ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΑΣ

July 4, 2018

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

Βάσιλας Κωνσταντίνος

Επιβλέπον καθηγητής
Βέργος Χαρίδημος

Περίληψη

Η πρόσθεση είναι η βασικότερη αριθμητική πράξη, αποτελώντας τμήμα των περισσότερων ψηφιακών συστημάτων. Επίσης, είναι η πιο συχνή πράξη που εκτελείται από μία επεξεργαστική μονάδα. Η επιτάχυνση της απασχολεί την επιστήμη των ψηφιακών κυκλωμάτων για αρκετά χρόνια. Στα πλαίσια αυτής της εργασίας παρουσιάζεται μια νέα υλοποίηση αθροιστών υπολοίπου $2^n - 1$. Τεχνικές και αρχιτεκτονικές που έχουν προταθεί στο παρελθόν, είτε αφορούν άμεσα τους αθροιστές υπολοίπου, είτε γενικότερα τους αθροιστές, συνδυάζονται για την ανάπτυξη του νέου σχεδιασμού. Συγκεκριμένα και περιεκτικά, έγινε χρήση των μοντέλων κατά Jackson και Talwar, που αφορούν την γενίκευση της παραγοντοποίησης που συστήθηκε από τον H. Ling. Εφαρμόζοντας αυτές τις λογικές συναρτήσεις σε μία τοπολογία που πλεονεκτεί σε προθεματικά επίπεδα, επιτεύχθηκε και το τελικό αποτέλεσμα. Οι αθροιστές που αναπτύχθηκαν συγκρίθηκαν με τις αποδοτικότερες τοπολογίες όσο άφορα την ταχύτητα, το εμβαδόν και την κατανάλωση ενέργειας, και τα αποτελέσματα είναι αρκετά υποσχόμενα. Ειδικότερα σε θεωρητικές προδιαγραφές η καθυστέρηση των νέων αθροιστών είναι αισθητά μικρότερη, ενώ σε πρακτικές συνθήκες οι αποδόσεις είναι θετικές σε ορισμένες περιπτώσεις και εξαρτώμενες της τεχνολογίας υλοποίησης.

Abstract

Η πρόσθεση είναι η βασικότερη αριθμητική πράξη, αποτελώντας τμήμα των περισσότερων ψηφιακών συστημάτων. Επίσης, είναι η πιο συχνή πράξη που εκτελείται από μία επεξεργαστική μονάδα. Η επιτάχυνση της απασχολεί την επιστήμη των ψηφιακών κυκλωμάτων για αρκετά χρόνια. Στα πλαίσια αυτής της εργασίας παρουσιάζεται μια νέα υλοποίηση αθροιστών υπολοίπου $2^n - 1$. Τεχνικές και αρχιτεκτονικές που έχουν προταθεί στο παρελθόν, είτε αφορούν άμεσα τους αθροιστές υπολοίπου, είτε γενικότερα τους αθροιστές, συνδυάζονται για την ανάπτυξη του νέου σχεδιασμού. Συγκεκριμένα και περιεκτικά, έγινε χρήση των μοντέλων κατά Jackson και Talwar, που αφορούν την γενίκευση της παραγοντοποίησης που συστήθηκε από τον H. Ling. Εφαρμόζοντας αυτές τις λογικές συναρτήσεις σε μία τοπολογία που πλεονεκτεί σε προθεματικά επίπεδα, επιτεύχθηκε και το τελικό αποτέλεσμα. Οι αθροιστές που αναπτύχθηκαν συγκρίθηκαν με τις αποδοτικότερες τοπολογίες όσο άφορα την ταχύτητα, το εμβαδόν και την κατανάλωση ενέργειας, και τα αποτελέσματα είναι αρκετά υποσχόμενα. Ειδικότερα σε θεωρητικές προδιαγραφές η καθυστέρηση των νέων αθροιστών είναι αισθητά μικρότερη, ενώ σε πρακτικές συνθήκες οι αποδόσεις είναι θετικές σε ορισμένες περιπτώσεις και εξαρτώμενες της τεχνολογίας υλοποίησης.

Περιεχόμενα

List of Figures

List of Tables

List of Equations

1 Εισαγωγή

Οι αριθμητικές μονάδες είναι βασικά στοιχεία στο σύνολο του υλικού των ηλεκτρονικών υπολογιστών. Οι περισσότερες επεξεργαστικές μονάδες έχουν ως βασικό δομικό στοιχείο την Αριθμητική Λογική Μονάδα (ΑΛΜ) ή Arithmetic Logic Unit (ALU) η οποία είναι υπεύθυνη για την υλοποίηση βασικών αριθμητικών πράξεων όπως πρόσθεση και πολλαπλασιασμό αλλά και λογικών πράξεων όπως OR, AND και XOR . Αυτές οι μονάδες είναι κρίσιμες στην ορθή λειτουργία ενός συστήματος αλλά και την ταχύτητα.

Ένα από τα μεγαλύτερα εμπόδια, όσο αφορά την ταχύτητα υπολογισμού του αθροίσματος, είναι ο χρόνος υπολογισμού των ενδιάμεσων κρατουμένων αλλά και του κρατούμενου εξόδου. Μία από τις μεγαλύτερες καινοτομίες που παρουσιάστηκαν με σκοπό την ελαχιστοποίηση των χρονικών αυτών διαστημάτων, είναι οι αρχιτεκτονικές πρόβλεψης κρατουμένων ή Carry Look Ahead (CLA). Η δομή των CLA αθροιστών απασχόλησε αρκετά την επιστημονική κοινότητα, με αρκετές προτάσεις υλοποίησης, η κάθε μία με διαφορετικές απαιτήσεις πόρων ανάλογα την εφαρμογή.

Στην παρούσα διπλωματική εργασία θα αναλυθούν οι αθροιστές και συγκεκριμένα μια υποομάδα αυτών , οι αθροιστές υπολοίπου $2^n - 1$. 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. Οι αθροιστές υπολοίπου δεν διαφέρουν πολύ από τους απλούς αθροιστές. Αν όχι όλες, τουλάχιστον οι περισσότερες κατηγορίες αθροιστών συμμερίζονται τον υπολογισμό των κρατουμένων στο κρίσιμο μονοπάτι τους.

Στα επόμενα κεφάλαια θα πραγματοποιηθεί μια ουσιαστική αναδρομή βασικών αρχιτεκτονικών που αφορούν την πράξη της πρόσθεσης. Θα αναλυθούν οι δομές που είναι πιο διαδεδομένες και συνέβαλαν στη πειραματική διαδικασία σχεδιασμού ενός αθροιστή υπολοίπου. Στην συνέχεια θα γίνει μία αναλυτική αναφορά στους αθροιστές υπολοίπου $2^n - 1$, καθώς και αρχιτεκτονικές βελτιστοποίησης τους. Τέλος, αναπτύσσεται ένας αθροιστής υπολοίπου, συνδυάζοντας διάφορες τεχνικές και σχεδιασμούς που έχουν αναφερθεί. Οι αθροιστές που θα αναπτυχθούν, εφόσον ελεγχθούν επαρκώς, θα συγκριθούν με παρελθοντικές υλοποιήσεις όσο αφορά το χρόνο, χώρο και κατανάλωση ενέργειας.

2 Πρόσθεση στο Δυαδικό

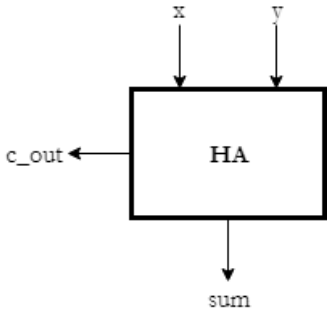
Οι αριθμητικές πράξεις σε αριθμούς εκφρασμένου σε μία συγκεκριμένη βάση ακολουθούν τους ίδιους κανόνες με αυτούς του δεκαδικού συστήματος, χρησιμοποιώντας σε κάθε περίπτωση μόνο τα διαθέσιμα ψηφία του κάθε συστήματος. Στα πλαίσια αυτής της διπλωματικής χρησιμοποιείται το δυαδικό σύστημα εφόσον κύριο 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

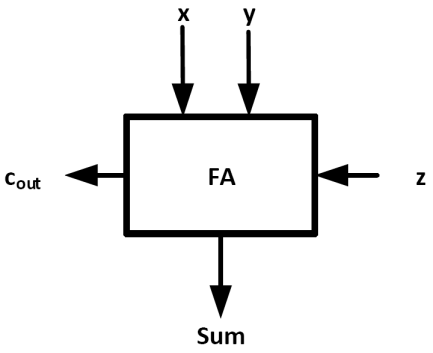
Το παραπάνω σύστημα ονομάζεται ημιαθροιστής ή Half-Adder (HA) , έχει δυο εισόδους τις x και y και δυο εξόδους τις c_{out} και sum και οι έξοδοι σύμφωνα με τον πίνακα αληθείας περιγράφονται από τις παρακάτω συναρτήσεις άλγεβρας Μπουλ :

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

Από τον ημιαθροιστή 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

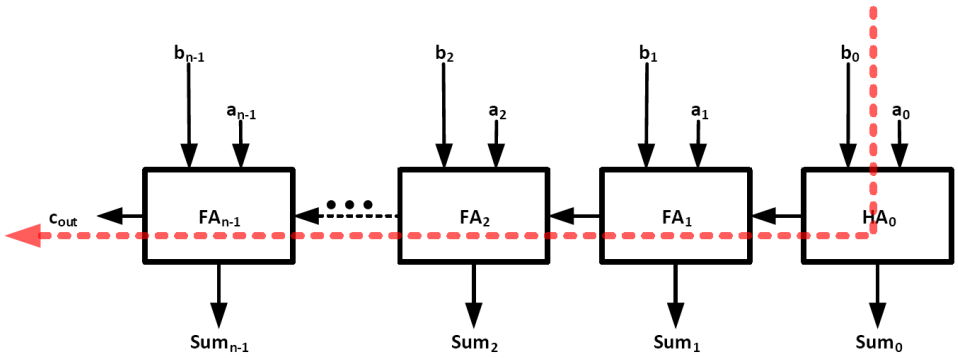
και οι εξισώσεις των εξόδων είναι :

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

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 $a_i + b_i$. Επομένως 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

Στο προηγούμενο κεφάλαιο περιγράφηκε σχηματικά ένας απλός σχεδιασμός αθροιστή. Σε αυτή την ενότητα θα πραγματοποιηθεί η κατάταξη αυτού του αθροιστή σε μία κατηγορία, καθώς και η παρουσίαση μερικών ακόμα βασικών αρχιτεκτονικών.Επίσης θα αποδοθεί μια σύντομη περιγραφή, αναφέροντας πλεονεκτήματα και μειονεκτήματα για κάθε δομή.

3.1 Διάδοσης Κρατουμένου

Αθροιστής Διάδοσης Κρατουμένου ή Ripple-Carry Adder (RCA) είναι ο πιο απλός αθροιστής και το σχηματικό του παρουσιάστηκε στην εικόνα 2.3. Παρατηρείται 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 c_0, αντίστοιχα για το τρίτο πρέπει να υπολογιστεί πρώτα το c_0 έπειτα από τον δεύτερο πλήρη αθροιστή να υπολογιστεί το c_1 με είσοδο στο c_{in} το c_0 δηλαδή το c_{out} του προηγούμενου FA. Επομένως κάθε FA δεν λειτουργεί παράλληλα με τα υπόλοιπα στοιχεία άθροισης, 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. Σημειώνεται πως τα σήματα generate και propagate αναφέροντε στα ζεύγη των δυαδικών ψηφίων των αριθμών εισόδου. 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 ισχύει $g_i = 1$ τότε γνωστοποιείται πως ο συγκεκριμένος πλήρης αθροιστής παράγει κρατούμενο εξόδου ανεξάρτητα του κρατουμένου εισόδου. 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 την ύπαρξη κρατούμενο εισόδου. Δηλαδή αν $g_i = 1$ τότε και $c\_out_i = 1$ χωρίς να ισχύει το αντίθετο.

Οι αθροιστές Παράλειψής Κρατουμένου ή Carry Skip είναι μια υλοποίηση που βελτιώνει την καθυστέρηση διάδοσης κρατουμένου με έναν απλό τρόπο αλλά όχι αρκετά αποτελεσματικό σε σχέση με άλλες αρχιτεκτονικές. Η χείριστη περίπτωση παρουσιάζεται παρομοίως με 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 ορίζεται ο όρος select όπου οδηγεί την είσοδο επιλογής ενός πολυπλέκτη 2-σε-1, όπου η έξοδος του είναι το c_{out} του αθροιστή, και όταν το select είναι αληθές 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 πυλών.

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.

Ομοίως με τον αθροιστή παράβλεψης κρατουμένου, η συγκεκριμένη αρχιτεκτονική έχει πρακτική εφαρμογή όταν εφαρμόζεται σε επιμέρους τμήματα.

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. Αυτά τα σήματα έχουν όμοια ερμηνεία με αυτά που παρουσιάστηκαν στους αθροιστές παράβλεψις αλλά διαφορετική χρήση.

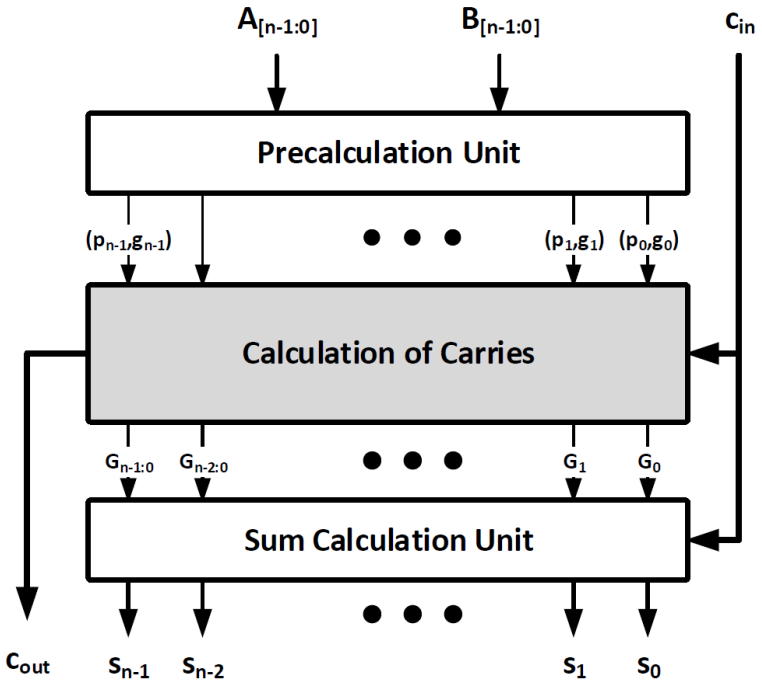
- Συνδυάζοντας αυτά τα σήματα δίνεται η δυνατότητα να προσδιοριστεί ταχύτερα αν ένα τμήμα ζευγών ψηφίων πρόκειται να διαδώσει ή να παράξει ένα κρατούμενο. Κάθε κρατούμενο, ενδιάμεσο ή τελικό, μπορεί να υπολογιστεί ξεχοριστά απο την Μονάδα Υπολογισμού Κρατουμέμων.

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. Παρακάτω παρουσιάζονται οι λογικές συναρτήσεις των κρατουμένων αυτού του αθροιστή.

$$\begin{aligned} c_0 &= g_0 + p_0 c_{in} \\ c_1 &= g_1 + p_1 g_0 + p_1 p_0 c_{in} \\ c_2 &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{in} \\ c_3 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{in} \end{aligned} \tag{3.3}$$

Το σήμα c_3 είναι το κρατούμενο εξόδου του 4-bit αθροιστή και όπως γίνεται αντιληπτό αντιπροσωπεύει το κρατούμενο που παράγεται από το πρώτο ζεύγος (A_0, B_0) έως το τέταρτο (A_3, B_3). Οπότε το c_3 είναι και το κρατούμενο που παράγεται από μία ομάδα των τεσσάρων εισόδων και μπορεί να αναπαρασταθεί και ως $G_{3:0}$. Δηλαδή κάθε κρατούμενο αναπαριστάται με την αντίστοιχη ομάδα $c_i = G_{i:0}$. Τα σήματα αυτά θα αναλυθούν στο επόμενο κεφάλαιο.

Είναι σημαντικό να τονιστεί πως η λογική για των υπολογισμό των κρατουμένων που μόλις αναφέρθηκε καταλαμβάνει επιπλέον πόρους. Επίσης κρατούμενα που αφορούν σημαντικότερα ψηφία εισόδου απαιτούν και περισσότερη λογική, όπως φαίνεται και στην εξίσωση 3.3.



Εικόνα 3.1: CLA Architecture

Στην εικόνα 3.1 παρουσιάζεται η βασική αρχιτεκτονική ενός αθροιστή πρόβλεψης κρατουμένου των n δυαδικών ψηφίων. Στο πρώτο επίπεδο υπολογίζονται τα σήματα generate και propagate, στο δεύτερο τμήμα υπολογίζονται τα σήματα Group Generate (κρατούμενα c_i) και Group Propagate και στο τρίτο πραγματοποιείται η άθροιση. Σημειώνεται στην συγκεκριμένη αρχιτεκτονική το μεγαλύτερο λειτουργικό φόρτο καθώς και εμβαδόν καταλαμβάνει η μονάδα υπολογισμού των κρατουμένων. Στις επόμενες ενότητες θα εξεταστούν εκτενώς τεχνικές βελτιστοποίησης της μονάδας αυτής.

4 Προθεματική Αθροιστές

Στην ενότητα αυτή θα παρουσιαστεί μια νέα προσέγγιση αθροιστών, οι αθροιστές προθέματος, ο οποίοι έλαβαν σημαντικό ρόλο στην επιτάχυνση καθώς και μείωση του συνολικού εμβαδού των αθροιστών πρόβλεψης κρατουμένου που περιγράφηκαν στο προηγούμενο κεφάλαιο. Όπως προαναφέρθηκε οι αθροιστές προθέματος που ακολουθούν, αποτελούν τροποποίηση της μονάδας υπολογισμού με στόχο την επιτάχυνση.

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 ένα σύνολο μεγέθους n από αριθμούς $(x_{n-1}, x_{n-2}, ..., x_1, x_0)$, σύμφωνα με τον ορισμό που δόθηκε παραπάνω ορίζεται ένα ακόμα σύνολο ίδιου μεγέθους $(y_{n-1}, y_{n-2}, ..., y_1, y_0)$ όπου κάθε στοιχείο του συνόλου αυτού υπολογίζεται αναδρομικά, σύμφωνα με την παρακάτω εξίσωση

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

και το τελικό αποτέλεσμα καταχωρείται στο y_{n-1}.

Το πρόβλημα υπολογισμού κρατουμένου είναι δυνατόν να μετατραπεί σε prefix problem δημιουργώντας τα ζεύγη (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 υπολογίζεται 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

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

Παρακάτω ακολουθεί η επαγωγική απόδειξη. Εφόσον δεν υπάρχει κρατούμενο εισόδου ($c_{in} = c_{-1} = 0$) έχουμε

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

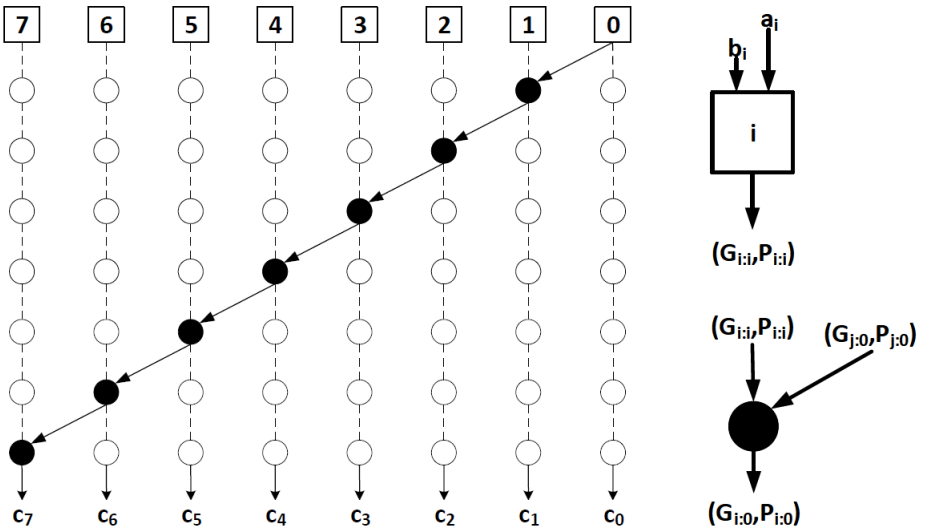
έστι το αποτέλεσμα ισχύει για $i - 1$
Αν $i > 0$ και $c_{i-1} = G_{i-1}$ τοτε

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

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 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 σε σχέση με άλλες τοπολογίες αθροιστών προθέματος που θα παρουσιαστούν στην συνέχεια. Σημειώνεται πως λιγότεροι κόμβοι συνεπάγεται μικρότερο εμβαδόν.

4.2 Παράλληλοι Προθεματικοί Αθροιστές

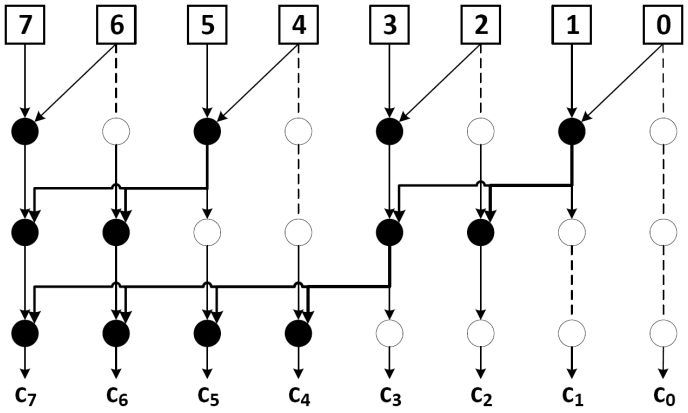
Η διάταξη που παρουσιάστηκε στην εικόνα 4.1 έχει την δομή του απλού αθροιστή διάδοσης κρατουμένου εκφρασμένο σε προθεματική μορφή. Ο προθεματικός όμως υπολογισμός κρατουμένων μπορεί να υπολογιστεί παράλληλα, αποτελώντας την προθεματική έκφραση ως κινητήριο παράγοντα διάφορων παράλληλων αρχιτεκτονικών που προτάθηκαν. Μερικές σημαντικές αρχιτεκτονικές διατυπώνονται στις επόμενες παραγράφους με σύντομη περιγραφή.

4.2.1 Δέντρα-Δομές Προθεμάτων

J. Sklansky

Μία από τις πρώτες παράλληλες τοπολογίες που προτάθηκε ήταν το 1960 απο τον Sklansky [Skl60]. Το πλεονέκτημα της είναι το ελάχιστο βάθος. Στην εικόνα 4.2 παρουσιάζεται ένα παράδειγμα υπολογισμού των κρατουμένων για είσοδο των οκτώ δυαδικών ψηφίων. Το βάθος σε αυτή την περίπτωση είναι τρία επίπεδα. Παρατηρείται πως ο κόμβος του κρατουμένου c_4 εκτός από το ίδιο το κρατούμενο οδηγεί επιπλέον τέσσερις εισόδους αλλών κόμβων (Fan-out

-4). Το υψηλό Fan-Out των κόμβων είναι το σημείο στο οποίο μειονεκτεί ο σχεδιασμός αυτός.

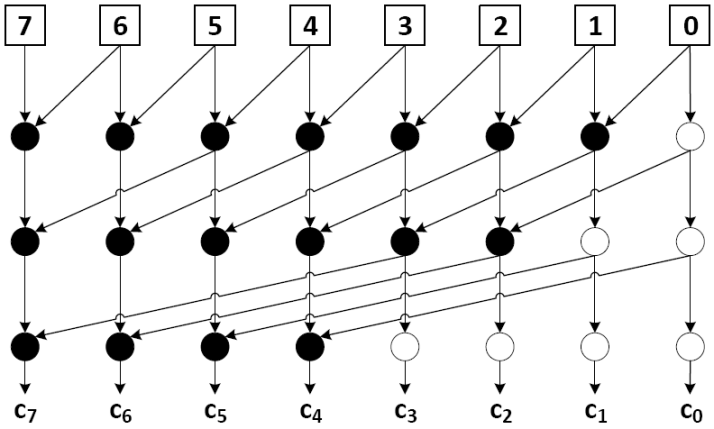


Εικόνα 4.2: J. Sklansky Prefix Tree Adder

Το 1980 παρουσιάστηκε μια αρχιτεκτονική από τους Ladner και Fischer [LF80] η οποία είχε την ίδια βασική δομή με αυτή του Sklansky, με κάποιες τροποποιήσεις ανάλογα την εφαρμογή, για καλύτερη απόδοση.

Kogge-Stone

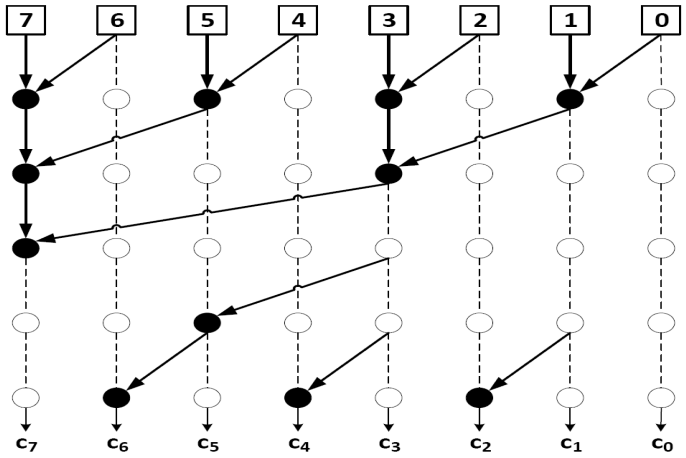
Όπως και η αρχιτεκτονική που προτάθηκε από τον Sklansky, έτσι και η αρχιτεκτονική Kogge-Stone [KS73] έχει μικρό βάθος. Η βασική του όμως διαφορά είναι πως δεν υπάρχει κόμβος με μεγαλύτερο του δυο fan-out, θυσιάζοντας το εμβαδόν του σχεδιασμού.



Εικόνα 4.3: Kogge-Stone Prefix Tree Adder

Brent-Kung

Στην περίπτωση της δομής Brent-Kung των Brent και Kung [BK82] όπως φαίνεται και στην εικόνα 4.4 παρουσιάζει μεγάλο βάθος αλλά καταλαμβάνει λιγότερο εμβαδόν.



Εικόνα 4.4: Brent-Kung Prefix Tree Adder

Σύγκριση των παράλληλων διατάξεων

Στον πίνακα 4.1 εκφράζεται μαθηματικα το πλήθος των επιπέδων, το πλήθος των κόμβων και το μέγιστο Fan-Out ανα αθροιστή, ανάλογα με το μήκος εισόδου του n.

Architecture	Levels(Delay)	nodes(Area)	max fan-out
Serial	$n-1$	$n-1$	2
Sklansky	$\log_2 n$	$n/2 \log_2 n$	$n/2+1$
Kogge-Stone	$\log_2 n$	$n(\log_2 n - 1) + 1$	2
Brent-Kung	$2\log_2 n - 1$	$2(n-1) - \log_2 n$	$\log_2 n + 1$

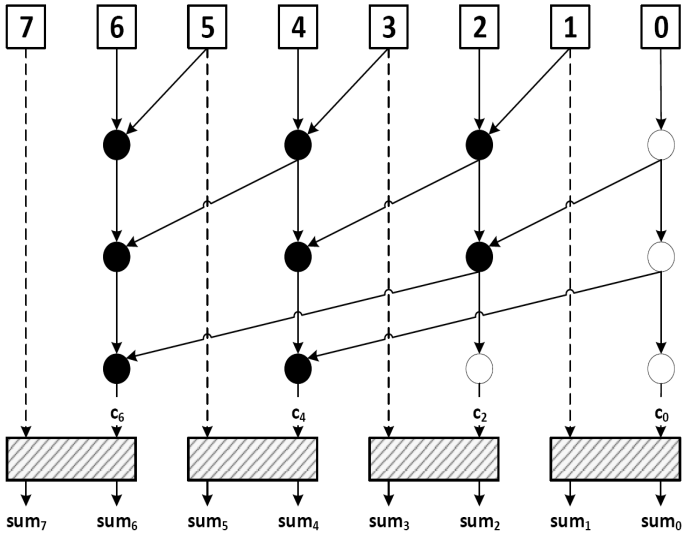
Πίνακας 4.1: Prefix Comparisson

Στις προθεματικές τοπολογίες που παρουσιάζονται σε αυτή την εργασία κάθε επίπεδο θεωρητικά έχει την ίδια χρονική καθυστέρηση με τα υπόλοιπα επίπεδα. Η παραδοχή αυτή στηρίζεται στο ότι σε κάθε επίπεδο, κάθε κόμβος είναι παράλληλα ή ανεξάρτητος συνδεδεμένος με τους υπόλοιπους του ίδιου επιπέδου, με αποτέλεσμα το κρίσιμο μονοπάτι του επιπέδου να είναι ίσο με αυτό του κόμβου. Συμπεραίνοντας πως κάθε επίπεδο έχει ίδια καθυστέρηση και ίση με την καθυστέρηση ενός κόμβου. Αποτέλεσμα του ισχυρισμού αυτού είναι η συσχέτιση του πλήθους των επιπέδων με την συνολική καθυστέρηση του αθροιστή.

Η παραπάνω αναλογία μεταξύ των επιπέδων και της καθυστέρησης έχει εφαρμογή και σε φυσικές υλοποιήσεις των προθεματικών αθροιστών. Συγκρίνοντας τους αθροιστές Kogge-Stone και Sklansky ως προς την ταχύτητα, σύμφωνα με τον παραπάνω ισχυρισμό θα ήταν αναμενόμενο να έχουν την ίδια απόδοση. Συμπεριλαμβάνοντας και το εμβαδόν στην εξίσωση επιλογής ταχύτερου αθροιστή, ως δευτερεύον παράγοντα, το πρόβλημα τείνει στην επιλογή του Sklansky. Ο λόγος, όμως που η αρχιτεκτονική Kogge-Stone είναι ιδανική αφορά το μικρό fan-out. Στην περίπτωση κυκλώματος με κόμβους που έχουν υψηλό fan-out, είναι αναγκαία η πρόσθεση ηλεκτρονικών στοιχείων για την σωστή οδήγηση, με αποτέλεσμα να επιβαρύνετε το σύστημα χρονικά.

4.3 Αραίωση των Δέντρων

Οι δομές που περιγράφηκαν για την παραλληλοποίηση του υπολογισμού των κρατουμένων είναι αρκετά πυκνές όσο αφορά τους κόμβου, το οποίο συνεπάγει υψηλές απαιτήσεις πόρων υλικού και ενέργειας. Με σκοπό την μείωση του εμβαδού εφαρμόζονται τεχνικές αραίωσης ή sparseness. Οι τεχνικές αυτές βασίζονται στον υπολογισμό λιγότερων κρατούμενων από των απαιτούμενων και την διάδοση αυτών για τον υπολογισμό των υπόλοιπων. Μειωνέκτημα αυτής της τεχνικής ειναι η αυξηση της λογικής πολυπλοκότητας του τελευταίου επιπέδου απου πραγματοποιείται η αθροιση, με αποτέλεσμα την μείωση της ολικής χρονικής απόδοσης.



Εικόνα 4.5: Προθεματικός αθροιστής με τεχνική αραίωσης

Στην εικόνα 4.5 παρουσιάζεται το διάγραμμα ενός αθροιστή οκτώ δυαδικών ψηφίων με την εφαρμογή της τεχνικής αραίωσης sparse-2, δηλαδή υπολογίζονται τα μισά κρατούμενα (ανά δύο), στην δομή Kogge-Stone. Στις εξισώσεις που ακολουθούν περιγράφεται η διάδοση των κρατουμένων στο τελευταίο

επίπεδο για τεχνική αραίωσης sparse-2 και sparse-4.

Sparsness-2

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

Sparsness-4

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

4.4 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 :

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

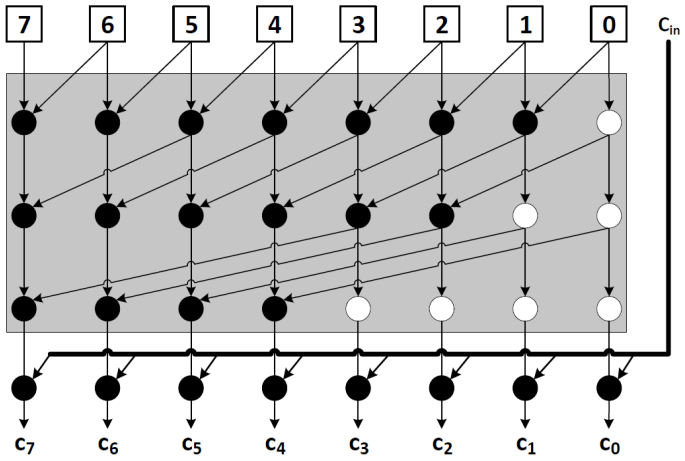
Υποθέτοντας πως η σχέση ισχύει και για $i = k - 1$, δηλαδή

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

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 $G_\iota + P_\iota * c_{-1}$. Στην εικόνα 4.6 παρουσιάζεται ένα παράδειγμα προθεματικού αθροιστή με κρατούμενο εισόδου. Εντός του σκιασμένου πλαισίου είναι δυνατό να εφαρμοστεί οποιαδίποτε προθεματική δομή (χωρίς κρατουμενο εισόδου). Στο παράδειγμα αυτό χρησιμοποιήθηκε η αρχιτεκτονική του Kogge-Stone.



Εικόνα 4.6: Kogge-Stone Prefix Tree Adder with Carry-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, 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

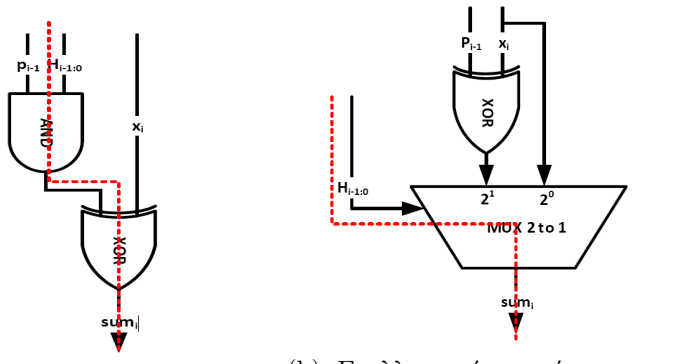
$$
\begin{aligned}
H_{7:2} &= H_{7:5} + P_{6:4} * H_{4:2} \\
H_{7:5} &= g_7 + g_6 + p_6 g_5 \\
H_{4:2} &= g_4 + g_3 + p_3 g_2 \\
P_{6:4} &= p_6 p_5 p_4 \\
H_{7:2} &= g_7 + g_6 + p_6 g_5 + p_6 p_5 p_4 * \left(g_4 + g_3 + p_3 g_2\right) \\
H_{7:2} &= g_7 + g_6 + p_6 g_5 + p_6 p_5 g_4 + p_6 p_5 p_4 g_3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

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. 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, όπως φαίνεται στην εικόνα 5.1a.



(a) Απλή προσέγγιση

(b) Εναλλακτική προσέγγιση με πολυπλέκτη

Εικόνα 5.1: Λογική υπολογισμού του αθροίσματος κατά Ling

Η εξίσωση 5.6 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 τροποποίηση που εκφράζεται στην εξίσωση 5.7 και απεικονίζεται στην εικόνα 5.1b με μία πρώτη ματιά φαίνεται να 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. 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.

Παρατηρείται, λοιπόν, πως η μέθοδος του Ling μειώνει την πολυπλοκότητα μόνο στο πρώτο επίπεδο του αθροιστή, ενώ η αύξηση πολυπλοκότητας του τελευταίου επιπέδου δεν επιβαρύνει τον σχεδιασμό χρονικά σύμφωνα με την παραπάνω τροποποίηση.

5.3 Αραίωση σε Ling Αρχιτεκτονικές

Οι τεχνικές αραίωσης που παρουσιάστηκαν στην παράγραφο 4.3 είναι δυνατό να εφαρμοστούν και στην περίπτωση της παραγοντοποίησης κατά Ling [DN05].

Sparsness-2

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

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 παρουσιαστεί μια επιπλέον μέθοδος 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 επιπέδου των αθροιστών, αλλά είναι δυνατό να εφαρμοστεί σε όλα τα επίπεδα. Ουσιαστικά στο κεφάλαιο αυτό συστήνεται η γενίκευση της παραγοντοποίησης που προτάθηκε παραπάνω.

Στις παρακάτω εξισώσεις παραγοντοποιείται η εξίσωση υπολογισμού κρατουμένου. Η πρώτη παραγοντοποίηση αναλύθηκε στην ενότητα 5, ενώ 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 εκφραστεί μια γενικευμένη μορφή για αυτές τις παραγοντοποιήσεις και επιπλέον θα υποδειχθούν τεχνικές μείωσης πολυπλοκότητας και στα επιμέρους τμήματα τις παραπάνω εξίσωσης (μέσα στις αγκύλες). Η παραγοντοποίηση αυτή καλείται 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, δηλαδή,

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

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

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

Γνωρίζοντας τα παραπάνω σήματα είναι δυνατό να εκφραστεί πλέων η βασική παραγοντοποίηση του υπολογισμού του κρατουμένου εξόδου που συστήνεται από τον Jackson. Ουσιαστικά επαναδιατυπώνεται η εξίσωση 6.1 με διαφορετική μορφή, χρησιμοποιώντας τα σήματα που συστήθηκαν παραπάνω.

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

Από την παραπάνω εξίσωση το τμήμα μέσα στις αγκύλες ορίζεται ως R και παρακάτω παρουσιάζονται τρείς διαφορετικές εκφράσεις της ίδιας συνάρτησης R.

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

οπού η υπογεγραμμένη αναφέρεται στην ομάδα των δυαδικών ψηφίων εισόδου και το υπερκείμενο στο πλήθος των ψηφίων που δέχεται σαν είσοδο το σήμα B. Η εξίσωση του κρατουμένου ομάδας λαμβάνει την τελική της έκφραση

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

Πρέπει να σημειωθεί πως στην περίπτωση που $j = k$ στην παραπάνω σχέση τότε εφαρμόζεται μία υποκατηγορία των Jackson συναρτήσεων που αφορά την παραγοντοποίηση που παρουσιάστηκε 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 υποομάδες αυτού του συνόλου. Παρακάτω θα αποδειχθεί πως και το σήμα R που ορίστηκε έχει επίσης αυτή την ιδιότητα.

Θα κατασκευαστεί 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 επιλέγονται 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 αναγράφεται

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

επίσης διασπώντας το B σε δυο σήματα αποτυπώνεται

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

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 διατυπώθηκε παραπάνω, η εξίσωση μετασχηματίζεται ως

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

Οπότε η τελική εξίσωση είναι

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

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

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

Μέχρι στιγμής έχει αποδειχθεί πως το σήμα R ενός συνόλου δυαδικών ψηφίων μπορεί να υπολογιστεί από τα σήματα R μικρότερων υποομάδων. Πρέπει να τονιστεί πως υπάρχουν πολλοί διαφορετικοί συνδυασμοί και τύποι, που υλοποιούν τον αναδρομικό υπολογισμό, οι οποίοι θα παρουσιαστούν στην συνέχεια. Παρακάτω θα αποδειχθεί πως και το σήμα Q υπολογίζεται αναδρομικά, αφού πρώτα αποτυπωθεί ο αναδρομικός υπολογισμός 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 ακολουθήθηκε προηγουμένως θα αποδειχθεί πώς υπολογίζεται αναδρομικά το σήμα Q με είσοδο το σύνολο των bits $n-1:0$. Το αρχικό σύνολο διασπάται σε τρία διαμερίσματα $n-1:k, k-1:j$ και $j-1:0$, ομοίως επιλέγονται τα μεσαία στοιχεία των τελευταίων δύο υποομάδων m και v αντίστοιχα. Ξεκινώντας από την εξίσωση του σήματος Q που εκφράστηκε παραπάνω, αρχικά γίνεται χρήση της πρώτης εξίσωσης από τις παραπάνω δυο αναδρομικές συναρτήσεις του D, ενώ παράλληλα διασπάται το σήμα P, και αποτυπώνεται

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

Σε αυτό το σημείο μέσα στις δύο πρώτες αγκύλες έχουν ήδη χτιστεί τα Q σήματα των πρώτων δύο υποσυνόλων, για την παραγωγή και του τρίτου στην τελευταία αγκύλη θα γίνει 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 τα αντίστοιχα σήματα R και Q παρουσιάζεται η τελική μορφή της αναδρομικής συνάρτησης

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

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 ή Radix-2 (σθένος-2) και τέσσερα ή Radix-4 (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:

$$\begin{aligned} R_{i:0}^{m} &= R_{i:j+1}^{m} + Q_{i-m:j+1-m}^{i-m-j} * R_{j:0}^{m} \\ R_{i:0}^{i-j+m} &= R_{i:j+1}^{m} + R_{j:0}^{m} \end{aligned} \tag{6.12}$$

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

Λογικές συναρτήσεις Radix-4:

$$\begin{aligned} R_{i:0}^{m} =& R_{i:j+1}^{m} + Q_{i-m:j+1-m}^{i-m-j} R_{j:k+1}^{m} + \\ & Q_{i-m:j+1-m}^{i-m-j} Q_{j-m:k+1-m}^{j-m-k} R_{k:l+1}^{m} \\ & Q_{i-m:j+1-m}^{i-m-j} Q_{j-m:k+1-m}^{j-m-k} Q_{k-m:l+1-m}^{k-m-l} R_{l:0}^{m} \\ R_{i:0}^{i-j+m} =& R_{i:j+1}^{m} + R_{j:k+1}^{m} + Q_{j-m:k+1-m}^{j-m-k} R_{k:l+1}^{m} \\ & Q_{j-m:k+1-m}^{j-m-k} Q_{k-m:l+1-m}^{k-m-l} R_{l:0}^{m} \\ R_{i:0}^{i-k+m} =& R_{i:j+1}^{m} + R_{j:k+1}^{m} + R_{k:l+1}^{m} + Q_{k-m:l+1-m}^{k-m-l} R_{l:0}^{m} \\ R_{i:0}^{i-l+m} =& R_{i:j+1}^{m} + R_{j:k+1}^{m} + R_{k:l+1}^{m} + R_{l:0}^{m} \end{aligned} \tag{6.14}$$

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

Παρατηρείται πως στην περίπτωση των Radix-2 δεν υπάρχει διαφορά σε σχέση με τις αντίστοιχες συναρτήσεις Prefix και Ling. Η χρονική καθυστέρηση και στις δύο 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 αυξάνεται η πολυπλοκότητα της υλοποίησης 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 6.5).

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 όπως φαίνεται και στην εξίσωση 6.7), το D πρέπει να έχει την τελική του τιμή τουλάχιστον δυο FO4 [1] 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

[1] FO4 : Fan-out of n (FOn) είναι μια μετρική καθυστέρησης πύλης που χρησιμοποιείται στα ψηφιακά CMOS τεχνολογίας. Στην περίπτωση του FO4 η καθυστέρηση αντιστοιχεί σε αυτή της πύλης XOR

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

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

1. Όπως προαναφέρθηκε, η κατασκευή του όρου D πρέπει να είναι τουλάχιστον δυο με τρία FO4 επίπεδα απο τον αντίστοιχο όρο R.

2. Δεν πρέπει να αφαιρείται 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[2]).

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 Αραίωση σε Jackson Αρχιτεκτονικές

Τεχνικές αραίωσης εφαρμόζονται επίσης και στις Jackson εκφράσεις. Sparsness-2

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

[2]AND-OR-INVERTER και OR-AND-INVERTER

Sparsness-4

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

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 είναι χρήσιμες 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 εκφράζεται μέσω αριθμητική υπολοίπου ως $(A + B) \ mod \ n$, 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 του παρόν έργου.

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

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

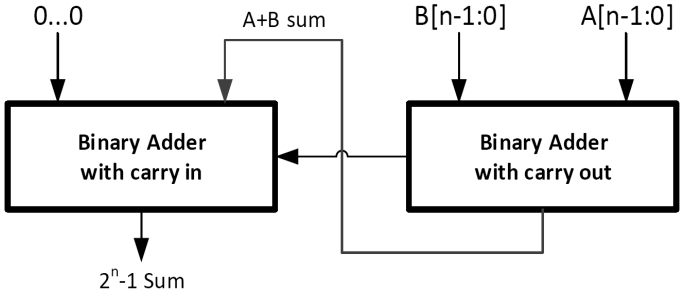
Ο ορισμός αυτός μαθηματικά έχει ένα λάθος το οποίο όμως, λαμβάνοντας μία παραδοχή, θα εξαλειφθεί. Για παράδειγμα έστω πως $n = 3$ άρα και $2^n - 1 = 7$, όπου n το πλήθος των δυαδικών ψηφίων που έχει κάθε αριθμός εισόδου. Για τον υπολογισμό του υπολοίπου, όπως προαναφέρθηκε, διαιρείται ο αριθμός εισόδου 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

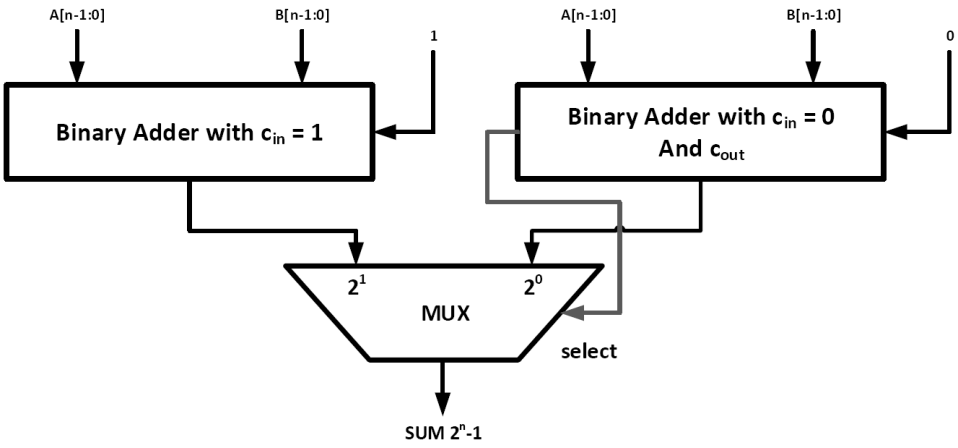
προφανής όπου όλα τα ψηφία είναι 0 και η δεύτερη είναι η $2^n - 1$, δηλαδή όλα τα ψηφία να είναι στο 1. Αυτό το φαινόμενο δεν είναι πάντα επιθυμητό. Στην συνέχεια του κεφαλαίου θα δοθεί μια πρόταση για την αντιμετώπιση της διπλής αυτής αναπαράστασης.

Υπάρχουν διάφοροι τρόποι για να υπολογιστεί στο υλικό το αποτέλεσμα ενός αθροιστή υπολοίπου 2^n-1. Η πιο απλή ιδέα 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. Στην εικόνα 7.1 αποτυπώνεται 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 είναι δυνατή η παράλληλη εκτέλεση δυο προθέσεων του A και B με τον ένα αθροιστή να έχει κρατούμενο εισόδου και τον άλλο χωρίς. Τα αποτελέσματα των δύο αθροιστών θα οδηγούνται σε έναν πολυπλέκτη με είσοδο επιλογής το κρατούμενο εξόδου του αθροιστή χωρίς κρατούμενο εισόδου. Αν η είσοδος επιλογής είναι ενεργή τότε θα επιλέγεται η έξοδος του αθροιστή με κρατούμενο εισόδου όπως φαίνεται στην εικόνα 7.2.



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

Πρέπει να σημειωθεί πως και στις δύο περιπτώσεις απαιτείται η υλοποίηση δύο αθροιστών με αποτέλεσμα αυξάνεται το εμβαδόν του αθροιστή στο διπλάσιο συγκριτικά με έναν απλό αθροιστή. Ένας ακόμα παράγοντας σε αυτές τις αρχιτεκτονικές είναι η δομή των επιμέρους αθροιστών. Η επίτευξη υψηλών ταχυτήτων είναι δυνατή επιλέγοντας μια παράλληλη προθεματική δομή για τον σχεδιασμό των αθροιστών που συντελούν το αθροιστή υπολοίπου, θυσιάζοντας αρκετά τον χώρο και την κατανάλωση.

Μία καθιερωμένη υλοποίηση των αθροιστών αυτής της κατηγορίας χρησιμοποιεί έναν τυπικό αθροιστή συνδέοντας το κρατούμενο εξόδου στο κρατούμενο εισόδου δημιουργώντας έναν αθροιστή κυκλικού κρατουμένου (end-round carry). Αυτή η τοπολογία μετατρέπει τον αθροιστή σε ένα ασύγχρονο ακολουθιακό κύκλωμα, του οποίου η κατάσταση εξαρτάται από προηγούμενες και από την καθυστέρηση της διάδοσης κρατουμένων. Εξαιτίας αυτής της συμπεριφοράς δημιουργούνται πρακτικά προβλήματα χρονισμού με αποτέλεσμα την επιπλέον αύξηση του χρόνου καθυστέρησης. Το πρόβλημα αυτό παρατηρείται στις περιπτώσεις που οι δυαδικοί αριθμοί εισόδου είναι συμπληρωματικοί ως προς ένα και αντιμετωπίζεται [She77] οδηγώντας την είσοδο κρατουμένου με την παρακάτω συνάρτηση και όχι απευθείας με το κρατούμενο εξόδου.

$$C_{in} = C_{out} + P_{n-1:0} \tag{7.2}$$

Με αυτήν την μικρή διαμόρφωση αντιμετωπίζεται επίσης και η διπλή αναπαράσταση του μηδέν. Η μόνη περίπτωση που προκύπτει η δεύτερη αναπαράσταση του μηδέν (1...11) πετυχαίνεται όταν οι αριθμοί εισόδου είναι συμπληρωματικοί. Με την παραπάνω τροποποίηση όμως στην περίπτωση αυτή το σήμα $P_{n-1:0}$ ενεργοποιείται και οδηγεί το κρατούμενο εισόδου, με προφανές τελικό αποτέλεσμα το μηδέν στην κανονική αναπαράστασή του.

Στην συνέχεια της εργασίας αυτής οι αθροιστές υπολοίπου που θα παρουσιαστούν ή θα σχεδιαστούν, θα είναι με την σύμβαση της διπλής αναπαράστασης.

Συγκριτικά, οι αθροιστές με διπλή αναπαράσταση είναι ταχύτεροι και απαιτούν λιγότερους πόρους από αυτούς με μονή αναπαράσταση [ENK94]. Αυτός είναι και ο λόγος που οι αυτοί αθροιστές προτιμούνται στις περιπτώσεις που είναι εφικτοί. Αν η εφαρμογή απαιτεί μονή αναπαράσταση τότε η πρόσθεση επιπλέον λογικής στους αθροιστές είναι αναπόφευκτη.

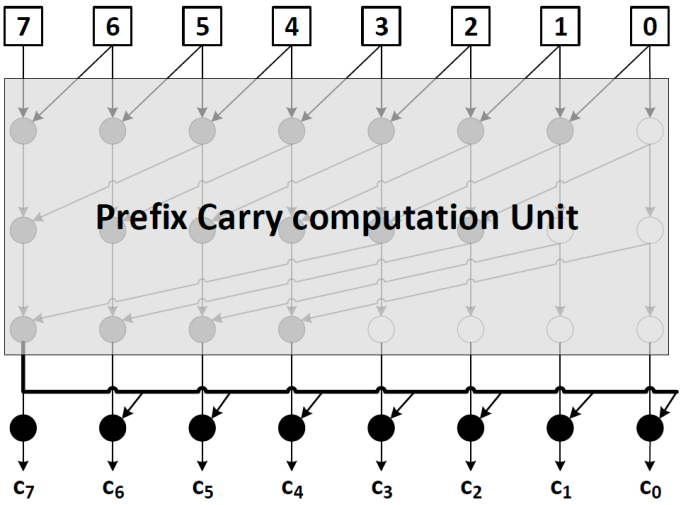
7.2 Prefix αθροιστές $2^n - 1$

Εξετάζεται η 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 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

$$(G'_i, P'_i) = (G_i + P_i G_{n-1}, P_i) \tag{7.3}$$



Εικόνα 7.3: Απλή δομή προθεματικού αθροιστή υπολοίπου $2^8 - 1$

Στην εικόνα 7.3 παρουσιάζεται η βασική δομή των αθροιστών υπολοίπου υλοποιημένη με προθεματικό αθροιστή σύμφωνα με τις εξισώσεις που αναπτύχθηκαν. Ένας σχεδιασμός σαν τον παραπάνω, εκτός από το γεγονός του επιπλέον επιπέδου, έχει και το μειονέκτημα στο ότι το κρατούμενο εξόδου του πρώτου τμήματος οδηγεί n κόμβους του τελευταίου επιπέδου, όπως εκφράζεται και την εξίσωση 7.3.

7.3 Αρχιτεκτονική βελτιστοποίηση

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

$$(G^*_i, P^*_i) = \begin{cases} (G_{n-1}, P_{n-1}), & i = -1 \\ (g_i, p_i) \circledast (G^*_{i-1}, P^*_{i-1}), & 0 \leq i \leq n-2 \end{cases} \tag{7.4}$$

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 7.4 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 αποδεικνύεται

$$\begin{aligned} (G^*_k, P^*_k) &= (g_k, p_k) \circledast (G^*_{k-1}, P^*_{k-1}) \\ &= (g_k, p_k) \circledast (c^*_{k-1}, P^*_{k-1}) \\ &= (g_k + p_k c^*_{k-1}, p_k P^*_{k-1}) \\ &= (c^*_k, P_k) \end{aligned}$$

   Σύμφωνα με τις παραπάνω ισότητες, 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

$$(G_i, P_i) \circledast (g, p) \circledast (G_i, P_i) = (G_i, P_i) \circledast (g, p) \tag{7.5}$$

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 7.4 αποδεικνύεται

$$\begin{aligned}(G_i^*, P_i^*) &= (g_i, p_i) \circledast (G_{i-1}^*, P_{i-1}^*) \\ &= (g_i, p_i) \circledast (g_{i-1}, p_{i-1}) \circledast ... \circledast (g_0, p_0) \circledast (G_{-1}^*, P_{-1}^*) \\ &= (g_i, p_i) \circledast ... \circledast (g_0, p_0) \circledast (G_{n-1}, P_{n-1}) \\ &= (g_i, p_i) \circledast ... \circledast (g_0, p_0) \circledast (g_{n-1}, p_{n-1}) \circledast (G_{n-2}, P_{n-2}) \\ &= (g_i, p_i) \circledast ... \circledast (g_0, p_0) \circledast (g_{n-1}, p_{n-1}) \circledast ... \circledast (g_i, p_i) \circledast ... \circledast (g_0, p_0) \\ &= (G_i, P_i) \circledast (g_{n-1}, p_{n-1}) \circledast ... \circledast (g_{i+1}, p_{i+1}) \circledast (G_i, P_i) \\ (G_i^*, P_i^*) &= (G_i, P_i) \circledast (g_{n-1}, p_{n-1}) \circledast ... \circledast (g_{i+1}, p_{i+1})\end{aligned} \tag{7.6}$$

Στην τελευταία ισότητα της παραπάνω διαδικασίας εφαρμόζεται ο ισχυρισμός της εξίσωσης 7.5 Η σχέση της εξίσωσης 7.6 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 $G_{i:i} = g_i$ και $G_{i:0} = c_i$. 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 προβλέπεται :

$$(G_{i:j}, P_{i:j}) = \begin{cases} (g_i, p_i) \circledast (g_{i-1}, p_{i-1}) \circledast ... \circledast (g_j, p_j), & i > j \\ (g_i, p_i), & i = j \\ (G_{i:0}, P_{i:0}) \circledast (G_{n-1:j}, P_{n-1:j}), & i < j \end{cases} \tag{7.7}$$

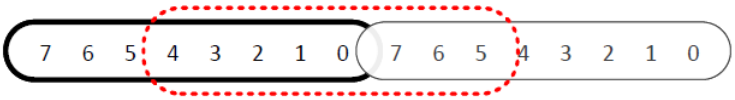
όπου n είναι το μέγεθος του αθροιστή. Έχοντας ορίσει την παραπάνω έκφραση, είναι αποδεκτή και μια εναλλακτική διατύπωση των σημάτων (G_i^*, P_i^*)

$$(G_i^*, P_i^*) = (G_{i:i+1}, P_{i:i+1}) \tag{7.8}$$

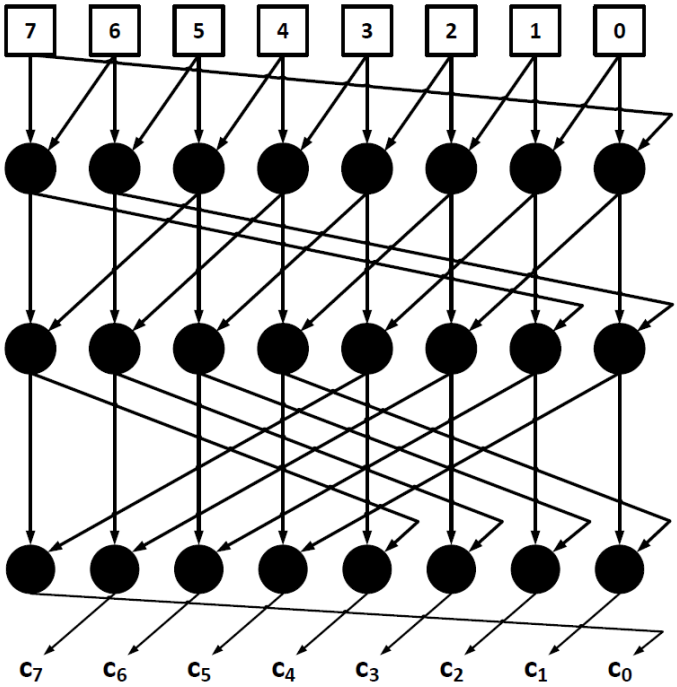
Το ίδιο ισχύει και για τους υπόλοιπους όρους (P,H,Q). Για παράδειγμα σε ένα αθροιστή των οκτώ δυαδικών ψηφίων $n = 8$, για τον υπολογισμό του $G_{1:6}$ υπονοείται η παρακάτω έκφραση :

$$G_{1:6} = g_1 + p_1 g_0 + p_1 p_0 g_7 + p_1 p_0 p_7 g_6$$

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. Στην παρακάτω εικόνα εφαρμόζεται η διαδικασία αυτή για το $G_{4:5} = G_4^*$. Είναι εμφανές πως το προηγούμενο στοιχείο του μηδέν είναι το $n - 1$, δηλαδή το επτά στην περίπτωση του $2^8 - 1$ αθροιστή υπολοίπου των 8-bits.



Στην εικόνα 7.4 παρουσιάζεται η αρχιτεκτονική ενός 8-bit αθροιστή υπολοίπου $2^n - 1$ με σκοπό την οπτικοποίηση της παραπάνω κυκλικής έκφρασης. Συγκρίνοντας αυτή την τοπολογία με αυτή της εικόνας 7.3, είναι εμφανές η διαφορά του επιπλέον επιπέδου.



Εικόνα 7.4: Παράλληλος προθεματικός αθροιστής υπολοίπου $2^8 - 1$

8 Ανάπτυξη Αθροιστών υπολοίπου $2^n - 1$

Σε αυτό το κεφάλαιο θα αναπτυχθούν συνολικά τριάντα-έξη αθροιστές υπολοίπου $2^n - 1$ ακολουθώντας την αρχιτεκτονική που παρουσιάστηκε στο κεφάλαιο 7 με τα ελάχιστα επίπεδα. Ανάλογα με το είδος της παραγοντοποίησης που τους εφαρμόζεται οι αθροιστές ομαδοποιούνται σε τρεις ομάδες, απλοί 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. Για κάθε αθροιστή επίσης εφαρμόζεται και η τακτική αραίωσης sparseness-2 και sparseness-4 για την μείωση των πόρων που καταλαμβάνει. 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].

8.1 Δομή και ανάπτυξή του κώδικα HDL

Για την περιγραφή των κυκλωμάτων δημιουργήθηκε ένα αρχείο verilog, στο οποίο περιγράφονται όλα τα δομικά στοιχεία από τα οποία δονούνται οι αθροιστές. Για κάθε δομικό στοιχείο-κόμβος υλοποιήθηκε ξεχωριστό module στην verilog που περιγράφει την λογική του συνάρτηση. Για παράδειγμα δημιουργήθηκε module για τον υπολογισμό των σημάτων propagate, generate και άθροισης χωρίς κρατούμενο. Ο συγκεκριμένος κόμβος είναι και αυτός που αποτελεί το πρώτο επίπεδο όλως των αθροιστών, έχει δύο εισόδους, την a και b, και τρεις εξόδους. Στον πίνακά A.1 του παραρτήματος Α δίνονται όλα τα modules που υλοποιήθηκαν μαζί με τις συναρτήσεις των εξόδων τους. Παρακάτω παρουσιάζεται η HDL περιγραφή των κόμβων του πρώτου επιπέδου όλων των αθροιστών, υπεύθυνα για τον υπολογισμό των σημάτων propagate (p), generate (g) και ημι-αθροίσματος (x).

```
1  module _gpx(a, b, g, p, x);
2
3      input a, b;
4      output g, p, x;
5
6      assign p = a | b ;    // propagate
7      assign g = a & b ;    // generate
8      assign x = a ^ b ;    // half-addition (witout carry in)
9
10 endmodule
```

Είναι προφανές πως για την περιγραφή ενός αθροιστή το μόνο που απαιτείται είναι η χρήση των κατάλληλων κόμβων, που αναφέρθηκαν στην προηγούμενη παράγραφο, με την σωστή σειρά και τις σωστές διασυνδέσεις μεταξύ των

άλλων κόμβων. Άρα ο αθροιστής δεν απαιτεί επιπλέον λογική από αυτή που έχει υλοποιηθεί ήδη, αρκεί να μην είναι ελλιπές το σύνολο των κόμβων που κατασκευάστηκε. Όσο αφορά, λοιπόν, τους αθροιστές, η μόνη διαδικασία που απομένει για την ανάπτυξη τους είναι η κλήση των modules-κόμβων και η ένωση τους με καλώδια.

Για τους αθροιστές υπολοίπου αναπτύχθηκε λογισμικό σε python, υπεύθυνο για την αυτοματοποίηση της γραφής του κώδικα. Στην ουσία το πρόγραμμα αυτό δέχεται τα ονόματα των έτοιμων verilog-modules κόμβων και τα εκτυπώνει σε ένα αρχείο με την σωστή σειρά και τις διασυνδέσεις καλωδίων που απαιτούνται για την ορθή λειτουργία του αθροιστή. Την αρχιτεκτονική την ορίζει ο χρήστης. Η αναγκαιότητα του λογισμικού αυτού είναι αισθητή στην περίπτωση αθροιστών μεγάλου μήκους εισόδου, όπου κάθε κόμβος καλείται αρκετές φορές, με αποτέλεσμα η διαδικασία γραφής του κώδικα να είναι χρονοβόρα και ευάλωτη σε λάθη.

8.2 Βασική δομή αθροιστών

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.

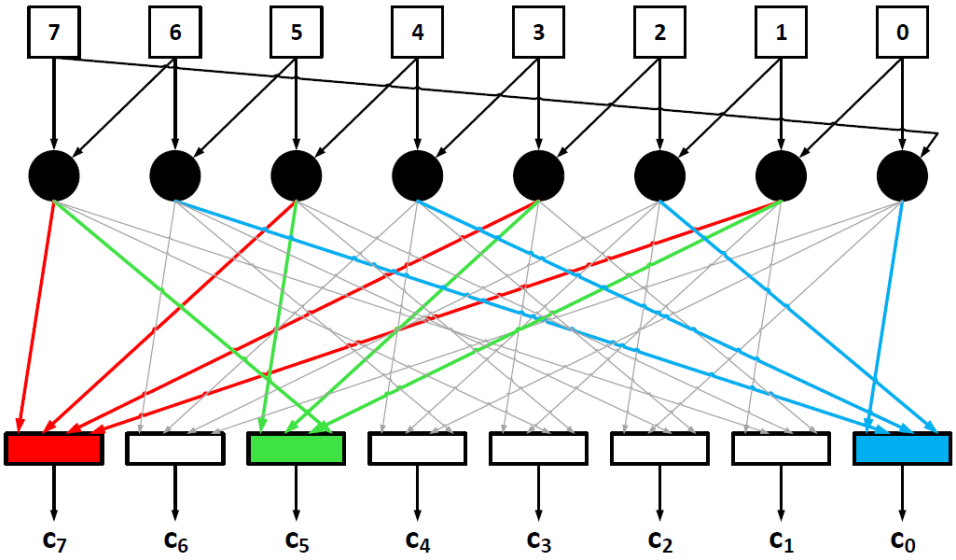
Όσο αφορά την επιλογή του πλήθους εισόδων στους κόμβους του κάθε επίπεδου αποφεύγεται η επιλογή του Radix-2 διότι, αναδιατυπώνοντας, οι αντίστοιχες συναρτήσεις κατά Jacskon δεν διαφέρουν με αυτές ενός απλού Prefix. Θα ήταν αρκετά αποδοτική η υλοποίηση ορισμένων επιπέδων με Radix-3, όπως και στους αθροιστές 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 των κόμβων αυτών πιο απλές, θα υπάρχουν περισσότερες αντιστοιχίες των έτοιμων 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 αφαιρείται το προνόμιο επαναχρησιμοποίησης κόμβων, με αποτέλεσμα η πολυπλοκότητα των αθροιστών καθώς και η επιφάνεια και κατανάλωση να είναι αρκετά μεγαλύτερες.

Σύμφωνα με αυτά που προ-ειπώθηκαν 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	Radix κάθε επιπέδου
8-bit	2x4
16-bit	4x4
32-bit	2x4x4
64-bit	4x4x4

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

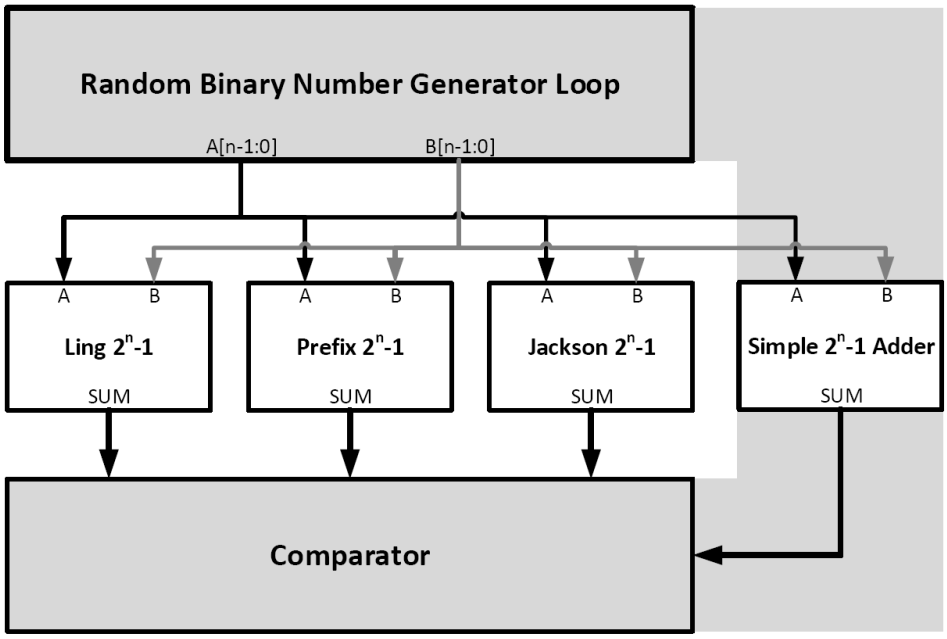
Στην εικόνα 8.1 παρουσιάζεται η βασική δομή των αθροιστών υπολοίπου $2^8 - 1$ που θα χρησιμοποιηθεί παρακάτω. Ενώ η δομή είναι κοινή και για τα τρία είδη αθροιστών το κάθε σχήμα-κόμβος αντιπροσωπεύει διαφορετική λογική συνάρτηση. Επίσης στο δέντρο αυτό ακολουθείται η κατασκευή μόνο των σημάτων που οδηγούν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



Εικόνα 8.1: Αρχιτεκτονική του $2^n - 1$ αθροιστή υπολοίπου

8.3 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.

Παρακάτω παρουσιάζεται η έξοδος του συστήματος προσομοίωσης για έλεγχο.

```
-------------------------------------------------------------------
 Length   | Architecture  | Rate                        | Results
-------------------------------------------------------------------
Prefix Adders :
-------------------------------------------------------------------
 8  - bit | 2x4           |       10000/      10000     | PASS
 16 - bit | 4x4           |       10000/      10000     | PASS
 32 - bit | 2x4x4         |       10000/      10000     | PASS
 64 - bit | 4x4x4         |       10000/      10000     | PASS
-------------------------------------------------------------------
Jackson Adders :
-------------------------------------------------------------------
 8  - bit | 2x4           |       10000/      10000     | PASS
 16 - bit | 4x4           |       10000/      10000     | PASS
 32 - bit | 2x4x4         |       10000/      10000     | PASS
 32 - bit | 4x4x2         |       10000/      10000     | PASS
 64 - bit | 4x4x4         |       10000/      10000     | PASS
-------------------------------------------------------------------
Ling Adders :
-------------------------------------------------------------------
 8  - bit | 2x4           |       10000/      10000     | PASS
 16 - bit | 4x4           |       10000/      10000     | PASS
 32 - bit | 2x4x4         |       10000/      10000     | PASS
 64 - bit | 4x4x4         |       10000/      10000     | PASS
-------------------------------------------------------------------
```

8.4 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 $G3_{i:i-k+1} = G_{i:i-63}$.

8.4.1 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

8.4.2 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

8.4.3 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 των 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 Μετρήσεις και Αποτελέσματα

Σε αυτό το κεφάλαιο παρουσιάζονται οι μετρήσεις χρόνου, χώρου και κατανάλωσης των αθροιστών που περιγράφηκαν στο προηγούμενο εδάφιο. Για τις μετρήσεις αυτές χρησιμοποιήθηκε το εργαλείο σύνθεσης Design Compiler, σε τοπογραφική και μη ρύθμιση.

9.1 Εργαλείο Μετρήσεων

Η λογική σύνθεση είναι μια υπό-διαδικασία του ηλεκτρονικού αυτόματου σχεδιασμού, κατά την οποία η αφηρημένη περιγραφή του ψηφιακού συστήματος μετατρέπεται σε κύκλωμα λογικών πυλών. Λογισμικά υπεύθυνα για αυτή την διαδικασία ονομάζονται εργαλεία σύνθεσις (Synthesis tools) και είτε παράγουν bitstreams για FPGAs είτε τα απαραίτητα αρχεία για την δημιουργία ASICs. Σε κάθε περίπτωση παράγουν επίσης κάποιες εκτιμήσεις σχετικά με την ταχύτητα, αξιοποίηση πόρων, κατανάλωση ενέργειας και άλλα.

Ένα από τα δημοφιλέστερα εργαλεία για σύνθεση αποτελεί το Design Compiler της Synopsys, το οποίο θα χρησιμοποιηθεί και για τις μετρήσεις των αθροιστών που αναπτύχθηκαν σε αυτή την διαδικασία. Το Design Compiler ουσιαστικα μεταφράζει τον HDL κώδικα σε λογικές μονάδες, υλοποιώντας μια διαδικασία αντιστοίχησης από των κώδικα στο φυσικό επίπεδο.

Οι λογικές μονάδες, πάνω στις οποίες θα αποτυπωθεί το κύκλωμα που σχεδιάστηκε, αποτελούνται από μικρά και απλά λογικά κυκλώματα, τα οποία μπορεί να περιέχουν μια λογική πύλη, ή να υλοποιούν μια απλή λογική συνάρτηση, όπως για παράδειγμα το λογικό στοιχείο AO21X1 το οποίο περιγράφεται απο την λογική συνάρτηση

$$Q = (IN1 * IN2) + IN3$$

Αυτά τα στοιχεία έχουν υλοποιηθεί σε φυσικό επίπεδο και έχει μετρηθεί αναλυτικά η συμπεριφορά τους. Μία τεχνολογική βιβλιοθήκη είναι ένα σύνολο από λογικά στοιχεία στην μορφή που περιγράφηκε παραπάνω.

Για την σύνθεση των αθροιστών έγινε χρήση της βιβλιοθήκης "Digital Standard Cell Library" των 32nm της Synopsys. Στην βιβλιοθήκη αυτή υλοποιούνται βασικές πύλες όπως AND, NAND, OR, NOR, XOR, XNOR, καθώς και άπλες λογικές συναρτήσεις όπως AND-OR, OR-AND, πολυπλέκτες δυο σε ένα, αποκωδικοποιητές, ήμι-αθροιστές και πλήρη αθροιστές και πολλά άλλα στοιχεία. Για κάθε ένα λογικό κελί που περιέχει η βιβλιοθήκη δίνεται μια αναλυτική αναφορά για τους χρόνους καθυστέρησης και την κατανάλωση ενέργειας ανάλογα με την συμπεριφορά της εισόδου, το εμβαδόν που καταλαμβάνει το λογικό κύκλωμά, καθώς και άλλες πληροφορίες που αφορούν την συχνότητα λειτουργίας, θερμοκρασία, τάση και χωρητικότητα.

9.2 Τακτική Μεταγλώττισης

Για λόγους βελτιστοποίησης απαιτείται η κατάλληλη επιλογή της τακτικής μεταγλώττισης (compile strategy) από το εργαλείο σύνθεσης. Το Design Compiler προσφέρει τις παρακάτω επιλογές :

- Top-Down Compilation
  Ο υψηλότερος ιεραρχικά σχεδιασμός μεταγλωττίζεται μαζί με τους επιμέρους από τους οποίους αποτελείται.

- Bottom-Up Compilation
  Κάθε sub-module μεταγλωττίζεται ξεκινώντας το κατώτερο ιεραρχικά επίπεδο και συνεχίζοντας στα επόμενα επίπεδα μέχρι να μεταγλωττιστεί και ο υψηλότερος ιεραρχικά σχεδιασμός.

- Mixed Compilation
  Συνδυασμός των παραπάνω δύο.

Για τους σκοπούς της παρούσας εργασίας χρησιμοποιείται η τεχνική Bottom-up, αποσκοπώντας στον περιορισμό των βελτιστοποιήσεων που παρέχει το εργαλείο μόνο στο εσωτερικό των κόμβων, που είναι και το χαμηλότερο επίπεδο της ιεραρχίας. Πέρα από τους κόμβους δεν επιτρέπεται καμία άλλη συναρτησιακή βελτιστοποίηση. Με αυτήν την στρατηγική το εργαλείο δεν έχει την δυνατότητα τροποποίησης της αρχιτεκτονικής των αθροιστών που αναπτύσσονται.

Επιπλέον, το εργαλείο της Synopsys προσφέρει την δυνατότητα τοπογραφικής μεταγλώττισης και μη. Στην περίπτωση μη-τοπογραφικής μεταγλώττισης αξιοποιούνται οι προδιαγραφές των κυκλωμάτων που παρέχονται από την βιβλιοθήκη και συνυπολογίζονται και επιπλέον λογικές που προστίθενται για την σωστή οδήγηση των πυλών. Στην περίπτωση της τοπογραφικής μεταγλώττισης συνυπολογίζονται στα μοντέλα σύνθεσης οι καθυστερήσεις και ο χώρος των καλωδίων μεταξύ των έτοιμων σχεδιασμένων κυκλωμάτων που παρέχονται από την βιβλιοθήκη, καθώς και η τοποθέτηση των κόμβων.

9.3 Μετρήσεις

Στην παράγραφο αυτή καταγράφονται τα αποτελέσματα των, τοπογραφικών και μη, μετρήσεων των αθροιστών σε μορφή πινάκων. Σημειώνεται πως για τις μετρήσεις δεν δόθηκε κανένας περιορισμός στο εργαλείο σύνθεσης. Οι μονάδες μέτρησης καθυστέρησης, εμβαδού και κατανάλωση ενέργειας είναι ns, nm^2 και pw αντίστοιχα.

9.3.1 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

9.3.2 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

9.3.3 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	0.73	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	0.67	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	0.85	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

9.3.4 Τοπογραφικές Μετρήσεις

	Delay	Area	Power
Prefix	0.289	166	43.56
Ling	0.313	178	48.60
Jackson	0.335	215	61.85

Πίνακας 9.13: Τοπογραφικές Μετρήσεις 8-bit

	Delay	Area	Power
Prefix	0.439	943	239.80
Ling	0.462	992	261.15
Jackson	0.467	1268	357.11

Πίνακας 9.15: Τοπογραφικές Μετρήσεις 32-bit

	Delay	Area	Power
Prefix	0.384	398	102.85
Ling	0.404	422	117.99
Jackson	0.397	455	133.82

Πίνακας 9.14: Τοπογραφικές Μετρήσεις 16-bit

	Delay	Area	Power
Prefix	0.548	2147	522.28
Ling	0.583	2244	587.19
Jackson	0.581	2960	821.75

Πίνακας 9.16: Τοπογραφικές Μετρήσεις 64-bit

9.3.5 Τοπογραφικές Μετρήσεις sparse-2

	Delay	Area	Power
Prefix	0.330	134	35.12
Ling	0.335	143	38.74
Jackson	0.343	161	44.82

Πίνακας 9.17: Τοπογραφικές sparse-2 Μετρήσεις 8-bit

	Delay	Area	Power
Prefix	0.423	300	78.92
Ling	0.421	319	89.23
Jackson	0.426	355	102.14

Πίνακας 9.18: Τοπογραφικές sparse-2 Μετρήσεις 16-bit

	Delay	Area	Power
Prefix	0.476	675	172.43
Ling	0.486	711	189.33
Jackson	0.465	849	232.75

Πίνακας 9.19: Τοπογραφικές sparse-2 Μετρήσεις 32-bit

	Delay	Area	Power
Prefix	0.571	1480	368.42
Ling	0.567	1553	407.71
Jackson	0.559	1911	516.89

Πίνακας 9.20: Τοπογραφικές sparse-2 Μετρήσεις 64-bit

9.3.6 Τοπογραφικές Μετρήσεις sparse-4

	Delay	Area	Power
Prefix	0.429	127	33.38
Ling	0.419	130	34.49
Jackson	0.427	150	42.43

Πίνακας 9.21: Τοπογραφικές sparse-4 Μετρήσεις 8-bit

	Delay	Area	Power
Prefix	0.499	252	66.87
Ling	0.501	261	71.30
Jackson	0.505	279	77.04

Πίνακας 9.22: Τοπογραφικές sparse-4 Μετρήσεις 16-bit

	Delay	Area	Power
Prefix	0.560	577	151.20
Ling	0.558	591	157.13
Jackson	0.550	660	180.03

Πίνακας 9.23: Τοπογραφικές sparse-4 Μετρήσεις 32-bit

	Delay	Area	Power
Prefix	0.649	1146	297.58
Ling	0.642	1183	313.25
Jackson	0.641	1476	392.74

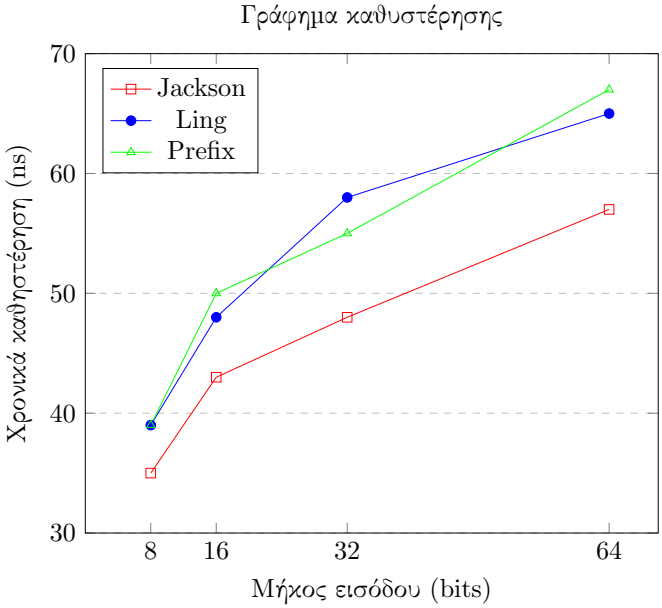
Πίνακας 9.24: Τοπογραφικές sparse-4 Μετρήσεις 64-bit

9.4 Ανάλυση αποτελεσμάτων

Τα αποτελέσματα των μετρήσεων, όπως ήταν και αναμενόμενο, διαφέρουν ανάλογα με τον τρόπο μεταγλώττισης (τοπογραφικό και μη). Στην συνέχεια σχολιάζονται και οι δύο περιπτώσεις όσο αφορά την απόδοση.

9.4.1 Σχολιασμός μη-τοπογραφικών αποτελεσμάτων

Σύμφωνα με τα αποτελέσματα των μη-τοπογραφικών μετρήσεων, παρατηρείται θετική απόδοση της ταχύτητας σε όλες τις περιπτώσεις. Η αναμενόμενη κατάταξή, όσο αφορά την ταχύτητα, ξεκινά με μεγαλύτερη καθυστέρηση στις απλές τοπολογίες προθεματικών αθροιστών υπολοίπου, και καταλήγει στις τοπολογίες κατά Jackson. Σε παρελθοντικές έρευνες [DN05] και [Dim+05] έχει αποδειχθεί πως οι αθροιστές υπολοίπου $2^n - 1$ κατά ling είναι πιο αποδοτικοί από τους simple-prefix. Στα αποτελέσματα, όπως φαίνεται και στο γράφημα της εικόνας 9.1, υπάρχουν περιπτώσεις όπου οι Ling αθροιστές είναι στην χειρότερη περίπτωση.

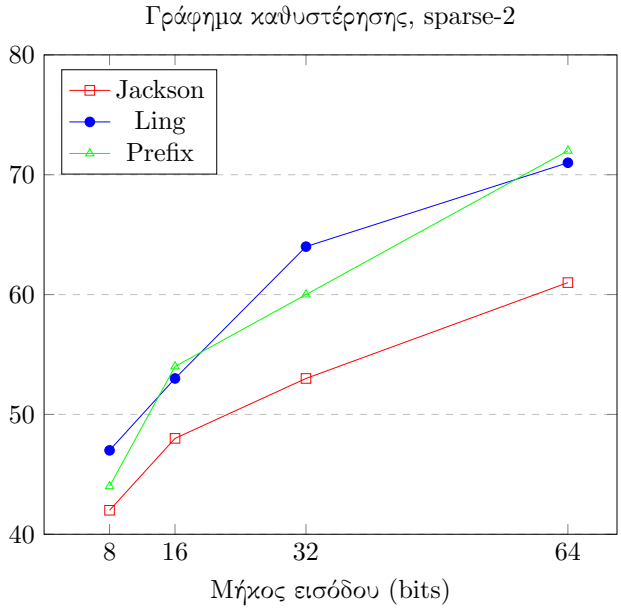


Εικόνα 9.1: Γράφημα καθυστέρησης - μήκος εισόδου αθροιστών υπολοίπου

Αυτό δεν σημαίνει πως οι μετρήσεις είναι άκυρες. Ο βασικός λόγος που προκαλεί την διαφορά αυτή είναι η επιλογή της βιβλιοθήκης. Οι μετρήσεις της παρούσας εργασίας πραγματοποιήθηκαν σε τεχνολογία 32nm, ενώ οι μετρήσεις που παρουσιάζονται σε προηγούμενες δημοσιεύσεις συγκρίνοντας τους προθεματικούς αθροιστές με αθροιστές Ling βασίζονται σε τεχνολογίες μεγαλύτερες των 100nm.

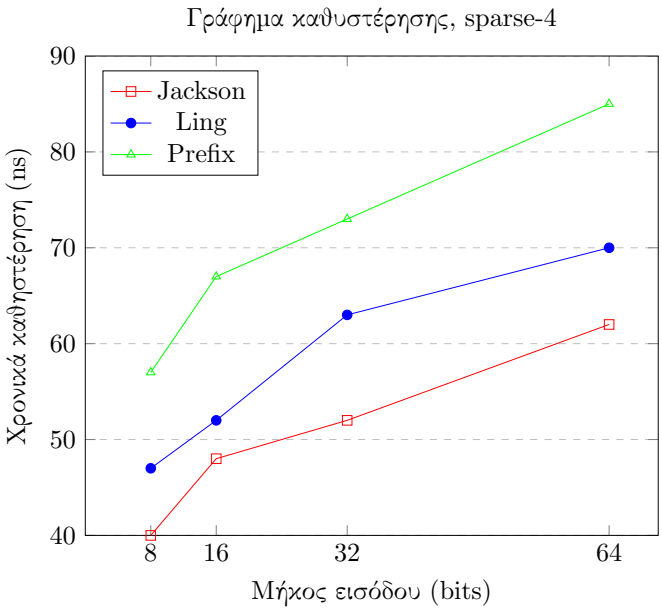
Όσο αφορά την επιτάχυνση των αθροιστών, όπου είναι και ο στόχος της παρούσας εργασίας, είναι εμφανές και από το διάγραμμα αλλά και από τους πίνακες των αποτελεσμάτων πως υπάρχει αισθητό προβάδισμα. Πρέπει όμως να σημειωθεί πως όσο αφορά την απαίτηση πόρων και ενέργειας τα αποτελέσματα είναι αρνητικά, κάτι που είναι επίσης αναμενόμενο εφόσον για την υλοποίηση της παραγοντοποίησης Jackson είναι αναγκαίο να εισαχθούν στο σχεδιασμό επιπλέον κόμβοι για τον υπολογισμό των σημάτων D.

Στις παραπάνω τρεις αρχιτεκτονικές προς σύγκριση, εφαρμόστηκε η ίδια ακριβός τεχνική αραίωσης, sparse-2 και sparse-4, με σκοπό την μείωση του εμβαδού. Στα γραφήματα 9.2 και 9.3 παρουσιάζονται τα διαγράμματα καθυστέρησης - μήκος εισόδου για sparse-2 και sparse-4 αντίστοιχα.



Εικόνα 9.2: Γράφημα καθυστέρησης - μήκος εισόδου αθροιστών υπολοίπου sparse-2

Η κατάταξη ταχύτητας δεν επηρεάζεται, αλλά κάθε αθροιστής επιβαρύνεται με ένα χρονικό διάστημα κερδίζοντας σε εμβαδόν. Αξιοσημείωτο είναι το αποτέλεσμα στην περίπτωση sparse-4, όπου οι αθροιστές παρουσιάζουν μεγαλύτερες διαφορές, όσο αφορά τον χρόνο, σε σχέση με τις δύο προηγούμενες περιπτώσεις, ενώ το εμβαδόν τους δεν έχει την αντίστοιχη απόκλιση.

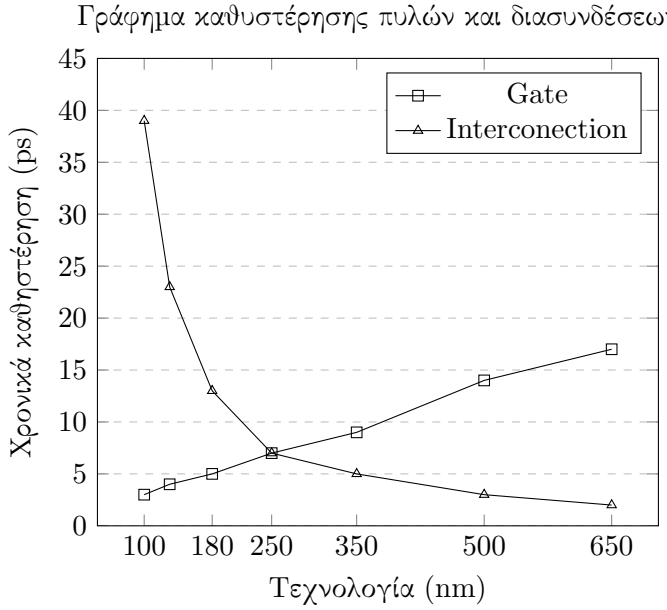


Εικόνα 9.3: Γράφημα καθυστέρησης - μήκος εισόδου αθροιστών υπολοίπου sparse-4

9.4.2 Σχολιασμός τοπογραφικών αποτελεσμάτων

Στην περίπτωση των τοπογραφικών μετρήσεων τα αποτελέσματα είναι αρνητικά κατά μέσο όρο με ελάχιστες εξαιρέσεις. Συγκεκριμένα στις sparse-2 και sparse-4 τεχνικές αραίωσης, στους αθροιστές των 32 και 64 δυαδικών ψηφίων παρατηρείται θετική απόδοση όσο αφορά την καθυστέρηση. Στις υπόλοιπες περιπτώσεις τα αποτελέσματα είναι αρνητικά και σε μερικές μη αναμενόμενα, διακρίνοντας ανάποδη χρονική κατάταξη.

Η προφανής εξήγηση των ανομοιόμορφων αποτελεσμάτων μεταξύ τοπογραφικών και μη μετρήσεων οφείλεται στην τεχνολογία που χρησιμοποιήθηκε. Σύμφωνα με έρευνες που πραγματοποιήθηκαν στο παρελθόν [Dar+17], καθώς και στην δημοσίευση του Saraswat και Mohammadi [SM82], διατυπώνεται πως στις προχωρημένες τεχνολογίες η καθυστέρηση λόγω των διασυνδέσεων έχει γίνει αρκετά σημαντικότερη από τις καθυστερήσεις των λογικών πυλών. Στην εικόνα 9.4 δίνεται ένα γράφημα που δείχνει την διαφορά αυτή χρησιμοποιώντας τα στοιχεία από τις έρευνες που αναφέρθηκαν.



Εικόνα 9.4: Γράφημα καθυστέρησης - μήκος εισόδου αθροιστών υπολοίπου

Στους αθροιστές Jackson σημειώνεται αρκετά μεγαλύτερο πλήθος κόμβων σε σχέση με τους υπόλοιπους δύο σχεδιασμούς. Συνεπάγεται, λοιπόν, πως και το πλήθος των διασυνδέσεων θα είναι μεγαλύτερο καθώς και λόγο δυσκολίας δρομολόγησης ίσως και πιο περίπλοκο.

9.5 Συμπεράσματα

Η χρονική απόδοση υπολογισμού του αθροίσματος υπολοίπου $2^n - 1$ στους αθροιστές, που αναπτύχθηκαν στην παρούσα εργασία, είναι θετική στις περισσότερες περιπτώσει. Εφόσον στις μη-τοπογραφικές μετρήσεις το αποτελέσματα είναι πάντα θετικά, η τελικές προδιαγραφές των Jackson $2^n - 1$ που αναπτύχθηκαν εξαρτώνται από την βιβλιοθήκη τεχνολογίας. Όσο μεγαλύτερο το τρανζίστορ της τεχνολογίας, τόσο πιο αμελητέα η αντίσταση των καλωδίων καθώς και άλλων φυσικών παραμέτρων που επηρεάζουν την ταχύτητα. Η χρήση τους είναι κατάλληλη όταν οι περιορισμοί διαθέσιμων πόρων και κατανάλωση ενέργειας είναι ελαστικοί.

A Παράρτημα - Node Modules

Symbol	Function
_gpx	$g_i = a_i * b_i$ $p_i = a_i + b_i$ $x_i = a_i \oplus b_i$
_4g2p_R4	$R = g_3 + g_2 + p_1 g_1 + p_1 p_0 g_0$
_4p_Q4	$Q = p_3 p_2 p_1 p_0$
_4R2Q_R4	$R = R_3 + R_2 + Q_1 R_1 + Q_1 Q_0 R_0$
_D16	$D = p_1 R + p_0 Q$
_Jsum	$sum = R\ ?\ x \oplus D\ :\ x$
_1R4Q_Q4	$Q = Q_3 Q_2 Q_1 (Q_0 + R)$
_D64_1	$D = g_1 + p_2 g_0 + p_2 p_1 p_0$
_D64_2	$D = D(R + Q)$
_2g_R2	$R = g_1 + g_0$
_2p_Q2	$Q = p_1 p_0$
_4g2p_H4	$H = g_3 + g_2 + p_1 g_1 + p_1 p_0 g_0$
_P4	$P = p_3 p_2 p_1 p_0$
_4G3P_G4	$G = G_3 + P_2 G_2 + P_2 P_1 G_1 + P_2 P_1 P_0 G_0$
_Lsum	$sum = H\ ?\ x \oplus p\ :\ x$
_2g_H2	$H = g_1 + g_0$
_2p_P2	$P = p_1 p_0$
_Psum	$sum = G \oplus x$
_2g1p_G2	$G = g_1 + p * g_0$
_2R1Q_R2	$R = R_1 + Q * R_0$

Πίνακας A.1: Modules Description

Βιβλιογραφία

[Skl60] J. Sklansky. “Conditional-Sum Addition Logic”. In: *IRE Transactions on Electronic Computers* EC-9.2 (June 1960), pp. 226–231. ISSN: 0367-9950. DOI: 10.1109/TEC.1960.5219822.

[KS73] P. M. Kogge and H. S. Stone. “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations”. In: *IEEE Transactions on Computers* C-22.8 (Aug. 1973), pp. 786–793. ISSN: 0018-9340. DOI: 10.1109/TC.1973.5009159.

[She77] J. J. Shedletsky. “Comment on the Sequential and Indeterminate Behavior of an End-Around-Carry Adder”. In: *IEEE Transactions on Computers* C-26.3 (Mar. 1977), pp. 271–272. ISSN: 0018-9340. DOI: 10.1109/TC.1977.1674817.

[LF80] Richard E. Ladner and Michael J. Fischer. “Parallel Prefix Computation”. In: *J. ACM* 27.4 (Oct. 1980), pp. 831–838. ISSN: 0004-5411. DOI: 10.1145/322217.322232. URL: http://doi.acm.org/10.1145/322217.322232.

[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.

[BK82] R. P. Brent and H. T. Kung. “A Regular Layout for Parallel Adders”. In: *IEEE Trans. Comput.* 31.3 (Mar. 1982), pp. 260–264. ISSN: 0018-9340. DOI: 10.1109/TC.1982.1675982. URL: http://dx.doi.org/10.1109/TC.1982.1675982.

[SM82] K. C. Saraswat and F. Mohammadi. “Effect of Scaling of Interconnections on the Time Delay of VLSI Circuits”. In: *IEEE Journal of Solid-State Circuits* 17.2 (Apr. 1982), pp. 275–280. ISSN: 0018-9200. DOI: 10.1109/JSSC.1982.1051729.

[ENK94] C. Efstathiou, D. Nikolos, and J. Kalamatianos. “Area-time efficient modulo 2n-1 adder design”. In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 41.7 (July 1994), pp. 463–467. ISSN: 1057-7130. DOI: 10.1109/82.298378.

[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] G. Dimitrakopoulos and D. Nikolos. “High-speed parallel-prefix VLSI Ling adders”. In: *IEEE Transactions on Computers* 54.2 (Feb. 2005), pp. 225–231. ISSN: 0018-9340. DOI: 10.1109/TC.2005.26.

[Dim+05] G. Dimitrakopoulos et al. “New architectures for modulo 2N - 1 adders”. In: *2005 12th IEEE International Conference on Electronics, Circuits and Systems*. Dec. 2005, pp. 1–4. DOI: 10.1109/ICECS.2005.4633502.

[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.

[Dar+17] Mohammed Darmi et al. “Integrated Circuit Conception: A Wire Optimization Technic Reducing Interconnection Delay in Advanced Technology Nodes”. In: *Electronics* 6.4 (2017). ISSN: 2079-9292. DOI: 10.3390/electronics6040078. URL: http://www.mdpi.com/2079-9292/6/4/78.