ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΜΙΚΡΟΗΛΕΚΤΡΟΝΙΚΗ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βελτιστοποίηση μονάδας υπολογισμού Butterfly για τον αλγόριθμο FFT

Φώτιος Χ. Ντούσκας

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Γκιζόπουλος Δημήτριος, Καθηγητής
Κιαμάλ Πεκμεστζή, Καθηγητής

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2017

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βελτιστοποίηση μονάδας υπολογισμού Butterfly για τον αλγόριθμο FFT

Όνομα Π. Επώνυμο

Α.Μ.: 266

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Γκιζόπουλος Δημήτριος,** Καθηγητής ΕΚΠΑ
Κιαμάλ Πεκμεστζή, Καθηγητής ΕΜΠ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Γκιζόπουλος Δημήτριος,** Καθηγητής
Κιαμάλ Πεκμεστζή, Καθηγητής

Μάρτιος 2017

ΠΕΡΙΛΗΨΗ

Ο σκοπός της διπλωματικής εργασίας αυτής είναι η διερεύνηση της λειτουργίας της μονάδας υπολογισμού πεταλούδας, Butterfly Computation Unit (BCU), και η παρουσίαση δύο εναλλακτικών αρχιτεκτονικών για την υλοποίηση της μονάδας αυτής.

Η μονάδα υπολογισμού Butterfly είναι η βασική μονάδα για την υλοποίηση μονάδων Fast Fourier Transform (FFT) για αποδεκατισμό στο πεδίο του χρόνου, η οποία χρησιμοποιείται ευρέως σε εφαρμογές Ψηφιακής Επεξεργασίας Σημάτων όπως η Ανάλυση Φάσματος σημάτων, η Συμπίεση Δεδομένων, η σχεδίαση φίλτρων, η λύση Μερικών Διαφορικών Εξισώσεων, ο Πολλαπλασιασμός Πολυωνύμων, και ο υπολογισμός Συνέλιξης.

Σκοπός της παρούσας εργασίας είναι η διερεύνηση μίας συμβατικής υλοποίησης της μονάδας υπολογισμού πεταλούδας, αλλά και τριών εναλλακτικών μονάδων οι οποίες έχουν σχεδιαστεί με σκοπό την αποδοτικότερη υλοποίησή της.

Συγκεκριμένα, προτείνεται η χρήση του αλγορίθμου του Gauss, για τον πολλαπλασιασμό δύο μιγαδικών αριθμών καθώς η πράξη αυτή περιέχει το μεγαλύτερο υπολογιστικό φόρτο στην μονάδα υπολογισμού πεταλούδας. Επίσης, προτείνεται η χρήση προ-κωδικοποιημένων πολλαπλασιαστών για την υλοποίηση της Μονάδας Μιγαδικού Πολλαπλασιασμού, με σκοπό την περαιτέρω βελτίωση της λειτουργίας της, σε αντιπαράθεση με την συμβατική της υλοποίηση, που κάνει χρήση πολλαπλασιαστών κωδικοποίησης Modified Booth.

Οι παραπάνω μονάδες περιγράφηκαν με χρήση της γλώσσας περιγραφής υλικού Verilog, στην συνέχεια επαληθεύσαμε την ορθότητα λειτουργίας τους και συνθέσαμε τα κυκλώματα αυτά, με σκοπό να τα συγκρίνουμε ως προς την καθυστέρηση, την επιφάνεια που καταλαμβάνει το κύκλωμά τους, αλλά και την κατανάλωση ενέργειάς τους, με χρήση των εργαλείων σύνθεσης και προσομοίωσης της Synopsys.

Τέλος, έγινε παρουσίαση των αποτελεσμάτων που προέκυψαν, καθώς και μία συγκριτική μελέτη τους για τα διάφορα μήκη λέξης εισόδου. Έτσι προέκυψαν τα απαραίτητα συμπεράσματα για την λειτουργία των διαφορετικών σχημάτων που υλοποιήθηκαν, καθώς, κάθε ένα από τα σχήματα που σχεδιάστηκαν, έδωσε αποτελέσματα που τα καθιστά κατάλληλα για διαφορετικές εφαρμογές, ανάλογα με το που εμφανίζουν την βέλτιστη συμπεριφορά τους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ψηφιακή Επεξεργασία Σημάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Γρήγορος Μετασχηματισμός Fourier, Μονάδα υπολογισμού πεταλούδας, Modified Booth, Αλγόριθμος του Gauss, ASIC, VLSI

ABSTRACT

The scope of the thesis is the exploration of the Butterfly Computation Unit, and the introduction of two alternative architectures for the implementation of the said unit.

The Butterfly Computation Unit is the most essential unit for the implementation of the algorithm of the Fast Fourier Transform, for Decimation in Time, that is used widely in applications for Digital Signals Processing like Spectral Analysis, Data Compression, and in solving Partial Differential equations, Filtering algorithms, Polynomial Multiplication and Convolution.

The purpose of this thesis is the exploration of a conventional implementation of the Butterfly Computation Unit, but also three more alternative schemes that have been designed in order to be more efficient than the conventional one.

Specifically, we propose the use of the Gauss algorithm for multiplying two complex numbers as this operation contains the largest computational effort in the Butterfly Computation Unit. Also, we proposed the use of pre-encoded multipliers to implement the complex number multiplication unit, in order to further improve its functioning, in juxtaposition with the conventional implementation, which uses Modified Booth multipliers.

The units described above have been implemented, using Verilog hardware description language, next, their functionality has been behaviorally verified and we synthesized the circuits in order to experimentally compare them in delay, area and power consumption, using the Synopsys tools.

Lastly, we presented the results that arose, and we also presented a comparative study for various input bit-widths. This way, we reached the necessary conclusions for the operation of the different proposed schemes that we designed and implemented, and according to the findings each proposed scheme showed its advantages, depending on the application that they are targeted for, where they appear to perform optimally.

SUBJECT AREA: Digital Signal Processing

KEYWORDS: Fast Fourier Transform, Buttergly Computation Unit, Modified Booth, Gauss's Algorithm, ASIC, VLSI

Θα ήθελα να αφιερώσω την εργασία αυτήν στην οικογένειά μου και στην Χλόη

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τους επιβλέποντες καθηγητές μου, με ιδιαίτερες ευχαριστίες στον κύριο Πεκμεστζή για την βοήθειά του, την υπομονή του και την καθοδήγησή του. Θα ήθελα επίσης να ευχαριστήσω την οικογένειά μου για την υποστήριξή τους, τους φίλους μου για τα χρόνια που με ανέχονται, την Χλόη που μένει μαζί μου, και τους Βαγγέλη, Παντελή, Νίκο και Μάριο που με βοήθησαν και συνεχίζουν να με βοηθούν να κάνω τα πράγματα που μόνος δεν θα μπορούσα ούτε σε εκατό χρόνια να κάνω.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η διπλωματική εργασία αυτή εκπονήθηκε στα πλαίσια του Διατμηματικού Προγράμματος Μεταπτυχιακών Σπουδών στην Μικροηλεκτρονική, της σχολής Πληροφορικής του ΕΚΠΑ, και γράφτηκε στο εργαστήριο Microlab της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, με την βοήθεια και την επίβλεψη του κύριου Κιαμάλ Πεκμεστζή.

1. ΕΙΣΑΓΩΓΗ

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 κυκλωμάτων. 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 ασχολείται με τον μαθηματικό χειρισμό ενός σήματος και τους τρόπους επεξεργασίας του. Κάποιες από τις βασικότερες εφαρμογές της ψηφιακής επεξεργασίας σήματος 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 στη μονάδα υπολογισμού Butterfly[1].

Η μονάδα αυτή, όπως περιεγράφηκε από τους Cooley και Tukey στο [1], εισήγαγε την ιδέα του FFT, και έδωσε σημαντική υπολογιστική μείωση του αλγόριθμου από $O(N^2)$ σε $O(N\log_2 N)$ κάνοντας αποδοτική χρήση της συμμετρίας και της περιοδικότητας των twiddle factors του αλγόριθμου του FFT.

1.2 Αντικείμενο διπλωματικής

Η παρούσα διπλωματική έχει ως αντικείμενο την διερεύνηση διάφορων μονάδων υπολογισμού Butterfly που υλοποιούνται με δύο διαφορετικούς μιγαδικούς πολλαπλασιαστές, οι οποίοι με την σειρά τους, βασίζονται σε δύο διαφορετικούς αλγόριθμους, τον συμβατικό και τον αλγόριθμο του Gauss [2], καθώς και δύο διαφορετικές παραλλαγές αυτών των υλοποιήσεων τόσο στην αρχιτεκτονική που ακολουθήσαμε όσο και στον τρόπο σχεδίασης των κυκλωμάτων.

Βασισμένοι στην φύση των twiddle factors, αριθμοί οι οποίοι είναι γνωστοί ανεξαρτήτως των δεδομένων εισόδου, θα επιχειρήσουμε να διερευνήσουμε την χρήση προ-κωδικοποιημένων πολλαπλασιαστών για την μείωση του κρίσιμου μονοπατιού της μονάδας υπολογισμού Butterfly.

Η βελτίωση του κρίσιμου μονοπατιού είναι μία από τις πτυχές της παρούσης εργασίας. Για την μείωση της επιφάνειας αλλά και της κατανάλωσης του κυκλώματος που θα σχεδιάσουμε, θα διερευνήσουμε έναν εναλλακτικό αλγόριθμο για τον πολλαπλασιασμό μιγαδικών αριθμών, τον αλγόριθμο του Gauss.

Στο πλαίσιο της εργασίας μας, υλοποιήσαμε τα διαφορετικά κυκλώματα της μονάδας υπολογισμού Butterfly σε γλώσσα Verilog σε διάφορα μήκη λέξεως (16, 24, 32, 48 και 64 bits). Η λειτουργία των κυκλωμάτων αυτών προσομοιώθηκε και επαληθεύτηκε με το περιβάλλον Modelsim [11]. Η σύνθεση των κυκλωμάτων σε ASIC έγινε με χρήση του

Synopsys Design Compiler [10] και τα αποτελέσματα καθυστέρησης, επιφάνειας και κατανάλωσης εξάχθηκαν από το Synopsys Design Compiler και το Synopsys Primepower [12].

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε τα προβλήματα της σχεδίασης αριθμητικών κυκλωμάτων.
2. Υλοποιήσαμε δύο διαφορετικές μονάδες υπολογισμού Butterfly με χρήση δύο διαφορετικών αλγόριθμων, του συμβατικού και του αλγόριθμου του Gauss, καθώς και δύο παραλλαγές των αρχικών υλοποιήσεων, οι οποίες μπορούν να παραχθούν παραμετρικά, για διάφορα μήκη λέξης.
3. Εξετάσαμε την χρήση προ-κωδικοποιημένων πολλαπλασιαστών, και παρουσιάσαμε τα αποτελέσματα που προέκυψαν από αυτήν την διερεύνηση.
4. Αξιολογήσαμε τα πλεονεκτήματα και τα μειονεκτήματα των δύο αλγόριθμων υλοποίησης του μιγαδικού πολλαπλασιαστή.
5. Αξιολογήσαμε την απόδοση και τα πλεονεκτήματα/μειονεκτήματα κάθε σχήματος και αρχιτεκτονικής που σχεδιάσαμε.
6. Παρουσιάσαμε τα συγκριτικά αποτελέσματα των διαφορετικών υλοποιήσεων του μιγαδικού πολλαπλασιαστή στα 90nm ως εγχειρίδιο αναφοράς.

1.3 Οργάνωση κειμένου

Στο κεφάλαιο 2 γίνεται μια παρουσίαση της θεωρίας του αλγόριθμου FFT, των διαφορετικών κωδικοποιήσεων που χρησιμοποιήσαμε, των δύο αλγόριθμων υλοποίησης του μιγαδικού πολλαπλασιαστή, καθώς και των κυκλωμάτων που θα χρησιμοποιηθούν ως δομικές μονάδες.

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 αρχιτεκτονικές των κυκλωμάτων που σχεδιάστηκαν.

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.

2. ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

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 ο αλγόριθμος Fast Fourier Transform (FFT) σε radix-2 για αποδεκατισμό στο πεδίο του χρόνου (DIT), καθώς και η μονάδα υπολογισμού Butterfly που είναι απαραίτητη για τον αλγόριθμο FFT.

Στην συνέχεια, παρουσιάζονται οι κωδικοποιήσεις Booth και Modified Booth που προσφέρουν αποδοτικότερες υλοποιήσεις της πράξης του πολλαπλασιασμού, ελαττώνοντας την πολυπλοκότητα των κυκλωμάτων πολλαπλασιασμού. Επίσης, παρουσιάζεται ένα ειδικό σχήμα μίας Non-Redundant, radix-4, προσημασμένου ψηφίου, τεχνικής κωδικοποίησης, η οποία χρησιμοποιήθηκε για την υλοποίηση των κυκλωμάτων πολλαπλασιασμού πραγματικών αριθμών.

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 Ο μετασχηματισμός Fourier

Ο μετασχηματισμός Fourier διακριτού χρόνου (Discrete Fourier Transform - DFT) είναι ένας αλγόριθμος με ευρεία χρήση στις επιστήμες. Παίζει σημαντικό ρόλο σε πολλές επιστημονικές και τεχνολογικές εφαρμογές, όπως η ανάλυση κυματομορφών, η επίλυση γραμμικών μερικών διαφορικών εξισώσεων, ο υπολογισμός της συνέλιξης, η ψηφιακή επεξεργασία σήματος και η επεξεργασία εικόνας. Ο DFT είναι ένας γραμμικός μετασχηματισμός που απεικονίζει n ομοιόμορφα δείγματα ενός κύκλου ενός περιοδικού σήματος, σε ίσο αριθμό σημείων που αντιπροσωπεύουν το φάσμα συχνοτήτων του σήματος. Η πολυπλοκότητά του όμως, που είναι $O(n^2)$, κρίνεται μη ικανοποιητική.

Το 1965, οι Cooley και Tukey [1] δημιούργησαν έναν αλγόριθμο για τον υπολογισμό ενός n-σημείων DFT (n-point DFT) με πολυπλοκότητα $O(N\log_2 N)$. Ο αλγόριθμος τους αποτέλεσε μια σημαντικότατη εξέλιξη στο πεδίο των DFT αλγορίθμων και αναφέρεται ως ταχύς μετασχηματισμός Fourier (Fast Fourier Transform - FFT). Λόγω της ευρείας χρήσης του αλγορίθμου, έχουν γίνει πολλές προσπάθειες για την υλοποίηση FFT αλγορίθμων σε παράλληλες αρχιτεκτονικές.

Υπάρχουν πολλές διαφορετικές μορφές FFT αλγορίθμων. Στην εργασία αυτή θα ασχοληθούμε με τον FFT αλγόριθμο βάσης-2 (radix-2 FFT) για αποδεκατισμό στο πεδίο του χρόνου (decimation in time, DIT).

2.1.1 Αλγόριθμος FFT, radix-2, DIT

Ο γρήγορος αλγόριθμος Fourier, βασης-2 με αποδεκατισμό στο πεδίο του χρόνου είναι η απλούστερη και πιο κοινή μορφή του αλγόριθμου των Cooley-Tukey. Ο αλγόριθμος αυτός χωρίζει τον DFT N-σημείων, σε δύο DFTs N/2-στοιχείων. Ο DFT μετασχηματισμός ορίζεται από τον παρακάτω τύπο:

$$X_k = \sum_{n=0}^{N-1} X_n e^{-\frac{2\pi i}{\mathrm{N}} nk},$$

όπου k είναι ακέραιος αριθμός με τιμή από 0 έως N-1.

Αρχικά, υπολογίζονται οι DFTs των άρτιων και περιττών όρων, και στην συνέχεια τα αποτελέσματά τους συνδυάζονται κατάλληλα για να παράγουν τα αποτελέσματα ολόκληρης της ακολουθίας. Αυτός ο τρόπος εφαρμόζεται αναδρομικά για να μειωθεί ο ολικός χρόνος εκτέλεσης σε $O(N\log_2 N)$. Η απλουστευμένη αυτή μορφή προϋποθέτει ο αριθμός N να είναι δύναμη του δύο. Έτσι η συνάρτηση x_n αναπροσαρμόζεται σε δύο μέρη, ένα άθροισμα των άρτιων όρων $(x_{2m} = x_0, x_2, ..., x_{N-2})$ και ένα άθροισμα των περιττών όρων $(x_{2m+1} = x_1, x_3, ..., x_{N-1})$:

$$x_k = \sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{\mathrm{N}} 2mk} + \sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{\mathrm{N}}(2m+1)k}$$

Παραγοντοποιώντας τον κοινό πολλαπλασιαστή του δεύτερου αθροίσματος προκύπτει το παρακάτω:

$$x_k = \underbrace{\sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{\mathrm{N}/2} mk}}_{\text{DFT άρτιων όρων του } x_m} + e^{-\frac{2\pi i}{\mathrm{N}} k} \cdot \underbrace{\sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{\mathrm{N}/2} mk}}_{\text{DFT περιττών όρων του } x_m}$$

Λόγω της περιοδικότητας των όρων γνωρίζουμε ότι:

$$\sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{\mathrm{N}/2} mk + \frac{\mathrm{N}}{2}} = \sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{\mathrm{N}/2} mk}$$

και

$$\sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{\mathrm{N}/2} mk + \frac{\mathrm{N}}{2}} = \sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{\mathrm{N}/2} mk}$$

Έτσι μπορούμε να ξαναγράψουμε την σχέση όπως βλέπουμε παρακάτω:

$$x_k = \begin{cases} \displaystyle\sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{\mathrm{N}/2} mk} + e^{-\frac{2\pi i}{\mathrm{N}} k} \cdot \sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{\mathrm{N}/2} mk} & \text{για } 0 \le k < N/2 \\ \displaystyle\sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{\mathrm{N}/2} m(k-\mathrm{N}/2)} + e^{-\frac{2\pi i}{\mathrm{N}} k} \cdot \sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{\mathrm{N}/2} m(k-\mathrm{N}/2)} & \text{για } N/2 \le k < N \end{cases}$$

Επίσης οι όροι της μορφής $e^{-\frac{2\pi i}{\mathrm{N}/2} mk}$ του FFT είναι μία πρωτόγονη ρίζα της ενότητας, και θα αναφερόμαστε σε αυτούς με τον όρο *"twiddle factors"*, που είναι και η διεθνής τους ονομασία, και γνωρίζουμε ότι έχει την παρακάτω ιδιότητα:

$$e^{-\frac{2\pi i}{\mathrm{N}}(k+N/2)} = e^{-\frac{2\pi ik}{\mathrm{N}} - \pi i} = e^{-\pi i} \cdot e^{-\frac{2\pi ik}{\mathrm{N}}} = -e^{-\frac{2\pi ik}{\mathrm{N}}}$$

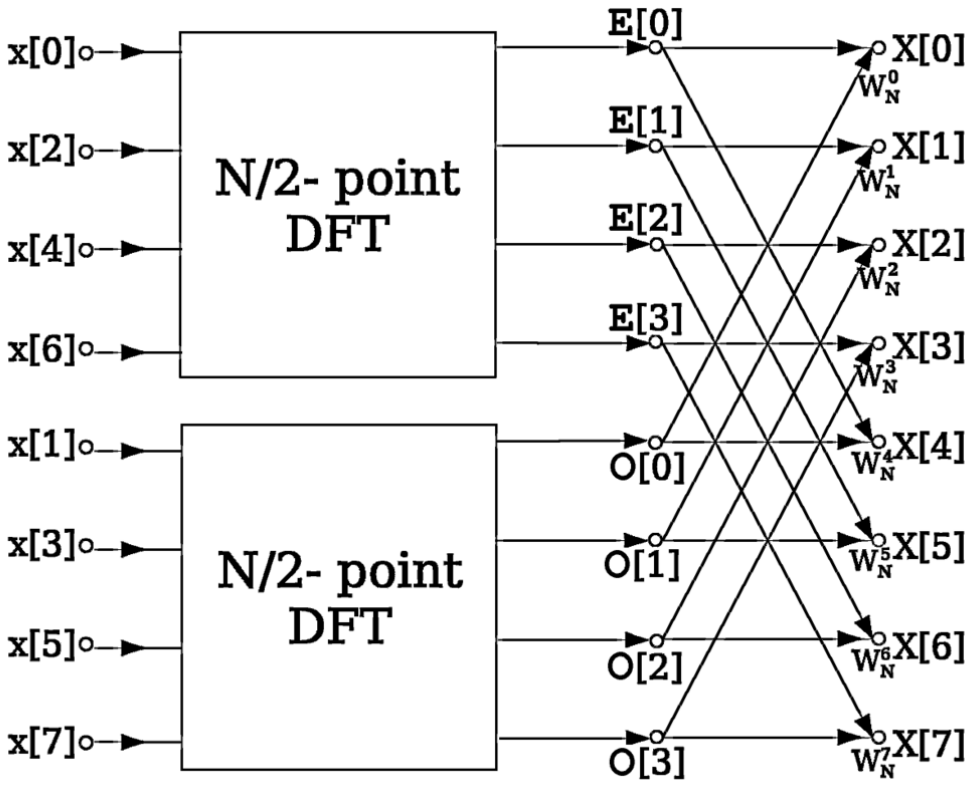
Με αυτόν τον τρόπο μπορούμε να μειώσουμε το πλήθος των twiddle factors στο μισό. Έτσι για $0 \le k < N/2$ είναι:

$$x_k = \sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{N/2}mk} + e^{-\frac{2\pi i}{N}k} \cdot \sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{N/2}mk}$$

$$x_{k+\frac{N}{2}} = \sum_{m=0}^{\frac{N}{2}-1} X_{2m} e^{-\frac{2\pi i}{N/2}mk} - e^{-\frac{2\pi i}{N}k} \cdot \sum_{m=0}^{\frac{N}{2}-1} X_{2m+1} e^{-\frac{2\pi i}{N/2}mk}$$

2.1.2 Μονάδα Υπολογισμού Butterfly

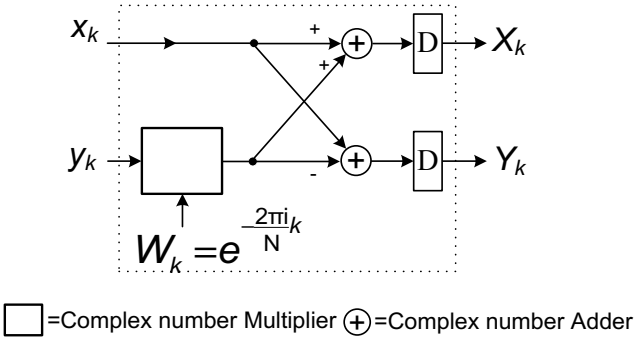
Όπως περιγράψαμε παραπάνω, ο αλγόριθμος FFT, όπως τον περιέγραψαν οι Cooley-Tukey, παρουσιάζει κέρδος σε ταχύτητα με την επαναχρησιμοποίηση των ενδιάμεσων αποτελεσμάτων για να υπολογίσει πολλαπλές εξόδους από DFTs. Η λειτουργία του, για οχτώ σημεία, φαίνεται στο παρακάτω σχήμα:



Σχήμα 1: Αλγόριθμος FFT, radix-2, DIT για N = 8

Οι ενδιάμεσες τιμές του παραπάνω σχήματος E[N/2] και O[N/2], αντιστοιχούν στους άρτιους (Even) και περιττούς (Odd) όρους, αντίστοιχα. Οι όροι $W_N[N]$ είναι οι twiddle factors, και βλέπουμε πως οι τελικοί αριθμοί προκύπτουν από τις προσθέσεις και αφαιρέσεις των γινομένων, μεταξύ των περιττών όρων και των twiddle factors, με τους άρτιους όρους.

Στην γενικευμένη της μορφή, η δοκιμή μονάδα του παραπάνω υπολογισμού έχει την παρακάτω μορφή:



Σχήμα 2: Γενικευμένη μορφή μονάδας Butterfly

Λόγω του σχήματός του, ο παραπάνω υπολογισμός ονομάζεται υπολογισμός πεταλούδας, και η μονάδα που παράγει το αποτέλεσμα αυτής πράξης καλείται Butterfly μονάδα. Στο παραπάνω σχήμα οι αριθμοί x_k, y_k, X_k, Y_k και W_k είναι μιγαδικοί αριθμοί.

2.2 Κωδικοποιήσεις Booth, Modified Booth και NR4SD

2.2.1. Κωδικοποίηση Booth

Ο αλγόριθμος φέρει το όνομα του Andrew Donald Booth και επινοήθηκε το 1950. Στην συνέχεια εξηγείται πως λειτουργεί η κωδικοποίηση Booth. Έστω δυαδικός αριθμός X σε μορφή συμπληρώματος ως προς δύο. Όπως γνωρίζουμε, 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: $\mathrm{X} = 2\mathrm{X} - \mathrm{X}$

2X=	X_{n-2}	X_{n-3}	…	X_0	0
-X =	$-X_{n-1}$	$-X_{n-2}$	…	$-X_1$	$-X_0$
	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 πρόσημου.

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 αποτέλεσμα 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 1 παρουσιάζονται όλες οι δυνατές περιπτώσεις κατά αντιστοιχία με τα κωδικοποιημένα ψηφία που προκύπτουν, καθώς και οι λογικές λειτουργίες που αντιπροσωπεύουν.

Πίνακας 1: Αντιστοίχιση δυαδικών ψηφίων με λειτουργίες και κωδικοποίηση Booth

Y_i	Y_{i-1}	Κωδικοποιημένα ψηφία z_i $(y_{i-1}-y_i)$	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 κωδικοποίηση 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_i, Y_{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 (MB), 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 zi είναι τα ψηφία που προέκυψαν από την απλή κωδικοποίηση Booth του Y.

Με περαιτέρω ανάλυση της σχέση αυτής έχουμε:

$$Y = \sum_{i=0}^{n-1} Z_i \cdot 2^i = \sum_{i=0}^{\frac{n}{2}-1} (Z_{2i} \cdot 2^{2i} + Z_{2i+1} \cdot 2^{2i+1}) = \sum_{i=0}^{\frac{n}{2}-1} 2^{2i} \cdot (Z_{2i} + Z_{2i+1} \cdot 2) \Rightarrow$$

$$Y = \sum_{i=0}^{\frac{n}{2}-1} 4^i \cdot (w_i)$$

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.

Πίνακας 2: Αντιστοίχιση δυαδικών ψηφίων με απλή κωδικοποίηση Booth & Modified Booth

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Κωδικοποίηση Booth Z_{2i+1}	Z_{2i}	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 3.

Πίνακας 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 Β κωδικοποιημένος σύμφωνα με τον αλγόριθμο Modified Booth γίνεται, $B = 2\bar{1}1\bar{1}$. Άρα θα μας χρειαστούν οι αριθμοί 2Α, Α και -Α. Τους υπολογίζουμε:

$$2A = 10011100,\ -A = 00110010$$

Ο πολλαπλασιασμός του Α με τον Β φαίνεται στον Πίνακα 4.

Πίνακας 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 -Α (Πρώτο μερικό γινόμενο)
1 0 0 1 1 1 0	+1 Πρόσθεσε Α (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 -Α (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 2Α (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 πρόσημου 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.

2.2.3 Κωδικοποίηση NR4SD

Σε αυτήν την παράγραφο, παρουσιάζουμε ένα ειδικό σχήμα μίας Non-Redundant, radix-4, προσημασμένου ψηφίου (Sign-Digit), τεχνικής κωδικοποίησης (NR4SD). Έστω ο πολλαπλασιασμός δύο αριθμών Α και Β σε μορφή συμπληρώματος ως προς 2, όπου ο καθένας έχει $n = 2k$ bits. Όπως στην MB μορφή, και εδώ, ο αριθμός των μερικών γινομένων μειώνεται στο μισό. Όταν κωδικοποιούμε έναν αριθμό Β σε μορφή συμπληρώματος ως προς 2, τα ψηφία b_j^{NR-} παίρνουν μία από τις παρακάτω τέσσερις τιμές: $\{-2,-1,0,+1\}$ ή $\{-1,0,+1,+2\}$, στον NR4SD$^-$ ή στον NR4SD$^+$ αλγόριθμο, αντίστοιχα. Σε αντίθεση με την κωδικοποίηση Modified Booth, χρησιμοποιούνται τέσσερις τιμές, αντί

για πέντε, το οποίο οδηγεί στον περιορισμό: $0 \le j \le k-2$. Έτσι για να καλυφθεί το δυναμικό εύρος των αριθμών σε μορφή συμπληρώματος ως προς 2, το πιο σημαντικό ψηφίο κωδικοποιείται σε μορφή MB. Στην συνέχεια παρουσιάζεται ο αλγόριθμος της κωδικοποίησης NR4SD$^-$. Ο αλγόριθμός της κωδικοποίησης NR4SD$^+$, είναι αντίστοιχος με αυτόν της κωδικοποίησης NR4SD$^-$, αλλά για τιμές $\{-1,0,+1,+2\}$ και μπορεί να βρεθεί στο [4]. Παρουσιάζεται μόνο η κωδικοποίηση NR4SD$^-$ γιατί αυτή χρησιμοποιήθηκε στην παρούσα εργασία.

NR4SD- Αλγόριθμος:

1. Θεωρούμε αρχικές τιμές $j = 0$ και $c_0 = 0$.
2. Υπολογίζουμε το κρατούμενο c_{2j+1} και το άθροισμα n_{2j}^- ενός ημιαθροιστή (HA) με εισόδους b_{2j} και c_{2j} (Σχήμα 3α).

$$c_{2j+1} = b_{2j} \wedge c_{2j},\ n_{2j}^+ = b_{2j} \oplus c_{2j}.$$

3. Υπολογίζουμε το θετικά προσημασμένο κρατούμενο $c_{2j+1}(+)$ και το αρνητικά προσημασμένο άθροισμα $n_{2j+1}^-(-)$ του τροποποιημένου ημιαθροιστή (HA*), με εισόδους $b_{2j+1}(+)$ και $c_{2j+1}(+)$ (Σχήμα 3α). Οι έξοδοι c_{2j+2} και n_{2j+1}^- του HA* σχετίζονται με την είσοδό του με βάση τις παρακάτω σχέσεις:

$$2c_{2j+2} - n_{2j+1}^- = b_{2j+1} + c_{2j+1}.$$

Η λειτουργία του HA* συνοψίζεται στις παρακάτω σχέσεις:

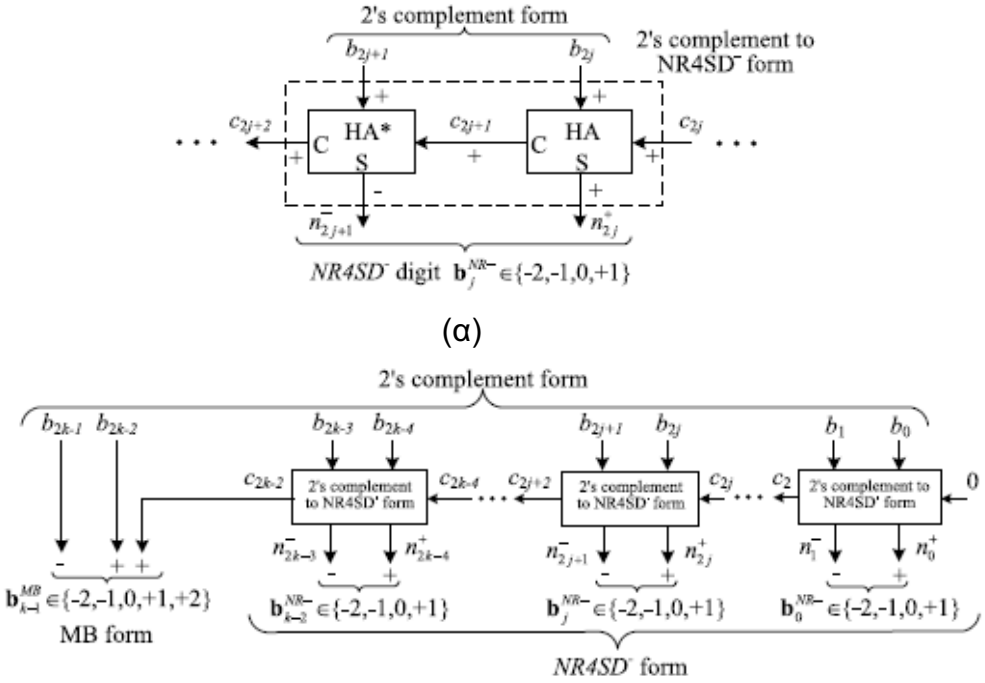
$$c_{2j+2} = b_{2j+1} \vee c_{2j+1},\ n_{2j+1}^- = b_{2j+1} \oplus c_{2j+1}.$$

4. Υπολογίζουμε την τιμή του ψηφίου b_j^{NR-}.

$$b_j^{NR-} = -2n_{2j+1}^- + n_{2j}^+.$$

Η παραπάνω εξίσωση προκύπτει, επειδή το n_{2j+1}^- είναι αρνητικά προσημασμένο ενώ το n_{2j}^+ είναι θετικά προσημασμένο.

5. Το j γίνεται j+1, $j := j+1$.
6. Εάν $j < k-1$, συνεχίζουμε από το βήμα 2, αλλιώς κωδικοποιούμε το πιο σημαντικό ψηφίο, σύμφωνα με τον αλγόριθμο της MB μορφής, θεωρώντας ως τρία συνεχόμενα ψηφία, τα ψηφία b_{2k-1}, b_{2k-2} και c_{2k-2} (Σχήμα 3β). Εάν j = k, σταματάμε.



Σχήμα 3: Σχηματικό διάγραμμα της NR4SD⁻ κωδικοποίησης σε επίπεδο ψηφίου (α), και λέξης (β).

Σύμφωνα με τα παραπάνω, προκύπτει ο πίνακας 5, της NR4SD⁻ κωδικοποίησης:

Πίνακας 5: NR4SD⁻ κωδικοποίησης

2's complement			NR4SD⁻ form			Digit	NR4SD⁻ Encoding		
b_{2j+1}	b_{2j}	c_{2j}	c_{2j+2}	n^-_{2j+1}	n^+_{2j}	$\mathbf{b_j^{NR-}}$	one^+_j	one^-_j	two^-_j
0	0	0	0	0	0	**0**	0	0	0
0	0	1	0	0	1	**+1**	1	0	0
0	1	0	0	0	1	**+1**	1	0	0
0	1	1	1	1	0	**-2**	0	0	1
1	0	0	1	1	0	**-2**	0	0	1
1	0	1	1	1	1	**-1**	0	1	0
1	1	0	1	1	1	**-1**	0	1	0
1	1	1	1	0	0	**0**	0	0	0

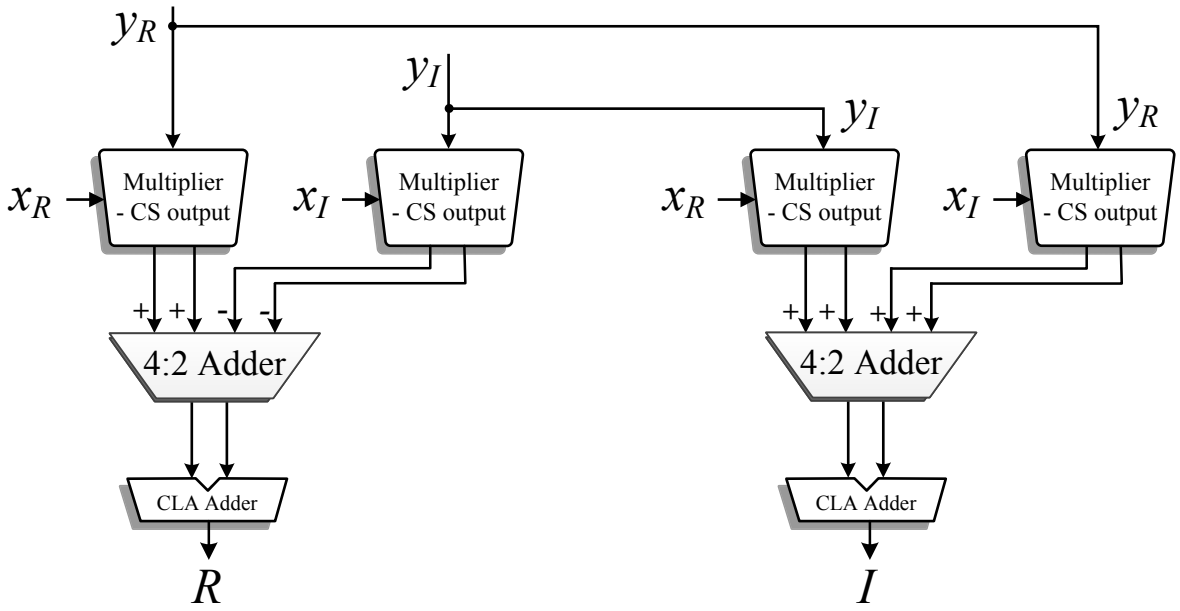
2.3 Μιγαδικός Πολλαπλασιαστής

2.3.1 Συμβατικός αλγόριθμος για Μιγαδικό Πολλαπλασιασμό

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

$$(x_R + j \cdot x_I) \cdot (y_R + j \cdot y_I) = (x_R \cdot y_R - x_I \cdot y_I) + j(x_R \cdot y_I + x_I \cdot y_R) = R + j \cdot I.$$

Το κύκλωμα που αντιστοιχεί στην παραπάνω κύκλωμα υλοποιείται όπως φαίνεται παρακάτω, στο Σχήμα 4:



Σχήμα 4: Διάγραμμα μονάδας Μιγαδικού Πολλαπλασιασμού με χρήση του συμβατικού αλγόριθμου

Οι μονάδες *Multiplier-CS output* είναι μονάδες πολλαπλασιασμού πραγματικών αριθμών, οι μονάδες *4:2 Adder* είναι συμπιεστές 4 προς 2 και στο τέλος οδηγούν δύο *Carry-Lookahead Adders(CLA Adder)* για την παραγωγή του τελικού αποτελέσματος σε μορφή συμπληρώματος ως προς 2.

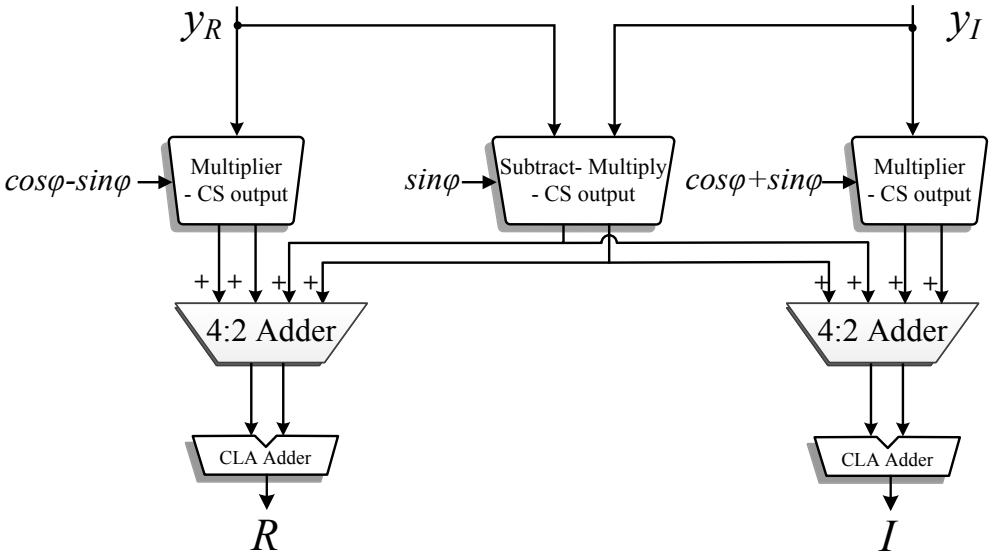
2.3.2 Αλγόριθμος του Gauss για Μιγαδικό Πολλαπλασιασμό

Ο αλγόριθμος του Gauss [2] 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 πού είδαμε στο **2.3.1**. Το πραγματικό και το φανταστικό μέρος μπορούν να παραχθούν ως εξής:

$$\left.\begin{aligned} m_0 &= (y_R - y_I) \cdot x_I \\ m_1 &= (x_R - x_I) \cdot y_R \\ m_2 &= (x_R + x_I) \cdot y_I \end{aligned}\right\} \Rightarrow \begin{aligned} R &= m_1 + m_0 \\ I &= m_2 + m_0 \end{aligned}$$

Με χρήση των μονάδων *Multiplier-CS output*, *4:2 Adder* και *CLA Adders* υλοποιούμε στο παρακάτω διάγραμμα (Σχήμα 5), το κύκλωμα για την παραγωγή του τελικού αποτελέσματος σε μορφή συμπληρώματος ως προς 2.



Σχήμα 5: Διάγραμμα μονάδας Μιγαδικού Πολλαπλασιασμού με χρήση του αλγόριθμου του Gauss

2.4 Δομικές Μονάδες

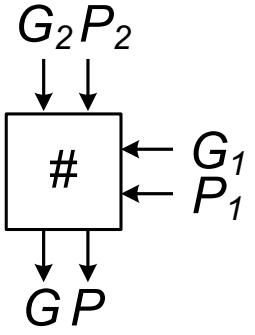
2.4.1 Αθροιστές

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 6: Κύκλωμα πρόβλεψης κρατουμένου

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

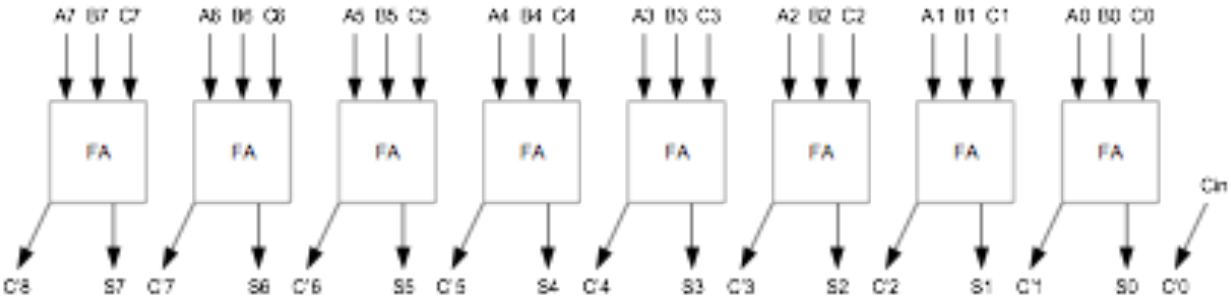
$$G = G2 + (G1 \cdot P2)$$
$$P = P1 \cdot P2$$

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 7: Κύκλωμα Αθροιστή Carry-Save

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.

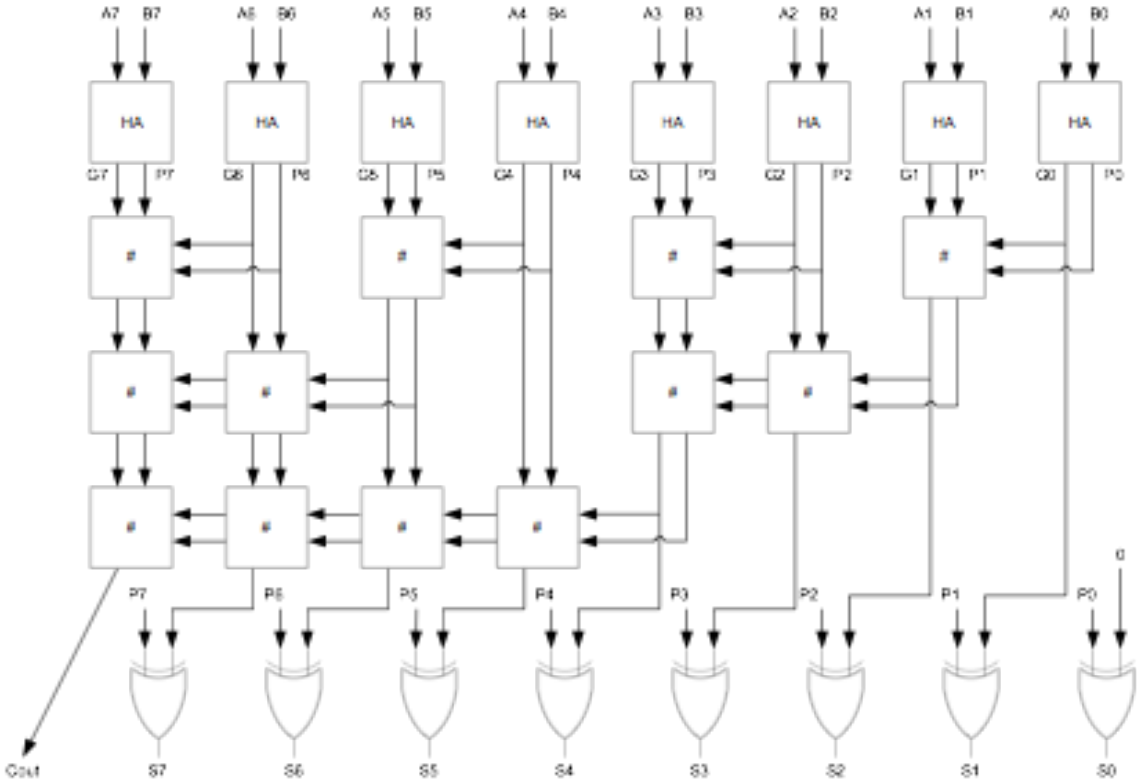
Ο αθροιστής σωσίματος κρατουμένου έχει απλή δομή και την ίδια επιφάνεια με τον αθροιστή διάδοσης κρατουμένου. Επίσης μπορεί να χειριστεί τρείς αντί για δύο δυαδικούς αριθμούς ταυτόχρονα. Το μεγάλο πλεονέκτημά του όμως είναι ότι η καθυστέρηση (critical path) ενός N-bit αθροιστή είναι μόνο 1 επίπεδο πλήρους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

Αθροιστής πρόβλεψης κρατουμένου (Carry Lookahead Adder)

Ο αθροιστής πρόβλεψης κρατουμένου (Carry Lookahead 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}.

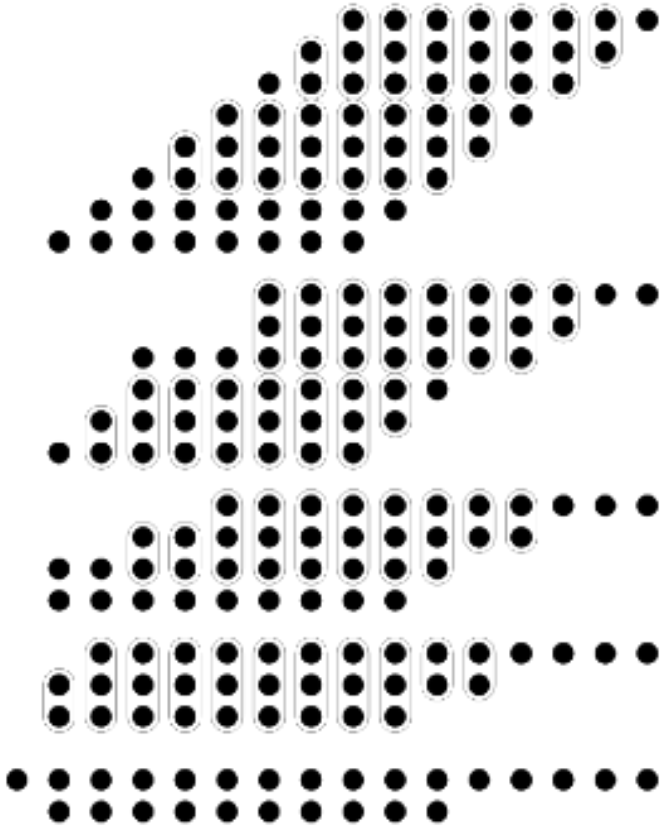


Σχήμα 8: Κύκλωμα αθροιστή πρόβλεψης κρατουμένου

Δεντρικός συμπιεστής Wallace (4:2 Adder)

Στόχος του δεντρικού 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).



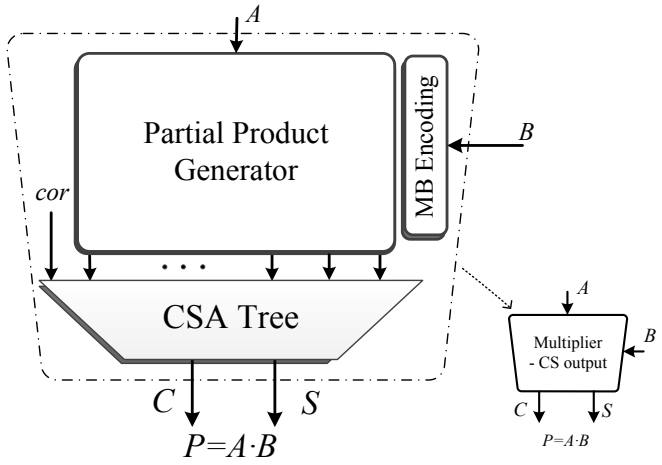
Σχήμα 9: Wallace δενδρικός συμπιεστής

2.4.2 Πολλαπλασιαστές

Στην παρούσα εργασία θα γίνει χρήση δύο ειδών πολλαπλασιαστών: ο Modified Booth – Carry-Save Output πολλαπλασιαστής και NR4SD Pre-Encoded πολλαπλασιαστής.

Modified Booth – Carry-Save Output πολλαπλασιαστής:

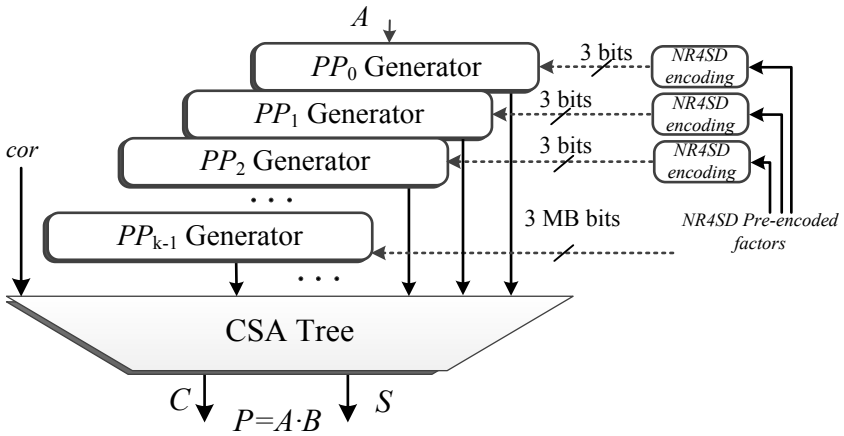
Ο Modified Booth – Carry-Save Output πολλαπλασιαστής, χρησιμοποιεί το αλγόριθμο Modified Booth, δέχεται σαν είσοδο δύο αριθμούς σε μορφή συμπληρώματος ως προς 2 και παράγει στην έξοδό του το αποτέλεσμα του πολλαπλασιασμού σε μορφή Carry-Save. Η αρχιτεκτονική αυτής της μονάδας φαίνεται στο παρακάτω διάγραμμα, Σχήμα 10:



Σχήμα 10: Αρχιτεκτονική MB Πολλαπλασιαστή

NR4SD Pre-Encoded πολλαπλασιαστής:

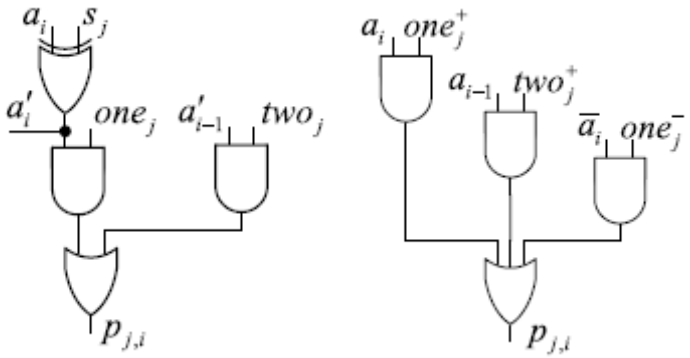
Αντίστοιχα με προηγουμένως, ο NR4SD Pre-Encoded πολλαπλασιαστής χρησιμοποιεί τον αλγόριθμο της NR4SD μορφής, όπως αυτός περιεγράφηκε στο **2.2.3**. Η διαφορά των δύο πολλαπλασιαστών βρίσκεται στην είσοδο του ενός παράγοντα του πολλαπλασιασμού, καθώς και στο κύκλωμα παραγωγής των μερικών γινομένων. Στην κωδικοποίηση NR4SD ο ένας παράγοντας κωδικοποιείται με ψηφία που μπορούν να πάρουν μία από τις τιμές $\{-2,-1,0,+1\}$, ένα λιγότερο από αυτά της κωδικοποίησης Modified Booth, κάτι που οδηγεί σε απλούστερα κυκλώματα παραγωγής μερικών γινομένων. Επίσης η μορφή της εισόδου του ενός παράγοντα στον πολλαπλασιασμό θεωρείται ότι λαμβάνεται στην ενδιάμεση μορφή NR4SD one^+, one^-, και two^-. Το διάγραμμα της αρχιτεκτονικής του NR4SD πολλαπλασιαστή φαίνεται στο σχήμα 11.



Σχήμα 11: Αρχιτεκτονική NR4SD Πολλαπλασιαστή

Βλέπουμε πως υπάρχει ένα κύκλωμα επανακωδικοποίησης, της εισόδου της μορφής one^+, one^-, και two^-, και συμπεραίνουμε πως οι δύο πολλαπλασιαστές έχουν τα ίδια επίπεδα καθυστέρησης, και συνεπώς οι διαφορές τους σε καθυστέρηση, επιφάνεια και κατανάλωσης ενέργειας οφείλεται στην πολυπλοκότητα των επιμέρους μονάδων.

Τέλος, στο σχήμα 12, βλέπουμε την διαφορά των κυκλωμάτων παραγωγής μερικών γινομένων, για ένα bit.



(α) (β)

Σχήμα 12: Κύκλωμα παραγωγής i-οστού bit μερικών γινομένων α) MB Πολλαπλασιαστή, και β) NR4SD πολλαπλασιαστή

3. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΣΧΕΔΙΑΣΗ ΥΛΟΠΟΙΗΣΕΩΝ

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

Αρχικά, παρουσιάζονται αναλυτικά τα κυκλώματα και οι αρχιτεκτονικές των επιμέρους μονάδων που χρησιμοποιήθηκαν για τις μονάδες υπολογισμού Butterfly που υλοποιήθηκαν.

Επίσης, παρουσιάζονται αναλυτικά δύο υλοποιήσεις των μονάδων υπολογισμού Butterfly, όπου ο μιγαδικός πολλαπλασιαστής έχει υλοποιηθεί σύμφωνα με τον συμβατικό αλγόριθμο του μιγαδικού πολλαπλασιαστή.

Τέλος, παρουσιάζονται δύο μονάδες υπολογισμού Butterfly που κάνουν χρήση του αλγόριθμου του Gauss για τον πολλαπλασιασμό μιγαδικών αριθμών. Όλες οι παραπάνω υλοποιήσεις κάνουν χρήση της αριθμητικής Carry-Save με σκοπό να έχουν την καλύτερη δυνατή απόδοση.

3.1 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** θεωρούμε ότι έχουμε τον πολλαπλασιασμό δύο αριθμών a, b μήκους λέξης, n bits.

3.1.1 Κύκλωμα κωδικοποίησης

Το σχήμα κωδικοποίησης που θα χρησιμοποιήσουμε στον πολλαπλασιαστή προτάθηκε το 2003 από τον Huang [3], 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, περιγράφεται στην συνέχεια.

Πίνακας 6: Πίνακας αληθείας σημάτων κωδικοποίησης Modified Booth

b_{2i+1}	b_{2i}	b_{2i-1}	$sign_i$	one_i	two_i	Modified Booth: W_i
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

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, το 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].

3.1.2 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:

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

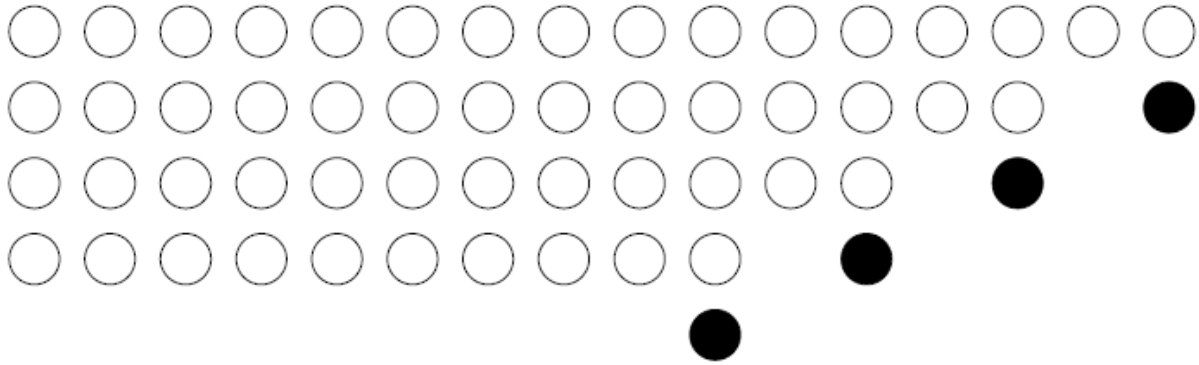
Τέλος το πιο σημαντικό bit του μερικού γινομένου είναι:

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

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 13 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 13: 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 αναλύεται 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:

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

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

$$ct = \sum_{k=0}^{3} \overline{s_k} 2^{8+2k} - \{2^{14} + 2^{12} + 2^{10} + 2^8\} = 2^{15} + 2^{13} + 2^{11} + 2^9 + 2^8$$

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

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

3.2 Μονάδα πολλαπλασιασμού NR4SD

Ο πολλαπλασιαστής NR4SD είναι ένας προκωδικοποιημένος πολλαπλασιαστής όπου ο πολλαπλασιαστέος είναι ένας αριθμός σε μορφή συμπληρώματος ως προς 2 ενώ ο πολλαπλασιαστής είναι στην προκωδικοποιημένη μορφή NR4SD$^-$, όπως την περιγράψαμε στο 2.2.3. Η χρήση αυτού του πολλαπλασιαστή οφείλεται στα twiddle factors του FFT αλγόριθμου. Οι αριθμοί αυτοί είναι σταθεροί όροι, και έτσι η χρήση του NR4SD πολλαπλασιαστή θα μας αποφέρει σημαντικά καλύτερα αποτελέσματα σε καθυστέρηση, επιφάνεια, και κατανάλωση του τελικού κυκλώματος.

Για τον πολλαπλασιασμό δύο αριθμών των n bit, ο πολλαπλασιαστέος *a* είναι σε μορφή συμπληρώματος ως προς 2, ενώ ο πολλαπλασιαστής στην είσοδο της μονάδας είναι κωδικοποιημένος σε δύο διανύσματα των $(n/2-1)$ bits, τα διανύσματα sign_tm και one_tm, και σε ένα Modified Booth ψηφίο, δηλαδή σε 3 bits με ονόματα sign_mb, two_mb και one_mb. Ο αριθμός αυτό επανακωδικοποιείται πριν οδηγηθεί στο κύκλωμα παραγωγής μερικών γινομένων.

3.2.1 Κύκλωμα κωδικοποίησης

Το κύκλωμα κωδικοποίησης βασίζεται στην κωδικοποίηση που περιεγράφηκε στο 2.3.2. Ο προκωδικοποιημένος αριθμός σε μορφή NR4SD-, χρειάζεται να επανακωδικοποιηθεί, στα σήματα twominus, oneminus, και oneplus, με μήκη λέξης $(n/2-1)$ bits, με ένα απλό κύκλωμα, όπου για το i-οστό bit, περιγράφεται στις παρακάτω σχέσεις:

$$twominus[i] = \overline{(sign\_tm[i] \wedge \overline{one\_tm[i]})}$$
$$oneminus[i] = \overline{(sign\_tm[i] \wedge one\_tm[i])}$$
$$oneplus[i] = \overline{sign\_tm[i]} \wedge one\_tm[i]$$

όπου $0 \le n \le (n/2-1)$.

Τα ψηφία Modified Booth, sign_mb, two_mb και one_mb, παράγουν το μερικό γινόμενο υψηλότερης θέσης, με χρήση του κυκλώματος που περιεγράφηκε στο 3.1.2.

Αυτό το στάδιο επανακωδικοποίησης απλοποιεί το κύκλωμα παραγωγής μερικών γινομένων που περιγράφεται στην συνέχεια.

3.2.2 Κύκλωμα δημιουργίας μερικών γινομένων

Η υπομονάδα αυτή, δέχεται σαν είσοδο τον πολλαπλασιαστέο σε μορφή συμπληρώματος ως προς 2 και τα διανύσματα twominus, oneminus, και oneplus. Σημειώνεται εδώ πως το πιο σημαντικό μερικό γινόμενο, κωδικοποιείται σε Modified Booth μορφή άρα η παραγωγή του γίνεται το αντίστοιχο κύκλωμα του 3.1.2.

Το κύκλωμα παραγωγής μερικών γινομένων της μορφής NR4SD, περιγράφεται από τις παρακάτω λογικές σχέσεις, για το i-οστό bit των διανυσμάτων εισόδου:

Για $i = 0$:

$$pp\_tm\_out[0] = \overline{\overline{(a[0] \wedge oneplus[0])} \wedge \overline{((a[0] \vee oneminus[0]) \vee twominus[0])}}$$

Για $0 < i < (n/2-1)$:

$$pp\_tm\_out[i] = \overline{\overline{(a[i] \wedge oneplus[i])} \wedge \overline{((a[i] \vee oneminus[i]) \vee (a[i-1] \vee twominus[i]))}}$$

Για $i = n$:

$$pp\_tm\_out[n] = ((a[n] \wedge oneplus[i]) \wedge \overline{\overline{(a[n] \wedge oneplus[i])} \wedge \overline{(a[n-1] \vee oneminus[i])} \vee \overline{(a[n-1] \vee twominus[i])}})$$

Τα παραπάνω μερικά γινόμενα που παράγονται συγκεντρώνονται σε ένα διάνυσμα, με τους απαραίτητους διορθωτικούς όρους και τις επεκτάσεις πρόσημου, και οδηγούνται στον δεντρικό συμπιεστή Wallace (4:2 Adder), που περιγράφεται στο 2.4.1.

3.3 Σχήμα επανακωδικοποίησης Αφαίρεσης-Πολλαπλασιασμού

Στην συνέχεια θα περιγράψουμε ένα σχήμα επανακωδικοποίησης για την πράξη Αφαίρεσης-Πολλαπλασιασμού, όπως αυτή περιεγράφηκε στο [4]. Βασίζεται στη μονάδα Πρόσθεσης-Πολλαπλασιασμού του [4], όπου με τις κατάλληλες αλλαγές στις δομικές του μονάδες μπορεί να μετατραπεί σε μονάδα Αφαίρεσης-Πολλαπλασιασμού.

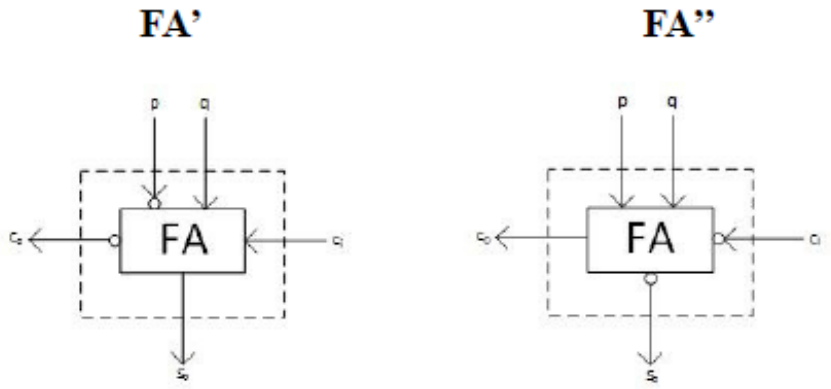
Η μονάδα αυτή δέχεται σαν είσοδο δύο αριθμούς σε μορφή συμπληρώματος ως προς 2, και παράγει στην έξοδό της το αποτέλεσμα της πράξης σε μορφής ψηφίων Modified Booth. Η πράξη που θα πραγματοποιήσουμε είναι Z=X·(A-B). Η συμβατική υλοποίηση μιας μονάδας Αφαίρεσης-Πολλαπλασιασμού απαιτεί την οδήγηση δύο εισόδων, Α και Β αντεστραμμένη, σε ένα αθροιστή όπου το κρατούμενο εισόδου είναι μονάδα, 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 διαδοθούν 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 Lookahead 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 Α-Β επανακωδικοποιείται απευθείας στην Modified Booth μορφή. Η συγχωνευμένη μονάδα Αφαίρεσης-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 Διαφοράς σε Modified Booth, η διαφορά δύο συνεχόμενων bits της εισόδου Α(a_{2j}, a_{2j+1}) με τα δύο συνεχόμενα bits της εισόδου Β(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. 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 14(α) τον αθροιστή FA’ και στο 14(β) τον αθροιστή FA’’ με την αξία των ψηφίων τους:



$$c_0 = ((\overline{p} \vee q) \wedge c_i) \vee (\overline{p} \wedge q) \qquad c_0 = ((p \vee q) \wedge \overline{c_i}) \vee (p \wedge q)$$
$$s = p \oplus q \oplus c_i \qquad s = p \oplus q \oplus c_i$$

Σχήμα 14: Προσημασμένοι πλήρης αθροιστές (α) FA’ και (b) FA’’

Στην συνέχεια παρουσιάζονται οι πίνακες αληθείας των παραπάνω τροποποιημένων αθροιστών.

Πίνακας 7: Πίνακας αληθείας 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

όπου το Output Value ορίζεται ως εξής:

$$Output\ \ Value = -2 \cdot c_0 + s = -p + q + c_i$$

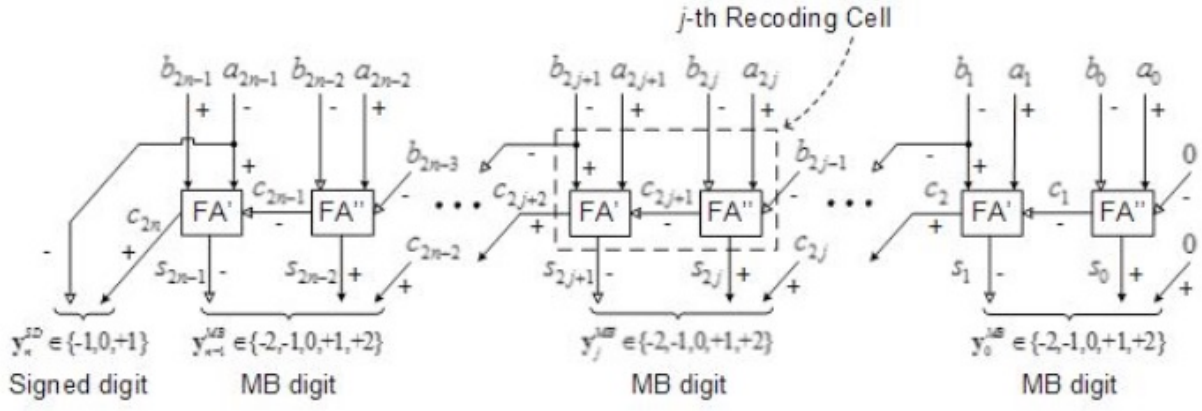
Πίνακας 8: Πίνακας αληθείας FA''

Inputs			Output Value[3]	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

όπου το Output Value ορίζεται ως εξής:

$$Output\ \ Value = 2 \cdot c_0 + s = p + q - c_i$$

Η παραπάνω μονάδες οδηγούνε στο σχήμα επανακωδικοποίησης το οποίο μας παράγει το επιθυμητό αποτέλεσμα για την παραγωγή της διαφοράς δύο αριθμών σε μορφή συμπληρώματος ως προς 2, απευθείας σε μορφής Modified Booth ψηφίων.



Σχήμα 15: Επανακωδικοποιητής Αφαίρεσης-Πολλαπλασιασμού

Η μονάδα που περιεγράφηκε παραπάνω είναι πολύ σημαντική στην εφαρμογή του αλγόριθμου του Gauss για τον μιγαδικό πολλαπλασιασμό όπως θα φανεί στην περιγραφή της αντίστοιχης μονάδας στην συνέχεια.

3.4 Μονάδα μιγαδικού πολλαπλασιασμού

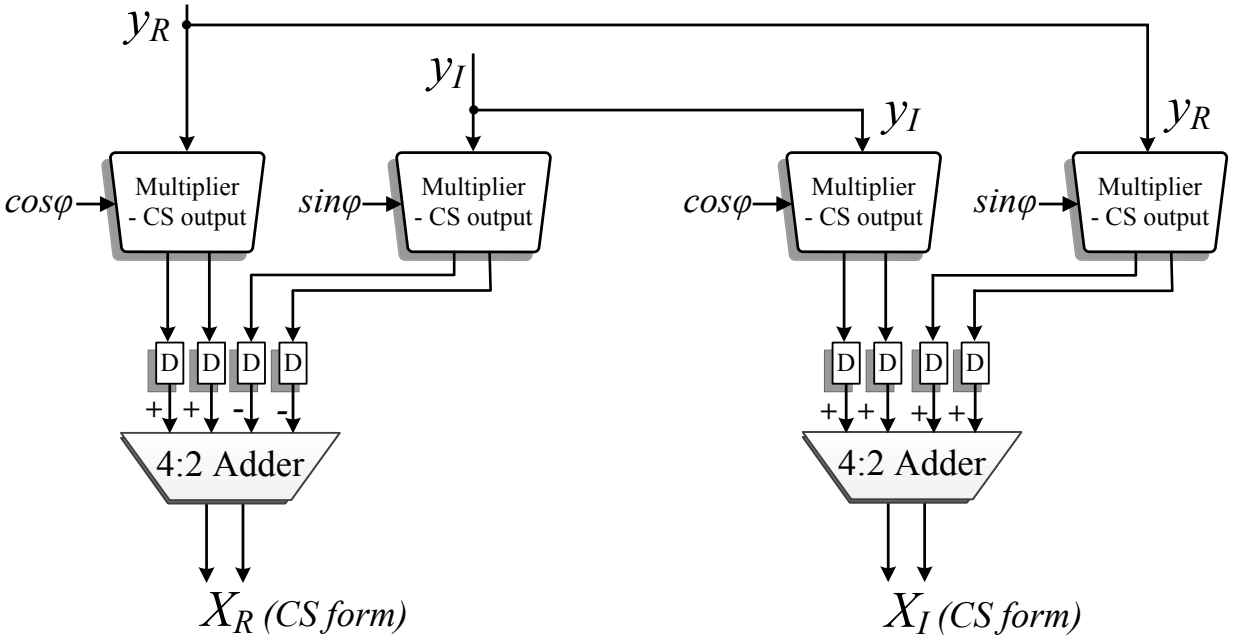
Στην παρούσα εργασία έχουμε σχεδιάσει και υλοποιήσει δύο διαφορετικές μονάδες για την πράξη του πολλαπλασιασμού ανάμεσα σε δύο μιγαδικούς αριθμούς. Στην συνέχεια θα παρουσιάσουμε τις δύο αυτές μονάδες που διαφοροποιούνται στον αλγόριθμο που χρησιμοποιούν, τον συμβατικό αλγόριθμο όπως αυτός περιεγράφηκε στο 2.3.1, και τον αλγόριθμο του Gauss που περιεγράφηκε στο 2.3.2.

3.4.1 Συμβατικός Μιγαδικός Πολλαπλασιαστής

Ο Συμβατικός Μιγαδικός Πολλαπλασιαστής βασίζεται στον συμβατικό αλγόριθμο για τον μιγαδικό πολλαπλασιασμό, όπως τον περιγράψαμε στο 2.3.1, και φαίνεται στο σχήμα 16. Ο μιγαδικός πολλαπλασιαστής είναι μέρος της μονάδας υπολογισμού Butterfly, όπως είδαμε στην περιγραφή της μονάδας αυτής στο 2.1.2, όπου ο ένας όρος είναι σταθερός αριθμός και αντιστοιχεί στο ημίτονο και συνημίτονο, που χρησιμοποιείται για τον αλγόριθμο FFT. Έτσι, ο υπολογισμός που φαίνεται παρακάτω, στο σχήμα 16, περιγράφεται από την παρακάτω σχέση:

$$(y_R + j \cdot y_i) \cdot (\cos\varphi + j \cdot \sin\varphi) = X_R + j \cdot X_I$$

όπου οι αριθμοί $y_R, y_i, \cos\varphi, \sin\varphi, X_R$ και X_I είναι πραγματικοί αριθμοί.



Σχήμα 16: Συμβατικός Μιγαδικός Πολλαπλασιαστής

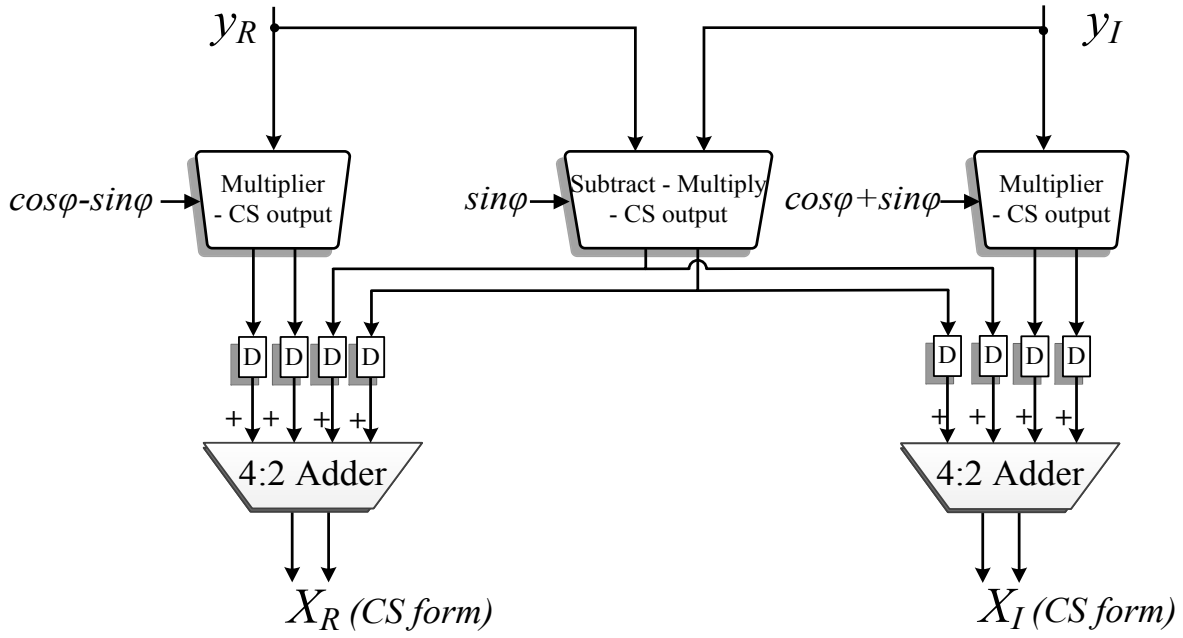
Στο παραπάνω σχήμα, βλέπουμε πως το κύκλωμα περιέχει τις μονάδες *Multiplier-CS output* και *4:2 Adder*. Η μονάδα *4:2 Adder*, είναι ο δεντρικός συμπιεστής που περιεγράφηκε στο 2.4.1 και η μονάδα *Multiplier-CS output* κάνει χρήση των πολλαπλασιαστών που περιγράψαμε στα 3.1 και 3.2. Ανάμεσα στο επίπεδο των πολλαπλασιαστών και των 4:2 Adders, έχουμε προσθέσει ένα επίπεδο pipeline, με χρήση καταχωρητών, για να μειώσουμε το κρίσιμο μονοπάτι και να έχουμε καλύτερη απόδοση.

Με αυτό τον τρόπο καταλήξαμε σε δύο διαφορετικούς μιγαδικούς πολλαπλασιαστές που χρησιμοποιούν το συμβατικό αλγόριθμο για τον μιγαδικό πολλαπλασιασμό, CM_Conv_1 και CM_Conv_2, που στο εσωτερικό τους χρησιμοποιούν πολλαπλασιαστές Modified Booth και NR4SD, αντίστοιχα.

Τέλος, αξίζει να σημειωθεί πως παντού κάνουμε χρήση Carry-Save αριθμητικής με σκοπό την καλύτερη απόδοση του τελικού κυκλώματος. Η χρήση της Carry-Save αριθμητικής έχει γίνει ομοιόμορφα σε όλα τα κυκλώματα που σχεδιάστηκαν στην παρούσα εργασία, όπως και το επίπεδο των pipeline καταχωρητών, ούτως ώστε οι συγκρίσεις που θα γίνουν να μας προσφέρουν ακριβή συμπεράσματα, βάσει των διαφορών των επιμέρους μονάδων που χρησιμοποιήσαμε, και των δύο αλγόριθμων για τον μιγαδικό πολλαπλασιασμό.

3.4.2 Gauss Μιγαδικός Πολλαπλασιαστής

Με την ίδια λογική που υλοποιήθηκαν τα κυκλώματα του συμβατικού μιγαδικού πολλαπλασιαστή, σχεδιάσαμε τον Gauss μιγαδικό πολλαπλασιαστή, η αρχιτεκτονική του οποίου φαίνεται στο διάγραμμα του σχήματος 17.



Σχήμα 17: Gauss Μιγαδικός Πολλαπλασιαστής

Όπως βλέπουμε στο σχήμα 17, το κύκλωμα περιέχει τις μονάδες *Multiplier-CS output* και *4:2 Adder*, όπως και στον συμβατικό μιγαδικό πολλαπλασιαστή, και ομοίως προσθέσαμε το pipeline επίπεδο καταχωρητών, και ακολουθήθηκε η Carry-Save αριθμητική.

Από τα παραπάνω, προέκυψαν δύο ακόμα μονάδες μιγαδικού πολλαπλασιαστή, με ονόματα CM_Gauss_1 και CM_Gauss_2, όπου γίνεται χρήση της αρχιτεκτονικής του σχήματος 17, με την διαφορά στην χρήση της μονάδας *Multiplier-CS output*, όπου κάναμε χρήση πολλαπλασιαστών Modified Booth και NR4SD, αντίστοιχα. Τέλος, η μονάδα

Subtract-Multiply-CS output, είναι υλοποιημένη σύμφωνα με το Σχήμα επανακωδικοποίησης Αφαίρεσης-Πολλαπλασιασμού, όπως αυτή περιεγράφηκε στο 3.3.

3.5 Υλοποιήσεις μονάδων υπολογισμού Butterfly

Στην ενότητα αυτή, θα παραθέσουμε τις τέσσερις μονάδες υπολογισμού Butterfly, οι οποίες, με βάση τις μονάδες που περιεγράφηκαν παραπάνω, αποτελούν το αντικείμενο της έρευνάς μας και στο επόμενο κεφάλαιο θα συγκριθούν πειραματικά.

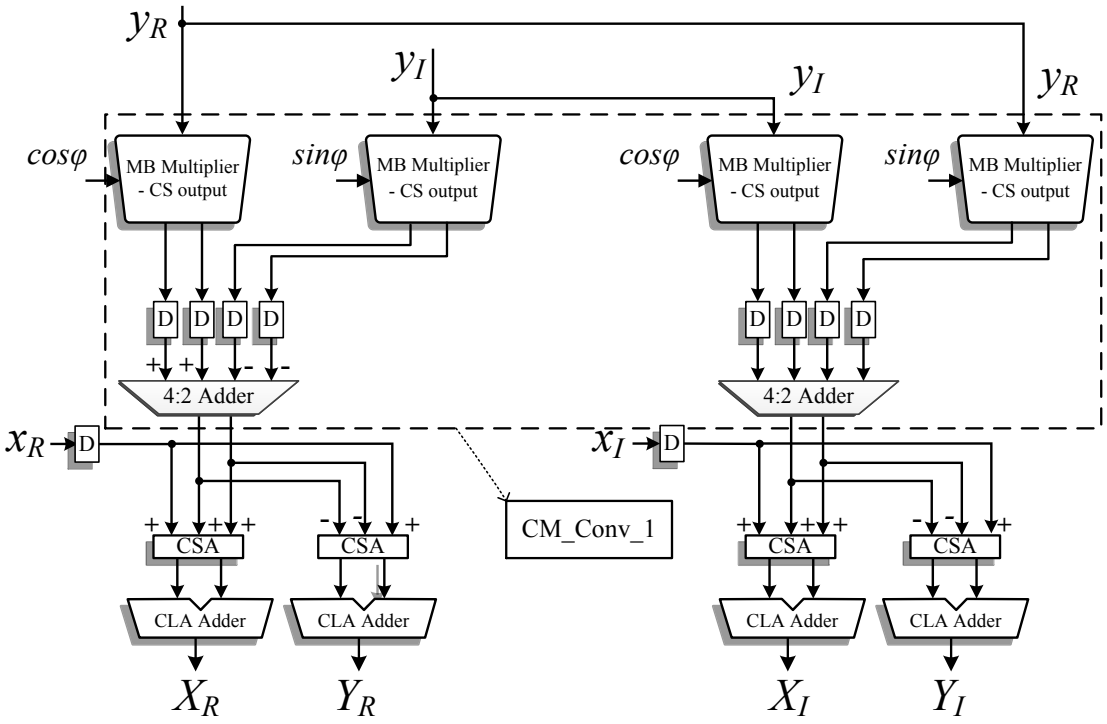
Οι τέσσερις μονάδες αυτές, είναι οι παρακάτω:

1. Μονάδα υπολογισμού Butterfly, Συμβατικού Αλγόριθμου, με χρήση κωδικοποίησης Modified Booth (*BCU_Conv_1*).
2. Μονάδα υπολογισμού Butterfly, Συμβατικού Αλγόριθμου, με χρήση κωδικοποίησης NR4SD (*BCU_Conv_2*).
3. Μονάδα υπολογισμού Butterfly, Αλγόριθμου του Gauss, με χρήση κωδικοποίησης Modified Booth (*BCU_Gauss_1*).
4. Μονάδα υπολογισμού Butterfly, Αλγόριθμου του Gauss, με χρήση κωδικοποίησης NR4SD (*BCU_ Gauss _1*).

Στην συνέχεια, παραθέτουμε τα διαγράμματα των παραπάνω τεσσάρων μονάδων, όπως αυτά υλοποιήθηκαν.

3.5.1 Μονάδα υπολογισμού Butterfly, Συμβατικού Αλγόριθμου, με χρήση κωδικοποίησης Modified Booth (*BCU_Conv_1*).

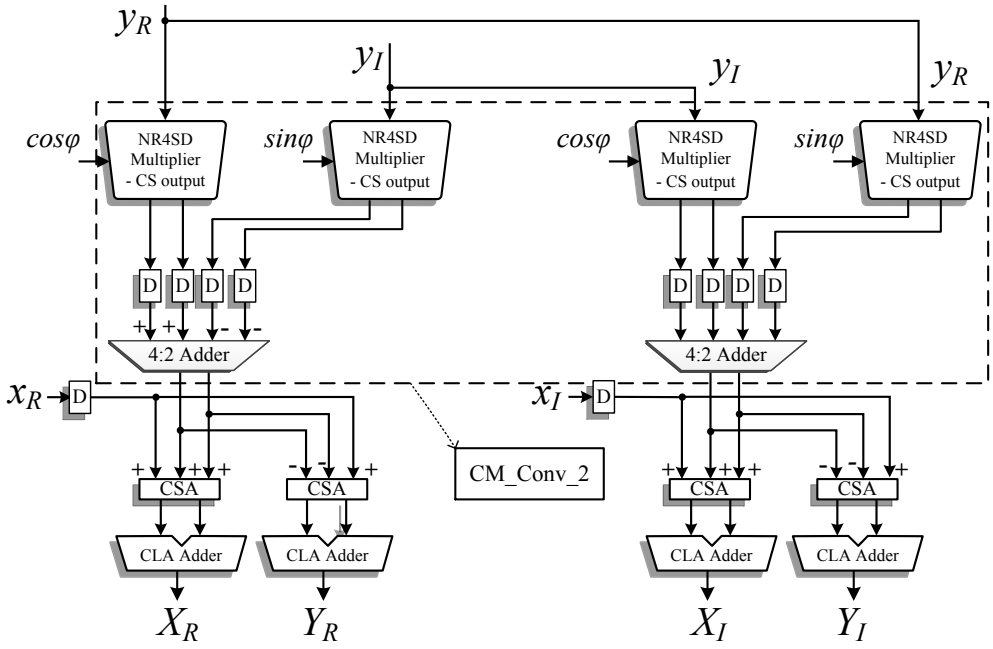
Η παρακάτω υλοποίηση είναι απλούστερη σχεδίαση, η οποία χρησιμοποιήθηκε ως μέτρο σύγκρισης των δύο υλοποιήσεων που κάνουν χρήση του συμβατικού αλγόριθμου, όπου θα συγκρίνουμε την απόδοση των δύο μονάδων για να δούμε πως επηρεάζεται το συνολικό κύκλωμα, από την χρήση των δύο διαφορετικών κωδικοποιήσεων για τον πολλαπλασιασμό πραγματικών αριθμών.



Σχήμα 18: Διάγραμμα υλοποίησης BCU_Conv_1

3.5.2 Μονάδα υπολογισμού Butterfly, Συμβατικού Αλγόριθμου, με χρήση κωδικοποίησης NR4SD (*BCU_Conv_2*).

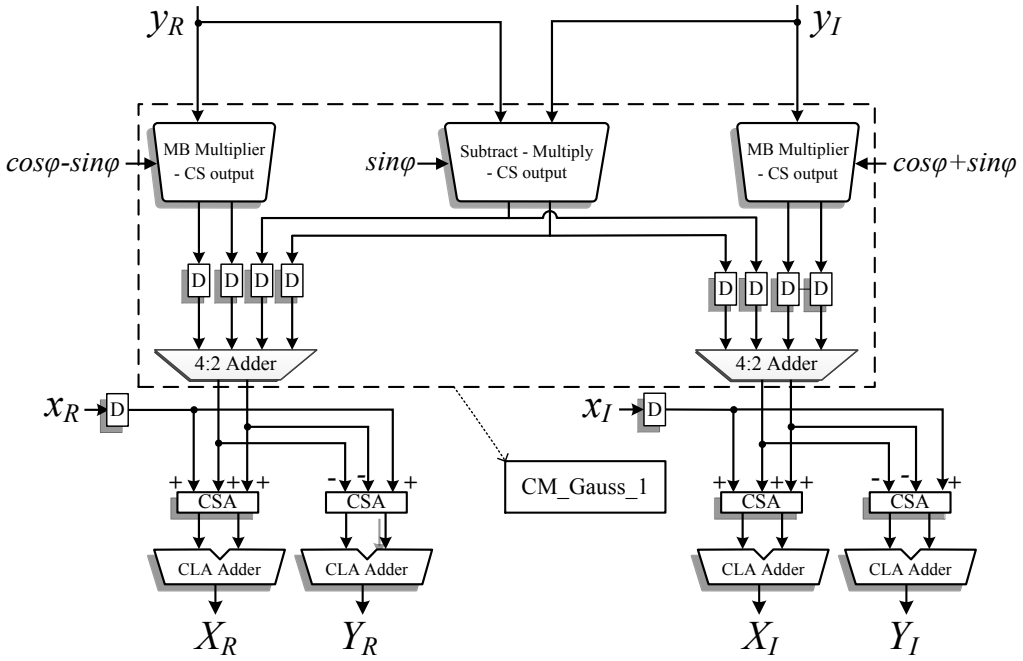
Στο επόμενο σχήμα, βλέπουμε την υλοποίηση BCU_Conv_2, με χρήση της κωδικοποίησης NR4SD, για τον πολλαπλασιαστή πραγματικών αριθμών.



Σχήμα 19: Διάγραμμα υλοποίησης BCU_Conv_2

3.5.3 Μονάδα υπολογισμού Butterfly, Αλγόριθμου του Gauss, με χρήση κωδικοποίησης Modified Booth (*BCU_Gauss_1*).

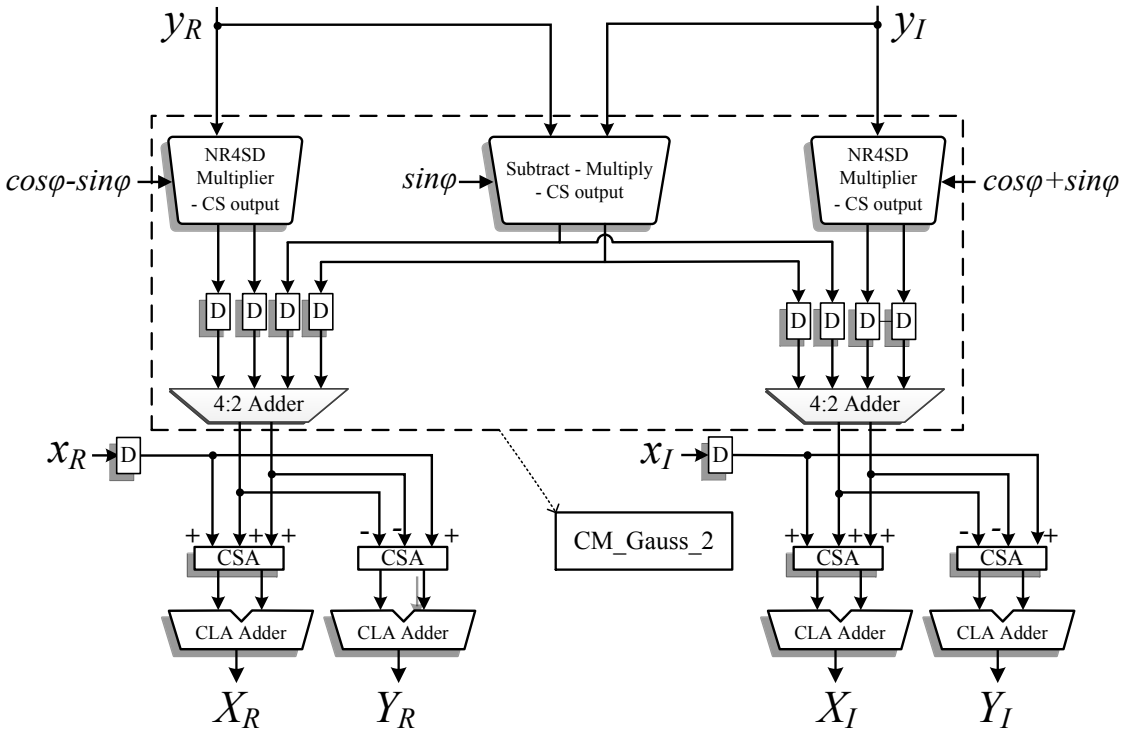
Στην συνέχεια παραθέτουμε το διάγραμμα της μονάδας BCU_Gauss_1, στο σχήμα 20, με χρήση του μιγαδικού πολλαπλασιαστή CM_Gauss_1, όπου χρησιμοποιείται η κωδικοποίηση Modified Booth, και περιεγράφηκε στο 3.4.1.



Σχήμα 20: Διάγραμμα υλοποίησης BCU_Gauss_1

3.5.4 Μονάδα υπολογισμού Butterfly, Αλγόριθμου του Gauss, με χρήση κωδικοποίησης NR4SD (*BCU_ Gauss _2*).

Τέλος, στο σχήμα 21, βλέπουμε το διάγραμμα της μονάδας BCU_Gauss_2, όπου ο μιγαδικός πολλαπλασιαστής έχει υλοποιηθεί με την μονάδα CM_Gauss_2, όπως αυτή περιεγράφηκε στο 3.4.2.



Σχήμα 21: Διάγραμμα υλοποίησης BCU_Gauss_2

4. ΣΥΓΚΡΙΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

4.1 Οργάνωση πειραμάτων

Οι τοπολογίες που παρουσιάστηκαν στο κεφάλαιο 3.5, περιγράφηκαν σε γλώσσα Verilog. Όσων αφορά την υλοποίηση των τοπολογιών αυτών σε ASIC συντέθηκαν από το Synopsys Design Compiler, κάνοντας χρήση της βιβλιοθήκης κελιών TSMC 90nm. Τα κυκλώματα που παρήχθησαν, προσομοιώθηκαν από το περιβάλλον Modelsim, για επιβεβαίωση της ορθής λειτουργίας τους.

Από το περιβάλλον του Design Compiler [10], εξήχθησαν οι τιμές καθυστέρησης και επιφάνειας κάθε κυκλώματος, ενώ από το περιβάλλον του Synopsys Primepower [12] (με βάση το .vcd αρχείο που προέκυπτε από κάθε προσομοίωση του Modelsim [11]) 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 = 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 κυκλώματα σε συνθήκες χαλαρότερων περιορισμών ρολογιού.

4.2 Πειραματική ανάλυση

Στην πειραματική ανάλυση θα παρουσιάσουμε τα αποτελέσματα που προέκυψαν από τις μετρήσεις των εργαλείων, στα παραπάνω κυκλώματα.

Αρχικά, θα παρουσιαστούν συγκεντρωτικά αποτελέσματα για την καθυστέρηση και θα αναλυθούν τα αποτελέσματα. Στην συνέχεια, θα παρουσιαστούν και θα εκτιμηθούν τα αποτελέσματα των μετρήσεων για το κάθε μήκος λέξης. Τέλος, θα γίνει μία συνολική αποτίμηση των αποτελεσμάτων, μέσα από τις επιμέρους μετρήσεις για τα διάφορα μήκη λέξης, και θα εξαχθούν συμπεράσματα για το συνολική εικόνα που θα μας αποφέρουν τα παραπάνω αποτελέσματα.

4.2.1 Συγκεντρωτικά αποτελέσματα

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

Πίνακας 9: Πίνακας Χαμηλότερης Συχνότητα λειτουργίας

	Critical Path (*n* sec)				
Bit length	**16**	**24**	**32**	**48**	**64**
BCU_Conv_1	0.88	1.01	1.09	1.33	1.50
BCU_Conv_2	0.89	0.96	1.03	1.21	1.31
BCU_Gauss_1	0.99	1.11	1.17	1.33	1.50
BCU_Gauss_2	0.96	1.09	1.15	1.34	1.48

Στον πίνακα 9, βλέπουμε πως οι μονάδες που κάνουν χρήση του συμβατικού αλγόριθμου δίνουν καλύτερα αποτελέσματα από τις μονάδες που κάνουν χρήση του αλγόριθμου του Gauss. Για μήκη λέξης 16, 24, και 32 bits, οι μονάδες BCU_Conv_1 και BCU_Conv_2 έχουν κέρδος της τάξης του 10%, επί των BCU_Gauss_1 και BCU_Gauss_2. Όταν το μήκος λέξης είναι 48 και 64 bits, παρατηρούμε πως οι μονάδες BCU_Gauss_1 και BCU_Gauss_2 έχουν παρόμοια συμπεριφορά με την μονάδα BCU_Conv_1. Η μονάδα BCU_Conv_2, έχει παντού τα καλύτερα αποτελέσματα σε ό,τι αφορά την καθυστέρηση του κυκλώματος που συντέθηκε.

Για να κατανοήσουμε καλύτερα τα αποτελέσματα πρέπει να βασιστούμε στις μονάδες που περιεγράφηκαν στο προηγούμενο κεφάλαιο. Όλες οι υλοποιήσεις μπορούν να αποδομηθούν σε δύο επίπεδα. Πρώτο είναι το επίπεδο των πολλαπλασιαστών, έως το επίπεδο των καταχωρητών, και δεύτερο, το επίπεδο όπου τα αποτελέσματα του μιγαδικού πολλαπλασιασμού, σε μορφή Carry-Save, συνδυάζονται κατάλληλα με τις εισόδους της μονάδας για να παραχθεί το τελικό αποτέλεσμα.

Οι ειδοποιός διαφορά των δύο αρχιτεκτονικών, βασίζεται στην μονάδα *Subtract-Multiply-CS output*, η οποία προσφέρει κέρδος από την απαλοιφή του τέταρτου πολλαπλασιαστή, αλλά αυτό το κέρδος δεν αναμένεται να είναι σε καθυστέρηση, αλλά σε επιφάνεια και κατανάλωση. Η μονάδα αυτή ενώ κάνει την πράξη γρηγορότερα από ό,τι θα γινόταν από επιμέρους μονάδες, βλέπουμε πως δημιουργεί μεγαλύτερο κρίσιμο μονοπάτι και είναι ο λόγος που ο αλγόριθμος του Gauss δίνει μεγαλύτερη καθυστέρηση σε σχέση με τον συμβατικό αλγόριθμο, κάτι που συμβαδίζει με την περιγραφή της μονάδας στο 3.3.

Ωστόσο, βλέπουμε πως σε μεγαλύτερα μήκη λέξης, όπου το κύκλωμα παραγωγής μερικών γινομένων των Modified Booth πολλαπλασιαστών γίνεται πιο βεβαρυμμένο, οι υλοποιήσεις BCU_Gauss_1 και BCU_Gauss_2 έχουν την ίδια καθυστέρηση με την μονάδα BCU_Conv_1.

Από την ανωτέρω ανάλυση, βλέπουμε πως οι πολλαπλασιαστές NR4SD και το ελαχιστοποιημένο κύκλωμά τους για την παραγωγή μερικών γινομένων, καθιστά την μονάδα BCU_Conv_2, την καλύτερη υλοποίηση σε ό,τι αφορά την καθυστέρηση. Παρόλα αυτά, περιμένουμε πως οι αλγόριθμος του Gauss και οι μονάδες BCU_Gauss_1 και BCU_Gauss_2, θα μας δώσουν κέρδος σε επιφάνεια και κατανάλωση, σύμφωνα με την περιγραφή τους στο προηγούμενη κεφάλαιο.

Για να εξηγήσουμε καλύτερα τις διαφορές που θα προκύψουν θα πρέπει να ανατρέξουμε στις παραμέτρους που δώσαμε στον Design Compiler της Synopsys [10] 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, με σκοπό το κύκλωμα να έχει την βέλτιστη συμπεριφορά ως προς την καθυστέρηση. Έτσι στην συνέχεια θα συγκρίνουμε τις υλοποιήσεις, έχοντας υπόψιν ότι οι μετρήσεις έχουν ληφθεί για να επιτευχθεί ο καλύτερος δυνατός χρονισμός.

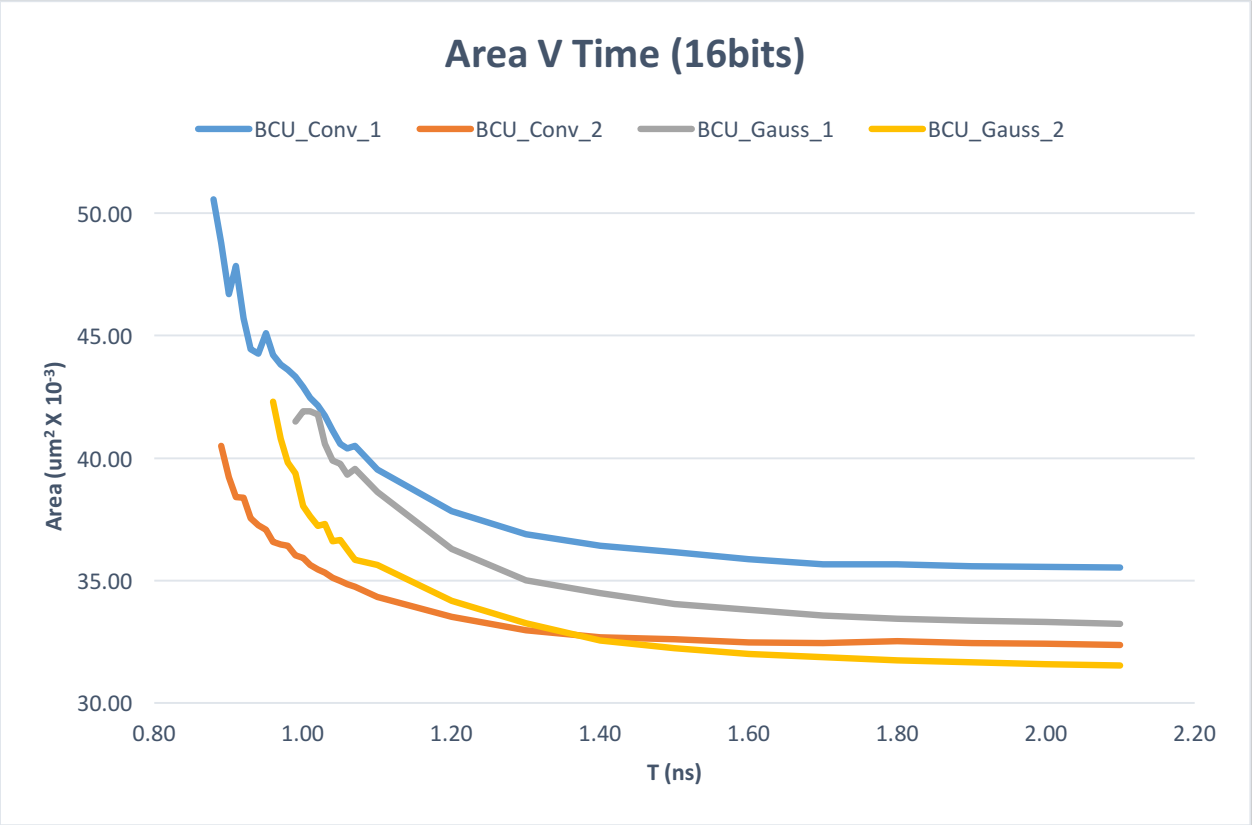
Για να εξετάσουμε τις μετρήσεις που έγιναν, θα τις χωρίσουμε ανάλογα με το μήκος λέξης των αριθμών στην είσοδο. Επίσης θα συγκρίνουμε και τις υλοποιήσεις μεταξύ τους, ξεχωριστά σε κάθε μήκος λέξης, αλλά και συνολικά για την συμπεριφορά τους από την χαμηλότερη συχνότητα ρολογιού μέχρι και τις συνθήκες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

4.2.2 Μήκος λέξης n = 16 bits

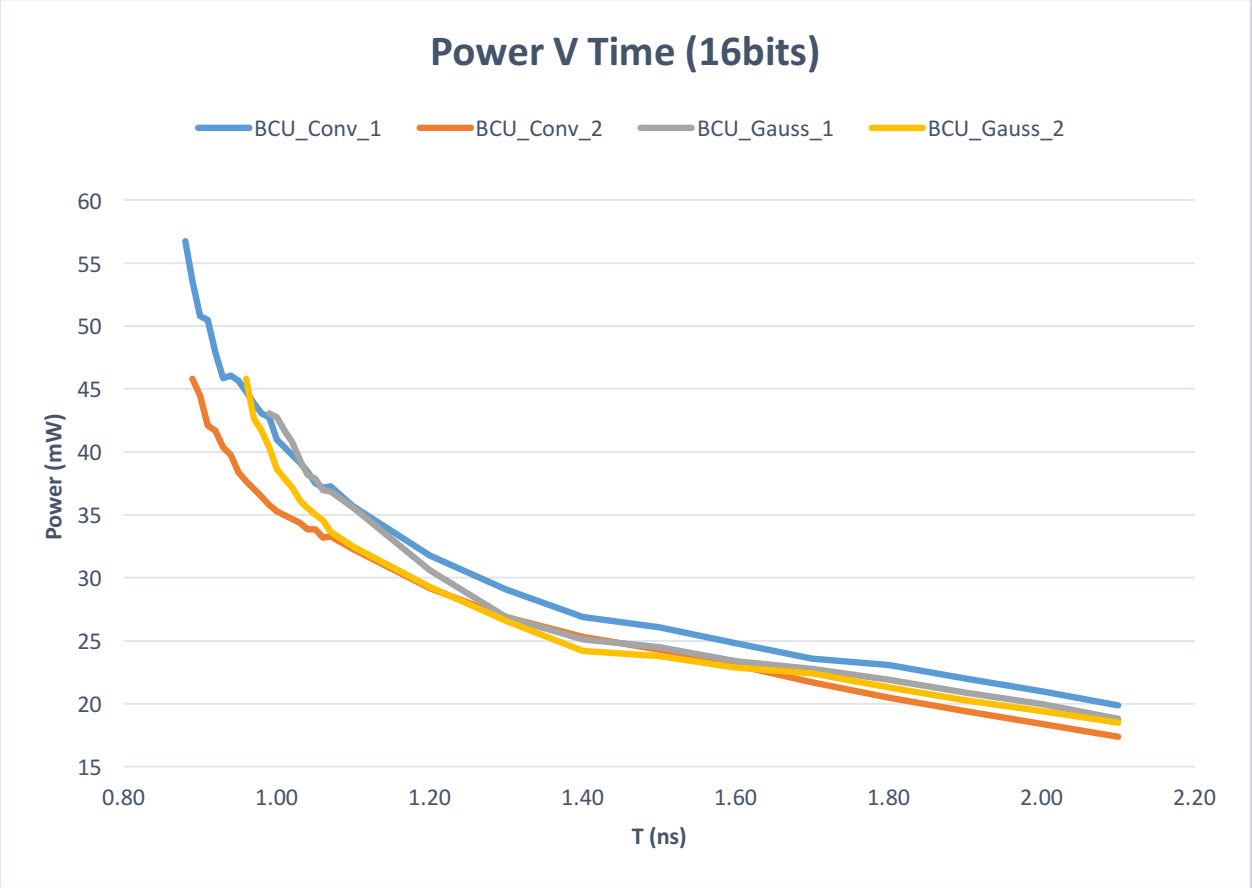
Στην συνέχεια παρουσιάζουμε τα αποτελέσματα που προκύπτουν από τις μετρήσεις για n = 16 bits, όπου έχουμε διαφορετικές γραφικές παραστάσεις, για την επιφάνεια του κυκλώματος σε συνάρτηση με το την περίοδο ρολογιού για το κύκλωμα μας, καθώς οι μετρήσεις που έγιναν, ακολούθησαν την περιγραφή που κάναμε στο 4.1.

Στο σχήμα 22, βλέπουμε γράφημα της επιφάνειας του κυκλώματος που συντέθηκε, συναρτήσει της περιόδου του ρολογιού και στο σχήμα 23 το γράφημα κατανάλωσης συναρτήσει του ρολογιού.

Το κύκλωμα BCU_Conv_2 δείχνει να έχει την μικρότερη επιφάνεια στο χαμηλότερο δυνατό ρολόι, αλλά οι υλοποιήσεις του αλγόριθμου του Gauss, ακολουθούν με μικρή διαφορά. Στο σχήμα 23, βλέπουμε πως ίδια είναι η εικόνα σε ό,τι αφορά την κατανάλωση. Αξίζει να σημειωθεί πως όταν βρισκόμαστε σε χαλαρότερες συνθήκες περιορισμού του ρολογιού, η μονάδα BCU_Gauss_2 συμπεριφέρεται καλύτερα από τα υπόλοιπα κυκλώματα.

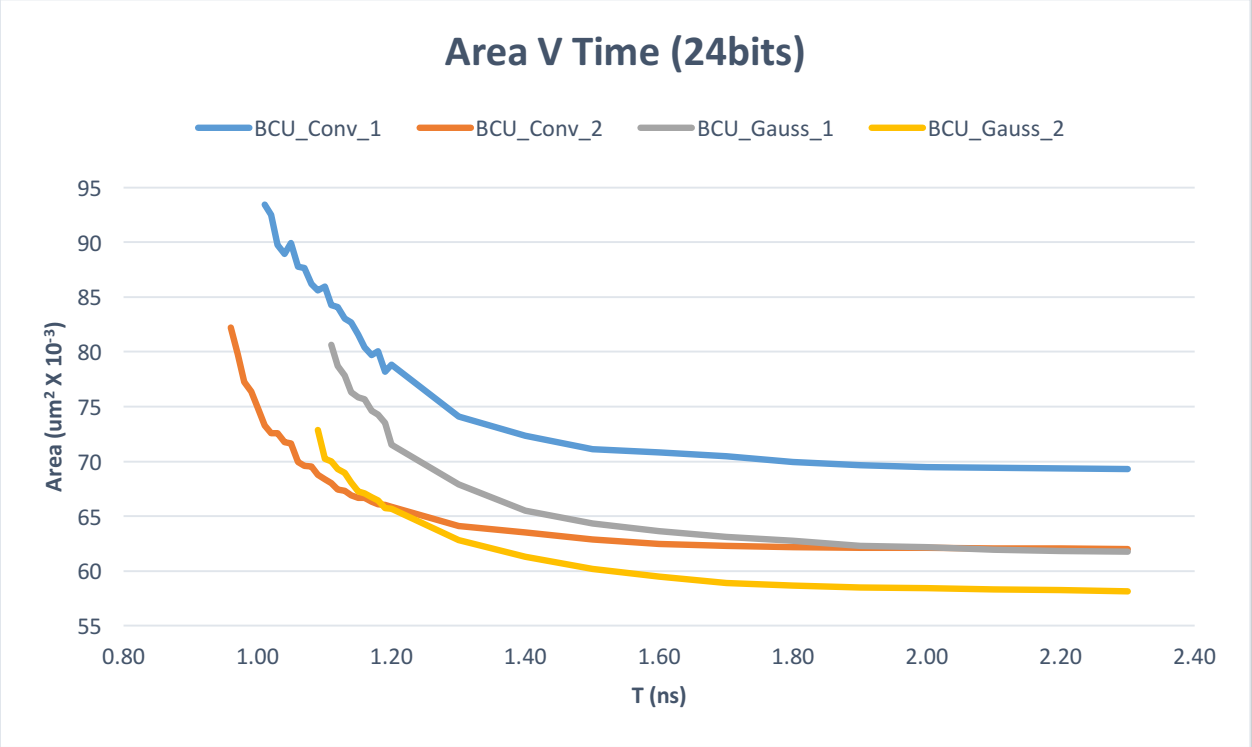


Σχήμα 22: Γράφημα επιφάνειας συναρτήσει περιόδου ρολογιού για n=16 bits



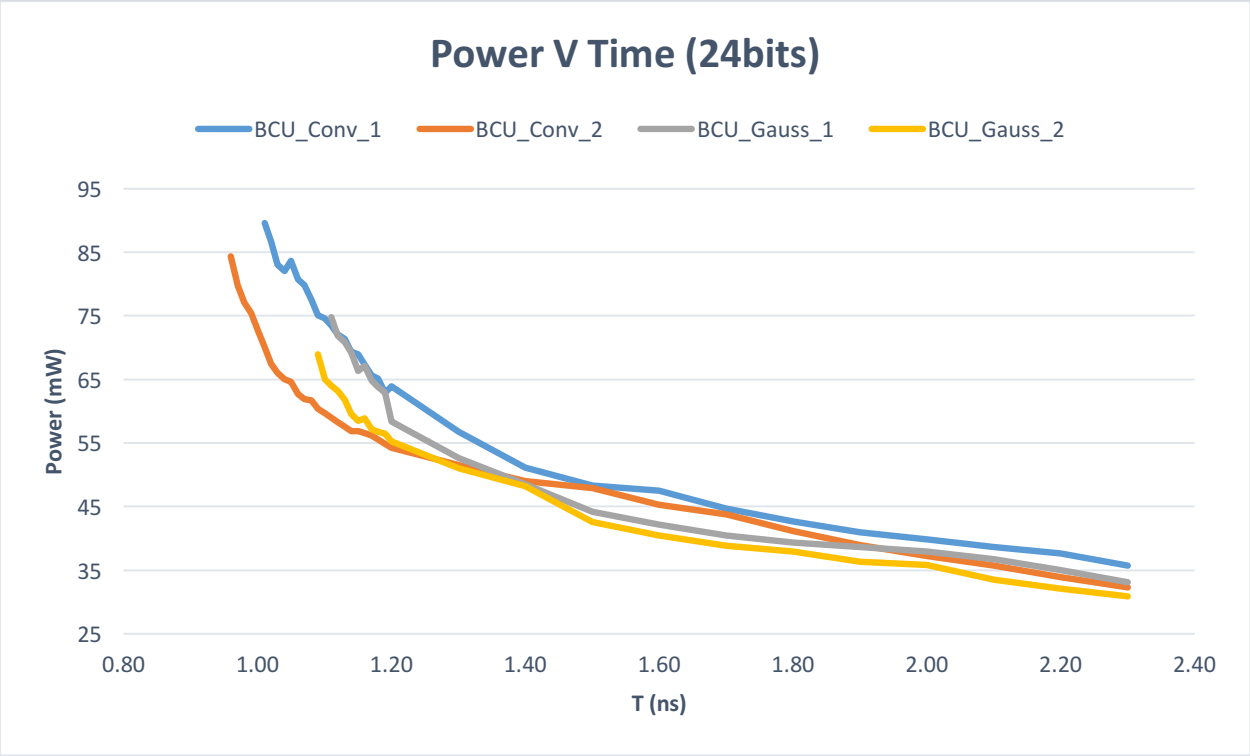
Σχήμα 23: Γράφημα κατανάλωσης συναρτήσει περιόδου ρολογιού για n=16 bits

4.2.1 Μήκος λέξης n = 24 bits



Σχήμα 24: Γράφημα επιφάνειας συναρτήσει περιόδου ρολογιού για n=24 bits

Για n=24 bits, στο σχήμα 24, βλέπουμε πως ο αλγόριθμος του Gauss, παρουσιάζει τα πρώτα του πλεονεκτήματα. Ενώ διατηρείται, η υπεροχή του συμβατικού αλγόριθμου σε ό,τι αφορά το κρίσιμο μονοπάτι, παρατηρούμε πως το κύκλωμα με την μικρότερη επιφάνεια και χαμηλότερη κατανάλωση, σε όλες τις περιοχές λειτουργίας, πλην των χαμηλότερων ρολογιών. Ομοίως παρακάτω, στο σχήμα 25 βλέπουμε πως οι μετρήσεις για την κατανάλωση, δίνουν στο αλγόριθμο του Gauss ένα πρώτο προβάδισμα.



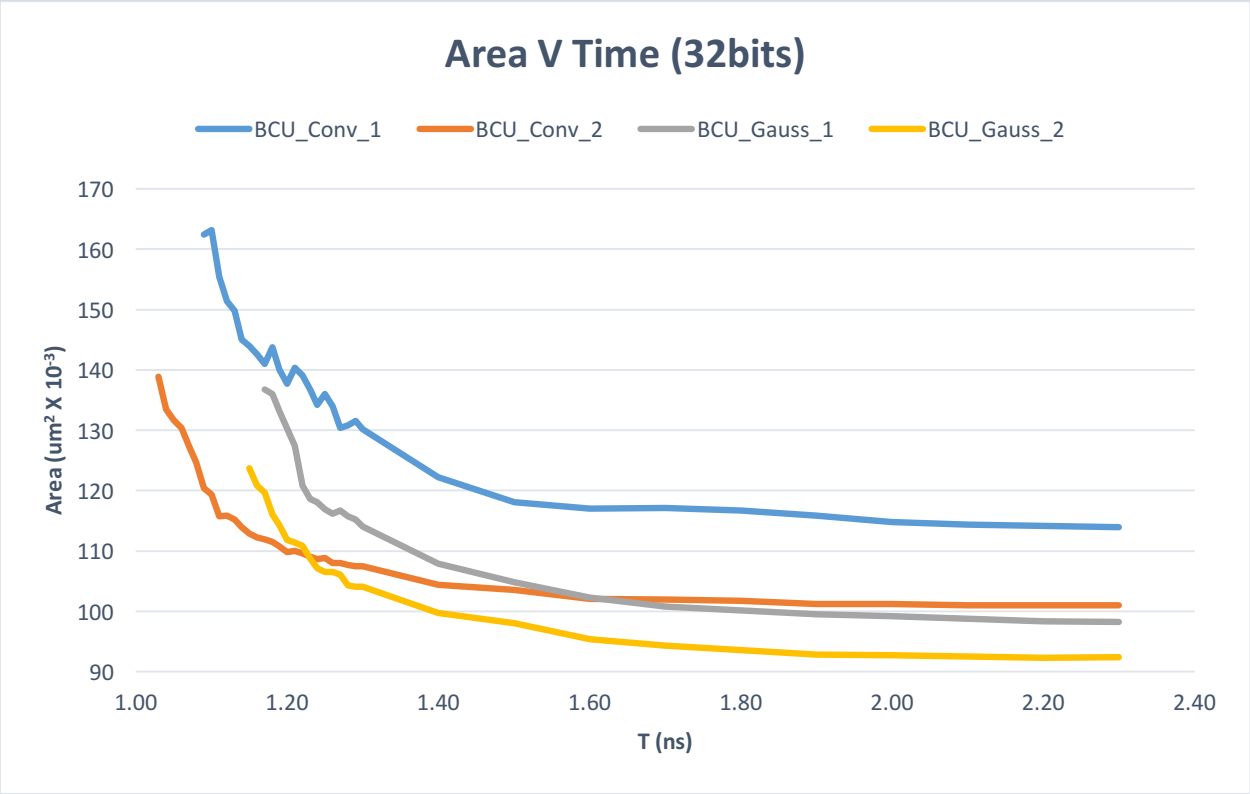
Σχήμα 25: Γράφημα κατανάλωσης συναρτήσει περιόδου ρολογιού για n=24 bits

4.2.2 Μήκος λέξης n = 32 bits

Μετά τα παραπάνω αποτελέσματα που παρουσιάστηκαν, σε συνδυασμό με την αναμενόμενη συμπεριφορά των κυκλωμάτων που συντέθηκαν, περιμένουμε πως τα 32 bits θα είναι σημείο καμπής για την ανάλυση των μετρήσεών μας. Καθώς το μέγεθος των κυκλωμάτων παραγωγής μερικών γινομένων αρχίζει να επηρεάζει το συνολικό κύκλωμα, βλέπουμε την επίδραση του ενός λιγότερου πολλαπλασιαστή στην μονάδα συνολικά.

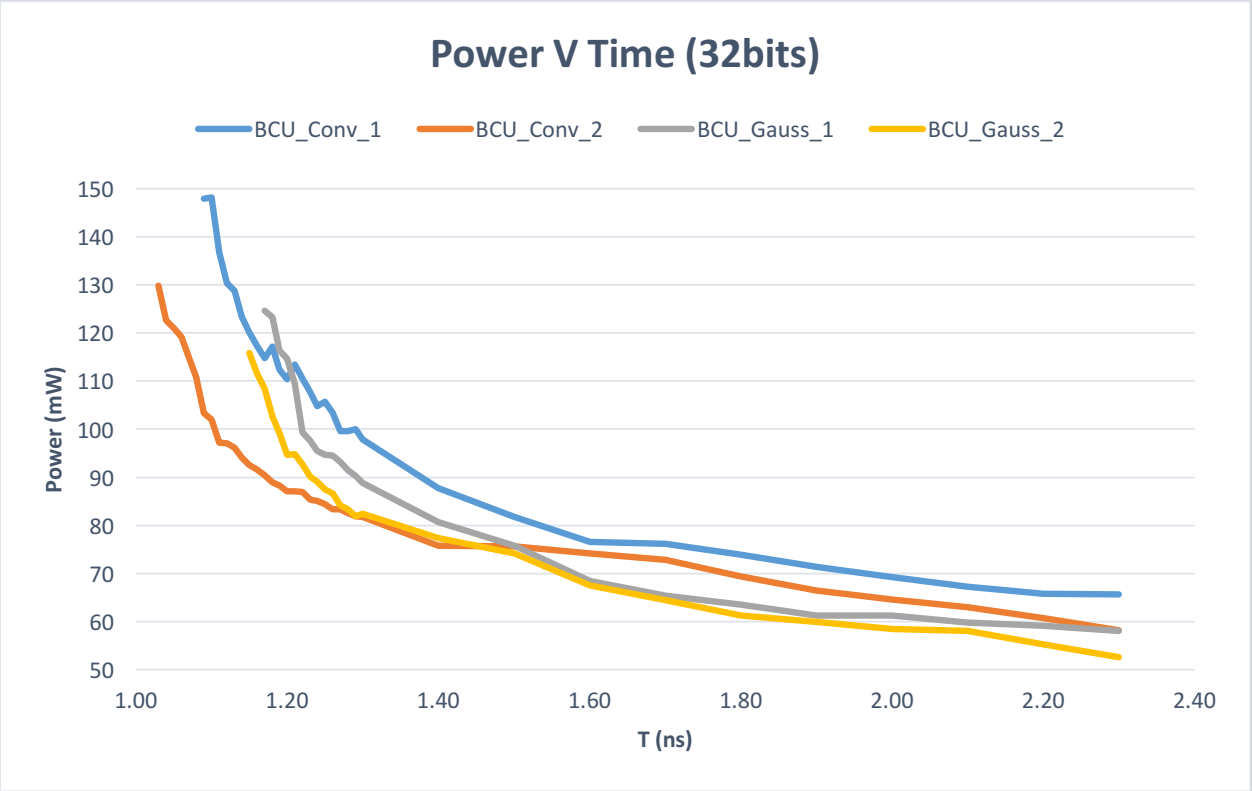
Όπως αναμενόταν, η απόδοση του BCU_Conv_2 είναι η βέλτιστη σε καθυστέρηση, αλλά η επιφάνεια και η κατανάλωση των κυκλωμάτων που χρησιμοποιούν τον αλγόριθμο του Gauss είναι χαμηλότερη από τις μονάδες που χρησιμοποιούν τον συμβατικό αλγόριθμο, όπως φαίνεται στα σχήματα 26 και 27, αντίστοιχα.

Η συμπεριφορά των κυκλωμάτων, στην χαμηλότερη περίοδο ρολογιού, μας δίνει την συμπεριφορά τους σε high performance συνθήκες. Ανάλογα με την εφαρμογή στην οποία σκοπεύουμε να εφαρμόσουμε τα κυκλώματα αυτά, μας υπαγορεύεται και η καταλληλότερη συχνότητα λειτουργία. Σε πολλές εφαρμογές, η λειτουργία στην χαμηλότερη δυνατή συχνότητα δεν είναι πρώτη προτεραιότητα. Όταν πρώτη προτεραιότητα είναι η χαμηλή κατανάλωση και η μικρή επιφάνεια του κυκλώματος είναι λογικό να γίνεται ένας συμβιβασμός, ώστε με μία μείωση της συχνότητας λειτουργίας να προκύψει μεγαλύτερη εξοικονόμηση στην κατανάλωση και στην επιφάνεια. Αυτό φαίνεται ότι με την χρήση του αλγόριθμου του Gauss μπορεί να επιτευχθεί και η διερεύνησή μας, θα επικεντρωθεί σε αυτό στην συνέχειά της.



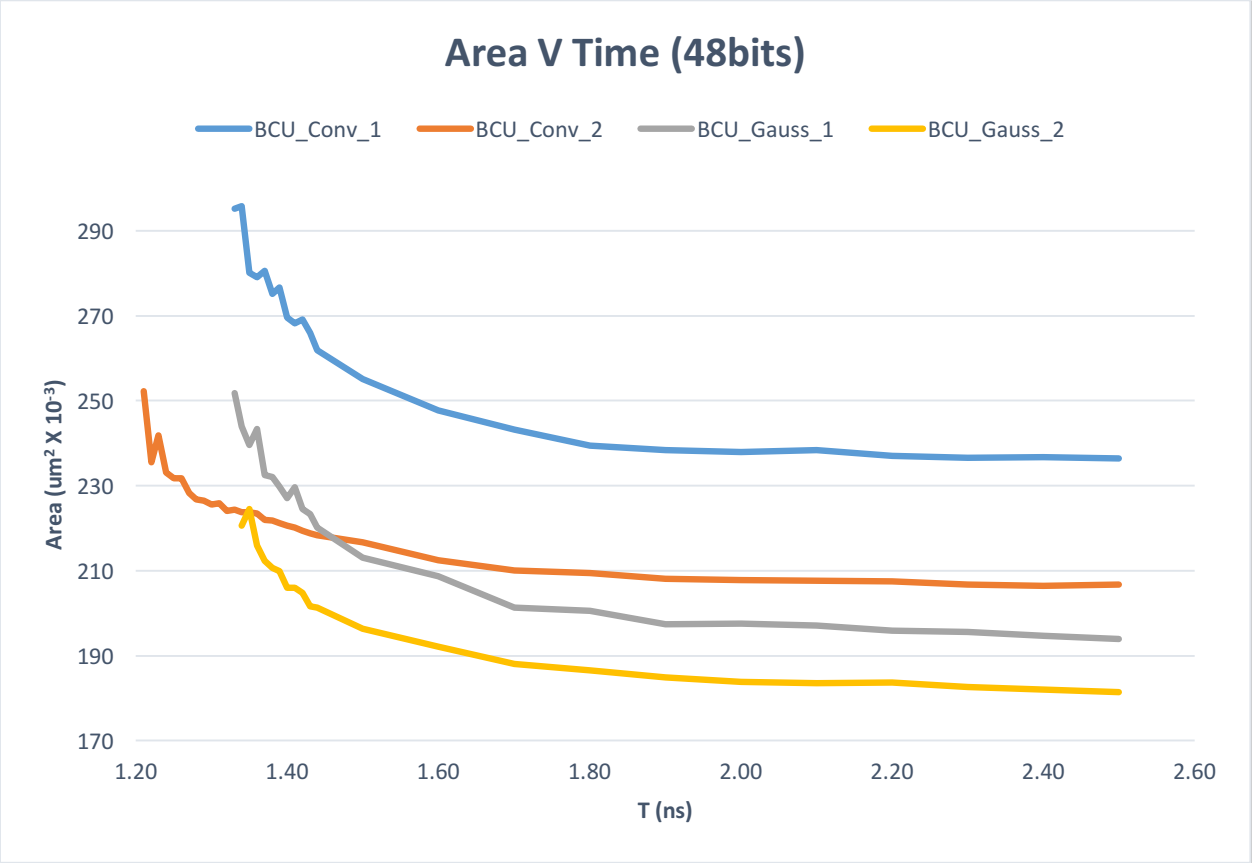
Σχήμα 26: Γράφημα επιφάνειας συναρτήσει περιόδου ρολογιού για n=32 bits

Στα σχήματα 26 και 27 βλέπουμε πως με μία αύξηση της περιόδου του ρολογιού των BCU_Gauss_1 και BCU_Gauss_2 κατά 5% καθιστά τα κυκλώματα ίδιας και καλύτερης απόδοσης με την BCU_Conv_2.



Σχήμα 27: Γράφημα κατανάλωσης συναρτήσει περιόδου ρολογιού για n=32 bits

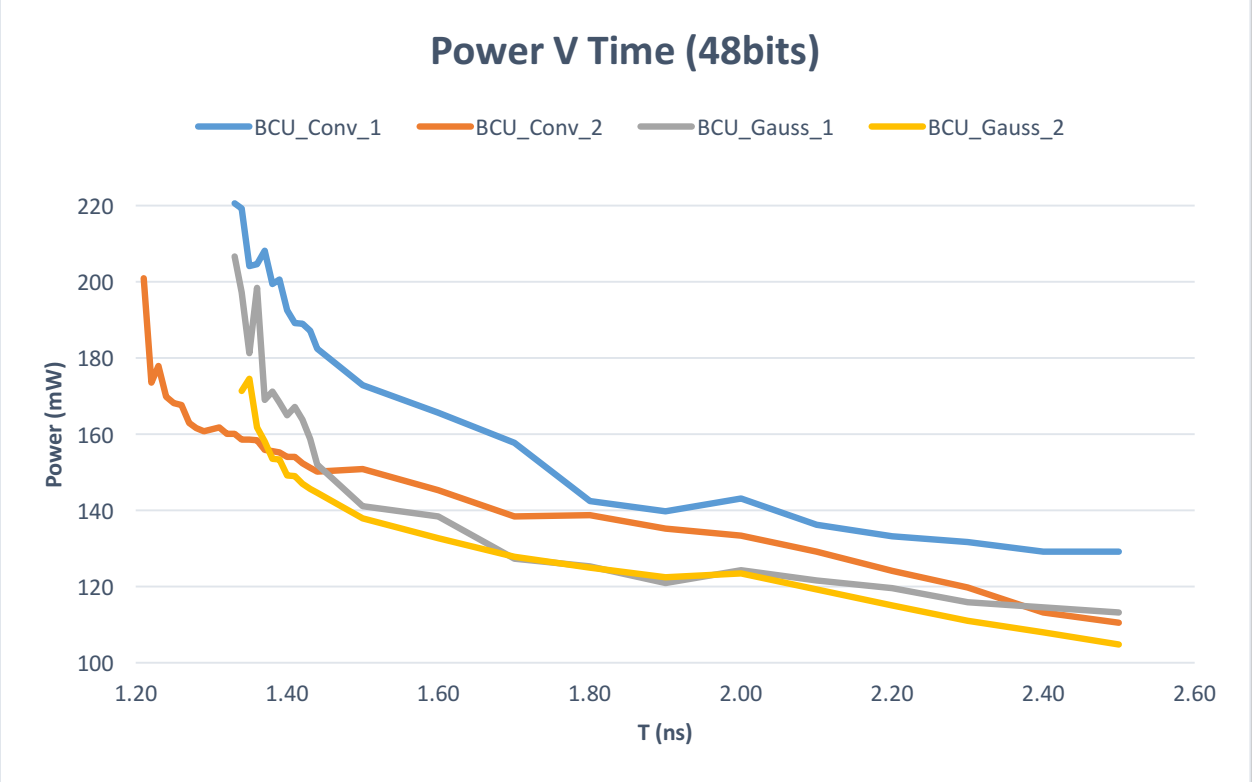
4.2.3 Μήκος λέξης n = 48 bits



Σχήμα 28: Γράφημα επιφάνειας συναρτήσει περιόδου ρολογιού για n=48 bits

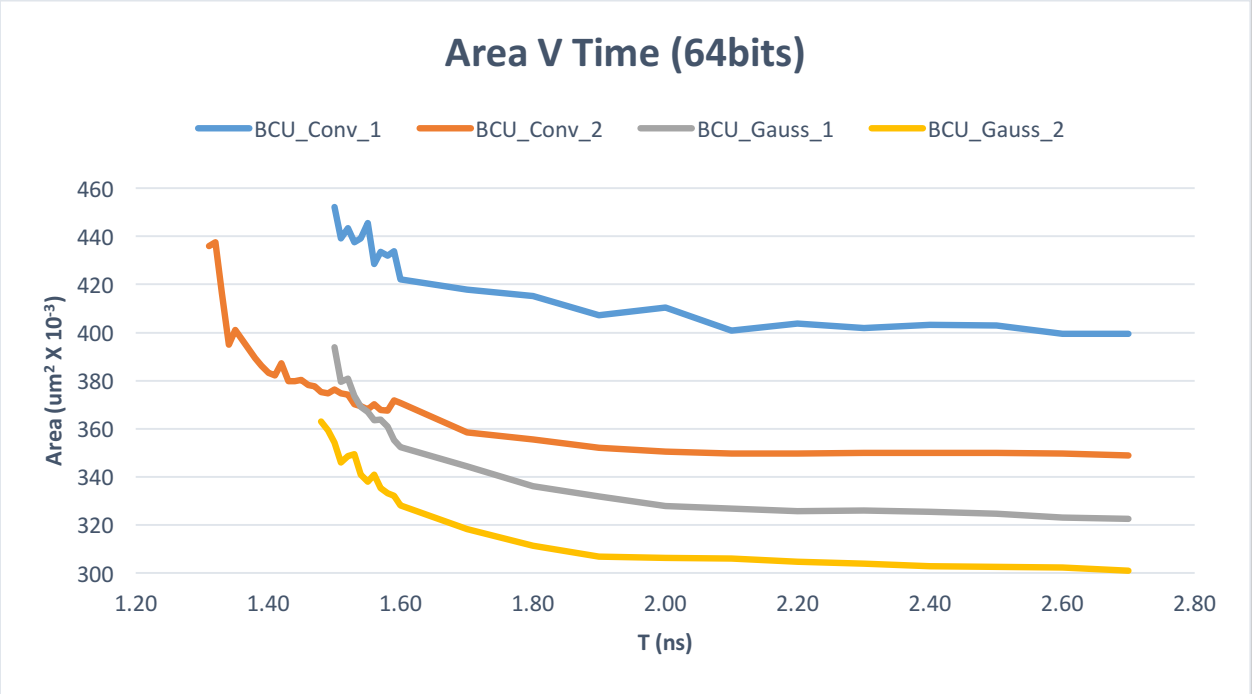
Στα παραπάνω σχήματα συνοψίζονται όσα περιεγράφηκαν παραπάνω, όπου η μονάδα BCU_Conv_2 παρουσιάζει την μικρότερη καθυστέρηση, ενώ η μονάδα BCU_Gauss_2 έχει την χαμηλότερη κατανάλωση και την μικρότερη επιφάνεια κυκλώματος. Έτσι βλέπουμε πως τα πλεονεκτήματα των πολλαπλασιαστών NR4SD στην καθυστέρηση, την επιφάνεια και την κατανάλωση των κυκλωμάτων είναι ανεξάρτητα του μήκους λέξης και αποφέρουν καλύτερα κυκλώματα. Επίσης μπορούμε να διακρίνουμε την χρησιμότητα του αλγόριθμου του Gauss για την παραγωγή κυκλωμάτων μικρής επιφάνειας και χαμηλής κατανάλωσης.

Αξίζει να παρατηρηθεί ότι σε χαλαρότερες συνθήκες περιορισμού του ρολογιού ο αλγόριθμος του Gauss, όπως εφαρμόστηκε στο BCU_Gauss_1, δίνει μικρότερο κύκλωμα με χαμηλότερη κατανάλωση από το κύκλωμα BCU_Conv_2, το οποίο σε συνθήκες αυστηρού περιορισμού του ρολογιού είναι βέλτιστο. Με αυτά υπόψιν, βλέπουμε πως αναδεικνύεται η χρησιμότητα του αλγόριθμου του Gauss, σε εφαρμογές όπου είναι σημαντική η χαμηλή κατανάλωση, και η μικρότερη δυνατή επιφάνεια του παραγόμενου κυκλώματος.

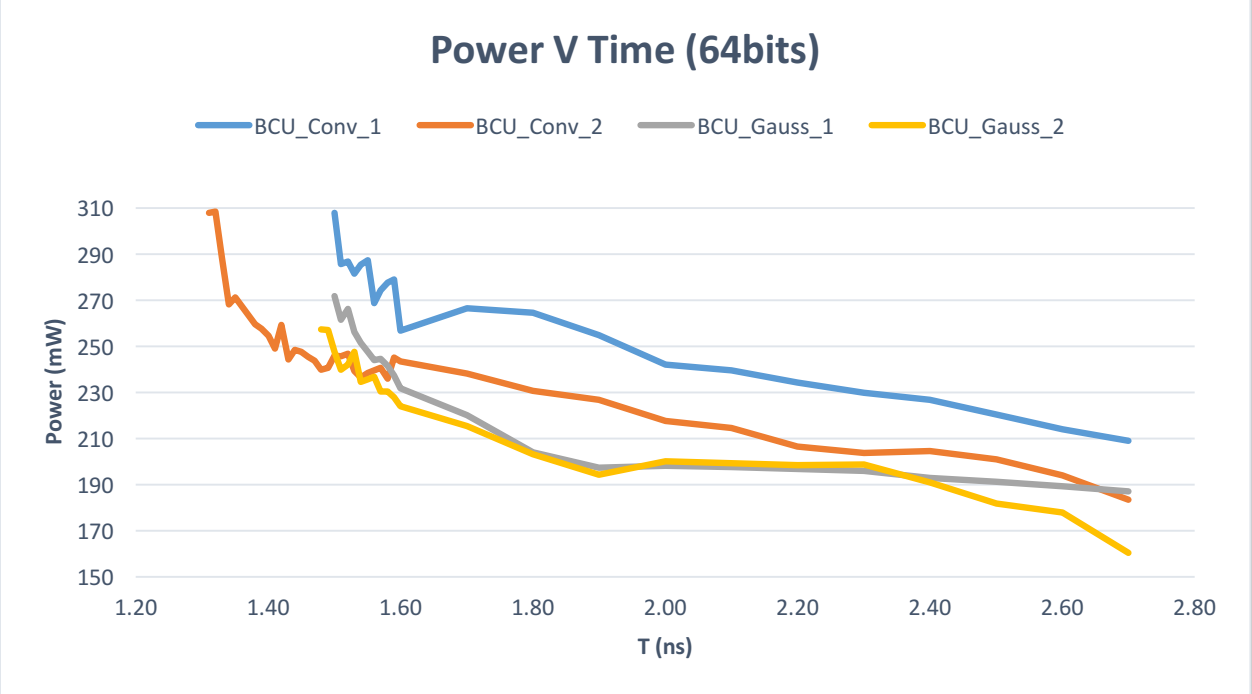


Σχήμα 29: Γράφημα κατανάλωσης συναρτήσει περιόδου ρολογιού για n=48 bits

4.2.4 Μήκος λέξης n = 64 bits



Σχήμα 30: Γράφημα επιφάνειας συναρτήσει περιόδου ρολογιού για n=64 bits



Σχήμα 31: Γράφημα κατανάλωσης συναρτήσει περιόδου ρολογιού για n=64 bits

Τα αποτελέσματα για n = 64 bits επιβεβαιώνουν τα συμπεράσματα που βγάλαμε στα μικρότερα μήκη λέξης, και βλέπουμε τα πλεονεκτήματα των NR4SD pre-encoded πολλαπλασιαστών, καθώς και τα πλεονεκτήματα του αλγόριθμου του Gauss για τον μιγαδικό πολλαπλασιασμό και την επίδρασή του στην μείωση επιφάνειας και κατανάλωσης έναντι αυτού του συμβατικού αλγόριθμου.

4.2.5 Συνολική αποτίμηση

Από την παραπάνω διερεύνηση προκύπτουν κάποια συμπεράσματα που είναι καθολικά, και άλλα τα οποία, για τα διαφορετικά μήκη λέξης αλλάζουν. Σε αυτήν την παράγραφο θα σχολιάσουμε τα αποτελέσματα αυτά.

Καθυστέρηση

Σε ό,τι αφορά την καθυστέρηση του κυκλώματος και το κρίσιμο μονοπάτι, είδαμε καθολικά την υλοποίηση BCU_Conv_2 να εμφανίζει την μικρότερη καθυστέρηση, ανεξάρτητα του μήκους λέξης, γεγονός που επιβεβαιώνει την καλύτερη λειτουργία του NR4SD προ-κωδικοποιημένου πολλαπλασιαστή, ο οποίος αξιοποιεί στο μέγιστο τον εκ των προτέρων υπολογισμό των twiddle factors καθώς και οι δύο μονάδες, BCU_Conv_1 και BCU_Conv_2, δέχονται στην είσοδό τους τα αθροίσματα και τις διαφορές των twiddle factors, σε μορφή συμπληρώματος ως προς 2 και NR4SD, αντίστοιχα. Και οι δύο μονάδες κωδικοποιούν τους αριθμούς που δέχονται στην είσοδό τους όπως είδαμε στο κεφάλαιο 3.4, αλλά το απλούστερο κύκλωμα της NR4SD κωδικοποίησης στην είσοδο, καθώς και το απλούστερο κύκλωμα παραγωγής μερικών γινομένων, επικρατεί αυτών της Modified Booth κωδικοποίησης, αντίστοιχα.

Επιφάνεια και κατανάλωση

Η επιφάνεια και η κατανάλωση των κυκλωμάτων που συντέθηκαν παρουσιάζουν κοινή συμπεριφορά, στα διαφορετικά μήκη λέξης. Όπως ήταν αναμενόμενο, όσο απομακρυνόμαστε από τις high performance περιοχές του ρολογιού και βρισκόμαστε σε χαλαρότερες συνθήκες περιορισμού, τα κυκλώματα γίνονται μικρότερα σε μέγεθος και κατανάλωση. Με αυτόν τον τρόπο τα κυκλώματα που κάνουν χρήση του αλγόριθμου του Gauss μας δίνουν μικρότερα και πιο ενεργειακά αποδοτικά αποτελέσματα, συμπεριφορά η οποία, όσο μεγαλώνει το μήκος λέξης εισόδου, γίνεται περισσότερο εμφανής. Έτσι μπορούμε να δούμε τις υλοποιήσεις BCU_Gauss_1 και 2 να έχουν την μικρότερη επιφάνεια και την μικρότερη κατανάλωση. Ταυτόχρονα, η χρήση πολλαπλασιαστών που βασίζονται στην NR4SD κωδικοποίηση, αποφέρει αποδοτικότερα κυκλώματα, με την μονάδα BCU_Conv_2 να είναι καλύτερη σε επιφάνεια και κατανάλωση από την BCU_Conv_1, και αντίστοιχη συμπεριφορά εμφανίζουμε και οι μονάδες που BCU_Gauss 1 και 2, μεταξύ τους.

Στην συνέχεια παρουσιάζουμε ένα πίνακα με συγκεντρωμένα τα σημαντικότερα αποτελέσματα από την παραπάνω διερεύνηση, στον οποίο έχουμε μετρήσεις καθυστέρησης, επιφάνειας και κατανάλωσης των υλοποιήσεών μας.

Στον παρακάτω πίνακα, ενώ βλέπουμε την προαναφερθείσα απόδοση της μονάδας BCU_Conv_2, και πως είναι η βέλτιστη λύση για το μικρότερο κρίσιμο μονοπάτι, οι τιμές για τις χαμηλότερες συνθήκες περιορισμού του ρολογιού, μας δείχνουν τα πλεονεκτήματα που είδαμε στα διαγράμματα παραπάνω. Για να διερευνήσουμε αυτό το συμπέρασμα θα συγκρίνουμε τις μονάδες μεταξύ τους, κατά μέσο όρο, σε λειτουργία αυστηρών περιορισμών ρολογιού, αλλά και σε χαλαρότερες συνθήκες.

Πίνακας 10: Συγκεντρωτικά αποτελέσματα

Design	Area(um2)	Power(mW)	Area(um2)	Power(mW)	Area(um2)	Power(mW)	Area(um2)	Power(mW)
16 bits								
	T = 0.89		**T = 1.00**		**T = 1.60**		**T = 2.00**	
BCU_Conv_1	48790.67	53.58	42909.89	41.00	35879.76	24.80	35563.02	21.00
BCU_Conv_2	40512.42	45.82	35922.88	35.30	32485.82	23.10	32419.97	18.40
BCU_Gauss_1	-	-	41903.23	42.74	33797.46	23.40	33311.38	20.00
BCU_Gauss_2	-	-	38037.33	38.62	32004.45	22.90	31580.30	19.40
24 bits								
	T = 1.01		**T = 1.12**		**T = 1.80**		**T = 2.20**	
BCU_Conv_1	93437.90	89.64	84098.11	72.06	69935.94	42.70	69389.49	37.60
BCU_Conv_2	73284.40	70.06	67459.28	58.32	62181.39	41.20	62052.03	33.90
BCU_Gauss_1	-	-	78730.85	71.86	62750.58	39.40	61843.49	35.00
BCU_Gauss_2	-	-	69313.44	63.21	58698.86	37.90	58264.53	32.10
32 bits								
	T = 1.09		**T = 1.18**		**T = 1.90**		**T = 2.30**	
BCU_Conv_1	162422.85	147.86	143783.25	117.15	115850.11	71.40	113960.67	65.60
BCU_Conv_2	120378.50	103.36	111466.77	88.88	101183.82	66.50	101068.58	58.20
BCU_Gauss_1	-	-	135980.88	123.25	99589.95	61.30	98234.42	58.10
BCU_Gauss_2	-	-	116097.07	102.61	92869.50	59.90	92430.46	52.60
48 bits								
	T = 1.33		**T = 1.41**		**T = 2.10**		**T = 2.50**	
BCU_Conv_1	295207.36	220.67	268207.97	189.25	238333.65	136.30	236436.37	129.20
BCU_Conv_2	224387.86	206.66	220185.62	154.05	207680.03	129.20	206717.28	110.60
BCU_Gauss_1	251822.37	158.63	229681.42	167.24	197064.67	121.70	193993.74	113.20
BCU_Gauss_2	-	-	205989.73	149.09	183583.01	119.30	181401.14	104.90
64 bits								
	T = 1.50		**T = 1.60**		**T = 2.40**		**T = 2.70**	
BCU_Conv_1	452237.07	308.05	421991.14	256.67	403276.27	226.70	399599.31	208.90
BCU_Conv_2	376380.37	245.83	370824.94	243.41	349865.49	204.50	348794.54	183.40
BCU_Gauss_1	393975.68	271.67	352314.70	231.66	325537.18	192.90	322475.66	187.00
BCU_Gauss_2	354310.77	247.76	328116.54	223.97	302916.43	190.90	301020.72	160.50

Πίνακας 11: Σύγκριση μονάδων BCU_Conv_1 και BCU_Conv_2

Bit width	BCU_Conv_2/BCU_Conv_1(%)		
	Delay	**Area**	**Power**
16 bits	-1.14	15.18	12.39
24 bits	4.95	18.03	16.84
32 bits	3.03	20.76	21.35
48 bits	5.50	19.92	22.26
64 bits	12.67	15.49	15.04
Mean Savings	5.00	17.88	17.58

Στον πίνακα 11, βλέπουμε ότι το κέρδος από την χρήση των προκωδικοποιημένων πολλαπλασιαστών σε επιφάνεια και κατανάλωση είναι της τάξης του 18%, με μέσο κέρδος στην καθυστέρηση στο 5%. Στην συνέχεια βλέπουμε την ίδια σύγκριση για τις υλοποιήσεις που κάνουν χρήση του αλγόριθμου του Gauss.

Πίνακας 12: Σύγκριση μονάδων BCU_Gauss_1 και BCU_Gauss_2

Bit width	BCU_Gauss_2/BCU_Gauss_1(%)		
	Delay	**Area**	**Power**
16 bits	3.03	8.52	7.96
24 bits	1.80	11.30	12.28
32 bits	1.71	11.40	11.56
48 bits	-0.75	9.12	10.17
64 bits	1.33	7.90	6.07
Mean Savings	1.42	9.65	9.61

Στη σύγκριση των μονάδων του αλγόριθμου του Gauss, βλέπουμε μικρότερο κέρδος, κάτι που εξηγείται από την χρήση της μονάδας επανακωδικοποίησης Αφαίρεσης-Πολλαπλασιασμού. Η μονάδα αυτή επηρεάζει το κρίσιμο μονοπάτι και συνεπώς δεν επιτρέπει στα εργαλεία σύνθεσης να κάνουν τις μέγιστες δυνατές ελαχιστοποιήσεις στο κύκλωμα.

Στις παραπάνω μετρήσεις, τα αποτελέσματα προκύπτουν ως, ο μέσος όρος των μετρήσεων που έγιναν για αυστηρές συνθήκες περιορισμού ρολογιού, και για χαλαρότερες συνθήκες, με σκοπό να κάνουμε μια συνολικότερη αποτίμηση των αποτελεσμάτων.

Με την ίδια λογική που ακολουθήθηκε παραπάνω, στην συνέχεια θα συγκρίνουμε την μονάδα BCU_Gauss_2 με βάση σύγκρισης την μονάδα BCU_Conv_1. Έτσι, θα δούμε την συνολική βελτιστοποίηση που παρέχουν του αλγόριθμου Gauss και των πολλαπλασιαστών NR4SD, επί της μονάδας αναφοράς, των συμβατικών τρόπων υλοποίησης.

Πίνακας 13: Σύγκριση μονάδων BCU_Gauss_1 και BCU_Conv_1

Bit width	BCU_Gauss_2/BCU_Conv_1(%)		
	Delay	**Area**	**Power**
16 bits	-9.09	10.71	6.95
24 bits	-7.92	16.93	12.93
32 bits	-5.50	19.40	13.91
48 bits	-0.75	23.39	21.53
64 bits	1.33	21.87	16.00
Mean Savings	-4.39	18.46	14.27

Στον πίνακα 13, βλέπουμε το κέρδος που μας προσφέρει η χρήση του αλγόριθμου Gauss για τον πολλαπλασιασμό μιγαδικών αριθμών, και η χρήση προ-κωδικοποιημένων πολλαπλασιαστών, και συγκεκριμένα η κωδικοποίηση NR4SD, σε σχέση με τον συμβατικό τρόπο υλοποίησης της μονάδας υπολογισμού Butterfly. Το κέρδος αυτό, αφορά στην επιφάνεια του κυκλώματος που παράγεται, αλλά και στην κατανάλωσή του.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στο παρόν κεφάλαιο θα παρουσιάσουμε τα συμπεράσματα που προέκυψαν από την διερεύνηση του αντικειμένου μας, δηλαδή της μονάδας υπολογισμού Butterfly, για την εφαρμογή του αλγόριθμου FFT για αποδεκατισμό στο επίπεδο του χρόνου.

5.1 Σύνοψη και συμπεράσματα

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

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

- ***Καθυστέρηση***

Οι υλοποιήσεις που κάνουν χρήση του συμβατικού αλγόριθμου για τον πολλαπλασιασμό δύο μιγαδικών αριθμών, BCU_Conv_1 και BCU_Conv_2, ήταν οι αποδοτικότερες σε ό,τι αφορά την καθυστέρηση. Επιπροσθέτως, όλες οι υλοποιήσεις κατά τον σχεδιασμό τους έχουν γίνει pipelined, περίπου στο μέσο του κρίσιμου μονοπατιού με σκοπό την μείωση της καθυστέρησης. Με τον τρόπο αυτό, τα κυκλώματα που κάνουν χρήση του συμβατικού αλγόριθμου, μας έδωσαν την μικρότερη καθυστέρηση για όλα τα μήκη λέξης. Οι επιμέρους μονάδες που χρησιμοποιήθηκαν, πριν το επίπεδο pipeline καταχωρητών, αποτελούνται από τους πολλαπλασιαστές πραγματικών αριθμών. Στον συμβατικό αλγόριθμο έχουμε μόνο μονάδες πολλαπλασιασμού, το κύκλωμα των οποίων είναι σχετικά απλό, και ο compiler που χρησιμοποιήθηκε για την σύνθεση των κυκλωμάτων αυτών μπορεί να απλοποιήσει την ιεραρχία του κυκλώματος και να επιτύχει το καλύτερο δυνατό κύκλωμα. Ο αλγόριθμος του Gauss, για λόγους που εξηγήθηκαν στην περιγραφή του, στο κεφάλαιο 3.4.2, πρέπει να κάνει χρήση μιας μονάδας επανακωδικοποίησης της πράξης Αφαίρεσης-Πολλαπλασιασμού. Το κρίσιμο μονοπάτι που δημιουργείται από την χρήση της μονάδας αυτής επιβαρύνει το κύκλωμα μας, και το καθιστά χαμηλότερης απόδοσης από αυτό του συμβατικού αλγόριθμου πολλαπλασιασμού μιγαδικών αριθμών.

Στην σύγκριση των δύο μονάδων, BCU_Conv_1 και BCU_Conv_2, βλέπουμε πως η χρήση προ-κωδικοποιημένων πολλαπλασιαστών πραγματικών αριθμών είναι η καλύτερη λύση σε σχέση με τους συμβατικούς MB πολλαπλασιαστές. Το κύκλωμα της μονάδας υπολογισμού Butterfly μας επιτρέπει την χρήση τους, δεδομένου ότι γνωρίζουμε τα twiddle factors σύμφωνα με τον αλγόριθμο FFT, και η κωδικοποίηση NR4SD έδωσε σημαντικό κέρδος επί των MB πολλαπλασιαστών, έως και 12 τοις εκατό για μήκος λέξης εισόδου 64 bits. Αυτό οφείλεται στο απλούστερο κύκλωμα παραγωγής γινομένων λόγο των λιγότερων καταστάσεων σε σχέση με τον αλγόριθμο MB, και η καλύτερη απόδοσή τους μπορεί είναι οριακή αλλά ουσιαστική, και στις υλοποιήσεις που χρησιμοποιούν τον αλγόριθμο του Gauss. Βέβαια στις δύο αυτές υλοποιήσεις, BCU_Gauss_1 και BCU_Gauss_2, η συνολική απόδοση του κυκλώματος επηρεάζεται σημαντικά από την μονάδα επανακωδικοποίησης Αφαίρεσης-Πολλαπλασιασμού. Τέλος, τα κυκλώματα μετά το επίπεδο των pipeline καταχωρητών, όπως φαίνεται και στα σχήματα 16 και 17, είναι παρόμοιο για όλες τις περιπτώσεις και δεν επηρεάζει το αποτέλεσμα. Με τον τρόπο αυτό, προσπαθήσαμε να συγκρίνουμε τις υλοποιήσεις ως προς τους παράγοντες που συζητήσαμε παραπάνω, χωρίς καμία επίδραση στο συνολικό κύκλωμα από τις μονάδες

που κάνουν πρόσθεση των μερικών γινομένων και τα φέρουν σε μορφή συμπληρώματος ως προς δύο, καθώς η Carry-Save αριθμητική που χρησιμοποιήσαμε αποφέρει τα αποδοτικότερα δυνατά κυκλώματα για τις πράξεις αυτές.

- ***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 βλέπουμε πως έχουμε μικρότερη επιφάνεια στις υλοποιήσεις του αλγόριθμου του Gauss, με την BCU_Gauss_2 να έχει την καλύτερη απόδοση.

Πιο συγκεκριμένα, η βελτιστοποίηση στην επιφάνεια των κυκλωμάτων βασίστηκε σε δύο παράγοντες, στην αρχιτεκτονική του μιγαδικού πολλαπλασιαστή, και στο είδος του πολλαπλασιαστή που χρησιμοποιήσαμε κατά την υλοποίησή του. Ο προ-κωδικοποιημένος πολλαπλασιαστής NR4SD, όπως περιγράφεται και στο [4], δίνει κυκλώματα με μικρότερη επιφάνεια σε σχέση με τον MB πολλαπλασιαστή, κάτι που επιβεβαιώθηκε από τις μετρήσεις μας για όλα τα διάφορα μήκη λέξης, μέσα από την σύγκριση ανάμεσα στις υλοποιήσεις BCU_Conv_1 και BCU_Conv_2, καθώς και από την σύγκριση BCU_Gauss_1 και BCU_Gauss_2, όπου οι υλοποιήσεις με δείκτη 1 κάνουν χρήση MB πολλαπλασιαστών ενώ οι υλοποιήσεις με δείκτη 2 χρησιμοποιούν NR4SD πολλαπλασιαστές. Στους πίνακες 11 και 12 του κεφαλαίου 4.2.5, βλέπουμε πως η χρήση NR4SD πολλαπλασιαστών μπορεί να δώσει κέρδος της τάξης του 15, έως 20%, στην επιφάνεια που καταλαμβάνει το κύκλωμά μας. Η αρχιτεκτονική των μιγαδικών πολλαπλασιαστών, επηρεάζει σημαντικά τις υλοποιήσεις μας και βλέπουμε πως υπάρχουν διαφορές στα διάφορα μήκη λέξης εισόδου. Στα μικρότερα μήκη λέξης, 16, 24 και 32 bits, βλέπουμε πως ο αλγόριθμος του Gauss δίνει μικρά κέρδη σε σχέση με τις συμβατικές υλοποιήσεις και λιγότερα σημαντικά σε σχέση με την υστέρησή του σε καθυστέρηση. Στα μεγαλύτερα μήκη λέξης, 48 και 64 bits, βλέπουμε πως το tradeoff ανάμεσα σε καθυστέρηση και επιφάνεια μπορεί να μας δώσει κυκλώματα που, όταν η επιφάνεια είναι σημαντικότερη της καθυστέρηση, η αρχιτεκτονική αυτή μας προσφέρει σημαντικό κέρδος

- ***Κατανάλωση***

Η κατανάλωση των κυκλωμάτων μας, ακολουθεί την περιγραφή μας για την επιφάνεια κάτι που ήταν αναμενόμενο, γιατί ένα μικρότερο κύκλωμα αναμένεται να έχει και μικρότερη κατανάλωση. Με αυτό τον τρόπο τα συμπεράσματα στα οποία καταλήξαμε για την επιφάνεια των κυκλωμάτων, εφαρμόζουν και για την κατανάλωση των κυκλωμάτων μας. Επίσης αξίζει να αναφέρουμε πως και οι NR4SD πολλαπλασιαστές έχουν απλοποιήσεις των καταστάσεών τους για να μειώσουν το switching activity του κυκλώματος, αλλά και ο επανακωδικοποιητής Αφαίρεσης-Πολλαπλασιασμού είναι σχεδιασμένος για να αποφέρει πιο αποδοτικά κυκλώματα και υλοποιεί πιο οικονομικά την πράξη αυτή.

Συνολικά, βλέπουμε πως από την συμβατική σχεδίαση της μονάδας υπολογισμού Butterfly, BCU_Conv_1, οι μονάδες που σχεδιάσαμε, BCU_Conv_2 και BCU_Gauss_2, έχουν να προσφέρουν σημαντικό κέρδος ανάλογα με την εφαρμογή για την οποία στοχεύουμε τον σχεδιασμό τους. Η μονάδα BCU_Conv_2 είναι σχεδιασμένη για να αποφέρει το μέγιστο σε ό,τι αφορά την απόδοση του κυκλώματος. Εφαρμογές οι οποίες απαιτούν μικρή επιφάνεια κυκλώματος και χαμηλή κατανάλωση, μπορούν να κάνουν χρήση της μονάδας BCU_Gauss_2, η οποία αποδείξαμε πως είναι βέλτιστη σε αυτούς τους τομείς.

5.2 Μελλοντικές επεκτάσεις

Οι μελλοντικές επεκτάσεις, αυτής της εργασίας συνοψίζονται παρακάτω:

- Η ενσωμάτωση των κυκλωμάτων που συντέθηκαν σε μία μονάδα FFT, θα μας έδινε σημαντικά συμπεράσματα για την βέλτιστη χρήση της, καθώς και θα μπορούσαμε να εξετάσουμε την περαιτέρω βελτίωση της λειτουργίας της.
- Η σχεδίαση της μονάδας FFT με χρήση των μονάδων BCU που εξετάσαμε στην παρούσα εργασία, πάνω σε συγκεκριμένες εφαρμογές, θα μας έδινε χρήσιμα συμπεράσματα, καθώς θα μπορούσαμε να ενσωματώσουμε διαφορετικές στρατηγικές σύνθεσης των κυκλωμάτων, με σκοπό τα εργαλεία που χρησιμοποιήσαμε να μας αποφέρουν το βέλτιστο αποτέλεσμα, μιας και στην εργασία αυτή κάναμε χρήση μίας, κοινής στρατηγικής σύνθεσης.
- Η μονάδα επανακωδικοποίηση της πράξης Αφαίρεσης-Πολλαπλασιασμού, έχει σημαντικό ρόλο στα αποτελέσματα των μονάδων που κάνουν χρήση του αλγόριθμου του Gauss. Μία επέκταση που θα μπορούσε να βελτιώσει συνολικά την εφαρμογή του αλγόριθμου του Gauss, θα ήταν η επανακωδικοποίηση των αριθμών της αφαίρεσης σε NR4SD κωδικοποίηση. Η μονάδα που χρησιμοποιήσαμε κάνει επανακωδικοποίηση των αριθμών που αφαιρούνται σε MB μορφή. Έτσι, θα μπορούσαμε να επωφεληθούμε σε επιφάνεια, κατανάλωση και καθυστέρηση από τα απλούστερα κυκλώματα παραγωγής μερικών γινομένων της NR4SD μορφής.
- Τέλος, ένα πλήρως pipelined σχήμα θα μπορούσε να σχεδιαστεί με σκοπό την ελαχιστοποίηση του κρίσιμου μονοπατιού, και με αυτόν τον τρόπο θα μπορούσαμε να ξεπεράσουμε το κέρδος, σε καθυστέρηση, του συμβατικού αλγόριθμου επί του αλγόριθμου του Gauss. Το αποτέλεσμα από μια τέτοια σχεδίαση θα μπορούσε να είναι ένα κύκλωμα το οποίο θα απολάμβανε των κερδών σε επιφάνεια και καθυστέρηση του αλγόριθμου του Gauss, ενώ δεν θα υπολειπόταν στο κρίσιμο μονοπάτι και στην καθυστέρησή του.

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

VLSI	Very-large-scale integration
FFT	Fast Fourier Transform
DIT	Decimation in Time
DIF	Decimation in Frequency
MB	Modified Booth
NR4SD	Non-Redundant Radix-4 Signed-Digit
BCU	Butterfly Computation Unit
CSA	Carry-Save Adder
CLA	Carry Look-Ahead Adder
FA	Full Adder
HA	Half Adder
CS	Carry-Save

ΑΝΑΦΟΡΕΣ

[1] G. W. Reitwiesner, “Binary arithmetic,” Adv. Comput., vol. 1, pp. 231–308, 1960.

[2] P. Duhamel and M. Vetterli, “Fast Fourier transforms: a tutorial review and a state of the art,” Signal Processing, vol. 19, pp. 259–299, Apr. 1990.

[3] Z. Huang, “High-Level Optimization Techniques for Low-Power Multiplier Design,” Ph.D., University of California, Department of Computer Science, Los Angeles, CA, 2003.

[4] K. Tsoumanis, N. Axelos, N. Moschopoulos, G. Zervakis, and K. Pekmestzi, “Pre-Encoded Multipliers Based on Non-Redundant Radix-4 Signed-Digit Encoding” IEEE Trans. Comput, vol. 65, no. 2, Feb. 2016.

[5] Kostas Tsoumanis, Sotiris Xydis, Constantinos Efstathiou, Nikos Moschopoulos, and Kiamal Pekmestzi, “An Optimized Modified Booth Recoder for Efficient Design of the Add-Multiply Operator” IEEE Trans. Circuits Syst., Regular Papers, vol. 61, no. 4, Apr. 2014 .

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

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

[8] F. Ntouskas, "Αποδοτική σχεδίαση μιγαδικού πολλαπλασιαστή σε ASIC και FPGA", Diploma Thesis, School of Electrical and Computer Engineering, National Technical University of Athens, 2014

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

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

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

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