Τμημα Μηχανικων Ηλεκτρονικων Υπολογιστων
και Πληροφορικης

Πανεπιστημιο Πατρας

May 24, 2018

Recursive Ling 2^{n-1} Binary Adder

Author
Βάσιλας Κωνσταντίνος

Supervisor
Βέργος Χαρίδημος

Περιεχόμενα

List of Figures

List of Tables

1 Εισαγωγη

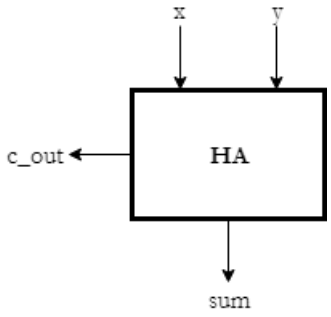
Οι αριθμητικές μονάδες είναι βασικά στοιχεία στο σύνολο του υλικού των ηλεκτρονικών υπολογιστών . Οι περισσότερες επεξεργαστικές μονάδες έχουν ως βασικό δομικό στοιχείο την Αριθμητική Λογική Μονάδα (ΑΛΜ) ή Arithmetic Logic Unit (ALU) η οποία είναι υπεύθυνη για την υλοποίηση βασικών αριθμητικών πράξεων 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 [1] [2] [3] [4].

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 1: Half Adder Truth Table



Εικόνα 1: Half-Adder schematic

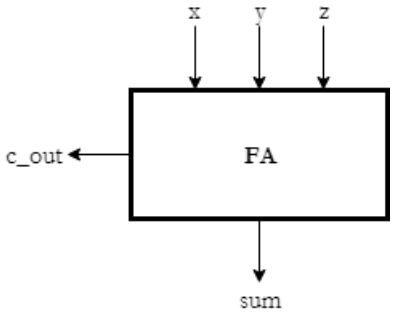
Το παραπάνω σύστημα ονομάζεται ημιαθροιστής ή Half-Adder (HA) , έχει δυο εισόδους τις x και y και δυο εξόδους τις c_out και sum και οι έξοδοι σύμφωνα με τον παραπάνω πίνακα αληθείας περιγράφονται από τις παρακάτω συναρτήσεις άλγεβρας Μπουλ :

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

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

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: Full Adder Truth Table



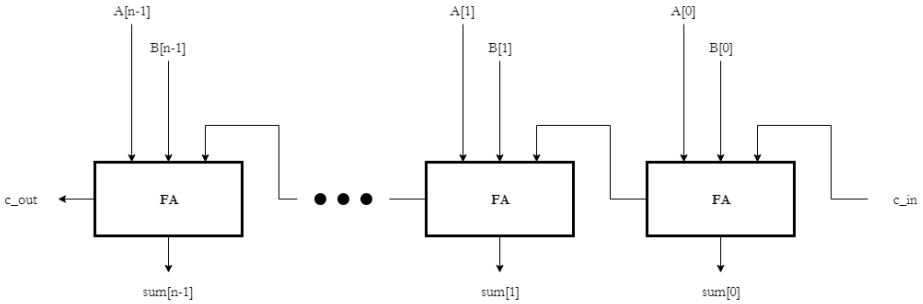
Εικόνα 2: Full-Adder schematic

και οι εξισώσεις των εξόδων είναι :

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

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 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 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 ελήθες 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]) ενός απλού διάδοσης κρατουμένου αθροιστή σε αναγόμενο σε πρόβλημα προθέματος.

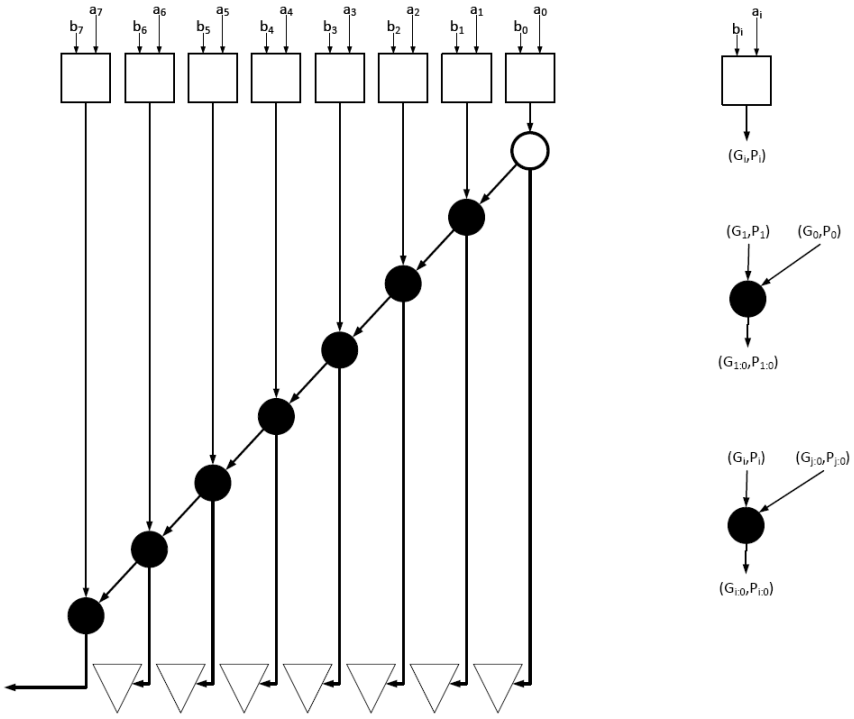
Σε κάθε μαύρο κόμβο ουσιαστικά υλοποιείται η λογική συνάρτηση του τελεστή $\circledast$ που παρουσιάστηκε προηγουμένως. Ο παραπάνω αθροιστής υλοποιεί τον προθεματικό αλγόριθμο σειριακά με αποτέλεσμα να είναι πολύ αργό το μοντέλο αλλά να καταλαμβάνει μικρότερο εμβαδόν από άλλες τοπολογίες αθροιστών προθέματος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

4.2 Παράλληλοι Προθεματικοί Αθροιστές

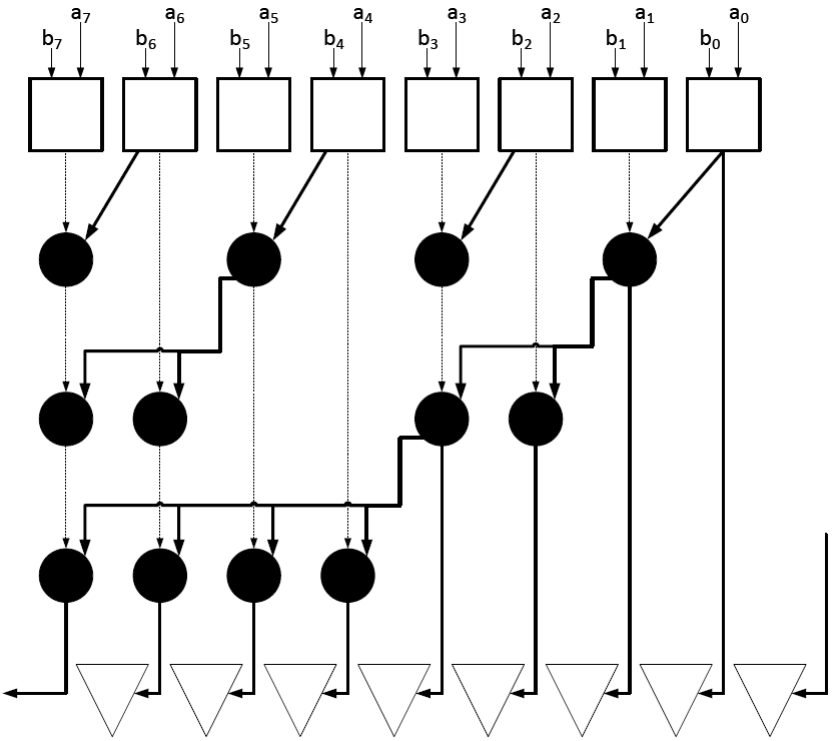
4.2.1 Brent-Kung Adder

4.2.2 Ladner-Fischer Adder

4.3 Δέντρα-Δομές Προθεμάτων



Εικόνα 4: Serial-Prefix Tree Adder



Εικόνα 5: Ladner-Fischer Prefix Tree Adder

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 απο την παρακάτω ισσοτητα

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

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 υπολογιστέι το H στην συνέχεια να πραγματοποιηθεί η λογική του πράξη AND με το σήμα p_i και τελος η έξοδος της πυλης AND να οδηγήσει την είσοδο της πύλης XOR σε συνδιασμό με το σήμα x_i.

[Figure with logic diagram of whats described above

Η εξίσωση 5.6 αυτή μπορεί να εκφραστεί διαφορετικά βοηθώντας στην μείωση της λογικής που απαιτείται για τον υπολογισμό του αθροίσματος.

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

[Figure with logic diagram of whats described above

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

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 αρχικής συνάρτησης.

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

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.

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

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

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

$$R^{n-m}_{n-1:0} = R^{n-k}_{n-1:k} + R^{k-m}_{k-1:j} + [P_{m-1:j}D_{j-1:v}]R^{j-v}_{j-1:0} \tag{6.7}$$

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

$$Q^{n-m}_{n-1:k} = P_{n-1:m}D_{m-1:k} \tag{6.8}$$

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

$$\begin{aligned} D_{j:i} &= D_{j:k}[B_{j:k} + D_{k-1:i}] \\ D_{j:i} &= G_{j:k} + P_{j:k}D_{k-1:i} \end{aligned} \tag{6.9}$$

Με την ιδια 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, ενω 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

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

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

$$Q_{n-1:0}^{n-m} = Q_{n-1:k}^{n-k}Q_{k-1:j}^{k-m}[R_{m-1:v}^{m-j} + Q_{j-1:0}^{j-v}] \tag{6.10}$$

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 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]

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 διαδική 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

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

[Γιατί είναι αυτός ο ορισμός ?
Γράψε την εξήγηση]

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 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 λιγότερο ο υπολογισμός του.

8 Ανάπτυξη Αθροιστών υπολοίπου $2^n - 1$

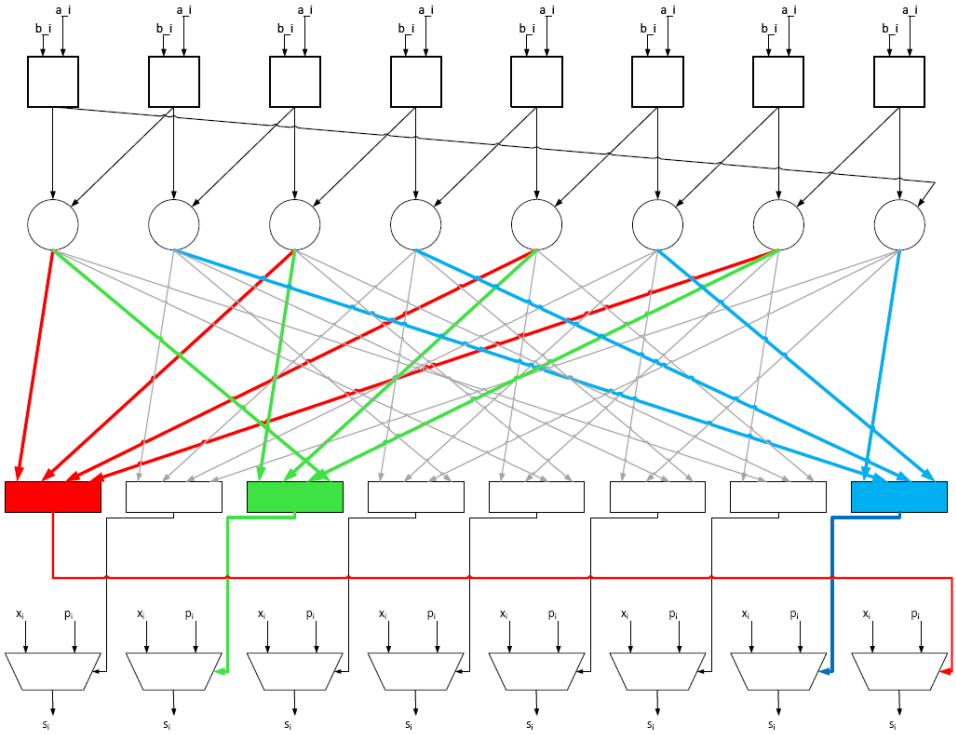
Σε αυτό το κεφάλαιο θα αναπτυχθούν συνολικά δώδεκα αθροιστές υπολοίπου $2^n - 1$ ακολουθώντας την αρχιτεκτονική που παρουσιάστηκε στο προηγούμενο κεφάλαιο με τα ελάχιστα επίπεδα. Οι αθροιστές θα χωριστούς σε τρείς ομάδες ανάλογα με την παραγοντοποίηση που τους εφαρμόζεται. Τις ομάδες αυτές είναι Prefix, 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 ο 8-bit αθροιστής λόγω των ευδιάκριτου σχήματος που τον περιγράφει, ενώ για τους υπόλοιπους θα δωθεί πλήρη η συναρτησιακή λογική 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 6: 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:

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

Επίπεδο 3:

$$\begin{aligned} R^3_{i:i-7} =& R^1_{i:i-1} + R^1_{i-2:i-3} + Q^1_{i-3:i-4} R^1_{i-4:i-5} \\ &+ Q^1_{i-3:i-4} Q^1_{i-5:i-6} R^1_{i-6:i-7} \end{aligned} \tag{8.3}$$

Group Generate:

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

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:

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

Για παράδειγμα:

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

---

9 Αποτελέσματα

9.1 Μετρήσεις

Αρχιτεκτονική	Max 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 3: Μετρήσεις 8-bit

Αρχιτεκτονική	Max 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 4: Μετρήσεις 16-bit

Αρχιτεκτονική	Max 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 5: Μετρήσεις 32-bit

Αρχιτεκτονική	Max 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 6: Μετρήσεις 64-bit

10 Αποδειξεις

10.1 Jackson recursion

$$
\begin{aligned}
G_{4:0} &= g_4 + p_4g_3 + p_4p_3g_2 + p_4p_3p_2g_1 + p_4p_3p_2p_1g_0 \\
&= p_4\Big[g_4 + g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0\Big] \\
&= p_4\Big[g_4 + p_3\Big[g_3 + g_2 + p_2g_1 + p_2p_1g_0\Big]\Big] \\
&= p_4g_4 + p_4p_3\Big[g_3 + g_2 + p_2g_1 + p_2p_1g_0\Big] \\
&= g_4 + p_4p_3\Big[g_3 + g_2 + p_2g_1 + p_2p_1g_0\Big] \\
&\text{Note} : x + y * z = (x + y) * (x + z) \\
&= \Big[g_4 + p_4p_3\Big]\Big[g_4 + g_3 + g_2 + p_2g_1 + p_2p_1g_0\Big] \\
&= p_4\Big[g_4 + p_3\Big[g_3 + g_2 + p_2g_1 + p_2p_1g_0\Big]\Big] \\
&= p_4\Big[g_4 + p_3\Big[g_3 + p_2(g_2 + g_1 + p_1g_0)\Big]\Big] \\
&= p_4\Big[g_4 + p_3g_3 + p_3p_2\Big[g_2 + g_1 + p_1g_0\Big]\Big] \\
&= p_4\Big[g_4 + \Big[p_3g_3 + p_3p_2\Big]\Big[p_3g_3 + g_2 + g_1 + p_1g_0\Big]\Big] \\
&= p_4g_4 + p_4\Big[g_3 + p_3p_2\Big]\Big[g_3 + g_2 + g_1 + p_1g_0\Big] \\
&= g_4 + \Big[p_4g_3 + p_4p_3p_2\Big]\Big[g_3 + g_2 + g_1 + p_1g_0\Big] \\
&= \Big[g_4 + p_4g_3 + p_4p_3p_2\Big]\Big[g_4 + g_3 + g_2 + g_1 + p_1g_0\Big]
\end{aligned}
\tag{10.1}
$$

10.2 Jackson D

$$\begin{aligned}
D_{i:k} &= G_{j:k} + P_{j:k} \\
&= G_{j:k+1} + P_{j:k-1}g_k + P_{j:k} \\
&= G_{j:k+1} + P_{j:k-1}g_k + P_{j:k-1}p_k \\
&= G_{j:k+1} + P_{j:k-1}(g_k + p_k) \\
&= G_{j:k+1} + P_{j:k-1}(a_k b_k + a_k + b_k) \\
&= G_{j:k+1} + P_{j:k-1}(a_k(b_k + 1) + b_k) \\
&= G_{j:k+1} + P_{j:k-1}(a_k + b_k) \\
&= G_{j:k+1} + P_{j:k-1}p_k \\
&= G_{j:k+1} + P_{j:k}
\end{aligned} \tag{10.2}$$

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] G.Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos, and C. Efstathiou. New architectures for module $2^n - 1$ adders.

[4] Lampros Kalampoukas, Dimitris Nikolos, Costas Efstathiou, Haridimos T. Vergos, and John Kalamatianos. High-speed parallel-prefix modulo $2^n - 1$ adders. 2000.

[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.