Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Σχολή Θετικών Επιστημών
Τμήμα Φυσικής
Τομέας Ηλεκτρονικής και Ηλεκτρονικών Υπολογιστών

Αποδοτικοί Αλγόριθμοι Πολλαπλασιασμού Κινητής Υποδιαστολής σε FPGA

Πτυχιακή Εργασία
του
Αλέξανδρου Συμεωνίδη

Επιβλέπων: Σιώζιος Κωνσταντίνος
Αναπληρωτής Καθηγητής Α.Π.Θ.

9 Ιουλίου 2024

Περίληψη

Η παρούσα πτυχιακή εργασία επικεντρώνεται στην υλοποίηση αλγορίθμων πολλαπλασιασμού στο πρότυπο IEEE 754 σε FPGA. Ο σκοπός της είναι να διερευνήσει την απόδοση τεσσάρων αλγορίθμων (Array, Sequential, Vedic, Wallace) σε σχέση με τον χρόνο εκτέλεσης, την κατανάλωση ενέργειας και την περιοχή υλικού, καθώς και να υλοποιήσει αυτούς τους αλγορίθμους σε FPGA και να συγκρίνει τα αποτελέσματα. Αρχικά, αναδεικνύεται η σημαντικότητα των υπολογιστικών συστημάτων, τον ρόλο των αλγορίθμων και η ανάγκη εστίασης στις βασικές μαθηματικές πράξεις και την αποδοτικότητά τους με σκοπό την χρήση τους σε απαιτητικές υπολογιστικές διεργασίες. Η υλοποίηση αυτών των αλγορίθμων περιλαμβάνει τον ορισμό και τις λειτουργίες των FPGA, τη χρήση της γλώσσας περιγραφής υλικού Verilog, καθώς και την ανάλυση του προτύπου IEEE 754. Στην συνέχεια, παρέχονται πληροφορίες σχετικά με την υλοποίηση και ανάλυση, καθώς και περιγραφές αλγορίθμων και υλοποιήσεων σε FPGA. Οι αλγόριθμοι πολλαπλασιασμού που υλοποιούνται περιλαμβάνουν τον Array, τον Sequential, τον Vedic και τον Wallace, με τον εκάστοτε να παρέχει διαφορετικά χαρακτηριστικά απόδοσης. Γίνεται σύγκριση των αλγορίθμων και επιλογή του κατάλληλου ανάλογα με τους προαναφερόμενους παράγοντες.

Abstract

This thesis focuses on the implementation of multiplication algorithms in IEEE 754 standard on FPGA. Its purpose is to investigate the performance of four algorithms (Array, Sequential, Vedic, Wallace) with respect to execution time, power consumption and hardware area, and to implement these algorithms on FPGA and compare the results. Initially, the importance of computing systems, the role of algorithms and the need to focus on basic mathematical operations and their efficiency in order to use them in demanding computing processes are highlighted. The implementation of these algorithms includes defining and explaining the main functions of FPGAs, using the Verilog hardware description language as well as analyzing the IEEE 754 standard. Implementation and analysis information is provided as well and descriptions of algorithms and implementations in FPGAs. Implemented multiplication algorithms include Array, Sequential, Vedic, and Wallace, each with different performance characteristics. Algorithms are compared and the appropriate one is chosen according to the aforementioned factors.

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου, Κωνσταντίνο Σιώζιο, για την καθοδήγηση και τεχνογνωσία του σε όλη αυτή τη διαδικασία. Εκφράζω επίσης τις ευχαριστίες μου στους φίλους και την οικογένειά μου για την αμέριστη υποστήριξη και κατανόησή τους.

Αποδοτικοί Αλγόριθμοι Πολλαπλασιασμού Κινητής Υποδιαστολής σε FPGA

Αλέξανδρος Συμεωνίδης
asymeoni@physics.auth.gr

10 Ιουλίου 2024

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Τομέας Υπολογιστικών Συστημάτων

Τα υπολογιστικά συστήματα αποτελούν τον πυλώνα του σύγχρονου ψηφιακού κόσμου, επιτρέποντας μια τεράστια γκάμα τεχνολογιών και εφαρμογών που έχουν αλλάξει τη ζωή μας. Αυτά τα συστήματα, που κυμαίνονται από ισχυρούς υπερυπολογιστές έως ενσωματωμένες συσκευές, αξιοποιούν τη δύναμη του υπολογισμού για την εκτέλεση σύνθετων εργασιών, την επεξεργασία τεράστιων ποσοτήτων δεδομένων και την προώθηση της καινοτομίας σε διάφορους τομείς. Στην καρδιά των υπολογιστικών συστημάτων βρίσκονται οι αλγόριθμοι, οι οδηγίες δηλαδή που καθοδηγούν τους υπολογιστές μέσω περίπλοκων υπολογισμών και διαδικασιών επίλυσης προβλημάτων. Αυτοί οι αλγόριθμοι, σε συνδυασμό με στοιχεία υλικού, όπως οι επεξεργαστές, η μνήμη και ο αποθηκευτικός χώρος, ενισχύουν τα υπολογιστικά συστήματα ώστε να αντιμετωπίζουν τις διάφορες προκλήσεις και να ξεκλειδώνουν νέες δυνατότητες.

Στην επίκεντρο αυτών των αλγορίθμων, βρίσκονται οι βασικές μαθηματικές πράξεις, μέσω των οποίων εξάγουμε λύσεις και συμπεράσματα στα διάφορα προβλήματα. Έτσι, είναι σημαντικό να ερευνήσουμε την αποδοτικότητα σε χρόνο, κατανάλωση ενέργειας και χώρου για την εκτέλεση αυτών των πράξεων. Στην συγκεκριμένη πτυχιακή εργασία, μελετάμε αλγορίθμους πολλαπλασιασμού στο δυαδικό σύστημα με βάση το μοντέλο του IEEE-754 και η προσομοίωση των αλγορίθμων έγινε σε ολοκληρωμένο κύκλωμα γενικής χρήσης FPGA (Field Programmable Gate Array).

1.2 Σημασία της Πράξης του Πολλαπλασιασμού στα Υπολογιστικά Συστήματα

Ο πολλαπλασιασμός είναι μια από τις θεμελιώδεις αριθμητικές πράξεις που είναι απαραίτητες για τους υπολογιστές, ώστε να εκτελούν ένα ευρύ φάσμα εργασιών. Είναι ένα βασικό δομικό στοιχείο για πολλούς αλγόριθμους και κατέχει κρίσιμο ρόλο σε διάφορες πτυχές της επιστήμης των υπολογιστών. Ακολουθούν ορισμένοι τομείς που η ακρίβεια και ο χρόνος εκτέλεσης της πράξης του πολλαπλασιασμού αποτελεί καθοριστικό ρόλο.

- **Γραφικά και Επεξεργασία εικόνας**: Ο πολλαπλασιασμός είναι απαραίτητος στα γραφικά και την επεξεργασία εικόνας για εργασίες όπως η κλιμάκωση εικόνων, η εφαρμογή φίλτρων και ο χειρισμός χρωμάτων. Χρησιμοποιείται για τη ρύθμιση της έντασης των εικονοστοιχείων (pixel[1]), την ανάμειξη διαφορετικών επιπέδων και τη δημιουργία ειδικών εφέ.
- **Μηχανική μάθηση και τεχνητή νοημοσύνη**: Ο πολλαπλασιασμός είναι μια θεμελιώδης λειτουργία στους αλγόριθμους μηχανικής μάθησης, ιδιαίτερα στα νευρωνικά δίκτυα. Χρησιμοποιείται για τον υπολογισμό βαρών, τη διάδοση σημάτων και την πραγματοποίηση προβλέψεων.
- **Κρυπτογραφία και ασφάλεια δεδομένων**: Ο πολλαπλασιασμός χρησιμοποιείται σε αλγόριθμους κρυπτογραφίας για την κρυπτογράφηση και την αποκρυπτογράφηση δεδομένων. Χρησιμοποιείται για τη δημιουργία ασφαλών κλειδιών, την εκτέλεση ασφαλούς επικοινωνίας και την προστασία ευαίσθητων πληροφοριών.

Συνολικά, ο πολλαπλασιασμός είναι μια απαραίτητη λειτουργία στην επιστήμη των υπολογιστών, που στηρίζει διάφορες υπολογιστικές διαδικασίες και επιτρέπει ένα ευρύ φάσμα εφαρμογών.

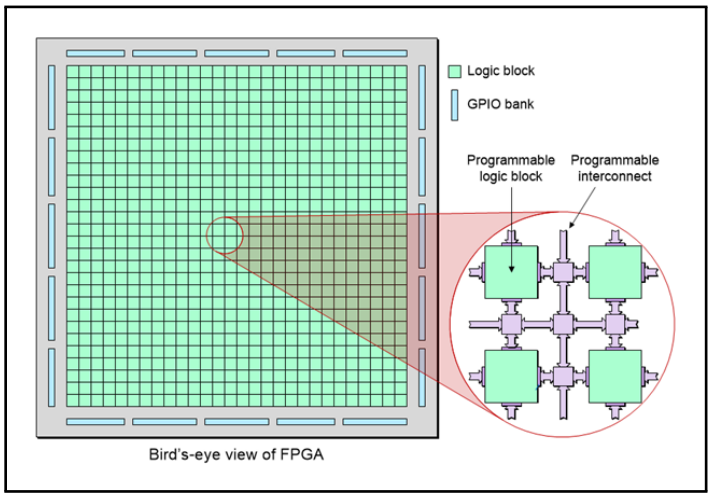
[1] Η μικρότερη διευθυνσιοδοτούμενη μονάδα σε μια συσκευή προβολής

Κεφάλαιο 2

FPGA

Το FPGA είναι ένα είδος ψηφιακού ολοκληρωμένου κυκλώματος (IC - Integrated Circuit), το οποίο παρέχει πεδίο προγραμματισμού για τον χρήστη. Κατά κύριο λόγο, το FPGA αποτελείται από μια σειρά από λογικές πύλες και διασυνδέσεις, τις οποίες ο χρήστης μπορεί να προγραμματίσει και να διαμορφώσει κατά τη διάρκεια της λειτουργίας του συστήματος.

Η ευελιξία των FPGA προέρχεται από τη δυνατότητά τους να ανακατασκευάζονται δυναμικά για να εκτελέσουν διάφορες λειτουργίες που έχουν προγραμματιστεί σε κάποια γλώσσα όπως η VHDL (VHSIC Hardware Description Language) ή η Verilog. Αυτό καθιστά τα FPGA ιδιαίτερα χρήσιμα για εφαρμογές που απαιτούν υψηλή επεξεργαστική ισχύ ή εξειδικευμένες λειτουργίες, όπως επεξεργασία εικόνας, κρυπτογραφία, και άλλες.



Σχήμα 2.1: Δομή ενός FPGA

2.1 Ιστορία των FPGA

Η εμφάνιση των FPGA στα μέσα της δεκαετίας του 1980, κυρίως με την εισαγωγή του XC2064 της Xilinx, σηματοδότησε μια μεταμορφωτική στιγμή στο τοπίο του ψηφιακού σχεδιασμού. Πριν από τα FPGA, οι μηχανικοί βασίζονταν σε ψηφιακά λογικά τσιπ

σταθερής λειτουργίας, περιορίζοντας την ευελιξία και την προσαρμοστικότητα των ηλεκτρονικών συστημάτων. Το πρωτοποριακό XC2064, παρείχε μια επαναστατική λύση — τον επαναπρογραμματισμό. Αυτή η ανακάλυψη έδωσε τη δυνατότητα στους μηχανικούς να εφαρμόσουν προσαρμοσμένα ψηφιακά λογικά κυκλώματα, προωθώντας την ταχεία δημιουργία πρωτοτύπων και τυχών αλλαγών. Κατά την δεκαετία του 1990, οι Altera, Lattice Semiconductor και άλλοι εισήλθαν στην αγορά των FPGA, συμβάλλοντας στην επέκτασή του.

Κατά τη διάρκεια των επόμενων δεκαετιών, τα FPGA συνέχισαν να εξελίσσονται με τις εξελίξεις στην τεχνολογία ημιαγωγών. Η αυξημένη πυκνότητα λογικών πυλών, οι μεγαλύτερες ταχύτητες και η βελτιωμένη απόδοση ισχύος ενίσχυσαν τις δυνατότητές τους, καθιστώντας τις απαραίτητες σε διάφορους κλάδους όπως οι τηλεπικοινωνίες και η αυτοκινητοβιομηχανία. Η δεκαετία του 2010 γνώρισε μια σημαντική εξέλιξη καθώς τα FPGA άρχισαν να ενσωματώνουν εξειδικευμένους πυρήνες επεξεργασίας, συμπεριλαμβανομένων των επεξεργαστών ARM. Αυτή η ενοποίηση άνοιξε το δρόμο για τις συσκευές System-on-Chip (SoC), διευρύνοντας το φάσμα των εφαρμογών.

Στη δεκαετία του 2020, η εξέλιξη των FPGA συνεχίζεται με συνεχείς καινοτομίες. Η επιτάχυνση της τεχνητής νοημοσύνης και η υποστήριξη για διεπαφές υψηλής ταχύτητας αποτελούν παράδειγμα των σύγχρονων εξελίξεων. Βασικοί παίκτες όπως η Xilinx και η Intel (Altera) συνεχίζουν να οδηγούν την πρόοδο στην αγορά των FPGA. Σήμερα, τα FPGA παραμένουν καθοριστικά για την ταχεία δημιουργία πρωτοτύπων, διευκολύνοντας την προσαρμογή της ψηφιακής λογικής και χρησιμεύουν ως βασικά στοιχεία για την ανάπτυξη εξειδικευμένου υλικού προσαρμοσμένου στις απαιτήσεις των σύγχρονων ηλεκτρονικών συστημάτων.

2.2 Τα βασικά στοιχεία ενός FPGA

Τα συγκεκριμένα στοιχεία ενός FPGA μπορεί να διαφέρουν ανάλογα με τον κατασκευαστή και το συγκεκριμένο μοντέλο του FPGA. Ωστόσο, μερικά βασικά στοιχεία είναι κοινά σε όλα τα FPGA και είναι τα εξής:

- **Ρυθμιζόμενα Λογικά Μπλοκ (CLB - Configurable Logic Blocks)**: Αυτά είναι τα βασικά δομικά στοιχεία ενός FPGA. Μπορούν να ρυθμιστούν ώστε να εφαρμόζουν διαφορετικούς τύπους λογικών πυλών, όπως πύλες AND, OR και XOR[1].
- **Κανάλια δρομολόγησης**: Αυτά είναι τα καλώδια που συνδέουν τα CLB μεταξύ τους. Επιτρέπουν τη σύνδεση των CLB με διαφορετικούς τρόπους για την υλοποίηση διαφορετικών κυκλωμάτων.
- **Μπλοκ εισόδου/εξόδου (I/O)**: Αυτά τα μπλοκ επιτρέπουν στο FPGA να επικοινωνεί με εξωτερικές συσκευές. Παρέχουν προσωρινή μνήμη (buffers), προγράμματα οδήγησης (drivers) και δέκτες (receivers) για τη μετατροπή σημάτων μεταξύ του FPGA και των εξωτερικών συσκευών.
- **Κύκλωμα ρολογιού (Clock circuitry)**: Το κύκλωμα ρολογιού παράγει το σήμα ρολογιού που συγχρονίζει τη λειτουργία του FPGA.
- **Κύκλωμα διαχείρισης ισχύος (Power management circuitry)**: Αυτό το κύκλωμα παρέχει την ισχύ στο FPGA και ρυθμίζει τα επίπεδα τάσης.

[1] Παράρτημα Β′ - Λογικές Πύλες και CLB

- **Μνήμη διαμόρφωσης (Configuration memory)**: Αυτή η μνήμη αποθηκεύει τα δεδομένα που ορίζουν το κύκλωμα που υλοποιείται στο FPGA.

2.3 Διαφορές μεταξύ CPU και FPGA

Τα FPGA και οι CPU είναι και τα δύο θεμελιώδη στοιχεία στα σύγχρονα υπολογιστικά συστήματα, το καθένα με ξεχωριστές δυνάμεις και εφαρμογές. Ενώ οι CPU υπερέχουν σε εργασίες υπολογιστών γενικής χρήσης, τα FPGA προσφέρουν μοναδικά πλεονεκτήματα για συγκεκριμένους τομείς, καθιστώντας τα πολύ χρήσιμα σε ορισμένα σενάρια.

Υπάρχουν διάφοροι λόγοι για τους οποίους μπορεί να επιλεγεί ένα FPGA έναντι μιας CPU για μια συγκεκριμένη εφαρμογή:

- **Υψηλή απόδοση και παραλληλισμός**: Τα FPGA μπορούν να επιτύχουν υψηλότερη απόδοση από τις CPU για συγκεκριμένες εργασίες λόγω της ικανότητάς τους να εφαρμόζουν τη λογική του συγκεκριμένου υλικού απευθείας σε ένα ολοκληρωμένο κύκλωμα. Αυτό επιτρέπει την παράλληλη επεξεργασία πολλαπλών λειτουργιών, οδηγώντας σε σημαντικές βελτιώσεις ταχύτητας.
- **Προσαρμογή και ευελιξία**: Τα FPGA μπορούν να προσαρμοστούν στις συγκεκριμένες απαιτήσεις μιας εφαρμογής. Τα λογικά μπλοκ και τα κανάλια δρομολόγησης σε ένα FPGA μπορούν να ρυθμιστούν ώστε να εφαρμόζουν συγκεκριμένες αρχιτεκτονικές υλικού, επιτρέποντας εξειδικευμένα σχέδια που είναι βελτιστοποιημένα για τη συγκεκριμένη εφαρμογή.
- **Χαμηλότερη κατανάλωση ενέργειας**: Τα FPGA μπορεί να είναι πιο αποδοτικά από τις CPU για ορισμένες εφαρμογές, ειδικά όταν αντιμετωπίζονται επαναλαμβανόμενες εργασίες ή αγωγούς επεξεργασίας δεδομένων. Η προσαρμοσμένη εφαρμογή υλικού μπορεί να βελτιστοποιηθεί για κατανάλωση ενέργειας, μειώνοντας το συνολικό ενεργειακό αποτύπωμα.
- **Χαμηλότερη καθυστέρηση**: Τα FPGA μπορούν να επιτύχουν χαμηλότερο λανθάνοντα χρόνο από τις CPU λόγω της άμεσης υλοποίησης υλικού τους. Αυτό τα καθιστά κατάλληλα για εφαρμογές όπου η απόκριση σε πραγματικό χρόνο είναι κρίσιμη, όπως η χρηματιστηριακές συναλλαγές υψηλής συχνότητας (High Frequency Trading), η επεξεργασία σήματος και τα συστήματα ελέγχου.

Οι διαφορές που εμφανίζουν τα FPGA σε σύγκριση με μια κεντρική μονάδα επεξεργασίας CPU (Central Processing Unit) εμφανίζονται στον Πίνακα 2.1.

Feature	FPGA	CPU
Flexibility	Moderate	High
Performance	High	Moderate
Latency	Low	Moderate
Cost	High	Low
Power consumption	Low	Moderate

Πίνακας 2.1: Διαφορές Μεταξύ FPGA και CPU

Κεφάλαιο 3

Γλώσσα Προγραμματισμού Verilog

Η Verilog είναι μια γλώσσα περιγραφής υλικού (HDL - Hardware Description Language) που χρησιμοποιείται για την περιγραφή ψηφιακών κυκλωμάτων και συστημάτων. Είναι μια γλώσσα υψηλού επιπέδου που επιτρέπει στους σχεδιαστές να προσδιορίζουν τη συμπεριφορά των κυκλωμάτων σε πιο αφηρημένο επίπεδο από τα σχηματικά σε επίπεδο τρανζίστορ. Η Verilog είναι μια από τις πιο δημοφιλής HDL στον κόσμο και χρησιμοποιείται για τη σχεδίαση μεγάλης ποικιλίας ψηφιακών κυκλωμάτων, από απλές λογικές πύλες έως πολύπλοκους μικροεπεξεργαστές.

3.1 Ιστορία της Verilog

Η ιστορία της Verilog ξεκινάει στις αρχές της δεκαετίας του 1980, όταν μια ομάδα μηχανικών στο Gateway Design Automation (GDA) ξεκίνησε να δημιουργήσει μια γλώσσα περιγραφής υλικού (HDL) που θα ήταν πιο διαισθητική και πιο εύκολη στη χρήση από υπάρχουσες γλώσσες όπως η VHDL. Το 1984, ο Phil Moorby και ο Prabhu Goel, δύο από τους βασικούς μηχανικούς στο GDA, ανέπτυξαν την αρχική έκδοση της Verilog, όπου σχεδιάστηκε αρχικά για την προσομοίωση ψηφιακών κυκλωμάτων, αλλά γρήγορα κέρδισε δημοτικότητα λόγω της φιλικής προς τον χρήστη σύνταξης και της ικανότητας της να περιγράφει τόσο συμπεριφορικές όσο και δομικές πτυχές των κυκλωμάτων.

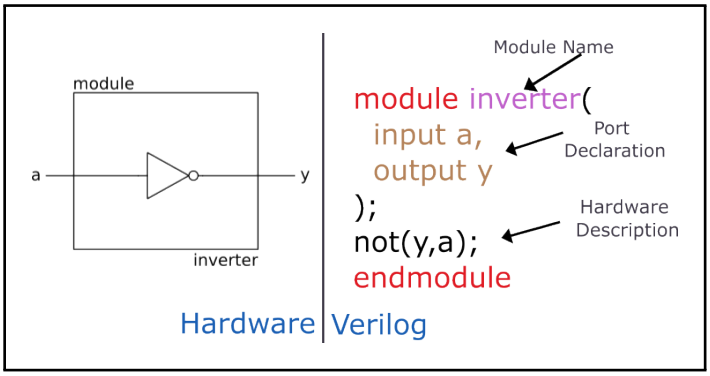
Το 1990, η Verilog έγινε ανοιχτού κώδικα, επιτρέποντας σε ανεξάρτητες εταιρείες να αναπτύξουν εργαλεία και προσομοιωτές για τη γλώσσα. Αυτή η προσέγγιση ανοιχτού κώδικα επιτάχυνε περαιτέρω την υιοθέτηση της Verilog, καθιστώντας την την κυρίαρχη HDL για τη σχεδίαση ψηφιακών κυκλωμάτων.

Το 2005 δημοσιεύτηκε το Verilog-2005 (IEEE Standard 1364-2005) με μικρές διορθώσεις και τροποποιήσεις. Επίσης το 2005 δημοσιεύτηκε η System Verilog, ένα υπερσύνολο του Verilog-2005, με πολλά νέα χαρακτηριστικά και δυνατότητες που βοηθούν στην επαλήθευση σχεδιασμού.

Το 2012, δημοσιεύθηκε το Πρότυπο IEEE 1800-2012 για τη Verilog και την SystemVerilog, επισημοποιώντας τις προδιαγραφές γλώσσας και διασφαλίζοντας τη συνέπεια μεταξύ των διαφορετικών υλοποιήσεων. Αυτή η τυποποίηση ενίσχυσε περαιτέρω την αξιοπιστία και την αποδοχή της Verilog στον κλάδο.

3.2 Χαρακτηριστικά της Verilog

Η Verilog είναι μια γλώσσα κειμένου που αποτελείται από λέξεις-κλειδιά, εκφράσεις, δηλώσεις και μπλοκ. Οι λέξεις-κλειδιά είναι δεσμευμένες λέξεις που έχουν ειδική σημασία στη γλώσσα, όπως **module**, **endmodule**, **always** και **if**. Εκφράσεις χρησιμοποιούνται για τον υπολογισμό τιμών, όπως **a** + **b** ή **c** ∗ **d**. Οι δηλώσεις χρησιμοποιούνται για την εκτέλεση ενεργειών, όπως **assign a = b + c**. Τα μπλοκ χρησιμοποιούνται για την ομαδοποίηση εντολών, όπως **always begin ... end** ή **module Half_adder ... endmodule**.



Σχήμα 3.1: Δομή της Verilog (Παράδειγμα πύλης NOT)

Η Verilog παρέχει μια ποικιλία χαρακτηριστικών για την περιγραφή ψηφιακών κυκλωμάτων, όπως:

- **Ορισμός μπλοκ (modules)**: Τα μπλοκ στην Verilog χρησιμοποιούνται για την ενθυλάκωση της συμπεριφοράς ενός κυκλώματος. Κάθε μπλοκ έχει το δικό της σύνολο εισόδων, εξόδων και εσωτερικών μεταβλητών.
- **Τύποι δεδομένων**: Η Verilog υποστηρίζει μια ποικιλία τύπων δεδομένων, συμπεριλαμβανομένων ακεραίων, δυαδικών και διανυσμάτων.
- **Κατασκευές χρονισμού**: Η Verilog υποστηρίζει κατασκευές για την περιγραφή του χρονισμού των γεγονότων σε ένα κύκλωμα, όπως always blocks και διαδικαστικές αναθέσεις.
- **Ταυτόχρονη Λειτουργία**: Η Verilog υποστηρίζει ταυτόχρονη λειτουργία, πράγμα που σημαίνει ότι πολλά μέρη ενός κυκλώματος (modules) μπορούν να λειτουργούν ταυτόχρονα. Αυτό είναι σημαντικό για την περιγραφή κυκλωμάτων που έχουν πολλαπλά σήματα ρολογιού ή που εκτελούν πολλαπλές λειτουργίες παράλληλα.

3.3 Testbench

Το testbench είναι ένα περιβάλλον προσομοίωσης που χρησιμοποιείται για την επαλήθευση της λειτουργικότητας των ψηφιακών κυκλωμάτων που περιγράφονται στην Verilog. Είναι ουσιαστικά ένα πρόγραμμα που δημιουργεί ερεθίσματα δοκιμής για το υπό δοκιμή κύκλωμα (UUT - Unit Under Test) και ελέγχει εάν οι έξοδοι του κυκλώματος ταιριάζουν με την αναμενόμενη συμπεριφορά. Τα testbench διαδραματίζουν κρίσιμο ρόλο στη διασφάλιση της ορθότητας και της αξιοπιστίας των ψηφιακών σχεδίων.

3.3.1 Βασικά Στοιχεία ενός testbench

Ένα τυπικό testbench στην Verilog αποτελείται από τα ακόλουθα στοιχεία:

- **Clock Generation**: Το testbench παράγει το σήμα ρολογιού που οδηγεί το UUT. Το σήμα ρολογιού είναι τυπικά μια περιοδική κυματομορφή, όπως ένα τετραγωνικό κύμα, που ενεργοποιεί τις λειτουργίες του κυκλώματος.
- **Δημιουργία ερεθίσματος**: Το testbench παράγει σήματα εισόδου στο UUT. Αυτά τα σήματα εισόδου αντιπροσωπεύουν τα δεδομένα ή τα σήματα ελέγχου που θα λάμβανε το κύκλωμα στο πραγματικό περιβάλλον λειτουργίας του.
- **Παρακολούθηση σήματος**: Το testbench παρακολουθεί τα σήματα εξόδου του UUT. Αυτά τα σήματα εξόδου αντιπροσωπεύουν την απόκριση του κυκλώματος στα εφαρμοζόμενα ερεθίσματα εισόδου.

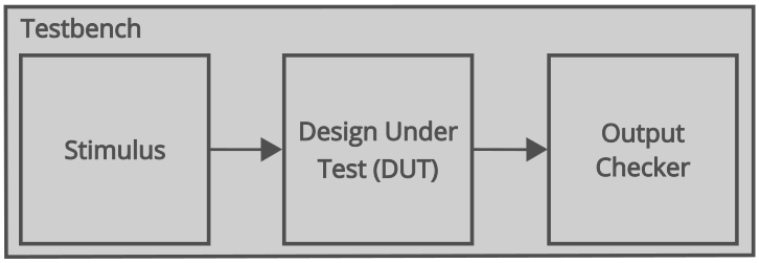
3.3.2 Τύποι testbench

Παράλληλα, έχουμε και διαφορετικούς τύπους testbench, ο καθένας προσαρμοσμένος για συγκεκριμένους σκοπούς επαλήθευσης.

Τα **λειτουργικά testbench** επικεντρώνονται στην επαλήθευση της σωστής συμπεριφοράς των λειτουργιών του κυκλώματος. Δημιουργούν ερεθίσματα εισόδου που ασκούν όλους τους πιθανούς συνδυασμούς εισόδων και ελέγχουν ότι το κύκλωμα παράγει τις αναμενόμενες εξόδους για κάθε συνδυασμό.

Τα **testbench δοκιμής χρονισμού** επικεντρώνονται στην επαλήθευση των περιορισμών χρονισμού του κυκλώματος. Παράγουν ερεθίσματα εισόδου με ακριβείς σχέσεις χρονισμού και ελέγχουν ότι οι έξοδοι του κυκλώματος συμμορφώνονται με τις καθορισμένες απαιτήσεις χρονισμού.

Τέλος, υπάρχουν και τα **testbench τυχαίων δοκιμών**, που δημιουργούν τυχαία ερεθίσματα εισόδου για να ελέγξουν την ¨αντοχή¨ του κυκλώματος έναντι απροσδόκητων ή μη έγκυρων εισόδων. Είναι χρήσιμα για την ανακάλυψη προβλημάτων που μπορεί να εμφανίζονται σε ακραίες τιμές εισόδων ή εξόδων.



Σχήμα 3.2: Τυπική αρχιτεκτονική ενός απλού testbench

Συμπεραίνοντας, το testbench αποτελεί ένα πολύ χρήσιμο εργαλείο στην ψηφιακή σχεδίαση κυκλωμάτων, διότι βοηθάει στον εντοπισμό σφαλμάτων στη σχεδίαση του κυκλώματος ελέγχοντας για αποκλίσεις μεταξύ των εξόδων του κυκλώματος και της αναμενόμενης συμπεριφοράς, παρέχει αποδείξεις ότι το κύκλωμα πληροί τις προδιαγραφές λειτουργίας/χρονισμού και μπορεί να βοηθήσει σε μεγάλο βαθμό τον σχεδιαστή του κυκλώματος ότι τυχών αλλαγές στον κώδικα δεν εισάγουν νέα σφάλματα.

Κεφάλαιο 4

Παράσταση Αριθμών Κινητής Υποδιαστολής - IEEE-754

Το πρότυπο IEEE-754, με επίσημο τίτλο "IEEE Standard for Floating-Point Arithmetic", είναι ένα τεχνικό πρότυπο για την αριθμητική κινητής υποδιαστολής που καθιερώθηκε το 1985 από το Institute of Electrical and Electronics Engineers (IEEE). Ορίζει ένα σύνολο μορφών και αλγορίθμων για την αναπαράσταση κατά προσέγγιση πραγματικών αριθμών σε συστήματα υπολογιστών. Το πρότυπο IEEE 754 έχει γίνει το κυρίαρχο πρότυπο για την αριθμητική κινητής υποδιαστολής στους περισσότερους σύγχρονους υπολογιστές και χρησιμοποιείται ευρέως στους επιστημονικούς υπολογιστές, τη μηχανική και τις οικονομικές εφαρμογές.

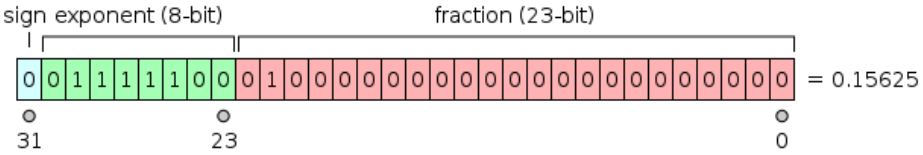
Το πρότυπο αντιμετώπισε πολλά προβλήματα που εντοπίζονταν στις διάφορες υλοποιήσεις κινητής υποδιαστολής που έκαναν δύσκολη την αξιόπιστη χρήση τους. Ορίζει μορφές για την αναπαράσταση αριθμών κινητής υποδιαστολής, συμπεριλαμβανομένων των θετικών και αρνητικών αριθμών, μηδέν, άπειρο και NaN (Not a Number). Το πρότυπο καθορίζει επίσης πράξεις όπως πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση και τετραγωνική ρίζα.

Το πρότυπο ορίζει πολλές μορφές κινητής υποδιαστολής, συμπεριλαμβανομένων της μισής ακρίβειας (16-bit - Half Precision), της απλής ακρίβειας (32-bit - Single Precision) και της διπλής ακρίβειας (64-bit - Double Precision). Κάθε μορφή καθορίζει το εύρος των αναπαραστάσιμων αριθμών, την ακρίβεια της αναπαράστασης και το μέγεθος του ψηφίου προσήμου, του εκθέτη και του κλασματικού μέρους.

Η αναπαράσταση ενός αριθμού με βάση το πρότυπο IEEE 754 ¨χωρίζει¨ την ψηφιολέξη σε τρία βασικά τμήματα.

- **Ψηφίο Προσήμου**: Το προσημασμένο μέγεθος υποδεικνύει το πρόσημο του αριθμού, είτε είναι θετικό είτε αρνητικό. Ένα bit τιμής 0 αντιπροσωπεύει έναν θετικό αριθμό, ενώ ένα bit τιμής 1 αντιπροσωπεύει έναν αρνητικό αριθμό.
- **Εκθέτης**: Ο εκθέτης αντιπροσωπεύει τη δύναμη της βάσης (συνήθως 2) με την οποία πολλαπλασιάζεται το κλασματικό μέρος. Καθορίζει το συνολικό μέγεθος του αριθμού κινητής υποδιαστολής.
- **Κλασματικό Μέρος**: Το κλασματικό μέρος, γνωστό και ως mantissa (ή fraction), είναι ένα δυαδικό κλάσμα που παρέχει την ακρίβεια της προσέγγισης. Το αρχικό

ψηφίο του κλασματικού μέρους είναι πάντα 1, το οποίο δεν αποθηκεύεται ρητά αλλά θεωρείται. Αυτό είναι γνωστό ως το κρυφό κομμάτι.



Σχήμα 4.1: Αναπαράσταση του αριθμού 0.15625 με βάση το πρότυπο IEEE 754

Ο συνδυασμός του ψηφίου προσήμου, του εκθέτη και του κλασματικού μέρους επιτρέπει την αποτελεσματική αναπαράσταση ενός ευρέος φάσματος πραγματικών αριθμών εντός ενός περιορισμένου αριθμού bit.

Στον παρακάτω πίνακα εμφανίζονται οι μορφές κινητής υποδιαστολής που θα μελετήσουμε, μαζί με τα βασικά χαρακτηριστηκά του καθενός.

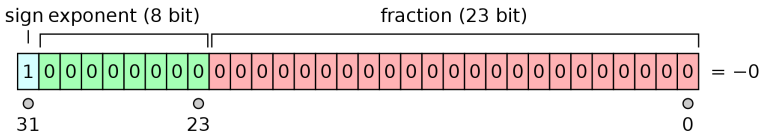
		Significand	**Exponent**		**Properties**	
Name	**Common name**	**Digits**	**Min**	**Max**	**Maximum Value**	**Minimum Value>0**
binary16	Half precision	11	−14	15	65504	$6.10 \cdot 10^{-5}$
binary32	Single precision	24	−126	+127	$3.40 \cdot 10^{38}$	$1.18 \cdot 10^{-38}$
binary64	Double precision	53	−1022	+1023	$1.80 \cdot 10^{308}$	$2.23 \cdot 10^{-308}$

Πίνακας 4.1: Διάφορες μορφές του προτύπου IEEE 754 και αντίστοιχες ιδιότητες

4.1 Ειδικές Τιμές

Το πρότυπο IEEE 754 ορίζει πέντε κατηγορίες αριθμών κινητής υποδιαστολής, συμπεριλαμβανομένων πολλών ειδικών τιμών που χρησιμοποιούνται για την αναπαράσταση εξαιρετικών περιπτώσεων και απροσδιόριστων αποτελεσμάτων σε υπολογισμούς κινητής υποδιαστολής. Αυτές οι ειδικές τιμές παίζουν καθοριστικό ρόλο στη διασφάλιση της ορθότητας και της συνέπειας των αριθμητικών υπολογισμών.

- **Μηδέν**: Με βάση το IEEE 754 υπάρχουν δύο είδη του μηδενός, ένα ¨θετικό¨ και ένα ¨αρνητικό¨. Πιο συγκεκριμένα, τα μηδέν ορίζονται όταν όλα τα bit του κλασματικό μέρος και του εκθέτη είναι ίσα με 0. Έτσι, ανάλογα με την τιμή του bit του προσήμου, έχουμε το ¨αρνητικό¨ και ¨θετικό¨ μηδέν.

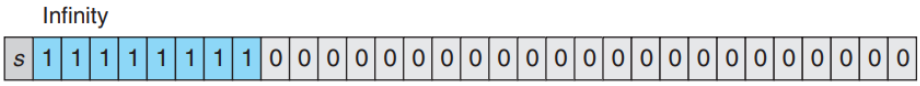


Σχήμα 4.2: Αναπαράσταση του αριθμού −0 με βάση το πρότυπο IEEE 754

- **Υποκανονικοί αριθμοί**: Οι υποκανονικοί αριθμοί αντιπροσωπεύουν αριθμούς που είναι πολύ μικροί για να αναπαρασταθούν στο κανονικό εύρος της μορφής κινητής υποδιαστολής. Στον εκθέτη, όλα τα ψηφία του είναι ίση με 0, ενώ δεν επηρεάζεται από τις τιμές που έχουμε στο κλασματικό μέρος και στο ψηφίο προσήμου. Οι υποκανονικοί αριθμοί χρησιμοποιούνται για την αποφυγή υπορροής (Underflow), η οποία συμβαίνει όταν ένα αποτέλεσμα γίνεται πολύ μικρό για να αναπαρασταθεί στο κανονικό εύρος.

Σχήμα 4.3: Αναπαράσταση ενός υποκανονικού αριθμού με βάση το πρότυπο IEEE 754

- **Άπειρο**: Τα άπειρα αντιπροσωπεύουν τα θετικά και αρνητικά όρια των αναπαραστάσιμων αριθμών κινητής υποδιαστολής. Το θετικό άπειρο ($+\infty$) χρησιμοποιείται για να αναπαραστήσει τιμές που είναι πολύ μεγάλες για να αναπαρασταθούν στο κανονικό εύρος, ενώ το αρνητικό άπειρο ($-\infty$) χρησιμοποιείται για να αναπαραστήσει τιμές που είναι πολύ μικρές. Τα άπειρα έχουν στον εκθέτη όλα τα bits του την τιμή 1, ενώ το κλασματικό μέρος έχει όλα του τα bits την τιμή 0 (το ψηφίο προσήμου ορίζει το $+\infty$ και το $-\infty$). Τα άπειρα είναι σημαντικά για τον χειρισμό της υπερχείλισης και για τη διασφάλιση ότι οι υπολογισμοί δεν παράγουν απροσδιόριστα αποτελέσματα.



Σχήμα 4.4: Αναπαράσταση του απείρου με βάση το πρότυπο IEEE 754

- **NaN - Not a Number**: Το NaN αντιπροσωπεύει ένα μη έγκυρο ή απροσδιόριστο αποτέλεσμα σε υπολογισμούς κινητής υποδιαστολής. Τα NaN χρησιμοποιούνται για να υποδείξουν ότι ένα αποτέλεσμα δεν έχει νόημα ή δεν μπορεί να αναπαρασταθεί, όπως όταν μια πράξη επιχειρεί να διαιρέσει με το μηδέν. Τα NaN έχουν στον εκθέτη όλα τα bits του την τιμή 1, ενώ το κλασματικό μέρος μπορεί να έχει όλες τις πιθανές τιμές πέρα από την περίπτωση που είναι όλα τα ψηφία ίσα με 0 (ανάγεται στην περίπτωση του απείρου). Τα NaN είναι σημαντικά για τον χειρισμό ιδιαίτερων περιπτώσεων και την αποτροπή των υπολογισμών από το να παράγουν λανθασμένα αποτελέσματα.

Σχήμα 4.5: Αναπαράσταση του NaN με βάση το πρότυπο IEEE 754

Το παρακάτω σχήμα συνοψίζει τις πιθανές προκύπτουσες τιμές με βάση τους πολλαπλασιασμούς που περιλαμβάνουν τους διάφορους συνδυασμούς αριθμών κινητής υποδιαστολής, συμπεριλαμβανομένων των ειδικών τιμών.

OP2 \ OP1	NaN	+Infinity	-Infinity	+0	-0	+Num	-Num
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
+Infinity	NaN	+Infinity	-Infinity	NaN	NaN	+Infinity	-Infinity
-Infinity	NaN	-Infinity	+Infinity	NaN	NaN	-Infinity	+Infinity
+0	NaN	NaN	NaN	+0	-0	+0	-0
-0	NaN	NaN	NaN	-0	+0	-0	+0
+Num	NaN	+Infinity	-Infinity	+0	-0	+Num +Infinity	-Num -Infinity
-Num	NaN	+Infinity	+Infinity	-0	+0	-Num -Infinity	+Num +Infinity

Σχήμα 4.6: Αποτελέσματα πολλαπλασιασμού των διαφόρων συνδυασμών αριθμών κινητής υποδιαστολής

4.2 Special Values Classifier

Με σκοπό να λάβουμε το σωστό αποτέλεσμα από τον πολλαπλασιαστή, είναι κρίσιμο πριν ξεκινήσουμε την διαδικασία του πολλαπλασιασμού, να γνωρίζουμε αν οι αριθμοί που πολλαπλασιάζονται βρίσκονται σε κάποια από τις ειδικές τιμές που έχει ορίσει το πρότυπο IEEE 754 (Μηδέν, Υποκανονικοί αριθμοί, Άπειρο, NaN). Γι' αυτό, κατασκευάζουμε ένα module το οποίο θα μπορεί να ξεχωρίζει μεταξύ των διαφόρων ειδικών τιμών και να πράττει ανάλογα σε κάθε περίπτωση.

- **Μηδέν**: Το πρότυπο IEEE 754 ορίζει το μηδέν ως την περίπτωση που όλα τα bit του κλασματικού μέρους και του εκθέτη είναι ίσα με μηδέν. Για να εντοπίσουμε άμα έχουμε όλα τα bit ίσα με 0, θα χρησιμοποιήσουμε την πύλη OR σε bit-wise operation και το τελικό αποτέλεσμα θα το αντιστρέψουμε μέσω μιας πύλης NOT. Το αποτέλεσμα το αποθηκεύουμε στις μεταβλητές expZeroes και sigZeroes, όπου όταν και οι δύο μεταβλητές είναι ίσες με 1 τότε έχουμε μηδέν (θετικό ή αρνητικό) με βάση το πρότυπο.
- **Υποκανονικοί Αριθμοί**: Σε αυτήν την περίπτωση, θέλουμε και πάλι όλα τα bit του εκθέτη να είναι ίσα με 0, αλλά το κλασματικό μέρος μπορεί να πάρει οποιαδήποτε τιμή πέρα από το να είναι όλα 0. Έτσι, πάλι με τις ίδιες μεταβλητές, θέλουμε το expZeroes να είναι ίσο με 1 και το sigZeroes να είναι ίσο με 0.
- **Άπειρο**: Για το άπειρο, θα χρειαστούμε μια νέα μεταβλητή, την expOnes, που θα γίνεται ίση με 1 όταν όλα τα ψηφία του εκθέτη είναι ίσα με 1. Για να εντοπίσουμε αυτήν την περίπτωση, θα χρησιμοποιήσουμε την πύλη AND σε bit-wise operation στα bit του εκθέτη. Έτσι, για το άπειρο θέλουμε το expOnes να είναι 1 και το sigZeroes να είναι 1.

- **NaN**: Για τους απροσδιόριστους αριθμούς, όλα τα ψηφία του εκθέτη είναι ίσα με 1, ενώ το κλασματικό μέρος μπορεί να λάβει όλες τις τιμές εκτός από την περίπτωση να είναι όλα 0. Δηλαδή, θέλουμε το expOnes να είναι ίσο με 1 και το sigZeroes να είναι 0.

Ο κώδικας που χρησιμοποιήσαμε για το Half Precision Format είναι ο εξής:

```
//hp_class.v
module hp_class(f, nan, inf, zero, subnormal, normal);

input [15:0] f;
output nan, inf, zero, subnormal, normal;

wire expOnes, expZeroes, sigZeroes;

assign expOnes = &f[14:10];
assign expZeroes = ~|f[14:10];
assign sigZeroes = ~|f[9:0];

assign nan = expOnes & ~sigZeroes;
assign inf = expOnes & sigZeroes;
assign zero = expZeroes & sigZeroes;
assign subnormal = expZeroes & ~sigZeroes;
assign normal = ~expOnes & ~expZeroes;

endmodule
```

Μέσω της προσομοίωσης βρίσκουμε ότι ο αριθμός των ειδικών τιμών είναι ίδιος με αυτόν που περιμέναμε:

Infinities	2
NaNs	2046
Zeroes	2
Subnormals	2046
Normals	61440

Πίνακας 4.2: Αριθμός ειδικών τιμών στην μορφή Half Precision

Για τις υπόλοιπες μορφές του προτύπου (Single Precision, Double Precision) έχουν κατασκευαστεί ανάλογοι classifiers, αλλά δεν μπορούμε να εξετάσουμε την ορθότητα τους λόγω του πολύ μεγάλου αριθμού περιπτώσεων που θα χρειαστεί να προσωμοιωθούν σε προσωπικό υπολογιστή (2^{32} για το Single Precision και 2^{64} για το Double Precision).

4.3 Πολλαπλασιασμός στο πρότυπο IEEE 754

Ο πολλαπλασιασμός κινητής υποδιαστολής είναι μία από τις θεμελιώδεις πράξεις στην αριθμητική κινητής υποδιαστολής. Περιλαμβάνει τον πολλαπλασιασμό δύο αριθμών κινητής υποδιαστολής, που αντιπροσωπεύονται σε δυαδική μορφή, για να παραχθεί ένα κατά προσέγγιση γινόμενο. Ο αλγόριθμος πολλαπλασιασμού στο IEEE 754 έχει σχεδιαστεί για να διατηρεί την ακρίβεια και να χειρίζεται τις ειδικές τιμές.

4.3.1 Βασικά Βήματα του Αλγορίθμου του Πολλαπλασιασμού

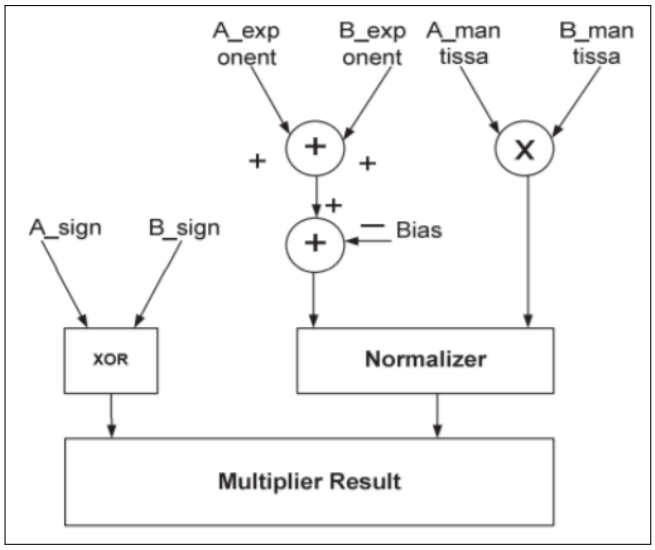
1. **Διαχειρισμός Προσήμου**: Τα ψηφία προσήμου των δύο αριθμών περνάνε σαν εισόδους σε μία πύλη XOR και η έξοδος είναι το πρόσημου του αποτελέσματος του πολλαπλασιασμού. Εάν και τα δύο ψηφία έχουν το ίδιο πρόσημο, το αποτέλεσμα είναι θετικό. εάν έχουν διαφορετικό πρόσημο, το αποτέλεσμα είναι αρνητικό.

2. **Υπολογισμός Εκθέτη**: Ο εκθέτης υπολογίζεται ως το άθροισμα των εκθετών που πολλαπλασιάζονται, λαμβάνοντας υπόψη το κρυφό bit του κλασματικού μέρους. Το κρυφό bit θεωρείται ότι είναι 1 και στις δύο περιπτώσεις. Έτσι, το άθροισμα των εκθετών προσαρμόζεται αφαιρώντας το αποτέλεσμα με έναν αριθμό ονόματι bias και υπολογίζεται σε κάθε μορφή του προτύπου με βάση την σχέση bias = $2^{(exp-1)} - 1$.

3. **Πολλαπλασιασμός Κλασματικού Μέρους**: Τα κλασματικά μέροι των δύο αριθμών πολλαπλασιάζονται χρησιμοποιώντας έναν αλγόριθμο πολλαπλασιασμού σε επίπεδο bit (bit-wise operation), παρόμοιο με τον πολλαπλασιασμό ακεραίων χωρίς πρόσημο. Το προϊόν των κλασματικών μερών κανονικοποιείται μετατοπίζοντας τα ψηφία αριστερά ή δεξιά (bit shift operation) έως ότου το πιο σημαντικό ψηφίο (MSB - Most Significant Bit) να είναι 1.

4. **Στρογγυλοποίηση**: Το προϊόν των κλασματικών μερών στρογγυλεύεται στην ακρίβεια της μορφής αποτελέσματος χρησιμοποιώντας καθορισμένες λειτουργίες στρογγυλοποίησης. Οι τρόποι στρογγυλοποίησης περιλαμβάνουν στρογγυλοποίηση προς το πλησιέστερο (Round to Nearest), στρογγυλοποίηση από το μηδέν (Round Away from Zero), στρογγυλοποίηση προς το μηδέν (Round Towards Zero) και στρογγυλή προς το θετικό ή αρνητικό άπειρο (Round Towards Positive/Negative Infinity). Στην δική μας περίπτωση, θα χρησιμοποιήσουμε την μέθοδο στρογγυλοποίησης προς το μηδέν, αφαιρώντας απλά τα επιπλέον bit που δεν χωράνε στο τελικό αποτέλεσμα από τον πολλαπλασιασμό των κλασματικών μερών.

5. **Κατασκευή Αποτελέσματος**: Το ψηφίο προσήμου, ο εκθέτης και το στρογγυλεμένο κλασματικό μέρος συνδυάζονται στην κατάλληλη μορφή για το αποτέλεσμα. Αν χρειαστεί γεμίζουμε τα λιγότερο σημαντικά ψηφία (LSB - Least Significant Bit) του κλασματικού μέρους με 0, ώστε να ταιριάζει στην σωστή μορφή,



Σχήμα 4.7: Block diagram ενός πολλαπλασιαστή αριθμών κινητής υποδιαστολής

Τελικά, όπως βλέπουμε, για την τέλεση της πράξης του πολλαπλασιασμού με βάση τα βήματα που αναφέρθηκαν, από πράξεις χρειάζεται να κάνουμε έναν πολλαπλασιασμό σε επίπεδο bit και μια πρόσθεση. Για την πρόσθεση, θα χρησιμοποιήσουμε έναν Ripple Carry Adder[1], λόγω της ευκολίας στην κατασκευή τους και της αποδοτικότητάς τους σε χρόνο και υλικό (Hardware). Για το τμήμα του πολλαπλασιαστή, θα χρησιμοποιήσουμε τέσσερις διαφορετικές υλοποιήσεις, με σκοπό να ερευνήσουμε την αποδοτικότητα σε χώρο, χρόνο και κατανάλωση της κάθε μεθόδου. Αυτοί οι τέσσεροις μέθοδοι είναι ο Array Multiplier, ο Sequential Multiplier, ο Vedic Multiplier και ο Wallace Tree Multiplier.

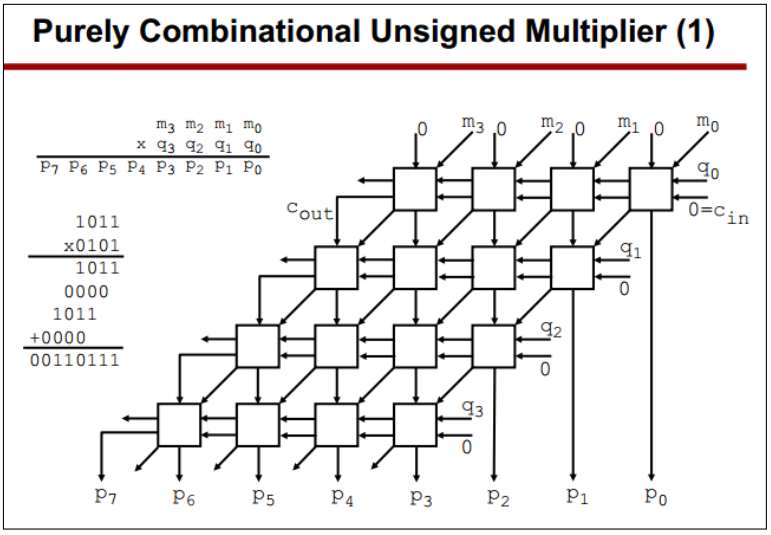
[1]Παράρτημα Γ′ - Ripple Carry Adder

Κεφάλαιο 5

Αλγόριθμοι Πολλαπλασιασμού

5.1 Array Multiplier

Ένας πολλαπλασιαστής πίνακα είναι ένα ψηφιακό κύκλωμα σχεδιασμένο να εκτελεί πολλαπλασιασμό δυαδικών αριθμών χρησιμοποιώντας μια σειρά βασικών λογικών πυλών. Σε αντίθεση με τα παραδοσιακά κυκλώματα πολλαπλασιαστή, τα οποία χρησιμοποιούν διαδοχικές πράξεις, οι πολλαπλασιαστές συστοιχιών αξιοποιούν τον παραλληλισμό για να βελτιώσουν την υπολογιστική απόδοση. Τα θεμελιώδη δομικά στοιχεία ενός πολλαπλασιαστή πίνακα περιλαμβάνουν σειρές και στήλες πλήρων αθροιστών, οργανωμένες σε μια δομή που μοιάζει με πλέγμα. Κάθε κελί στον πίνακα επεξεργάζεται ένα συγκεκριμένο ζεύγος bit από τον πολλαπλασιαστή και τον πολλαπλασιαστέο, παράγοντας μερικά γινόμενα. Αυτά τα μερικά προϊόντα στη συνέχεια μετατοπίζονται κατάλληλα και προστίθενται για να δώσουν το τελικό προϊόν. Το πλεονέκτημα των πολλαπλασιαστών συστοιχιών έγκειται στην ικανότητά τους να χειρίζονται ταυτόχρονα πολλά ζεύγη bit, με αποτέλεσμα μειωμένο χρόνο πολλαπλασιασμού σε σύγκριση με τις διαδοχικές μεθόδους. Αυτό τα καθιστά κατάλληλα για εφαρμογές που απαιτούν αριθμητικές λειτουργίες υψηλής ταχύτητας, όπως στην επεξεργασία ψηφιακών σημάτων.



Σχήμα 5.1: Δομή ενός 4 bit Array Multiplier

Παρά την πολυπλοκότητά τους, οι σύγχρονες τεχνικές σχεδιασμού και οι αλγόριθμοι

βελτιστοποίησης συμβάλλουν στην αποτελεσματικότητα των array multipliers, εξασφαλίζοντας ότι ανταποκρίνονται στην αυξανόμενη ζήτηση για υπολογισμό υψηλής ταχύτητας και ενεργειακού αποδοτικού σε διαφορετικά περιβάλλοντα υπολογιστών.

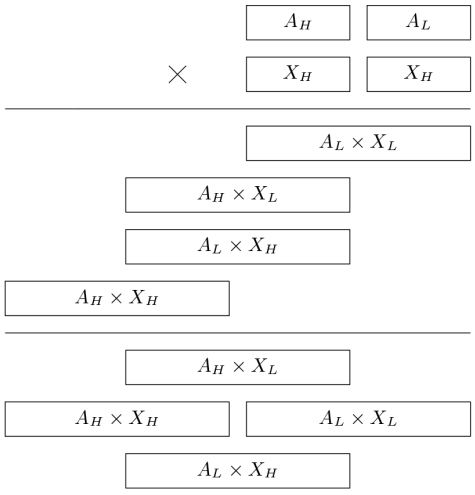
5.1.1 Κατασκευή του Array Multiplier

Μια μέθοδος μείωσης των μερικών προϊόντων ενός πολλαπλασιαστής γίνεται δημιουργώντας υψηλότερους πολλαπλασιαστές από μικρότερους. Τα μεγαλύτερα μπλοκ πολλαπλασιαστή μπορούν να πραγματοποιηθούν χρησιμοποιώντας μικρότερα μπλοκ πολλαπλασιασμού. Ένας πολλαπλασιασμός $2N \times 2n$ μπορεί να πραγματοποιηθεί χρησιμοποιώντας τέσσερις $n \times n$ μπλοκ πολλαπλασιαστών. Αυτό βασίζεται στην ακόλουθη εξίσωση:

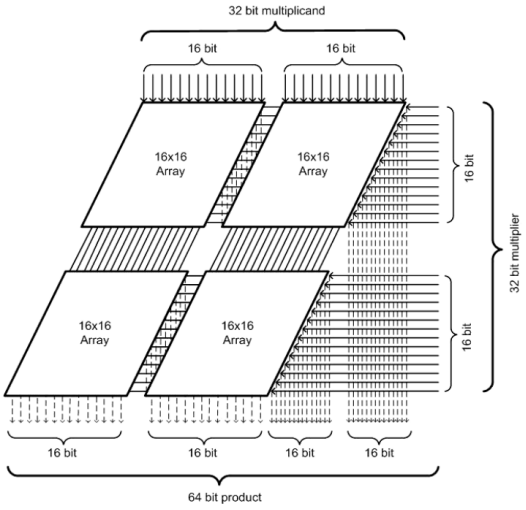
$$A \cdot X = (A_H \cdot 2^n + A_L) \cdot (X_H \cdot 2^n + X_L) = A_H \cdot X_H \cdot 2^{2n} + (A_H \cdot X_L + A_L \cdot X_H) \cdot 2^n + A_L \cdot X_L \quad (5.1)$$

Όπου το A_H είναι το σημαντικότερο μισό του A, το X_H είναι το πιο σημαντικό μισό του X, το A_L είναι το λιγότερο σημαντικό μισό του A και X_L είναι το λιγότερο σημαντικό μισό του X.

Τα μερικά προϊόντα από τα μικρότερα μπλοκ πολλαπλασιαστή θα πρέπει να είναι σωστά διατεταγμένα και συσσωρευμένα από adders. Ένα σχήμα εφαρμογής ενός πολλαπλασιαστή 8 bit χρησιμοποιώντας τέσσερις 4 bit πολλαπλασιαστές παρουσιάζεται στο παρακάτω σχήμα.



Σχήμα 5.2: Δομή ενός 8 bit Array Multiplier using four 4 bit Array Multipliers

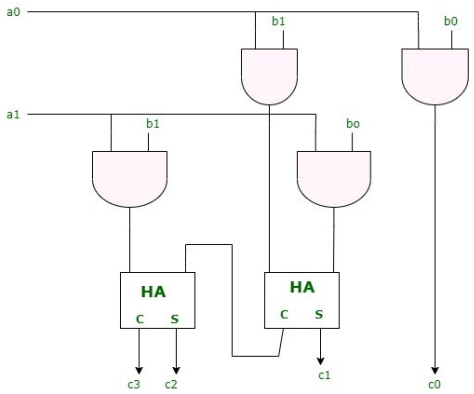


Σχήμα 5.3: Δομή ενός 32 bit Array Multiplier using four 16 bit Array Multipliers

Έτσι, μπορούμε να ξεκινήσουμε με την κατασκευή ενός 2×2 bit array multiplier και μέσω της μεθόδου που εξηγήσαμε να κατασκευάσουμε όλους τους απαραίτητους πολλαπλασιαστές για το πρότυπο IEEE 754 (στην δική μας περίπτωση χρειάζεται να κατασκευάσουμε μέχρι και πολλαπλασιαστή 64×64 bits για το double precision).

5.1.2 2×2 Array Multiplier

Ο σχεδιασμός του 2×2 Array Multiplier είναι πολύ απλός αφού το μόνο που χρειαζόμαστε είναι 2 Half Adders και 4 λογικές πύλες AND. Με βάση το Σχήμα 6.3, δημιουργούμε τον αντίστοιχο κώδικα στην Verilog.



Σχήμα 5.4: Δομή ενός 2 bit Array Multiplier

```
//Array_2x2.v
module Array_2x2(a,b,c);

input [1:0] a,b;
output [3:0]c;
wire [3:0] c,temp;

assign c[0]=a[0]&b[0];
assign temp[0]=a[1]&b[0];
assign temp[1]=a[0]&b[1];
assign temp[2]=a[1]&b[1];
half_adder U1 (.a(temp[0]),.b(temp[1]),.sum(c[1]),.carry(temp[3]));
half_adder U2 (.a(temp[2]),.b(temp[3]),.sum(c[2]),.carry(c[3]));

endmodule
```

5.1.3 Κατασκευή των υπόλοιπων πολλαπλασιαστών

Για την υλοποίηση μεγαλύτερων πολλαπλασιαστών, χρησιμοποιούμε μικρότερους πολλαπλασιαστές και εφαρμόζουμε την μέθοδο που ακολουθήσαμε στο 5.1.1. Ως παράδειγμα, θα σχεδιάσουμε το ψηφιακό κύκλωμα του 4 bit array multiplier χρησιμοποιώντας τον πολλαπλασιαστή που σχεδιάσαμε στο 5.1.2.

```
//Array_4x4.v
module Array_4x4(a,b,c);

input [3:0]a, b;
output [7:0]c;

   wire [3:0]q0,q1,q2,q3,q4,temp1;
```

```
    wire [7:0]c;
    wire [5:0]q5,q6,temp2,temp3,temp4;

    Array_2x2 U1 (a[1:0],b[1:0],q0[3:0]);
    Array_2x2 U2 (a[3:2],b[1:0],q1[3:0]);
    Array_2x2 U3 (a[1:0],b[3:2],q2[3:0]);
    Array_2x2 U4 (a[3:2],b[3:2],q3[3:0]);

    assign temp1 ={2'b0,q0[3:2]};
    assign q4 = q1[3:0]+temp1;
    assign temp2 ={2'b0,q2[3:0]};
    assign temp3 ={q3[3:0],2'b0};
    assign q5 = temp2+temp3;
    assign temp4={2'b0,q4[3:0]};
    assign q6 = temp4+q5;

    assign c[1:0] = q0[1:0];
    assign c[7:2] = q6[5:0];

endmodule
```

Οι μεγαλύτεροι πολλαπλασιαστές κατασκευάζονται με την ίδια λογική, με την διαφορά ότι αλλάζουμε κάθε φορά το μήκος των μεταβλητών.

5.2 Sequential Multiplier

Στον τομέα του σχεδιασμού ψηφιακής λογικής, οι διαδοχικοί πολλαπλασιαστές προσφέρουν μια εναλλακτική προσέγγιση για τον πολλαπλασιασμό δυαδικών αριθμών. Αυτά τα κυκλώματα αξιοποιούν έναν απλό αθροιστή n bit (όπου το n αντιπροσωπεύει το μήκος bit του πολλαπλασιαστή) για να εκτελέσει τη διαδικασία πολλαπλασιασμού επαναληπτικά. Αυτή η επαναληπτική προσέγγιση έρχεται σε αντίθεση με τη συνδυαστική προσέγγιση, η οποία χρησιμοποιεί μια αποκλειστική δομή υλικού για κάθε μερική παραγωγή προϊόντος. Ο διαδοχικός πολλαπλασιαστής επιτυγχάνει απόδοση περιοχής (Area Efficiency) θυσιάζοντας την υπολογιστική ταχύτητα (Computational Speed). Κάθε επανάληψη εντός του διαδοχικού πολλαπλασιαστή περιλαμβάνει τρεις διακριτές πράξεις:

- Δημιουργία ενός μερικού γινομένου με βάση το τρέχον bit του πολλαπλασιαστή και ολόκληρου του πολλαπλασιαστέου.
- Συσσώρευση του παραγόμενου μερικού γινομένου με το μερικό άθροισμα που προκύπτει από προηγούμενες επαναλήψεις σε έναν καταχωρητή (Register).
- Μετατόπιση προς τα δεξιά του συσσωρευμένου μερικού αθροίσματος για διευκόλυνση της ευθυγράμμισης με το επόμενο μερικό γινόμενο

Αυτός ο κύκλος μερικής δημιουργίας προϊόντος, συσσώρευσης και μετατόπισης επαναλαμβάνεται για όλα τα n bit του πολλαπλασιαστή, δίνοντας τελικά το γινόμενο. Έτσι, ο αριθμός των κύκλων ρολογιού που απαιτείται για να έχουμε ορθό αποτέλεσμα είναι ανάλογος με τον αριθμό των bits του πολλαπλασιαστή (Clock Cycles $\propto n$).

5.2.1 Αλγόριθμος

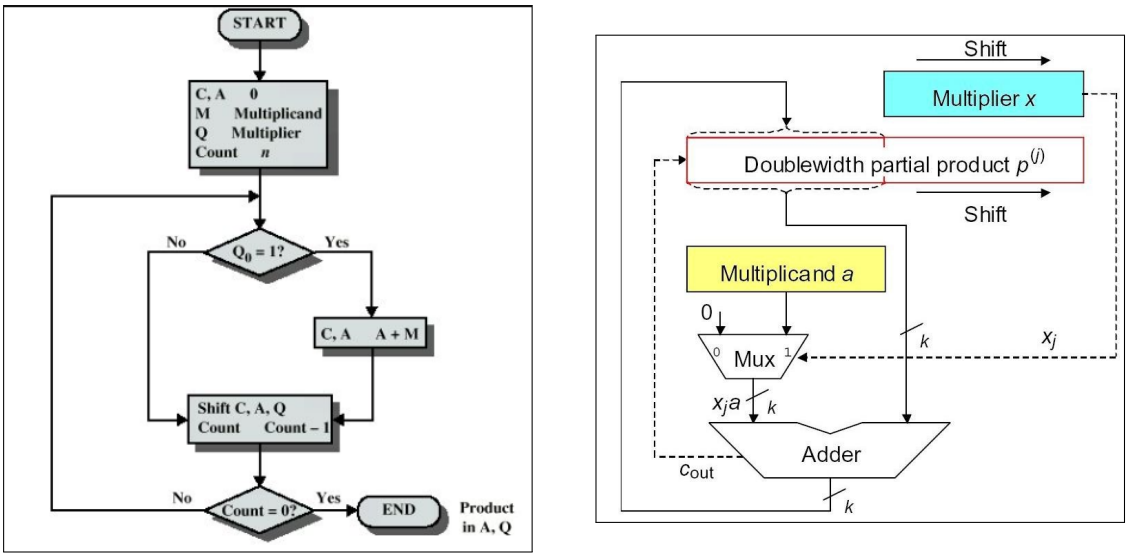
- Αρχικοποίηση
  - Θέτουμε την τιμή του πολλαπλασιαστή σε καταχωρητή M και του πολλαπλασιαστέου σε καταχωρητή Q.
  - Αρχικοποιούμε τις τιμές των καταχωρητών C και A στην τιμή μηδέν.
- Επαναλαμβάνουμε τα ακόλουθα βήματα n φορές, όπου n είναι ο αριθμός των bit στον πολλαπλασιαστή
  - Εάν το LSB του καταχωρητή $Q == 1 \rightarrow A = A + M$ (η υπερχείλιση πηγαίνει στον καταχωρητή C).
  - Εάν το LSB του καταχωρητή $Q \neq 1$, συνεχίζουμε στο επόμενο βήμα.
  - Αντιμετωπίζουμε τους καταχωρητές C, A και Q ως έναν συνεχόμενο καταχωρητή και μετατοπίζουμε τα περιεχόμενα του δεξιά κατά μια θέση bit.
- Μετά την ολοκλήρωση των βημάτων n
  - Ο καταχωρητής A περιέχει τα MSBs του τελικού προϊόντος.
  - Ο καταχωρητής Q περιέχει τα LSBs του τελικού προϊόντος.

Algorithm 1: Sequential Multiplier

```
1  Q = x;
2  M = y;
3  A, C ← 0;
4  counter ← n ;                          //n is the bit length of x,y
5  while counter > 0 do
6      if Q[0] = 1 then
7          {C, A} ← A + M;
8          {C, A, Q} >> 1;
9          counter ← counter − 1;
10     else
11         {AC, QR} >> 1;
12         counter ← counter − 1;
13 result ← {A, Q};
```

Παρακάτω βλέπουμε την υλοποίηση του αλγορίθμου στην γλώσσα Verilog όπου παρατηρούμε ότι ο ίδιος κώδικας μπορεί να χρησιμοποιηθεί για οποιοδήποτε μέγεθος πολλαπλασιαστή ή πολλαπλασιαστέου, αλλάζοντας απλά τις παραμέτρους width και counter_value. Παρακάτω εμφανίζεται η υλοποίηση για τον πολλαπλασιασμό αριθμών των τεσσάρων bit.

```
//Sequential_Mul.v
module Sequential_Mul#(
parameter width = 4,
        counter_value = 4
)(
input wire [width - 1 : 0] a,b,
input wire start,clk,
output wire [2*width - 1 : 0] result,
output reg valid
);

reg [width - 1 : 0] A,Q,M;
reg C;
reg [width - 1 : 0] counter;
wire [width : 0] diff,sum;

   always @(posedge clk, posedge start)
     begin
       if (start)
         begin
            A <= 4'b0;
            M <= b;
            Q <= a;
            counter <= counter_value;
            valid <= 1'b0;
         end
       else
         begin
            case (Q[0])
              1'b1 : begin
                    {C, A} = A + M;
                    {C,A,Q} = {1'b0,C,A,Q[width-1:1]};
                        end
              1'b0 : {C,A,Q} = {1'b0,C,A,Q[width-1:1]};
            endcase
            counter <= counter + 1'b1;
          end
         valid = (&counter) ? 1'b1 : 1'b0;
     end

assign result = {A,Q};

endmodule
```



Σχήμα 5.5: Flow Chart αλγορίθμου Sequential Multiplier και Κυκλωματική Δομή

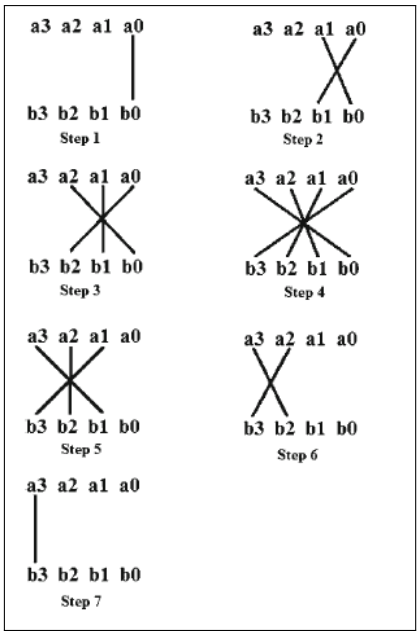
5.3 Vedic Multiplier

Στο πεδίο του ψηφιακού σχεδιασμού, οι πολλαπλασιαστές Vedic έχουν αναδειχθεί ως μια ισχυρή εναλλακτική λύση σε σχέση με τις συμβατικές μεθόδους πολλαπλασιασμού. Αντλώντας έμπνευση από το αρχαίο ινδικό μαθηματικό σύστημα, αυτοί οι αλγόριθμοι προσφέρουν μια ξεχωριστή προσέγγιση, που χαρακτηρίζεται από ταχύτητα, area efficiency και μείωση της κατανάλωσης ενέργειας. Η θεμελιώδης αρχή αυτού του αλγορίθμου είναι η επιλεκτική δημιουργία και πρόσθεση μερικών προϊόντων, οδηγώντας σε σημαντική μείωση του αριθμού των απαιτούμενων πράξεων. Αυτό μεταφράζεται σε βελτιωμένη ταχύτητα, καθιστώντας τους Vedic πολλαπλασιαστές ιδιαίτερα ελκυστικούς για εφαρμογές που απαιτούν απόδοση σε πραγματικό χρόνο, όπως η επεξεργασία ψηφιακού σήματος (DSP - Digital Signal Processing).

Με την ελαχιστοποίηση του αριθμού πράξεων και τον μειωμένο αριθμό μερικών προϊόντων, αυτοί οι αλγόριθμοι απαιτούν λιγότερο υλικό σε σύγκριση με τις συμβατικές μεθόδους. Αυτό είναι ιδιαίτερα πλεονεκτικό σε καταστάσεις περιορισμένων πόρων, όπως ενσωματωμένα συστήματα και ολοκληρωμένα κυκλώματα, όπου η ελαχιστοποίηση των πόρων υλικού μεταφράζεται άμεσα σε μείωση κόστους και κατανάλωσης ενέργειας.

5.3.1 Αλγόριθμος

Η βασική ιδέα αυτού του αλγορίθμου είναι ο κατακόρυφος και σταυρωτός πολλαπλασιασμός. Έτσι, θεωρούμε δύο δυαδικούς αριθμούς των 4 bit που έχουν τιμές $a[3:0]$ και $b[3:0]$. Το προϊόν του πολλαπλασιασμού είναι οι τιμές $p[7:0]$. Τα μερικά προϊόντα έχουν τις τιμές $t[7:0]$ και κατασκευάζονται με βάση τα παρακάτω δύο σχήματα:



Step1: t_0 (1 bit) $= a_0 b_0$

Step2: t_1 (2 bit) $= a_1 b_0 + a_0 b_1 = t_1[1]\, t_1[0]$

Step3: t_2 (2 bit) $= a_2 b_0 + a_1 b_1 + a_0 b_2 = t_2[1]\, t_2[0]$

Step4: t_3 (3 bit) $= a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3 = t_3[2]\, t_3[1]\, t_3[0]$

Step5: t_4 (2 bit) $= a_3 b_1 + a_2 b_2 + a_1 b_3 = t_4[1]\, t_4[0]$

Step6: t_5 (2 bit) $= a_3 b_2 + a_2 b_3 = t_5[1]\, t_5[0]$

Step7: t_6 (1 bit) $= a_3 b_3$

Σχήμα 5.6: Line Notation και τιμές μερικών προϊόντων

Το τελικό αποτέλεσμα είναι η πρόσθεση των s_1, s_2 και s_3:

$$s_1 = t_6 t_5[0]\, t_4[0]\, t_3[0]\, t_2[0]\, t_1[0]\, t_0$$

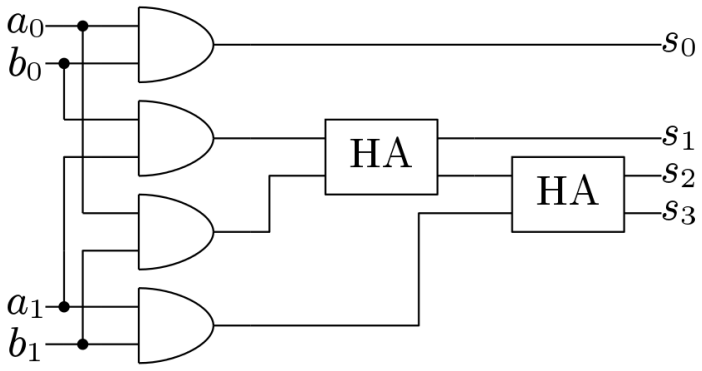
$$s_2 = t_5[1]\, t_4[1]\, t_3[1]\, t_2[1]\, t_1[1]$$

$$s_3 = t_3[2]$$

$$\begin{array}{rcccccccc} \text{Product} = & t_6 & t_5[0] & t_4[0] & t_3[0] & t_2[0] & t_1[0] & & t_0 \\ + & t_5[1] & t_4[1] & t_3[1] & t_2[1] & t_1[1] & & 0 & \\ + & t_3[2] & 0 & 0 & 0 & 0 & & & \\ \hline & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 \end{array}$$

Σχήμα 5.7: Τιμές των s_1, s_2, s_3 και τελικό αποτέλεσμα

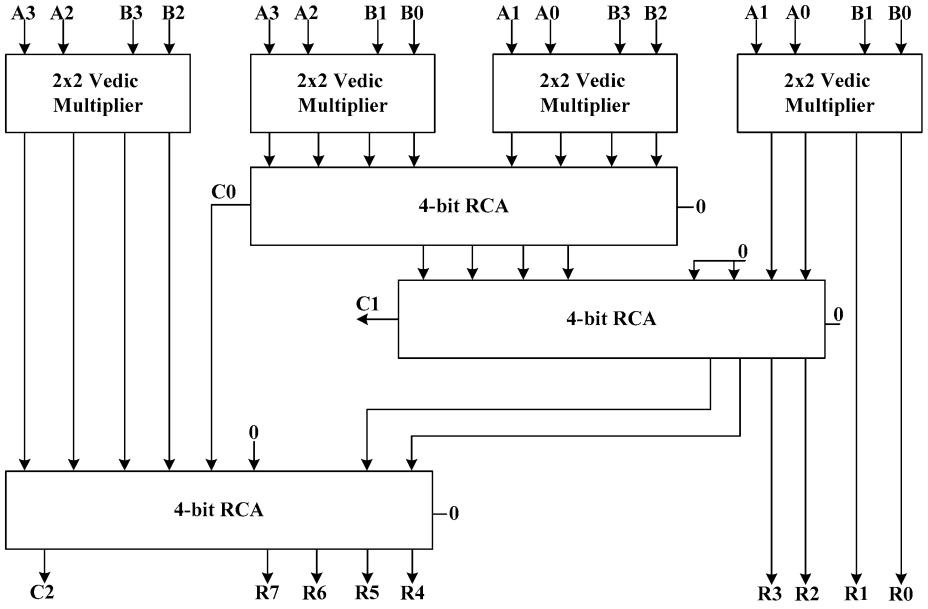
Με βάση την λογική που περιγράψαμε, κατασκευάζουμε στην Verilog ένα 2×2 bit Vedic πολλαπλασιαστή του οποίου το κύκλωμα και το προγραμματιστικό τμήμα είναι:



Σχήμα 5.8: Κυκλωματική Δομή ενός 2×2 Vedic Multiplier

```
//vedic_2x2.v
module vedic_2x2(a, b, result);

input [1:0] a,b;
output [3:0] result;

wire [3:0] w;

assign result[0] = a[0]&b[0];
assign w[0] = a[1]&b[0];
assign w[1] = a[0]&b[1];
assign w[2] = a[1]&b[1];

half_adder H1(w[0], w[1], result[1], w[3]);
half_adder H2(w[2], w[3], result[2], result[3]);

endmodule
```

Για την κατασκευή πολλαπλασιαστών μεγαλύτερου μεγέθους, όπως και στην περίπτωση του πολλαπλασιαστή πίνακα (Array Multiplier), μπορούμε να χρησιμοποιήσουμε ως δομικά στοιχεία τέσσερεις πολλαπλασιαστές 2×2 για να δημιουργήσουμε έναν 4×4 bit Vedic πολλαπλασιαστή.



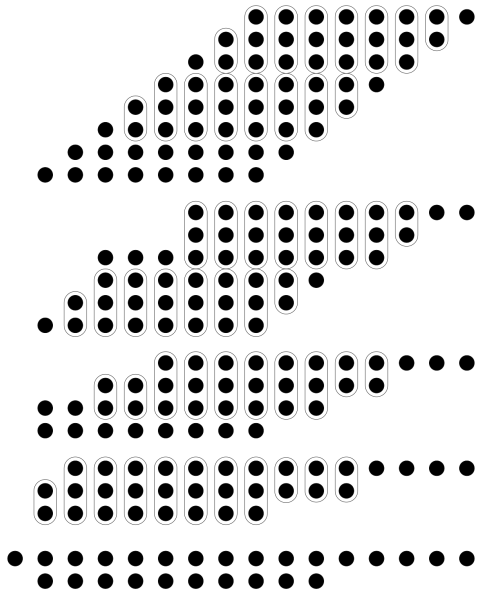
Σχήμα 5.9: 4×4 bit Vedic Multiplier

Αντίστοιχα κατασκευάζονται και οι Vedic πολλαπλασιαστές μεγαλύτερου αριθμού bits.

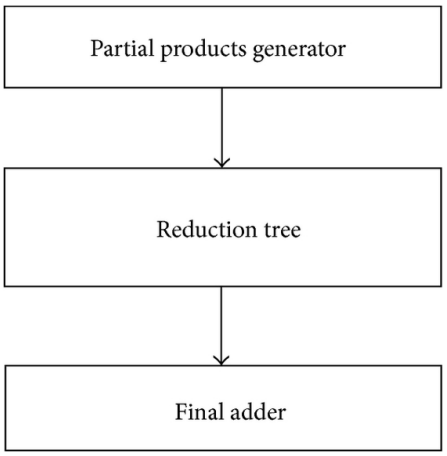
5.4 Wallace Tree Multiplier

Στον τομέα του ψηφιακού σχεδιασμού, οι πολλαπλασιαστές δέντρων Wallace αντιπροσωπεύουν μια εξειδικευμένη αρχιτεκτονική υλικού για δυαδικό πολλαπλασιασμό υψηλής ταχύτητας. Σε αντίθεση με τις βασικές προσεγγίσεις που περιλαμβάνουν έναν πλήρη αθροιστή για κάθε μερικό προϊόν, οι πολλαπλασιαστές Wallace χρησιμοποιούν μια δομή πολλαπλών σταδίων γνωστή ως Wallace Tree Reduction. Αυτό το δέντρο, που αποτελείται από στρατηγικά διατεταγμένους πλήρεις και μισούς αθροιστές, εκτελεί μια μερική μείωση των παραγόμενων μερικών προϊόντων. Με την αποτελεσματική ομαδοποίηση και μερική άθροιση αυτών των όρων (συχνά σε σετ των τριών), το δέντρο Wallace μειώνει σημαντικά τον αριθμό των τελικών σταδίων προσθήκης που απαιτούνται. Αυτή η μείωση σταδίων μεταφράζεται σε ταχύτερο πολλαπλασιασμό σε σύγκριση με τις συμβατικές τεχνικές. Ωστόσο, η αρχιτεκτονική του δέντρου Wallace έχει ένα κόστος. Η αυξημένη πολυπλοκότητα της δομής του δέντρου απαιτεί μεγαλύτερη περιοχή κυκλώματος σε σύγκριση με απλούστερους πολλαπλασιαστές.

Στόχος του δεντρικού συμπιεστή Wallace είναι να γίνει η συμπίεση των μερικών γινομένων σε όσο το δυνατόν λιγότερα επίπεδα Full Adder και Half Adder και συνεπώς σε όσο το δυνατόν μικρότερο χρονικό διάστημα. Για το σκοπό αυτό, σε κάθε επίπεδο, τα ψηφία ίδιου βάρους (στην ίδια στήλη) ομαδοποιούνται ανά τρία και εισέρχονται σαν είσοδοι σε έναν Full Adder, εάν περισσέψουν δύο, τότε ομαδοποιούνται ανά δύο και εισέρχονται σαν είσοδοι σε έναν Half Adder, ενώ εάν περισσέψει μόνο ένα, μεταφέρεται στο επόμενο επίπεδο. Αμέσως παρακάτω παρουσιάζεται το τμήμα συμπίεσης των μερικών γινομένων ενός 8x8 bit πολλαπλασιαστή Wallace. Κάθε κουκκίδα συμβολίζει ένα bit του αντίστοιχου μερικού γινομένου. Όπου υπάρχει ομαδοποίηση τριών bit, τα τρία αυτά bit εισέρχονται σαν είσοδοι σε έναν FA και όπου υπάρχει ομαδοποίηση δύο bit, τα δύο αυτά bit εισέρχονται σαν είσοδοι σε έναν HA. Προφανώς, κάθε FA και HA παράγουν ως αποτέλεσμα ένα bit ίδιου βάρους (Sum) και ένα bit με το αμέσως μεγαλύτερο βάρος (Carry - C_{out}).



Σχήμα 5.10: 4 layer Wallace reduction of an 8x8 partial product matrix



Σχήμα 5.11: Δομή του Wallace πολλαπλασιαστή

Επίσης, με βάση την μέθοδο που χρησιμοποιήθηκε και στον Array Multiplier, θα κατασκευάσουμε τον πολλαπλασιαστή Wallace για 8×8 bit πολλαπλασιασμό και οι μεγαλύτεροι τάξης πολλαπλασιαστές θα κατασκευαστούν με βάση τον τελευταίο.

```
//Wallace_8x8.v
module Wallace_8x8(
input wire [7:0] a,b,
output [15:0] result
);

wire [63:0] pp;
wire [62:0] c;
wire [52:0] s;

genvar i,j;
generate
   for (i=0; i < 8; i = i + 1) begin
      for (j=0; j < 8; j = j + 1) begin
         assign pp[8*i+j] = a[i] & b[j];
      end
   end
endgenerate

//Level 1 - Stage 1
half_adder HA1 (pp[1],pp[8],s[0],c[0]);

generate
   for (i=0; i < 6; i = i + 1) begin: L1_S1
      full_adder FAi_L1_S1
         (pp[2+i],pp[9+i],pp[16+i],s[i+1],c[i+1]);
   end
endgenerate

half_adder HA2 (pp[15],pp[22],s[7],c[7]);


//Level 1 - Stage 2
half_adder HA3 (pp[25],pp[32],s[8],c[8]);

generate
   for (i=0; i < 6; i = i + 1) begin: L1_S2
      full_adder FAi_L1_S2
         (pp[26+i],pp[33+i],pp[40+i],s[i+9],c[i+9]);
   end
endgenerate

half_adder HA4 (pp[39],pp[46],s[15],c[15]);


//Level 2 - Stage 1
half_adder HA5 (s[1],c[0],s[16],c[16]);
```

```
full_adder FA11_L2 (s[2],c[1],pp[24],s[17],c[17]);

generate
   for (i=0; i < 5; i = i + 1) begin: L2_S1
      full_adder FAi_L2_S1 (s[3+i],c[2+i],s[8+i],s[i+18],c[i+18]);
   end
endgenerate

full_adder FA22_L2 (pp[23],c[7],s[13],s[23],c[23]);


//Level 2 - Stage 2
half_adder HA6 (c[9],pp[48],s[24],c[24]);

generate
   for (i=0; i < 6; i = i + 1) begin: L2_S2
      full_adder FAi_L2_S2
         (c[10+i],pp[49+i],pp[56+i],s[i+25],c[i+25]);
   end
endgenerate

half_adder HA7 (pp[55],pp[62],s[31],c[31]);


//Level 3 - Stage 1
half_adder HA8 (s[17],c[16],s[32],c[32]);
half_adder HA9 (s[18],c[17],s[33],c[33]);

full_adder FA11_L3 (s[19],c[18],c[8],s[34],c[34]);

generate
   for (i=0; i < 4; i = i + 1) begin: L3_S1
      full_adder FAi_L3_S1
         (s[20+i],c[19+i],s[24+i],s[i+35],c[i+35]);
   end
endgenerate

full_adder FA22_L3 (s[14],c[23],s[28],s[39],c[39]);

half_adder HA10 (s[15],s[29],s[40],c[40]);
half_adder HA11 (pp[47],s[30],s[41],c[41]);


//Level 4 - Stage 1
assign result[0] = pp[0];
assign result[1] = s[0];
assign result[2] = s[16];
assign result[3] = s[32];

generate
   for (i=0; i < 3; i = i + 1) begin: HA_L4_S1
```

```
      half_adder HAi_L4_S1 (s[33+i],c[32+i],s[42+i],c[i+42]);
   end
endgenerate

generate
   for (i=0; i < 6; i = i + 1) begin: L4_S1
      full_adder FAi_L4_S1
         (c[24+i],s[36+i],c[35+i],s[i+45],c[i+45]);
   end
endgenerate

full_adder FA11_L4 (c[30],s[31],c[41],s[51],c[51]);
half_adder HA12 (pp[63],c[31],s[52],c[52]);


//Level 5 - Stage 1
assign result[4] = s[42];

half_adder HA13 (s[43],c[42],result[5],c[53]);

generate
   for (i=0; i < 9; i = i + 1) begin: L5_S1
      full_adder FAi_L5_S1
         (s[44+i],c[43+i],c[53+i],result[i+6],c[i+54]);
   end
endgenerate

half_adder HA14 (c[52],c[62],result[15],result[16]);

endmodule
```

Παρακάτω κατασκευάζουμε έναν 16×16 bit πολλαπλασιαστή Wallace με βάση τον 8×8 πολλαπλασιαστή που κατασκευάσαμε παραπάνω.

```
\\Wallace_16.v
module Wallace_16(
input wire [15:0] a,b,
output [31:0] result,
wire [7:0] Pa,Pb,Qa,Qb,Ra,Rb,Sa,Sb,
wire [15:0] tempP,tempQ,tempR,tempS,temp1,temp2,temp3,
wire cout1,cout2,cout3
   );

assign Pa = a[7:0];
assign Pb = b[7:0];
assign Qa = a[15:8];
assign Qb = b[7:0];
assign Ra = a[7:0];
assign Rb = b[15:8];
assign Sa = a[15:8];
assign Sb = b[15:8];
```

```
Wallace_8x8 U1 (.a(Pa), .b(Pb), .result(tempP));
Wallace_8x8 U2 (.a(Qa), .b(Qb), .result(tempQ));
Wallace_8x8 U3 (.a(Ra), .b(Rb), .result(tempR));
Wallace_8x8 U4 (.a(Sa), .b(Sb), .result(tempS));

adder_16bits U5 (.a(tempQ), .b(tempR), .sum(temp1), .cout(cout1));
adder_16bits U6 (.a({16'b0,tempP[15:8]}), .b(temp1), .sum(temp2),
   .cout(cout2));
adder_16bits U7 (.a({15'b0,cout1,temp2[15:8]}), .b(tempS),
   .sum(temp3), .cout(cout3));

assign result = {temp3,temp2[7:0],tempP[7:0]};

endmodule
```

Κεφάλαιο 6

Προσομοίωση και Αποτελέσματα Αλγορίθμων

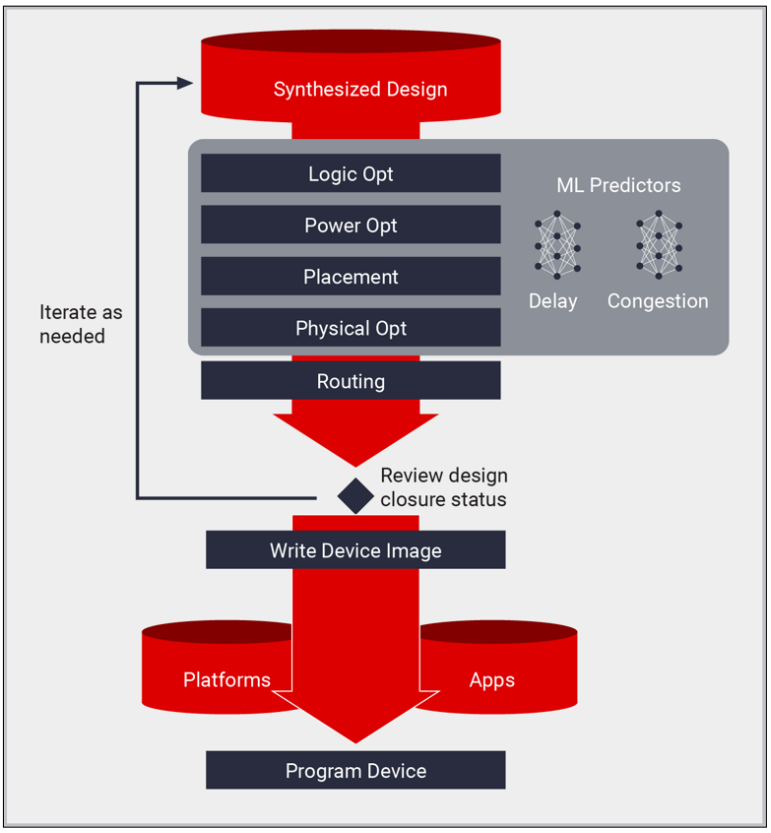
6.1 Προσομοιώσεις στο Xilinx Vivado

Το Xilinx Vivado παρέχει ένα ισχυρό και ευέλικτο περιβάλλον για το σχεδιασμό και την υλοποίηση ψηφιακών κυκλωμάτων σε FPGA. Αυτές οι διαδικασίες περιλαμβάνουν πολλά κρίσιμα στάδια, καθένα από τα οποία παίζει ζωτικό ρόλο στη μετατροπή ενός ψηφιακού κυκλώματος που περιγράφεται σε μια γλώσσα περιγραφής υλικού (HDL) σε ένα λειτουργικό κύκλωμα στο FPGA.

- **Λειτουργική επαλήθευση μέσω προσομοίωσης**: Το αρχικό στάδιο περιγράφτηκε εκτενώς και στο Κεφάλαιο 3 και δίνει προτεραιότητα στην επαλήθευση της επιδιωκόμενης συμπεριφοράς του σχεδίου πριν από τη δέσμευση για υλοποίηση υλικού. Αυτό επιτυγχάνεται μέσω της προσομοίωσης, όπου τα testbenches είναι κατασκευασμένα για να μιμούνται τις εισόδους του πραγματικού κόσμου και να συγκρίνουν τα αποτελέσματα του σχεδιασμού με τις προκαθορισμένες προσδοκίες. Ουσιαστικά, η προσομοίωση λειτουργεί ως σχεδίαση σε ελεγχόμενο ψηφιακό περιβάλλον, εκθέτοντας πιθανά λειτουργικά ελαττώματα πριν από την εφαρμογή του πραγματικού υλικού.
- **Στατική ανάλυση και έλεγχος κανόνων σχεδίασης με ανάλυση RTL**: Μετά την επιτυχή προσομοίωση, ο σχεδιασμός υποβάλλεται σε ενδελεχή εξέταση χρησιμοποιώντας τα εργαλεία ανάλυσης RTL (Register-Transfer Level - Επίπεδο Εγγραφής-Μεταφοράς) του Vivado. Η βασική περιγραφή του κυκλώματος, ελέγχεται για συντακτικά λάθη, τήρηση προτύπων κωδικοποίησης και πιθανές παραβιάσεις κανόνων σχεδίασης. Αυτή η στατική ανάλυση χρησιμεύει ως κρίσιμος έλεγχος ποιότητας, διασφαλίζοντας ότι ο σχεδιασμός συμμορφώνεται με τις καθιερωμένες βέλτιστες πρακτικές και εντοπίζει δομικά ζητήματα που μπορεί να οδηγήσουν σε απροσδόκητη συμπεριφορά στο υλικό.
- **Σύνθεση**: Έχοντας περάσει τον έλεγχο της ανάλυσης RTL, ο σχεδιασμός προχωρά στο στάδιο της σύνθεσης. Εδώ, το Vivado μετατρέπει τον αναγνώσιμο από τον άνθρωπο κώδικα HDL σε ένα netlist χαμηλού επιπέδου, ουσιαστικά ένα λεπτομερές σχέδιο του κυκλώματος που έχει κατασκευαστεί από θεμελιώδεις λογικές πύλες. Αυτό το netlist χρησιμεύει ως μια ενδιάμεση αναπαράσταση, αποτυπώνοντας τη

λειτουργικότητα του σχεδιασμού πάνω στο υλικό. Η διαδικασία σύνθεσης συχνά περιλαμβάνει βελτιστοποιήσεις για τη μείωση του μεγέθους του κυκλώματος, τη βελτίωση της απόδοσης ή τη βελτίωση της απόδοσης ισχύος.

- **Υλοποίηση**: Το τελικό στάδιο γεφυρώνει το χάσμα μεταξύ του αφηρημένου netlist και της αρχιτεκτονικής του FPGA. Το Vivado τοποθετεί και δρομολογεί τα λογικά κελιά που περιγράφονται στο netlist στο FPGA. Αυτή η περίπλοκη διαδικασία περιλαμβάνει την εκχώρηση συγκεκριμένων λογικών κυψελών για την υλοποίηση των λειτουργιών που περιγράφονται στο netlist, τη δρομολόγηση των καλωδίων μεταξύ τους για τη δημιουργία των απαραίτητων συνδέσεων και τη δημιουργία περιορισμών χρονισμού για να διασφαλιστεί ότι το κύκλωμα λειτουργεί αξιόπιστα σύμφωνα με τις προδιαγραφές χρονισμού του FPGA. Η επιτυχής υλοποίηση παράγει το bitstream, ένα αρχείο διαμόρφωσης που προγραμματίζει το FPGA για να πραγματοποιήσει τη σχεδιασμένη λειτουργικότητα.

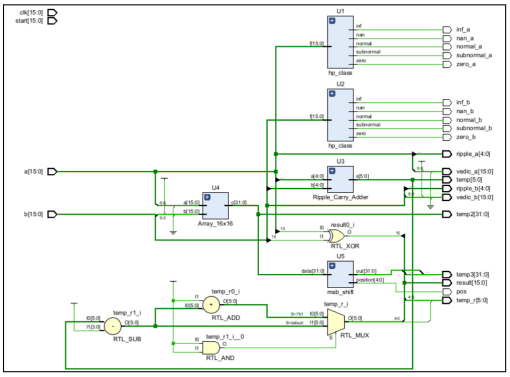


Σχήμα 6.1: Μεθοδολογία Synthesis και Implementation

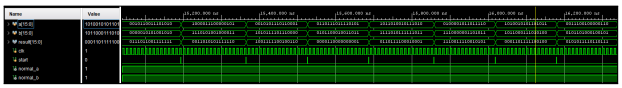
Με βάση τα παραπάνω στάδια, θα μελετήσουμε τα αποτελέσματα των προσομοιώσεων για κάθε αλγόριθμο πολλαπλασιασμού που αναφέρθηκε στο Κεφάλαιο 5 μαζί με την υλοποίησή τους στο πρότυπο IEEE-754 και το FPGA στο οποίο πραγματοποιήσαμε τις προσομοιώσεις είναι το μοντέλο Artix-7 AC701 Evaluation Platform (xc7a200tfbg676-2).

6.2 Αποτελέσματα Half Precision

- Array Multiplier



Σχήμα 6.2: Schematic

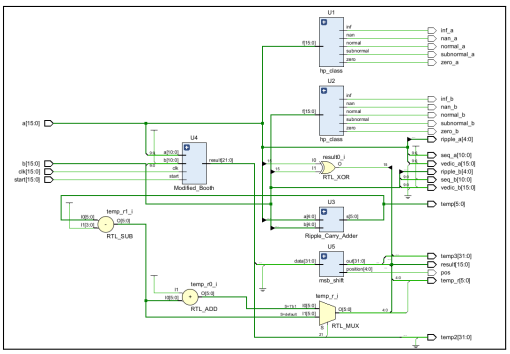


Σχήμα 6.3: Behavioral Simulation

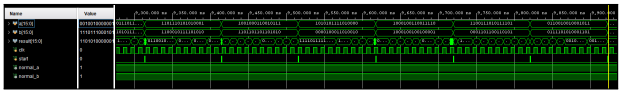
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	247	0	0	134600	0.18
LUT as Logic	247	0	0	134600	0.18
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.4: Slice Logic

- Sequential Multiplier



Σχήμα 6.5: Schematic

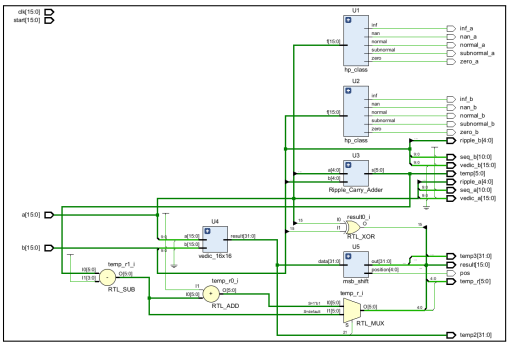


Σχήμα 6.6: Behavioral Simulation

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	296	0	0	134600	0.22
LUT as Logic	296	0	0	134600	0.22
LUT as Memory	0	0	0	46200	0.00
Slice Registers	72	0	0	269200	0.03
Register as Flip Flop	52	0	0	269200	0.02
Register as Latch	20	0	0	269200	<0.01
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.7: Slice Logic

- Vedic Multiplier



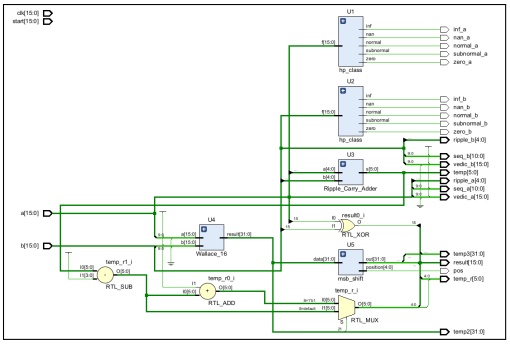
Σχήμα 6.8: Schematic

Σχήμα 6.9: Behavioral Simulation

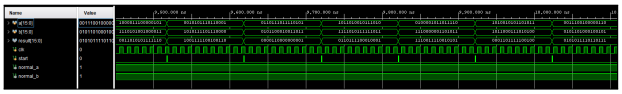
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	248	0	0	134600	0.18
LUT as Logic	248	0	0	134600	0.18
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.10: Slice Logic

- Wallace Multiplier



Σχήμα 6.11: Schematic

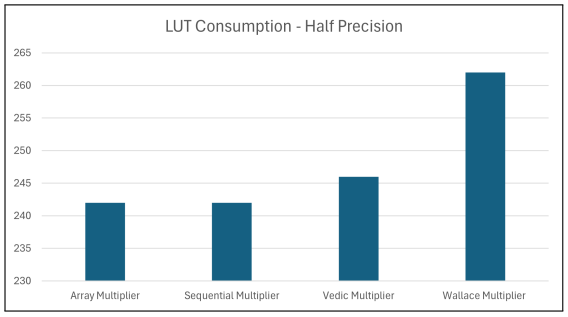


Σχήμα 6.12: Behavioral Simulation

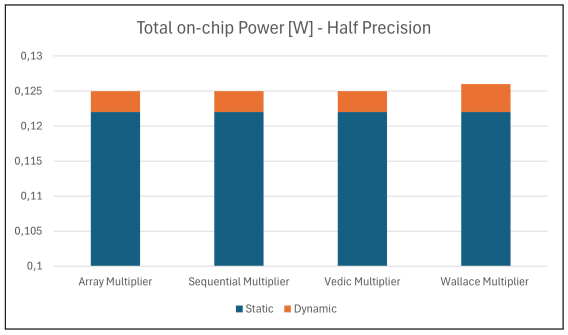
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	245	0	0	134600	0.18
LUT as Logic	245	0	0	134600	0.18
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.13: Slice Logic

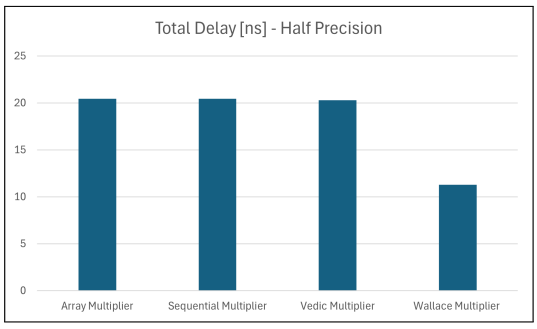
Με βάση τα αποτελέσματα που έχουμε λάβει από το synthesis και implementation του κάθε αλγορίθμου, συγκρίνουμε τις τιμές που έχουμε λάβει για τα LUTs που απαιτούνται, την ενέργεια που χρησιμοποιούν και το total delay που δημιουργούν τα σήματα στο συνολικό κύκλωμα.



Σχήμα 6.14



Σχήμα 6.15

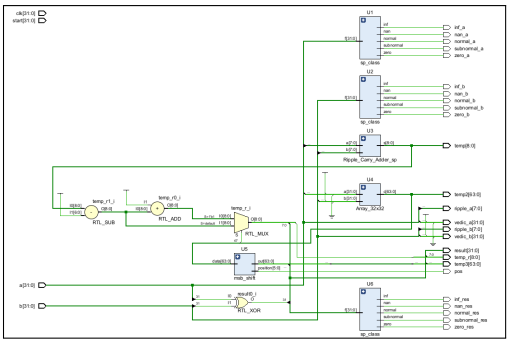


Σχήμα 6.16

Παρατηρούμε ότι για το Half-Precision εμφανίζονται παρόμοια αποτελέσματα για τους αλγορίθμους του Array, Sequential και Vedic Multiplier. Αλλά, ο Wallace Multiplier εμφανίζει μεγαλύτερες ταχύτητες εκτέλεσης της πράξης, λόγω μειωμένου delay κατά 44.6%, με κόστος την αυξημένη απαίτηση υλικού (hardware) κατά ένα μικρό ποσό (από 240 – 245 στα 262 LUTs).

6.3 Αποτελέσματα Single Precision

- Array Multiplier



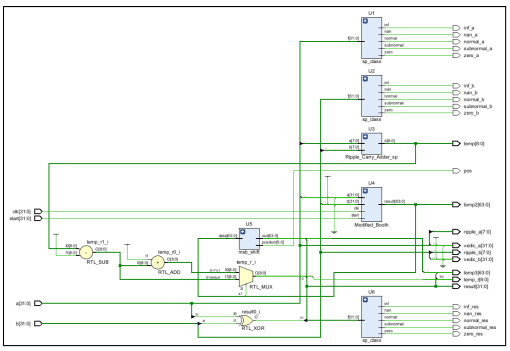
Σχήμα 6.17: Schematic

Σχήμα 6.18: Behavioral Simulation

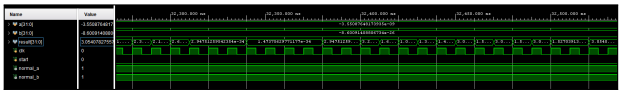
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1039	0	0	134600	0.77
LUT as Logic	1039	0	0	134600	0.77
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.19: Slice Logic

- Sequential Multiplier



Σχήμα 6.20: Schematic

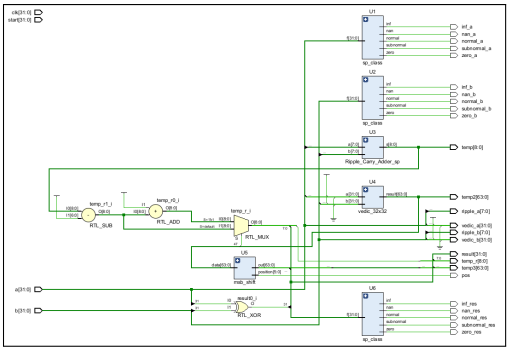


Σχήμα 6.21: Behavioral Simulation

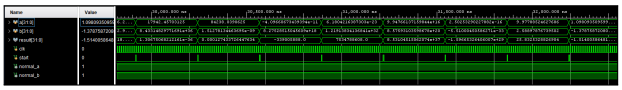
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	583	0	0	134600	0.43
LUT as Logic	583	0	0	134600	0.43
LUT as Memory	0	0	0	46200	0.00
Slice Registers	171	0	0	269200	0.06
Register as Flip Flop	125	0	0	269200	0.05
Register as Latch	46	0	0	269200	0.02
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.22: Slice Logic

- Vedic Multiplier



Σχήμα 6.23: Schematic

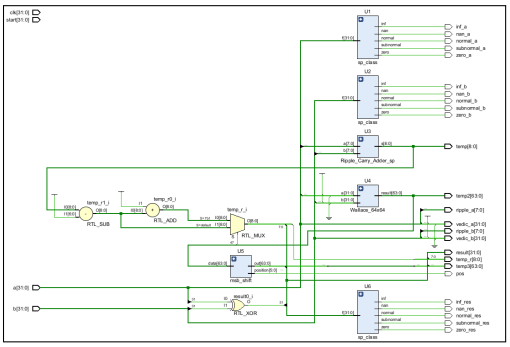


Σχήμα 6.24: Behavioral Simulation

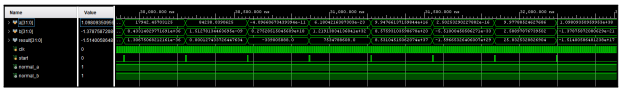
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1016	0	0	134600	0.75
LUT as Logic	1016	0	0	134600	0.75
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.25: Slice Logic

- Wallace Multiplier



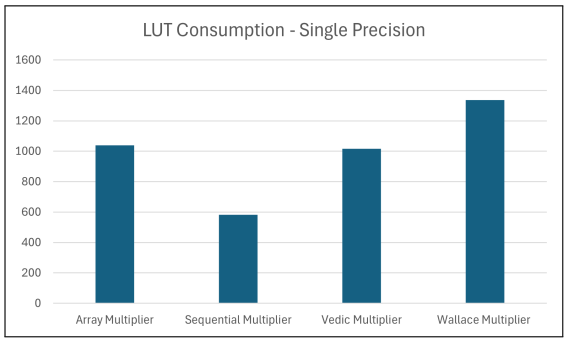
Σχήμα 6.26: Schematic



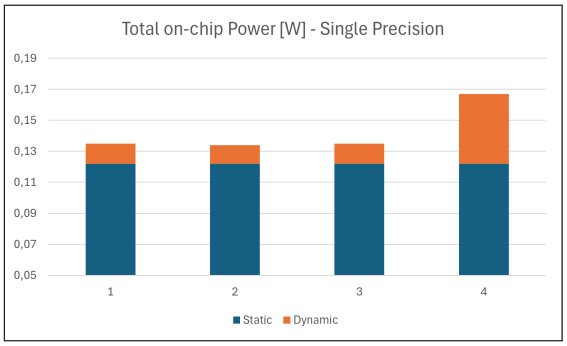
Σχήμα 6.27: Behavioral Simulation

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1337	0	0	134600	0.99
LUT as Logic	1337	0	0	134600	0.99
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

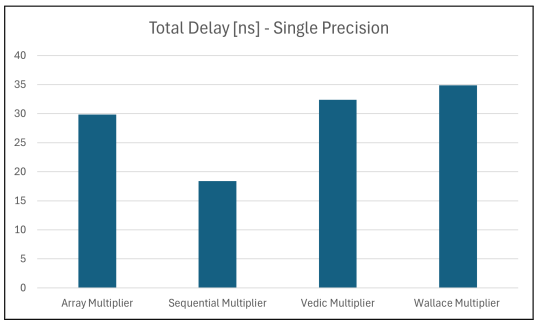
Σχήμα 6.28: Slice Logic



Σχήμα 6.29



Σχήμα 6.30

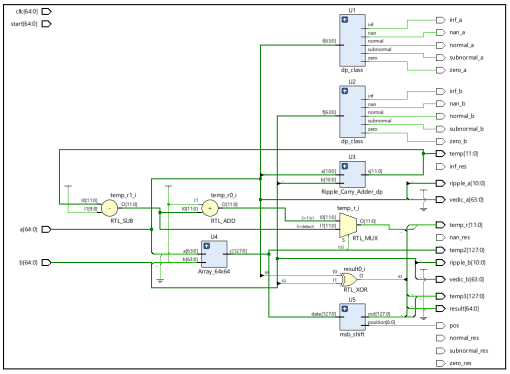


Σχήμα 6.31

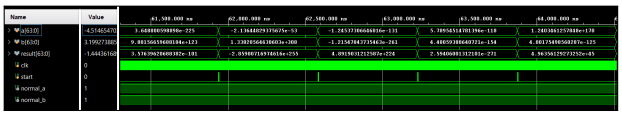
Στην περίπτωση του Single Precision, βλέπουμε ότι ο Wallace Multiplier αποτελεί την χείριστη επιλογή σε όλες τις κατηγορίες, αφού έχει την μεγαλύτερη απαίτηση σε υλικό, την μεγαλύτερη κατανάλωση και το μεγαλύτερο delay. Καλύτερη επιλογή δείχνει να είναι ο Sequential Multiplier που χρειάζεται το λιγότερο υλικό (43.88% λιγότερο συγκριτικά με τον Array Multiplier) και έχει τον μικρότερο χρόνο εκτέλεσης, αλλά το πρόβλημα αυτού του αλγορίθμου είναι ότι απαιτεί πολλούς κύκλους ρολογιού μέχρι να δώσει ορθό αποτέλεσμα. Έτσι, μια μέση λύση αποτελούν οι αλγόριθμοι του Array και Vedic Multiplier.

6.4 Αποτελέσματα Double Precision

- Array Multiplier



Σχήμα 6.32: Schematic

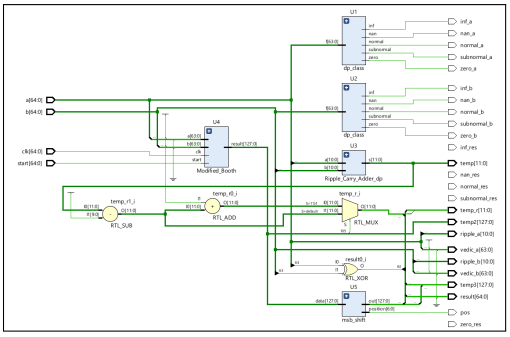


Σχήμα 6.33: Behavioral Simulation

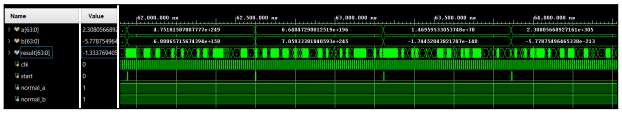
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	4727	0	0	134600	3.51
LUT as Logic	4727	0	0	134600	3.51
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	26	0	0	67300	0.04
F8 Muxes	1	0	0	33650	<0.01

Σχήμα 6.34: Slice Logic

- Sequential Multiplier



Σχήμα 6.35: Schematic

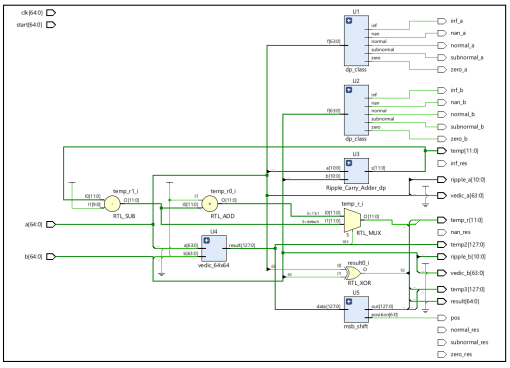


Σχήμα 6.36: Behavioral Simulation

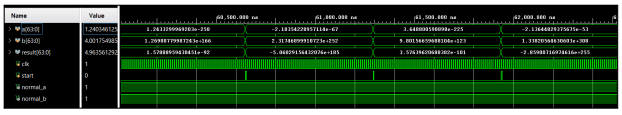
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1089	0	0	134600	0.81
LUT as Logic	1089	0	0	134600	0.81
LUT as Memory	0	0	0	46200	0.00
Slice Registers	377	0	0	269200	0.14
Register as Flip Flop	273	0	0	269200	0.10
Register as Latch	104	0	0	269200	0.04
F7 Muxes	3	0	0	67300	<0.01
F8 Muxes	0	0	0	33650	0.00

Σχήμα 6.37: Slice Logic

- Vedic Multiplier



Σχήμα 6.38: Schematic

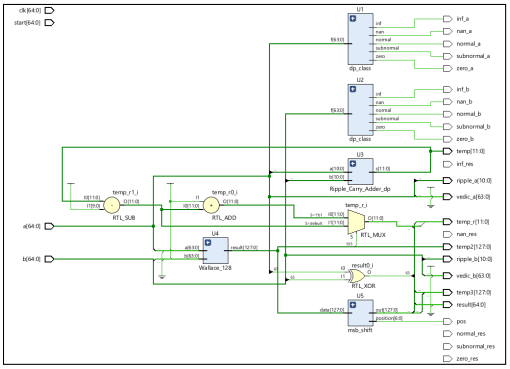


Σχήμα 6.39: Behavioral Simulation

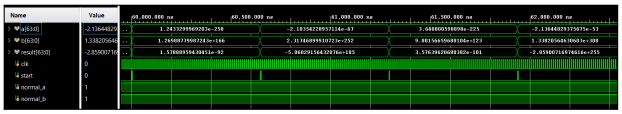
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	4857	0	0	134600	3.61
LUT as Logic	4857	0	0	134600	3.61
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	139	0	0	67300	0.21
F8 Muxes	4	0	0	33650	0.01

Σχήμα 6.40: Slice Logic

- Wallace Multiplier



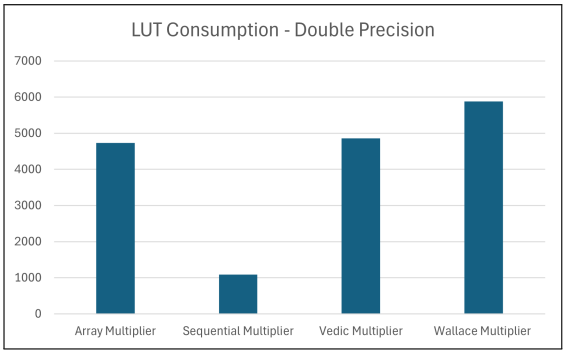
Σχήμα 6.41: Schematic



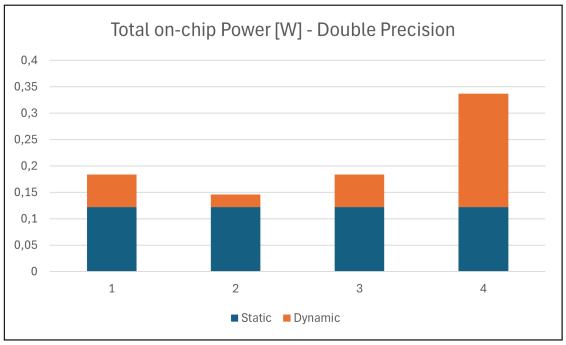
Σχήμα 6.42: Behavioral Simulation

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	5876	0	0	134600	4.37
LUT as Logic	5876	0	0	134600	4.37
LUT as Memory	0	0	0	46200	0.00
Slice Registers	0	0	0	269200	0.00
Register as Flip Flop	0	0	0	269200	0.00
Register as Latch	0	0	0	269200	0.00
F7 Muxes	1	0	0	67300	<0.01
F8 Muxes	0	0	0	33650	0.00

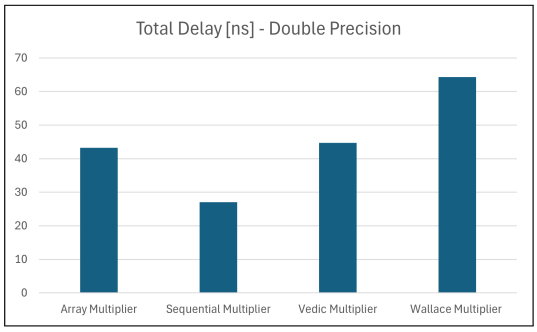
Σχήμα 6.43: Slice Logic



Σχήμα 6.44



Σχήμα 6.45



Σχήμα 6.46

Τέλος, η περίπτωση του Double Precision θυμίζει αρκετά αυτή του Single Precision, όπου ο Sequential Multiplier χρειάζεται **76.9**% λιγότερο υλικό συγκριτικά με τον αμέσως καλύτερο αλγόριθμο, ενώ ο Wallace Multiplier λόγω της αυξημένης πολυπλοκότητας του καταλήγει με τις μεγαλύτερες απαιτήσεις σε υλικό, ενέργεια και χρόνο εκτέλεσης.

Κεφάλαιο 7

Συμπεράσματα

Συμπεραίνοντας από τα αποτελέσματα που λάβαμε μέσω του Xilinx Vivado για τους 4 διαφορετικούς αλγορίθμους πολλαπλασιασμού στο πρότυπο IEEE 754, οι Array και Vedic πολλαπλασιαστές αποτελούν αξιόπιστοι και σχετικά απλοί αλγόριθμοι ως προς την υλοποίησή, με ικανοποιητική απόδοση ισχύος, υλικού και χρόνου εκτέλεσης. Όσον αφορά τον Wallace Multiplier, στο Half-Precision αποτελεί μια καλή λύση όταν δεν υπάρχει περιορισμός στο υλικό, λόγω του χαμηλού Delay που εμφανίζει, αλλά στο Single και Double Precision δεν μπορεί να πετύχει εξίσου καλές ταχύτητές εκτέλεσης της πράξης συγκριτικά με τους υπόλοιπους αλγορίθμους, ενώ παράλληλα δεν σημειώνει μειωμένη κατανάλωση ενέργειας ή και υλικού. Ο Sequential Multiplier είναι μια λύση όταν οι διαθέσιμοι πόροι σε υλικό είναι χαμηλοί (για όλες τις μορφές του προτύπου IEEE 754), ωστόσο ο περιορισμός του βασίζεται στο γεγονός ότι ορθό αποτέλεσμα στην πράξη εμφανίζεται μετά από κάποιους κύκλους ρολογιού (ανάλογο με τον αριθμό σε bits του πολλαπλασιαστή και πολλαπλασιαστέου).

Κλείνοντας, για την καλύτερη αντιμετώπιση των αλγορίθμων, δύναται να εφαρμοστούν μελλοντικές αλλαγές που θα εστιάζουν στην επιλογή πολλαπλών και διαφορετικών μεθόδων πρόσθεσης του εκθέτη. Με αυτόν τον τρόπο, θα παρέχεται η δυνατότητα επιλογής του κατάλληλου τρόπου πρόσθεσης σε κάθε αλγόριθμο πολλαπλασιασμού, με σκοπό την βελτιστοποίηση του χρόνου εκτέλεσης και κατανάλωσης σε υλικό και ενέργεια.

Παράρτημα Α΄

Ακρωνύμια και συντομογραφίες

CLB Configurable Logic Block

CPU Central Processing Unit

DSP Digital Signal Processing

FA Full Adder

FF Flip-Flop

HA Half Adder

HDL Hardware Description Language

IEEE Institute of Electrical and Electronics Engineers

LSB Least Significant Bit

LUT Look-Up Table

MSB Most Significant Bit

NaN Not a Number

RCA Ripple Carry Adder

RTL Register-Transfer Level

SoC System-on-Chip

UUT Unit Under Test

FPGA Field Programmable Gate Array

IC Integrated circuit

VHDL VHSIC Hardware Description Language

Παράρτημα Β΄

Λογικές Πύλες και CLB

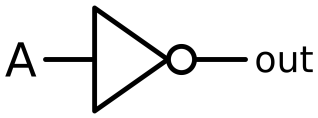
Β΄.1 Λογικές Πύλες

Μία λογική πύλη είναι ηλεκτρονικό κύκλωμα το οποίο πραγματοποιεί μία λογική πράξη στις εισόδους της και παράγει μία έξοδο. Οι λογικές πύλες έχουν δημιουργηθεί για να δουλεύουν στο δυαδικό σύστημα. Στα ηλεκτρονικά κυκλώματα ως λογικό 0 θεωρείται η τάση εκείνη η οποία είναι κάτω από ένα κατώφλι που έχουν ορίσει οι κατασκευαστές της λογικής πύλης (π.χ. $0,5[V]$). Αντίστοιχα το λογικό 1 αντιστοιχεί σε τάση η οποία υπερβαίνει κάποια τάση . Με άλλα λόγια το λογικό 0 αντιστοιχεί στην τάση γείωσης και το λογικό 1 σε τάση τροφοδοσίας.

Β΄.1.1 Είδη Λογικών Πυλών

- **Πύλη NOT**: Η πύλη NOT έχει μόνο μία είσοδο και δίνει μόνο μία έξοδο. Η λειτουργία της είναι η αντιστροφή του λογικού σήματος της εισόδου.

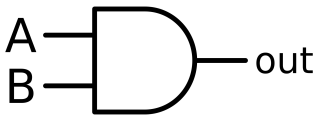
Input	Output
0	1
1	0



Σχήμα Β΄.1: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης NOT

- **Πύλη AND**: Η πύλη AND βγάζει στην έξοδο 1 αν και μόνο αν και οι δύο είσοδοι είναι 1. Διαφορετικά, η έξοδος είναι 0.

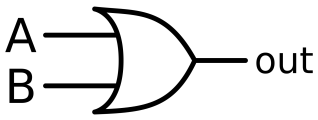
Input		Output
A	B	$A * B$
0	0	0
0	1	0
1	0	0
1	1	1



Σχήμα Β΄.2: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης AND

- **Πύλη OR**: Η πύλη OR βγάζει 1 εάν μία ή και οι δύο είσοδοι είναι 1. Διαφορετικά, η έξοδος είναι 0.

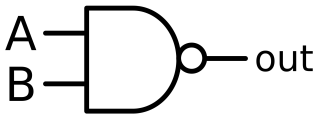
Input		Output
A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1



Σχήμα Β΄.3: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης OR

- **Πύλη NAND**: Η πύλη NAND είναι το ισοδύναμο μιας πύλης AND που ακολουθείται από μια πύλη NOT. Βγάζει 0 αν και μόνο αν και οι δύο είσοδοι είναι 1. Διαφορετικά, η έξοδος είναι 1.

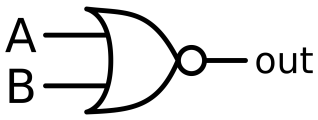
Input		Output
A	B	$(A * B)'$
0	0	1
0	1	1
1	0	1
1	1	0



Σχήμα Β΄.4: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης NAND

- **Πύλη NOR**: Η πύλη NOR είναι το ισοδύναμο μιας πύλης OR που ακολουθείται από μια πύλη NOT. Βγάζει 1 αν και μόνο αν και οι δύο είσοδοι είναι 0. Διαφορετικά, η έξοδος είναι 0.

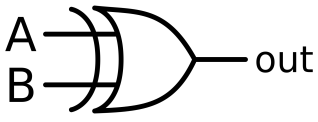
Input		Output
A	B	$(A + B)'$
0	0	1
0	1	0
1	0	0
1	1	0



Σχήμα Β΄.5: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης NOR

- **Πύλη XOR**: Η πύλη XOR βγάζει 1 εάν και μόνο εάν οι είσοδοι είναι διαφορετικές. Εάν οι είσοδοι είναι ίδιες, η έξοδος είναι 0.

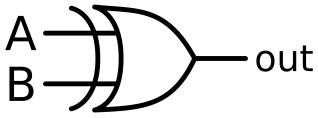
Input		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



Σχήμα Β΄.6: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης XOR

- **Πύλη XNOR**: Η πύλη XNOR είναι το ισοδύναμο μιας πύλης XOR που ακολουθείται από μια πύλη NOT. Βγάζει 1 αν και μόνο αν οι είσοδοι είναι ίδιες. Εάν οι είσοδοι είναι διαφορετικές, η έξοδος είναι 0.

Input		Output
A	B	$(A \oplus B)'$
0	0	1
0	1	0
1	0	0
1	1	1



Σχήμα Β΄.7: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα Πύλης XOR

Β΄.2 CLB - Configurable Logic Block

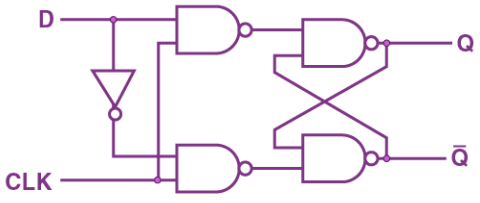
Στον τρόπο λειτουργίας ενός FPGA, το βασικό δομικό στοιχείο που είναι υπεύθυνο για την εφαρμογή της ψηφιακής λογικής είναι το Configurable Logic Block (CLB). Αυτά τα ευέλικτα εξαρτήματα χρησιμεύουν ως οι βάσεις εργασίας των FPGA, επιτρέποντας στους μηχανικούς να σχεδιάσουν μια μεγάλη ποικιλία κυκλωμάτων διαμορφώνοντάς τα ώστε να εκτελούν συγκεκριμένες λειτουργίες.

Β΄.2.1 Εμβαθύνοντας στις εσωτερικές λειτουργίες ενός CLB

Ένα CLB δεν είναι μια ξεχωριστή μονάδα, αλλά μια συλλογή από μικρότερα, διαμορφώσιμα στοιχεία που συνεργάζονται για την εκτέλεση πολύπλοκων λογικών πράξεων. Τα βασικά στοιχεία που βρίσκονται συνήθως σε ένα CLB περιλαμβάνουν:

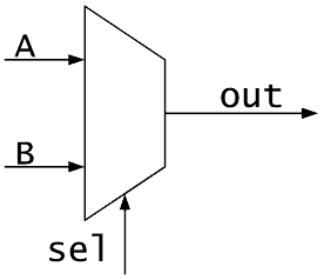
- **Look-up Tables (LUT)**: Αυτά τα ευέλικτα στοιχεία αποτελούν τον πυρήνα ενός CLB. Τα LUT μπορούν να προγραμματιστούν ώστε να λειτουργούν ως οποιαδήποτε αυθαίρετη λογική συνάρτηση με περιορισμένο αριθμό εισόδων. Λειτουργούν ουσιαστικά ως ένας προκαθορισμένος πίνακας αλήθειας, επιτρέποντας στο CLB να υλοποιήσει οποιονδήποτε συνδυασμό λογικών πυλών (AND, OR, XOR, κλπ.) με βάση την προγραμματισμένη διαμόρφωση.
- **Flip-Flops:** Αυτά τα στοιχεία μνήμης είναι ζωτικής σημασίας για την υλοποίηση διαδοχικών λογικών κυκλωμάτων (Sequential Logic Circuits). Μπορούν να αποθηκεύσουν ένα μόνο bit δεδομένων και να ενημερώσουν την τιμή του με βάση το σήμα ρολογιού και τα εισερχόμενα δεδομένα. Αυτό επιτρέπει στο CLB να δημιουργεί κυκλώματα με μνήμη, όπως μετρητές, καταχωρητές και μηχανές κατάστασης (State Machines).

Q	D	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1



Σχήμα Β΄.8: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα D Flip-Flop

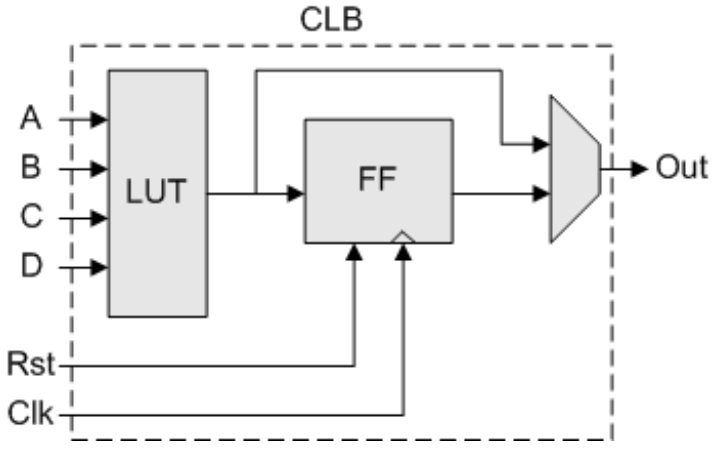
- **Multiplexers - Πολυπλέκτες**: Αυτοί οι προγραμματιζόμενοι διακόπτες επιτρέπουν στο CLB να επιλέξει την επιθυμητή έξοδο με βάση ένα σήμα ελέγχου. Λειτουργούν ως ένας ευέλικτος μηχανισμός δρομολόγησης εντός του CLB, επιτρέποντας την επιλογή δεδομένων από διάφορες πηγές εντός του μπλοκ.



Σχήμα Β΄.9: Κυκλωματικό Σχεδιάγραμμα Πολυπλέκτη

Ενώ τα CLB είναι οι θεμελιώδεις λογικές μονάδες, δεν λειτουργούν μεμονωμένα. Μια κρίσιμη πτυχή του σχεδιασμού σε FPGA είναι η διασύνδεση μεταξύ των CLB. Οι προγραμματιζόμενοι πόροι δρομολόγησης εντός του FPGA επιτρέπουν τη σύνδεση διαφόρων CLB για να σχηματιστούν πιο πολύπλοκα κυκλώματα. Συνδέοντας στρατηγικά τις εξόδους ενός CLB με τις εισόδους ενός άλλου, μπορούν να δημιουργηθούν περίπλοκες λογικές δομές που πληρούν την επιθυμητή λειτουργικότητα.

Συμπερασματικά, τα CLB αποτελούν την ραχοκοκαλιά των FPGA, προσφέροντας μια ευέλικτη δομική μονάδα για την εφαρμογή της ψηφιακής λογικής. Η εσωτερική τους σύνθεση από LUT, Flip-Flops και πολυπλέκτες, σε συνδυασμό με την προγραμματιζόμενη φύση τους, επιτρέπει την δημιουργία μιας μεγάλης ποικιλίας κυκλωμάτων.

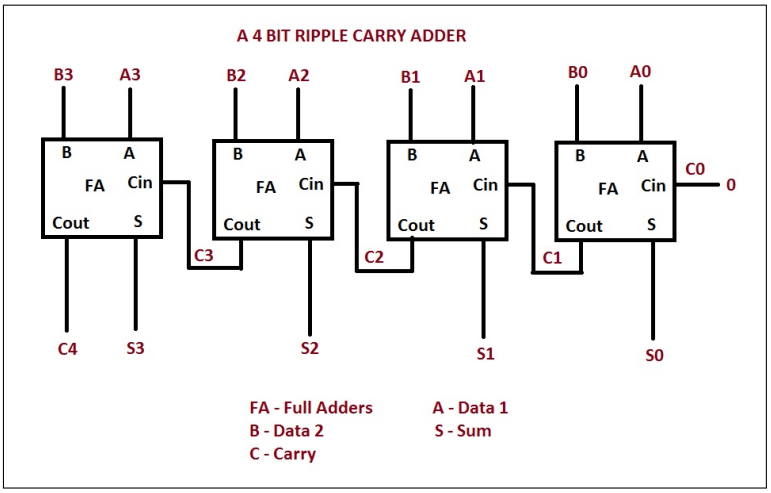


Σχήμα Β΄.10: Κυκλωματική Δομή ενός CLB

Παράρτημα Γ΄

Ripple Carry Adder

Ένας αθροιστής Ripple Carry Adder (RCA) είναι ένα ψηφιακό λογικό κύκλωμα που εκτελεί την πρόσθεση δύο δυαδικών αριθμών. Αποτελείται από έναν καταρράκτη πλήρων αθροιστών (Full Adders), όπου το κρατούμενο της εξόδου κάθε πλήρους αθροιστή τροφοδοτείται ως είσοδος στον επόμενο πλήρη αθροιστή. Αυτή η διάταξη ονομάζεται μεταφορά κυματισμού επειδή το σήμα μεταφοράς πρέπει να κυματίζει σε κάθε στάδιο πλήρους αθροιστή πριν φτάσει στο τελικό στάδιο.



Σχήμα Γ΄.1: Block diagram ενός Ripple Carry Adder

Για να ερευνήσουμε όμως την λειτουργία του Ripple Carry Adder αθροιστή, πρέπει πρώτα να μελετήσουμε την κατασκευή ενός πλήρους αθροιστή.

Γ΄.1 Ημιαθροιστής

Ο ημιαθροιστής είναι ένα βασικό δομικό στοιχείο στα ψηφιακά ηλεκτρονικά. Είναι ένα συνδυαστικό λογικό κύκλωμα που εκτελεί την πρόσθεση δύο δυαδικών αριθμών ενός bit. Ο ημιαθροιστής έχει δύο εισόδους και δύο εξόδους. Οι είσοδοι είναι τα δύο δυαδικά ψηφία που προστίθενται και οι έξοδοι είναι το άθροισμα (Sum) και το κρατούμενο (Carry).

Η έξοδος του αθροίσματος είναι το αποτέλεσμα της προσθήκης των δύο δυαδικών ψηφίων. Εάν τα δύο ψηφία είναι και τα δύο 0, το άθροισμα είναι 0. Εάν ένα από τα δύο ψηφία είναι 1, το άθροισμα είναι 1. Αν και τα δύο ψηφία είναι 1, τότε το άθροισμα είναι ίσο με 0.

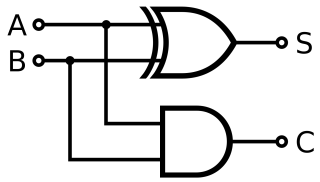
Η έξοδος του κρατούμενου είναι ένα σήμα που υποδεικνύει εάν υπάρχει κρατούμενο από την προσθήκη των δύο δυαδικών ψηφίων. Εάν τα δύο ψηφία είναι και τα δύο 0 ή μόνο το ένα από τα δύο ψηφία είναι 1, δεν υπάρχει κρατούμενο και η έξοδος του είναι 0. Μόνο στην περίπτωση που και τα δύο ψηφία εισόδου είναι ίσα με 1, η έξοδος του κρατούμενου γίνεται και αυτή ίση με 1.

Ο ημιαθροιστής αποτελείται από δύο λογικές πύλες:

- Μια πύλη XOR χρησιμοποιείται για τον υπολογισμό του αθροίσματος εξόδου. Η πύλη XOR παίρνει δύο εισόδους και παράγει μια έξοδο που είναι 1 εάν και μόνο εάν ακριβώς μία από τις εισόδους είναι 1.
- Μια πύλη AND χρησιμοποιείται για τον υπολογισμό του κρατούμενου. Η πύλη AND λαμβάνει δύο εισόδους και παράγει μια έξοδο που είναι 1 εάν και μόνο εάν και οι δύο είσοδοι είναι 1.

Παρακάτω φαίνονται ο πίνακας αληθείας ενός ημιαθροιστή και η σχηματική αναπαράστασή του.

A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Σχήμα Γ΄.2: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα ενός Ημιαθροιστή

Οι ημιαροιστές μπορούν να χρησιμοποιηθούν για την κατασκευή ενός πλήρους αθροιστή.

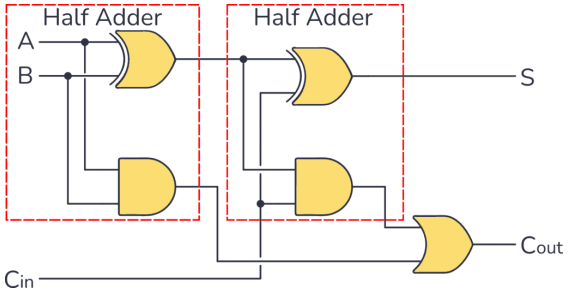
Γ΄.2 Πλήρης Αθροιστής

Ένας πλήρης αθροιστής είναι ένα συνδυαστικό λογικό κύκλωμα που εκτελεί την προσθήκη δύο δυαδικών αριθμών ενός bit και ενός κρατούμενου (C_{in}). Παράγει ένα άθροισμα (Sum) και μια έξοδο (C_{out}). Ο πλήρης αθροιστής είναι ένα από τα θεμελιώδη δομικά στοιχεία της ψηφιακής ηλεκτρονικής και χρησιμοποιείται σε πολλά αριθμητικά κυκλώματα, συμπεριλαμβανομένων των αθροιστών, των αφαιρετών και των πολλαπλασιαστών.

Ένας πλήρης αθροιστής μπορεί να κατασκευαστεί χρησιμοποιώντας δύο ημιαθροιστές και μία πύλη OR. Ο πρώτος ημιαθροιστής χρησιμοποιείται για την προσθήκη των δύο δυαδικών ψηφίων εισόδου A και B και ο δεύτερος ημιαθροιστής χρησιμοποιείται για την προσθήκη της εξόδου του πρώτου ημιαθροιστή και του ψηφίου κρατούμενου C_{in}. Η έξοδος του πρώτου ημιαθροιστή είναι το bit αθροίσματος (Sum), και η έξοδος του δεύτερου ημιαθροιστή είναι το bit κρατούμενου C_{out}.

Παρακάτω φαίνονται ο πίνακας αληθείας και η σχηματική αναπαραστασή του πλήρη αθροιστή.

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



Σχήμα Γ΄.3: Πίνακας αληθείας και Κυκλωματικό Σχεδιάγραμμα ενός Πλήρη Αθροιστή

Γ΄.3 Πλεονεκτήματα/Μειονεκτήματα Ripple Carry Adder

Ο RCA είναι ένας απλός και εύκολος στην εφαρμογή αθροιστής, αλλά έχει το μειονέκτημα ότι είναι αργός. Αυτό συμβαίνει επειδή κάθε στάδιο πλήρους αθροιστή έχει τη δική του καθυστέρηση διάδοσης, που είναι ο χρόνος που χρειάζεται για να δημιουργηθεί η έξοδος του κρατούμενου. Ως αποτέλεσμα, ο RCA έχει υψηλότερο λανθάνοντα χρόνο από άλλους τύπους αθροιστών, όπως ο αθροιστής Carry-Lookahead Adder.

Ωστόσο, ο RCA εξακολουθεί να χρησιμοποιείται ευρέως σε ψηφιακά κυκλώματα επειδή είναι φθηνό και εύκολο στη σχεδίασή του. Χρησιμοποιείται επίσης συχνά σε κυκλώματα συνδυαστικής λογικής, όπου η καθυστέρηση του αθροιστή δεν είναι κρίσιμη, όπως στην δική μας περίπτωση που ο πολλαπλασιασμός των κλασματικών μερών σε επίπεδο bit αφορά μεγαλύτερο αριθμό bit και γενικότερα είναι μια πολύ πιο χρονοβόρα διαδικασία από την πρόσθεση.

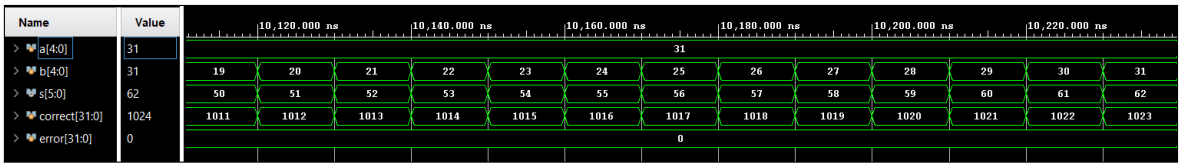
Ακολουθεί ένας πίνακας με τα πλεονεκτήματα και τα μειονεκτήματα του RCA:

Advantage	**Disadvantage**
Easy to implement	Slow
Inexpensive	Large latency
Can be used for any number of bits	Not efficient for large numbers

Πίνακας Γ΄.1: Πλεονεκτήματα και Μειονεκτήματα του RCA

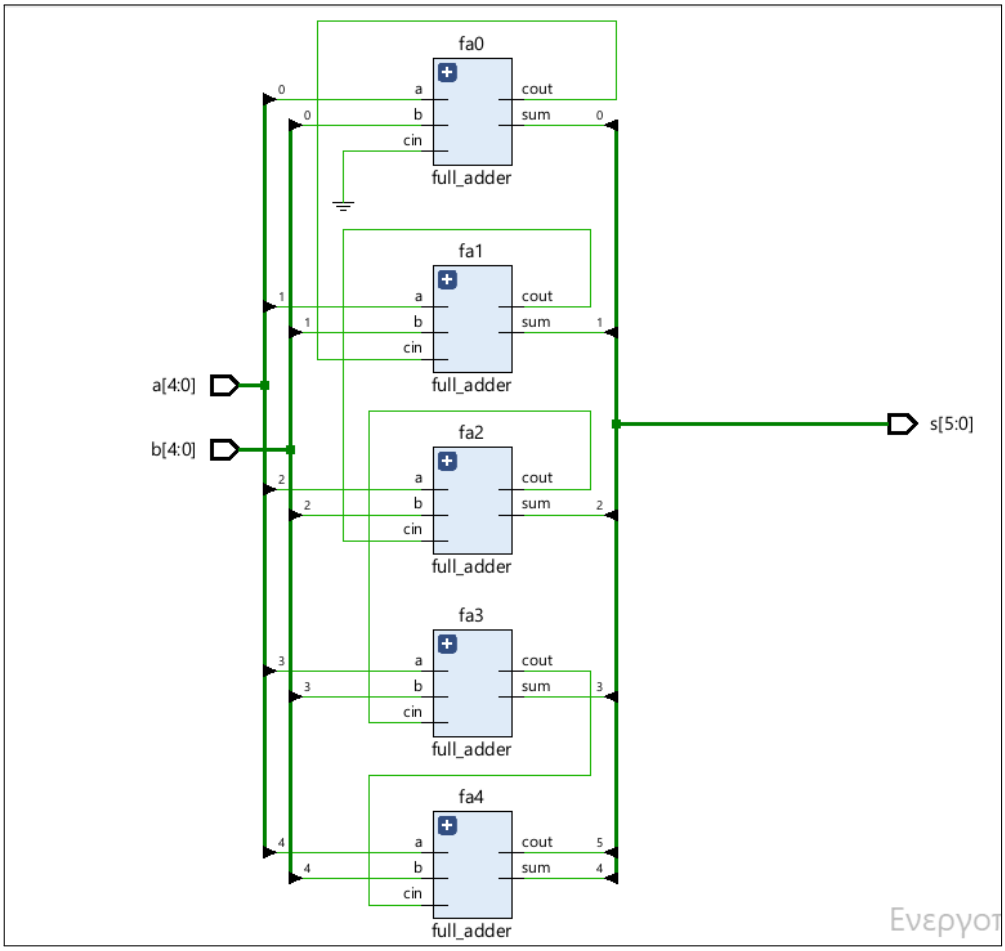
Γ΄.4 Κατασκευή του RCA

Για αρχή, κατασκευάζουμε έναν 5 bit RCA για την περίπτωση που ο μορφή του προτύπου IEEE 754 είναι Half Precision (16 bit). Τα αποτελέσματα του simulation τα βλέπουμε στο Σχήμα Γ.2, όπου εξετάσαμε όλες τις πιθανές περιπτώσεις και προέκυψε ότι το κύκλωμα μας δεν εμφανίζει σφάλματα (correct: 1024, error: 0).

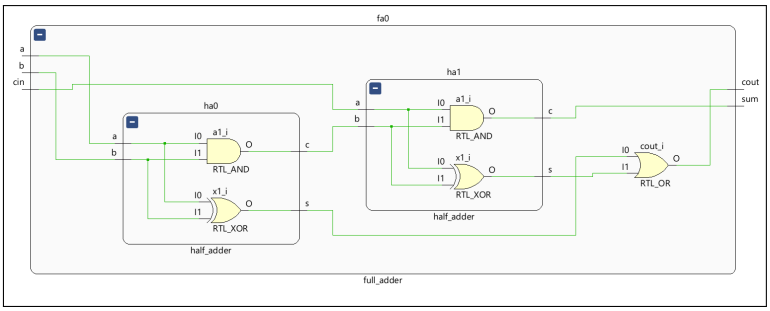


Σχήμα Γ΄.4: Testbench Waveform - 5 bit Ripple Carry Adder

Παρακάτω βλέπουμε το σχηματικό από το RTL Analysis για ολόκληρο τον αθροιστή (Σχήμα Γ.3), αλλά και για τον κάθε πλήρη αθροιστή ξεχωριστά (Σχήμα Γ.4).

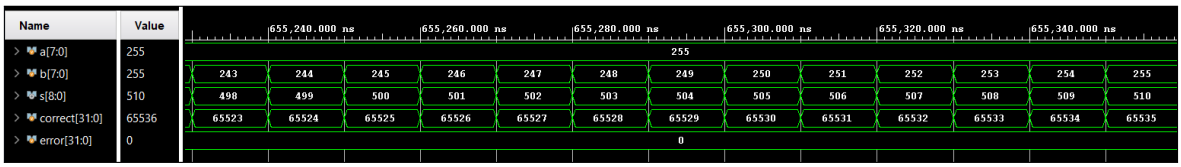


Σχήμα Γ΄.5: Schematic ενός 5 bit Ripple Carry Adder

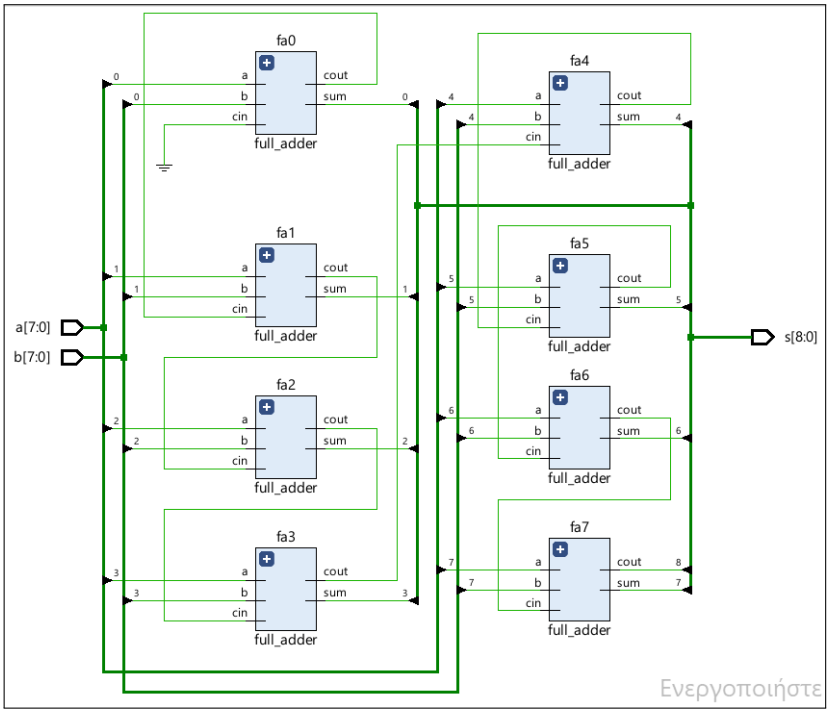


Σχήμα Γ΄.6: Schematic ενός Full Adder (Κατασκευασμένος από 2 Half Adders)

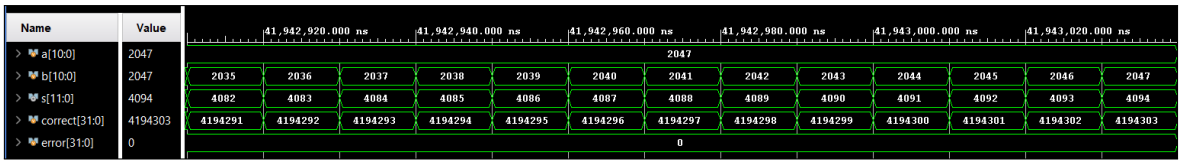
Στην συνέχεια, εφαρμόζουμε την ίδια διαδικασία για τους αθροιστές που θα χρειαστούμε και για τις υπόλοιπες μορφές του προτύπου IEEE 754 (Signle Precision - 8 bits, Double Precision - 11 bits).



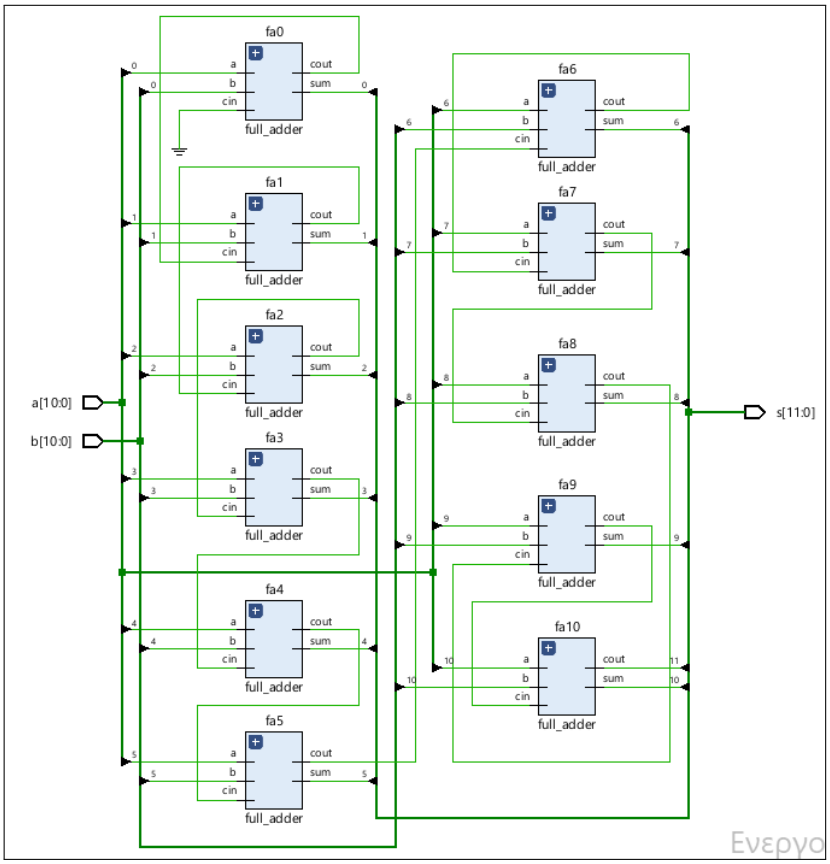
Σχήμα Γ΄.7: Schematic ενός 8 bit Ripple Carry Adder



Σχήμα Γ΄.8: Testbench Waveform - 8 bit Ripple Carry Adder



Σχήμα Γ.9: Testbench Waveform - 11 bit Ripple Carry Adder



Σχήμα Γ.10: Schematic ενός 11 bit Ripple Carry Adder

Bibliography

[1] "Artificial intelligence (ai) market - global industry analysis, size, share, growth, trends, regional outlook, and forecast 2023-2032." [Online]. Available: https://www.precedenceresearch.com/artificial-intelligence-market

[2] R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*. Prentice Hall, 2011.

[3] M. Ercegovac and T. Lang, *Digital Arithmetic*, ser. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2004. [Online]. Available: https://books.google.gr/books?id=p79cu3nZ6yoC

[4] K. M. Yong, R. Hussin, A. Kamarudin, R. C. Ismail, M. N. M. Isa, and S. Z. M. Naziri, "Design and analysis of 32-bit signed and unsigned multiplier using booth, vedic and wallace architecture," *Journal of Physics: Conference Series*, feb 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1755/1/012008

[5] K. Kumar, V. Tyagi, H. Kukreja, S. Thakral, and M. Verma, "A state-ofthe-art study on multipliers : Advancement and comparison," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:201679456

[6] P. Soni, S. Kadam, H. Dhurape, and N. Gulavani, "Implementation of 16x16 bit multiplication algorithm by using vedic mathematics over booth algorithm," *IJRET: International Journal of Research in Engineering and Technology SSN*, pp. 2319–1163, 2015.

[7] R. K. Kodali, L. Boppana, and S. S. Yenamachintala, "Fpga implementation of vedic floating point multiplier," in *2015 IEEE international conference on signal processing, informatics, communication and energy systems (SPICES)*. IEEE, 2015, pp. 1–4.

[8] K. F. Pang, "Architectures for pipelined wallace tree multiplier-accumulators," in *Proceedings., 1990 IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 1990, pp. 247–250.

[9] C. Y. Lee, L. H. Hiung, S. W. Lee, and N. H. Hamid, "A performance comparison study on multiplier designs," in *2010 International Conference on Intelligent and Advanced Systems*. IEEE, 2010, pp. 1–6.

[10] S. Waser, "High-speed monolithic multipliers for real-time digital signal processing," *Computer*, vol. 11, no. 10, pp. 19–29, 1978.

[11] Z. Huang and M. D. Ercegovac, "High-performance left-to-right array multiplier design," in *Proceedings 2003 16th IEEE Symposium on Computer Arithmetic*. IEEE, 2003, pp. 4–11.

[12] N. Strader and V. Rhyne, "A canonical bit-sequential multiplier," *IEEE Transactions on Computers*, vol. C-31, no. 8, pp. 791–795, 1982.

[13] K. Pichhode, M. D. Patil, D. Shah, and B. C. Rohit, "Fpga implementation of efficient vedic multiplier," in *2015 International Conference on Information Processing (ICIP)*. IEEE, 2015, pp. 565–570.

[14] S. Asif and Y. Kong, "Low-area wallace multiplier," *Vlsi Design*, vol. 2014, pp. 1–1, 2014.

[15] S. Asif and Kong, "Design of an algorithmic wallace multiplier using high speed counters," 2015, pp. 133–138.