ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

Συγκριτική Ανάλυση αριθμητικών κυκλωμάτων Πολλαπλασιαστών σε Επαναπρογραμματιζόμενες Μονάδες Υλικού

Δημήτριος Κουτσογεώργος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Αντώνιος Δαδαλιάρης

Λαμία, 2023

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

Comparative Analysis of numerical circuits in Reprogrammable Hardware Units

Dimitrios Koutsogeorgos

Master thesis
Supervisot
Antonios Dadaliaris

Lamia, 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ

Συγκριτική Ανάλυση αριθμητικών κυκλωμάτων Πολλαπλασιαστών σε Επαναπρογραμματιζόμενες Μονάδες Υλικού

Δημήτριος Κουτσογεώργος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Αντώνιος Δαδαλιάρης

4

Λαμία, 2023

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο [«τίτλος εργασίας»] αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ
Δημήτριος Κουτσογεώργος

Ημερομηνία
13/07/2023

Υπογραφή

Συγκριτική Ανάλυση αριθμητικών κυκλωμάτων
Πολλαπλασιαστών σε Επαναπρογραμματιζόμενες Μονάδες Υλικού

Δημήτριος Κουτσογεώργος

Τριμελής Επιτροπή:

Ονοματεπώνυμο, Μαρία Κοζύρη – Αναπληρωτής Καθηγήτρια

Ονοματεπώνυμο, Γεώργιος Δημητρίου – Επίκουρος Καθηγητής

Ονοματεπώνυμο, Αντώνιος Δαδαλιάρης – Επίκουρος Καθηγητής

Επιστημονικός Σύμβουλος:

Ονοματεπώνυμο, Αντώνιος Δαδαλιάρης

Αριθμός Διπλωματικής Εργασίας: (γραμματείας)

ΤΙΤΛΟΣ: " ΣΥΓΚΡΙΤΙΚΗ ΑΝΑΛΥΣΗ ΑΡΙΘΜΗΤΙΚΩΝ ΚΥΚΛΩΜΑΤΩΝ ΠΟΛΛΑΠΛΑΣΙΑΣΤΩΝ ΣΕ ΕΠΑΝΑΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΕΣ ΜΟΝΑΔΕΣ ΥΛΙΚΟΥ"

Φοιτητής: Δημήτριος Κουτσογεωργος
Επιβλέπων: Αντώνιος Δαδαλιάρης

Περίληψη

Στην εποχή μας, οι περισσότεροι επεξεργαστές απαιτούν πολύ υψηλές ταχύτητες λειτουργίας. Αριθμητικές πράξεις όπως πρόσθεση, αφαίρεση και πολλαπλασιασμός χρησιμοποιούνται σε διάφορα ψηφιακά κυκλώματα για την επιτάχυνση της υπολογιστικής διαδικασίας. Ο πολλαπλασιασμός αποτελεί μια από τις σημαντικότερες αριθμητικές πράξεις σε εφαρμογές επεξεργασίας σήματος, όπως για παράδειγμα στην ψηφιακή επεξεργασία σήματος.

Η πράξη του πολλαπλασιασμού αποτελεί περιοριστικό παράγοντα στη ταχύτητα επεξεργασίας. Για τον λόγο αυτό αναπτύσσονται όλο και περισσότερες μέθοδοι μείωσης του αριθμού των βημάτων της διαδικασίας του πολλαπλασιασμού. Στο σχεδιασμό κάθε ψηφιακού κυκλώματος οι τρεις βασικές παράμετροι που καθορίζουν την απόδοση είναι η ταχύτητα, η ισχύς και ο χώρος που καταλαμβάνουν οι λογικές πύλες. Σχετικά με τις επιμέρους διεργασίες ενός ψηφιακού συστήματος, η πράξη του πολλαπλασιασμού απαιτεί μεγάλο χρόνο εκτέλεσης καθιστώντας τον βασικό παράγοντα περάτωσης μιας λειτουργίας. Επιπλέον η υλοποίηση ενός κυκλώματος πολλαπλασιασμού απαιτεί μεγάλο αριθμό λογικών πυλών και ως αποτέλεσμα έχει και τη μεγαλύτερη κατανάλωση ισχύος. Για όλους τους

παραπάνω λόγους ο σχεδιασμός κυκλωμάτων πολλαπλασιασμού που βελτιώνουν έναν η περισσότερους από τους σχεδιαστικούς στόχους της υψηλής ταχύτητας, της κατανάλωσης και του αριθμού των λογικών πυλών, αποτελεί προτεραιότητα.

Η παρούσα διπλωματική εργασία έχει ως στόχο την προσομοίωση της διαδικασίας «σύνθεσης» (“Synthesis”) και «υλοποίησης» (“Implementation”) διάφορων κυκλωμάτων πολλαπλασιασμού σε επιτόπια συστοιχία προγραμματιζόμενων πυλών (FPGA). Για την προσομοίωση επιλέχθηκαν διάφοροι αλγόριθμοι πολλαπλασιασμού των 8bit, 16bit, 32bit και 64bit. Ως περιβάλλον προσομοίωσης χρησιμοποιήθηκε το Vivado Design Suite στο οποίο έγιναν διάφοροι συνδυασμοί διαδικασιών σχεδίασης “Synthesis” και “Implementation”. Οι παραπάνω διαδικασίες σχεδίασης προσομοιώθηκαν στο evaluation board Artix - 7 FPGA AC701 το οποίο είναι μια έτοιμη πλατφόρμα ανάπτυξης ψηφιακών κυκλωμάτων βασισμένη στο πιο πρόσφατο Artix – 7 FPGA της Xilinx, το XC7A200T-2FBG676C.

Τα αποτελέσματα της προσομοίωσης θα είναι οι συγκρίσεις των κυκλωμάτων πολλαπλασιασμού τόσο ως προς τους χρόνους υλοποίησης των διαδικασιών σχεδίασης “Synthesis” και “Implementation” όσο και ως προς την κατανάλωση ισχύος, τον αριθμό των λογικών πυλών καθώς και το συσχετισμό όλων των παραπάνω μεταξύ τους. Οι συγκρίσεις αυτές θα γίνουν τόσο σε επίπεδο μεμονωμένων αλγορίθμων όσο και σε επίπεδο αλγορίθμων πολλαπλασιασμού ίδιας λογικής αλλά διαφορετικών bit.

ΠΡΟΛΟΓΟΣ

Η πράξη του πολλαπλασιασμού αποτελεί ένα από τα βασικά στοιχεία της ψηφιακής επεξεργασίας σήματος (Digital Signal Processing) (DSP) καθώς χρησιμοποιείται σε εφαρμογές όπως ο μετασχηματισμός Fourier στο διακριτό χρόνο, ο «γρήγορος» μετασχηματισμός Fourier, η συνέλιξη σημάτων και τα ψηφιακά φίλτρα. Λόγω λοιπόν της ευρείας χρήσης κυκλωμάτων πολλαπλασιασμού υπάρχει συνεχής έρευνα για τη βελτιστοποίηση των προκαθορισμένων αλγορίθμων πολλαπλασιασμού και την ανάπτυξη νέων προσεγγίσεων υλοποίησης για αποδοτικότερη αρχιτεκτονική. Σήμερα υπάρχουν πολλοί αλγόριθμοι για τη διενέργεια της πράξης του πολλαπλασιασμού κάποιοι από τους οποίους είναι: ο Booth, ο Wallace Tree, ο Dadda, ο array, ο serial και ο Vedic. Κάποιοι από τους παραπάνω αλγορίθμους μπορούν επίσης να τροποποιηθούν αλλάζοντας τον αλγόριθμο που χρησιμοποιούν για τη διενέργεια των μερικών αθροισμάτων τους προκύπτοντας μια μεγάλη ποικιλία διαφορετικών αλγορίθμων πολλαπλασιασμού.

Κάποιοι από αυτούς τους αλγορίθμους πολλαπλασιασμού επιλέχθηκαν για τη παρούσα διπλωματική εργασία. Σκοπός της εργασίας είναι η προσομοίωση της διαδικασίας «Synthesis» και «Implementation» των αλγορίθμων αυτών σε FPGA ώστε να μελετηθούν οι διάφορες αλληλεπιδράσεις των διαδικασιών σχεδίασης (design flows) και των αλγορίθμων πολλαπλασιασμού.

Στο Κεφάλαιο 1 θα αναλυθούν οι αλγόριθμοι πολλαπλασιασμού που χρησιμοποιήθηκαν στη παρούσα διπλωματική εργασία ώστε να υπάρξει μια πιο εμπεριστατωμένη εικόνα πάνω στον τρόπο που λειτουργούν οι αλγόριθμοι αυτοί.

Στο Κεφάλαιο 2 θα γίνει ανάλυση του περιβάλλοντος προσομοίωσης Vivado Design Suite με τη βοήθεια του οποίου έγιναν όλες οι απαραίτητες προσομοίωσης των διαδικασιών σχεδίασης «Synthesis» και «Implementation» και από το οποίο εξάχθηκαν όλες οι απαραίτητες πληροφορίες εξαγωγής των αποτελεσμάτων της παρούσας διπλωματικής εργασίας.

Στο Κεφάλαιο 3 θα παρουσιαστούν τα αποτελέσματα των συγκρίσεων των αλγορίθμων πολλαπλασιασμού και των διαδικασιών σχεδίασης «Synthesis» και «Implementation».

Στο Κεφάλαιο 4 θα παρουσιαστούν τα συμπεράσματα των παραπάνω συγκρίσεων και θα προταθούν τρόποι εξέλιξης της παρούσας έρευνας καθώς και νέες προοπτικές που αυτή δημιουργεί.

ΕΥΧΑΡΙΣΤΙΕΣ

Οι λέξεις είναι λίγες για να περιγράψουν την ευγνωμοσύνη μου προς τον Καθηγητή και επιβλέποντα της Διπλωματικής μου Εργασίας κ. Αντώνιο Δαδαλιάρη. Το αστείρευτο ενδιαφέρον του και η μεταδοτικότητά του τόσο στις παραδόσεις του μαθήματος Ανάπτυξης Τηλεπικοινωνιακών Συστημάτων σε Υλικό, όσο και στις πολύωρες συζητήσεις μας κατά τη διάρκεια εκπόνησης της Διπλωματικής Εργασίας, με καθοδήγησαν σωστά και σταθερά. Οι γνώσεις του αποτέλεσαν και συνεχίζουν να αποτελούν πηγή θαυμασμού και έμπνευσης για μένα.

Πολλές ευχαριστίες αρμόζουν στο συνάδελφο κ. Αντώνη Χορμόβα για τις πολύτιμες συμβουλές που μου προσέφερε σε όλα τα στάδια εκπόνησης της παρούσας διπλωματικής εργασίας.

Θα ήθελα επίσης να ευχαριστήσω θερμά τη σύντροφό μου για την αμέριστη ψυχολογική και ηθική στήριξη που μου προσέφερε κατά τη διάρκεια εκπόνησης της διπλωματικής εργασίας καθώς χωρίς την πίστη της στις δυνατότητες μου και την κατανόηση της τίποτα από αυτά δε θα ήταν εφικτό.

Τέλος, πολλές ευχαριστίες αρμόζουν στους φίλους μου για την υπομονή και κατανόηση που έδειξαν στη προσπάθεια μου για την εκπόνηση της παρούσας διπλωματικής εργασίας.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΥΠΟΜΝΗΜΑ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1

ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ

1.1 ΤΥΠΟΙ ΑΛΓΟΡΙΘΜΩΝ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ

1.1.1 BOOTH [1]

Ο αλγόριθμος πολλαπλασιασμού Booth πολλαπλασιάζει δυο προσημασμένους αριθμούς κάνοντας χρήση του συμπληρώματος ως προς δύο. Ο Booth έχει τη δυνατότητα να εκτελεί λιγότερες προσθέσεις και αφαιρέσεις σε σύγκριση με τον βασικό αλγόριθμο πολλαπλασιασμού (shift –and – add). Πρόκειται για μια διαδικασία κωδικοποίησης που ελαχιστοποιεί τον αριθμό των μερικών παραγόντων της διαδικασίας του πολλαπλασιασμού. Η κωδικοποίηση αυτή βασίζεται στη σχέση:

$$X = -x_{n-1} + \sum_{i=0}^{n-2} 2^i \times x_i$$

Η παραπάνω σχέση μπορεί να γραφεί και ισοδύναμα ως:

$$X = 2 \times X - X$$

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

Όπου: $z_0 = x_0 - x_1$

$z_2 = x_1 - x_2$

.

.

.

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

Αναλύοντας το ψηφίο z_{n-1} προκύπτει η παρακάτω σχέση:

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

Ο αλγόριθμος πολλαπλασιασμού Booth εξετάζει διαδοχικά τα N-bit ψηφιά του πολλαπλασιαστή Y σε παράσταση συμπληρώματος ως προς δύο ανεξάρτητα αν αυτοί είναι θετικοί ή αρνητικοί αριθμοί. Αν υποθέσουμε το γινόμενο $P = X \times Y$ όπου $Y = y_{n-1}y_{n-2} \dots y_1y_0$ και $X = x_{n-1}x_{n-2} \dots x_1x_0$ τότε σύμφωνα με τον αλγόριθμο πολλαπλασιασμού Booth ισχύει η σχέση:

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

Σύμφωνα με τη παραπάνω σχέση το X πολλαπλασιάζεται με ένα από τα στοιχεία του συνόλου {-1, 0, 1} σε κάθε βήμα, σύμφωνα με το αποτέλεσμα της αφαίρεσης των δυο διαδοχικών ψηφίων του πολλαπλασιαστή Y. Με τη κωδικοποίηση Booth ελέγχουμε τις ομάδες μονάδων που παρουσιάζονται μέσα σε ένα διάδικό αριθμό. Η διαδικασία της κωδικοποίησης αρχίζει ένα bit πριν τη διαδικασία του αθροίσματος, δηλαδή αν η άθροιση ξεκινά από το y_0 η κωδικοποίηση αρχίζει από το y_{-1}. Ύστερα, με φορά από τα δεξιά προς τα αριστερά εξετάζονται τα ψηφία ανά δύο. Μεταξύ των ψηφιακών ζευγών που εξετάζονται υπάρχει επικάλυψη με αποτέλεσμα μόνο ένα bit του πολλαπλασιαστή να μένει εκτός. Το αποτέλεσμα της κωδικοποίησης παρουσιάζεται στονΠίνακας 1.1.

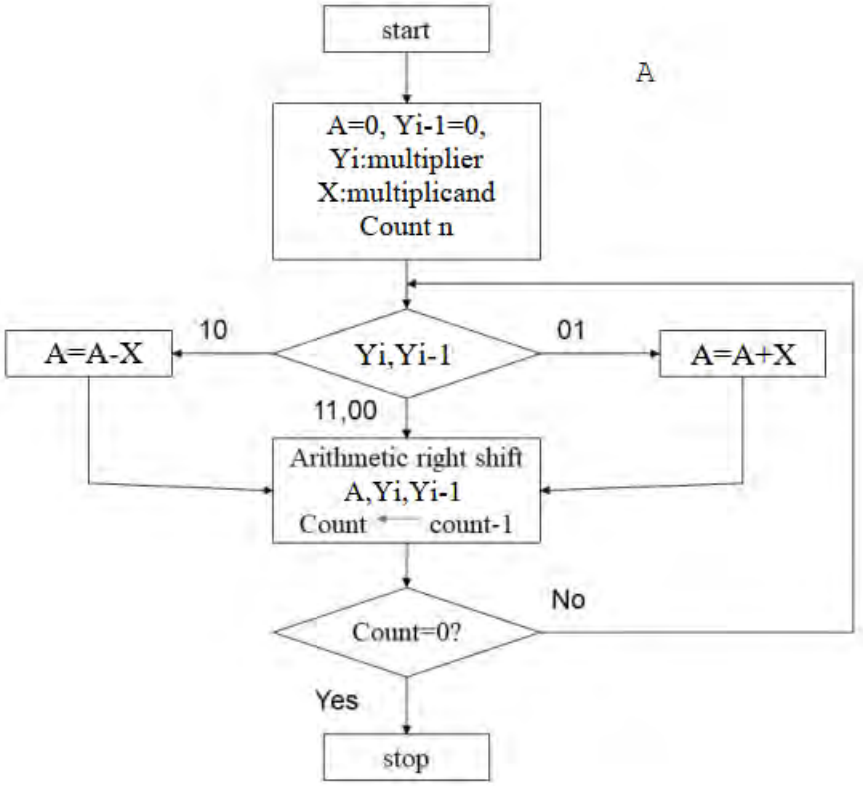
Πίνακας 1.1 Πίνακας κωδικοποίησης Radix-4

$\mathbf{Y_i}$	$\mathbf{Y_{i-1}}$	**Κωδικοποίηση Booth** $(Y_{i-1} - Y_i)$	**Λειτουργία**
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

Τα βήματα της λειτουργίας του αλγορίθμου Booth συνοψίζονται στα ακόλουθα:

- Αν $Y_iY_{i-1} = 00$ ή $Y_iY_{i-1} = 11$ τότε ολίσθησε το άθροισμα μερικών γινομένων κατά ένα bit προς τα αριστερά

- Αν $Y_i Y_{i-1} = 10$ τότε αφαίρεσε τον πολλαπλασιαστή X από το τρέχον άθροισμα των ενδιάμεσων γινομένων και μετά ολίσθησε το αποτέλεσμα κατά μια θέση προς τα αριστερά.
- Αν $Y_i Y_{i-1} = 01$ τοτε αφαίρεσε τον πολλαπλασιαστή X από το τρέχον άθροισμα των ενδιάμεσων γινομένων και μετά ολίσθησε το αποτέλεσμα κατά μια θέση προς τα αριστερά.
- Το Y_{i-1} είναι ένα bit και αποτελεί αρχική συνθήκη του αλγορίθμου
- Το τρέχον άθροισμα των μερικών γινομένων A αρχικοποιείται στην αρχή του αλγορίθμου και έχει τον ίδιο αριθμό bits με τον πολλαπλασιαστή και τον πολλαπλασιαστέο
- Ο αριθμός των επαναλήψεων του αλγορίθμου εξαρτάται από τον αριθμό των bits του πολλαπλασιαστή και του πολλαπλασιαστέου.



Εικόνα 1.1 Λογικό διάγραμμα αλγόριθμου πολλαπλασιασμού Booth

Τα κύρια πλεονεκτήματα του αλγορίθμου πολλαπλασιασμού Booth είναι:

- Η κωδικοποίηση Booth είναι κοινή για προσημασμένους και μη προσημασμένους αριθμούς.
- Το κάθε ψηφίο του κωδικοποιημένου αριθμού μπορεί να παραχθεί άμεσα χωρίς εξάρτηση από προηγούμενα μερικά γινόμενα μιας και είναι συνάρτηση μόνο των $Y_i Y_{i-1}$
- Με την κωδικοποίηση του δυαδικού αριθμού μπορούμε να μειώσουμε τον αριθμό των προσθέσεων που απαιτούνται για τον πολλαπλασιασμό.

Τα σημαντικότερα μειονεκτήματα του αλγόριθμου Booth είναι:

- Η πολυπλοκότητά του: Η λογική του είναι αρκετά πιο πολύπλοκη στη κατανόηση και την υλοποίηση σε σχέση με τους παραδοσιακούς αλγορίθμους.
- Η μεγάλή καθυστέρηση μετάδοσης της πληροφορίας: Ο αλγόριθμος απαιτεί μεγάλο αριθμό επαναλήψεων για να υπολογίσει το γινόμενο ενός πολλαπλασιασμού, γεγονός που αυξάνει το χρόνο μετάδοσης της πληροφορίας.
- Η μεγάλη κατανάλωση: Η υλοποίηση του αλγορίθμου καταναλώνει μεγαλύτρα ποσα ισχύος σε σχέση με τους παραδοσιακού αλγορίθμους, ειδικά για μεγαλύτερο αριθμό bits εισόδου.

1.1.2 WALLACE TREE [2]

Ο αλγόριθμος Wallace Tree είναι μια αποδοτική υλοποίηση υλικού ενός ψηφιακού κυκλώματος που πολλαπλασιάζει δύο ακέραιους αριθμούς (Εικόνα 1.2). Ο αλγόριθμος Wallace Tree αποτελείται από τα παρακάτω βήματα:

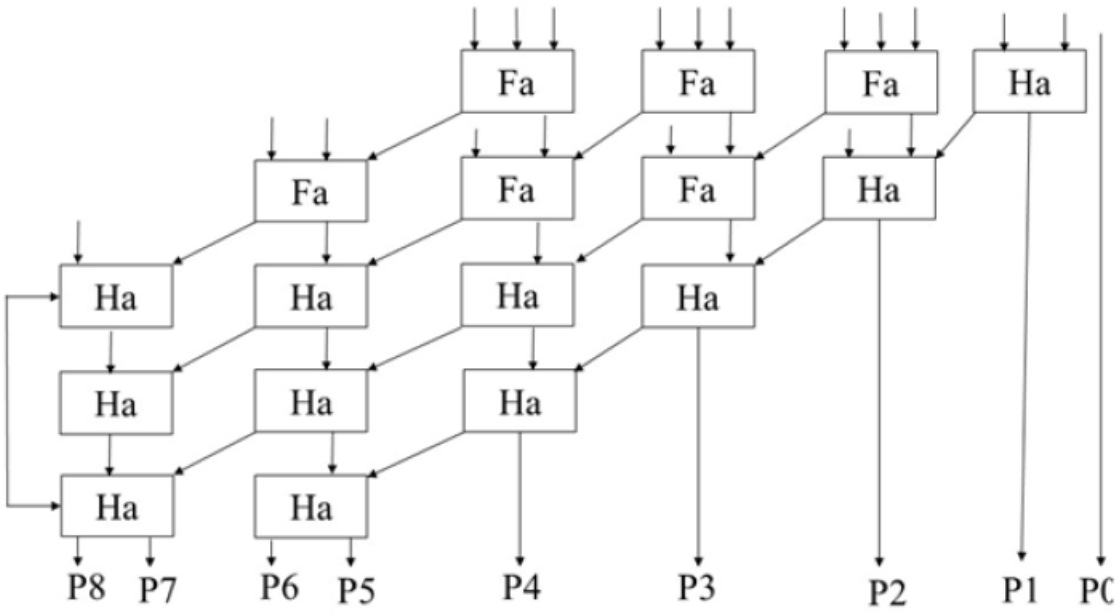
1. Πολλαπλασιάζουμε κάθε ψηφίο από το ένα αριθμητικό σύνολο με κάθε ψηφίο από το άλλο αριθμητικό σύνολο παράγοντας έτσι n^2 αποτελέσματα. Ανάλογα με τη θέση των μερικών παραγόντων τα καλώδια διασύνδεσης φέρουν διαφορετικά βάρη, για παράδειγμα το καλώδιο που μεταφέρει το αποτέλεσμα $a_2 b_3$ έχει βάρος 32.
2. Ακολουθεί η μείωση του αριθμού των μερικών γινομένων σε δύο επίπεδα με χρήση κυκλωμάτων full και half adders

3. Τέλος ακολουθεί η ομαδοποίηση όλων των καλωδιώσεων σε δύο αριθμούς οι οποίοι προστίθενται με συμβατική πρόσθεση.

				a_3	a_2	a_1	a_0
				$x\ b_3$	b_2	b_1	b_0
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Εικόνα 1.2 Πολλαπλασιασμός 4x4 bits με χρήση αλγορίθμου Wallace Tree

Η Εικόνα 1.3 δείχνει την άθροιση των μερικών γινομένων σε έναν αλγόριθμο πολλαπλασιασμού Wallace tree 4x4 bits. Στο πρώτο στάδιο τα μερικά γινόμενα των τριών πρώτων σειρών αθροίζονται μεταξύ τους με τη βοήθεια αθροιστών CSA (Carry Save Adder) και λαμβάνονται ξεχωριστά τα αθροίσματα και τα υπόλοιπα τους. Οι μερικές αυτές αθροίσεις με τα υπόλοιπα τους προστίθενται με την τελευταία σειρά στο δεύτερο στάδιο του αλγορίθμου. Τέλος η διαδικασία ολοκληρώνεται στο τελικό στάδιο κατά το οποίο δεν προκύπτει κάποιο κρατούμενο.



Εικόνα 1. 3 Διαδικασία άθροισης μερικών γινομένων αλγορίθμου Wallace Tree

Γενικά, ένας n-bit αλγόριθμος πολλαπλασιασμού Wallace Tree περιλαμβάνει n+1 στάδια, δηλαδή για το παραπάνω παράδειγμα του 4x4 bits πολλαπλασιασμού απαιτούνται 5 στάδια. Η χρήση CSA αθροιστή βελτιώνει την καθυστέρηση μετάδοσης μιας και το κάθε στάδιο είναι ανεξάρτητο από τα προηγούμενα κρατούμενα.

Παίρνοντας οποιεσδήποτε τρείς καλωδιώσεις με το ίδιο βάρος και εισάγοντάς τες σε έναν full adder το αποτέλεσμα θα είναι μια καλωδίωση ίδιου βάρους με τις καλωδιώσεις εισόδου και μια καλωδίωση μεγαλύτερου βάρους από τις τρεις καλωδιώσεις εισόδου. Αν προκύψουν δύο εναπομείνασες καλωδιώσεις ίδιου βάρους το αποτέλεσμα τους προκύπτει από την άθροιση τους μέσω ενός ημιαθροιστή. Τέλος, αν απομένει μόνο μια καλωδίωση, αυτή συνδέεται με το επόμενο επίπεδο.

Ο αλγόριθμος πολλαπλασιασμού Wallace Tree έχει τα εξής πλεονεκτήματα:

1. Κάθε επίπεδο του δέντρου μειώνει τον αριθμό των διανυσμάτων κατά έναν παράγοντα 3:2.

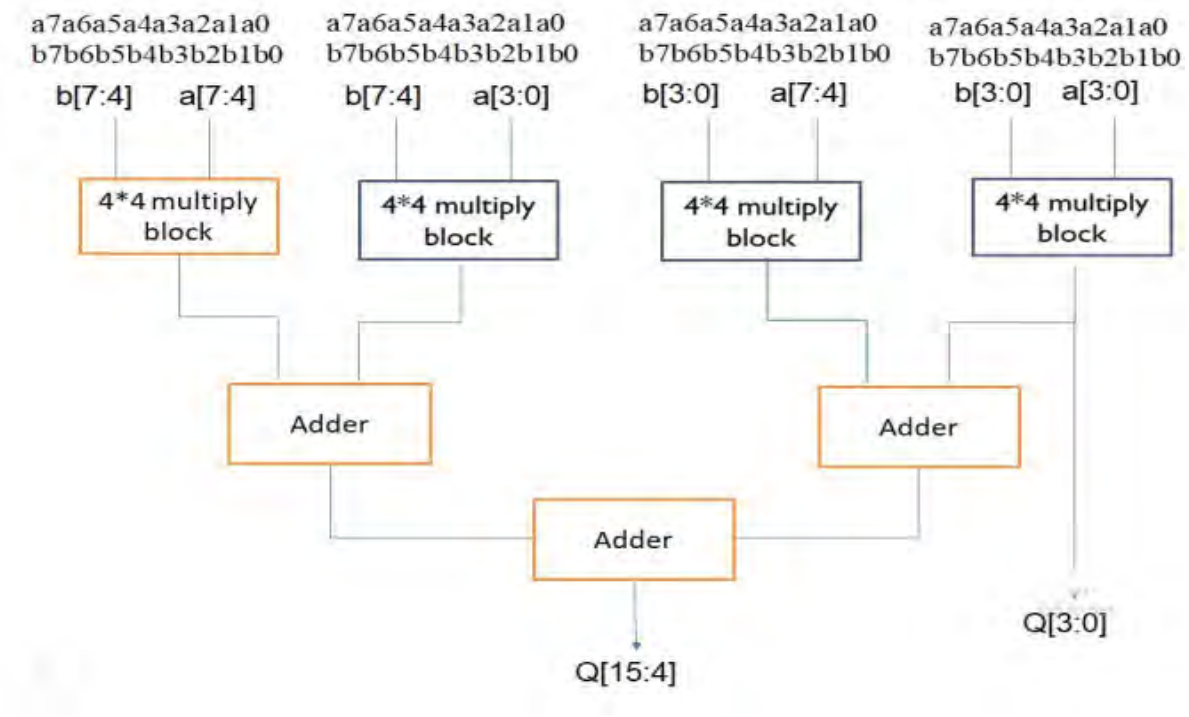
2. Προσφέρει την ελάχιστη καθυστέρηση μετάδοσης της πληροφορίας.
3. Ο αλγόριθμος Wallace Tree παρέχει $O(\log n)$ χρόνο καθυστέρησης μετάδοσης πληροφορίας, αλλά προσθέτοντας τα μερικά γινόμενα με συμβατικούς αλγόριθμους άθροισης θα απαιτούσε $O(log^2 n)$ χρόνο.

Ο αλγόριθμος πολλαπλασιασμού Wallace Tree έχει τα εξής μειονεκτήματα:

1. Δεν παρέχει κανένα πλεονέκτημα εναντι δέντρων ripple adder στα περισσότερα FPGA.
2. Λόγω της δομής διασύνδεσης ο αριθμός των αθροιστών αυξάνεται μαζί με το πλήθος των bit που πρέπει να πολλαπλασιαστούν.

1.1.3 VEDIC [3]

Ο αλγόριθμος πολλαπλασιασμού Vedic είναι ο παραδοσιακός αλγόριθμος πολλαπλασιασμού δύο αριθμών στο δεκαδικό σύστημα. Η ίδια λογική ακολουθείται και στο δυαδικό σύστημα. Είναι μια γενική μέθοδος πολλαπλασιασμού που έχει εφαρμογή σε όλες τις περιπτώσεις. Το όνομα Vedic εσωκλείει τον τρόπο που λειτουργεί ο αλγόριθμος δηλαδή «Κατακόρυφα και Διαγώνια»(«Vertical and Crosswise»). Βασίζεται στην ιδέα πως όλα τα μερικά γινόμενα μπορούν να γενικευτούν για πολλαπλασιασμούς nxn bits. Εφόσον τα μερικά αθροίσματα και γινόμενα υπολογίζονται παράλληλα η ταχύτητα υλοποίησης του αλγορίθμου είναι ανεξάρτητη από την ταχύτητα του επεξεργαστή. Η υπολογιστική ισχύς του αλγορίθμου αυξάνεται μαζί με το μέγεθος των διαύλων εισόδου και εξόδου του. Λόγω της απλής δομής του μπορεί εύκολα να ενσωματωθεί σε οποιοδήποτε chip. Ο αλγόριθμος πολλαπλασιασμού Vedic έχει το πλεονέκτημα πως, καθώς ο αριθμός των bits πολλαπλασιασμού αυξάνεται, η καθυστέρηση των λογικών πυλών και η επιφάνεια που απαιτεί αυξάνονται με πολύ αργούς ρυθμούς σε σύγκριση με άλλους συμβατικούς πολλαπλασιαστές. Ο 8bit αλγόριθμος πολλαπλασιασμού Vedic φαίνεται στην Εικόνα 1.4.



Εικόνα 1.4 Αλγόριθμος πολλαπλασιασμού Vedic για 8x8 bits

Μερικά από τα πλεονεκτήματα του αλγόριθμου πολλαπλασιασμού Vedic είναι:

1. Ο χρόνος διάδοσης της πληροφορίας είναι ανεξάρτητος από τον χρόνο λειτουργίας του επεξεργαστή καθώς τα μερικά γινόμενα και τα αθροίσματά τους υπολογίζονται παράλληλα.
2. Συγκριτικά με άλλους αλγόριθμους, έχει μικρή σχετικά κατανάλωση για πολλαπλασιασμούς 8bits και 16bits οι οποίοι χρησιμοποιούν συγκριτικά λιγότερες πύλες.
3. Καθώς ο αριθμός των απαιτούμενων πυλών αυξάνεται, η περιοχή που χρειάζεται αυξάνεται σχετικά αργά σε σχέση με άλλους αλγορίθμους, γεγονός που τον καθιστά χωρικά αποδοτικό.
4. Η καθυστέρηση μετάδοσης αυξάνεται αργά καθώς τα bits εισόδου αυξάνονται.

Τα πιο σημαντικά μειονεκτήματα του αλγορίθμου Vedic είναι:

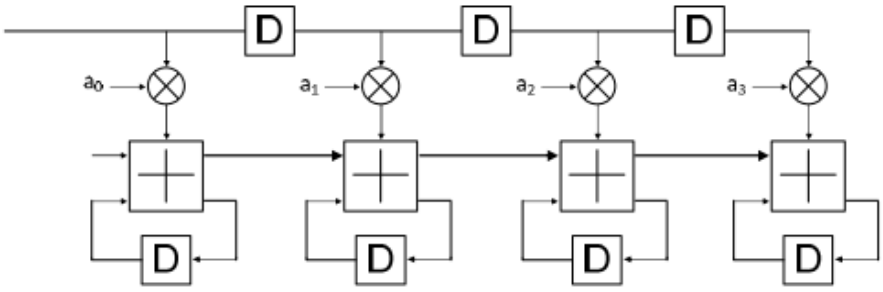
1. Λόγω της δομής του, επιβαρύνεται από τη καθυστέρηση της μετάδοσης του κρατουμένου των αθροιστών, φαινόμενο που εντείνεται όσο μεγαλώνει ο αριθμός των bits πολλαπλασιασμού.
2. Όταν ο αριθμός των bits ξεπερνά τα 32bits ή 64bits η καθυστέρηση μετάδοσης της πληροφορίας αυξάνεται σημαντικά.

1.1.4 SERIAL [4]

1.1.4.1 Serial – Parallel

1.1.4.1.1 Τύπος 1ος

Ο πρώτος τύπος σειριακού πολλαπλασιαστή αποτελείται από τέσσερεις full adders, τέσσερις πύλες AND για τη παραγωγή μερικών γινομένων και τέσσερα στοιχεία καθυστέρησης (Εικόνα 1.5). Στην παρούσα δομή το ψηφίο μεταφοράς (carry out bit) κάθε αθροιστή ανατροφοδοτείται στον ίδιο αθροιστή ως σήμα εισόδου μετά από τον παλμό ενός ρολογιού.

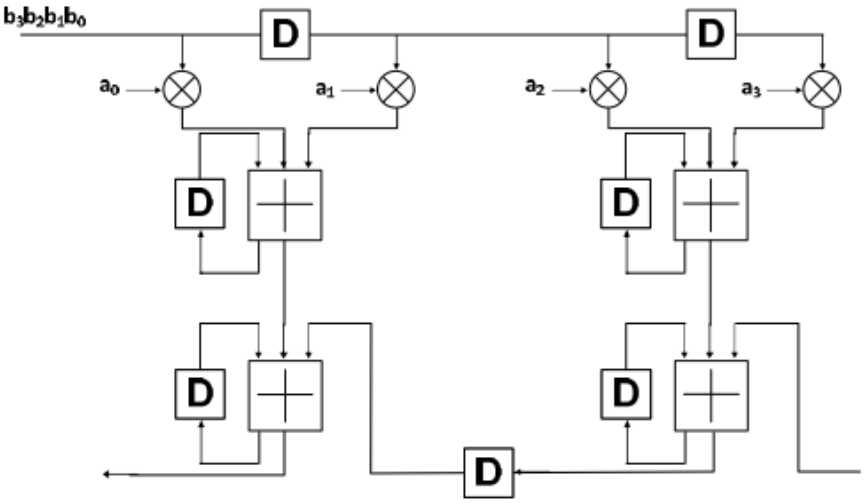


Εικόνα 1.5 Αλγόριθμος πολλαπλασιασμού Serial – Parallel Τύπου 1

1.1.4.1.2 Τύπος 2ος

Η μόνη διάφορα του δεύτερου τύπου σειριακού αθροιστή από τον πρώτο έγκειται στην καθυστέρηση μετάδοσης του σήματος που στον πρώτο τύπο εξαρτάται κυρίως από δύο full adders (Εικόνα 1.6). Επίσης, αυτή η αρχιτεκτονική μπορεί γίνει μέρος μιας μεγαλύτερης αρχιτεκτονικής

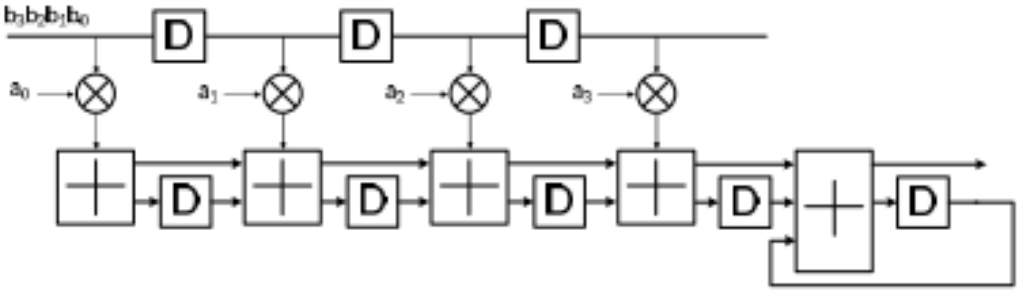
καθώς προσφέρει τη δυνατότητα διασύνδεσης με επιπλέον κυκλώματα (pipeline). Σε αυτή την περίπτωση, καθώς αλλάζει η δομή του κυκλώματος, το βασικό στοιχείο καθυστέρησης θα είναι N+1 full adders.



Εικόνα 1.6 Αλγόριθμος πολλαπλασιασμού Serial – Parallel Τύπου 2

1.1.4.1.3 Τύπος 3ος

Σε αυτόν τον τύπο το σήμα μεταφοράς δεν ανατροφοδοτείται στην είσοδο του full adder όπως συμβαίνει στον τύπο 1 (Εικόνα 1.7). Σε αυτή την περίπτωση ο αριθμός των full adders καθορίζει την καθυστέρηση μετάδοσης σήματος. Η δομή αυτή μπορεί να διασυνδεθεί με άλλες δομές καθώς δεν περιέχει ανάδραση κρατουμένου.

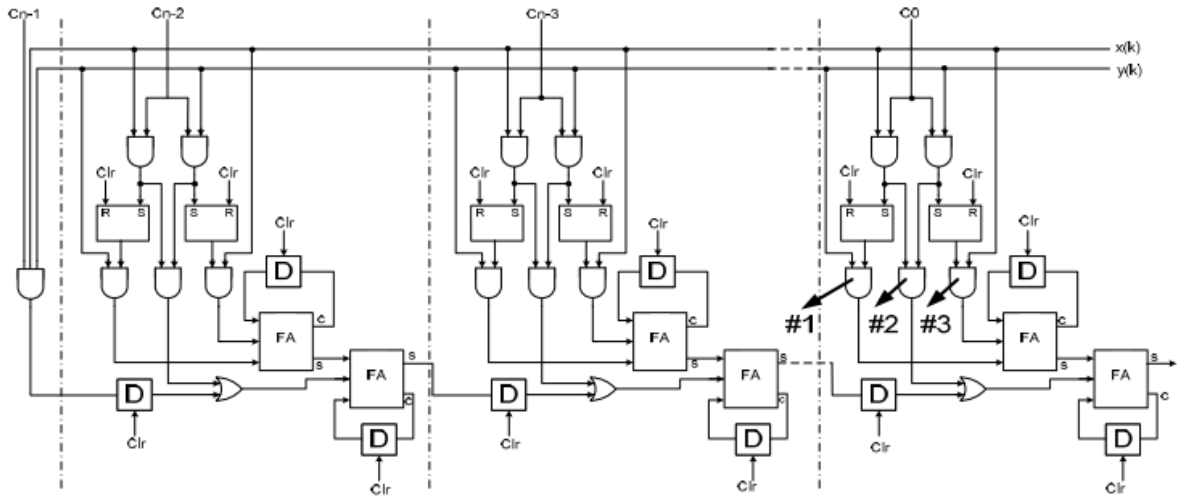


Εικόνα 1.7 Αλγόριθμος πολλαπλασιασμού Serial – Parallel Τύπου 3

1.1.4.2 Serial – Serial

Όλες οι παραπάνω δομές που αναφέρθηκαν ανήκουν στη κατηγορία των serial – parallel σειριακών αλγορίθμων πολλαπλασιασμού. Στις σειριακές επικοινωνίες τα δεδομένα εισέρχονται σειριακά, επομένως ο σειριακός αλγόριθμος πολλαπλασιασμού πρέπει να πολλαπλασιάζει τον πολλαπλασιαστή με τον πολλαπλασιαστέο σειριακά.

Ο αλγόριθμος πολλαπλασιασμού serial – serial αποθηκεύει τα bits εισόδου σε RS καταχωρητές όπως φαίνεται και στην Εικόνα 1.8. Ένας αλγόριθμος πολλαπλασιασμού N-bits μπορεί να δώσει αποτέλεσμα μετά από 2N παλμούς ρολογιού. Επομένως το σημαντικότερο πλεονέκτημα του συγκεκριμένου αλγορίθμου είναι πως μπορεί να παρέχει αποτέλεσμα μετά από χρόνο 2N παλμών ρολογιού, ο οποίος είναι ανεξάρτητος από την καθυστέρηση των full adders, σε αντίθεση με τους παραπάνω serial – parallel αλγορίθμους.



Εικόνα 1.8 Αλγόριθμος πολλαπλασιασμού Serial – Serial

Στη συνέχεια παρουσιάζεται ο αλγόριθμος πολλαπλασιασμού serial – serial για δύο αριθμούς x και y με N ψηφία ο καθένας τους. Ο παρακάτω αλγόριθμος πολλαπλασιασμού αφορά ακέραιες δεκαδικές αναπαραστάσεις αριθμών στο δυαδικό σύστημα.

Στο αρχικό βήμα όταν $f_0 = f(x, y)$, όπως φαίνεται στην (1), ο κάθε αριθμός αναπαρίσταται από το άθροισμα των λιγότερο σημαντικών ψηφιών και των εναπομείναντων bits.

$$f_0 = x \times y = \left(\sum_{i=0}^{n-1} x_i \times 2^i\right) \times \left(\sum_{i=0}^{n-1} y_i \times 2^i\right)$$

$$= \left(\sum_{i=1}^{n-1} \left(x_i \times 2^i + x_0\right)\right) \times \left(\sum_{i=1}^{n-1} \left(y_i \times 2^i + y_0\right)\right) \quad (1)$$

$$= \left(\sum_{i=1}^{n-1} x_i \times 2^i\right) \times \left(\sum_{i=1}^{n-1} y_i \times 2^i\right) + x_0 \times y_0 + \left(x_0 \times \sum_{i=1}^{n-1} y_i \times 2^i\right) + \left(y_0 \times \sum_{i=1}^{n-1} x_i \times 2^i\right)$$

Αφού γίνει ο πολλαπλασιασμός γραμμή προς γραμμή γίνεται χρήση της συνάρτησης f_1.

$$f_1 = f(x - x_0, y - y_0) = (x - x_0) \times (y - y_0) \quad (2)$$

Η συνάρτηση f_1 μπορεί να αποσυντεθεί με παρόμοιο τρόπο στο επόμενο βήμα. Επαναλαμβάνοντας τη διαδικασία αυτή η γενική συνάρτηση (3) μπορεί να εξαχθεί.

$$f_j = f_{j+1} + \Delta_j, j = 0,1, \dots, n-1$$

$$\Delta_j = x_j \times y_j \times 2^{2\times j} + \left(x_j \times 2^j \times \sum y_i \times 2^i\right) + \left(y_j \times 2^j \times \sum x_i \times 2^i\right) \quad (3)$$

Η j-οστή επανάληψη σύμφωνα με την εξίσωση (3) είναι η f_{j+1} συν την αθροιστική μεταβολή Δ_j. Σε αυτή την περίπτωση αν $X_j = 1$ τα βάρη των bits του αριθμού y αθροίζονται όλα μαζί. Επίσης, αν $Y_j = 1$ τα βάρη των bits του αριθμού x αθροίζονται μαζί. Αν και τα δύο bits είναι 1, αν είναι αναγκαίο, τα bit βάρους του πολλαπλασιασμού $X_j \times Y_j$ μπορούν να προστεθούν σε προηγούμενο άθροισμα. Επομένως κατά την εκάστοτε επανάληψη, τα bits (x_j, y_j) πρέπει να αποθηκευτούν για την επόμενη επανάληψη. Συνεπώς οι δομές πρέπει να σχεδιαστούν έτσι ώστε σε κάθε βήμα να δημιουργούνται τα bits των εισόδων και να αποθηκεύονται οι μερικοί πολλαπλασιασμοί.

Οι σειριακοί αλγόριθμοί είναι η καλύτερη επιλογή όταν η ταχύτητα μετάδοσης του σήματος είναι λιγότερο σημαντική από τον χώρο και την κατανάλωση. Οι σειριακοί αλγόριθμοι προσφέρουν υλοποιήσεις με λιγότερες λογικές πύλες άρα και με λιγότερη κατανάλωση. Το αντίτιμο αυτού όμως, είναι η μείωση της συνολικής τους απόδοσης, καθώς η καθυστέρηση μετάδοσης της πληροφορίας αυξάνεται.

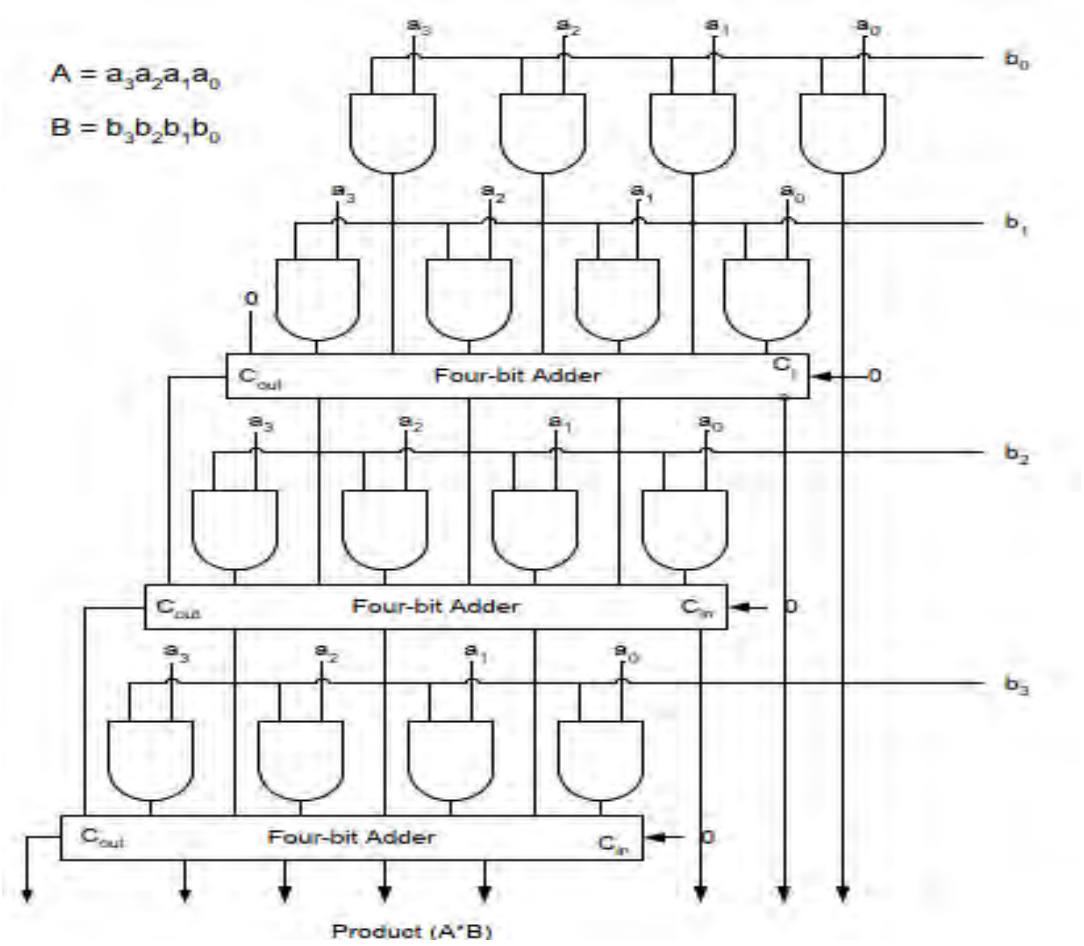
1.1.5 ARRAY [5],[6]

Οι αλγόριθμοι πολλαπλασιασμού πίνακα είναι ευρέως γνωστοί λόγω της απλής δομής τους. Το κύκλωμα του πολλαπλασιαστή βασίζεται στον αλγόριθμο προσθήκης και μετατόπισης. Κάθε μερικό γινόμενο παράγεται από τον πολλαπλασιασμό του πολλαπλασιαστέου με κάθε ένα από τα bits του πολλαπλασιαστή. Τα μερικά γινόμενα στη συνέχεια μετατοπίζονται σύμφωνα με τη θέση τους και ύστερα προστίθενται (Εικόνα 1.9). Η πρόσθεση μπορεί να πραγματοποιηθεί και με έναν απλό αθροιστή Carry Propagate. Οι αλγόριθμοί πολλαπλασιασμού πίνακα απαιτούν N-1 αριθμό αθροιστών, υποθέτοντας πως το μήκος των αριθμών πολλαπλασιασμού είναι N.

			x	A3 B3	A2 B2	A1 B1	A0 B0	Inputs
			C	B0 x A3	B0 x A2	B0 x A1	B0 x A0	
		+	B1 x A3	B1 x A2	B1 x A1	B1 x A0		
		C	sum	sum	sum	sum		
	+	B2 x A3	B2 x A2	B2 x A1	B2 x A0			Internal Signals
	C	sum	sum	sum	sum			
+	B3 x A3	B3 x A2	B3 x A1	B3 x A0				
C	sum	sum	sum	sum				
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Outputs

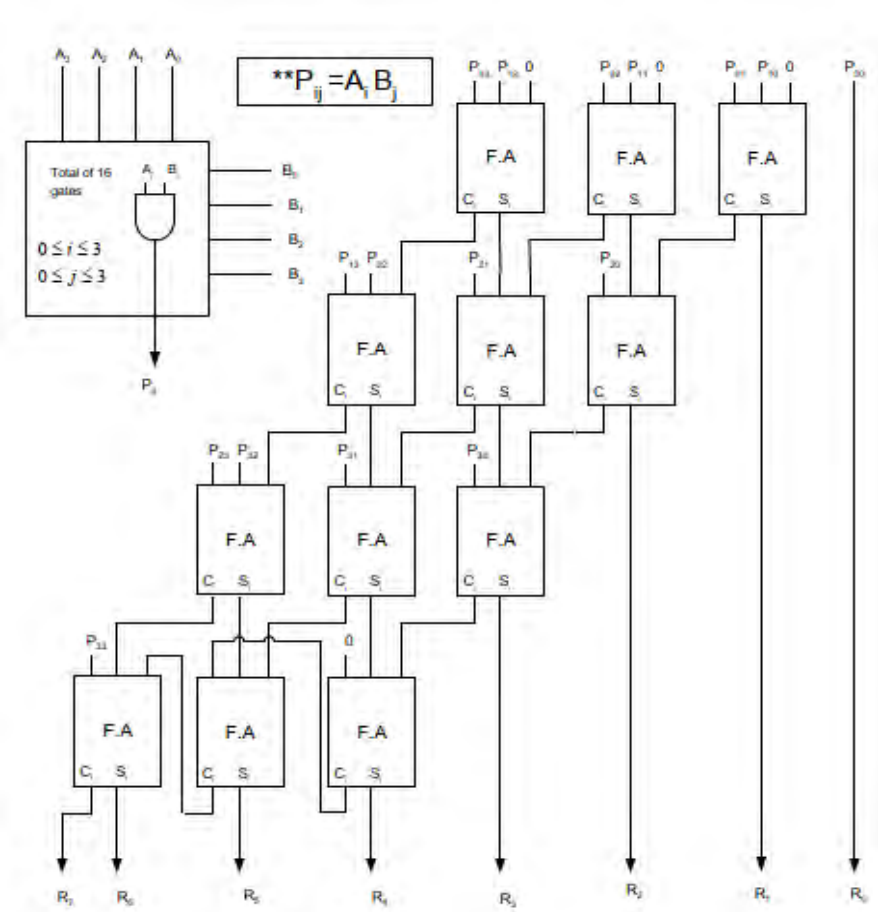
Εικόνα 1.9 Αλγόριθμος πολλαπλασιασμού πίνακα για 4x4 bits

Ένα παράδειγμα πολλαπλασιασμού με τη μέθοδο πίνακα φαίνεται με τη μορφή λογικών πυλών για δύο 4bits αριθμούς στην Εικόνα 1.10.



Εικόνα 1.10 Δομή λογικών πυλών αλγορίθμου πολλαπλασιασμού πίνακα για 4x4 bits

Παρόλο που η μέθοδος αυτή είναι φαινομενικά απλή, όπως φαίνεται και από το παραπάνω σχήμα, η πρόσθεση γίνεται ταυτόχρονα, παράλληλα και σειριακά. Με σκοπό τη βελτίωση της καθυστέρησης μετάδοσης, καθώς και τη περιοχής που καταλαμβάνει στο FPGA, οι CRA αθροιστές αντικαθίστανται συνήθως από αθροιστές Carry Save, στους οποίους το αποτέλεσμα και το κρατούμενο περνά σαν πληροφορία στους αθροιστές του επόμενου σταδίου. Το τελικό γινόμενο αποκτάται από οποιονδήποτε αθροιστή ταχείας μετάδοσης σήματος (συνήθως αυτός είναι αθροιστής Carry Ripple). Στους πολλαπλασιαστές πίνακα ο αριθμός των μερικών γινομένων που πρέπει να προστεθούν είναι ίδιος με τον αριθμό των bits του πολλαπλασιαστή. Αυτή η διάταξη παρουσιάζεται στην Εικόνα 1.11.



Εικόνα 1.11 Δομή λογικών πυλών αλγορίθμου πολλαπλασιασμού πίνακα με χρήση Carry Save αθροιστών

Η περιοχή που καταλαμβάνει ένας αλγόριθμος πολλαπλασιασμού πίνακα ανέρχεται σε:

$$Total\ Area = (N-1) * M * Area_{FA}\ ,$$

- N: τα bits του πολλαπλασιαστή
- M: τα bits του πολλαπλασιαστέου
- $Area_{FA}$: περιοχή κατάληψης πλήρους αθροιστή

Η καθυστέρηση μετάδοσης της πληροφορίας δίνεται από τη σχέση:

$$Delay = 2 * (M-1) *\ \tau_{FA}$$

- τ_{FA}: Η καθυστέρηση μετάδοσης πληροφορίας ενός πλήρους αθροιστή.

Τα πλεονεκτήματα χρήσης ενός αλγορίθμου πολλαπλασιασμού πίνακα είναι:

1. Έχει μικρή πολυπλοκότητα.
2. Παράγεται εύκολα για πολλαπλασιασμούς τόσο μικρών όσο και μεγάλων bits
3. Διαρθρώνεται εύκολα σε συστήματα που απαιτούν παραπάνω από ένα κύκλωμα.
4. Το σχήμα του είναι απλό με αποτέλεσμα να είναι εύκολη η διαδικασία του placement και routing.

Τα μειονεκτήματα του αλγορίθμου πολλαπλασιασμού πίνακα είναι:

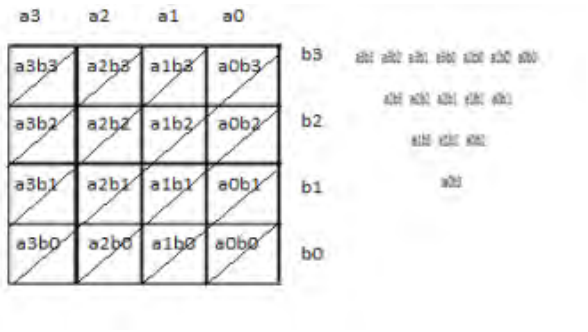
1. Έχει μεγάλη κατανάλωση η οποία αυξάνεται εκθετικά με την αύξηση των αριθμών εισόδου
2. Όσο ο αριθμός των bits εισόδου αυξάνεται το ίδιο κάνουν και οι λογικές πύλες, γεγονός που οδηγεί σε ανάγκη κάλυψης μεγαλύτερης περιοχής.

1.1.6 DADDA [7], [8]

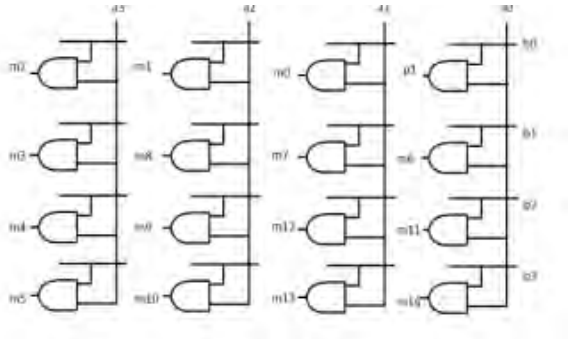
Ο αλγόριθμος πολλαπλασιασμού Dadda προκύπτει από τον αλγόριθμο παράλληλου πολλαπλασιασμού. Ο Dadda είναι ελαφρώς ταχύτερος και απαιτεί λιγότερο αριθμό πυλών από τον παράλληλο αλγόριθμο πολλαπλασιασμού. Ο αλγόριθμος Dadda είναι μια από τις πολλές μεθόδους παράλληλου πολλαπλασιασμού και ελαχιστοποιεί τον αριθμό των σταδίων πρόσθεσης που απαιτούνται για τον υπολογισμό του αθροίσματος των μερικών γινομένων. Αυτό το επιτυγχάνει χρησιμοποιώντας πλήρεις και ημιπλήρεις αθροιστές ώστε να μειωθεί ο αριθμός των γραμμών του πίνακα γινομένων σε κάθε στάδιο πρόσθεσης. Παρόλο που ο αλγόριθμός πολλαπλασιασμού Dadda έχει απλή δομή, η διαδικασία μετάδοσης της πληροφορίας είναι πιο αργή λόγω της σειριακής διαδικασίας πολλαπλασιασμού.

1.1.6.1 Αλγόριθμος Dadda

Ο αλγόριθμός Dadda βασίζεται στον πίνακα που φαίνεται στην Εικόνα 1.12. Ο πίνακας των μερικών γινομένων κατασκευάζεται στο πρώτο στάδιο από μια αλληλουχία AND πυλών όπως φαίνεται στην Εικόνα 1.13.



Εικόνα 1.12 Πίνακας μερικών γινομένων αλγορίθμου πολλαπλασιασμού Dadda



Εικόνα 1.13 Παραγωγή μερικών γινομένων με χρήση πυλών AND

Ο αλγόριθμος Dadda περιλαμβάνει τα εξής παρακάτω βήματα:

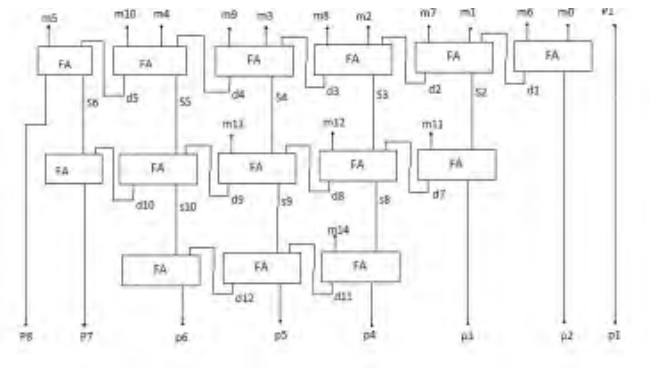
1. Πολλαπλασιασμός κάθε bit του πολλαπλασιαστή με κάθε bit του πολλαπλασιαστέου. Ανάλογα με τη θέση των μερικών γινομένων τα καλώδια μεταφοράς τους έχουν διαφορετικά βάρη.
2. Μείωση του αριθμού των μερικών γινομένων σε δύο επίπεδα πλήρους αθροιστών.
3. Ομαδοποίηση των καλωδίων σε δύο αριθμούς και πρόσθεσή τους με συμβατικό αθροιστή.

1.1.6.2 Αλγόριθμος Dadda με χρήση αθροιστή Ripple Carry

Το λογικό διάγραμμα ενός αλγορίθμου πολλαπλασιασμού Dadda με χρήση αθροιστών Ripple Carry φαίνεται στην Εικόνα 1.14. Σύμφωνα με το σχήμα, παίρνοντας οποιαδήποτε τρία καλώδια

με το ίδιο βάρος και εισάγοντάς τα ως εισόδους σε έναν αθροιστή το αποτέλεσμα της άθροισης θα είναι ένα καλώδιο εξόδου του ίδιο βάρους. Η διαδικασία του αλγορίθμου φαίνεται παρακάτω:

1. Τρία από τα μερικά γινόμενα που προκύπτουν από το πρώτο στάδιο μεταφέρονται με τρία καλώδια ως είσοδοι στους αθροιστές. Το αποτέλεσμα της άθροισης μεταφέρεται ως είσοδος στους αθροιστές του επόμενου σταδίου, ενώ το κρατούμενο της άθροισης αθροίζεται με τα επόμενα δύο μερικά γινόμενα του αθροιστή του ίδιου σταδίου.
2. Με την επανάληψη της διαδικασίας αυτής τα μερικά γινόμενα μειώνονται σε δύο επίπεδα πλήρους αθροιστών.
3. Στο τελικό επίπεδο ακολουθείται η ίδια μέθοδος άθροισης με τον Ripple Carry αθροιστή και προκύπτει το γινόμενο p1 έως p8.



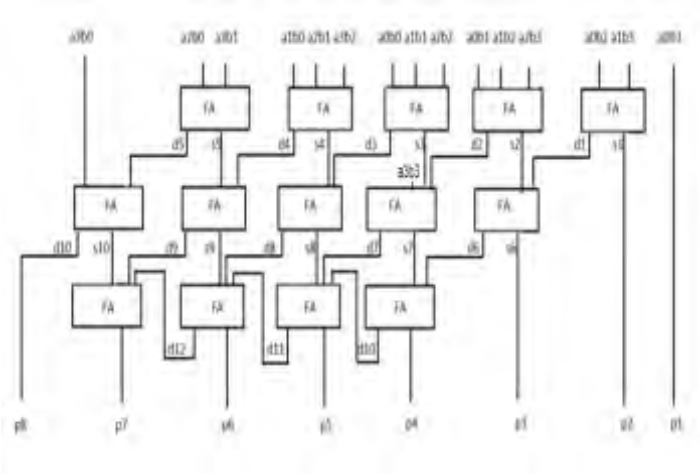
Εικόνα 1.14 Άθροιση μερικών γινομένων με χρήση αθροιστών Ripple Carry

1.1.6.3 Αλγόριθμος Dadda με χρήση αθροιστή Carry Save

Η αρχιτεκτονική του αλγορίθμου Dadda με χρήση αθροιστή Carry Save φαίνεται στην Εικόνα 1.15. Η διαδικασία του αλγορίθμου ορίζει πως, παίρνοντας οποιαδήποτε τρία διαδοχικά καλώδια που μεταφέρουν μερικά γινόμενα και αθροίζοντάς τα, το αποτέλεσμα της άθροισης θα είναι ένα καλώδιο ίδιου βάρους με αυτά της εισόδου και ένα καλώδιο μεγαλύτερου βάρους. Τα βήματα της διαδικασίας είναι:

1. Τρία μερικά διαδοχικά γινόμενα εισάγονται με καλώδια ως είσοδοι ενός πλήρους αθροιστή.

2. Το κρατούμενο της άθροισης γίνεται είσοδος των αθροιστών του επόμενου σταδίου.
3. Στο επόμενο στάδιο τα αθροίσματα και τα κρατούμενα των αθροιστών του προηγούμενου σταδίου αθροίζονται μεταξύ τους.
4. Στο τελικό στάδιο γίνεται χρήση ενός γρήγορου αθροιστή ripple carry ώστε να μειωθεί ο αριθμός των σταδίων και να προκύψει το γινόμενο p1 ως p8.



Εικόνα 1.15 Άθροιση μερικών γινομένων με χρήση αθροιστών Carry Save

Τα πλεονεκτήματα του αλγορίθμου πολλαπλασιασμού Dadda παρουσιάζονται παρακάτω:

1. Είναι ταχύτερος από τον Wallace Tree πολλαπλασιαστή.
2. Έχει μικρότερη πολυπλοκότητα από τον Wallace Tree.
3. Έχει χαμηλότερες απώλειες από τον Wallace Tree.
4. Έχει λιγότερη καθυστέρηση μετάδοσης πληροφορίας από τον Wallace Tree.

Ο αλγόριθμος Dadda είναι γενικά καλύτερος από τον αντίστοιχο αλγόριθμο Wallace Tree με τα ίδια όμως μειονεκτήματα με αυτόν[9].

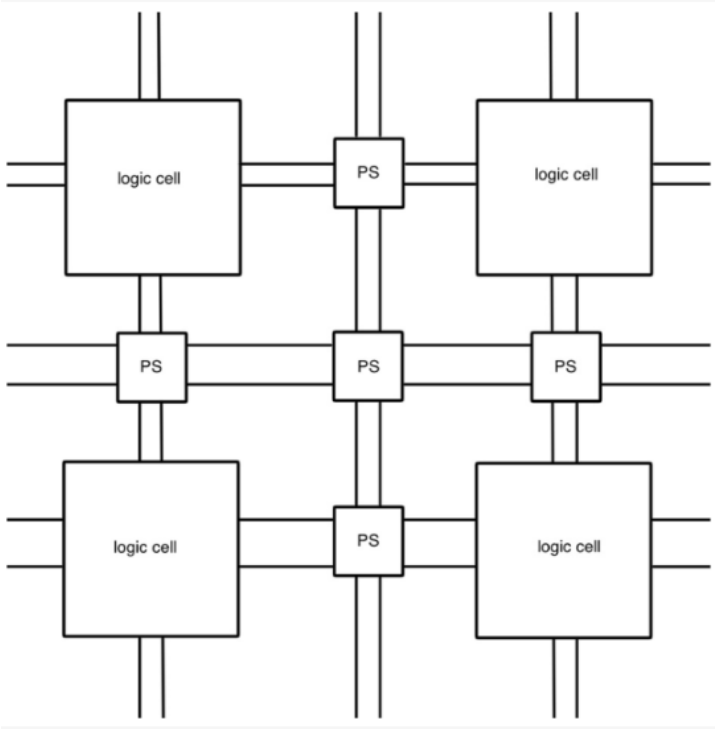
ΚΕΦΑΛΑΙΟ 2

ΠΡΟΣΟΜΟΙΩΣΗ FPGA

2.1 ΤΟ FPGA

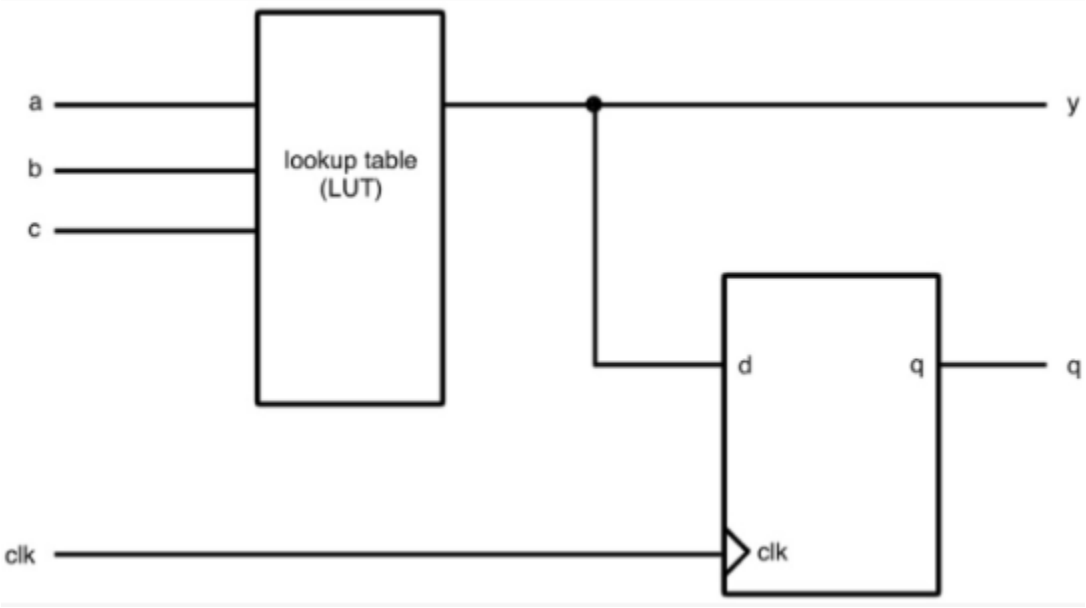
2.1.1 ΓΕΝΙΚΗ ΑΝΑΛΥΣΗ FPGA[10]

Το FPGA είναι μια συσκευή που στο δυσδιάστατο επίπεδο περιλαμβάνει μια συστοιχία λογικών κελιών και προγραμματιζόμενων διακοπτών. Το απλοποιημένο δομικό διάγραμμα ενός FPGA φαίνεται στην Εικόνα 2.1. Κάθε λογικό κελί μπορεί να διαμορφωθεί ώστε να εκτελεί μια απλή λειτουργία και ένας προγραμματιζόμενος διακόπτης προσαρμόζεται ώστε να παρέχει ή όχι σύνδεση μεταξύ των λογικών κελιών. Κάθε εξατομικευμένος σχεδιασμός υλοποιείται καθορίζοντας τη λειτουργία κάθε λογικού κελιού και επιλεκτικά ρυθμίζοντας τις συνδέσεις των προγραμματιζόμενων διακοπτών.



Εικόνα 2.1 Απλοποιημένο δομικό διάγραμμα FPGA

Κάθε λογικό κελί αποτελείται από ένα LUT(Look Up Table) και ένα D flip – flop. Το LUT χρησιμοποιείται για την εκτέλεση οποιασδήποτε συνδυαστικής λειτουργίας. Ο κάθε κατασκευαστής FPGA μπορεί να επιλέξει διαφορετικό αριθμό LUTs είσοδού για το προϊόν του, με αποτέλεσμα κάθε FPAGA να διαφοροποιείται από κάθε άλλο. Στην Εικόνα 2.2 φαίνεται η εσωτερική δομή ενός λογικού κυττάρου. Σύμφωνα με αυτή βλέπουμε πως βασίζεται σε ένα LUT τριών εισόδων και ένα D flip – flop. Το FPGA μπορεί να σχηματίσει ένα ακολουθιακό ή ένα συνδυαστικό κύκλωμα ανάλογα με την έξοδο που επιλέγεται. Δηλαδή, η έξοδος y οδηγεί σε συνδυαστική υλοποίηση ενώ η q σε ακολουθιακή. Εκτός από λογικά κύτταρα και προγραμματιζόμενους διακόπτες κάθε FPGA αποτελείται και από αλλά στοιχεία τα οποία είναι υπεύθυνα για τη μνήμη, τη διαχείριση του ρολογιού τις διεπαφές εισόδου και εξόδου κ.α..



Εικόνα 2.2 Εσωτερικό διάγραμμα λογικού κυττάρου

Με σκοπό την αύξηση της ευελιξίας και της απόδοσης οκτώ λογικά κύτταρα συνδυάζονται μεταξύ τους σε μια ειδική δομή εσωτερικής δρομολόγησης. Σύμφωνα με τους όρους που έχει

εισάγει η Xilinx δύο λογικά κύτταρα που συνδυάζονται μαζί δημιουργούν ένα “slice”. Συνδυάζοντας τέσσερα “slices” δημιουργείται ένα CLB (Configurable Logic Block).

2.1.2 ΑΝΑΛΥΣΗ ΤΟΥ FPGA ΠΟΥ ΠΡΟΣΟΜΟΙΩΘΗΚΕ

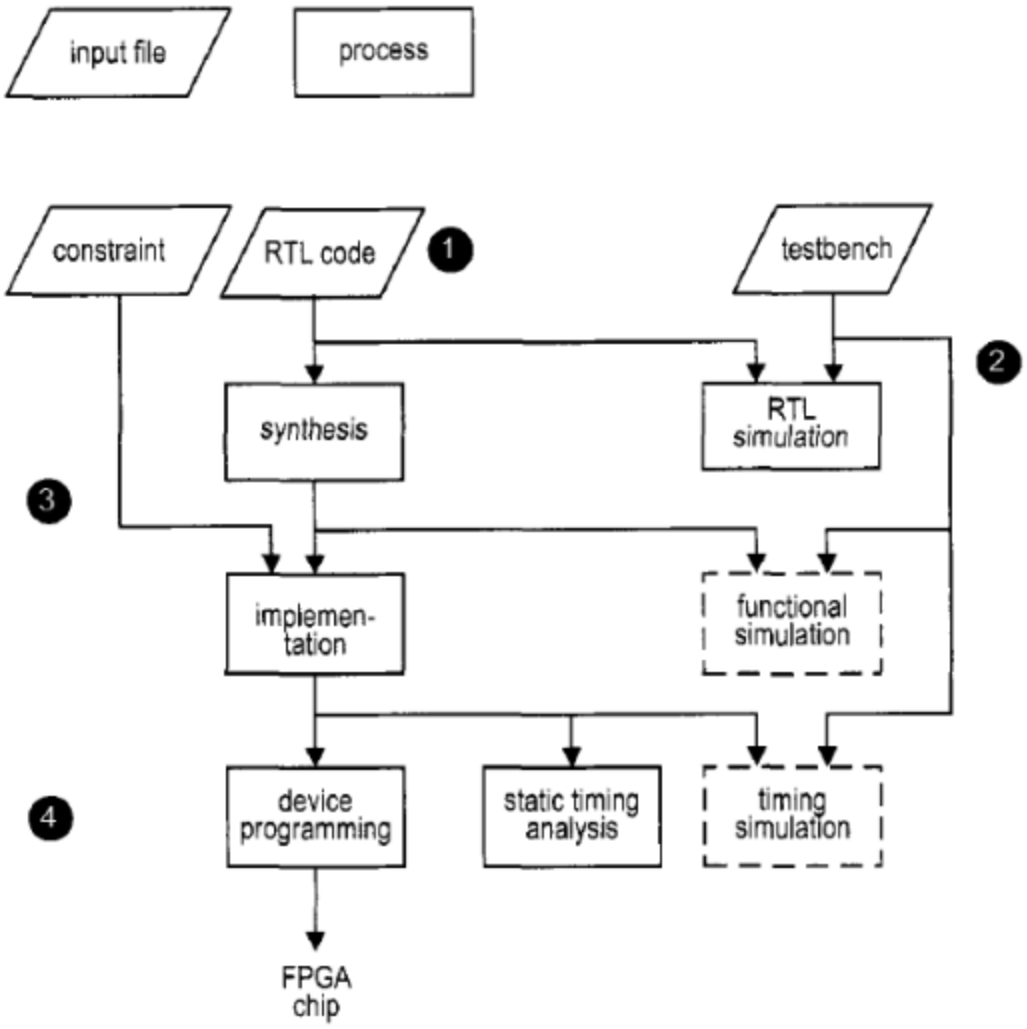
Η διάταξη που προσομοιώθηκε στην παρούσα διπλωματική εργασία για την εξαγωγή των παρακάτω δεδομένων είναι το Evaluation Board της Xilinx, AMD Artix 7 FPGA Evaluation Kit. Το συγκεκριμένο Evaluation Board προσφέρει αρκετά προεγκατεστημένα περιφερειακά όπως επιταχυνσιόμετρο, θερμοκρασιακό αισθητήρα, ψηφιακό μικρόφωνο, ενισχυτή ήχου και μεγάλο αριθμό συσκευών εισόδου / εξόδου. Η μεγάλη ποικιλία των περιφερειακών συστημάτων παρέχει στο σχεδιαστή τη δυνατότητα διενέργειας μιας ευρείας γκάμας εφαρμογών χωρίς την ανάγκη επιπλέον εξωτερικών συστημάτων. Με το ισχυρών δυνατοτήτων FPGA που ενσωματώνει (XC7A200T-2FBG484C), το μεγάλο αριθμό εξωτερικών μνημών και τη μεγάλη ποικιλία θυρών, όπως USB, ETHERNET κ.α., μπορεί να υλοποιήσει σχεδιασμούς που κυμαίνονται από εκπαιδευτικά κυκλώματα έως και ισχυρούς επεξεργαστές.

Το FPGA Artix – 7 XC7A200T-2FBG484C που ενσωματώνει το παραπάνω Evaluation Board είναι βελτιστοποιημένο για λογική υψηλής απόδοσης και προσφέρει μεγαλύτερη χωρητικότητα, υψηλότερη απόδοση και περισσότερους πόρους σε σχέση με παλαιότερους σχεδιασμούς της ίδιας οικογένειας. Κάποια από τα χαρακτηριστικά του XC7A200T-2FBG484C αναφέρονται παρακάτω:

- 33.650 CLBs.
- 740 DSP Slices.
- 13.140Kbits γρήγορης μνήμης RAM
- 10 CMTs (Clock Management Tiles)
- 1 XADC
- Μέγιστη συχνότητα εσωτερικού ρολογιού 640MHz

2.1.3 ΔΙΑΔΙΚΑΣΙΑ ΑΝΑΠΤΥΞΗΣ

Η απλοποιημένη ροή ανάπτυξης ενός συστήματος που βασίζεται σε FPGA παρουσιάζεται στην Εικόνα 2.3. Το αριστερό μέρος της ροής ανάπτυξης αφορά τη μετατροπή της εννοιολογικής γλώσσας μηχανής σε όλες τις απαραίτητες λειτουργίες των φυσικών λογικών κελιών του FPGA. Το δεξί μέρος της ροής αφορά την επιβεβαίωση ορθής λειτουργίας του κώδικα μηχανής σύμφωνα με τις απαιτήσεις σχεδίασης. Τα κύρια βήματα της διαδικασίας αναλύονται παρακάτω:



Εικόνα 2.3 Διαδικασία Ανάπτυξης

1. Επίπεδο σχεδίασης: Κατά το στάδιο αυτό παράγεται ο περιγραφικός κώδικάς μηχανής σύμφωνα με τις απαιτήσεις σχεδίασης. Στο στάδιο αυτό μπορεί να παραχθεί και ένα αρχείο περιορισμών σχεδίασης ώστε να προδιαγράψει τους όποιους περιορισμούς υλοποίησης του κώδικα μηχανής.

2. Προσομοίωση του κώδικα μηχανής: Η προσομοίωση του κώδικα μηχανής γίνεται με τη χρήση κώδικα test bench. Ο κώδικας test bench όπως και ο κώδικας σχεδίασης είναι κώδικες περιγραφής υλικού. Η προσομοίωση όμως γίνεται σε επίπεδο registers (RTL) και όχι σε επίπεδο κώδικα περιγραφής υλικού (HDL). Αυτό σημαίνει πως οι HDL κώδικες μεταφράζονται σε registers και ύστερα προσομοιώνονται.
3. Διαδικασία "Synthesis": Η διαδικασία αυτή μετατρέπει τον κώδικα περιγραφής υλικού σε απλές λογικές πύλες, flip – flops και άλλα λογικά εξαρτήματα.
4. Διαδικασία "Implementation": Η διαδικασία "Implementation" αποτελείται από τρεις μικρότερες διεργασίες: 1)τη μετάφραση (translation) 2)τη χαρτογράφηση (mapping) και 2)την τοποθέτηση και δρομολόγηση (placement & routing). Οι τρεις αυτές υποδιαδικασίες περιγράφονται επιγραμματικά παρακάτω:
    a. Η διαδικασία της μετάφρασης συγχωνεύει όλα τα σχηματικά διαγράμματα σε ένα ενιαίο δίκτυο.
    b. Η διαδικασία της χαρτογράφησης αντιστοιχίζει τις εννοιολογικές πύλες του σχηματικού σε φυσικά λογικά κελιά και κελιά εισόδων / εξόδων του FGPA.
    c. Η διαδικασία τοποθέτησης και δρομολόγησης παράγει τη φυσική διάταξη στο εσωτερικού του FPGA. Τοποθετεί τα κελιά στις φυσικές τους θέσεις και καθορίζει τις διαδρομές διασύνδεσης.

    Μετά το τέλος του "Implementation" και πρίν γίνει ο προγραμματισμός του FPGA πραγματοποιείται ανάλυση στατικού χρονισμού (static time analysis) κατά την οποία εξάγονται διάφορες χρονικές παράμετροι όπως η μέγιστη καθυστέρηση μετάδοσης σήματος και η μέγιστη συχνότητα ρολογιού.
5. Προγραμματισμός του FPGA: Στη διαδικασία αυτή παράγεται ένα αρχείο διαμόρφωσης σύμφωνα με το τελικό δίκτυο όπως αυτό προέκυψε από το "Implementation". Το αρχείο αυτό «φορτώνεται» στο FPGA σειριακά ώστε να διαμορφώσει κατάλληλα τα λογικά κελιά και τους προγραμματιζόμενους διακόπτες. Στη συνέχεια μπορεί προαιρετικά να γίνει και λειτουργική και χρονική προσομοίωση ώστε να διαπιστωθεί η ορθότητα του και σε φυσικό επίπεδο. Η λειτουργική προσομοίωση χρησιμοποιεί ένα συνθετικό δίκτυο ώστε να αντικαταστήσει την RTL περιγραφή και ελέγχει την ορθότητα της διαδικασίας "Synthesis". Η χρονική προσομοίωση χρησιμοποιεί το τελικό δίκτυο και μαζί με

λεπτομερή δεδομένα χρονισμού πραγματοποιεί την προσομοίωση. Λόγω της πολυπλοκότητας του δικτύου, η λειτουργική και χρονική προσομοίωση μπορεί να απαιτούν σημαντικό χρονικό διάστημα. Εφόσον όμως γίνει χρήση «καλών» πρακτικών σχεδίασης, η λειτουργική και χρονική προσομοίωση μπορούν να παραλειφθούν και να διατηρηθεί μόνο η RTL προσομοίωση, ώστε να γίνει έλεγχος του κώδικα περιγραφής υλικού.

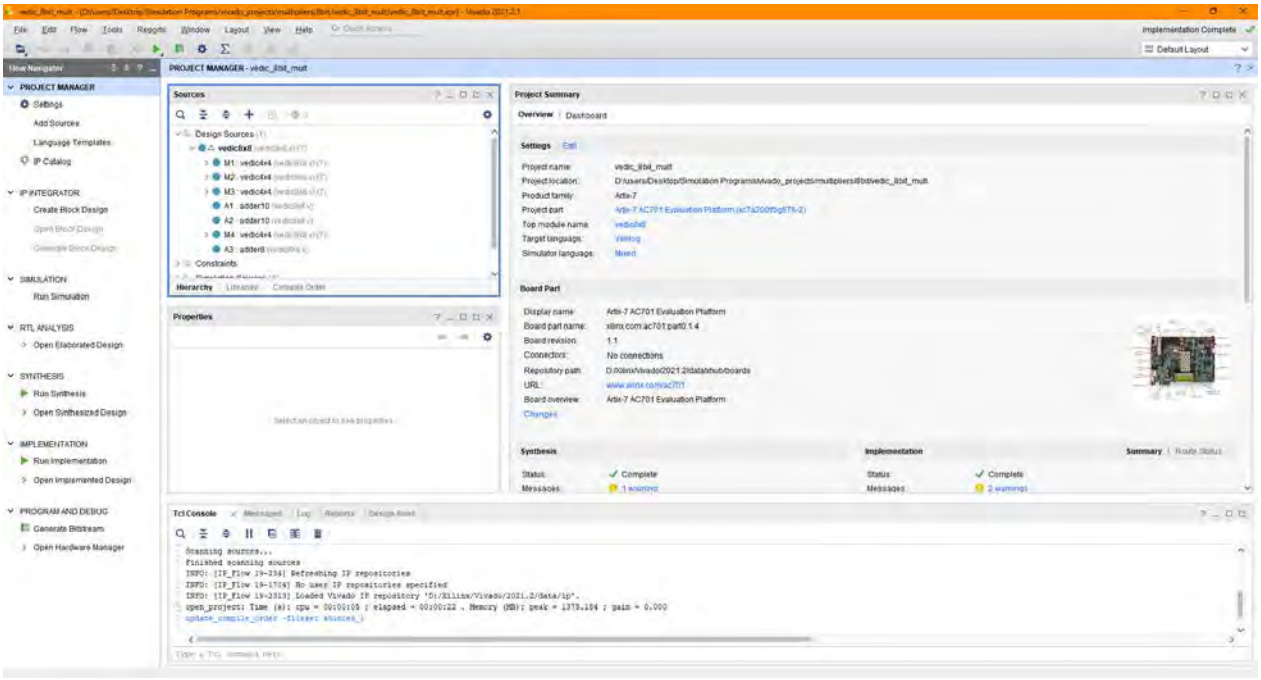
2.2 ΤΟ ΠΡΟΓΡΑΜΜΑ ΠΡΟΣΟΜΟΙΩΣΗΣ VIVADO

Στη συνέχεια θα γίνει περιγραφή του προγράμματος προσομοίωσης Vivado. Με αυτό έγιναν όλες οι απαραίτητες προσομοιώσεις και εξάχθηκαν όλες οι απαραίτητες αναφορές, βάσει των οποίων παράχθηκαν τα αποτελέσματα του επόμενου κεφαλαίου.

Για την παρουσίαση του προγράμματος χρησιμοποιήθηκε ο 8bit αλγόριθμός πολλαπλασιασμού Vedic.

2.2.1 ΤΟ ΚΥΡΙΟ ΠΕΡΙΒΑΛΛΟΝ

Στην Εικόνα 2.4 φαίνεται το κύριο περιβάλλον του προγράμματος προσομοίωσης Vivado για τον 8bit αλγόριθμο Vedic.



Εικόνα 2.4 Κύριο περιβάλλον Vivado

Στο πάνω αριστερά μέρος του περιβάλλοντος υπάρχουν τα design sources τα οποία φαίνονται αναλυτικότερα στην Εικόνα 2.5. Στο σημείο αυτό γράφονται από το μηδέν ή προστίθενται εξωτερικά όλα τα απαραίτητα αρχεία κώδικα περιγραφής υλικού, τα αρχεία περιορισμών καθώς και ο κώδικας προσομοίωσης test bench.

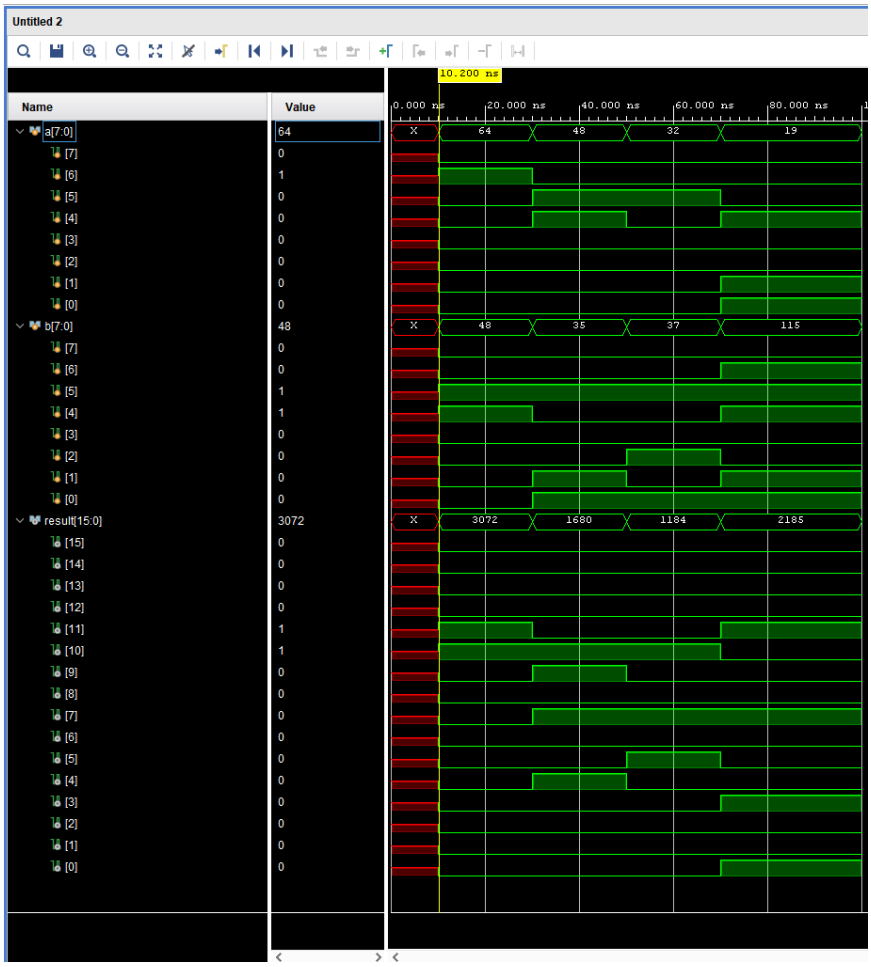
Εικόνα 2.5 Ιεραρχική δομή αρχείων σχεδίασης

Στους αλγόριθμους που αφορούν την παρούσα διπλωματική εργασία δε προστέθηκε κάποιο αρχείο περιορισμών σχεδίασης, μιας και σε κάθε προσομοίωση το FPGA φιλοξενούσε μόνο μια σχεδίαση.

Όπως φαίνεται και στην Εικόνα 2.5, το αρχείο test bench test_8x8 καλεί τον 8bit αλγόριθμο Vedic, ώστε να τον προσομοιώσει, όπως θα δούμε παρακάτω.

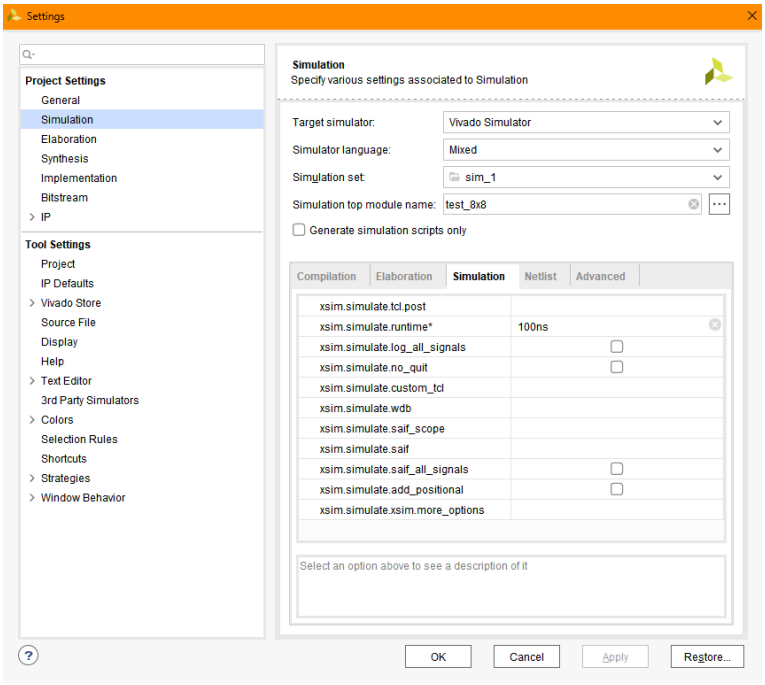
2.2.2 Η ΠΡΟΣΟΜΟΙΩΣΗ RTL

Στη συνέχεια, όπως φαίνεται και στην Εικόνα 2.6, εκτελείται η προσομοίωση RTL του 8bit αλγορίθμου Vedic ώστε να διαπιστωθεί η ορθή λειτουργία του.



Εικόνα 2.6 Προσομοίωση του κώδικα περιγραφής υλικού

Ανάλογα με το είδος του κώδικα που προσομοιώνεται μπορεί να κριθεί απαραίτητο ο χρόνος προσομοίωσης να μεταβληθεί ώστε να γίνει πιο εύκολη ανάλυση των αποτελεσμάτων της. Στην Εικόνα 2.7 φαίνονται το παράθυρο των ρυθμίσεων προσομοίωσης και ο τρόπος μεταβολής του χρόνου προσομοίωσης.



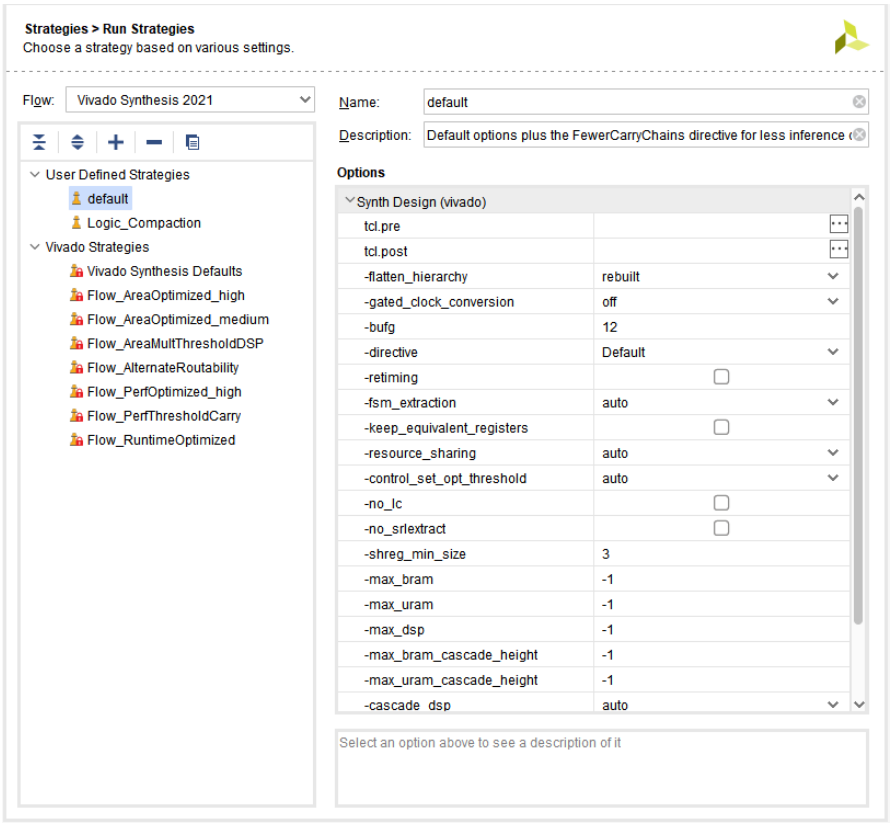
Εικόνα 2.7 Ρυθμίσεις Προσομοίωσης και μεταβολής του χρόνου προσομοίωσης

2.2.3 ΣΤΡΑΤΗΓΙΚΕΣ SYNTHESIS ΚΑΙ IMPLEMENTATION

Αφού ολοκληρωθεί η RTL προσομοίωση με επιτυχία, η διαδικασία συνεχίζεται με το "Synthesis" και μετά με το "Implementation" του κώδικα. Το Vivado δίνει μια πληθώρα διαφορετικών στρατηγικών "Synthesis" και "Implementation» στο σχεδιαστή ώστε αυτός να επιλέξει βάσει της αρχιτεκτονικής που επιθυμεί αυτές που του ταιριάζουν καλύτερα.

2.2.3.1 Στρατηγικές "Synthesis"

Οι στρατηγικές της διαδικασίας "Synthesis" φαίνονται στην Εικόνα 2.8. Οι στρατηγικές που εμφανίζονται με το «κλειδωμένο πιόνι» είναι οι εργοστασιακές στρατηγικές του Vivado, ενώ αυτές που εμφανίζονται με το «σκέτο πιόνι» είναι αυτές που ορίζει ο σχεδιαστής. Κάθε στρατηγική προσφέρει και επιπλέον επιλογές παραμετροποίησης όπως φαίνεται στο δεξί παράθυρο της Εικόνας 2.8, παρέχοντας στον σχεδιαστή επιπλέον ευελιξία στη σχεδίασή του.



Εικόνα 2.8 Στρατηγικές “Synthesis”

Στη συνέχεια θα γίνει μια επιγραμματική αναφορά στη λειτουργία των στρατηγικών “Synthesis” που χρησιμοποιήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας:

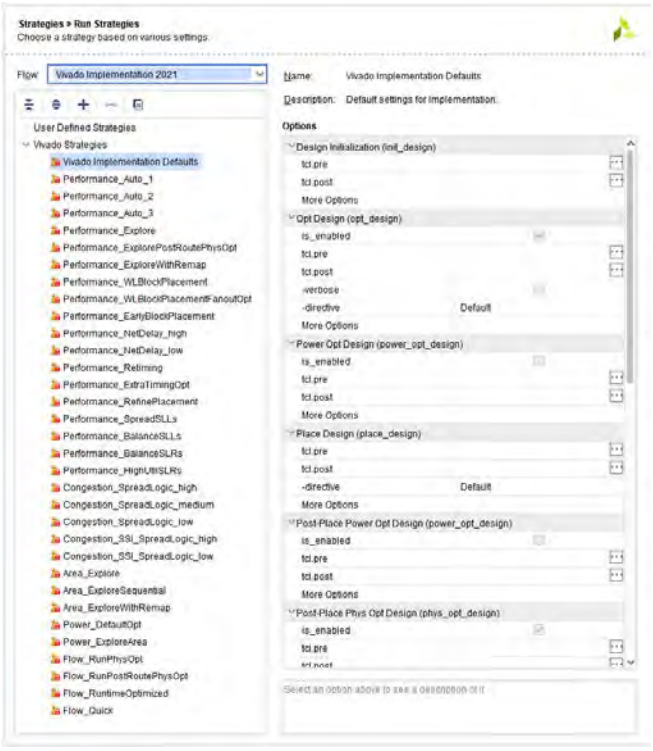
a) Default: Είναι η βασική στρατηγική υλοποίησης του “Synthesis” καθώς δε δίνει βάρος σε κάποιο συγκεκριμένο τομέα.
b) Logic Compaction: Οργανώνει τις αλυσίδες Carry και τα LUTs χρησιμοποιώντας λιγότερα slices ώστε να καθιστά τη λογική πιο συμπαγή.
c) Performance optimized: Στο σχεδιασμό υψηλής απόδοσης η κοινή χρήση πόρων απενεργοποιείται, το fanout ρυθμίζεται σε χαμηλότερο αριθμό, ο συνδυασμός των LUTs

απενεργοποιείται, γίνεται διατήρηση των ισοδύναμων registers και τα SRL εκτιμώνται με μεγαλύτερη ανοχή.

d) Runtime Optimized: Θυσιάζει απόδοση και χωρική περιοχή για λιγότερη καθυστέρηση μετάδοσης σήματος.

2.2.3.2 Στρατηγικές "Implementation"

Σε πλήρη αναλογία με τις στρατηγικές της διαδικασίας "Synthesis" οι στρατηγικές της διαδικασίας "Implementation" φαίνονται στην Εικόνα 2.9. Για τις στρατηγικές σύνδεσης και την παραμετροποίησή τους ισχύουν τα ίδια με τις στρατηγικές "Synthesis" που περιγράφηκαν παραπάνω.



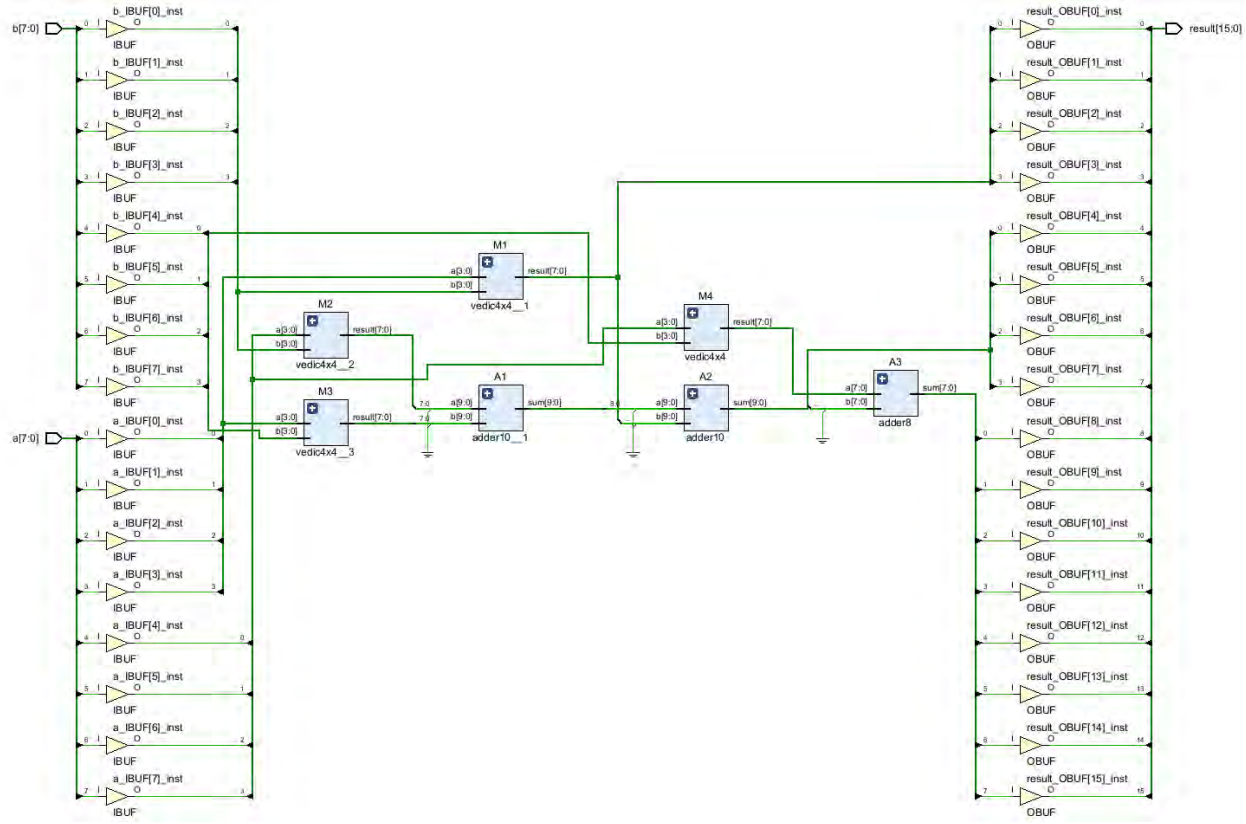
Εικόνα 2.9 Στρατηγικές "Implementation"

Στη συνέχεια θα γίνει μια επιγραμματική αναφορά στη λειτουργία των στρατηγικών "Implementation" που χρησιμοποιήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας:

a) Default: Η βασική λειτουργία “Implementation” η οποία δε δίνει βάρος σε κάποιο συγκεκριμένο τομέα.
b) Explore Area: Κάνει χρήση πολλαπλών αλγορίθμων βελτιστοποίησης ώστε να επιτύχει δυνητικά λιγότερο αριθμό LUTs.
c) Explore with Remap: Εισάγει τη βελτιστοποίηση remap με σκοπό να μειώσει το μέγεθος της λογικής
d) Runtime Optimized: Κάθε βήμα της διαδικασίας “Implementation” θυσιάζει σχεδιαστική απόδοση για καλύτερο χρόνο μετάδοσης σήματος. Στη στρατηγική αυτή η φυσική βελτιστοποίηση (phys_opt_design) απενεργοποιείται.

2.2.4 ΔΙΑΔΙΚΑΣΙΑ SYNTHESIS

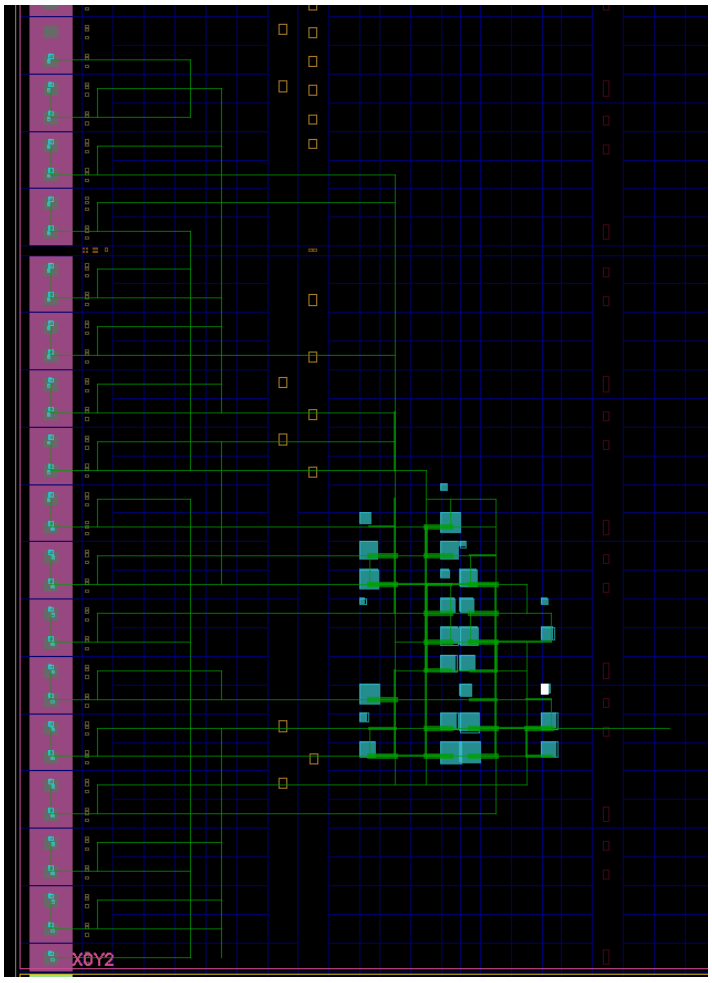
Εφόσον επιλέξουμε τη στρατηγική που θέλουμε να εφαρμόσουμε για τη διαδικασία του “Synthesis” προχωρούμε με την εκτέλεσή της. Το αποτέλεσμα της διαδικασίας είναι ένα RTL σχηματικό διάγραμμα με όλες τις απαραίτητες πύλες για τη διενέργεια της διεργασίας μας. Στην Εικόνα 10 φαίνεται το σχηματικό διάγραμμα λογικών πυλών του 8bit αλγορίθμου πολλαπλασιασμού όπως αυτό προέκυψε από τη διαδικασία synthesis. Η στρατηγική “Synthesis” που χρησιμοποιήθηκε στην Εικόνα 2.10 ήταν η Runtime Optimized.



Εικόνα 2.10 Σχηματικό διάγραμμα λογικών πυλών

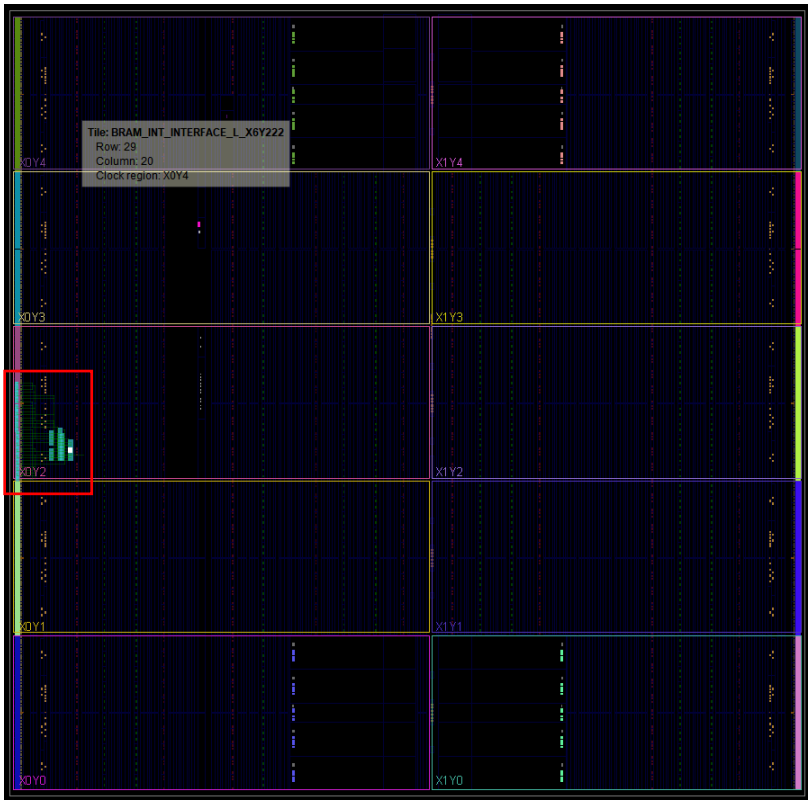
2.2.5 ΔΙΑΔΙΚΑΣΙΑ IMPLEMENTATION

Μετά την περάτωση της διαδικασίας “Synthesis” ακολούθησε η διαδικασία του “Implementation” για τον 8bit αλγόριθμο πολλαπλασιασμού Vedic. Η στρατηγική “Implementation” που επιλέχθηκε ήταν αυτή του “Runtime Optimized”. Το αποτέλεσμα της διαδικασίας του Implementation φαίνεται στην Εικόνα 2.11.



Εικόνα 2.11 Φυσική σχεδίαση 8bit αλγορίθμου Vedic στο FPG

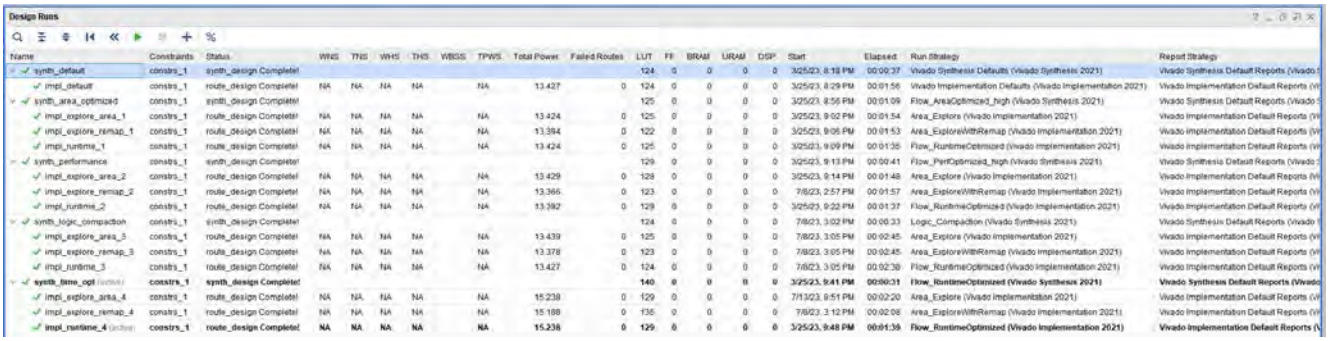
Η Εικόνα 2.12 δείχνει την περιοχή κατάληψης του 8bit αλγορίθμου Vedic σε σχέση με το μέγεθος του FPGA.



Εικόνα 2.12 Περιοχή κατάληψης του 8bit αλγορίθμου πολλαπλασιασμού Vedic από την περιοχή του FPGA

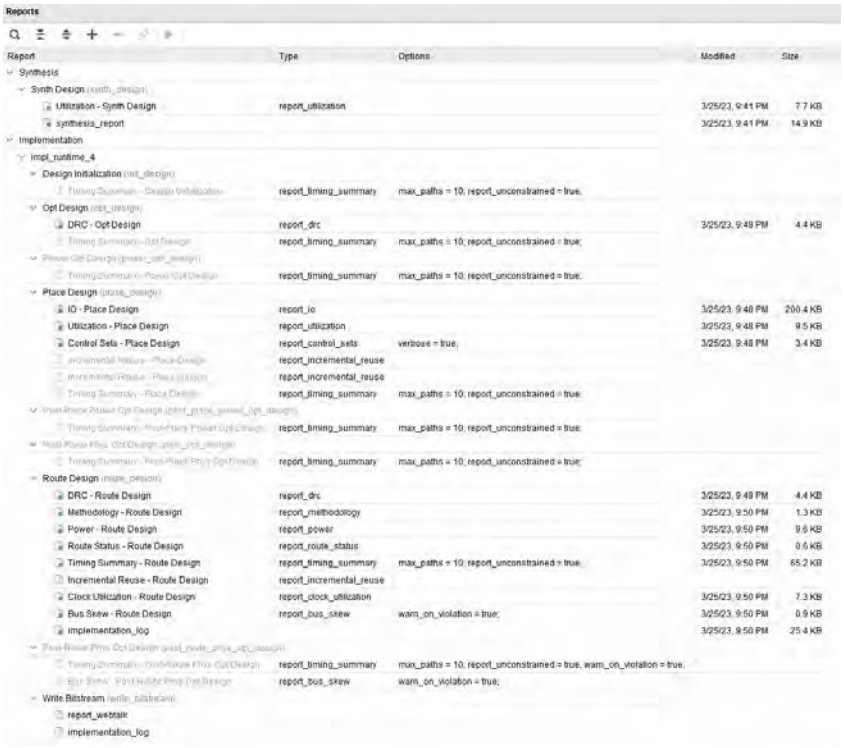
2.2.6 DESIGN FLOWS & ΑΝΑΦΟΡΕΣ

Για τον 8bit αλγόριθμό με τον ίδιο τρόπο που περιγράφηκε παραπάνω έγιναν και οι υπόλοιπες προσομοιώσεις των διάφορων συνδυασμών στρατηγικών “Synthesis” και “Implementation”. Όλοι οι συνδυασμοί των στρατηγικών ανέρχονται συνολικά σε δεκαοχτώ προσομοιώσεις “Synthesis” και “Implementation” για κάθε αλγόριθμο. Οι προσομοιώσεις αυτές για τον 8bit αλγόριθμο πολλαπλασιασμού Vedic παρουσιάζονται στην Εικόνα 2.13.



Εικόνα 2.13 Προσομοιώσεις “Synthesis” και “Implementation” 8bit αλγορίθμου πολλαπλασιασμού Vedic

Στη συνέχεια έγινε η εξαγωγή των αναφορών με βάση τις οποίες έγινε παραγωγή των αποτελεσμάτων της παρούσας διπλωματικής εργασίας στο επόμενο κεφάλαιο. Η καρτέλα των αναφορών κάθε συνδυασμού προσομοιώσεων “Synthesis” και “Implementation” φαίνεται στην Εικόνα 2.14.



Εικόνα 2.14 Αναφορές προγράμματος προσομοίωσης Vivado για τις προσομοιώσεις “Synthesis” και “Implementation”

Με την ίδια λογική που περιγράφηκε παραπάνω για τις προσομοιώσεις “Synthesis” και “Implementation” του 8bit αλγορίθμου πολλαπλασιασμού Vedic έγιναν και οι προσομοιώσεις όλων των υπόλοιπων αλγορίθμων. Από την εξαγωγή των αναφορών των προσομοιώσεων αυτών παράχθηκαν τα αποτελέσματα που παρουσιάζονται στο επόμενο κεφαλαιο.

ΚΕΦΑΛΑΙΟ 3

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΕΩΝ

Η παρουσίαση των αποτελεσμάτων θα γίνει με κατηγοριοποίηση των αλγορίθμων βάσει του αριθμού των bits που καλούνται να πολλαπλασιάσουν. Οι γραφικές παραστάσεις που θα παρατεθούν έχουν να κάνουν με:

- τον χρόνο προσομοίωσης της στρατηγικής “Synthesis”
- με την κατανάλωση ισχύος
- με τη συσχέτιση κατανάλωσης ισχύος και “Implementation” LUTs
- με τη συσχέτιση “Synthesis” LUTs και “Implementation” LUTs

Τέλος γίνεται σύγκριση της μέγιστης, ελάχιστης και μέσης τιμής μεταξύ του αλγορίθμου Dadda διαφορετικών bits σε:

- χρόνο προσομοίωσης της στρατηγικής “Synthesis”
- κατανάλωση ισχύος
- αριθμό “Synthesis” LUTs
- αριθμό “Implementation” LUTs

Οι αλγόριθμοι που προσομοιώθηκαν καθώς και ο αριθμός εμφάνισής τους σε δύο ή περισσότερες κατηγορίες αλγορίθμων ανάλογα με τον αριθμό των bits παρουσιάζονται στον Πίνακας 3.1. Στον ίδιο πίνακα φαίνεται ο αριθμός εμφάνισης τους καθώς και η κατηγορία των bits πολλαπλασιασμού που εμφανίζονται.

Πίνακας 3.1 Οι Υπό εξέταση Αλγόριθμοι Πολλαπλασιασμού

Αλγόριθμος	8bit	16bit	32bit	64bit	Αριθμός Εμφάνισης
Booth	✓			✓	2
Dadda	✓	✓	✓	✓	4
Vedic	✓		✓		2
Serial – Serial		✓			1

Αλγόριθμος	8bit	16bit	32bit	64bit	Αριθμός Εμφάνισης
Wallace Tree		✓	✓		2
Array				✓	1

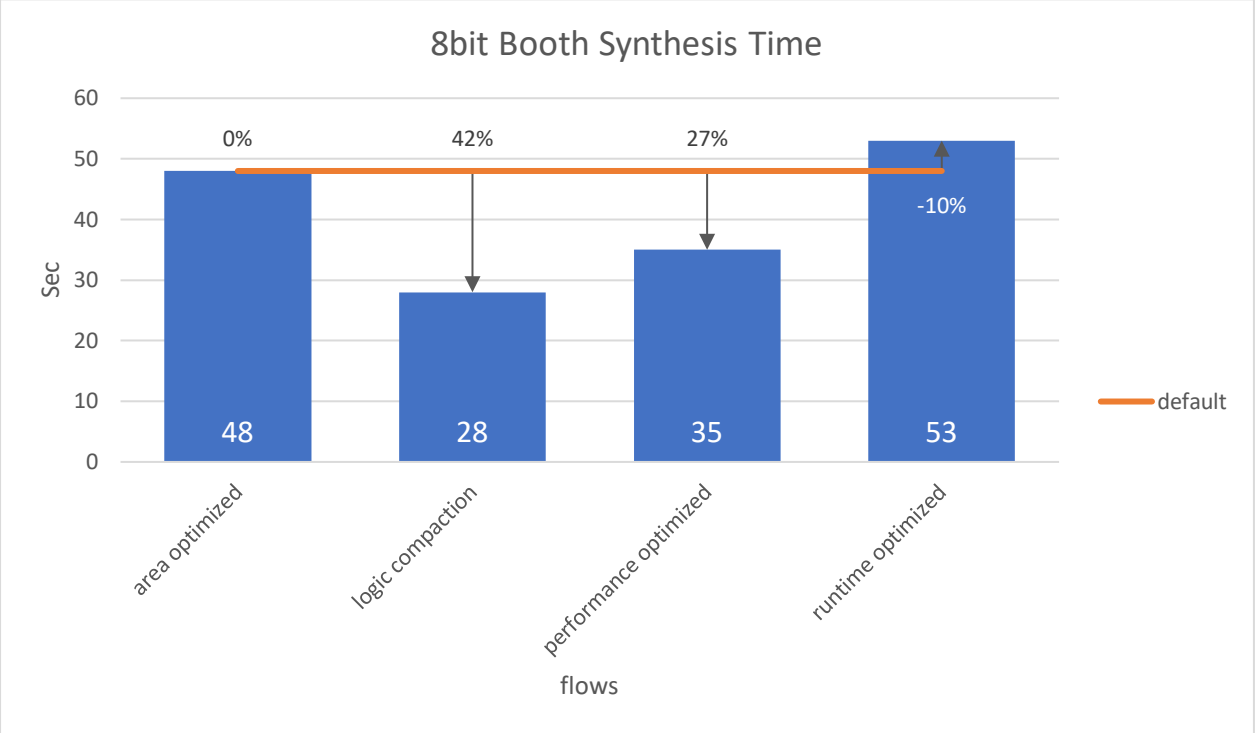
3.1 ΧΡΟΝΟΣ ΠΡΟΣΟΜΟΙΩΣΗΣ “SYNTHESIS”

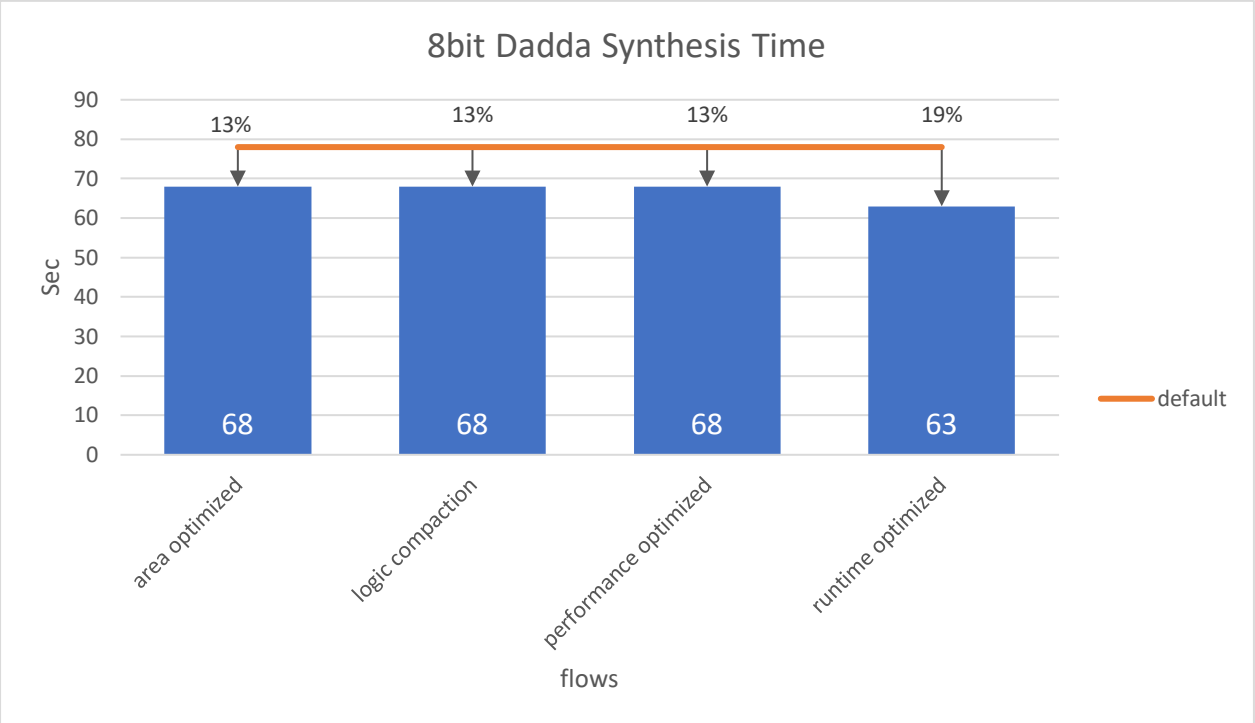
Παρακάτω παρατίθενται οι γραφικές παραστάσεις του χρόνου προσομοίωσης της διαδικασίας “Synthesis” για τους παραπάνω αλγόριθμους. Η στρατηγικές “Synthesis” που εφαρμόστηκαν στον καθένα από αυτούς αναλύθηκαν στο προηγούμενο κεφαλαιο.

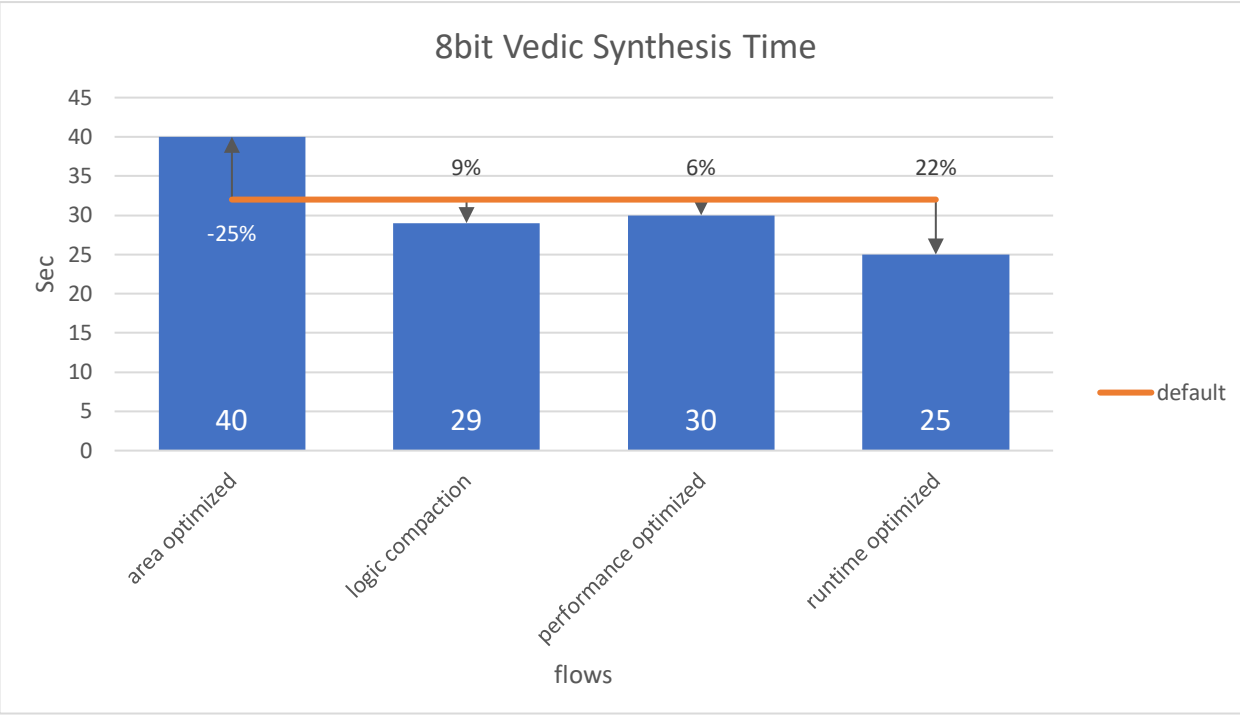
3.1.1 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 8BIT

Οι 8bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Booth
- O Dadda
- O Vedic



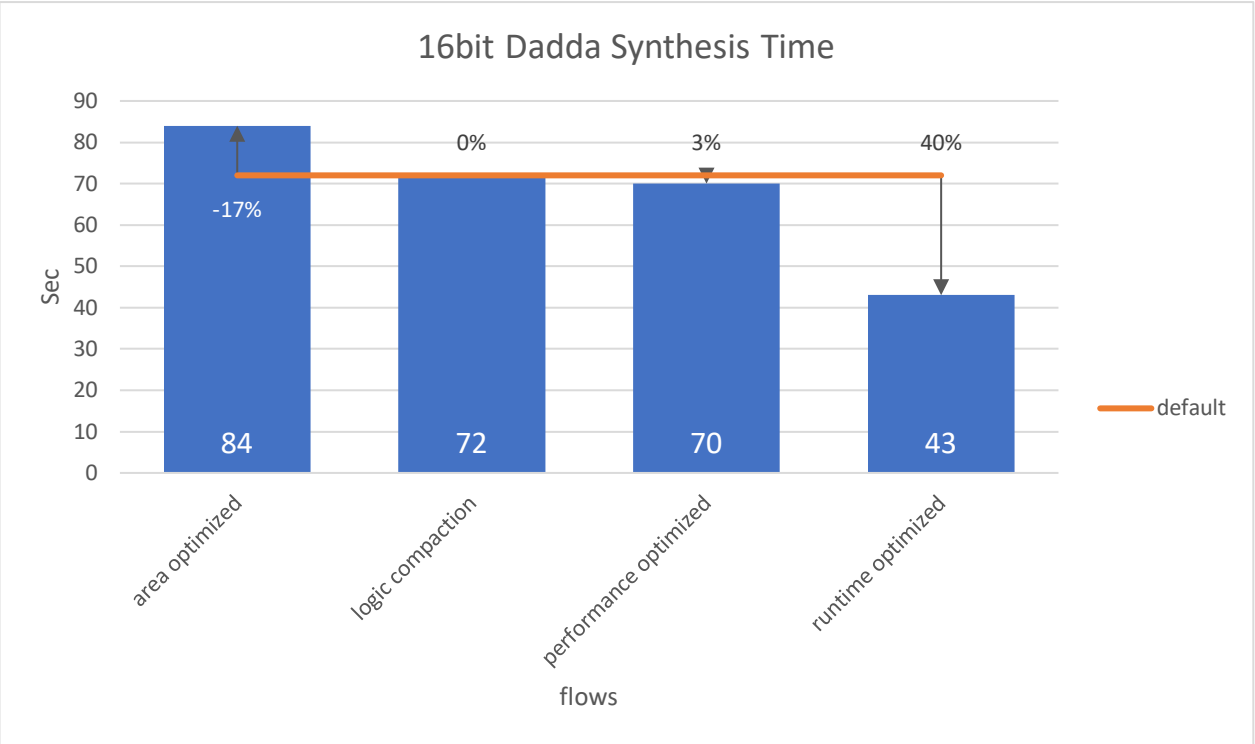


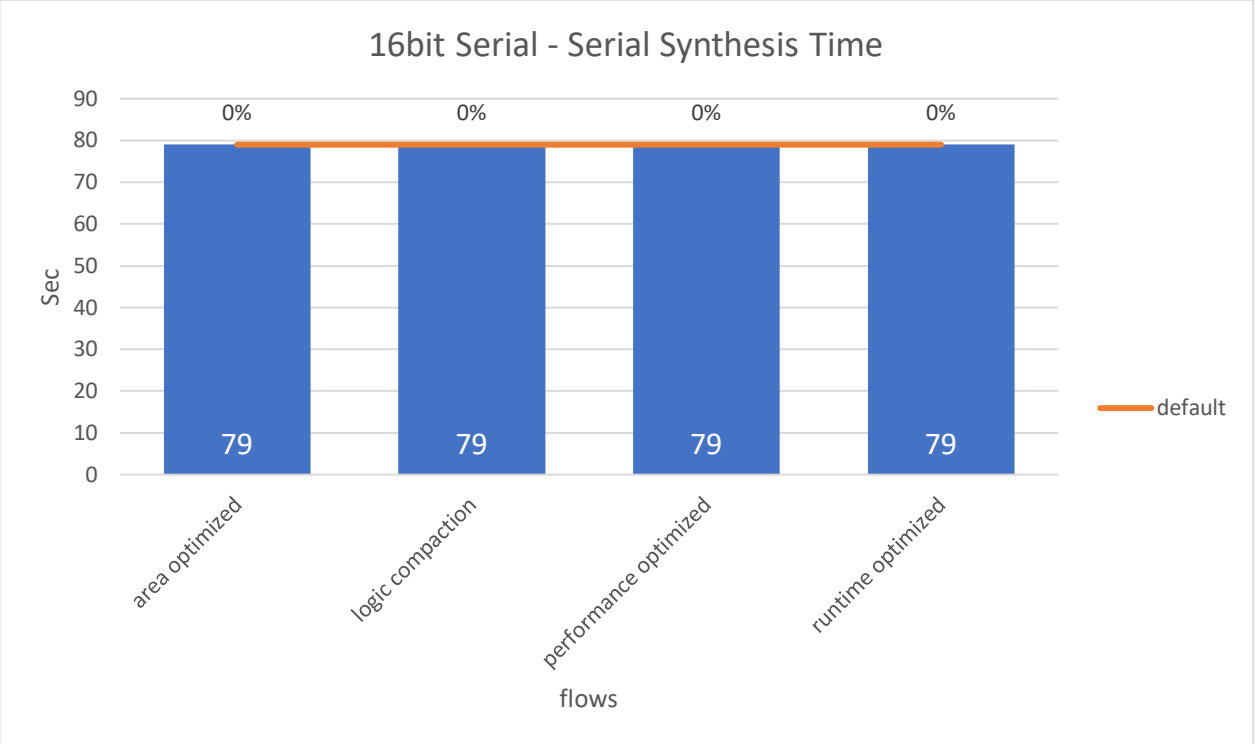


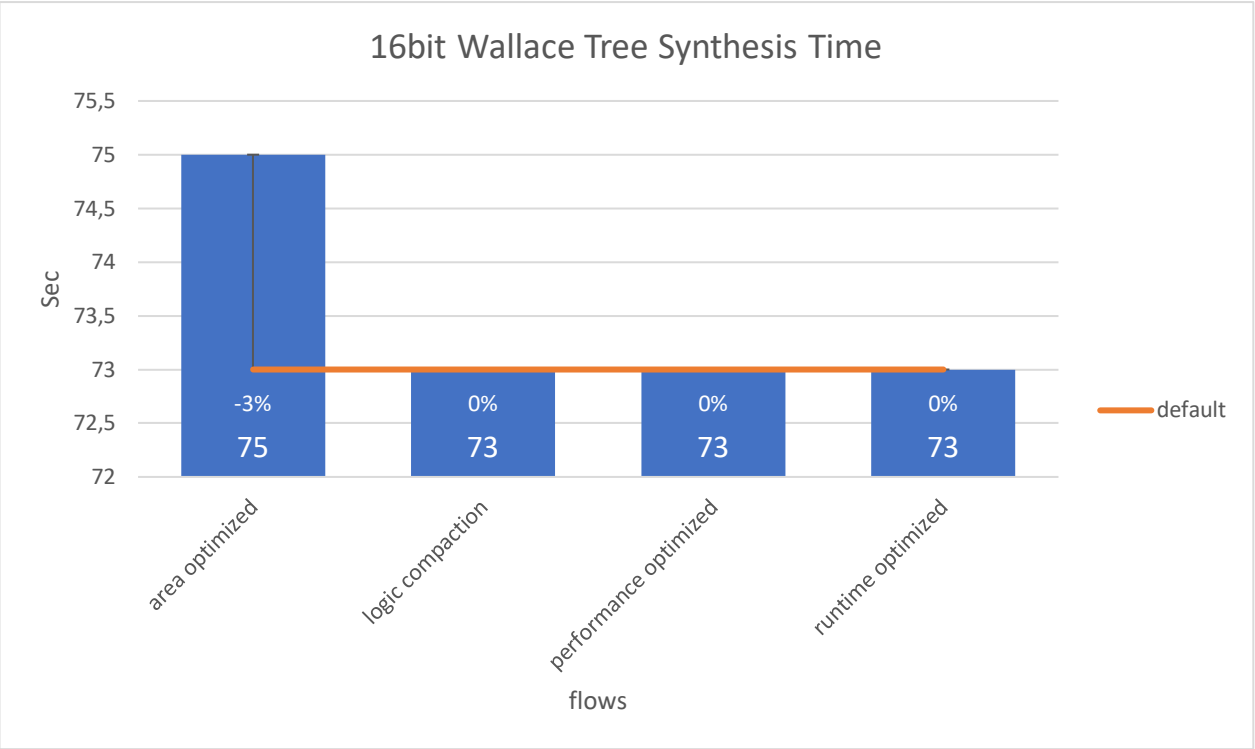
3.1.2 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 16BIT

Οι 16bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Dadda
- O Serial – Serial
- O Wallace Tree



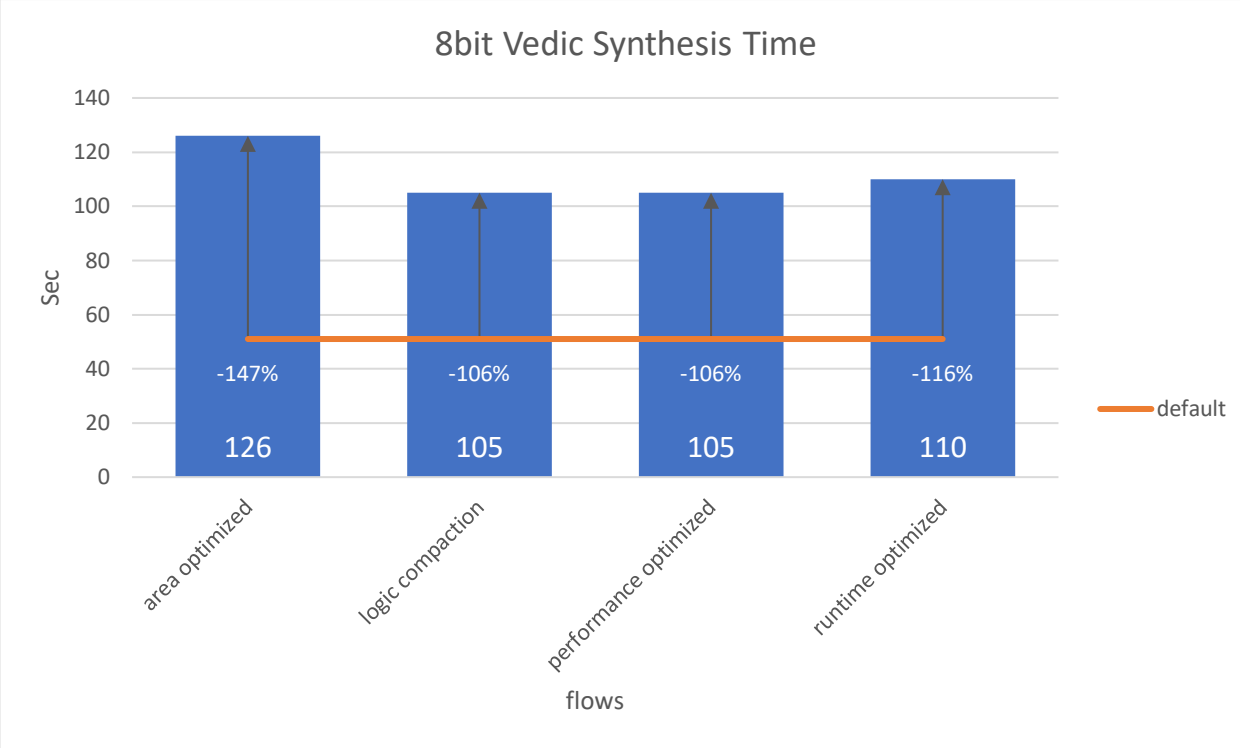


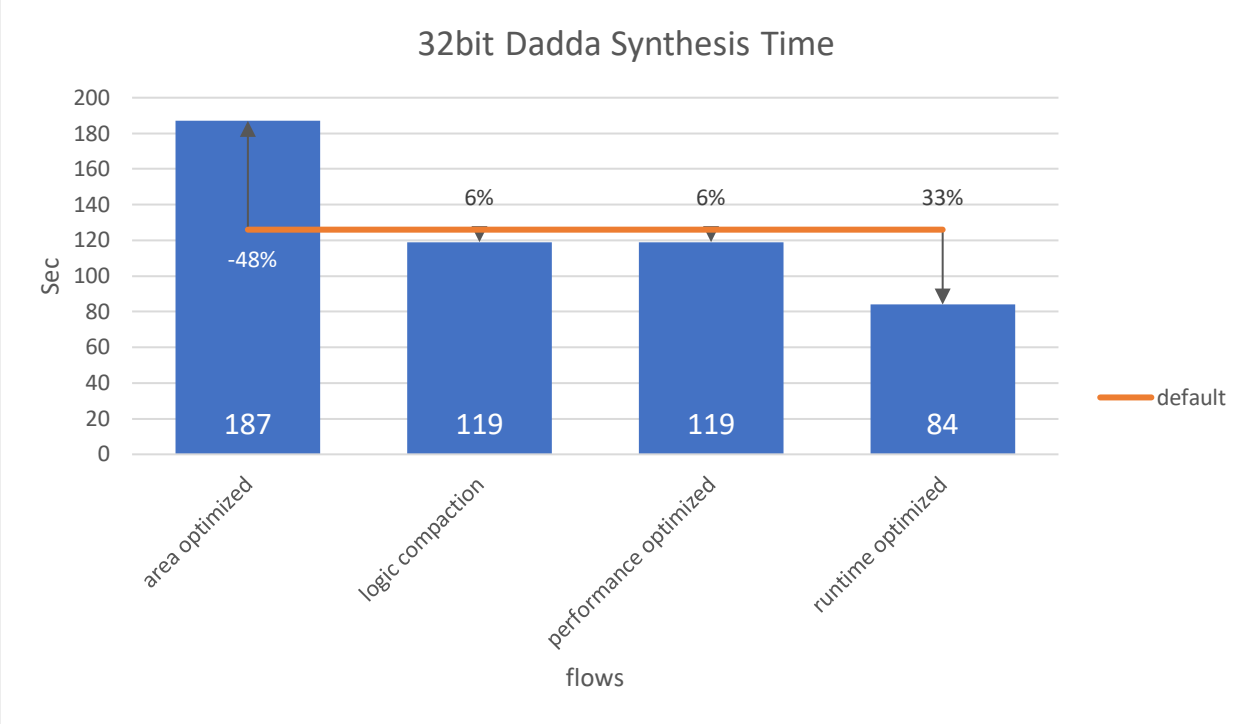


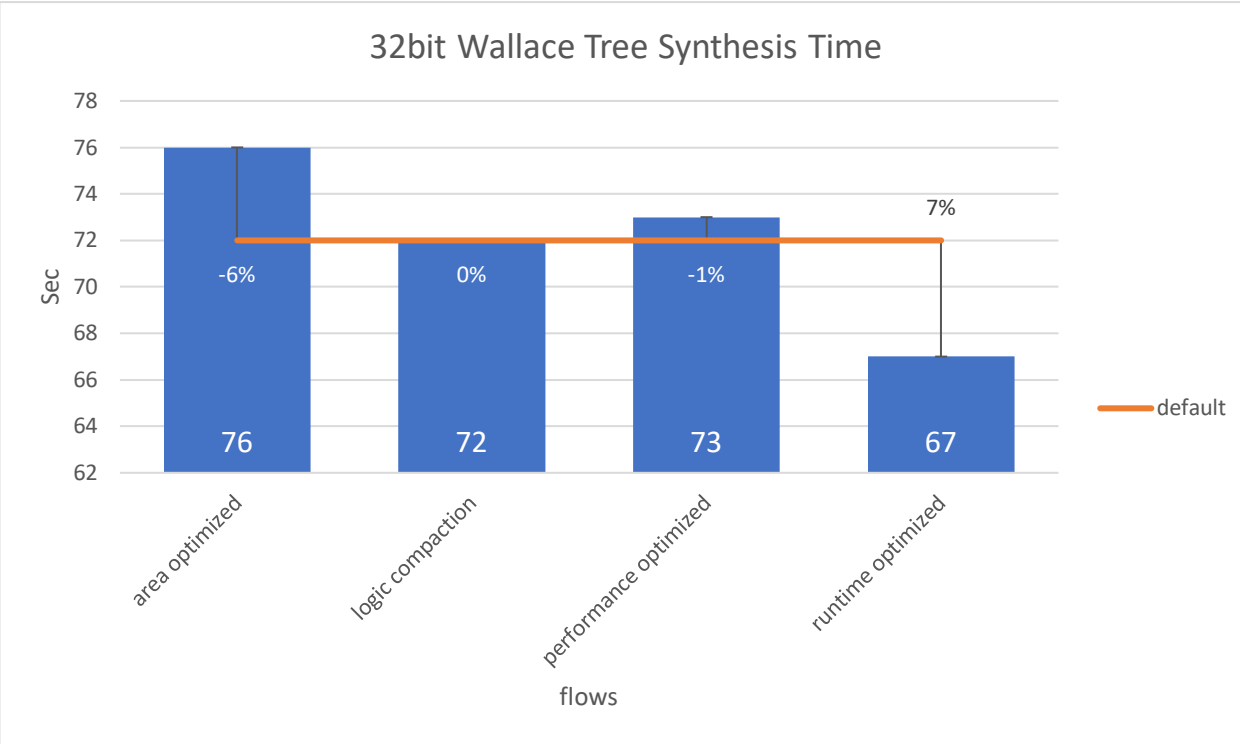
3.1.3 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 32BIT

Οι 32bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Vedic
- O Dadda
- O Wallace Tree



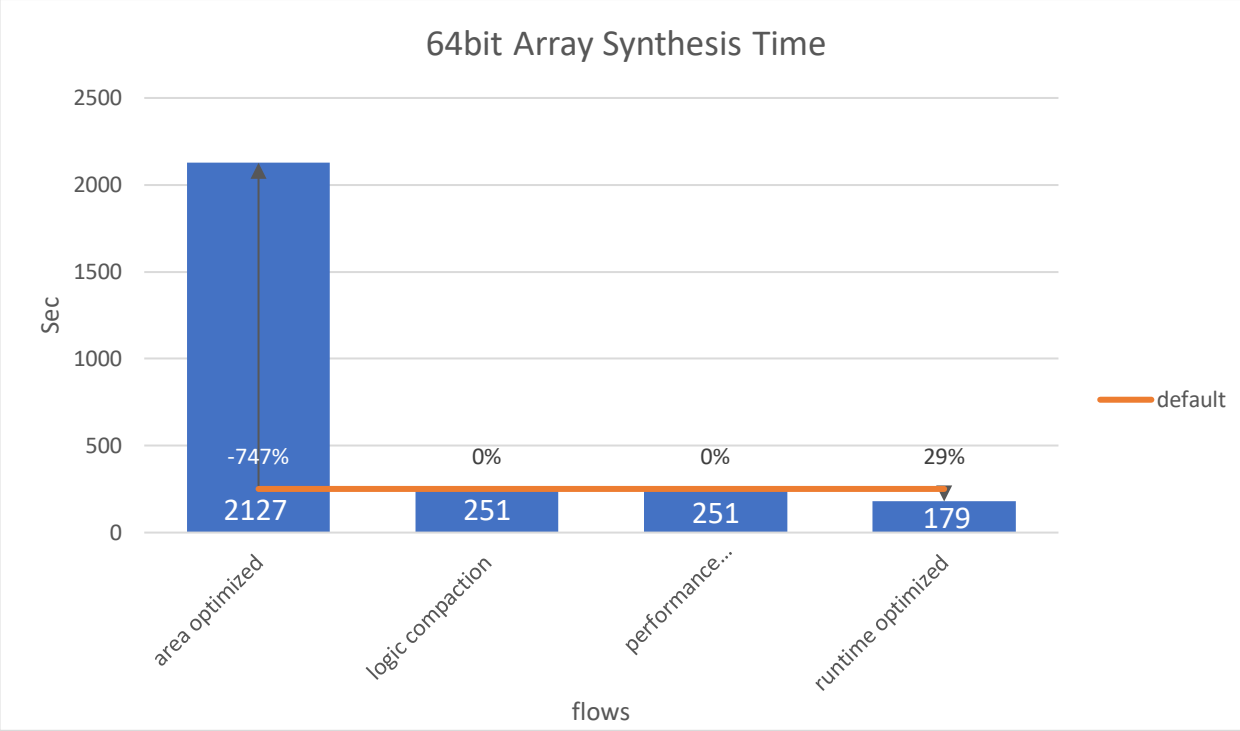


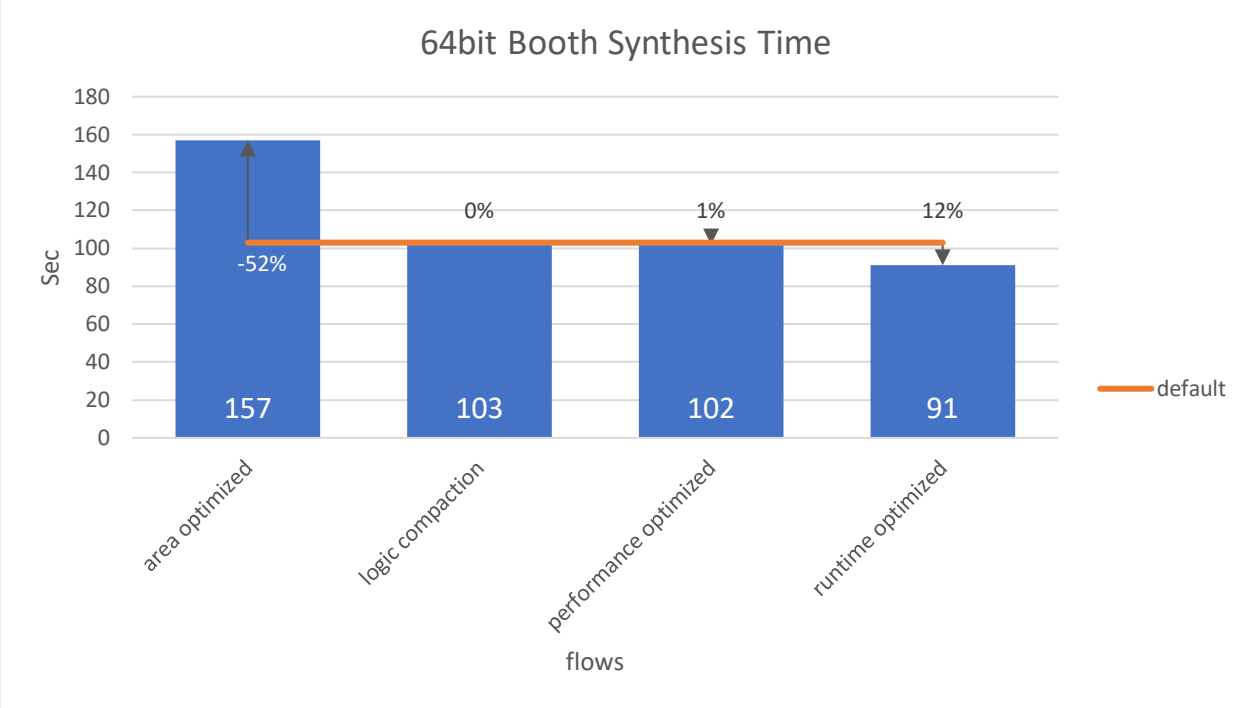


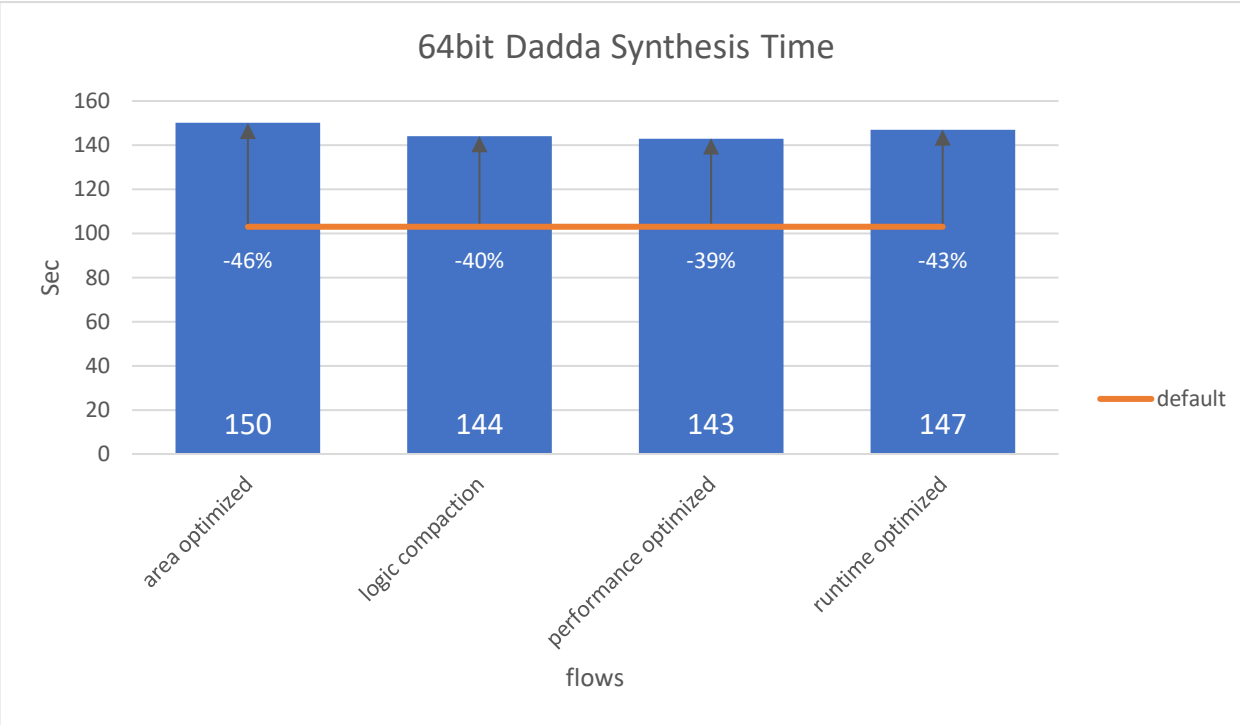
3.1.4 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 64BIT

Οι 64bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Array
- O Booth
- O Dadda







3.1.5 ΣΥΜΠΕΡΑΣΜΑΤΑ ΧΡΟΝΟΥ ΠΡΟΣΟΜΟΙΩΣΗΣ "SYNTHESIS"

Στον Πίνακα 1 βλέπουμε τη συχνότητα που παρατηρήθηκε αύξηση, μείωση ή μηδενική μεταβολή του χρόνου προσομοίωσης "Synthesis" ανάλογα με τη στρατηγική προσομοίωσης για τους παραπάνω αλγορίθμους. Οι συγκρίσεις έγιναν με βάση τον χρόνο προσομοίωσης της "default" στρατηγικής "Synthesis".

Πίνακας 3.2 Συχνότητα Μεταβολής Χρόνου "Synthesis"

Στρατηγική "Synthesis"	Αύξηση (%)	Μηδενική μεταβολή (%)	Μείωση (%)
explore area	75%	17%	8%
logic compaction	17%	50%	33%
performance optimized	25%	25%	50%
runtime optimized	25%	17%	58%

Σύμφωνα με τον παραπάνω πίνακα παρατηρούμε ότι η μεγαλύτερη συχνότητα αύξησης χρόνου προσομοίωσης “Synthesis” παρατηρείται στη στρατηγική “Explore Area”, ενώ η μεγαλύτερη συχνότητα μείωσης του χρόνου προσομοίωσης “Synthesis” παρατηρείται στη στρατηγική “Runtime Optimized”. Η στρατηγική “Synthesis” “Logic Compaction” κατέχει το μεγαλύτερο ποσοστό μηδενικής μεταβολής χρόνου προσομοίωσής σε σχέση με τη “default” στρατηγική “Synthesis”.

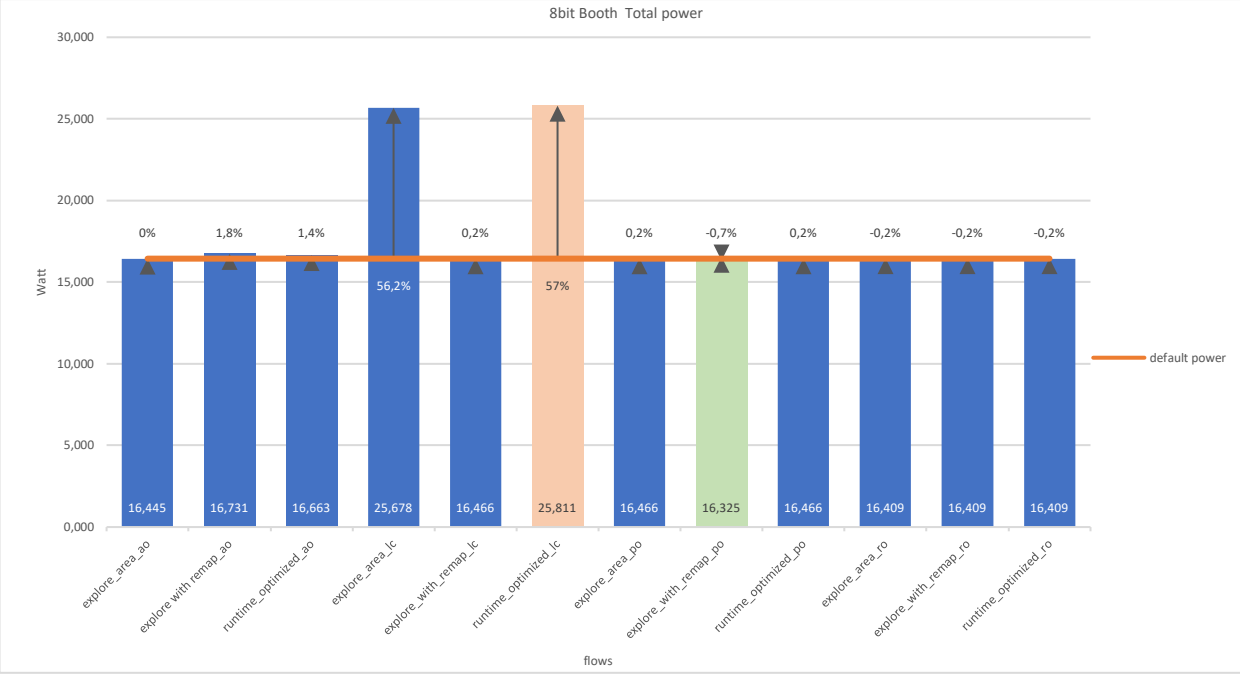
3.2 ΚΑΤΑΝΑΛΩΣΗ ΙΣΧΥΟΣ

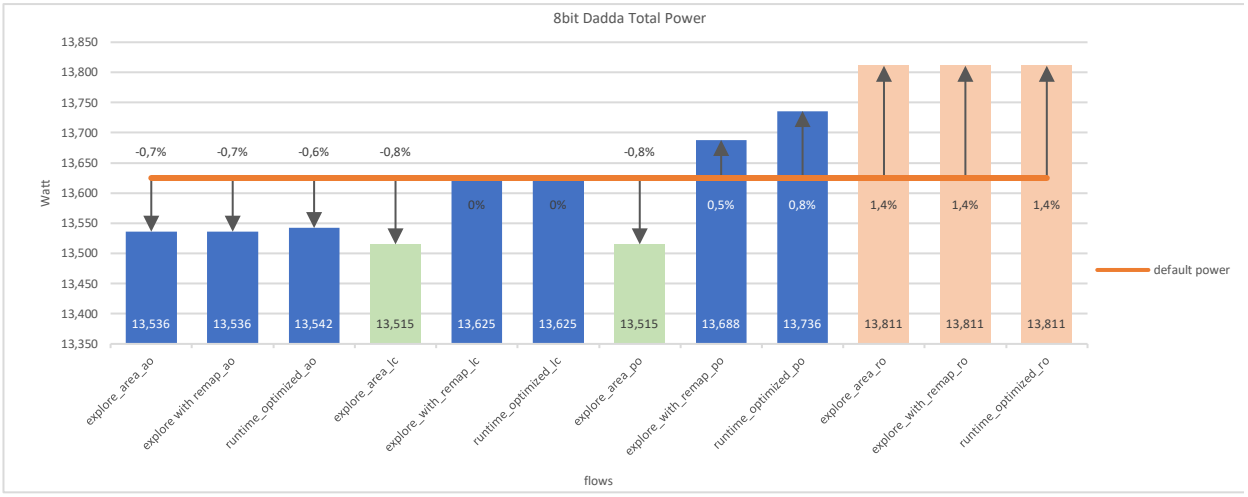
Παρακάτω παρατίθενται οι γραφικές παραστάσεις της κατανάλωσης ισχύος των υπό εξέταση αλγορίθμων. Οι στρατηγικές “Synthesis” και “Implementation” που εφαρμόστηκαν στον καθένα από αυτούς αναλύθηκαν στο προηγούμενο κεφάλαιο.

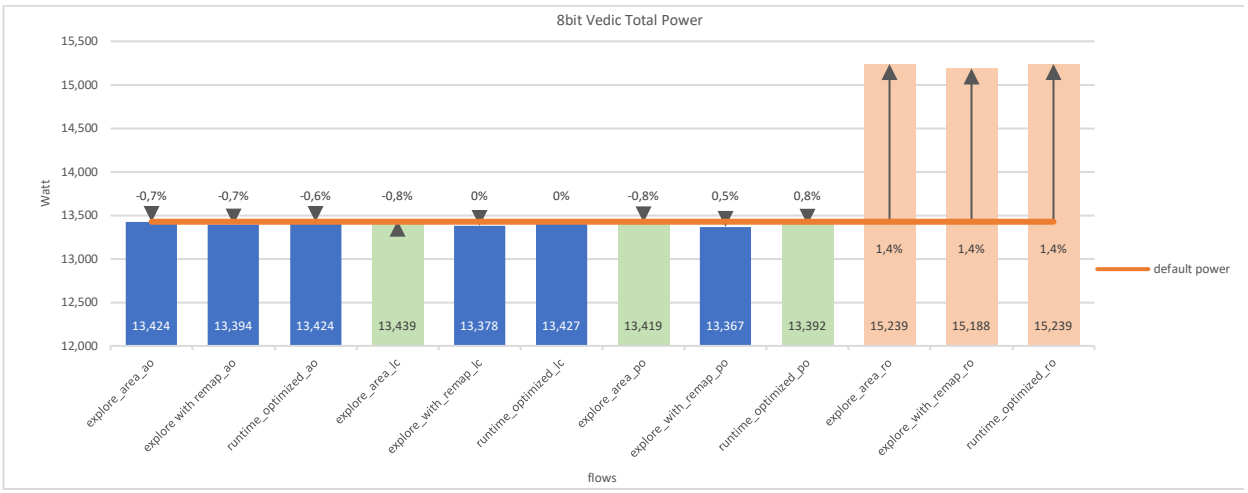
3.2.1 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 8BIT

Οι 8bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Booth
- O Dadda
- O Vedic



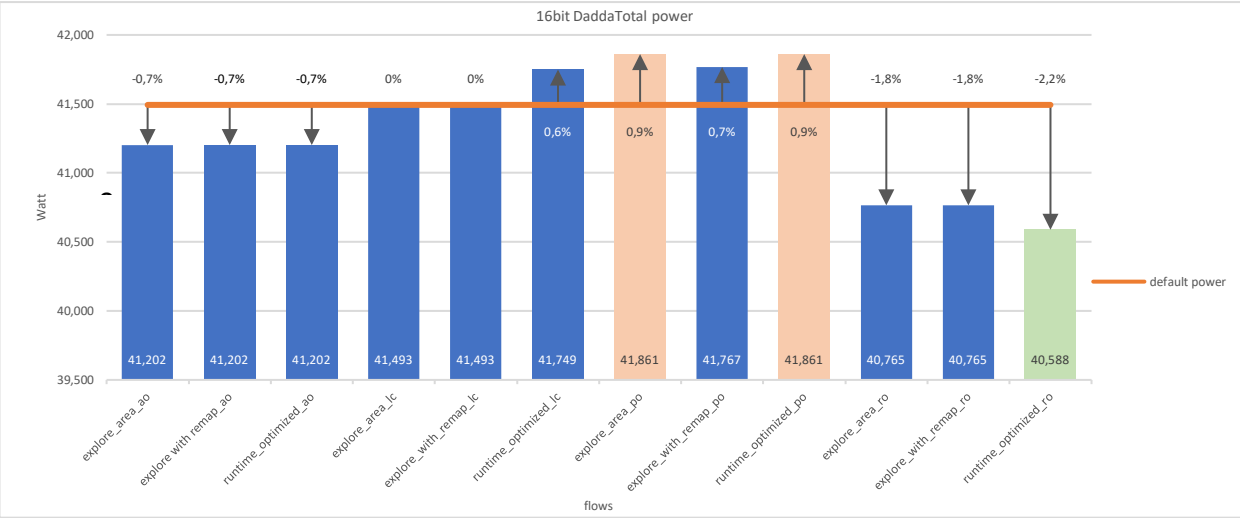


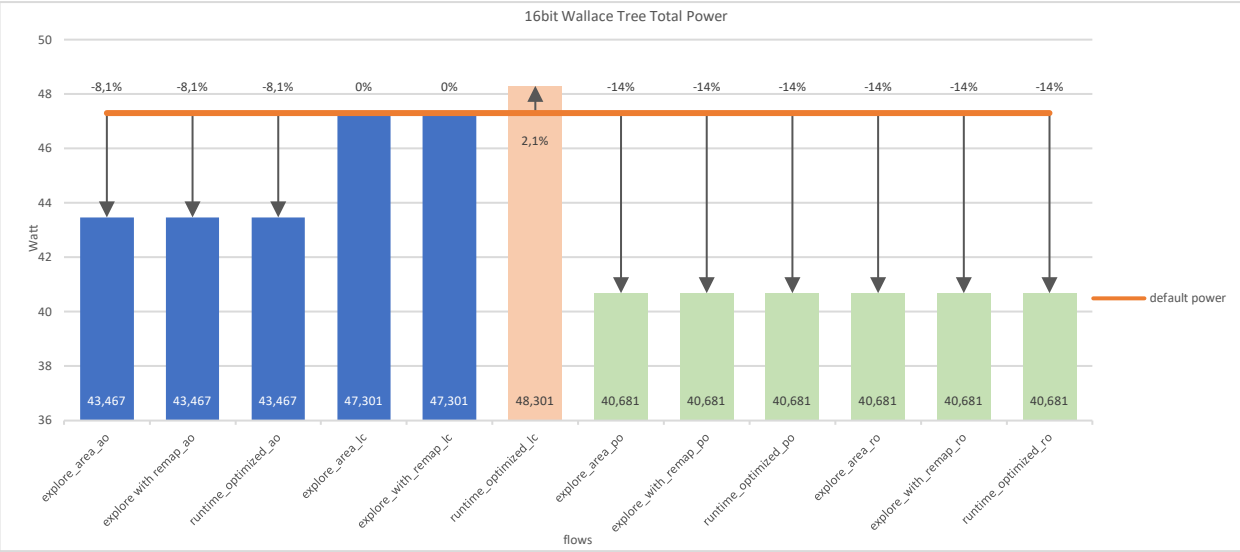


3.2.2 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 16BIT

Οι 16bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Dadda
- O Serial – Serial
- O Wallace Tree

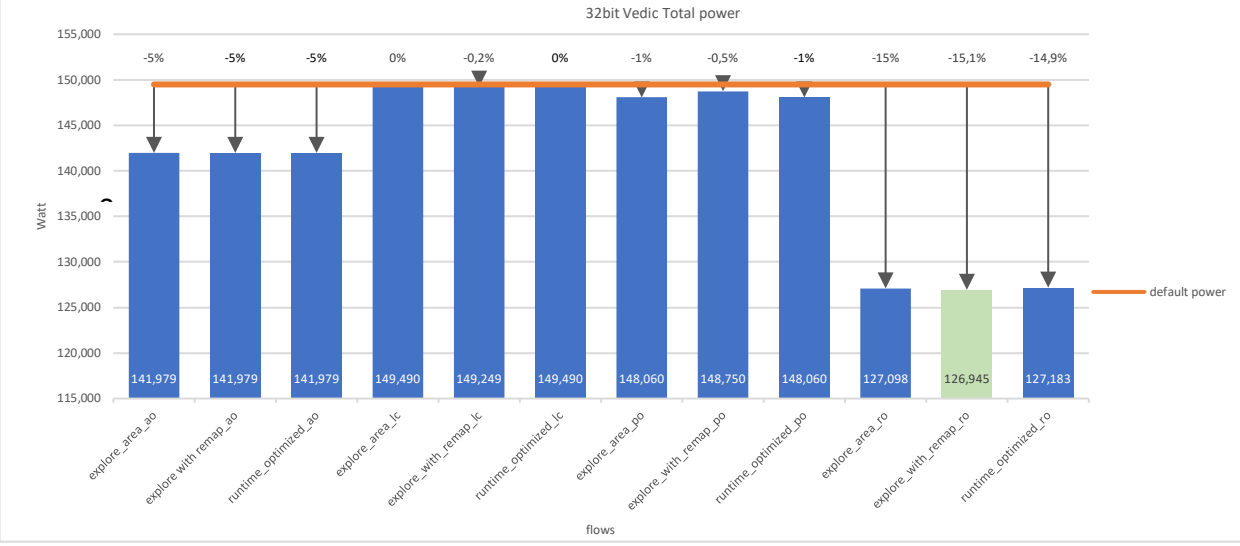


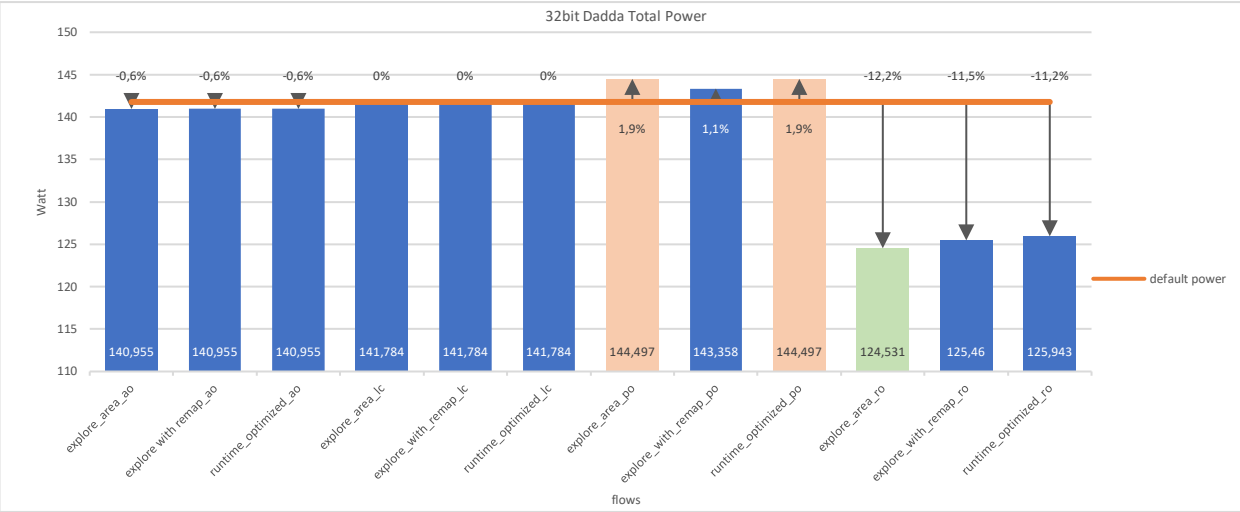


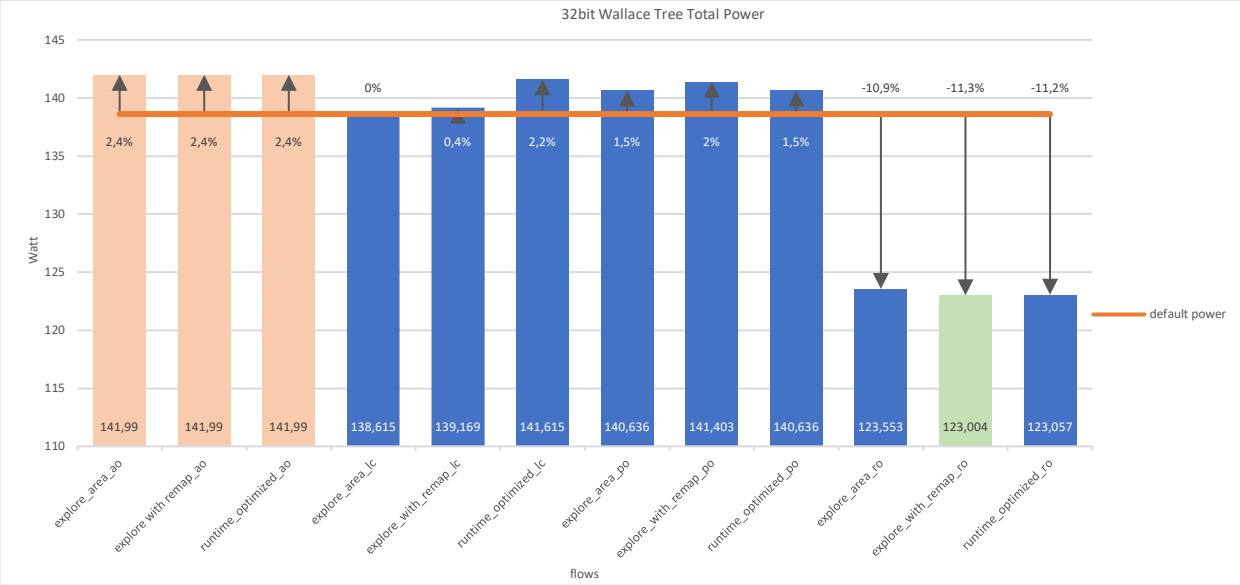
3.2.3 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 32BIT

Οι 32bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- Ο Vedic
- Ο Dadda
- Ο Wallace Tree



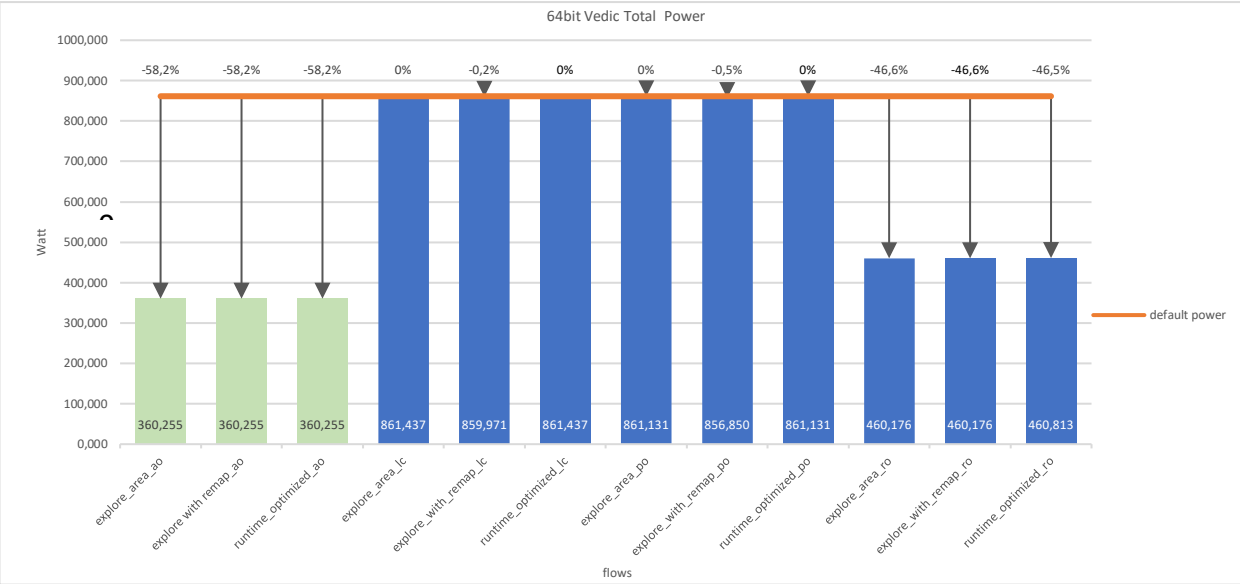


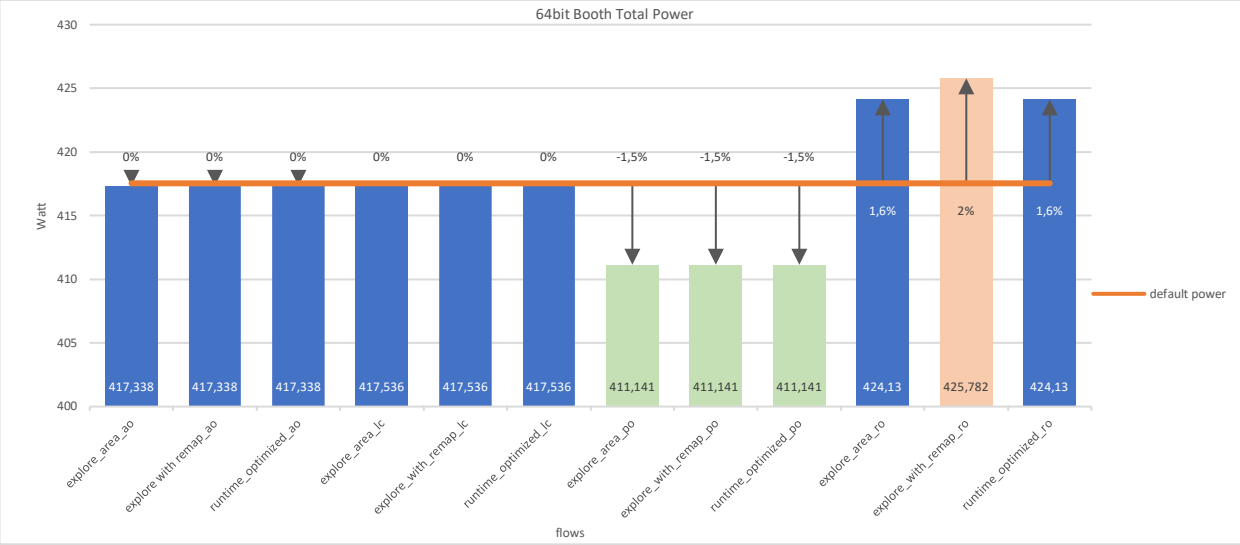


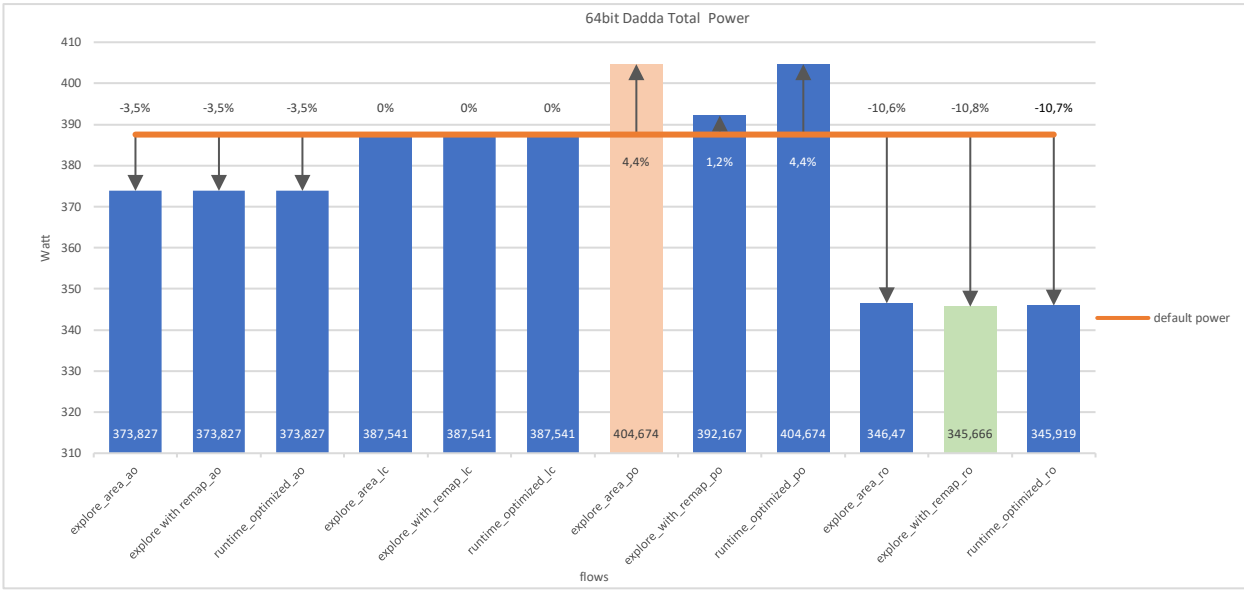
3.2.4 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 64BIT

Οι 64bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- Ο Array
- Ο Booth
- Ο Dadda







3.2.5 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΤΑΝΑΛΩΣΗΣ ΙΣΧΥΟΣ

Στον Πίνακα 3.3 βλέπουμε τη συχνότητα που παρατηρήθηκε αύξηση, μείωση ή μηδενική μεταβολή της κατανάλωσης ισχύος ανάλογα με τη στρατηγική προσομοίωσης “Synthesis” και “Implementation για τους υπό μελέτη αλγορίθμους πολλαπλασιασμού. Οι συγκρίσεις έγιναν με βάση την κατανάλωση των “default” στρατηγικών “Synthesis” και “Implementation”.

Πίνακας 3.3 Συχνότητα Μεταβολής κατανάλωσης ισχύος

Στρατηγική “Synthesis	Στρατηγική “Implementation”	Αύξηση(%)	Μηδενική Μεταβολή (%)	Μείωση (%)
Area Optimized	Explore Area	8%	17%	75%
	Explore with Remap	17%	8%	75%
	Runtime Optimized	17%	8%	75%
Logic Compaction	Explore Area	17%	75%	8%
	Explore with Remap	17%	66%	17%

Στρατηγική "Synthesis	Στρατηγική "Implementation"	Αύξηση(%)	Μηδενική Μεταβολή (%)	Μείωση (%)
	Runtime Optimized	33%	67%	0%
Performance Optimized	Explore Area	50%	8%	42%
	Explore with Remap	50%	0%	50%
	Runtime Optimized	58%	8%	34%
Runtime Optimized	Explore Area	75%	0%	25%
	Explore with Remap	75%	0%	25%
	Runtime Optimized	75%	0%	25%

Σύμφωνα με τον παραπάνω Πίνακα παρατηρούμε πως η μεγαλύτερη συχνότητα αύξησης της κατανάλωσης συμβαίνει κατά τη "Runtime Optimized" "Synthesis" στρατηγική και τις στρατηγικές "Implementation", "Explore Area", "Explore with Remap" και "Runtime Optimized" που έτρεξαν βάσει αυτής.

Αντίθετα το μεγαλύτερο ποσοστό συχνότητας μείωσης της κατανάλωσης παρατηρήθηκε κατά τη στρατηγική "Synthesis" "Explore Area" και τις στρατηγικές "Implementation", "Explore Area", "Explore with Remap" και "Runtime Optimized" που έτρεξαν βάσει αυτής.

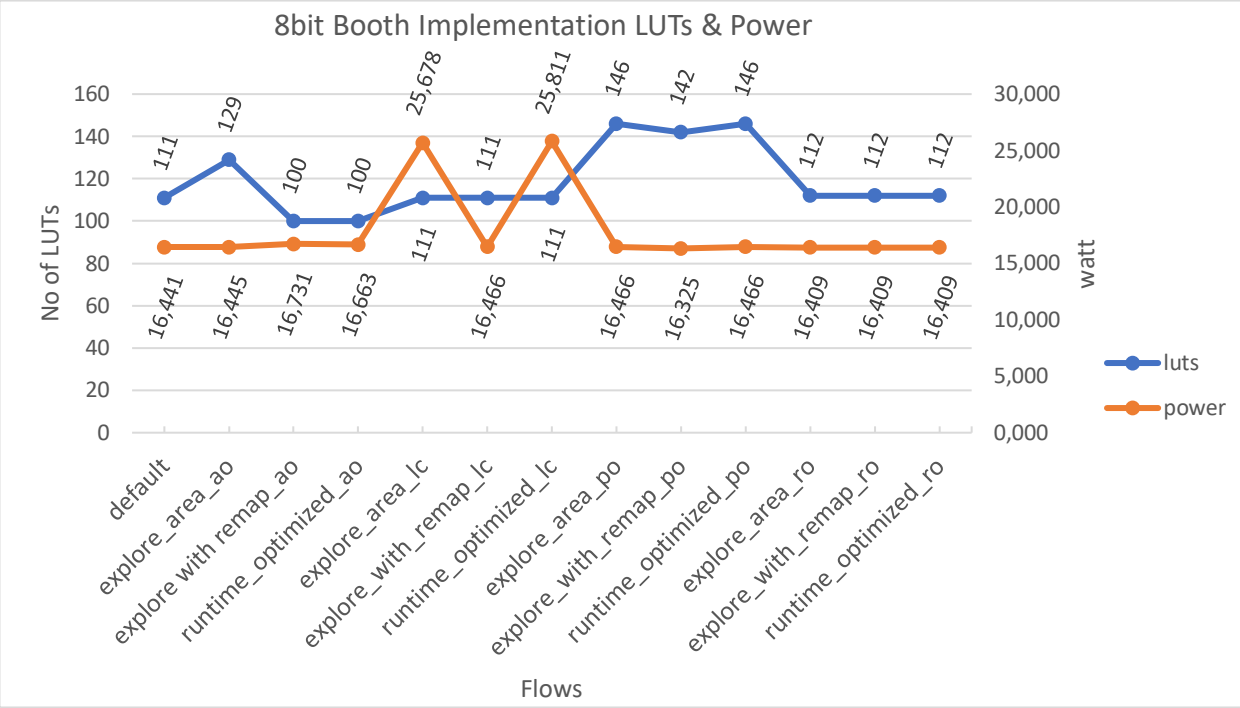
Τέλος το μεγαλύτερο ποσοστό συχνότητας μηδενικής μεταβολής της κατανάλωσης παρατηρήθηκε κατά τη στρατηγική "Synthesis" "Logic Compaction" και τη στρατηγική "Implementation" "Explore Area" που έτρεξε βάσει αυτής.

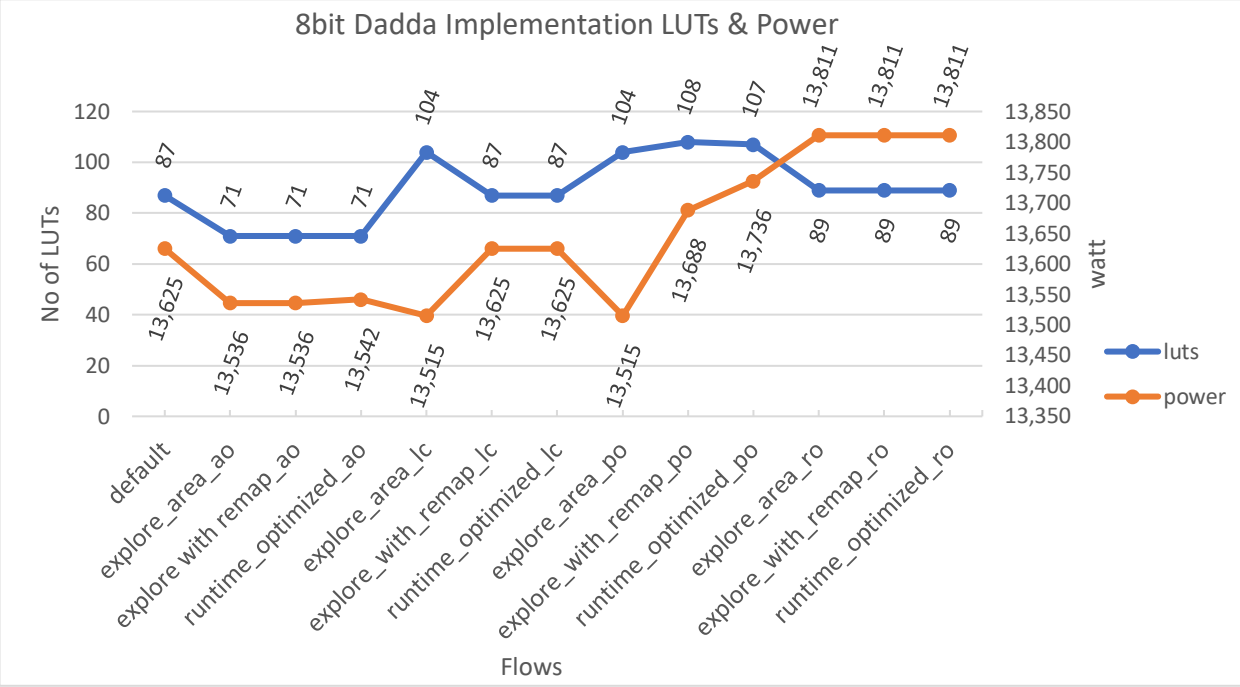
3.3 ΣΥΣΧΕΤΙΣΗ ΑΡΙΘΜΟΥ "IMPLEMENTATION" LUTS ΚΑΙ ΚΑΤΑΝΑΛΩΣΗΣ ΙΣΧΥΟΣ

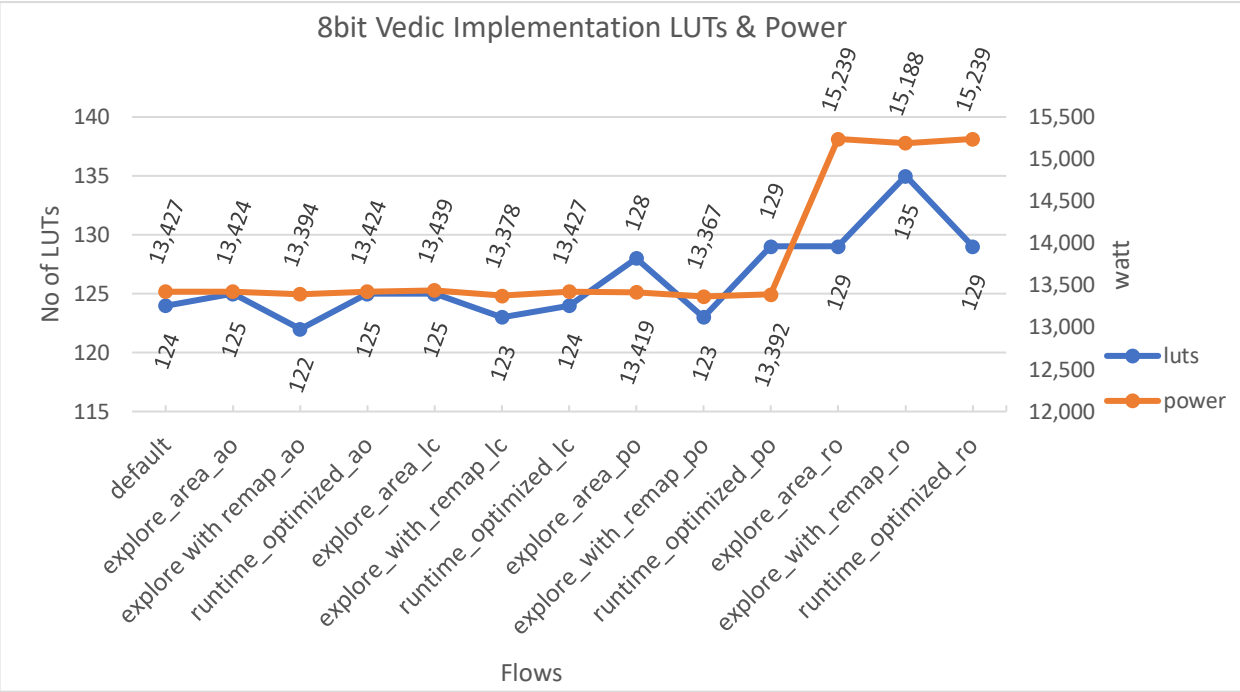
3.3.1 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 8BIT

Οι 8bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- O Booth
- O Dadda
- O Vedic







Ο Πίνακας 3.4 δείχνει τη συνάφεια της μεταβολής Implementation LUTs και ισχύος για όλους τους συνδυασμούς στρατηγικών των τριών 8bit αλγορίθμων.

Πίνακας 3.4 Συνάφεια "Implementation LUTs & Ισχύος για τους 8bit αλγορίθμους

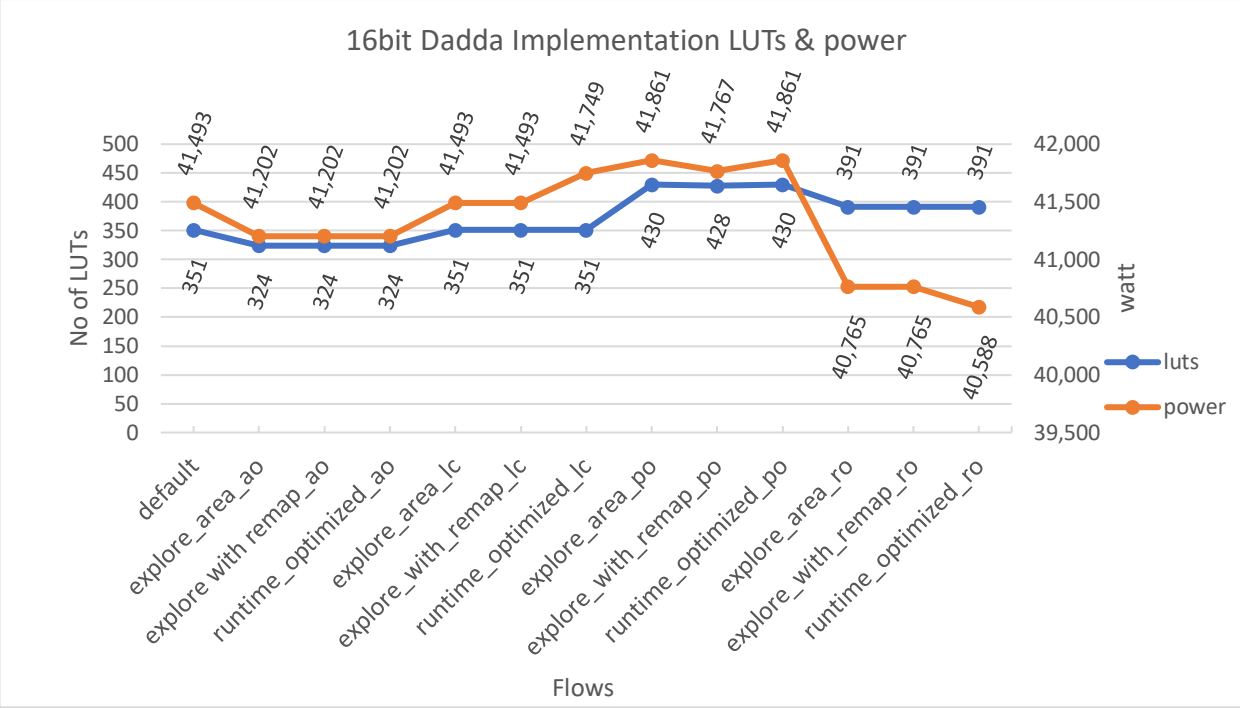
8bit Αλγόριθμος	**Στρατηγική "Synthesis**	**Στρατηγική "Implementation"**	**LUTs**	**Power**	**Ποσοστό Συνάφειας (%)**
Booth	Default	Default	-	-	-
	Area Optimized	Explore Area	↗	↗	✓
		Explore with Remap	↘	↗	✗
		Runtime Optimized	→	↘	✗
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	↘	✗
		Runtime Optimized	→	↗	✗
	Performance Optimized	Explore Area	↗	↘	✗
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Booth					**58%**
Dadda	Default	Default	-	-	
	Area Optimized	Explore Area	↘	↘	✓

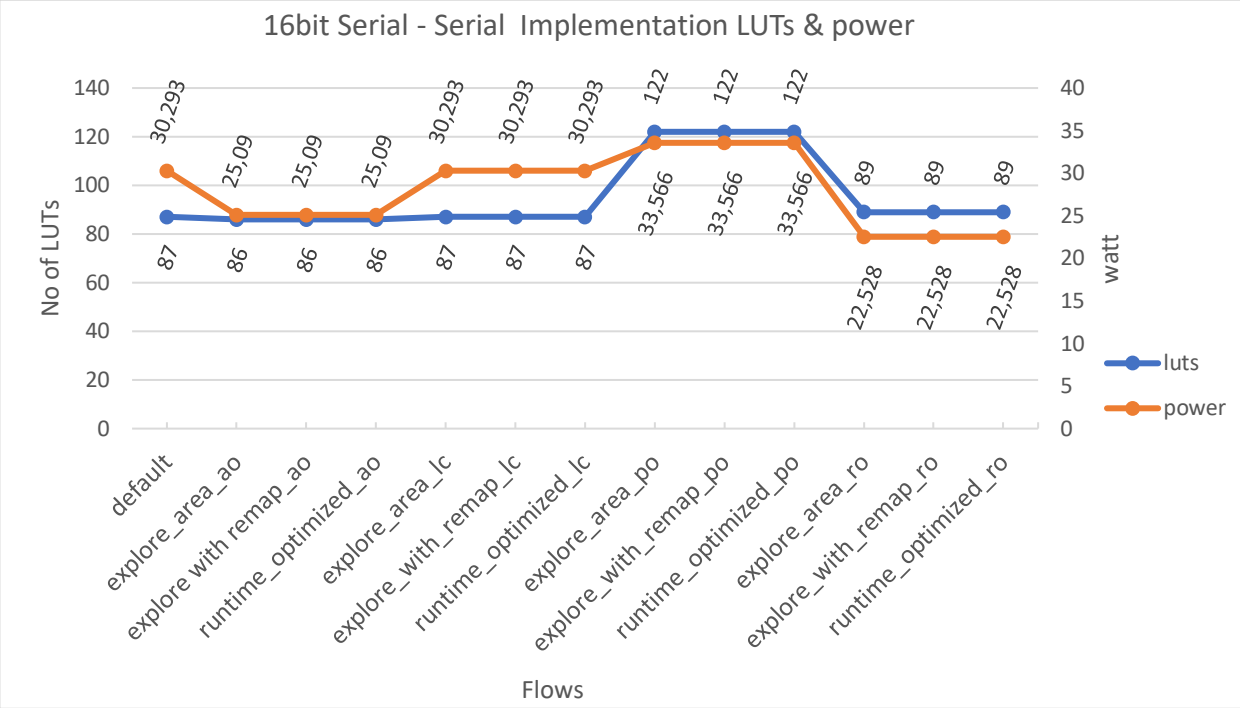
8bit Αλγόριθμος	Στρατηγική "Synthesis	Στρατηγική "Implementation"	LUTs	Power	Ποσοστό Συνάφειας (%)
		Explore with Remap	→	→	✓
		Runtime Optimized	→	↗	✘
	Logic Compaction	Explore Area	↗	↘	✘
		Explore with Remap	↘	↗	✘
		Runtime Optimized	→	→	✓
	Performance Optimized	Explore Area	↗	↘	✘
		Explore with Remap	↗	↗	✓
		Runtime Optimized	↘	↗	✘
	Runtime Optimized	Explore Area	↘	↗	✘
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Dadda					**50%**
Vedic	Default	Default	-	-	
	Area Optimized	Explore Area	↗	↘	✘
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Logic Compaction	Explore Area	→	↗	✘
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Performance Optimized	Explore Area	↗	↘	✘
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	→	↗	✘
		Explore with Remap	↗	↘	✘
		Runtime Optimized	↘	↗	✘
Ποσοστό συνάφειας μεταβολών αλγορίθμου Vedic					**50%**
Μέσο Ποσοστό συνάφειας μεταβολών 8bit αλγορίθμων					**53%**

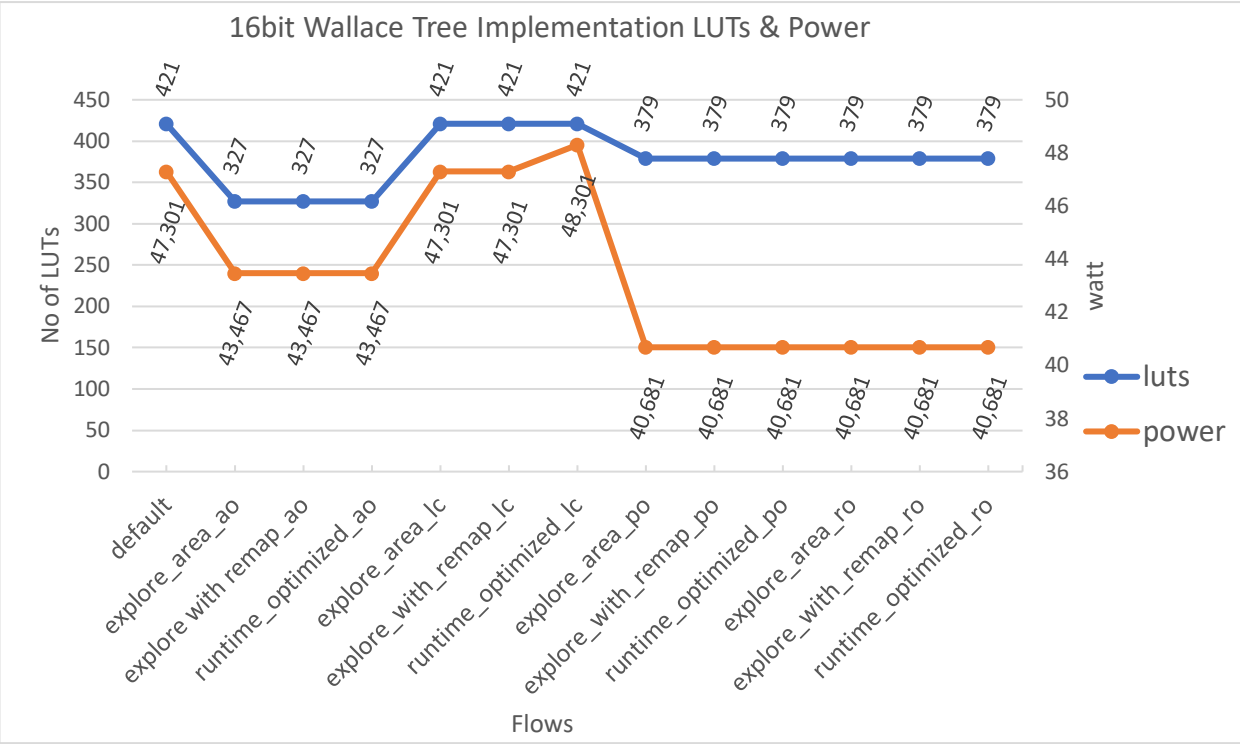
3.3.2 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 16BIT

Οι 16bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- Ο Dadda
- Ο Serial – Serial
- Ο Wallace Tree







Ο Πίνακας 3.5 δείχνει τη συνάφεια της μεταβολής Implementation LUTs και ισχύος για όλους τους συνδυασμούς στρατηγικών των τριών 16bit αλγορίθμων.

Πίνακας 3.5 Συνάφεια "Implementation" LUTs & Ισχύος για τους 16bit αλγορίθμους

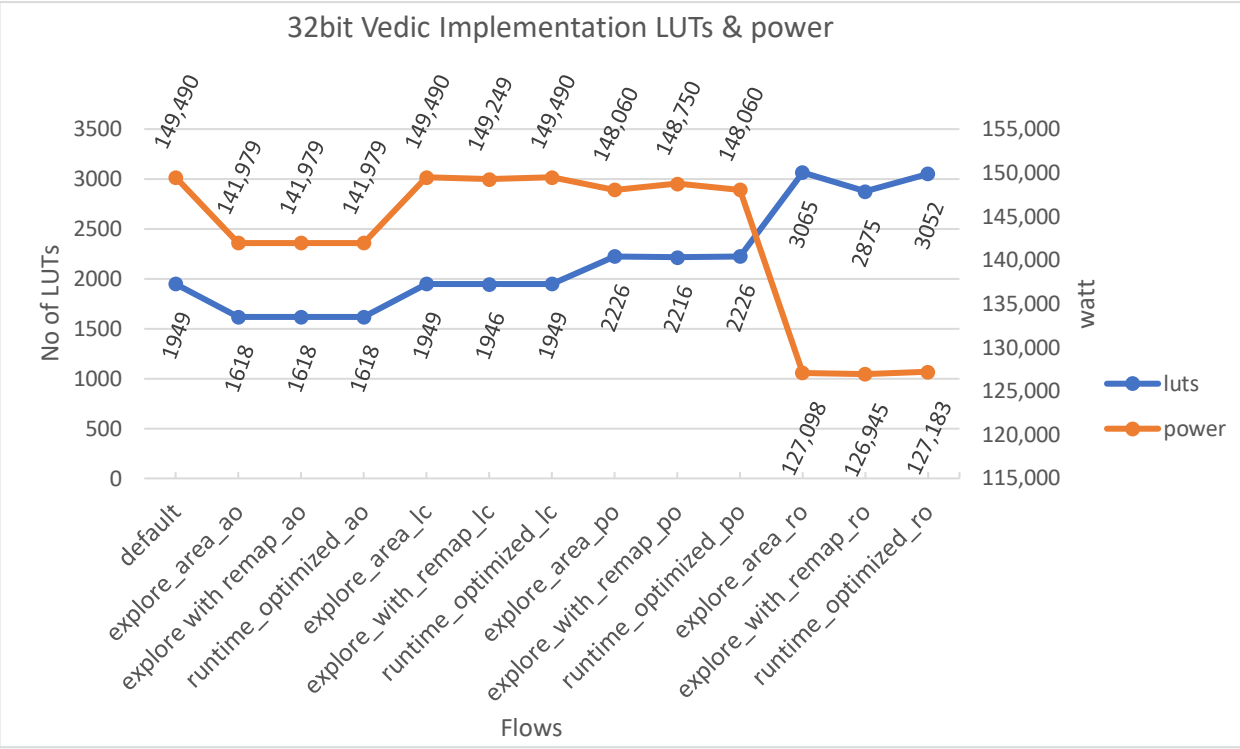
16bit Αλγόριθμος	**Στρατηγική "Synthesis**	**Στρατηγική "Implementation"**	**LUTs**	**Power**	**Ποσοστό Συνάφειας (%)**
Dadda	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	↗	✗
	Performance Optimized	Explore Area	↗	↗	✓
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	↘	✗
Ποσοστό συνάφειας μεταβολών αλγορίθμου Dadda					**83%**
Serial – Serial	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↘	✓

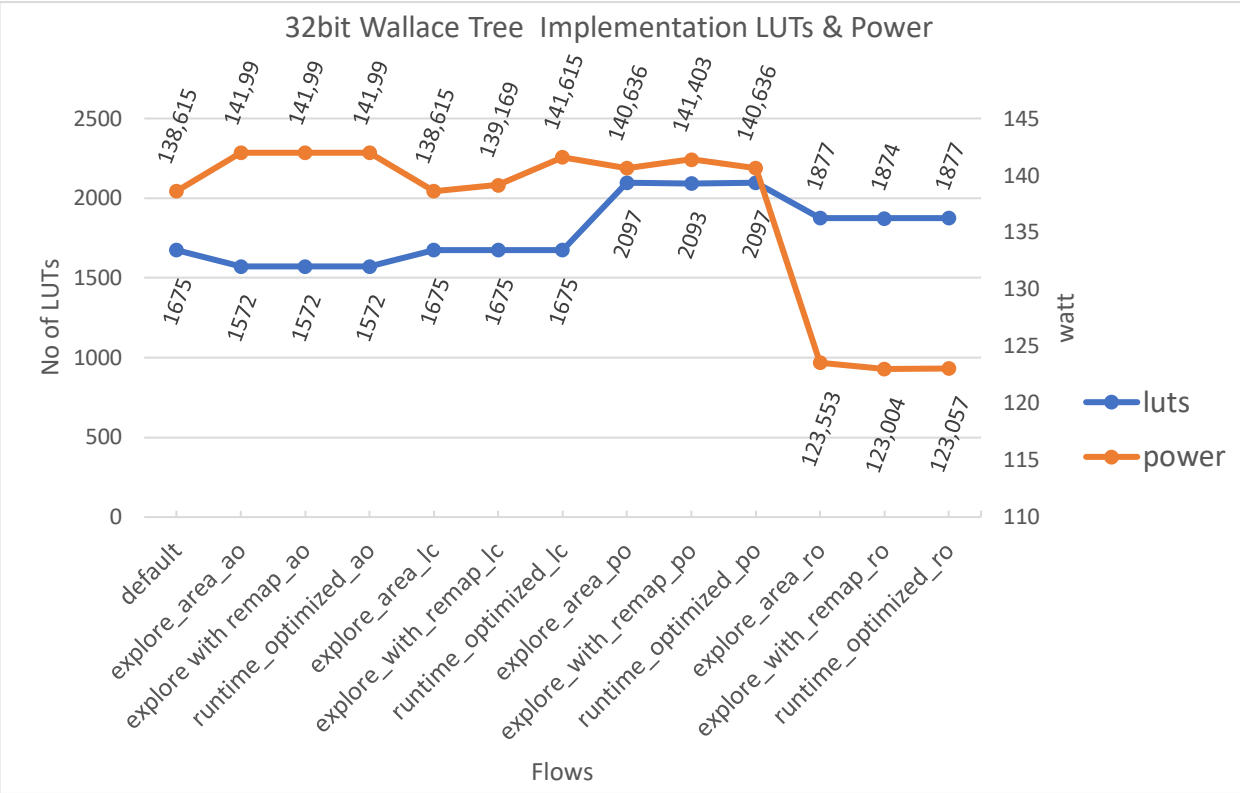
16bit Αλγόριθμος	Στρατηγική "Synthesis	Στρατηγική "Implementation"	LUTs	Power	Ποσοστό Συνάφειας (%)
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Performance Optimized	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Serial – Serial					**100%**
Wallace Tree	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	↗	✗
	Performance Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Runtime Optimized	Explore Area	→	→	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Wallace Tree					**92%**
Μέσο Ποσοστό συνάφειας μεταβολών 16bit αλγορίθμων					**92%**

3.3.3 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 32BIT

Οι 32bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- Ο Vedic
- Ο Dadda
- Ο Wallace Tree





Στον Πίνακα 3.6 φαίνεται η συνάφεια της μεταβολής Implementation LUTs και ισχύος για όλους τους συνδυασμούς στρατηγικών των τριών 32bit αλγορίθμων.

Πίνακας 3.6 Συνάφεια "Implementation LUTs & Ισχύος για τους 32bit αλγορίθμους

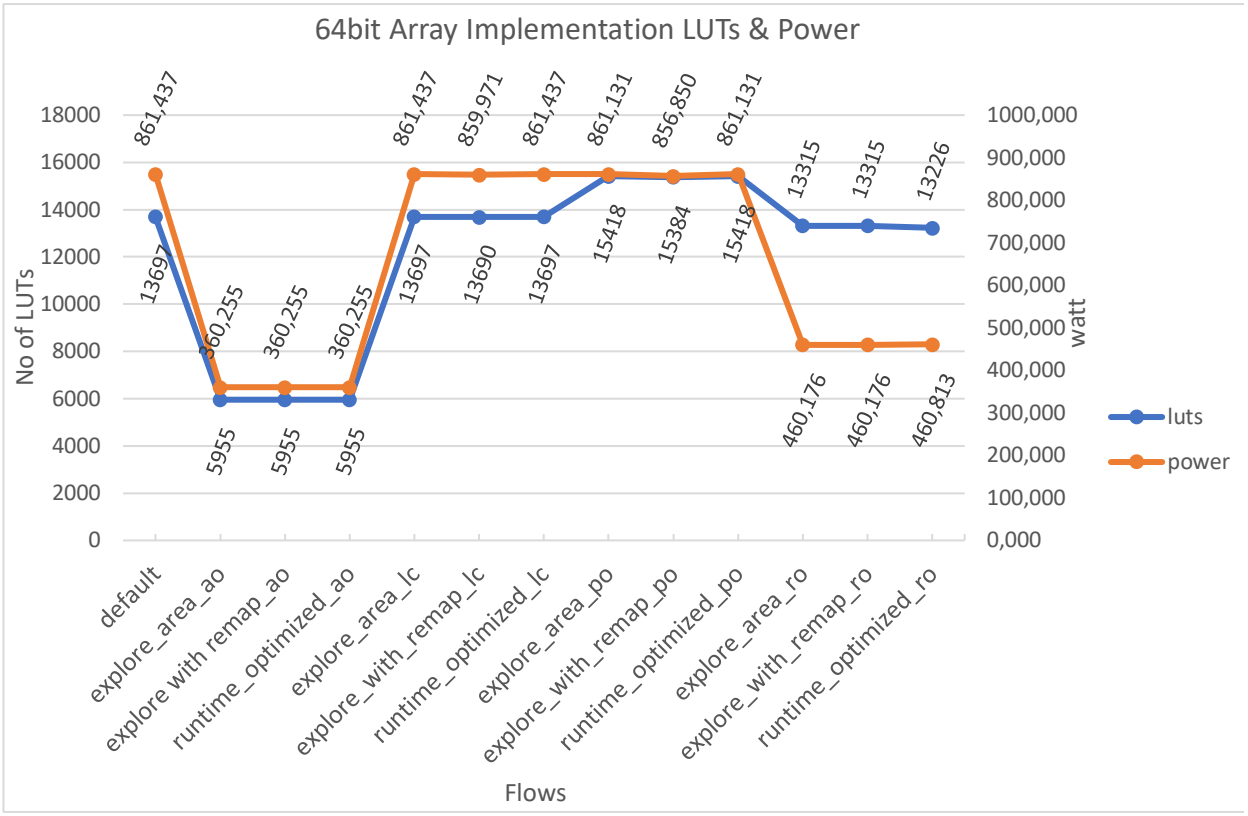
32bit Αλγόριθμος	**Στρατηγική "Synthesis**	**Στρατηγική "Implementation"**	**LUTs**	**Power**	**Ποσοστό Συνάφειας (%)**
Vedic	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Performance Optimized	Explore Area	↗	↗	✓
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	↘	↗	✗
		Runtime Optimized	↗	↗	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Vedic					**92%**
Dadda	Default	Default	-	-	-

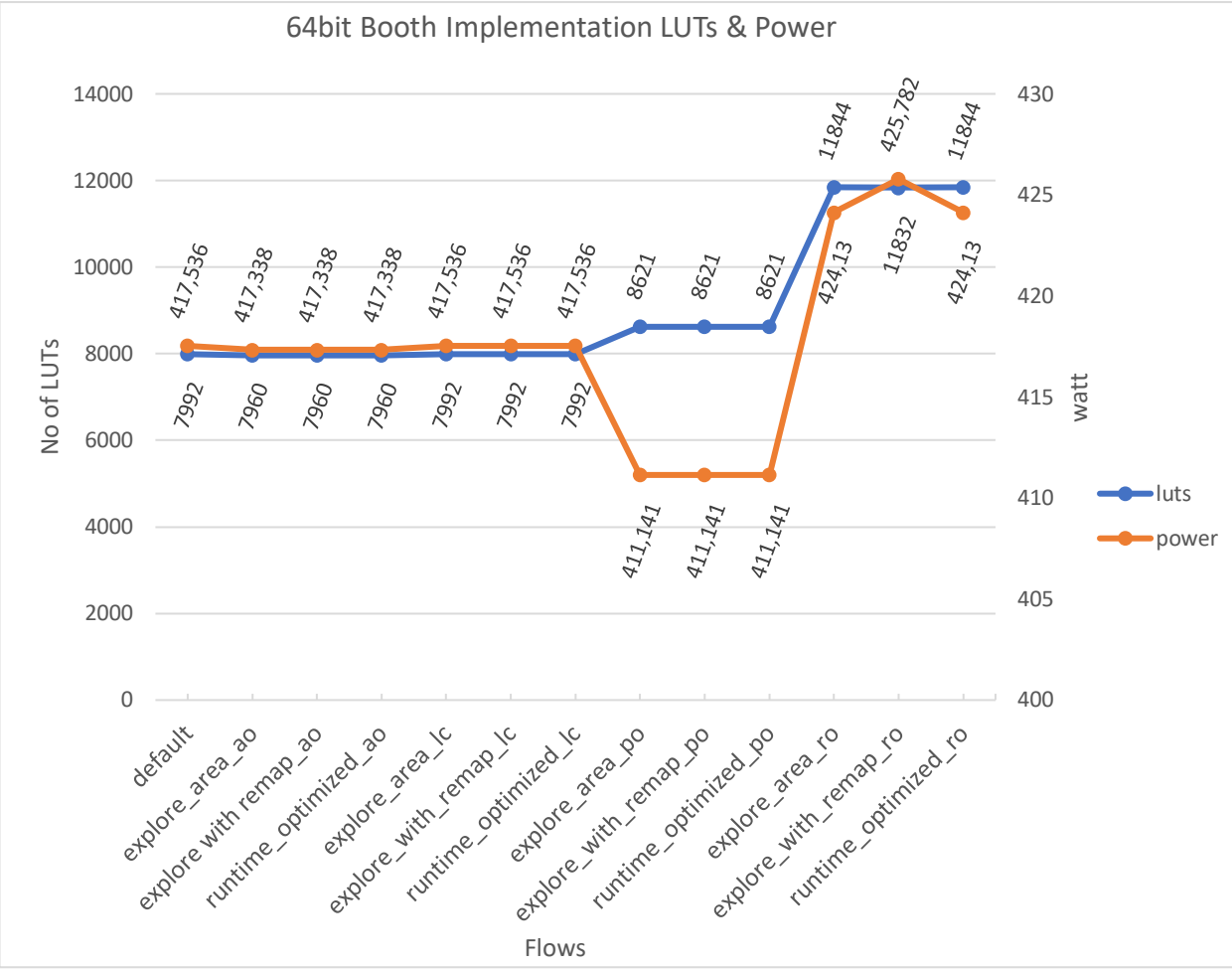
32bit Αλγόριθμος	Στρατηγική "Synthesis	Στρατηγική "Implementation"	LUTs	Power	Ποσοστό Συνάφειας (%)
	Area Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Performance Optimized	Explore Area	↗	↗	✓
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	↘	↗	✗
		Runtime Optimized	↗	↗	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Dadda					**92%**
Wallace Tree	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↗	✗
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↘	✗
		Explore with Remap	→	↗	✗
		Runtime Optimized	→	↗	✗
	Performance Optimized	Explore Area	↗	↘	✗
		Explore with Remap	↘	↗	✗
		Runtime Optimized	↗	↘	✗
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Wallace Tree					**42%**
Μέσο Ποσοστό συνάφειας μεταβολών 32bit αλγορίθμων					**75%**

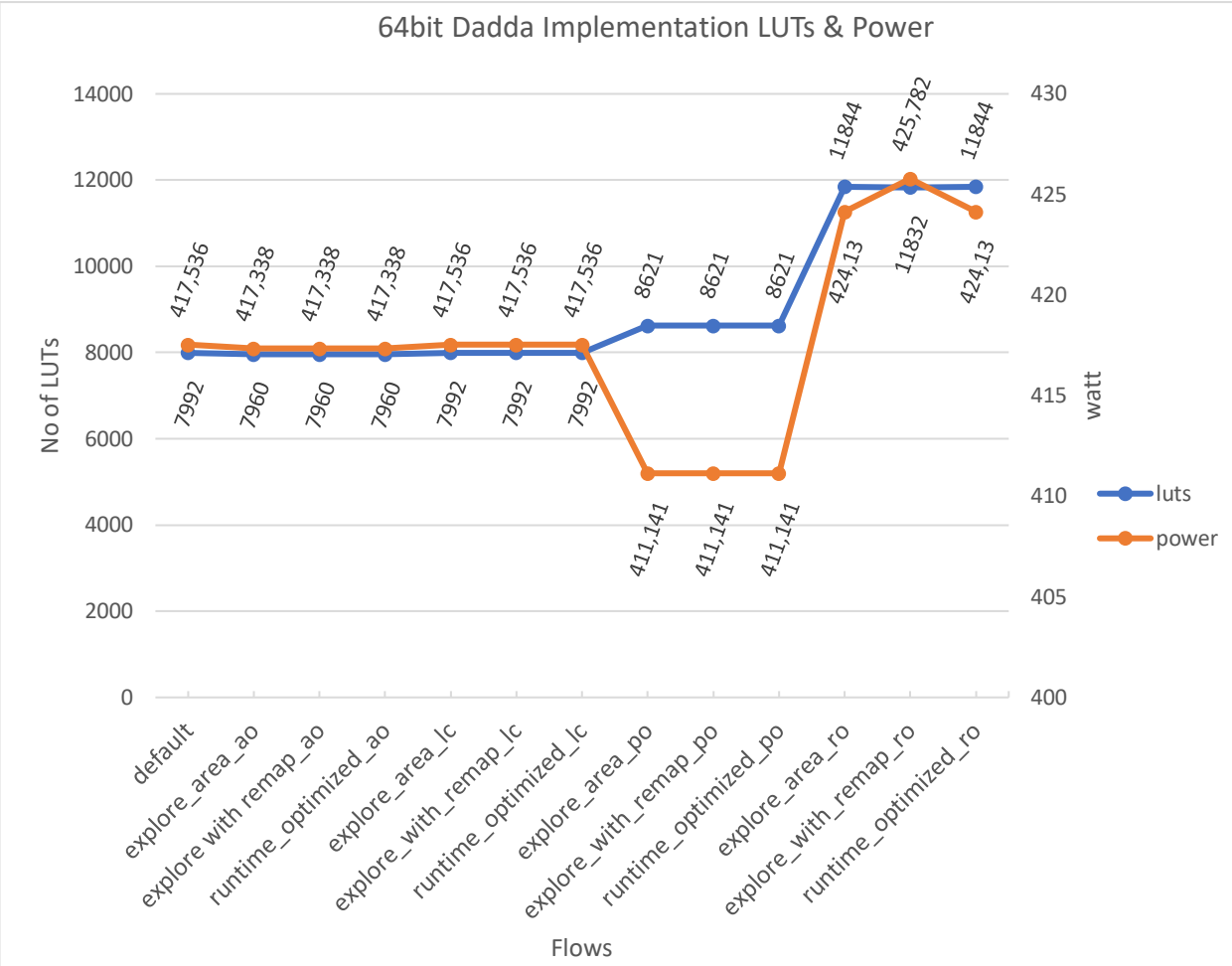
3.3.4 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 64BIT

Οι 64bit αλγόριθμοι πολλαπλασιασμού των οποίων τα αποτελέσματα παρατίθενται στη συνέχεια είναι:

- Ο Array
- Ο Booth
- Ο Dadda







Ο Πίνακας 3. 7 δείχνει τη συνάφεια της μεταβολής Implementation LUTs και ισχύος για όλους τους συνδυασμούς στρατηγικών των τριών 64bit αλγορίθμων.

Πίνακας 3. 7 Συνάφεια "Implementation LUTs & Ισχύος για τους 64bit αλγορίθμους

64bit Αλγόριθμος	**Στρατηγική "Synthesis**	**Στρατηγική "Implementation"**	**LUTs**	**Power**	**Ποσοστό Συνάφειας (%)**
Array	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Performance Optimized	Explore Area	↗	↘	✗
		Explore with Remap	↘	↘	✓

64bit Αλγόριθμος	Στρατηγική "Synthesis	Στρατηγική "Implementation"	LUTs	Power	Ποσοστό Συνάφειας (%)
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	↘	↗	✗
Ποσοστό συνάφειας μεταβολών αλγορίθμου Array					**83%**
Booth	Default	Default	-	-	-
	Area Optimized	Explore Area	↘	↘	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	↗	↗	✓
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Performance Optimized	Explore Area	↗	↘	✗
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Runtime Optimized	Explore Area	↗	↗	✓
		Explore with Remap	↘	↗	✗
		Runtime Optimized	↗	↘	✗
Ποσοστό συνάφειας μεταβολών αλγορίθμου Booth					**75%**
Dadda	Default	Default	-	-	-
	Area Optimized	Explore Area	→	↘	✗
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Logic Compaction	Explore Area	→	↗	✗
		Explore with Remap	→	→	✓
		Runtime Optimized	→	→	✓
	Performance Optimized	Explore Area	↗	↗	✓
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
	Runtime Optimized	Explore Area	↗	↘	✗
		Explore with Remap	↘	↘	✓
		Runtime Optimized	↗	↗	✓
Ποσοστό συνάφειας μεταβολών αλγορίθμου Dadda					**75%**
Μέσο Ποσοστό συνάφειας μεταβολών 64bit αλγορίθμων					**78%**

3.3.5 ΣΥΜΠΕΡΑΣΜΑΤΑ ΣΥΣΧΕΤΙΣΗΣ "IMPLEMENTATION" LUTS & ΙΣΧΥΟΣ

Παίρνοντας τον μέσο όρο των αλγορίθμων πολλαπλασιασμού 8bit, 16bit, 32bit και 64bit προκύπτει πως το μέσο ποσοστό συνάφειας "Implementation" LUTs και ισχύος για όλους τους αλγορίθμους αγγίζει το 75%. Το ποσοστό αυτό παρέχει μια ασφάλεια στο να συμπεράνουμε πως στη πλειονότητα των περιπτώσεων υπάρχει συνάφεια ανάμεσα στον αριθμό των LUTs και τη κατανάλωση ισχύος.

Για τις περιπτώσεις που δεν επιτυγχάνεται αυτή η συνάφεια μπορεί να συντρέχουν πολλοί λόγοι. Ένας από αυτούς και ίσως ο πιο σημαντικός είναι το utilization των LUTs που επιτυγχάνεται σε κάθε προσομοίωση. Δηλαδή στις περιπτώσεις που ο αριθμός των LUTs αυξάνεται ενώ η κατανάλωση ισχύος μειώνεται το utilization των LUTs αυτών ίσως είναι συγκριτικά μικρότερο σε σχέση με άλλες περιπτώσεις. Αντίθετα στις περιπτώσεις που ο αριθμός των LUTs μειώνεται ενώ η κατανάλωση ισχύος αυξάνεται το utilization των LUTs ίσως είναι μεγαλύτερο σε σχέση με άλλες περιπτώσεις.

3.4 ΣΥΣΧΕΤΙΣΗ "SYNTHESIS LUTS ΜΕ "IMPLEMENTATION" LUTS

3.4.1 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 8BIT

Οι παρακάτω πίνακες δείχνουν τον τρόπο συσχέτισης των "Synthesis" LUTs με των "Implementation" LUTs της ίδιας υλοποίησης "Synthesis" για τους 8bit αλγορίθμους πολλαπλασιασμού:

- Ο Booth
- Ο Dadda
- Ο Vedic

Πίνακας 3. 8 Συσχέτιση "Synthesis" LUTs με "Implementation" LUTs για τον 8bits αλγόριθμο πολλαπλασιασμού Booth

8bit Booth Algorithm				
Synthesis directive	**Implementation directive**	**Synthesis LUTs**	**Implementation LUTs**	**LUTs' Improvement**
Default	Default	111	111	0%
Area Optimized	Explore Area	100	129	-29%
	Explore with Remap		100	0%
	Runtime Optimized		100	0%
Logic Compaction	Explore Area	111	111	0%
	Explore with Remap		111	0%
	Runtime Optimized		111	0%
Performance Optimized	Explore Area	146	146	0%
	Explore with Remap		142	3%
	Runtime Optimized		146	0%
Runtime Optimized	Explore Area	802	547	32%
	Explore with Remap		173	78%
	Runtime Optimized		522	35%

Πίνακας 3. 9 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 8bits αλγόριθμο πολλαπλασιασμού Dadda

8bit Dadda Algorithm				
Default	Default	87	87	0%
Area Optimized	Explore Area	71	71	0%
	Explore with Remap		71	0%
	Runtime Optimized		71	0%
Logic Compaction	Explore Area	87	104	-20%
	Explore with Remap		87	0%
	Runtime Optimized		87	0%
Performance Optimized	Explore Area	110	104	5%
	Explore with Remap		108	2%
	Runtime Optimized		107	3%
Runtime Optimized	Explore Area	89	89	0%
	Explore with Remap		89	0%
	Runtime Optimized		89	0%

Πίνακας 3. 10 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 8bits αλγόριθμο πολλαπλασιασμού Vedic

8bit Vedic Algorithm				
Default	Default	124	124	0%
Area Optimized	Explore Area	125	125	0%
	Explore with Remap		122	2%
	Runtime Optimized		125	0%
Logic Compaction	Explore Area	124	125	-1%
	Explore with Remap		123	1%
	Runtime Optimized		124	0%
Performance Optimized	Explore Area	129	128	1%
	Explore with Remap		123	5%
	Runtime Optimized		129	0%
Runtime Optimized	Explore Area	140	129	8%
	Explore with Remap		135	4%
	Runtime Optimized		129	8%

3.4.2 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 16BIT

Οι παρακάτω πίνακες δείχνουν τον τρόπο συσχέτισης των "Synthesis" LUTs με των "Implementation" LUTs της ίδιας υλοποίησης "Synthesis" για τους 16bit αλγορίθμους πολλαπλασιασμού:

- Ο Dadda
- Ο Serial – Serial
- Ο Wallace Tree

Πίνακας 3. 11 Συσχέτιση "Synthesis" LUTs με "Implementation" LUTs για τον 16bits αλγόριθμο πολλαπλασιασμού Dadda

16bit Dadda Algorithm				
Default	Default	351	351	0%
Area Optimized	Explore Area	324	125	0%
	Explore with Remap		351	0%
	Runtime Optimized		324	0%
Logic Compaction	Explore Area	351	324	0%
	Explore with Remap		324	0%
	Runtime Optimized		351	0%
Performance Optimized	Explore Area	431	351	0%
	Explore with Remap		351	0%
	Runtime Optimized		430	0%
Runtime Optimized	Explore Area	391	428	1%
	Explore with Remap		430	0%
	Runtime Optimized		391	0%

Πίνακας 3. 12 Συσχέτιση "Synthesis" LUTs με "Implementation" LUTs για τον 16bits αλγόριθμο πολλαπλασιασμού Serial – Serial

16bit Serial - Serial Algorithm				
Default	Default	87	87	0%
Area Optimized	Explore Area	86	86	0%
	Explore with Remap		86	0%
	Runtime Optimized		86	0%

16bit Serial - Serial Algorithm				
Logic Compaction	Explore Area	87	87	0%
	Explore with Remap		87	0%
	Runtime Optimized		87	0%
Performance Optimized	Explore Area	122	122	0%
	Explore with Remap		122	0%
	Runtime Optimized		122	0%
Runtime Optimized	Explore Area	89	89	0%
	Explore with Remap		89	0%
	Runtime Optimized		89	0%

Πίνακας 3. 13 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 16bits αλγόριθμο πολλαπλασιασμού Wallace Tree

16bit Wallace Tree Algorithm				
Default	Default	421	421	0%
Area Optimized	Explore Area	327	327	0%
	Explore with Remap		327	0%
	Runtime Optimized		327	0%
Logic Compaction	Explore Area	421	421	0%
	Explore with Remap		421	0%
	Runtime Optimized		421	0%
Performance Optimized	Explore Area	380	379	0%
	Explore with Remap		379	0%
	Runtime Optimized		379	0%
Runtime Optimized	Explore Area	380	379	0%
	Explore with Remap		379	0%
	Runtime Optimized		379	0%

3.4.3 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 32BIT

Οι παρακάτω πίνακες δείχνουν τον τρόπο συσχέτισης των “Synthesis” LUTs με των “Implementation” LUTs της ίδιας υλοποίησης “Synthesis” για τους 32bit αλγορίθμους πολλαπλασιασμού:

- Ο Vedic

- O Dadda
- O Wallace Tree

Πίνακας 3. 14 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 32bits αλγόριθμο πολλαπλασιασμού Vedic

32bit Vedic Algorithm				
Default	Default	1961	1949	1%
Area Optimized	Explore Area	1919	1618	0%
	Explore with Remap		1618	0%
	Runtime Optimized		1618	0%
Logic Compaction	Explore Area	1961	1949	1%
	Explore with Remap		1946	1%
	Runtime Optimized		1949	1%
Performance Optimized	Explore Area	2253	2226	1%
	Explore with Remap		2216	2%
	Runtime Optimized		2226	1%
Runtime Optimized	Explore Area	3617	3065	15%
	Explore with Remap		2875	21%
	Runtime Optimized		3052	16%

Πίνακας 3. 15 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 32bits αλγόριθμο πολλαπλασιασμού Dadda

32bit Dadda Algorithm				
Default	Default	1428	1427	0%
Area Optimized	Explore Area	1366	1366	0%
	Explore with Remap		1366	0%
	Runtime Optimized		1366	0%
Logic Compaction	Explore Area	1428	1427	0%
	Explore with Remap		1427	0%
	Runtime Optimized		1427	0%
Performance Optimized	Explore Area	1811	1805	0%
	Explore with Remap		1804	0%
	Runtime Optimized		1805	0%
Runtime Optimized	Explore Area	1643	1643	0%
	Explore with Remap		1642	0%

32bit Dadda Algorithm				
	Runtime Optimized		1643	0%

Πίνακας 3. 16 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 32bits αλγόριθμο πολλαπλασιασμού Wallace Tree

32bit Wallace Tree Algorithm				
Default	Default	1678	1675	0%
Area Optimized	Explore Area	1572	1572	0%
	Explore with Remap		1572	0%
	Runtime Optimized		1572	0%
Logic Compaction	Explore Area	1678	1675	0%
	Explore with Remap		1675	0%
	Runtime Optimized		1675	0%
Performance Optimized	Explore Area	2117	2097	1%
	Explore with Remap		2093	1%
	Runtime Optimized		2097	1%
Runtime Optimized	Explore Area	1877	1877	0%
	Explore with Remap		1874	0%
	Runtime Optimized		1877	0%

3.4.4 ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ 64BIT

Οι παρακάτω πίνακες δείχνουν τον τρόπο συσχέτισης των “Synthesis” LUTs με των “Implementation” LUTs της ίδιας υλοποίησης “Synthesis” για τους 8bit αλγορίθμους πολλαπλασιασμού:

- Ο Array
- Ο Booth
- Ο Dadda

Πίνακας 3. 17 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 64bits αλγόριθμο πολλαπλασιασμού Array

64bit Array Algorithm				
Default	Default	13697	13697	0%
Area Optimized	Explore Area	5955	5955	0%
	Explore with Remap		5955	0%
	Runtime Optimized		5955	0%
Logic Compaction	Explore Area	13697	13697	0%
	Explore with Remap		13690	0%
	Runtime Optimized		13697	0%
Performance Optimized	Explore Area	16255	15418	5%
	Explore with Remap		15384	5%
	Runtime Optimized		15418	5%
Runtime Optimized	Explore Area	14017	13315	5%
	Explore with Remap		13315	5%
	Runtime Optimized		13226	6%

Πίνακας 3. 18 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 64bits αλγόριθμο πολλαπλασιασμού Booth

64bit Booth Algorithm				
Default	Default	7994	7992	0%
Area Optimized	Explore Area	7962	7960	0%
	Explore with Remap		7960	0%
	Runtime Optimized		7960	0%
Logic Compaction	Explore Area	7994	7992	0%
	Explore with Remap		7992	0%
	Runtime Optimized		7992	0%
Performance Optimized	Explore Area	9968	8621	14%
	Explore with Remap		8621	14%
	Runtime Optimized		8621	14%
Runtime Optimized	Explore Area	12193	11844	3%
	Explore with Remap		11832	3%
	Runtime Optimized		11844	3%

Πίνακας 3. 19 Συσχέτιση “Synthesis” LUTs με “Implementation” LUTs για τον 64bits αλγόριθμο πολλαπλασιασμού Dadda

64bit Dadda Algorithm				
Default	Default	6325	6321	0%
Area Optimized	Explore Area	6132	6321	0%
	Explore with Remap		6321	0%
	Runtime Optimized		6321	0%
Logic Compaction	Explore Area	6325	6321	-3%
	Explore with Remap		6321	-3%
	Runtime Optimized		6321	-3%
Performance Optimized	Explore Area	7078	7059	0%
	Explore with Remap		7046	0%
	Runtime Optimized		7059	0%
Runtime Optimized	Explore Area	11343	11158	0%
	Explore with Remap		11143	0%
	Runtime Optimized		11234	0%

3.4.5 ΣΥΜΠΕΡΑΣΜΑΤΑ ΣΥΣΧΕΤΙΣΗΣ “SYNTHESIS” ΚΑΙ “IMPLEMENTATION” LUTs

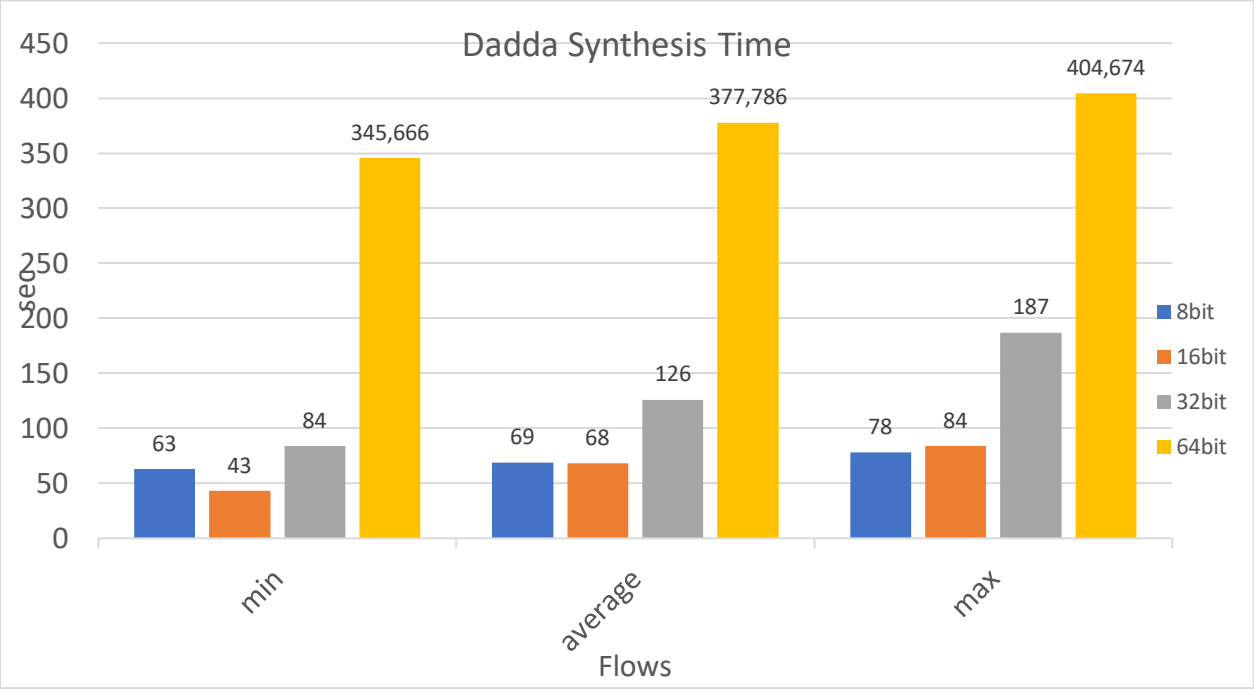
Από τους παραπάνω πίνακες φαίνεται πως στα περισσότερα “Implementations” που παράγονται από τη στρατηγική “Synthesis” “Runtime Optimized” υπάρχει βελτίωση στον αριθμό των LUTs. Στις υπόλοιπες υλοποιήσεις, εκτός μερικών εξαιρέσεων, δεν φαίνεται να υπάρχει ιδιαίτερη μεταβολή στα LUTs του “Synthesis” από αυτά του “Implementation”.

3.5 ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΟΥ DADDA ΔΙΑΦΟΡΕΤΙΚΩΝ BITS

Στη συνέχει παρουσιάζονται οι γραφικές παραστάσεις των αλγορίθμων πολλαπλασιασμού Dadda 8bit, 16bit, 32bit και 64bit. Αν και όπως φαίνεται στην αρχή του κεφαλαίου υπάρχουν και άλλοι αλγόριθμοι που εμφανίζονται σε διαφορετικά bits πολλαπλασιασμού, επιλέχθηκε να παρουσιαστεί μόνο ο αλγόριθμος Dadda καθώς αυτός υπάρχει σε όλους τους πολλαπλασιασμούς διαφορετικών bits. Τα συμπεράσματα των αλγορίθμων που δεν παρουσιάζονται είναι σε συμφωνία με τα συμπεράσματα που προκύπτουν από τη σύγκριση των διαφορετικών bits πολλαπλασιασμού του αλγορίθμου Dadda.

3.5.1 ΧΡΟΝΟΣ ΠΡΟΣΟΜΟΙΩΣΗΣ ΣΤΡΑΤΗΓΙΚΗΣ “SYNTHESIS”

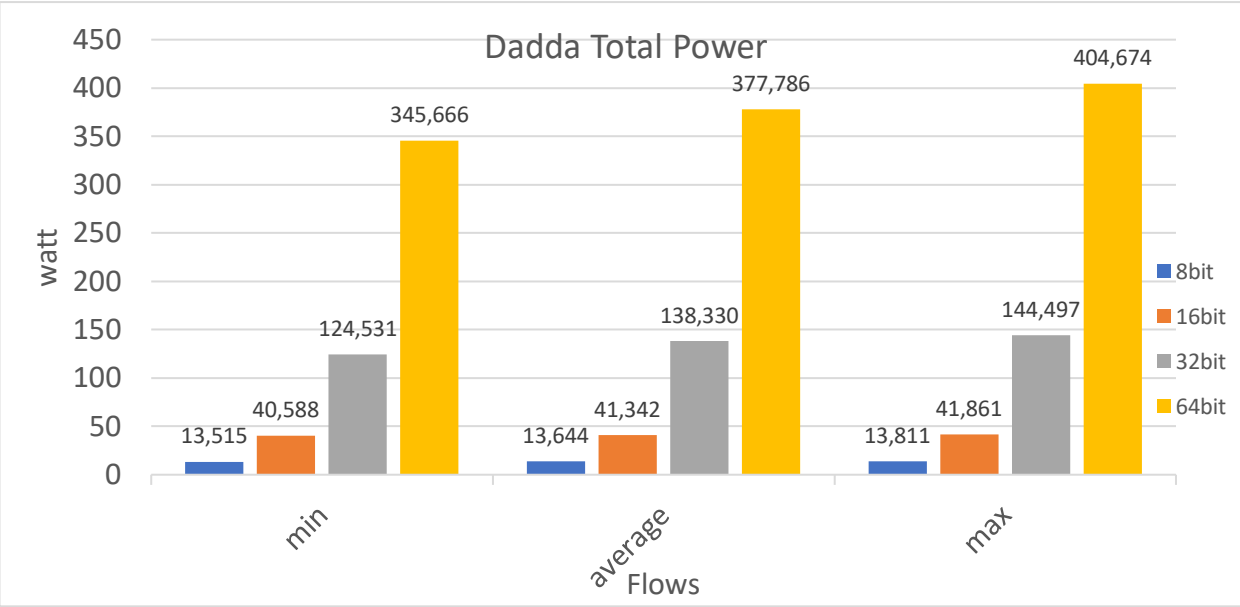
Η παρακάτω γραφική παράσταση δείχνει την ελάχιστη, τη μέγιστη και τη μέση μεταβολή του χρόνο προσομοίωσης “Synthesis” για τον αλγόριθμο Dadda σε 8bits, 16bits, 32bits και 64bits.



Από τη παραπάνω γραφική παράσταση φαίνεται πως ενώ για τους πολλαπλασιασμούς των 8bits και 16bits η μέση τιμή του χρόνου προσομοίωσης “Synthesis” δεν μεταβάλλεται σημαντικά για τους επόμενους δύο ακολουθεί εκθετική αύξηση.

3.5.2 ΚΑΤΑΝΑΛΩΣΗ ΙΣΧΥΟΣ

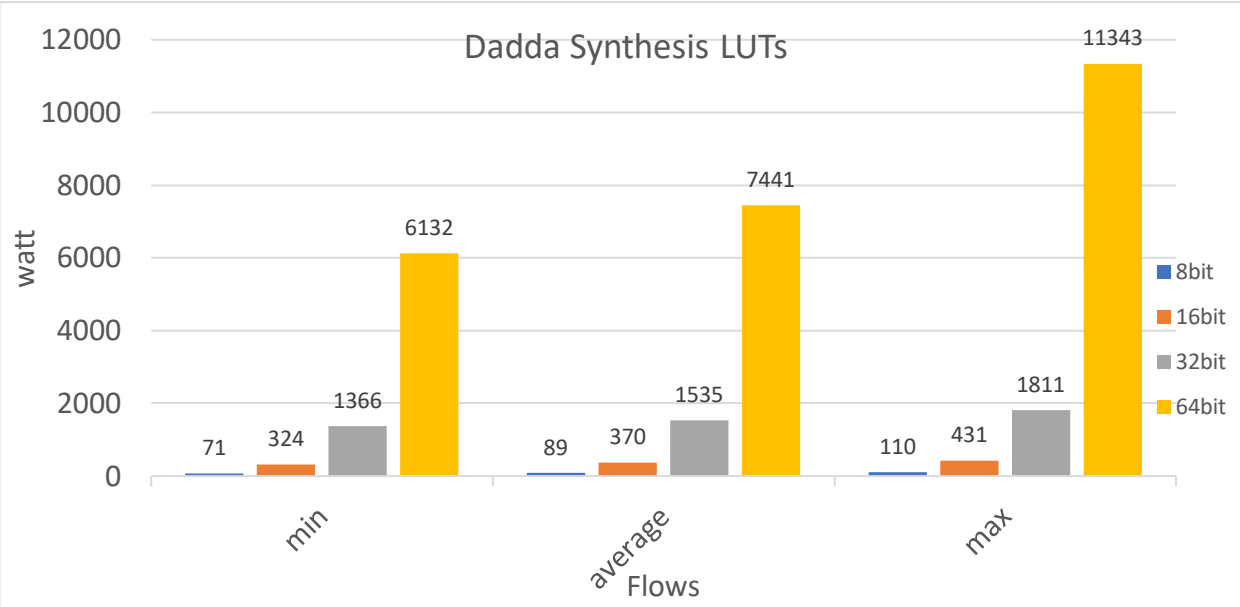
Η παρακάτω γραφική παράσταση δείχνει την ελάχιστη, τη μέγιστη και τη μέση μεταβολή της ισχύος για τον αλγόριθμο Dadda σε 8bits, 16bits, 32bits και 64bits.



Από τη παραπάνω γραφική παράσταση βλέπουμε πως με την μεταβολή των bits εισόδου από 8 σε 64 η μεταβολή της ισχύος ακολουθεί γραμμική μεταβολή καθώς η μία ισχύς από την αλλή έχει ένα σταθερό παράγοντα που κυμαίνεται γύρω από την τιμή 3.

3.5.3 ΑΡΙΘΜΟΣ “SYNTHESIS” LUTS

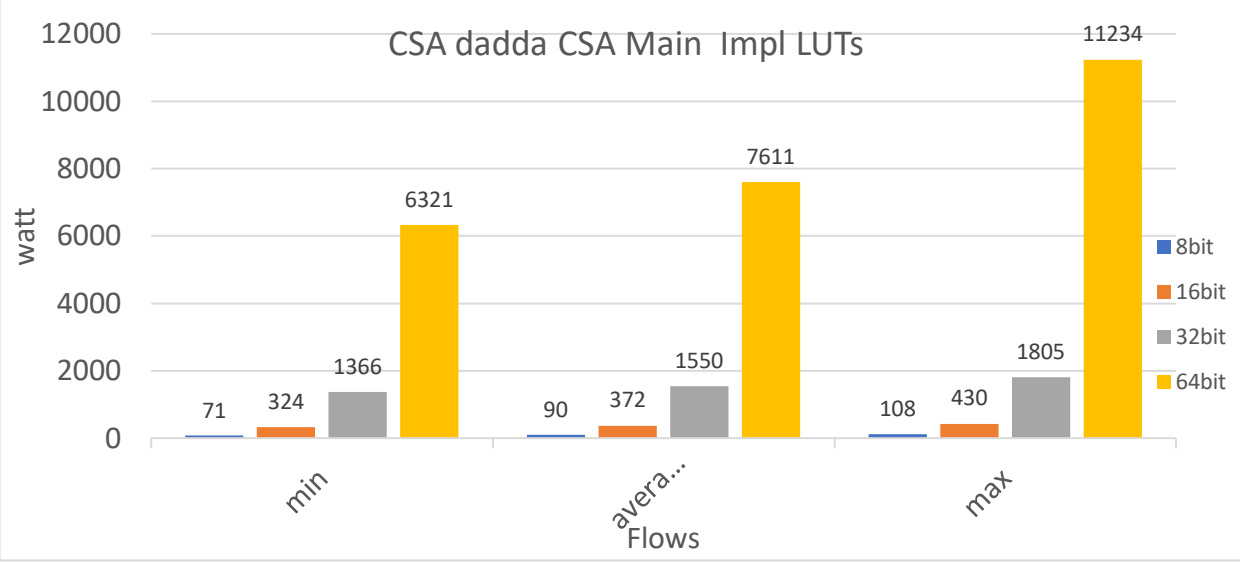
Η παρακάτω γραφική παράσταση δείχνει την ελάχιστη, τη μέγιστη και τη μέση μεταβολή των “Synthesis” LUTs για τον αλγόριθμο Dadda σε 8bits, 16bits, 32bits και 64bits.



Και εδώ παρατηρούμε πως η μεταβολή των LUTs σε σχέση με τα bits εισόδου ακολουθεί γραμμική αύξηση με ένα παράγοντα που κυμαίνεται γύρω από το 4,5.

3.5.4 ΑΡΙΘΜΟΣ “IMPLEMENTATION” LUTS

Η παρακάτω γραφική παράσταση δείχνει την ελάχιστη, τη μέγιστη και τη μέση μεταβολή των “Implementation” LUTs για τον αλγόριθμο Dadda σε 8bits, 16bits, 32bits και 64bits.



Και εδώ παρατηρούμε πως η μεταβολή των LUTs σε σχέση με τα bits εισόδου ακολουθεί γραμμική αύξηση με ένα παράγοντα που κυμαίνεται γύρω από το 4,5.

ΚΕΦΑΛΑΙΟ 4

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΟΠΤΙΚΕΣ

Σε αυτή τη διπλωματική εργασία έγινε προσομοίωση μιας μεγάλης ποικιλίας αλγορίθμων όπως αυτοί περιγράφηκαν παραπάνω για πολλαπλασιασμούς των 8bits, 16bits 32bits και 64bits. Οι προσομοιώσεις αυτές έγιναν με τη χρήση του προγράμματος Vivado Design Suit της Xilinx. Η προσομοίωση των διαδικασιών "Synthesis" και "Implementation" έγινε μέσω του evaluation board AMD Artix 7 το οποίο ενσωματώνει το FPGA XC7A200T-2FBG484C. Η εύρεση του πιο αποδοτικού τρόπου "Synthesis" και "Implementation" για τοπολογίες πολλαπλασιαστών είναι ιδιαίτερα σημαντική καθώς αυτοί εκτελούν σημαντικές διεργασίες σε διάφορους VLSI σχεδιασμού οι οποίοι πρέπει να είναι ιδιαίτερα αποδοτικοί από άποψη ταχύτητας και απόδοσης.

Οι μέθοδοι "Synthesis" και "Implementation" αναλύθηκαν ως προς τον χρόνο προσομοίωσης, τον αριθμό των LUTs και την ισχύ τους για όλους τους προς εξέτασή αλγορίθμους με τα αποτελέσματα των αναλύσεων να φαίνονται στο προηγούμενο κεφάλαιο. Επίσης έγινε σύγκριση και των αλληλεξαρτήσεων διαφορετικών μεταβλητών ώστε να εξαχθεί η λογική με την οποία η κάθε μεταβλητή επηρεάζει την άλλη.

Από τα αποτελέσματα των προσομοιώσεων της παρούσας διπλωματικής έχουν εξαιρεθεί οι χρόνοι προσομοίωσης της διαδικασίας "Implementation". Ο λόγος της μη ένταξης των προσομοιώσεων αυτών είναι η μεγάλη διαφοροποίηση των αποτελεσμάτων τους σε επαναλαμβανόμενες προσομοιώσεις της ίδιας στρατηγικής. Αν και οι προσομοιώσεις αυτές διεξήχθησαν στον ίδιο υπολογιστή δεν έγιναν όλες την ίδια χρονική περίοδο. Αυτό είχε ως αποτέλεσμα να υπάρχουν διαφορές από τα tasks του background έως και τα updates του λειτουργικού συστήματος με τα οποία διεξήχθησαν οι προσομοιώσεις. Οι διαφορές αυτές είναι ικανές να επηρεάσουν το χρόνο προσομοίωσης της διαδικασίας simulation. Παρόλα αυτά οι χρόνοι simulation της διαδικασίας "Synthesis" δε φάνηκε να μεταβάλλονται ιδιαίτερα από τις διαδοχικές εκτελέσεις και για αυτό το λόγο συμπεριελήφθησαν στη παρούσα διπλωματική εργασία.

Ως προοπτική, η παρούσα διπλωματική εργασία θα μπορούσε να έχει τη διεξαγωγή των παραπάνω προσομοιώσεων σε έναν απομακρυσμένο ηλεκτρονικό υπολογιστή ο οποίος θα έχει τον ελάχιστο αριθμό προγραμμάτων που απαιτούνται για τη διενέργεια μόνο αυτής της λειτουργίας,

δηλαδή των προσομοιώσεων. Στο συγκεκριμένο υπολογιστή καλό θα ήταν επίσης να μην εφαρμόζονται ενημερώσεις λειτουργικού συστήματος, τουλάχιστον κατά τη διενέργεια των προσομοιώσεων. Μια δεύτερη προοπτική θα ήταν οι παραπάνω προσομοιώσεις να εφαρμοστούν σε πραγματικό FPGA ώστε να γίνει η σύγκριση πραγματικών αποτελεσμάτων με αυτών της διαδικασίας προσομοίωσης.

ΒΙΒΛΙΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

[1] Booth A.D.: ‘A signed binary multiplication technique’, Q. J. Mech. Appl. Math., 1951, 4, (2), pp. 236–240

[2] Wallace C.S.: ‘A suggestion for a fast multiplier’, IEEE Trans. Electron. Comput., 1964, (1), pp. 14–17

[3] Akshata R., Prof. V.P. Gejji, Prof. B.R. Pandurangi, “ANALYSIS OF VEDIC MULTIPLIER”, INTERNATIONAL CONFERENCE ON COMPUTING, COMMUNICATION AND ENERGY SYSTEMS (ICCCES-16)

[4] S. Sabbagh and J. Baseri, "Optimization of serial-serial multiplier and implementation of a 4-bit multiplier," 2014 22nd Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, 2014, pp. 476-479, doi: 10.1109/IranianCEE.2014.6999588.

[5] A. Kumar, E. Gupta, R. P. Agarwal Shobhit and R. K. Jain, "Comparative Research for Managing Delay in Signal Processing via Multipliers," 2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, India, 2018, pp. 1213-1218, doi: 10.1109/ICPEICES.2018.8897312.

[6] Guoping Wang and J. Shield, "The efficient implementation of an array multiplier," 2005 IEEE International Conference on Electro Information Technology, Lincoln, NE, USA, 2005, pp. 5 pp.-5, doi: 10.1109/EIT.2005.1626958.

[7] P. Samundiswary, K.Anitha, “Design and Analysis of CMOS Based DADDA Multiplier”, IJCEMInternational Journal of Computational Engineering & Management, Vol. 16 Issue 6,November2013, epartmentof Electronics Engineering, Pondicherry University, Puducherry, India

[8] L. Dadda, “Some Schemes for Parallel Multipliers,” Alta Frequenza, vol. 34, pp. 349-356, 1965.

[9] Malti Bansal, Vishal Bharti, Vibhor Chanderm, “Comparison between Conventional Fast Multipliers and Improved Fast Multipliers using PTL Logic”, Malti Bansal et al 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1126 012041

[10] Cao H, Meyer-Baese U. XML-Based Automatic NIOS II Multi-Processor System Generation for Intel FPGAs. Electronics. 2022; 11(18):2840. https://doi.org/10.3390/electronics11182840