1} p_{i-2} g_{i-1}$ $P1_i = p_i p_{i-1} p_{i-2} p_{i-3}$
2 (x4)	$G2_i = G1_i + P1_i G1_{i-4} + P1_i P1_{i-4} G1_{i-8} +$ $P1_i P1_{i-4} P1_{i-8} G1_{i-12}$
	32-bit
1 (x2)	$G1_i = g_i + p_i g_{i-1}$ $P1_i = p_i * p_{i-1}$
2 (x4)	$G2_i = G1_i + P1_i G1_{i-2} + P1_i P1_{i-2} G1_{i-4} +$ $P1_i P1_{i-2} P1_{i-4} G1_{i-6}$ $P2_i = P1_i P1_{i-2} P1_{i-4} P_{i-6}$
3 (x4)	$G3_i = G2_i + P2_i G2_{i-8} + P2_i P2_{i-8} G2_{i-16} +$ $P2_i P2_{i-8} P2_{i-16} G2_{i-24}$
	64-bit
1 (x4)	$G1_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-1} + p_i p_{i-1} p_{i-2} g_{i-1}$ $P1_i = p_i p_{i-1} p_{i-2} p_{i-3}$
2 (x4)	$G2_i = G1_i + P1_i G1_{i-4} + P1_i P1_{i-4} G1_{i-8} +$ $P1_i P1_{i-4} P1_{i-8} G1_{i-12}$ $P2_i = P1_i P1_{i-4} P1_{i-8} P1_{i-12}$
3 (x4)	$G3_i = G2_i + P2_i G2_{i-16} + P2_i P2_{i-16} G2_{i-32} +$ $P2_i P2_{i-16} P2_{i-32} G2_{i-48}$
SUM	$sum_i = G_{i-1} \oplus x_i$

Πίνακας 8.2: Prefix $2^n - 1$ Equations

8.3.2 Ling $2^n - 1$

level	H-P Equations
0	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
	8-bit
1 (x2)	$H1_i = g_i + g_{i-1}$ $P1_i = p_i * p_{i-1}$
2 (x4)	$H2_i = H1_i + P1_{i-1}H1_{i-2} + P1_{i-1}P1_{i-3}H1_{i-4} +$ $P1_{i-1}P1_{i-3}P1_{i-5}H1_{i-6}$
	16-bit
1 (x4)	$H1_i = g_i + g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3}$ $P1_i = p_i p_{i-1} p_{i-2} p_{i-3}$
2 (x4)	$H2_i = H1_i + P1_{i-1}H1_{i-4} + P1_{i-1}P1_{i-5}H1_{i-8} +$ $P1_{i-1}P1_{i-5}P1_{i-9}H1_{i-12}$
	32-bit
1 (x2)	$H1_i = g_i + g_{i-1}$ $P1_i = p_i * p_{i-1}$
2 (x4)	$H2_i = H1_i + P1_{i-1}H1_{i-2} + P1_{i-1}P1_{i-3}H1_{i-4} +$ $P1_{i-1}P1_{i-3}P1_{i-5}H1_{i-6}$ $P2_i = P1_i P1_{i-2} P1_{i-4} P1_{i-6}$
3 (x4)	$H3_i = H2_i + P2_{i-1}H2_{i-8} + P2_{i-1}P2_{i-9}H2_{i-16} +$ $P2_{i-1}P2_{i-9}P2_{i-17}H2_{i-24}$
	64-bit
1 (x4)	$H1_i = g_i + g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3}$ $P1_i = p_i p_{i-1} p_{i-2} p_{i-3}$
2 (x4)	$H2_i = H1_i + P1_{i-1}H1_{i-4} + P1_{i-1}P1_{i-5}H1_{i-8} +$ $P1_{i-1}P1_{i-5}P1_{i-9}H1_{i-12}$ $P2_i = P1_i P1_{i-4} P1_{i-8} P1_{i-12}$
3 (x4)	$H3_i = H2_i + P2_{i-1}H2_{i-16} + P2_{i-1}P2_{i-17}H2_{i-32} +$ $P2_{i-1}P2_{i-17}P2_{i-33}H2_{i-48}$
SUM	$sum_i = H_{i-1}\ ?\ (x_i \oplus p_{i-1})\ :\ x_i$

Πίνακας 8.3: Ling $2^n - 1$ Equations

8.3.3 Jackson $2^n - 1$

Στους Jackson αθροιστές υπολοίπου $2^n - 1$ η αλγεβρική περιγραφή γίνεται με αναλυτικό τρόπο εφόσον αποτελούν το κύριο κομμάτι αυτής της αναφοράς. Συγκεκριμένα σε κάθε πίνακα εκτός από την συναρτησιακή περιγραφή δίνεται και η συμβολική ερμηνεία, έτσι ώστε ο αναγνώστης να είναι σε θέση να

αναπαράγει τους αθροιστές επικαλούμενος και τον εξισώσεων της παραγράφου 6.4.

level	R-Q Equations
0	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
1 (x2)	$R1_i = g_i + g_{i-1}$ $Q1_i = p_i * p_{i-1}$
2 (x4)	$R2_i = R1_i + R1_{i-2} + Q1_{i-3} * R1_{i-4} + Q1_{i-3} * Q1_{i-5} * R1_{i-6}$
D	$D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
SUM	$sum_i = R2_{i-1}\ ?\ (x_i \oplus D_{i-1})\ :\ x_i$

Symbols	$R1_i$	$Q1_i$	$R2_i$	D_i
Equations	$R^1_{i:i-1}$	$Q^1_{i:i-1}$	$R^3_{i:i-7}$	$D_{i:i-2}$

Πίνακας 8.4: Jackson $2^8 - 1$ Equations

level	R-Q Equations
0	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
1 (x4)	$R1_i = g_i + g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3}$ $Q1_i = p_i p_{i-1} p_{i-2} p_{i-3}$
2 (x4)	$R2_i = R1_i + R1_{i-4} + Q1_{i-5} * R1_{i-8} + Q1_{i-5} * Q1_{i-9} * R1_{i-12}$
D	$D_i = p_i R1_i + p_{i-1} Q1_i$
SUM	$sum_i = R2_{i-1}\ ?\ (x_i \oplus D_{i-1})\ :\ x_i$

Symbols	$R1_i$	$Q1_i$	$R2_i$	D_i
Equations	$R^1_{i:i-3}$	$Q^3_{i:i-3}$	$R^5_{i:i-15}$	$D_{i:i-4}$

Πίνακας 8.5: Jackson $2^{16} - 1$ Equations

level	R-Q Equations
0	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
1 (x2)	$R1_i = g_i + g_{i-1}$ $Q1_i = p_i * p_{i-1}$
2 (x4)	$R2_i = R1_i + R1_{i-2} + Q1_{i-3} * R1_{i-4} + Q1_{i-3} * Q1_{i-5} * R1_{i-6}$ $Q2_i = Q1_i Q1_{i-2} Q1_{i-4}(R1_{i-5} + Q1_{i-6})$
3 (x4)	$R3_i = R2_i + R2_{i-8} + Q2_{i-11} * R2_{i-16} + Q2_{i-11} * Q2_{i-17} * R2_{i-24}$
D	$D1_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$ $D_i = D1_i(R2_i + Q2_{i-3})$
SUM	$sum_i = R2_{i-1}\ ?\ (x_i \oplus D_{i-1})\ :\ x_i$

Symbols	$R1_i$	$Q1_i$	$R2_i$	$Q2_i$	$R3_i$	D_i
Equations	$R^1_{i:i-1}$	$Q^1_{i:i-1}$	$R^3_{i:i-7}$	$Q^5_{i:i-7}$	$R^{11}_{i:i-31}$	$D_{i:i-10}$

Πίνακας 8.6: Jackson $2^{32} - 1$ Equations

level	R-Q Equations
0	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
1 (x4)	$R1_i = g_i + g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3}$ $Q1_i = p_i p_{i-1} p_{i-2} p_{i-3}$
2 (x4)	$R2_i = R1_i + R1_{i-4} + Q1_{i-5} * R1_{i-8} + Q1_{i-5} * Q1_{i-9} * R1_{i-12}$ $Q2_i = Q1_i Q1_{i-4} Q1_{i-8}(R1_{i-11} + Q1_{i-12})$
3 (x4)	$R3_i = R2_i + R2_{i-16} + Q2_{i-21} * R2_{i-32} + Q2_{i-21} * Q2_{i-37} * R3_{i-48}$
D	$D1_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$ $D2_i = D1_i(R1_i + Q1_{i-3})$ $D_i = D2_i(R2_i + Q2_{i-5})$
SUM	$sum_i = R2_{i-1}\ ?\ (x_i \oplus D_{i-1})\ :\ x_i$

Symbols	$R1_i$	$Q1_i$	$R2_i$	$Q2_i$	$R3_i$	D_i
Equations	$R^1_{i:i-3}$	$Q^3_{i:i-3}$	$R^5_{i:i-15}$	$Q^{11}_{i:i-15}$	$R^{21}_{i:i-63}$	$D_{i:i-20}$

Πίνακας 8.7: Jackson $2^{64} - 1$ Equations

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	0.50	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	0.55	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	0.67	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	0.54	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	0.60	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	0.72	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 8-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 16-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 32-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	0.589	384	97
Ling	0.597	397	103
Jackson	0.542	449	120

Πίνακας 9.14: Τοπογραφικές Μετρήσεις 16-bit

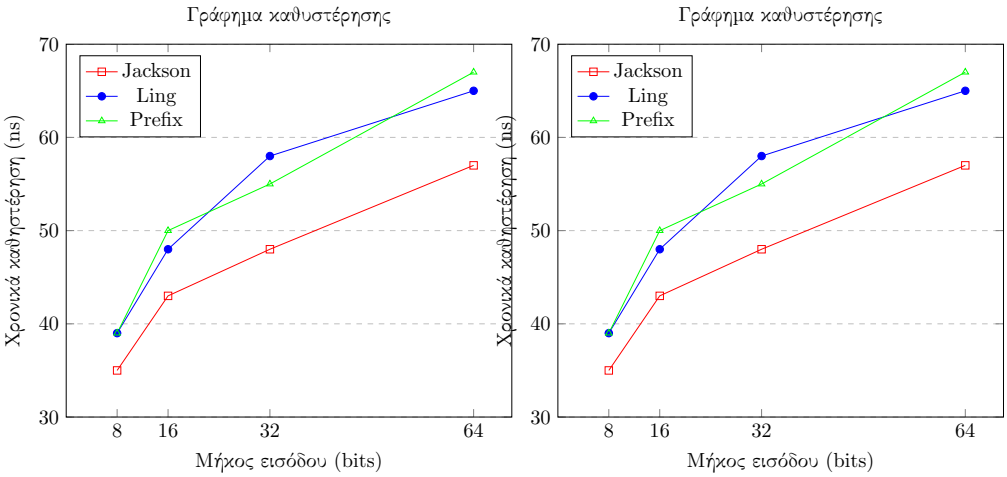
	Delay	Area	Power
Prefix	0.667	949.8	252.9
Ling	0.747	917.2	245
Jackson	0.652	1109	299.6

Πίνακας 9.15: Τοπογραφικές Μετρήσεις 32-bit

	Delay	Area	Power
Prefix	0.726	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

[Lin81] H. Ling. "High-Speed Binary Adder". In: *IBM Journal of Research and Development* 25.3 (Mar. 1981), pp. 156–166. ISSN: 0018-8646. DOI: 10.1147/rd.252.0156.

[Kal+00] L. Kalampoukas et al. "High-speed parallel-prefix module 2n-1 adders". In: *IEEE Transactions on Computers* 49.7 (July 2000), pp. 673–680. ISSN: 0018-9340. DOI: 10.1109/12.863036.

[JT04] R. Jackson and S. Talwar. "High speed binary addition". In: *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.* Vol. 2. Nov. 2004, 1350–1353 Vol.2. DOI: 10.1109/ACSSC.2004.1399373.

[DN05] Giorgos Dimitrakopoulos and Dimitris Nikolos. "High-Speed Parallel-Prefix VLSI Ling Adders". In: (2005).

[IEE06] IEEE. "IEEE Standard for Verilog Hardware Description Language". In: *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)* (2006), pp. 0-560. DOI: 10.1109/IEEESTD.2006.99495.

[Bur09] N. Burgess. "Implementation of recursive Ling adders in CMOS VLSI". In: *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers.* Nov. 2009, pp. 1777–1781. DOI: 10.1109/ACSSC.2009.5470204.

[Kee+11] M. Keeter et al. "Implementation of 32-bit Ling and Jackson adders". In: *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR).* Nov. 2011, pp. 170–175. DOI: 10.1109/ACSSC.2011.6189978.

[McA+13] T. McAuley et al. "Implementation of a 64-bit Jackson adder". In: *2013 Asilomar Conference on Signals, Systems and Computers.* Nov. 2013, pp. 1149–1154. DOI: 10.1109/ACSSC.2013.6810474.

[Syn15] Synopsys. *VCS User Guide.* 2015.