ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση νέων αρχιτεκτονικών παράλληλων πολλαπλασιαστών με χαμηλή κατανάλωση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΕΥΑΓΓΕΛΟΥ ΚΟΓΧΥΛΑΚΗ

Επιβλέπων : Κιαμάλ Πεκμεστζή,

Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2010

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

Υλοποίηση νέων αρχιτεκτονικών παράλληλων πολλαπλασιαστών με χαμηλή κατανάλωση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΕΥΑΓΓΕΛΟΥ ΚΟΓΧΥΛΑΚΗ

Επιβλέπων : Κιαμάλ Πεκμεστζή,

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Νοεμβρίου 2010.

.....	.....	.....
Κ. Πεκμεστζή	Δ. Σούντρης	Γ. Οικονομάκος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2010

.....

ΕΥΑΓΓΕΛΟΣ ΚΟΓΧΥΛΑΚΗΣ

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

Περίληψη

Ο σκοπός της διπλωματικής εργασίας ήταν η ανάπτυξη νέων τύπων αλγορίθμων πολλαπλασιασμού, οι οποίοι έχουν χαμηλή κατανάλωση ισχύος σε σήματα που χρησιμοποιούνται σε DSP εφαρμογές. Οι αλγόριθμοι αυτοί χωρίστηκαν σε δύο βασικές κατηγορίες. Η πρώτη ήταν οι αλγόριθμοι πολλαπλασιασμού με δενδρική δομή πρόσθεσης των μερικών γινομένων, ενώ η δεύτερη ήταν η ιδέα του Most Significant First πολλαπλασιαστή.

Οι δύο τύποι πολλαπλασιαστών εκμεταλλεύονται την ύπαρξη μηδενικών ή μονάδων στα υψηλότερης αξίας bits των εισόδων που είναι πολύ σύνηθες φαινόμενο σε σήματα φωνής και γενικότερα σήματα που χρησιμοποιούνται σε DSP εφαρμογές. Με αυτή την τεχνική επιτυγχάνεται η αποδοτικότερη υλοποίηση των αλγορίθμων, ώστε να επιτυγχάνεται χαμηλή δυναμική κατανάλωση, ταυτόχρονα με τη μέγιστη δυνατή επίδοση, στα κυκλώματα πολλαπλασιασμού που προέκυψαν από τους προτεινόμενους αλγορίθμους.

Συγκεκριμένα, στα πλαίσια της εργασίας αυτής έγινε μελέτη τριών πολλαπλασιαστών δενδρικής δομής, οι οποίοι χωρίζονται σε δύο ή περισσότερα επιμέρους blocks. Δύο απ' αυτούς υλοποιήθηκαν με τη χρήση έτοιμων μονάδων. Τα αποτελέσματα που παρουσιάστηκαν βασίστηκαν σε μια CMOS βιβλιοθήκη τυποποιημένων κυττάρων των 90nm και συγκρίνουν τις τρεις αρχιτεκτονικές ως προς τη δυναμική και τη συνολική κατανάλωση, για διαφορετικούς τύπους δεδομένων και στις δύο εισόδους των πολλαπλασιαστών. Επίσης, υλοποιήθηκε και ο MSF πολλαπλασιαστής και επεκτάθηκε η ιδέα του και σε έναν τετραγωνιστή (MSF Squarer). Παρουσιάστηκαν αποτελέσματα για τους MSF αλγορίθμους και συγκεκριμένα ο MSF Squarer συγκρίθηκε με τον αλγόριθμο τετραγωνισμού Booth Folding, που έχει παρουσιαστεί στη βιβλιογραφία. Τέλος παρουσιάστηκαν συνολικές συγκρίσεις μεταξύ των πολλαπλασιαστών δενδρικού τύπου και του MSF.

Λέξεις Κλειδιά:

Παράλληλοι πολλαπλασιαστές, DSP, block, Most significant first, MSF squarer, αρχιτεκτονική

Abstract

The scope of this thesis was the development of new types of multiplication algorithms that have low power dissipation when using signal that are common in DSP applications. The development of these algorithms followed two main chapters. The first was block multipliers, while the second one was a multiplier that uses the Most Significant First technique.

Specifically, three types of block multipliers were studied, that were broken into two or more blocks. Some of these were also created by using modules from Synopsys DesignWare IP library and results were presented for a variety of data types in comparison with directly comparable architectures. Also, another multiplier, the MSF multiplier was created and the same idea was applied to a squarer (MSF Squarer). The results were shown for all MSF algorithms for different type of data. Lastly, a broad comparison between block and MSF multipliers was presented.

These two types of multipliers take advantage of high order ones and zeros that is a very common phenomenon when dealing with signals that are used in DSP applications. With these techniques we managed to create a few multipliers and also a squarer that have as little area as possible while having low power consumption.

Keywords:

Parallel multipliers, DSP, block, Most significant first, MSF squarer, architecture

Πίνακας περιεχομένων

1

ΕΙΣΑΓΩΓΗ

1.1 Πεδίο εφαρμογής αλγορίθμων

Ένα μεγάλο πεδίο εφαρμογής των κυκλωμάτων CMOS VLSI αριθμητικών πράξεων και κυρίως των μονάδων πολλαπλασιασμού είναι η υλοποίηση συστημάτων Ψηφιακής επεξεργασία σήματος (Digital Signal Processing, DSP). Η ψηφιακή επεξεργασία σήματος έχει να κάνει με την αναπαράσταση των σημάτων από μια ακολουθία ψηφίων και έπειτα την επεξεργασία τους. Στις περιπτώσεις που τα σήματα που θα υποστούν επεξεργασία βρίσκονται σε αναλογική μορφή αυτά μετατρέπονται μέσω ειδικών κυκλωμάτων σε ψηφιακά (analog to digital converters). Στην συνέχεια εφαρμόζονται οι DSP αλγόριθμοι που επεξεργάζονται το σήμα με κάποιον επιθυμητό τρόπο πριν οδηγηθεί στην έξοδο.

Υπάρχουν αρκετές κατηγορίες DSP αλγορίθμων με τα κυριότερα πεδία εφαρμογών τους να είναι τα εξής:

- Επεξεργασία ήχου και σημάτων φωνής
- Συστήματα Ραντάρ
- Αισθητήρες
- Επεξεργασία στοχαστικών σημάτων
- Ψηφιακή επεξεργασία εικόνας
- Επεξεργασία σημάτων στις τηλεπικοινωνίες
- Επεξεργασία ιατρικών σημάτων

Οι DSP αλγόριθμοι μπορούν να εκτελεσθούν είτε σε εξειδικευμένους επεξεργαστές (DSP processors), είτε σε πυρήνες ειδικού σκοπού, είτε, τέλος, από το υλικό. Για το σκοπό αυτό, μπορούν να σχεδιαστούν κυκλώματα που υλοποιούν μόνο έναν συγκεκριμένο αλγόριθμο, είτε μια σειρά από αλγορίθμους, με χρήση επαναδιατάξιμων αρχιτεκτονικών, πυρήνας των οποίων είναι η μονάδα που υλοποιεί τις αριθμητικές πράξεις, βασικότερη των οποίων είναι ο πολλαπλασιασμός.

Συνήθως, στον πολλαπλασιασμό δεδομένων με συντελεστές, όπως συμβαίνει σε αλγορίθμους DSP, το σήμα δεδομένων είναι κάποια αλληλουχία ψηφίων της οποίας πολλές φορές αρκετά από τα bit υψηλότερης αξίας είναι μηδενικά (στην περίπτωση που ο αριθμός είναι θετικός) ή μονάδες (στην περίπτωση που ο αριθμός είναι αρνητικός). Επομένως, οι πράξεις με αυτά τα ψηφία δίνουν μηδενικά μερικά γινόμενα, αν ακολουθήσει κανείς την κωδικοποίηση Booth, όπως θα δούμε και στο επόμενο κεφάλαιο. Επομένως, αυτό το γεγονός αποτελεί από μόνο το σχεδιαστικό κίνητρο για την όσο το δυνατό μεγαλύτερη μείωση της κατανάλωσης.

Γενικότερα, ο σχεδιασμός πυρήνων πολλαπλασιαστών που να έχουν υψηλή απόδοση και μικρή επιφάνεια είναι αναγκαίος για την αποδοτική υλοποίηση DSP αλγορίθμων σε real-time ενσωματωμένα συστήματα. Πλέον είναι αναγκαίο οι πολλαπλασιαστές αυτοί να έχουν και χαμηλή κατανάλωση καθώς η μπαταρία είναι η μόνη πηγή ενέργειας σε ενσωματωμένα συστήματα όπως τα συστήματα κινητής τηλεφωνίας.

Σε εφαρμογές κρυπτογραφίας βρίσκει εφαρμογή η αριθμητική σε επίπεδο bit, όπου χρειάζονται πολλοί αριθμητικοί υπολογισμοί. Οι τετραγωνιστές βρίσκουν εφαρμογή σε αλγόριθμους κρυπτογραφίας όπως για παράδειγμα ο αλγόριθμος RSA. Η ανάπτυξη της κρυπτογραφίας μαζί με την ζήτηση για χαμηλή κατανάλωση δικαιολογούν τις σχεδιαστικές προσπάθειες για μία υλοποίηση του αλγορίθμου τετραγωνισμού η οποία θα συμβαδίζει με τις ανάγκες που δημιουργούνται για χαμηλή κατανάλωση και ταυτόχρονα υψηλή απόδοση.

1.2 Αντικείμενο διπλωματικής

Η παρούσα διπλωματική έχει ως αντικείμενο την υλοποίηση αλγορίθμων πολλαπλασιασμού που να λειτουργούν όσο πιο αποδοτικά γίνεται με βάση τα παραπάνω κριτήρια. Έτσι για την μέγιστη δυνατή απόδοση χρησιμοποιήθηκαν παράλληλοι πολλαπλασιαστές Wallace με modified booth κωδικοποίηση. Για την τεχνολογία 90nm που χρησιμοποιήθηκε η δυναμική κατανάλωση είναι ο κυρίαρχος παράγοντας στην συνολική κατανάλωση συγκρινόμενη με την στατική κατανάλωση και για αυτό επικεντρωθήκαμε σε αυτή. Επίσης ένας ακόμα λόγος που δώσαμε βάση κυρίως στην δυναμική κατανάλωση είναι ότι τα κυκλώματα αυτά, όταν χρησιμοποιούνται για την υλοποίηση συστημάτων DSP δουλεύουν αδιάκοπα, δηλαδή νέα δεδομένα εισόδου εισέρχονται για επεξεργασία στο σύστημα. Επομένως, συνήθως τα

κυκλώματα πολλαπλασιασμού, που αποτελούν τον πυρήνα των DSP συστημάτων, δουλεύουν σε κάθε κύκλο ρολογιού και επομένως μένουν άεργα για ελάχιστους κύκλους, με αποτέλεσμα η βαρύτητα της δυναμικής κατανάλωσης να είναι σημαντικά μεγαλύτερη.

Εφαρμόζονται διάφοροι τρόποι κωδικοποίησης των αριθμών που υπόκεινται σε πολλαπλασιασμό και σε διάφορα blocks χωριστά το ένα από το άλλο, σε αντίθεση με την κλασσική υλοποίηση του πολλαπλασιασμού. Στόχος ήταν να εκμεταλλευτούμε την ύπαρξη μεγάλου αριθμού μηδενικών δυαδικών ψηφίων των δύο αριθμών, ώστε να επιτευχθεί η μέγιστη εξοικονόμηση ενέργειας.

Επίσης εφαρμόζεται στην παρούσα διπλωματική η ιδέα των left-to-right πολλαπλασιαστών για να υπάρχει κέρδος σε σήματα που έχουν μεγάλο αριθμό μηδενικών ή άσσων στα υψηλότερης αξίας bit τους. Αυτοί οι πολλαπλασιαστές καλούνται Most Significant First (MSF) Multipliers. Στα πλαίσια αυτά, επεκτείναμε την ιδέα των MSF Multipliers σε κυκλώματα υπολογισμού του τετραγώνου και έτσι, παρουσιάζεται και ο MSF Squarer, που βρίσκει εφαρμογή στην κρυπτογραφία.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε συστήματα πολλαπλασιασμού και τεχνικές κωδικοποίησης με στόχο το συνδυασμό υψηλής επίδοσης και χαμηλής κατανάλωσης ισχύος.
2. Υλοποιήσαμε 3 αλγορίθμους πολλαπλασιασμού με ανεξάρτητα blocks.
3. Υλοποιήσαμε 1 MSF πολλαπλασιαστή και 1 ίδιας τεχνικής τετραγωνιστή.
4. Αξιολογήσαμε την επίδοση των αλγορίθμων ως προς την επιφάνεια και την κατανάλωση τους στα 90nm.

1.3 Οργάνωση κειμένου

Το θεωρητικό υπόβαθρο για το αντικείμενο της διπλωματικής παρουσιάζεται στο Κεφάλαιο 2. Το Κεφάλαιο 3 παρουσιάζει τους αλγόριθμους block και αναλύει το πώς υλοποιήθηκαν ενώ ταυτόχρονα παρουσιάζει και τα αποτελέσματα τους. Στο Κεφάλαιο 4 αναπτύσσουμε τον MSF αλγόριθμο και επεκτείνουμε την ιδέα και στον τετραγωνιστή και παρουσιάζουμε πάλι τις υλοποιήσεις αυτών καθώς και τα αποτελέσματα τους όπως και μια συνολική παρουσίαση των αποτελεσμάτων όλων των πολλαπλασιαστών. Το κεφάλαιο 5 παρουσιάζει μία σύνοψη της διπλωματικής μαζί με τα συμπεράσματα και με πιθανές προεκτάσεις αυτής. Τέλος στο κεφάλαιο 6 παρατίθεται η βιβλιογραφία που χρησιμοποιήθηκε για την συγγραφή της παρούσας διπλωματικής.

2

ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Σε αυτό το κεφάλαιο καλύπτεται το θεωρητικό υπόβαθρο που είναι αναγκαίο ώστε να γίνει κατανοητή η παρούσα διπλωματική από τον αναγνώστη πριν από την παρουσίαση της ανάλυσης και σχεδίασης των αλγορίθμων. Οι προαπαιτούμενες αυτές γνώσεις αρχίζουν με την παράσταση των αριθμών στα ψηφιακά κυκλώματα και την κωδικοποίηση των δυαδικών αριθμών και την δικαιολόγηση των πλεονεκτημάτων αυτών. Το θεωρητικό αυτό υπόβαθρο ολοκληρώνεται με την παρουσίαση των δομικών μονάδων των αθροιστών καθώς και την παρουσίαση διαφόρων δομών αθροιστών και πολλαπλασιαστών.

2.1 Αριθμητικά συστήματα

2.1.1 Δυαδικό σύστημα

Το δυαδικό σύστημα είναι το πιο διαδεδομένο αριθμητικό σύστημα στα ψηφιακά κυκλώματα. Αποτέλεσε το πρώτο και το πιο απλό σύστημα που χρησιμοποιήθηκε, και πάνω σε αυτό βασίστηκαν οι επόμενες επεκτάσεις του που χρησιμοποιούνται σήμερα. Τα μοναδικά ψηφία κάθε δυαδικού αριθμού ανήκουν στο σύνολο {0,1} δηλαδή όπως υποδηλώνει και το όνομα του το δυαδικό σύστημα έχει βάση το 2. Αυτές οι δύο διαφορετικές καταστάσεις είναι εύκολο να διαχωριστούν στο επίπεδο των κυκλωμάτων θέτοντας την μία τιμή ως χαμηλή τάση και την άλλη τιμή ως ένα διαφορετικό από το προηγούμενο επίπεδο τάσης. Έχει επικρατήσει στα κυκλώματα να χρησιμοποιείται κάποιο επίπεδο τάσης πιο χαμηλό για την αναπαράσταση του

‘0’ από ότι για την αναπαράσταση του ‘1’. Από την παραπάνω πρόταση καταλαβαίνουμε ότι θα ήταν πιο δύσκολο να διαχωριστούν για παράδειγμα τα 16 επίπεδα ενός δεκαεξαδικού συστήματος.

Η σχέση ισότητας που αντιστοιχεί ανάμεσα σε έναν δεκαδικό και έναν δυαδικό αριθμό είναι η παρακάτω:

$$a_{(10)} = \sum_{i=0}^{n-1} b_i \cdot 2^i = b_{n-1}b_{n-2}...b_2b_1b_{0(2)}$$

όπου,

a: η ισοδύναμη δεκαδική τιμή του αριθμού

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

n: το πλήθος των δυαδικών ψηφίων του αριθμού

Για παράδειγμα ο δεκαδικός αριθμός 154 αντιστοιχίζεται στο δυαδικό σύστημα με τον εξής τρόπο:

$$154_{(10)} = 10011010_{(2)} = 1\cdot2^7 + 0\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 0\cdot2^0$$

Παρόμοιες σχέσεις ισχύουν και για μετατροπή δυαδικού σε δεκαεξαδικό σύστημα.

Εδώ αξίζει να σημειώσουμε πως από ένα πλήθος n ψηφία στο δυαδικό σύστημα μπορούν να παραχθούν 2^n διαφορετικοί αριθμοί.

Στο συμβατικό δυαδικό σύστημα που εξετάζουμε υπάρχει μία αντιστοιχία ένα προς ένα μεταξύ δυαδικών και δεκαδικών. Επίσης όταν αναφερόμαστε στο bit με το μεγαλύτερο βάρος (Most Significant Bit- MSB) εννοούμε το αριστερότερο ψηφίο του δυαδικού αριθμού ενώ όταν αναφερόμαστε στο bit με το μικρότερο βάρος (Least Significant Bit- LSB) εννοούμε το δεξιότερο ψηφίο του δυαδικού αριθμού.

2.1.1.1 Πρόσθεση αριθμών

Η πρόσθεση των αριθμών στο συμβατικό δυαδικό σύστημα δεν διαφέρει από την πρόσθεση στο δεκαδικό σύστημα αφού κάθε φορά το κρατούμενο διαδίδεται στην αριστερότερη βαθμίδα με την μόνη προφανή διαφορά να είναι η διαφορετική βάση του συστήματος. Ένα παράδειγμα μιας τέτοιας πρόσθεσης είναι η πρόσθεση των αριθμών 154 και 169:

154	=	10011010
+169	=	+ 10101001
323		101000011

Όπου οι δύο αυτοί αριθμοί προφανώς ισούνται.

Το συμβατικό δυαδικό σύστημα όμως παρουσιάζει ένα σημαντικό μειονέκτημα, και αυτό είναι ότι δεν μπορεί να παρουσιάσει προσημασμένους αριθμούς (signed) παρά μόνο απρόσημους (unsigned). Έτσι το αποτέλεσμα μιας αφαίρεσης όπου το δεύτερος αριθμός θα ήταν μεγαλύτερος από τον πρώτο δεν θα μπορούσε να παρασταθεί με αυτό το σύστημα.

2.1.2 Αναπαράσταση σε μορφή συμπληρώματος ως προς δύο

Η προηγούμενη αδυναμία του συμβατικού δυαδικού συστήματος αντιμετωπίστηκε από την αναπαράσταση σε μορφή συμπληρώματος ως προς δύο με την εξής παραδοχή: Αν το ψηφίο με το μεγαλύτερο βάρος (MSB) είναι 0 τότε ο αριθμός είναι θετικός ενώ αν το MSB είναι 1 τότε ο αριθμός είναι αρνητικός. Δηλαδή εν ολίγοις το ψηφίο με το μεγαλύτερο βάρος έχει αρνητική αξία.

Με αυτή την παραδοχή η σχέση που αναπαριστά έναν αριθμό σε μορφή συμπληρώματος ως προς δύο γίνεται:

$$a_{(10)} = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i = b_{n-1} b_{n-2} \ldots b_2 b_1 b_0$$

όπου,

a: η ισοδύναμη δεκαδική τιμή του αριθμού

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

n: το πλήθος των δυαδικών ψηφίων του αριθμού

Όπως ακριβώς και στο συμβατικό δυαδικό, έτσι και στο παρόν αριθμητικό σύστημα η αναπαράσταση ενός αριθμού είναι αμφιμονοσήμαντη.

Επί παραδείγματι, ο δυαδικός αριθμός 10011010 ενώ στο συμβατικό δυαδικό σύστημα αντιστοιχεί στην δεκαδική τιμή 154, στην αναπαράσταση σε μορφή συμπληρώματος ως προς δύο αντιστοιχεί σε αρνητικό δεκαδικό αριθμό ίσο με την τιμή

$$10011010_{(2)} = -1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -102_{(10)}$$

2.1.2.1 Συμπλήρωμα ως προς δύο

Συμπλήρωμα ως προς δύο ενός αριθμού που έχει αναπαρασταθεί στο δυαδικό σύστημα ως προσημασμένος αριθμός είναι ένας προσημασμένος δυαδικός αριθμός που έχει το ίδιο μέτρο αλλά αντίθετο πρόσημο. Συγκεκριμένα για να βρούμε το συμπλήρωμα ως προς δύο ενός αριθμού ακολουθούμε τα εξής βήματα:

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

Β) Έπειτα προσθέτουμε μία μονάδα στον αριθμό που προέκυψε.

Για παράδειγμα θα βρούμε το συμπλήρωμα ως προς δύο του αριθμού 10011010= $-102_{(10)}$.

Αντιστρέφοντας τα ψηφία του παίρνουμε τον αριθμό:

01100101

και προσθέτουμε μία μονάδα σε αυτόν τον αριθμό:

$$\begin{array}{r} 01100101 \\ +\quad\quad 1 \\ \hline 01100110_{(2)} \end{array}$$

Έτσι, βλέπουμε πως το συμπλήρωμα ως προς δύο του $-102_{(10)}$ είναι το $102_{(10)}$ που επαληθεύει τον ορισμό.

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

Η πρόσθεση και αφαίρεση δύο αριθμών σε μορφή συμπληρώματος ως προς δύο γίνεται με παρόμοιο τρόπο με την πρόσθεση και αφαίρεση αριθμών στο συμβατικό δυαδικό σύστημα με δύο διαφοροποιήσεις:

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

Β) Για να προστεθούν δύο αριθμοί και το αποτέλεσμα να είναι πάντα σωστό θα πρέπει να έχουν τον ίδιο αριθμό ψηφίων. Για να συμβεί αυτό αυξάνουμε τον αριθμό με τα λιγότερα

ψηφία κατά όσα ψηφία χρειάζεται, ώστε οι δύο αριθμοί να έχουν τον ίδιο αριθμό ψηφίων. Η επαύξηση αυτή πραγματοποιείται τοποθετώντας τα επιπλέον bits στα αριστερά του MSB του αριθμού και δίνοντας σε όλα την τιμή του MSB. Με αυτό τον τρόπο δεν μεταβάλλεται η δεκαδική τιμή του αριθμού. Η τεχνική αυτή ονομάζεται επέκταση προσήμου (sign extension).

Παρακάτω παρατίθεται ένα παράδειγμα για την πρόσθεση προσημασμένων δυαδικών αριθμών:

Έστω ότι θέλουμε να προσθέσουμε τους $01100110_{(2)}= 102_{(10)}$ και $1000110_{(2)}= -58_{(10).}$

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

$1000110_{(2)}= (1)1000110_{(2)}$ που έχει και αυτός 8 ψηφία..

Στην συνέχεια προσθέτουμε όλα τα bit εκτός από τα MSB:

$$\begin{array}{ll} & (0)1100110 \\ + & \underline{(1)1000110} \\ & \underline{1}0101100 \end{array}$$

Με το υπογραμμισμένο ψηφίο του αποτελέσματος να είναι το κρατούμενο που προέρχεται από την προηγούμενη βαθμίδα των MSB, είναι θετικής αξίας και στην δικιά μας περίπτωση έχει την τιμή 1.

Προσθέτουμε τα MSB και το κρατούμενο:

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

Το αποτέλεσμα παριστάνεται ως 00 (και αυτό γιατί είναι ίσο με $(-1) \cdot 2^1 \cdot 0 + 2^0 \cdot 0 = 0$) και το τελικό αποτέλεσμα προκύπτει από την συνένωση αυτού του αριθμού με το υπόλοιπο αποτέλεσμα που βγάλαμε πιο πάνω όπου παραλείψαμε τα MSB και το κρατούμενο της προηγούμενης βαθμίδας.

Άρα έχουμε:

$\{00, 0101100\}= 000101100_{(2)}=44_{(10)}= 102_{(10)} + (-58_{(10)})$.

Παρακάτω παρατίθεται ένα δεύτερο παράδειγμα για την αφαίρεση προσημασμένων δυαδικών αριθμών:

Έστω ότι θέλουμε να αφαιρέσουμε τον $0100110_{(2)}= 38_{(10)}$ από τον $01000110_{(2)}=70_{(10).}$

Πρώτο βήμα μας είναι να εφαρμόσουμε επέκταση προσήμου στον πρώτο αριθμό αφού έχει 7 και όχι 8 ψηφία όπως ο δεύτερος. Έτσι παίρνουμε τον δυαδικό: $00100110_{(2)}= 38_{(10)}$ και εν συνεχεία βρίσκουμε το συμπλήρωμα ως προς 1 του α το οποίο είναι το $11011001_{(2)}$ και του προσθέτουμε μία μονάδα για να πάρουμε το συμπλήρωμα ως προς δύο: $11011010_{(2)}$.

Το τελικό αποτέλεσμα θα βρεθεί αν προσθέσουμε τον αριθμό που μόλις υπολογίσαμε με τον $01000110_{(2)}=70_{(10)}$:

$$\begin{array}{r} (1)1011010 \\ +\ (\underline{0})1000110 \\ \underline{1}0100000 \end{array}$$

Όπου πάλι το υπογραμμισμένο ψηφίο του αποτελέσματος είναι το κρατούμενο που προέρχεται από την προηγούμενη βαθμίδα των MSB, είναι θετικής αξίας και σε αυτή την περίπτωση έχει την τιμή 1.

Προσθέτουμε τα MSB και το κρατούμενο:

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

Το οποίο όπως δείξαμε πιο πάνω αναπαρίσταται με τον αριθμό 00.

Έτσι το τελικό αποτέλεσμα βρίσκεται ως εξής:

$\{00, 0100000\} = 000100000_{(2)} = 0100000_{(2)} = 32_{(10)} = 70_{(10)} - 38_{(10)}$.

2.2 Κωδικοποιήσεις booth και modified booth

2.2.1 Κωδικοποίηση booth

Έστω δυαδικός αριθμός X σε μορφή συμπληρώματος ως προς δύο. Έτσι 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 μπορει να γραφεί ισοδύναμα: $X = 2X - X$

$2X =$	$-x_{n-1}$	x_{n-2}	x_{n-3}	$\cdots$	x_0	0
$-X =$	0	$-x_{n-1}$	$-x_{n-2}$	$\cdots$	x_1	x_0
		z_{n-1}	z_{n-2}	$\cdots$	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} που χρίζει ανάλυσης προκύπτει από την παρακάτω σχέση:

$z_{n-1} = x_{n-2} - x_{n-1}$

Επομένως, ο αριθμός X δίνεται από την σχέση:

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

με z_i να είναι τα ψηφία που προέκυψαν από την κωδικοποίηση του X. Αυτή είναι και η κωδικοποίηση Booth και το μεγάλο της πλεονέκτημα είναι ότι εφαρμόζεται σε κάθε αριθμό σε μορφή συμπληρώματος ως προς δύο, ανεξάρτητα από το πρόσημο του.

Έστω το γινόμενο $P = X \cdot Y$, με $X = x_{n-1}x_{n-2}\ldots x_1x_0$ και $Y = y_{n-1}y_{n-2}\ldots y_1y_0$. Για το γινόμενο αυτό ισχύει η παρακάτω σχέση σύμφωνα με τον αλγόριθμο του Booth:

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

Η παραπάνω σχέση υποδεικνύει ότι σε κάθε βήμα i, το X πολλαπλασιάζεται με ένα από τα στοιχεία του συνόλου {-1, 0, 1} σύμφωνα με το αποτέλεσμα της αφαίρεσης των δύο διαδοχικών ψηφίων του πολλαπλασιαστή Y. Στην πραγματικότητα με την κωδικοποίηση Booth ελέγχουμε τις σειρές από μονάδες που παρουσιάζονται μέσα σε κάθε δυαδικό αριθμό. Πιο συγκεκριμένα, ο έλεγχος που γίνεται σε σχέση με τα κωδικοποιημένα ψηφία φαίνεται στον Πίνακα 2.1.

Πίνακας 2.1 Έλεγχος ύπαρξης μονάδων σε σχέση με τα κωδικοποιημένα ψηφία

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 σειράς από 1
1	**0**	-1	Αρχή σειράς από 1
1	**1**	0	Μέση σειράς από 1

Συνοψίζοντας, η κωδικοποίηση Booth παρουσιάζει τα παρακάτω πλεονεκτήματα:

- Η κωδικοποίηση Booth θετικών και αρνητικών αριθμών γίνεται ακριβώς με τον ίδιο τρόπο όπως αναφέραμε και πιο πάνω.
- Κάθε ψηφίο του κωδικοποιημένου αριθμού παράγεται ανεξάρτητα από τα άλλα αφού είναι συνάρτηση μόνο των δύο bit Y_iY_{i-1}. Αυτό σημαίνει ότι και κάθε μερικό γινόμενο σε έναν πολλαπλασιασμό όπου χρησιμοποιείται η κωδικοποίηση Booth παράγεται αμέσως. Το γεγονός αυτό βοηθά στη γρήγορη εκτέλεση του πολλαπλασιασμού, καθώς τα μερικά γινόμενα μπορούν να διοχετευθούν παράλληλα σε ένα δένδρο carry save αθροιστών για την παραγωγή του τελικού γινομένου.

Το μεγαλύτερο πλεονέκτημα είναι ότι η κωδικοποίηση Booth 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. Από την άλλη, αν ο αριθμός είχε απομονωμένες μονάδες, τότε είναι πολύ πιθανόν η κωδικοποίηση Booth να μην δώσει τα αναμενόμενα αποτελέσματα και να αυξήσει τον αριθμό των μη μηδενικών μερικών γινομένων και έτσι να αυξήσει το κύκλωμα υλοποίησης της πράξης στην οποία λαμβάνει μέρος ο αριθμός. Για να αντιμετωπιστεί αυτό το πρόβλημα, προτάθηκε μια διαφοροποιημένη μέθοδος κωδικοποίησης, η οποία παρουσιάζεται στη συνέχεια.

2.2.2 Κωδικοποίηση modified booth

Μια παραλλαγή της απλής κωδικοποίησης Booth είναι ο τροποποιημένος αλγόριθμος Booth που επεκτείνει το σύνολο των ψηφίων κωδικοποίησης και προκύπτει αλγεβρικά από τον απλό αλγόριθμο του Booth. Σε αντίθεση με τον απλό αλγόριθμο κωδικοποίησης Booth αυτή η μέθοδος κωδικοποιεί τους αριθμούς από την μορφή συμπληρώματος ως προς δύο στο σύνολο {-2, -1, 0 , +1, +2}.

Στην αμέσως προηγούμενη παράγραφο είδαμε πως ισχύει η σχέση:

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

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

Αυτή η σχέση μπορεί να αναλυθεί περαιτέρω ως εξής:

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

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

Με το w_i να είναι το κωδικοποιημένο ψηφίο με Modified Booth και μπορεί να αναλυθεί ως εξής:

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

Όπως φαίνεται και από την παραπάνω σχέση ο τροποποιημένος αλγόριθμος Booth δεν κωδικοποιεί δύο μόνο ψηφία του δυαδικού αριθμού αλλά πάντα τριάδες ψηφίων. Οι τριάδες αυτές είναι επικαλυπτόμενες κατά ένα ψηφίο και τέλος για την σωστή κωδικοποίηση θεωρούμε το χαμηλότερου βάρους (LSB) bit προς κωδικοποίηση ότι είναι το 0.

Στον Πίνακα 2.2 φαίνεται η αντιστοίχηση τόσο των δυαδικών ψηφίων όσο και των κωδικοποιημένων σε απλό Booth με τα bits της τροποποιημένης κωδικοποίησης:

Πίνακας 2.2 Αντιστοίχηση δυαδικών ψηφίων με απλή κωδ. Booth & Modified Booth

			Κωδικοποίηση Booth		**Κωδικοποίηση Modified**
$\mathbf{Y_{2i+1}}$	$\mathbf{Y_i}$	$\mathbf{Y_{2i-1}}$	$\mathbf{Z_{2i+1}}$	$\mathbf{Z_{2i}}$	**Booth: $\mathbf{W_i}$**
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

Η κωδικοποίηση Modified Booth χρησιμοποιείται ευρέως για την υλοποίηση πολλαπλασιαστών. Κάθε κωδικοποιημένο ψηφίο καθορίζει τη λειτουργία που πρόκειται να γίνει στον πολλαπλασιασμό ακριβώς όπως και στον απλό αλγόριθμο κωδικοποίησης Booth. Έστω ότι πρόκειται να πολλαπλασιάσουμε δύο αριθμούς 8-bit A και B, όπου B είναι ο πολλαπλασιαστής. Τα ψηφία του B κωδικοποιούνται κατά Modified Booth. Τα Modified Booth ψηφία που προκύπτουν, ανάλογα με την τιμή τους καθορίζουν τις λειτουργίες που θα εκτελεστούν. Οι λειτουργίες αυτές φαίνονται στον Πίνακα 2.3.

Πίνακας 2.3 Αντιστοιχία λειτουργιών και κωδικοποιημένων ψηφίων

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

Για να γίνει κατανοητή η διαδικασία πολλαπλασιασμού δύο αριθμών παρακάτω παρατίθεται ένα παράδειγμα όπου ο πολλαπλασιαστής κωδικοποιείται με τον αλγόριθμο Modified Booth:

Έστω A = $10110101_{(2)} = -75_{(10)}$ και B=$01110010_{(2)}=114_{(10)}$.

Κωδικοποιημένος ο B με τον αλγόριθμο Modified Booth γίνεται:

B= $2\ \overline{1}\ 1\ \overline{2}$.

Επειδή θα μας χρειαστούν οι αριθμοί 2A, $\overline{A}$ και $2\overline{A}$ τους υπολογίζουμε:

2A = 101101010, $\overline{A}$ = 01001011 και $2\overline{A}$ = 010010110.

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

Πίνακας 2.4 Παράδειγμα πολλαπλασιασμού με χρήση κωδικοποίησης Modified Booth

A = 10110101 B = 01110010 Μερικό γινόμενο = 0 0 0 0 0 0 0 0	
$\underline{0}$ 0 1 0 0 1 0 1 1 0	-2 Πρόσθεσε -2A (Πρώτο μερικό γινόμενο)
1 0 1 1 0 1 0 1	+1 Πρόσθεσε A (Δεύτερο μερικό γινόμενο)
$\underline{1}$ $\underline{1}$ 1 1 0 1 1 0 1 0 1 0	Άθροισμα των δύο πρώτων μερικών γινομένων
0 1 0 0 1 0 1 1	−1 Πρόσθεσε −A (Τρίτο μερικό γινόμενο)
$\underline{0}$ $\underline{0}$ $\underline{0}$ 0 1 0 0 0 0 0 0 1 1 0 1 0	Άθροισμα των τριών μερικών γινομένων
1 0 1 1 0 1 0 1 0	+2 Πρόσθεσε +2A (Τέταρτο μερικό γινόμενο)
1 0 1 1 1 1 0 1 0 0 1 1 0 1 0	Τελικό αποτέλεσμα = -8550

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

Ο τροποποιημένος αλγόριθμος του Booth όπως και ο απλός έχει σαν μεγάλο του πλεονέκτημα το ότι εφαρμόζεται ανεξάρτητα από το αν οι αριθμοί είναι σε απλή δυαδική ή σε μορφή συμπληρώματος ως προς 2. Επίσης με τον τροποποιημένο αλγόριθμο του Booth έχουμε περαιτέρω μείωση του αριθμού των μερικών γινομένων και επομένως λιγότερες

προσθέσεις να εκτελέσουμε. Προφανώς αυτό συμβάλει τόσο στην αύξηση της ταχύτητας του πολλαπλασιαστή όσο και στην μείωση της επιφάνειας που καταλαμβάνει.

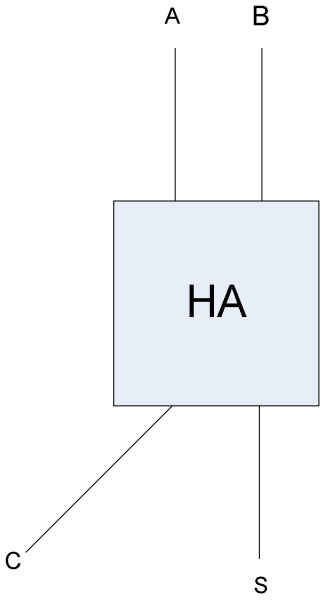
2.3 Είδη αθροιστών και πολλαπλασιαστών

2.3.1 Δομικά στοιχεία

Κάθε αθροιστής ή πολλαπλασιαστής αποτελείται από κάποια βασικά στοιχεία τα οποία χρησιμοποιούνται σαν μονάδες πρόσθεσης από το όλο κύκλωμα. Αυτές οι μονάδες είναι ο ημιαθροιστής και ο πλήρης αθροιστής. Κάθε κύκλωμα που επιτελεί κάποια επεξεργασία πάνω σε αριθμητικά δεδομένα χρησιμοποιεί αυτά τα δομικά στοιχεία σε κάποια συγκεκριμένη διάταξη για να επιτελέσει την λειτουργία που πρέπει σωστά.

2.3.1.1 Ημιαθροιστής

Ο ημιαθροιστής που ονομάζεται και Half Adder έχει δύο εισόδους, όπου και οι δύο είναι ψηφία της ίδιας βαθμίδας και επιτελεί την πρόσθεση αυτών των δύο ψηφίων έχοντας δύο εξόδους, η μία στην ίδια βαθμίδα με τις εισόδους (Save, S) και μία άλλη στην αμέσως επόμενη(Carry, C). Το σχήμα του ημιαθροιστή παρατίθεται στο Σχήμα 2.1.



Σχήμα 2.1 Ημιαθροιστής

Οι έξοδοι σε σχέση με τις εισόδους για τον ΗΑ ικανοποιούν τις παρακάτω σχέσεις:

$$S = A \oplus B$$

$$C = A \cdot B$$

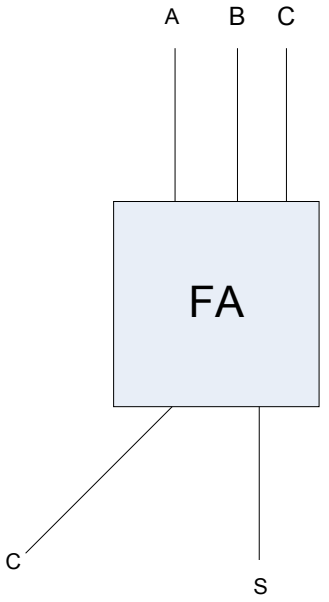
Στον Πίνακα 2.5 παρουσιάζονται οι αντιστοιχίες εισόδου εξόδου.

Πίνακας 2.5 Πίνακας αληθείας ημιαθροιστή

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

2.3.1.2 Πλήρης αθροιστής

Ο πλήρης αθροιστής που ονομάζεται και Full Adder έχει τρεις εισόδους, όπου και οι τρεις είναι ψηφία της ίδιας βαθμίδας και επιτελεί την πρόσθεση αυτών των ψηφίων έχοντας δύο εξόδους, η μία στην ίδια βαθμίδα με τις εισόδους (Save, S) και μία άλλη στην αμέσως επόμενη (Carry, C). Το σχήμα του πλήρη αθροιστή παρατίθεται στο Σχήμα 2.2.



Σχήμα 2.2 Πλήρης αθροιστής

Οι έξοδοι σε σχέση με τις εισόδους για τον FA ικανοποιούν τις παρακάτω σχέσεις:

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

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

Στον Πίνακα 2.6 παρουσιάζονται οι αντιστοιχίες εισόδου εξόδου του FA.

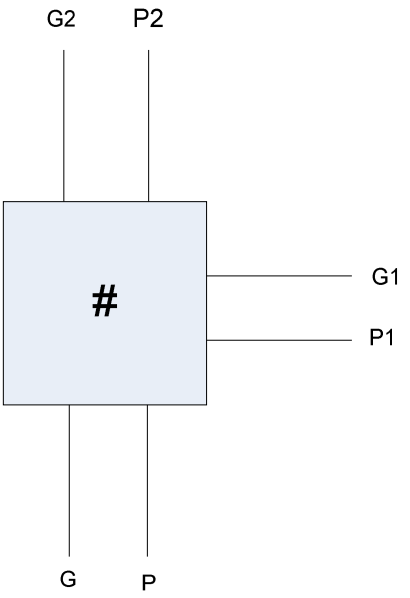
Πίνακας 2.6 Πίνακας αληθείας πλήρη αθροιστή

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

2.3.1.3 Κύκλωμα πρόβλεψης κρατουμένου

Το κύκλωμα πρόβλεψης κρατουμένου αποτελεί δομικό στοιχείο των αθροιστών πρόβλεψης κρατουμένου. Δέχεται σαν εισόδους τα σήματα G1, P1, G2, P2 τα οποία είναι τα σήματα γέννησης (G, Generation) και διάδοσης (P, Propagation) κρατουμένου δύο χαμηλότερων βαθμίδων ενώ δίνει σαν έξοδο τα σήματα G,P τα οποία αποτελούν δύο νέα σήματα γεννήσεως και διάδοσης κρατουμένου.

Το κύκλωμα παριστάνεται στο Σχήμα 2.3.



Σχήμα 2.3 Κύκλωμα πρόβλεψης κρατουμένου

Ενώ οι έξοδοι σε σχέση με τις εισόδους για αυτή την μονάδα ικανοποιούν τις παρακάτω σχέσεις:

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

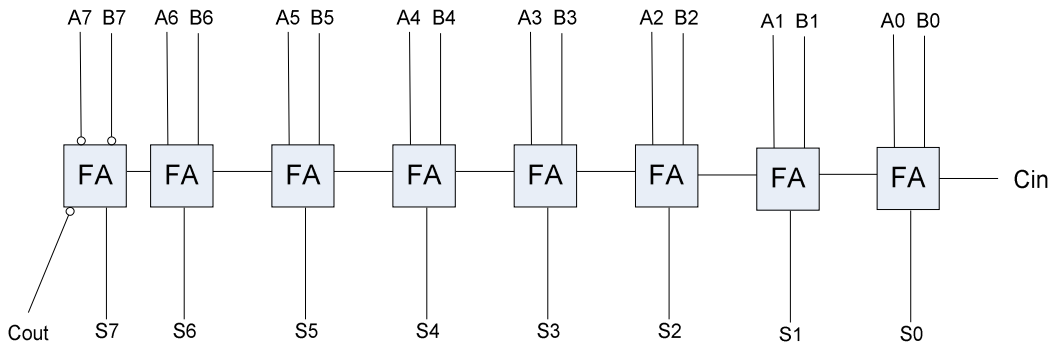
$$P = P1 \cdot P2$$

2.3.2 Αθροιστές

2.3.2.1 Αθροιστής διάδοσης κρατουμένου

Ο αθροιστής διάδοσης κρατουμένου (carry propagate adder – CPA) αποτελείται από συνδεδεμένους πλήρεις αθροιστές, όπου κάθε αθροιστής αντιστοιχεί σε μία βαθμίδα. Η έξοδος του κρατουμένου (Carry) του κάθε αθροιστή συνδέεται με την θύρα εισόδου του αθροιστή της επόμενης βαθμίδας ενώ το αποτέλεσμα της πράξης αποτελείται από όλα τα SAVE ψηφία των δομικών μονάδων (Full adders) και το κρατούμενο εξόδου της πιο σημαντικής βαθμίδας. Έτσι τα κρατούμενα διαδίδονται μέσα στην δομή του αθροιστή μέχρι την υψηλότερη βαθμίδα, ο οποίος είναι και ο λόγος που ο αθροιστής πήρε αυτό το όνομα.

Ο CPA μπορεί να δεχτεί σαν είσοδο είτε δύο δυαδικούς αριθμούς σε μορφή συμπληρώματος ως προς δύο, είτε έναν carry save αριθμό σε μορφή συμπληρώματος ως προς δύο. Προφανώς η έξοδος είναι ένας δυαδικός αριθμός σε μορφή συμπληρώματος ως προς δύο και διαμορφώνεται όπως περιγράψαμε παραπάνω. Η διασύνδεση ενός αθροιστή διάδοσης κρατουμένου 8-bit φαίνεται στο Σχήμα 2.4.



Σχήμα 2.4 Κύκλωμα αθροιστή διάδοσης κρατουμένου

Όπως φαίνεται και από το σχήμα στην τελευταία βαθμίδα οι είσοδοι καθώς και η έξοδος είναι αρνητικής αξίας.

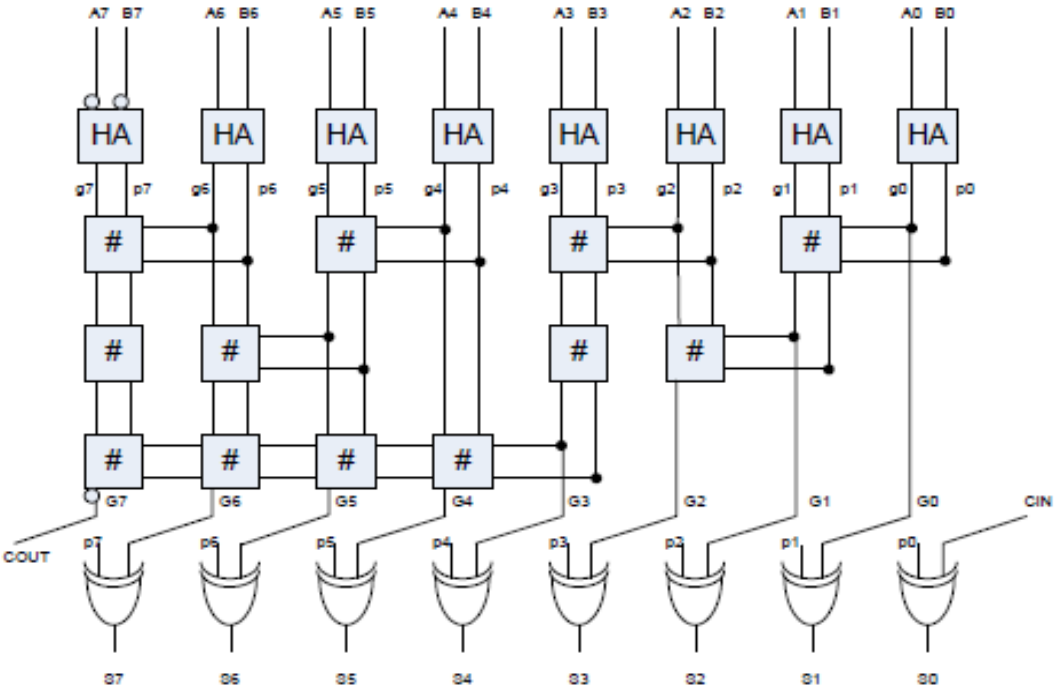
Το τελικό αποτέλεσμα είναι: {Cout,S7,S6,S5,S4,S3,S2,S1,S0}

Αν και ο CPA έχει πολύ απλή δομή και πολύ μικρή επιφάνεια, η διάδοση του κρατουμένου καθιστά απαγορευτική την χρησιμοποίηση του καθώς είναι πολύ πιο αργός από τους αθροιστές που χρησιμοποιούνται.

2.3.2.2 Αθροιστής πρόβλεψης κρατουμένου

Ο αθροιστής πρόβλεψης κρατουμένου (Carry lookahead adder – CLA) για να αποφύγει την καθυστέρηση από την διάδοση κρατουμένου χρησιμοποιεί ημιαθροιστές, κυκλώματα πρόβλεψης κρατουμένου και πύλες XOR (αποκλειστικού-ή).

Ο CPA λειτουργεί αθροιστικά σαν τον CLA και όπως πριν μπορεί να δεχτεί σαν είσοδο είτε δύο δυαδικούς αριθμούς σε μορφή συμπληρώματος ως προς δύο, είτε έναν carry save αριθμό σε μορφή συμπληρώματος ως προς δύο. Προφανώς η έξοδος είναι ένας δυαδικός αριθμός σε μορφή συμπληρώματος ως προς δύο και αποτελεί το άθροισμα των δύο αριθμών. Η διασύνδεση ενός αθροιστή πρόβλεψης κρατουμένου 8-bit φαίνεται και στο Σχήμα 2.5.



Σχήμα 2.5 Κύκλωμα αθροιστή πρόβλεψης κρατουμένου

Όπως φαίνεται και από το σχήμα στην τελευταία βαθμίδα οι είσοδοι καθώς και η έξοδος είναι αρνητικής αξίας.

Το τελικό αποτέλεσμα είναι: {Cout,S7,S6,S5,S4,S3,S2,S1,S0}

Επίσης από το σχήμα φαίνεται πως ο αθροιστής πρόβλεψης κρατουμένου είναι πιο γρήγορος από τον CPA καθώς αποφεύγει την διάδοση κρατουμένου αφού η άθροιση γίνεται χρησιμοποιώντας κυκλώματα πρόβλεψης κρατουμένου. Αφού όμως η δομή του CLA είναι λιγότερο απλή σε σχέση με την δομή του CPA, εύκολα εξάγεται το συμπέρασμα ότι η επιφάνεια που καλύπτει ο CLA θα είναι αρκετά μεγαλύτερη από την επιφάνεια του CPA. Για μεγάλα μήκη λέξεων η επιφάνεια που καταλαμβάνει ένας αθροιστής διάδοσης κρατουμένου είναι εξαιρετικά μεγάλη αφού η συνάρτηση επιφάνειας/μήκους λέξης είναι πολυπλοκότητας O(N·logN), αλλά χρησιμοποιείται λόγω του ότι ο ρυθμός με τον οποίο αυξάνεται το κέρδος σε ταχύτητα ενός CLA είναι μικρότερος από τον ρυθμό με τον οποίο αυξάνεται η επιφάνεια του καθώς μεγαλώνει το μήκος λέξης.

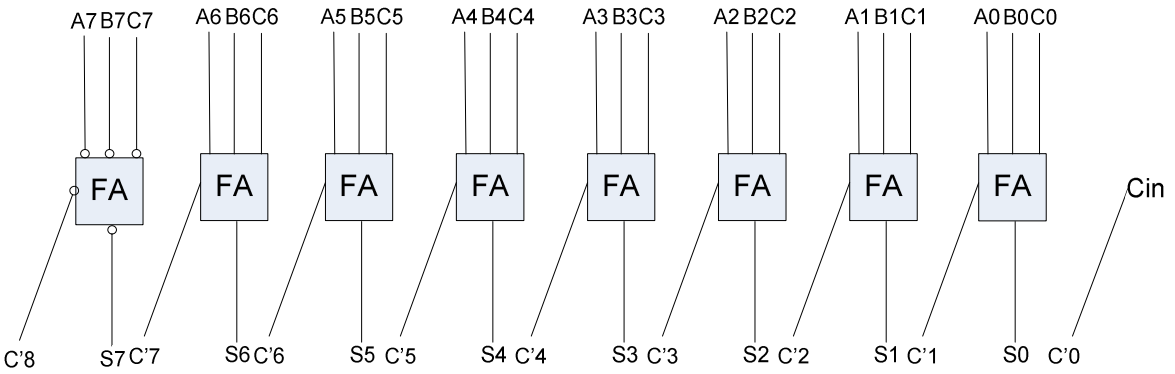
Εύκολα κανείς καταλαβαίνει ότι ενώ ο CLA υπερτερεί σε χρόνο, ο CPA υπερτερεί σε επιφάνεια, οπότε ανάλογα με την εκάστοτε εφαρμογή που αναπτύσσεται αν δεν υπάρχει συγκεκριμένο όριο στον χρόνο μπορεί να επιλεγεί ο CPA που προσφέρει την ελάχιστη δυνατή επιφάνεια για την πρόσθεση μιας σειράς από bit, ενώ αν δεν υπάρχει συγκεκριμένο

όριο στην επιφάνεια μπορεί κάλλιστα να επιλεγεί ο CLA που προσφέρει το αποτέλεσμα στον ελάχιστο δυνατό χρόνο.

2.3.2.3 Αθροιστής σωσίματος κρατουμένου

Ο Αθροιστής σωσίματος κρατουμένου (Carry save adder – CSA) όπως και ο carry propagation adder χρησιμοποιεί ως δομικές μονάδες πλήρεις αθροιστές. Παρομοίως κάθε FA αντιστοιχεί σε μία συγκεκριμένη βαθμίδα αλλά η διαφορά έγκειται στο ότι ο CSA δρα σαν συμπιεστής 3 σειρών από bit σε 2. Η έξοδος κάθε πλήρους αθροιστή δεν είναι παράλληλα είσοδος σε κάποιον άλλο, απλά στην περίπτωση του Carry αυτό το bit θα πάει σε μία παραπάνω βαθμίδα στην έξοδο. Εν ολίγοις, ενώ ο CSA δέχεται σαν είσοδο είτε 3 δυαδικούς αριθμούς, είτε έναν αριθμό σε carry save μορφή και έναν δυαδικό, πάντα στην έξοδο του παράγει έναν αριθμό σε μορφή carry save.

Στο Σχήμα 2.6 παρουσιάζεται η δομή ενός 8-bit CSA.



Σχήμα 2.6 Κύκλωμα αθροιστή σωσίματος κρατουμένου

Όπως φαίνεται και από το σχήμα στην τελευταία βαθμίδα οι είσοδοι καθώς και οι έξοδοι αφού είναι τα MSB των Carry & Save, είναι αρνητικής αξίας.

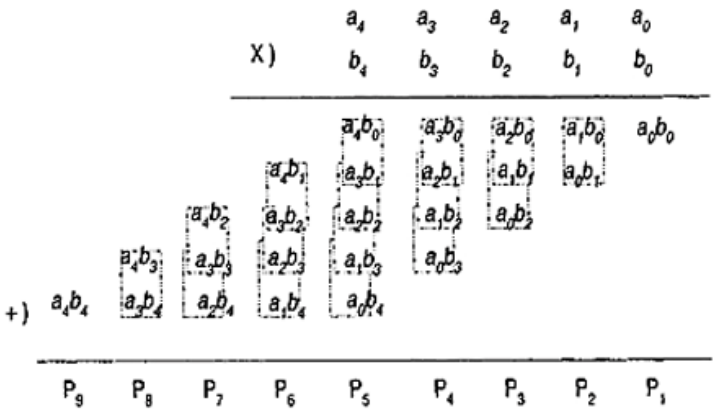
Το αποτέλεσμα του αθροιστή σωσίματος κρατουμένου είναι σε CS μορφή με

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}.

Από το σχήμα βλέπουμε πως ενώ ένας CPA αθροιστής με μήκος λέξης N bit καθυστερεί κατά N επίπεδα πλήρων αθροιστών, ένας CSA με ίδιο μήκος λέξης καθυστερεί μόνο κατά ένα επίπεδο ενώ έχουν την ίδια επιφάνεια τα δύο αυτά κυκλώματα. Τέλος πρέπει να επισημανθεί για τον CSA ότι δεν υπολογίζει κάποιο τελικό αποτέλεσμα, αλλά μόνο συμπιέζει τις 3 σειρές από bit σε 2 (3 to 2 compressor) και για να υπολογιστεί το τελικό αποτέλεσμα θα χρειαστεί κάποιος CLA ή CPA ανάλογα με τις απαιτήσεις της εφαρμογής.

2.3.3 Πολλαπλασιαστές

Έστω ότι θέλουμε να πολλαπλασιάσουμε δύο αριθμούς Α και Β που ο καθένας τους είναι 5 bit. Οι πράξεις που πρέπει να γίνουν συνοψίζονται στο Σχήμα 2.7.



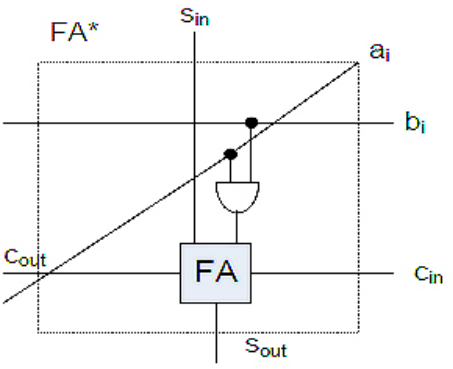
Σχήμα 2.7 Πράξεις πολλαπλασιασμού

2.3.3.1 Πολλαπλασιαστές τύπου πίνακα

Οι πολλαπλασιαστές τύπου πίνακα που παρατίθενται είναι ο παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου και ο παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου. Βασικό πλεονέκτημα και των δύο μεθόδων είναι πως η δομή των πολλαπλασιαστών παραμένει η ίδια και είναι ομοιόμορφη και πως είναι αρκετά εύκολη η μετατροπή τους σε συνεχούς διοχέτευσης και συστολικούς. Και οι δύο πολλαπλασιαστές χρησιμοποιούν N^2 πλήρεις αθροιστές αφού για κάθε μερικό γινόμενο με μήκος λέξης Ν χρησιμοποιούν Ν πλήρεις αθροιστές. Κάθε νέα σειρά από FA είναι ολισθημένη κατά μια βαθμίδα προς τα αριστερά συγκρινόμενη με την σειρά που βρίσκεται από πάνω της.

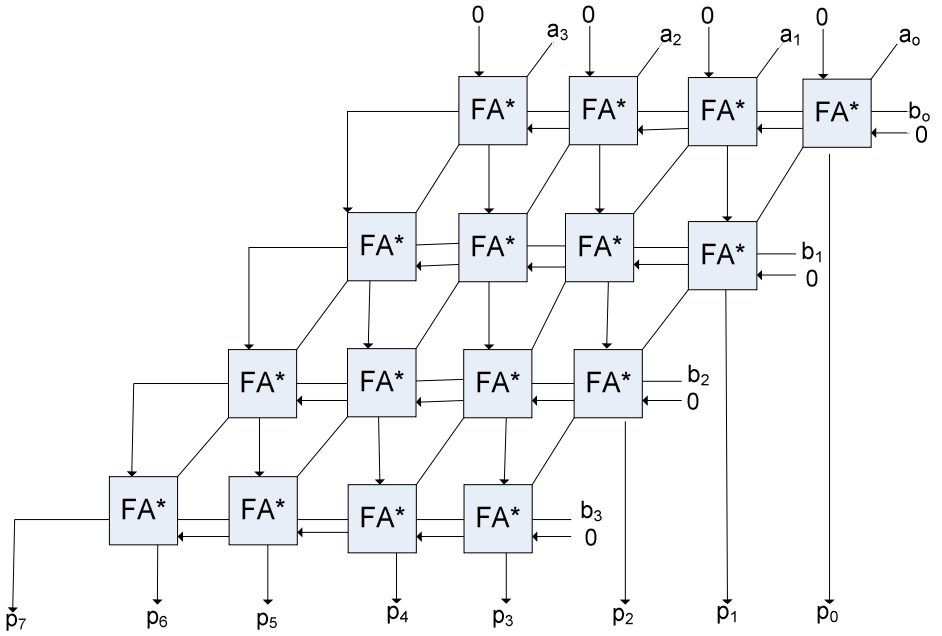
Α. Παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου

Ο παράλληλος πολλαπλασιαστής με διάδοση κρατουμένου χρησιμοποιεί μία δομική μονάδα που χρησιμοποιεί έναν πλήρη αθροιστή και μία πύλη AND. Το Σχήμα 2.8 απεικονίζει τον FA*.



Σχήμα 2.8 Κύκλωμα FA*

και όλο το σχήμα του παράλληλου πολλαπλασιαστή με διάδοση κρατουμένου παρουσιάζεται στο Σχήμα 2.9 για έναν πολλαπλασιαστή 4-bit:



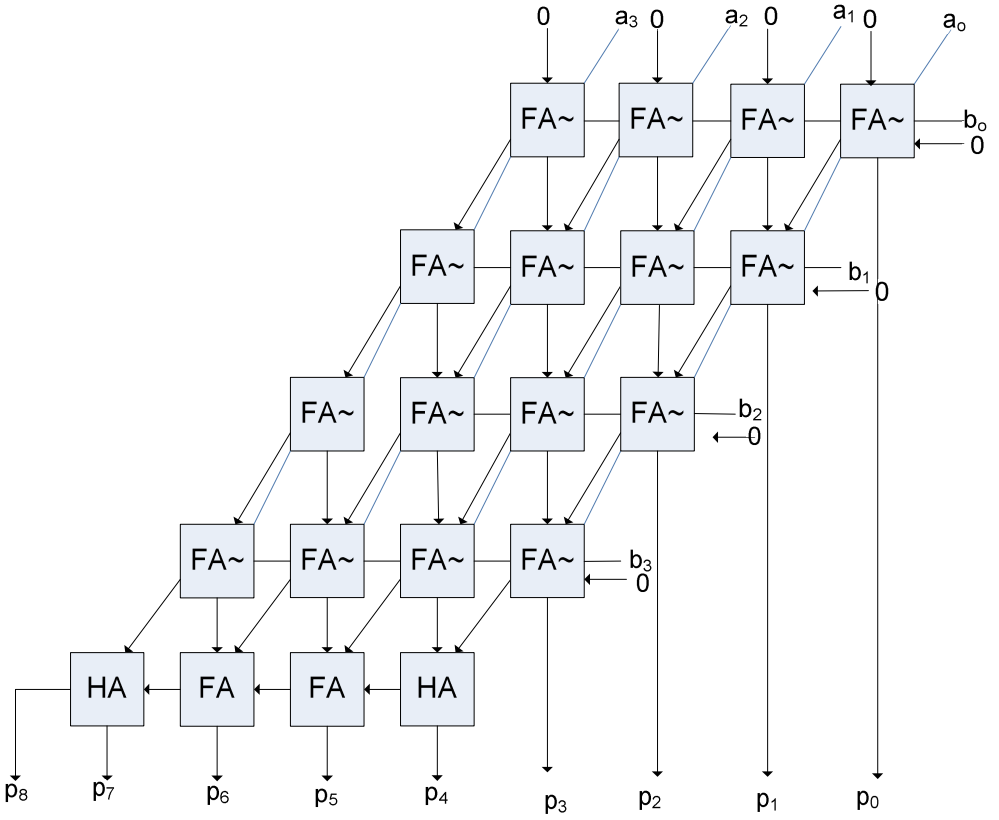
Σχήμα 2.9 Κύκλωμα παράλληλου πολλαπλασιαστή με διάδοση κρατουμένου

Όπως φαίνεται και από το σχήμα το κρίσιμο μονοπάτι για τον παράλληλο πολλαπλασιαστή με διάδοση κρατουμένου τεσσάρων bit αποτελείται από 10 FA*, ενώ χρειάζεται ειδική μεταχείριση για τα MSB εισόδου (a_3 , b_3) καθώς και για το MSB εξόδου (p_7) αφού αυτά είναι αρνητικής αξίας.

Β. Παράλληλος πολλαπλασιαστής με σώσιμο κρατουμένου

Στην περίπτωση του παράλληλου πολλαπλασιαστή με σώσιμο κρατουμένου χρησιμοποιείται σαν δοκιμή μονάδα μία διαφορετική μονάδα από τον FA* όχι ως προς την ουσία αλλά ως προς την διάταξη των γραμμών, η οποία ονομάζεται FA~.

Η μορφή του όλου δικτύου του πολλαπλασιαστή (για δύο 4-bit αριθμούς) φαίνεται στο Σχήμα 2.10.



Σχήμα 2.10 Κύκλωμα παράλληλου πολλαπλασιαστή με σώσιμο κρατουμένου

Στο τέλος της τέταρτης βαθμίδας έχουμε τα τέσσερα πιο σημαντικά bits του αποτελέσματος σε μορφή αθροίσματος-κρατουμένου και για αυτόν τον λόγο πρέπει να χρησιμοποιήσουμε ή έναν απλό αθροιστή διάδοσης κρατουμένου όπως στο σχήμα ή έναν αθροιστή πρόβλεψης κρατουμένου. Τέλος η ταχύτητα ώστε να παραχθεί το αποτέλεσμα είναι μικρότερη από το κύκλωμα του παράλληλου πολλαπλασιαστή με διάδοση κρατουμένου αφού το κρίσιμο μονοπάτι εδώ αποτελείται από 4 FA~, 2 FA και 2 HA.

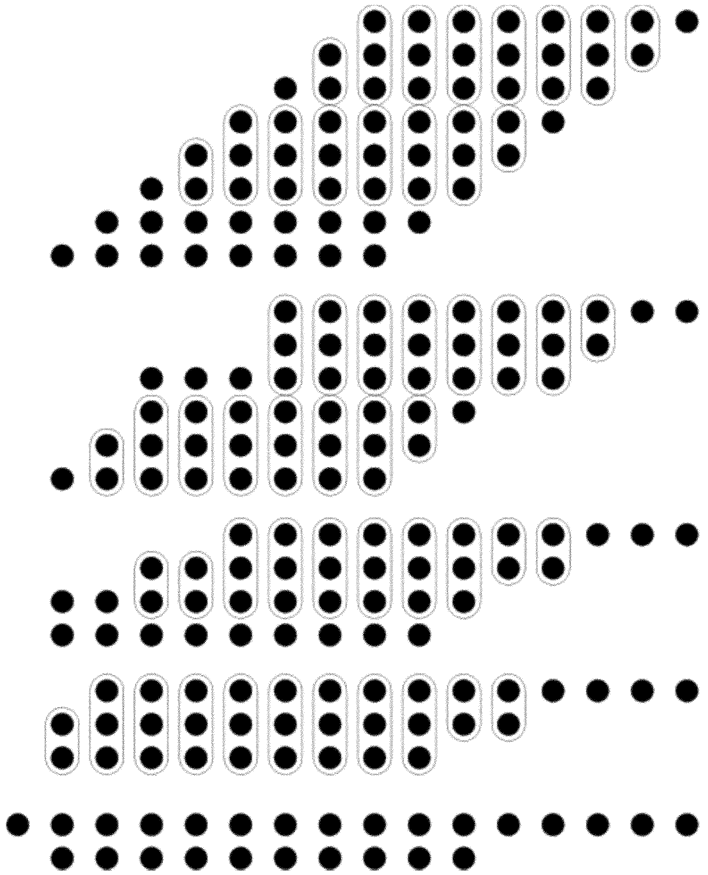
2.3.3.2 Δενδρικοί πολλαπλασιαστές

Οι δενδρικοί πολλαπλασιαστές που παρουσιάζονται είναι ο πολλαπλασιαστής Wallace και ο πολλαπλασιαστής Dadda. Η δημιουργία των μερικών γινομένων στους δενδρικούς πολλαπλασιαστές γίνεται ανεξάρτητα από το στάδιο της πρόσθεσης και ύστερα τα μερικά γινόμενα οδηγούνται σε συμπιεστές όπου ανάλογα με την μέθοδο που χρησιμοποιείται αλλάζει το δίκτυο των FA & HA που χρησιμοποιείται όμως στο τέλος παράγεται ένας αριθμός σε μορφή αθροίσματος- κρατουμένου. Στο επόμενο στάδιο, ο αριθμός αυτός χρησιμοποιείται από έναν CPA ή έναν CLA αθροιστή για να παραχθεί το τελικό αποτέλεσμα σε δυαδική μορφή.

Γνωρίζοντας πως η δημιουργία των μερικών γινομένων γίνεται ανεξάρτητα από το στάδιο της δενδρικής πρόσθεσης, είναι δυνατή η κωδικοποίηση του ενός από τους δύο αριθμούς ή και των δύο με χρήση την κωδικοποίηση Booth ή Modified Booth. Με αυτόν τον τρόπο μπορεί να μειωθεί ο αριθμός των μερικών γινομένων έτσι ώστε να αυξηθεί η ταχύτητα του δενδρικού πολλαπλασιαστή.

Α. Δενδρικός πολλαπλασιαστής Wallace

Ο πολλαπλασιαστής Wallace χρησιμοποιεί ένα δίκτυο από FA και HA για να συμπιέσει τα μερικά γινόμενα σε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου. Στόχος του πολλαπλασιαστή Wallace είναι να γίνει η συμπίεση αυτή 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, οι σειρές των μερικών γινομένων ομαδοποιούνται ανά τρεις. Εάν δύο σειρές περισσέψουν, τότε ομαδοποιούνται ανά δύο και εισέρχονται σαν είσοδοι σε έναν HA. Αν περισσέψει μία μόνο σειρά αφήνεται ως έχει μέχρι το επόμενο επίπεδο. Στις σειρές που ομαδοποιήθηκαν, για κάθε βαθμίδα στην οποία υπάρχουν τρία bits χρησιμοποιείται ένας FA και για κάθε βαθμίδα στην οποία υπάρχουν 2 bits χρησιμοποιείται ένας HA. Κάθε FA και HA δίνει ως έξοδο ένα bit στην ίδια βαθμίδα (SAVE) και ένα bit στην επόμενη (CARRY). Στο Σχήμα 2.11 παρουσιάζεται ο τρόπος με τον οποίο ένας 8x8-bit πολλαπλασιαστής Wallace συμπιέζει τα μερικά γινόμενα.



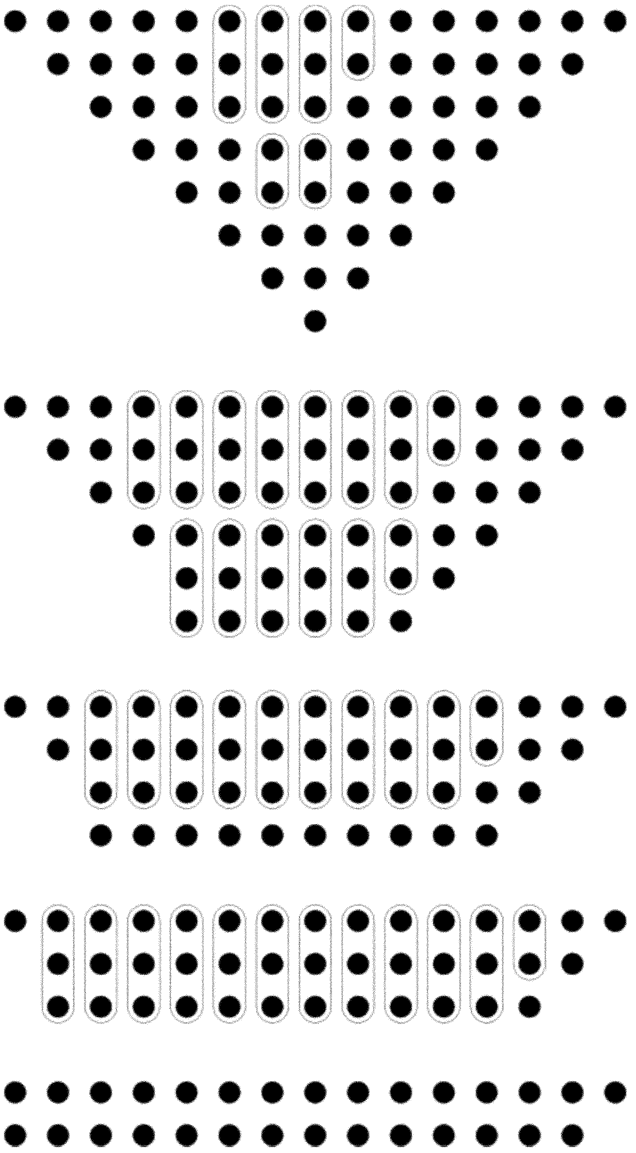
Σχήμα 2.11 Wallace δενδρικός συμπιεστής

Οι τελείες προφανώς συμβολίζουν τα bits των μερικών γινομένων και όπου υπάρχει ομαδοποίηση 3 bit, τα 3 αυτά bit εισέρχονται σαν είσοδοι σε έναν FA και όπου υπάρχει ομαδοποίηση 2 bit, τα 2 αυτά bit εισέρχονται σαν είσοδοι σε έναν HA.

Β. Δενδρικός πολλαπλασιαστής Dadda

Ο δενδρικός πολλαπλασιαστής Dadda ακολουθεί μία διαφορετική προσέγγιση στην συμπίεση των bit των μερικών γινομένων έχοντας όμως ως κοινό παράγοντα με τον Wallace ότι και αυτός χρησιμοποιεί ένα δίκτυο από FA & HA και ως τελικό αποτέλεσμα δίνει έναν αριθμό σε μορφή αθροίσματος-κρατουμένου. Ο δενδρικός πολλαπλασιαστής Dadda προσπαθει σε κάθε βήμα να εξομαλύνει όσο γίνεται το ύψος του δέντρου ώστε στο επόμενο βήμα να υπάρχει μία ομοιομορφία στους FA και στους HA που θα χρησιμοποιηθούν.

Στο Σχήμα 2.12 παρουσιάζεται ο τρόπος με τον οποίο ένας 8x8-bit πολλαπλασιαστής Dadda συμπιέζει τα μερικά γινόμενα.



Σχήμα 2.12 Dadda δενδρικός συμπιεστής

Όπως και στον Wallace, έτσι και εδώ ισχύει ότι οι τελείες συμβολίζουν τα bits των μερικών γινομένων και όπου υπάρχει ομαδοποίηση 3 bit, τα 3 αυτά bit εισέρχονται σαν είσοδοι σε έναν FA και όπου υπάρχει ομαδοποίηση 2 bit, τα 2 αυτά bit εισέρχονται σαν είσοδοι σε έναν HA.

Έχοντας δει τις κύριες μορφές πολλαπλασιαστών, οι πολλαπλασιαστές τύπου πίνακα παρουσιάζουν το βασικό πλεονέκτημα ότι μπορούν να μετατραπούν εύκολα σε συστολικούς και να λειτουργούν σε κατάσταση συνεχούς διοχέτευσης λόγω της κανονικότητας της δομής τους, ενώ οι δενδρικοί πολλαπλασιαστές έχουν το πλεονέκτημα ότι καταλαμβάνουν μικρότερη επιφάνεια σε σχέση με τους πρώτους λόγω των λιγότερων FA & HA και συνεπώς τα αποτελέσματα είναι διαθέσιμα σε μικρότερο χρόνο λόγω του μικρότερου κρίσιμου μονοπατιού (critical path).

3

ΑΛΓΟΡΙΘΜΟΙ BLOCK

Αυτό το κεφάλαιο πραγματεύεται τους αλγόριθμους block που υλοποιήθηκαν και αναλύει τις υλοποιήσεις αυτών με τα επιμέρους δομικά στοιχεία που χρησιμοποιούνται από κάθε αλγόριθμο.

3.1 Γενική μορφή αλγορίθμων

Οι αλγόριθμοι που υλοποιήθηκαν αποτελούνται από 3 ή 4 διακριτά μέρη και όλοι αφορούν πολλαπλασιασμό δύο δυαδικών αριθμών στην μορφή συμπληρώματος ως προς δύο.

Συγκεκριμένα τα μέρη αυτά είναι

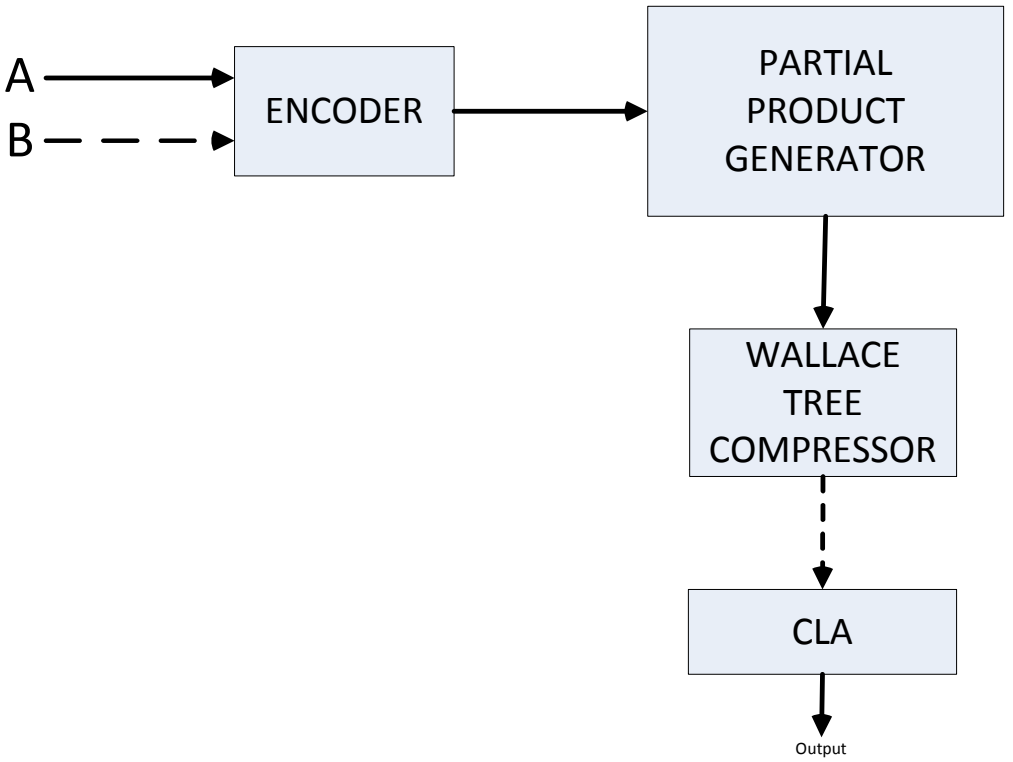
α) Η μονάδα που κωδικοποιεί (encoder) τον έναν ή και τους δύο αριθμούς

β) Η μονάδα που παράγει τα μερικά γινόμενα (partial product generator)

γ) Η μονάδα που συμπιέζει τα μερικά γινόμενα σε έναν αριθμό μορφής carry save (wallace tree compressor)

δ) (προαιρετικά) Η μονάδα που παίρνει ως είσοδο τον carry save αριθμό από το προηγούμενο βήμα και δίνει ως έξοδο έναν αριθμό στην μορφή συμπληρώματος ως προς δύο (carry look-ahead adder- CLA)

Οι μονάδες αυτές απεικονίζονται στο Σχήμα 3.1.



Σχήμα 3.1 Γενική δομή πολλαπλασιαστών

3.2 Αλγόριθμος WHOLE

Ο αλγόριθμος WHOLE υλοποιήθηκε για να συγκριθούν οι υπόλοιποι αλγόριθμοι πολλαπλασιασμού και αποτελεί τον κλασσικό αλγόριθμο πολλαπλασιασμού δύο αριθμών σύμφωνα με το Σχήμα 3.1.

3.2.1 Γενική δομή αλγορίθμου WHOLE

Αυτός ο αλγόριθμος έχει δύο εισόδους, τις A και B, όπου μόνο η μία από τις δύο εισόδους κωδικοποιείται. Η δομή αντίθετα με τους επόμενους αλγορίθμους δεν χωρίζεται σε δύο ή περισσότερες ξεχωριστές μονάδες, αλλά χρησιμοποιεί μία μονάδα κωδικοποίησης, μία μονάδα παραγωγής μερικών γινομένων και τέλος μία μονάδα δενδρικού συμπιεστή wallace και αυτή η δομή φαίνεται στο Σχήμα 3.2.

Σχήμα 3.2 Δομή WHOLE

Το αποτέλεσμα αυτού του πολλαπλασιασμού θα είναι προφανώς ένας αριθμός carry save μορφής 64-bit.

3.2.2 Πρώτη υλοποίηση WHOLE

Η πρώτη υλοποίηση του αλγορίθμου WHOLE έγινε εξ' ολοκλήρου με στοιχεία από την βιβλιοθήκη της DesignWare, με βασικό εργαλείο να είναι το DW02_MULTP του οποίου οι είσοδοι και οι έξοδοι έχουν εξηγηθεί σε επόμενο κεφάλαιο.

Σε αυτή την περίπτωση δεν θα χρειαστεί να αλλάξουμε το netlist του κυκλώματος καθώς κωδικοποιούμε ολόκληρο το A και όχι κάποιο τμήμα του.

3.2.2.1 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το κύκλωμα του Whole έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Όταν το αποτέλεσμα έπρεπε να είναι για λόγους σύγκρισης σε δυαδική μορφή αντί να γίνει η πρόσθεση των δύο σειρών από bit που βγαίνουν σαν έξοδοι από το DW02_MULTP, χρησιμοποιείται ο πολλαπλασιαστής της DesignWare DW02_MULT όπου το αποτέλεσμα αυτού του πολλαπλασιαστή είναι ένας αριθμός 64-bit σε δυαδική μορφή με το MSB του να αναπαριστά το πρόσημο του.

3.2.3 Δεύτερη υλοποίηση WHOLE

Η δεύτερη υλοποίηση του αλγορίθμου WHOLE έγινε και με δικές μας μονάδες αλλά και με στοιχεία από την βιβλιοθήκη της Designware, όπως το DW02_TREE. Η δεύτερη υλοποίηση δεν διαφέρει από την πρώτη ως προς το σχήμα και επίσης και αυτή έχει ως έξοδο έναν αριθμό μορφής carry save που το άθροισμα του είναι το γινόμενο τον εισόδων του A και B όπου οι A και B είναι προσημασμένοι αριθμοί σε μορφή συμπληρώματος ως προς δύο.

3.2.3.1 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το κύκλωμα του Whole έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή όπως ακριβώς και η πρώτη υλοποίηση.

Encoder

Για να κωδικοποιηθούν τα ψηφία του A σε modified booth ψηφία χρησιμοποιείται μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τρία σήματα για την κάθε τριάδα. Τα τρία αυτά σήματα είναι τα: m, x, και x2 και στον Πίνακα 3.4 αναφέρονται εκτεταμένα οι τιμές των προαναφερθέντων σημάτων.

Οι πύλες του κωδικοποιητή φαίνονται στο Σχήμα 3.12.

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από τον κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων λαμβάνει ως είσοδο 3 σήματα των 16 bit αφού έχουν κωδικοποιηθεί 16 τριάδες του A (τα τρία αυτά σήματα είναι τα: m, x, και x2) και άλλο 1 σήμα 32-bit που έχει την τιμή του B.

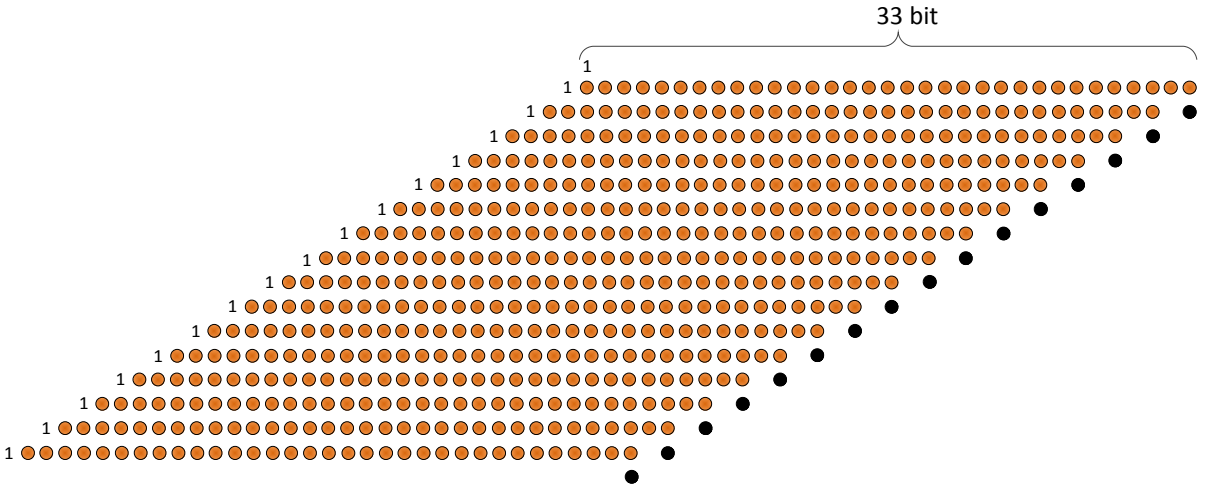
Η διεργασία παραγωγής μερικών γινομένων χωρίζεται σε 3 μέρη. Στο Σχήμα 3.13 φαίνονται οι πύλες που χρησιμοποιήθηκαν για κάθε ένα από αυτά τα 3 μέρη αφού αυτή η μονάδα είναι όμοια με την αντίστοιχη μονάδα που χρησιμοποιήθηκε από τον αλγόριθμο 2encA με την μόνη διαφορά ότι η κάθε μία από τις δύο μονάδες για τον 2encA λάμβανε ως είσοδο 3 σήματα των 8 bit αντί για 16.

Έτσι η μονάδα παραγωγής μερικών γινομένων βγάζει στην έξοδο της 16 αριθμούς από 33 bit ο καθένας.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Αυτή η μονάδα παίρνει ως είσοδο 16 33-bit αριθμούς και μαζί με τα κρατούμενα και τους διορθωτικούς όρους τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα, τα κρατούμενα και οι διορθωτικές μονάδες παρουσιάζονται στο Σχήμα 3.3 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 3.3 Μερικά γινόμενα, διορθωτικοί όροι και κρατούμενα

Με ίδια αιτιολόγηση με αυτή που γίνεται και στο κεφάλαιο του αλγορίθμου 2encA, έτσι και και στον αλγόριθμο WHOLE χρειάζονται διορθωτικές μονάδες αντί για επέκταση προσήμου και κρατούμενα για την περίπτωση που θέλουμε πρέπει να προστεθεί το συμπλήρωμα του μερικού γινομένου. Τα κρατούμενα στο σχήμα φαίνονται ως μαύρες τελείες. Η πρώτη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο, η δεύτερη οριζόντια σειρά από bit αποτελεί το δεύτερο μερικό γινόμενο, και ούτω καθεξής. Κάθε μερικό γινόμενο βρίσκεται 2 θέσεις αριστερότερα από το προηγούμενο μερικό γινόμενο λόγω της MB κωδικοποίησης.

Οι διορθωτικοί όροι έχουν εξηγηθεί και αναλυθεί αλγεβρικά σε επόμενο κεφάλαιο και με τις ίδιες σχέσεις αλλά για την περίπτωση του 32x32 πολλαπλασιασμού που αντιμετωπίζει ο παρών αλγόριθμος η σχέση των διορθωτικών όρων γίνεται:

$$ct_{32} = 2^{63} + 2^{61} + 2^{59} + 2^{57} + 2^{55} + 2^{53} + 2^{51} + 2^{49} + 2^{47} + 2^{45} + 2^{43} + 2^{41} + 2^{39} + 2^{37} + 2^{35} + 2^{33} + 2^{32}$$

Όλοι οι παραπάνω διορθωτικοί όροι προστίθενται και φαίνονται στο παραπάνω σχήμα μαζί με το $1 \cdot 2^{47}$ σε αντίθεση με επόμενους αλγορίθμους αφού δεν χρειάζεται να γίνει επέκταση προσήμου σε αυτόν τον αλγόριθμο.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η λειτουργία αυτής της μονάδας του πολλαπλασιασμού έχει εξηγηθεί σε επόμενα κεφάλαια και δεν χρίζει περαιτέρω ανάλυσης. Το αποτέλεσμα θα είναι ένας αριθμός σε μορφή αθροίσματος-κρατουμένου 64-bit.

Όταν το αποτέλεσμα έπρεπε να είναι για λόγους σύγκρισης σε δυαδική μορφή η πρόσθεση των δύο σειρών από bit που βγαίνουν σαν έξοδοι από τον δενδρικό συμπιεστή Wallace χρησιμοποιούνται ως είσοδοι στον αθροιστή της DesignWare DW01_ADD και το αποτέλεσμα θα είναι προφανώς ένας αριθμός 64-bit σε δυαδική μορφή με το MSB του να αναπαριστά το πρόσημο του.

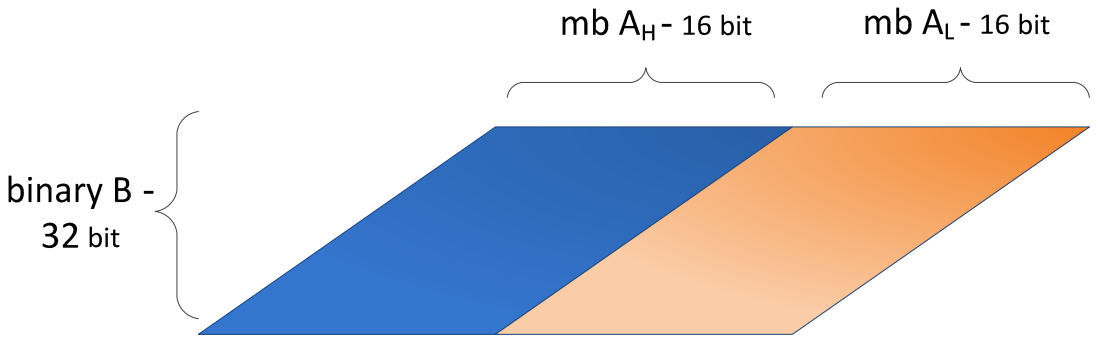
3.3 Αλγόριθμος 2encA

Ο αλγόριθμος 2encA είναι ο πρώτος αλγόριθμος που υλοποιήθηκε για να έχει καλύτερα αποτελέσματα δηλαδή χαμηλότερη κατανάλωση όταν οι είσοδοι είναι σήματα φωνής.

3.3.1 Γενική δομή αλγορίθμου 2encA

Αυτός ο αλγόριθμος έχει δύο εισόδους, τις A και B, όπου η είσοδος A προορίζεται να έχει σήματα φωνής. Η λειτουργία του αλγορίθμου προσπαθεί να ελαχιστοποιήσει την κατανάλωση με χρήση καταμερισμού υλοποιώντας δύο επιμέρους πολλαπλασιασμούς με εισόδους στην πρώτη μονάδα τα 16 MSB του A και το B και στην δεύτερη μονάδα τα 16 LSB του A και το B.

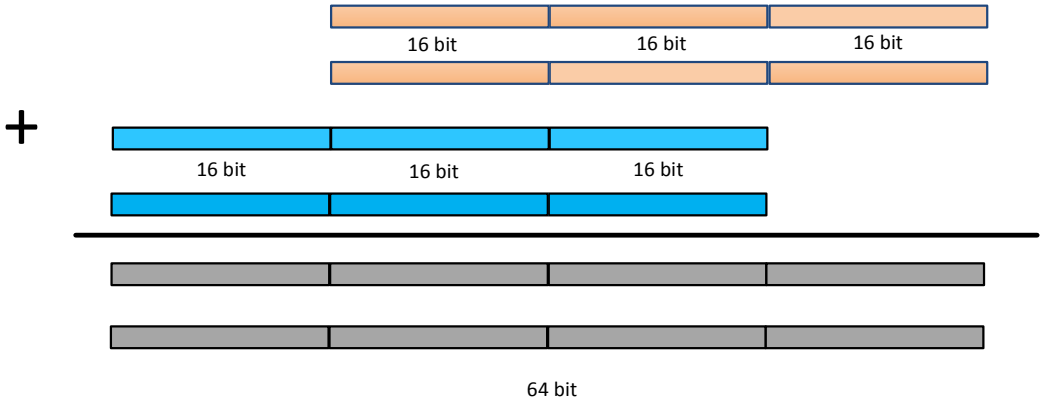
Αυτή η δομή φαίνεται και στο Σχήμα 3.4.



Σχήμα 3.4 Δομή 2encA

Το αποτέλεσμα αυτών των δύο επιμέρους πολλαπλασιασμών θα είναι δύο αριθμοί carry save μορφής 48-bit από κάθε μονάδα, με τα 48 bit που εξάγονται σαν αποτέλεσμα από τον πολλαπλασιασμό του A_H με το B να βρίσκονται κατά 16 bit αριστερότερα από τα 48 bit της άλλης μονάδας. Αυτό φαίνεται και στο Σχήμα 3.5.

Οι δύο carry save αριθμοί θα πρέπει να προστεθούν ώστε να βγει στην έξοδο του 2encA μόνο ένας carry save αριθμός μήκους 64 bit.



Σχήμα 3.5 Έξοδοι μονάδων 2encA

3.3.2 Πρώτη υλοποίηση 2encA

Η πρώτη υλοποίηση του αλγορίθμου 2encA έγινε εξ' ολοκλήρου με στοιχεία από την βιβλιοθήκη της Designware, με βασικό εργαλείο να είναι το DW02_MULTP. Το DW02_MULTP έχει ως έξοδο έναν αριθμό μορφής carry save που το άθροισμα του είναι το γινόμενο των εισόδων του A και B όπου οι A και B είναι προσημασμένοι αριθμοί σε μορφή συμπληρώματος ως προς δύο.

Χαρακτηριστικά του DW02_MULTP είναι ότι πάντα η πρώτη είσοδος (A) κωδικοποιείται με χρήση της κωδικοποίησης modified Booth, ενώ από τις εξόδους της out0 & out1 οι οποίες είναι 50 bit η κάθε μία (και αυτό γιατί το DW02_MULTP δεν δέχεται η έξοδος του να είναι 48 bit τα οποία είναι αποδεδειγμένα αρκετά για να παραστήσουν το αποτέλεσμα της πράξης, αλλά τουλάχιστον 50 bit), η έξοδος out0 είναι πάντα θετική, δηλαδή το 50ο bit του out0 είναι πάντα μηδέν. Τα επιπλέον δύο bit μπορούν προφανώς να αγνοηθούν.

3.3.2.1 Χρήση και αλλαγή του DW02_MULTP

Όπως είδαμε και από το Σχήμα 3.4 ο αλγόριθμος βασίζεται στην κωδικοποίηση κατά modified Booth του A_H και A_L. Βάζοντας σαν είσοδο του DW02_MULTP το A_L και το B μας δίνει το σωστό αποτέλεσμα, αυτό δεν συμβαίνει όμως πάντα όταν βάζουμε σαν είσοδο του DW02_MULTP το A_H και το B. Αυτό είναι λογικό αν κανείς σκεφτεί την κωδικοποίηση με Modified Booth.

Στην κωδικοποίηση με χρήση του αλγορίθμου Modified Booth γνωρίζουμε πως το δεξιότερο Modified Booth ψηφίο που κωδικοποιείται, κωδικοποιεί την δεξιότερη τριάδα αριθμών με το LSB από αυτά τα τρία να είναι πάντα το μηδέν. Στην περίπτωση όμως που το MSB του A_L είναι 1 τότε το αποτέλεσμα δεν θα βγει σωστό.

Το παρακάτω παράδειγμα επεξηγεί καλύτερα το παραπάνω πρόβλημα.

Έστω ο 32-bit αριθμός A= 0110 1001 1010 0101 1000 1111 1010 1010

Έτσι θα έχουμε:

A_H = 0110 1001 1010 0101 και

A_L = 1000 1111 1010 1010

Αν βάλουμε σαν είσοδο στο DW02_MULTP το A_H και το οποιοδήποτε B, τότε η δεξιότερη τριάδα αριθμών που θα κωδικοποιήσει το εργαλείο θα είναι το 01 (ως LSB του A_H) και το 0 ως στοιχείο του αλγορίθμου Modified Booth, δηλαδή θα είναι η τριάδα 010 ενώ θα έπρεπε να είναι η τριάδα 011 αφού κοιτώντας τον A, το προηγούμενο ψηφίο του 18ου και 17ου ψηφίου είναι το 1. Αυτό βέβαια δεν συμβαίνει όταν το 16ο bit του αριθμού A είναι 0.

Μη έχοντας πρόσβαση στον κώδικα του DW02_MULTP μόνη λύση ήταν να εξεταστεί προσεκτικά το netlist. Στο netlist παρατηρήθηκε ότι το a[16] (δηλαδή το LSB του A_H), χρησιμοποιείται σε 2 πύλες. Η πρώτη πύλη ήταν μία XOR η οποία είχε σαν εισόδους το a[16] και το a[17], και η δεύτερη ήταν μία πύλη αντιστροφής που είχε σαν είσοδο μόνο το a[16]. Το a[18] το οποίο είναι και αυτό το μεσαίο ψηφίο της επόμενης τριάδας προς κωδικοποίηση, χρησιμοποιείται σε 2 είδη πυλών. Το πρώτο είδος είναι XOR πύλες οι οποίες έχουν σαν είσοδο το a[18] και το a[19] και το δεύτερο είδος είναι XNOR πύλες οι οποίες έχουν σαν είσοδο το a[18] και το a[17].

Στο Σχήμα 3.6 φαίνεται μία πύλη XNOR και στον Πίνακα 3.1 παρουσιάζεται ο πίνακας αληθείας αυτής.

Σχήμα 3.6 Πύλη XNOR

Πίνακας 3.1 Πίνακας αληθείας πύλης XNOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

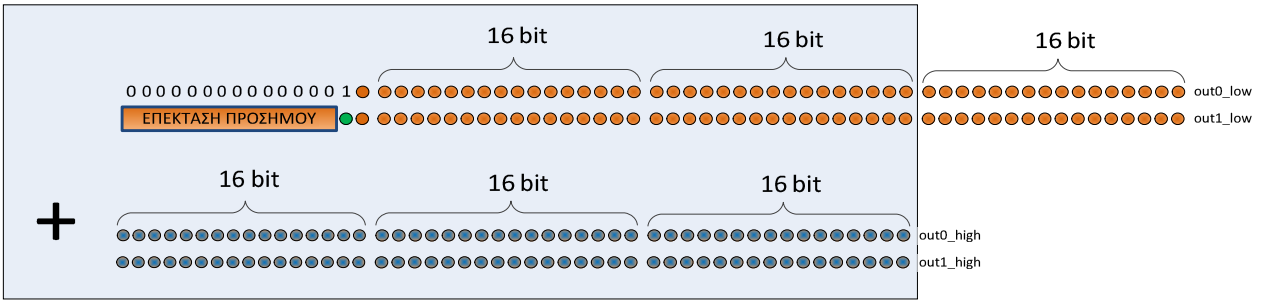
Από τον πίνακα αληθείας της XNOR φαίνεται πως αν μία από τις δύο εισόδους είναι μηδενική τότε η έξοδος είναι η αντίστροφη της άλλης. Το γεγονός αυτό εξηγεί την πύλη αντιστροφής που είχε σαν είσοδο μόνο το a[16].

Αλλάζοντας λοιπόν στο netlist την παραπάνω πύλη αντιστροφής σε μια πύλη XNOR με εισόδους τα ψηφία a[16] και a[15] το αποτέλεσμα είναι το επιθυμητό.

3.3.2.2 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το όλο κύκλωμα του 2encA έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Κάθε ένα από τα δύο DW02_MULTP που χρησιμοποιούνται έχει ως έξοδο δύο καταχωρητές 48-bit ο καθένας όπως φαίνεται και στο Σχήμα 3.5. Οι τέσσερις αυτοί καταχωρητές τροφοδοτούν δύο διαδοχικούς Carry Save Adders όπως φαίνεται στο Σχήμα 3.7.



Σχήμα 3.7 Τροφοδότηση των δύο CSA

Το μεγάλο τετράγωνο υποδεικνύει τους καταχωρητές που θα μπουν ως είσοδοι στο DW01_csa που είναι το εργαλείο της DesignWare που υλοποιεί τον carry save adder. Στο Σχήμα 3.7 είναι εμφανείς οι απλοποιήσεις που γίνονται θέτοντας τα 15 MSB του out0_low ως μηδενικά και κάνοντας την αναγκαστική επέκταση προσήμου στο out1_low για να βγει σωστό το αποτέλεσμα.

Για την βελτίωση αυτού του κυκλώματος βάλαμε στην θέση του 50ου bit του out0_low μία μονάδα αντί για μηδέν και αντιστρέφουμε το αντίστοιχο bit του out1_low (πράσινο bit στο Σχήμα 3.7) κάνοντας επέκταση προσήμου με μονάδες στο out1_low.

Οι τρεις πρώτοι καταχωρητές μπαίνουν ως είσοδοι στον πρώτο Carry Save Adder και ο αριθμός 48-bit σε carry save μορφή που εξέρχεται ως έξοδος από αυτή την μονάδα εισέρχεται σαν είσοδος μαζί με τον τέταρτο καταχωρητή (out1_high) στον δεύτερο Carry Save Adder. Το αποτέλεσμα αυτού είναι ένας 48-bit carry save αριθμός ο οποίος μαζί με τα 16 LSB των out0_low και out1_low, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του 2encA, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

3.3.3 Σύνθεση και οργάνωση πειραμάτων

Η αξιολόγηση των κυκλωμάτων έγινε με βάση την επιφάνεια τους σε μm^2 και την κατανάλωση ισχύος σε W.

Οι πύλες που συνθέτουν τα παραπάνω modules (κωδικοποιητής, partial product generator, wallace tree) περιγράφτηκαν σε VHDL σε structural programming σε διαφορετικά αρχεία. Η βιβλιοθήκη (Digital Standard Cell Library) που χρησιμοποιήθηκε ήταν η SAED90nm και οι τελικές μετρήσεις έγιναν για το πλησιέστερο πολλαπλάσιο του 0.1ns ώστε όμως να προλαβαίνουν τα αποτελέσματα να γίνονται διαθέσιμα στις εξόδους και άρα να μην βγαίνει SLACK VIOLATED στην ανάλυση χρονισμού. Η σύνθεση έγινε με compile_ultra και όχι για απλό compile για βέλτιστα αποτελέσματα. Στην σύνθεση χρησιμοποιήθηκαν οι εντολές:

set_max_area 0

set_max_dynamic_power 0.0

set_max_leakage_power 0.0

ώστε να υπάρχει βελτιστοποίηση ως προς την επιφάνεια και την ισχύ που καταναλώνεται. Από την σύνθεση που έγινε με τον Design Compiler εξήχθη η επιφάνεια για το κάθε κύκλωμα. Για κάθε τέτοιο κύκλωμα με την εντολή:

write -h -f verilog -output NAME.sv

γινόταν διαθέσιμο το netlist σε ένα αρχείο sv το οποίο μετά το χρησιμοποιούσαμε μέσω του modelsim για να κάνουμε post-synthesis simulation. Η προσομοίωση από την μία γινόταν για την αριθμητική επαλήθευση των κυκλωμάτων που υλοποιήθηκαν και από την άλλη για την εξαγωγή των αποτελεσμάτων πραγματικής κατανάλωσης ισχύος. Πιο συγκεκριμένα μέσω του modelsim διαβάζαμε το netlist από ένα αρχείο sv, το αρχείο της βιβλιοθήκης saed90nm.v και

ένα testbench σε verilog το οποίο κάθε 5ns έδινε στις εισόδους του κυκλώματος κάποιους συγκεκριμένους δυαδικούς αριθμούς, των οποίων οι τιμές εξαρτώνται από την περίπτωση που θέλουμε να εξετάσουμε. Οι τιμές αυτών των αριθμών προέρχονταν άλλες φορές από συναρτήσεις της verilog που παράγουν τυχαίους αριθμούς και άλλες φορές διαβαζόντουσαν από αρχεία που περιείχαν αριθμούς σε ακέραια δεκαδική μορφή και εσωτερικά η verilog τους μετέτρεπε σε προσημασμένους δυαδικούς. Το modelsim εν συνεχεία παρήγαγε ένα vcd αρχείο το οποίο διαβάζαμε από το PrimeTime PX (όπως και το αρχείο sv) και το οποίο πρόγραμμα μέσω ενός script και χρησιμοποιώντας την εντολή:

set power_analysis_mode averaged

έδινε την κατανάλωση ισχύος για κάθε διαφορετική περίπτωση.

Για κάθε διαφορετικό αλγόριθμο ακολουθήθηκαν οι ίδιες ακριβώς διαδικασίες για να είναι σωστές οι συγκρίσεις μεταξύ τους.

3.3.4 Αποτελέσματα πρώτης υλοποίησης 2encA

Οι συγκρινόμενοι αλγόριθμοι είναι η πρώτη υλοποίηση του αλγορίθμου 2encA με την πρώτη υλοποίηση του αλγορίθμου Whole όπου και οι δύο είναι άμεσα συγκρινόμενοι αφού και οι δύο χρησιμοποιούν modules της DesignWare. Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους πίνακες 3.2 και 3.3:

Πίνακας 3.2 Πίνακας επιφάνειας 2encA σε Carry Save μορφή στα 1.6ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	33066
2encA	39352

Πίνακας 3.3 Πίνακας επιφάνειας 2encA σε δυαδική μορφή στα 2.1ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	44213
2encA	51869

Έχει μετρηθεί η κατανάλωση για τις εξής περιπτώσεις:

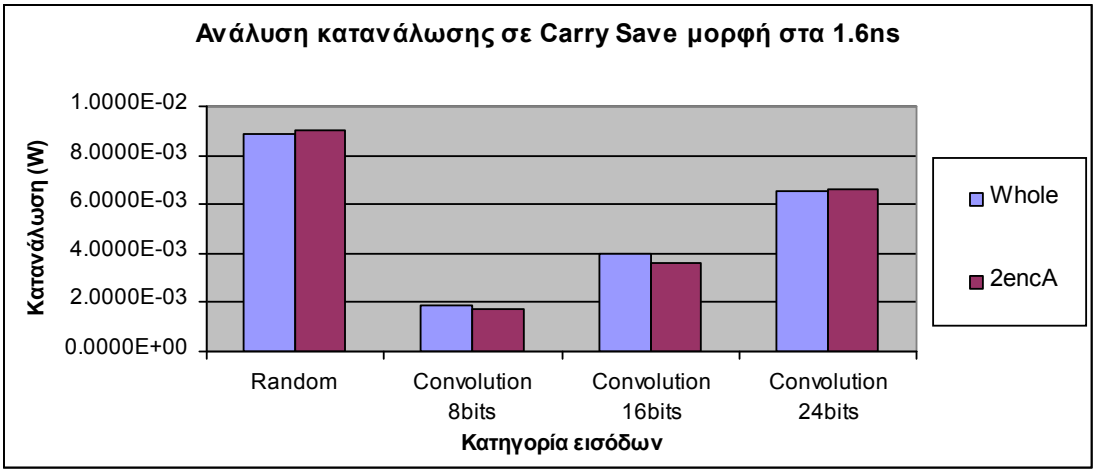
α) οι δύο είσοδοι είναι τυχαίοι δυαδικοί αριθμοί (random)

β) οι δύο είσοδοι είναι σήματα φωνής 8, 16 και 24 bits αντίστοιχα (convolution)

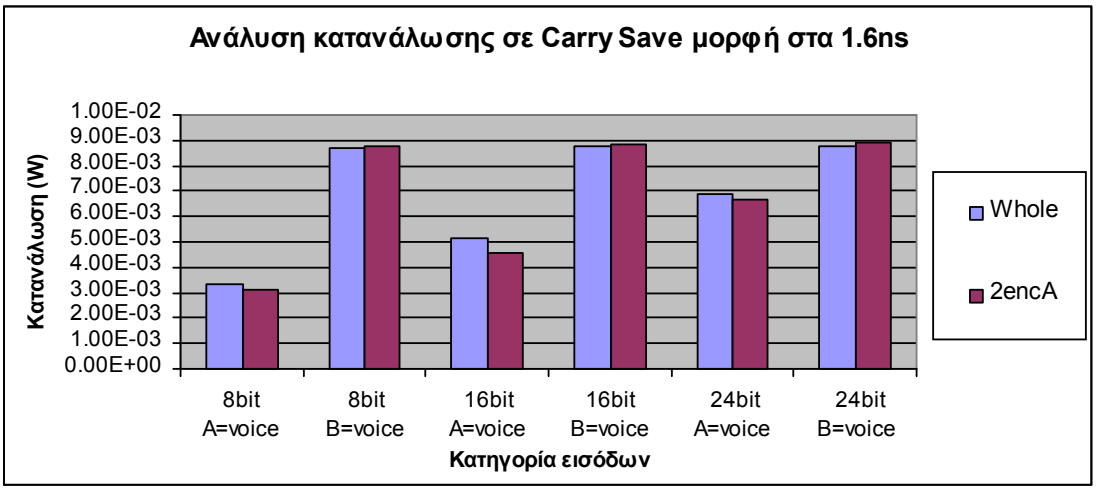
γ) η μία είσοδος (A) είναι σήμα φωνής 8, 16 και 24 bits αντίστοιχα (A=voice)

δ) η μία είσοδος (B) είναι σήμα φωνής 8, 16 και 24 bits αντίστοιχα (B=voice)

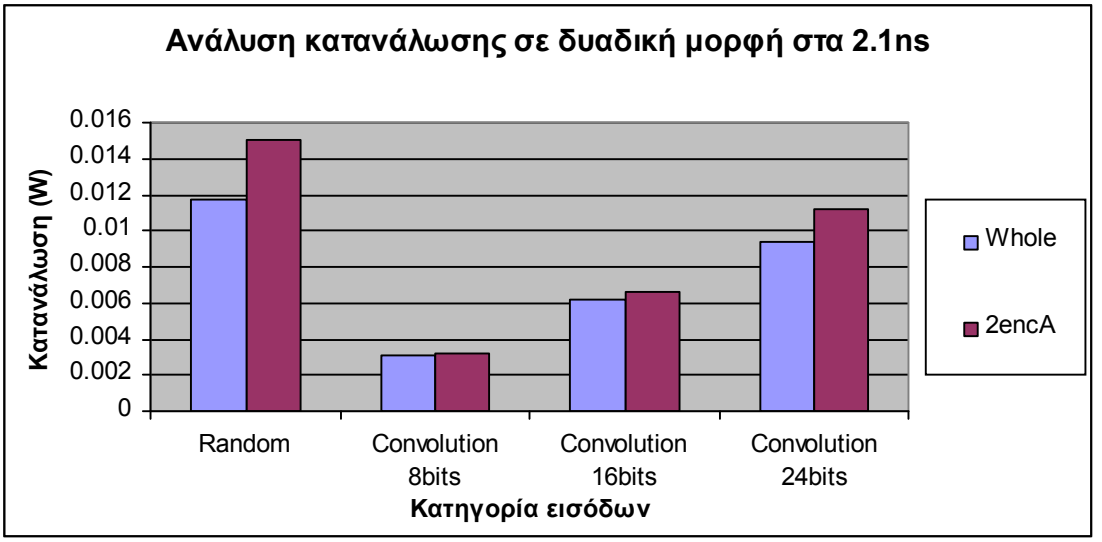
Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα.



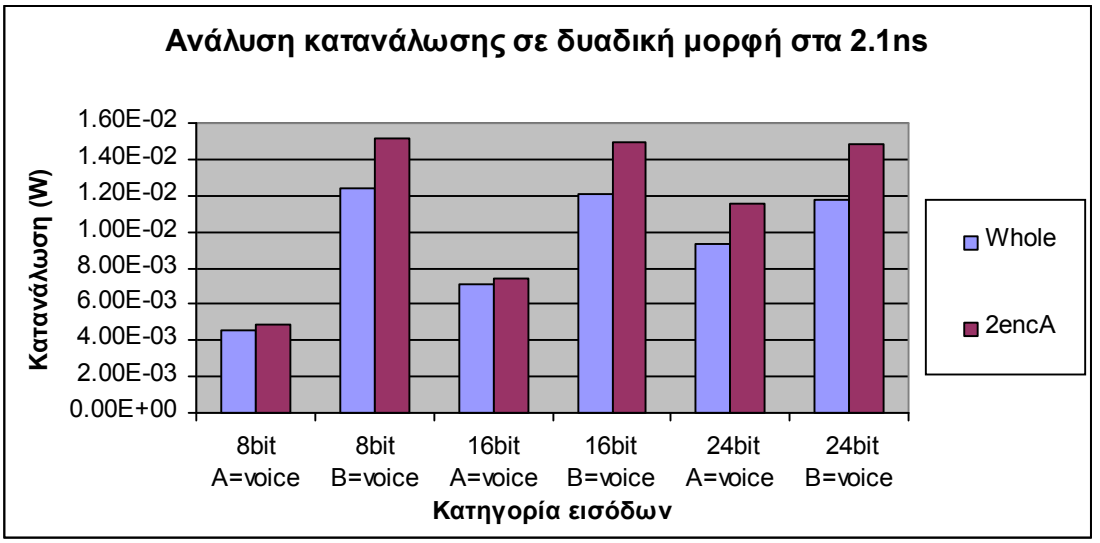
Σχήμα 3.8 Αποτελέσματα 1ης υλοποίησης 2encA σε CS μορφή στα 1.6ns



Σχήμα 3.9 Αποτελέσματα 1ης υλοποίησης 2encA σε CS μορφή στα 1.6ns



Σχήμα 3.10 Αποτελέσματα 1ης υλοποίησης 2encA σε Binary μορφή στα 2.1ns



Σχήμα 3.11 Αποτελέσματα 1ης υλοποίησης 2encA σε Binary μορφή στα 2.1ns

Βλέπουμε από το Σχήμα 3.8 ότι για τυχαίους αριθμούς καθώς και για όταν και οι δύο είσοδοι είναι σήματα φωνής 24 bit ο αλγόριθμος 2encA παρουσιάζει μεγαλύτερη κατανάλωση από τον κλασσική υλοποίηση. Αυτό συμβαίνει γιατί χρησιμοποιούνται και τα δύο μέρη του 2encA και η κατανάλωση επιβαρύνεται και από τους 2 carry save adders που είναι παραπάνω κύκλωμα σε σχέση με την κλασσική υλοποίηση του πολλαπλασιαστή το οποίο φαίνεται και από τον Πίνακα 3.2. Αυτό όμως δεν συμβαίνει για convolution σημάτων φωνής 8 και 16 bits, όπως και για τις περιπτώσεις που μόνο το ένα σήμα είναι 8 και 16 bits αφού χρησιμοποιείται ουσιαστικά μόνο το ένα από τα δύο μέρη του 2encA άρα η κατανάλωση είναι λιγότερη. Αυτό συμβαίνει, γιατί τα 8 υψηλότερης αξίας μερικά γινόμενα που προκύπτουν από τον πολλαπλασιασμό του B με το AH είναι τις περισσότερες φορές ίσα με 0, δεδομένου ότι τα σήματα φωνής των 8 και των 16 bits (που εδώ είναι η είσοδος A) έχουν τα υπόλοιπα ψηφία μηδενικά. Επομένως, το switching activity των αριστερότερων 8 και 16 bits του αποτελέσματος, που είναι σε Carry-Save μορφή και απεικονίζονται στο Σχήμα 3.7, παραμένει το ίδιο γι'αυτή την κατηγορία δεδομένων. Όταν το σήμα φωνής είναι μόνο το B είναι λογικό ότι ο 2encA αλγόριθμος θα υστερεί σε σχέση με την πρώτη υλοποίηση του κλασσικού πολλαπλασιασμού αφού κωδικοποιούμε με modified booth το A πάντα και επιπλέον έχουμε και παραπάνω κύκλωμα από τους 2 carry save adders.

Η διαφορά σε σχέση με τα αποτελέσματα σε binary μορφή είναι ότι κατά τη μετατροπή του αποτελέσματος από Carry-Save, οι τιμές των αριστερότερων bits του τελικού πια αποτελέσματος αλλοιώνονται, λόγω της διάδοσης κρατουμένου στον 64-bit Parallel-Prefix adder, με αποτέλεσμα να αυξάνεται το switching activity των ψηφίων αυτών αλλά και γιατί ο Parallel-Prefix adder που χρησιμοποιείται έχει καλύτερα αποτελέσματα ως προς την κατανάλωση όταν δεν υπάρχουν πριν από αυτόν οι carry save adders που επιβαρύνουν κατά πολύ σε αυτή την περίπτωση την κατανάλωση.

3.3.5 Δεύτερη υλοποίηση 2encA

Η δεύτερη υλοποίηση του αλγορίθμου 2encA έγινε με δικές μας μονάδες αλλά και με στοιχεία από την βιβλιοθήκη της Designware, όπως το DW02_TREE. Η δεύτερη υλοποίηση δεν διαφέρει από την πρώτη ως προς το σχήμα και επίσης και αυτή έχει ως έξοδο έναν αριθμό μορφής carry save που το άθροισμα του είναι το γινόμενο τον εισόδων του A και B όπου οι A και B είναι προσημασμένοι αριθμοί σε μορφή συμπληρώματος ως προς δύο που είναι οι είσοδοι του 2encA.

3.3.5.1 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το κύκλωμα του 2encA έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Encoder

Για να κωδικοποιηθούν τα ψηφία του A σε modified booth ψηφία χρησιμοποιείται μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τρία σήματα για την κάθε τριάδα. Το κύκλωμα είναι κοινό και για το HIGH κομμάτι του 2encA αλλά και για το LOW κομμάτι του, αφού κωδικοποιούνται ιδίου αριθμού τριάδες (συγκεκριμένα κωδικοποιούνται σε κάθε ένα από τα δύο κομμάτια του 2encA 8 τριάδες σε 3·8=24 σήματα.

Τα τρία αυτά σήματα είναι τα: m, x, και x2.

Το x2 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +2 ή −2, το οποίο σημαίνει ότι το μερικό γινόμενο που πρέπει να προστεθεί είναι ή το B ολισθημένο κατά μία θέση ή το συμπλήρωμα αυτού. Σε κάθε άλλη περίπτωση το x2 είναι 0.

Το x1 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +1 ή −1, το οποίο σημαίνει ότι το μερικό γινόμενο που πρέπει να προστεθεί είναι ή το B ή το συμπλήρωμα αυτού. Σε κάθε άλλη περίπτωση το x1 είναι 0.

Το m σε άλλες κωδικοποιήσεις συνήθως είναι 1 όταν και το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1, που αυτό συνεπάγεται ότι πρέπει να προστεθεί το συμπλήρωμα του B είτε ολισθημένο είτε όχι, εκτός από την περίπτωση που η τριάδα είναι το 111, οπότε δεν προστίθεται τίποτα. Επειδή όμως ο αλγόριθμος είναι κατασκευασμένος να πολλαπλασιάζει πολλούς μικρούς αριθμούς, άρα και μικρούς αρνητικούς αριθμούς, θα επιβαρυνόταν εξαιρετικά η κατανάλωση αν αντιμετωπίζαμε το 111 σαν αρνητικό μηδέν, οπότε το m είναι 1 όταν το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1 εκτός από την περίπτωση της τριάδας 111.

Στον Πίνακα 3.4 αναφέρονται εκτεταμένα οι τιμές των προαναφερθέντων σημάτων.

Πίνακας 3.4 Πίνακας αληθείας σημάτων m, x, x2

A_{2i+1}	A_{2i}	A_{2i-1}	m_i	x_i	$x2_i$	**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	0	0	0	0

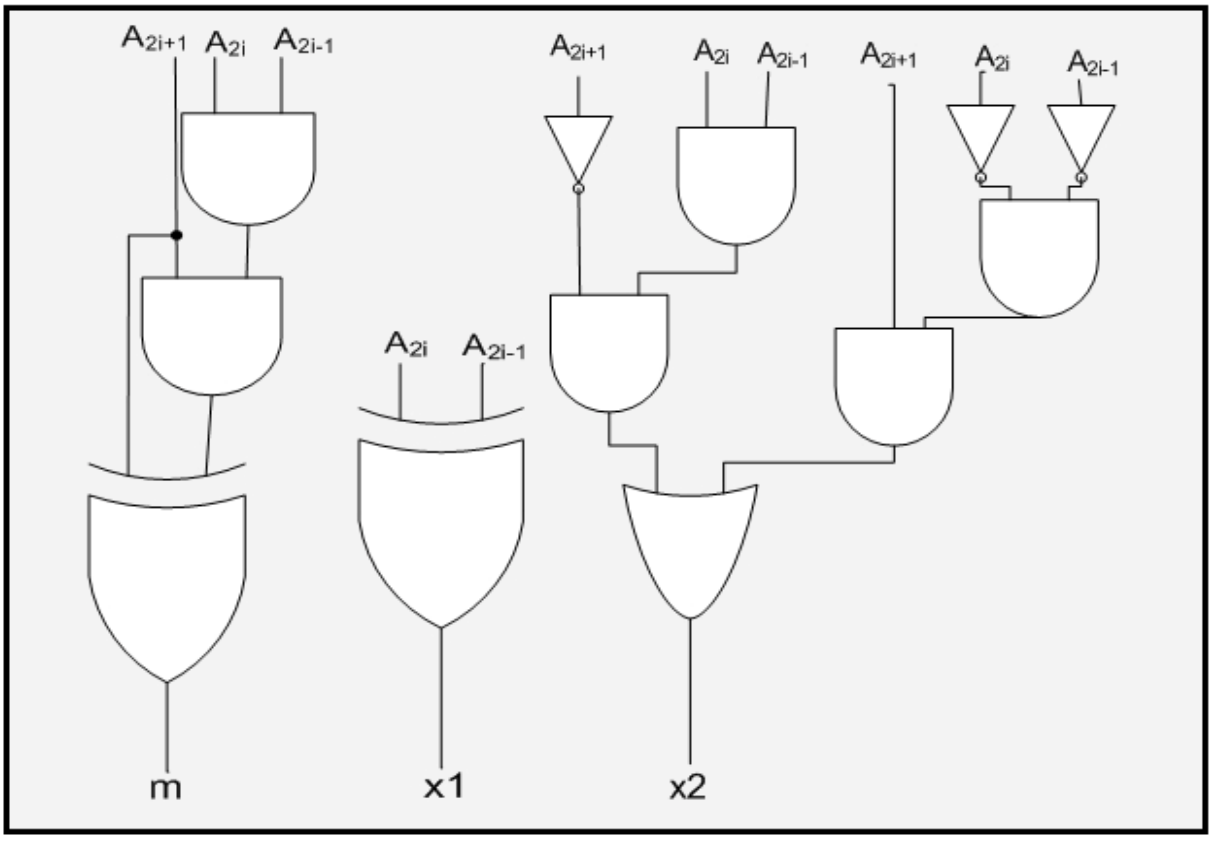
Έτσι οι πύλες που χρησιμοποιήθηκαν για να υλοποιηθούν αυτά τα σήματα είναι οι παρακάτω:

$$m = A_{2i+1} \text{ xor } (A_{2i+1} \text{ and } A_{2i} \text{ and } A_{2i-1})$$

$$x2 = (\text{not}(A_{2i+1}) \text{ and } A_{2i} \text{ and } A_{2i-1}) \text{ or } (A_{2i+1} \text{ and not}(A_{2i}) \text{ and not}(A_{2i-1}))$$

$$x = (A_{2i} \text{ xor } A_{2i-1})$$

Και φαίνονται και στο Σχήμα 3.12.



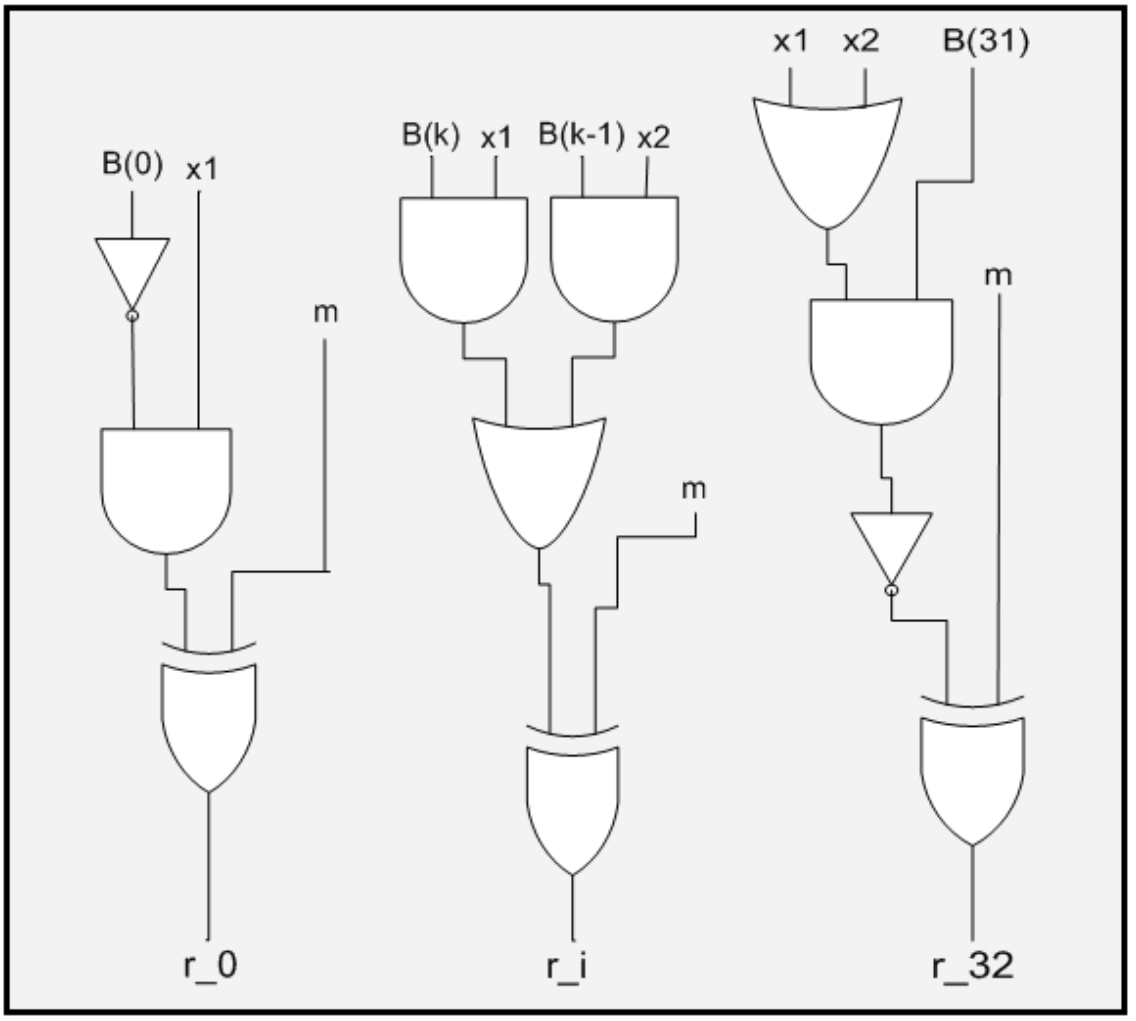
Σχήμα 3.12 Πύλες του κωδικοποιητή

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από την κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων λαμβάνει ως είσοδο 3 σήματα των 8 bit αφού έχουν κωδικοποιηθεί 8 τριάδες του A_H ή του A_L (τα τρία αυτά σήματα είναι τα: m, x, και x2) και άλλο 1 σήμα 32-bit που έχει την τιμή του B.

Η μονάδα αυτή χρησιμοποιεί τα παραπάνω σήματα ώστε στις εξόδους της να γίνουν διαθέσιμα τα μερικά γινόμενα. Λόγω της πιθανής ολίσθησης στις περιπτώσεις όπου το x2 είναι ίσο με 1, τα μερικά γινόμενα που θα παραχθούν θα είναι 33 bit. Η διεργασία αυτή χωρίζεται σε 3 μέρη. Το πρώτο σκέλος είναι οι πύλες για το πρώτο bit του μερικού γινομένου, το δεύτερο σκέλος είναι οι πύλες για όλα τα υπόλοιπα bit εκτός από το 33ο, δηλαδή ξεκινώντας από το δεύτερο μέχρι και το 32ο, και το τρίτο σκέλος είναι οι πύλες που θα χρησιμοποιηθούν για το 33ο bit.

Στο Σχήμα 3.13 φαίνονται οι πύλες που χρησιμοποιήθηκαν.



Σχήμα 3.13 Πύλες της μονάδας παραγωγής μερικών γινομένων

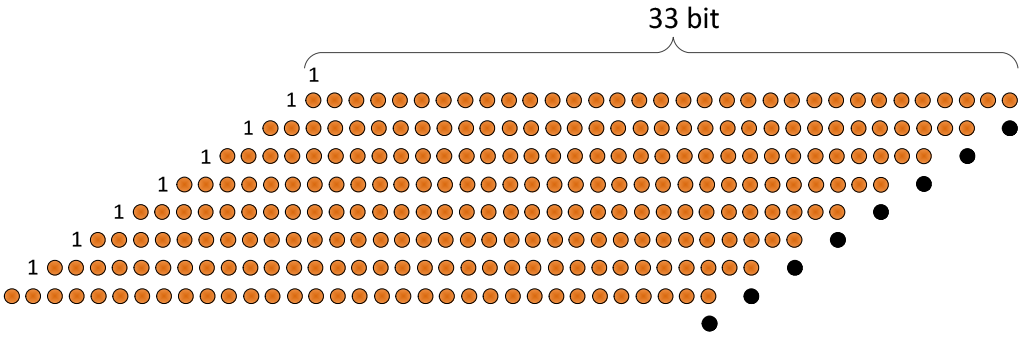
Ξεκινώντας την αρίθμηση από το μηδέν, το 33ο bit απεικονίζεται στο παραπάνω σχήμα ως r_32 ενώ τα bit ανάμεσα στο 1ο και στο 33ο απεικονίζονται ως r_i. Το 33ο bit κάθε μερικού γινομένου έχει πάντα αρνητική αξία. Προφανώς οι παραπάνω πύλες εφαρμόζονται κοινά σε κάθε κωδικοποιημένη τριάδα του A (πλέον ως m, x1, x2).

Έτσι με αυτές τις πύλες η μονάδα παραγωγής μερικών γινομένων βγάζει στην έξοδο της 8 αριθμούς από 33 bit ο καθένας.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Αυτή η μονάδα παίρνει ως είσοδο 8 33-bit αριθμούς και μαζί με τα κρατούμενα και τους διορθωτικούς όρους τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα, τα κρατούμενα και οι διορθωτικές μονάδες παρουσιάζονται στο Σχήμα 3.14 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 3.14 Μερικά γινόμενα, διορθωτικοί όροι και κρατούμενα

Στην περίπτωση που το m είναι ίσο με 1, άρα εκτελείται είτε το -2·B ή το –B, το κύκλωμα της μονάδας παραγωγής μερικών γινομένων χρησιμοποιώντας την πύλη XOR και έχοντας ως μία από τις δύο εισόδους της το m, βρίσκει το συμπλήρωμα ως προς 1 του αριθμού και το ολισθαίνει όταν αυτό χρειάζεται. Όμως εμείς πρέπει να προσθέσουμε τα συμπληρώματα ως προς δύο αυτών των αριθμών. Αυτό επιτυγχάνεται προσθέτοντας μία μονάδα στο LSB κάθε μερικού γινομένου όταν το σήμα m αυτής της σειράς είναι ίσο με 1. Στο Σχήμα 3.14 αυτό απεικονίζεται με μαύρες τελείες. Η πρώτη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο, η δεύτερη οριζόντια σειρά από bit αποτελεί το δεύτερο μερικό γινόμενο, και ούτω καθεξής. Κάθε μερικό γινόμενο βρίσκεται 2 θέσεις αριστερότερα από το προηγούμενο μερικό γινόμενο λόγω της MB κωδικοποίησης.

Οι διορθωτικοί όροι εξηγούνται ως εξής. Ας θεωρήσουμε πως έχουμε ένα παρόμοιο σχήμα με το παραπάνω μόνο που κάθε μερικό γινόμενο είναι 9 bit και υπάρχουν 4 μερικά γινόμενα, τα οποία έχουν προέλθει από παρόμοια επεξεργασία ενός πολλαπλασιασμού 8x8. Για να βγει σωστό το αποτέλεσμα θα πρέπει να κάνουμε 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:

$$ct = \sum_{k=0}^{3} \mathrm{s}_k \sum_{i=8+2k}^{15} 2^i = \sum_{k=0}^{3} \mathrm{s}_k \{2^{16} - 2^{8+2k}\} = \sum_{k=0}^{3} -\mathrm{s}_k 2^{8+2k}$$

Αυτή η εξίσωση μπορεί να αναλυθεί περαιτέρω χρησιμοποιώντας τις επόμενες δύο ισότητες:

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

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

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

Η τελική σχέση των διορθωτικών όρων βρίσκεται από την παρακάτω σχέση

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

Με τις ίδιες σχέσεις αλλά για την περίπτωση του 32x32 πολλαπλασιασμού που αντιμετωπίζει ο παρόν αλγόριθμος οι σχέση των διορθωτικών όρων γίνεται:

$$ct_{32} = 2^{47} + 2^{45} + 2^{43} + 2^{41} + 2^{39} + 2^{37} + 2^{35} + 2^{33} + 2^{32}$$

Όλοι οι παραπάνω διορθωτικοί όροι προστίθενται και φαίνονται στο παραπάνω σχήμα εκτός από το 2^{47} το οποίο θα προστεθεί αργότερα.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως το αποτέλεσμα ενός πολλαπλασιασμού 16x32 bits θα είναι 48 bits, έτσι ο αριθμός X θα είναι 48 bit. Τα μερικά γινόμενα που είναι δεν είναι 48 bits εισάγονται με μηδενικά στα αριστερά και

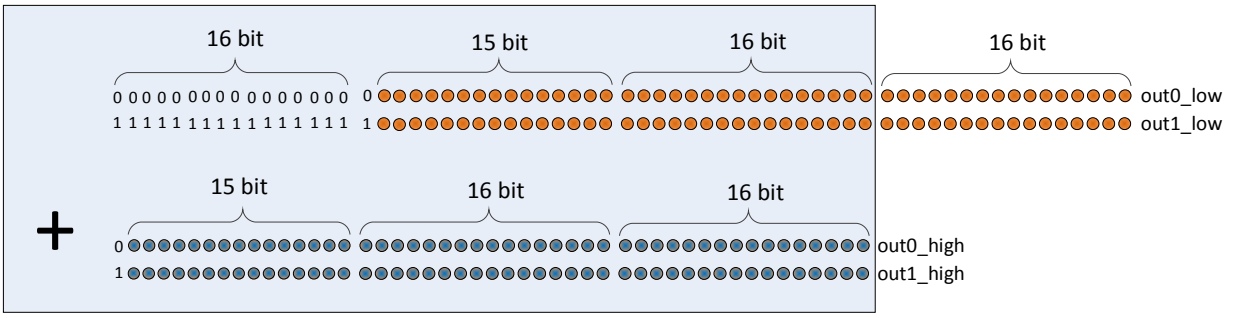
στα δεξιά τους για να γίνεται σωστά η άθροιση. Για παράδειγμα το δεύτερο μερικό γινόμενο εισάγεται στην είσοδο του DW02_tree ως: "0000000000000"&r1&"00", όπου r1 είναι τα παραπάνω 33 bits που φαίνονται στο σχήμα 3.8 στην δεύτερη σειρά μερικών γινομένων. Τα μηδενικά απλοποιούνται από τον design compiler και είναι σαν να μην μπαίνουν ψηφία σε αυτά τα σημεία δηλαδή δεν θα υπάρξει κανένας full adder ή half adder που να πάρει ως είσοδο το 0.

Πριν από αυτή την υλοποίηση προηγήθηκε η υλοποίηση με καθαρά δικό μας κώδικα αλλά το εργαλείο της DesignWare παρέχει μία πολύ καλύτερη υλοποίηση και αυτό θα φαίνεται στο netlist που η αλληλουχία πυλών υποδεικνύει ότι «σπάει» τους ημιαθροιστές και τους πλήρεις αθροιστές όπου χρειάζεται για να εξοικονομήσει ταχύτητα σε απλές πύλες XOR και AND για το save και το carry κάθε βαθμίδας με δύο ή παραπάνω bits σε αυτή τη βαθμίδα.

Τελική άθροιση

Εφόσον δεν έχουμε προσθέσει τον διορθωτικό όρο με βάρος 2^{47} γνωρίζουμε πως το αποτέλεσμα κάθε μονάδας wallace θα έχει το μηδέν ως MSB και στο carry αλλά και στο save.

Έτσι δεν έχουμε παρά να προσθέσουμε την μονάδα με βάρος 2^{47} και να κάνουμε επέκταση προσήμου στο αποτέλεσμα της LOW μονάδας wallace στο τώρα όμως γνωστό πρόσημο. Αυτό φαίνεται στο Σχήμα 3.15 όπως και ποια ψηφία πρέπει να προστεθούν για να βγει το τελικό αποτέλεσμα σε carry save.



Σχήμα 3.15 Τελικός αθροιστής και επέκταση προσήμου

Η πρόσθεση που εσωκλείεται στο παραλληλεπίπεδο του προηγούμενου σχήματος υλοποιήθηκε με 2 carry save adders. Το παραπάνω τετράγωνο υποδεικνύει τους καταχωρητές που θα μπουν ως είσοδοι στο DW01_csa που είναι το εργαλείο της DesignWare που υλοποιεί τον carry save adder. Στο Σχήμα 3.15 είναι εμφανείς οι απλοποιήσεις που γίνονται θέτοντας τα 17 MSB του out0_low ως 0 και τα 17 MSB του out1_low ως 1. Οι τρεις πρώτοι καταχωρητές μπαίνουν ως είσοδοι στον πρώτο Carry Save Adder και ο αριθμός 48-bit σε carry save μορφή που εξέρχεται ως έξοδος από αυτή την μονάδα εισέρχεται σαν είσοδος μαζί με τον τέταρτο καταχωρητή (out1_high) στον δεύτερο Carry Save Adder. Το αποτέλεσμα

αυτού είναι ένας 48-bit carry save αριθμός ο οποίος μαζί με τα 16 LSB των out0_low και out1_low, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του TWO, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

3.3.6 Αποτελέσματα δεύτερης υλοποίησης 2encA

Οι συγκρινόμενοι αλγόριθμοι είναι η δεύτερη υλοποίηση του αλγορίθμου 2encA με την δεύτερη υλοποίηση του αλγορίθμου Whole όπου και οι δύο είναι άμεσα συγκρινόμενοι αφού και οι δύο δεν χρησιμοποιούν modules της DesignWare παρά τον Wallace tree compressor.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους παρακάτω πίνακες 3.5 και 3.6:

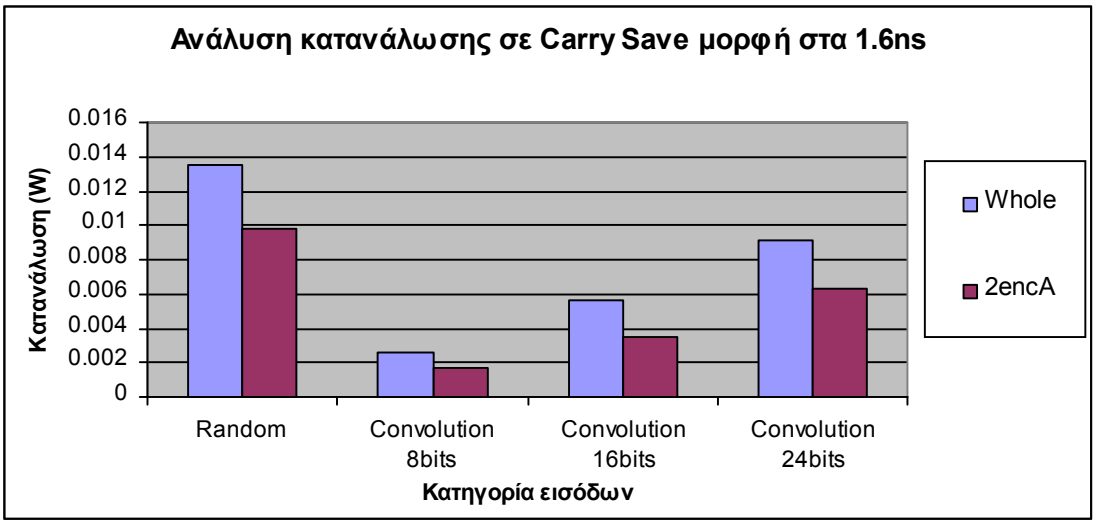
Πίνακας 3.5 Πίνακας επιφάνειας 2encA σε Carry Save μορφή στα 1.6ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	36182
2encA	38598

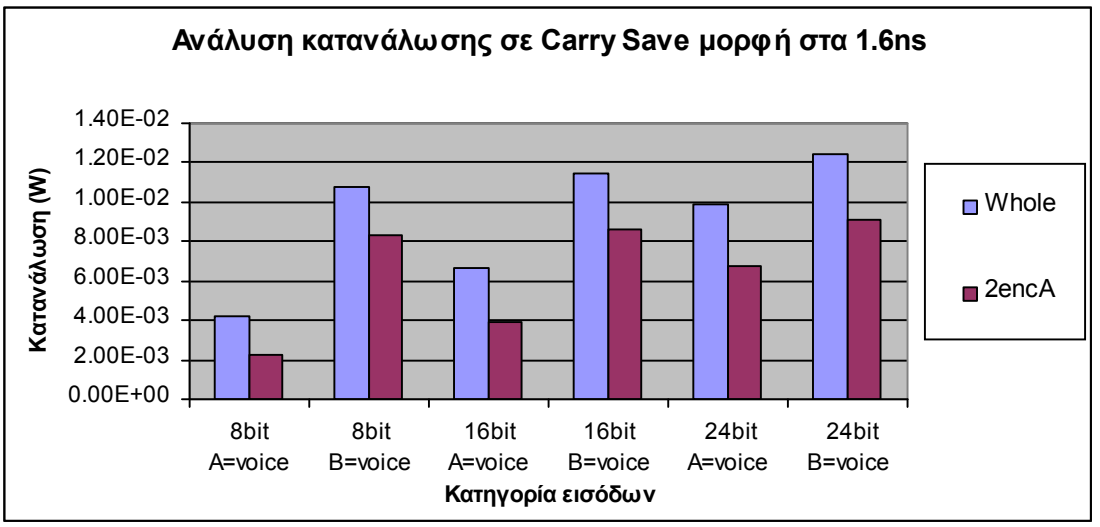
Πίνακας 3.6 Πίνακας επιφάνειας 2encA σε δυαδική μορφή στα 2.1ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	58197
2encA	54352

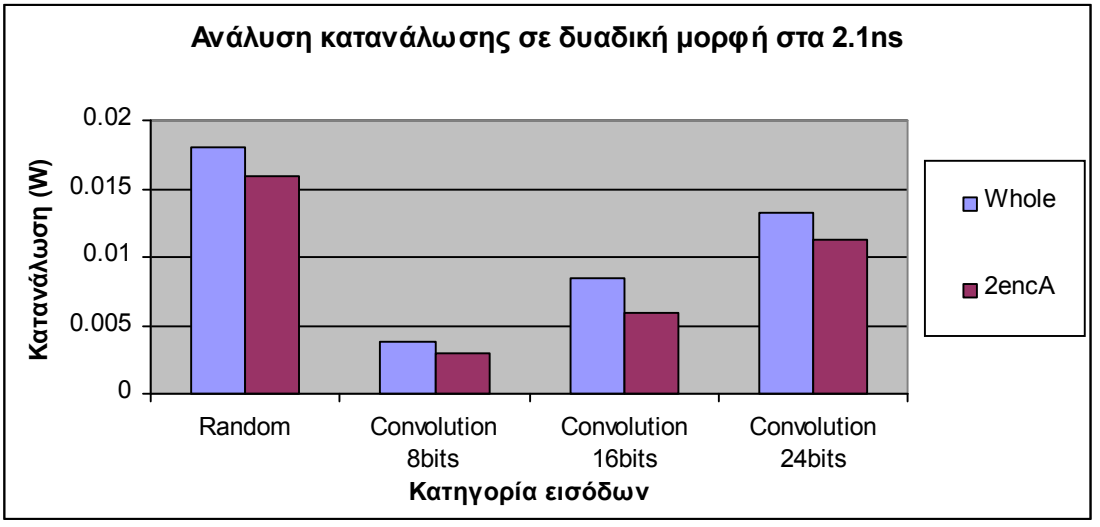
Έχει μετρηθεί η κατανάλωση για τις ίδιες περιπτώσεις όπως και για την πρώτη υλοποίηση του 2encA. Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα.



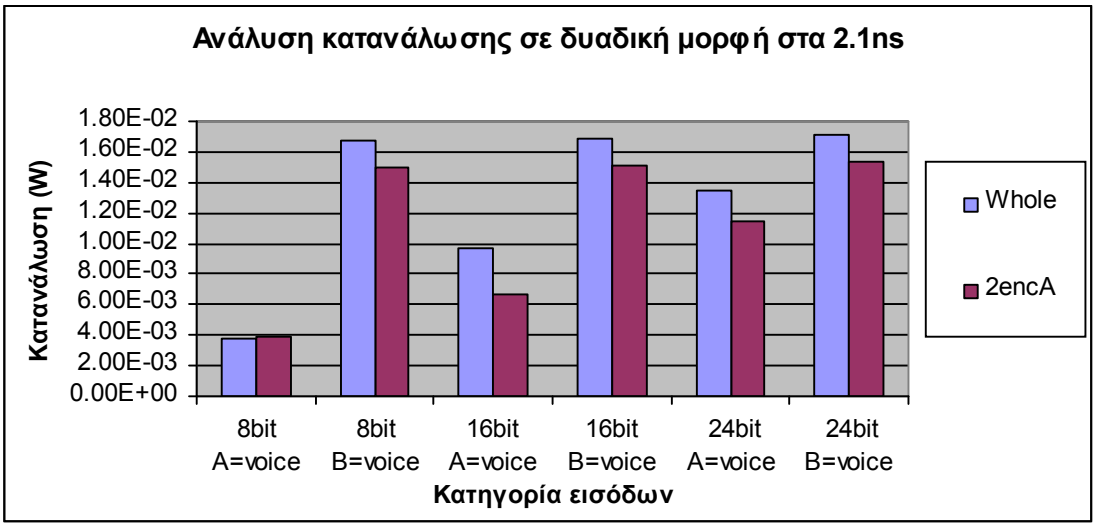
Σχήμα 3.16 Αποτελέσματα 2ης υλοποίησης 2encA σε CS μορφή στα 1.6ns



Σχήμα 3.17 Αποτελέσματα 2^{ης} υλοποίησης 2encA σε CS μορφή στα 1.6ns



Σχήμα 3.18 Αποτελέσματα 2^{ης} υλοποίησης 2encA σε binary μορφή στα 2.1ns



Σχήμα 3.19 Αποτελέσματα 2^{ης} υλοποίησης 2encA σε binary μορφή στα 2.1ns

Βλέπουμε από το Σχήμα 3.16 ότι για τυχαίους αριθμούς καθώς και για όταν και οι δύο είσοδοι είναι σήματα φωνής 24, 16 ή 8 bit ο αλγόριθμος 2encA παρουσιάζει μικρότερη κατανάλωση από τον κλασσική υλοποίηση. Αυτό συμβαίνει, γιατί τα υψηλότερης αξίας μερικά γινόμενα που προκύπτουν από τον πολλαπλασιασμό του B με το AH είναι τις περισσότερες φορές ίσα με 0, δεδομένου ότι τα σήματα φωνής (που εδώ είναι η είσοδος A) έχουν πολλά υψηλής αξίας ψηφία μηδενικά. Επομένως, το switching activity των αριστερότερων bits του αποτελέσματος, που είναι σε Carry-Save μορφή, παραμένει το ίδιο.

Επίσης ο χωρισμός σε δύο block του πολλαπλασιασμού έχει πλεονέκτημα και για τυχαίες τιμές αφού τα ενδιάμεσα bits που επικαλύπτονται στα δύο block παράγονται πιο γρήγορα άρα υπόκεινται σε περαιτέρω απλοποιήσεις στην συνέχεια του κυκλώματος με τους carry save adders και τον Parallel-Prefix adder.

Επίσης δεν υπάρχει μεγάλη διαφορά σε σχέση με τα αποτελέσματα σε binary μορφή αφού η επιφάνεια του 2encA γίνεται μικρότερη σε σχέση με το Whole εξαιτίας των ταχύτατα διαθέσιμων αποτελεσμάτων και συνεπώς των απλοποιήσεων που γίνονται από τον Design Compiler με την προσθήκη του 64-bit Parallel-Prefix adder.

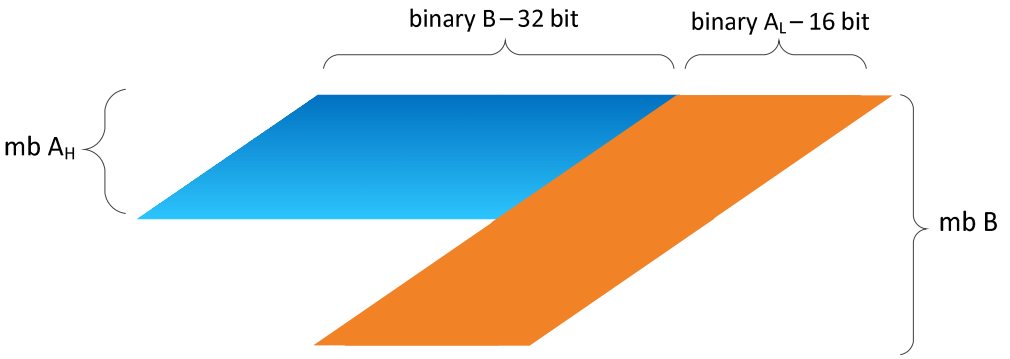
3.4 Αλγόριθμος TWO

Ο αλγόριθμος TWO είναι ο πρώτος αλγόριθμος που υλοποιήθηκε για να έχει καλύτερα αποτελέσματα δηλαδή χαμηλότερη κατανάλωση όταν η μία είσοδος είναι σήμα φωνής αλλά και ταυτόχρονα να έχει όσο το δυνατόν χαμηλότερη κατανάλωση όταν και οι δύο είσοδοι είναι σήματα φωνής.

3.4.1 Γενική δομή αλγορίθμου TWO

Αυτός ο αλγόριθμος έχει δύο εισόδους, τις A και B, όπου η είσοδος A προορίζεται να έχει σήματα φωνής αλλά και η B σε μερικές περιπτώσεις μπορεί να είναι κάποια τιμή από ένα σήμα φωνής. Η όλη λειτουργία του αλγορίθμου προσπαθεί να ελαχιστοποιήσει την κατανάλωση με χρήση καταμερισμού υλοποιώντας δύο επιμέρους πολλαπλασιασμούς με εισόδους στην πρώτη μονάδα τα 16 MSB του A και το B κωδικοποιώντας τα 16 MSB του A και στην δεύτερη μονάδα τα 16 LSB του A και το B κωδικοποιώντας το B.

Αυτή η δομή φαίνεται και στο Σχήμα 3.20.

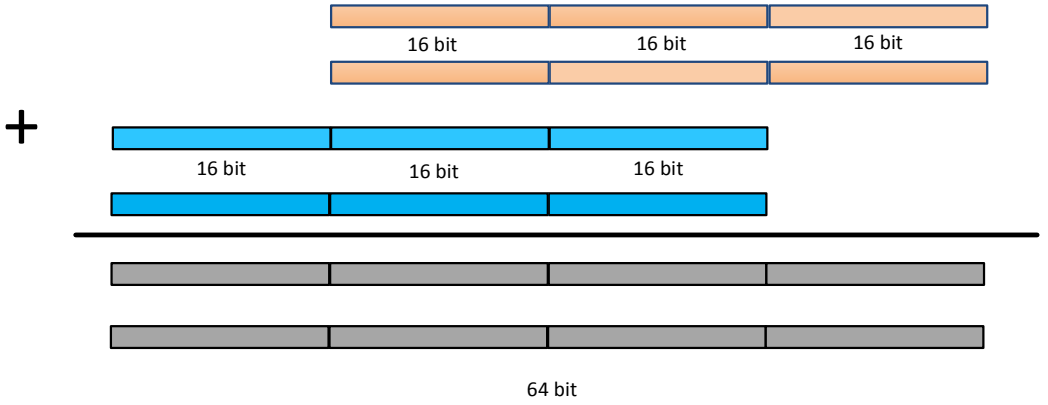


Σχήμα 3.20 Δομή TWO

Το άνω μέρος του πολλαπλασιασμού (μπλε παραλληλόγραμμο) θα αναφέρεται ως HIGH και το κάτω μέρος (πορτοκαλί παραλληλόγραμμο) θα αναφέρεται ως LOW.

Το αποτέλεσμα αυτών των δύο επιμέρους πολλαπλασιασμών θα είναι δύο αριθμοί carry save μορφής 48-bit από κάθε μονάδα, με τα 48 bit που εξάγονται σαν αποτέλεσμα από τον πολλαπλασιασμό του A_H με το B να βρίσκονται κατά 16 bit αριστερότερα από τα 48 bit της άλλης μονάδας. Αυτό φαίνεται και στο Σχήμα 3.21.

Οι δύο carry save αριθμοί θα πρέπει να προστεθούν ώστε να βγει στην έξοδο του TWO μόνο ένας carry save αριθμός μήκους 64 bit.



Σχήμα 3.21 Έξοδοι μονάδων TWO

3.4.2 Πρώτη υλοποίηση TWO

Η πρώτη υλοποίηση του αλγορίθμου TWO έγινε με εξ' ολοκλήρου με στοιχεία από την βιβλιοθήκη της Designware, με βασικό εργαλείο να είναι το DW02_MULTP του οποίου οι είσοδοι, οι έξοδοι και τα χαρακτηριστικά έχουν εξηγηθεί σε προηγούμενο κεφάλαιο.

Όπως και στον αλγόριθμο 2encA έτσι και σε αυτόν τον αλγόριθμο θα πρέπει να γίνουν οι ίδιες αλλαγές στο netlist για να βγαίνει σωστό το αποτέλεσμα.

3.4.2.1 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το όλο κύκλωμα του TWO έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Κάθε ένα από τα δύο DW02_MULTP που χρησιμοποιούνται έχει ως έξοδο δύο καταχωρητές 48-bit ο καθένας όπως φαίνεται και στο Σχήμα 3.21. Οι τέσσερις αυτοί καταχωρητές μετά την επέκταση προσήμου τροφοδοτούν δύο διαδοχικούς Carry Save Adders όπως φαίνεται στο Σχήμα 3.7.

Η πρόσθεση έγινε όπως ακριβώς και για τον αλγόριθμο 2ecnA πάλι όμως βάζοντας στην θέση του 50ου bit του out0_low μία μονάδα αντί για μηδέν και αντιστρέφοντας το αντίστοιχο bit του out1_low κάνοντας επέκταση προσήμου με μονάδες στο out1_low και τέλος χρησιμοποιώντας 2 carry save adders.

3.4.3 Αποτελέσματα πρώτης υλοποίησης TWO

Οι συγκρινόμενοι αλγόριθμοι είναι η πρώτη υλοποίηση του αλγορίθμου TWO με την πρώτη υλοποίηση του αλγορίθμου Whole όπου και οι δύο είναι άμεσα συγκρινόμενοι αφού και οι δύο χρησιμοποιούν modules της DesignWare.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους παρακάτω πίνακες 3.7 και 3.8:

Πίνακας 3.7 Πίνακας επιφάνειας TWO σε Carry Save μορφή στα 1.6ns

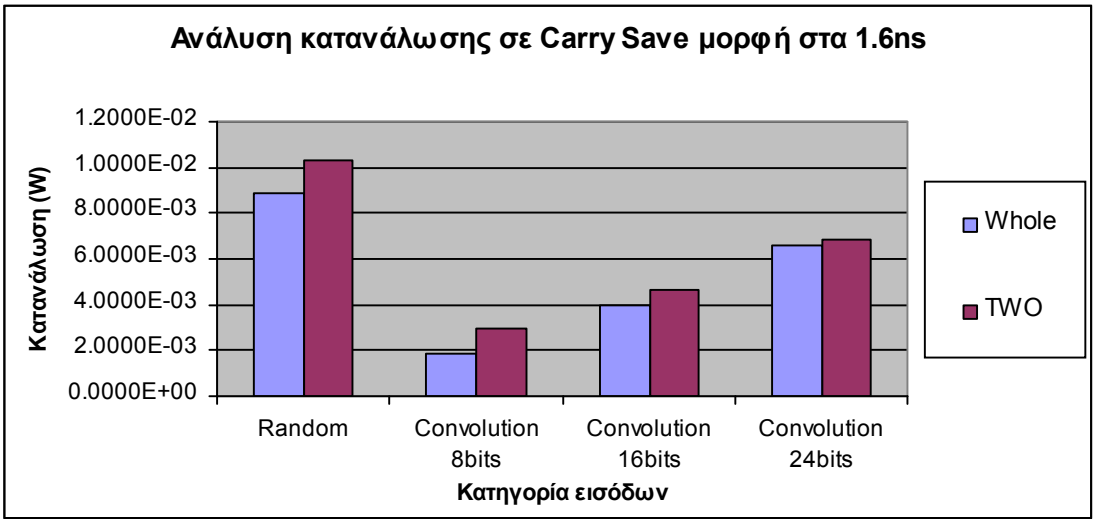
Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	33066
TWO	42425

Πίνακας 3.8 Πίνακας επιφάνειας TWO σε δυαδική μορφή στα 2.1ns

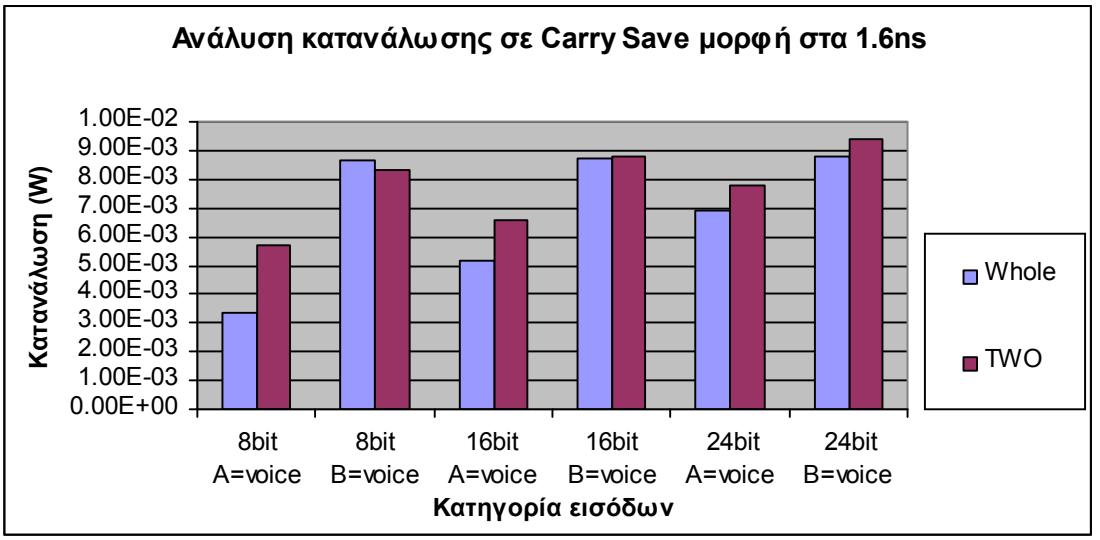
Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	44213
TWO	57858

Έχει μετρηθεί η κατανάλωση για τις ίδιες περιπτώσεις όπως και για την πρώτη υλοποίηση του 2encA.

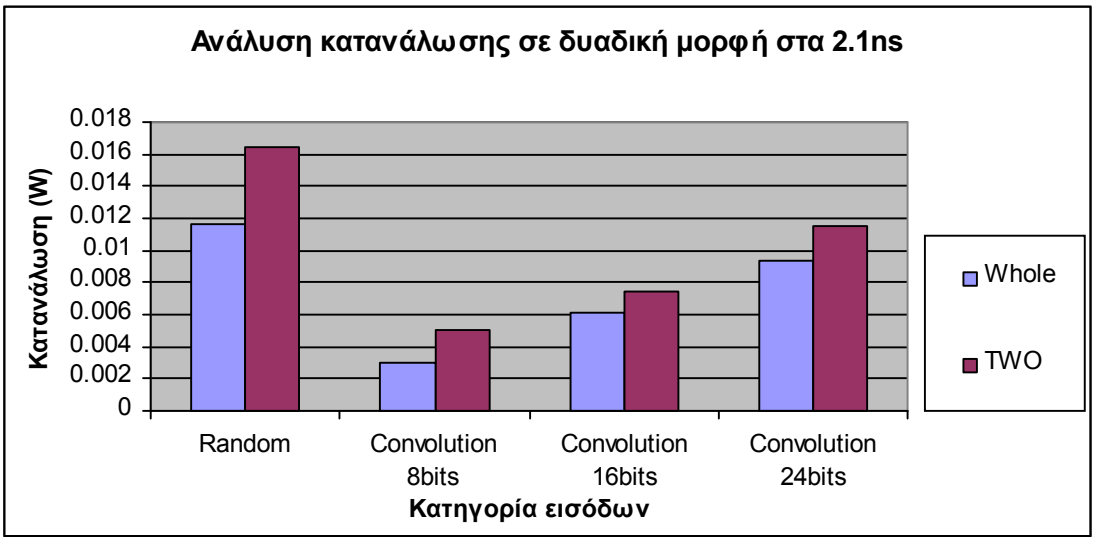
Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα.



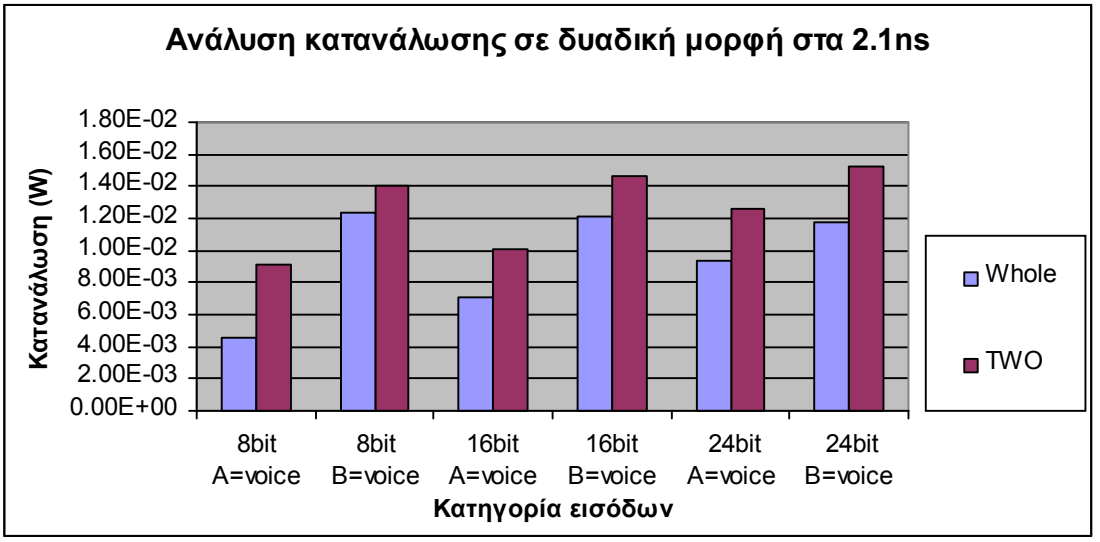
Σχήμα 3.22 Αποτελέσματα 1^{ης} υλοποίησης TWO σε CS μορφή στα 1.6ns



Σχήμα 3.23 Αποτελέσματα 1ης υλοποίησης TWO σε CS μορφή στα 1.6ns



Σχήμα 3.24 Αποτελέσματα 1ης υλοποίησης TWO σε binary μορφή στα 2.1ns



Σχήμα 3.25 Αποτελέσματα 1ης υλοποίησης TWO σε binary μορφή στα 2.1ns

Βλέπουμε πως όταν το αποτέλεσμα είναι είτε σε carry save ή σε δυαδική μορφή, ότι σε κάθε περίπτωση από τις προαναφερθείσες, ο αλγόριθμος TWO παρουσιάζει μεγαλύτερη κατανάλωση και αυτό συμβαίνει γιατί το κύκλωμα που χρειάζεται για το παραπάνω encoding και οι 2 carry save αθροιστές επιβαρύνουν το συνολικό κύκλωμα με μεγαλύτερη επιφάνεια άρα και περισσότερη κατανάλωση. Επιπρόσθετα φαίνεται πως το κύκλωμα που χρησιμοποιεί η DesignWare για τις κωδικοποιήσεις και την παραγωγή των μερικών γινομένων δεν είναι κατάλληλο για να αναδειχτούν τα πλεονεκτήματα του αλγόριθμου TWO αλλά έχει σχεδόν πάντοτε καλύτερα αποτελέσματα ως ένα block (αλγόριθμος Whole).

3.4.4 Δεύτερη υλοποίηση TWO

Η δεύτερη υλοποίηση του αλγορίθμου TWO έγινε με δικές μας μονάδες αλλά και με στοιχεία από την βιβλιοθήκη της Designware, όπως το DW02_TREE. Η δεύτερη υλοποίηση δεν διαφέρει από την πρώτη ως προς το σχήμα και επίσης και αυτή έχει ως έξοδο έναν αριθμό μορφής carry save που το άθροισμα του είναι το γινόμενο τον εισόδων του A και B όπου οι A και B είναι προσημασμένοι αριθμοί σε μορφή συμπληρώματος ως προς δύο που είναι οι είσοδοι του TWO.

3.4.4.1 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το όλο κύκλωμα του TWO έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Encoder

Για να κωδικοποιηθούν τα ψηφία του A_H και του B σε modified booth ψηφία χρησιμοποιείται μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τρία σήματα για την κάθε τριάδα. Η μόνη διαφορά για το HIGH κομμάτι του TWO αλλά και για το LOW κομμάτι του, είναι ότι για το HIGH κομμάτι του κωδικοποιούνται 8 τριάδες του A ενώ για το LOW κωδικοποιούνται 16 τριάδες του B. Συνολικά κωδικοποιούνται από τα δύο μέρη του TWO 24 τριάδες σε $3 \cdot 24 = 72$ σήματα.

Τα τρία αυτά σήματα είναι τα: m, x, και x2 και θα εξηγηθούν με βάση την κωδικοποίηση του A_H ενώ παρόμοια είναι η σημασία τους και για την κωδικοποίηση του B απλά το μόνο που αλλάζει είναι τα bit εισόδου.

Το x2 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +2 ή −2, το οποίο σημαίνει ότι το μερικό γινόμενο που πρέπει να προστεθεί είναι ή το B ολισθημένο κατά μία θέση ή το συμπλήρωμα αυτού. Σε κάθε άλλη περίπτωση το x2 είναι 0.

Το x1 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +1 ή −1, το οποίο σημαίνει ότι το μερικό γινόμενο που πρέπει να προστεθεί είναι ή το B ή το συμπλήρωμα αυτού. Σε κάθε άλλη περίπτωση το x1 είναι 0.

Το m σε άλλες κωδικοποιήσεις συνήθως είναι 1 όταν και το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1, που αυτό συνεπάγεται ότι πρέπει να προστεθεί το συμπλήρωμα του B είτε ολισθημένο είτε όχι, εκτός από την περίπτωση που η τριάδα είναι το 111, οπότε δεν προστίθεται τίποτα. Επειδή όμως ο αλγόριθμος είναι κατασκευασμένος να πολλαπλασιάζει πολλούς μικρούς αριθμούς, άρα και μικρούς αρνητικούς αριθμούς, θα επιβαρυνόταν εξαιρετικά η κατανάλωση αν αντιμετωπίζαμε το 111 σαν αρνητικό μηδέν, οπότε το m είναι 1 όταν το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1 εκτός από την περίπτωση της τριάδας 111.

Στον Πίνακα 3.4 αναφέρονται εκτεταμένα οι τιμές των προαναφερθέντων σημάτων.

Έτσι οι πύλες που χρησιμοποιήθηκαν για να υλοποιηθούν αυτά τα σήματα είναι οι παρακάτω:

$$m = A_{2i+1} \text{ xor } (A_{2i+1} \text{ and } A_{2i} \text{ and } A_{2i-1})$$

$$x2 = (\text{not}(A_{2i+1}) \text{ and } A_{2i} \text{ and } A_{2i-1}) \text{ or } (A_{2i+1} \text{ and not}(A_{2i}) \text{ and not}(A_{2i-1}))$$

$$x = (A_{2i} \text{ xor } A_{2i-1})$$

Και φαίνονται και στο Σχήμα 3.12.

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από την κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Η μονάδα όμως που χρησιμοποιείται από το HIGH μέρος του αλγορίθμου TWO δεν είναι πανομοιότυπη με την μονάδα που χρησιμοποιείται από το LOW μέρος, όπως φαίνεται και από το Σχήμα 3.20.

Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων του HIGH μέρους λαμβάνει ως είσοδο 3 σήματα των 8 bit αφού έχουν κωδικοποιηθεί 8 τριάδες του A_H (τα τρία αυτά σήματα είναι τα: m, x, και x2) και άλλο 1 σήμα 32-bit που έχει την τιμή του B. Όμως η μονάδα παραγωγής μερικών γινομένων του LOW μέρους λαμβάνει ως είσοδο 3 σήματα των 16 bit αφού έχουν κωδικοποιηθεί 16 τριάδες του B και άλλο 1 σήμα 16-bit που έχει την τιμή του A_L.

Οι μονάδες αυτές χρησιμοποιούν τα παραπάνω σήματα ώστε στις εξόδους αυτών να γίνουν διαθέσιμα τα μερικά γινόμενα. Λόγω της πιθανής ολίσθησης στις περιπτώσεις όπου το x2 είναι ίσο με 1, τα μερικά γινόμενα που θα παραχθούν θα είναι 33 bit για το HIGH μέρος και 17 bit για το LOW μέρος. Η διεργασία αυτή χωρίζεται σε 3 σκέλη. Το πρώτο σκέλος είναι οι πύλες για το πρώτο bit του μερικού γινομένου, το δεύτερο σκέλος είναι οι πύλες για όλα τα υπόλοιπα bit εκτός από το 33ο ή το 17ο, και το τρίτο σκέλος είναι οι πύλες που θα χρησιμοποιηθούν για το 33ο ή το 17ο αντίστοιχα bit.

Στο Σχήμα 3.13 φαίνονται οι πύλες που χρησιμοποιήθηκαν. Οι μόνες διαφοροποιήσεις είναι ότι για το LOW μέρος η συνθήκη που θα ισχύει για το k θα είναι $1 \leq k < 16$,στο πρώτο και στο δεύτερο σχήμα οι είσοδοι αντί για B θα είναι A, και στο τρίτο σχήμα του 3.13 αντί για B(31) η πύλη AND θα δέχεται ως είσοδο το A(15).

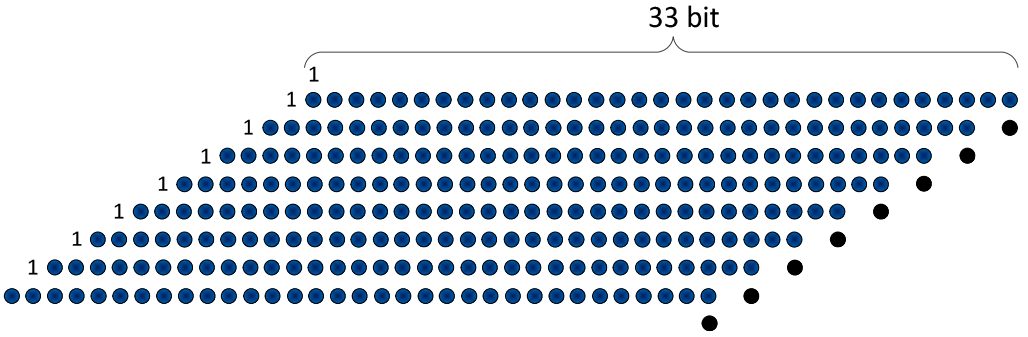
Έτσι με αυτές τις πύλες η μονάδα παραγωγής μερικών γινομένων για το HIGH μέρος βγάζει στην έξοδο της 8 αριθμούς από 33 bit ο καθένας και η μονάδα παραγωγής μερικών γινομένων για το LOW μέρος βγάζει στην έξοδο της 16 αριθμούς από 17 bit ο καθένας.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Όπως και στην μονάδα παραγωγής μερικών γινομένων έτσι και εδώ θα χρειαστεί μία μονάδα δενδρικού συμπιεστή τύπου Wallace στο HIGH και στο LOW μέρος αντίστοιχα οι οποίες δεν είναι ταυτόσημες.

Η πρώτη μονάδα για το HIGH μέρος παίρνει ως είσοδο 8 33-bit αριθμούς και μαζί με τα κρατούμενα και τους διορθωτικούς όρους τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

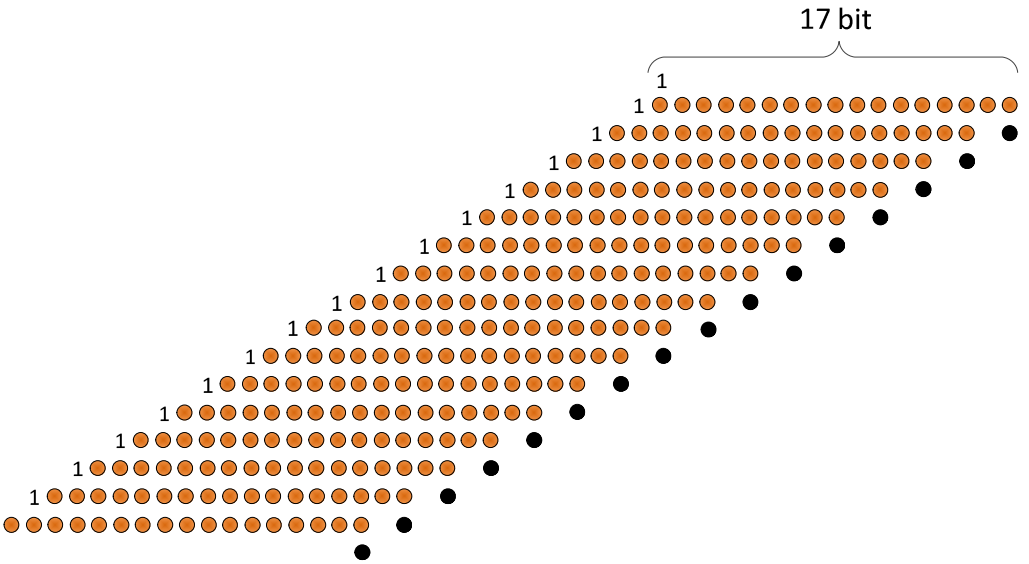
Τα μερικά γινόμενα, τα κρατούμενα και οι διορθωτικές μονάδες παρουσιάζονται στο Σχήμα 3.26 στην σωστή διάταξη που πρέπει να προστεθούν πάντα για το HIGH μέρος του πολλαπλασιασμού.



Σχήμα 3.26 Μερικά γινόμενα, διορθωτικοί όροι και κρατούμενα (HIGH)

Η δεύτερη μονάδα για το LOW μέρος παίρνει ως είσοδο 16 17-bit αριθμούς και μαζί με τα κρατούμενα και τους διορθωτικούς όρους τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα, τα κρατούμενα και οι διορθωτικές μονάδες παρουσιάζονται στο Σχήμα 3.27 στην σωστή διάταξη που πρέπει να προστεθούν για το LOW μέρος του πολλαπλασιασμού.



Σχήμα 3.27 Μερικά γινόμενα, διορθωτικοί όροι και κρατούμενα (LOW)

Με βάση προηγούμενη ανάλυση θα πρέπει να προσθέσουμε τα συμπληρώματα ως προς δύο των μερικών γινομένων στα οποία το m για την αντίστοιχη κωδικοποιημένη τριάδα του A_H ή του B είναι ίσο με 1. Αυτό επιτυγχάνεται προσθέτοντας μία μονάδα στο LSB κάθε μερικού γινομένου όταν το σήμα m αυτής της σειράς είναι ίσο με 1. Στα σχήματα 3.27 και 3.26 αυτό απεικονίζεται με μαύρες τελείες. Η πρώτη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο, η δεύτερη οριζόντια σειρά από bit αποτελεί το δεύτερο μερικό γινόμενο, και ούτω καθεξής. Κάθε μερικό γινόμενο βρίσκεται 2 θέσεις αριστερότερα από το προηγούμενο μερικό γινόμενο λόγω της MB κωδικοποίησης.

Ακριβώς έτσι όπως δικαιολογήθηκαν οι διορθωτικοί όροι για τον αλγόριθμο 2encA, μπορούν να χρησιμοποιηθούν και εδώ με τον ίδιο τρόπο.

Εν κατακλείδι για την περίπτωση του TWO αλγορίθμου οι σχέσεις των διορθωτικών όρων είναι οι εξής:

$$ct_H = 2^{47} + 2^{45} + 2^{43} + 2^{41} + 2^{39} + 2^{37} + 2^{35} + 2^{33} + 2^{32}$$

$$ct_L = 2^{47} + 2^{45} + 2^{43} + 2^{41} + 2^{39} + 2^{37} + 2^{35} + 2^{33} + 2^{31} + 2^{29} + 2^{27} + 2^{25} + 2^{23} + 2^{21} + 2^{19} + 2^{17} + 2^{16}$$

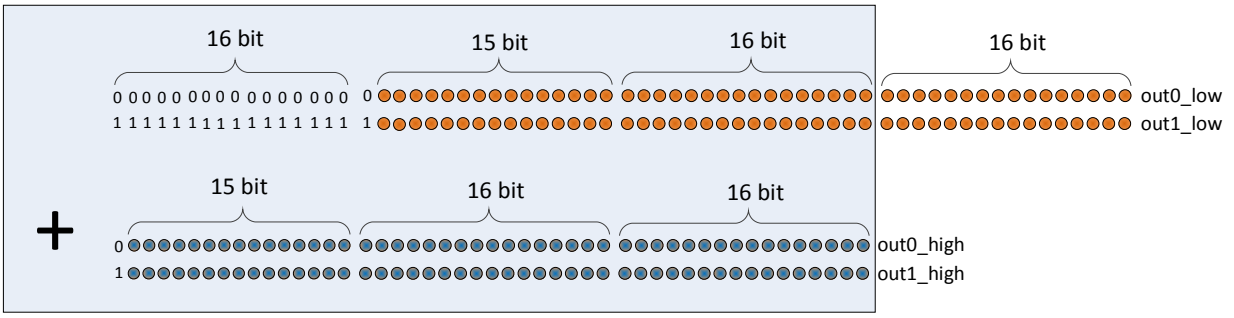
Όλοι οι παραπάνω διορθωτικοί όροι προστίθενται και φαίνονται στα δύο παραπάνω σχήματα εκτός από το 2^{47} το οποίο θα προστεθεί αργότερα.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως το αποτέλεσμα ενός πολλαπλασιασμού 16x32 bits θα είναι 48 bits, έτσι ο αριθμός X θα είναι 48 bit. Τα μερικά γινόμενα που είναι δεν είναι 48 bits εισάγονται με μηδενικά στα αριστερά και στα δεξιά τους για να γίνεται σωστά η άθροιση. Τα μηδενικά απλοποιούνται από τον design compiler και είναι σαν να μην μπαίνουν ψηφία σε αυτά τα σημεία δηλαδή δεν θα υπάρξει κανένας full adder ή half adder που να πάρει ως είσοδο το 0.

Τελική άθροιση

Εφόσον δεν έχουμε προσθέσει τον διορθωτικό όρο με βάρος 2^{47} γνωρίζουμε πως το αποτέλεσμα κάθε μονάδας wallace θα έχει το μηδέν ως MSB και στο carry αλλά και στο save.

Έτσι δεν έχουμε παρά να προσθέσουμε την μονάδα με βάρος 2^{47} και να κάνουμε επέκταση προσήμου στο αποτέλεσμα της LOW μονάδας wallace στο τώρα όμως γνωστό πρόσημο. Αυτό φαίνεται στο Σχήμα 3.28 όπως και ποια ψηφία πρέπει να προστεθούν για να βγει το τελικό αποτέλεσμα σε carry save.



Σχήμα 3.28 Τελικός αθροιστής και επέκταση προσήμου

Η πρόσθεση που εσωκλείεται στο παραλληλεπίπεδο του προηγούμενου σχήματος υλοποιήθηκε με 3 τρόπους:

Α. 2 csa

Το παραπάνω τετράγωνο υποδεικνύει τους καταχωρητές που θα μπουν ως είσοδοι στο DW01_csa που είναι το εργαλείο της DesignWare που υλοποιεί τον carry save adder. Στο σχήμα 3.26 είναι εμφανείς οι απλοποιήσεις που γίνονται θέτοντας τα 17 MSB του out0_low ως 0 και τα 17 MSB του out1_low ως 1. Οι τρεις πρώτοι καταχωρητές μπαίνουν ως είσοδοι στον πρώτο Carry Save Adder και ο αριθμός 48-bit σε carry save μορφή που εξέρχεται ως έξοδος από αυτή την μονάδα εισέρχεται σαν είσοδος μαζί με τον τέταρτο καταχωρητή (out1_high) στον δεύτερο Carry Save Adder. Το αποτέλεσμα αυτού είναι ένας 48-bit carry save αριθμός ο οποίος μαζί με τα 16 LSB των out0_low και out1_low, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του TWO, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

Β. wallace tree

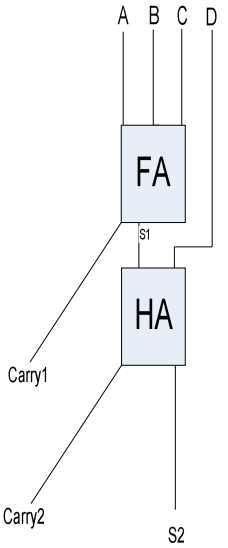
Το παραπάνω τετράγωνο υποδεικνύει τους καταχωρητές που θα μπουν ως είσοδοι στο DW02_tree που είναι το εργαλείο της DesignWare που υλοποιεί τον δενδρικό συμπιεστή Wallace.

Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως οι είσοδοι μας έχουν μήκος 48 bit, έτσι ο αριθμός X θα είναι 48 bit. Προφανώς και θα υπάρξουν απλοποιήσεις στο δέντρο wallace λόγω των μηδενικών και των άσσων του out0_low και out1_low αντίστοιχα όπως και των MSB των δύο άλλων καταχωρητών. Το αποτέλεσμα αυτού είναι ένας 48-bit carry save αριθμός ο οποίος μαζί με τα 16 LSB των out0_low και out1_low, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του TWO, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

Γ. adder 4 to 2

Τελευταία επιλογή για την άθροιση των παραπάνω ψηφίων ήταν η υλοποίηση ενός αθροιστή 4 προς 2. Ας θεωρήσουμε τις εξής εισόδους: out0_low2 (48- bit με τα 17 MSB ίσα με 0), out1_low2 (48- bit με τα 17 MSB ίσα με 1), out0_high2 (48- bit με το MSB ίσο με 0), out1_high2 (48- bit με το MSB ίσο με 1). Οι έξοδοι του αθροιστή είναι δύο σήματα 48-bit, τα οποία ονομάζονται CARRY και SAVE.

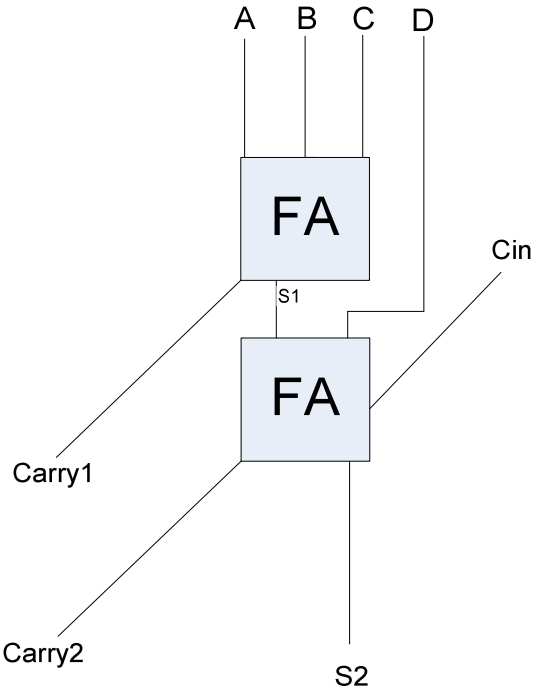
Οι δομικές μονάδες που χρησιμοποιεί ο αθροιστής 4 προς 2 είναι τρείς με την πρώτη μονάδα να ονομάζεται F4ADD, η οποία απεικονίζεται στο σχήμα 3.29 και παίρνει ως είσοδο 4 ψηφία (δηλαδή όσα έχουμε για την δεξιότερη πρόσθεση) και έχει ως έξοδο 2 κρατούμενα και 1 save.



Σχήμα 3.29 Κύκλωμα F4ADD

Το S2 είναι προφανώς το SAVE(0), το Carry2 είναι το CARRY(1) και το Carry1 θα μεταφερθεί ως κρατούμενο στην επόμενη δομική μονάδα η οποία αθροίζει 5 ψηφία.

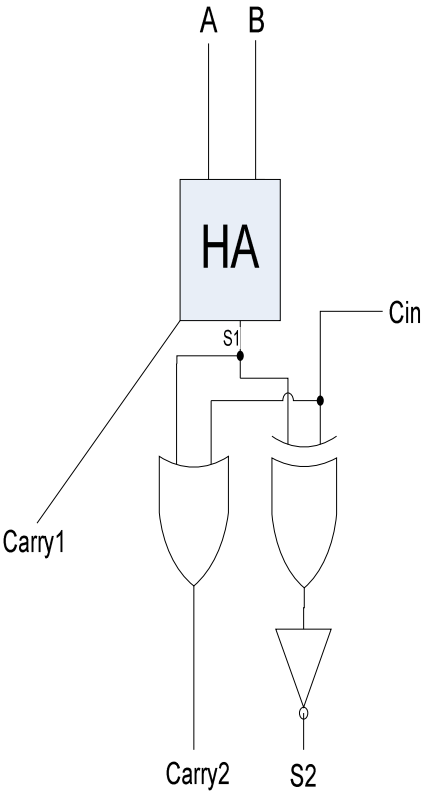
Η δεύτερη μονάδα ονομάζεται F5ADD, και απεικονίζεται στο Σχήμα 3.30. Παίρνει ως είσοδο 5 ψηφία (4 και ένα το οποίο είναι το κρατούμενο) και έχει ως έξοδο 2 κρατούμενα και 1 save.



Σχήμα 3.30 Κύκλωμα F5ADD

Εδώ το S2 είναι προφανώς το SAVE(k), το Carry2 είναι το CARRY(k+1) και το Carry1 θα μεταφερθεί ως κρατούμενο στην επόμενη βαθμίδα.

Η τρίτη μονάδα ονομάζεται F3ADD_AST, και απεικονίζεται στο σχήμα 3.31. Παίρνει ως είσοδο 3 ψηφία (2 και ένα το οποίο είναι το κρατούμενο) αλλά έχουν γίνει οι απαραίτητες απλοποιήσεις αφού ως είσοδος υπάρχει και μία μονάδα η οποία απλοποιεί το κύκλωμα και η μονάδα F3ADD_AST έχει ως έξοδο 2 κρατούμενα και 1 save.



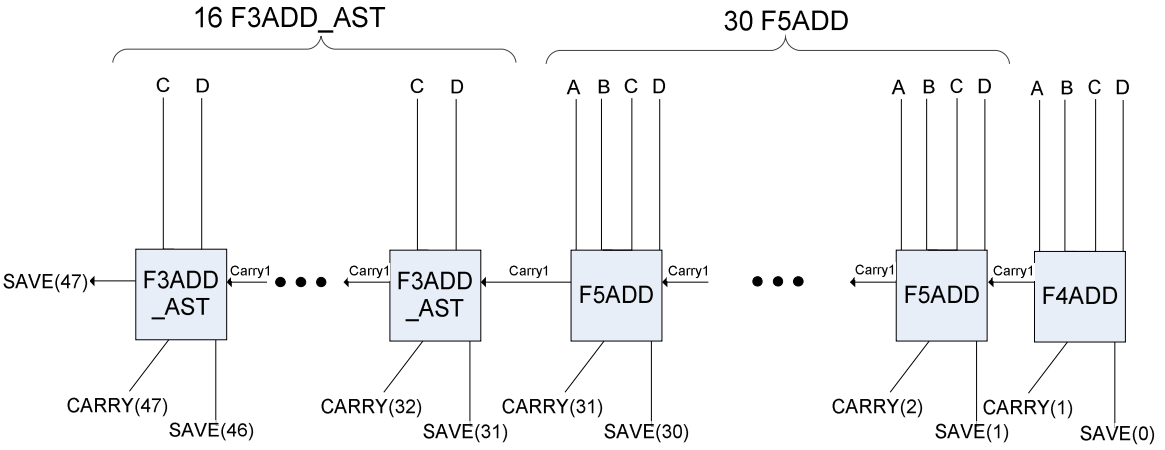
Σχήμα 3.31 Κύκλωμα F3ADD_AST

Εδώ το S2 είναι προφανώς το SAVE(k), το Carry2 είναι το CARRY(k+1) και το Carry1 θα μεταφερθεί ως κρατούμενο στην επόμενη βαθμίδα.

Στην τελευταία βαθμίδα προστίθενται δύο άσσοι, δύο μηδενικά και το κρατούμενο από την προηγούμενη βαθμίδα, άρα αφού ως γνωστόν η πρόσθεση δύο άσσων δίνει μόνο κρατούμενο το SAVE(47) θα είναι ίσο με το κρατούμενο από την προηγούμενη βαθμίδα.

Συνοψίζοντας ο αθροιστής 4 προς 2 παίρνει την μορφή που παρουσιάζεται στο σχήμα 3.32, και έχει ως αποτέλεσμα έναν 48-bit carry save αριθμό ο οποίος μαζί με τα 16 LSB των out0_low και out1_low, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του

TWO, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.



Σχήμα 3.32 Κύκλωμα αθροιστή 4 προς 2

3.4.5 Αποτελέσματα δεύτερης υλοποίησης TWO

Αρχικά συγκρίθηκαν ως προς την κατανάλωση οι τρεις διαφορετικές εκδοχές της τελικής άθροισης και τα αποτελέσματα παρατίθενται στον παρακάτω πίνακα για σήμα φωνής 16bit.

Πίνακας 3.9 Πίνακας κατανάλωσης TWO σε Carry Save μορφή στα 1.6ns

	2 csa	**wallace tree**	**adder 4 to 2**
random	0.0107	0.0107	0.0108
A=voice	0.00543	0.00557	0.00547
convolution	0.00236	0.00249	0.00236

Παρατηρούμε ότι οι διαφορές είναι μικρές αφού ο Design Compiler έχει κάνει τις απαραίτητες απλοποιήσεις αλλά διαλέγουμε την εκδοχή με την χαμηλότερη κατανάλωση.

Οπότε στο δεύτερο μέρος των μετρήσεων, χρησιμοποιήθηκε η εκδοχή της τελικής άθροισης με του δύο carry save adders και οι συγκρινόμενοι αλγόριθμοι είναι η δεύτερη υλοποίηση του αλγορίθμου TWO με την δεύτερη υλοποίηση του αλγορίθμου Whole.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους παρακάτω πίνακες 3.10 και 3.11:

Πίνακας 3.10 Πίνακας επιφάνειας TWO σε Carry Save μορφή στα 1.6ns

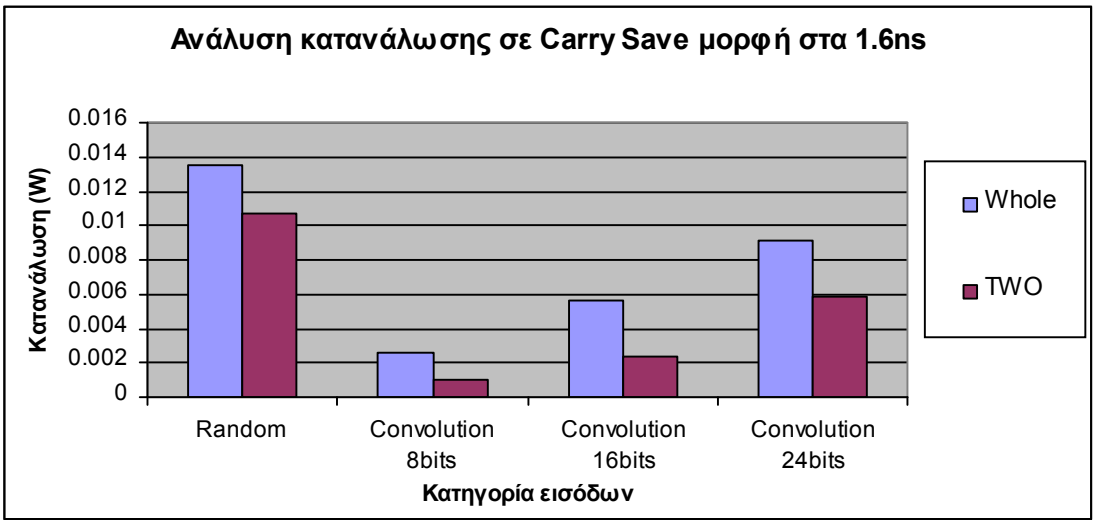
Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	36182
TWO	42465

Πίνακας 3.11 Πίνακας επιφάνειας TWO σε δυαδική μορφή στα 2.1ns

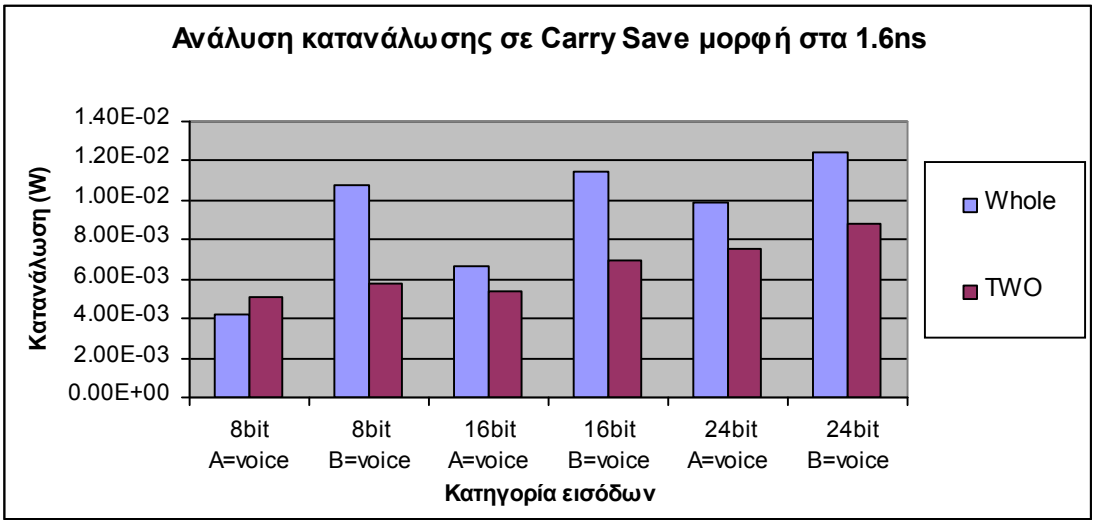
Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	58197
TWO	60031

Έχει μετρηθεί η κατανάλωση για τις ίδιες περιπτώσεις όπως και για την πρώτη υλοποίηση του 2encA.

Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα.



Σχήμα 3.33 Αποτελέσματα $2^{ης}$ υλοποίησης TWO σε CS μορφή στα 1.6ns



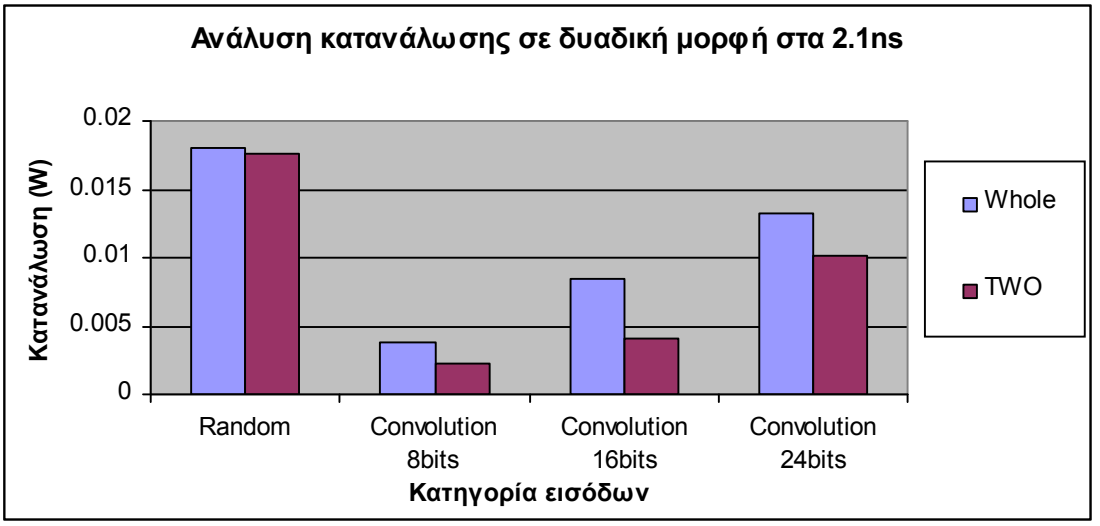
Σχήμα 3.34 Αποτελέσματα $2^{ης}$ υλοποίησης TWO σε CS μορφή στα 1.6ns

Σε αντίθεση με τον κλασσικό αλγόριθμο πολλαπλασιασμού εδώ κωδικοποιούμε τα 16MSB του A και όλο το B, οπότε έχουμε μεγαλύτερη επιφάνεια για την παραπάνω κωδικοποίηση (κωδικοποιούμε 16 περισσότερα bit σε σχέση με τον αλγόριθμο WHOLE) αλλά και για τους 2 carry save αθροιστές.

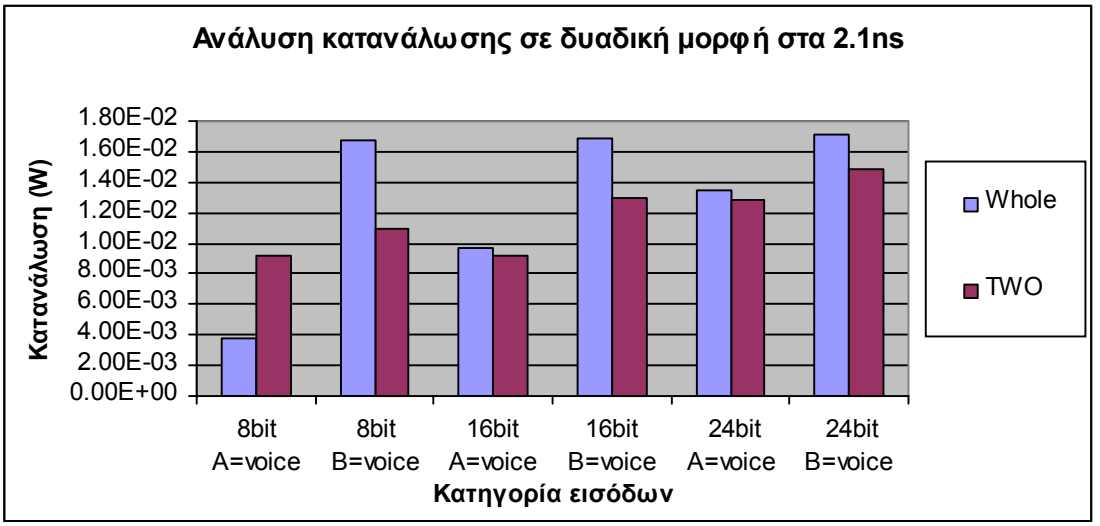
Βλέπουμε από το Σχήμα 3.33 ότι για τυχαίους αριθμούς καθώς και για όταν και οι δύο είσοδοι είναι σήματα φωνής 24, 16 bit ο αλγόριθμος TWO παρουσιάζει μικρότερη κατανάλωση από τον κλασσική υλοποίηση. Αυτό συμβαίνει, γιατί τα υψηλότερης αξίας μερικά γινόμενα που προκύπτουν από τον πολλαπλασιασμό του B με το AH είναι τις περισσότερες φορές ίσα με 0, δεδομένου ότι τα σήματα φωνής (που εδώ είναι η είσοδος A) έχουν πολλά υψηλής αξίας ψηφία μηδενικά. Επομένως, το switching activity των αριστερότερων bits του αποτελέσματος, που είναι σε Carry-Save μορφή, παραμένει το ίδιο αφού τα περισσότερα (στην περίπτωση των σημάτων φωνής 16 bit είναι όλα) από τα μερικά γινόμενα που φαίνονται στο Σχήμα 3.26 και αντιστοιχούν στο HIGH block του πολλαπλασιασμού είναι μηδενικά. Βλέπουμε ακόμα πως η βέλτιστη απόδοση του αλγορίθμου επιτυγχάνεται όταν το σήμα φωνής εισέλθει στην είσοδο A παρά στην είσοδο B, καθώς από αυτό επιτυγχάνεται το χαμηλό switching activity στο HIGH block. Επίσης ο χωρισμός σε δύο block του πολλαπλασιασμού έχει πλεονέκτημα και για τυχαίες τιμές αφού τα ενδιάμεσα bits που επικαλύπτονται στα δύο block παράγονται πιο γρήγορα άρα υπόκεινται σε περαιτέρω απλοποιήσεις στην συνέχεια του κυκλώματος με τους carry save adders και τον Parallel-Prefix adder. Τέλος στην περίπτωση που έχουμε στην είσοδο A σήμα φωνής 8-bit ο αλγόριθμος WHOLE πλεονεκτεί έναντι του αλγορίθμου TWO γιατί τα μηδενικά στο A έχουν πλέον γίνει τόσα πολλά που το κομμάτι LOW που φαίνεται στο Σχήμα 3.27 θα συνεχίσει να δουλεύει αφού έχει κωδικοποιηθεί σε Modified Booth το B (το block HIGH συνεχίζει προφανώς να μην δουλεύει), ενώ ταυτόχρονα τα περισσότερα μερικά γινόμενα του WHOLE

θα είναι μηδενικά λόγω της μικρής τιμής του A με αποτέλεσμα χαμηλότερο switching activity σε σχέση με τον αλγόριθμο TWO.

Η διαφορά σε σχέση με τα αποτελέσματα σε binary μορφή (Σχήμα 3.35 και Σχήμα 3.36) είναι ότι κατά τη μετατροπή του αποτελέσματος από Carry-Save, οι τιμές των αριστερότερων bits του τελικού πια αποτελέσματος αλλοιώνονται, λόγω της διάδοσης κρατουμένου στον 64-bit Parallel-Prefix adder, με αποτέλεσμα να αυξάνεται το switching activity των ψηφίων αυτών αλλά και γιατί ο Parallel-Prefix adder που χρησιμοποιείται έχει καλύτερα αποτελέσματα ως προς την κατανάλωση όταν δεν υπάρχουν πριν από αυτόν οι carry save adders που επιβαρύνουν κατά πολύ σε αυτή την περίπτωση την κατανάλωση, εξαιτίας απλοποιήσεων που δεν μπορούν να γίνουν.



Σχήμα 3.35 Αποτελέσματα 2ης υλοποίησης TWO σε binary μορφή στα 2.1ns



Σχήμα 3.36 Αποτελέσματα 2ης υλοποίησης TWO σε binary μορφή στα 2.1ns

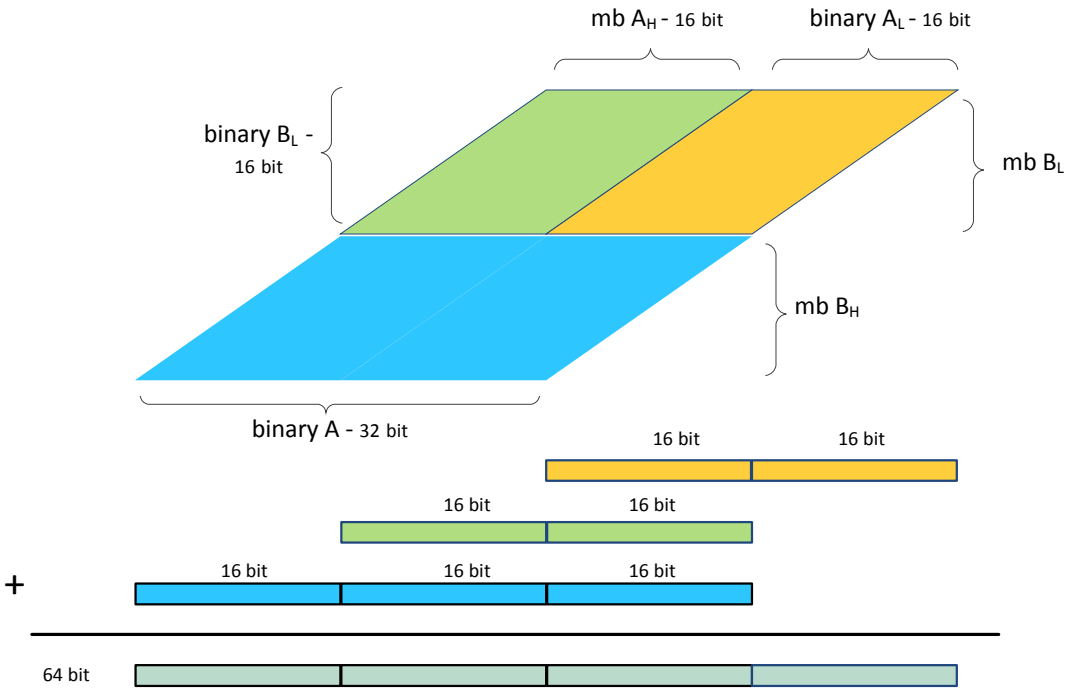
3.5 Αλγόριθμος MULT3

Ο αλγόριθμος MULT3 είναι ένας αλγόριθμος που υλοποιήθηκε για να έχει καλύτερα αποτελέσματα δηλαδή χαμηλότερη κατανάλωση όταν η μία είσοδος είναι σήμα φωνής αλλά και ταυτόχρονα να έχει χαμηλότερη κατανάλωση όταν και οι δύο είσοδοι είναι σήματα φωνής.

3.5.1 Γενική δομή αλγορίθμου MULT3

Αυτός ο αλγόριθμος έχει δύο εισόδους, τις A και B, όπου η είσοδος A προορίζεται να έχει σήματα φωνής αλλά και η B σε μερικές περιπτώσεις μπορεί να είναι κάποια τιμή από ένα σήμα φωνής. Η όλη λειτουργία του αλγορίθμου προσπαθεί να ελαχιστοποιήσει την κατανάλωση με χρήση καταμερισμού υλοποιώντας επιμέρους πολλαπλασιασμούς με εισόδους στην πρώτη μονάδα τα 16 MSB του A και τα 16 LSB του B κωδικοποιώντας τα 16 MSB του A, στην δεύτερη μονάδα τα 16 LSB του A τα 16 LSB του B κωδικοποιώντας τα 16 LSB του B και στην τρίτη μονάδα το A και τα 16 MSB του B κωδικοποιώντας τα 16 MSB του B.

Αυτή η δομή φαίνεται και στο Σχήμα 3.37.



Σχήμα 3.37 Δομή MULT3

Το άνω μέρος του πολλαπλασιασμού (μπλε παραλληλόγραμμο) θα αναφέρεται ως HIGH, το μεσαίο μέρος του πολλαπλασιασμού (πράσινο παραλληλόγραμμο) θα αναφέρεται ως MID και το κάτω μέρος (πορτοκαλί παραλληλόγραμμο) θα αναφέρεται ως LOW.

Το αποτέλεσμα αυτών των επιμέρους πολλαπλασιασμών θα είναι δύο αριθμοί carry save μορφής 32-bit από κάθε μονάδα MID και LOW, και ένας αριθμός 48 bit από την μονάδα HIGH. Αυτό φαίνεται και στο σχήμα 3.37.

Οι τρεις carry save αριθμοί θα πρέπει να προστεθούν ώστε να βγει στην έξοδο του MULT3 μόνο ένας carry save αριθμός μήκους 64 bit.

3.5.2 Υλοποίηση MULT3

Η υλοποίηση του αλγορίθμου MULT3 έγινε με εξ' ολοκλήρου με στοιχεία από την βιβλιοθήκη της Designware, με βασικό εργαλείο να είναι το DW02_MULTP του οποίου οι είσοδοι, οι έξοδοι και τα χαρακτηριστικά έχουν εξηγηθεί σε προηγούμενο κεφάλαιο.

Όπως και στον αλγόριθμο 2encA έτσι και σε αυτόν τον αλγόριθμο θα πρέπει να γίνουν οι ίδιες αλλαγές στο netlist για να βγαίνει σωστό το αποτέλεσμα με την μόνη διαφορά να είναι πως σε αυτή την περίπτωση η ίδια διαδικασία θα πρέπει να επαναληφθεί και για το A αλλά και για το B.

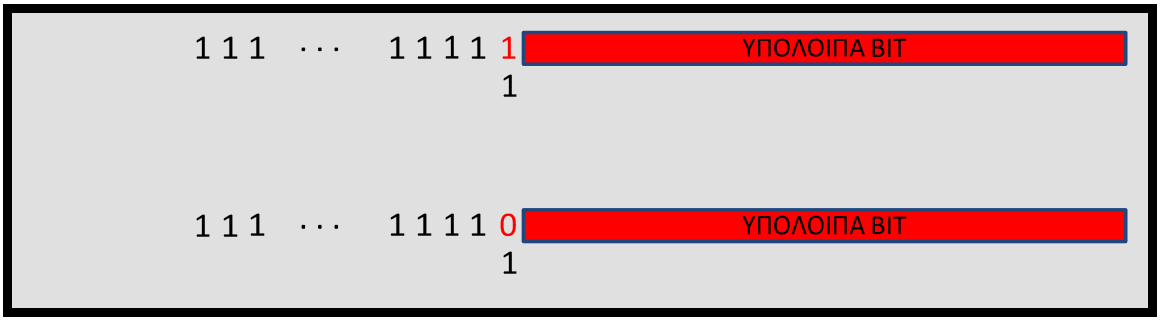
3.5.2.1 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το κύκλωμα του MULT3 έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Κάθε ένα από τα τρία DW02_MULTP που χρησιμοποιούνται έχει ως έξοδο δύο καταχωρητές, οι οποίοι είναι 48-bit για το HIGH μέρος, και 32-bit ο καθένας για τα MID και LOW μέρη, όπως φαίνεται και στο σχήμα 3.37. Είναι γνωστό πως για να γίνει η άθροιση σωστά θα πρέπει θα κάνουμε επέκταση προσήμου, όμως μία επέκταση προσήμου ενός μη γνωστού προσήμου θα επιβάρυνε το κύκλωμα. Είναι γνωστό επίσης πως ο ένας από τους δύο καταχωρητές που αποτελούν την έξοδο των DW02_MULTP μονάδων είναι πάντα θετικός, δηλαδή το 32ο bit του ενός καταχωρητή του MID και του LOW και το 48ο του ενός πάλι κταχωρητή του HIGH είναι πάντα μηδέν, άρα δεν χρειάζεται επέκταση προσήμου. Για τον άλλο καταχωρητή που είναι είτε θετικός ή αρνητικός ακολουθείται το εξής τέχνασμα:

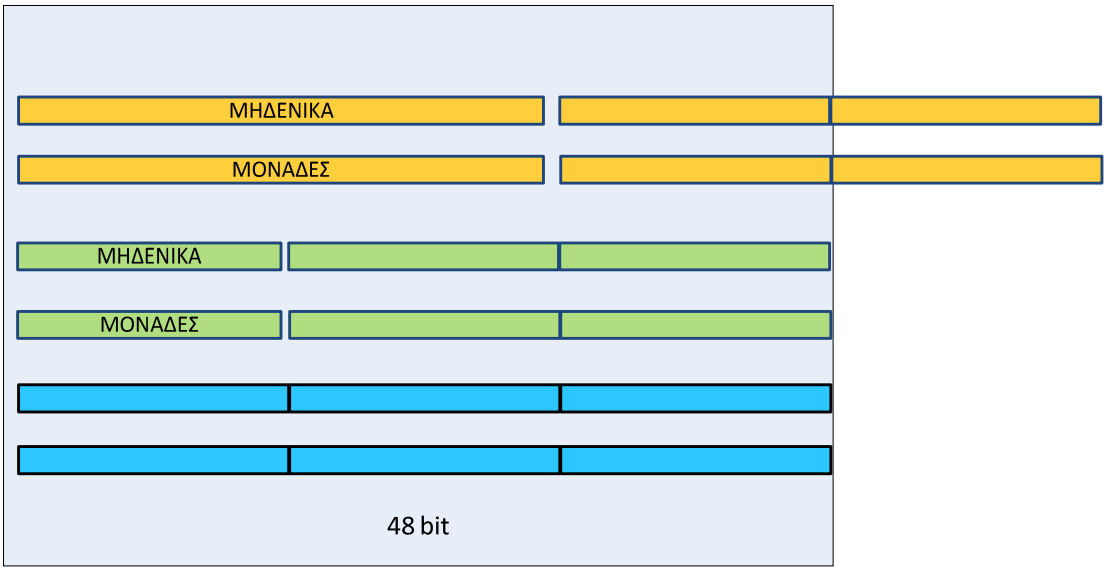
Αντιστέφουμε το MSB αυτού του καταχωρητή και προσθέτουμε μία μονάδα σε αυτή τη βαθμίδα ενώ παράλληλα κάνουμε επέκταση προσήμου σαν να ήταν το MSB μονάδα. Έτσι αν

το MSB του εν λόγω καταχωρητή ήταν 0 (άρα το αντίστροφο του θα ήταν το 1) θα έχουμε το άνω σχέδιο του σχήματος 3.38 και εν τέλει δεν προστίθεται έτσι κάποια παραπάνω αξία αφού είναι σαν να έχει γίνει επέκταση προσήμου στο 0. Αν όμως το MSB του εν λόγω καταχωρητή ήταν 1 (άρα το αντίστροφο του θα ήταν το 0) θα έχουμε το κάτω σχέδιο του σχήματος 3.38 και εν τέλει προστίθεται μία σειρά από μονάδες αφού είναι σαν να έχει γίνει επέκταση προσήμου στο 1.



Σχήμα 3.38 Τεχνική για επέκταση προσήμου

Όλοι οι έξι καταχωρητές μετά την αναστροφή των MSB όπου χρειάζεται φαίνονται στο σχήμα 3.39. Επιπλέον όμως θα πρέπει να προστεθούν και οι δύο μονάδες που αναφέρθησαν παραπάνω.



Σχήμα 3.39 Επέκταση προσήμου στα αποτελέσματα για τον MULT3

Ακολουθήθηκαν δύο τρόποι για την πρόσθεση αυτών των ψηφίων.

α) 5 csa

Το παραπάνω τετράγωνο υποδεικνύει τους καταχωρητές που θα μπουν ως είσοδοι στο DW01_csa που είναι το εργαλείο της DesignWare που υλοποιεί τον carry save adder. Στο σχήμα 3.39 είναι εμφανείς οι απλοποιήσεις που γίνονται θέτοντας τα 33 MSB του πρώτου καταχωρητή και τα 17 MSB του τρίτου καταχωρητή, ως μηδενικά και θέτοντας τα 32 MSB του δεύτερου και τα 16 MSB του τέταρτου καταχωρητή ως μονάδες για να βγει σωστό το αποτέλεσμα. Οι τρεις πρώτοι καταχωρητές μπαίνουν ως είσοδοι στον πρώτο Carry Save Adder και ο αριθμός 48-bit σε carry save μορφή που εξέρχεται ως έξοδος από αυτή την μονάδα εισέρχεται σαν είσοδος μαζί με τον τέταρτο καταχωρητή στον δεύτερο Carry Save Adder και με την ίδια διαδικασία προσθέτουμε και τις υπόλοιπες σειρές από ψηφία ενώ στο τέλος θα προστεθούν και οι δύο μονάδες που υποδεικνύονται στο σχήμα 3.38.

Το αποτέλεσμα των πέντε CSA είναι ένας 48-bit carry save αριθμός ο οποίος μαζί με τα 16 LSB των δύο πρώτων σειρών από bit, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του MULT3, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

b) Wallace

Το παραπάνω τετράγωνο υποδεικνύει τους καταχωρητές που θα μπουν ως είσοδοι στο DW02_tree που είναι το εργαλείο της DesignWare που υλοποιεί τον δενδρικό συμπιεστή Wallace.

Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως οι είσοδοι μας έχουν μήκος 48 bit, έτσι ο αριθμός X θα είναι 48 bit. Προφανώς και θα υπάρξουν απλοποιήσεις στο δέντρο wallace λόγω των μηδενικών και των μονάδων. Το αποτέλεσμα του wallace είναι ένας 48-bit carry save αριθμός ο οποίος μαζί με τα 16 LSB των δύο πρώτων σειρών από bit, αποτελεί τον 64-bit carry save αριθμό ο οποίος είναι η έξοδος του TWO, εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

3.5.3 Αποτελέσματα υλοποίησης MULT3

Αρχικά συγκρίθηκαν ως προς την κατανάλωση οι δύο διαφορετικές εκδοχές της τελικής άθροισης και τα αποτελέσματα παρατίθενται στον παρακάτω πίνακα για σήμα φωνής 16bit.

Πίνακας 3.12 Πίνακας κατανάλωσης MULT3 σε Carry Save μορφή στα 1.6ns

	5 csa	**wallace tree**
random	0.0110	0.0107
A=voice	0.00855	0.00834
convolution	0.00374	0.00332

Οπότε στο δεύτερο μέρος των μετρήσεων, χρησιμοποιήθηκε η εκδοχή της τελικής άθροισης με το wallace tree και οι συγκρινόμενοι αλγόριθμοι είναι η υλοποίηση του αλγορίθμου MULT3 με την πρώτη υλοποίηση του αλγορίθμου Whole.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους παρακάτω πίνακες 3.13 και 3.14:

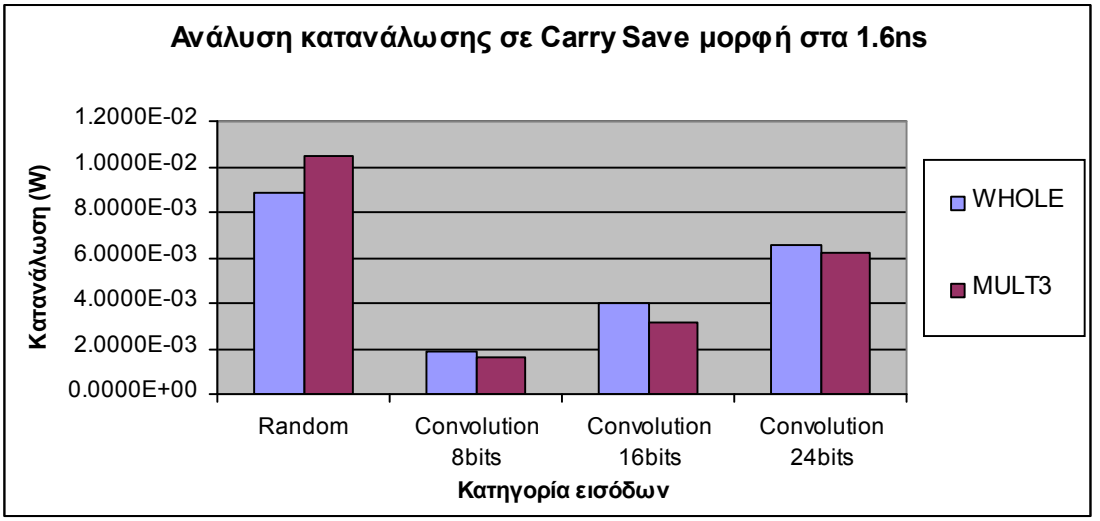
Πίνακας 3.13 Πίνακας επιφάνειας MULT3 σε Carry Save μορφή στα 1.6ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	33071
MULT3	42688

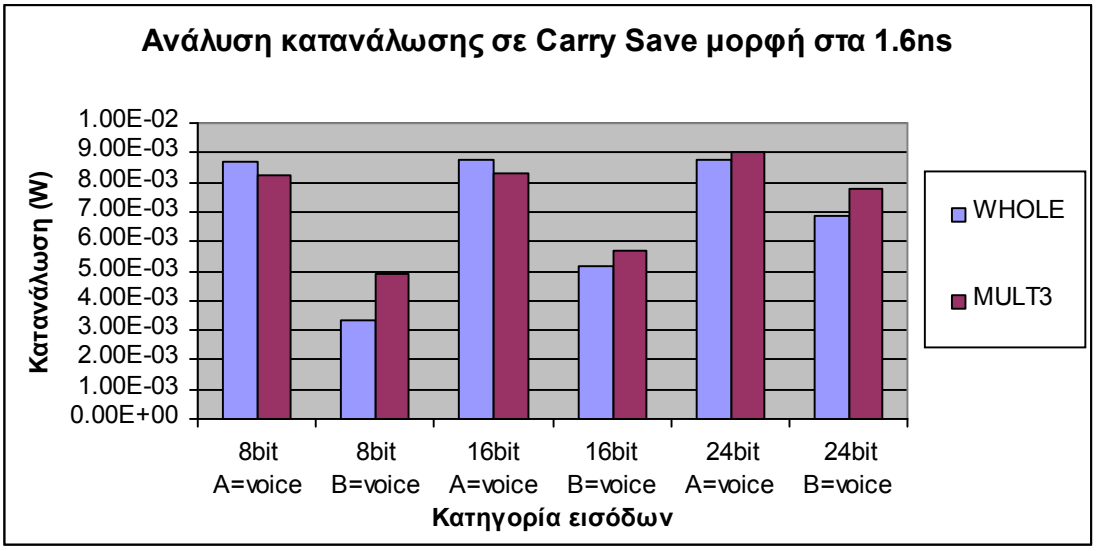
Πίνακας 3.14 Πίνακας επιφάνειας MULT3 σε δυαδική μορφή στα 2.1ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	42575
MULT3	52846

Έχει μετρηθεί η κατανάλωση για τις ίδιες περιπτώσεις όπως και για την πρώτη υλοποίηση του 2encA. Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα.

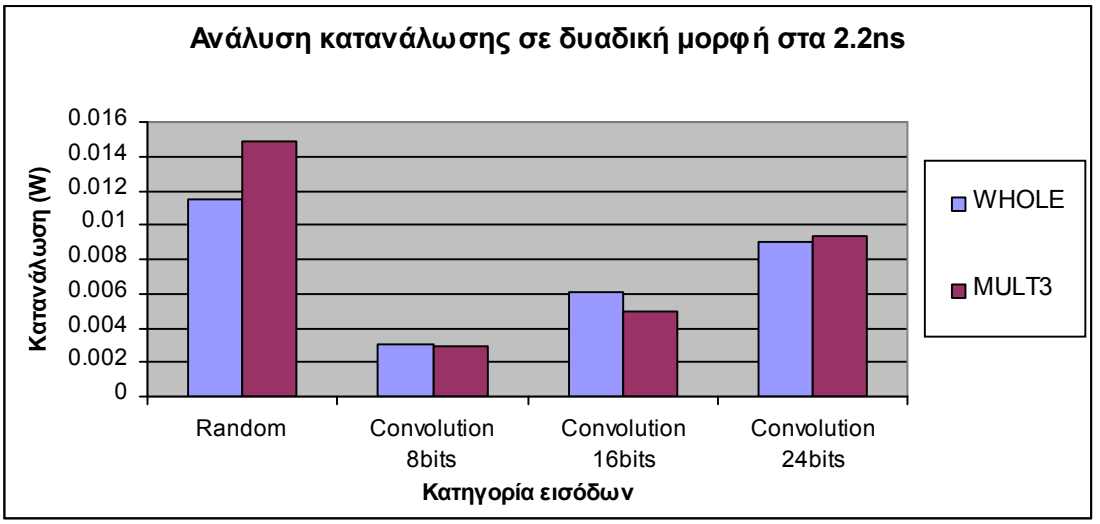


Σχήμα 3.40 Αποτελέσματα MULT3 σε CS μορφή στα 1.6ns

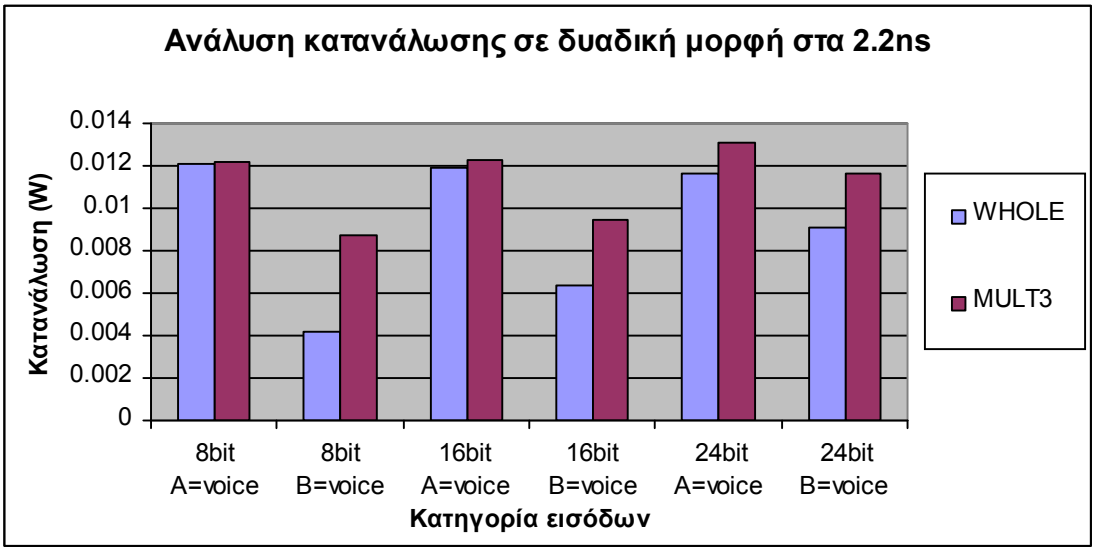


Σχήμα 3.41 Αποτελέσματα MULT3 σε CS μορφή στα 1.6ns

Αναλύοντας τα παραπάνω αποτελέσματα βλέπουμε πως για τυχαίους δυαδικούς αριθμούς ο αλγόριθμος MULT3 παρουσιάζει μεγαλύτερη κατανάλωση από τον πολλαπλασιαστή της DW και αυτό είναι λογικό αφού χρειάστηκε να χρησιμοποιήσουμε περισσότερο κύκλωμα για την τελική πρόσθεση wallace και τις επιπλέον κωδικοποιήσεις. Στην περίπτωση που γνωρίζουμε από πριν πιο από τα δύο σήματα θα είναι το σήμα φωνής τότε θα επιλέξουμε κάθε φορά το σήμα φωνής να εισέρχεται στην είσοδο B (B=voice) όπου και έχουμε την χαμηλότερη κατανάλωση και στους δύο αλγορίθμους. Σε αυτή την περίπτωση ο αλγόριθμος MULT3 υστερεί του κλασσικού αλγορίθμου πολλαπλασιασμού για τον ίδιο λόγο που υστερεί και στις τυχαίες τιμές. Αν πάλι δεν γνωρίζουμε από πριν πιο από τα δύο σήματα θα είναι το σήμα φωνής και το σήμα φωνής τελικά θα είναι σαν είσοδος στο A, τότε από τα αποτελέσματα φαίνεται πως ο αλγόριθμος MULT3 παρουσιάζει χαμηλότερη κατανάλωση αφού κωδικοποιεί τα 16MSB του A, ενώ ο κλασσικός αλγόριθμος πολλαπλασιασμού κωδικοποιεί μόνο το B. Το πλεονέκτημα αυτής της κωδικοποίησης του MULT3 φαίνεται ξεκάθαρα για την περίπτωση convolution και για τα τρία σήματα όπου με την κωδικοποίηση του αλγορίθμου αυτού χρησιμοποιούμε το γεγονός ότι και οι δύο είσοδοι θα έχουν στα MSB τους πολλές μονάδες ή πολλά μηδενικά άρα μπορούμε να το εκμεταλλευτούμε αυτό με κωδικοποιώντας τους με χρήση της κωδικοποίησης modified booth για να μειωθεί η κατανάλωση θέτοντας τα ως κωδικοποιημένα μηδενικά μειώνοντας έτσι το συνολικό switching activity.



Σχήμα 3.42 Αποτελέσματα MULT3 σε binary μορφή στα 2.2ns



Σχήμα 3.43 Αποτελέσματα MULT3 σε binary μορφή στα 2.2ns

Η διαφορά σε σχέση με τα αποτελέσματα σε binary μορφή είναι ότι κατά τη μετατροπή του αποτελέσματος από Carry-Save, οι τιμές των αριστερότερων bits του τελικού πια αποτελέσματος αλλοιώνονται, λόγω της διάδοσης κρατουμένου στον 64-bit Parallel-Prefix adder, με αποτέλεσμα να αυξάνεται το switching activity των ψηφίων αυτών αλλά και γιατί ο Parallel-Prefix adder που χρησιμοποιείται έχει καλύτερα αποτελέσματα ως προς την κατανάλωση όταν δεν υπάρχουν πριν από αυτόν οι carry save adders που επιβαρύνουν κατά πολύ σε αυτή την περίπτωση την κατανάλωση, εξαιτίας απλοποιήσεων του Design Compiler που δεν μπορούν να γίνουν.

4

MSF ΑΛΓΟΡΙΘΜΟΙ

4.1 Αλγόριθμος MSF Multiplier

Ο αλγόριθμος MSF Multiplier υλοποιήθηκε για να έχει καλύτερα αποτελέσματα δηλαδή χαμηλότερη κατανάλωση όταν και οι δύο είσοδοι είναι σήματα φωνής για πιθανές επεξεργασίες σημάτων που αφορούν συνελίξεις.

Αυτός ο αλγόριθμος έχει δύο εισόδους, τις A και B, όπου και οι δύο είσοδοι A και B προορίζονται να είναι σήματα φωνής και επίσης τα A και B είναι 32 bits. Η όλη λειτουργία του αλγορίθμου προσπαθεί να ελαχιστοποιήσει την κατανάλωση με χρήση κωδικοποίησης modified booth και στο A αλλά ταυτόχρονα και στο B.

4.1.1 Θεωρητική τεκμηρίωση αλγορίθμου MSF Multiplier

Έστω ότι έχουμε τους αριθμούς X και Y σε μορφή συμπληρώματος ως προς δύο. Χάριν απλούστευσης θεωρούμε ότι οι δύο αυτοί αριθμοί είναι 8 bit ο καθένας και βρίσκονται από τις παρακάτω σχέσεις:

$$X = \overline{x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0} = -2^7 x_7 + \sum_{i=0}^{6} 2^i x_i \quad (1)$$

$$Y = \overline{y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0} = -2^7 y_7 + \sum_{i=0}^{6} 2^i y_i \quad (2)$$

Οι αριθμοί X και Y μπορούν να κωδικοποιηθούν και να γραφτούν σε μορφή Modified Booth:

$$X = \overline{\mathbf{x}_{76}\mathbf{x}_{54}\mathbf{x}_{32}\mathbf{x}_{10}} = \sum_{k=0}^{3} 2^{2k}\mathbf{x}_{k+1,k} \tag{3}$$

$$Y = \overline{\mathbf{y}_{76}\mathbf{y}_{54}\mathbf{y}_{32}\mathbf{y}_{10}} = \sum_{k=0}^{3} 2^{2k}\mathbf{y}_{k+1,k} \tag{4}$$

Υπενθυμίζεται πως τα ψηφία $\mathbf{y}_k \in \{-2, -1, 0, 1, 2\}$ αντιστοιχούν στα τρία συνεχόμενα bits x_{2k+1}, x_{2k} και x_{2k-1}, με ένα κοινό bit όπως αναλύθηκε στο κεφάλαιο 2.

Τα κωδικοποιημένα Modified-Booth ψηφία $\boldsymbol{x}_{k+1,k}$ δίνονται επίσης και από την παρακάτω εξίσωση:

$$\boldsymbol{x}_{k+1,k} = -2x_{2k+1} + x_{2k} + x_{2k-1} \tag{5}$$

Ο αριθμός X μπορεί να γραφτεί σε μια υβριδική μορφή ως Modified-Booth και συμπληρώματος ως προς δύο:

$$X = 2^6(-2x_7 + x_6 + x_5) - 2^5 x_5 + \sum_{i=0}^{4} 2^i x_i = 2^6\mathbf{x}_{7,6} + \overline{x_5x_4x_3x_2x_1x_0} = 2^6\mathbf{x}_{7,6} + X_6 \tag{6}$$

Όπου $\mathbf{x}_{7,6} = -2x_7 + x_6 + x_5$ και $X_6 = \overline{x_5x_4x_3x_2x_1x_0} = -2x_5 + \sum_{i=0}^{4} 2^i x_i$ (7)

Ο αριθμός Y μπορεί επίσης να γραφτεί με παρόμοιο τρόπο:

$$Y = 2^6\mathbf{y}_{7,6} + \overline{y_5y_4y_3y_2y_1y_0} = 2^6\mathbf{y}_{7,6} + Y_6 \tag{8}$$

Όπου $\mathbf{y}_{7,6} = -2y_7 + y_6 + y_5$ και $Y_6 = \overline{y_5y_4y_3y_2y_1y_0} = -2y_5 + \sum_{i=0}^{4} 2^i y_i$ (9)

Το γινόμενο $X{\cdot}Y$ δίνεται από τις παρακάτω σχέσεις:

$$\begin{aligned} X \cdot Y &= 2^6(\mathbf{x}_{7,6} + X_6)\cdot(2^6\mathbf{x}_{7,6} + Y_6) = 2^{12}\mathbf{x}_{7,6}\mathbf{y}_{7,6} + 2^6\mathbf{x}_{7,6}Y_6 + 2^6\mathbf{y}_{7,6}X_6 + X_6 \cdot Y_6 \\ X \cdot Y &= 2^6\mathbf{x}_{7,6}(2^6\mathbf{y}_{7,6} + Y_6) + 2^6\mathbf{y}_{7,6}X_6 + X_6 \cdot Y_6 = 2^6\mathbf{x}_{7,6}Y + 2^6\mathbf{y}_{7,6}X_6 + X_6 \cdot Y_6 \\ X \cdot Y &= 2^6 Z_6 + X_6 \cdot Y_6 \text{ όπου } Z_6 = \mathbf{x}_{7,6}Y_6 + \mathbf{y}_{7,6}X \end{aligned} \tag{10}$$

Παρομοίως ο όρος $X_6 \cdot Y_6$ με βάση την Εξ. 10 μπορεί να εκφραστεί ως:

$$X_6 \cdot Y_6 = 2^4\mathbf{x}_{5,4}Y_6 + 2^4\mathbf{y}_{5,4}X_4 + X_4 \cdot Y_4 = 2^4 Z_4 + X_4 \cdot Y_4 \tag{11}$$

Όπου $\mathbf{x}_{5,4} = -2x_5 + x_4 + x_3$, $X_4 = \overline{x_3 x_2 x_1 x_0} = -2^3 x_3 + \sum_{i=0}^{2} 2^i x_i$, $\mathbf{y}_{5,4} = -2y_5 + y_4 + y_3$ και

$Y_4 = \overline{y_3 y_2 y_1 y_0} = -2^3 y_3 + \sum_{i=0}^{2} 2^i y_i$

Επομένως:

$$X_4 \cdot Y_4 = 2^2 \mathbf{x}_{3,2} Y_4 + 2^2 \mathbf{y}_{3,2} X_2 + X_2 \cdot Y_2 = 2^2 Z_2 + X_2 \cdot Y_2 \tag{12}$$

Ο τελευταίος όρος $Z_0 = X_2 \cdot Y_2 \in \{-2, -1, 0, 1, 2, 4\}$ μπορεί να υπολογιστεί σε μορφή συμπληρώματος ως προς δύο κατευθείαν από τα bits x_1, x_0, y_1, y_0.

$$Z_0 = X_2 \cdot Y_2 = \overline{x_1 x_0} \cdot \overline{y_1 y_0} = (-2x_1 + x_0) \cdot (-2y_1 + y_0) = p_2 \overline{p_1} p_0 \tag{13}$$

Τα βάρη των p_0, p_2 είναι θετικά ενώ το βάρος του p_1 είναι αρνητικό.

Πίνακας 4.1 Πίνακας τιμών p_0, p_2, $-p_1$

$+p_2$	$-p_1$	$+p_0$	Τιμή
0	0	0	0
0	0	1	1
0	1	0	-2
0	1	1	-1
1	0	0	4
1	1	0	2

Έτσι τα bits αυτά δίνονται από τις επόμενες σχέσεις:

$$\begin{aligned} p_0 &= x_0 \cdot y_0 \\ p_1 &= x_0 \cdot y_1 (x_1 \cdot y_0)' + x_1 \cdot y_0 (x_0 \cdot y_1)' = (x_0 \cdot y_1) \oplus (x_1 \cdot y_0) \\ p_2 &= x_1 \cdot y_1 (x_0 \cdot y_0)' \end{aligned} \tag{14}$$

Τελικά έχουμε:

$$X \cdot Y = 2^6 Z_6 + 2^4 Z_4 + 2^2 Z_2 + 2^0 Z_0 \tag{15}$$

$$X \cdot Y = 2^6 (\mathbf{x}_{7,6} Y + \mathbf{y}_{7,6} X_6) + 2^4 (\mathbf{x}_{5,4} Y_6 + \mathbf{y}_{5,4} X_4) + 2^2 (\mathbf{x}_{3,2} Y_4 + \mathbf{y}_{3,2} X_2) + 2^2 p_2 - 2p_1 + p_0$$

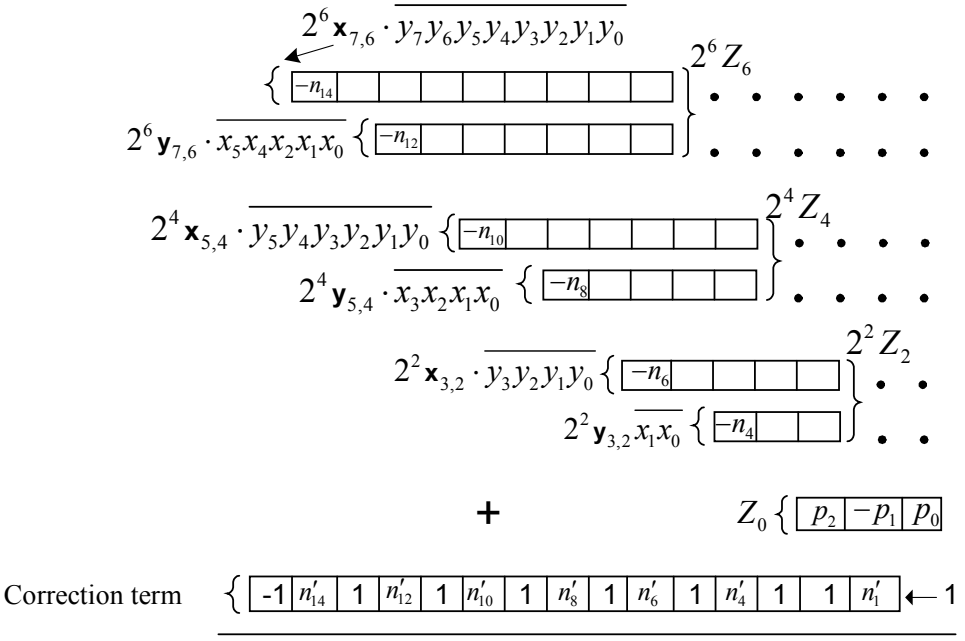
Ισοδύναμα:

$$X \cdot Y = 2^6 Z_6^* + 2^4 Z_4^* + 2^2 Z_2^* + 2^0 Z_0^* \tag{16}$$

$$X \cdot Y = 2^6(\mathbf{x}_{7,6} Y_6 + \mathbf{y}_{7,6} X) + 2^4(\mathbf{x}_{5,4} Y_4 + \mathbf{y}_{5,4} X_6) + 2^2(\mathbf{x}_{3,2} Y_2 + \mathbf{y}_{3,2} X_4) + 2^2 p_2 - 2p_1 + p_0$$

Όπου $Z_6^* = \mathbf{x}_{7,6} Y_6 + \mathbf{y}_{7,6} X$, $Z_4^* = \mathbf{x}_{5,4} Y_4 + \mathbf{y}_{5,4} X_6$, $Z_2^* = \mathbf{x}_{3,2} Y_2 + \mathbf{y}_{3,2} X_4$ και $Z_0^* = X_2 \cdot Y_2$

Από αυτή την ανάλυση λαμβάνουμε το σχήμα 4.1 για X και Y 8-bit αριθμούς για τον αλγόριθμο MSF.



Σχήμα 4.1 Δομή μερικών γινομένων MSF και διορθωτικοί όροι

4.1.2 Είσοδοι και έξοδοι επιμέρους μονάδων και ανάλυση αυτών

Το όλο κύκλωμα του MSF έχει σαν εισόδους δύο αριθμούς 32-bit σε μορφή συμπληρώματος ως προς δύο και σαν έξοδο έχει, είτε έναν αριθμό σε μορφή αθροίσματος-κρατουμένου 64-bit ή έναν αριθμό σε μορφή συμπληρώματος ως προς δύο που είναι το άθροισμα του παραπάνω αριθμού που βρίσκεται σε carry save μορφή.

Encoder

Για να κωδικοποιηθούν τα ψηφία του A και του B σε modified booth ψηφία χρησιμοποιείται μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τρία σήματα για την κάθε τριάδα εκτός από την κάθε πρώτη τριάδα (A_1A_00 και B_1B_00) αφού όπως φαίνεται από το Σχήμα 4.1 δεν χρειάζεται. Το κύκλωμα είναι κοινό και για την μετατροπή του A σε ψηφία modified-booth αλλά και για την μετατροπή του B σε ψηφία modified-booth αφού κωδικοποιούνται οι ίδιες τριάδες ψηφίων (συγκεκριμένα κωδικοποιούνται σε κάθε ένα από τα δύο κομμάτια της κωδικοποίησης 15 τριάδες σε 3·15=45 σήματα.

Τα τρία αυτά σήματα είναι τα: m, x, και x2.

Το x2 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +2 ή −2, το οποίο σημαίνει ότι το μερικό γινόμενο που πρέπει να προστεθεί είναι ή το B ολισθημένο κατά μία θέση ή το συμπλήρωμα αυτού. Σε κάθε άλλη περίπτωση το x2 είναι 0.

Το x1 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +1 ή −1, το οποίο σημαίνει ότι το μερικό γινόμενο που πρέπει να προστεθεί είναι ή το B ή το συμπλήρωμα αυτού. Σε κάθε άλλη περίπτωση το x1 είναι 0.

Το m σε άλλες κωδικοποιήσεις συνήθως είναι 1 όταν και το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1, που αυτό συνεπάγεται ότι πρέπει να προστεθεί το συμπλήρωμα του B είτε ολισθημένο είτε όχι, εκτός από την περίπτωση που η τριάδα είναι το 111, οπότε δεν προστίθεται τίποτα. Επειδή όμως ο αλγόριθμος είναι κατασκευασμένος να πολλαπλασιάζει πολλούς μικρούς αριθμούς, άρα και μικρούς αρνητικούς αριθμούς, θα επιβαρυνόταν εξαιρετικά η κατανάλωση αν αντιμετωπίζαμε το 111 σαν αρνητικό μηδέν, οπότε το m είναι 1 όταν το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1 εκτός από την περίπτωση της τριάδας 111.

Στον Πίνακα 3.4 αναφέρονται εκτεταμένα οι τιμές των προαναφερθέντων σημάτων.

Έτσι οι πύλες που χρησιμοποιήθηκαν για να υλοποιηθούν αυτά τα σήματα είναι οι παρακάτω:

$$m = A_{2i+1} \text{ xor } (A_{2i+1} \text{ and } A_{2i} \text{ and } A_{2i-1})$$

$$x2 = (\text{not}(A_{2i+1}) \text{ and } A_{2i} \text{ and } A_{2i-1}) \text{ or } (A_{2i+1} \text{ and not}(A_{2i}) \text{ and not}(A_{2i-1}))$$

$$x = (A_{2i} \text{ xor } A_{2i-1})$$

και

$$m = B_{2i+1} \text{ xor } (B_{2i+1} \text{ and } B_{2i} \text{ and } B_{2i-1})$$

$$x2 = (\text{not}(B_{2i+1}) \text{ and } B_{2i} \text{ and } B_{2i-1}) \text{ or } (B_{2i+1} \text{ and not}(B_{2i}) \text{ and not}(B_{2i-1}))$$

$$x = (B_{2i} \text{ xor } B_{2i-1})$$

Και επίσης οι παραπάνω πύλες φαίνονται σχηματικά για την πρώτη περίπτωση και στο Σχήμα 3.12. Ακριβώς οι ίδιες πύλες χρησιμοποιήθηκαν και για την μετατροπή του B σε modified booth ψηφία.

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από την κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων λαμβάνει ως είσοδο 6 σήματα των 15 bit αφού έχουν κωδικοποιηθεί 15 τριάδες του A και του B (τα τρία σήματα για την κωδικοποίηση του A είναι τα: m, x, και x2 και τα τρία σήματα για την κωδικοποίηση του A είναι τα: m_b, x_b, και x2_b), άλλο 1 σήμα 32-bit που έχει την τιμή του B και τέλος 1 σήμα 30-bit που έχει τα 30 LSB του A.

Η μονάδα αυτή χρησιμοποιεί τα παραπάνω σήματα ώστε στις εξόδους της να γίνουν διαθέσιμα τα μερικά γινόμενα. Θεωρείται ότι τα r_i είναι τα μερικά γινόμενα που δημιουργούνται με βάση τα modified booth ψηφία του A και τα δυαδικά ψηφία του B ενώ r_i_b είναι τα μερικά γινόμενα που δημιουργούνται με βάση τα modified booth ψηφία του B και τα δυαδικά ψηφία του A. Ως όμοια μερικά γινόμενα θα αναφέρονται τα μερικά γινόμενα r_i και r_k με $i \neq k$ και παρομοίως τα r_i_b και r_k_b πάλι με $i \neq k$.

Τα όμοια μερικά γινόμενα ξεκινάνε από 5 ψηφία και σε κάθε επόμενο μερικό γινόμενο προσθέτονται άλλα δύο ψηφία για την περίπτωση των r_i και τα μερικά γινόμενα τύπου r_i_b ξεκινάνε από 3 ψηφία.

Τα μερικά γινόμενα έχουν την μορφή που φαίνεται στο σχήμα 4.2 δηλαδή για παράδειγμα τα δεύτερα μερικά γινόμενα ξεκινώντας την μέτρηση από τα LSB είναι:

$r_2 = 2^4 A_{5.4} \cdot B_5 B_4 B_3 B_2 B_1 B_0$, το οποίο είναι 7 bits

$r_2\_b = 2^4 B_{5.4} \cdot A_3 A_2 A_1 A_0$, το οποίο είναι 5 bits

Λόγω της πιθανής ολίσθησης στις περιπτώσεις όπου το x2 είναι ίσο με 1, τα μερικά γινόμενα που θα παραχθούν θα είναι ένα bit παραπάνω από το δυαδικά ψηφία που χρειάζεται το εκάστοτε μερικό γινόμενο από το A ή το B, για αυτό στο παραπάνω r_2 ενώ χρειάζονται μόνο 6 δυαδικά ψηφία σε μορφή συμπληρώματος ως προς δύο το r_2 είναι τελικά 7 bits για να μπορεί να συμπεριλάβει και την τυχούσα ολίσθηση. Η διεργασία απόδοσης τιμής στα μερικά γινόμενα χωρίζεται σε 3 μέρη. Το πρώτο σκέλος είναι οι πύλες για το πρώτο bit του μερικού γινομένου, το δεύτερο σκέλος είναι οι πύλες για όλα τα υπόλοιπα bit εκτός από το τελευταίο, και το τρίτο σκέλος είναι οι πύλες που θα χρησιμοποιηθούν για το τελευταίο bit.

Στο σχήμα 3.13 φαίνονται οι πύλες που χρησιμοποιήθηκαν με την μόνη διαφορά ότι το τελευταίο bit δεν είναι πάντα το 33ο. Το τελευταίο bit κάθε μερικού γινομένου έχει πάντα αρνητική αξία. Προφανώς οι παραπάνω πύλες εφαρμόζονται κοινά σε κάθε κωδικοποιημένη τριάδα του A και B (πλέον ως m, x1, x2).

Τα τρία ανεξάρτητα bits που προστίθενται στα τρία LSB του δέντρου που σχηματίζεται υπολογίζονται και αυτά σε αυτή την μονάδα με τις πύλες που παρουσιάζονται στις παρακάτω σχέσεις:

p0 = B(0) and A(0)

p1 = not((B(0) and A(1)) xor (B(1) and A(0)))

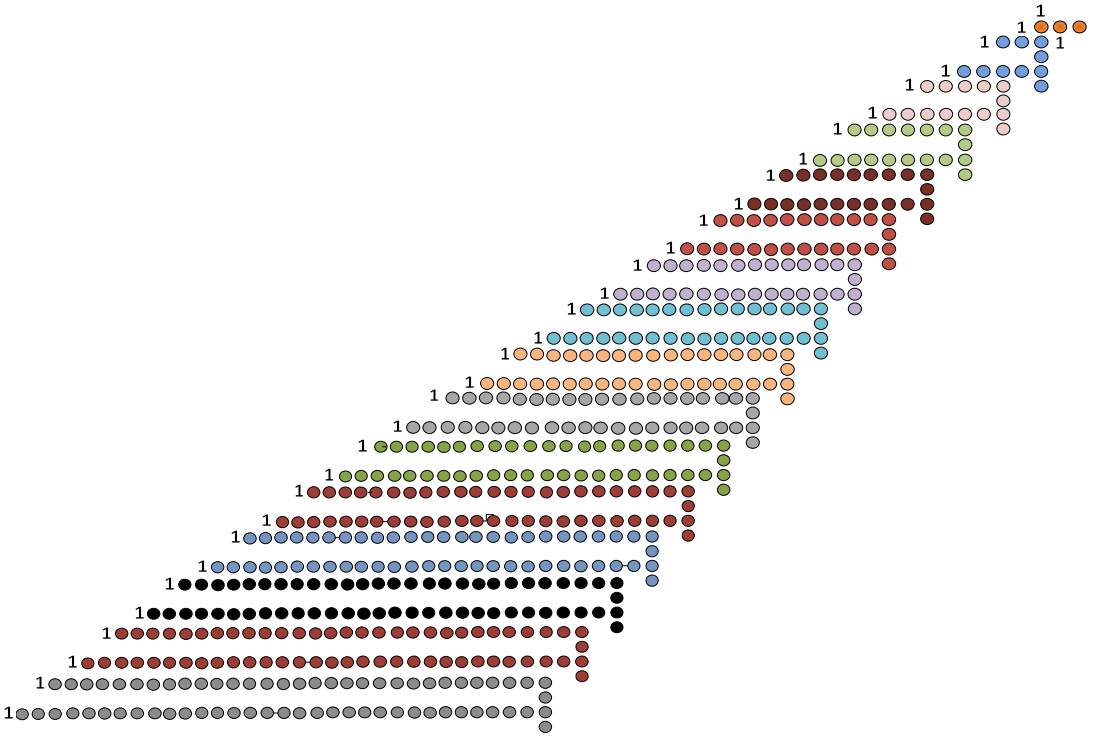
p2 = ((B(1) and A(1)) and (not(B(0) and A(0))))

Έτσι με αυτές τις πύλες η μονάδα παραγωγής μερικών γινομένων βγάζει στην έξοδο της όλους τους αριθμούς που πρέπει να προστεθούν για να βγει σωστό αποτέλεσμα εκτός από τις διορθωτικές μονάδες και τα κρατούμενα.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Αυτή η μονάδα παίρνει ως είσοδο όλες τις εξόδους από την μονάδα παραγωγής μερικών γινομένων και μαζί με τα κρατούμενα και τους διορθωτικούς όρους τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα, τα κρατούμενα και οι διορθωτικές μονάδες παρουσιάζονται ενδεικτικά (καθώς δεν έχει νόημα να σχεδιάσουμε κάθε ένα μερικό γινόμενο) στο Σχήμα 4.2 στην σωστή διάταξη που πρέπει να προστεθούν



Σχήμα 4.2 Μερικά γινόμενα, διορθωτικοί όροι και κρατούμενα MSF

Στην περίπτωση που το m είναι ίσο με 1, το κύκλωμα της μονάδας παραγωγής μερικών γινομένων χρησιμοποιώντας την πύλη XOR και έχοντας ως μία από τις δύο εισόδους της το m, βρίσκει το συμπλήρωμα ως προς 1 του αριθμού και το ολισθαίνει όταν αυτό χρειάζεται. Στο αποτέλεσμα όμως πρέπει να προστεθούν τα συμπληρώματα ως προς δύο αυτών των αριθμών. Αυτό επιτυγχάνεται προσθέτοντας μία μονάδα στο LSB κάθε μερικού γινομένου όταν το σήμα m αυτής της σειράς είναι ίσο με 1. Στο Σχήμα 4.2 αυτό απεικονίζεται με μία τελεία ακριβώς κάτω από το LSB του μερικού γινομένου. Η πρώτη οριζόντια σειρά (ξεκινώντας την μέτρηση από πάνω) από bit αποτελεί τα τρία ανεξάρτητα bits, η δεύτερη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο r_0_b, η τρίτη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο r_0, και ούτω καθεξής. Κάθε μερικό γινόμενο βρίσκεται 2 θέσεις αριστερότερα από το προηγούμενο όμοιο μερικό γινόμενο του λόγω της Modified Booth κωδικοποίησης.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως το αποτέλεσμα ενός πολλαπλασιασμού 32x32 bits θα είναι 64 bits, έτσι ο αριθμός X θα είναι 64 bit. Τα μερικά γινόμενα που είναι δεν είναι 64 bits εισάγονται με μηδενικά στα αριστερά και στα δεξιά τους για να γίνεται σωστά η άθροιση. Για παράδειγμα το μερικό γινόμενο r_0_b εισάγεται στην είσοδο του DW02_tree ως:

"00000000000000000000000000000000000000000000000000000000000000"&r0_b&"00"

όπου r0_b είναι τα παραπάνω 3 bits που φαίνονται στο Σχήμα 4.2 στην δεύτερη σειρά μερικών γινομένων. Τα μηδενικά απλοποιούνται από τον design compiler και είναι σαν να μην μπαίνουν ψηφία σε αυτά τα σημεία δηλαδή δεν θα υπάρξει κανένας full adder ή half adder που να πάρει ως είσοδο το 0.

Έτσι το αποτέλεσμα του DW02_tree και όλου του αλγορίθμου MSF είναι ένας αριθμός σε carry save μορφή 64 bit, δηλαδή δύο σειρές από 64 bit εκτός αν χρησιμοποιήσουμε τον DW01_add για να τους προσθέσουμε και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

4.1.3 Αποτελέσματα υλοποίησης MSF Multiplier

Οι συγκρινόμενοι αλγόριθμοι είναι η πρώτη υλοποίηση του αλγορίθμου 2encA με την δεύτερη υλοποίηση του αλγορίθμου Whole όπου και οι δύο είναι άμεσα συγκρινόμενοι αφού και οι δύο δεν χρησιμοποιούν modules της DesignWare παρά μόνο το wallace tree.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους παρακάτω πίνακες 4.2 και 4.3:

Πίνακας 4.2 Πίνακας επιφάνειας MSF Multiplier σε Carry Save μορφή στα 1.5ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	40184
MSF Multiplier	38260

Πίνακας 4.3 Πίνακας επιφάνειας MSF Multiplier σε δυαδική μορφή στα 2.0ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
Whole	63693
MSF Multiplier	67669

Έχει μετρηθεί η κατανάλωση για τις εξής περιπτώσεις:

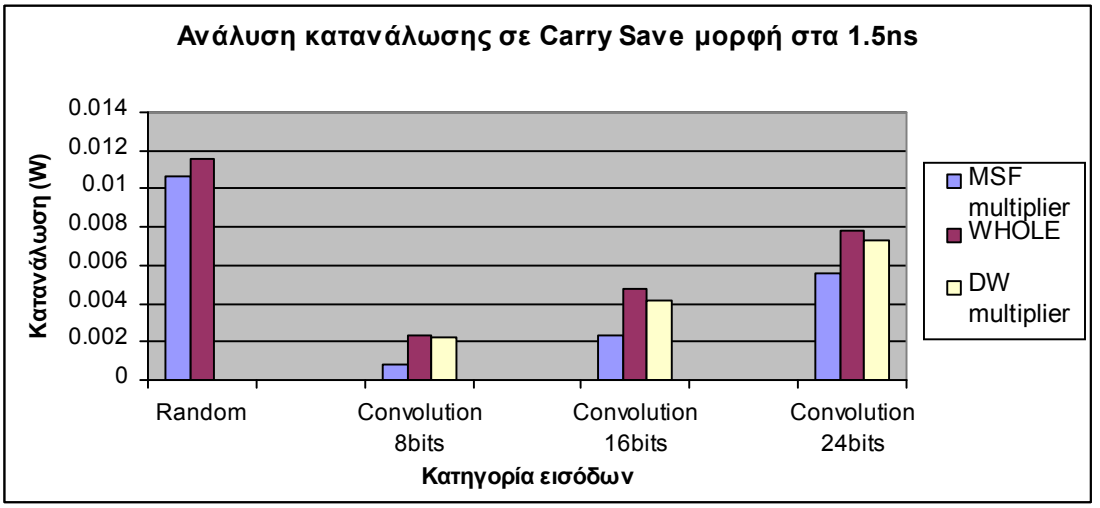
α) οι δύο είσοδοι είναι τυχαίοι δυαδικοί αριθμοί (random)

β) οι δύο είσοδοι είναι σήματα φωνής 8, 16 και 24 bits αντίστοιχα (convolution)

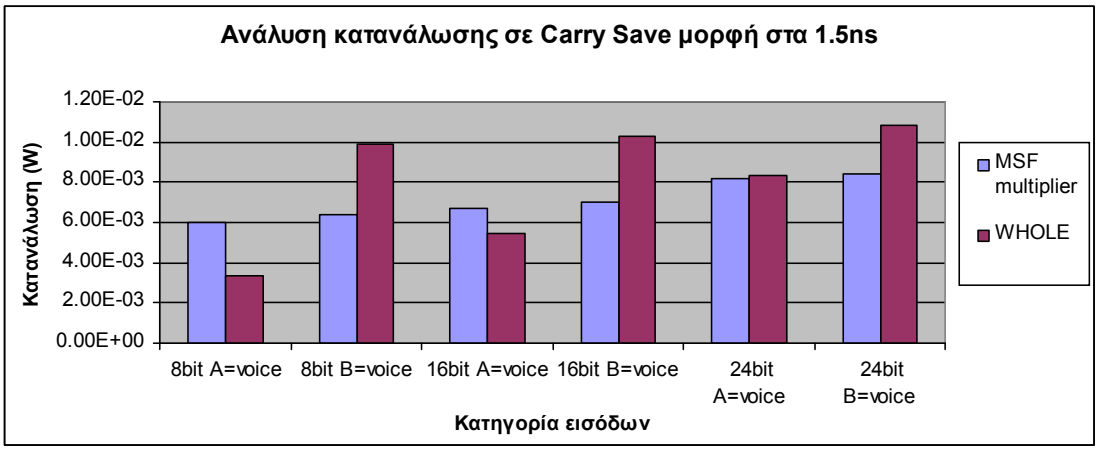
γ) η μία είσοδος (A) είναι σήμα φωνής 8, 16 και 24 bits αντίστοιχα (A=voice)

δ) η μία είσοδος (B) είναι σήμα φωνής 8, 16 και 24 bits αντίστοιχα (B=voice)

Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα με DW multiplier να τιτλοφορείται η πρώτη υλοποίηση του αλγορίθμου Whole και είναι συγκρίσιμο μόνο για τιμές όπου και οι δύο είσοδοι είναι σήματα φωνής.

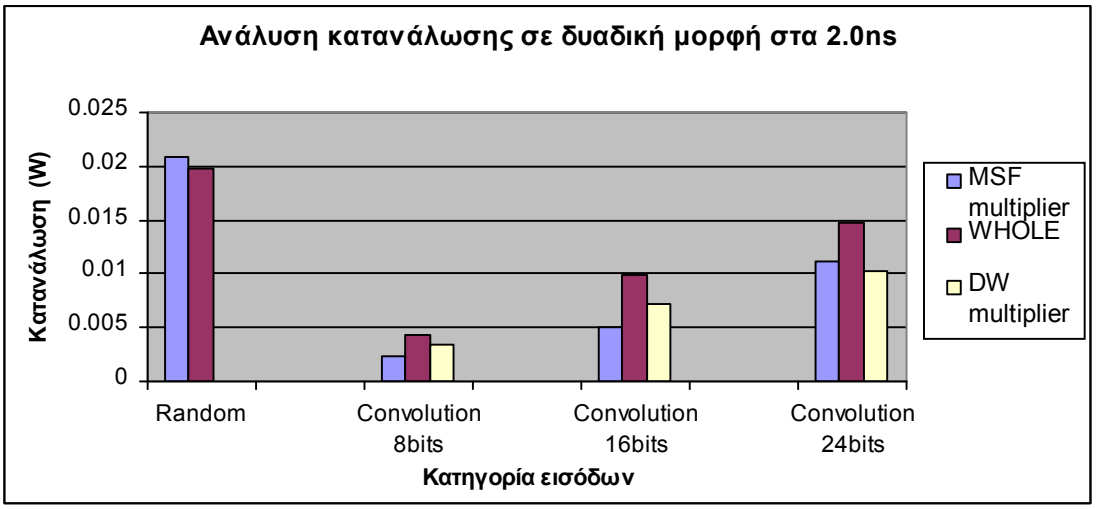


Σχήμα 4.3 Αποτελέσματα α) & β) MSF Multiplier σε CS μορφή στα 1.5ns

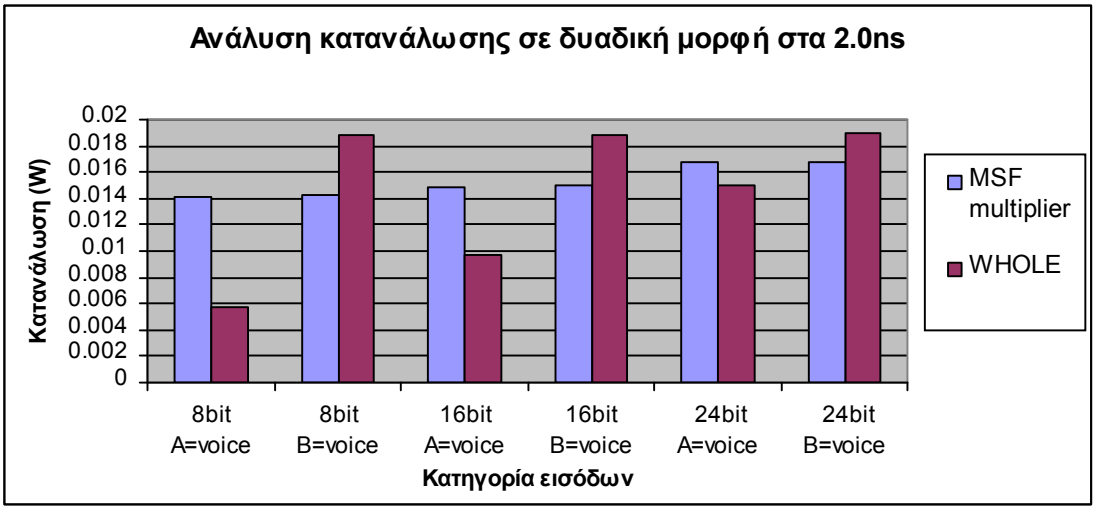


Σχήμα 4.4 Αποτελέσματα γ) & δ) MSF Multiplier σε CS μορφή στα 1.5ns

Αναλύοντας τα αποτελέσματα που φαίνονται στο Σχήμα 4.4 συμπεραίνουμε στα σήματα φωνής ο MSF Multiplier παρουσιάζει υψηλότερη κατανάλωση όταν μόνο ένα από τα δύο σήματα είναι το σήμα φωνής και ταυτόχρονα ο αλγόριθμος WHOLE κωδικοποιεί σε Modified Booth την είσοδο που αντιστοιχεί σε αυτό το σήμα. Αυτό συμβαίνει γιατί η υβριδική κωδικοποίηση των ψηφίων που έχουμε υλοποιήσει απαιτεί την κωδικοποίηση σε Modified Booth και των δύο σημάτων αλλά σε αυτές τις περιπτώσεις το δεύτερο σήμα είναι μία τυχαία αλληλουχία δυαδικών ψηφίων οπότε δεν κερδίζουμε κάτι και από την κωδικοποίηση του δεύτερου σήματος αφού τα modified booth μηδενικά που παρουσιάζει θα είναι ελάχιστα. Όταν όμως και τα δύο σήματα είναι σήματα φωνής (περίπτωση β) τότε βλέπουμε από το Σχήμα 4.3 πως έχουμε χαμηλότερη κατανάλωση για τον MSF Multiplier. Αυτό συμβαίνει διότι στα σημεία του Wallace Tree που έχουμε τις μεγαλύτερες επικαλύψεις (βλέπε Σχήμα 4.2 και 4.1), εκεί τα μερικά γινόμενα προκύπτουν με βάση τα κωδικοποιημένα MSB ψηφία των B και A, οπότε αφού αυτά τα ψηφία είναι μηδενικά για σήματα φωνής θα έχουμε μικρότερο switching activity. Για την περίπτωση random ψηφίων η χαμηλότερη κατανάλωση εξηγείται λόγω της μικρότερης επιφάνειας και προφανώς γιατί ο αλγόριθμος WHOLE δεν έχει κανένα πλεονέκτημα απέναντι στον MSF Multiplier.



Σχήμα 4.5 Αποτελέσματα α) & β) MSF Multiplier σε Binary μορφή στα 2.0ns



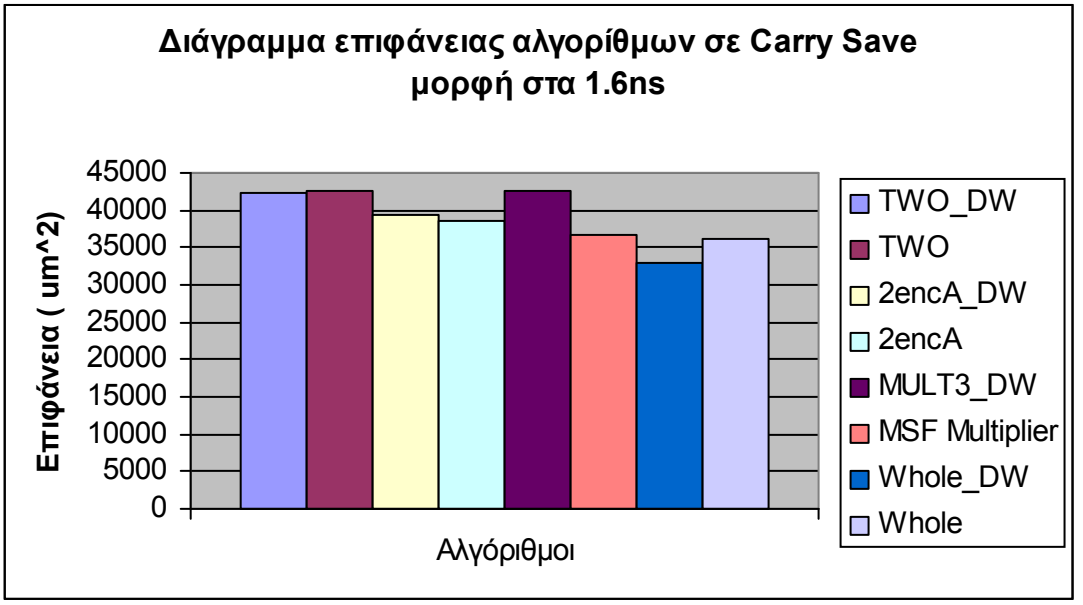
Σχήμα 4.6 Αποτελέσματα γ) & δ) MSF Multiplier σε Binary μορφή στα 2.0ns

Η διαφορά ανάμεσα στα αποτελέσματα που είναι σε carry save και σε binary μορφή οφείλεται στην μεγαλύτερη επιφάνεια του κυκλώματος που επηρεάζει τις περιπτώσεις που αναλύσαμε, αφού εφόσον υπάρχει μεγαλύτερη επιφάνεια σε σχέση με τον συγκρινόμενο αλγόριθμο. Οι απλοποιήσεις που γίνονται από τον Design Compiler με την προσθήκη του Parallel-Prefix adder είναι διαφορετικές σε κάθε ένα από τα δύο κυκλώματα με αποτέλεσμα να έχουμε μεγαλύτερη επιφάνεια στον MSF Multiplier. Το τελευταίο συμβαίνει λόγω μεγάλης συσσώρευσης ψηφίων στα χαμηλότερης αξίας bit με αποτέλεσμα την διάδοση κρατουμένου σε περισσότερα επίπεδα.

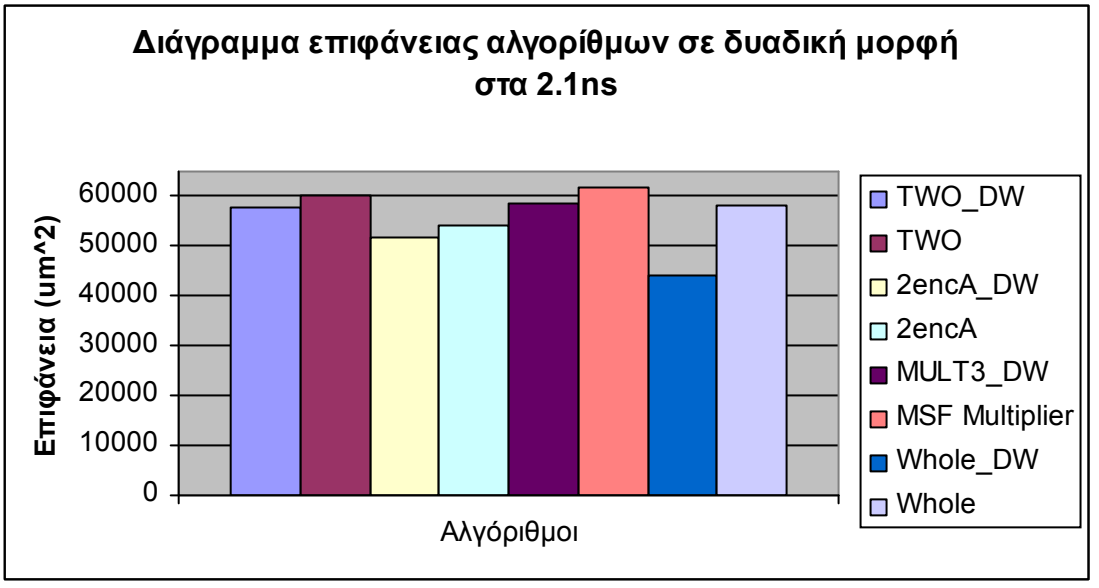
4.1.4 Συγκριτικά αποτελέσματα MSF και BLOCK πολλαπλασιαστών

Έχοντας πλέον διαθέσιμα όλα τα αποτελέσματα των πολλαπλασιαστών μπορούμε να κάνουμε μία γενική σύγκριση μεταξύ όλων των αλγορίθμων για κάθε περίπτωση. Οι μετρήσεις έγιναν στα 1.6ns για αποτέλεσμα σε carry-save μορφή και στα 2.1ns σε binary μορφή.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στα Σχήματα 4.7 και 4.8:



Σχήμα 4.7 Διάγραμμα επιφάνειας αλγορίθμων σε Carry Save μορφή στα 1.6ns



Σχήμα 4.8 Διάγραμμα επιφάνειας αλγορίθμων σε δυαδική μορφή στα 2.1ns

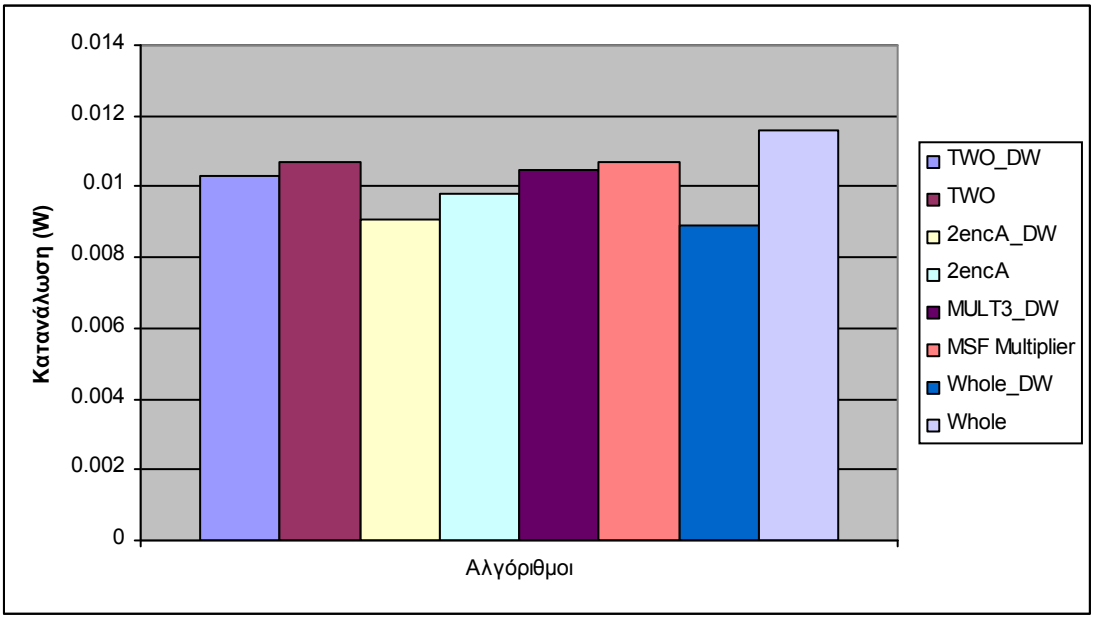
Στην συνέχεια συγκρίνεται η κατανάλωση για τις εξής περιπτώσεις:

α) οι δύο είσοδοι είναι τυχαίοι δυαδικοί αριθμοί (random)

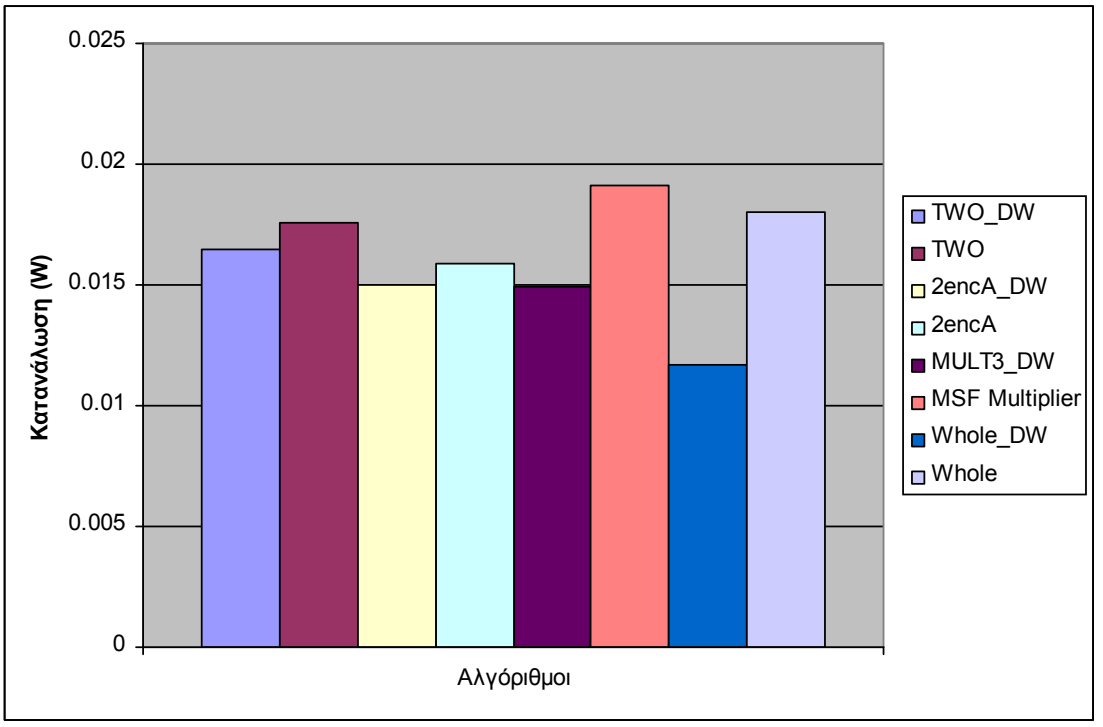
β) οι δύο είσοδοι είναι σήματα φωνής 8, 16 και 24 bits αντίστοιχα (convolution)

γ) η μία είσοδος (A) είναι σήμα φωνής 8, 16 και 24 bits αντίστοιχα (A=voice)

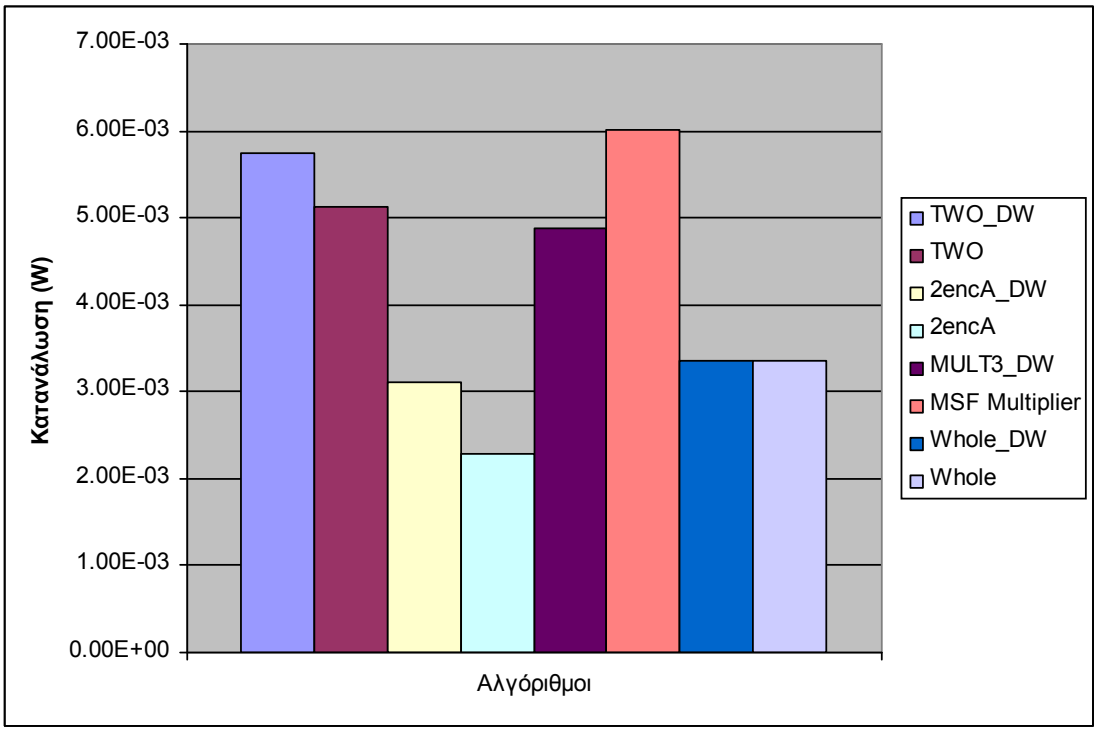
Αυτές οι περιπτώσεις εξετάζονται για carry-save στα 1.6ns & για binary μορφή στα 2.1ns.



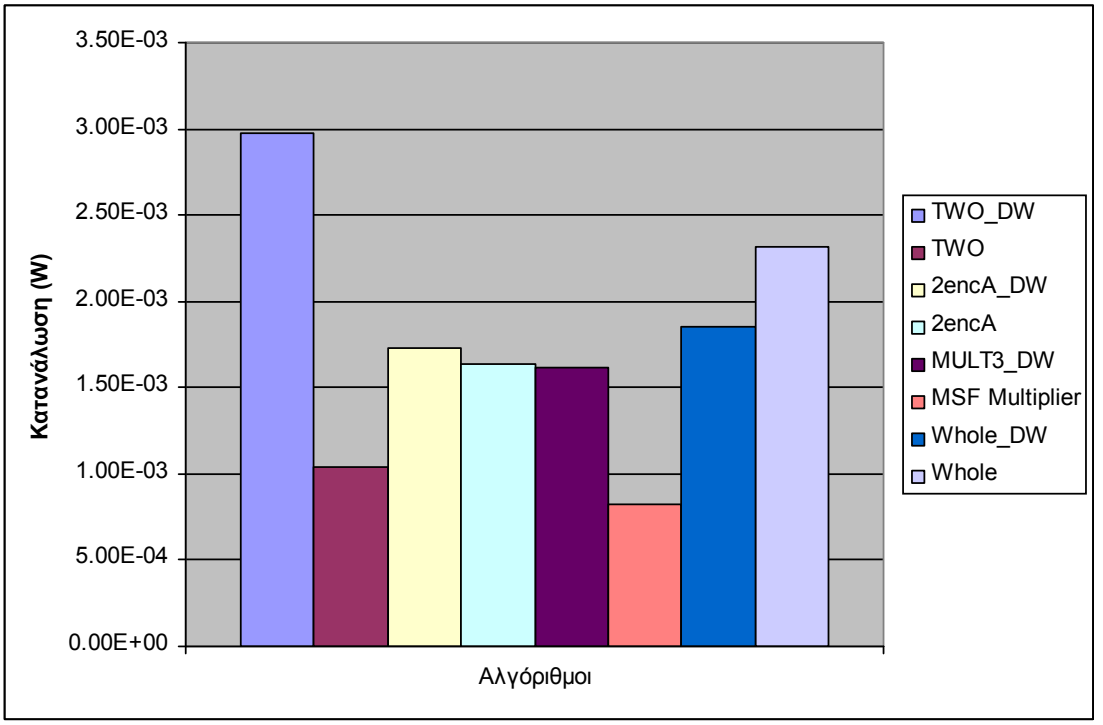
Σχήμα 4.9 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (random)



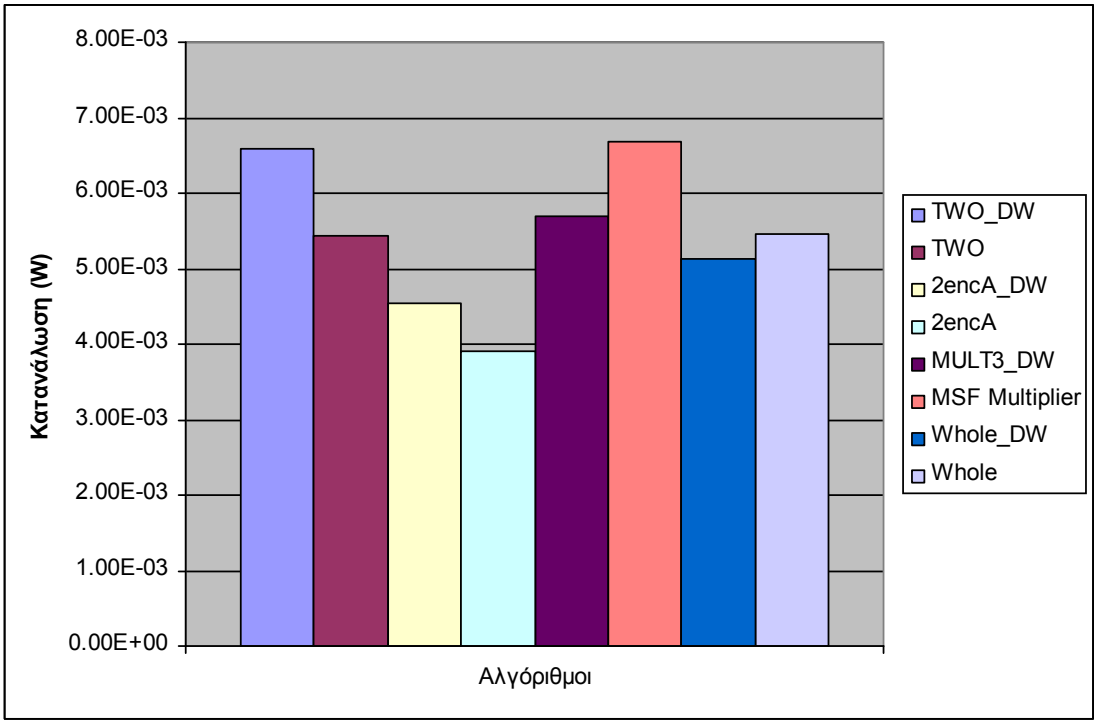
Σχήμα 4.10 Αποτελέσματα κατανάλωσης αλγορίθμων σε δυαδική μορφή (random)



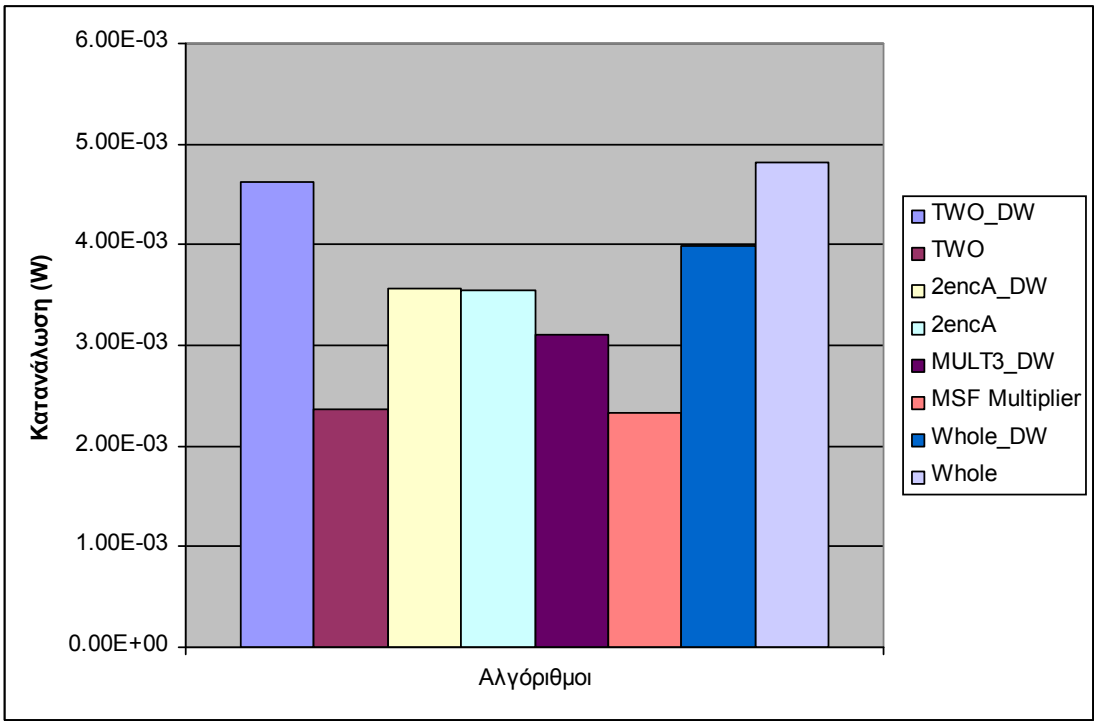
Σχήμα 4.11 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (8-bit A=voice)



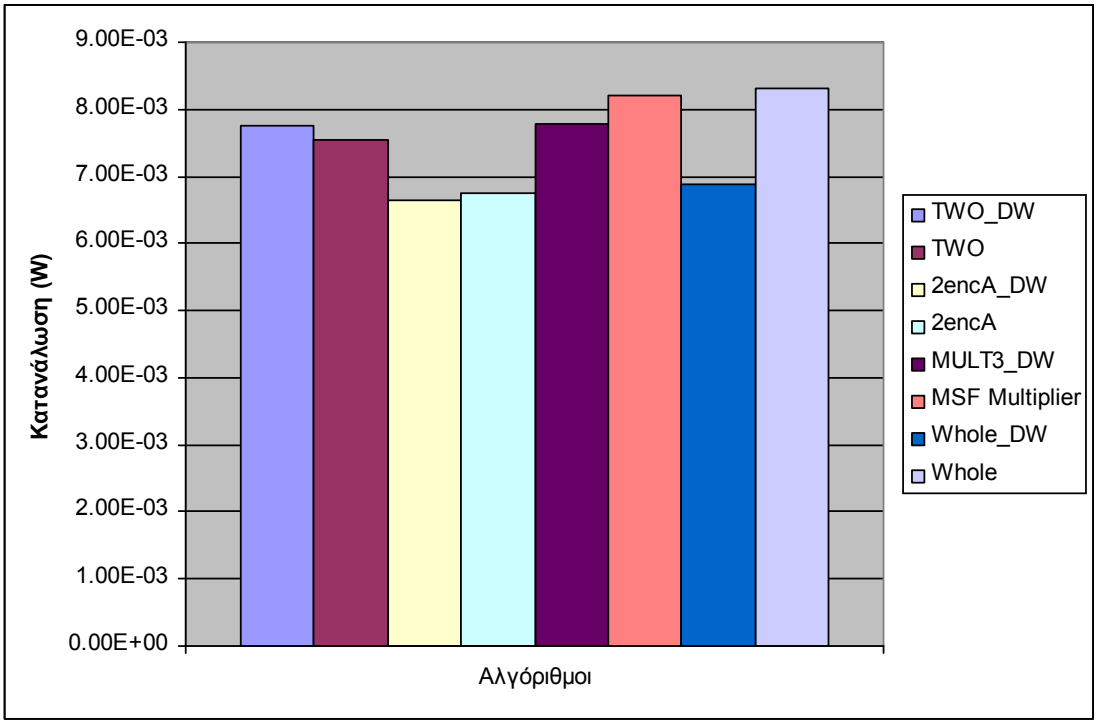
Σχήμα 4.12 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (8-bit convolution)



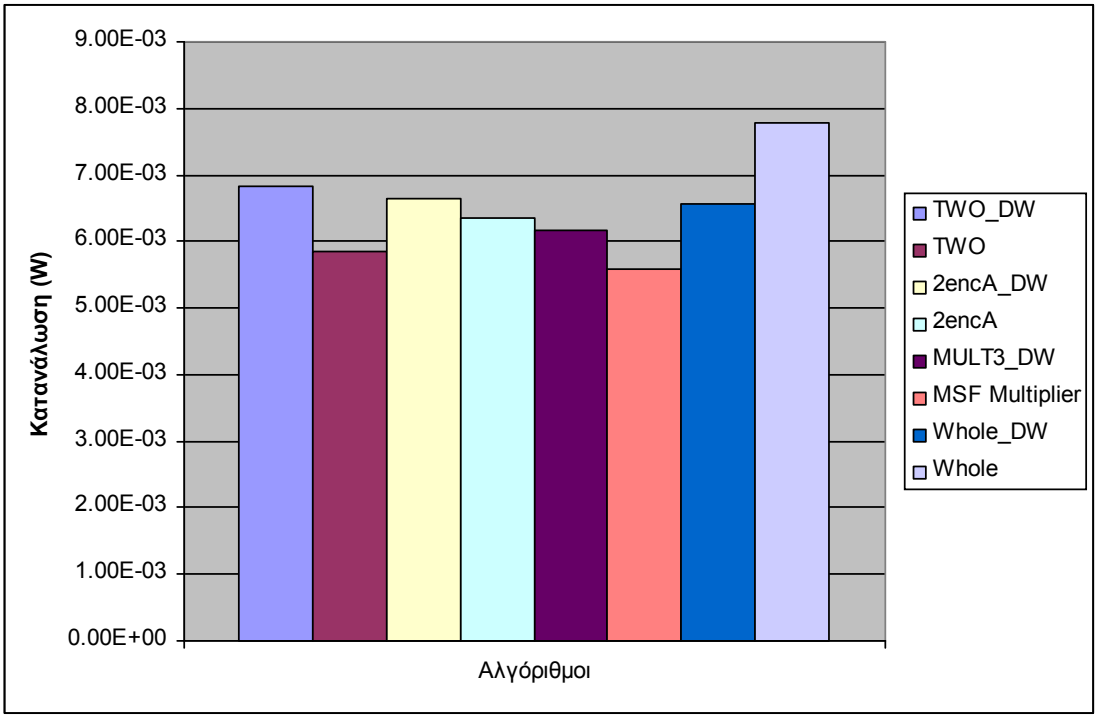
Σχήμα 4.13 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (16-bit A=voice)



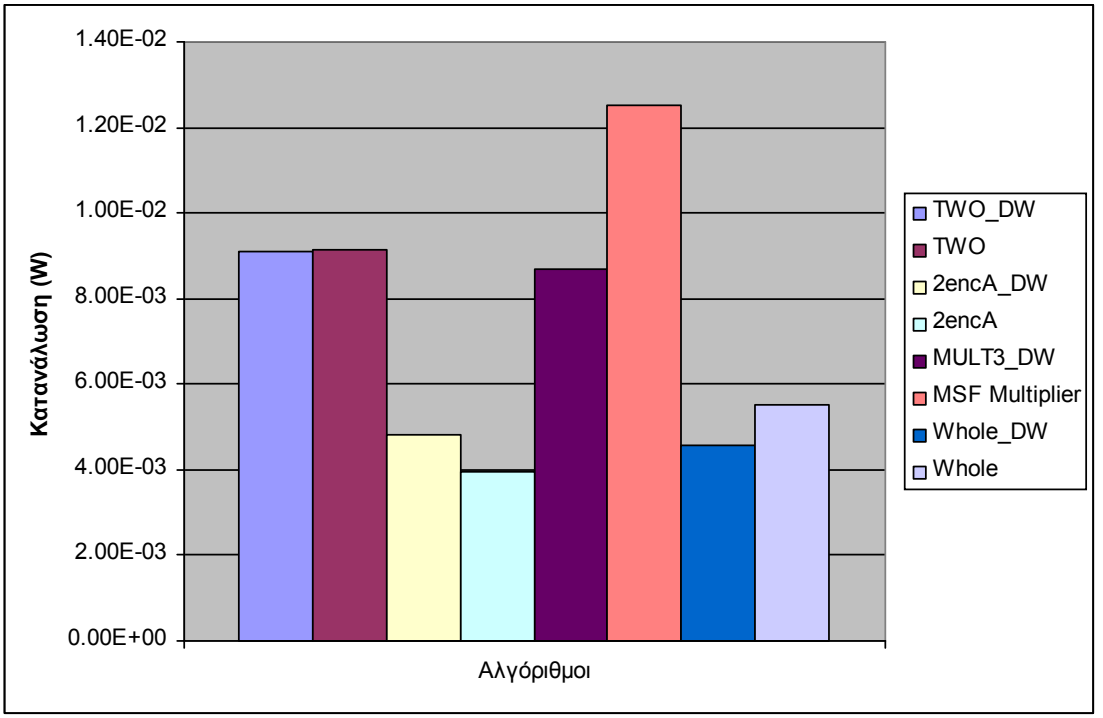
Σχήμα 4.14 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (16-bit convolution)



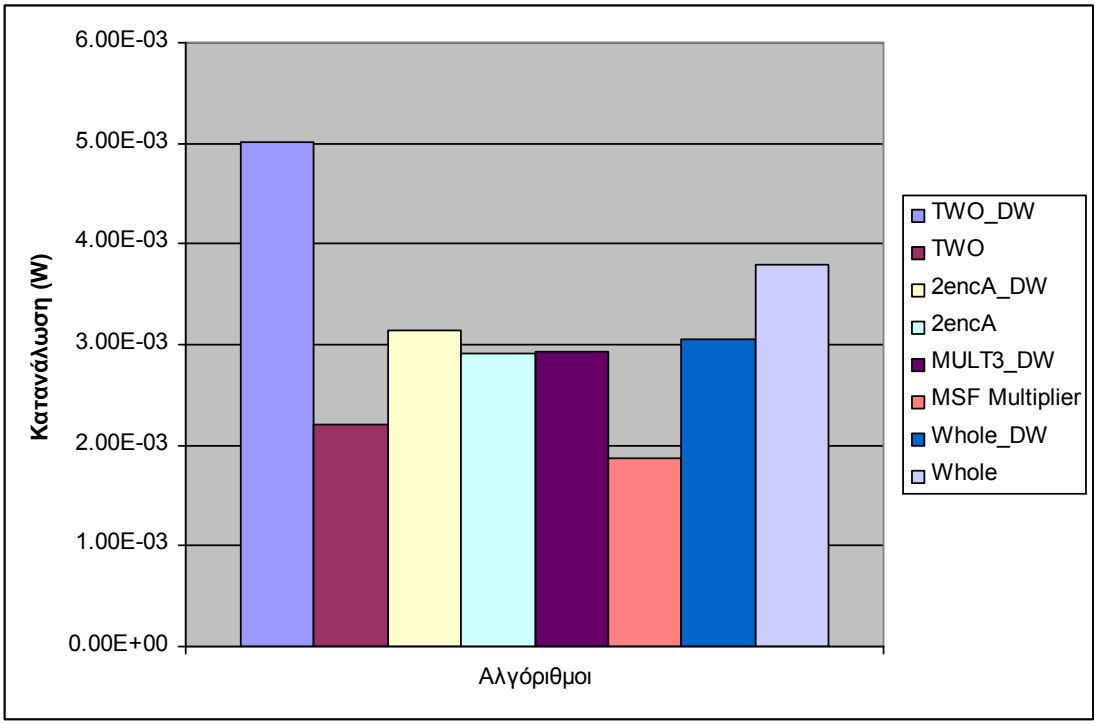
Σχήμα 4.15 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (24-bit A=voice)



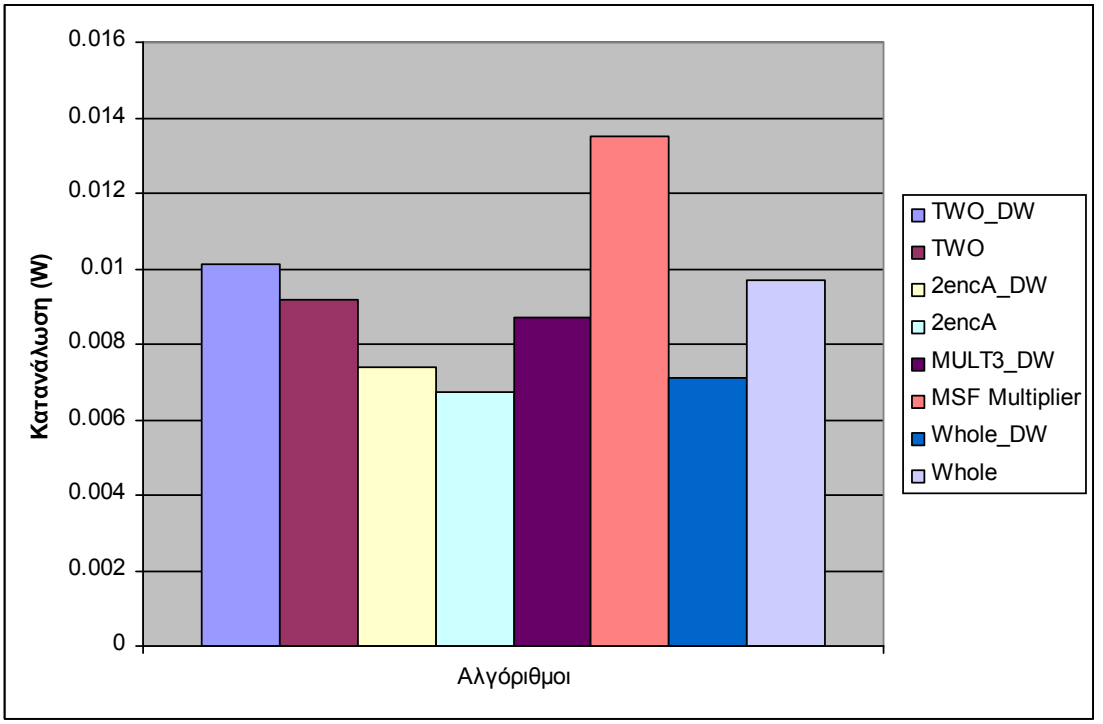
Σχήμα 4.16 Αποτελέσματα κατανάλωσης αλγορίθμων σε carry-save μορφή (24-bit convolution)



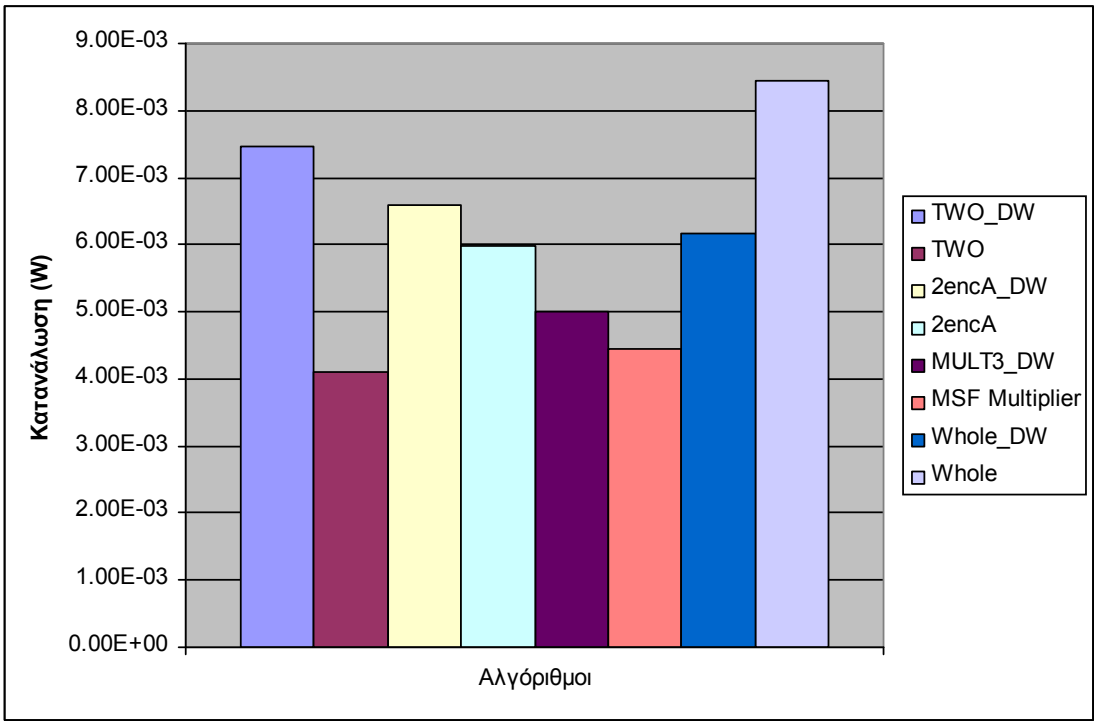
Σχήμα 4.17 Αποτελέσματα κατανάλωσης αλγορίθμων σε binary μορφή (8-bit A=voice)



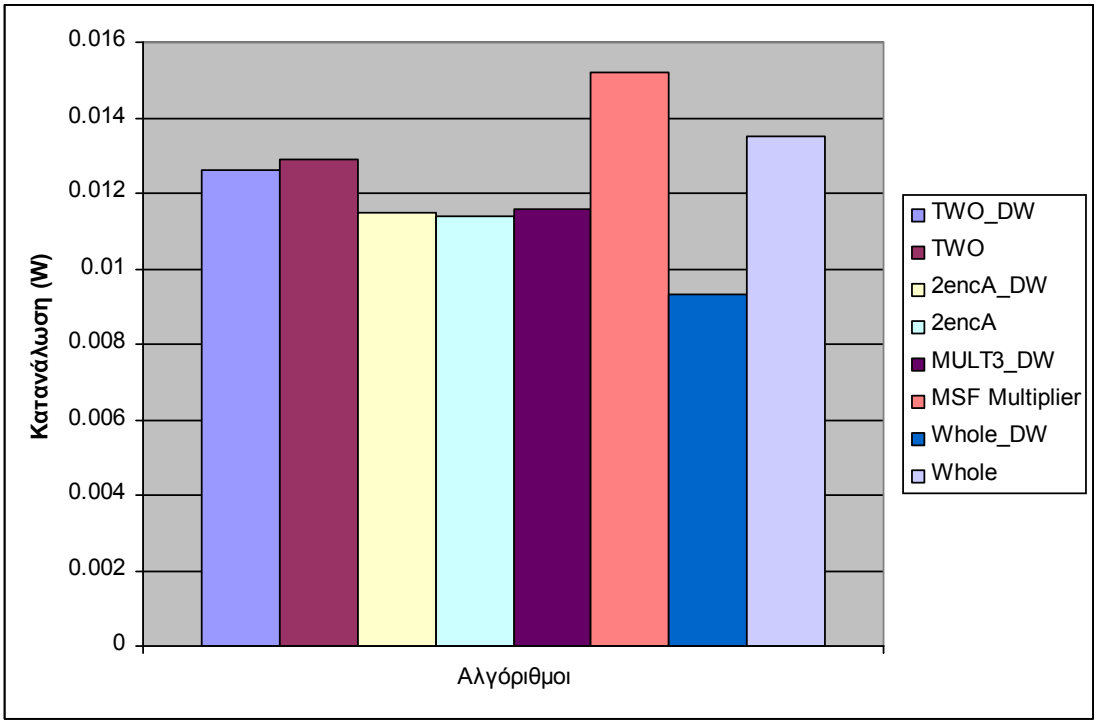
Σχήμα 4.18 Αποτελέσματα κατανάλωσης αλγορίθμων σε binary μορφή (8-bit convolution)



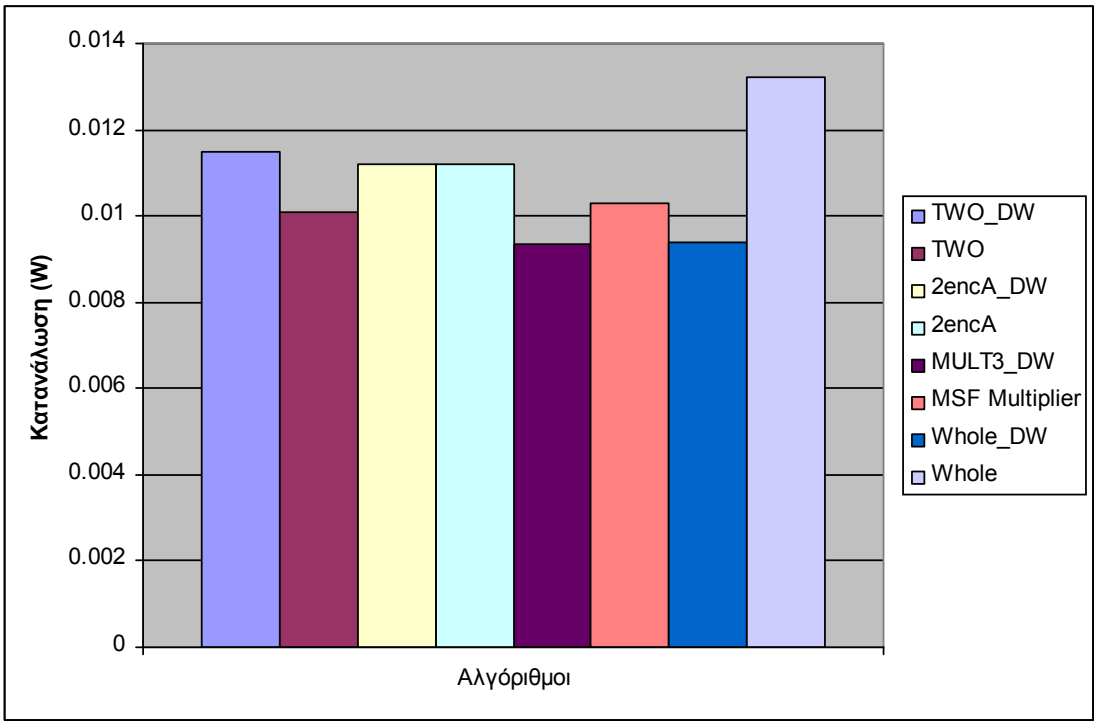
Σχήμα 4.19 Αποτελέσματα κατανάλωσης αλγορίθμων σε binary μορφή (16-bit A=voice)



Σχήμα 4.20 Αποτελέσματα κατανάλωσης αλγορίθμων σε binary μορφή (16-bit convolution)



Σχήμα 4.21 Αποτελέσματα κατανάλωσης αλγορίθμων σε binary μορφή (24-bit A=voice)



Σχήμα 4.22 Αποτελέσματα κατανάλωσης αλγορίθμων σε binary μορφή (24-bit convolution)

Στα παραπάνω σχήματα βλέπουμε συγκεντρωτικά τα αποτελέσματα όλων των πολλαπλασιαστών για κάθε περίπτωση από τις α, β και γ. Παρατηρούμε πως ο αλγόριθμος TWO στην πρώτη υλοποίηση του (με στοιχεία της DesignWare) δεν αποδίδει σε κάθε

περίπτωση καλύτερα από την δεύτερη υλοποίηση γιατί οι κωδικοποιητές του DW02_multp έχουν προφανώς φτιαχτεί για να έχουν βέλτιστα αποτελέσματα όταν κωδικοποιούν την μικρότερη από τις δύο εισόδους. Ο MSF Multiplier βλέπουμε πως έχει καλά αποτελέσματα σε περιπτώσεις που και οι δύο είσοδοι είναι σήματα φωνής ενώ υστερεί στις υπόλοιπες περιπτώσεις. Διαφορετικές επιφάνειες καθώς και διαφορετικοί κωδικοποιητές (encoders) και αποκωδικοποιητές (decoders-partial product generators) αποφέρουν διαφορετικά αποτελέσματα τα οποία συνθέτουν την παραπάνω εικόνα και έχουν εξηγηθεί για κάθε αλγόριθμο ξεχωριστά.

4.2 Αλγόριθμος MSF SQUARER

Ο αλγόριθμος MSF SQUARER υλοποιήθηκε ως ένας τετραγωνιστής που θα έχει καλύτερα αποτελέσματα δηλαδή χαμηλότερη κατανάλωση όταν και η είσοδος προέρχεται από σήματα φωνής.

Αυτός ο αλγόριθμος έχει μία προφανώς είσοδο αφού πρόκειται για τετραγωνιστή, το A. Η όλη λειτουργία του αλγορίθμου προσπαθεί να ελαχιστοποιήσει την κατανάλωση με χρήση κωδικοποίησης modified booth στο A.

4.2.1 Θεωρητική τεκμηρίωση αλγορίθμου MSF SQUARER

Χωρίς βλάβη της γενικότητας θεωρούμε πως η είσοδος είναι ένας 8-bit αριθμός X που βρίσκεται σε μορφή συμπληρώματος ως προς δύο:

$$X = \overline{x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0} = -2^7 x_7 + \sum_{i=0}^{6} 2^i x_i \qquad (1)$$

Ο αριθμός X μπορεί να γραφτεί σε Modified-Booth μορφή:

$$X = \overline{\mathbf{y}_3 \mathbf{y}_2 \mathbf{y}_1 \mathbf{y}_0} = \sum_{k=0}^{3} 2^{2k} \mathbf{y}_k \qquad (2)$$

με τα ψηφία $\mathbf{y}_k \in \{-2, -1, 0, 1, 2\}$ να αντιστοιχούν σε τρία συνεχόμενα bits x_{2k+1}, x_{2k} και x_{2k-1}, με ένα κοινό bit όπως αναλύθηκε στο κεφάλαιο 3.

Τα κωδικοποιημένα Modified-Booth ψηφία $\mathbf{y}_k$ δίνονται επίσης και από την παρακάτω εξίσωση:

$$\mathbf{y}_k = -2x_{2k+1} + x_{2k} + x_{2k+1} \qquad (3)$$

Ο αριθμός X μπορεί να γραφεί σε μια υβριδική μορφή ως Modified-Booth και συμπληρώματος ως προς δύο:

$$X = 2^6(-2x_7 + x_6 + x_5) - 2^5 x_5 + \sum_{i=0}^{4} 2^i x_i = 2^6 \mathbf{y}_3 + \overline{x_5 x_4 x_3 x_2 x_1 x_0} = 2^6 \mathbf{y}_3 + X_6 \qquad (4)$$

Όπου $\mathbf{y}_3 = -2x_7 + x_6 + x_5$ και $X_6 = \overline{x_5 x_4 x_3 x_2 x_1 x_0} = -2x_5 + \sum_{i=0}^{4} 2^i x_i$ (5)

Τετραγωνίζοντας τον αριθμό X στην μορφή που είναι στην εξίσωση (4) παίρνουμε:

$$X^2 = \left(2^6 \mathbf{y}_3 + X_6\right)^2 = 2^{12} \mathbf{y}_3^2 + 2^7 \mathbf{y}_3 X_6 + X_6^2 = 2^7 \mathbf{y}_3 \left(2^5 \mathbf{y}_3 + X_6\right) + X_6^2 \qquad (6)$$

Η ποσότητα $2^5 \mathbf{y}_3 + X_6 = 2^5\left(-2x_7 + x_6 + x_5\right) - 2^5 x_5 + \sum_{i=0}^{4} 2^i x_i = \mathbf{y}_3 \overline{x_7 x_6 x_4 x_3 x_2 x_1 x_0}$

Αντικαθιστώντας αυτή την ποσότητα στην εξίσωση (6) παίρνουμε:

$$X^2 = 2^7 \mathbf{y}_3 \overline{x_7 x_6 x_4 x_3 x_2 x_1 x_0} + X_6^2 \qquad (7)$$

Παρομοίως X_6^2 μπορεί να γραφτεί ως:

$$X_6^2 = 2^5 \mathbf{y}_2 \overline{x_5 x_4 x_2 x_1 x_0} + X_4^2 \text{ με } X_4 = \overline{x_3 x_2 x_1 x_0} \qquad (8)$$

και το X_4^2 είναι: $X_4^2 = 2^3 \mathbf{y}_1 \overline{x_3 x_2 x_0} + X_2^2$ με $X_2 = \overline{x_1 x_0}$ (9)

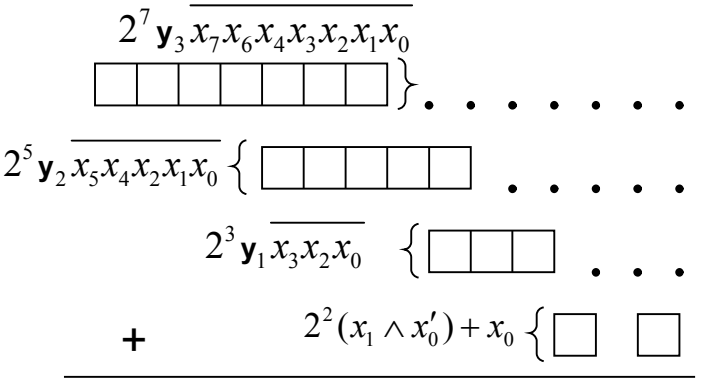
Η ποσότητα X_4^2 μπορεί να πάρει τις τιμές {0, 1, 2} και εύκολα μπορεί να αποδειχτεί πως

$$X_2^2 = \left(\overline{x_1 x_0}\right)^2 = 2^2(x_1 \wedge x_0') + x_0 \qquad (10)$$

Τελικά έχουμε:

$$X^2 = 2^7 \mathbf{y}_3 \overline{x_7 x_6 x_4 x_3 x_2 x_1 x_0} + 2^5 \mathbf{y}_2 \overline{x_5 x_4 x_2 x_1 x_0} + 2^3 \mathbf{y}_1 \overline{x_3 x_2 x_0} + 2^2(x_1 \wedge x_0') + x_0 \qquad (11)$$

Από αυτή την ανάλυση λαμβάνουμε το Σχήμα 4.23 για X 8-bit αριθμό για τον αλγόριθμο MSF SQUARER.



Σχήμα 4.23 Δομή μερικών γινομένων MSF SQUARER

4.2.2 Πρώτη υλοποίηση MSF SQUARER

Encoder

Για να κωδικοποιηθούν τα ψηφία του A σε modified booth ψηφία χρησιμοποιείται μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τρία σήματα για την κάθε τριάδα εκτός από την κάθε πρώτη τριάδα (A_1A_00) αφού όπως φαίνεται από το Σχήμα 4.23 δεν χρειάζεται. Συγκεκριμένα κωδικοποιούνται 15 τριάδες σε 3·15=45 σήματα.

Τα τρία αυτά σήματα είναι τα: m, x, και x2 και το κύκλωμα του κωδικοποιητή είναι το ίδιο με αυτό που χρησιμοποιήθηκε από τον MSF multiplier.

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από την κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων λαμβάνει ως είσοδο 3 σήματα των 15 bit αφού έχουν κωδικοποιηθεί 15 τριάδες του A (τα τρία σήματα για την κωδικοποίηση του A είναι τα: m, x, και x2) και τέλος 1 σήμα 32 bit που αντιπροσωπεύει την τιμή του A.

Η μονάδα αυτή χρησιμοποιεί τα παραπάνω σήματα ώστε στις εξόδους της να γίνουν διαθέσιμα τα μερικά γινόμενα. Θεωρείται ότι τα r_i είναι τα μερικά γινόμενα που δημιουργούνται με βάση τα modified booth ψηφία του A. Τα μερικά γινόμενα r_i ξεκινάνε από 3 ψηφία και σε κάθε επόμενο μερικό γινόμενο προσθέτονται άλλα δύο ψηφία.

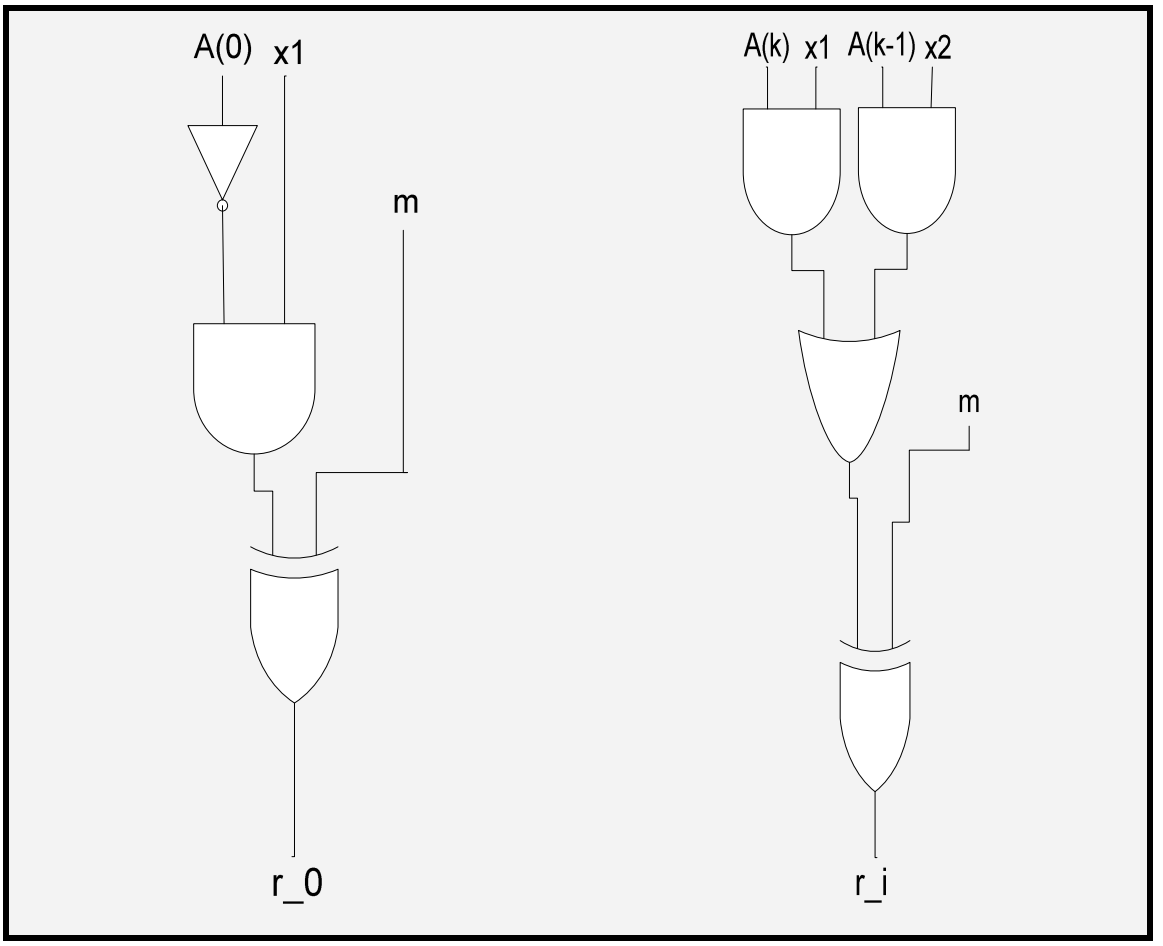
Τα μερικά γινόμενα έχουν την μορφή που φαίνεται στο σχήμα 4.23 δηλαδή για παράδειγμα το δεύτερο μερικό γινόμενο ξεκινώντας την μέτρηση από τα LSB είναι:

$$r_2 = 2^5 A_{MB,5.4,3} \cdot A_5 A_4 A_2 A_1 A_0 ,$$

το οποίο r_2 είναι 5 bits και το $A_{MB,5.4,3}$ είναι το Modified-Booth ψηφίο της κωδικοποιημένης τριάδας $A_5A_4A_3$.

Όπως είδαμε στην παραπάνω εξίσωση στην περίπτωση του r_2 το μερικό γινόμενο είναι θετικό αφού η τριάδα που κωδικοποιείται σε Modified-Booth και τα πολλαπλασιαζόμενα ψηφία έχουν το ίδιο bit προσήμου και συγκεκριμένα είναι το A_5. Όπως με το r_2 έτσι συμβαίνει και με τα υπόλοιπα μερικά γινόμενα, άρα δεν χρειάζεται να χρησιμοποιήσουμε ένα παραπάνω bit για να παραστήσουμε το μερικό γινόμενο. Επιπροσθέτως δεν χρειάζεται να χρησιμοποιηθούν διορθωτικοί όροι για τον ίδιο λόγο το οποίο σημαίνει λιγότερο κύκλωμα για τον τετραγωνιστή MSF SQUARER.

Η διεργασία απόδοσης τιμής στα μερικά γινόμενα χωρίζεται σε δύο μέρη. Το πρώτο σκέλος είναι οι πύλες για το πρώτο bit του μερικού γινομένου και το δεύτερο σκέλος είναι οι πύλες για όλα τα υπόλοιπα bit εκτός από το τελευταίο.



Σχήμα 4.24 Πύλες της μονάδας παραγωγής μερικών γινομένων του MSF SQUARER

Στο Σχήμα 4.24 φαίνονται οι πύλες που χρησιμοποιήθηκαν για την παραγωγή των μερικών γινομένων. Προφανώς οι παραπάνω πύλες εφαρμόζονται κοινά σε κάθε κωδικοποιημένη τριάδα του A (πλέον ως m, x1, x2).

Το μόνο που δεν απεικονίζεται στο σχήμα και χρίζει αναφοράς είναι ότι ενώ εξετάζονται οι δυάδες των A_1-A_0 και A_2-A_1 και ούτω καθεξής, θα πρέπει λόγω της φύσης του αλγορίθμου να εξεταστεί και η δυάδα A_{n-1}-A_{n-3} όπου n είναι το πλήθος των εξεταζόμενων δυαδικών bit για την παραγωγή κάθε μερικού γινομένου.

Τα δύο ανεξάρτητα bits που προστίθενται στις θέσεις 2^0 και 2^2 (p_0 και p_2 αντίστοιχα) του δέντρου που σχηματίζεται υπολογίζονται και αυτά σε αυτή την μονάδα με τις πύλες που παρουσιάζονται στις παρακάτω σχέσεις:

$p_0 = A(0)$

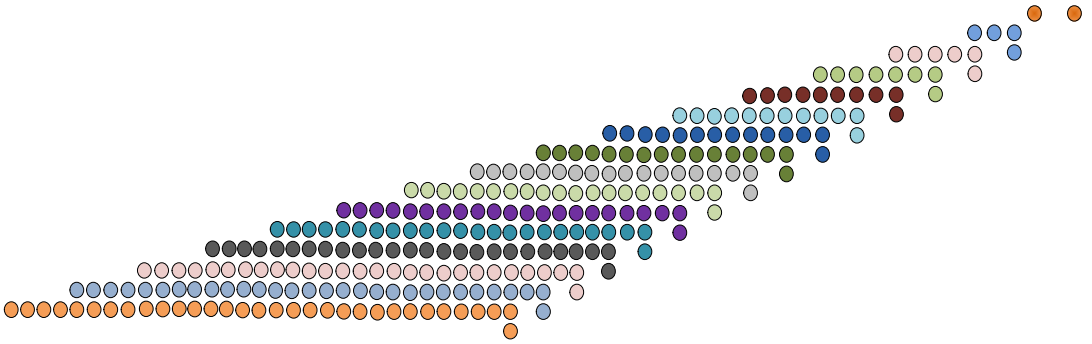
$p_2 = A(1)$ and (not $A(0)$)

Έτσι με αυτές τις πύλες η μονάδα παραγωγής μερικών γινομένων βγάζει στην έξοδο της όλους τους αριθμούς που πρέπει να προστεθούν για να βγει σωστό αποτέλεσμα εκτός από τα κρατούμενα.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Αυτή η μονάδα παίρνει ως είσοδο όλες τις εξόδους από την μονάδα παραγωγής μερικών γινομένων και μαζί με τα κρατούμενα τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα και τα κρατούμενα παρουσιάζονται στο Σχήμα 4.25 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 4.25 Μερικά γινόμενα και κρατούμενα MSF SQUARER

Στην περίπτωση που το m είναι ίσο με 1, το κύκλωμα της μονάδας παραγωγής μερικών γινομένων χρησιμοποιώντας την πύλη XOR και έχοντας ως μία από τις δύο εισόδους της το m, βρίσκει το συμπλήρωμα ως προς 1 του αριθμού και το ολισθαίνει όταν αυτό χρειάζεται. Στο αποτέλεσμα όμως πρέπει να προστεθούν τα συμπληρώματα ως προς δύο αυτών των αριθμών. Αυτό επιτυγχάνεται προσθέτοντας μία μονάδα στο LSB κάθε μερικού γινομένου όταν το σήμα m αυτής της σειράς είναι ίσο με 1. Στο σχήμα 4.25 αυτό απεικονίζεται με μία τελεία ακριβώς κάτω από το LSB του μερικού γινομένου. Η πρώτη οριζόντια σειρά (ξεκινώντας την μέτρηση από πάνω) από bit αποτελεί τα δύο ανεξάρτητα bits, η δεύτερη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο r_0, η τρίτη οριζόντια σειρά από bit αποτελεί το δεύτερο μερικό γινόμενο r_1, και ούτω καθεξής. Κάθε μερικό γινόμενο

βρίσκεται 2 θέσεις αριστερότερα από το προηγούμενο όμοιο μερικό γινόμενο του λόγω της Modified Booth κωδικοποίησης.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως το αποτέλεσμα ενός πολλαπλασιασμού 32x32 bits θα είναι 64 bits, έτσι ο αριθμός X θα είναι 64 bit. Τα μερικά γινόμενα που είναι δεν είναι 64 bits εισάγονται με μηδενικά στα αριστερά και στα δεξιά τους για να γίνεται σωστά η άθροιση. Για παράδειγμα το μερικό γινόμενο r_0_b εισάγεται στην είσοδο του DW02_tree ως:
"00000000000000000000000000000000000000000000000000000000000000"&r0&"000"
όπου r0 είναι τα παραπάνω 3 bits που φαίνονται στο σχήμα 4.25 στην πρώτη σειρά μερικών γινομένων (μη συμπεριλαμβανομένου των δύο ανεξάρτητων bit). Τα μηδενικά απλοποιούνται από τον design compiler και είναι σαν να μην μπαίνουν ψηφία σε αυτά τα σημεία δηλαδή δεν θα υπάρξει κανένας full adder ή half adder που να πάρει ως είσοδο το 0.

Έτσι το αποτέλεσμα του DW02_tree και όλου του αλγορίθμου MSF είναι ένας αριθμός σε carry save μορφή 64 bit, δηλαδή δύο σειρές από 64 bit εκτός αν χρησιμοποιήσουμε τον DW01_add για να προσθέσουμε το carry & το save (δηλαδή τους δύο καταχωρητές) και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

4.2.3 Δεύτερη υλοποίηση MSF SQUARER

Η δεύτερη υλοποίηση του MSF SQUARER έγινε με διαφορετική μονάδα κωδικοποίησης άρα και αναγκαστικά με διαφορετική μονάδα παραγωγής μερικών γινομένων και παρουσιάζεται παρακάτω.

Encoder

Για να κωδικοποιηθούν τα ψηφία του A σε modified booth ψηφία χρησιμοποιείται μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τέσσερα σήματα για την κάθε τριάδα εκτός από την κάθε πρώτη τριάδα ($A_1A_0$0) αφού όπως φαίνεται από το Σχήμα 4.23 δεν χρειάζεται. Συγκεκριμένα κωδικοποιούνται 15 τριάδες σε 4·15=60 σήματα.

Τα τέσσερα αυτά σήματα είναι τα: n, z, nx1 και nx2.

Στον Πίνακα 4.4 αναφέρονται εκτεταμένα οι τιμές των προαναφερθέντων σημάτων.

Πίνακας 4.4 Πίνακας αληθείας σημάτων n, z, nx1, nx2

A_{2i+1}	A_{2i}	A_{2i-1}	n_i	z_i	$nx1_i$	$nx2_i$	Booth: W_i
0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	+1
0	1	0	0	0	0	1	+1
0	1	1	0	0	1	0	+2
1	0	0	1	0	1	0	-2
1	0	1	1	0	0	1	-1
1	1	0	1	1	0	1	-1
1	1	1	1	1	1	0	0

Το nx2 είναι 1 όταν το ψηφίο κωδικοποίησης W είναι ίσο με +1 ή −1, ενώ σε κάθε άλλη περίπτωση το x2 είναι 0.

Το nx1 είναι 1 όταν το nx2 είναι ίσο με 0 και χρησιμοποιείται για την ολίσθηση του μερικού γινομένου.

Το z είναι το σήμα που χρησιμοποιείται για να επιβληθεί το μηδέν όταν το nx1 είναι 1, στο μερικό γινόμενο είτε στο θετικό μηδέν (000) είτε στο αρνητικό (111).

Το n είναι 1 όταν το $\mathbf{A_{2i+1}}$ κάθε τριάδας είναι 1, που αυτό συνεπάγεται ότι πρέπει να προστεθεί το συμπλήρωμα του δυαδικών bit που μπαίνουν σαν είσοδος για την δημιουργία του μερικού γινομένου.

Έτσι οι πύλες που χρησιμοποιήθηκαν για να υλοποιηθούν αυτά τα σήματα είναι οι παρακάτω:

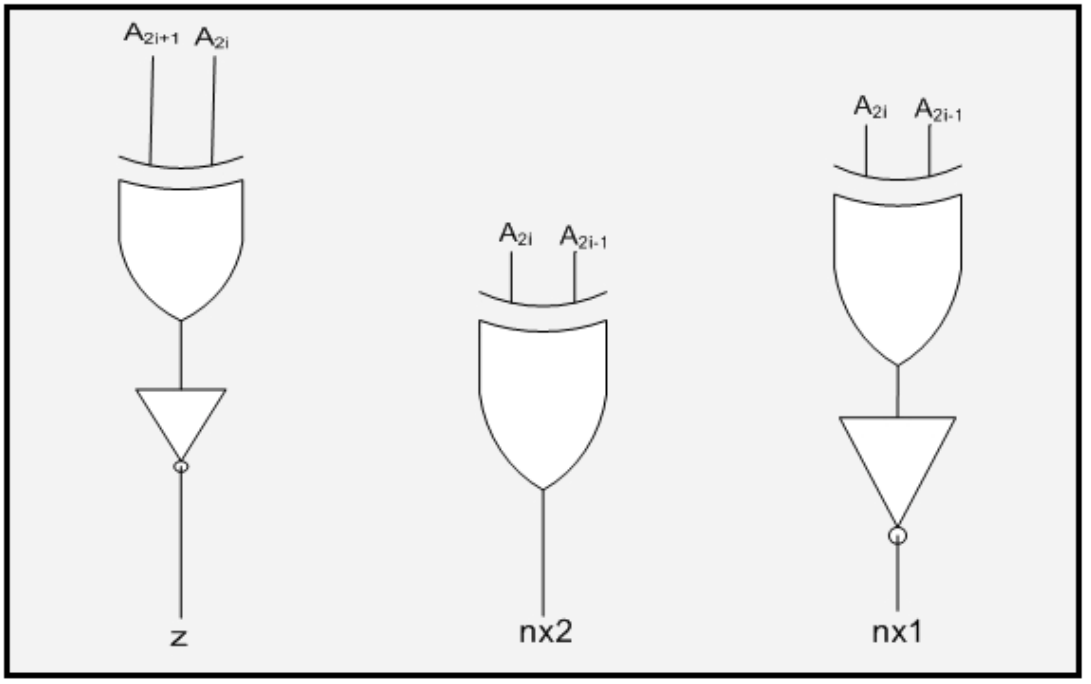
$$n = \mathbf{A_{2i+1}}$$

$$z = \text{not}(\mathbf{A_{2i+1}} \text{ xor } \mathbf{A_{2i}})$$

$$nx2 = (\mathbf{A_{2i}} \text{ xor } \mathbf{A_{2i-1}})$$

$$nx1 = \text{not } (\mathbf{A_{2i}} \text{ xor } \mathbf{A_{2i-1}})$$

Και επίσης οι παραπάνω πύλες φαίνονται σχηματικά και στο Σχήμα 4.26.



Σχήμα 4.26 Πύλες του κωδικοποιητή της δεύτερης υλοποίησης του MSF SQUARER

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από την κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων λαμβάνει ως είσοδο 4 σήματα των 15 bit αφού έχουν κωδικοποιηθεί 15 τριάδες του A (τα τέσσερα σήματα για την κωδικοποίηση του A είναι τα: n, z, nx1 και nx2.) και τέλος 1 σήμα 32 bit που αντιπροσωπεύει την τιμή του A.

Η μονάδα αυτή χρησιμοποιεί τα παραπάνω σήματα ώστε στις εξόδους της να γίνουν διαθέσιμα τα μερικά γινόμενα. Θεωρείται ότι τα r_i είναι τα μερικά γινόμενα που δημιουργούνται με βάση τα modified booth ψηφία του A. Τα μερικά γινόμενα r_i ξεκινάνε από 3 ψηφία και σε κάθε επόμενο μερικό γινόμενο προσθέτονται άλλα δύο ψηφία.

Τα μερικά γινόμενα έχουν την μορφή που φαίνεται στο σχήμα 4.23 δηλαδή για παράδειγμα το δεύτερο μερικό γινόμενο ξεκινώντας την μέτρηση από τα LSB είναι:

$$r_2 = 2^5 A_{MB,5.4,3} \cdot A_5 A_4 A_2 A_1 A_0 ,$$

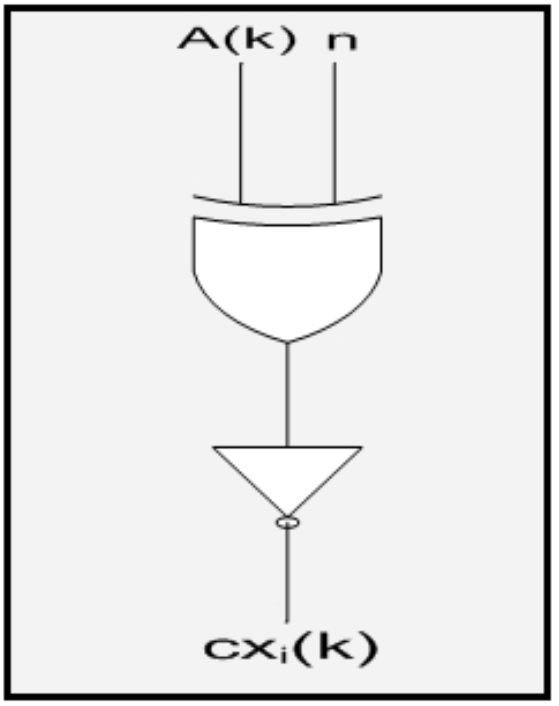
το οποίο r_2 είναι 5 bits και το $A_{MB,5.4,3}$ είναι το Modified-Booth ψηφίο της κωδικοποιημένης τριάδας $A_5A_4A_3$.

Όπως και στην προηγούμενη υλοποίηση έτσι και εδώ, στην περίπτωση του r_2 το μερικό γινόμενο είναι θετικό αφού η τριάδα που κωδικοποιείται σε Modified-Booth και τα

πολλαπλασιαζόμενα ψηφία έχουν το ίδιο bit προσήμου και συγκεκριμένα είναι το A_5. Όπως με το r_2 έτσι συμβαίνει και με τα υπόλοιπα μερικά γινόμενα, άρα δεν χρειάζεται να χρησιμοποιήσουμε ένα παραπάνω bit για να παραστήσουμε το μερικό γινόμενο. Επιπροσθέτως δεν χρειάζεται να χρησιμοποιηθούν διορθωτικοί όροι για τον ίδιο λόγο το οποίο σημαίνει λιγότερο κύκλωμα για τον τετραγωνιστή MSF SQUARER.

Η διεργασία απόδοσης τιμής στα μερικά γινόμενα χωρίζεται σε δύο διακριτά μέρη. Το πρώτο σκέλος είναι οι πύλες για να παραχθούν ενδιάμεσα εσωτερικά σήματα τα οποία θα χρησιμοποιήσει εν συνεχεία η μονάδα παραγωγής μερικών γινομένων. Τα σήματα ονομάζονται cx_i και συμπιέζουν τις πληροφορίες του A και του n, δηλαδή κατά πόσο τα ψηφία του A που θα χρησιμοποιηθούν, θα πρέπει να συμπληρωθούν ή όχι.

Στο Σχήμα 4.27 φαίνονται οι πύλες που χρησιμοποιήθηκαν για την παραγωγή των cx_i.



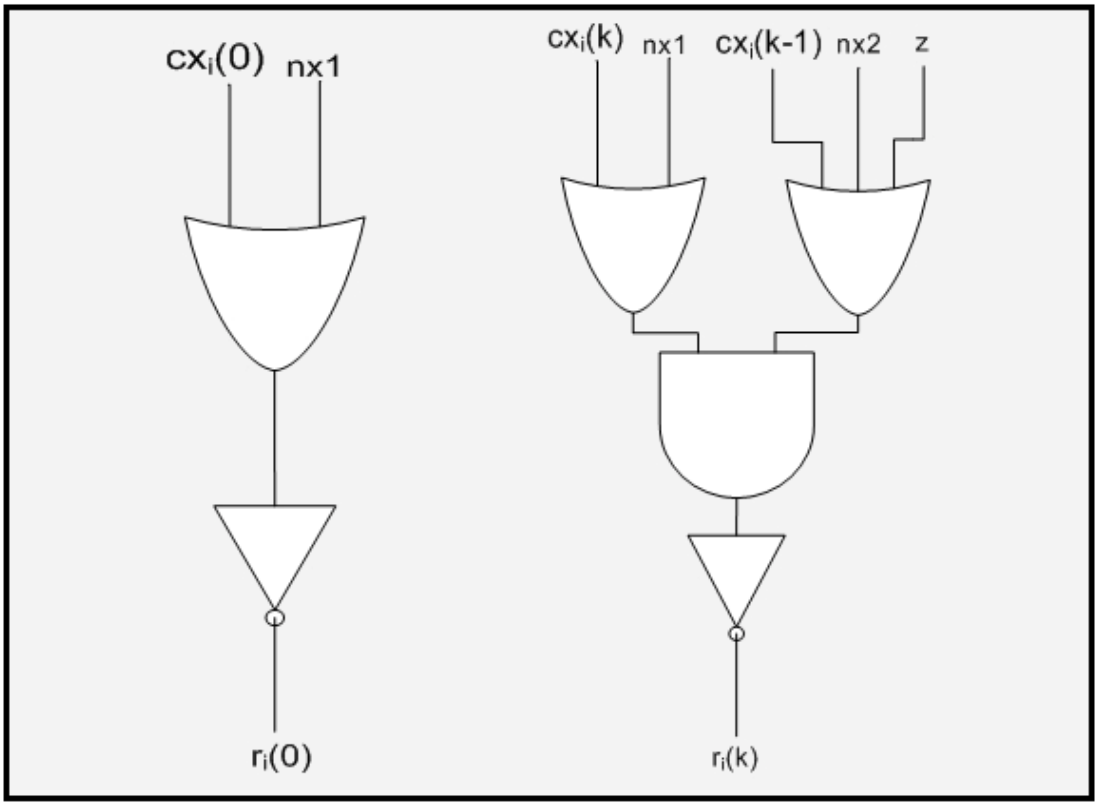
Σχήμα 4.27 Πύλες παραγωγής σημάτων cx_i του MSF SQUARER

Προφανώς οι παραπάνω πύλες εφαρμόζονται κοινά σε κάθε κωδικοποιημένη τριάδα του A (πλέον ως n, z, nx1 και nx2).

Το μόνο που δεν απεικονίζεται στο σχήμα και χρίζει αναφοράς είναι ότι ενώ εξετάζονται τα ψηφία A_0, A_1, A_2 και ούτω καθεξής, θα πρέπει λόγω της φύσης του αλγορίθμου να εξεταστεί και μετά το A_{n-3}, το A_{n-1} για την παραγωγή του επόμενου ψηφίου του cx_i, όπου n είναι το πλήθος των εξεταζόμενων δυαδικών bit για την παραγωγή κάθε συγκεκριμένου μερικού γινομένου.

Το δεύτερο σκέλος της μονάδας παραγωγής μερικών γινομένων χρησιμοποιεί τα σήματα cx_i για να παράγει το τελικό αποτέλεσμα των μερικών γινομένων. Αυτό γίνεται σε δύο βήματα με το πρώτο βήμα να αφορά μόνο το bit με το μικρότερο βάρος (LSB) και το δεύτερο να είναι οι πύλες για όλα τα υπόλοιπα bit.

Στο σχήμα 4.28 φαίνονται οι πύλες που χρησιμοποιήθηκαν για την παραγωγή των μερικών γινομένων.



Σχήμα 4.28 Πύλες της μονάδας παραγωγής μερικών γινομένων του MSF SQUARER

Προφανώς οι παραπάνω πύλες εφαρμόζονται κοινά σε κάθε κωδικοποιημένη τριάδα του A (πλέον ως n, z, nx1 και nx2).

Τα δύο ανεξάρτητα bits που προστίθενται στις θέσεις 2^0 και 2^2 (p_0 και p_2 αντίστοιχα) του δέντρου που σχηματίζεται υπολογίζονται και αυτά σε αυτή την μονάδα με τις πύλες που παρουσιάζονται στις παρακάτω σχέσεις:

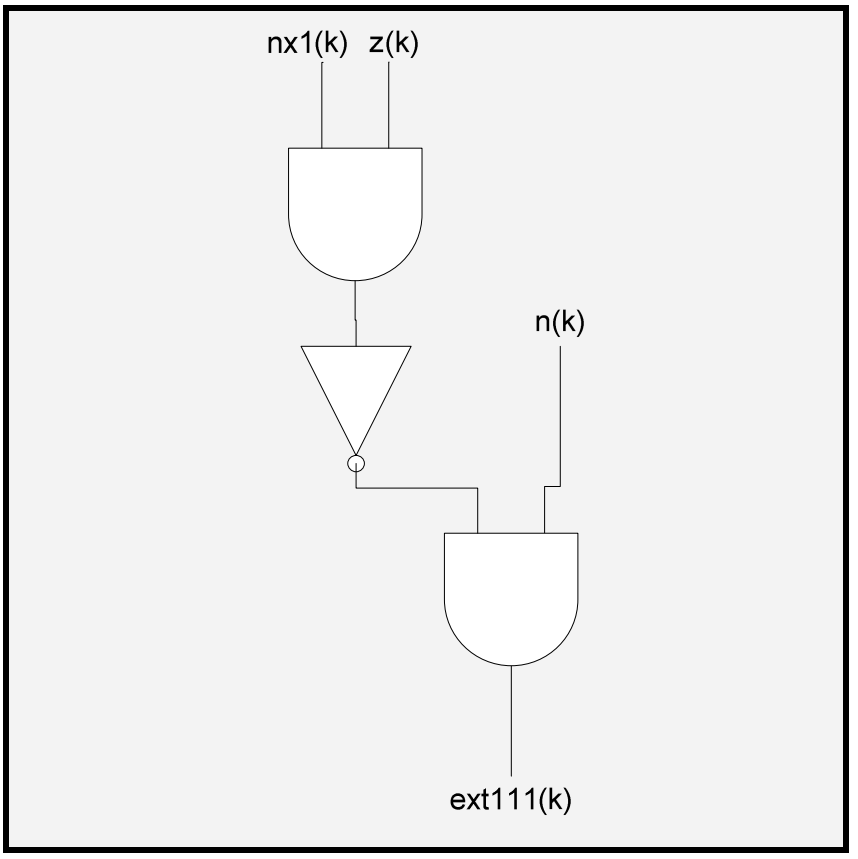
$p_0 = A(0)$

$p_2 = A(1)$ and (not $A(0)$)

Όταν κωδικοποιείται σε modified booth μία τυχαία τριάδα του A και αυτή είναι 111, τότε δεν θα πρέπει να προστίθεται μία μονάδα στο LSB του μερικού γινομένου με την ίδια τεχνική

που ακολουθήθηκε και σε προηγούμενους αλγορίθμους. Άρα δεν μπορεί να χρησιμοποιηθεί το σήμα n για την πρόσθεση κρατουμένων αφού το n είναι ίσο με 1 ακόμα και όταν η τριάδα προς κωδικοποίηση του Α είναι η 111.

Για αυτό θα χρησιμοποιηθεί το σήμα ext111 για την εν συνεχεία πρόσθεση κρατουμένων και το σχήμα 4.29 απεικονίζει τις πύλες που χρησιμοποιούνται για να παράγουν αυτό το σήμα.



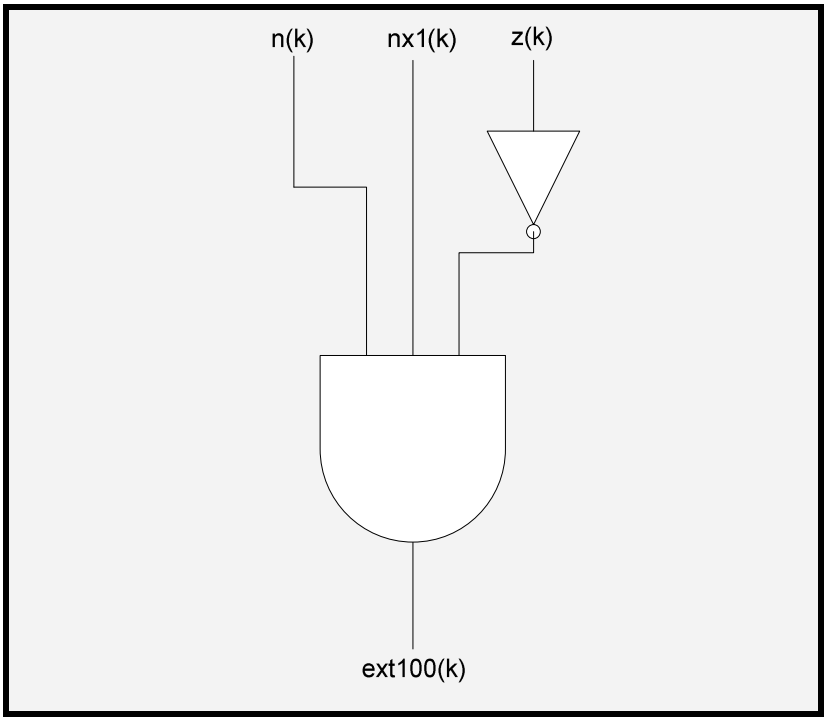
Σχήμα 4.29 Πύλες της μονάδας παραγωγής του σήματος ext111

Στον Πίνακα 4.5 αναφέρονται εκτεταμένα οι τιμές των σήματος ext111.

Πίνακας 4.5 Πίνακας αληθείας σήματος ext111

A_{2i+1}	A_{2i}	A_{2i-1}	n_i	z_i	$nx1_i$	**ext111**
0	0	0	0	1	1	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	1	0	0	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0

Τέλος, όταν κωδικοποιείται σε modified booth μία τυχαία τριάδα του A και αυτή είναι 100, τότε θα πρέπει να προστίθεται μία επιπλέον μονάδα στο LSB του μερικού γινομένου με την ίδια τεχνική που ακολουθήθηκε και σε προηγούμενους αλγορίθμους για την πρόσθεση του κρατουμένου. Για αυτό θα χρησιμοποιηθεί το σήμα ext100 για την εν συνεχεία πρόσθεση επιπλέον κρατουμένων και το Σχήμα 4.30 απεικονίζει τις πύλες που χρησιμοποιούνται για να παράγουν αυτό το σήμα.



Σχήμα 4.30 Πύλες της μονάδας παραγωγής του σήματος ext100

Στον Πίνακα 4.6 αναφέρονται εκτεταμένα οι τιμές του σήματος ext100.

Πίνακας 4.6 Πίνακας αληθείας σήματος ext100

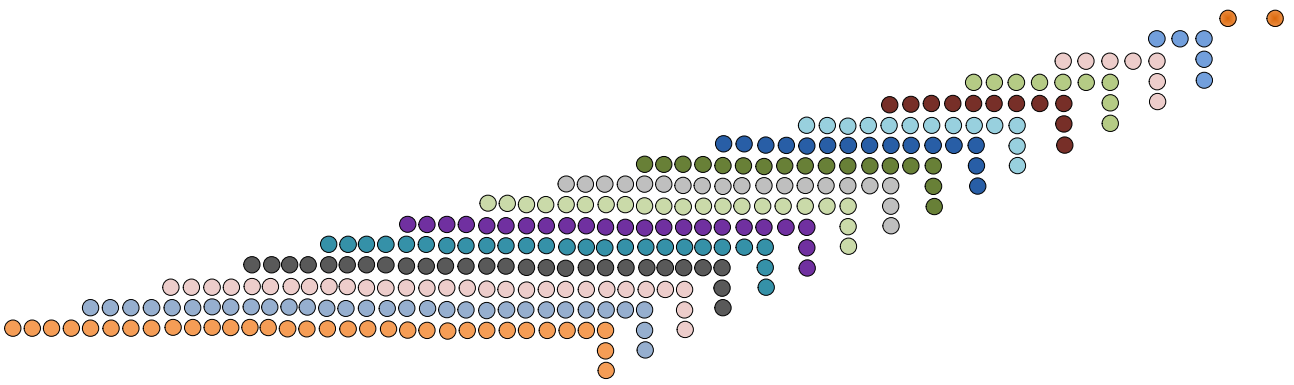
A_{2i+1}	A_{2i}	A_{2i-1}	n_i	z_i	$nx1_i$	**ext100**
0	0	0	0	1	1	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Έτσι με αυτές τις πύλες η μονάδα παραγωγής μερικών γινομένων βγάζει στην έξοδο της όλους τους αριθμούς που πρέπει να προστεθούν για να βγει σωστό αποτέλεσμα, τα πρωτεύοντα κρατούμενα καθώς και τα επιπλέον κρατούμενα για την περίπτωση της τριάδας 100.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Αυτή η μονάδα παίρνει ως είσοδο όλες τις εξόδους από την μονάδα παραγωγής μερικών γινομένων και μαζί με τα κρατούμενα τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα και τα κρατούμενα παρουσιάζονται στο σχήμα 4.31 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 4.31 Μερικά γινόμενα και κρατούμενα MSF SQUARER

Στην περίπτωση που το n είναι ίσο με 1, το κύκλωμα της μονάδας παραγωγής μερικών γινομένων χρησιμοποιώντας την πύλη XOR και έχοντας ως μία από τις δύο εισόδους της το n, βρίσκει το συμπλήρωμα ως προς 1 του αριθμού και το ολισθαίνει όταν αυτό χρειάζεται. Στο αποτέλεσμα όμως πρέπει να προστεθούν τα συμπληρώματα ως προς δύο αυτών των αριθμών. Αυτό επιτυγχάνεται προσθέτοντας μία μονάδα στο LSB κάθε μερικού γινομένου όταν το σήμα ext111 αυτής της σειράς είναι ίσο με 1. Στο σχήμα 4.31 αυτό απεικονίζεται με μία τελεία ακριβώς κάτω από το LSB του μερικού γινομένου στο ίδιο χρώμα με αυτό. Επιπλέον η μία ακόμα διαφορά είναι η δεύτερη τελεία ακριβώς κάτω από την προαναφερθείσα η οποία απεικονίζει το σήμα ext100 για να προστεθεί μία μονάδα όταν η τριάδα που κωδικοποιείται στο συγκεκριμένο μερικό γινόμενο είναι η 100 όπως αναφέρθηκε και παραπάνω.

Η πρώτη οριζόντια σειρά (ξεκινώντας την μέτρηση από πάνω) από bit αποτελεί τα δύο ανεξάρτητα bits, η δεύτερη οριζόντια σειρά από bit αποτελεί το πρώτο μερικό γινόμενο r_0, η τρίτη οριζόντια σειρά από bit αποτελεί το δεύτερο μερικό γινόμενο r_1, και ούτω καθεξής. Κάθε μερικό γινόμενο βρίσκεται 2 θέσεις αριστερότερα από το προηγούμενο όμοιο μερικό γινόμενο του λόγω της Modified Booth κωδικοποίησης.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως το αποτέλεσμα ενός πολλαπλασιασμού 32x32 bits θα είναι 64 bits, έτσι ο αριθμός X θα είναι 64 bit. Τα μερικά γινόμενα αφού δεν είναι 64 bits εισάγονται με μηδενικά στα αριστερά και στα δεξιά τους για να γίνεται σωστά η άθροιση. Για παράδειγμα το μερικό γινόμενο r_0 εισάγεται στην είσοδο του DW02_tree ως:
"00000000000000000000000000000000000000000000000000000000000000000"&r0&"000"
Τα μηδενικά απλοποιούνται από τον design compiler και έτσι το αποτέλεσμα του DW02_tree και όλου του αλγορίθμου MSF είναι ένας αριθμός σε carry save μορφή 64 bit, δηλαδή δύο σειρές από 64 bit.

4.3 BOOTH FOLDING SQUARER

Η υλοποίηση του Booth folding squarer έγινε με την ίδια μονάδα κωδικοποίησης που χρησιμοποιήθηκε για την δεύτερη υλοποίηση του MSF squarer και αναγκαστικά με παρόμοια μονάδα παραγωγής μερικών γινομένων. Στην βιβλιογραφία αυτή η υλοποίηση παρουσιάζει καλύτερα αποτελέσματα από την απλή υλοποίηση χρησιμοποιώντας μονάδες που είναι παρόμοιες σε αυτές που χρησιμοποιήθηκαν στην πρώτη υλοποίηση του MSF squarer για αυτό και προτιμήθηκε.

Encoder

Για να κωδικοποιηθούν τα ψηφία του A σε modified booth ψηφία χρησιμοποιείται πάλι μια μονάδα κωδικοποιητή που κωδικοποιεί κάθε τριάδα δυαδικών αριθμών σε τέσσερα σήματα (n, z, nx1 και nx2)για την κάθε τριάδα εκτός από την τελευταία τριάδα ($A_{29}A_{30}A_{31}$) αφού δεν χρειάζεται. Συγκεκριμένα κωδικοποιούνται 15 τριάδες σε 4·15=60 σήματα και για την τριάδα $A_{29}A_{30}A_{31}$ υπολογίζονται μόνο τα z και nx2.

Στον Πίνακα 4.4 αναφέρονται εκτεταμένα οι τιμές των προαναφερθέντων σημάτων.

Οι πύλες που χρησιμοποιήθηκαν για να υλοποιηθούν αυτά τα σήματα είναι οι παρακάτω:

$n = \mathbf{A_{2i+1}}$

$z = not(\mathbf{A_{2i+1}}\ xor\ \mathbf{A_{2i}})$

$nx2 = (\mathbf{A_{2i}}\ xor\ \mathbf{A_{2i-1}})$

$nx1 = not\ (\mathbf{A_{2i}}\ xor\ \mathbf{A_{2i-1}})$

Και επίσης φαίνονται σχηματικά και στο Σχήμα 4.26.

Partial Product Generator

Τα σήματα που δημιουργήθηκαν από τον κωδικοποιητή που χρησιμοποίησε τον αλγόριθμο κωδικοποίησης της τροποποιημένης μεθόδου Booth χρησιμοποιούνται από την επόμενη μονάδα η οποία παράγει τα μερικά γινόμενα. Πιο συγκεκριμένα η μονάδα παραγωγής μερικών γινομένων λαμβάνει ως είσοδο 2 σήματα των 15 bit (n και nx1) και 2 σήματα των 14 bit (z και nx2) και τέλος 1 σήμα 32 bit που αντιπροσωπεύει την τιμή του A.

Η μονάδα αυτή χρησιμοποιεί τα παραπάνω σήματα ώστε στις εξόδους της να γίνουν διαθέσιμα τα μερικά γινόμενα. Θεωρείται ότι τα r_i είναι τα μερικά γινόμενα που δημιουργούνται με βάση τα modified booth ψηφία του A. Τα μερικά γινόμενα r_i ξεκινάνε από 34 ψηφία και σε κάθε επόμενο μερικό γινόμενο μειώνονται δύο ψηφία εκτός από το r_{16} το οποίο παρίσταται ως τρία δυαδικά ψηφία.

Σε αντιδιαστολή με την προηγούμενη υλοποίηση, στην περίπτωση των r_i το μερικό γινόμενο δεν είναι πάντα θετικό αφού η τριάδα που κωδικοποιείται σε Modified-Booth και τα πολλαπλασιαζόμενα ψηφία δεν έχουν το ίδιο bit προσήμου. Άρα χρειάζεται να χρησιμοποιήσουμε ένα παραπάνω bit για να παραστήσουμε το μερικό γινόμενο το οποίο θα έχει αρνητική αξία. Επιπροσθέτως χρειάζεται να χρησιμοποιηθούν διορθωτικοί όροι για τον ίδιο λόγο.

Η διεργασία απόδοσης τιμής στα μερικά γινόμενα χωρίζεται σε δύο διακριτά μέρη. Το πρώτο σκέλος είναι οι πύλες για να παραχθούν ενδιάμεσα εσωτερικά σήματα τα οποία θα χρησιμοποιήσει εν συνεχεία η μονάδα παραγωγής μερικών γινομένων. Τα σήματα ονομάζονται cx_i και συμπιέζουν τις πληροφορίες του A και του n, δηλαδή κατά πόσο τα ψηφία του A που θα χρησιμοποιηθούν, θα πρέπει να συμπληρωθούν ή όχι.

Στο σχήμα 4.27 φαίνονται οι πύλες που χρησιμοποιήθηκαν για την παραγωγή των cx_i.

Το μόνο που δεν απεικονίζεται στο σχήμα και χρίζει αναφοράς είναι ποια ψηφία του A χρησιμοποιούνται κάθε φορά. Για την παραγωγή του cx_1 το οποίο είναι 30 bit, χρησιμοποιούνται σαν είσοδο στις παραπάνω πύλες τα 30 ψηφία του A με το μεγαλύτερο

βάρος δηλαδή τα 30 MSB του A. Ομοίως για την παραγωγή του cx_2 το οποίο είναι 28 bit, χρησιμοποιούνται σαν είσοδο στις παραπάνω πύλες τα 28 ψηφία του A με το μεγαλύτερο βάρος δηλαδή τα 28 MSB του A. Συνεχίζοντας έτσι για τα υπόλοιπα ενδιάμεσα στάδια, για την παραγωγή του cx_{15} το οποίο είναι 2 bit, χρησιμοποιούνται σαν είσοδο στις παραπάνω πύλες τα 2 MSB του A.

Το δεύτερο σκέλος της μονάδας παραγωγής μερικών γινομένων χρησιμοποιεί τα σήματα cx_i για να παράγει το τελικό αποτέλεσμα των μερικών γινομένων μαζί με τα υπόλοιπα σήματα που έχουν προέλθει από την κωδικοποίηση. Αυτό γίνεται σε τέσσερα βήματα:

Α) το πρώτο βήμα να αφορά μόνο την δημιουργία των 3 LSB του μερικού γινομένου,

Β) το δεύτερο βήμα να αφορά μόνο το τέταρτο bit,

Γ) το τρίτο βήμα να είναι οι πύλες για όλα τα υπόλοιπα bit εκτός από το τελευταίο bit

Δ) το τέταρτο βήμα είναι οι πύλες που χρησιμοποιούνται για την δημιουργία του τελευταίου bit κάθε μερικού γινομένου που έχει αρνητική αξία.

Σχετικά με το πρώτο βήμα, τα τρία bit με το μικρότερο βάρος των μερικών γινομένων είναι πάντοτε καθορισμένα ως εξής:

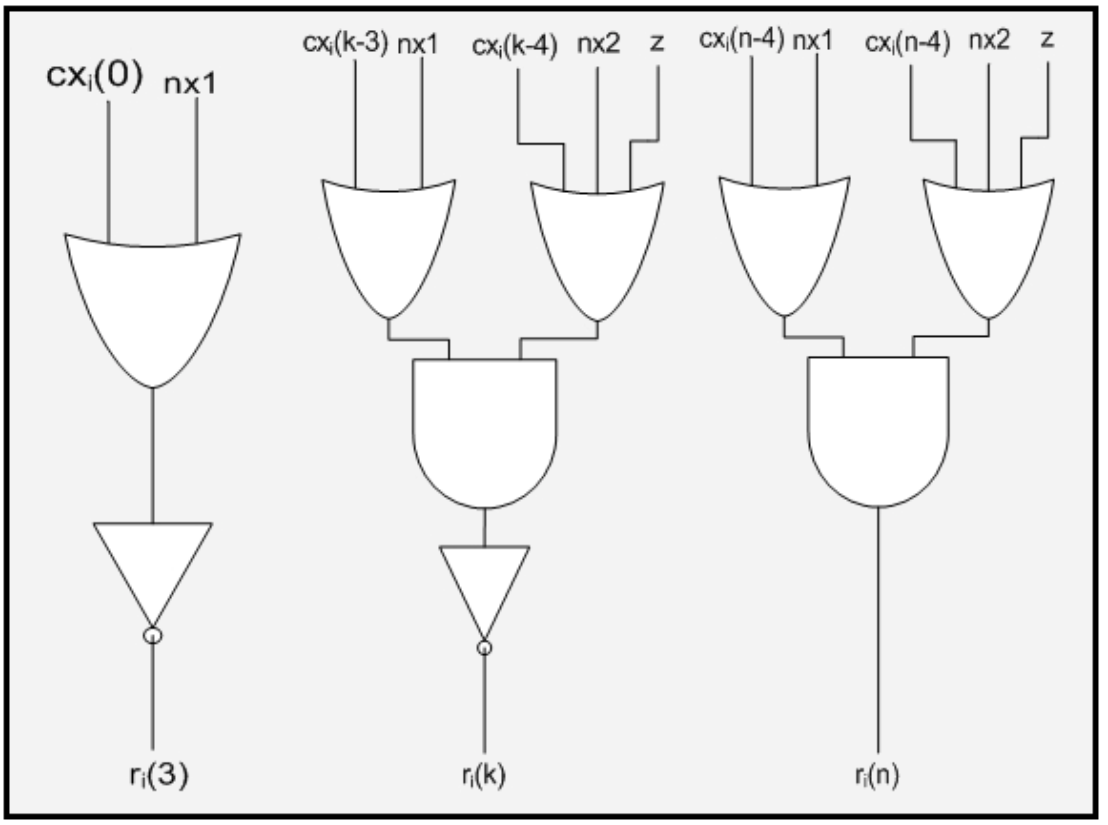
$r_i(0)$ = nx2(i)

$r_i(1) = 0$

$r_i(2)$ = not(z(i) or nx2(i))

και αυτές οι πύλες ισχύουν για κάθε μερικό γινόμενο, όπου προφανώς όταν αλλάζουμε μερικό γινόμενο αλλάζει και η τιμή των nx2 και z.

Σχετικά με το δεύτερο, τρίτο και τέταρτο βήμα, στο σχήμα 4.32 φαίνονται οι πύλες που χρησιμοποιήθηκαν για την παραγωγή των μερικών γινομένων με $4 \leq k < n$ όπου n είναι ίσο με το πλήθος των bit του r_i μειωμένο κατά ένα.



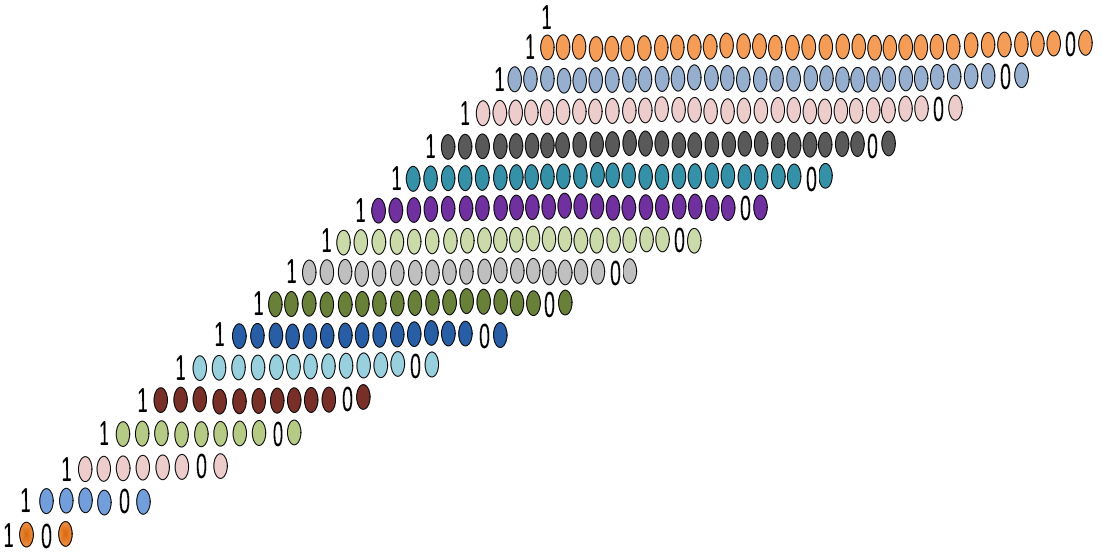
Σχήμα 4.32 Πύλες παραγωγής μερικών γινομένων του Booth folding Squarer

Η μόνη ειδική περίπτωση είναι το r_{16} όπου δεν χρειάζεται να δημιουργηθούν τα εσωτερικά σήματα cx_{16} αλλά μόνο τα $r_{16}(0)$, $r_{16}(1)$ και $r_{16}(2)$.

Wallace tree

Η επόμενη μονάδα είναι ένας δενδρικός συμπιεστής τύπου Wallace. Αυτή η μονάδα παίρνει ως είσοδο όλες τις εξόδους από την μονάδα παραγωγής μερικών γινομένων και μαζί με τους αναγκαίους διορθωτικούς όρους τα προσθέτει όλα μαζί σε ένα δέντρο Wallace.

Τα μερικά γινόμενα και τα οι διορθωτικοί όροι παρουσιάζονται στο σχήμα 4.33 στην σωστή διάταξη που πρέπει να προστεθούν.



Σχήμα 4.33 Μερικά γινόμενα και διορθωτικοί όροι του Booth folding Squarer

Σε κάθε μερικό γινόμενο το δεύτερο LSB είναι πάντα μηδέν όπως απεικονίζεται και στο σχήμα 4.33 λόγω του αλγορίθμου του booth folding.

Η πρώτη οριζόντια σειρά (ξεκινώντας την μέτρηση από πάνω) από bit αποτελεί τα τρία ανεξάρτητα bits ακολουθούμενα από τα υπόλοιπα 31 bit του r_0, η δεύτερη οριζόντια σειρά από bit αποτελεί το δεύτερο μερικό γινόμενο r_1, η τρίτη οριζόντια σειρά από bit αποτελεί το τρίτο μερικό γινόμενο r_2, και ούτω καθεξής. Κάθε μερικό γινόμενο βρίσκεται 4 θέσεις αριστερότερα από το προηγούμενο μερικό γινόμενο.

Η υλοποίηση του δενδρικού συμπιεστή Wallace έγινε με το module DW02_tree της DesignWare. Η μονάδα αυτή παίρνει ως είσοδο σειρές από έναν X αριθμό bits και βγάζει ως έξοδο το αποτέλεσμα σε μορφή carry save και πάλι μήκους X bit. Επειδή είναι γνωστό πως το αποτέλεσμα ενός πολλαπλασιασμού 32x32 bits θα είναι 64 bits, έτσι ο αριθμός X θα είναι 64 bit. Τα μερικά γινόμενα αφού δεν είναι 64 bits εισάγονται με μηδενικά στα αριστερά και στα δεξιά τους για να γίνεται σωστά η άθροιση. Για παράδειγμα το μερικό γινόμενο r_2 εισάγεται στην είσοδο του DW02_tree ως:

"00000000000000000000000000"&r2&"00000000"

Τα μηδενικά απλοποιούνται από τον design compiler και έτσι το αποτέλεσμα του DW02_tree και όλου του αλγορίθμου MSF είναι ένας αριθμός σε carry save μορφή 64 bit, δηλαδή δύο σειρές από 64 bit εκτός αν χρησιμοποιήσουμε τον DW01_add για να προσθέσουμε τους δύο καταχωρητές και να εξάγουμε το αποτέλεσμα σε μορφή συμπληρώματος ως προς δύο σε έναν μόνο καταχωρητή.

4.3.1 Αποτελέσματα υλοποίησης MSF SQUARER

Η πρώτη σύγκριση που έγινε ήταν μεταξύ των δύο υλοποιήσεων του MSF SQUARER για να φανεί ποια είναι η καλύτερη. Έτσι πήραμε μετρήσεις στα 1.3ns για τρεις περιπτώσεις:

α) η είσοδος Α είναι τυχαίος δυαδικός αριθμός (random)

β) η είσοδος Α είναι σήμα φωνής 24 bits (24 bits)

γ) η είσοδος Α είναι σήμα φωνής 16 bits (16 bits)

Η επιφάνεια των δύο κυκλωμάτων για Carry Save μορφή φαίνεται στον Πίνακα 4.7 και η κατανάλωση για τις τρεις παραπάνω περιπτώσεις στον Πίνακα 4.8.

Πίνακας 4.7 Πίνακας επιφάνειας MSF SQUARER σε Carry Save μορφή στα 1.3ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
MSF SQUARER 1	16906
MSF SQUARER 2	15408

Πίνακας 4.8 Πίνακας κατανάλωσης MSF SQUARER σε Carry Save μορφή στα 1.3ns

	MSF SQUARER 1	**MSF SQUARER 2**
Random	0.005251	0.004808
24 bit	0.002599	0.002427
16 bit	0.001138	0.0009542

Βλέπουμε ότι η επιφάνεια της δεύτερης υλοποίησης είναι μικρότερη και η κατανάλωση είναι λιγότερη για ίδια δεδομένα. Αυτό συμβαίνει γιατί το κρίσιμο μονοπάτι της δεύτερης υλοποίησης αποτελείται από λιγότερες λογικές πύλες άρα τα αποτελέσματα γίνονται πιο γρήγορα διαθέσιμα για να δοθούν στο δενδρικό συμπιεστή Wallace.

Στην συνέχεια συγκρίνουμε την δεύτερη υλοποίηση του MSF SQUARER (η οποία στην συνέχεια του κεφαλαίου τιτλοφορείται απλά με τον τίτλο MSF SQUARER, εννοώντας πάντα την δεύτερη υλοποίηση) με τον αλγόριθμο Booth folding. Επίσης, για την μορφή δυαδικής αναπαράστασης έγινε η σύγκριση και με τον τετραγωνιστή της βιβλιοθήκης της DesignWare που συγκεκριμένα υλοποιείται αρχικοποιώντας το DW_square και ο οποίος τιτλοφορείται στις μετρήσεις ως DW squarer.

Η επιφάνεια των κυκλωμάτων για Carry Save και Binary μορφή φαίνεται στους Πίνακες 4.9 και 4.10:

Πίνακας 4.9 Πίνακας επιφάνειας MSF SQUARER σε Carry Save μορφή στα 1.1ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
MSF SQUARER	17217
Booth folding	19297

Πίνακας 4.10 Πίνακας επιφάνειας MSF SQUARER σε δυαδική μορφή στα 1.6ns

Αλγόριθμος	**Επιφάνεια (μm^2)**
MSF SQUARER	31602
Booth folding	30552
DW squarer	35257

Έχει μετρηθεί η κατανάλωση για τις εξής περιπτώσεις:

α) η είσοδος Α είναι τυχαίος δυαδικός αριθμός (random)

β) η είσοδος Α είναι σήμα φωνής 24 bits (24 bit)

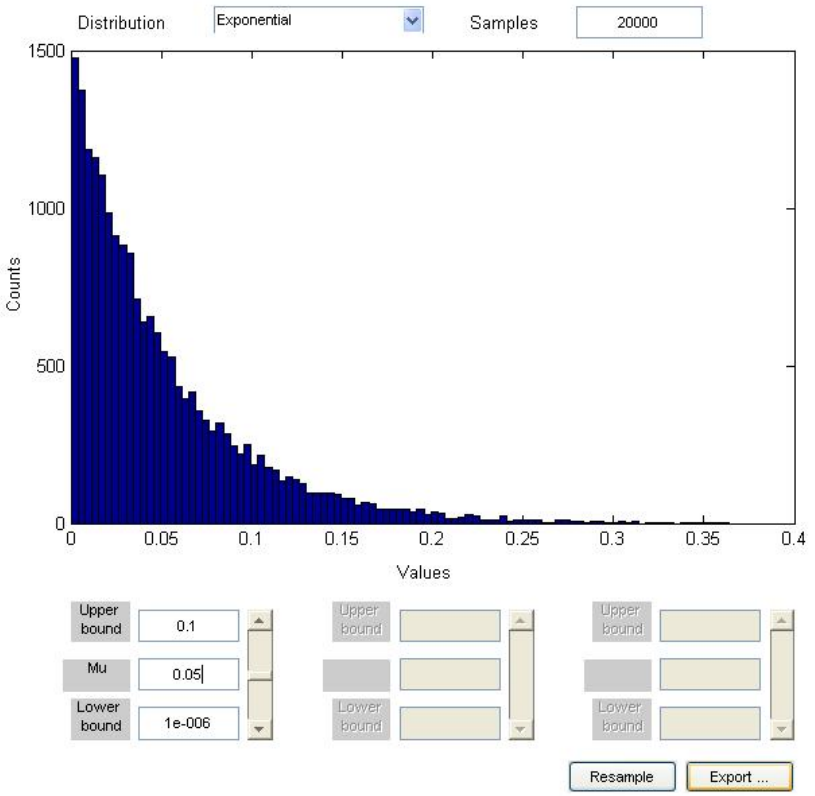
γ) η είσοδος Α είναι σήμα φωνής 16 bits (16 bit)

δ) η είσοδος Α είναι σήμα φωνής 8 bits (8 bit)

ε) η είσοδος Α είναι ειδικό σήμα (Custom Signal)

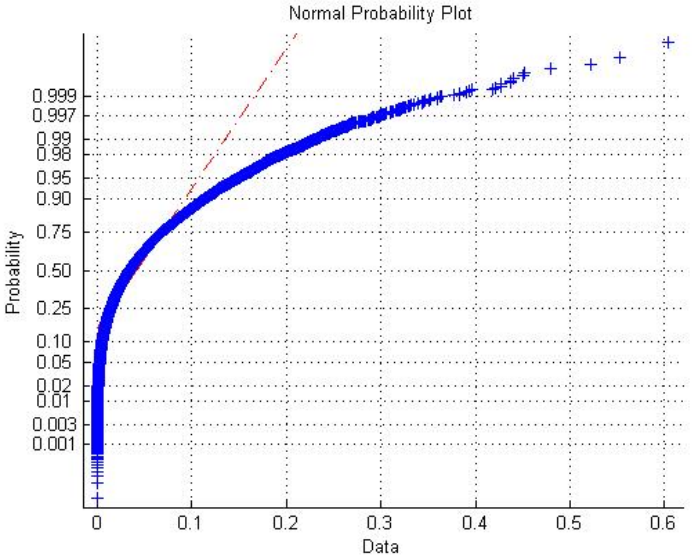
Το σήμα της περίπτωσης (ε) δημιουργήθηκε από το MATLAB με το εργαλείο *randtool* το οποίο δίνει τυχαίους αριθμούς με βάση συγκεκριμένες κατανομές. Επιλέχθηκε η εκθετική κατανομή και με συντελεστές που φαίνονται στο Σχήμα 4.34.

Το σήμα αυτό έχει ως σκοπό να αξιολογήσει περαιτέρω τον αλγόριθμο του MSF Squarer έχοντας τέτοια κατανομή ώστε να έχει πολλούς αριθμούς που να χρειάζονται λίγα bits για αναπαράσταση και όσο αυξάνονται τα bits που χρειάζονται αυτοί οι αριθμοί τόσο να μειώνεται το πλήθος αυτών των αριθμών. Υπενθυμίζεται πως αν θεωρήσουμε πως οι τυχαίοι αριθμοί ανήκουν στο [0,1] τότε οι αριθμοί που είναι μεγαλύτεροι από 0.5 χρειάζονται 32 bits για να αναπαρασταθούν, οι αριθμοί που είναι μεγαλύτεροι από 0.25 χρειάζονται 31 bits για να αναπαρασταθούν, οι αριθμοί που είναι μεγαλύτεροι από 0.125 χρειάζονται 30 bits για να αναπαρασταθούν και ούτω καθεξής.



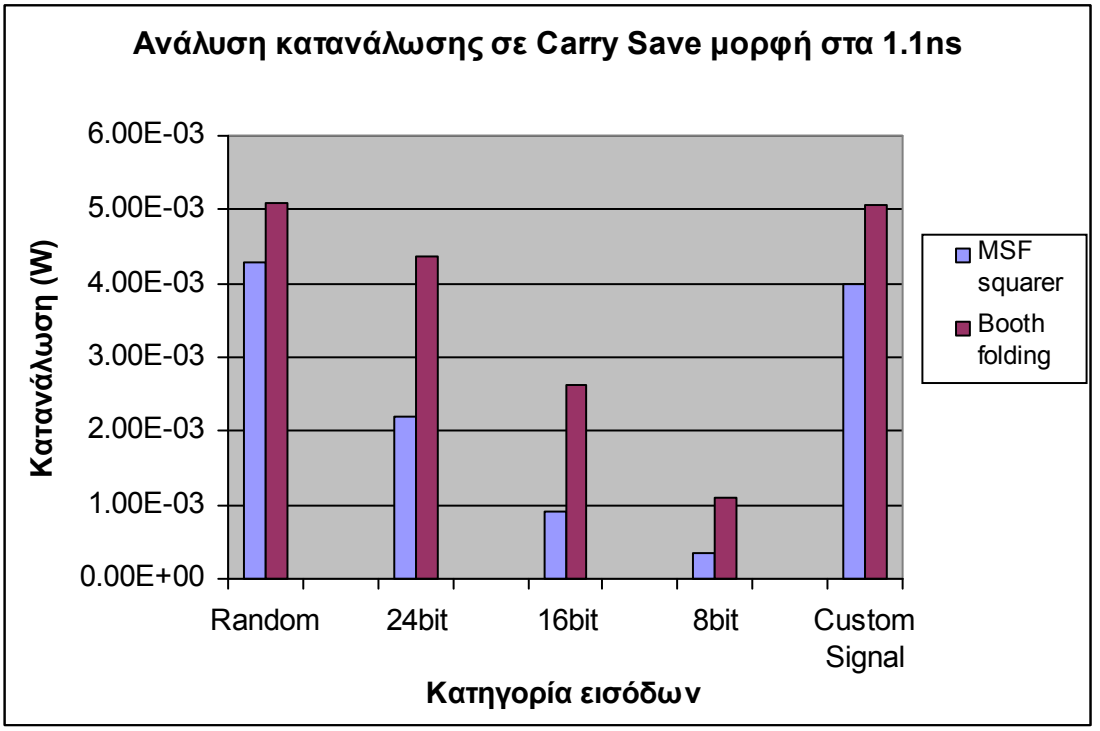
Σχήμα 4.34 Κατανομή πλήθους δειγμάτων για το Custom Signal

Η πιθανοτική κατανομή των αριθμών που ανήκουν στο Custom Signal φαίνεται στο Σχήμα 4.35. Αξίζει επίσης να σημειωθεί πως η ίδια διεργασία έγινε και με αρνητικούς αριθμούς που άνηκαν στο [-1,0] και αυτοί οι αριθμοί οι οποίοι πολλαπλασιάστηκαν με 2^{31} (όπως προφανώς και οι θετικοί) συγχωνεύτηκαν σε τυχαία σειρά με τους θετικούς και αποτέλεσαν την τελική μορφή του Custom Signal.

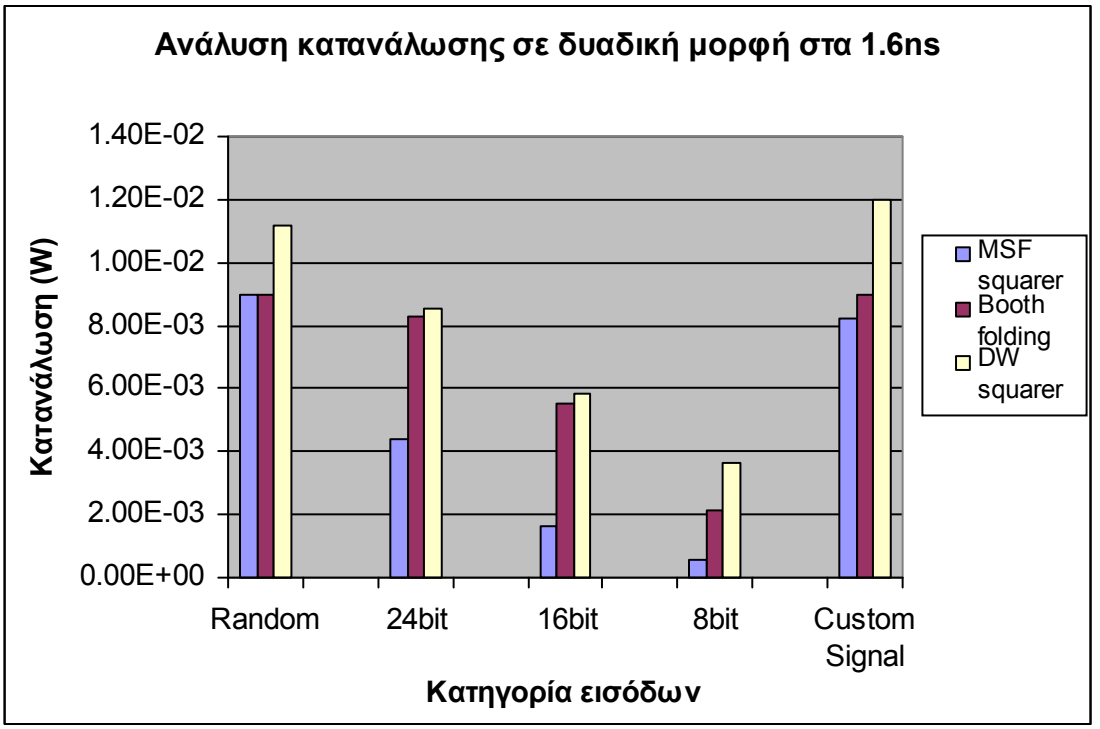


Σχήμα 4.35 Πιθανοτική κατανομή πλήθους δειγμάτων για το Custom Signal

Στα επόμενα διαγράμματα παρατίθενται τα αποτελέσματα.



Σχήμα 4.35 Αποτελέσματα κατανάλωσης MSF SQUARER σε CS μορφή στα 1.1ns



Σχήμα 4.36 Αποτελέσματα κατανάλωσης MSF SQUARER σε δυαδική μορφή στα 1.6ns

Από το Σχήμα 4.35 βλέπουμε πως ο MSF Squarer έχει χαμηλότερη κατανάλωση σε κάθε μία από τις εξεταζόμενες περιπτώσεις. Στον αλγόριθμο MSF Squarer τα υψηλότερης αξίας bits της εισόδου A, αν είναι μηδενικά θα είναι μηδενικές οι τελευταίες σειρές από bits οι οποίες είναι και οι πολυπληθέστερες με αποτέλεσμα να έχουμε χαμηλότερο switching activity ενώ ο αλγόριθμος Booth Folding έχει ως πολυπληθέστερα μερικά γινόμενα αυτά που πολλαπλασιάζονται με τα κωδικοποιημένα χαμηλότερης αξίας bits της εισόδου A. Έτσι εξηγείται η χαμηλότερη κατανάλωση για σήματα φωνής 8, 16, 24-bit και για την περίπτωση του Custom Signal. Για την περίπτωση που η είσοδος είναι μία αλληλουχία τυχαίων δυαδικών αριθμών η κατανάλωση είναι πάλι μικρότερη λόγω της χαμηλότερης επιφάνειας του κυκλώματος το οποίο εν μέρει οφείλεται και στο ότι τα μερικά γινόμενα στον MSF Squarer είναι πάντα θετικά άρα δεν χρειάζονται διορθωτικές μονάδες που όμως χρειάζονται στον Booth Folding.

Όταν τα αποτελέσματα των τετραγωνιστών είναι σε δυαδική μορφή (Σχήμα 4.36), βλέπουμε πως ο MSF Squarer συνεχίζει να διατηρεί το πλεονέκτημα του ως προς την κατανάλωση έναντι του Booth Folding για σήματα φωνής 8, 16, 24-bit και για την περίπτωση του Custom Signal. Στην περίπτωση όμως του τυχαίου σήματος το αποτέλεσμα είναι το ίδιο σχεδόν με του Booth Folding γιατί υπάρχει μεγαλύτερη επιφάνεια σε σχέση με τον συγκρινόμενο αλγόριθμο. Οι απλοποιήσεις που γίνονται από τον Design Compiler με την προσθήκη του Parallel-Prefix adder είναι διαφορετικές σε κάθε ένα από τα δύο κυκλώματα με αποτέλεσμα να έχουμε μεγαλύτερη επιφάνεια στον MSF Squarer.

5

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΕΠΕΚΤΑΣΕΙΣ

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

5.1 Σύνοψη και συμπεράσματα

Συνοψίζοντας την μελέτη της διπλωματικής, μελετήσαμε δύο διαφορετικών ειδών πολλαπλασιαστές. Όλοι ήταν παράλληλοι πολλαπλασιαστές wallace αλλά η φιλοσοφία του πρώτου είδους πολλαπλασιασμού είναι ο χωρισμός σε blocks για να επιτύχουμε καλύτερα αποτελέσματα σε DSP εφαρμογές ενώ η φιλοσοφία του δεύτερου ήταν μια διαφορετική κωδικοποίηση επονομαζόμενη Most Significant First (MSF) όπου κωδικοποιούνται πρώτα τα bit υψηλότερης αξίας. Συνολικά μελετήθηκαν 3 block αλγόριθμοι (με κάποιους από αυτούς να έχουν υλοποιηθεί με δύο τρόπους) και 2 MSF αλγόριθμοι, ένας πολλαπλασιαστής και ένας τετραγωνιστής.

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

1. Οι αλγόριθμοι TWO, 2encA (και συγκεκριμένα η δεύτερη υλοποίηση τους) καθώς και ο αλγόριθμος MSF Multiplier εμφανίζουν πολύ καλά αποτελέσματα όταν και οι δύο είσοδοι τους είναι σήματα που εμφανίζουν από λίγα έως πολλά μηδενικά ή μονάδες στα υψηλότερης αξίας bit. Άρα συμπεραίνουμε πως θα μπορούσαν να χρησιμοποιηθούν ως πυρήνες σε συστήματα που πραγματοποιούν σχεδόν αποκλειστικά συνελίξεις (convolutions) όπου πραγματοποιείται ο πολλαπλασιασμός ενός σήματος με την τιμή που είχε πριν από κάποιο χρονικό διάστημα.

2. Ο αλγόριθμος MSF Squarer παρουσιάζει καλύτερα αποτελέσματα από κάθε άλλο τετραγωνιστή για αποτέλεσμα της μορφής carry-save αλλά για binary και η απόδοση του μπορεί να βελτιστοποιηθεί ακόμα περισσότερο με την προσθήκη ενός διαφορετικού CLA τελικού αθροιστή όπου θα φαίνεται το πλεονέκτημα που υπάρχει σε carry-save μορφή ακόμα και για τυχαίες τιμές. Παρά ταύτα είναι δυνατή και η εφαρμογή του τετραγωνιστή σε εφαρμογές με συντεταγμένες όπου συνήθως υπολογίζεται η παράσταση $x^2+y^2+z^2$ άρα οι επιμέρους υπολογισμοί γίνονται σε carry-save μορφή. Ο αλγόριθμος αυτός έχει καλύτερα αποτελέσματα πάντοτε για σήματα που εμφανίζουν από λίγα έως πολλά μηδενικά ή μονάδες στα υψηλότερης αξίας bit και μπορεί να χρησιμοποιηθεί σε εφαρμογές που χειρίζονται τέτοια σήματα. Τελευταία και ίσως σημαντικότερη εφαρμογή του MSF Squarer είναι στην κρυπτογραφία και ειδικότερα σε συστήματα που χρησιμοποιούν τον αλγόριθμο RSA.

5.2 Μελλοντικές επεκτάσεις

1. Από την φύση της κωδικοποίησης του MSF multiplier γνωρίζουμε πως είναι αναδρομικός. Στην υλοποίηση που παρουσιάσαμε έχουμε εξαντλήσει την αναδρομή μέχρι και το τελευταίο βήμα της. Μία ιδέα προς επέκταση της παρούσας διπλωματικής θα ήταν να σταματήσει η αναδρομή σε ένα συγκεκριμένο σημείο και στην συνέχεια ο υπόλοιπος πολλαπλασιασμός να πραγματοποιηθεί σε ένα block.

   Για παράδειγμα, όπως είχαμε αναφέρει στο $4^{ο}$ κεφάλαιο ισχύει για X και Y 8-bit αριθμούς:

   $X \cdot Y = 2^6 Z_6 + X_6 \cdot Y_6$ όπου $Z_6 = \mathbf{x}_{7,6} Y_6 + \mathbf{y}_{7,6} X$

   Παρομοίως ο όρος $X_6 \cdot Y_6$ μπορεί να εκφραστεί ως:

   $X_6 \cdot Y_6 = 2^4 \mathbf{x}_{5,4} Y_6 + 2^4 \mathbf{y}_{5,4} X_4 + X_4 \cdot Y_4 = 2^4 Z_4 + X_4 \cdot Y_4$

   Άρα μπορούμε το κομμάτι που παραμένει, δηλαδή το $X_4 \cdot Y_4$ να το υπολογίσουμε με έναν block αλγόριθμο.
2. Μία δεύτερη μελλοντική επέκταση της διπλωματικής και συγκεκριμένα του MSF Multiplier είναι η εφαρμογή του αλγορίθμου αυτού στους truncated πολλαπλασιαστές, όπου μας ενδιαφέρει μια εκτίμηση του αποτελέσματος η οποία δεν χρειάζεται να είναι ακριβής. Αυτό συμβαίνει γιατί για πολλές εφαρμογές δεν χρειάζεται τόσο μεγάλη ακρίβεια στις πράξεις γιατί από ένα σημείο και μετά η πλεονάζουσα λεπτομέρεια δεν γίνεται αισθητή. Έτσι μπορεί να μειωθεί δραματικά το κύκλωμα για τα bit χαμηλότερης αξίας και έτσι η κατανάλωση και μειωθεί αισθητά.
3. Οι υλοποιήσεις των πολλαπλασιαστών μπορούν να γίνουν με διαφορετικούς encoders/decoders όπως για παράδειγμα έγινε στην δεύτερη υλοποίηση του MSF Squarer.

6

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Neil H. E. Weste, David F. Harris, **"CMOS VLSI design: a circuits and systems perspective",Third Edition**, pp. 698-705.

[2] Kuan-Hung Chen, Yuan-Sun Chu, **"A Low-Power Multiplier With the Spurious Power Suppression Technique",** IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 15, NO.7, pp. 846-850, July 2007.

[3] Antonio G. M. Strollo and Davide De Caro, **"Booth Folding Encoding for High Performance Squarer Circuits"**, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS – II: ANALOG AND DIGITAL SIGNAL PROCESSING, VOL. 50, NO.5, AMY 2003

[4] Dimitris Bekiaris, George Economakos, Kiamal Z. Pekmestzi, **"A High-Speed Radix-16 Array Multiplier".**

[5] Kiamal Pekmestzi, "**DIGITAL VLSI SYSTEMS**", NTUA Lectures Notes, Athens 2003.

[6] Synopsys Design Compiler, www.synopsys.com.

[7] ModelSim Corporation, www.model.com.

[8[Synopsys Primetime PX, www.synopsys.com.