ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών

Αποδοτική σχεδίαση μιγαδικού πολλαπλασιαστή σε ASIC και FPGA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΦΩΤΙΟΥ ΝΤΟΥΣΚΑ

Επιβλέπων : Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αποδοτική σχεδίαση μιγαδικού πολλαπλασιαστή σε ASIC και FPGA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΦΩΤΙΟΥ ΝΤΟΥΣΚΑ

Επιβλέπων : Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Ιουλίου 2014.

.....	.....	.....
Κιαμάλ Πεκμεστζή Καθηγητής Ε.Μ.Π.	Δημήτριος Σούντρης Καθηγητής Ε.Μ.Π.	Γιώργος Οικονομάκος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

.....

ΦΩΤΙΟΣ ΝΤΟΥΣΚΑΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας ήταν η σχεδίαση διαφόρων τοπολογιών μιγαδικών πολλαπλασιαστών και η διερεύνησή τους ως προς τις επιδόσεις τους στην επιφάνεια, κατανάλωση και καθυστέρηση σε ASIC και FPGA.

Υλοποιήθηκαν έξι τοπολογίες βασισμένες σε δύο αλγόριθμους πολλαπλασιασμού δύο μιγαδικών αριθμών, τον συμβατικό αλγόριθμο και τον αλγόριθμο του Gauss. Ταυτόχρονα, υλοποιήθηκαν δύο διαφορετικές μονάδες επανακωδικοποίησης για την υλοποίηση της πράξης X∗(A+B). Σχεδιάστηκαν διαφορετικές αρχιτεκτονικές υλοποίησής τους και διερευνήθηκε ποια είναι η αποδοτικότερη.

Συγκεκριμένα, υλοποίηθηκαν σε γλώσσα Verilog δύο διαφορετικά κυκλώματα ενός μιγαδικού πολλαπλασιαστή σύμφωνα με τον συμβατικό αλγόριθμο και τέσσερα διαφορετικά κυκλώματα σύμφωνα με τον αλγόριθμο του Gauss σε μήκη λέξεως 8 ως 64 bits. Τα κυκλώματα προσομοιώθηκαν, συντέθηκαν και εφαρμόστηκαν σε ASIC και FPGA.

Τα αποτελέσματα καθυστέρησης του κρίσιμου μονοπατιού, κατανάλωσης και επιφάνειας χρησιμοποιήθηκαν για την σύγκριση των υλοποιήσεων με σκοπό την εύρεση της αποδοτικότερης υλοποίησης στο κάθε μέσο (ASIC και FPGA). Επίσης διενεργήθηκε μια συγκριτική αξιολόγηση των αποτελεσμάτων στα δύο μέσα, με σκοπό την διερεύνηση της επίδρασης του μέσου στην απόδοση της υλοποίησης.

Λέξεις Κλειδιά: Μιγαδικός πολλαπλασιαστής, Modified Booth, Αλγόριθμος του Gauss, Επανακωδικοποιητής Πρόσθεσης-Πολλαπλασιασμού, ASIC, FPGA

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The scope of this thesis was the design of various complex multiplier schemes, as well as the investigation of their performance, on area, energy consumption and critical path delay in ASIC and FPGA.

A total of six (6) topologies were implemented based on two complex multiplication algorithms, the conventional and Gauss's complex multiplication algorithm. Moreover, two different recoding units where implemented to execute the X*(A+B) operation and architectures to encompass all of the aforementioned functionality were designed and their respective performances where analysed in order to determine the most efficient.

More specifically, the first two topologies are designed on the conventional multiplication algorithm and the subsequent four used Gauss's complex multiplication algorithm using Verilog HDL, while the inputs can be 8, 16, 24, 32, 48, or 64 bits wide. The designs were simulated, synthesised and implemented in ASIC and FPGA.

The energy, delay and area results are used to compare these schemes to determine the most efficient in each device (ASIC and FPGA). Also there is a comparison of the results on these two devices in order to examine their effect on the efficiency of the scheme.

Keywaords: Complex Multiplier, Modified Booth, Gauss's Complex Multiplication Algorithm, Add-Multiply Operator Recoder, ASIC, FPGA

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1. Εισαγωγή

1.1. Ψηφιακή σχεδίαση και αποδοτικές υλοποιήσεις

Η αλματώδης διείσδυση της τεχνολογίας σε όλους τους τομείς της ανθρώπινης καθημερινότητας έχει ως αποτέλεσμα τη συνεχή εξέλιξη των ψηφιακών συστημάτων με σκοπό να καλύψουν τις ολοένα αυξανόμενες ανάγκες των εφαρμογών.

Στον τομέα της ψηφιακής σχεδίασης η εξέλιξη αυτή εξαρτάται από την καθυστέρηση, την επιφάνεια και την κατανάλωση των ψηφιακών κυκλωματών. Οι περιορισμοί που προέκυψαν όσων αφορά τις παραπάνω παραμέτρους, κυρίως λόγω της χρήσης των ψηφιακών κυκλωμάτων σε φορητές συσκευές με χαρακτηριστικά τους, το μικρό μέγεθος, την τροφοδοσία τους από μπαταρία καθώς και ο αυξημένος υπολογιστικός φόρτος εργασίας των συσκευών αυτών, οδήγησε στην προσπάθεια εύρεσης λύσεων που να αφορούν και τις τρεις παραπάνω παραμέτρους, σε αντίθεση με μια παλαιότερη λογική που επικεντρωνόταν στην κάθε μία ξεχωριστά.

Ο τομέας της ψηφιακής επεξεργασίας σήματος (DSP) ασχολείται με τον μαθηματικό χειρισμό ενός σήματος και τους τρόπους επεξεργασίας του. Κάποιες από τις βασικότερες εφαρμογές της ψηφιακής επεξεργασίας σημάτος είναι η επεξεργασία ήχου και ομιλίας, η επεξεργασία εικόνας και γενικότερα πληροφοριών με εφαρμογές στις τηλεπικοινωνίες, στον αυτόματο έλεγχο, στην βιοϊατρική και σε πολλούς άλλους τομείς. Στα συστήματα ψηφιακής επεξεργασίας σήματος βασικό στοιχείο αποτελεί ο αλγόριθμος Γρήγορου Μετασχηματισμού Fourier (Fast Fourier Transform) ο οποίος βασίζεται στον πολλαπλασιασμό δύο μιγαδικών αριθμών.

Ο μιγαδικός πολλαπλασιαστής έχει ενδιαφέρον σε επίπεδο αλγόριθμου και σε τεχνολογικό επίπεδο για τον σχεδιασμό αποδοτικότερων κυκλωμάτων υλοποίησής του. Ο σημαντικότερος λόγος για τον οποίο έχει ενδιαφέρον ο μιγαδικός πολλαπλασιαστής είναι η έλλειψη έρευνας για την λειτουργία του αλλά κυρίως για την υλοποίησή του με την βοήθεια των σημερινών τεχνολογιών και εργαλείων που καθιστούν παλαιότερες μελέτες ξεπερασμένες.

1.2. Αντικείμενο διπλωματικής

Η παρούσα διπλωματική έχει ως αντικείμενο την διερεύνηση διάφορων μιγαδικών πολλαπλασιαστών που υλοποιούνται με δύο διαφορετικούς αλγόριθμους, τον συμβατικό και τον αλγόριθμο του Gauss, καθώς και διαφορετικές παραλλαγές αυτών των υλοποίησεων τόσο στην αρχιτεκτονική που ακολουθήσαμε όσο και στον τρόπο σχεδίασης των κυκλωμάτων.

Συνοπτικά, σχεδιάσαμε δύο διαφορετικές παραλλαγές ενός μιγαδικού πολλαπλασιαστή σύμφωνα με τον συμβατικό αλγόριθμο και τέσσερις διαφορετικές υλοποιήσεις του μιγαδικού πολλαπλασιαστή σύμφωνα με τον αλγόριθμο του Gauss. Στις υλοποιήσεις του μιγαδικού πολλαπλασιαστή με τον συμβατικό αλγόριθμο χρησιμοποιήσαμε δύο διαφορετικές αρχιτεκτονικές με σκοπό την εύρεση της αποδοτικότερης σε καθυστέρηση, κατανάλωση και επιφάνεια. Στις υλοποιήσεις όπου κάνουμε χρήση του αλγόριθμου του Gauss, χρησιμοποιήσαμε δύο μεθόδους εφαρμογής του αλγόριθμου όπως αυτοί έχουν περιγραφεί σε δύο διαφορετικές δημοσιεύσεις και στην μία από αυτές σχεδιάσαμε τρεις παραλλαγές της μεθόδου αυτής, για τη εύρεση της καταλληλότερης αρχιτεκτονικής για την υλοποίησή της. Ο λόγος που διαλέξαμε την δημοσίευση αυτή για να βρούμε την καταλληλότερη αρχιτεκτονική, είναι γιατί βασίζεται σε μια δημοσίευση που έγινε τον προηγούμενο χρόνο και ως πιο σύγχρονη, θεωρήσαμε ότι θα εκμεταλλευεται καλύτερα τις δυνατότητες των σύγρονων εργαλείων ψηφιακής σχεδίασης.

Στο πλαίσιο της εργασίας μας, υλοποίησαμε τα διαφορετικά κυκλώματα του μιγαδικού πολλαπλασιαστή σε γλώσσα Verilog σε διάφορα μήκη λέξεως (8 ως και 64 bits), όπου ο απλός πολλαπλασιασμός έγινε με χρήση της κωδικοποίησης Modified Booth σε radix 4. Η λειτουργία των κυκλωμάτων αυτών προσομοιώθηκε και επαληθεύτηκε με το περιβάλλον Modelsim. Η σύνθεση των κυκλωμάτων σε ASIC έγινε με χρήση του Synopsys Design Compiler και τα αποτελέσματα καθυστέρησης, επιφάνειας και κατανάλωσης εξάχθηκαν από το Synopsys Design Compiler και το Synopsys Primepower. Στην συνέχεια, μεταφέραμε την υλοποίηση των συστημάτων πάνω σε ένα FPGA. Συγκεκριμένα επιλέξαμε την οικογένεια Virtex 7 της Xilinx, όπου η σύνθεση και εφαρμογή του κώδικα έγινε με την χρήση της σουίτας ISE 14.7 της Xilinx, από την οποία εξάγαμε τις απαραίτητες μετρήσεις καθυστέρησης και επιφάνειας. Οι απαραίτητες μετρήσεις κατανάλωσης εξάχθηκαν με χρήση του XPower Analyzer της Xilinx. Τέλος πραγματοποιήσαμε μία σύγκριση της απόδοσης των υλοποιήσεων του μιγαδικού πολλαπλασιαστή ανάλογα με το μέσο στο οποίο εφαρμόστηκαν και πως επηρεάζεται η χρήση των διαφορετικών αλγόριθμων και αρχιτεκτονικών, από τα μέσα αυτά, αλλά και τα εργαλεία που τα συνοδεύουν.

1.2.1. Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε τα προβλήματα της σχεδίασης αριθμητικών κυκλωμάτων.
2. Υλοποιήσαμε δύο διαφορετικά σχήματα του μιγαδικού πολλαπλασιαστή με χρήση δύο διαφορετικών αλγόριθμων, του συμβατικού και του αλγόριθμου του Gauss, καθώς και τέσσερις παραλλαγές των αρχικών υλοποιήσεων, οι οποίες μπορούν να παραχθούν παραμετρικά για διάφορα μήκη λέξης.
3. Αξιολογήσαμε τα πλεονεκτήματα και τα μειονεκτήματα των δύο αλγόριθμων υλοποίησης του μιγαδικού πολλαπλασιαστή.
4. Αξιολογήσαμε την απόδοση και τα πλεονεκτήματα/μειονεκτήματα κάθε σχήματος και αρχιτεκτονικής που σχεδιάσαμε.
5. Παρουσιάσαμε τα συγκριτικά αποτελέσματα των διαφορετικών υλοποιήσεων του μιγαδικού πολλαπλασιαστή στα 90nm ως εγχειρίδιο αναφοράς.
6. Μεταφέραμε αυτές τις υλοποιήσεις σε ένα FPGA, σε τεχνολογία 28nm.
7. Συγκρίναμε την απόδοση των υλοποιήσεων σε ένα FPGA και μελετήσαμε την επίδραση του μέσου στην απόδοση αυτών.

1.3. Οργάνωση κειμένου

Στο κεφάλαιο 2 γίνεται μια παρουσίαση της θεωρίας των αριθμητικών συστημάτων, των κυκλωμάτων που θα χρησιμοποιηθούν, καθώς και των δύο αλγόριθμων υλοποίησης του μιγαδικού πολλαπλασιαστή.

Στο κεφάλαιο 3 αναλύεται συνοπτικά η πορεία του κατασκευαστικού μέρους της εργασίας με τις απαραίτητες αναφορές στις εργασίες στις οποίες βασίστηκε η σχεδίαση των κυκλωμάτων, και παρουσιάστηκαν οι διαφορετικές υλοποιήσεις του μιγαδικού πολλαπλασιαστή που σχεδιάσαμε.

Στο κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα των προσομοιώσεων καθώς και μια αναλυτική συγκριτική παρουσίαση των επιμέρους υλοποιήσεων.

Στο κεφάλαιο 5 γίνεται μια σύνοψη των συμπερασμάτων που προέκυψαν, και προτείνονται πιθανές εφαρμογές και μελλοντικές επεκτάσεις της εργασίας.

Στο κεφάλαιο 6 αναφέρεται η βιβλιογραφία που χρησιμοποιήθηκε κατά την εκπόνηση της παρούσας εργασίας.

2. Θεωρητικό υπόβαθρο

Σε αυτό το κεφάλαιο καλύπτεται το θεωρητικό υπόβαθρο το οποίο είναι απαραίτητο ώστε να γίνει κατανοητή η παρούσα διπλωματική εργασία από τον αναγνώστη.

Αρχικά, παρουσιάζεται το δυαδικό σύστημα αρίθμησης, και πιο συγκεκριμένα η αναπαράσταση αριθμών στη μορφή συμπληρώματος του δύο, που είναι και η πιο διαδεδομένη στα αριθμητικά κυκλώματα.

Επίσης, παρουσιάζονται οι κωδικοποιήσεις Booth και Modified Booth που προσφέρουν αποδοτικότερες υλοποιήσεις της πράξης του πολλαπλασιασμού, ελαττώνοντας την πολυπλοκότητα των κυκλωμάτων πολλαπλασιασμού.

Τέλος, παρουσιάζονται οι δύο διαφορετικοί αλγόριθμοι που θα εξετάσουμε για την υλοποίηση του μιγαδικού πολλαπλασιαστή, ο συμβατικός αλγόριθμος και ο αλγόριθμος του Gauss.

2.1. Αριθμητικά συστήματα

2.1.1. Δυαδικό Σύστημα

Το δυαδικό σύστημα είναι το πιο γνωστό, απλό αλλά και ευρέως χρησιμοποιούμενο αριθμητικό σύστημα στα ψηφιακά κυκλώματα. Είναι η βάση πάνω στην οποία στηρίχθηκε η ανάπτυξη της ψηφιακής λογικής, και σχεδόν όλα τα επόμενα συστήματα αποτελούν παραλλαγές ή επεκτάσεις του.

Το δυαδικό σύστημα, σαν σύστημα αρίθμησης, είναι ακριβώς όμοιο με το γνωστό μας δεκαδικό, με την μόνη διαφορά ότι χρησιμοποιεί μόνο δύο διαφορετικά ψηφία για την αναπαράσταση όλων των αριθμών. Το δυαδικό σύστημα καθιερώθηκε στον κόσμο των ψηφιακών συστημάτων διότι η χρήση δύο διακριτών καταστάσεων (που αντιστοιχούν στα ψηφία 0 και 1) καθιστά πιο εύκολο τον διαχωρισμό τους σε επίπεδα τάσεων, κάτι που θα ήταν πιο πολύπλοκο αν υιοθετούσαμε την χρήση ενός δεκαδικού ή οκταδικού συστήματος.

Η δυαδική αναπαράσταση ακολουθεί το ίδιο σύστημα με βάρη που χρησιμοποιεί και η δεκαδική αναπαράσταση, μόνο που σε αυτή την περίπτωση η βάση είναι το δύο (και όχι το δέκα) και επομένως τα δυνατά ψηφία ανήκουν στο σύνολο {0,1} (αντί του {0,1,2,3,4,5,6,7,8,9}) ενώ τα βάρη είναι και αυτά δυνάμεις του δύο (αντί του δέκα).

Κάθε αριθμός παριστάνεται σε δυαδική μορφή σύμφωνα με την παρακάτω σχέση

$$a = \sum_{i=0}^{n-1} 2^i \cdot b_i$$

όπου,

a: η ισοδύναμη δεκαδική τιμή του αριθμού

b_i: τα ψηφία του δυαδικού αριθμού που ανήκουν στο σύνολο {0,1}

n: το πλήθος των δυαδικών ψηφίων του αριθμού

Για παράδειγμα, ο δεκαδικός αριθμός 178 αντιστοιχίζεται στο δυαδικό σύστημα με τον εξής τρόπο:

$$178_{(10)} = 10110010_{(2)} = 2^7 \cdot 1 + 2^6 \cdot 0 + 2^5 \cdot 1 + 2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 0$$

Μπορούμε εύκολα να παρατηρήσουμε ότι κάθε θέση στο διάνυσμα του αριθμού έχει διαφορετικό βάρος. Το ψηφίο με το χαμηλότερο βάρος είναι το δεξιότερο (Least Significant Bit) ενώ το ψηφίο με το μεγαλύτερο βάρος είναι το αριστερότερο (Most Significant Bit).

Πρόσθεση αριθμών

Το άθροισμα δύο δυαδικών αριθμών υπολογίζεται με την χρήση των ίδιων κανόνων που ισχύουν για τους δεκαδικούς. Κάθε φορά αθροίζονται ψηφία ίδιου βάρους και εάν προκύψει κρατούμενο σε κάποια θέση, τότε αυτό προστίθεται στο ψηφίο της αμέσως πιο σημαντικής θέσης. Στην συνέχεια έχουμε ένα παράδειγμα μιας τέτοιας πρόσθεσης μεταξύ δύο αριθμών:

$$\begin{array}{rcl} 178 & = & 10110010 \\ +156 & = & +10011100 \\ \hline 334 & = & 101001110 \end{array}$$

2.1.2. Αναπαράσταση σε μορφή συμπληρώματος ως προς δύο

Η ανάγκη για την παράσταση προσημασμένων αριθμών στο δυαδικό σύστημα οδήγησε στην μορφή του συμπληρώματος ως προς 2. Στην μορφή συμπληρώματος του 2 το πιο σημαντικό bit του αριθμού (MSB) έχει αρνητική αξία. Η αναπαράσταση αυτή έχει τα καλύτερα αποτελέσματα προς αυτή την κατεύθυνση, δίνει δηλαδή μια σχεδιαστικά εύκολη υλοποίηση τόσο της πρόσθεσης όσο και της αφαίρεσης, για θετικούς και αρνητικούς αριθμούς, σε συνδυασμό με μια απλή διαδικασία για την εύρεση του αντίθετου ενός αριθμού, ενώ είναι ταυτόχρονα κατανοητή με μια απλή ανάγνωση.

Στην αναπαράσταση αυτή, το πιο σημαντικό ψηφίο του αριθμού (MSB) λαμβάνει το αντίστοιχο αρνητικό βάρος, μετατοπίζοντας έτσι το εύρος τιμών που μπορούν να αναπαρασταθούν με N bits από τις τιμές 0 ως 2^N-1 στις τιμές -2^{N-1} ως 2^{N-1}-1.

Η τιμή ενός δυαδικού αριθμού που αποτελείται από n bits και είναι σε μορφή συμπληρώματος ως προς 2 δίνεται από την παρακάτω σχέση:

$$a_{(10)} = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Όπως ακριβώς και στο συμβατικό δυαδικό σύστημα, έτσι και στην μορφή του συμπληρώματος του δύο η αναπαράσταση ενός αριθμού είναι αμφιμονοσήμαντη.

Η διαφορά των δύο αναπαραστάσεων φαίνεται με το επόμενο παράδειγμα, ο δυαδικός αριθμός 10110010 ενώ στο συμβατικό δυαδικό σύστημα αντιστοιχεί στην δεκαδική τιμή 178, στην αναπαράσταση σε μορφή συμπληρώματος ως προς δύο αντιστοιχεί σε αρνητικό δεκαδικό αριθμό ίσο με την τιμή

$$10110010_{(2)} = -1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -78_{(10)}$$

Προφανώς, στην μορφή συμπληρώματος του δύο, όλοι οι θετικοί αριθμοί έχουν το πιό σημαντικό τους ψηφίο (MSB) ίσο με το μηδέν, και τα υπόλοιπα bits είναι τα ίδια με αυτά που θα είχαν στη συμβατική δυαδική αναπαράσταση.

2.1.2.1 Συμπλήρωμα (ως προς δύο)

Συμπλήρωμα (ως προς δύο) ενός αριθμού που έχει αναπαρασταθεί στο δυαδικό σύστημα ως προσημασμένος αριθμός, είναι ένας προσημασμένος δυαδικός αριθμός που έχει το ίδιο μέτρο αλλά αντίθετο πρόσημο.

Συγκεκριμένα, για να βρούμε το συμπλήρωμα (ως προς δύο) ενός αριθμού ακολουθούμε τα εξής βήματα:

Α) Αντιστρέφουμε όλα τα ψηφία του αριθμού μετατρέποντας τα 0 σε 1 και τα 1 σε 0. Πλέον ο αριθμός βρίσκεται στην μορφή συμπληρώματος ως προς 1.

Β) Έπειτα προσθέτουμε μία μονάδα στον αριθμό που προέκυψε.

Για παράδειγμα θα βρούμε το συμπλήρωμα ως προς δύο του αριθμού 10110010= $-78_{(10)}$. Αντιστρέφοντας τα ψηφία του παίρνουμε τον αριθμό:

01001101

και προσθέτουμε μία μονάδα σε αυτόν:

$$\begin{array}{r} 01001101 \\ + \quad\quad 1 \\ \hline 01001110 \end{array}$$

Εύκολα μπορούμε να καταλάβουμε πως το συμπλήρωμα ως προς 2 του $-78_{(10)}$ είναι το $78_{(10)}$ και έτσι επαληθεύεται ο ορισμός.

2.1.2.2 Πρόσθεση και αφαίρεση δύο αριθμών σε μορφή συμπληρώματος ως προς 2

Η πρόσθεση και η αφαίρεση δύο αριθμών σε μορφή συμπληρώματος ως προς δύο γίνεται με παρόμοιο τρόπο με την πρόσθεση και την αφαίρεση αριθμών στο συμβατικό δυαδικό σύστημα, με δύο όμως βασικές τροποποιήσεις:

Α) Όλα τα bits των δύο αριθμών προστίθενται όπως τα bits των συμβατικών δυαδικών αριθμών, εκτός από τα MSB που έχουν αρνητικό βάρος. Για την αναπαράσταση του αποτελέσματος της πρόσθεσης στη βαθμίδα των MSB χρειαζόμαστε δύο ψηφία, τα οποία να καλύπτουν το εύρος τιμών -2 ... +1. Ο λόγος που το αποτέλεσμα έχει αυτό το εύρος τιμών, είναι το ότι στη βαθμίδα αυτή, προστίθενται δύο ψηφία αρνητικού βάρους (τα MSB των αριθμών) και το κρατούμενο εισόδου από την προηγούμενη βαθμίδα, το οποίο είναι θετικού βάρους. Επομένως όλα τα πιθανά αποτελέσματα αναπαρίστανται σωστά με ένα bit θετικού βάρους στη βαθμίδα των MSB και ένα επιπλέον bit αρνητικού βάρους στην αμέσως μεγαλύτερη βαθμίδα. Το τελικό αποτέλεσμα προκύπτει από τη συνένωση των δύο αυτών bit με τα υπόλοιπα bits των χαμηλότερων βαθμίδων του αποτελέσματος που υπολογίστηκαν με το συμβατικό τρόπο.

Β) Για να προστεθούν δύο αριθμοί και το αποτέλεσμα να είναι πάντα σωστό, θα πρέπει να έχουν τον ίδιο αριθμό ψηφίων. Για να συμβεί αυτό, αυξάνουμε τον αριθμό με τα λιγότερα ψηφία κατά όσα ψηφία χρειάζεται, έτσι ώστε οι δ ύο αριθμοί να έχουν τον ίδιο αριθμό ψηφίων. Η επαύξηση αυτή πραγματοποιείται τοποθετώντας τα επιπλέον bits στα αριστερά του MSB του αριθμού και δίνοντας σε όλα την τιμή του MSB. Με αυτό τον τρόπο δεν μεταβάλλεται η δεκαδική τιμή του αριθμού. Η τεχνική αυτή ονομάζεται επέκταση προσήμου (sign extension).

Παρακάτω δίνεται ένα παράδειγμα για την πρόσθεση δυαδικών αριθμών σε μορφή συμπληρώματος του δύο:

Έστω ότι θέλουμε να προσθέσουμε τους $00101001_{(2)} = 41_{(10)}$ και $1000110_{(2)} = -58_{(10)}$.

Βλέπουμε πως ενώ ο πρώτος αριθμός έχει 8 ψηφία, ο δεύτερος έχει 7. Άρα όπως αναφέραμε παραπάνω, θα πραγματοποιήσουμε επέκταση προσήμου στον δεύτερο αριθμό. Έτσι ο δεύτερος αριθμός θα γίνει:

$$1000110_{(2)} = (1)\ 1000110_{(2)}$$

που έχει και αυτός 8 ψηφία.

Τώρα που έχουμε δύο αριθμούς με το ίδιο μήκος λέξης προχωράμε στην πρόσθεση όλων των bits εκτός το MSB:

$$\begin{array}{r} (0)0101001 \\ +(1)1000110 \\ \hline \underline{(0)}1101111 \end{array}$$

Για τον υπολογισμό του MSB πρέπει να λάβουμε υπόψιν τα αρνητικής αξίας MSB των δύο όρων της πρόσθεσης καθώς και το υπογραμμισμέο MSB που είναι το κρατούμενο εξόδου και έχει θετική αξία. Η πρόσθεση θα γίνει ως εξής:

$$(-1)\cdot MSB_1 + (-1)\cdot MSB_2 + C_{in} = (-1)\cdot 0 + (-1)\cdot 1 + 0 = -1$$

Το αποτέλεσμα παριστάνεται ως 11 (και αυτό γιατί είναι ίσο με $(-1)\cdot 2^1 + 1\cdot 2^0 = -1$) και το τελικό αποτέλεσμα προκύπτει από την συνένωση αυτού του αριθμού με το υπόλοιπο αποτέλεσμα που προέκυψε παραπάνω, όπου παραλείψαμε τα MSB και το κρατούμενο της προηγούμενης βαθμίδας.

Άρα έχουμε:

$$\{11, 1101111\} = 111101111_{(2)} = -17_{(10)} = 41_{(10)} + (-58_{(10)})$$

Για να υλοποιήσουμε μια αφαίρεση δύο προσημασμένων αριθμών συνδυάζουμε ότι κάναμε στην πρόσθεση καθώς και στον υπολογισμό του συμπληρώματος ως προς δύο, όπως φαίνεται στο παράδειγμα παρακάτω:

Έστω ότι θέλουμε να αφαιρέσουμε τον $0101101_{(2)} = 45_{(10)}$ από τον $01001110_{(2)} = 78_{(10)}$. Ακολουθούμε την ίδια μέθοδο με πριν και εφαρμόζουμε επέκταση προσίμου στον πρώτο αριθμό ώστε να έχει 8 ψηφία όπως ο δεύτερος.

Έτσι πρώτος αριθμός γίνεται: $000101101_{(2)} = 45_{(10)}$. Στην συνέχεια υπολογίζουμε το συμπλήρωμά του ως προς δύο. Αρχικά υπολογίζουμε το συμπλήρωμά του ως προς ένα, το οποίο είναι 11010010 και προσθέτουμε μία μονάδα. Εύκολα μπορούμε να υπολογίσουμε το άθροισμα που είναι ο αριθμός:

$$11010011_{(2)} = -45_{(10)}$$

Για να καταλήξουμε στον τελικό υπολογισμό προσθέτουμε τους δύο προσημασμένους δυαδικούς αριθμούς όπως κάναμε παραπάνω:

(0)1001110

+(1)1010011

(1) 0100001

Τέλος υπολογίζουμε το MSB όπως πριν:

$$(-1)\cdot MSB_1 + (-1)\cdot MSB_2 + C_{in} = (-1)\cdot 0 + (-1)\cdot 1 + 1 = 0$$

Άρα ο αριθμός που προκύπτει είναι:

$$00100001_{(2)} = 33_{(10)} = 78_{(10)} - 45_{(10)}$$

2.2. Κωδικοποιήσεις Booth και Modified Booth

2.2.1. Κωδικοποίηση Booth

Ο αλγόριθμος φέρει το όνομα του Andrew Donald Booth και επινοήθηκε το 1950. Παρακάτω εξηγείται πως λειτουργεί η κωδικοποίηση.

Έστω δυαδικός αριθμός X σε μορφή συμπληρώματος ως προς δύο. Όπως είδαμε, ο X δίνεται από την παρακάτω σχέση:

$$X_{(10)} = -x_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

όπου x_i τα ψηφία του δυαδικού αριθμού X.

Ο αριθμός X μπορεί να γραφεί ισοδύναμα: X = 2X − X

2X = -x_{n-1}	x_{n-2}	x_{n-3}	…	x_0	0
-X = 0	-x_{n-1}	-x_{n-2}	…	x_1	x_0
	z_{n-1}	z_{n-2}	…	z_1	z_0

όπου:

$z_0 = 0 - x_0$

$z_1 = x_0 - x_1$

$z_2 = x_1 - x_2$

.

.

$z_{n-2} = x_{n-3} - x_{n-2}$

Το z_{n-1} υπολογίζεται όπως βλέπουμε παρακάτω:

$$z_{n-1} = -2 \cdot x_{n-1} + x_{n-2} + x_{n-1} = x_{n-2} - x_{n-1}$$

Επομένως καταλήγουμε στην γενική σχέση για τον αριθμό Χ:

$$X = \sum_{i=0}^{n-1} z_i \cdot 2^i$$

όπου z_i είναι τα ψηφία που προέκυψαν από την κωδικοποίηση Booth του Χ.

Το σημαντικό πλεονέκτημα της κωδικοποίησης Booth είναι ότι εφαρμόζεται σε οποιονδήποτε αριθμό σε μορφή συμπλήρωμα ως προς δύο, ανεξαρτήτως προσήμου.

Πολλαπλασιασμός με την μέθοδο Booth

Έστω το γινόμενο $P = X \cdot Y$, με $X = x_{n-1}x_{n-2}...x_1x_0$ και $Y = y_{n-1}y_{n-2}...y_1y_0$.

Με χρήση της κωδικοποίησης Booth για τον παράγοντα Υ, ο πολλαπλασιασμός των δύο αριθμών γίνεται:

$$P = X \cdot Y = \sum_{i=0}^{n-1} (y_{i-1} - y_i) \cdot X \cdot 2^i$$

Από την παραπάνω σχέση βλέπουμε ότι σε κάθε βήμα i, το Χ πολλαπλασιάζεται με ένα από τα στοιχεία του συνόλου {-1, 0, 1}, ανάλογα με το αποτελέσμα της αφαίρεσης των δύο διαδοχικών ψηφίων του πολλαπλασιαστή Υ. Στην ουσία με την κωδικοποίηση Booth ελέγχουμε σειρές από μονάδες που παρουσιάζονται μέσα σε κάθε δεκαδικό αριθμός. Στον Πίνακα 2.1 παρουσιάζονται όλες οι δυνατές περπτώσεις κατά αντιστοιχία με τα κωδικοποιησμένα ψηφία που προκύπτουν, καθώς και οι λογικές λειτουργίες που αντιπροσωπεύουν.

Πίνακας 2.1 Αντιστοίχηση δυαδικών ψηφίων με λειτουργίες και κωδικοποίηση Booth

Y		Κωδικοποιημένα ψηφία (y_{i-1}	Αποτέλεσμα ελέγχου
0	0	0	Δεν υπάρχει σειρά από 1
0	1	1	Τέλος σειρά από 1
1	0	-1	Αρχή σειράς από 1
1	1	0	Μέση σειράς από 1

Η κωδικοποιήση Booth είναι ένα σύστημα αναπαράστασης προσημασμένου ψηφίου. Συνοψίζοντας βλέπουμε ότι η κωδικοποίηση Booth παρουσιάζει τα παρακάτω πλεονεκτήματα:

- Η κωδικοποίηση τόσο θετικών όσο και αρνητικών αριθμών συμπληρώματος ως προς δύο, γίνεται με τον ίδιο τρόπο.
- Κάθε ψηφίο του προς κωδικοποίηση αριθμού, παράγεται ανεξάρτητα από τα υπόλοιπα, όντας συνάρτηση μόνο των δύο συνεχόμενων bit Y_iY_{i-1}. Αυτό μας εξασφαλίζει άμεση παραγωγή των επιμέρους μερικών γινομένων σε έναν πολλαπλασιασμό, το οποίο βοηθά στη γρήγορη εκτέλεσή του.
- Το σημαντικότερο πλεονέκτημα της κωδικοποίησης Booth είναι ότι μπορεί να μειώσει τον αριθμό των μη μηδενικών μερικών γινομένων και έτσι να μειωθούν οι προσθέσεις που πρέπει να γίνουν εάν ο αριθμός που κωδικοποιήθηκε είχε μεγάλο αριθμό μονάδων. Ωστόσο, αν ο αριθμός είχε απομονωμένες μονάδες, τότε είναι πιθανόν η κωδικοποίηση Booth, όχι μόνον να μην δώσει τα αναμενόμενα αποτελέσματα, αλλά και να αυξήσει τον αριθμό των μη μηδενικών μερικών γινομένων με αποτέλεσμα να επιβαρύνει το κύκλωμα υλοποίησης της πράξης στην οποία λαμβάνει μέρος ο αριθμός.

Για την αντιμετώπιση του προβλήματος αυτού, προτάθηκε μια διαφοροποιημένη μέθοδος κωδικοποίησης, η οποία παρουσιάζεται στην συνέχεια.

2.2.2. Κωδικοποίηση Modified Booth

Μια επέκταση του απλού αλγόριθμου Booth είναι ο τροποποιημένος αλγόριθμος Booth που επεκτείνει το σύνολο των ψηφίων κωδικοποίησης από το σύνολο {-1, 0, +1} στο σύνολο {-2, -1, 0, +1, +2} και προκύπτει αλγεβρικά από τον απλό αλγόριθμο του Booth. Η κωδικοποίηση Modified Booth, όπως και η απλή Booth, ανήκει στα συστήματα με αναπαράσταση προσημασμένου ψηφίου (signed digit).

Στην κωδικοποίηση Booth είδαμε πως ισχύει η σχέση:

$$Y = \sum_{i=0}^{n-1} z_i \cdot 2^i$$

όπου z_i είναι τα ψηφία που προέκυψαν από την απλή κωδικοποίηση Booth του Y.

Με περαιτέρω ανάλυση της σχέση αυτής έχουμε:

$$Y = \sum_{i=0}^{n-1} z_i \cdot 2^i = \sum_{j=0}^{\frac{n}{2}-1} (z_{2j} \cdot 2^{2j} + z_{2j+1} \cdot 2^{2j+1}) = \sum_{j=0}^{\frac{n}{2}-1} (z_{2j} + z_{2j+1} \cdot 2) \cdot 2^{2j}$$

$$= \sum_{j=0}^{\frac{n}{2}-1} (w_j) \cdot 4^j$$

όπου w_i είναι το κωδικοποιημένο ψηφίο κατά Modified Booth, το οποίο μπορεί να αναλυθεί ως εξής:

$$w_i = y_{2i} - 2 \cdot y_{2i+1} + y_{2i-1}$$

Όπως εύκολα μπορούμε να διαπιστώσουμε από την παραπάνω σχέση, ο τροποποιημένος αλγόριθμος Booth χρησιμοποιεί τριάδες ψηφίων του δυαδικού αριθμού για την κωδικοποίηση. Οι τριάδες αυτές έχουν μια επικάλυψη, κατά ένα ψηφίο, και επειδή για i=0 χρειάζεται μια τιμή y_{-1} για την σωστή κωδικοποίηση, πρέπει να θεωρήσουμε ότι ισούται με 0 για να έχουμε σωστά αποτελέσματα.

Όλες οι δυνατές περιπτώσεις, μαζί με τα κωδικοποιημένα ψηφία που προκύπτουν, καθώς και η αντιστοιχία με την κωδικοποίηση Booth, φαίνονται αναλυτικά στον Πίνακα 2.2.

Πίνακας 2.2 Αντιστοίχηση δυαδικών ψηφίων με απλή κωδικοποίηση Booth & Modified Booth

Y_{2i+1}			Κωδικοποίηση Booth Z_{2i+1}		Κωδικοποίηση Modified Booth W_i
0	0	0	0	0	0
0	0	1	0	+1	+1
0	1	0	+1	-1	+1
0	1	1	+1	0	+2
1	0	0	-1	0	-2
1	0	1	-1	+1	-1
1	1	0	0	-1	-1
1	1	1	0	0	0

Πολλαπλασιασμός με την μέθοδο Modified Booth

Η κωδικοποίηση Modified Booth χρησιμοποιείται ευρέως στην υλοποίηση πολλαπλασιαστών. Κάθε κωδικοποιημένο ψηφίο καθορίζει τη λειτουργία που πρόκειται να γίνει στον πολλαπλασιασμό, όπως στον απλό Booth. Έστω ότι θέλουμε να πολλαπλασιάσουμε δύο αριθμούς των 8 bits A και B, όπου B είναι ο πολλαπλασιαστής του οποίου τα ψηφία θα κωδικοποιηθούν κατά Modified Booth. Τα Modified Booth ψηφία που θα προκύψουν, ανάλογα με την τιμή τους, θα καθορίσουν τις λειτουργίες που θα εκτελεστούν. Οι λειτουργίες αυτές φαίνονται στον Πίνακα 2.3.

Πίνακας 2.3 Αντιστοιχία λειτουργιών και κωδικοποιημένων ψηφίων

Κωδικοποιημένο ψηφίο	**Λειτουργία**
0	Πρόσθεσε το 0 στο μερικό γινόμενο
+1	Πρόσθεσε το (A) στο μερικό γινόμενο
+2	Πρόσθεσε το (2A) στο μερικό γινόμενο
-2	Αφαίρεσε το (2A) στο μερικό γινόμενο
-1	Αφαίρεσε το (A) στο μερικό γινόμενο

Για καλύτερη κατανόηση της διαδικασίας πολλαπλασιασμού δύο αριθμών με χρήση της κωδικοποίηση Modified Booth, παρατίθεται το παράδειγμα:

Έστω $A = 1001110_{(2)} = -50_{(10)}$ και $B = 01110011_{(2)} = 115_{(10)}$.

Ο αριθμός B κωδικοποιημένος σύμφωνα με τον αλγόριθμο Modified Booth γίνεται:

$B = 2\bar{1}1\bar{1}$. Άρα θα μας χρειαστούν οι αριθμοί 2A, A και -A. Τους υπολογίζουμε:

2A = 10011100, -A = 00110010.

Ο πολλαπλασιασμός του A με τον B φαίνεται στον Πίνακα 2.4.

Πίνακας 2.4 Παράδειγμα πολλαπλασιασμού με χρήση κωδικοποίησης Modified Booth

A = 1001110 B = 01110011 Μερικό γινόμενο = 00000000	
0 0 0 1 1 0 0 1 0	-1 Πρόσθεσε -A (Πρώτο μερικό γινόμενο)
1 0 0 1 1 1 0	+1 Πρόσθεσε A (Δεύτερο μερικό γινόμενο)
1 1 1 1 0 1 1 0 1 0 1 0	Άθροισμα των δύο πρώτων μερικών γινομένων
0 0 1 1 0 0 1 0	-1 Πρόσθεσε -A (Τρίτο μερικό γινόμενο)
1 1 1 0 0 1 0 1 0 0 0 1 0 1 0	Άθροισμα των τριών πρώτων μερικών γινομένων
1 1 0 0 1 1 1 0 0	+2 Πρόσθεσε 2A (Τέταρτο μερικό γινόμενο)
1 1 1 0 1 0 0 1 1 0 0 0 1 0 1 0	Τελικό αποτέλεσμα = -5750

Όπου τα υπογραμμισμένα bits αποτελούν την επέκταση προσήμου για να βγει σωστό το αποτέλεσμα.

Ο τροποποιημένος αλγόριθμος του Booth έχει σαν πλεονέκτημα το ότι εφαρμόζεται ανεξάρτητα από το αν οι αριθμοί είναι σε απλή δυαδική αναπαράσταση ή σε μορφή συμπληρώματος ως προς δυο, μαζί με τα υπόλοιπα πλεονεκτήματα της απλής κωδικοποίησης Booth.

Το μεγαλύτερο όμως πλεονέκτημα που επιτυγχάνεται με τον τροποποιημένο αλγόριθμο του Booth, είναι το ότι έχουμε μείωση του αριθμού των μερικών γινομένων στο μισό, σε σχέση με τις προηγούμενες κωδικοποιήσεις, και επομένως λιγότερες προσθέσεις να εκτελέσουμε, με προφανές όφελος το κέρδος σε ταχύτητα λειτουργίας του πολλαπλασιαστή αλλά και την μείωση της επιφάνειας που καταλαμβάνει.

2.3. Είδη αθροιστών και πολλαπλασιαστών

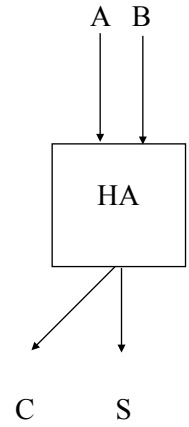
2.3.1 Δομικά στοιχεία

Τα περισσότερα κυκλώματα που επιτελούν τις πράξεις της πρόσθεσης ή του πολλαπλασιασμού αποτελούνται από κάποια βασικά στοιχεία τα οποία χρησιμοποιούνται σαν μονάδες πρόσθεσης των επιμέρους όρων (μερικών γινομένων).

Οι βασικότερες από αυτές τις μονάδες είναι ο ημιαθροιστής και ο πλήρης αθροιστής, που παρουσιάζονται στη συνέχεια. Επίσης, παρουσιάζεται και η γεννήτρια πρόβλεψης κρατουμένου, που αποτελεί βασικό στοιχείο των αθροιστών πρόβλεψης κρατουμένου.

2.3.1.1 Ημιαθροιστής

Ο ημιαθροιστής (Half Adder) είναι ένα στοιχειώδες κύκλωμα που επιτελεί την πράξη της πρόσθεσης δύο bits. Ως εκ τούτου, έχει δύο εισόδους, οι οποίες είναι ψηφία του ίδιου βάρους και επιτελεί την πρόσθεση αυτών των δύο ψηφίων, παράγοντας δύο εξόδους, η μία του ίδιου βάρους με τις εισόδους (Save, S) και μία άλλη στο αμέσως μεγαλύτερο (Carry, C). Το σχήμα του ημιαθροιστή φαίνεται αμέσως παρακάτω:



Σχήμα 2.1 Ημιαθροιστής

Οι εξισώσεις που συνδέουν τις εισόδους με τις εξόδους του ημιαθροιστή είναι οι εξής:

$$S = A \oplus B$$
$$C = A \cdot B$$

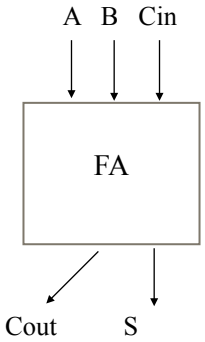
Οι παραπάνω εξισώσεις προκύπτουν από τον παρακάτω πίνακα αληθείας:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Πίνακας 2.5 Πίνακας αληθείας ημιαθροιστή

2.3.1.2 Πλήρης αθροιστής

Ο πλήρης αθροιστής (Full Adder) είναι ένα στοιχειώδες κύκλωμα που επιτελεί την πράξη της πρόσθεσης τριών bits, γι' αυτό αναφέρεται στην βιβλιογραφία και ως 3-2 συμπιεστής. Ως εκ τούτου, έχει τρείς εισόδους, οι οποίες είναι ψηφία του ίδιου βάρους και επιτελεί την πρόσθεση αυτών των τριών ψηφίων, παράγοντας δύο εξόδους, η μία του ίδιου βάρους με τις εισόδους (Save, S) και μία άλλη στο αμέσως μεγαλύτερο (Carry, C). Το σχήμα του πλήρους αθροιστή φαίνεται αμέσως παρακάτω:



Σχήμα 2.2 Πλήρης αθροιστής

Οι σχέσεις που συνδέουν τις εισόδους και τις εξόδους του πλήρους αθροιστή είναι οι εξής:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \cdot B) + (A \cdot C_{in}) + (B \cdot C_{in})$$

Οι παραπάνω εξισώσεις προκύπτουν από τον παρακάτω πίνακα αληθείας:

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

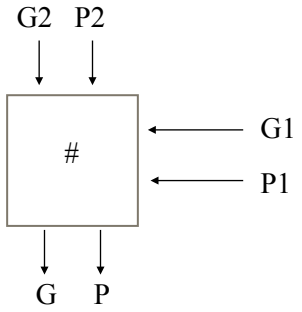
Πίνακας 2.6 Πίνακας αληθείας πλήρους αθροιστή

2.3.1.3 Κύκλωμα πρόβλεψης κρατουμένου

Το κύκλωμα πρόβλεψης κρατουμένου αποτελεί βασικό δομικό στοιχείο της γεννήτριας κρατουμένου, η οποία είναι η «καρδιά» ενός αθροιστή πρόβλεψης κρατουμένου.

Δέχεται ως εισόδους τα σήματα G1, P1, G2, P2 τα οποία είναι τα σήματα γέννησης (G, Generation) και διάδοσης (P, Propagation) κρατουμένου δύο βαθμίδων διαφορετικής αξίας, ενώ δίνει σαν έξοδο τα σήματα G, P τα οποία αποτελούν δύο νέα σήματα γεννήσεως και διάδοσης κρατουμένου, στην μεγαλύτερη από τις δύο βαθμίδες.

Το σχήμα του τελεστή # (κύκλωμα πρόβλεψης κρατουμένου) φαίνεται αμέσως παρακάτω:



Σχήμα 2.3 Κύκλωμα πρόβλεψης κρατουμένου

Οι σχέσεις που συνδέουν τις εισόδους και τις εξόδους του κυκλώματος πρόβλεψης κρατουμένου είναι οι εξής:

$$G = G2 + (G1 \cdot P2)$$

$$P = P1 \cdot P2$$

2.3.2 Αθροιστές

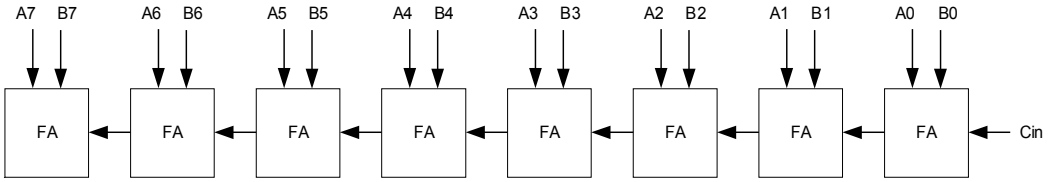
2.3.2.1 Αθροιστής διάδοσης κρατουμένου (Carry Propagate Adder)

Ο αθροιστής διάδοσης κρατουμένου (Carry Propagate Adder) πραγματοποιεί την πράξη της πρόσθεσης με βάση τον αλγόριθμο που ξέρουμε για την πρόσθεση δεκαδικών αριθμών, δηλαδή προσθέτει δύο ψηφία ίσου βάρους και παράγει ένα ψηφίο ίσου βάρους, το οποίο είναι το άθροισμά τους, καθώς και ένα κρατούμενο με το αμέσως μεγαλύτερο βάρος.

Κυκλωματικά, αποτελείται από ένα δίκτυο πλήρων αθροιστών συνδεδεμένων σε σειρά. Κάθε πλήρης αθροιστής αντιστοιχεί σε μία βαθμίδα συγκεκριμένου βάρους. Το κρατούμενο εξόδου (Cout) του κάθε πλήρους αθροιστή συνδέεται με το κρατούμενο εισόδου (Cin) του πλήρους αθροιστή της επόμενης βαθμίδας, και το αποτέλεσμα της πράξης αποτελείται από τα Save ψηφία όλων των πλήρων αθροιστών, καθώς και το κρατούμενο εξόδου της πιο σημαντικής βαθμίδας.

Με αυτόν τον τρόπο τα κρατούμενα διαδίδονται μέσα στην δομή του αθροιστή από την χαμηλότερη μέχρι την υψηλότερη βαθμίδα, και αυτός είναι ο λόγος που ο αθροιστής πήρε αυτό το όνομα.

Παρακάτω φαίνεται ένα κύκλωμα αθροιστή διάδοσης κρατουμένου 8-bit, το οποίο μπορεί να δεχτεί σαν είσοδο είτε δύο δυαδικούς αριθμούς είτε έναν carry save αριθμό. Η έξοδός του είναι ένας δυαδικός αριθμός που αποτελείται από τα ψηφία {Cout,S7,S6,S5,S4,S3,S2,S1,S0}.



Σχήμα 2.4 Κύκλωμα αθροιστή διάδοσης κρατουμένου

Το παραπάνω κύκλωμα μπορεί να λειτουργήσει και για δυαδικούς αριθμούς σε μορφή συμπληρώματος του δύο, αν προστεθεί έλεγχος για τις περιπτώσεις υπερχείλισης, ή αν απλά αντιστραφούν τα ψηφία αρνητικής αξίας {A7, B7, Cout}.

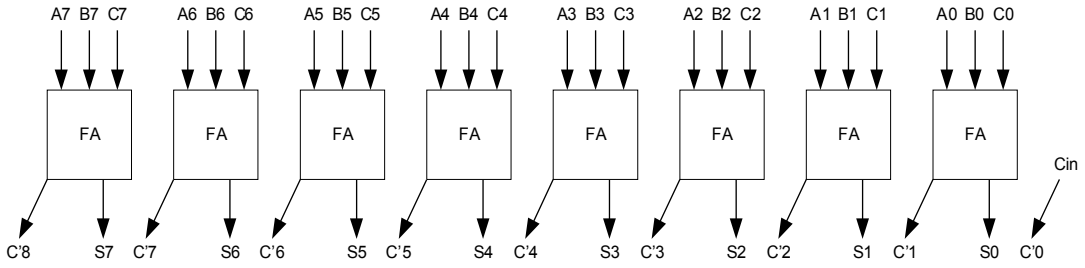
Αν και ο αθροιστής διάδοσης κρατουμένου έχει πολύ απλή δομή και πολύ μικρή επιφάνεια, η διάδοση του κρατουμένου τον καθιστά απαγορευτικά αργό για τις περισσότερες εφαρμογές καθώς η καθυστέρηση (critical path) ενός N-bit αθροιστή είναι N επίπεδα πλήρων αθροιστών.

2.3.2.2 Αθροιστής σωσίματος κρατουμένου (Carry Save Adder)

Ο αθροιστής σωσίματος κρατουμένου (Carry Save Adder) λειτουργεί παρόμοια με τον αθροιστή διάδοσης κρατουμένου, όμως δεν παράγει έναν δυαδικό αριθμό ως αποτέλεσμα, αντί αυτού διατηρεί το αποτέλεσμα σε μορφή Carry-Save όπως προκύπτει από τις επιμέρους αθροίσεις κάθε βαθμίδας.

Η δομή του αθροιστή σωσίματος κρατουμένου είναι όμοια με τον αθροιστή διάδοσης κρατουμένου, με τη διαφορά ότι το κρατούμενο εξόδου (Cout) κάθε πλήρους αθροιστή δεν είναι είσοδος σε κάποιον άλλο, αλλά αποτελεί αποτέλεσμα στην συγκεκριμένη βαθμίδα, μαζί με το αντίστοιχο Save ψηφίο.

Παρακάτω φαίνεται ένα κύκλωμα αθροιστή σωσίματος κρατουμένου 8-bit, το οποίο μπορεί να δεχτεί σαν είσοδο είτε τρείς δυαδικούς αριθμούς είτε ένα δυαδικό και έναν Carry-Save αριθμό. Η έξοδός του είναι ένας Carry-Save αριθμός που αποτελείται από τα ψηφία
C = {C΄8, C΄7, C΄6, C΄5, C΄4, C΄3, C΄2, C΄1, C΄0} και S = {S7, S6, S5, S4, S3, S2, S1, S0}.



Σχήμα 2.5 Κύκλωμα αθροιστή σωσίματος κρατουμένου

Το παραπάνω κύκλωμα μπορεί να λειτουργήσει και για δυαδικούς αριθμούς σε μορφή συμπληρώματος του δύο, αν αντιστραφούν τα ψηφία αρνητικής αξίας {A7, B7, C7, S7, C΄8}.

Η παραπάνω δομή εξηγεί και την ονομασία του πλήρους αθροιστή ως 3-2 συμπιεστή.

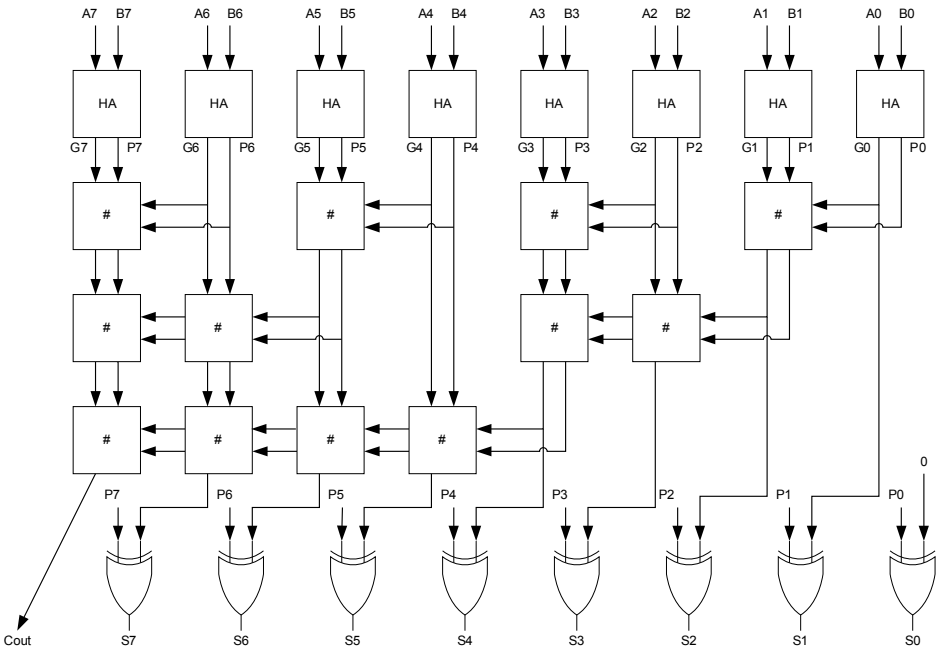
Ο αθροιστής σωσίματος κρατουμένου έχει απλή δομή και την ίδια επιφάνεια με τον αθροιστή διάδοσης κρατουμένου. Επίσης μπορεί να χειριστεί τρείς αντί για δύο δυαδικούς αριθμούς ταυτόχρονα. Το μεγάλο πλεονέκτημά του όμως είναι ότι η καθυστέρηση (critical path) ενός N-bit αθροιστή είναι μόνο 1 επίπεδο πλήρους αθροιστή. Το βασικό μειονέκτημα του είναι ότι το αποτέλεσμα δεν βρίσκεται σε δυαδική μορφή, οπότε θα χρειαστεί περαιτέρω επεξεργασία.

2.3.2.3 Αθροιστής πρόβλεψης κρατουμένου (Carry Look ahead Adder)

Ο αθροιστής πρόβλεψης κρατουμένου (Carry Look ahead Adder) ακολουθεί μια διαφορετική φιλοσοφία άθροισης ώστε να αποφύγει την καθυστέρηση από την διάδοση κρατουμένου. Χρησιμοποιεί κυκλώματα ημιαθροιστών για την δημιουργία των σημάτων P και G, κυκλώματα πρόβλεψης κρατουμένου (#) για την πρόβλεψη του τελικού κρατουμένου και ένα τελευταίο στάδιο που είναι η γεννήτρια του τελικού αθροίσματος, η οποία στην απλοποιημένη περίπτωση που δεν έχουμε κρατούμενο εισόδου, αποτελείται μόνο από N πύλες αποκλειστικού-ή (XOR), όπου N το πλήθος των ψηφίων του αθροιστή.

Το κύκλωμα δημιουργίας κρατουμένου, εκμεταλλεύεται την προσεταιριστικότητα του τελεστή # για να υπολογίσει παράλληλα τα τελικά κρατούμενα. Αυτό επιτυγχάνεται με την διάταξη των κυκλωμάτων πρόβλεψης κρατουμένου σε μια δενδρική δομή, και επομένως την παραγωγή των τελικών αποτελεσμάτων με καθυστέρηση logN κυκλωμάτων #.

Παρακάτω φαίνεται ένα κύκλωμα αθροιστή πρόβλεψης κρατουμένου 8-bit χωρίς κρατούμενο εισόδου, το οποίο μπορεί να δεχτεί σαν είσοδο είτε δυο δυαδικούς αριθμούς είτε έναν Carry-Save αριθμό. Η έξοδός του είναι ένας δυαδικός αριθμός που αποτελείται από τα ψηφία {Cout,S7,S6,S5,S4,S3,S2,S1,S0}.



Σχήμα 2.6 Κύκλωμα αθροιστή πρόβλεψης κρατουμένου

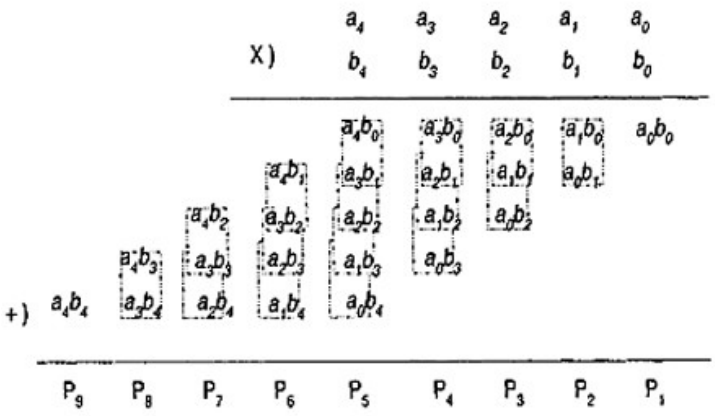
Το παραπάνω κύκλωμα μπορεί να λειτουργήσει και για δυαδικούς αριθμούς σε μορφή συμπληρώματος του δύο, αν αντιστραφούν τα ψηφία αρνητικής αξίας {A7, B7, Cout}.

Ο αθροιστής πρόβλεψης κρατουμένου έχει πιο πολύπλοκη δομή, αλλά είναι πιο γρήγορος από τον αθροιστή διάδοσης κρατουμένου, καθώς αποφεύγει την διάδοση κρατουμένου και παράγει τα αποτελέσματά του με καθυστέρηση ενός HA συν logN κυκλωμάτων # συν μιας πύλης XOR. Το μειονέκτημα του αθροιστή πρόβλεψης κρατουμένου είναι η επιφάνειά του, η οποία αυξάνει πολύ σε μεγάλα μήκη λέξης. Συγκεκριμένα, ο αριθμός των κυκλωμάτων πρόβλεψης κρατουμένου (#) που απαιτούνται είναι N*logN/2, όπου N το πλήθος των ψηφίων του αθροιστή. Σε γενικές γραμμές ωστόσο, το κέρδος σε ταχύτητα είναι πιο σημαντικό από την επιβάρυνση σε επιφάνεια.

2.3.3 Πολλαπλασιαστές

Η πράξη του πολλαπλασιασμού στο δυαδικό σύστημα εκτελείται ακριβώς όμοια με το δεκαδικό. Επειδή όμως στο δυαδικό σύστημα έχουμε μόνο δυο ψηφία {0,1}, ο πολλαπλασιασμός ενός N-bit αριθμού μπορεί να έχει μόνο δύο δυνατά αποτελέσματα, είτε τον αριθμό αυτούσιο, είτε ένα μηδενικό μερικό γινόμενο. Ως εκ τούτου, η παραγωγή των μερικών γινομένων γίνεται με πύλες και (AND).

Ο πολλαπλασιασμός δύο αριθμών A και B των πέντε bit φαίνεται στο παρακάτω σχήμα:

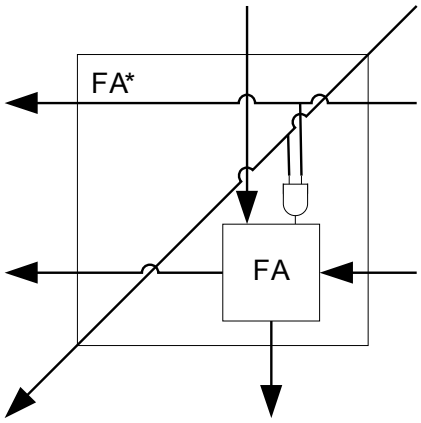


Σχήμα 2.7 Πράξεις πολλαπλασιασμού

2.3.3.1 Παράλληλοι πολλαπλασιαστές τύπου πίνακα (Array Multipliers)

Η παραπάνω διαδικασία υποδεικνύει και μια βασική τοπολογία κυκλώματος πολλαπλασιασμού. Συγκεκριμένα, παράγονται παράλληλα N μερικά γινόμενα, αποτελούμενα από N-bits το καθένα, και αθροίζονται τα ψηφία ίσου βάρους, κάνοντας χρήση ενός δικτύου N2 πλήρων αθροιστών σε μορφή πίνακα.

Θα παρουσιάσουμε δύο βασικές τοπολογίες αυτής της οικογενείας, τον παράλληλο πολλαπλασιαστή με διάδοση κρατουμένου και τον παράλληλο πολλαπλασιαστή με σώσιμο κρατουμένου. Και οι δύο τοπολογίες κάνουν χρήση της ίδιας βασικής δομικής μονάδας που αποτελείται, όπως προαναφέραμε, από μια πύλη AND, για την παραγωγή ενός ψηφίου ενός μερικού γινομένου, και έναν πλήρη αθροιστή. Η δομική αυτή μονάδα, παρουσιάζεται αμέσως παρακάτω.

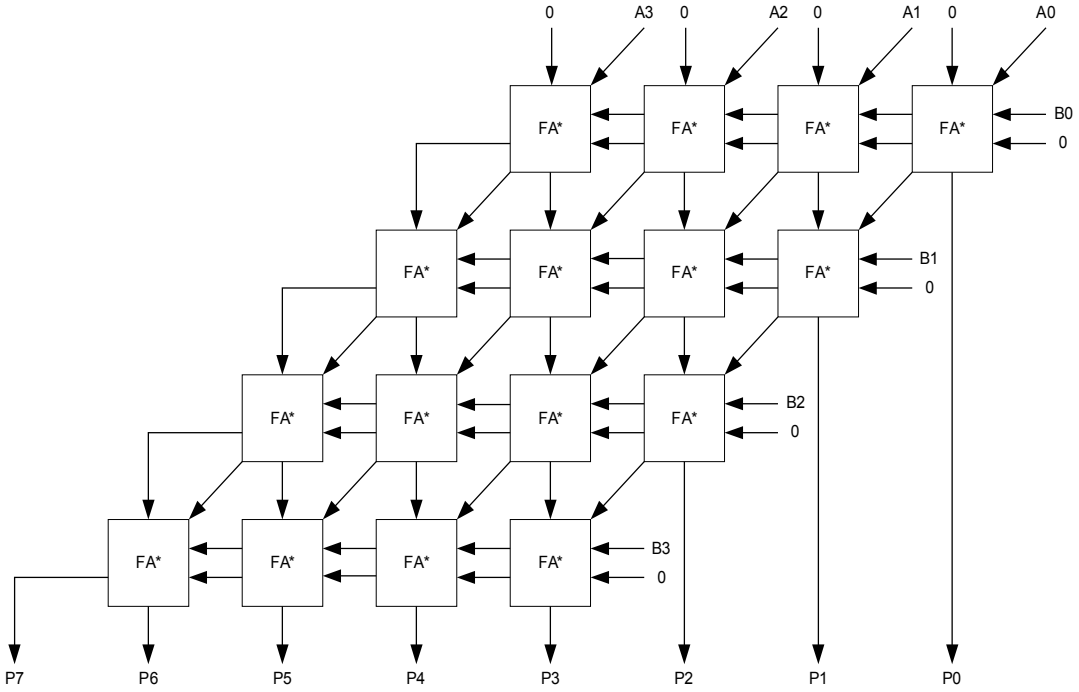


Σχήμα 2.8 Κύκλωμα FA*

Βασικό πλεονέκτημα και των δύο αυτών σχημάτων είναι πως η δομή των πολλαπλασιαστών παραμένει κανονική και επίσης είναι αρκετά εύκολη η μετατροπή τους σε συνεχούς διοχέτευσης και συστολικούς.

Α. Παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου

Ο παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου, αποτελείται από αθροιστές διάδοσης κρατουμένου συνδεδεμένους σε σειρά (οι έξοδοι του ενός δηλαδή είναι είσοδοι του επομένου) αλλά ο κάθε ένας από αυτούς είναι ολισθημένος κατά μία θέση αριστερά, ως προς τον προηγούμενο. Αμέσως παρακάτω παρουσιάζεται ένας παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου, τεσσάρων bit.



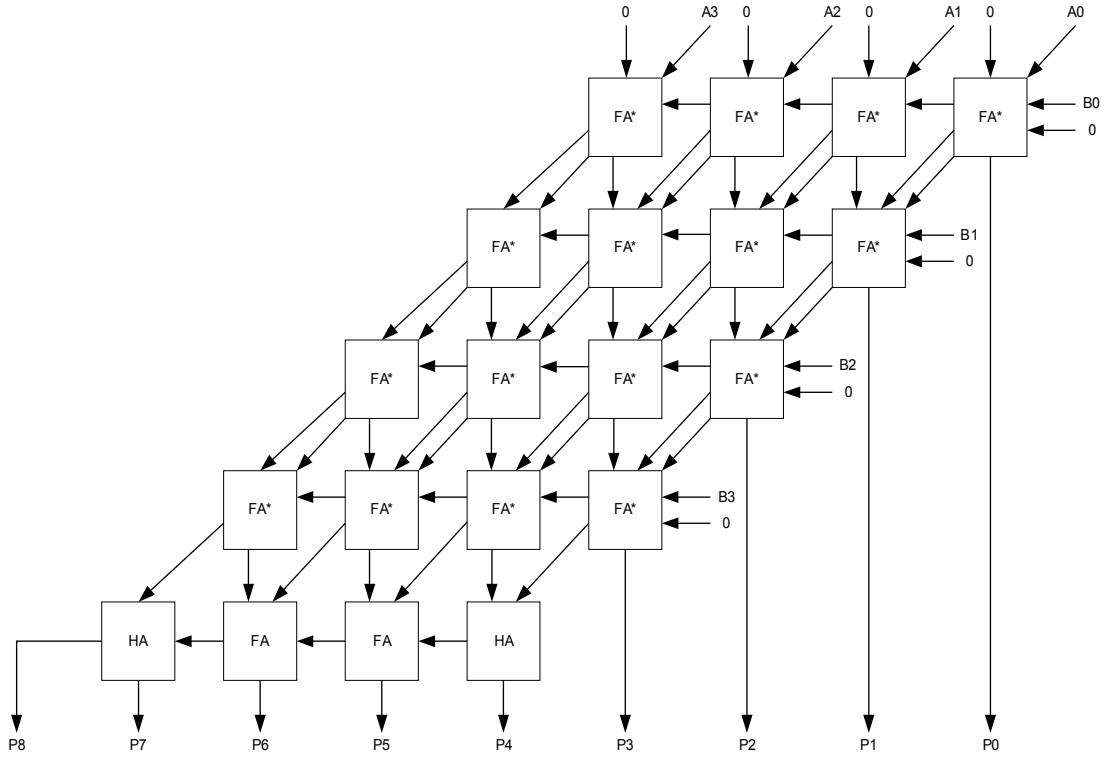
Σχήμα 2.9 Κύκλωμα παράλληλου πολλαπλασιαστή με διάδοση κρατουμένου

Ο παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου τεσσάρων bit παράγει τα αποτελέσματά του με καθυστέρηση 10 FA*, όπως φαίνεται και από το παραπάνω σχήμα. Στην γενική περίπτωση, η επιφάνεια του πολλαπλασιαστή με διάδοση κρατουμένου είναι N2 FA* και η καθυστέρησή του είναι 3N-2 FA*, όπου N το μήκος λέξης των όρων A και B.

Β. Παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου

Ο παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου, αποτελείται από αθροιστές σωσίματος κρατουμένου συνδεδεμένους σε σειρά (οι έξοδοι του ενός δηλαδή είναι είσοδοι του επομένου) αλλά ο κάθε ένας από αυτούς είναι ολισθημένος κατά μία θέση αριστερά, ως

προς τον προηγούμενο. Επειδή στο τελευταίο στάδιο της άθροισης το πιο σημαντικό τμήμα του αποτελέσματος είναι σε μορφή Carry-Save, πρέπει να ενσωματώσουμε έναν επιπλέον αθροιστή διάδοσης κρατουμένου για να έχουμε το σωστό δυαδικό αποτέλεσμα. Αμέσως παρακάτω παρουσιάζεται ένας παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου, τεσσάρων bit.



Σχήμα 2.10 Κύκλωμα παράλληλου πολλαπλασιαστή με σώσιμο κρατουμένου

Ο παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου τεσσάρων bit παράγει τα αποτελέσματά του με καθυστέρηση 4 FA* συν 2 FA συν 2 HA, όπως φαίνεται και από το παραπάνω σχήμα. Στην γενική περίπτωση, η επιφάνεια του πολλαπλασιαστή με σώσιμο κρατουμένου είναι N^2 FA* συν 2 HA συν N-2 FA και η καθυστέρησή του είναι N FA* συν 2 HA συν N-2 FA, όπου N το μήκος λέξης των όρων A και B.

Ο παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου επομένως, καταλαμβάνει ελαφρώς περισσότερη επιφάνεια από τον παράλληλο πολλαπλασιαστή με διάδοση κρατουμένου, όμως είναι σημαντικά ταχύτερος.

2.3.3.2 Σειριακός Παράλληλος Πολλαπλασιαστής (Serial Parallel Multiplier)

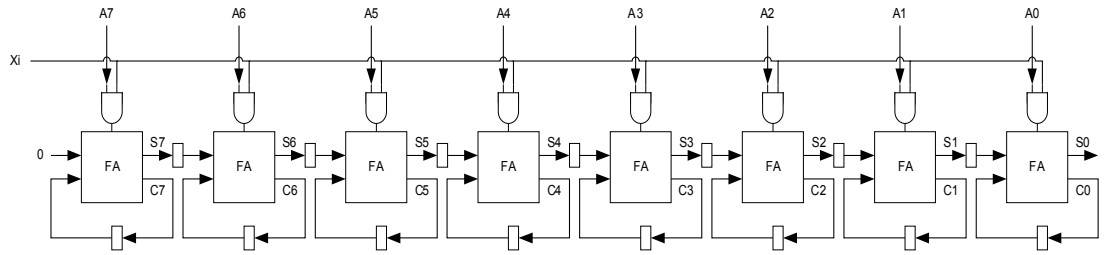
Οι σειριακοί πολλαπλασιαστές είναι κυκλώματα συνεχούς διοχέτευσης ή συστολικά, που προκύπτουν από το χρονικό «δίπλωμα» των αντίστοιχων παράλληλων πολλαπλασιαστών. Οι παράλληλοι πολλαπλασιαστές τύπου πίνακα μετατρέπονται εύκολα σε σειριακούς, λόγω της επαναληπτικής δομής τους, και λόγω του ότι αποτελούνται από όμοια στοιχεία.

Στην παράγραφο αυτή θα παρουσιάσουμε τον σειριακό-παράλληλο πολλαπλασιαστή, ο οποίος έχει όμοια λειτουργία με ένα συστολικό κύκλωμα που προκύπτει από τον παράλληλο πολλαπλασιαστή με σώσιμο κρατουμένου.

Ο σειριακός παράλληλος πολλαπλασιαστής είναι ένα συστολικό κύκλωμα, δηλαδή υποστηρίζει την λογική της συνεχούς διοχέτευσης, ενώ ταυτόχρονα αποτελείται από πανομοιότυπα στοιχεία.

Η λογική που ακολουθεί, είναι παρόμοια με αυτή του παράλληλου πολλαπλασιαστή σωσίματος κρατουμένου, μόνο που χρησιμοποιεί ένα μόνο επίπεδο αθροιστών, οι οποίοι σε συνδυασμό με ένα σύνολο καταχωρητών επιτελούν τις λειτουργίες της πρόσθεσης και της ολίσθησης, ώστε το τελικό αποτέλεσμα να είναι το ίδιο και στις δύο περιπτώσεις.

Για το πολλαπλασιασμό δύο N-bit αριθμών A και X ακολουθείται η εξής διαδικασία, τα ψηφία του A εισέρχονται παράλληλα στο κύκλωμα άθροισης, ενώ τα ψηφία του X εισάγονται σειριακά, ένα σε κάθε κύκλο ρολογιού, ξεκινώντας από το λιγότερο σημαντικό και καταλήγοντας στο περισσότερο σημαντικό. Παράγεται ένα bit του αποτελέσματος σε κάθε κύκλο ρολογιού, ξεκινώντας από το λιγότερο σημαντικό. Αμέσως παρακάτω παρουσιάζεται ένας σειριακός παράλληλος πολλαπλασιαστής οκτώ bit.



Σχήμα 2.11 Κύκλωμα σειριακού παράλληλου πολλαπλασιαστή 8 bit

Ο σειριακός παράλληλος πολλαπλασιαστής παράγει τα αποτελέσματά του με συνολική καθυστέρηση N πύλες AND συν N FA συν N καταχωρητές, ενώ έχει συνολική επιφάνεια ίση με N FA* συν N καταχωρητές, όπου N το μήκος λέξης του όρου A. Είναι δηλαδή αρκετά πιο αργός από τον παράλληλο πολλαπλασιαστή σωσίματος κρατουμένου, ωστόσο καταλαμβάνει σημαντικά μικρότερη επιφάνεια. Επίσης, έχει την δυνατότητα να υποστηρίξει διάφορα μήκη λέξης για τον παράγοντα X.

2.3.3.3 Δενδρικοί πολλαπλασιαστές (Tree Multipliers)

Οι δενδρικοί πολλαπλασιαστές χρησιμοποιούν μια διαφορετική φιλοσοφία άθροισης, στοχεύοντας στην παραγωγή κυκλωμάτων πολλαπλασιασμού με την ελάχιστη καθυστέρηση. Η δημιουργία των μερικών γινομένων στους δενδρικούς πολλαπλασιαστές γίνεται ανεξάρτητα από το στάδιο της πρόσθεσης. Αφού παραχθούν όλα τα μερικά γινόμενα, οδηγούνται σε ένα δίκτυο FA και HA, από το οποίο παράγεται ένας αριθμός σε μορφή σωσίματος - κρατουμένου. Στο τελικό στάδιο, μπορεί να χρησιμοποιηθεί ένας αθροιστής διάδοσης κρατουμένου ή πρόβλεψης κρατουμένου για να παραχθεί το τελικό αποτέλεσμα σε δυαδική μορφή.

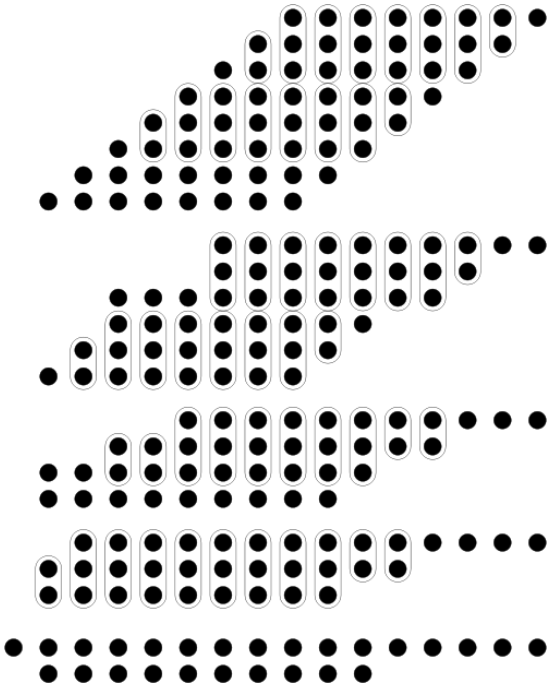
Θα παρουσιάσουμε δύο τύπους δενδρικών πολλαπλασιαστών, που χρησιμοποιούν διαφορετικό αλγόριθμο για την δημιουργία του δέντρου συμπίεσης, τους πολλαπλασιαστές Wallace και τους πολλαπλασιαστές Dadda.

Επειδή η δημιουργία των μερικών γινομένων γίνεται ανεξάρτητα από το στάδιο της δενδρικής συμπίεσης, είναι δυνατή η κωδικοποίηση του ενός από τους δύο αριθμούς ή και των δύο με βάση κάποιο άλλο σύστημα. Με αυτόν τον τρόπο μπορεί να μειωθεί ο αριθμός των μερικών γινομένων έτσι ώστε να αυξηθεί η ταχύτητα του δενδρικού πολλαπλασιαστή.

Α. Δενδρικός πολλαπλασιαστής Wallace

Στόχος του δενδρικού συμπιεστή Wallace είναι να γίνει η συμπίεση των μερικών γινομένων μετά από όσο το δυνατόν λιγότερα επίπεδα FA και HA και συνεπώς σε όσο το δυνατόν μικρότερο χρονικό διάστημα. Για το σκοπό αυτό, σε κάθε επίπεδο, τα ψηφία ίδιου βάρους ομαδοποιούνται ανά τρία και εισέρχονται σαν είσοδοι σε έναν FA, εάν περισσέψουν δύο, τότε ομαδοποιούνται ανά δύο και εισέρχονται σαν είσοδοι σε έναν HA, ενώ εαν περισσέψει μόνο ένα, μεταφέρεται στο επόμενο επίπεδο.

Αμέσως παρακάτω παρουσιάζεται το τμήμα συμπίεσης των μερικών γινομένων ενός 8x8 bit πολλαπλασιαστή Wallace. Κάθε κουκκίδα συμβολίζει ένα bit του αντίστοιχου μερικού γινομένου. Όπου υπάρχει ομαδοποίηση τριών bit, τα τρία αυτά bit εισέρχονται σαν είσοδοι σε έναν FA και όπου υπάρχει ομαδοποίηση δύο bit, τα δύο αυτά bit εισέρχονται σαν είσοδοι σε έναν HA. Προφανώς, κάθε FA και HA παράγουν ως αποτέλεσμα ένα bit ίδιου βάρους (Save) και ένα bit με το αμέσως μεγαλύτερο βάρος (Carry).

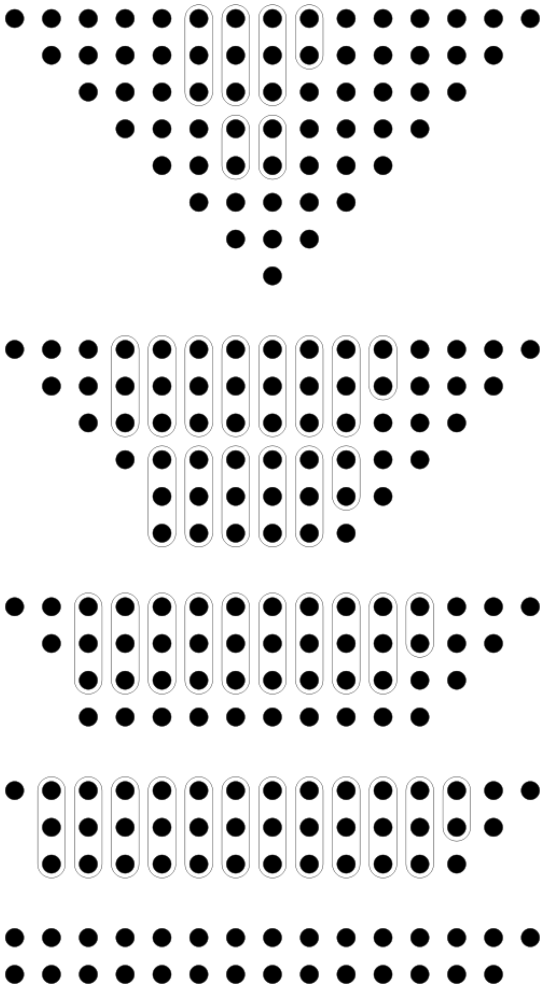


Σχήμα 2.12 Wallace δενδρικός συμπιεστής

Β. Δενδρικός πολλαπλασιαστής Dadda

Ο δενδρικός συμπιεστής Dadda ακολουθεί μία ελαφρώς διαφορετική προσέγγιση στην συμπίεση των bit των μερικών γινομένων. Στόχος του είναι να διατηρήσει σε κάθε βήμα τα bits που έχουν το ίδιο βάρος όσο το δυνατόν πλησιέστερα σε ένα πολλαπλάσιο του τρία, ώστε να χρησιμοποιήσει πιο «επιθετικά» τα κυκλώματα των FA. Ως εκ τούτου, επιτυγχάνει μικρότερη επιφάνεια, καθώς χρησιμοποιεί πιο αποδοτικά την 3:2 συμπίεση που προσφέρει ο FA, έναντι της 2:2 που προσφέρει ο HA. Επιπλέον, νεώτερες μελέτες έδειξαν ότι επιτυγχάνει και μικρότερη καθυστέρηση έναντι του δενδρικού συμπιεστή Wallace.

Αμέσως παρακάτω παρουσιάζεται το τμήμα συμπίεσης των μερικών γινομένων ενός 8x8 bit πολλαπλασιαστή Dadda. Κάθε κουκκίδα συμβολίζει ένα bit του αντίστοιχου μερικού γινομένου. Όπου υπάρχει ομαδοποίηση τριών bit, τα τρία αυτά bit εισέρχονται σαν είσοδοι σε έναν FA και όπου υπάρχει ομαδοποίηση δύο bit, τα δύο αυτά bit εισέρχονται σαν είσοδοι σε έναν HA. Προφανώς, κάθε FA και HA παράγουν ως αποτέλεσμα ένα bit ίδιου βάρους (Save) και ένα bit με το αμέσως μεγαλύτερο βάρος (Carry).



Σχήμα 2.13 Dadda δενδρικός συμπιεστής

Συνοψίζοντας, έχοντας δει τις βασικές μορφές πολλαπλασιαστών, συμπεραίνουμε ότι οι δενδρικοί πολλαπλασιαστές έχουν τα καλύτερα αποτελέσματα στον τομέα της ταχύτητας, οι πολλαπλασιαστές τύπου πίνακα προσφέρουν μια κανονική δομή, η οποία μπορεί να οδηγήσει σε σειριακούς πολλαπλασιαστές, οι οποίοι έχουν σημαντικά μικρότερη επιφάνεια και μπορούν να λειτουργήσουν σε μεγαλύτερες συχνότητες, λόγω του μικρότερου κρίσιμου μονοπατιού (critical path).

2.4. Μιγαδικός Πολλαπλασιαστής

2.4.1 Συμβατικός αλγόριθμος για Μιγαδικό Πολλαπλασιασμό

Η απ'ευθείας εφαρμογή ενός πολλαπλασιασμού δύο μιγαδικών αριθμών χρησιμοποιεί τέσσερις πολλαπλασιαστές:

$$(A + jB) \cdot (C + jD) = R + jI$$

b	b	b	sign	one	two	Booth: W
0	0	0	0	0	0	0
0	0	1	0	1	0	+1
0	1	0	0	1	0	+1
0	1	1	0	0	1	+2
1	0	0	1	0	1	-2
1	0	1	1	1	0	-1
1	1	0	1	1	0	-1
1	1	1	1	0	0	0

όπου R = AC - BD και I = AD + BC

Ο συμβατικός αλγόριθμος για τον μιγαδικό πολλαπλασιασμό χρειάζεται τέσσερις πολλαπλασιασμούς και δύο προσθαφαιρέσεις για την παραγωγή του αποτελέσματος.

2.4.2 Αλγόριθμος του Gauss για Μιγαδικό Πολλαπλασιασμό

Ο αλγόριθμος του Gauss για τον πολλαπλασιασμό δύο μιγαδικών αριθμών μειώνει τον αριθμό των απαραίτητων πολλαπλασιασμών σε τρεις, μέσα από αλγεβρικές πράξεις, με κόστος τριών προσθέσεων.

Έστω ο μιγαδικός πολλαπλασιασμός (A+jB)(C+jD) = R+jI, όπου R = AC-BD και I = AD+BC. Το πραγματικό και το φανταστικό μέρος μπορούν να παραχθούν ως εξής:

$$m_0 = (C - D) \cdot B$$
$$m_1 = (A - B) \cdot C$$
$$m_2 = (A + B) \cdot D$$

$$R = m_1 + m_0$$
$$I = m_2 + m_0$$

3. Σχεδιασμός Κυκλώματος Μιγαδικού Πολλαπλασιαστή

Σε αυτό το κεφάλαιο περιγράφεται αναλυτικά η υλοποίηση των αλγορίθμων που εξετάστηκαν, με έμφαση στα δομικά στοιχεία που χρησιμοποιήθηκαν καθώς και στις παραλλαγές των υλοποιήσεων που εξετάσαμε.

Αρχικά, παρουσιάζονται οι υλοποιήσεις του συμβατικού αλγόριθμου του μιγαδικού πολλαπλασιασμού με χρήση της αριθμητικής σωσίματος κρατουμένου (carry-save).

Επίσης, παρουσιάζονται οι υλοποιήσεις του μιγαδικού πολλαπλασιαστή σύμφωνα με τον αλγόριθμο του Gauss με χρήση δύο διαφορετικών recoders τελεστή πρόσθεσης-πολλαπλασμού.

Τέλος, παρουσιάζονται οι δύο διαφορετικοί αλγόριθμοι που θα εξετάσουμε για την υλοποίηση του μιγαδικού πολλαπλασιαστή, ο συμβατικός αλγόριθμος και ο αλγόριθμος του Gauss.

3.1. Δομικά στοιχεία

Για την υλοποίηση του κυκλώματος που υλοποιεί την πράξη του μιγαδικού πολλαπλασιασμού χρειαζόμαστε διάφορα υποκυκλώματα που θα πραγματοποιούν τις πράξεις των επιμέρους πολλαπλασιασμών, προσθέσεων και αφαιρέσεων. Στην συνέχεια παρουσιάζονται οι μονάδες που σχεδιάσαμε για την υλοποίηση των παραπάνω.

Στο υπόλοιπο του κεφαλαίου χάρην ευκολίας θεωρούμε τον μιγαδικό πολλαπλασιασμό

$$(A+jB)(C+jD) = R+jI$$

όπου R = AC-BD και I = AD+BC.

3.1.1 Μονάδα πολλαπλασιασμού Modified Booth

Η μονάδα αυτή πραγματοποιεί τον πολλαπλασιασμό ανάμεσα σε δύο αριθμούς σε μορφή συμπληρώματος του 2, με χρήση του αλγόριθμου Modified Booth. Η χρήση της κωδικοποίησης Modified Booth έχει θετικά αποτελέσματα τόσο στην ταχύτητα λειτουργίας του πολλαπλασιαστή αλλά και στην μείωση της επιφάνειας που καταλαμβάνει.

Στην συνέχεια του *3.1.1* θεωρούμε ότι έχουμε τον πολλαπλασιασμό δύο αριθμών a, b μήκους λέξης n bits.

3.1.1.1 Κύκλωμα κωδικοποίησης

Το σχήμα κωδικοποίησης που θα χρησιμοποιήσουμε στον πολλαπλασιαστή προτάθηκε το 2003 από τον Huang, ο οποίος μελέτησε διάφορα σχήματα κωδικοποίησης Modified Booth, τα οποία προσομοιώθηκαν με την βιβλιοθήκη TSMC 180nm της Artisan.

Το σχήμα που εμφάνισε τα περισσότερα πλεονεκτήματα, ιδίως ελάχιστη κατανάλωση, αλλά και ελάχιστη καθυστέρηση, όπως επίσης και τον ελάχιστο αριθμό σημάτων κωδικοποίησης, φαίνεται αμέσως παρακάτω.

b	b	b	sign	one	two	Modified Booth: W
0	0	0	0	0	0	0
0	0	1	0	1	0	+1
0	1	0	0	1	0	+1
0	1	1	0	0	1	+2
1	0	0	1	0	1	-2
1	0	1	1	1	0	-1
1	1	0	1	1	0	-1
1	1	1	1	0	0	0

Πίνακα 3.1 Πίνακας αληθείας σημάτων κωδικοποίησης Modified Booth

Ο παραπάνω πίνακας δείχνει τις πράξεις που πρέπει να εκτελεστούν σε κάθε περίπτωση. Ο πολλαπλασιασμός με το 0 ή με το -0 παράγει ένα μηδενικό μερικό γινόμενο, ο πολλαπλασιασμός με το +1 παράγει ένα μερικό γινόμενο ίσο με τον παράλληλο όρο ενώ με το -1 το συμπλήρωμα ως προς δύο του παράλληλου όρου, και τέλος ο πολλαπλασιασμός με το +2 ολισθαίνει κατά μια θέση αριστερά τον παράλληλο όρο ενώ με το -2 παράγει το συμπλήρωμα ως προς δύο του ολισθημένου όρου. Υπενθυμίζουμε ότι για τον υπολογισμό του συμπληρώματος ως προς δύο ενός αριθμού πρέπει να αντιστρέψουμε όλα τα ψηφία του, και κατόπιν να τους προσθέσουμε μια μονάδα. Η επιπλέον αυτή μονάδα, είναι ένα επιπλέον σήμα, απαραίτητο για την μετατροπή σε μορφή συμπληρώματος του δύο, οποίο έχει την τιμή $sign_i$.

Για την κωδικοποίηση σε Modified Booth ενός αριθμού b μήκους λέξης n σε μορφή συμπλήρωμα ως προς 2, η υλοποίηση της παραπάνω κωδικοποίησης προκύπτει από τις παρακάτω εξισώσεις:

$$one_i = b_{2i} \oplus b_{2i-1}$$
$$two_i = (b_{2i+1}' \cdot b_{2i} \cdot b_{2i-1}) + (b_{2i+1} \cdot b_{2i}' \cdot b_{2i-1}')$$
$$sign_i = b_{2i+1}$$

όπου: $one_0 = b_0$, $two_0 = b_1 + b_0'$, $sign_0 = b_1$ και i = [0, 1, …, n].

Με την παραπάνω κωδικοποίηση υπολογίζονται τα γινόμενα AC, AD και BC. Για τον υπολογισμό του γινομένου BD χρησιμοποιούμε ένα ξεχωριστό υποκύκλωμα.

Η έξοδος του πολλαπλασιαστή κατά Modified Booth είναι σε μορφή Carry-Save. Στην συνέχεια προσθέτουμε τα γινόμενα που προκύπτουν από την έξοδο του πολλαπλασιαστή με δύο δένδρα Wallace τα οποία συμπιέζουν τα τέσσερα παραχθέντα γινόμενα σε δύο αριθμούς σε μορφή Carry-Save. Έτσι για να υπολογίσουμε το πραγματικό μέρος του μιγαδικού αριθμού, R = AC-BD, το γινόμενο BD υπολογίζεται με αντίθετο πρόσημο απευθείας. Για να το επιτύχουμε αυτό αντιστρέφουμε τον τρόπο υπολογισμού του σήματος $sign_i$, δηλαδή πλέον είναι $sign_i = b_{2i+1}'$. Με αυτόν τον τρόπο επωφελούμαστε από την χρήση του δένδρου Wallace με μηδενικό επιπλέον κόστος στο design.

3.1.1.2 Κύκλωμα δημιουργίας μερικών γινομένων

Το βέλτιστο κύκλωμα δημιουργίας των μερικών γινομένων με βάση την παραπάνω κωδικοποίηση προκύπτει από τις παρακάτω εξισώσεις.

Το λιγότερο σημαντικό bit του μερικού γινομένου υπολογίζεται ως εξής:

$$pp\_out_0 = ((a_0 \oplus sign) \cdot one) + (sign \cdot two)$$

Η γενική εξίσωση υπολογισμού του μερικού γινομένου είναι:

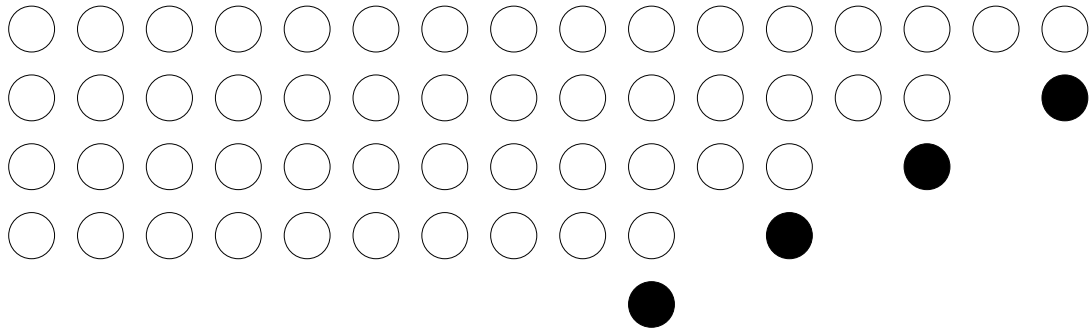
$$pp\_out_i = ((a_i \oplus sign) \cdot one) + ((a_{i-1} \oplus sign) \cdot two)$$

Τέλος το πιο σημαντικό bit του μερικού γινομένου είναι:

$$pp\_out_n = (((a_{n-1} \oplus sign) \cdot one) + ((a_{n-1} \oplus sign) \cdot two))'$$

Για την άθροιση των επιμέρους μερικών γινομένων κάνουμε χρήση ενός δένδρου Wallace. Για να γίνει αυτό πρέπει να συγκεντρώσουμε σε ένα διάνυσμα όλα τα μερικά γινόμενα λαμβάνοντας υπόψιν τα μηδενικά που πρέπει να προσθέσουμε πριν από κάθε μερικό γινόμενο ανάλογα με την βαθμίδα στην οποία βρίσκονται στον πολλαπλασιασμό, μαζί με τα κρατούμενα και τους διορθωτικούς όρους. Στην συνέχεια θα εξετάσουμε έναν πολλαπλασιασμό 8x8 bits για να καταλήξουμε στην γενική περίπτωση ενός πολλαπλασιασμού nxn bits.

Τα μερικά γινόμενα, τα κρατούμενα και οι διορθωτικές μονάδες παρουσιάζονται στο σχήμα 3.1 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 3.1 Μερικά γινόμενα διορθωτικοί όροι και κρατούμενα

Οι λευκές κουκκίδες παριστάνουν τα bits των μερικών γινομένων, οι μαύρες κουκκίδες τους όρους που πρέπει να προστεθούν για την μετατροπή κάθε μερικού γινομένου σε μορφή συμπληρώματος του δύο, και οι οποίοι προφανώς ισούνται με ένα αν το αντίστοιχο μερικό γινόμενο προκύπτει από αντιστροφή του παράλληλου όρου, και οι σκιασμένες κουκκίδες παριστάνουν τα bits της επέκτασης προσήμου. Τα ψηφία της επέκτασης προσήμου πρέπει να καλύψουν μέχρι και τη βαθμίδα βάρους 15, καθώς το αποτέλεσμα ενός 8bit x 8bit πολλαπλασιασμού έχει 16 ψηφία στα βάρη 15 – 0. Τα ψηφία αυτά λαμβάνουν την τιμή του προσήμου του αντίστοιχου μερικού γινομένου.

Όσων αφορά τους διορθωτικούς όρους εργαζόμαστε ως εξής. Για να έχουμε σωστό αποτέλεσμα, έχοντας κάνει την επέκταση προσήμου του MSB κάθε μερικού γινομένου μέχρι την δύναμη 2^{15}, ο διορθωτικός όρος περιλαμβάνει το άθροισμα όλων των sign-extension bits, ct, και αναλύτεται ως εξής:

$$ct = \sum_{k=0}^{3} s_k \sum_{i=8+2k}^{15} 2^i = \sum_{k=0}^{3} s_k (2^{16} - 2^{8+2k}) = \sum_{k=0}^{3} -s_k 2^{8+2k}$$

Για την περαιτέρω ανάλυση της παραπάνω εξίσωσης, χρησιμοποιούμε τις δύο παρακάτω ισότητες:

$$-s_k = \overline{s_k} - 1 \quad \text{και} \quad \sum_{q=j}^{k} 2^q = 2^{k+1} - 2^j$$

Έτσι η εξίσωση που δίνει το ολικό άθροισμα ct, γράφεται:

$$ct = \sum_{k=0}^{3} \overline{s_k} 2^{8+2k} - \{2^{14} + 2^{12} + 2^{10} + 2^8\}$$

Η τελική σχέση των διορθωτικών όρων βρίσκεται από την παρακάτω σχέση:

$$ct = \sum_{k=0}^{3} \overline{s_k} 2^{8+2k} = 2^{15} + 2^{13} + 2^{11} + 2^{9} + 2^{8}$$

Τέλος, μπορούμε να καταλήξουμε στην γενική περίπτωση για τον πολλαπλασιασμό nxn bits:

$$ct = \sum_{k=0}^{n/2} \overline{s_k} \cdot 2^{n+2k} + \{2^{2n-1} + ... + 2^{n+3} + 2^{n+1} + 2^{n}\}$$

3.1.2 Προτεινόμενος Επανακωδικοποιητής Πρόσθεσης-Πολλαπλασιασμού

Πρόσφατες μελέτες στο τομέα της βελτιστοποίησης αριθμητικών κυκλωμάτων έχουν δείξει ότι ο σχεδιασμός των κυκλωμάτων που μοιράζονται δεδομένα, με έναν συνδυαστικό τρόπο μπορεί να οδηγήσει σε σημαντικές βελτιώσεις στην απόδοση του συνολικού κυκλώματος. Βασιζόμενοι σε αυτό το συμπέρασμα χρησιμοποιούμε μια μονάδα που υπολογίζει το άθροισμα δύο αριθμών και στην συνέχεια κωδικοποιεί απευθείας το αποτέλεσμα σε μορφή Modified Booth, πράγμα που μας διευκολύνει πολύ για την υλοποίηση του Μιγαδικού Πολλαπλασιαστή με τον αλγόριθμο του Gauss.

Το σχήμα επανακωδικοποίησης είναι σχεδιασμένο να μπορεί να τροποποιηθεί εύκολα για να χρησιμοποιηθεί είτε για προσημασμένους αριθμούς, σε μορφή συμπληρώματος ως προς 2, είτε για μη προσημασμένους αριθμούς με κριτήριο αν είναι αριθμοί ζυγού ή άρτιου αριθμού bits. Στην παρούσα εργασία εξετάζουμε σχήματα με μήκος λέξης άρτιου αριθμού.

Στην συνέχεια θα εξετάσουμε δύο παραλλαγές αυτής της υλοποιήσης, την υλοποίηση Πρόσθεσης-Πολλαπλασιασμού και την υλοποίηση Αφαίρεσης-Πολλαπλασιασμού.

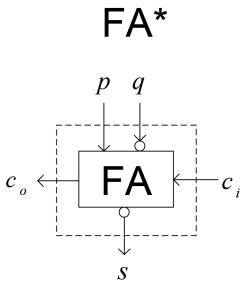
3.1.2.1 Προτεινόμενο Σχήμα επανακωδικοποίησης Πρόσθεσης-Πολλαπλασιασμού

Η πράξη που θα πραγματοποιήσουμε είναι το $Z = X \cdot (A+B)$. Η συμβατική υλοποίηση μιας μονάδας Πρόσθεσης-Πολλαπλασιασμού απαιτεί την οδήγηση δύο εισόδων, Α και Β, σε ένα αθροιστή και στην συνέχεια η οδήγηση της εισόδου Χ και της εξόδου του αθροιστή σε έναν πολλαπλασιαστή για να υπολογιστεί το Z.

Το μειονέκτημα της χρήσης ενός αθροιστή είναι ότι μας εισάγει μια σημαντική καθυστέρηση στο κρίσιμο μονοπάτι. Επειδή υπάρχουν κρατούμενα που πρέπει να διαδωθούν στο εσωτερικό του αθροιστή, το κρίσιμο μονοπάτι εξαρτάται από το μήκος λέξης των εισόδων. Για να μειώσουμε αυτή την καθυστέρηση μπορεί να χρησιμοποιηθεί ένας Αθροιστής Πρόβλεψης Κρατουμένου (Carry Look ahead Adder) όμως αυξάνεται σημαντικά η επιφάνεια που καταλαμβάνει το κύκλωμα καθώς και η ενέργεια που καταναλώνει. Μια βελτιωμένη υλοποίηση αυτής της μονάδας βασίζεται στην συγχώνευση του αθροιστή με την μονάδα κωδικοποίησης σε Modified Booth σε μια κοινή μονάδα, όπου το αποτέλεσμα του αθροίσματος A+B επανακωδικοποιείται απευθείας στην Modified Booth μορφή. Η συγχωνευμένη μονάδα Πρόσθεσης-Πολλαπλασιασμού περιέχει μόνο έναν αθροιστή, στο τελικό στάδιο του παράλληλου πολλαπλασιαστή που χρησιμοποιούμε. Βέβαια, στα σχήματα που έχουμε υλοποιήσει έχουμε αφαιρέσει τον τελικό αθροιστή και χρησιμοποιούμε την έξοδο του πολλαπλασιαστή σε μορφή σωσίματος κρατουμένου, αλλά οι υλοποιήσεις πιο συγκεκριμένα παρουσιάζονται και στην συνέχεια. Σαν αποτέλεσμα έχουμε σημαντική οικονομία τόσο στην επιφάνεια που καταλαμβάνει το κύκλωμά μας καθώς και στην κατανάλωση.

Στην τεχνική επανακωδικοποίησης Αθροίσματος σε Modified Booth, το άθροισμα δύο συνεχόμενων bits της εισόδου $A(a_{2j}, a_{2j+1})$ με τα δύο συνεχόμενα bits της εισόδου $B(b_{2j}, b_{2j+1})$ κωδικοποιούνται σε ένα Modified Booth ψηφίο y_j^{MB}. Όπως γνωρίζουμε για τον σχηματισμό ενός Modified Booth ψηφίου χρειαζόμαστε τρία bits. Το πιο σημαντικό bit από αυτά είναι αρνητικής αξίας και τα δύο λιγότερα σημαντικά bits έχουν θετική αξία. Συνεπώς, για τον σχηματισμό των δύο προαναφερθέντων ζευγαριών bits σε Modified Booth μορφή πρέπει να χρησιμοποιήσουμε αριθμητική προσημασμένων ψηφίων. Για αυτόν τον σκοπό αναπτύχθηκαν προσημασμένοι ημιαθροιστές και πλήρεις αθροιστές, σε επίπεδο bit, ανάλογα με το πρόσημο που θέλουμε για τα bits εισόδου και εξόδου. Στην παρούσα διπλωματική εργασία εργαστήκαμε με προσημασμένους πλήρεις αθροιστές για λόγους απλότητας.

Συγκεκριμένα, χρησιμοποιήσαμε έναν συμβατικό πλήρη αθροιστή και ένα προσημασμένο πλήρη αθροιστή, όπου για όλους ευκολίας θα ονομάσουμε FA*. Στην συνέχεια παραθέτουμε στο σχήμα 3.2 τον αθροιστή FA* με την αξία των ψηφίων του, καθώς και τις εξισώσεις που τον χαρακτηρίζουν.



$$c_o = ((p \vee \overline{q}) \wedge c_i) \vee (p \wedge \overline{q})$$

$$s = p \oplus q \oplus c_i$$

Σχήμα 3.2 Προσημασμένος πλήρης αθροιστής FA*

Στον παρακάτω πίνακα αληθείας παρουσιάζεται η λειτουργία του.

Inputs			Output Value	Outputs	
p(+)	q(-)	c_i		c_o	s(-)
0	0	0	0	0	0
0	0	1	+1	1	1
0	1	0	-1	0	1
0	1	1	0	0	0
1	0	0	+1	1	1
1	0	1	+2	1	0
1	1	0	0	0	0
1	1	1	+1	1	1

$$^{1}OutputValue = 2 \cdot c_o - s = p - q + c_i$$

Πίνακας 3.2 Πίνακας αληθείας FA*

Θεωρούμε ότι τα p,q και c_i είναι οι είσοδοι σε δυαδικό και τα c_o, s είναι το κρατούμενο εξόδου και το άθροισμα αντίστοιχα, ο FA* υλοποιεί την πράξη:

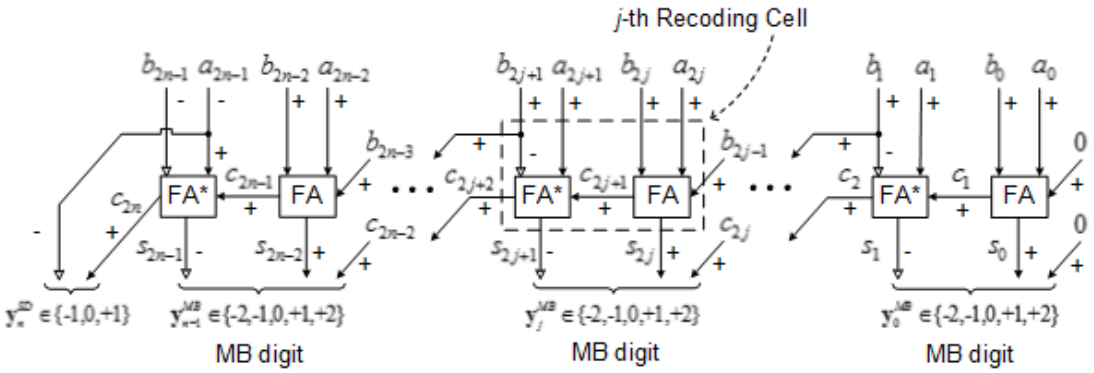
$$2 \cdot c_o - s = p - q + c_i$$

όπου τα bits s και q είναι αρνητικά προσημασμένα.

Έτσι βλέπουμε ότι οι τιμές εξόδου της μονάδας FA* είναι {-1, 0, +1, +2}.

Για την υλοποίηση μας θεωρούμε ότι και οι δύο είσοδοι, A και B, είναι σε μορφή συμπληρώματος ως προς 2 και έχουν μήκος λέξης 2k bits, στην γενική περίπτωση. Σκοπός μας είναι να μετατρέψουμε το άθροισμα των A και B, σε μορφή Modified Booth, έτσι θεωρούμε τα bits a_{2j}, a_{2j+1}, και b_{2j}, b_{2j+1} ως εισόδους του j κελιού επανακωδικοποίησης με σκοπό να πάρουμε σαν έξοδο τα 3 bits που χρειαζόμαστε για να σχηματίσουμε το Modified Booth ψηφίο y_j^{MB}.

Στο σχήμα που υλοποιήσαμε ακολουθούμε την τεχνική κωδικοποίησης που φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.3 Επανακωδικοποιητής Αθροίσματος-Πολλαπλασιασμού

(Sum to MB)

Όπως φαίνεται και στο παραπάνω σχήμα, το άθροισμα των A και B δίνεται από την παρακάτω σχέση:

$$Y = A + B = y_k \cdot 2^{2k} + \sum_{j=0}^{k-1} y_j^{MB} \cdot 2^{2j}$$

$$\text{όπου: } y_j^{MB} = -2s_{2j+1} + s_{2j} + c_{2j}$$

Η κωδικοποίηση των Modified Booth ψηφίων y_j^{MB}, $0 \le j \le k-1$, της παραπάνω σχέσης πραγματοποιήται όπως περιγράψαμε στο 3.1.1.1.

Και τα δύο ψηφία s_{2j+1} και s_{2j} παράγονται από το j κελί επανακωδικοποίησης, όπως φαίνεται στο σχήμα 3.3. Ένας συμβατικός πλήρης αθροιστής με εισόδους a_{2j}, b_{2j} και b_{2j-1} παράγει το κρατούμενο c_{2j+1} και το άθροισμα s_{2j}, όπου:

$$c_{2j+1} = (a_{2j} \wedge b_{2j}) \vee (b_{2j-1} \wedge (a_{2j} \vee b_{2j-1})\ ) \text{ και } s_{2j} = a_{2j} \oplus b_{2j} \oplus b_{2j-1}$$

Όσων αφορά το bit s_{2j+1} χρειάζεται να είναι αρνητικά προσημασμένο, έτσι χρησιμοποιούμε έναν FA* με εισόδους a_{2j+1}, b_{2j+1}(-) και c_{2j+1}, που παράγει το κρατούμενο c_{2j+2} και το άθροισμα s_{2j+1}(-):

$$c_{2j+2} = (a_{2j+1} \wedge \bar{b}_{2j+1}) \vee (c_{2j+1} \wedge (a_{2j+1} \vee \bar{b}_{2j+1}))$$
$$s_{2j+1} = a_{2j+1} \oplus b_{2j+1} \oplus c_{2j+1}$$

Επίσης όπως φαίνεται και στο σχήμα 3.3 στην είσοδο του πρώτου κελιού επανακωδικοποίησης αρχικοποιούμε τα $b_{-1} = 0$ και $c_o = 0$.

Όσων αφορά το πιο σημαντικό ψηφίο του επανακωδικοποιητή y_k^{SD} είναι προσημασμένο ψηφίο που δίνεται από την παρακάτω σχέση:

$$y_k^{SD} = -a_{2k-1} + c_{2k}$$

Η καθυστέρηση του κρίσιμου μονοπατιού του επανακωδικοποιητή είναι σταθερή και ανεξάρτητη του μήκους λέξης της εισόδου και δίνεται από την παρακάτω σχέση:

$$T_{S-MB} = T_{FA,carry} + T_{FA^*,sum}$$

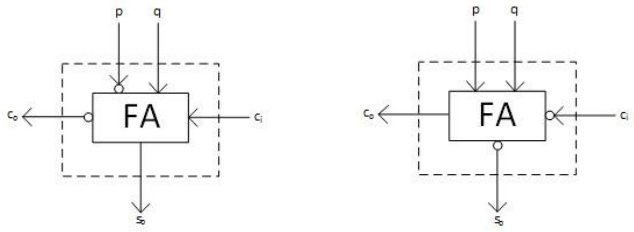
όπου $T_{FA,carry}$ είναι η καθυστέρηση για τον υπολογισμό του κρατουμένου εξόδου ενός συμβατικού πλήρους αθροιστή και $T_{FA^*,sum}$ είναι η καθυστέρηση για τον σχηματισμό του αθροίσματος του προσημασμένου FA*.

3.1.2.2 Προτεινόμενο Σχήμα επανακωδικοποίησης Διαφοράς-Πολλαπλασιασμού

Μια παραλλαγή της παραπάνω μονάδας είναι ο επανακωδικοποιητής Διαφοράς-Πολλαπλασιασμού που κάνει την πράξη Z = X · (A-B). Με κάποιες αλλαγές στις δύο δομικές μονάδες μετατρέπουμε το παραπάνω κύκλωμα επανακωδικοποίησης από Πρόσθεσης-Πολλαπλασιασμού σε Διαφοράς-Πολλαπλασιασμού.

Αυτό το επιτυγχάνουμε με την χρήση των δύο παρακάτω μονάδων, έστω FA' και FA'', που είναι προσημασμένοι πλήρεις αθροιστές που υλοποιούν την αφαίρεση για την απευθείας κωδικοποίηση σε Modified Booth .Στην συνέχεια παραθέτουμε στο σχήμα 3.4(α) τον αθροιστή FA' και στο 3.4(b) τον αθροιστή FA'' με την αξία των ψηφίων τους:

FA' **FA''**



$$c_o = ((\overline{p} \vee q) \wedge c_i) \vee (\overline{p} \wedge q) \qquad c_o = ((p \vee q) \wedge \overline{c_i}) \vee (p \wedge q)$$
$$s = p \oplus q \oplus c_i \qquad s = p \oplus q \oplus c_i$$

(α) (b)

Σχήμα 3.4 Προσημασμένοι πλήρης αθροιστές (α) FA' και (b) FA''

Inputs			Output Value	Outputs	
p(-)	q(+)	c_i		c_o	s(+)
0	0	0	0	0	0
0	0	1	-1	1	1
0	1	0	-1	1	1
0	1	1	-2	1	0
1	0	0	+1	0	1
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	-1	1	1

$$^{2}OutputValue = -2 \cdot c_o + s = -p + q + c_i$$

Πίνακας 3.3 Πίνακας αληθείας FA'

Και στην συνέχεια ο πίνακας του FA'':

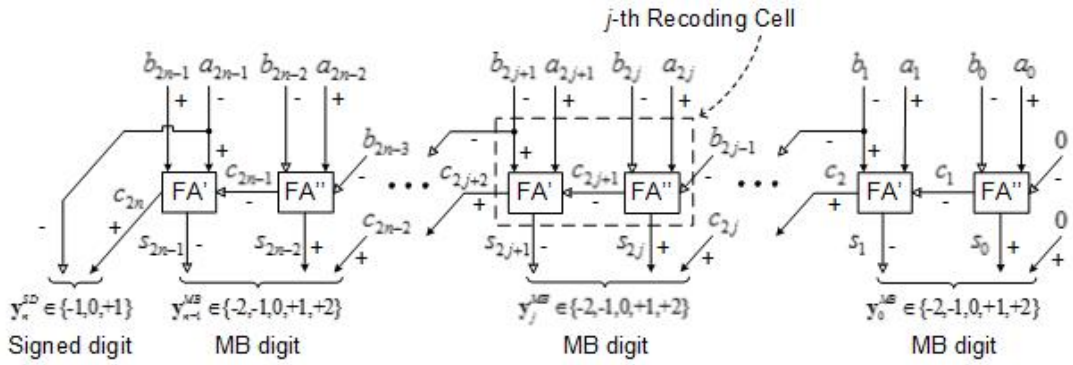
Inputs			Output Value	Outputs	
p(-)	q(+)	c_i		c_o	s(-)
0	0	0	0	0	0
0	0	1	-1	0	1
0	1	0	+1	1	1
0	1	1	0	0	0
1	0	0	+1	1	1
1	0	1	0	0	0
1	1	0	+2	1	0
1	1	1	+1	1	1

FA''

$$^{3}OutputValue = 2 \cdot c_o - s = p + q - c_i$$

Πίνακας 3.4 Πίνακας αληθείας FA''

Στο σχήμα που υλοποιήσαμε ακολουθούμε την τεχνική κωδικοποίησης που φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.5 Επανακωδικοποιητής Διαφοράς-Πολλαπλασιασμού

(Difference to MB)

3.1.3 Επανακωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen

Στο παρόν κεφάλαιο εξετάζουμε έναν Επανακωδικοποιητή σε Modified Booth, αριθμού δυαδικής πλεονασματικής μορφής, όπως παρουσιάστηκε από τους Belle W.Y. Wei, He Du και Hongly Chen και βασίζεται στην υλοποίηση του Μιγαδικού Πολλαπλασιαστή με τον αλγόριθμο του Gauss. Ο Επανακωδικοποιητής αυτός χρησιμοποιεί έναν διαφορετικό τρόπο για τον υπολογισμό του αθροίσματος πριν την κωδικοποίηση του αποτελέσματος σε μορφή Modified Booth.

Ο Επανακωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής προσθέτει, ή αφαιρεί, δύο αριθμούς σε μορφή συμπληρώματος ως προς δύο και παράγει το αποτέλεσμα σε πλεονασματική δυαδική μορφή. Θεωρούμε το άθροισμα (ή διαφορά δύο) αριθμών A και B με μήκος λέξης (n+1) bits όπως φαίνεται παρακάτω:

$$A + B \text{ or } (\mathrm{A} - \mathrm{B}) = d_n \cdot 2^n + \sum_{i=0}^{n-1} d_i \cdot 2^i + g \cdot 2^0$$

όπου

$\mathrm{A} = (a_n a_{n-1} .. a_1 a_0),$
$B = (b_n b_{n-1} .. b_1 b_0),$
$a_n, a_{n-1}, ..., a_1, a_0, b_n, b_{n-1}, ..., b_1, b_0 \in \{0,1\},$
$d_n, d_i, g \in \{\bar{1}, 0, 1\}.$

Θεωρούμε τις δυάδες bits 00, 01, 10, και 11 ότι αναπαρίστανται από τους προσημασμένους αριθμούς, $\bar{1}$, 0, 0 και 1 αντίστοιχα. Η λογική για αυτήν την πλεονασματική δυαδική αναπαράσταση περιγράφεται στο πίνακα 3.5, όπου ο προσημασμένος δυαδικός d_i ($d_i \in \{\bar{1}, 0, 1\}$) έχει δύο ψηφία που τον κωδικοποιούν, τα $d_i^- d_i^+$. Ο Επανακωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής κάνει χρήση μόνο αντιστροφέων σε συνδυασμό με έναν διορθωτικό όρο και έτσι παίρνουμε το άθροισμα σε αυτήν την πλεονασματική δυαδική μορφή για να επανακωδικοποιηθεί μετά σε μορφή Modified Booth.

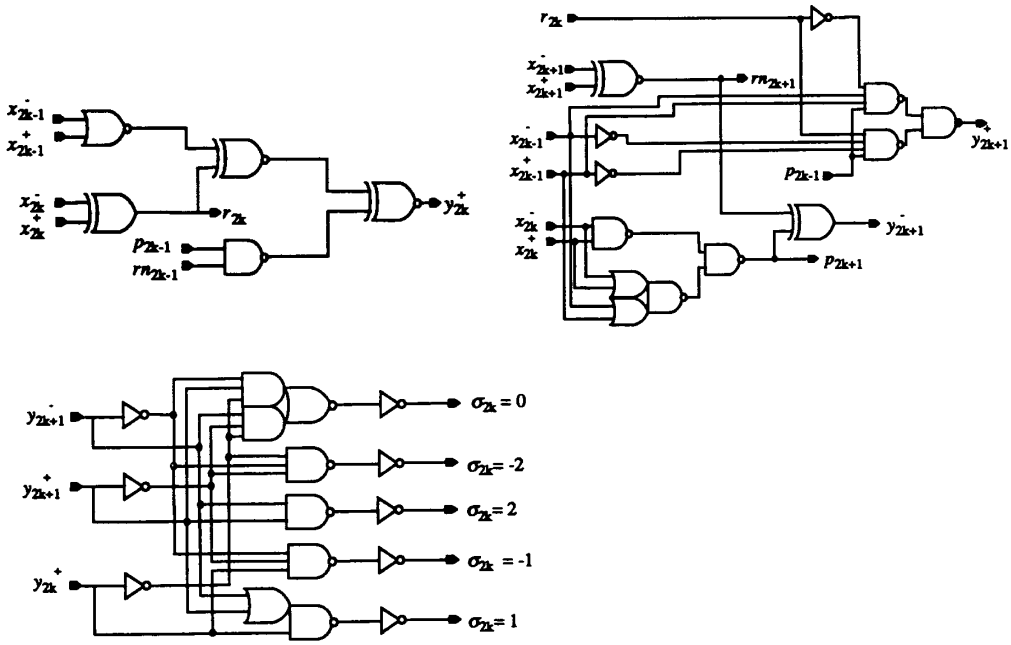
	d_n		$d_i (0 \le i \le n-1)$		$g(g^- g^+)$
	d_n^-	d_n^+	d_i^-	d_i^+	
$A+B$	$\overline{a_n}$	$\overline{b_n}$	a_i	b_i	-1 (00)
$A-B$	$\overline{a_n}$	b_n	a_i	$\overline{b_i}$	0 (01)

Πίνακας 3.5 Ψηφία Πλεονασματικής Δυαδική μορφής για πρόσθεση και αφαίρεση

Ο διορθωτικός όρος g, σύμφωνα με την αριθμητική που ακολουθήθηκε είναι 0 στην αφαίρεση και -1 στην πρόσθεση. Για να εφαρμόσουμε τον διορθωτικό όρο στο αποτέλεσμα της προσθαφαίρεσης τον εισάγαμε σαν κρατούμενο εισόδου στην κωδικοποίηση του αποτελέσματος σε μορφή Modified Booth. Όπως περιγράφηκε και παραπάνω στο κεφάλαιο *2.2.2,* κατά την κωδικοποίηση των τριών πρώτων bits ενός αριθμού στο πρώτο Modified Booth ψηφίο υπάρχει το bit τάξης -1 το οποίο και θεωρούμε ίσο με το 0. Έτσι στην συγκεκριμένη αριθμητική το -1 ζεύγος bits χρησιμοποιείται σαν κρατούμενο εισόδου της μονάδας και αναθέτουμε το -1(00) στην πρόσθεση και το 0(01) στην αφαίρεση για να έχουμε τα σωστά αποτελέσματα.

Στην συνέχεια θα κωδικοποιήσουμε το αποτέλεσμα της πρόσθεσης, ή αφαίρεσής μας από την πλεονασματική δυαδική μορφή που βρίσκεται σε ψηφία Modified Booth. Οι Wei, Du και Chen χρησιμοποιούν μια δική τους κωδικοποίηση για την μετατροπή του αποτελέσματος σε μορφή Modified Booth, με την μόνη διαφορά ότι στην έξοδο ο αριθμός κωδικοποιείται σε Modified Booth μορφή με πέντε σήματα, όσα και οι διαφορετικές τιμές που μπορούμε να πολλαπλασιάσουμε τα μερικά γινόμενά μας {-2, -1, 0, +1, +2}, σε αντίθεση με την προηγούμενη υλοποίηση που έκανε την ίδια κωδικοποίηση με τρία σήματα. Για τον λόγο αυτό υλοποιήσαμε μια ενδιάμεση μονάδα κωδικοποίησης που με σχεδόν μηδενικό κόστος, συμπλέκει τα πέντε σήματα σε τρία, για να χρησιμοποιήσουμε το κύκλωμα του παράλληλου πολλαπλασιαστή όπως περιγράφηκε και παραπάνω.

Στην συνέχεια παραθέτουμε το κύκλωμα που κάνει την κωδικοποίηση του αποτελέσματος της πρόσθεσης, ή αφαίρεσης, σε Modified Booth:



Σχήμα 3.6 Κύκλωμα κωδικοποίησης Wei, Du και Chen

όπου: $x_{2k-1}\ x_{2k}\ x_{2k+1}$ τα τρία προς MB κωδικοποίηση συνεχόμενα ψηφία και

$x_{2k-1} = (x_{2k-1}^{-}\ ,\ x_{2k-1}^{+})$,

$x_{2k} = (x_{2k}^{-}\ ,\ x_{2k}^{+})$,

$x_{2k+1} = (x_{2k+1}^{-}\ ,\ x_{2k+1}^{+})$

Έτσι καταλήγουμε στα 5 σήματα για την παραγωγή των μερικών γινομένων πολλαπλασιασμένα κατά -2, -1, 0, +1 ή +2. Στην κωδικοποίηση που χρησιμοποιούμε τα κωδικοποιημένα ψηφία κατά Modified Booth, αποτελούνται από τρία bits, τα sign, one και two. Στην συνέχεια μετατρέψαμε τα παραπάνω πέντε σήματα των Wei, Du και Chen στα τρία δικά μας, όπως φαίνεται στις παρακάτω εξισώσεις:

$$sign_{2k} = (\sigma_{2k\_-1} \vee \sigma_{2k\_-2})$$
$$one_{2k} = (\sigma_{2k\_-1} \vee \sigma_{2k\_+1})$$
$$two_{2k} = (\sigma_{2k\_-2} \vee \sigma_{2k\_-2})$$

Με αυτό τον τρόπο εξαλείψαμε και το σήμα σ_{2k_0} το οποίο δεν χρειάζεται για τον υπολογισμό των κωδικοποιημένων Modified Booth ψηφίων στη δική μας υλοποίηση.

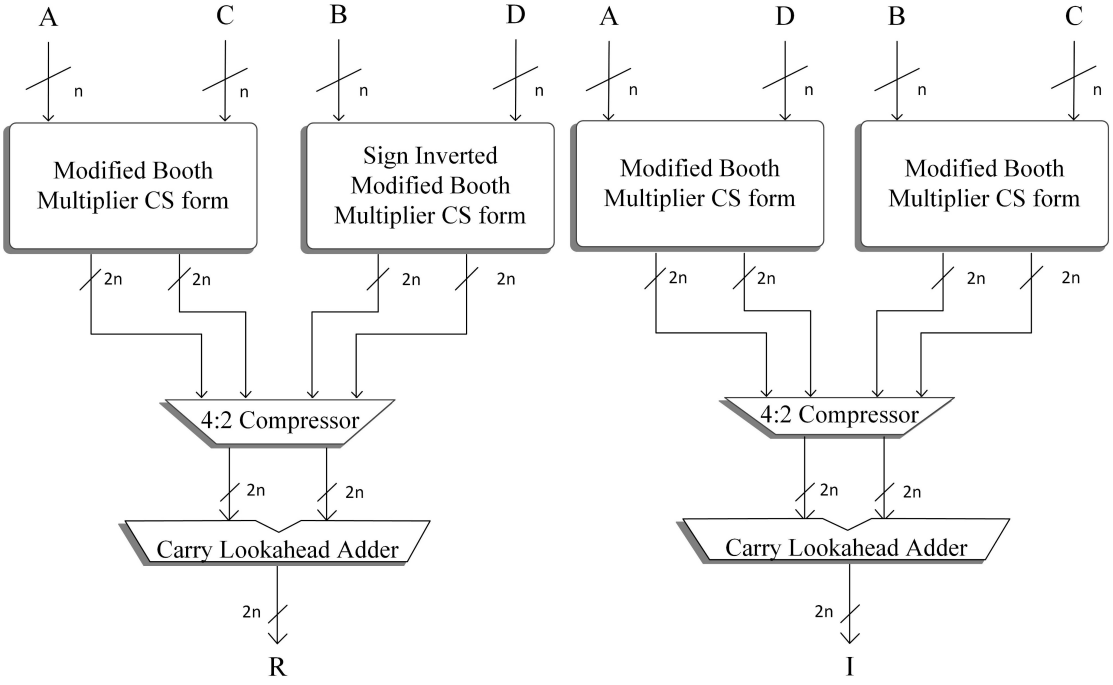
Τέλος για το σημαντικότερο ψηφίο έπρεπε να λάβουμε υπ'όψην το ενδεχόμενο της υπερχείλησης. Στην περίπτωση που έχουμε κρατούμενο εξόδου στο τελευταίο κελί, θα πρέπει να το λάβουμε υπ'όψην, και οι πιθανές τιμές του τελευταίου ψηφίου μπορεί να -1, 0 ή 1. Έτσι για το σημαντικότερο ψηφίο υλοποιούμε μια ξεχωριστή μονάδα από την οποία όμως, απαλείφουμε το υποκύκλωμα για τον υπολογισμό των -2, +2 που είναι τιμές που δεν μπορεί να λάβει αυτό το ψηφίο.

3.2. Περιγραφή υλοποιήσεων

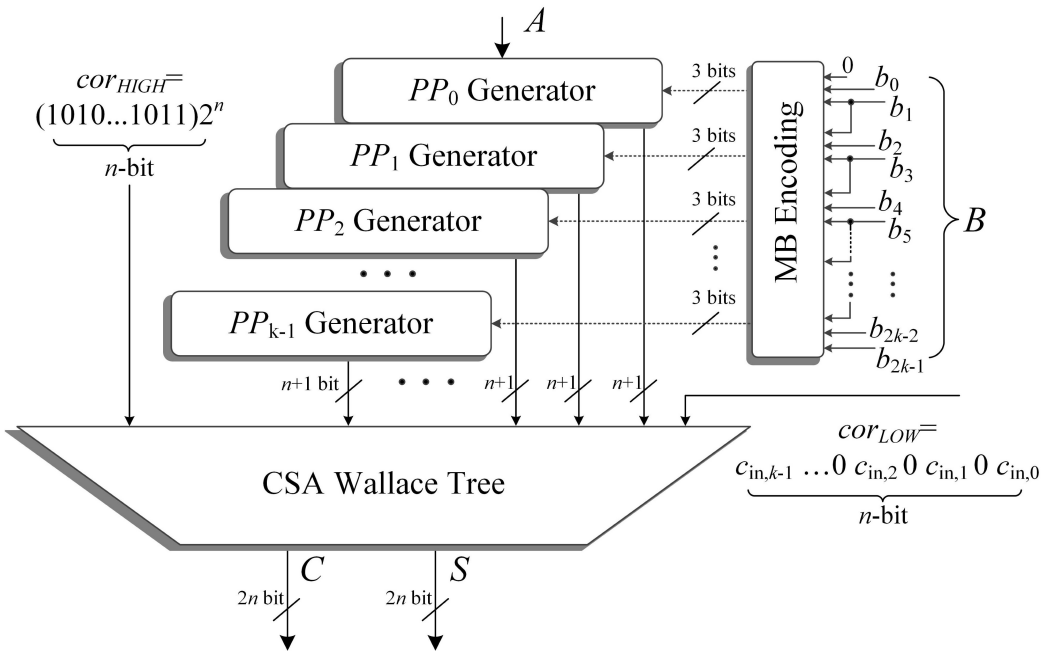
3.2.1 Υλοποίηση συμβατικού αλγόριθμου σε μορφή Carry-Save (A)

Στην πρώτη υλοποίηση έχουμε χρησιμοποιήσει τον συμβατικό αλγόριθμο για τον υπολογισμό ενός μιγαδικού πολλαπλασιασμού. Έτσι χρησιμοποιούμε τέσσερις πολλαπλασιαστές δυαδικών αριθμών σε μορφή συμπληρώματος ως προς 2, όπως περιγράφηκαν παραπάνω, δύο δένδρικούς συμπιεστές Wallace και δύο γρήγορους αθροιστές. Η υλοποίηση του δενδρικού συμπιεστή Wallace, όπου έδω λειτουργεί σαν ένας 4:2 συμπιεστής, όπως και η υλοποίηση του αθροιστή που χρησιμοποιούμε έγιναν με τα modules DW02_tree και DW01_add αντίστοιχα.

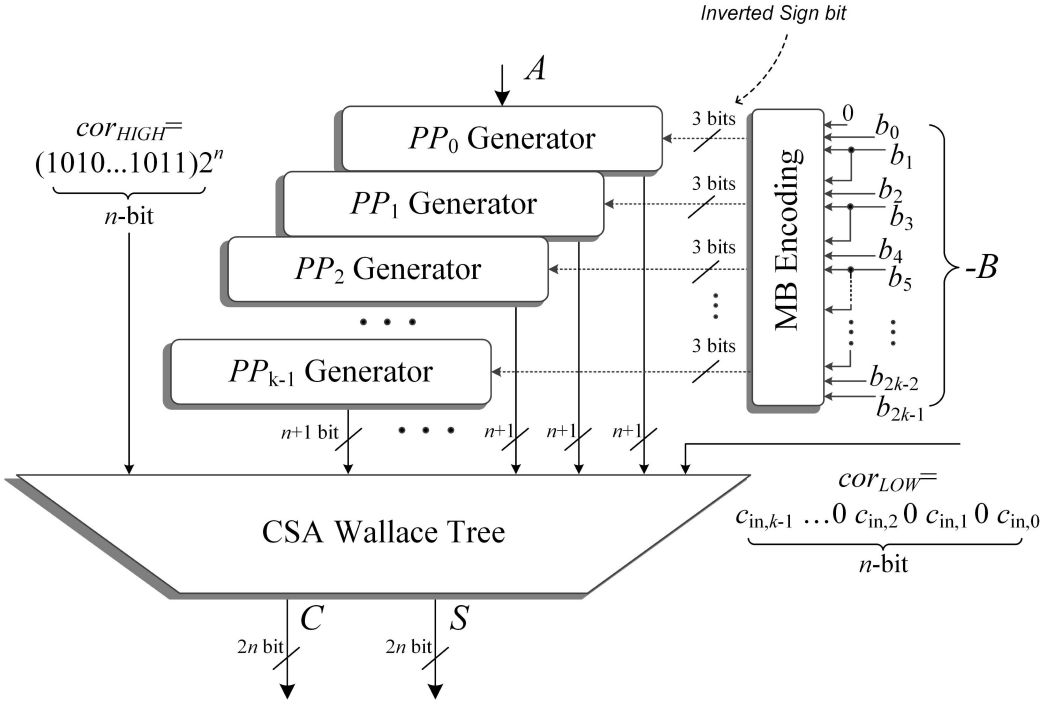
Έστω ο πολλαπλασιασμός δύο μιγαδικών αριθμών X και Y, όπου $A = A + j \cdot B$ και $B = C + j \cdot D$ με αποτέλεσμα $R + j \cdot I$. Έτσι η υλοποίησή μας φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.7 Υλοποίηση συμβατικού αλγόριθμου σε μορφή Carry-Save (A)



Σχήμα 3.8 Υλοποίηση Carry-Save MB Πολλαπλασιαστή A · B

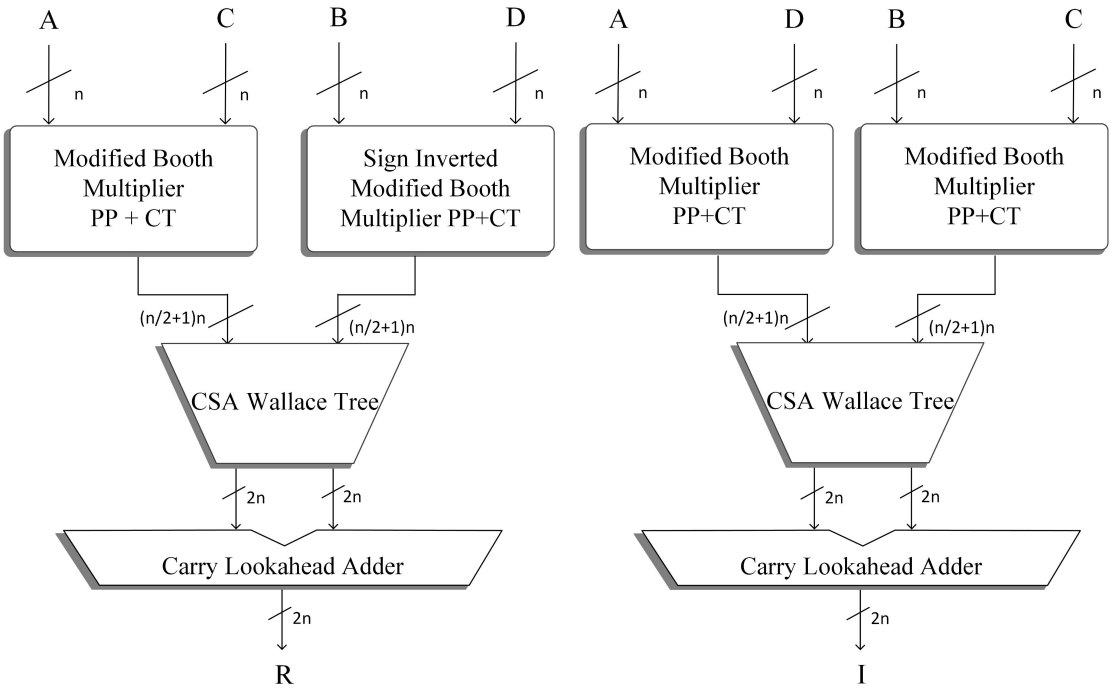


Σχήμα 3.9 Υλοποίηση Sign Inverted Carry-Save MB Πολλαπλασιαστή A · (-B)

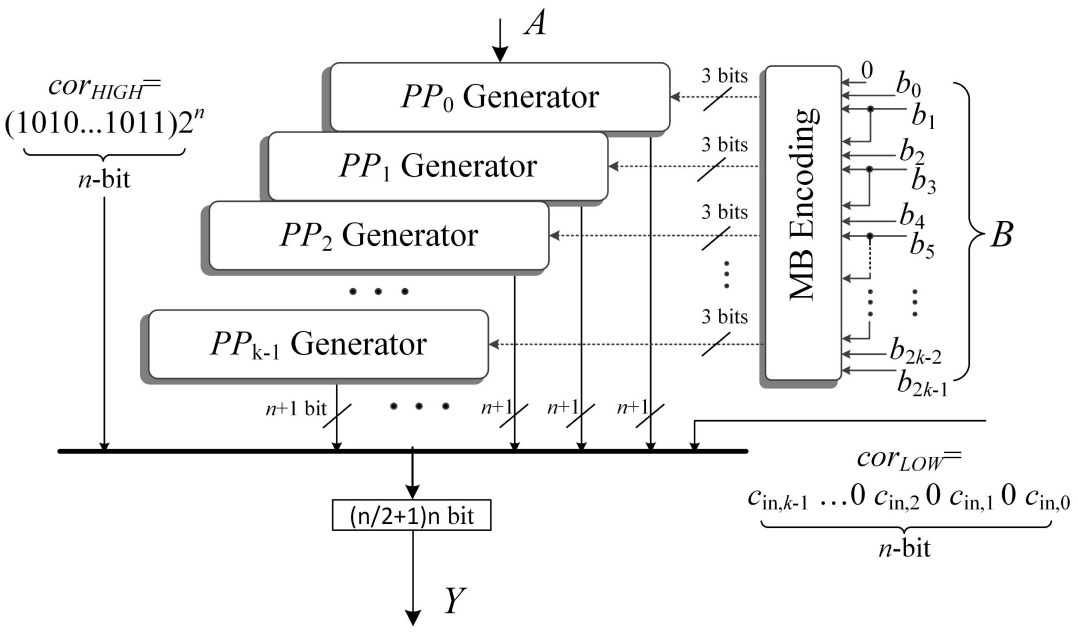
Η μονάδα SI Modified Booth Multiplier CS είναι η μονάδα που κάνει την παραγωγή του όρου BD στον μιγαδικό πολλαπλασιασμό και υλοποιήθηκε όπως περιγράφηκε και παραπάνω στο υποκεφάλαιο *3.1.1.1* για το κύκλωμα κωδικοποίησης σε Modified Booth, και αποτελεί ξεχωριστό κύκλωμα γιατί έτσι παράγουμε τον όρο BD με το αντεστραμένο χωρίς να κάνουμε καμία αλλαγή προσήμου.

3.2.2 Υλοποίηση συμβατικού αλγόριθμου σε μορφή Carry-Save (B)

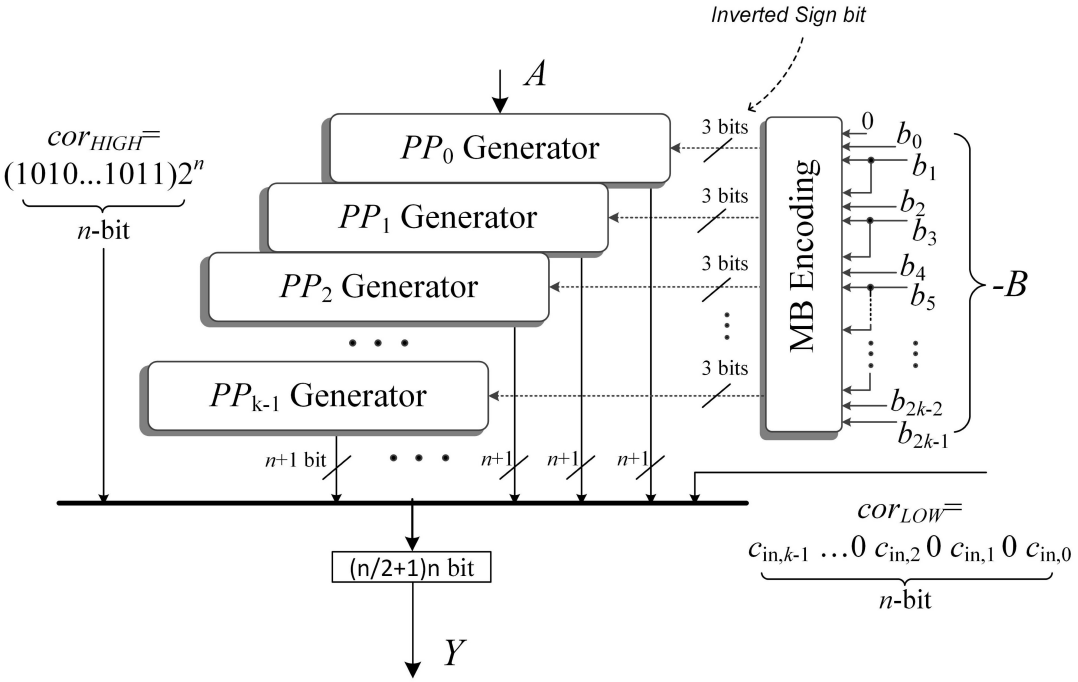
Στην δεύτερη υλοποίηση κάναμε χρήση της προηγούμενης υλοποίησης με μια παραλλαγή στο στάδιο του πολλαπλασιαστή. Στο τελικό στάδιο του πολλαπλασιαστή, αφού συγκεντρώσουμε όλα τα μερικά γινόμενα με τις απαραίτητες επεκτάσεις προσήμου και τους διορθωτικούς όρους, οδηγούνται σε ένα διάνυσμα που με την σειρά του χρησιμοποιείται σαν είσοδος στον δενδρικό συμπιεστή Wallace. Την έξοδο του πολλαπλασιαστή όμως, παίρνουμε το αποτέλεσμα σε Carry-Save μορφή και το οδηγούμε σε ένα καινούργιο διάνυσμα μαζί με το αποτέλεσμα του δεύτερου πολλαπλασιαστή με σκοπό να τους προσθέσουμε με έναν δενδρικό συμπιεστή Wallace για να πάρουμε το τελικό αποτέλεσμα, είτε πραγματικό είτε φανταστικό μέρος, σε Carry-Save μορφή. Δεδομένης της αποδοτικότητας και της οικονομίας σε κατανάλωση και επιφάνεια του δενδρικού συμπιεστή Wallace, χρησιμοποιούμε από την μονάδα πολλαπλασιασμού, το διάνυσμα που περιέχει τα μερικά γινόμενα ως έξοδο του πολλαπλασιαστή τα οποία οδηγούμε σε ένα καινούργιο διάνυσμα μαζί με το αντίστοιχό του από τον έταιρο πολλαπλασιαστή που πρέπει να προστεθεί στο αποτέλεσμα του πρώτου. Έτσι καταλήγουμε με ένα διάνυσμα που περιέχει τα μερικά γινόμενα από δύο πολλαπλασιαστές τα οποία εισέρχονται σε έναν κοινό δενδρικό συμπιεστή Wallace. Από την έξοδο του δένδρου λαμβάνουμε και πάλι είτε το πραγματικό, είτε το φανταστικό μέρος του μιγαδικού αριθμού που είναι αποτέλεσμα του πολλαπλασιασμού δύο μιγαδικών A και B. Στην συνέχεια έχουμε το σχήμα της υλοποίησής μας.



Σχήμα 3.10 Υλοποίηση συμβατικού αλγόριθμου σε μορφή Carry-Save (B)



Σχήμα 3.11 Κυκλώματος παραγωγής μερικών γινομένων και διορθωτικών όρων (Partial Products + Correction Term) του MB πολλαπλασιασμού A · B



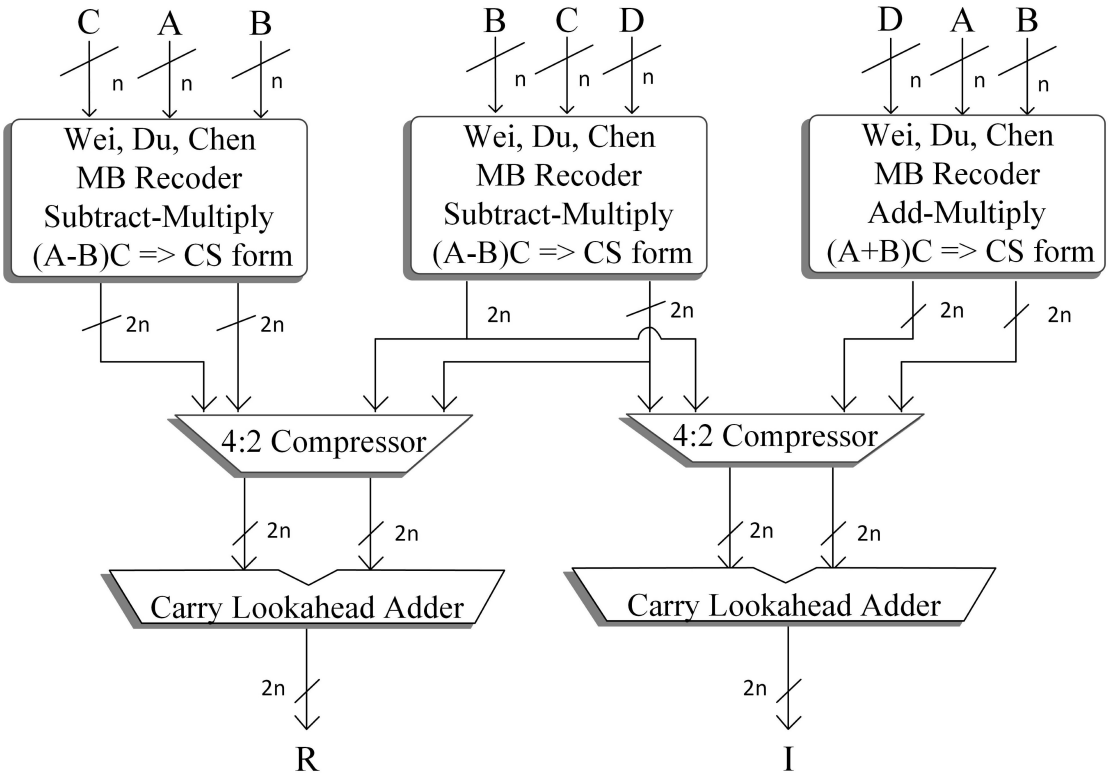
Σχήμα 3.12 Κυκλώματος παραγωγής μερικών γινομένων και διορθωτικών όρων (Partial Products + Correction Term) του MB πολλαπλασιασμού A · (-B)

3.2.3 Υλοποίηση αλγόριθμου Gauss με χρήση του Επανακωδικοποιητή σε Modified Booth, αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen

Η υλοποίηση αυτή χρησιμοποιεί τον αλγόριθμό του Gauss για τον πολλαπλασιασμό δύο μιγαδικών, μήκους λέξης n bits. Έστω ο μιγαδικός πολλαπλασιασμός (A+jB)(C+jD) = R+jI, όπου R = AC-BD και I = AD+BC. Το πραγματικό και το φανταστικό μέρος μπορούν να παραχθούν ως εξής:

$$m_0 = (C - D) \cdot B \qquad R = m_1 + m_0$$
$$m_1 = (A - B) \cdot C \qquad I = m_2 + m_0$$
$$m_2 = (A + B) \cdot D$$

Οι είσοδοί μας οδηγούνται σε τρεις μονάδες για να γίνουν οι πράξεις για τον υπολογισμό των m_0, m_1, m_2. Οι τρεις αυτές μονάδες υλοποιούν τις πράξεις $(A+B) \cdot X$ και $(A-B) \cdot X$ όπως τις περιγράψαμε παραπάνω στο 3.1.3. Στην έξοδό τους έχουμε το αποτέλεσμα σε carry-save μορφή με μήκος λέξης 2n bits που με την χρήση δύο δενδρικών συμπιεστών και δύο αθροιστών της Designware υπολογίζουμε το τελικό αποτέλεσμα, όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.13 Υλοποίηση αλγόριθμου Gauss με χρήση του Επανακωδικοποιητή σε Modified Booth, αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen

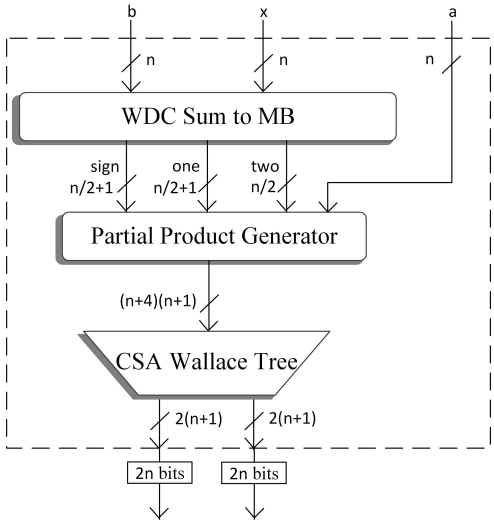
Όπως φαίνεται στο παραπάνω σχήμα κάνουμε χρήση δύο είδη μονάδων για την υλοποίηση των πράξεων Πρόσθεσης-Πολλαπλασιασμού και Αφαίρεσης-Πολλαπλασιασμού.

Η μονάδα Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού υλοποιείται σύμφωνα με το παρακάτω σχήμα όπου κάνουμε χρήση των μονάδων:

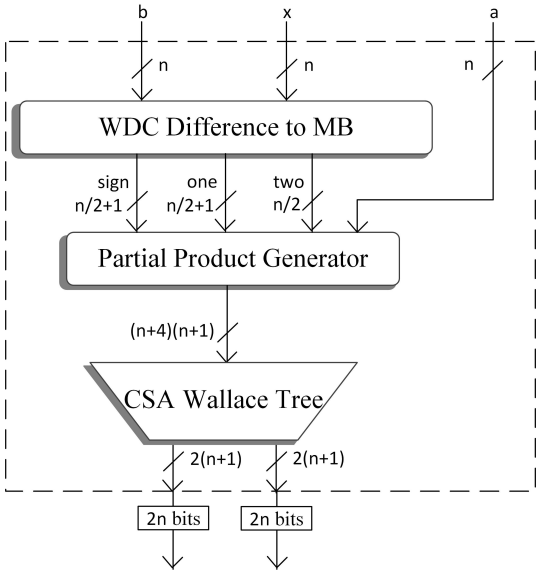
Partial Product Generator = Μονάδα παραγωγής μερικών γινομένων όπως περιγράφηκε στο *3.1.1.2*

WDC Sum to MB = Κωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής Πρόσθεσης των Wei, Du, Chen

WDC Difference to MB = Κωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής Αφαίρεσης των Wei, Du, Chen



Σχήμα 3.14 Μονάδα Πρόσθεσης-Πολλαπλασιασμού των Wei, Du και Chen

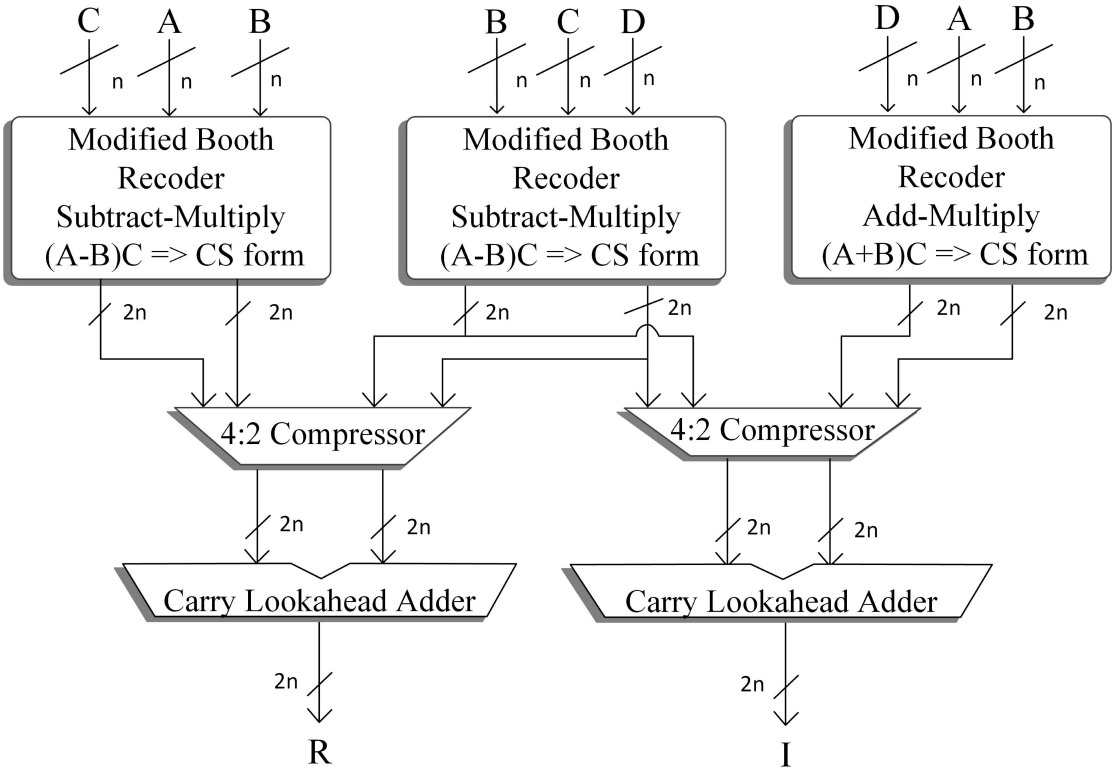


Σχήμα 3.15 Μονάδα Αφαίρεσης-Πολλαπλασιασμού των Wei, Du και Chen

Ο υπολογισμός του πολλαπλασιασμού γίνεται με έξοδο 2(n+1) bits για τον συνυπολογισμό του κρατουμένου εξόδου. Στην συνέχεια τα επιπλέον 2 bits απαλείφονται δεδομένου ότι ο μεγαλύτερος αριθμός στην έξοδο είναι μήκους 2n bits.

3.2.4 Υλοποίηση αλγόριθμου Gauss με χρήση του Προτεινόμενου σχήματος επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού (Α).

Με την ίδια λογική αλλά με διαφορετικό σχήμα επανακωδικοποίησης υλοποιήσαμε το προηγούμενο σχήμα. Εδώ κάνουμε χρήση του επανακωδικοποιητή Πρόσθεσης-Πολλαπλασιασμού και Αφαίρεσης-Πολλαπλασιασμού και στην συνέχεια θα τροποποιήσουμε το πρώτο σχήμα, όπως αυτό φαίνεται παρακάτω, για να αποκτήσουμε δύο καινούργιες υλοποιήσεις και να δούμε την επίδραση των παραλλαγών αυτών στην ταχύτητα, την κατανάλωση και στην επιφάνεια που καταλαμβάνει το κύκλωμά μας.

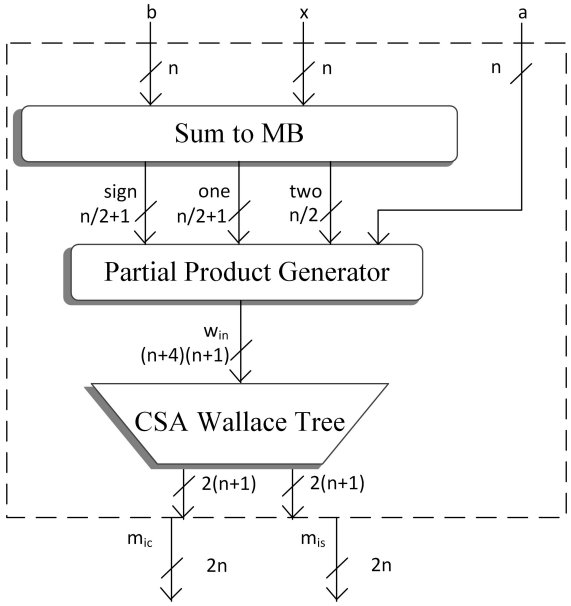


Σχήμα 3.16 Υλοποίηση αλγόριθμου Gauss με χρήση του Προτεινόμενου σχήματος επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού (Α)

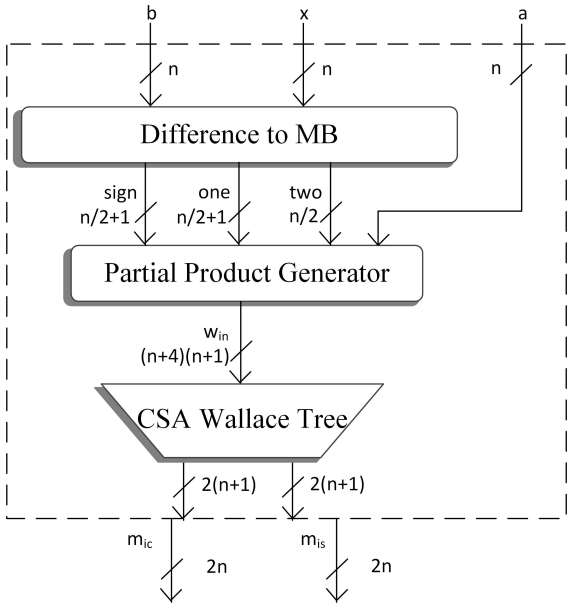
Στην παρούσα υλοποίηση κάνουμε χρήση δύο είδη μονάδων για την υλοποίηση των πράξεων Πρόσθεσης-Πολλαπλασιασμού και Αφαίρεσης-Πολλαπλασιασμού, τις μονάδες MB Recoder Add-Multiply και MB Recoder Subtract-Multiply αντίστοιχα. Στα παρακάτω σχήματα γίνεται η περιγραφή των δύο αυτών μονάδων, στο εσωτερικό των οποίων κάνουμε χρήση των μονάδων που περιγράφηκαν παραπάνω:

Sum to MB = Σχήμα επανακωδικοποίησης Πρόσθεσης-Πολλαπλασιασμού

Difference to MB = Σχήμα επανακωδικοποίησης Αφαίρεσης-Πολλαπλασιασμού



Σχήμα 3.17 Μονάδα Πρόσθεσης-Πολλαπλασιασμού Προτεινόμενου Σχήματος επανακωδικοποίησης (MB Recoder Add-Multiply)

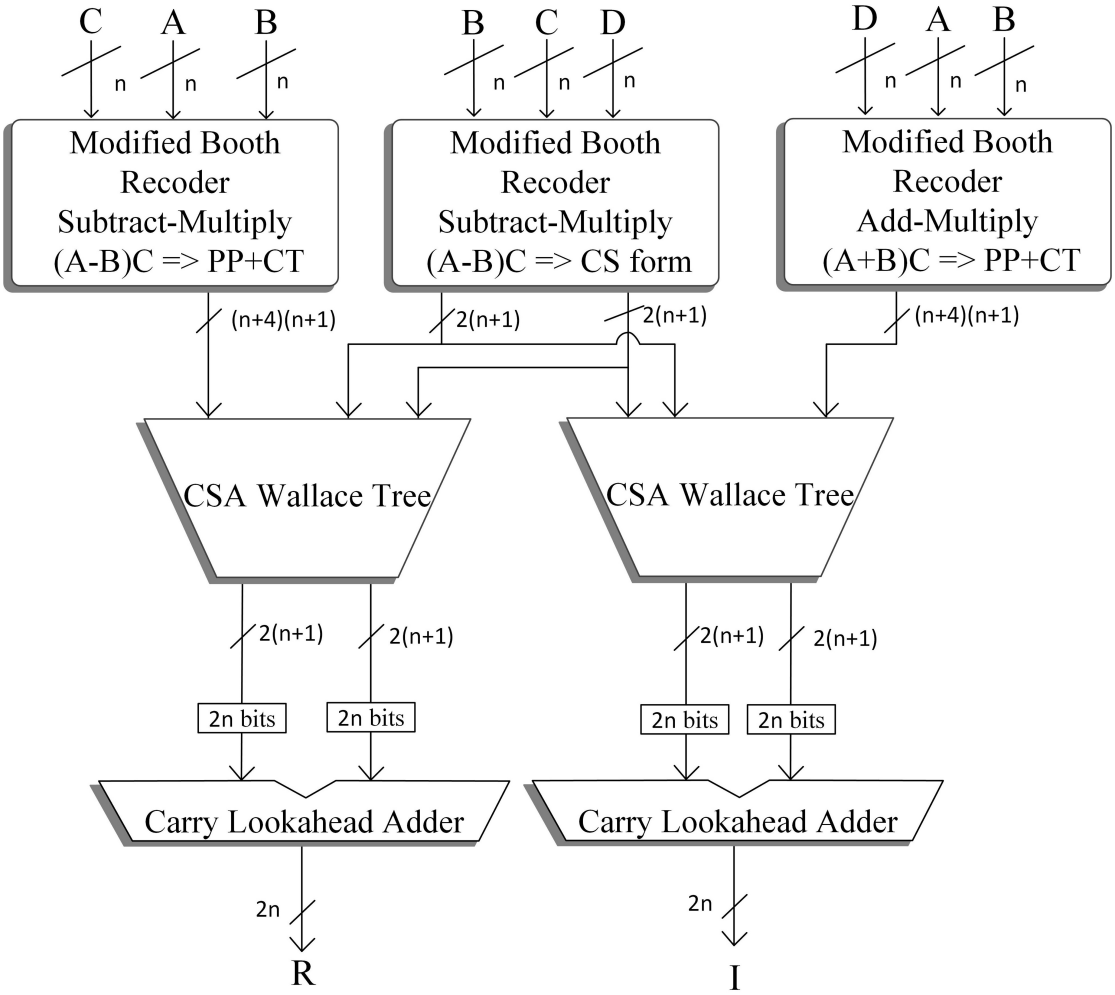


Σχήμα 3.18 Μονάδα Αφαίρεσης-Πολλαπλασιασμού Προτεινόμενου Σχήματος επανακωδικοποίησης (MB Recoder Subtract-Multiply)

Εδώ παρατηρούμε ότι ο υπολογισμός του m_0 είναι κοινός για τα δύο δένδρα. Έτσι οδηγούμαστε στην παρατήρηση ότι θα μπορούσε να γίνει ένας καλύτερος χειρισμός του δενδρικού συμπιεστή Wallace για τον γρηγορότερο υπολογισμό του αποτελέσματος, που μας οδηγεί στην επόμενη υλοποίηση.

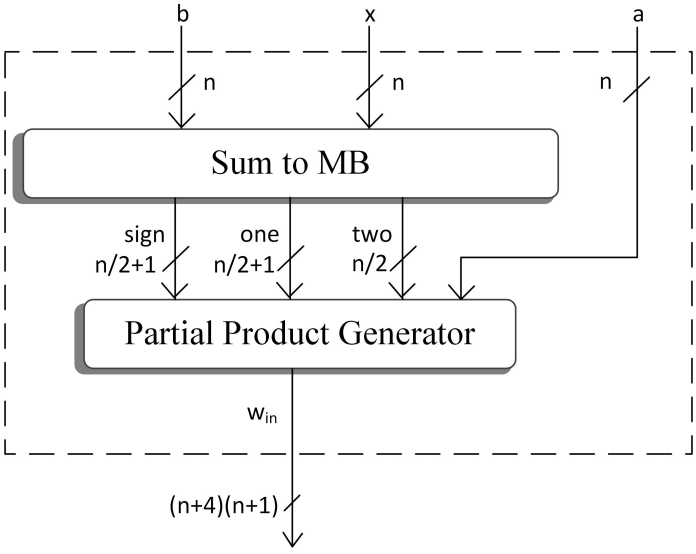
3.2.5 Υλοποίηση αλγόριθμου Gauss με χρήση του Προτεινόμενου σχήματος επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού (B).

Σε αυτήν την υλοποίηση θα εξετάσουμε κατά πόσο μπορούμε να επωφεληθούμε από το γεγονός ότι ο όρος m_0 είναι κοινός στον υπολογισμό του πραγματικού και του φανταστικού μέρους στον αλγόριθμο του Gauss. Έτσι προχωρήσαμε σε μία υλοποίηση όπου ο υπολογισμός του m_0 φθάνει στο σημείο όπου έχουμε το αποτέλεσμα σε carry-save μορφή με μήκος λέξης 2(n+1). Ο υπολογισμός των m_1, m_2 γίνεται με απαλοιφή του τελικού δενδρικού συμπιεστή Wallace στο σχήμα επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού. Έτσι με κατάλληλη τοποθέτηση στον κώδικα του m_0, στα διανύσματα μαζί με τα m_1 και m_2, που οδήγουμε στο Wallace tree, θα γίνει η συμπίεση των bits του m_0 στα χαμηλότερα επίπεδα της υλοποίηση το οποίο μπορεί να μας προσφέρει ένα γρηγορότερο κύκλωμα. Στην συνέχεια έχουμε το σχήμα αυτής της υλοποίησης:

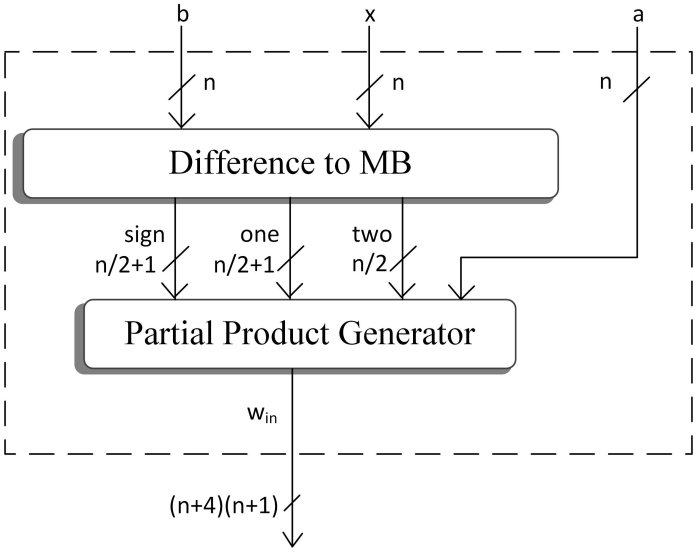


Σχήμα 3.19 Υλοποίηση αλγόριθμου Gauss με χρήση του Προτεινόμενου σχήματος επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού (B)

Βλέπουμε ότι κάνουμε χρήση της μονάδας MB Recoder Subtract-Multiply, η οποία περιγράφηκε στην προηγούμενη παράγραφο, αλλά κάνουμε χρήση δύο ακόμα μονάδων, τις MB Recoder Add-Multiply (A+B)C=>PP+CT και MB Recoder Subtract-Multiply (A-B)C=>PP+CT. Αυτές οι μονάδες είναι ίδιες με τις δύο προηγούμενες, MB Recoder Add-Multiply και MB Recoder Subtract-Multiply αντίστοιχα, με την διαφορά ότι έχουμε απαλείψει τους τελικούς δενδρικούς συμπιεστές Wallace. Στα παρακάτω σχήματα βλέπουμε το εσωτερικό των δύο αυτών μονάδων:



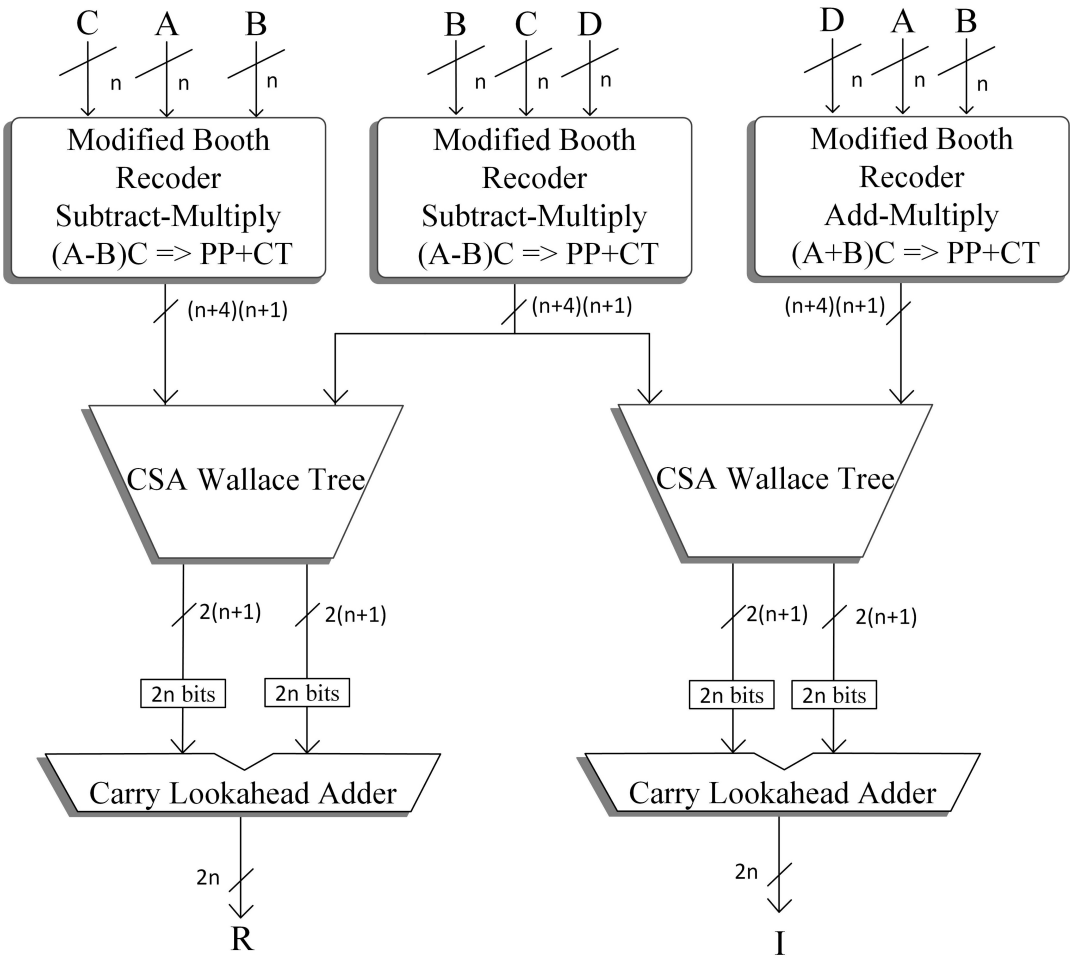
Σχήμα 3.20 Μονάδα Πρόσθεσης-Πολλαπλασιασμού Προτεινόμενου Σχήματος επανακωδικοποίησης (MB Recoder Add-Multiply => PP + CT)



Σχήμα 3.21 Μονάδα Αφαίρεσης-Πολλαπλασιασμού Προτεινόμενου Σχήματος επανακωδικοποίησης (MB Recoder Subtract-Multiply => PP + CT)

3.2.6 Υλοποίηση αλγόριθμου Gauss με χρήση του Προτεινόμενου σχήματος επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού (C).

Στην τελευταία υλοποίηση κάνουμε χρήση μόνο των δύο τελευταίων μονάδων που παρουσιάστηκαν παραπάνω στα σχήματα 3.16 και 3.17. Έτσι θα ελέγξουμε αν με την απαλοιφή όλων των ενδιάμεσων δενδρικών συμπιεστών Wallace και με την χρήση δύο τελικών δένδρων, θα έχουμε την μέγιστη δυνατή απλοποίηση του κυκλώματος. Αυτό εξετάζεται πειραματικά δεδομένου ότι τα εργαλεία που χρησιμοποιούμε για την εφαρμογή του κώδικα είτε σε ASIC, είτε σε FPGA, θα προβεί στον σχεδιασμό του κυκλώματος με τον βέλτιστο τρόπο ανάλογα με το μέσο στο οποίο θα έχουμε τελικά. Έτσι θα δούμε ανάλογα με τα εργαλεία που θα χρησιμοποιήσουμε ποια υλοποίηση αποδίδει καλύτερα, ανάλογα με την πλατφόρμα που χρησιμοποιούμε. Στην συνέχεια παρατίθεται το σχήμα της τελευταίας υλοποίησης με χρήση των μονάδων MB Recoder Subtract-Multiply (A-B)C=>PP+CT και MB Recoder Subtract-Multiply (A-B)C=>PP+CT και τους δενδρικούς συμπιεστές Wallace και τους adders της DesignWare.



Σχήμα 3.22 Υλοποίηση αλγόριθμου Gauss με χρήση του Προτεινόμενου σχήματος επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού (C)

4. Συγκριτικά Αποτελέσματα

4.1. Οργάνωση πειραμάτων

Οι τοπολογίες που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, περιγράφηκαν σε γλώσσα Verilog. Όσων αφορά την υλοποίηση των τοπολογιών αυτών σε ASIC συντέθηκαν από το Synopsys Design Compiler, κάνοντας χρήση της βιβλιοθήκης κελιών Faraday 90nm, και της εντολής –compile_ultra για βέλτιστα αποτελέσματα. Τα κυκλώματα που παρήχθησαν, προσομοιώθηκαν από το περιβάλλον Modelsim, για επιβεβαίωση της ορθής λειτουργίας τους.

Από το περιβάλλον του Design Compiler, εξήχθησαν οι τιμές καθυστέρησης και επιφάνειας κάθε κυκλώματος, ενώ από το περιβάλλον του Synopsys Primepower (με βάση το .vcd αρχείο που προέκυπτε από κάθε προσομοίωση του Modelsim) υπολογίστηκε η μέση κατανάλωση κάθε κυκλώματος.

Για την προσομοίωση της λειτουργίας κάθε κυκλώματος χρησιμοποιήθηκαν 65536 ζεύγη τυχαίων δυαδικών αριθμών, μήκους λέξης n.

Όλα τα κυκλώματα υλοποιήθηκαν για μήκη λέξης n = 8, 16, 24, 32, 48, 64.

Για όλα τα κυκλώματα, λάβαμε μετρήσεις στην χαμηλότερη δυνατή συχνότητα ρολογιού που μπορέσαμε να προσομοιώσουμε το εκάστοτε κύκλωμα. Η περίοδος του ρολογιού για αυτό το σκοπό μεταβάλλονταν με βήμα 0.01ns για 10 επαναλήψεις. Στην συνέχεια πήραμε 10 μετρήσεις από τον αμέσως μεγαλύτερο αριθμό κατά ένα δέκατο, της τελευταίας μέτρησης, με βήμα 0.1ns για να εξετάσουμε τα κυκλώμα σε συνθήκες χαλαρότερων περιορισμών ρολογιού.

Στην συνέχεια για την ίδια υλοποίηση σε FPGA επιλέχθηκε η οικογένεια Virtex 7, της Xilinx όπου χρησιμοποιήθηκε το εργαλείο ISE. Τα κυκλώματα που παρήχθησαν προσομοιώθηκαν στο ISim(Xilinx) για την επαλήθευση της ορθής λειτουργίας τους, και η σύνθεση έγινε με το XST(Xilinx).

Για όλα τα κυκλώματα υπολογίστηκε η χαμηλότερη περίοδος ρολογιού με την βοήθεια των timing reports του ISE, και ο υπολογισμός της καταναλησκώμενης ενέργειας έγινε με το εργαλείο XPower Analyzer της Xilinx.

Τέλος τα κυκλώματα συγκρίθηκαν ως προς την καθυστέρηση, την επιφάνεια και την ισχύ που καταναλώνουν.

4.2. Υλοποίηση σε ASIC

Αρχικά πραγματοποιήσαμε έναν θεωρητικό υπολογισμό της επιφάνειας που θα καταλαμβάνει η κάθε υλοποίηση καθώς και το κρίσιμο μονοπάτι τους. Για αυτόν τον σκοπό μελετήσαμε τις δομικές μονάδες ξεχωριστά, σε επίπεδο πυλών, για να καταλήξουμε σε μια θεωρητική εκτίμηση των υλοποιήσεων. Για την ανάλυσή μας βασιστήκαμε στο μοντέλο μονάδας-πύλης (unit-gate model) όπου για τις ποσοτικές μας συγκρίσεις οι λογικές πύλες NAND, AND, NOR, OR δύο εισόδων μετράνε σαν μία πύλη, και όσων αφορά καθυστέρηση αλλά και όσων αφορά την επιφάνεια που καταλαμβάνουν. Σε αντίθεση, οι πύλες XOR και XNOR δύο εισόδων μετράνε σαν δύο πύλες και στις δύο περιπτώσεις. Η επιφάνεια ενός πλήρους αθροιστή και ενός ημιαθροιστή είναι 4 και 3 πύλες αντίστοιχα και η καθυστέρηση του αθροίσματος και του κρατουμένου εξόδου του πλήρους αθροιστή είναι 4 και 3 πύλες και αντίστοιχα στον ημιαθροιστή είναι 2 και 1 πύλες. Όλα τα παραπάνω συνοψίζονται στον παρακάτω πίνακα:

AREA AND DELAY OF VARIOUS COMPONENTS IN THE UNIT GATE MODEL.

Components	*Area (Gate equivalents)*	*Delay (Gate equivalents)*
NAND-2, NOR-2	A_g	T_g
NAND-3, NOR-3	$2A_g$	$2T_g$
XOR, XNOR	$2A_g$	$2T_g$
HA	$3A_g$	$T_{HA,carry}=T_g$, $T_{HA,sum}=2T_g$
FA	$7A_g$	$T_{FA,carry}=3T_g$, $T_{FA,sum}=4T_g$

Πίνακας 4.1 Επιφάνεια και καθυστέρηση δομικών μονάδων

4.2.1. Θεωρητική ανάλυση

1. Κύκλωμα κωδικοποίησης

Όπως περιγράφηκε και προηγουμένως το κύκλωμα κωδικοποιήσεις προκύπτει από τις παρακάτω εξισώσεις:

$$one_i = b_{2i} \oplus b_{2i-1}$$
$$two_i = (b_{2i+1}' \cdot b_{2i} \cdot b_{2i-1}) + (b_{2i+1} \cdot b_{2i}' \cdot b_{2i-1}')$$
$$sign_i = b_{2i+1}$$

Έτσι βλέπουμε ότι το κρίσιμο μοναπάτι θα έχει καθυστέρηση 2 πυλών και η επιφάνεια που θα καταλαμβάνει 1XOR + 4AND+1OR = $7A_g$.

Γενικεύοντας για τον πολλαπλασιασμό δύο προσημασμένων αριθμών σε μορφή συμπληρώματος ως προς 2, με μήκος λέξης n bits, ξέρουμε ότι τα Modified Booth ψηφία σε

έναν τέτοιο πολλαπλασιασμό είναι n/2, άρα στο κύκλωμά μας πρέπει να λάβουμε υπόψιν ότι η επιφάνεια που θα καταλαμβάνει θα είναι ανάλογη του μήκους λέξης.

2. Κύκλωμα δημιουργίας μερικών γινομένων

Ομοίως με πριν γνωρίζουμε ότι για το κύκλωμα δημιουργία μερικών γινομένων ισχύουν οι παρακάτω εξισώσεις.

Το λιγότερο σημαντικό bit του μερικού γινομένου υπολογίζεται ως εξής:

$$pp\_out_0 = ((a_0 \oplus sign) \cdot one) + (sign \cdot two)$$

Η γενική εξίσωση υπολογισμού του μερικού γινομένου είναι:

$$pp\_out_i = ((a_i \oplus sign) \cdot one) + ((a_{i-1} \oplus sign) \cdot two)$$

Τέλος το πιο σημαντικό bit του μερικού γινομένου είναι

$$pp\_out_n = (((a_{n-1} \oplus sign) \cdot one) + ((a_{n-1} \oplus sign) \cdot two))' :$$

όπου n είναι το μήκος λέξης των δύο αριθμών που πολλαπλασιάζουμε.

Έτσι μπρούμε να καταλήξουμε ότι για το υποκύκλωμα δημιουργίας ενός μερικού γινομένου μήκους n+1 bits θα έχουμε καθυστέρηση 2XOR+AND+OR = $6T_g$, και η επιφάνειά του θα είναι (n-1)(2XOR+2AND+2OR)+1XOR+1OR+2AND = (n-1)$4A_g$+$5A_g$. Και για το συνολικό κύκλωμα θα έχουμε n/2 μερικά γινόμενα.

3. Δενδρικός Συμπιεστής Wallace

Ο δενδρικός συμπιεστής Wallace, όπως έχει περιγραφεί και παραπάνω, χρησιμοποιείται για την περίπτωση που θέλουμε να προσθέσουμε παραπάνω από δύο αριθμούς με μήκος λέξης μεγαλύτερο των 16 bits μιας και τότε αποδίδει καλύτερα από την χρήση οποιουδήποτε τύπου αθροιστή. Αυτό επιτυγχάνεται με την χρήση 3:2 συμπιεστών, δηλαδή FA, για κάθε τρια bits σε κάθε επίπεδο ελαχιστοποίησής του και 2:1 συμπιεστών, δηλαδή HA, όταν σε ένα επίπεδο έχουμε 2 bits ελεύθερα. Όταν ένα bit έχει μείνει μόνο του θα ληφθεί υπ'όψην στο επόμενο επίπεδο. Έτσι καταλήγουμε ότι το δένδρο Wallace απαιτεί

$$\log_{3/2}(\frac{n}{2})$$

επίπεδα FA και HA, για να μειώσει τις εισόδους σε 2 εξόδους σε μορφή σωσίματος-κρατουμένου(carry-save), όπου n είναι το πλήθος των εισόδων που έχουμε.

Δεδομένου του τρόπου υλοποίησης του δενδρικού συμπιεστή Wallace ο θεωρητικός υπολογισμός, της επιφάνειας του δένδρου Wallace θα γίνει για κάθε μήκος λέξης και για κάθε υλοποίηση ξεχωριστά.

4. Αθροιστής Πρόβλεψης Κρατουμένου

Σε κάθε μια από τις διαφορετικές υλοποιήσεις έχουμε ένα κοινό στοιχείο, τον τελικό "γρήγορο" αθροιστή όπου αθροίζουμε το αποτέλεσμα που βρίσκεται σε carry-save μορφή. Ο αθροιστής αυτός είναι ένας Αθροιστής Πρόβλεψης Κρατουμένου και αθροίζει δύο αριθμούς μήκους λέξης 2n bits. Όπως είδαμε και στο 2.3.2.3 ο αθροιστής πρόβλεψης κρατουμένου έχει πιο πολύπλοκη δομή, αλλά είναι πιο γρήγορος από τον αθροιστή διάδοσης κρατουμένου, καθώς αποφεύγει την διάδοση κρατουμένου και παράγει τα αποτελέσματά του με καθυστέρηση ενός HA συν logN κυκλωμάτων (#) συν μιας πύλης XOR. Όπως είδαμε και παραπάνω το κύκλωμα # έχει καθυστέρηση δύο πυλών T_g και επιφάνεια τριών A_g. Ο αριθμός των κυκλωμάτων πρόβλεψης κρατουμένου (#) που απαιτούνται είναι $N*\log N/2$, όπου N το πλήθος των ψηφίων του αθροιστή. Συγκεκριμένα στα δικά μας κυκλώματα όπου έχουμε πρόσθεση 2n bits θα έχουμε καθυστέρηση $1HA+2T_{g*}\log 2n+1XOR = 4T_g + \log 2n$, και συνολική επιφάνεια $2n(HA+XOR)+3A_{g*}2n\log n = 10nA_g+6nA_{g*}\log n$.

5. Προτεινόμενο Σχήμα επανακωδικοποίησης Πρόσθεσης(Αφαίρεσης)-Πολλαπλασιασμού

Όπως είδαμε και παραπάνω στο 3.1.2.1 η καθυστέρηση του κρίσιμου μοναπατιού του επανακωδικοποιητή είναι σταθερή ανεξάρτητα του μήκους λέξης της εισόδου και ίση με:

$$T_{S-MB} = T_{FA,carry} + T_{FA^*,sum}$$

όπου, όπως είδαμε και παραπάνω, η καθυστέρηση υπολογισμού του αθροίσματος ενός FA* είναι ίση με αυτήν του απλού FA. Όσων αφορά την επιφάνεια που θα καταλαμβάνει το κύκλωμα, όπως είδαμε και στο αντίστοιχο κεφάλαιο, το κύκλωμα αποτελείται από m κελιά που το κάθε κελί περιέχει έναν FA και έναν FA* δηλαδή συνολική επιφάνεια $14A_g$, και όπου m είναι τα πλήθος των Modified Booth ψηφίων, άρα m = n/2 και δεν λαμβάνουμε υπ'όψην το τελευταίο Modified Booth ψηφίο που μπορεί να είναι -1, 0, +1, μιας και το λαμβάνουμε σαν κρατούμενο εξόδου από τον τελευταίο FA*.

6. Επανακωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen

Ο επανακωδικοποιητής αυτός έχει την ιδιαιτερότητα, λόγω της αριθμητικής στην οποία βασίζεται και περιγράφηκε στον πίνακα 3.5, να υπολογίζει το άθροισμα-διαφορά που μετά θα κωδικοποιήθει σε μορφή Modified Booth με χρήση μόνο αντιστροφέων. Έτσι το κύκλωμα κωδικοποίησης είναι η μόνη επιβάρυνση του επανακωδικοποιητή στο κρίσιμο μονοπάτι και στην επιφάνεια που καταλαμβάνει. Στο σχήμα 3.6 βλέπουμε ότι το κρίσιμο μονοπάτι και η καθυστέρηση είναι 3XOR, 1AND τριών εισόδων και μία OR άρα συνολικά θα έχουμε καθυστέρηση ίση με $9T_g$. Όσων αφορά την συνολική επιφάνεια το κάθε κελί κωδικοποίησης σε Modified Booth, εκτός του τελευταίου που μπορεί να είναι -1, 0, +1, περιλαμβάνει 5XOR,

7AND,5OR και 6AND τριών εισόδων. Στο τελευταίο κελί θα έχουμε μια AND 2 εισόδων και μια AND 3 εισόδων λιγότερες, επειδή στο κελί αυτό η έξοδος μπορεί να είναι -1, 0, +1.

Άρα για την πράξη A(B+X) και A(B-X) όπου το μήκος λέξης είναι n bits θα έχουμε κύκλωμα επιφάνειας $17A_{g*}n+31A_g$, λαμβάνωντας υπ'όψην ότι το πλήθος των κελιών θα n/2, όσα και τα Modified Booth ψηφία που απαιτούνται για τον πολλαπλασιασμό.

Στην συνέχεια παραθέτουμε έναν πίνακα όπου συγκεντρώνουμε τα αποτελέσματα του θεωρητικού υπολογισμού για τις υλοποιήσεις μας:

	Conventional Carry-Save (A)	Conventional Carry-Save (B)	Gauss Recoder Wei, Du, Chen	Gauss Optimized Modified Booth Recoder (A)	Gauss Optimized Modified Booth Recoder (B)	Gauss Optimized Modified Booth Recoder (C)
n	Area/Ag	Area/Ag	Area/Ag	Area/Ag	Area/Ag	Area/Ag
8	2880	2880	3305	2972	3028	3532
16	12352	10560	10165	9592	9200	11552
24	28296.47	22920.47	20553.47	19740.47	18452.47	23996.47
32	50688	39936	34445	33392	30760	35296
48	114768.94	87888.94	72702.94	71169.94	64496.94	79448.94
64	204544	154368	124861	122848	110360	152024

Πίνακας 4.2 Θεωρητική σύγκριση υλοποιήσεων ως προς την επιφάνεια με παράγοντα A_g

	Conventional Carry-Save (A)	Conventional Carry-Save (B)	Gauss Recoder Wei, Du, Chen	Gauss Optimized Modified Booth Recoder (A)	Gauss Optimized Modified Booth Recoder (B)	Gauss Optimized Modified Booth Recoder (C)
n	Delay/Tg	Delay/Tg	Delay/Tg	Delay/Tg	Delay/Tg	Delay/Tg
8	40	40	41	39	53	45
16	54	46	51	49	67	55
24	63.17	51.17	56.17	54.17	76.17	56.17
32	72	56	57	55	81	57
48	81.17	61.17	62.17	60.17	86.17	62.17
64	90	66	67	65	95	71

Πίνακας 4.3 Θεωρητική σύγκριση υλοποιήσεων ως προς την καθυστέρηση με παράγοντα T_g

Από τους δύο παραπάνω πίνακες μπορούμε εύκολα να διαπιστώσουμε τα πλεονεκτήματα και τα μειονεκτήματα που θα περιμένουμε από τις υλοποιήσεις μας σε επιφάνεια και καθυστέρηση.

Από τον πίνακα 4.2 βλέπουμε πως θα πρέπει να περιμένουμε από την υλοποίηση του αλγόριθμου του Gauss να είναι η καλύτερη υλοποίηση. Αυτό φαίνεται από τα 16 bits και μετά όπου όλες οι υλοποιήσεις με τον αλγόριθμο του Gauss έχουν μικρότερη επιφάνεια από αυτές του συμβατικού αλγόριθμου για τον μιγαδικό πολλαπλασιασμό. Άλλη μια σημαντική παρατήρηση αφορά την επίδραση του δενδρικού συμπιεστή Wallace ανάμεσα στις υλοποιήσεις. Βλέπουμε ότι η χρήση του κοινού δενδρικού συμπιεστή Wallace στην υλοποίηση Conventional Carry-Save (B) επωφελεί το design, πράγμα που συμβαίνει και στο Gauss Optimized Modified Booth Recoder (B), αλλά όχι στο Gauss Optimized Modified Booth Recoder (C). Αυτό μπορεί να εξηγηθεί από τον κοινό όρο που παράγουμε στην carry-save μορφή στον αλγόριθμο του Gauss και γι'αυτό τον λόγο η δεύτερη υλοποίηση επωφελείται από την αποδοτικότητα του δενδρικού συμπιεστή Wallace. Έτσι παράγουμε το m_0 (κεφ.2.4.2) σε μορφή σωσίματος-κρατουμένου και εξοικονομούμε επιφάνεια από την τρίτη υλοποίηση όπου έχουμε τα ίδια δένδρα αλλά με πολύ μεγαλύτερο διάνυσμα εισόδου. Επίσης βλέπουμε ότι η υλοποίηση Gauss Optimized Modified Booth Recoder (A) είναι αρκετά κοντά στην (B) αλλά υπολείπεται, γιατί ο δενδρικός συμπιεστής Wallace αποδίδει καλύτερα σε μεγαλύτερα μήκη λέξης στην είσοδό του, έτσι όταν συγχωνεύσουμε τα δένδρα πριν τους τελικούς αθροιστές θα περιμένουμε καλύτερα αποτελέσματα. Τέλος, η υλοποίηση που κάνει χρήση του Gauss Recoder των Wei, Du, Chen αναμένεται να αποδίδει περίπου όμοια με τον Gauss Optimized Modified Booth Recoder (A), αφού το κέρδος της υλοποίησης από την έλλειψη χρήσης πλήρων αθροιστών στην παραγωγή του όρου πριν τον πολλαπλασιασμό, χάνεται από το κύκλωμα κωδικοποίησης που είναι εμφανώς πιο επιβαρυμένο.

Στον πίνακα 4.3 εξετάζουμε την καθυστέρηση στις υλοποιήσεις μας και βλέπουμε ότι κατά κανόνα ακολουθούν τα αποτελέσματα που είχαμε αντίστοιχα στην επιφάνεια των κυκλωμάτων, με μια σημαντική διαφορά. Βλέπουμε ότι η βέλτιστη υλοποίηση σε επιφάνεια, η υλοποίηση Gauss Optimized Modified Booth Recoder (B) είναι πλέον η χειρότερη σε καθυστέρηση. Αυτό το αποτέλεσμα είναι αναμενόμενο γιατί τα οφέλη του συγκεκριμένου design είναι σχεδιασμένο για να έχει την μικρότερη επιφάνεια σε βάρος της καθυστέρησης, μιας και η παραγωγή του όρου m_0 όπως περιγράφηκε και παραπάνω καθυστερεί το κύκλωμα κατά ένα στάδιο την παραγωγή του τελικού αποτελέσματος, όπως φαίνεται και στην περιγραφή της αρχιτεκτονικής που ακολουθήθηκε. Σε κάθε περίπτωση, βλέπουμε ότι όσων αφορά τον δενδρικό συμπιεστή Wallace, όπως χρησιμοποιήθηκε στα κυκλώματά μας, η χρήση κοινών δένδρων στον συμβατικό αλγόριθμο είναι καλύτερη και στον τομέα της καθυστέρηση, όπως ήταν στην επιφάνεια του κυκλώματος, σε αντίθεση με τις υλοποιήσεις με τον αλγόριθμο του Gauss όπου συμβαίνει το αντίθετο, και η καθυστέρηση ελαχιστοποιείται όταν κάνουμε χρήση επιμέρους δενδρικών συμπιεστών Wallace.

Στην συνέχεια θα εξετάσουμε κατά πόσο η σύνθεση και προσομοίωση των παραπάνω κυκλωμάτων θα επαληθεύσει τους θεωρητικούς υπολογισμούς που κάναμε καθώς και το πως συμπεριφέρονται όσων αφορά την κατανάλωση.

4.2.2. Πειραματική ανάλυση

Στον παρακάτω πίνακα έχουμε συγκεντρώσει τα αποτελέσματα για την χαμηλότερη δυνατή συχνότητα ρολογιού που μπορέσαμε να προσομοιώσουμε το εκάστοτε κύκλωμα:

Complex Multiplier	n = 8	n = 16	n = 24	n = 32	n = 48	n = 64
Conventional Carry-Save (A)	1.01	1.38	1.54	1.62	1.85	2.07
Conventional Carry-Save (B)	0.98	1.37	1.50	1.61	1.84	2.03
Gauss Recoder Wei, Du, Chen	1.23	1.54	1.72	1.82	2.02	2.28
Gauss Optimized Modified Booth Recoder (A)	1.17	1.51	1.65	1.77	2.01	2.24
Gauss Optimized Modified Booth Recoder (B)	1.22	1.68	1.92	1.79	2.00	2.22
Gauss Optimized Modified Booth Recoder (C)	1.14	1.53	1.68	1.78	2.04	2.25

Πίνακας 4.4 Σύγκριση υλοποιήσεων ως προς την χαμηλότερη περίοδο ρολογιού

Μια πρώτη εκτίμηση των αποτελεσμάτων όσων αφορά την ελάχιστη καθυστέρηση δείχνει ότι θεωρητικές και πειραματικές μετρήσεις συμφωνούν στα μικρότερα μήκη λέξης, αλλά όσο ανεβαίνουμε σε μήκος λέξης περιμέναμε τις υλοποιήσεις με τον αλγόριθμο του Gauss να έχουν την ίδια περίπου απόδοση με αυτές του συμβατικού αλγόριθμου, κάτι το οποίο δεν συμβαίνει. Επίσης βλέπουμε πως η καθυστέρηση της υλοποίησης Gauss Optimized Modified Booth Recoder (B) είναι καλύτερη από τις υπόλοιπες υλοποιήσεις του αλγόριθμου του Gauss. Για να εξηγήσουμε αυτές τις διαφορές θα πρέπει να ανατρέξουμε στις παραμέτρους που δώσαμε στον Design Compiler της Synopsys που καθορίζουν πως θα λειτουργήσει το εργαλείο κατά την μετάφραση του κώδικα και εν συνεχεία πως θα εφαρμόσει το σχέδιο στους περιορισμούς που του θέσαμε. Η εντολή που κάνει την επιλογή για αυτές τις παραμέτρους είναι η εντολή compile_ultra. Η εντολή αυτή αφορά designs υψηλής ταχύτητας που είναι σημαντικά ως προς τον χρονισμό τους, και χρησιμοποιείται για την βελτιστοποίησή τους, έτσι επιτρέπει στον compiler να εφαρμόσει το καλύτερο δυνατό σύνολο χρονοκεντρικών

μεταβλητών και εντολών ώστε η καθυστέρηση να είναι βέλτιστη. Επειδή αυτή η εντολή περιέχει όλες τις επιλογές για τον compiler και επηρεάζει όλη την διαδικασία της μετάφρασης πρέπει να γνωρίζουμε πως ακριβώς επιδρά στο κύκλωμά μας.

Η πρώτη παράμετρος [-scan] αφορά τα ακολουθιακά στοιχεία, όπου με την εντολή compile_ultra αντικαθίστανται με ισοδύναμα κελιά. Στην συνέχεια, έχουμε την παράμετρο [-no_uniquify] που είναι εύχρηστη για υλοποιήσεις με υποκυκλώματα που χρησιμοποιούνται πολλές φορές μέσα στον κώδικά μας. Η εντολή αυτή είναι ενεργοποιημένη και ευνοεί τα designs του συμβατικού αλγόριθμου που χρησιμοποιούν περισσότερες κοινές μονάδες στο εσωτερικό του από αυτές του αλγόριθμου του Gauss. Τέλος η παράμετρος [-no_autoungroup] έχει ως προεπιλογή να επιτρέπει στον compiler να καταργεί την ομαδοποίηση των μονάδων στο εσωτερικό του κυκλώματός μας, και να καταργεί τις ιεραρχίες στο κρίσιμο μονοπάτι με σκοπό την ελάχιστη χρονική καθυστέρηση.

Έτσι καταλήγουμε στο συμπέρασμα ότι ο Design Compiler τροποποιεί το κρίσιμο μονοπάτι το οποίο εξετάσαμε στο θεωρητικό μέρος, με σκοπό το κύκλωμα να έχει την βέλτιστη συμπεριφορά ως προς την καθυστέρηση. Έτσι στην συνέχεια θα συγκρίνουμε τις υλοποιήσεις, έχοντας υπ'όψην ότι οι μετρήσεις έχουν ληφθεί για να επιτευχθεί ο καλύτερος δυνατός χρονισμός και αυτό ενδεχομένως να έχει αρνητικές επιπτώσεις στις αναμενόμενες επιδόσεις των υλοποιήσεων όσων αφορά την επιφάνεια.

Για να εξετάσουμε τις μετρήσεις που έγιναν, θα τις χωρίσουμε ανάλογα με το μήκος λέξης των αριθμών σην είσοδο του μιγαδικού πολλαπλασιαστή και θα συγκρίνουμε τα αποτελέσματα με τα αντίστοιχα θεωρητικά. Επίσης θα συγκρίνουμε και τις υλοποιήσεις μεταξύ τους, ξεχωριστά σε κάθε μήκος λέξης, αλλά και συνολικά για την συμπεριφορά τους από την χαμηλότερη συχνότητα ρολογιού μέχρι και τις συνθήκες χαλαρότερων περιορισμών ρολογιού.

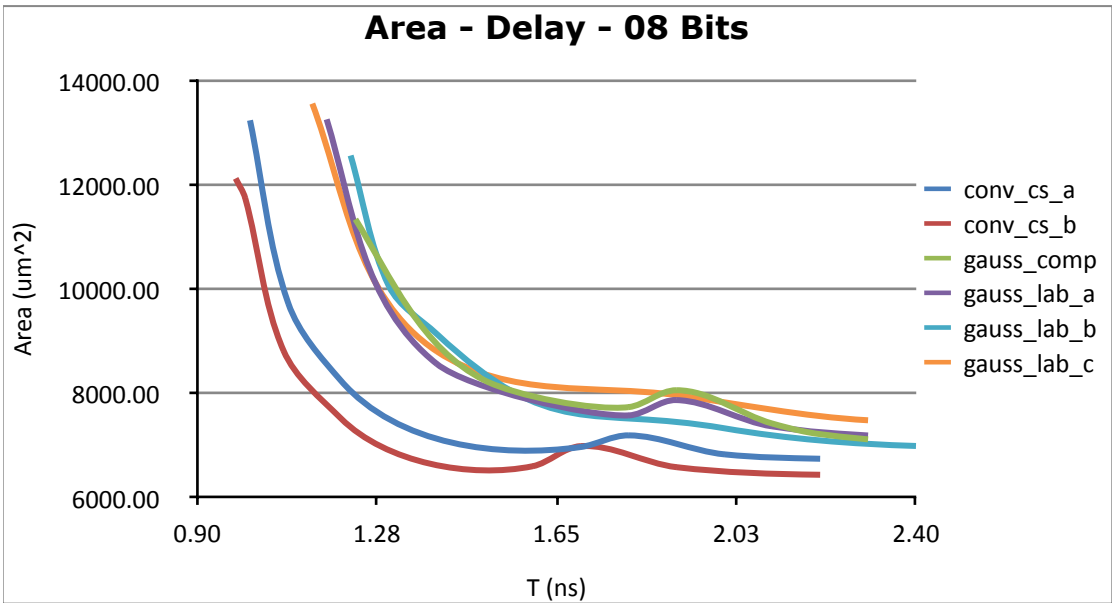
Στον παρακάτω πίνακα έχουμε την αντιστοιχία των υλοποιήσεων, με τα ονόματα που τους δώσαμε στα Verilog αρχεία:

Complex Multiplier	Project Name
Conventional Carry-Save (A)	conv_cs_a
Conventional Carry-Save (B)	conv_cs_b
Gauss Recoder Wei, Du, Chen	gauss_comp
Gauss Optimized Modified Booth Recoder (A)	gauss_lab_a
Gauss Optimized Modified Booth Recoder (B)	gauss_lab_b
Gauss Optimized Modified Booth Recoder (C)	gauss_lab_c

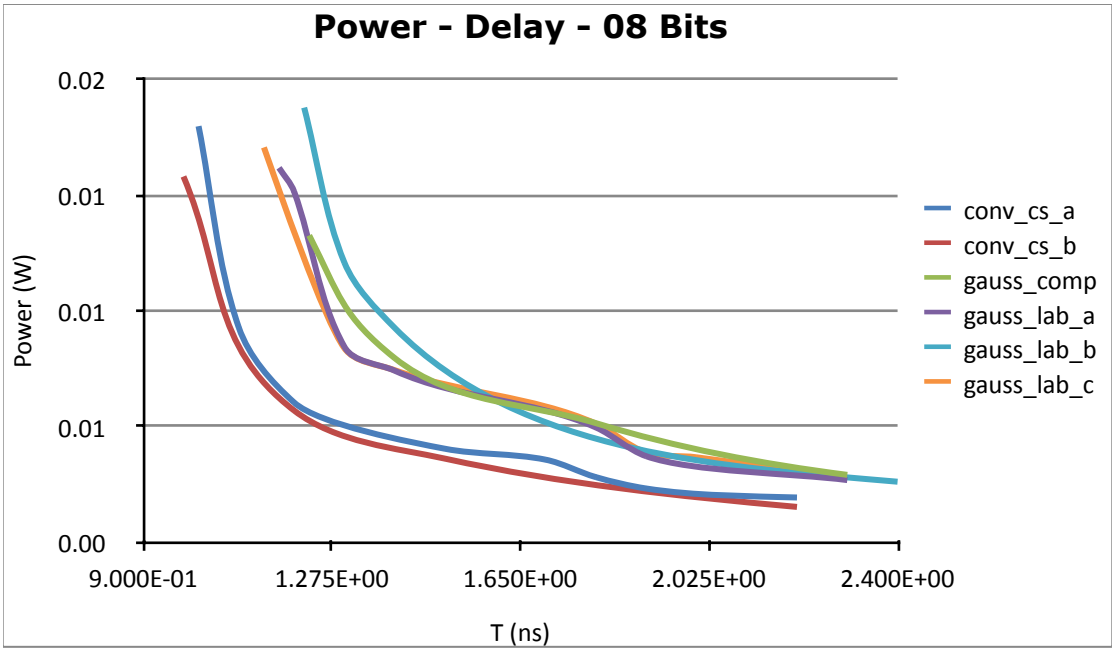
Πίνακας 4.5 Αντιστοιχία υλοποιήσεων με ονόματα στα Verilog modules

1. Μήκος λέξης 08 bits

Στα παρακάτω διάγραμματα παραθέτουμε τα αποτελέσματα των μετρήσεων που έγιναν:



Σχήμα 4.1 Σύγκριση υλοποιήσεων επιφάνειας-καθυστέρησης στα 08 bits

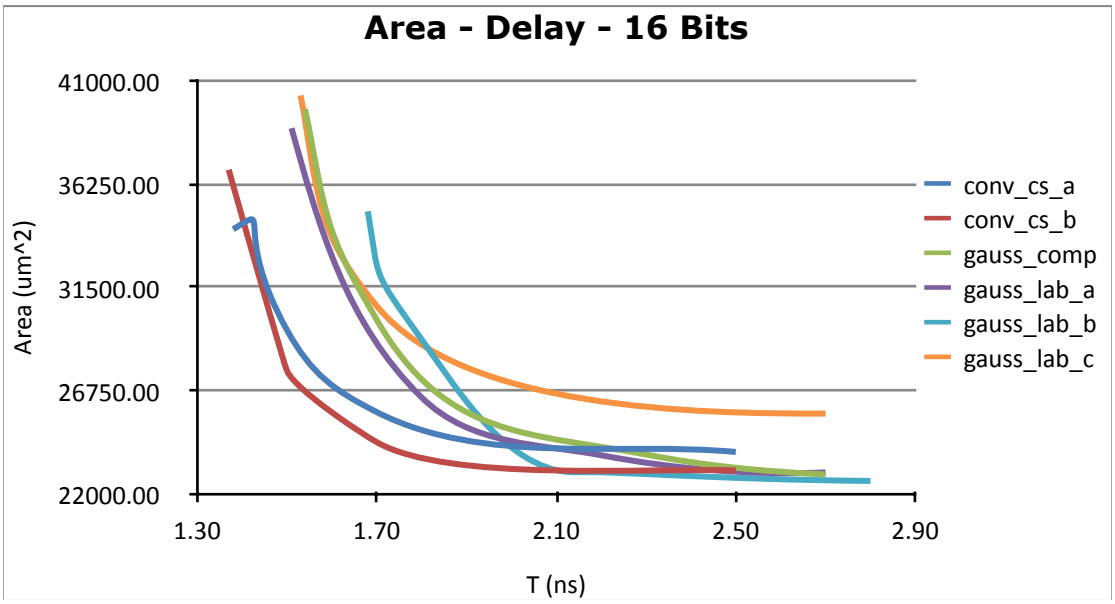


Σχήμα 4.2 Σύγκριση υλοποιήσεων κατανάλωσης-καθυστέρησης στα 08 bits

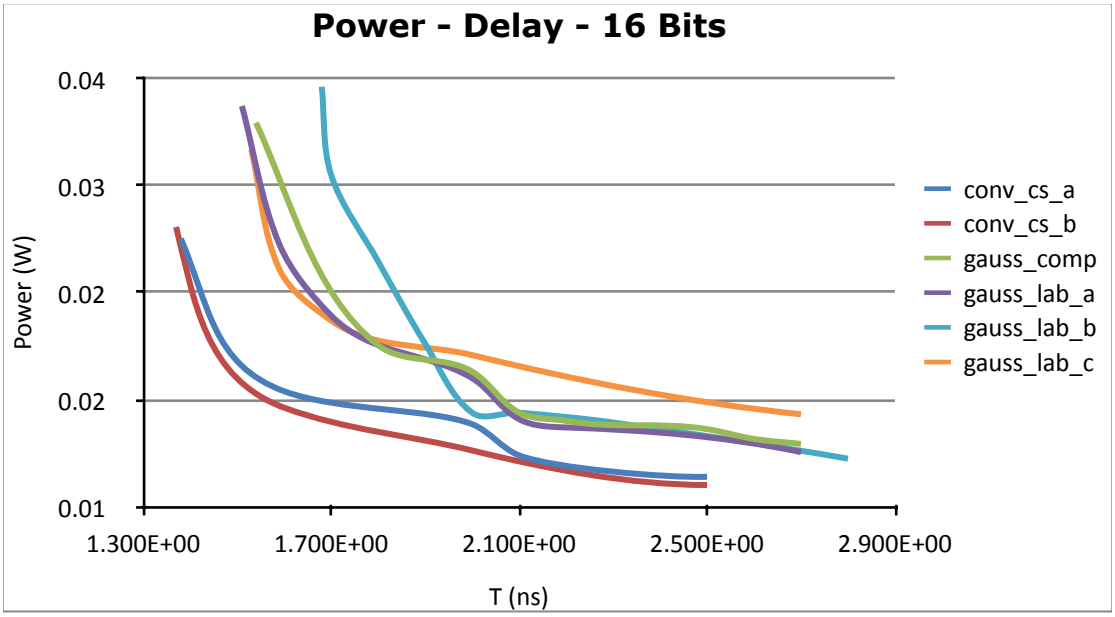
Στα παραπάνω διαγράμματα είναι εμφανής η υπεροχή των υλοποιήσεων που χρησιμοποιούν τον συμβατικό αλγόριθμο του μιγαδικού πολλαπλασιαστή, όπως ήταν αναμενόμενο, με τις υλοποιήσεις που χρησιμοποιούν τον αλγόριθμο του Gauss να έχουν παρόμοια συμπεριφορά μεταξύ τους.

2. Μήκος λέξης 16 bits

Ομοίως παρουσιάζουμε τα αποτελέσματα για 16 bits:



Σχήμα 4.3 Σύγκριση υλοποιήσεων επιφάνειας-καθυστέρησης στα 16 bits

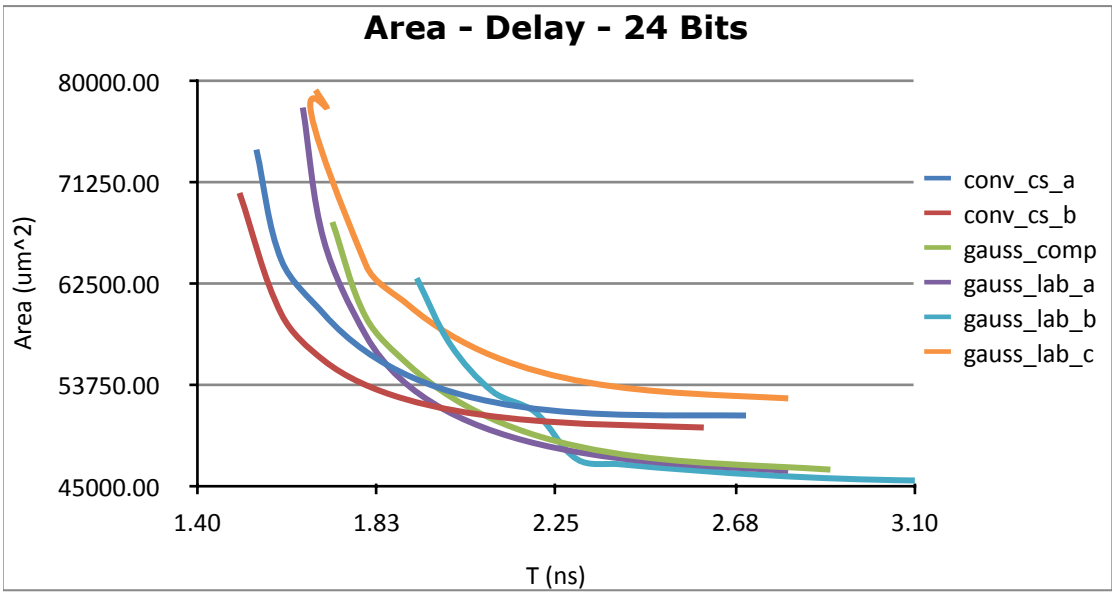


Σχήμα 4.4 Σύγκριση υλοποιήσεων κατανάλωσης-καθυστέρησης στα 16 bits

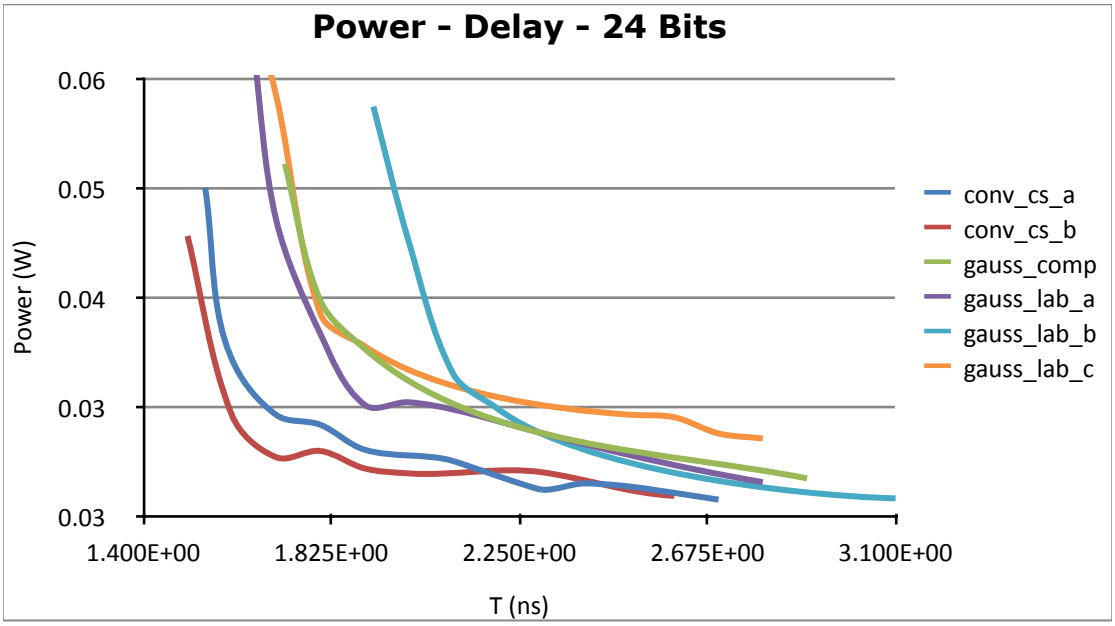
Στα διαγράμματα βλέπουμε ήδη τις υλοποιήσεις του αλγόριθμου του Gauss να πλησιάζουν σε απόδοση τις υλοποιήσεις του συμβατικού αλγόριθμου, ιδίως όσων αφορά την επιφάνεια, αλλά είναι εμφανές ότι οι υλοποιήσεις του συμβατικού αλγόριθμου υπερτερούν σε ότι αφορά την καθυστέρηση. Άξιο αναφοράς είναι το γεγονός ότι όταν βρισκόμαστε σε χαλαρότερες συνθήκες περιορισμού του ρολογιού βλέπουμε ότι η συμπεριφορά των υλοποιήσεων εμφανίζει μια τάση σύγκλισης των μετρήσεων, την οποία θα εξετάσουμε και στην συνέχεια.

3. Μήκος λέξης 24 bits

Στην συνέχεια θα εξετάσουμε τα αποτελέσματα για τα 24 bits:



Σχήμα 4.5 Σύγκριση υλοποιήσεων επιφάνειας-καθυστέρησης στα 24 bits

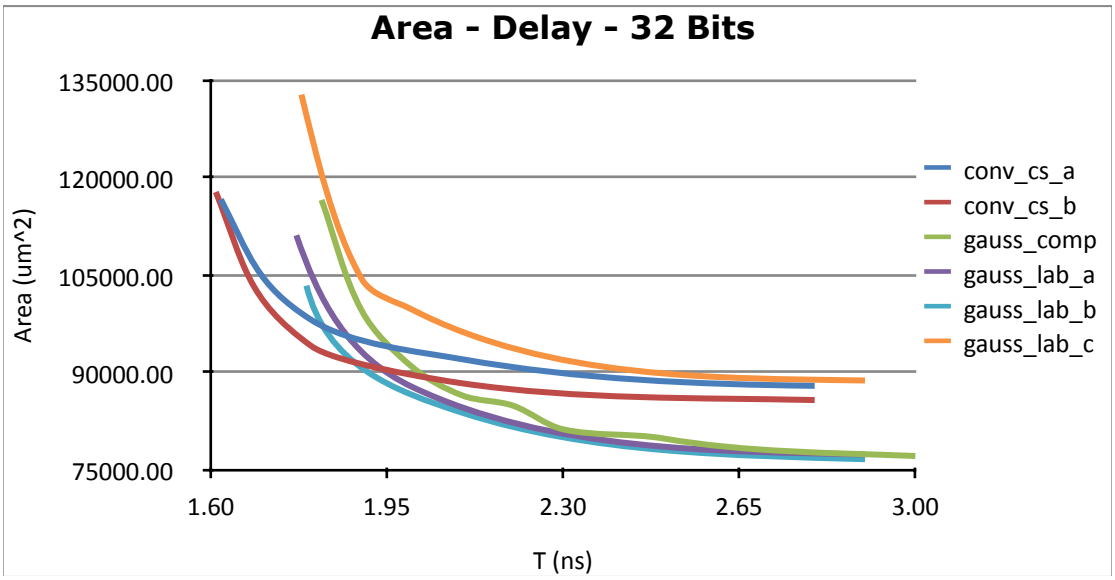


Σχήμα 4.6 Σύγκριση υλοποιήσεων κατανάλωσης-καθυστέρησης στα 24 bits

Από τα διαγράμματα βλέπουμε ότι όσων αφορά την καθυστέρηση οι υλοποιήσεις του συμβατικού αλγόριθμου εξακολουθούν να υπερτερούν. Τα αποτελέσματα δεν συμφωνούν με τους θεωρητικούς υπολογισμούς, αλλά αυτό μπορεί να εξηγηθεί από την εντολή compile_ultra. Στις high performance μετρήσεις, ο Design Compiler ευνοεί τις υλοποιήσεις του συμβατικού αλγόριθμου, όπως περιγράφηκε και παραπάνω, αλλά βλέπουμε ότι σε χαλαρότερες συνθήκες περιορισμού του ρολογιού οι υλοποιήσεις με τον αλγόριθμο του Gauss έχουν μικρότερη επιφάνεια και η κατανάλωσή τους μειώνεται σταθερά όσο ανεβαίνουμε μήκος λέξης.

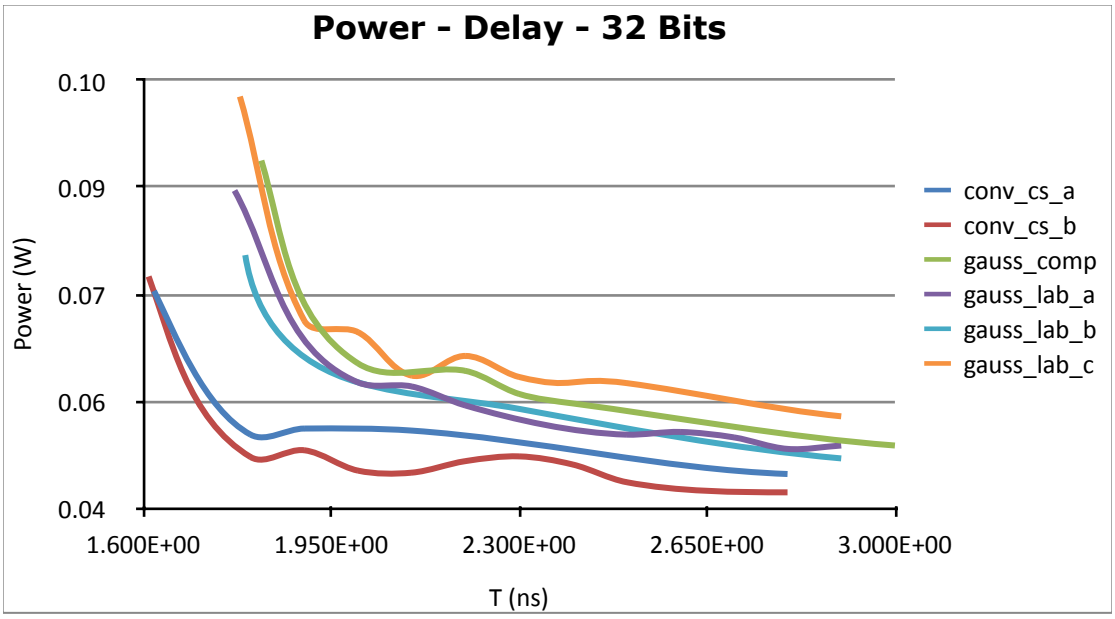
4. Μήκος λέξης 32 bits

Και στην συνέχεια για τα 32 bits έχουμε ομοίως:



Σχήμα 4.7 Σύγκριση υλοποιήσεων επιφάνειας-καθυστέρησης στα 32 bits

Εδώ είναι εμφανές ότι ενώ οι υλοποιήσεις του συμβατικού αλγόριθμου έχουν καλύτερα αποτελέσματα για την καθυστέρηση, βλέπουμε ότι όσων αφορά την επιφάνεια τα Gauss Optimized Modified Booth Recoder (A), (B) και Gauss Recoder Wei, Du, Chen υπερέχουν πλέον σταθερά από τις υπόλοιπες υλοποιήσεις.

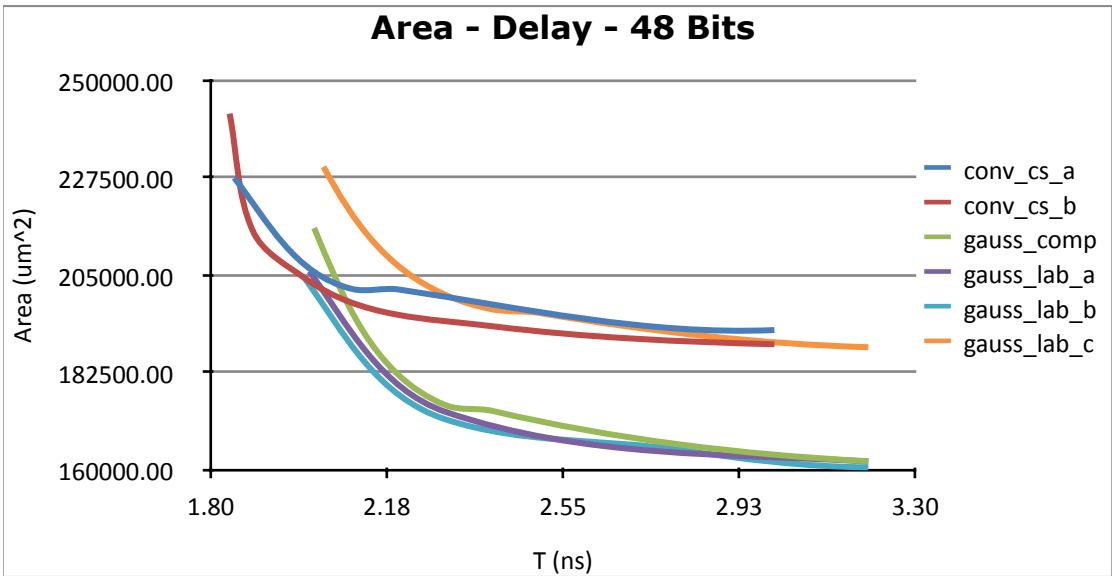


Σχήμα 4.8 Σύγκριση υλοποιήσεων κατανάλωσης-καθυστέρησης στα 32 bits

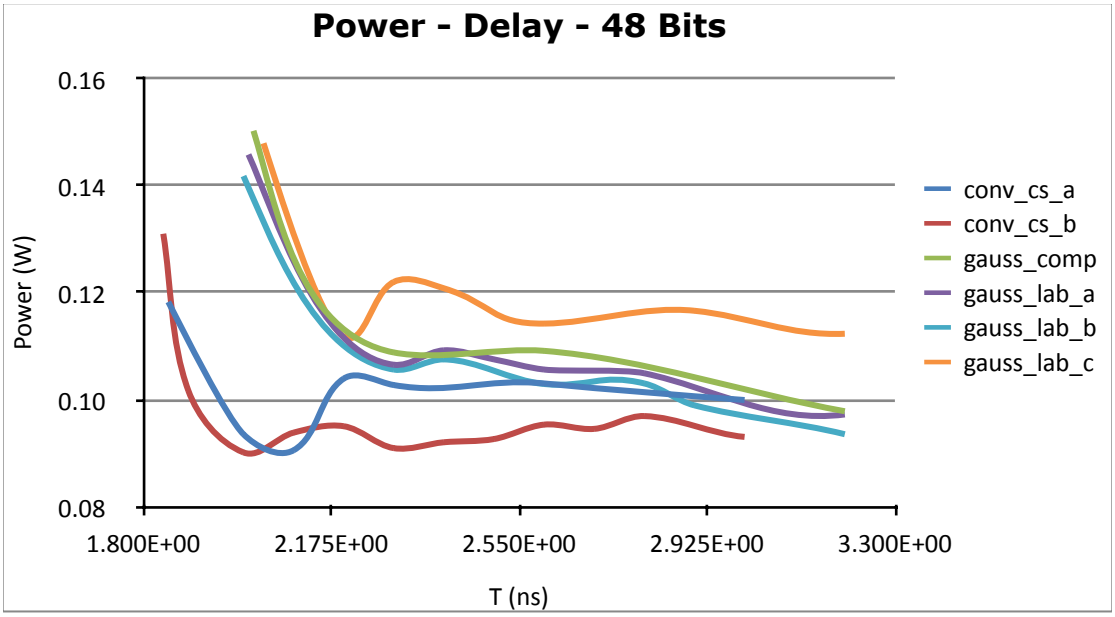
Οι υλοποιήσεις του συμβατικού αλγόριθμου είναι ακόμα καλύτερες στον τομέα της κατανάλωσης με την υλοποίηση Conventional Carry-Save (B) να έχει την καλύτερη απόδοση σε κατανάλωση, επιφάνεια και καθυστέρηση ενώ η υλοποίηση Gauss Optimized Modified Booth Recoder (C) να έχει την χειρότερη.

5. Μήκος λέξης 48 bits

Στο μήκος λέξης των 48 bits έχουμε τα παρακάτω αποτελέσματα:



Σχήμα 4.9 Σύγκριση υλοποιήσεων επιφάνειας-καθυστέρησης στα 48 bits

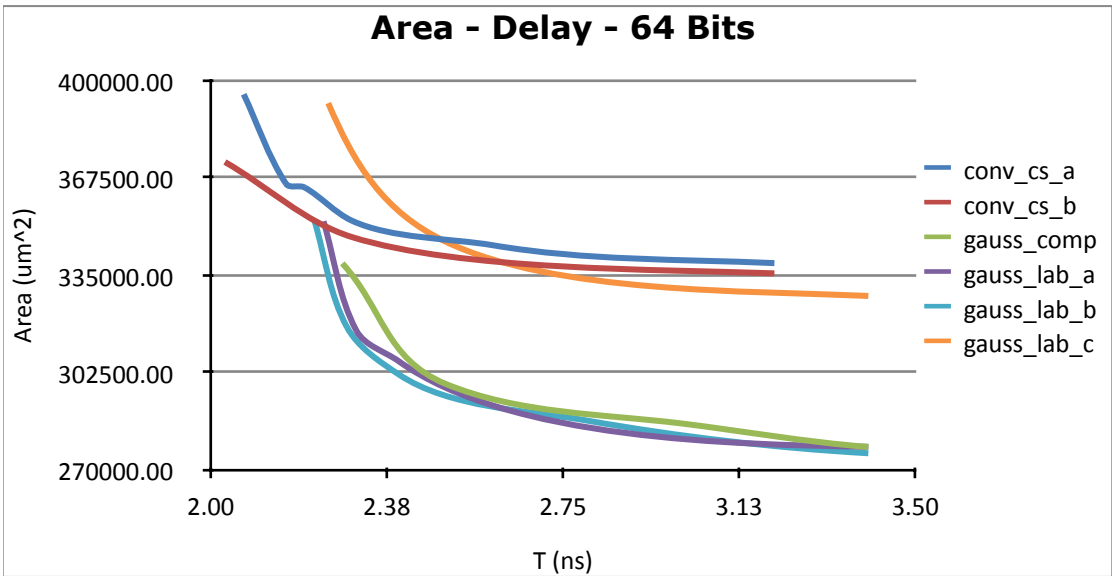


Σχήμα 4.10 Σύγκριση υλοποιήσεων κατανάλωσης-καθυστέρησης στα 48 bits

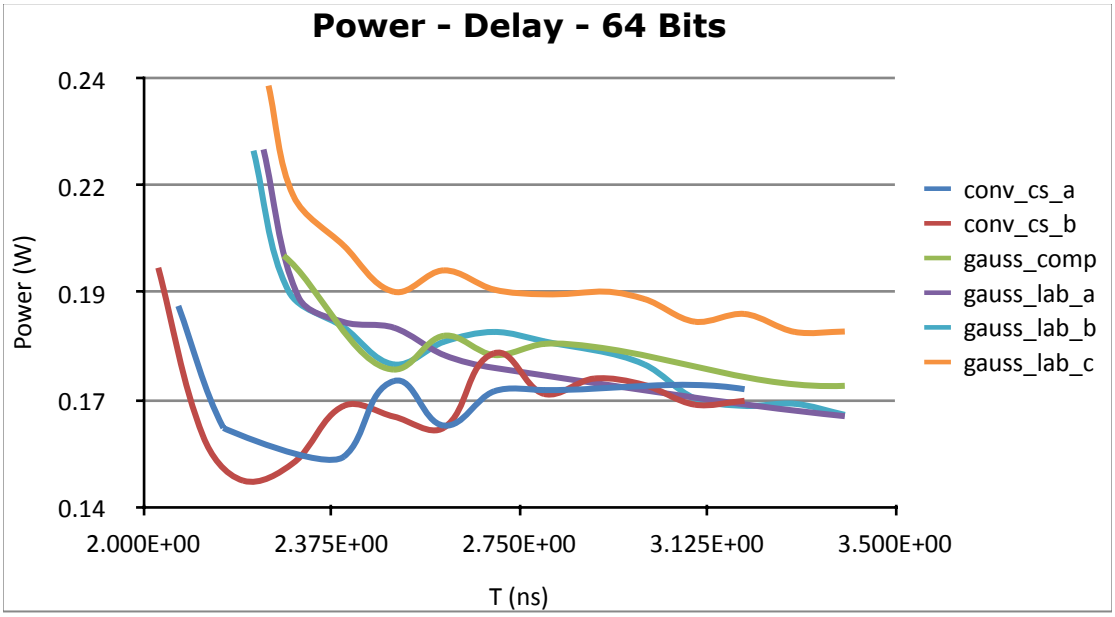
Σε αυτό το σημείο βλέπουμε ότι σε αυτό το μήκος λέξης, τα αποτελέσματα μας δίνουν μια σαφή εικόνα της συμπεριφοράς των υλοποιήσεων. Οι υλοποιήσεις του συμβατικού αλγόριθμου έχουν το πλεονέκτημα στην καθυστέρηση, όπου προηγούνται για περίπου 0.15ns και η υλοποίηση (B) να είναι καλύτερη και στην κατανάλωση ανεξαρτήτως περιορισμών του ρολογιού. Από την άλλη βλέπουμε ότι οι υλοποιήσεις του αλγόριθμου του Gauss έχουν σαφές προβάδισμα σε ότι αφορά την επιφάνεια. Όλες οι υλοποίησεις, εκτός της Gauss Optimized Modified Booth Recoder (C) έχουν μικρότερη επιφάνεια, με την Gauss Optimized Modified Booth Recoder (B) να έχει την καλύτερη απόδοση σε επιφάνεια και κατανάλωση.

6. Μήκος λέξης 64 bits

Τέλος στα 64 bits θα δούμε πως διαμορφώνεται η συμπεριφορά των υλοποιήσεων:



Σχήμα 4.11 Σύγκριση υλοποιήσεων επιφάνειας-καθυστέρησης στα 64 bits



Σχήμα 4.12 Σύγκριση υλοποιήσεων κατανάλωσης-καθυστέρησης στα 64 bits

Στα 64 bits μπορούμε να καταλήξουμε σε ότι αφορά τα πλεονεκτήματα και τα μειονεκτήματα της κάθε υλοποίησης και κυρίως στην σύγκριση μεταξύ του συμβατικού αλγόριθμου και του αλγόριθμου του Gauss. Οι υλοποιήσεις που χρησιμοποιούν τον συμβατικό αλγόριθμο έχουν την μικρότερη καθυστέρηση και υπερτερούν στην κατανάλωση ιδίως σε συνθήκες ισχυρών περιορισμών στο ρολόι. Οι υλοποιήσεις που χρησιμοποιούν το αλγόριθμο του Gauss δείχνουν να έχουν την μικρότερη επιφάνεια και σε συνθήκες χαλαρών περιορισμών ρολογιού έχουν ίδια κατανάλωση με τις υλοποιήσεις του συμβατικού αλγόριθμου.

Από την πορεία των μετρήσεων και την συμπεριφορά των υλοποιήσεων όσο ανεβαίνουμε σε μήκος λέξης βλέπουμε ότι ο αλγόριθμος του Gauss προσφέρει την μικρότερη δυνατή επιφάνεια και είναι αποδοτικό και σε κατανάλωση.

Στην συνέχεια, θα συγκρίνουμε τις υλοποιήσεις μεταξύ τους για να εξετάσουμε την λειτουργία και την επίδραση του δενδρικού συμπιεστή Wallace πάνω στο κύκλωμα. Από τις υλοποιήσεις του συμβατικού αλγόριθμου μπορούμε να δούμε ότι όταν μεταφέραμε τα διανύσματα σε δύο δενδρικούς συμπιεστές Wallace ώστε να έχουμε δύο μεγάλα δένδρα αντί για τέσσερα επιμέρους, κερδίσαμε σε επιφάνεια, κατανάλωση και καθυστέρηση, αλλά οι υλοποιήσεις του αλγόριθμου του Gauss μας έδειξαν ότι η λογική του να δημιουργούμε όσο το δυνατόν μεγαλύτερα δένδρα δεν είναι η βέλτιστη. Στην περίπτωση του αλγόριθμου του Gauss έχουμε τον κοινό όρο m_0. Με την λογική αυτή, το τρίτο σχήμα που υλοποιήσαμε με την προτεινόμενο επανακωδικοποιητή Πρόσθεσης-Πολλαπλασιασμού, θα πρέπει να είχε την βέλτιστη συμπεριφορά ανάμεσα στις τρεις αυτές παραλλαγές του ίδιου σχήματος. Είδαμε όμως ότι αυτή ήταν η χειρότερη υλοποίηση, με την δεύτερη παραλλαγή να είναι η καλύτερη, όπου είχαμε δύο μεγάλα δένδρα Wallace για να παράγουν το πραγματικό και το φανταστικό μέρος σε μορφή carry-save, από τους τρεις όρους m_0, m_1 και m_2, αλλά για τον κοινό όρο m_0 είχαμε ένα ξεχωριστό δένδρο το όποιο παραγόταν σε carry-save μορφή και στην συνέχεια το οδηγούσαμε στα δύο κεντρικά δένδρα Wallace, με τα m_1 και m_2 αντίστοιχα, τα οποία ήταν ακόμα στην μορφή διανύσματος που περιέχει όλα τα μερικά γινόμενα, με τους διορθωτικούς όρους και τις απαραίτητες επεκτάσεις προσήμου. Στο σημείο αυτό είναι σημαντικό να λάβουμε υπ'όψην τον Verilog κώδικα της υλοποίησης Gauss Optimized Modified Booth Recoder (B). Στον Verilog κώδικα ο όρος m_0 οδηγήθηκε στα δύο δένδρα με τρόπο που ο compiler θα τον προσθέτει στα χαμηλότερα "φύλλα" των δένδρων. Με τον τρόπο αυτό μειώνουμε το κρίσιμο μονοπάτι της υλοποίησης αυτής, όπως αυτό υπολογίστηκε στους θεωρητικούς υπολογισμούς, αφού οι υπολογισμοί του δένδρου ξεκινάνε πριν ολοκληρωθεί ο υπολογισμός του όρου m_0 και ουσιαστικά παράγεται παράλληλα με τους πρώτους υπολογισμούς των δύο κεντρικών δέδρων Wallace. Έτσι βλέπουμε ότι λόγω της κοινής χρήσης του όρου m_0, με τον υπολογισμό του σε carry-save μορφή, κερδίζουμε σε επιφάνεια, κατανάλωση και καθυστέρηση.

Τέλος η υλοποίηση Gauss Recoder Wei, Du, Chen βλέπουμε ότι είναι οριακά χειρότερη από την Gauss Optimized Modified Booth Recoder (A). Συγκρίνουμε αυτές τις δύο υλοποιήσεις γιατί χρησιμοποιούν τους ίδιους δενδρικούς συμπιεστές Wallace και διαφέρουν μόνο στη μονάδα επανακωδικοποίησης Πρόσθεσης-Πολλαπλασιασμού. Στις δύο αυτές υλοποιήσεις βλέπουμε ότι έχουν παρόμοια συμπεριφορά στην επιφάνεια την οποία καταλαμβάνουν αλλά η Gauss Optimized Modified Booth Recoder (A) μας δίνει καλύτερη κατανάλωση, ιδίως όσο ανεβαίνουμε σε μήκος λέξης και είμαστε σε high performance συχνότητες.

4.3. Υλοποίηση σε FPGA

Στο παρόν κεφάλαιο μεταφέρουμε τις παραπάνω τοπολογίες σε ένα FPGA για να δούμε πως θα συμπεριφερθούν. Οι μετρήσεις έγιναν για μήκος λέξης εισόδου 8, 16, 24, 32, 48 και 64 bits, έτσι χρειαζόμαστε ένα FPGA με πολλές εισόδους/εξόδους δεδομένου ότι στα 64 bits θα πρέπει να έχουμε 64*4 = 256 εισόδους 2*128 = 256 εξόδους και δύο εισόδους για ρολόι και επαναφορά. Έτσι βλέπουμε ότι θα χρειαστούμε 514 εισόδους/εξόδους στο μεγαλύτερο μήκος λέξης, αριθμός μεγάλος για ένα FPGA. Για τον λόγο αυτό επιλέξαμε την οικογένεια Virtex 7 της Xilinx που έχουν 1200 εισόδους/εξόδους, μιας και το αμέσως μικρότερο FPGA έχει 500. Με αυτό τον τρόπο θα δούμε τι επιπτώσεις θα έχουμε από την εφαρμογή του κυκλώματός μας σε ένα FPGA, δεδομένου ότι τα FPGA υλοποιούν το κύκλωμα με την χρήση λογικών blocks τα οποία προγραμματίζονται για την εφαρμογή του κώδικα, έτσι τα εργαλεία που χρησιμοποιούμε κάνουν διαφορετική ελαχιστοποίηση από αυτήν που είχαμε στην ASIC υλοποίηση.

Οι μετρήσεις λήφθηκαν με χρήση του ISE 14.7 της Xilinx σε τεχνολογία 28nm, με τυπικές ρυθμίσεις θερμοκρασίας. Στις μετρήσεις που λαμβάνουμε το ISE παράγει μια αναφορά για την καθυστέρηση, όπου υπολογίζει την μικρότερη δυνατή περίοδο ρολογιού. Για αυτήν την τιμή ρολογιού, και με χρήση του εργαλείου XPower Analyzer της Xilinx υπολογίζουμε την κατανάλωση του κυκλώματος. Τέλος για τον αντίστοιχο υπολογισμό της επιφάνειας που καταλαμβάνει το κάθε κύκλωμα θα πρέπει να εξηγήσουμε την αρχιτεκτονική των FPGA της οικογένειας Virtex 7. Τυπικά σε ένα FPGA τα κυκλώματα υλοποιούνται με την χρήση λογικών blocks. Αυτά τα blocks συγκεντρώνονται σε Slices τα οποία περιέχουν LUTs (lookup tables) τα οποία προγραμματίζονται από ένα αρχείο το οποίο παράγεται από το εργαλείο της εκάστοτε εταιρείας παραγωγής του FPGA. Τα LUTs είναι πίνακες αναζήτησεις οι οποίοι διατρέχονται μέσω ενός πολυπλέκτη, οι είσοδοι του οποίου είναι σταθερές, ορισμένες από το αρχείο που παράγει το εργαλείο μετάφρασης του κώδικα μας. Έτσι ένα n-bit LUT μπορεί να κωδικοποιήσει μια Boolean συνάρτηση των n bits και ανάλογα με την λειτουργία που θέλουμε μπορεί να χρησιμοποιηθεί για διάφορες χρήσεις, από την υλοποίηση λογικών συναρτήσεων, μέχρι και την υλοποίηση μιας μνήμης RAM. Στην περίπτωση της οικογένειας Virtex 7, η συσκευή που επιλέξαμε περιέχει 326400 slices όπου κάθε slice περιέχει 4 LUTs και 8 flip-flops. Κατ'αναλογία του υπολογισμού επιφανείας στην ASIC υλοποίηση, στα FPGA θα χρησιμοποιήσουμε τα στατιστικά χρησιμοποίησης των slices, LUTs και flip-flops της συσκευής.

Με την ίδια λογική που ακολουθήσαμε και παραπάνω θα χωρίσουμε τις μετρήσεις ανάλογα με το μήκος λέξης εισόδου, για να συγκρίνουμε τα αποτελέσματα.

1. Μήκος λέξης 08 bits

Εδώ βλέπουμε συγκεντρωμένα τα αποτελέσματα για τα 08 bits:

Design ->	conv_cs_a	conv_cs_b	gauss_comp	gauss_lab_a	gauss_lab_b	gauss_lab_c
Power (mW)	235.71	228.44	238.03	236.77	236.79	237.86
T (ns)	6.723	7.703	7.756	7.631	7.160	7.074
Number of Slice Registers	72	72	91	100	77	84
Number of Slice LUTs	518	481	732	584	571	602
Number of occupied Slices	262	254	377	340	301	316
Number of LUT Flip Flop	525	485	750	614	582	616
Number of bonded IOBs	66					

Πίνακας 4.6 Σύγκριση υλοποιήσεων 08 bits

Στον παραπάνω πίνακα βλέπουμε την κατανάλωση, την καθυστέρηση και το πόσα slices, LUTs, καταχωρητές και flip-flops χρησιμοποιήθηκαν. Με μια πρώτη ματιά βλέπουμε ότι η υλοποίηση με την μικρότερη καθυστέρηση είναι η υλοποίηση Conventional Carry-Save (B) (conv_cs_b) αλλά με εμφανώς μικρό προβάδισμα. Ομοίως στην κατανάλωση προηγείται η υλοποίηση Conventional Carry-Save (A) (conv_cs_a) και πάλι με μικρή διαφορά.

Σε ότι αφορά την επιφάνεια που καταλαμβάνει ένα κύκλωμα, πρέπει να ορίσουμε πως θα μετράμε την επιφάνεια, δεδομένου ότι τα FPGAs έχουν συγκριμένη επιφάνεια, εμείς σαν επιφάνεια θα ορίσουμε τον χώρο που απασχολεί η υλοποίησή μας. Δηλαδή στις μετρήσεις μας, στο Design Summary, το ISE μας ενημερώνει για το hardware το οποίο απασχολεί η εκάστοτε υλοποίηση. Έτσι αναφέρεται στο πλήθος των slices, στο πλήθος των LUTs, τα οποία είναι μέρος των slices όπως είδαμε παραπάνω, και στο αριθμό των flip-flops και καταχωρητών, που είναι και αυτά μέρος των slices. Για να επιλέξουμε πως θα προχωρήσουμε στην σύγκριση των υλοποιήσεων στον τομέα της επιφάνειας ανατρέξαμε στην εργασία των Ian Kuon και Jonathan Rose [5] όπου περιγράφεται ένας τρόπος μέτρησης. Περιγράφεται ότι για να μετρήσουμε το ανάλογο της επιφάνειας που μετρήσαμε στην ASIC υλοποίηση, λαμβάνουμε υπ'όψην τα λογικά blocks που απασχολούνται καθώς και τα flip-flops και οι καταχωρητές. Δεν λαμβάνονται υπ'όψην οι είσοδοι/έξοδοι που χρησιμοποιούνται, δεδομένου ότι τα FPGAs χρησιμοποιούν διαφορετικό τρόπο υλοποίησης από αυτόν των εισόδων/εξόδων σε ASIC. Τέλος, δεν θα λάβουμε υπ'όψην τα slices που χρησιμοποιούνται, αλλά μόνο τα LUTs μιας και το εργαλείο εφαρμογής του κώδικα[8] στο FPGA κάνει το mapping και routing χωρίς να κάνει πλήρη χρήση όλων των LUTs σε κάθε slice.

2. *Μήκος λέξης 16 bits*

16 Bits						
Design ->	**conv_cs_a**	**conv_cs_b**	**gauss_comp**	**gauss_lab_a**	**gauss_lab_b**	**gauss_lab_c**
Power (mW)	276.91	276.30	282.25	286.46	287.89	300.30
T (ns)	7.484	6.732	8.477	8.108	8.482	8.392
Number of Slice Registers	186	226	202	214	172	207
Number of Slice LUTs	1802	1813	2019	1940	1776	2283
Number of occupied Slices	777	743	847	780	796	840
Number of LUT Flip Flop pairs	1835	1863	2049	1983	1805	2315
Number of bonded IOBs	130					

Πίνακας 4.7 Σύγκριση υλοποιήσεων 16 bits

3. *Μήκος λέξης 24 bits*

24 Bits						
Design ->	**conv_cs_a**	**conv_cs_b**	**gauss_comp**	**gauss_lab_a**	**gauss_lab_b**	**gauss_lab_c**
Power (mW)	387.57	368.75	384.07	399.49	386.53	415.22
T (ns)	8.936	8.252	10.101	9.516	10.438	9.605
Number of Slice Registers	321	280	390	321	270	302
Number of Slice LUTs	3966	3699	4013	3992	3450	4463
Number of occupied Slices	1695	1369	1657	1631	1372	1498
Number of LUT Flip Flop pairs used	4028	3733	4087	4046	3481	4492
Number of bonded IOBs	194					

Πίνακας 4.8 Σύγκριση υλοποιήσεων 24 bits

4. Μήκος λέξης 32 bits

32 Bits						
Design ->	**conv_cs_a**	**conv_cs_b**	**gauss_comp**	**gauss_lab_a**	**gauss_lab_b**	**gauss_lab_c**
Power (mW)	469.29	471.36	460.03	535.25	485.44	572.81
T (ns)	10.641	10.302	10.489	11.230	11.305	11.330
Number of Slice Registers	522	446	396	490	365	552
Number of Slice LUTs	6729	6537	6482	9002	5904	9173
Number of occupied Slices	2728	3057	2342	3307	2254	3677
Number of LUT Flip Flop pairs used	6863	6631	6499	9110	5943	9259
Number of bonded IOBs	258					

Πίνακας 4.9 Σύγκριση υλοποιήσεων 32 bits

5. Μήκος λέξης 48 bits

48 Bits						
Design ->	**conv_cs_a**	**conv_cs_b**	**gauss_comp**	**gauss_lab_a**	**gauss_lab_b**	**gauss_lab_c**
Power (mW)	747.34	727.60	735.78	908.85	741.47	1003.79
T (ns)	11.941	10.428	11.747	11.979	11.995	11.997
Number of Slice Registers	857	839	839	883	513	1188
Number of Slice LUTs	14596	14074	13985	19368	11536	21430
Number of occupied Slices	5632	5244	5034	7374	4482	7472
Number of LUT Flip Flop pairs used	14801	14277	14115	19550	11580	21680
Number of bonded IOBs	386					

Πίνακας 4.10 Σύγκριση υλοποιήσεων 48 bits

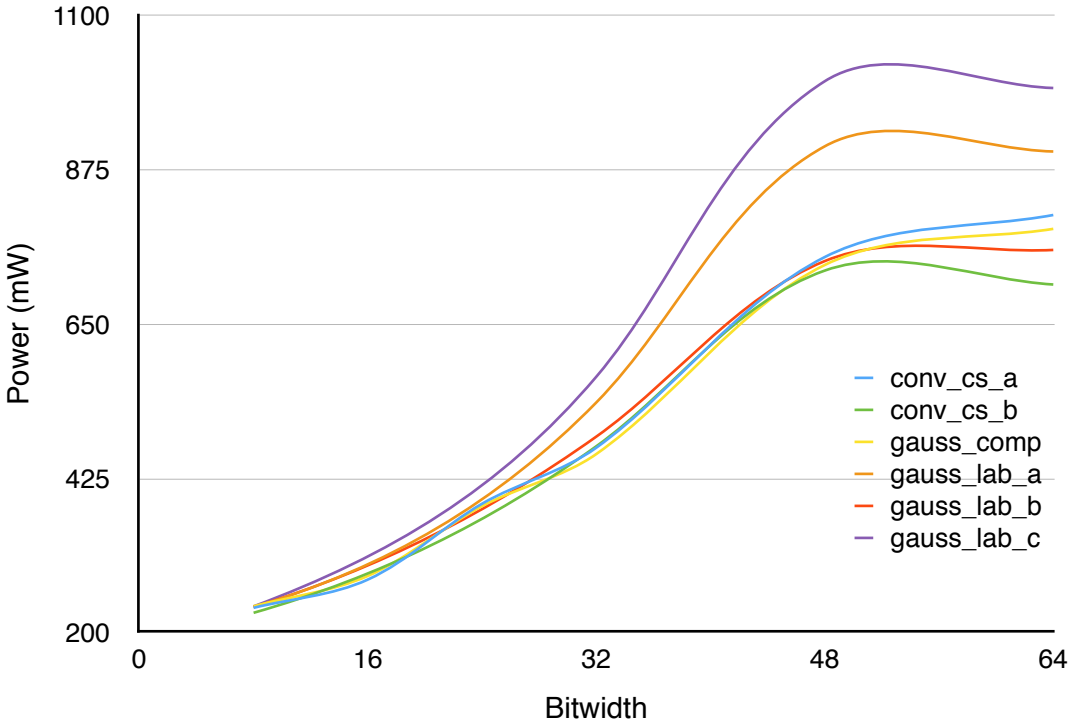
6. Μήκος λέξης 64 bits

64 Bits						
Design ->	**conv_cs_a**	**conv_cs_b**	**gauss_comp**	**gauss_lab_a**	**gauss_lab_b**	**gauss_lab_c**
Power (mW)	808.66	707.26	788.43	901.27	757.68	993.87
T (ns)	19.963	11.967	15.999	19.446	15.435	16.755
Number of Slice Registers	719	952	1103	1022	750	1380
Number of Slice LUTs	26482	23851	31130	34863	20406	37044
Number of occupied Slices	13349	7417	9080	13481	7042	11793
Number of LUT Flip Flop pairs used	26607	23928	31204	35068	20452	37168
Number of bonded IOBs	386					

Πίνακας 4.11 Σύγκριση υλοποιήσεων 64 bits

Για να έχουμε μία καλύτερη σύγκριση των παραπάνω αποτελεσμάτων θα συγκεντρώσουμε όλα τα αποτελέσματα σε διαγράμματα κατανάλωσης, καθυστέρησης και επιφάνειας συναρτήσει του μήκους λέξης εισόδου για να έχουμε καλύτερη εποπτεία των αποτελεσμάτων.

4.3.1. Κατανάλωση συναρτήσει μήκους λέξης εισόδου

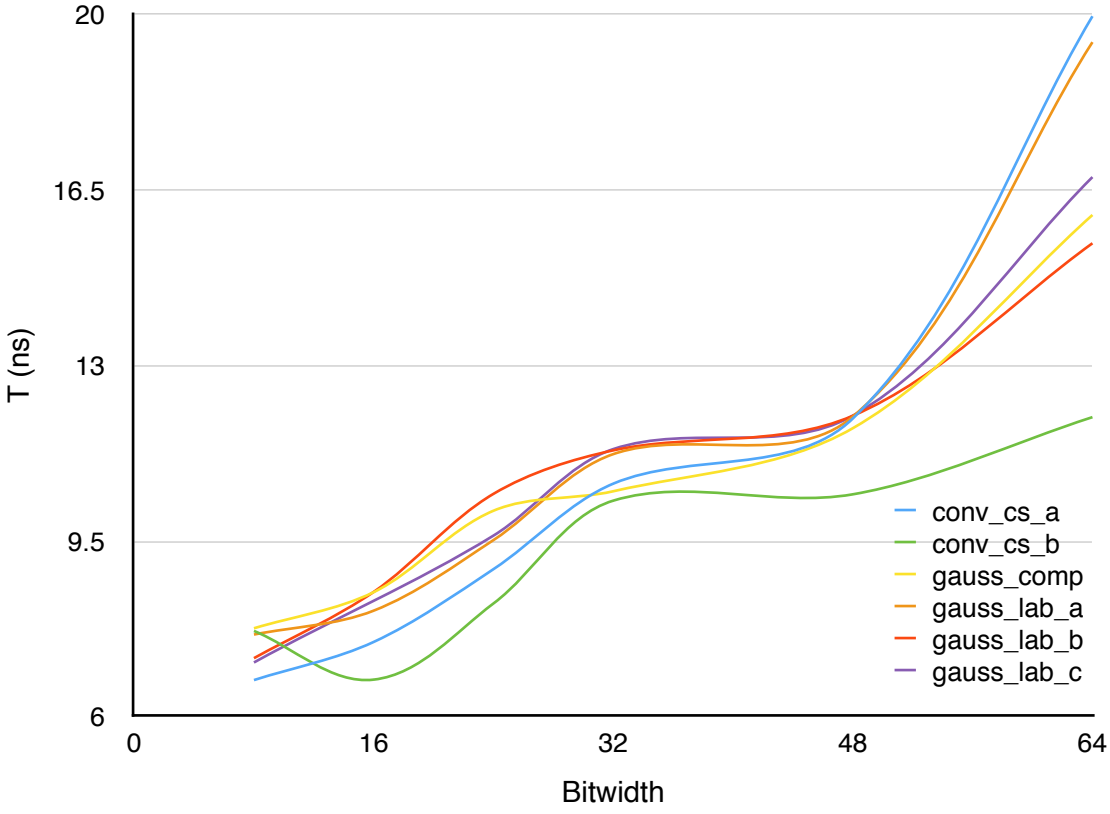


Σχήμα 4.13 Συγκεντρωτικό διάγραμμα κατανάλωσης-μήκους λέξης

Από το παραπάνω διάγραμμα βλέπουμε ότι για τα μικρότερα μήκη λέξης έχουμε μικρές διαφορές και μια παρόμοια συμπεριφορά των υλοποιήσεων. Από τα 24 bits και μετά βλέπουμε ότι οι υλοποιήσεις Gauss Optimized Modified Booth Recoder (A) και (C) έχουν την μεγαλύτερη κατανάλωση ενώ οι υπόλοιπες έχουν μια παρόμοια συμπεριφορά. Στα 64 bits έχουμε την υλοποίηση Conventional Carry-Save (B) να είναι και πάλι αυτή με την χαμηλότερη κατανάλωση με τις Gauss Optimized Modified Booth Recoder (B), Gauss Recoder Wei, Du, Chen, και Conventional Carry-Save (A) να ακολουθούν με αυτήν την σειρά. Από τις θεωρητικές μετρήσεις και εν συνεχεία από τις high-performance μετρήσεις στις υλοποιήσεις σε ASIC είδαμε πως η υλοποίηση Gauss Optimized Modified Booth Recoder (C) θα είναι η χειρότερη κάτι που εξηγείται, όπως και παραπάνω, από τον κοινό όρο m_0, ο οποίος όταν δεν μετατραπεί σε carry-save μορφή επιβαρύνει πολύ το κύκλωμα, με τους δύο κεντρικούς δενδρικούς συμπιεστές Wallace. Αντίστοιχα η υλοποίηση Gauss Optimized Modified Booth Recoder (A) δεν εκμεταλλεύεται πλήρως το Wallace δένδρο με τους επιμέρους υπολογισμούς των m_0, m_1 και m_2 σε carry-save μορφή. Το ίδιο ισχύει και για την υλοποίηση Conventional Carry-Save (A) η οποία μπορεί η απόδοσή της να μην είναι τόσο χαμηλή όσο οι δύο προηγούμενες, αλλά εξακολουθεί να είναι η χειρότερη από την Conventional Carry-Save (B).

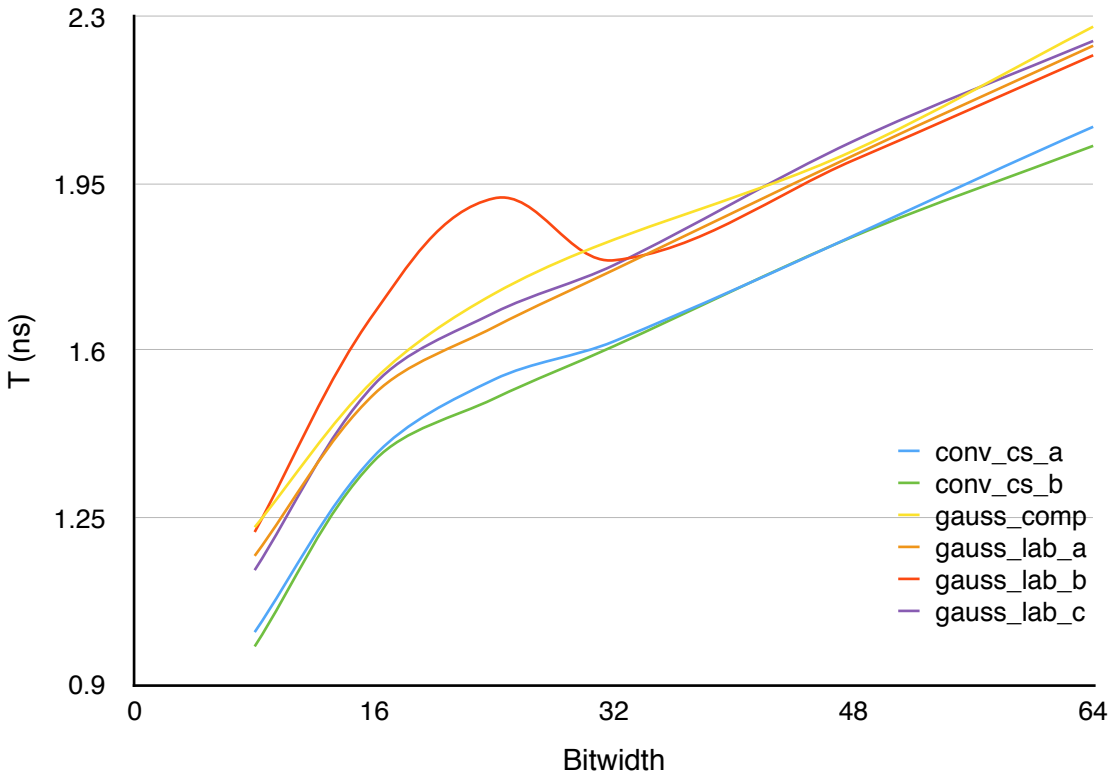
Σε γενικές γραμμές τα αποτελέσματα που προκύπτουν από την μεταφορά των υλοποιήσεων από ASIC σε FPGA συμφωνούν σε γενικές γραμμές σε θέματα κατανάλωσης με την διαφορά ότι στα FPGA έχουμε μια σαφέστερη διαφοροποίηση όσο ανεβαίνουμε σε μήκος λέξης και αυτό οφείλεται στην ευελιξία που έχουν τα εργαλεία των ASIC υλοποιήσεων να καταργούν την ιεραρχία των κυκλωμάτων κάτι που κάνει τις διαφορετικές παραλλαγές των υλοποιήσεων που χρησιμοποιούν των αλγόριθμο του Gauss και αυτές που χρησιμοποιούν τον συμβατικό αλγόριθμο να έχουν μια πιο κοινή συμπεριφορά.

4.3.2. Καθυστέρηση συναρτήσει μήκους λέξης εισόδου



Σχήμα 4.14 Συγκεντρωτικό διάγραμμα καθυστέρησης-μήκους λέξης

Στο παραπάνω διάγραμμα βλέπουμε την σύγκριση της καθυστέρησης των υλοποιήσεων. Η υλοποίηση Conventional Carry-Save (B) εξακολουθεί να είναι η αποδοτικότερη. Οι υπόλοιπες υλοποιήσεις έχουν παρόμοια συμπεριφορά, με τις υλοποιήσεις του συμβατικού αλγόριθμου του μιγαικού πολλαπλασιαστή να έχουν καλύτερη απόδοση στα μηκρότερα μήκη λέξης, ενώ στα 48 bits βλέπουμε μια σύγκλιση όλων των υλοποιήσεων, εκτός της Conventional Carry-Save (B), και στα 64 bits βλέπουμε τις υλοποιήσεις που κάνουν χρήση του αλγόριθμου του Gauss να αποκτούν πλεονέκτημα επί της υλοποίησης Conventional Carry-Save (A). Για να έχουμε μια καλύτερη σύγκριση με τις υλοποιήσεις σε ASIC σχεδιάζουμε το αντίστοιχο διάγραμμα σε ASIC:



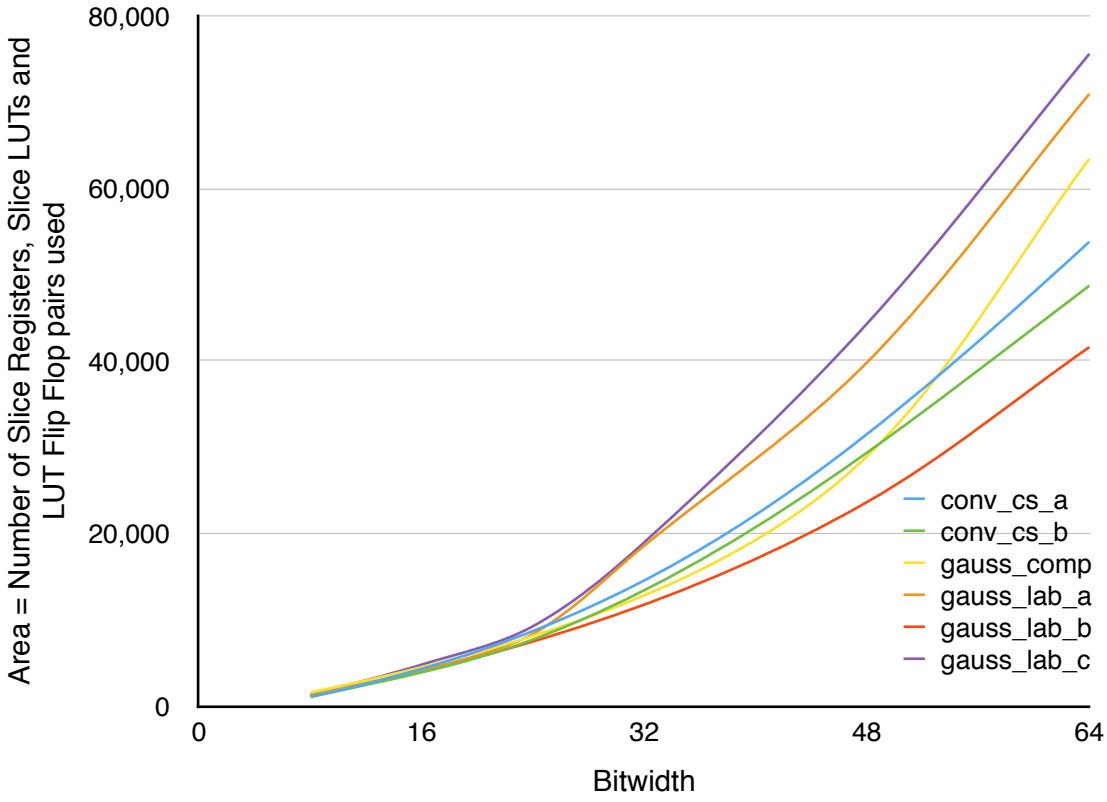
Σχήμα 4.15 Συγκεντρωτικό διάγραμμα καθυστέρησης-μήκους λέξης σε ASIC

Στην ASIC υλοποίηση του μιγαδικού πολλαπλασιαστή βλέπουμε ότι ο συμβατικός αλγορίθμος δίνει μικρότερη καθυστέρηση από τον αλγόριθμο του Gauss, σε αναντιστοιχία με τον θεωρητικό υπολογισμό που πραγματοποιήσαμε αρχικά, όπου η υλοποίηση Gauss Optimized Modified Booth Recoder (A) εμφανιζόταν, έστω κι οριακά, η βέλτιστη υλοποίηση. Όμως και οι υλοποιήσεις που κάνουν χρήση του αλγόριθμου του Gauss αναμέναμε να δώσουν διαφορετικά αποτελέσματα και στην μεταξύ τους σύγκριση. Στο παραπάνω διάγραμμα βλέπουμε την υλοποίηση Gauss Optimized Modified Booth Recoder (B) να ακουλουθεί τις θεωρητικές μετρήσεις μέχρι το μήκος λέξης των 24 bits, όπου στα 32, 48 και 64 bits αλλάζει και γίνεται η καλύτερη. Αυτό εξηγείται από την εντολή compile_ultra που επιτρέπει στον Design Compiler να κάνεις όλες τις απαραίτητες ελαχιστοποιήσεις στο κρίσιμο μονοπάτι του κυκλώματος για να έχουμε την μικρότερη δυνατή καθυστέρηση. Και με τον ίδιο τρόπο επηρεάζονται και οι υλοποιήσεις που χρησιμοποιούν τον συμβατικό αλγόριθμο. Αν κάνουμε μια παραδοχή ότι και στους δύο αλγόριθμους οι υπολογισμοί γίνονται παράλληλα, με μια μακροσκοπική ανάλυση, βλέπουμε ότι ο συμβατικός αλγόριθμος χρειάζεται έναν πολλαπλασιασμό και μια πρόσθεση για να δώσει απότελεσμα, σε αντίθεση με τον αλγόριθμο του Gauss όπου υπολογίζουμε ένα άθροισμα, έναν πολλαπλασιασμό και μια πρόσθεση. Με τις υλοποιήσεις του αλγόριθμου του Gauss δημιουργήσαμε δύο διαφορετικές μονάδες που συγχωνεύουν την πράξη $A*(X+Y)$ στο κόστος ενός πολλαπλασιασμού. Βλέπουμε όμως ότι οι ελαχιστοποιήσεις που κάνει ο Design Compiler στο κρίσιμο μονοπάτι των υλοποιήσεων του συμβατικού αλγόριθμου καθιστούν αυτές τις υλοποίησεις γρηγορότερες.

4.3.3. Επιφάνεια συναρτήσει μήκους λέξης εισόδου

Στην προηγούμενη παράγραφο είδαμε τα πλεονεκτήματα των υλοποίησεων του συμβατικού αλγόριθμου επί των υλοποιήσεων του αλγόριθμου του Gauss στην καθυστέρηση των κυκλωμάτων και εδώ θα δούμε την συμπεριφορά τους σε ότι αφορά την επιφάνεια που καταλαμβάνουν. Όπως περιγράψαμε και παραπάνω για τον υπολογισμό της επιφάνειας που καταλαμβάνουν τα κυκλώματα αθροίσαμε το πλήθος των slices, των καταχωρητών και των flip-flops που κάνει χρήση η κάθε υλοποίηση, δεδομένου ότι όλα τα slices καλύπτουν την ίδια επιφάνεια και αντίστοιχα όλα τα flip-flops και οι καταχωρητές. Έτσι μετράμε την επιφάνεια πυριτίου που καταλαμβάνει το κύκλωμά μας στο εσωτερικό ενός FPGA.

Έτσι συγκεντρώσαμε τα αποτελέσματα των μετρήσεων στο παρακάτω διάγραμμα:



Σχήμα 4.16 Συγκεντρωτικό διάγραμμα επιφάνειας-μήκους λέξης

Στο παραπάμω διάγραμμα βλέπουμε να επιβεβαιώνεται η υπεροχή της υλοποίησης Gauss Optimized Modified Booth Recoder (B) σε ότι αφορά την επιφάνεια που καταλαμβάνει. Όπως και στην ASIC υλοποίηση, έτσι και στην FPGA υλοποίηση, με την σωστή χρήση των δενδρικών συμπιεστών Wallace, βλέπουμε να αναδεικνύονται τα πλεονεκτήματα του αλγόριθμου του Gauss επί του συμβατικού αλγόριθμου, με την υλοποίηση των Wei, Du, και Chen να έχει χειρότερα αποτελέσματα από τις δύο υλοποίησεις του συμβατικού αλγόριθμου μόνο στα 64 bits, πράγμα που μας δείχνει ότι αν είχαμε υλοποιήσει και αυτήν με σωστή χρήση των δενδρικών συμπιεστών Wallace ενδεχομένως να είχε ξεπεράσει και αυτή καθολικά τις συμβατικές υλοποιήσεις.

5. Συμπεράσματα και επεκτάσεις

Σε αυτό το κεφάλαιο θα συνοψίσουμε την παρουσίαση της διπλωματικής εργασίας μας.

5.1. Σύνοψη και συμπεράσματα

Συνοψίζοντας τα αποτελέσματα της παρούσας διπλωματικής εργασίας, διερευνήσαμε διαφορετικές υλοποιήσεις του μιγαδικού πολλαπλασιαστή, με χρήση του συμβατικού αλγόριθμου και του αλγόριθμου του Gauss, διαφορετικών αρχιτεκτονικών και μονάδων υπολογισμού των ενδιάμεσων αποτελεσμάτων.

Τα συμπεράσματα που εξήγχθησαν από την μελέτη των αποτελεσμάτων είναι τα εξής:

- *Καθυστέρηση*

  Ο συμβατικός αλγόριθμος είναι ταχύτερος από τον αλγόριθμο του Gauss κάτι το οποίο διαπιστώνουμε εύκολα από όλες τις μετρήσεις που κάναμε. Συγκεκριμένα για την εφαρμογή των υλοποιήσεων σε ASIC βλέπουμε και στο *σχήμα 4.5* ότι και οι δύο αρχιτεκτονικές εφαρμογής του συμβατικού αλγόριθμου κερδίζουν τις υλοποιήσεις αλγόριθμου του Gauss για όλα τα μήκη λέξης από 8 έως 64 bits. Από την άλλη στις υλοποιήσεις σε FPGA στο *σχήμα 4.14* βλέπουμε ότι η αρχιτεκτονική που ακολουθούμε επηρεάζει τα αποτελέσματα. Η υλοποίηση που εκμεταλλεύεται την χρήση του δενδρικού συμπιεστή Wallace (Conventional Carry-Save (B)) μας αποδίδει την ταχύτερη υλοποίηση με την υλοποίηση Conventional Carry-Save (A) να έχει την χειρότερη απόδοση όσο ανεβαίνουμε σε μήκος λέξης. Στις υλοποιήσεις του αλγόριθμου του Gauss, στα ASIC βλέπουμε το σχήμα Gauss Optimized Modified Booth Recoder (A) να είναι το καλύτερο στα μικρότερα μήκη λέξης ενώ στα μεγαλύτερα μήκη λέξης καλύτερο είναι το σχήμα (B) μιας και κάνει και αυτό καλύτερη χρήση του δενδρικού συμπιεστή Wallace. Στα FPGA, παρατηρούμε μια κοινή συμπεριφορά των υλοποιήσεων μεταξύ τους με μικρές διαφορές ανάλογα με το μήκος λέξης, με την υλοποίηση Gauss Recoder Wei, Du, Chen να έχει μια σταθερά καλή απόδοση σε όλα τα μήκη λέξης, εκτός από τα 24 bits.

- *Κατανάλωση*

  Είδαμε πως οι υλοποιήσεις του συμβατικού αλγόριθμου σε ASIC έχουν χαμηλότερη κατανάλωση, ιδίως σε high performance συνθήκες, ενώ σε χαλαρώτερες συνθήκες περιορισμού του ρολογιού, και καθώς ανεβαίνουμε σε μήκος λέξης, οι υλοποιήσεις του αλγόριθμου του Gauss συγκλίνουν προς την απόδοση αυτών του συμβτικού αλγόριθμου. Στα FPGA, βλέπουμε την επίδραση των δενρικών συμπιεστών Wallace στην κατανάλωση

των κυκλωμάτων. Ο υλοποιήσεις Conventional Carry-Save (B) και Gauss Optimized Modified Booth Recoder (B), που κάνουν την καλύτερη χρήση των δένδρων Wallace, έχουν την χαμηλότερη κατανάλωση όσο ανεβαίνουμε σε μήκος λέξης.

- *Επιφάνεια*

  Ο αλγόριθμος του Gauss είναι ο οικονομικότερος σε ότι αφορά την επιφάνεια που καταλαμβάνουν τα κύκλωματα υλοποίησής του. Από την high performance λειτουργία μέχρι τις χαλαρότερες συνθήκες περιορισμού του ρολογιού και στα δύο μέσα εφαρμογής, ASIC και FPGA, βλέπουμε πως έχουμε μικρότερη επιφάνεια στις υλοποιήσεις του αλγόριθμου του Gauss, με την Gauss Optimized Modified Booth Recoder (B) να έχει την καλύτερη απόδοση.

Παραπάνω είδαμε συγκεντρωτικά, την σύγκριση των υλοποιήσεων κυρίως σε ότι αφορά τον αλγόριθμο υλοποίησης, με κριτήρια, την καθυστέρηση, την κατανάλωση και την επιφάνεια.

Πιο στοχευμένα μπορούμε να κάνουμε τις εξής συγκρίσεις:

- Η επίδραση της χρήσης του δενδρικού συμπιεστή Wallace, είδαμε ότι είναι πολύ μεγάλη στα κυκλώματά μας. Συγκρίνωντας τις δύο διαφορετικές υλοποιήσεις του συμβατικού αλγόριθμου είδαμε πως η βέλτιστη χρήση των δένδρων Wallace έγκειται στην δημιουργία όσο το δυνατόν μεγαλύτερων δένδρων. Έτσι αποδοτικότερη ήταν η υλοποίηση που δεν υπολόγιζε το αποτέλεσμα του πολλαπλασιασμού στην παράσταση $A*(X+Y)$, αλλά το κράταγε σε μορφή μερικών γινομένων τα οποία προστίθονταν ή αφαιρούνταν, ανάλογα με τον όρο που θέλαμε να παράξουμε. Στην συνέχεια συγκρίναμε τις τρεις υλοποιήσεις Gauss Optimized Modified Booth Recoder μεταξύ τους, οι οποίες σύμφωνα με τα αποτέλεσματα των υλοποιήσεων του συμβατικού αλγόριθμου, θα έπρεπε να έχουν σαν αποδοτικότερη υλοποίηση, την (C) η οποία λειτουργεί με τον ίδιο τρόπο με την υλοποίηση Conventional Carry-Save (B). Είδαμε όμως ότι η υλοποίηση Gauss Optimized Modified Booth Recoder (B) είναι η αποδοτικότερη, λόγω του υπολογισμού του κοινού όρου m_0. Έτσι βλέπουμε πως η υλοποίηση αυτή κάνει την καλύτερη χρήση του δενδρικού συμπιεστή Wallace γιατί επωφελείται των πλεονεκτημάτων του, αλλά κάνει και μια ελαχιστοποίηση στο μέγεθος των δένδρων. Όπως είπαμε και παραπάνω στον Verilog κώδικα ο όρος m_0 οδηγήθηκε στα δύο δένδρα με τρόπο που ο compiler θα τον προσθέτει στα χαμηλότερα “φύλλα” των δένδρων. Με τον τρόπο αυτό κάναμε την βέλτιστη χρήση του δενδρικού συμπιεστή Wallace στο κρίσιμο μονοπάτι, το οποίο ήταν εξ’αρχής το αδύναμο σημείο αυτής της υλοποίησης.

- Σε ότι αφορά τον αλγόριθμο του Gauss κάναμε χρήση δύο μονάδων υλοποίησης της πράξης A*(X+Y), του Προτεινόμενου Σχήματος επανακωδικοποίησης Πρόσθεσης (Αφαίρεσης)-Πολλαπλασιασμού και του Επανακωδικοποιητή σε Modified Booth, αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen. Για να κάνουμε μια σύγκριση των δύο μονάδων θα χρησιμοποιήσουμε τις υλοποιήσεις Gauss Optimized Modified Booth Recoder (A) και Gauss Recoder Wei, Du, Chen, γιατί κάνουν χρήση των ίδιων δένδρων Wallace και έτσι θα συγκρίνουμε την επίδραση των μονάδων μόνο, και όχι της υλοποίησης συνολικά. Από το *σχήμα 4.15* βλέπουμε πως το Προτεινόμενο Σχήμα επανακωδικοποίησης Πρόσθεσης (Αφαίρεσης)-Πολλαπλασιασμού έχει την μικρότερη καθυστέρηση σε ASIC, ενώ από το *σχήμα 4.14* βλέπουμε ότι σε FPGA, ο Επανακωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen υπέρεχει, όσο ανεβαίνουμε σε μήκος λέξης. Ομοίως από το *σχήμα 4.13*, βλέπουμε ότι και στην κατανάλωση στα FPGA ο Επανακωδικοποιητής σε Modified Booth αριθμού δυαδικής πλεονασματικής μορφής των Wei, Du και Chen είναι καλύτερος, ενώ σε ASIC συμβαίνει το αντίθετο με το Προτεινόμενο Σχήμα επανακωδικοποίησης Πρόσθεσης (Αφαίρεσης)-Πολλαπλασιασμού, να είναι για όλα τα μήκη λέξης καλύτερος. Το ίδιο συμβαίνει και για την επιφάνεια που καταλαμβάνουν τα κυκλώματα των δύο μονάδων με το Προτεινόμενο Σχήμα επανακωδικοποίησης Πρόσθεσης (Αφαίρεσης)-Πολλαπλασιασμού να είναι καλύτερο σε ASIC ενώ να χάνει όταν μεταφέρουμε τις υλοποιήσεις σε FPGA. Αυτό συμβαίνει λόγω των full adders που χρησιμοποιούμε στην μονάδα αυτή ενώ η μονάδα των Wei, Du και Chen με τους αντιστροφείς και την αριθμητική που χρησιμοποιεί ευνοείται σε FPGA μιας και αξιοποιεί καλύτερα τα λογικά blocks.

5.2. Μελλοντικές επεκτάσεις

- Για την περαιτέρω μελέτη του μιγαδικού πολλαπλασιαστή ένα πρώτο βήμα για την καλύτερη υλοποιήσή του είναι η εξερεύνηση διαφορετικών στρατηγικών του Design Compiler. Στην παρούσα διπλωματική κάναμε την σύνθεση με χρήση της εντολής compile_ultra, ενώ θα μπορούσαμε να ερευνήσουμε περαιτέρω τις DC-Ultra βελτιστοποιήσεις με τα ενσωματωμένα scripts, –area_high_effort_script αλλά και –timing_high_effort_script, για να δούμε την επίδρασή τους στις υλοποιήσεις.
- Με την ίδια λογική, για την υλοποίηση του μιγαδικού πολλαπλασιαστή σε FPGA, θα μπορούσαν να ερευνηθούν διαφορετικές αρχιτεκτονικές, οι οποίες με χρήση των αριθμητικών μονάδων (IP blocks) που έχουν τα FPGA θα μας δώσουν ενδεχομένως αποτελέσματα καλύτερα προσαρμοσμένα στο μέσο.
- Τέλος, η χρήση pipelined σχημάτων, με την κατάλληλη επιλογή αρχιτεκτονικών σχεδίασης, μπορεί να μας δώσει σημαντικά αποτελέσματα για τον μιγαδικό πολλαπλασιαστή, όπως και η διερεύνηση άλλων αλγόριθμων υλοποίησης του μιγαδικού

πολλαπλασιαστή, όπως η χρήση προενταμίευσης των όρων $\alpha_R+\alpha_I$ και $\alpha_R-\alpha_I$, και ο μιγαδικός πολλαπλασιαστής βασισμένος σε πολυπλέκτη.

6. Βιβλιογραφία

[1] Neil H. Weste, Kamran Eshraghian, “Principles of CMOS VLSI Design: A Systems Perspective”, Fourth Edition.

[2] Kiamal Pekmestzi, “DIGITAL VLSI SYSTEMS”, NTUA Lectures Notes, Athens 2003.

[3] Kostas Tsoumanis, Sotiris Xydis, Constantinos Efstathiou, Nikos Moschopoulos, and Kiamal Pekmestzi, “An Optimized Modified Booth Recoder for Efficient Design of the Add-Multiply Operator”.

[4] Belle W.Y.Wei, He Du and Honglu Chen, “Complex-Number Multiplier Using Radix-4 Digits”.

[5] Ian Kuon and Jonathan Rose, “Measuring the Gap Between FPGAs and ASICs”.

[6] Xilinx 7 Series FPGAs Overview, [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[7] Xilinx ISE, [Online]. Available: http://www.xilinx.com/

[8] Synopsys Design Compiler User Guide, [Online]. Available: http://www.cse.psu.edu/~cg577/READINGS/dcug_8.pdf

[9] Synopsys Design Compiler, [Online]. Available: http://www.synopsys.com

[10] ModelSim Corporation, [Online]. Available: http://www.model.com

[11] Synopsys Primetime PX, [Online]. Available: http://www.synopsys.com