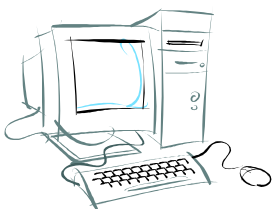
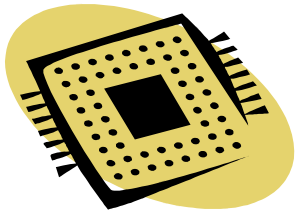
**Τ.Ε.Ι. ΚΡΗΤΗΣ**
**Σχολή Εφαρμοσμένων Επιστημών**
**Τμήμα Ηλεκτρονικών Μηχανικών**





# Μελέτη και σχεδίαση αριθμητικής και λογικής μονάδας, στα 32 μπιτ, συμπεριλαμβάνοντας πράξεις κινητής υποδιαστολής απλής ακρίβειας.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

**Ντογκρασβίλι Λεωνίδα**

Επιβλέπων: **Δρ. Μηχ. Νικόλαος Σ. Πετράκης**
**Καθηγητής Εφαρμογών**

**Χανιά,**
**Φεβρουάριος 2015**

**ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ**
Σχολή Εφαρμοσμένων Επιστημών
Τμήμα Ηλεκτρονικών Μηχανικών

# ΠΑΡΟΥΣΙΑΣΗ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

*με θέμα:*

## Μελέτη και σχεδίαση αριθμητικής και λογικής μονάδας, στα 32 μπιτ, συμπεριλαμβάνοντας πράξεις κινητής υποδιαστολής απλής ακρίβειας.

*του*

**Ντογκρασβίλι Λεωνίδα**

**Παρασκευή 13 Φεβρουαρίου 2015, ώρα 11:00, Αίθουσα 4**

*Εξεταστική Επιτροπή:* Καθ. Εφαρμ. Νικόλαος Πετράκης (επιβλέπων)
Επίκ. Καθ. Αντώνιος Κωνσταντάρας
Εργ. Συνεργάτης Ηρακλής Ρηγάκης

### Εν' συντομία

Στόχος της παρούσας πτυχιακής εργασίας ήταν η κατανόηση σε βάθος της λειτουργίας της αριθμητικής και λογικής μονάδας (ALU) ενός επεξεργαστή καθώς και της γλώσσας περιγραφής υλικού VHDL. Στα πλαίσια της εργασίας αυτής σχεδιάστηκαν κυκλώματα τα οποία μπορούν να πραγματοποιούν στοιχειώδεις πράξεις ακεραίων στα 32 μπιτ (όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση), καθώς και κυκλώματα τα οποία μπορούν να πραγματοποιούν πράξεις με πραγματικούς αριθμούς εκφρασμένους σε κινητή υποδιαστολή, απλής ακρίβειας, σύμφωνα με το πρότυπο IEEE 754. Υλοποιήθηκαν σε VHDL τα σχεδιασμένα κυκλώματα και επαληθεύτηκε η λειτουργία τους μέσω προσομοίωσης στο ολοκληρωμένο περιβάλλον ISE της Xilinx.

# Πίνακας Περιεχομένων

## Περίληψη

Σκοπός αυτής της πτυχιακής εργασίας είναι η κατανόηση σε βάθος τόσο της γλώσσας περιγραφής υλικού VHDL όσο και της λειτουργίας της αριθμητικής και λογικής μονάδας (ALU) ενός επεξεργαστή. Σχεδιάστηκαν κυκλώματα τα οποία μπορούν να πραγματοποιήσουν τις στοιχειώδεις πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση) μεταξύ ακεραίων στα 32 μπιτ, καθώς και κυκλώματα τα οποία μπορούν να πραγματοποιήσουν τις πράξεις (πρόσθεση, αφαίρεση και πολλαπλασιασμό) μεταξύ πραγματικών αριθμών εκφρασμένων σε κινητή υποδιαστολή, απλής ακρίβειας, σύμφωνα με το πρότυπο IEEE 754. Υλοποιήθηκαν σε VHDL τα σχεδιασμένα κυκλώματα και επαληθεύτηκε η καλή λειτουργία τους μέσω προσομοίωσης στο περιβάλλον ISE της Xilinx. Κάποια κυκλώματα υλοποιήθηκαν χρησιμοποιώντας διαφορετικές τεχνικές οπότε δόθηκε η ευκαιρία να γίνουν συγκρίσεις στις επιδόσεις των.

## Abstract

The purpose of this thesis is to deepen the understanding of both the hardware description language VHDL and the functioning of arithmetic and logic unit (ALU) of a processor. Were designed circuits which can perform the elementary operations (addition, subtraction, multiplication and division) between integers in 32 bits and circuits which can perform the operations (addition, subtraction and multiplication) between real numbers expressed in single-precision floating point, according to the standard IEEE 754. The designed circuits were implemented in VHDL and verified that they operate conforming the initial specifications through simulation using Xilinx ISE. Some circuits have been implemented using different techniques giving the opportunity to make comparisons in performance.

# 1. Εισαγωγή

Η παρούσα πτυχιακή εργασία έχει να κάνει με την μελέτη της αριθμητικής και λογικής μονάδας ενός επεξεργαστή στα 32 μπιτ. Ο κύριος λόγος που επέλεξα να κάνω την πτυχιακή αυτή, είναι γιατί ήθελα να διευρύνω τις γνώσεις μου όσο αφορά τα ψηφιακά κυκλώματα, να μάθω και να καταλάβω την εσωτερική δομή ενός επεξεργαστή και να καταλάβω αρκετά καλά το πώς γίνονται και το πώς υλοποιούνται οι βασικότερες πράξεις των μαθηματικών στο δυαδικό. Επίσης άλλος ένας λόγος ο οποίος με ώθησε στην επιλογή της παρούσας πτυχιακής είναι γιατί ήθελα να μάθω και μια διαφορετική γλώσσα προγραμματισμού χαμηλού επιπέδου από την assembly του MPIS και την assembly του 8051. Κατά συνέπεια λοιπόν επέλεξα να μάθω την γλώσσα περιγραφής υλικού γιατί αυτή μου έδινε την δυνατότητα να υλοποιήσω το κομμάτι της ALU, και γι' αυτό στην εργασία αυτή, θα γίνει ανάλυση και υλοποίηση της μονάδας με την χρήση της γλώσσας περιγραφής υλικού VHDL.

Η εργασία είναι δομημένη σε έξι κεφάλαια. Εκτός από το παρόν κεφάλαιο που αποτελεί μια σύντομη εισαγωγή στα περιεχόμενα της εργασίας ανά κεφάλαιο, στο κεφάλαιο 2 παρουσιάζουμε συνοπτικά την θεωρία των πράξεων με τους ακέραιους αριθμούς, συγκεκριμένα έχουμε κάνει μια μελέτη πάνω στα συστήματα αρίθμησης, το δυαδικό σύστημα και την μετατροπή δυαδικών αριθμών σε δεκαδικό και αντίστροφα. Στο ίδιο κεφάλαιο παρουσιάζονται όλοι οι δυνατοί τρόποι απεικόνισης αρνητικών ακέραιων αριθμών (συμπλήρωμα ως προς ένα, συμπλήρωμα ως προς δύο, προσημασμένου μέτρου, πολωμένη σημειογραφία), μελετάμε τους αθροιστές (ημιαθροιστής, πλήρης αθροιστής, αθροιστής ριπής κρατουμένου, αλυσίδα μεταφοράς κρατουμένου), τον πολλαπλασιασμό των δυαδικών αριθμών (σειριακή και παράλληλη εκδοχή) καθώς και τη διαίρεση των δυαδικών αριθμών που είναι ένα από τα μεγαλύτερα και δυσκολότερα κομμάτια των πράξεων στους ακέραιους αριθμούς. Στο κεφάλαιο 3 μελετάμε τις πράξεις με τους πραγματικούς αριθμούς, ξεκινώντας με μια ευρεία εισαγωγή σχετικά με τους αριθμούς κινητής υποδιαστολής, την μορφή που έχουν οι αυτοί, είτε στα 32 είτε στα 64 μπιτ, σύμφωνα με το πρότυπο IEEE-754 που αναφέρεται στην απεικόνιση των πραγματικών αριθμών κινητής υποδιαστολής σε απλή και διπλή ακρίβεια. Παρουσιάζουμε, βήμα-βήμα, τη μετατροπή ενός δεκαδικού αριθμού σε δυαδικό, είτε σε απλή είτε σε διπλή ακρίβεια, κατά το πρότυπο αυτό, μαζί με παραδείγματα τα οποία βοηθούν τον αναγνώστη να κατανοήσει την διαδικασία. Επίσης, το ίδιο αναλυτικά παρουσιάζουμε και την αντίστροφη διαδικασία, δηλαδή από δυαδικό αριθμό (εκφρασμένο σε κινητή υποδιαστολή) σε δεκαδικό. Παρουσιάζουμε κάποιους τρόπους στρογγυλοποίησης του αποτελέσματος και εξηγούμε πώς λειτουργεί ο καθένας από αυτούς. Το κεφάλαιο 4, περιέχει αναλυτικά την υλοποίηση της μονάδας των ακεραίων αριθμών. Από την μελέτη που κάναμε στην πρόσθεση των δυαδικών αριθμών, είδαμε ότι έχουμε αρκετούς τρόπους για να υλοποιήσουμε την πράξη της πρόσθεσης. Καταλήγοντας λοιπόν ανάμεσα σε δυο μεθόδους αυτή της πρόβλεψης κρατουμένου και την μέθοδο της ριπής κρατουμένου, υλοποιήσαμε και τα δυο σε 8 μπιτ πρώτα, τα προσομοιώσαμε για να βεβαιωθούμε για την ορθή τους λειτουργία και τέλος υλοποιήσαμε και με τις δυο μεθόδους το τελικό το οποίο είναι στα 32 μπιτ. Για την πράξη της διαίρεσης είχαμε να επιλέξουμε ανάμεσα στην παράλληλη και στην σειριακή διαίρεση. Το πλεονέκτημα της σειριακής διαίρεσης είναι ότι χρειάζεται λιγότερους πόρους για να υλοποιηθεί ενώ το μειονέκτημα είναι ότι θέλει αρκετό χρόνο για να γίνει η διαίρεση. Εμείς επιλέξαμε να υλοποιήσουμε την παράλληλη έκδοση, αλλά για τον λόγο ότι το κύκλωμα είναι αρκετά πολύπλοκο για να υλοποιηθεί απευθείας, το χωρίσαμε σε κάποια επιμέρους κομμάτια, υλοποιήσαμε το κάθε κομμάτι ξεχωριστά, το προσομοιώσαμε και στο τέλος τα δέσαμε όλα μαζί σε ένα τελικό σχέδιο. Στο κεφάλαιο 5 παρουσιάζεται αναλυτικά η

υλοποίηση των πράξεων ανάμεσα στους πραγματικούς αριθμούς κινητής υποδιαστολή κατά το πρότυπο IEEE 754. Οι είσοδοι των κυκλωμάτων δέχονται αριθμούς απλής ακρίβεια στα 32 μπιτ, οι οποίοι είναι εκφρασμένοι σύμφωνα με το πρότυπο αυτό. Να σημειωθεί εδώ ότι τα κυκλώματα δεν ελέγχουν τους αριθμούς που δέχονται εάν είναι ή όχι εκφρασμένοι κατά το πρότυπο και για τον λόγο αυτό θα πρέπει η πηγή η οποία δίνει τους αριθμούς να έχει φροντίσει και να έχει κάνει τον έλεγχο αυτό προτού εισάγει τα δεδομένα στις εισόδους των κυκλωμάτων, για να μην γίνει η πράξη άσκοπα χάνοντας χρόνο γιατί και η έξοδος σε αυτή τη περίπτωση θα δώσει λανθασμένα δεδομένα. Η υλοποίηση έγινε με βάση τα διαγράμματα ροής που βρίσκονται στο βιβλίο [1]. Για να υλοποιήσουμε την πρόσθεση, την αφαίρεση και τον πολλαπλασιασμό, κατασκευάσαμε με βάση τα διαγράμματα ροής, το διάγραμμα καταστάσεων του καθενός από αυτά, ώστε στην συνέχεια να μπορέσουμε να τα περάσουμε στο περιβάλλον της ISE Xilinx και με την χρήση της μηχανής τύπου Moore, περιγράψαμε την διαδικασία κάθε κατάστασης όσο πιο αναλυτικά γινόταν στο περιβάλλον και βγάλαμε το τελικό αποτέλεσμα. Τα κυκλώματα της πρόσθεσης και του πολλαπλασιασμού αρχικά υλοποιήθηκαν και προσομοιώθηκαν χωρίς βελτιστοποιήσεις. Ο έλεγχος για τις περιπτώσεις της βελτιστοποίησης έγινε μετά την ολοκλήρωση της υλοποίησης των κυκλωμάτων και την προσομοίωσή τους. Για τις πράξεις της πρόσθεσης και της αφαίρεσης έγινε ένα κύκλωμα. Αυτό το πετύχαμε κάνοντας μια μικρή αλλαγή στο κύκλωμα της πρόσθεσης. Με μια παραπάνω είσοδο και με μερικές γραμμές κώδικα επιπλέον καταφέραμε να έχουμε σε ένα κύκλωμα την μονάδα που κάνει πρόσθεση και αφαίρεση. Για την πράξη του πολλαπλασιασμού δημιουργήθηκε μια ξεχωριστή μονάδα. Για να λειτουργήσουν σωστά τα κυκλώματα αυτά, προσθέσαμε μια είσοδο και μια έξοδο στο καθένα, αντίστοιχα με ονομασία VI (Valid Input) και VO (Valid Out). Η είσοδος VI καθορίζει εάν τα δεδομένα τα οποία δόθηκαν στις εισόδους είναι έγκυρα και η μηχανή μπορεί να τα πάρει να τα χειριστεί. Η έξοδος VO από την άλλη μας δείχνει πότε το αποτέλεσμα που βγήκε στην έξοδο μετά την διεργασία της πράξης είναι έγκυρο. Στο έκτο και τελευταίο κεφάλαιο, αναφέρουμε τα συμπεράσματα που βγάλαμε μετά την ολοκλήρωση της πτυχιακής.

Ασφαλώς στο τέλος της εργασίας υπάρχει η σχετική βιβλιογραφία με αριθμημένα τα συγγράμματα και τις ιστοσελίδες που συμβουλευτήκαμε. Για πληρέστερη κατανόηση στο παράρτημα Α παραθέτονται εκτεταμένα αποσπάσματα κώδικα από την υλοποίηση των κυκλωμάτων.

# 2. Αριθμητικές πράξεις με ακέραιους.

## 2.1 Συστήματα αρίθμησης.

Υπάρχουν διάφορα συστήματα αρίθμησης και το κάθε σύστημα έχει μια βάση. Τα πιο κοινά συστήματα είναι το δεκαδικό, με βάση το 10, και αυτά που έχουν ως βάση δυνάμεις του δύο, όπως το δυαδικό, το οκταδικό και το δεκαεξαδικό. Για κάθε σύστημα ορίζονται οι βασικές αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση) και έχουμε την δυνατότητα να μετατρέψουμε έναν αριθμό συστήματος a σε ένα σύστημα b (αλλαγή βάσης). Τα πιο κοινά αριθμητικά συστήματα που χρησιμοποιούνται είναι το Δεκαδικό με βάση το 10 και ψηφία από το 0 μέχρι το 9, το Δυαδικό με βάση το 2 και ψηφία το 0 και το 1, το Οκταδικό με βάση το 8 και ψηφία από το 0 μέχρι το 7 και το Δεκαεξαδικό με βάση το 16 και ψηφία από το 0 μέχρι το F όπου τα A,B,C,D,E και F αντιστοιχούν σε 10, 11, 12, 13, 14 και 15. Η γενική μορφή των αριθμητικών συστημάτων μπορεί να γραφτεί ως ένα αριθμητικό σύστημα το οποίο έχει βάση b και χρησιμοποιεί b ψηφία με αξία από το 0 μέχρι το b-1.

## 2.2 Δυαδικό σύστημα.

Το δυαδικό σύστημα έχει σαν βάση b το 2 και χρησιμοποιεί ψηφία μέχρι και το b-1 δηλαδή μόνο το 0 και 1. Κάθε ψηφίο στο δυαδικό ανήκει σε μια τάξη μεγέθους μεγαλύτερη κατά ένα από αυτήν του ψηφίου στα δεξιά του. Έτσι κάθε ψηφίο του δυαδικού δηλώνει μονάδες, δυάδες, τετράδες, οκτάδες από τα δεξιά προς τα αριστερά.

## 2.3 Μετατροπή αριθμών από το δυαδικό στο δεκαδικό σύστημα.

Ας υποθέσουμε ότι έχουμε έναν αριθμό του δυαδικού στα 8 μπιτ $(11010011)_2$. Η μετατροπή γίνεται ως εξής :$1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$ = $(211)_{10}$.

## 2.4 Μετατροπή αριθμών από το δεκαδικό στο δυαδικό σύστημα.

Η διαδικασία για να γίνει μια τέτοια μετατροπή έχει πέντε απλά βήματα.

(α) Διαίρεση του δεκαδικού αριθμού με το δύο.

(β) Καταγραφή του πηλίκου και του υπόλοιπου της διαίρεσης.

(γ) Διαίρεση του πηλίκου με το δύο.

(δ) Επανάληψη του (β) και (γ) έως ότου το πηλίκο είναι 0.

(ε) Σχηματισμός του δυαδικού αριθμού από τα υπόλοιπα των διαιρέσεων με την ανάποδη σειρά.

Ας υποθέσουμε ότι έχουμε ένα αριθμό του δεκαδικού $(74)_{10}$. Ακολουθώντας τα βήματα θα έχουμε: $74 \div 2 = 37$ με υπόλοιπο 0. → LSB (Least Significant Bit).

$37 \div 2 = 18$ με υπόλοιπο 1.

$18 \div 2 = 9$ με υπόλοιπο 0.

$9 \div 2 = 4$ με υπόλοιπο 1.

$4 \div 2 = 2$ με υπόλοιπο 0.

$2 \div 2 = 1$ με υπόλοιπο 0.

$1 \div 2 = 0$ με υπόλοιπο 1. → MSB (Most Significant Bit).

Έτσι λοιπόν έχουμε ότι $(74)_{10}$ = $(1001010)_2$.

## 2.5 Απεικόνιση αρνητικών ακέραιων αριθμών.

### 2.5.1 Συμπλήρωμα ως προς 1.

Στην μέθοδο αυτή αντιστρέφονται όλα τα ψηφία του δυαδικού αριθμού, και ο αριθμός που προκύπτει θεωρείται ο αρνητικός του πρώτου. Ας υποθέσουμε ότι έχουμε έναν αριθμό σε ένα Byte τον +7 00000111. Σε συμπλήρωμα ως προς ένα θα έχουμε -7 11111000. Το πρόβλημα που παρουσιάζεται με την συγκεκριμένη μέθοδο είναι πως υπάρχουν δυο αναπαραστάσεις για το μηδέν, 00000000 για θετικό μηδέν και 11111111 για αρνητικό μηδέν. Για να αντιμετωπιστεί αυτό το πρόβλημα δημιουργήθηκε μια δεύτερη μέθοδος καλούμενη ως Συμπλήρωμα ως προς 2.

### 2.5.2 Συμπλήρωμα ως προς 2.

Στην μέθοδο αυτή μετά την αντιστροφή όλων των ψηφίων προσθέτουμε επιπλέον 1. Έτσι λοιπόν σύμφωνα με το προηγούμενο παράδειγμα το +7 σε συμπλήρωμα ως προς 2 θα είναι : 11111000 + 00000001 = 11111001.

### 2.5.3 Αριθμοί προσημασμένου μέτρου.

Ένας άλλος τρόπος για την αναπαράσταση αρνητικών αριθμών είναι με την μέθοδο αυτή στη οποία η αναπαράσταση του προσήμου γίνεται με ένα επιπλέον μπιτ, 0 για θετικό πρόσημο και 1 για αρνητικό. Έστω ότι έχουμε N μπιτ για να αναπαραστήσουμε ένα ακέραιο αριθμό θετικό ή αρνητικό. Το εύρος των αριθμών που μπορούμε να δείξουμε κυμαίνεται στα όρια : $-2^{N-1}$ *ως* $2^{N-1}-1$.

### 2.5.4 Πολωμένη σημειογραφία (Biased Notation).

Στη αναπαράσταση αυτή προστίθεται στον αριθμό που θέλουμε να απεικονίσουμε η πόλωση (bias) που ισούται με $2^{k-1}-1$. Όπου $k$ είναι ο αριθμός των ψηφίων (μπιτ) στο δυαδικό. Στην αντίστροφη διαδικασία όταν μας δοθεί ένας αριθμός με πολωμένη σημειογραφία αφαιρείται η πόλωση για να εξάγουμε τον πραγματικό αριθμό. Ας υποθέσουμε ότι θέλουμε να απεικονίσουμε τον αριθμό -20 στο δυαδικό στα 8 μπιτ, στην περίπτωση αυτή η πόλωση θα είναι $(2^{8-1}-1)=127$. Προσθέτοντας την πόλωση στον αριθμό θα έχουμε $-20+127=107$. Η τελική τιμή του αριθμού στο δυαδικό είναι $(01101011)_2$.

## 2.6 Πρόσθεση, Αφαίρεση και Πολλαπλασιασμός δυαδικών αριθμών.

Για τις βασικές πράξεις πρόσθεση, αφαίρεση και πολλαπλασιασμό υπάρχουν κανόνες τους οποίους ακολουθούμε για να κάνουμε την αντίστοιχη πράξη. Οι κανόνες φαίνονται στον παρακάτω πίνακα.

| | Πράξη | Αποτέλεσμα | Κρατούμενο |
|---|---|---|---|
| Πρόσθεση | 0 + 0 | 0 | 0 |
| | 0 + 1 | 1 | 0 |
| | 1 + 0 | 1 | 0 |
| | 1 + 1 | 0 | 1 |
| Αφαίρεση | 0 – 0 | 0 | 0 |
| | 0 – 1 | 1 | 1 |
| | 1 – 0 | 1 | 0 |
| | 1 – 1 | 0 | 0 |
| Πολλαπλασιασμός | 0 × 0 | 0 | 0 |
| | 0 × 1 | 0 | 0 |
| | 1 × 0 | 0 | 0 |
| | 1 × 1 | 1 | 0 |

Πίνακας 2.1 : Κανόνες Πρόσθεσης, Αφαίρεσης και Πολλαπλασιασμού.

Έστω ότι έχουμε δυο αριθμούς σε 4 μπιτ, τον αριθμό $(5)_{10} = (0101)_2$ και τον αριθμό $(3)_{10} = (0011)_2$. Ακλουθώντας τους κανόνες του παραπάνω πίνακα για και τους κανόνες που γνωρίζουμε στο δεκαδικό για κάθε πράξη θα έχουμε :

$(0101)_2 - (0011)_2 = (0010)_2$

$(0101)_2 + (0011)_2 = (1000)_2$

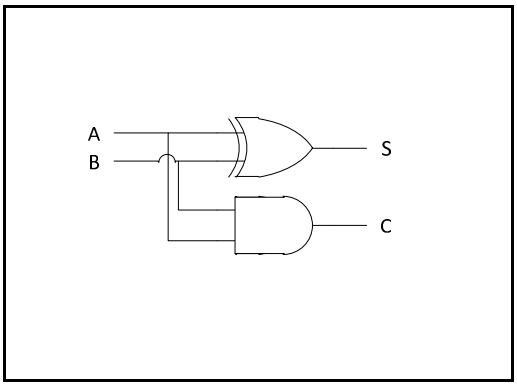
$(0101)_2 \times (0011)_2 = (1111)_2$

## 2.7 Αθροιστές (Adders).

Ο αθροιστής είναι ένα ψηφιακό λογικό κύκλωμα αποτελούμενο από λογικές πύλες. Ο αθροιστής έχει δυο ή περισσότερες εισόδους και σαν έξοδο έχει το αποτέλεσμα της άθροισης των εισόδων αυτών. Τα δυο βασικότερα ψηφιακά κυκλώματα αθροιστών, τα οποία έχουν εφαρμογή στου ηλεκτρονικούς υπολογιστές είναι ο ημιαθροιστής και ο πλήρης αθροιστής.

### 2.7.1 Ημιαθροιστής (Half Adder).

Ο ημιαθροιστής είναι ένα συνδυαστικό κύκλωμα με δυο εισόδους $A$ για τον πρώτο προσθετέο, $B$ για τον δεύτερο προσθετέο και δύο εξόδους $S$ $(Sum)$ για το αποτέλεσμα της πρόσθεσης και $C$ $(Carry)$ για το κρατούμενο εξόδου. Παρακάτω φαίνεται ο πίνακας αληθείας καθώς και το σχηματικό διάγραμμα του ημιαθροιστή.



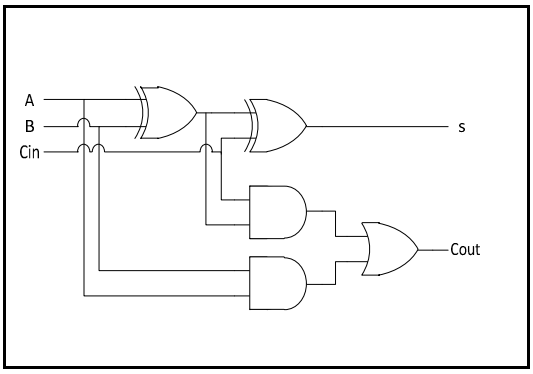

Εικόνα 2.1 : Σχηματικό διάγραμμα ημιαθροιστή.

| A | B | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Πίνακας 2.2 : Πίνακας αληθείας ημιαθροιστή.

### 2.7.2 Πλήρης αθροιστής.

Ο πλήρης αθροιστής κάνει ακριβώς την ίδια εργασία με τον ημιαθροιστή με την διαφορά ότι έχει την δυνατότητα να προσθέσει ακόμα ένα μπιτ (μπιτ) το οποίο είναι το κρατούμενο εισόδου και συμβολίζεται συνήθως με το $C_{in}$. Η υπόλοιπη λογική διάταξη παραμένει ίδια με αυτή του ημιαθροιστή. Παρακάτω φαίνεται ο πίνακας αληθείας και το σχηματικό του διάγραμμα.




Εικόνα 2.2 : Σχηματικό διάγραμμα πλήρη αθροιστή.

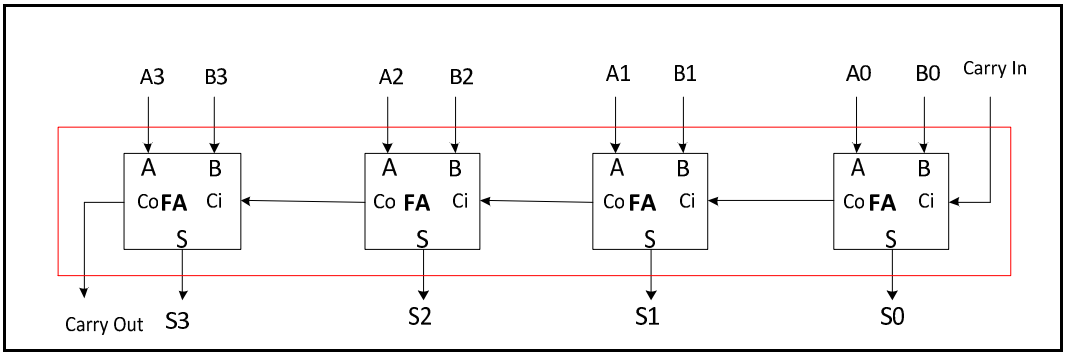
| A | B | Cin | Cout | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Πίνακας 2.3 : Πίνακας αληθείας πλήρη αθροιστή.

Ο πλήρης αθροιστής μπορεί να υλοποιηθεί με αρκετούς τρόπους όπως για παράδειγμα σε επίπεδο τρανζίστορ, σε επίπεδο πύλης ή χρησιμοποιώντας δύο ημιαθροιστές. Το Σχηματικό διάγραμμα 2.2 δείχνει την υλοποίηση σε επίπεδο πύλης η οποία ικανοποιείται από τις παρακάτω εκφράσεις : $S = A \oplus B \oplus C_{in}$ και $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$. Οι χρόνοι καθυστέρησης είναι : $T_S = 2D$ και $T_{cout} = 2D$, όπου $D$ ο χρόνος διάδοσης κάθε πύλης.

### 2.7.3 Αθροιστής ριπής κρατουμένου (Ripple-carry adder).

Ο αθροιστής ριπής κρατουμένου είναι μια λογική διάταξη αποτελούμενη από τόσους πλήρεις αθροιστές όσο είναι και το πλήθος των μπιτ των αριθμών που προστίθενται. Στην συνδεσμολογία αυτής της διάταξης η είσοδος $C_{in}$ του πλήρη αθροιστή συνδέεται με την έξοδο $C_{out}$ της προηγούμενης βαθμίδας. Ένα από τα πλεονεκτήματα αυτού του αθροιστή είναι η γρήγορη σχεδίασή του, ενώ το μεγαλύτερο του μειονέκτημα είναι η «ταχύτητα» πρόσθεσης γιατί η κάθε βαθμίδα περιμένει την προηγούμενή της να κάνει την πρόσθεση και να βγάλει το κρατούμενο εξόδου. Παρακάτω φαίνεται το σχηματικό διάγραμμα ενός τέτοιου αθροιστή στα 4 μπιτ.




Εικόνα 2.3 : Σχηματικό διάγραμμα Ripple-carry Adder στα 4 μπιτ.

### 2.7.4 Αθροιστής Πρόβλεψης Κρατουμένου (Carry Look Ahead).

Η λογική μονάδα της πρόβλεψης κρατουμένου χρησιμοποίει τις έννοιες της παραγωγής (generate) και διάδοσης (propagate) κρατουμένου [3]. Στην πρόσθεση δυο 1-μπιτ δυαδικών αριθμών έστω $A + B$ θα γίνει παραγωγή κρατούμενου $(carry\ generate)$ μόνο στην περίπτωση που και οι δυο είναι 1, επομένως μπορούμε να γράψουμε την εξίσωση $G(A,B) = A \cdot B$. Για να γίνει διάδοση κρατουμένου $(carry\ propagate)$ θα πρέπει ή το $A = 1$ ή το $B = 1$, οπότε μπορούμε να γράψουμε $P(A,B) = A \oplus B$. Σύμφωνα με τις παραπάνω έννοιες μπορούμε να πούμε ότι η πρόσθεση δυο 1-μπιτ αριθμών θα παράγει κρατούμενο στην περίπτωση που ισχύει η παρακάτω εξίσωση. $C_{i+1} = G_i + (P_i \cdot C_i)$, όπου $C_i$ το κρατούμενο της βαθμίδας $i$. Για κάθε ζευγάρι μπιτ που προστίθεται σε μια δυαδική ακολουθία η μονάδα Carry Look Ahead θα πρέπει να προβλέψει εάν θα παράγει (generate) ή θα διαδώσει (propagate) κρατούμενο. Με τον τρόπο αυτό η μονάδα προ επεξεργάζεται τα δυο μπιτ που προστίθενται ώστε να γίνει η πρόβλεψη του κρατουμένου μειώνοντας την καθυστέρηση (delay). Λαμβάνοντας υπόψη τις παραπάνω εξισώσεις, παρακάτω φαίνεται μια μονάδα γενικής χρήσης στα $N = 8\ bit$.

- $C_1 = G_0 + P_0 \cdot C_0$
- $C_2 = G_1 + P_1 \cdot C_1$
- $C_3 = G_2 + P_2 \cdot C_2$
- $C_4 = G_3 + P_3 \cdot C_3$
- $C_5 = G_4 + P_4 \cdot C_4$
- $C_6 = G_5 + P_5 \cdot C_5$
- $C_7 = G_6 + P_6 \cdot C_6$
- $C_8 = G_7 + P_7 \cdot C_7$

αντικαθιστώντας το $C_1$ στο $C_2$, το $C_2$ στο $C_3$,το $C_3$ στο $C_4$, το $C_4$ στο $C_5$, το $C_5$ στο $C_6$, το $C_6$ στο $C_7$ και το $C_7$ στο $C_8$ παίρνουμε τις παρακάτω τελικές εξισώσεις για το κρατούμενο του κάθε προσθετέου :

$C_1 = G_0 + P_0 \cdot C_0$

$C_2 = G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$

$C_3 = G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0))$

$C_4 = G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))$

$C_5 = G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0))))$

$C_6 = G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))))$

$C_7 = G_6 + P_6 \cdot (G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0))))))$

$C_8 = G_7 + P_7 \cdot (G_6 + P_6 \cdot (G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))))))$

Όπου το $C_0$ είναι το κρατούμενο εισόδου και $C_8$ είναι το τελικό κρατούμενο εξόδου.

### 2.7.5 Αλυσίδα μεταφοράς κρατουμένου (Manchester carry chain).

Η αλυσίδα μεταφοράς κρατουμένου είναι μια παραλλαγή της μονάδας πρόβλεψης κρατουμένου το οποίο χρησιμοποιεί μια κοινόχρηστη λογική μονάδα μειώνοντας έτσι κατά πολύ μεγάλο ποσοστό τον αριθμό των τρανζίστορ που χρειάζεται στην υλοποίησή του [5]. Στην μονάδα της πρόβλεψης κρατουμένου για να γίνει η παραγωγή κάθε κρατουμένου

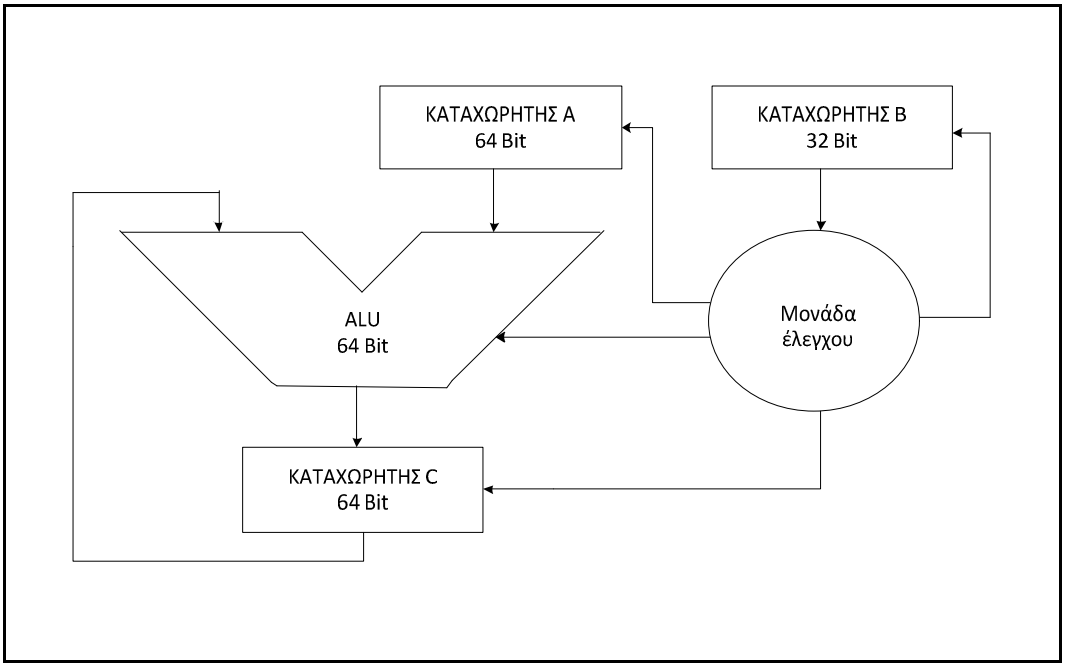
περιέχει την λογική που χρειάζεται για να γίνει παραγωγή των προηγούμενων κρατουμένων. Μια τέτοια μονάδα παράγει κρατούμενα ενδιάμεσα λαμβάνοντας υπόψη κόμβους από τις πύλες που υπολογίζουν την αξία του σημαντικότερου κρατουμένου. Ένα από τα μεγάλα μειονεκτήματα της μονάδας αυτής είναι η πολύ μεγάλη καθυστέρηση διάδοσης λόγω του χωρητικού φορτίου των εξόδων και την αντίσταση των τρανζίστορ. Η μονάδα αυτή χρησιμοποιείται πολύ σπάνια και δεν ξεπερνά τις περισσότερες φορές τα 4 μπιτ.

## 2.8 Πολλαπλασιασμός δυαδικών ακέραιων αριθμών.

Όπως και ο πολλαπλασιασμός των δεκαδικών αριθμών έτσι και στον πολλαπλασιασμό δυαδικών αριθμών ακολουθούμε την ίδια τεχνική. Στους δυαδικούς αριθμούς η πράξη είναι λίγο πιο εύκολη από αυτή με τους δεκαδικούς γιατί έχουμε μόνο δύο ψηφία το 0 και το 1, επομένως είτε θα γράφουμε τον πολλαπλασιαστέο $(multiplicand)$ όπως είναι αν ο πολλαπλασιαστής $(multiplier)$ είναι 1 ή μόνο μηδενικά εάν είναι 0. Το τελικό αποτέλεσμα ονομάζεται γινόμενο $(product)$ και δύναται έχει μέγεθος το άθροισμα των ψηφίων των δύο τελεστέων. Για παράδειγμα εάν έχουμε να πολλαπλασιάσουμε δυο τελεστέους με $m\ bit$ μεγέθους ο πολλαπλασιαστέος και $n\ bit$ ο πολλαπλασιαστής. Το αποτέλεσμα θα έχει μέγεθος $m + n\ bit$ για να μπορεί να γίνει η αναπαράσταση όλων των πιθανών γινομένων. Όπως και στην πρόσθεση έτσι και εδώ πρέπει να γίνει έλεγχος για υπερχείλιση.

### 2.8.1 Σειριακή έκδοση πολλαπλασιασμού.

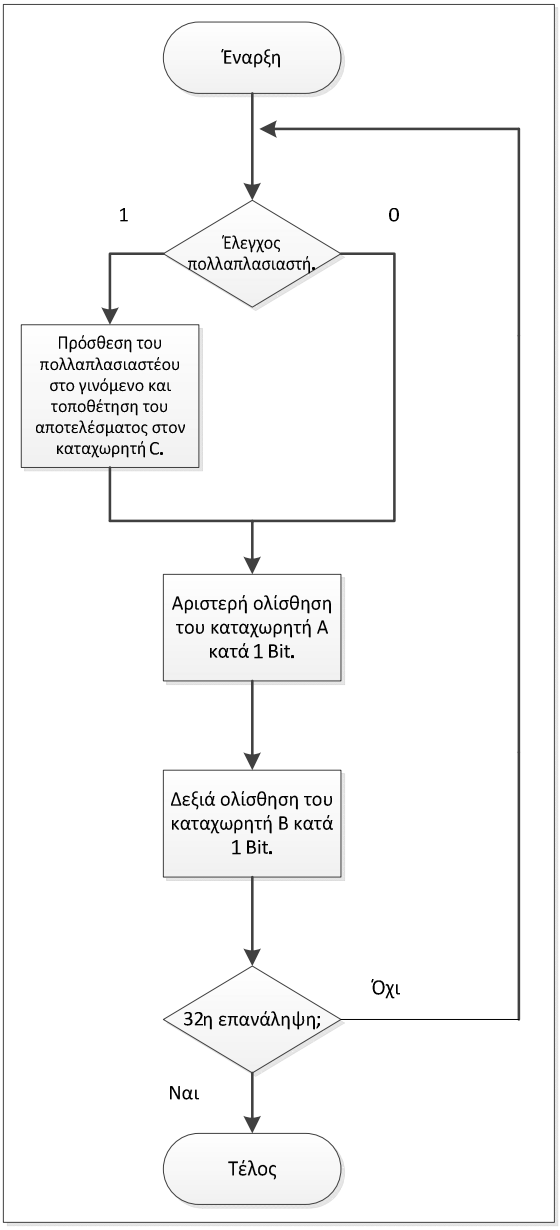
Η τεχνική αυτή μοιάζει κατά πολύ με αυτή που γίνεται με το μολύβι και το χαρτί. Ας υποθέσουμε ότι έχουμε να πολλαπλασιάσουμε δυο τελεστές των 32 μπιτ ο καθένας.




Εικόνα 2.4 : Διάγραμμα υλικού σειριακού πολλαπλασιαστή στα 32 μπιτ.

Ο πολλαπλασιαστής βρίσκεται στον καταχωρητή $B$ και ο πολλαπλασιαστέος στον $A$. Ο καταχωρητής $A$ ολισθαίνει κατά 1 προς τα αριστερά σε κάθε παλμό ρολογιού και ο $B$ προς

τα δεξιά. Το γινόμενο των τελεστών γράφεται σε έναν καταχωρητή 64 μπιτ, τον $C$. Σε κάθε βήμα ο πολλαπλασιαστέος ολισθαίνει κατά μια φορά ένα ψηφίο προς τα αριστερά ώστε να γίνει η πρόσθεση στα ενδιάμεσα γινόμενα εάν ο πολλαπλασιαστής είναι 1. Σε 32 βήματα ο πολλαπλασιαστέος των 32 μπιτ θα έχει μετακινηθεί κατά 32 θέσεις προς τα αριστερά και για αυτό χρειαζόμαστε έναν καταχωρητή στα 64 μπιτ με αρχική τιμή τα 32 μπιτ του πολλαπλασιαστέου στο δεξιό μισό και τα υπόλοιπα 32 μπιτ αρχικοποιημένα σε τιμή 0.



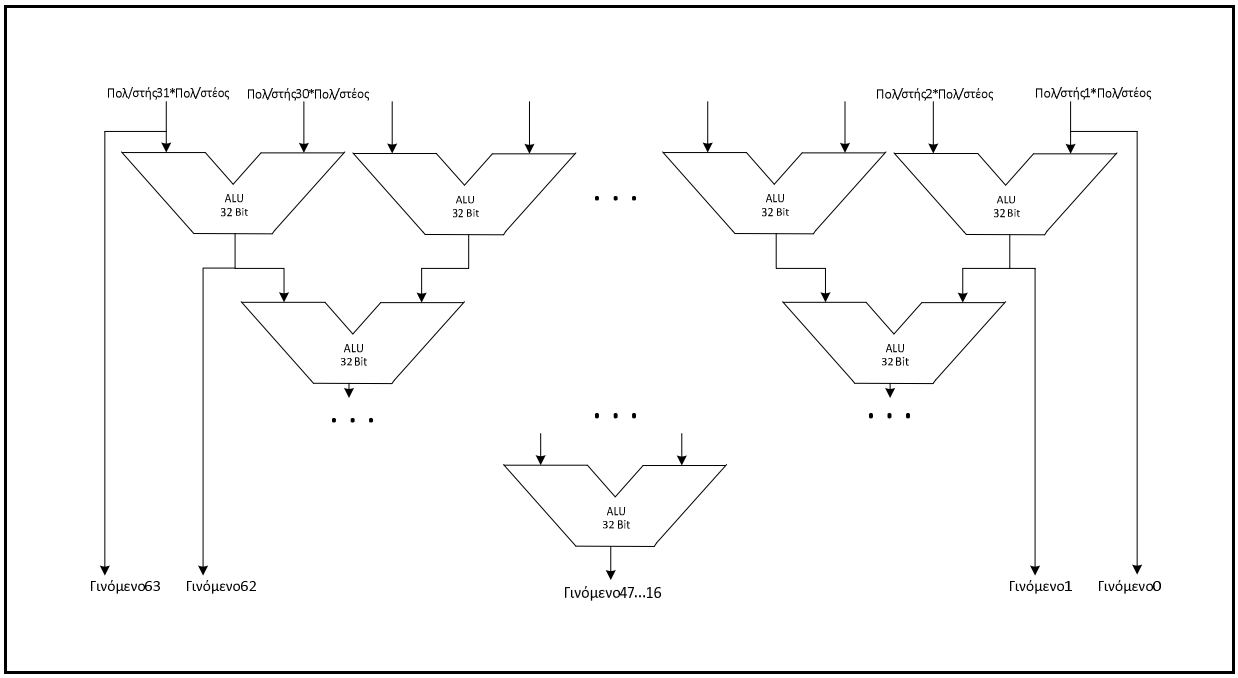

Εικόνα 2.5 : Αλγόριθμος Σειριακού Πολλαπλασιαστή [1].

Ο καταχωρητής αυτός ολισθαίνει προς τα αριστερά κατά 1 μπιτ σε κάθε παλμό ρολογιού, ώστε να ευθυγραμμίσει τον πολλαπλασιαστέο με το άθροισμα το οποίο συσσωρεύεται στον καταχωρητή $C$. Η αριθμητική και λογική μονάδα (ALU) που χρειάζεται εδώ είναι στα 64 μπιτ, και όλα τα κομμάτια ελέγχονται από την μονάδα ελέγχου (Control Unit). Στην

εικόνα 2.4 απεικονίζεται το διάγραμμα υλικού ενώ ο αλγόριθμος ο οποίος υλοποιεί την σειριακή έκδοση παρουσιάζεται στην εικόνα 2.5.

### 2.8.2 Παράλληλη έκδοση πολλαπλασιασμού.

Η τεχνική αυτή εξασφαλίζει μεγαλύτερη ταχύτητα της πράξης από την σειριακή έκδοση, και αυτό γίνεται γιατί αντί για την χρήση ενός μόνο αθροιστή 32 μπιτ για 31 φορές, το υλικό αυτό «ξετυλίγει το βρόχο» ώστε να χρησιμοποιήσει 31 αθροιστές, τους οργανώνει και τους τοποθετεί κατάλληλα ώστε να ελαχιστοποιηθεί η καθυστέρηση. Το βασικότερο μειονέκτημα αυτής της έκδοσης είναι η μεγάλη κατανάλωση του υλικού που έχει αφού χρησιμοποιεί πολλούς αθροιστές μαζί. Το διάγραμμα υλικού φαίνεται παρακάτω.



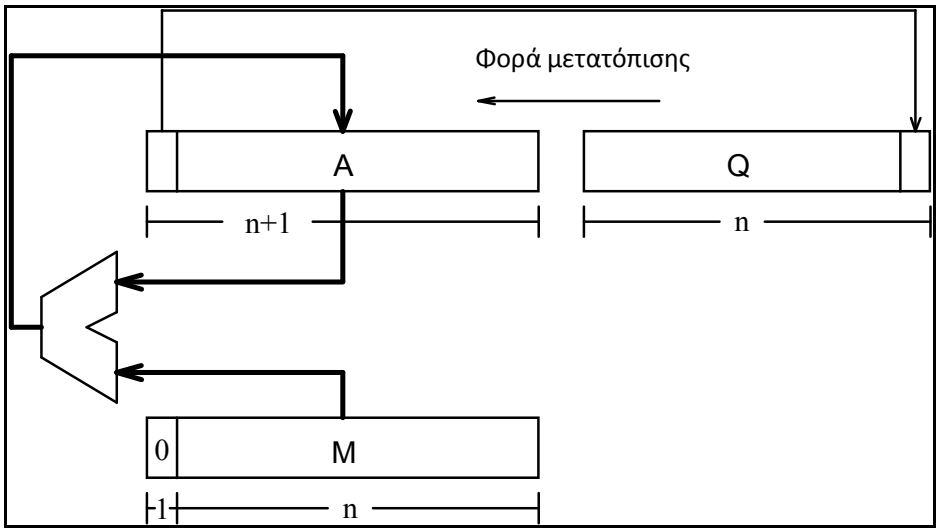

Εικόνα 2.4 : Διάγραμμα υλικού παράλληλου πολλαπλασιαστή στα 32 μπιτ [1].

## 2.9 Η πράξη της διαίρεσης

Είναι γνωστό ότι η πράξη της διαίρεσης συγκρινόμενη με άλλες θεμελιώδεις αριθμητικές ή λογικές πράξεις απαιτεί περισσότερο χρόνο για να εκτελεστεί. Το πιο απλό σχήμα διαίρεσης λειτουργεί με ακεραίους χωρίς πρόσημο και παράγει ένα μπιτ πηλίκου σε κάθε βήμα. Το σχήμα 2.5 παρουσιάζει έναν τέτοιο δυαδικό διαιρέτη. Για να υπολογιστεί το X/Y, εισάγεται ο διαιρετέος X στον καταχωρητή Q, ο διαιρέτης Y στον καταχωρητή M, μηδενίζεται ο καταχωρητής A και στη συνέχεια ακολουθούν τα επόμενα βήματα.

1. Ολίσθηση του ζεύγους των καταχωρητών **AQ** προς τα αριστερά κατά ένα μπιτ.
2. Αφαιρείται από το περιεχόμενο του καταχωρητή **A** το περιεχόμενο του καταχωρητή **M**.
3. Εάν το αποτέλεσμα του βήματος 2 είναι αρνητικό τοποθετείται 0 στο λιγότερο σημαντικό μπιτ (LSB) του καταχωρητή **Q**, ενώ σε αντίθετη περίπτωση τοποθετείται 1.
4. Εάν το αποτέλεσμα του βήματος 2 είναι αρνητικό, αποκαθίσταται η προηγούμενη τιμή του καταχωρητή A με την πρόσθεση σε αυτό του περιεχομένου του **M**.

Μετά την διενέργεια αυτών των βημάτων κατά ***n*** φορές, ο καταχωρητής Q θα περιέχει το πηλίκο ενώ ο καταχωρητής **A** θα περιέχει το υπόλοιπο της διαίρεσης.




Σχήμα 2.5 : Διάγραμμα μπλοκ ενός απλού διαιρέτη απρόσημων ακεραίων με **n** δυαδικά ψηφία.

Ο παραπάνω αλγόριθμος διαίρεσης ονομάζεται «με αποκατάσταση υπολοίπου» (restoring division) διότι αν προκύψει αρνητικός αριθμός μετά την αφαίρεση του διαιρέτη Y (καταχωρητής M) από το υπόλοιπο (καταχωρητής A), το υπόλοιπο αποκαθίσταται. Παρόλα αυτά το βήμα 4 της αποκατάστασης μπορεί να παραλειφθεί αρκεί στο επόμενο βήμα να μην αφαιρεθεί ο διαιρέτης αλλά να προστεθεί και έτσι προκύπτει ο γνωστός από την βιβλιογραφία αλγόριθμος της διαίρεσης χωρίς αποκατάσταση υπολοίπου (non-restoring division).

Π.χ. (rd) : υπόλοιπο R, αν R-Y < 0, αποκατάσταση R-Y+Y, R, ολίσθηση 2R, τελικά 2R-Y.

(n-rd) : υπόλοιπο R, αν R-Y < 0, ολίσθηση 2(R-Y), πρόσθεση 2(R-Y)+Y άρα ισοδύναμα 2R-Y.

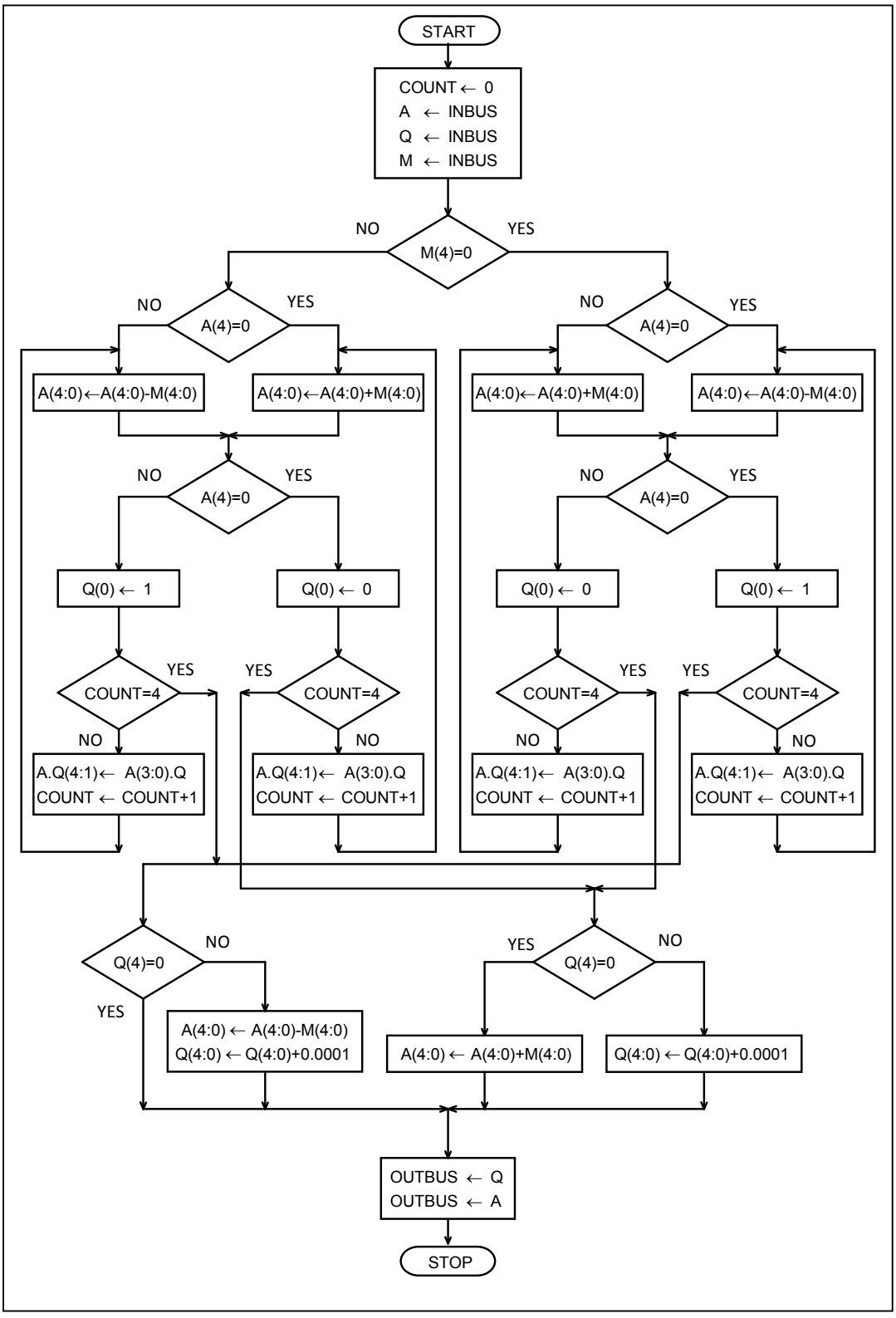
Για δύο μη αρνητικούς ακεραίους αριθμούς, έστω *a* και *b*, η διαίρεση και το υπόλοιπο πρέπει να ικανοποιούν τις παρακάτω σχέσεις : $a = (a \text{ DIV } b)b + a \text{ REM } b, 0 \le a \text{ REM } b < b$.

### 2.9.1 Διαίρεση δύο δυαδικών αριθμών με πρόσημο

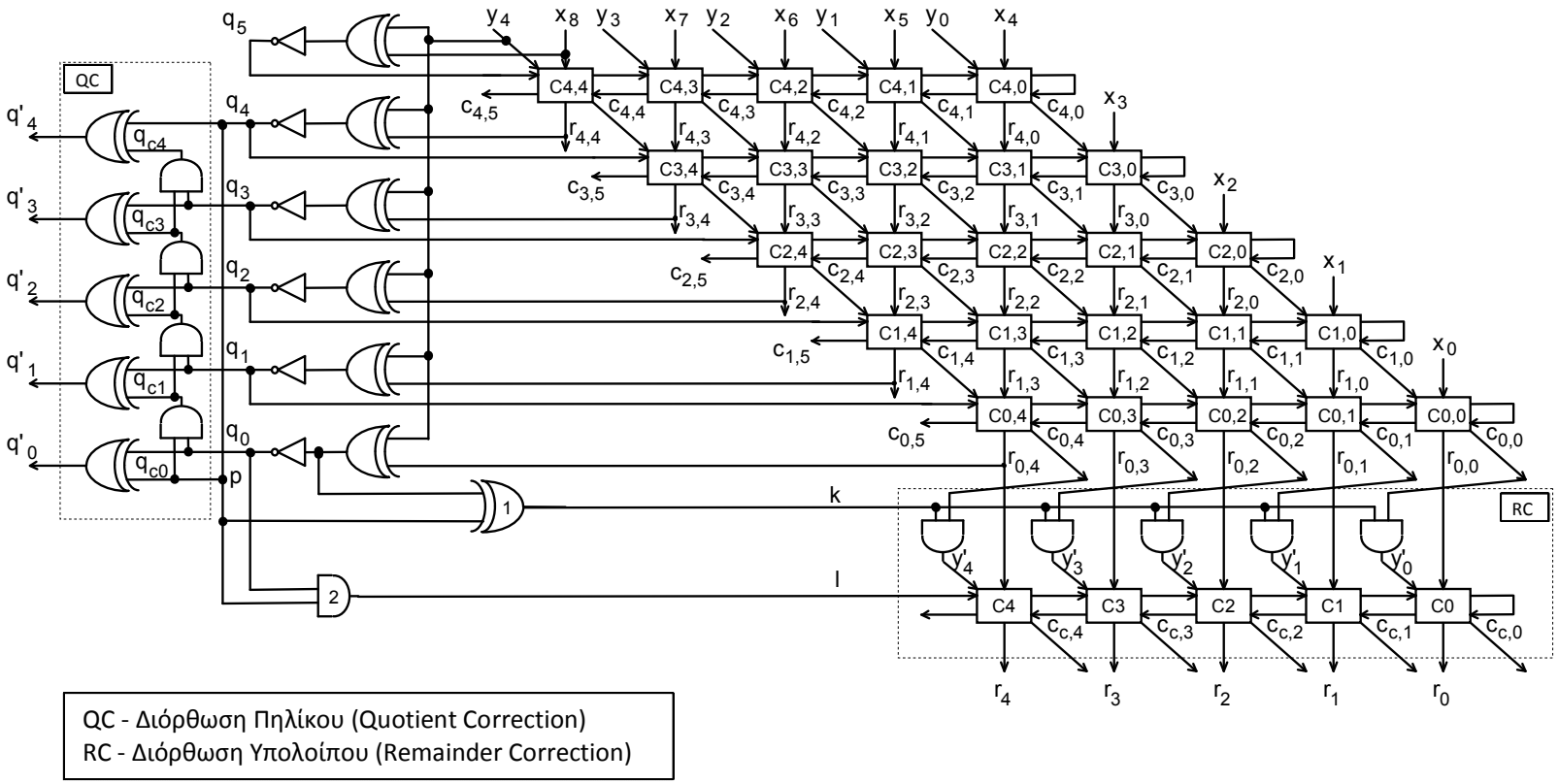
Από την σχετική βιβλιογραφία προκύπτει ότι για την πράξη της δυαδικής διαίρεσης ακολουθούνται τα ακόλουθα βήματα:

i. γίνεται κανονικοποίηση των τελεστέων έτσι ώστε ο διαιρετέος να είναι μικρότερος του διαιρέτη, έχοντας και οι δύο θετικό πρόσημο,
ii. διενεργείται η πράξη της διαίρεσης, με τελεστέους πάντα θετικούς,
iii. Αν χρειάζεται, εφαρμόζεται διόρθωση αποτελεσμάτων (πηλίκου ή/και υπολοίπου),
iv. υπολογίζονται τα πρόσημα των αποτελεσμάτων σε συνάρτηση με τα πρόσημα των τελεστέων.

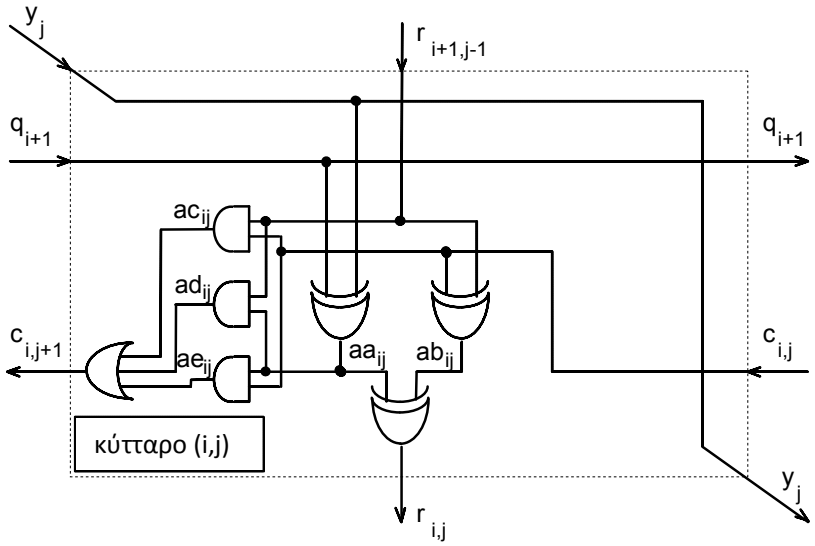
Από τα τέσσερα αυτά βήματα μόνο το ii (ίσως και το iii) υλοποιούνται στο υλικό (hardware), ενώ τα υπόλοιπα αντιμετωπίζονται στο λογισμικό (software), γεγονός που οδηγεί στην μείωση της απόδοσης.




Σχήμα 2.6: Διάγραμμα ροής για την διαίρεση δυαδικών αριθμών με πρόσημο. Αρχικά στους καταχωρητές A και Q ευρίσκεται ο διαιρετέος, ενώ στον καταχωρητή M ευρίσκεται ο διαιρέτης. Μετά την διενέργεια της πράξης ο καταχωρητής A περιέχει το υπόλοιπο της διαίρεσης, ο καταχωρητής Q περιέχει το πηλίκο, ενώ ο καταχωτητής M παραμένει αμετάβλητος [2].




Σχήμα 2.7: Κυτταρικός ή παράλληλος διαιρέτης στα πέντε δυαδικά ψηφία και με πέντε επίπεδα [2].




Σχήμα 2.8: Κύτταρο διαίρεσης.

Χωρίς να χαθεί η γενικότητα θα υποθέσουμε τελεστέους με πέντε δυαδικά ψηφία, ένα εκ των οποίων θα είναι το πρόσημο και τα υπόλοιπα τέσσερα η αξία. Μόνο ο διαιρετέος μπορεί να έχει περισσότερα δυαδικά ψηφία ανάλογα με τον αριθμό των επιπέδων του παράλληλου διαιρέτη.

Έτσι έστω ο αριθμός $X = (x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0)$ ο διαιρετέος όπου το $x_8$ αντιπροσωπεύει το πρόσημό του, και $Y = (y_4 y_3 y_2 y_1 y_0)$ ο διαιρέτης, όπου το $y_4$ παριστά το πρόσημό του.

Βασιζόμενοι στην διαίρεση χωρίς αποκατάσταση (επαναφορά) υπολοίπου (non-restoring type division) χρησιμοποιούμε το διάγραμμα ροής του σχήματος 2.6, το λογικό διάγραμμα του σχήματος 2.7 με κύτταρο που παρουσιάζεται στο σχήμα 2.8. Τα σήματα k, l και p τα οποία χρησιμοποιούνται στα κυκλώματα διόρθωσης των αποτελεσμάτων (RC και QC) περιγράφονται από τις παρακάτω συναρτήσεις [2]:

$$k = \overline{q_4 \oplus q_0} = q_4 \oplus \overline{q_0}$$
$$l = q_4 q_0$$
$$p = q_4$$

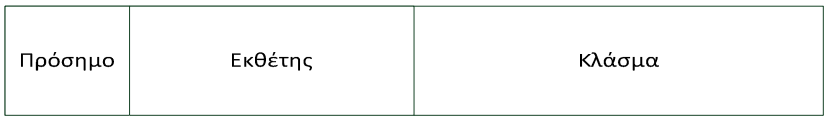
# 3. Αριθμητικές πράξεις με πραγματικούς

## 3.1 Γενικά για τους αριθμούς κινητής υποδιαστολής.

Υπάρχουν διάφοροι τρόποι με τους οποίους μπορούν να αναπαρασταθούν οι πραγματικοί αριθμοί στους υπολογιστές. Ένας από αυτούς είναι η κινητή υποδιαστολή. Η αναπαράσταση αυτή δεν είναι προκαθορισμένη, αλλά μεταβλητή και προσαρμόζεται αυτόματα στις μεταβαλλόμενες απαιτήσεις των υπολογισμών. Στην περίπτωση αυτή η θέση της δυαδικής υποδιαστολής λέμε ότι μετακινείται και οι αριθμοί αυτοί ονομάζονται **αριθμοί κινητής υποδιαστολής**, το οποίο ξεχωρίζει από τους **αριθμούς σταθερής υποδιαστολής** στους οποίους η υποδιαστολή μένει πάντα στην ίδια θέση. Για την αναπαράσταση των αριθμών κινητής υποδιαστολής χρησιμοποιείται η επιστημονική σημειογραφία στους οποίους οι αριθμοί γράφονται με την μορφή $d.dddd \times 10^{exponent}$ όπου d είναι τα δεκαδικά ψηφία με βάση το 10 και exponent ο εκθέτης του. Με λίγα λόγια αυτό που κάνουμε είναι να μετατοπίσουμε την υποδιαστολή σε μια βολική θέση και χρησιμοποιούμε δυνάμεις του 10 για να γνωρίζουμε ανά πάσα στιγμή την θέση της υποδιαστολής. Έστω ότι έχουμε τον δεκαδικό αριθμό $123{,}456_{(10)}$. Ο αριθμός αυτός μπορεί να γραφτεί ως εξής : $1{,}23456 \times 10^{2}{}_{(10)}$. Η δύναμη $10^2$ μας δείχνει πως η υποδιαστολή έχει μετακινηθεί δύο θέσεις προς τα αριστερά. Στην περίπτωση που η δύναμη ήταν αρνητική αυτό σήμαινε ότι η υποδιαστολή θα είχε μετακινηθεί προς τα δεξιά.

## 3.2 Μορφή αριθμών κινητής υποδιαστολής.

Η μορφή των αριθμών κινητής υποδιαστολής έχει ως εξής : $\pm X_1.X_2X_3X_4X_5X_6X_7X_i \times 10^{\pm Y_1Y_2Y_i}$ όπου τα $X_i$ και $Y_i$ είναι δεκαδικά ψηφία. Το εύρος των αριθμών που μπορεί να αναπαρασταθεί με την μορφή αυτή εξαρτάται από τα δεκαδικά ψηφία X για το δεκαδικό μέρος και από τα ψηφία Y για το εύρος του εκθέτη. Συνήθως το εύρος αυτό αντιστοιχεί σε 32 μπιτ το οποίο είναι ένα τυποποιημένο μέγεθος μιας λέξης ενός υπολογιστή όταν αναφερόμαστε σε δυαδικούς αριθμούς. Οι αριθμοί αυτοί αποθηκεύονται σε τρία πεδία : το πρώτο πεδίο το ονομάζεται πρόσημο (Sign), το δεύτερο πεδίο εκθέτης (Exponent) και το τρίτο κλάσμα (Fraction ή Mantissa).




Σχήμα 3.1 : Τυπική μορφή αριθμού κινητής υποδιαστολής.

### 3.2.1 Πρόσημο (Sign).

Στους αριθμούς κινητής υποδιαστολής το πεδίο αυτό ή αλλιώς το πρώτο μπιτ της δυαδικής λέξης δηλώνει το πρόσημο του αριθμού που είναι αποθηκευμένο στα επόμενα μπιτ. Αν το μπιτ είναι 0 ο αριθμός είναι θετικός ενώ αν είναι 1 τότε ο αριθμός είναι αρνητικός.

### 3.2.2 Εκθέτης (Exponent).

Στο πεδίο αυτό αποθηκεύεται ο εκθέτης του αριθμού με βάση το 2 αφού αναφερόμαστε στους δυαδικούς αριθμούς. Στην αναπαράσταση του εκθέτη χρησιμοποιείται η πολωμένη σημειογραφία. Η πόλωση είναι ίση με : $2^{k-1} - 1$.

### 3.2.3 Κλάσμα ή σημαντικό μέρος (Fraction or Mantissa).

Στο τρίτο και το τελευταίο πεδίο το οποίο καταλαμβάνει συνήθως και τα περισσότερα μπιτ σε μια δυαδική λέξη αποθηκεύεται η δυαδική πληροφορία μετά την υποδιαστολή κανονικοποιημένη. Στην τυπική μορφή της λέξης που συνήθως είναι 32 μπιτ το κλάσμα καταλαμβάνει τα 23 λιγότερο σημαντικά μπιτ εκτός κρυφού μπιτ (hidden μπιτ) το οποίο δεν καταλαμβάνει χώρο για την αποθήκευση. Η τελική μορφή στην οποία αποθηκεύεται ο δυαδικός αριθμός έχει την εξής μορφή : $(-1)S \times 1.bbbb \times 2^{exp}$ .

### 3.2.4 Κανονικοποίηση.

Ένας αριθμός είναι κανονικοποιημένος όταν το πιο σημαντικό μέρος του αριθμού είναι μη μηδενικό. Για την αναπαράσταση των αριθμών με βάση το 2 ο αριθμός θεωρείται κανονικοποιημένος όταν το σημαντικότερο μπιτ του σημαντικού μέρους είναι 1. Στην περίπτωση που ο αριθμός δεν είναι κανονικοποιημένος τότε πρέπει να μετακινηθεί η υποδιαστολή αριστερά ή δεξιά και ο εκθέτης να αυξηθεί ή να μειωθεί αντίστοιχα για να κανονικοποιηθεί ο αριθμός. Η τυπική σύμβαση είναι ότι υπάρχει πάντα το hidden μπιτ στα αριστερά της υποδιαστολής που είναι 1, για τον λόγο αυτό δεν χρειάζεται να αποθηκεύσουμε το μπιτ γιατί υπονοείται. Παρακάτω φαίνονται μερικά παραδείγματα κανονικοποίησης αριθμών :

- Έστω ότι έχουμε τον παρακάτω αριθμό ο οποίος σύμφωνα με τα παραπάνω δεν είναι κανονικοποιημένος $0.00011 \times 2^4$, επομένως θα πρέπει να μετακινήσουμε την υποδιαστολή προς τα δεξιά και να μειώσουμε τον εκθέτη κατά τόσες φορές όσες και η μετακίνηση τις υποδιαστολής.
  $0.00011 \times 2^4 = 0.0011 \times 2^3 = 0.011 \times 2^2 = 0.11 \times 2^1 = 1.1 \times 2^0$
- Έστω ότι έχουμε τον παρακάτω αριθμό $10010.001 \times 2^0$. Στην περίπτωση αυτή για να γίνει κανονικοποίηση του αριθμού θα πρέπει να μετακινήσουμε την υποδιαστολή προς τα αριστερά και να αυξήσουμε τον εκθέτη.
  $10010.001 \times 2^0 = 1001.0001 \times 2^1 = 100.10001 \times 2^2 = 10.010001 \times 2^3 = 1.0010001 \times 2^4$.

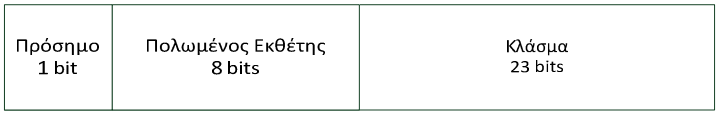
## 3.3 Περιοχή τιμών και ακρίβεια.

Αν είχαμε ακέραιους ο μέγιστος αριθμός που μπορεί να αναπαρασταθεί στα 32 μπιτ είναι $2^{32}$. Στους πραγματικούς όμως με κινητή υποδιαστολή ο εκθέτης σε μια λέξη των 32 μπιτ καταλαμβάνει 8 μπιτ και το κλάσμα τα 23 μπιτ. Αν αυξήσουμε τον

αριθμό των μπιτ στον εκθέτη δεσμεύοντας περισσότερα μπιτ στα 32, από την μια επεκτείνουμε την περιοχή τιμών των αριθμών που μπορεί να εκφραστούν από την άλλη όμως έχουμε μειώσει την ακρίβεια γιατί τα επιπρόσθετα μπιτ δεσμεύονται από τα 23 λιγότερα σημαντικά μπιτ δηλαδή από το κλάσμα. Ο μόνος τρόπος για να αυξηθεί τόσο η περιοχή των τιμών όσο και η ακρίβεια είναι να χρησιμοποιηθούν περισσότερα από 32 μπιτ. Έτσι οι περισσότεροι υπολογιστές μπορούν να δουλέψουν τόσο σε απλή ακρίβεια στα 32 μπιτ όσο και σε διπλή ακρίβεια στα 64 μπιτ.

## 3.4 Το πρότυπο IEEE 754.

Η IEEE είναι ένα διεθνούς κύρους επιστημονικό ινστιτούτο ηλεκτρολόγων και ηλεκτρονικών μηχανικών το οποίο εκτός των άλλων έχει στόχο την κατάρτιση προτύπων σχετικών με το αντικείμενό του. Το συγκεκριμένο πρότυπο 754 αναπτύχθηκε για να διευκολυνθεί η φορητότητα των προγραμμάτων από τον έναν επεξεργαστή στον άλλον, και να προωθηθεί η ανάπτυξη σύνθετων προγραμμάτων με προσανατολισμό τους υπολογισμούς. Το πρότυπο έχει γίνει ευρέως αποδεκτό και χρησιμοποιείται σε όλους τους σύγχρονους επεξεργαστές και αριθμητικούς συνεπεξεργαστές. Το πρότυπο αυτό ορίζει δύο βασικές μορφές. Έχουν αναπτυχθεί ακόμη δυο εκτεταμένες μορφές. Οι δυο βασικές μορφές είναι αυτής της απλής ακρίβειας (single precision) και αυτή της διπλής ακρίβειας (double precision). Ενώ η εκτεταμένες μορφές είναι οι απλή-εκτεταμένη (single-extended precision) και η διπλή-εκτεταμένη (double-extended precision).




Σχήμα 3.2 : Single Precision


Σχήμα 3.3 :Double Precision

Όπως φαίνεται από τα παραπάνω σχήματα στην απλή ακρίβεια έχουμε ένα μπιτ για το πρόσημο, 8 μπιτ για τον εκθέτη και 23 μπιτ για το κλάσμα ενώ για την διπλή ακρίβεια έχουμε επίσης ένα μπιτ για το πρόσημο, 11 μπιτ για τον εκθέτη και 52 μπιτ για το κλάσμα. Είναι ολοφάνερο ότι στην διπλή ακρίβεια μπορούμε να αναπαραστήσουμε και μεγαλύτερο εύρος αριθμών από αυτό της απλής ακρίβειας. Η μορφή των εκτεταμένων μορφών εξαρτάται από την υλοποίηση. Οι εκτεταμένες μορφές περιλαμβάνουν επιπλέον μπιτ στον εκθέτη (εκτεταμένη περιοχή τιμών) και

επιπλέον μπιτ στο σημαντικό μέρος (εκτεταμένη ακρίβεια). Οι μορφές αυτές χρησιμοποιούνται για την χρήση σε ενδιάμεσους υπολογισμούς. Με την μεγαλύτερη ακρίβεια των εκτεταμένων μορφών μειώνεται η πιθανότητα εμφάνισης ενός τελικού αποτελέσματος το οποίο έχει υποβαθμιστεί από υπερβολικό σφάλμα στρογγυλοποίησης. Με την μεγαλύτερη περιοχή τιμών μειώνουν την πιθανότητα να ακυρωθεί ένας ενδιάμεσος υπολογισμός λόγω υπερχείλισης, ενώ το τελικό αποτέλεσμα θα μπορούσε να αναπαρασταθεί με μια βασική μορφή. Ένα ακόμη κίνητρο για την χρήση αυτής της μορφής απλής επέκτασης είναι ότι έχει κάποια από τα πλεονεκτήματα μιας μορφής διπλής ακρίβειας, χωρίς όμως το κόστος σε χρόνο που συνήθως απαιτεί η υψηλότερη ακρίβεια. Παρακάτω απεικονίζεται ένας πίνακας που δείχνει περιληπτικά τα χαρακτηρίστηκα των τεσσάρων μορφών.

| ΠΑΡΑΜΕΤΡΟΣ | ΑΠΛΗ ΑΚΡΙΒΕΙΑ | ΔΙΠΛΗ ΑΚΡΙΒΕΙΑ |
|---|---|---|
| Εύρος λέξης (μπιτ) | 32 | 64 |
| Εύρος εκθέτη (μπιτ) | 8 | 11 |
| Πόλωση εκθέτη | 127 | 1023 |
| Μέγιστος εκθέτης | 127 | 1023 |
| Ελάχιστος εκθέτης | -126 | -1022 |
| Περιοχή τιμών αριθμών (βάση 10) | $10^{-38} - 10^{+38}$ | $10^{-308} - 10^{+308}$ |
| Εύρος του σημαντικού μέρους (μπιτ) | 23 | 52 |
| Αριθμών εκθετών | 254 | 2046 |
| Αριθμός κλασμάτων | $2^{33}$ | $2^{52}$ |
| Αριθμός τιμών | $1.98 \times 2^{31}$ | $1.98 \times 2^{63}$ |

Πίνακας 3.1 : Χαρακτηριστικά των μορφών του προτύπου IEEE 754.

## 3.5 Μετατροπή αριθμών κινητής υποδιαστολής (IEEE 754).

### 3.5.1 Μετατροπή από δεκαδικό σε δυαδικό.

Ας υποθέσουμε ότι έχουμε τον αριθμό 150,75 του δεκαδικού. Για να μετατρέψουμε τον αριθμό αυτό σε δυαδικό απλής ή διπλής ακρίβειας σύμφωνα με το πρότυπο IEEE 754, θα ακολουθήσουμε τα παρακάτω βήματα.

- Απλή ακρίβεια 32 μπιτ.
  Στην απλή ακρίβεια ξέρουμε ότι έχουμε 1 μπιτ για το πρόσημο (S), 8 μπιτ για τον εκθέτη (exponent), 22 μπιτ για το κλάσμα (fraction) και η πόλωση ισούται με127.

**Βήμα 1 :** γράφουμε το μπιτ προσήμου ανάλογα με το πρόσημο του αριθμού. Στην περίπτωσή μας ο αριθμός είναι θετικός επομένως το μπιτ θα είναι S = 0.
**Βήμα 2 :** μετατρέπουμε τον δεκαδικό αριθμό σε δυαδικό με τον κλασικό τρόπο που έχουμε μάθει. $(150{,}75)_{10} = (10010110{,}11)_2$.
**Βήμα 3**: κανονικοποιούμε τον αριθμό μεταφέροντας την υποδιαστολή προς τα αριστερά τόσες θέσεις ώστε να μείνει μόνο ένα ψηφίο αριστερά της υποδιαστολής και να είναι μη μηδενικό, $(10010110{,}11) = (1{,}001011011) \times 2^7$.
**Βήμα 4** : προσθέτουμε το 7 στην πόλωση και μετατρέπουμε τον αριθμό σε δυαδικό ώστε να έχουμε τον εκθέτη στα 8 μπιτ. $127 + 7 = (134)_2 = (10000110)_2$.
**Βήμα 5** : γράφουμε το πρόσημο, τον εκθέτη και το κλάσμα στα 32 μπιτ στο αντίστοιχο πεδίο το καθένα λαμβάνοντας υπόψη ότι το πρώτο μπιτ του παραπάνω κλάσματος δεν το γράφουμε στον αντίστοιχο πεδίο διότι θεωρείται κρυφό όπως προαναφέραμε παραπάνω.

| | | |
|---|---|---|
| 0 | 10000110 | 00101101100000000000 |

- Διπλή ακρίβεια 64 μπιτ.
  Στην διπλή ακρίβεια το κλάσμα και η δύναμη του εκθέτη θα παραμείνουν το ίδιο. Το μόνο που θα αλλάξει είναι η πόλωση που στην διπλή ακρίβεια είναι 1023 και το μέγεθος του κάθε πεδίου θα είναι : για το πρόσημο 1 μπιτ, για τον εκθέτη 11 μπιτ και για το κλάσμα 52 μπιτ. Σύμφωνα με τα παραπάνω ο αριθμός στα 64 μπιτ θα γραφτεί :

| | | |
|---|---|---|
| 0 | 10000000110 | 0010110110000000000000000000000000000000000000000000000000 |

### 3.5.2 Μετατροπή από δυαδικό στο δεκαδικό.

Ας υποθέσουμε ότι έχουμε την δυαδική ακολουθία 10111111001000000000000000000000 στα 32 μπιτ κωδικοποιημένο σύμφωνα με το πρότυπο IEEE 754. Για να μετατρέψουμε την δυαδική ακολουθία σε δεκαδικό ακολουθούμε τα παρακάτω βήματα.
**Βήμα 1 :** χωρίζουμε τα 32 μπιτ σε τμήματα σύμφωνα με το πρότυπο. Θα έχουμε ένα μπιτ για το πρόσημο, 8 μπιτ για τον εκθέτη και 23 μπιτ για το κλάσμα.
**Βήμα 2 :** αναγνωρίζουμε το πρόσημο του δεκαδικού αριθμού σύμφωνα με το μπιτ πρόσημου. Αφού το μπιτ αυτό είναι 1, ο αριθμός είναι αρνητικός.
**Βήμα 3 :** μετατρέπουμε τα 8 μπιτ του εκθέτη σε δεκαδικό αριθμό και αφαιρούμε την πόλωση τής απλής ακρίβειας που είναι 127, ώστε να μπορούμε να βρούμε την θέση της υποδιαστολής. Επομένως θα έχουμε ότι : $(01111110)_2 = (126)_{10}$ και στην συνέχεια $126 - 127 = -1$.
**Βήμα 4 :** γράφουμε την τελική μορφή του δυαδικού αριθμού με το κλάσμα με το κρυφό μπιτ και στην συνέχεια μετακινούμε την υποδιαστολή 1 θέση προς τα αριστερά : $-1.01 \times 2^{-1} = -0.101 \times 2^0$.
**Βήμα 5 :** τέλος μετατρέπουμε και τα δύο μέρη της δυαδικής ακολουθίας σε δεκαδικό για να πάρουμε τον αριθμό : $(-0.101)_2 = (-0.625)_{10}$.

Στην περίπτωση που ο αριθμός ήταν κωδικοποιημένος κατά πρότυπο IEEE 754 σε διπλή ακρίβεια (64 μπιτ), το μόνο που θα άλλαζε στην διαδικασία μετατροπής του σε δεκαδικό θα ήτανε στο **Βήμα 3**. Αντί για 127 που είναι η πόλωση της απλής ακρίβειας θα είχαμε 1023 που είναι η πόλωση της διπλής ακρίβειας.

## 3.6 Στρογγυλοποίηση

Η τεχνική της στρογγυλοποίησης βοηθά στην πιο σωστή απεικόνιση του αριθμού σε απλή και διπλή ακρίβεια. Για την τεχνική αυτή χρησιμοποιούνται τρία επιπλέον μπιτ sticky, guard και round μπιτ. Το πρότυπο IEEE 754 ορίζει τέσσερεις βασικούς τρόπους στρογγυλοποίησης που θα τους δούμε παρακάτω.

### 3.6.1 Στρογγυλοποίηση προς το πλησιέστερο.

Στην τεχνική αυτή το αποτέλεσμα στρογγυλοποιείται προς το πλησιέστερο δυαδικό αριθμό ο οποίος μπορεί να αναπαρασταθεί.

### 3.6.2 Στρογγυλοποίηση στο συν άπειρο.

Αυτή η τεχνική είναι χρήσιμη στ ην αριθμητική των διαστημάτων. Το αποτέλεσμα στρογγυλοποιείται προς το συν άπειρο. Εάν το πρόσημο είναι θετικό και τα τρία επιπλέον μπιτ είναι διάφορα του μηδενός δηλ. «000», τότε η στρογγυλοποίηση του αποτελέσματος γίνεται προσθέτοντας έναν άσο στο τέλος του σημαντικού μέρους.

### 3.6.3 Στρογγυλοποίηση προς το μείον άπειρο.

Η τεχνική αυτή είναι παρόμοια με αυτή της στρογγυλοποίησης προς το συν άπειρο. Εάν δηλαδή το πρόσημο είναι αρνητικό αυτή τη φορά και τα επιπλέον μπιτ είναι και πάλι διάφορα του μηδενός δηλ «000», τότε για να κάνουμε στρογγυλοποίηση προσθέτουμε έναν άσο στο τέλος του σημαντικού.

### 3.6.4 Στρογγυλοποίηση προς το μηδέν.

Η στρογγυλοποίηση προς το μηδέν είναι η πιο απλή τεχνική της αποκοπής των επιπλέων μπιτ. Στην τεχνική αυτή το μέτρο της τιμής που έχει υποστεί περικοπή είναι πάντοτε μικρότερο ή ίσο με την πιο κοντινή αρχική τιμή. Στην διαδικασία αυτή εισάγεται μια τιμή πόλωσης προς το μηδέν πάντοτε.

# 4. Υλοποίηση μονάδας ακεραίων.

## 4.1 Πρόσθεση – Αφαίρεση.

Η πρόσθεση και η αφαίρεση υλοποιήθηκε με την τεχνική Carry Look Ahead. Χρησιμοποιήσαμε μια μονάδα πρόβλεψης κρατουμένου στα 8 μπιτ, και στην συνέχεια με την ένωση των τεσσάρων μονάδων των 8 μπιτ, υλοποιήσαμε μια μεγαλύτερη στα 32 μπιτ. Για αρχή γράψαμε τις τελικές εξισώσεις για μια τέτοια μονάδα με την βοήθεια των εξισώσεων που αναφέρονται στο 2.7.4.

$$C_1 = G_0 + P_0C_0, C_2 = G_1 + P_1C_1$$

$$C_3 = G_2 + P_2C_2, C_4 = G_3 + P_3C_3$$

$$C_5 = G_4 + P_4C_4, C_6 = G_5 + P_5C_5$$

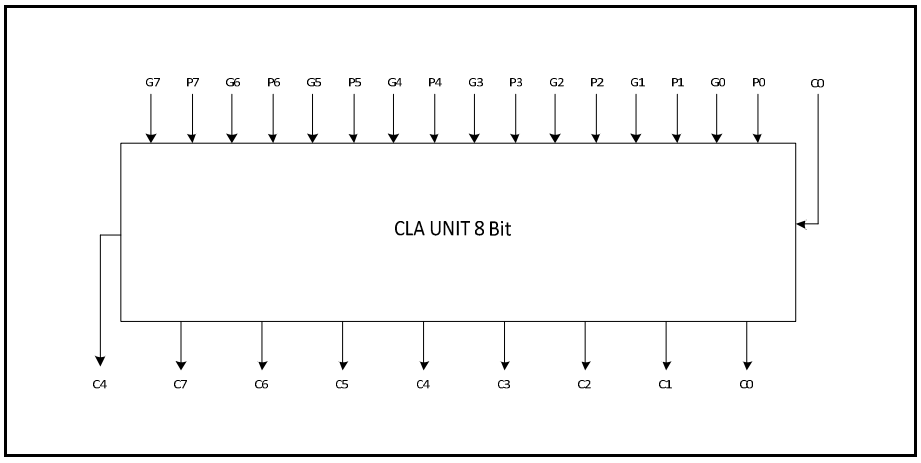
$$C_7 = G_6 + P_6C_6, C_8 = G_7 + P_7C_7$$

Αντικαταστήσαμε την εξίσωση

$C_1\ στην\ C_2, την\ C_2\ στην\ C_3, την\ C_3\ στην\ C_4, την\ C_4\ στην\ C_5, την\ C_5\ στην\ C_6, την\ C_6\ στην\ C_7\ και\ την\ C_7\ στην\ C_8$ και προκύπτουν οι παρακάτω τελικές συναρτήσεις για την μονάδα :

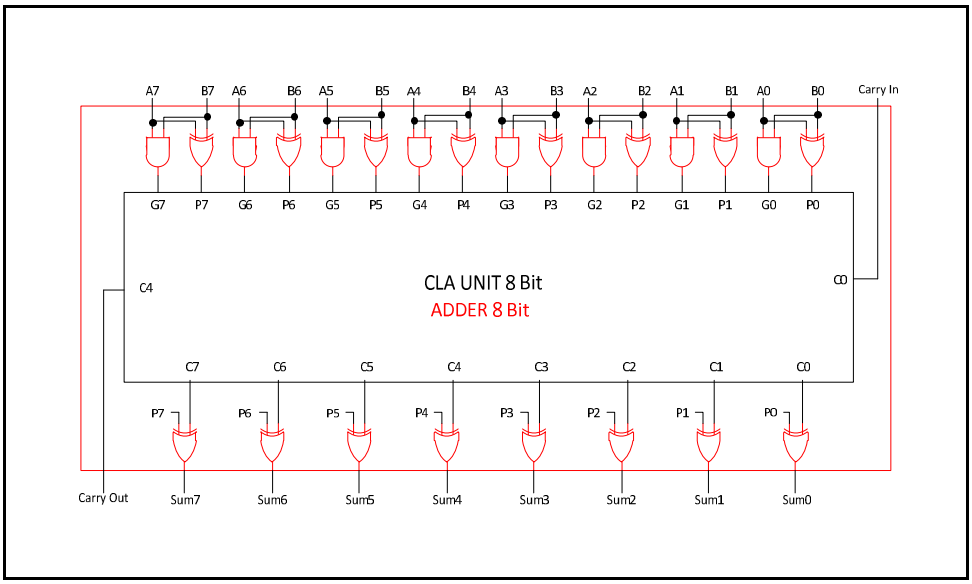
$C_1 = G_0 + P_0 \cdot C_0$
$C_2 = G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$
$C_3 = G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0))$
$C_4 = G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))$
$C_5 = G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0))))$
$C_6 = G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))))$
$C_7 = G_6 + P_6 \cdot (G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0))))))$
$C_8 = G_7 + P_7 \cdot (G_6 + P_6 \cdot (G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))))))$

Στην συνέχεια υλοποιήσαμε την μονάδα αυτή σε γλώσσα περιγραφής υλικού VHDL στο περιβάλλον ISE Design Suite της Xilinx. Παρακάτω παρουσιάζεται το μπλοκ διάγραμμα της μονάδας.



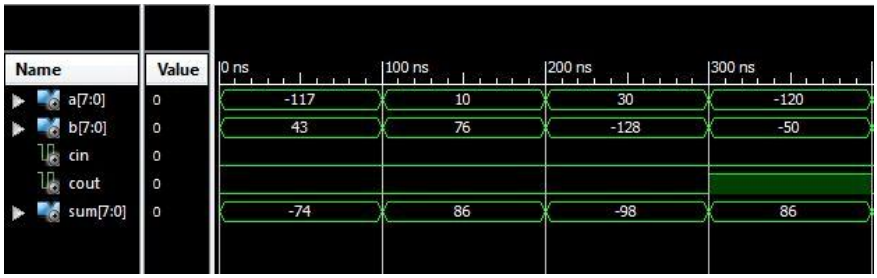

Σχήμα 4.1: Μπλοκ διάγραμμα CLA στα 8 μπιτ.

Το επόμενο βήμα ήταν με την βοήθεια της μονάδας και της εξίσωσης $Sum_i = P_i \oplus C_i$ να κατασκευάσουμε έναν αθροιστή δυο τελεστέων με πρόσημο στα 8 μπιτ. Μετά την υλοποίησή του, κάναμε προσομοίωση για να εξακριβώσουμε την ορθή λειτουργία του. Παρακάτω παρουσιάζονται το μπλοκ διάγραμμα και οι τιμές από την προσομοίωσή του.



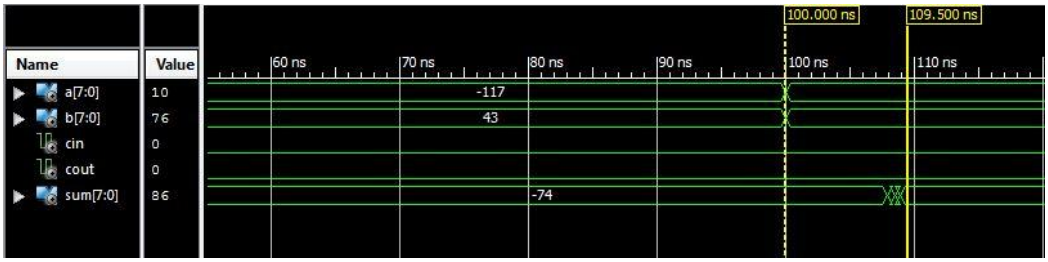

Σχήμα 4.2: Μπλοκ διάγραμμα Αθροιστή στα 8 μπιτ.

Στο παραπάνω μπλοκ διάγραμμα υπάρχουν δυο μονάδες, η κορυφαία μονάδα με το κόκκινο «πλαίσιο» το οποίο αντιστοιχεί στον αθροιστή στα 8 μπιτ και μέσα σε αυτό υπάρχει μια ακόμη μονάδα η CLA UNIT που παρουσιάζεται με το μαύρο «πλαίσιο».



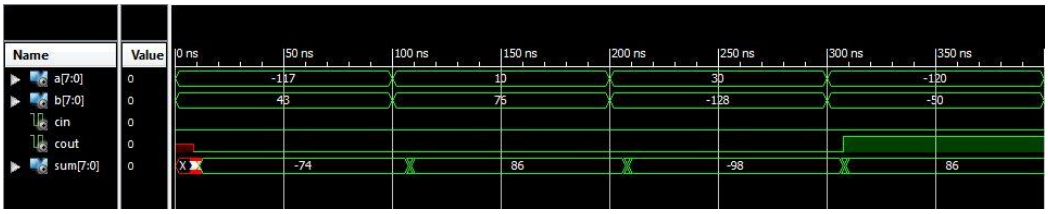

Εικόνα 4.1: Αποτελέσματα προσομοίωσης (Behavioral Simulation) αθροιστή στα 8 μπιτ.

Στο Σχήμα 4.1 παρουσιάζονται τα αποτελέσματα της λειτουργικής προσομοίωσης. Επίσης όπως φαίνεται και παραπάνω δεν υπάρχουν χρόνοι καθυστέρησης στις μεταβάσεις της εισόδου και της εξόδου. Μέχρι και τα 300 ns τα αποτελέσματα της εξόδου συμφωνούν με τις αναμενόμενες τιμές (-74, 86, -98). Μετά τα 300 ns η έξοδος δεν συμφωνεί με το αναμενόμενο αποτέλεσμα το οποίο είναι (-170), και αυτό συμβαίνει διότι η τιμή της εξόδου δεν μπορεί να απεικονιστεί στα προκαθορισμένα όρια με 8 μπιτ που είναι [-128,127].



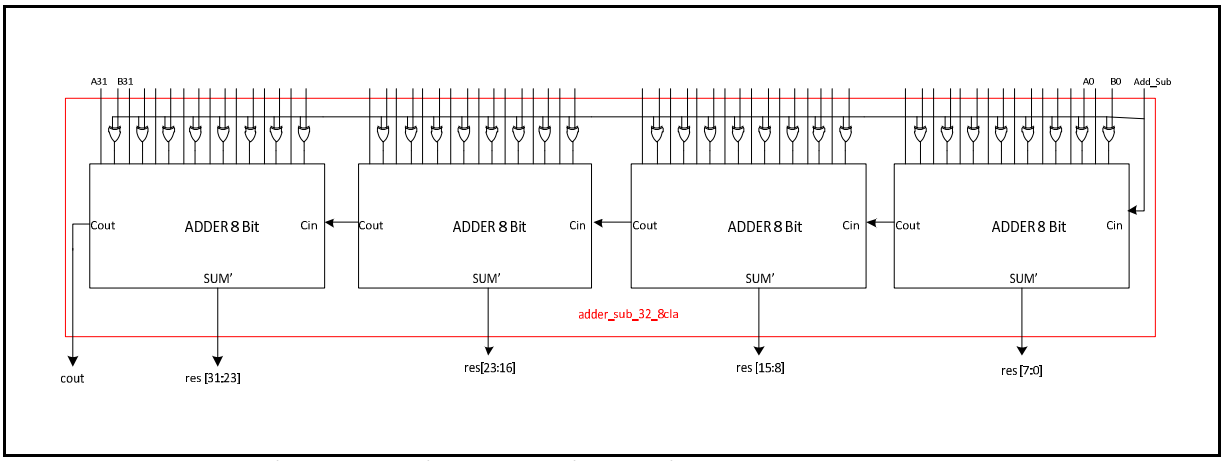

Εικόνα 4.2: Αποτελέσματα προσομοίωσης (Post-Route Simulation) αθροιστή στα 8 μπιτ.

Για να δούμε τους χρόνους καθυστέρησης, θα πρέπει να γίνει προσομοίωση μετά την δρομολόγηση (Post-Route Simulation). Στην Εικόνα 4.2 παρουσιάζονται τα αποτελέσματα της προσομοίωσης αυτής, και βλέπουμε ότι ενώ οι είσοδοι a, b παίρνουν τιμές χωρίς να υπάρχει κάποια καθυστέρηση, η έξοδος χρειάζεται κάποιο χρόνο για να καταλήξει σε κάποια τιμή. Ο χρόνος αυτός για τις τιμές των εισόδων στα 100 ns είναι 9.5 ns, και μπορεί να διαφέρει από σημείο σε σημείο αλλά συνεχίζεται για όσο οι τιμές των εισόδων αλλάζουν. Οι καθυστερήσεις αυτές φαίνονται στην παρακάτω εικόνα που ακολουθεί.



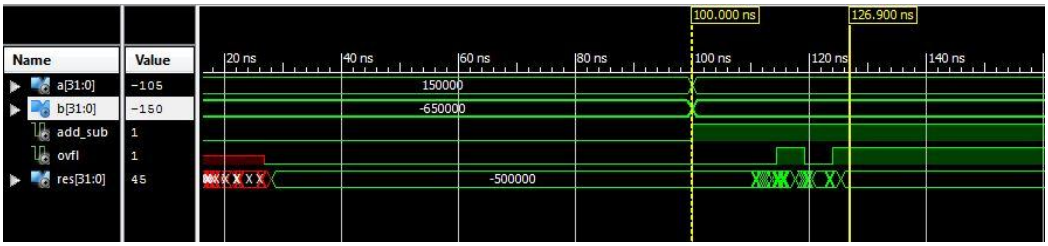

Εικόνα 4.3: Αποτελέσματα προσομοίωσης (Post-Route Simulation) αθροιστή στα 8 μπιτ.

Αφού εξομοιώσαμε τον αθροιστή και σιγουρευτήκαμε για την ορθή του λειτουργία, με την χρήση τεσσάρων τέτοιων αθροιστών φτιάξαμε έναν μεγαλύτερο στα 32 μπιτ. Επίσης βάλαμε πύλες XOR σε όλες τις εισόδους του ενός από τους δυο τελεστέους (στην περίπτωση μας στον τελεστέο B). Η μία είσοδος της κάθε πύλης XOR συνδέεται με την κύρια είσοδο **Add_Sub** ενώ ή άλλη της είσοδος συνδέεται με το αντίστοιχο μπιτ του τελεστέου B. Την τεχνική αυτή την χρησιμοποιήσαμε για να μπορέσουμε με το ίδιο κύκλωμα να υλοποιήσουμε και την πράξη της αφαίρεσης ώστε να μην χρειαστεί να φτιάξουμε μια ξεχωριστή μονάδα για την αφαίρεση. Η είσοδος **Add_Sub** είναι αυτή η οποία καθορίζει την πράξη που θα εκτελεστεί ανάλογα με την τιμή που έχει. Εάν η τιμή είναι 0 τότε θα εκτελεστεί η πράξη της πρόσθεσης, και το κρατούμενο εισόδου της αρχικής μονάδας (πρώτη από τα αριστερά) Adder 8 μπιτ θα είναι 0. Στην περίπτωση που το μπιτ είναι 1 τότε η πύλη XOR θα αντιστρέψει όλα τα μπιτ του τελεστέου B και θα εκτελεστεί η πράξη της αφαίρεσης. Στα σχήματα που ακολουθούν παρουσιάζεται το μπλοκ διάγραμμα και τα αποτελέσματα από την προσομοίωση του αθροιστή στα 32 μπιτ.



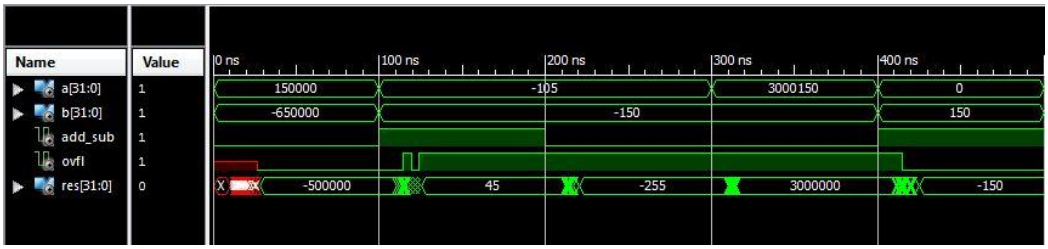

Σχήμα 4.3: Μπλοκ διάγραμμα αθροιστή και αφαιρέτη στα 32 μπιτ.

Όπως φαίνεται και από το Σχήμα 4.3 με το κόκκινο περίγραμμα είναι ο αθροιστής/αφαιρέτης στα 32 μπιτ το οποίο είναι και η κορυφαία μονάδα. Μέσα σε αυτήν όπως θα δείτε υπάρχουν τέσσερις επιμέρους μονάδες των αθροιστών στα 8 μπιτ. Η έξοδος κρατουμένου cout της μονάδας i συνδέεται στην είσοδο κρατουμένου cin της μονάδας i+1 ώστε να μεταφέρεται το κρατούμενο από την μια μονάδα στην άλλη. Για παράδειγμα η έξοδος κρατουμένου της πρώτης μονάδας (πρώτο από τα δεξιά) συνδέεται στην αμέσως επόμενη και ούτω καθεξής.




Εικόνα 4.4: Αποτελέσματα προσομοίωσης (Post-Route Simulation) αθροιστή/αφαιρέτη στα 32 μπιτ.

Η Εικόνα 4.4 δείχνει τα αποτελέσματα προσομοίωσης μετά την δρομολόγηση της συνολικής μονάδας Αθροιστή/Αφαιρέτη στα 32 μπιτ. Εδώ βλέπουμε ότι η καθυστέρηση είναι 26.90 ns. Η καθυστέρηση που φαίνεται είναι πολύ μεγαλύτερη από αυτή του Αθροιστή στα 8 μπιτ. Αναμενόμενο γιατί ο καθένας αθροιστής προσθέτει και κάποιο χρόνο παραπάνω που στο σύνολο προστίθενται. Η μέγιστη καθυστέρηση που μετρήθηκε στον Αθροιστή/Αφαιρέτη είναι 47.560 ns.

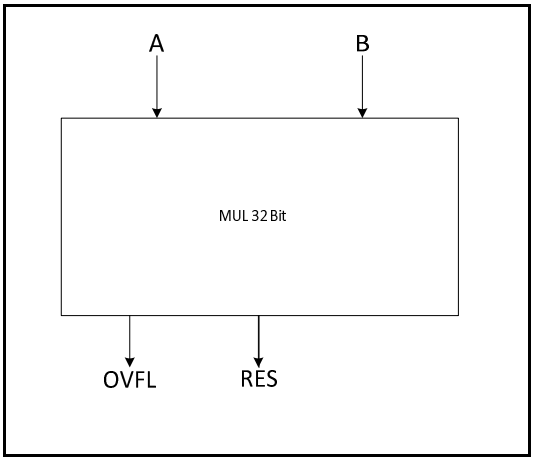



Εικόνα 4.5: Αποτελέσματα προσομοίωσης (Post-Route Simulation) αθροιστή/αφαιρέτη στα 32 μπιτ.

Από την παραπάνω εικόνα φαίνεται η ορθή λειτουργία της τελικής μονάδας adder_sub_328cla.

## 4.2 Πολλαπλασιασμός.

Για την υλοποίηση της μονάδας του πολλαπλασιασμού περιγράψαμε απλώς την διαδικασία η οποία πρέπει να εκτελεστεί, και στην συνέχεια κάναμε έλεγχο για υπερχείλιση του αποτελέσματος. Το μπλοκ διάγραμμα, ο κώδικας και τα αποτελέσματα από την προσομοίωση της μονάδας ακολουθούν παρακάτω.




Σχήμα 4.4: Μπλοκ διάγραμμα πολλαπλασιαστή στα 32 μπιτ.

Η μονάδα αυτή έχει δυο εισόδους και δυο εξόδους, όλες οι είσοδοι και έξοδοι έχουν μήκος 32 μπιτ.

```
signal zeros : STD_LOGIC_VECTOR (31 downto 0);
signal sproduct : STD_LOGIC_VECTOR (63 downto 0);zeros <= (others => '0');
```

Σε πρώτη φάση δημιουργήσαμε δυο εσωτερικά σήματα το zeros και το sproduct. Το ένα έχει μέγεθος 32 μπιτ και το άλλο 64 μπιτ. Το σήμα zeros έχει αρχικοποιηθεί σε τιμή 0 δηλαδή και τα 32 μπιτ του σήματος είναι 0.

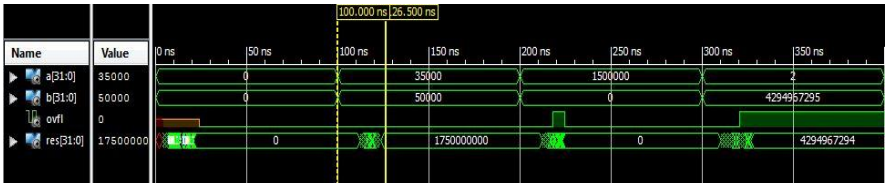
```
sproduct <= A * B;
```

Όπως ξέρουμε από την θεωρία όταν πολλαπλασιάζουμε δυο ακέραιους με n μπιτ μέγεθος το αποτέλεσμα χρειάζεται n+n μπιτ για την απεικόνιση του, για αυτό τον λόγο το σήμα sproduct έχει μέγεθος 64 μπιτ. Από αυτά τα 64 μπιτ όμως τα χρήσιμα είναι τα 32 πρώτα μπιτ, ενώ τα άλλα χρησιμεύουν για τον έλεγχο της υπερχείλισης.

```
process(sproduct)
   begin
      if(sproduct(63 downto 32) = zeros) then
         ovfl <= '0';
      else
         ovfl <= '1';
     end if;
end process;
res <= sproduct(31 downto 0);
```

Ο έλεγχος για την υπερχείλιση γίνεται με μια απλή εντολή συνθήκης. Εάν τα 32 σημαντικότερα μπιτ του sproduct είναι ίσα με το 0 τότε το αποτέλεσμα χώρεσε στα 32

μπιτ, επομένως και το μπιτ της υπερχείλισης θα τεθεί σε λογικό 0. Στην περίπτωση που τα 32 μπιτ αυτά είναι διάφορα του μηδενός τότε έχουμε υπερχείλιση και το μπιτ ovfl θα τεθεί σε λογικό 1, δείχνοντάς μας ότι το αποτέλεσα δεν χώρεσε στα 32 μπιτ. Τέλος, αναθέτουμε στην έξοδο res τα 32 λιγότερα σημαντικά μπιτ από τα 64 που έχει το σήμα sproduct.

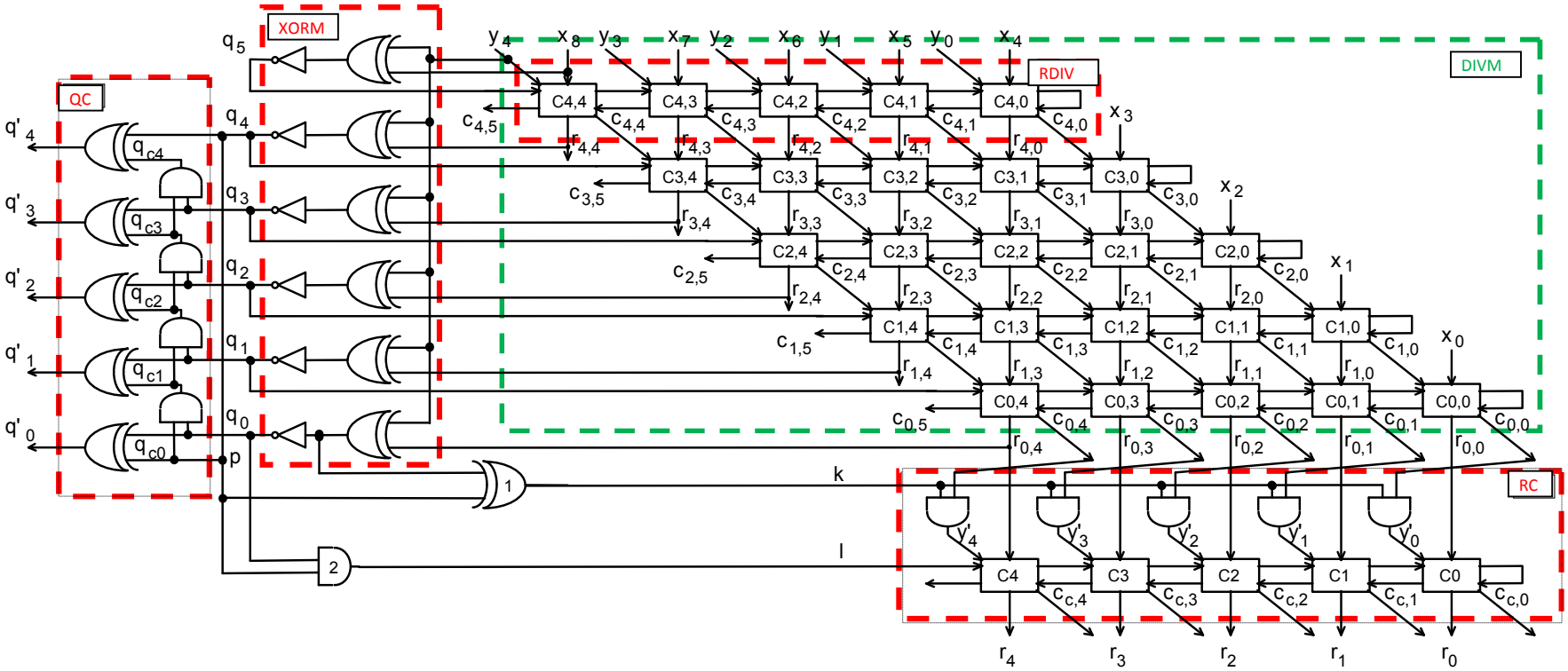



Εικόνα 4.6: Αποτελέσματα προσομοίωσης πολλαπλασιαστή στα 32 μπιτ.

Όπως φαίνεται και από την Εικόνα 4.6 το κύκλωμα του πολλαπλασιασμού λειτουργεί όπως πρέπει. Μέχρι τα πρώτα 300 ns λειτουργεί σωστά. Στα επόμενα 100 ns όμως η τιμή της εξόδου res δεν συμφωνεί με το αναμενόμενο αποτέλεσμα το οποίο είναι 2* 4294967295 = 8589934590, και αυτό συμβαίνει διότι το αποτέλεσμα ξεπερνά το όριο το οποίο μπορεί να αναπαρασταθεί στα 32 μπιτ που είναι $2^N - 1 = 2^{32} - 1 = 4294967295$. Σε αυτή τη περίπτωση το μπιτ ovfl τίθεται σε 1. Η μέγιστη καθυστέρηση που μετρήθηκε είναι 33.07 ns.

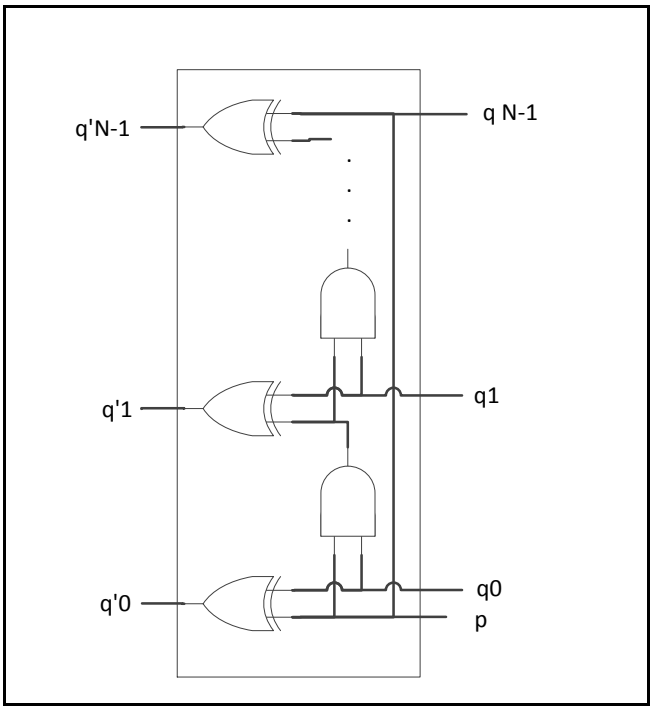
## 4.3 Διαίρεση.

Για την πράξη της διαίρεσης υλοποιήθηκε η παράλληλη έκδοση όπως φαίνεται στο Σχήμα 2.7 που είναι ένα κύκλωμα διαίρεσης δυο τελεστέων στα 5 μπιτ, και βλέπουμε ότι για μια τέτοια διαίρεση χρειαζόμαστε συνολικά 25 κύτταρα και τρείς ακόμα διατάξεις με πύλες που χρησιμεύουν στην διόρθωση υπολοίπου και πηλίκου. Για τον λόγο του ότι οι διατάξεις και τα κύτταρα διαίρεσης απαιτούν πολλές συνδέσεις μεταξύ τους υπάρχει μεγάλη πιθανότητα να γίνει κάποιο λάθος στην σύνδεση, και ακόμη περισσότερα θα είναι τα κύτταρα και το μήκος κάθε διάταξης για μια διαίρεση στα 32 μπιτ συνεπώς και περισσότερες συνδέσεις και ακόμη μεγαλύτερο το ενδεχόμενο να γίνει κάποιο λάθος στην σύνδεση μεταξύ τους. Για αυτό το λόγο χωρίσαμε το κύκλωμα σε πέντε κομμάτια, υλοποιήσαμε το κάθε κομμάτι ξεχωριστά και στο τέλος τα συνδέσαμε μεταξύ τους για να πάρουμε το τελικό αποτέλεσμα. Τα κομμάτια αυτά είναι τα QC, XORM, RDIV, DIVM και RC. Στο Σχήμα 4.7 που ακολουθεί δείχνονται τα κομμάτια και τα συμπεριλαμβανόμενα στοιχεία στο κάθε κομμάτι. Η πράξη η οποία εκτελείται είναι X/Y.



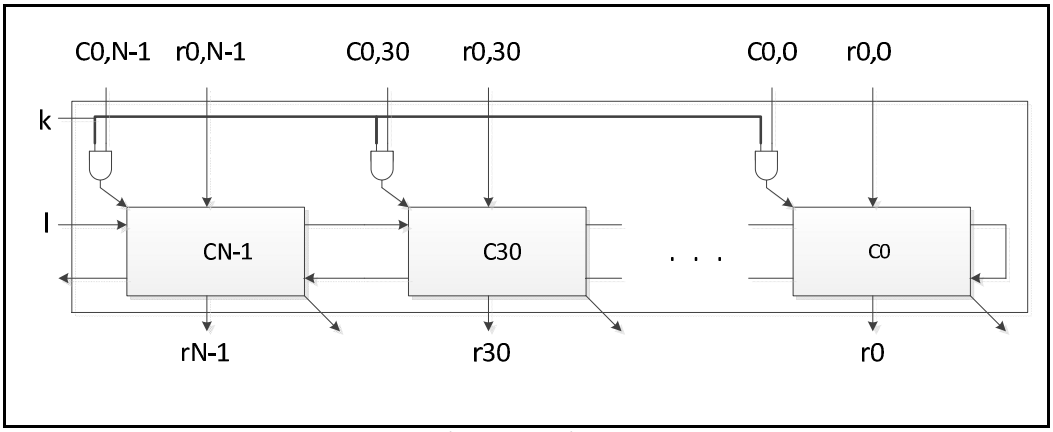

Σχήμα 4.5 : Κύκλωμα διαίρεσης, με τα επιμέρους κομμάτια.

Όπως φαίνεται από το Σχήμα 4.4 το κάθε κομμάτι έχει μέσα αρκετές πύλες διασυνδεμένες κατάλληλα μεταξύ τους. Το κάθε κομμάτι έχει τις απαραίτητες πύλες για δύο τελεστέους στα 5 μπιτ. Η υλοποίηση του κάθε κομματιού δεν σχεδιάστηκε για 5 μπιτ τελεστέους αλλά έγινε με την δυνατότητα άμεσης αλλαγής (παραμετροποιημένα) του μήκους των εισόδων / εξόδων. Παρακάτω ακολουθούν τα σχήματα του κάθε κομματιού.




Σχήμα 4.6 : Κομμάτι QC.

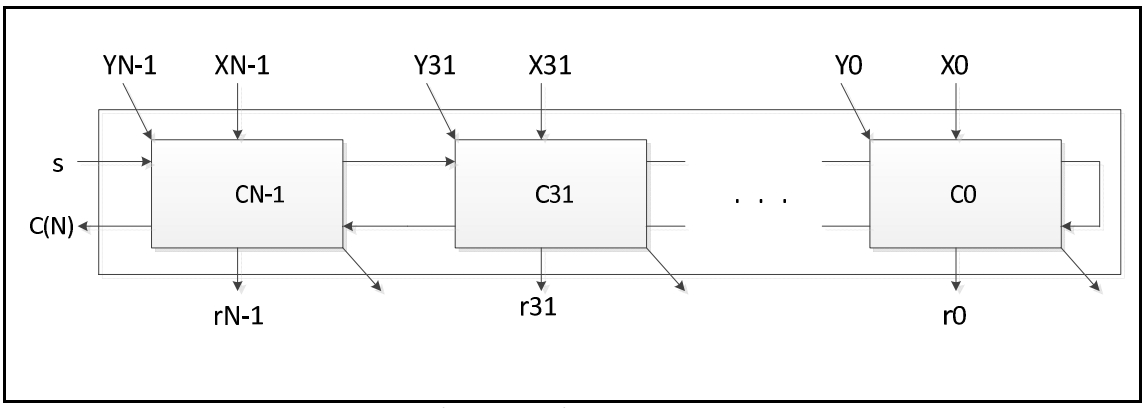
Στο παραπάνω Σχήμα παρουσιάζεται το κύκλωμα που κάνει την διόρθωση του πηλίκου εάν χρειαστεί. Το κύκλωμα αποτελείται από μια διάταξη πυλών AND και XOR που στην ουσία στο σύνολο μπορούμε να πούμε ότι είναι ένας μη-ολοκληρωμένος αθροιστής. Εάν το σήμα εισόδου p είναι 1 τότε θα γίνει πρόσθεση +1 στο τελικό πηλίκο αλλιώς θα παραμείνει ως έχει.




Σχήμα 4.7 : Κομμάτι RC.

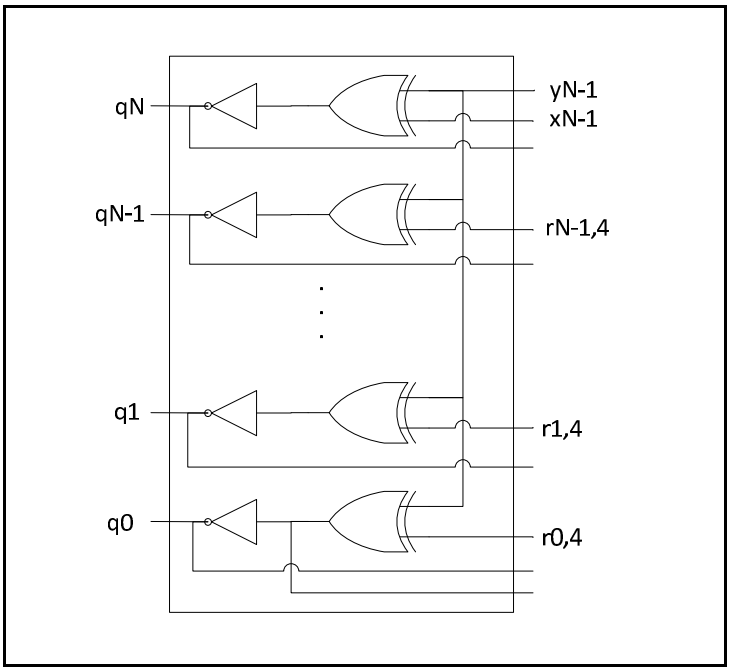
Στο Σχήμα 4.6 παρουσιάζεται το κύκλωμα που κάνει διόρθωση υπολοίπου. Αποτελείται από N-1 κύτταρο διαίρεσης και κάποιες πύλες AND. Το κύκλωμα δέχεται δύο επιπλέον εισόδους

k και l. Ανάλογα με τις τιμές των εισόδων αυτών γίνεται η επιλογή εάν θα γίνει διόρθωση ή όχι. Εάν το σήμα k είναι '1' τότε θα γίνει διόρθωση του τελικού υπόλοιπου, και ανάλογα με την τιμή που θα έχει το σήμα εισόδου l '0' ή '1', θα γίνει διόρθωση με την πρόσθεση +1 στο τελικό κρατούμενο ή με την αφαίρεση -1 στο τελικό κρατούμενο.

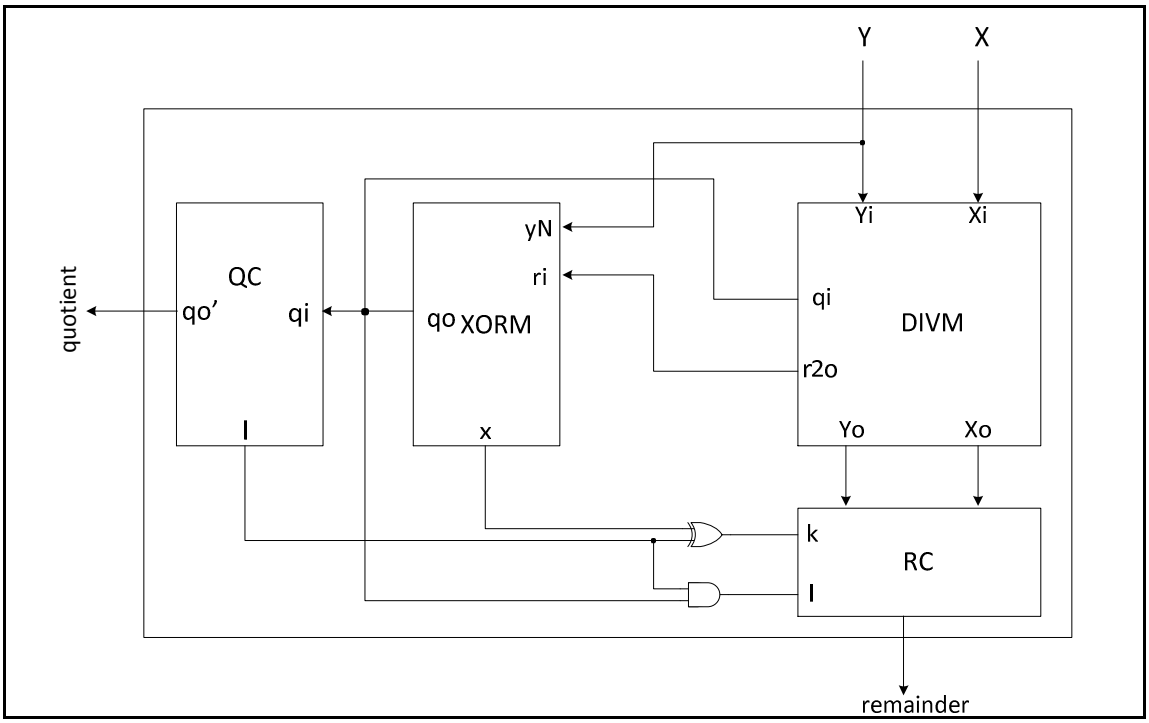



Σχήμα 4.8 : Κομμάτι RDIV.

Στο Σχήμα 4.7 παρουσιάζεται το σχηματικό του RDIV το οποίο είναι μια σειρά από το σύνολο των κυττάρων διαίρεσης που χρειάζεται το τελικό κύκλωμα. Για λόγους διευκόλυνσης στην σχεδίαση επιλέξαμε να πάρουμε μόνο την μια σειρά, να την υλοποιήσουμε και στην συνέχεια με την βοήθεια αυτού να σχεδιάσουμε την επόμενη μονάδα σε ιεραρχία που είναι το DIVM, που περιέχει όλα τα κύτταρα μαζί με τις κατάλληλες συνδέσεις. Το κομμάτι του RDIV περιέχει N-1 κύτταρα διαίρεσης το οποίο θα κάνει πρόσθεση η αφαίρεση του X,Y μπιτ προς μπιτ ανάλογα με την τιμή του σήματος εισόδου s.

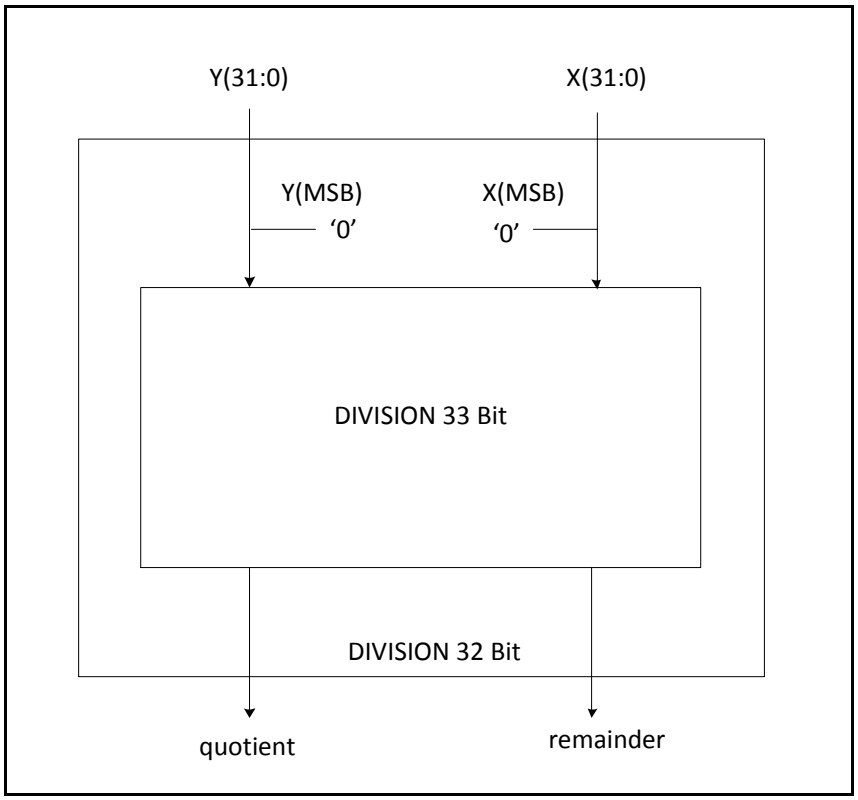



Σχήμα 4.9 : Κομμάτι XORM.



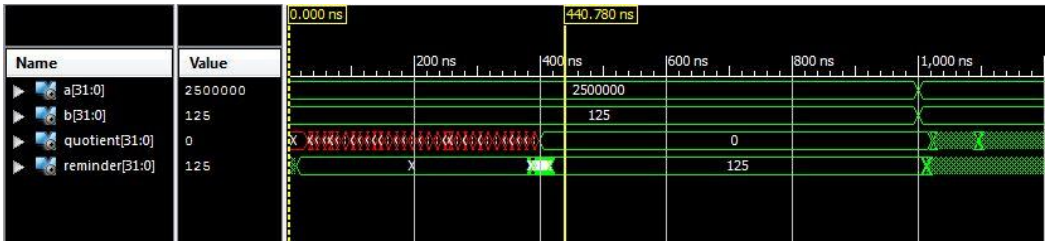

Σχήμα 4.10 : Τελικό σχηματικό διάγραμμα διαιρέτη με όλα τα κομμάτια διασυνδεμένα.

Στο παραπάνω Σχήμα παρουσιάζεται το τελευταίο σε ιεραρχία κύκλωμα της διαίρεσης το οποίο περιέχει, όλα τα επιμέρους κυκλώματα που προαναφερθήκαν στο κεφάλαιο αυτό. Το κύκλωμα αυτό εκτελεί την πράξη της διαίρεσης ανάμεσα σε δύο τελεστέους με πρόσημο.



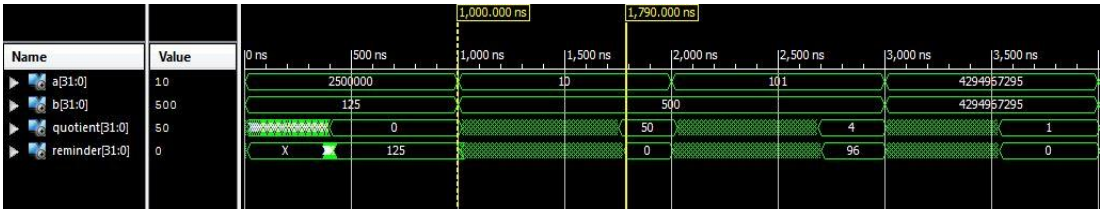

Σχήμα 4.11 : Τελικό διάγραμμα διαιφέτη στα 32 μπιτ.

Για να πετύχουμε την διαίρεση τελεστέων χωρίς πρόσημο και να μην χρειαστεί να αλλάξουμε την συνδεσμολογία των συμπεριλαμβανομένων στοιχείων στο κύκλωμα, υλοποιήσαμε το παραπάνω κύκλωμα στα 33 μπιτ και κάνοντας μικρές αλλαγές σε αυτό πετύχαμε το ζητούμενο αποτέλεσμα. Από την στιγμή που το περισσότερο σημαντικό μπιτ κάθε τελεστέου καθορίζει και το πρόσημό του, θέτουμε μόνιμα σε τιμή '0' το μπιτ αυτό. Έτσι μένουν τα υπόλοιπα δύο 32 μπιτ από κάθε τελεστέο για την εισαγωγή της τιμής και η πράξη γίνεται μόνο με μη προσημασμένους ακέραιους. Ακολουθούν το σχηματικό διάγραμμα και τα αποτελέσματα της προσομοίωσης του κυκλώματος.




Εικόνα 4.7 : Αποτελέσματα προσομοίωσης διαιρέτη στα 32 μπιτ.

Όπως φαίνεται και από την Εικόνα 4.7 το κύκλωμα της διαίρεσης απαιτεί περισσότερο χρόνο για να υλοποιηθεί η πράξη και να έχουμε ένα έγκυρο αποτέλεσμα στην έξοδο. Για τις δεδομένες παραπάνω τιμές ο χρόνος αυτός είναι 440.780 ns. Να σημειωθεί εδώ ότι X = B και Y = A.




Εικόνα 4.8 : Αποτελέσματα προσομοίωσης διαιρέτη στα 32 μπιτ.

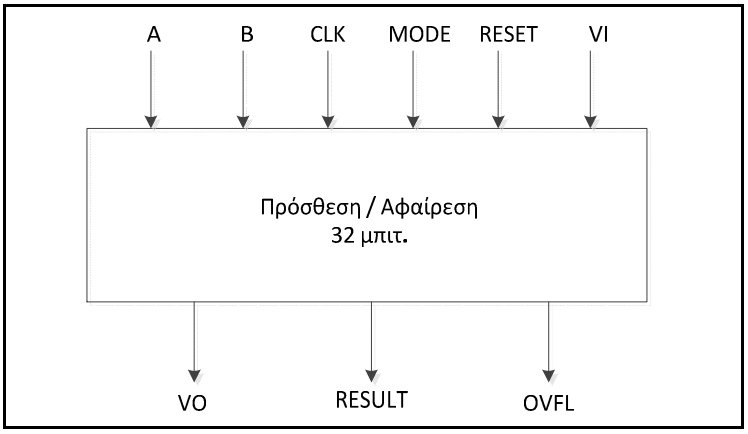
Στην παραπάνω εικόνα φαίνεται ξεκάθαρα η διαφορά χρόνου που χρειάζεται για να εκτελεστεί η πράξη ανάλογα με τις τιμές που δίνουμε. Στο σημείο αυτό η καθυστέρηση είναι 790 ns.

# 5. Υλοποίηση μονάδας πραγματικών αριθμών.

Οι πράξεις που εκτελούνται στην μονάδα αυτή είναι η πρόσθεση, αφαίρεση και ο πολλαπλασιασμός. Για να γίνει μια από τις παραπάνω πράξεις ανάμεσα σε δύο τελεστέους που ανήκουν στους πραγματικούς αριθμούς, θα πρέπει οι αριθμοί αυτοί να δοθούν κωδικοποιημένοι σύμφωνα με το πρότυπο IEEE 754 σε απλή ακρίβεια.

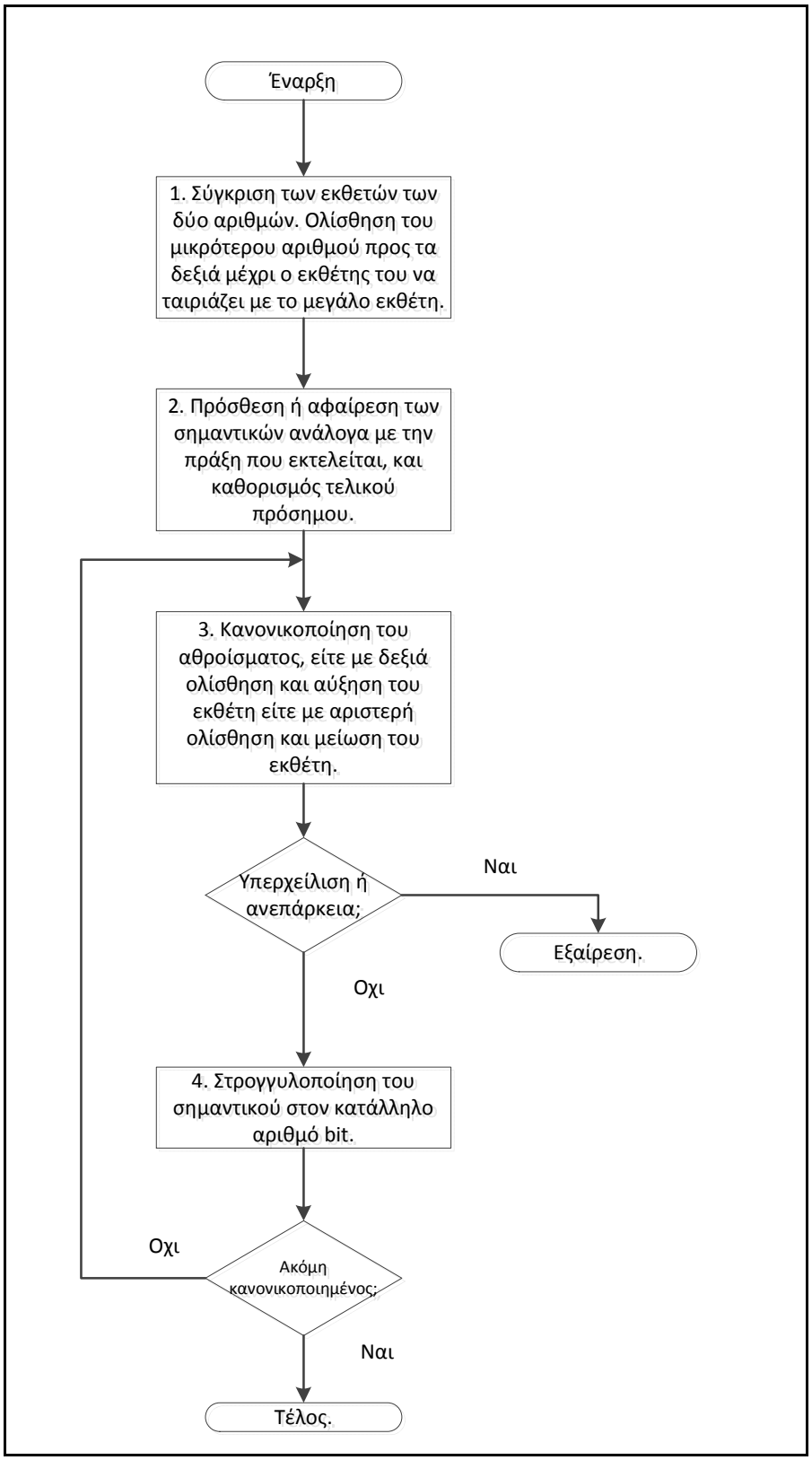
## 5.1 Πρόσθεση / Αφαίρεση.

Για την πρόσθεση και την αφαίρεση χρησιμοποιήθηκε μια μονάδα με είσοδο επιλογής που ανάλογα με την τιμή της γίνεται πρόσθεση ή αφαίρεση των τελεστέων. Το κύκλωμα της πρόσθεσης υλοποιήθηκε με βάση το παρακάτω διάγραμμα ροής που ακολουθεί, ενώ για να υλοποιήσουμε και την αφαίρεση στο ίδιο κύκλωμα το μόνο που κάναμε ήταν να μετατρέψουμε τον έναν από τους δύο τελεστέους σε θετικό στην περίπτωση που ήταν αρνητικός και κάναμε πρόσθεση αυτών των δυο. Εμείς επιλέξαμε να ελέγχουμε τον τελεστέο B αν η πράξη που χρειάζεται να εκτελεστεί είναι η αφαίρεση του αλλάζουμε το πρόσημο. Το βασικό κύκλωμα της πρόσθεσης δεν υλοποιήθηκε ιεραρχικά, δηλαδή δεν σχεδιάστηκαν τα κομμάτια ένα-ένα και τέλος μια τελική μονάδα σε ιεραρχία που τα σύνδεε όλα μαζί, αλλά υλοποιήθηκε με την βοήθεια μηχανής καταστάσεων. Με λίγα λόγια αυτό που κάναμε ήταν να περιγράψουμε την λειτουργία του κυκλώματος σύμφωνα με το διάγραμμα ροής. Η μηχανή καταστάσεων που χρησιμοποιήθηκε είναι τύπου Moore. Οι έξοδοι αυτής της μηχανής εξαρτώνται αποκλειστικά από την παρούσα κατάσταση. Παρακάτω ακολουθούν το σχηματικό διάγραμμα, το διάγραμμα ροής και το διάγραμμα καταστάσεων του κυκλώματος.

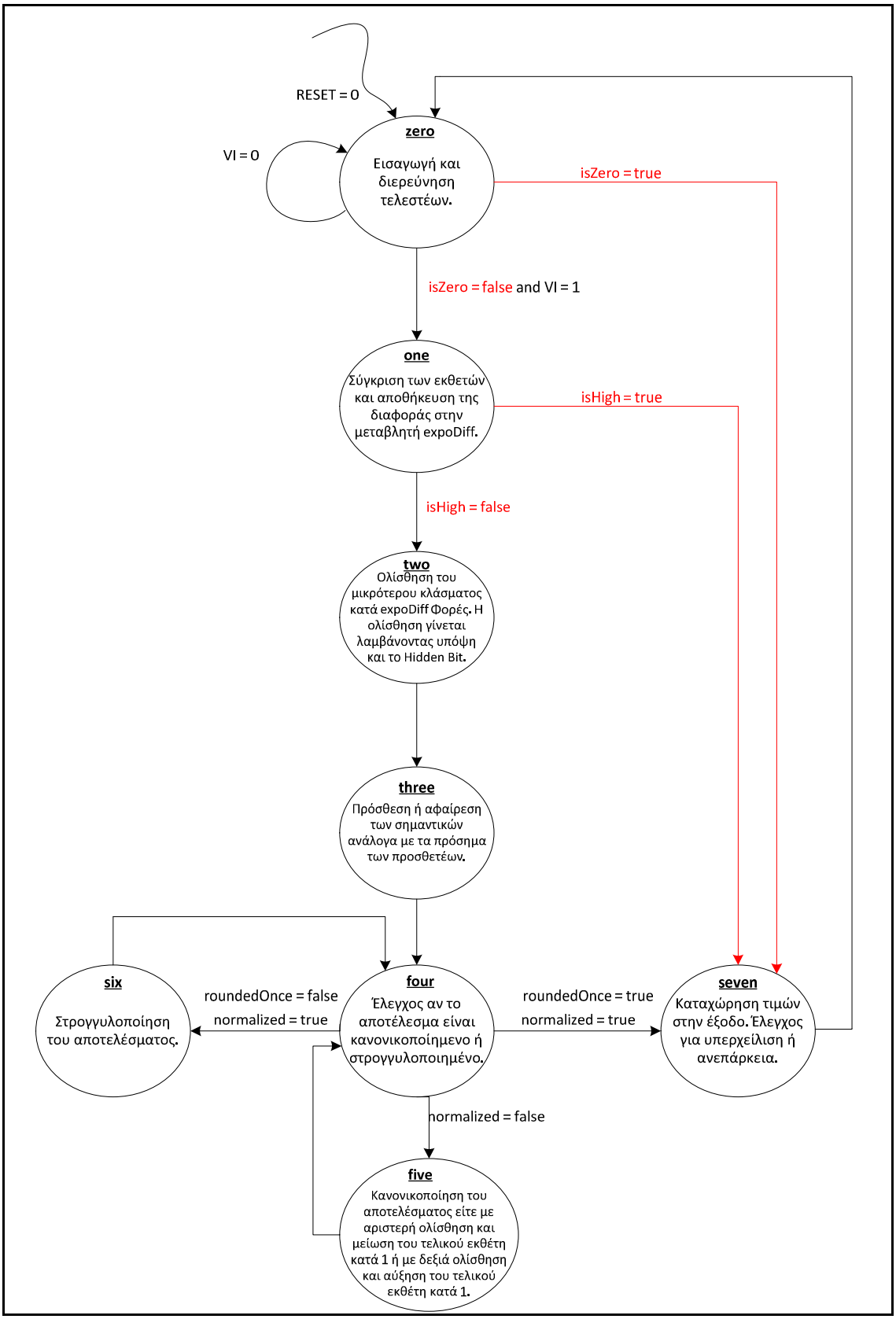



Σχήμα 5.1 : Σχηματικό διάγραμμα πρόσθεσης / αφαίρεσης στα 32 μπιτ.

Το κύκλωμα λοιπόν έχει έξι εισόδους και τρείς εξόδους όπως φαίνεται από το παραπάνω σχηματικό διάγραμμα, εκ των οποίων οι είσοδοι A, B και η έξοδος RESULT έχουν μήκος 32 μπιτ, ενώ οι υπόλοιπες είσοδοι και έξοδοι είναι ένα μπιτ η κάθε μια. Ανάλογα με την τιμή της εισόδου MODE, η πράξη που εκτελείται είναι πρόσθεση ή αφαίρεση. Όταν είναι '1' γίνεται αφαίρεση των τελεστών (A-B) και όταν είναι '0' γίνεται πρόσθεση. Για όσο χρόνο η μηχανή επεξεργάζεται τα δεδομένα που δόθηκαν η έξοδος παίρνει διάφορες τιμές. Για να πάρουμε μια έγκυρη έξοδο η έξοδος VO πρέπει να είναι '1', για όσο είναι η έξοδος αυτή στο '0' σημαίνει ότι το αποτέλεσμα της εξόδου δεν είναι σωστό. Για παρόμοιους λόγους όταν η είσοδος VI είναι '1', αυτό σημαίνει ότι τα δεδομένα που δόθηκαν είναι έγκυρα και θα εκτελεστεί ή αντίστοιχη πράξη που είναι να γίνει.

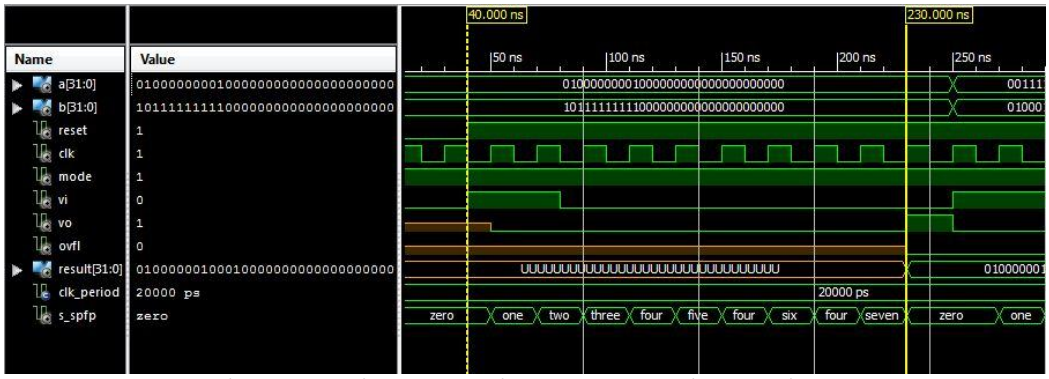



Σχήμα 5.2 : Διάγραμμα ροής πρόσθεσης.



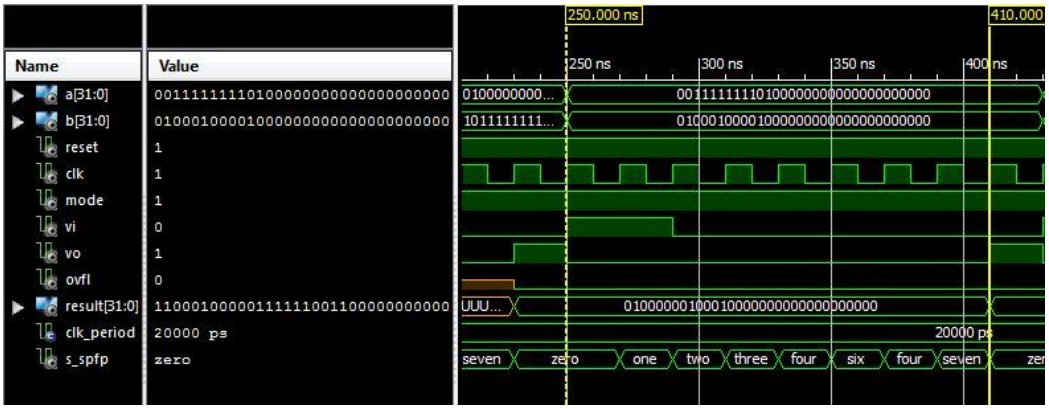

Σχήμα 5.3 : Διάγραμμα καταστάσεων πρόσθεσης.

Όπως φαίνεται και από το παραπάνω Σχήμα, για να υλοποιηθεί το κύκλωμα της αφαίρεσης και της πρόσθεσης χρειάστηκαν επτά διαφορετικές καταστάσεις (one, two, three, four, five, six, seven). Μετά τις μαύρες γραμμές είναι η μετάβαση που κάνει η μια κατάσταση σε μια άλλη με κάθε παλμό ρολογιού, ενώ με κόκκινα είναι οι μεταβάσεις της βελτιστοποίησης που έγινε στον αλγόριθμο. Οι βελτιστοποιήσεις που γίνανε στο κύκλωμα είναι οι εξής : Ο έλεγχος για την πρώτη βελτιστοποίηση γίνεται στην πρώτη κατάσταση (one), ελέγχονται οι δυο τελεστές και αν ο ένας από τους δυο έχει την τιμή 0 τότε η μηχανή μεταβαίνει κατευθείαν στην τελευταία κατάσταση στην (seven) αφού πρώτα η κατάσταση (one) φροντίσει να προετοιμάσει κατάλληλα τις μεταβλητές που αποθηκεύονται οι τιμές των εξόδων. Ο δεύτερος έλεγχος για την βελτιστοποίηση γίνεται στην κατάσταση (two) όπου εκεί ελέγχονται οι δυο τελεστέοι και εάν ο ένας από τους δυο είναι πολύ μεγαλύτερος από τον άλλο, τότε σε αυτή τη περίπτωση δεν έχει νόημα να εκτελεστεί η πράξη διότι στην απλή ακρίβεια η διαφορά δεν θα φανεί και για τον λόγο αυτό η μηχανή μεταβαίνει και πάλι στην τελευταία κατάσταση αφού πρώτα προετοιμαστούν κατάλληλα οι μεταβλητές εξόδου. Να σημειωθεί εδώ πέρα ότι μια πρόσθεση ή αφαίρεση μπορεί να πάρει διαφορετικό χρόνο στην εκτέλεση της πράξης. Ακολουθούν εικόνες από τα αποτελέσματα της προσομοίωσης.



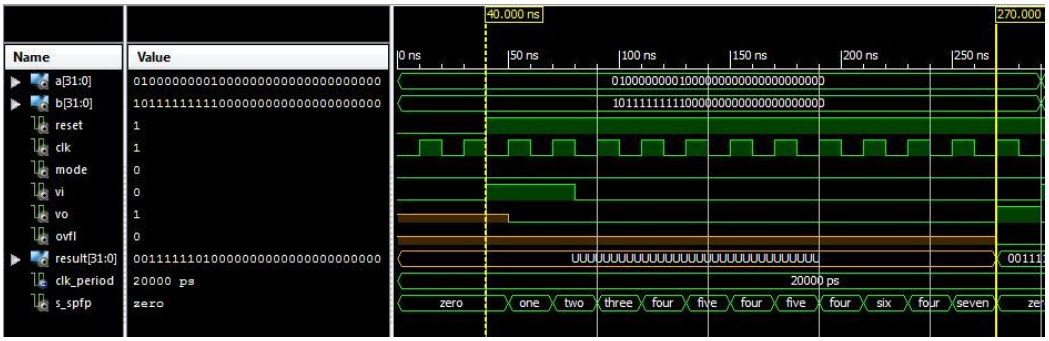

Εικόνα 5.1 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος.

Στην Εικόνα 5.1 βλέπουμε τα αποτελέσματα τις από την προσομοίωση τις μονάδας. Οι δυο τελεστές έχουν τιμές **a = 2.5**, **b = -1.75** και η πράξη που εκτελείτε ανάμεσα στους δυο τελεστές είναι αφαίρεση αφού η είσοδος **mode** είναι '1'. Όπως δείχνει και η προσομοίωση για όσο η είσοδος **VI** (**0 έως 40 ns**) είναι ίσο με το '0' η μηχανή δεν λαμβάνει υπόψη τις τιμές των τελεστών και το σήμα **spfp** το οποίο μας δείχνει σε ποια κατάσταση είναι η μηχανή είναι μόνιμα στην κατάσταση **zero**. Όταν τώρα η είσοδος αυτή γίνει ίσο με το λογικό '1' (**50 ns**) τότε η μηχανή ξεκινά να επεξεργάζεται τα δεδομένα τις εισόδου. Κατά την επεξεργασία των δεδομένων η έξοδος **VI** είναι '0', όταν η έξοδος αυτή μεταβεί στην τιμή '1' (**230 ns**) τότε η μηχανή έχει τελειώσει την επεξεργασία και μπορούμε να πάρουμε τα δεδομένα τις εξόδου που είναι η αφαίρεση των τελεστών και έχει αποτέλεσμα **result = 4.25**. Από την παραπάνω εικόνα βλέπουμε ότι για τις συγκεκριμένες τιμές εισόδου, χρειάστηκε να γίνει κανονικοποίηση μόνο μια φορά. Αυτό φαίνεται από τις μεταβάσεις καταστάσεων του εσωτερικού σήματος **spfp** και βλέπουμε ότι μεταβαίνει μόνο μια φορά την κατάσταση **"five"** στην οποία και γίνεται η κανονικοποίηση. Παρακάτω θα δούμε ότι δια διαφορετικές τιμές των εισόδων η μηχανή μπορεί να μεταβεί πιο πολλές φορές από μια στην κατάσταση αυτή ή και σε ορισμένες περιπτώσεις καθόλου.




Εικόνα 5.2 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος.

Στην περίπτωση αυτή για τιμές των **a = 1.625** και **b = 640.0** φαίνεται ξεκάθαρα από τις καταστάσεις στις οποίες μεταβαίνει το εσωτερικό σήμα **spfp**, ότι ο αριθμός δεν χρειάζεται να κανονικοποιηθεί συνεπώς και δεν μεταβαίνει στην κατάσταση **five**. Η πράξη που εκτελείται είναι αφαίρεση και το αποτέλεσμα είναι **result = 638.375**.




Εικόνα 5.3 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος.

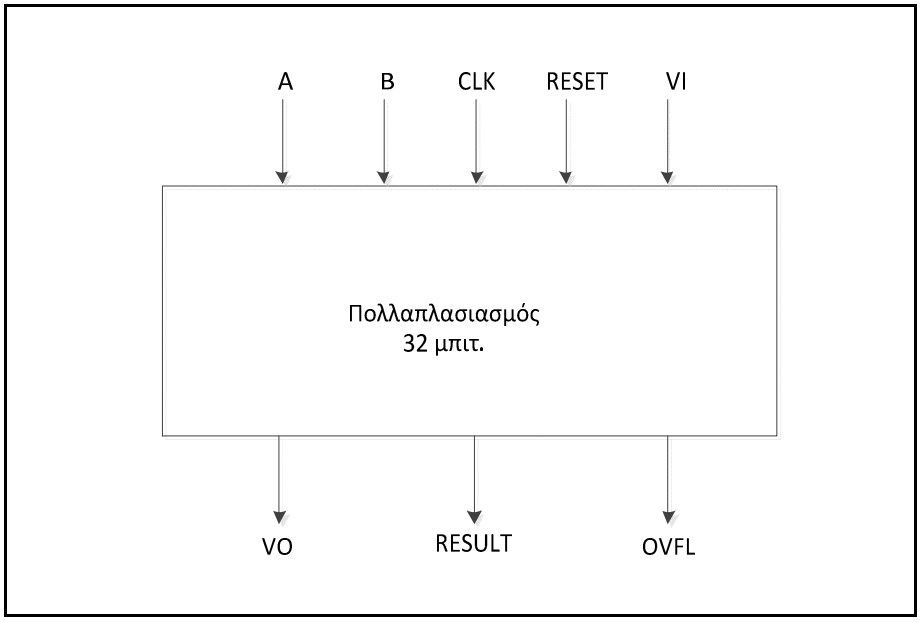
Εδώ πέρα όπως φαίνεται και από την παραπάνω εικόνα, η πράξη που εκτελείται είναι πρόσθεση εφόσον η είσοδος **mode** είναι ‘0’. Η πρόσθεση των δυο τελεστέων με τιμές **a = 2.5** και **b = -1.75** θα δώσει ένα αποτέλεσμα **result = 0.75**.


Εικόνα 5.4 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος.

Εδώ οι τιμές είναι **a = 0.00625**, **b = 0.025** και **result = 0.03125**.

## 5.2 Πολλαπλασιασμός.

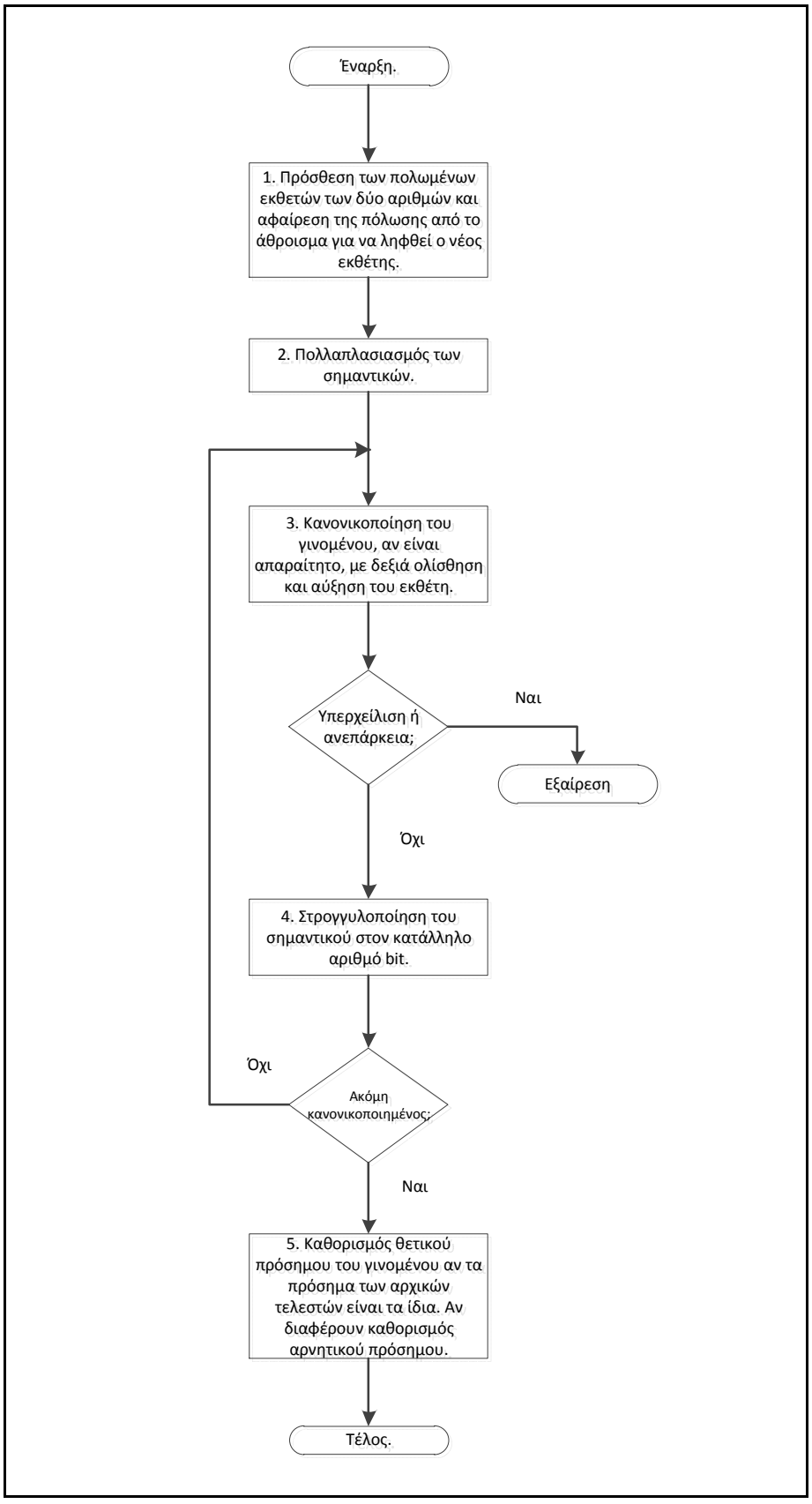
Ο πολλαπλασιασμός υλοποιήθηκε ακριβώς με την ίδια τεχνική με αυτή της πρόσθεσης/αφαίρεσης, δηλαδή το κύκλωμα και αυτό υλοποιήθηκε με την χρήση της μηχανής καταστάσεων και η μηχανή καταστάσεων που χρησιμοποιήθηκε και εδώ είναι τύπου Moore. Το μπλοκ διάγραμμα του κυκλώματος πολλαπλασιασμού ακολουθεί παρακάτω.



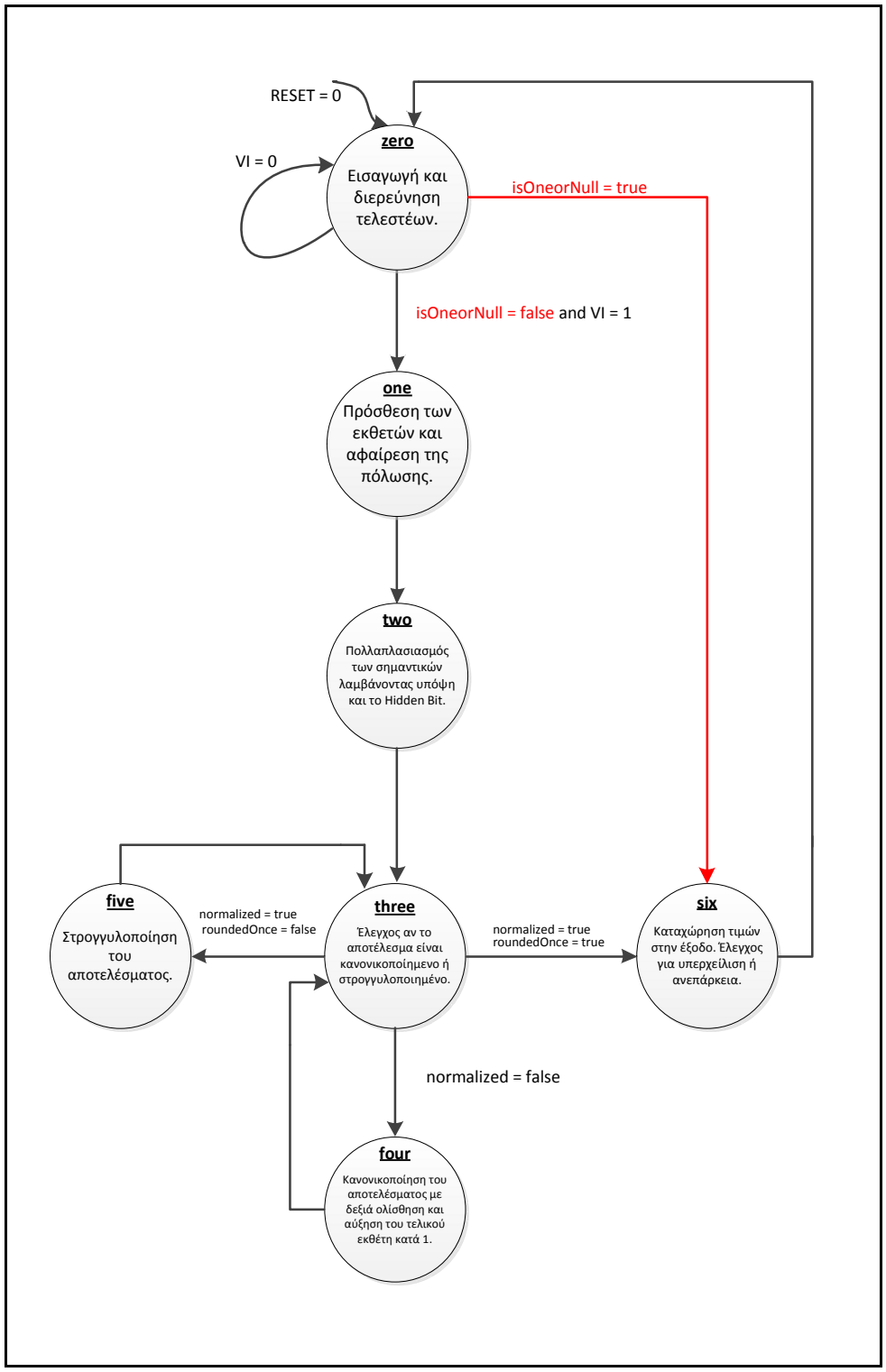

Σχήμα 5.4 : Σχηματικό διάγραμμα πολλαπλασιασμού στα 32 μπιτ.

Το σχηματικό διάγραμμα του πολλαπλασιαστή έχει πέντε εισόδους και τρείς εξόδους. Οι ακροδέκτες του κυκλώματος, έχουν ακριβώς τον ίδιο ρόλο και το ίδιο μέγεθος με αυτό της πρόσθεσης στα 32 μπιτ.

Στο Σχήμα 5.5 παρουσιάζεται το διάγραμμα ροής, με βάση το οποίο έχει υλοποιηθεί και η πράξη αυτή. Η πράξη αυτή ένα βήμα παραπάνω στην υλοποίηση του, από αυτό της πρόσθεσης.



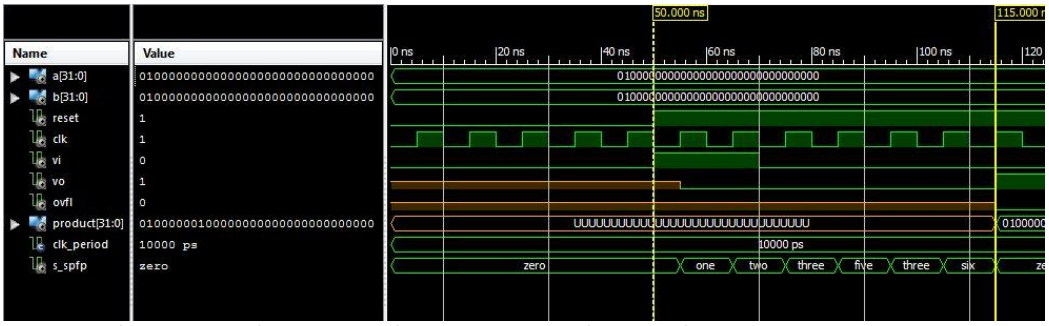

Σχήμα 5.5 : Διάγραμμα ροής πολλαπλασιασμού στα 32 μπιτ.



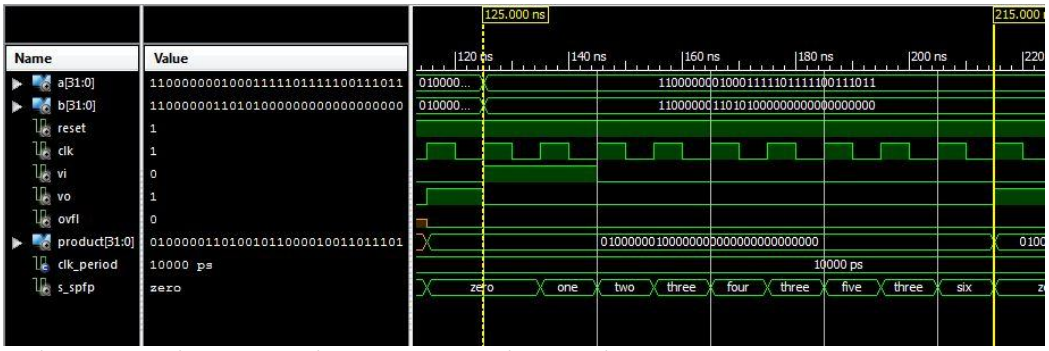

Σχήμα 5.6 : Διάγραμμα καταστάσεων πολλαπλασιασμού στα 32 μπιτ.

Όπως φαίνεται και από το παραπάνω σχήμα, η μηχανή που υλοποιεί την πράξη της διαίρεσης έχει μόνο έξι καταστάσεις (μια λιγότερη από αυτή της πρόσθεσης) παρόλο που στο διάγραμμα ροής έχουμε μια κίνηση παραπάνω. Η βελτιστοποίηση που έγινε στην πράξη αυτή είναι, εάν ο ένας από τους δύο αριθμούς είναι 0.0 τότε η μηχανή μεταβαίνει

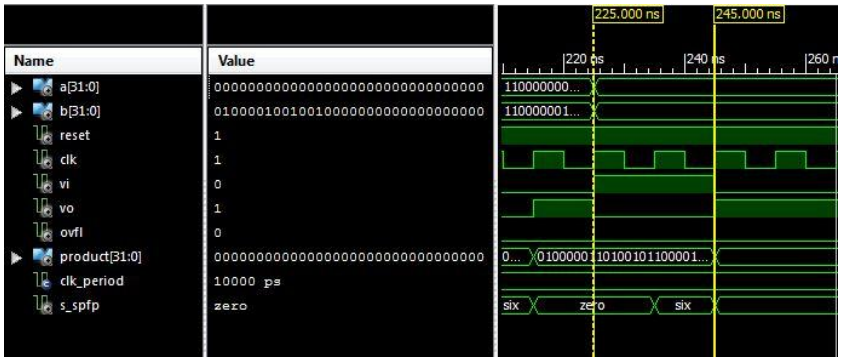
απευθείας στην τελευταία κατάσταση που στην περίπτωση του πολλαπλασιασμού είναι η κατάσταση (six) αφού πρώτα η κατάσταση (zero) στην οποία και γίνεται ο έλεγχος των δύο αριθμών, φροντίσει να προετοιμάσει κατάλληλα τις μεταβλητές που αποθηκεύονται οι τιμές των εξόδων.




Εικόνα 5.5 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος. **a, b = 2.0** και **result = 4.0**.




Εικόνα 5.6 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος. **a = -3.123, b = -6.625** και **result = 20.689875**.




Εικόνα 5.7 : Αποτελέσματα προσομοίωσης της συμπεριφοράς του κυκλώματος.

Στην περίπτωση της παραπάνω εικόνας, φαίνεται ξεκάθαρα η περίπτωση της βελτιστοποίησης. Οι τιμές που έχουν οι τελεστές είναι **a = 0.0** και **b = 50.0** και το **result = 0.0.** Επειδή λοιπόν ο a είναι ίσος με το 0.0, η μηχανή μεταβαίνει στην τελευταία κατάσταση όπως φαίνεται και από την εικόνα 5.7.

# 6. Συμπεράσματα.

Αφού πρώτα μελετήσαμε σε βάθος τα σχέδια και τα διαγράμματα ροής από το κάθε ψηφιακό κύκλωμα, τα υλοποιήσαμε στο Xilinx ISE. Μετά την υλοποίηση των κυκλωμάτων προβήκαμε στην προσομοίωση των κυκλωμάτων για να επιβεβαιώσουμε την ορθή τους λειτουργία, καταγράφοντας τους μετρούμενους χρόνους διάδοσης που θέλει το κάθε κύκλωμα. Ο χρόνος διάδοσης στα κυκλώματα που δεν έχουν ρολόι μάς δίνει την δυνατότητα να προσδιορίσουμε τον μέγιστο ρυθμό με τον οποίο θα εισάγουμε δεδομένα στις εισόδους των κυκλωμάτων, για να πάρουμε στην έξοδο το σωστό αποτέλεσμα. Αρχικά το κύκλωμα της πρόσθεσης των ακεραίων έγινε με δύο τρόπους. Η πρώτη μέθοδος είναι η πρόσθεση ριπής κρατουμένου και η δεύτερη είναι η πρόσθεση πρόβλεψης κρατουμένου. Δεν το υλοποιήσαμε απευθείας στα 32 μπιτ, γιατί μια τέτοια σχεδίαση με την μέθοδο πρόβλεψης κρατουμένου θα είχε μεγάλο κόστος, και για τον λόγο αυτό το υλοποιήσαμε στα 8 μπιτ, τα συγκρίναμε και στην συνέχεια με τέσσερα τέτοια υλοποιήσαμε το τελικό στα 32 μπιτ όπως αναφέρεται και στο κεφάλαιο 4. Τα κυκλώματα που έχουν ρολόι, στην περίπτωσή μας η πρόσθεση, η αφαίρεση και ο πολλαπλασιασμός, έχουν σήματα ελέγχου τα οποία καθορίζουν πότε θα πάρουν τις τιμές της εισόδου και πότε είναι διαθέσιμο το σωστό αποτέλεσμα (VI και VO). Ο χρόνος σε αυτά τα κυκλώματα διαφέρει ανάλογα με τις τιμές που δίνουμε στις εισόδους τους, όπως αναφέρεται και στο κεφάλαιο 5.

Από τα αποτελέσματα που καταγράψαμε στις μετρούμενες τιμές για τον αθροιστή στα 32 μπιτ με την μέθοδο πρόβλεψης κρατουμένου και με την μέθοδο ριπής κρατουμένου, οι χρόνοι διάδοσης του αθροιστή ριπής κρατουμένου ήταν μικρότερη από αυτή της πρόβλεψης κρατουμένου. Για να δούμε γιατί συμβαίνει αυτό αναγκαστήκαμε να υλοποιήσουμε και κάποιες παραλλαγές των αθροιστών, να τα προσομοιώσουμε, να καταγράψουμε τους χρόνους διάδοσης και στην συνέχεια να βγάλουμε κάποια χρήσιμα συμπεράσματα. Οι χρόνοι που καταγράψαμε παρουσιάζονται στον επόμενο πίνακα που ακλουθεί.

| **Ψηφιακό Κύκλωμα** | **Spartan 3 xc3s1000-5ft256** | **Virtex-7 xc7vx330t-2ffg1157** |
|---|---|---|
| Αθροιστής 4 μπιτ αποτελούμενος από 4 πλήρεις αθροιστές. | 9.575 ns | 5.460 ns |
| Αθροιστής 4 μπιτ, με πρόβλεψη κρατουμένου. | 9.396 ns | 5.330 ns |
| Αθροιστής 8 μπιτ αποτελούμενος από 8 πλήρεις αθροιστές. | 14.052 ns | 7.311 ns |
| Αθροιστής 8 μπιτ με, πρόβλεψη κρατουμένου. | 14.255 ns | 8.234 ns |
| Αθροιστής 32 μπιτ αποτελούμενος από 8 αθροιστές των 4 μπιτ, με πλήρεις αθροιστές. | 40.607 ns | 13.641 ns |
| Αθροιστής 32 μπιτ αποτελούμενος από 8αθροιστές των 4 μπιτ, με πρόβλεψη κρατουμένου. | 38.115 ns | 12.285 ns |
| Αθροιστής 32 μπιτ αποτελούμενος από 4 αθροιστές των 8 μπιτ, με πλήρης αθροιστές. | 38.972 ns | 17.105 ns |
| Αθροιστής 32 μπιτ αποτελούμενος από 4 αθροιστές των 8 μπιτ, με πρόβλεψη κρατουμένου. | 39.522 ns | 18.832 ns |

Πίνακας 6.1 : Συγκριτικός πίνακας χρόνων διάδοσης ανά κύκλωμα για δύο FPGA από διαφορετικές οικογένειες.

Από τα αποτελέσματα που παρουσιάζονται στον παραπάνω πίνακα, βλέπουμε ότι η μέθοδος της πρόβλεψης κρατουμένου στα 8 μπιτ χρειάζεται περισσότερο χρόνο διάδοσης από αυτόν της πρόσθεσης ριπής κρατουμένου, και αυτό οφείλεται στην εσωτερική συνάρτηση που έχει η μονάδα, όπου στα 8 μπιτ είναι αρκετά μεγάλη ($C_8 = G_7 + P_7 \cdot (G_6 + P_6 \cdot$

$(G_5 + P_5 \cdot (G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)))))))$. Αυτό σημαίνει ότι για να υλοποιηθεί η συνάρτηση αυτή εσωτερικά στο FPGA, χρειάζεται πύλες με πολλές εισόδους, συνεπώς η πύλη αυτή θα χρειαστεί και περισσότερο χρόνο διάδοσης. Οπότε καταλήγουμε στο συμπέρασμα ότι εάν υλοποιούσαμε έναν επεξεργαστή στα 32 μπιτ σε ένα αναπτυξιακό με πολύ γρήγορο FPGA για κάποια συγκεκριμένη διαδικασία, θα επιλέγαμε την υλοποίηση με την μέθοδο πρόβλεψης κρατουμένου με 8 αθροιστές των 4 μπιτ ο καθένας.

Η απόδοση στα κυκλώματα μπορεί πολύ εύκολα να αυξηθεί εάν χρησιμοποιηθεί ένα FPGA από μια ισχυρότερη οικογένεια, όπως η Virtex 7, οπότε θα μπορούσε να λειτουργεί σε μεγαλύτερη συχνότητα, και να έχει εσωτερικές πύλες με χρόνο διάδοσης πολύ μικρότερο από το FPGA που διαθέτουμε το οποίο ανήκει στην οικογένεια Spartan 3.

Σε μια ψηφιακή σχεδίαση που υλοποιείται με την χρήση της VHDL καλό είναι πρώτα να σχεδιάζεται στο χαρτί αυτό που επρόκειτο να περαστεί στο περιβάλλον ανάπτυξης, να δώσουμε ονομασίες στα επιμέρους κομμάτια, να ονομάσουμε επίσης όλες τις ενώσεις μεταξύ των διάφορων κομματιών και να καλό είναι επίσης όλα τα κομμάτια και τα σήματα που υπάρχουν στην σχεδίαση στο χαρτί, να περαστούν με την ίδια ονομασία και στο περιβάλλον ανάπτυξης, ώστε να ελαχιστοποιηθεί το ενδεχόμενο λάθους και στην περίπτωση του λάθους να γίνει εύκολα ο εντοπισμός και η διόρθωσή του.

Η υλοποίηση ακεραίων και πραγματικών έχει γίνει στα 32 μπιτ. Δεν είναι δύσκολο να γίνει μια παραλλαγή η μια βελτιστοποίηση θα μπορούσε να πει κανείς, ώστε να μπορούν να εκτελεστούν πράξεις στα 64 μπιτ. Η διαφορά στο τελικό αποτέλεσμα θα είναι ο χρόνος που θα χρειαζόταν η πράξη στα 64 μπιτ, το μέγεθος του τελικού σχεδίου και οι πόροι που θα κατανάλωνε.

Στις προσομοιώσεις των κυκλωμάτων που υλοποιήσαμε, θέσαμε κάποιους χρόνους (περιόδους ρολογιού) για να γίνει σωστά η προσομοίωση του κάθε κυκλώματος. Για να επιλέξουμε τις κατάλληλες τιμές για τους χρόνους αυτούς συμβουλευτήκαμε ένα από τα εργαλεία που είναι διαθέσιμα στο περιβάλλον ανάπτυξης ISE Design Suite 14.7, το «Σύνοψη της Σχεδίασης/Αναφορές» (Design Summary/Reports).

## Βιβλιογραφία.

[1] D. Patterson & J. Hennessy, «ΟΡΓΑΝΩΣΗ ΚΑΙ ΣΧΕΔΙΑΣΗ ΥΠΟΛΟΓΙΣΤΩΝ», μετάφραση και επιστ. επιμέλεια Δ. Γκιζόπουλου, $4^{η}$ ΕΚΔΟΣΗ, ΤΟΜΟΣ Α', Κλειδάριθμος, 2010.

[2] Πετράκης Νικόλαος, «Αύξηση της Αξιοπιστίας και της Διαθεσιμότητας των Συστημάτων Υπολογισμού με την Αποτελεσματική Χρήση του Αυτοελέγχου», *Διδακτορική διατριβή*, Τιμισοάρα, Ιούλιος 1995. (στην Ρουμανική)

[3] M. Mano, «Ψηφιακή Σχεδίαση», μετάφραση και επιστ. επιμέλεια Απ. Τραγανίτη, $2^{η}$ ΕΚΔΟΣΗ, Παπασωτηρίου, 1992.

Ιστότοποι :

[4] http://en.wikipedia.org/wiki/Division_algorithm

[5] http://en.wikipedia.org/wiki/Carry-lookahead_adder

[6] http://en.wikipedia.org/wiki/IEEE_floating_point

# Παράρτημα Α: Αποσπάσματα κώδικα σε VHDL

--Πρόσθεση κινητής υποδιαστολής

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;

entity adder_32_fp is
  Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
    B : in  STD_LOGIC_VECTOR (31 downto 0);
    CLK : in STD_LOGIC;
    RESET : in STD_LOGIC;
    VI : in STD_LOGIC;
    VO : out STD_LOGIC;
    OVFL : out STD_LOGIC;
    SUM : out STD_LOGIC_VECTOR (31 downto 0));
end adder_32_fp;

architecture Behavioral of adder_32_fp is
--Δημιουργία διάφορων signal με εμβέλεια σε όλο το πρόγραμμα  (source file).
type FP_STATES is (zero, one, two, three, four, five, six, seven);
signal signifA, signifB : STD_LOGIC_VECTOR (26 downto 0);
signal exponentA, exponentB : STD_LOGIC_VECTOR (7 downto 0);
signal zeros31 : STD_LOGIC_VECTOR (30 downto 0);
signal s_spfp : FP_STATES;
signal s_signOut : std_logic;
signal s_exponentOut : std_logic_vector (7 downto 0);
signal s_fractionOut : std_logic_vector (22 downto 0);

begin
zeros31 <= (others => '0');
process(CLK)
   -- Δημιουργία διάφορων μεταβλητών (variables) με εμβέλεια μόνο στην process ().
  variable exponentOut : std_logic_vector (7 downto 0);
  variable signOut : std_logic;
  variable signifOut ,v_signifA, v_signifB: std_logic_vector (27 downto 0);
  variable spfp : FP_STATES;
  variable expoDiff : integer;
  variable aGb : boolean;
  variable bGa : boolean;
  variable aEb : boolean;
  variable normalized : boolean;
  variable tmp: bit_vector (26 downto 0);
  variable tmp2 : bit_vector (27 downto 0);
  variable roundedOnce : boolean;
  variable isZero : boolean;
  variable isHigh : boolean;
  begin
  if CLK'event and CLK = '1' then
    if RESET = '0' then
      spfp := zero;
    else
      case spfp is
        -- Αρχικοποίηση όλων των μεταβλητών, signal και έλεγχος για τυχόν περίπτωση
        -- βελτιστοποίησης.
        when zero =>
          -- Ελέγχω να δω εάν ο ένας από τους δυο αριθμούς είναι 0.
          -- Θέτω το isZero σε true, αναθέτω στις μεταβλητές εξόδου  κατάλληλες τιμές και
          -- μεταβαίνω στην τελευταία κατάσταση. (Βελτιστοποίηση).
          if(A(30 downto 0) = zeros31) then
            signOut := B(31);
            exponentOut := B(30 downto 23);
            signifOut := ("00" & B(22 downto 0) & "000");
            isZero := true;
```

```
  elsif(B(30 downto 0) = zeros31) then
    signOut := A(31);
    exponentOut := A(30 downto 23);
    signifOut := ("00" & A(22 downto 0) & "000");
    isZero := true;
  -- Ελέγχω να δω εάν οι αριθμοί έχουν διαφορετικά πρόσημα αλλά ίδιο μέτρο.
  -- Το αποτέλεσμα στην περίπτωση αυτή θα είναι 0. Αναθέτω στις μεταβλητές εξόδου
  -- τις κατάλληλες τιμές,
  -- θέτω το isZero σε true και μεταβαίνω στην τελευταία κατάσταση.
  -- (Βελτιστοποίηση).
  elsif( (A(31) /= B(31)) and (A(30 downto 0) = B(30 downto 0)) ) then
    signOut := '0';
    exponentOut := (others => '0');
    signifOut := ("00" & zeros31(22 downto 0) & "000");
    isZero := true;
  else
    signifA <= ('1' & A(22 downto 0) & "000");
    signifB <= ('1' & B(22 downto 0) & "000");
    isZero := false;
  end if;
  exponentA <= A(30 downto 23);
  exponentB <= B(30 downto 23);
  isHigh := false;
  aGb := false;
  bGa := false;
  aEb := false;
  normalized := false;
  tmp := (others => '0');
  tmp2 := (others => '0');
  roundedOnce := false;
  VO <= '0';
  if(VI = '0') then
    spfp := zero;
  else
    if(isZero = true) then
      spfp := seven;
    else
      spfp := one;
    end if;
  end if;

-- Συγκρίνω τους εκθέτες, θέτω το αντίστοιχο flag σε true ή false και
-- αποθηκεύω στην μεταβλητή expoDiff την διαφορά τους.
when one =>
  -- Εάν το expoDiff είναι μεγαλύτερο ή ίσο με το 27 τότε ο ένας από τους δυο
  -- αριθμούς είναι
  -- πολύ μικρότερος από τον άλλον. Σε αυτή τη περίπτωση θέτω το isHigh σε true,
  -- αναθέτω στις μεταβλητές εξόδου την τιμή του μεγαλυτέρου και μεταβαίνω στην
  -- τελευταία κατάσταση.
  if(exponentA > exponentB) then -- Ο A είναι μεγαλύτερος από τον B.
    expoDiff := to_integer(unsigned(exponentA - exponentB));
    if(expoDiff >= 26) then
      signOut := A(31);
      signifOut := ("00" & A(22 downto 0) & "000");
      isHigh := true;
    end if;
    aGb := true;
    exponentOut := exponentA;
  elsif (exponentB > exponentA) then -- Ο B είναι μεγαλύτερος από τον A.
    expoDiff := to_integer(unsigned(exponentB - exponentA));
    if(expoDiff >= 26) then
      signOut := B(31);
      signifOut := ("00" & B(22 downto 0) & "000");
      isHigh := true;
    end if;
    bGa := true;
    exponentOut := exponentB;
```

```
  else
    expoDiff := 0;
    aEb := true;
    exponentOut := exponentA;
  end if;
  if(isHigh = true) then
    spfp := seven;
  else
    spfp := two;
  end if;

-- Ολισθαίνω το σημαντικό του A ή του B προς τα δεξιά ανάλογα με
-- το ποιο flag είναι true.
-- Στην ολίσθηση λαμβάνω υπόψη και το Hidden Bit.
when two =>
  -- Εάν το aGb είναι true τότε ολισθαίνω το σημαντικό του B προς τα δεξιά κατά
  -- expoDiff φορές.
  if (aGb = true) then
    tmp := To_bitvector(signifB);
    tmp := (tmp srl expoDiff);
    signifB <= To_StdLogicVector(tmp);
  -- Εάν το bGa είναι true τότε ολισθαίνω το σημαντικό του A προς τα δεξιά κατά
  -- expoDiff φορές.
  elsif (bGa = true) then
    tmp := To_bitvector(signifA);
    tmp := (tmp srl expoDiff);
    signifA <= To_StdLogicVector(tmp);
  end if;
  spfp := three;

-- Προσθέτω ή αφαιρώ τα σημαντικά των αριθμών ανάλογα με τα πρόσημα των αριθμών,
-- και αναθέτω τιμή στο πρόσημο της εξόδου.
when three =>
  v_signifA:='0' & signifA;
  v_signifB:='0' & signifB;
  if (A(31) /= B(31)) then
    if (signifA > signifB) then
      signOut := A(31);
      signifOut := v_signifA - v_signifB;
    elsif (signifB > signifA) then
      signOut := B(31);
      signifOut := v_signifB - v_signifA;
    end if;
  else
    signOut := A(31);
    signifOut := v_signifA + v_signifB;
  end if;
  spfp := four;

-- Ελέγχω το τελικό σημαντικό της εξόδου που είναι αποθηκευμένο στην μεταβλητή
-- signifOut
-- χρειάζεται κανονικοποίηση και θέτω την μεταβλητή normalized σε true ή false
-- ανάλογα.
  when four =>
  -- Εάν τα δυο σημαντικότερα bits του signifOut έχουν τιμή "01" τότε δεν
  -- χρειάζεται κανονικοποίηση
  -- και θέτω το normalized σε true. Διαφορετικά το αποτέλεσμα
  -- χρειάζεταικανονικοποίηση,
  -- στην περίπτωση αυτή θέτω το normalized σε false.
  if signifOut(27 downto 26) = "01" then
    normalized := true;
    if roundedOnce = false then
      spfp := six;
    else
      spfp := seven;
    end if;
```

```
        else
          normalized := false;
          spfp := five;
        end if;

      when five =>
        -- Εάν το Bit 28 του signifOut είναι '1' τότε ολισθαίνω προς τα δεξιά το
        -- signifOut και,
        -- αυξάνω τον τελικό εκθέτη κατά 1. Στην περίπτωση που τα δυο σημαντικότερα
        -- Bits,
        -- της μεταβλητής αυτής έχουν τιμή "00" τότε γίνεται αριστερή ολίσθηση και
        -- μειώνω τον εκθέτη κατά 1.
        if(signifOut(27) = '1') then
          tmp2 := To_bitvector(signifOut);
          tmp2 := (tmp2 srl 1);
          signifOut := To_StdLogicVector(tmp2);
          exponentOut := exponentOut + "1";
        elsif(signifOut(27 downto 26) = "00") then
          tmp2 := To_bitvector(signifOut);
          tmp2 := (tmp2  sll 1);
          signifOut := To_StdLogicVector(tmp2);
          exponentOut := exponentOut - 1;
        end if;
        spfp := four;

      -- Κάνω στρογγυλοποίηση προσθέτοντας +1 στο αποτέλεσμα και,
      -- θέτω το roundedOnce σε true για να δείξω ότι έχει γίνει στρογγυλοποίηση έστω
      -- και μια φορά.
      -- Η στρογγυλοποίηση γίνεται εάν τα 3 τελευταία Bits της μεταβλητής signifOut,
      -- που αντιστοιχούν σε guard, round και sticky bits έχουν τιμή μεγαλύτερη από
      -- "011".
      when six =>
        if(signifOut(2 downto 0) > "011") then
          signifOut(27 downto 3) := signifOut(27 downto 3) + 1;
        end if;
        roundedOnce := true;
        spfp := four;

      -- Στην κατάσταση αυτή απλά εξάγω από την μεταβλητή signifOut την ωφέλιμη,
      -- πληροφορία στην έξοδο,
      -- και επίσης θέτω την έξοδο Valid Out (VO) σε  '1'.
      when seven =>
        SUM(30 downto 23) <= exponentOut;
        SUM(22 downto 0) <= signifOut(25 downto 3);
        SUM(31) <= signOut;
        VO <= '1';
        -- Ελέγχω για υπερχείλιση ή ανεπάρκεια και θέτω την έξοδο OVFL  σε '0' ή '1'
        -- αντίστοιχα.
        if(exponentOut >= "00000001" and exponentOut <= "11111110") then
          OVFL <= '0';
        else
          OVFL <= '1';
        end if;
        spfp:=zero;
    end case;
  end if;
 end if;
 s_spfp <= spfp;
 s_signOut <= signOut;
 s_exponentOut <= exponentOut;
 s_fractionOut <= signifOut(25 downto 3);
end process;
end Behavioral;
```

--Πολλαπλασιασμός κινητής υποδιαστολής

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity mul_32_fp is
  Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
    B : in  STD_LOGIC_VECTOR (31 downto 0);
    RESET : in  STD_LOGIC;
    CLK : in  STD_LOGIC;
    VI : in STD_LOGIC;
    VO : out STD_LOGIC;
    OVFL : out  STD_LOGIC;
    PRODUCT : out  STD_LOGIC_VECTOR (31 downto 0));
end mul_32_fp;

architecture Behavioral of mul_32_fp is
-- Δημιουργία διάφορων signal με εμβέλεια σε όλο το πρόγραμμα (source file).
type FP_STATES is (zero, one, two, three, four, five, six);
signal s_spfp : FP_STATES;
begin
  process(CLK)
    -- Δημιουργία διάφορων μεταβλητών (variables) με εμβέλεια μόνο στην process ().
    variable spfp : FP_STATES;
    variable signifA, signifB : std_logic_vector (23 downto 0);
    variable signifOut : std_logic_vector (27 downto 0);
    variable b_signifOut : bit_vector (27 downto 0);
    variable exponentA, exponentB, exponentOut : std_logic_vector (8 downto 0);
    variable tmpMul : std_logic_vector (47 downto 0);
    variable isOneorNull : boolean;
    variable normalized : boolean;
    variable isRounded : boolean;
    constant BIAS : std_logic_vector (7 downto 0) := "01111111";
    constant C_ZERO : std_logic_vector (30 downto 0) := (others => '0');
    constant C_ONE : std_logic_vector (30 downto 0) :=
    "0111111100000000000000000000000";
    begin
      if(CLK'event and CLK = '1') then
        if(RESET = '0') then
          spfp := zero;
        else
          case spfp is
            -- Αρχικοποίηση όλων των μεταβλητών, signal και έλεγχος
            -- για τυχόν περίπτωση βελτιστοποιήσεις.
            when zero =>
            -- Ελέγχω να δω εάν ο ένας από τους δυο αριθμούς είναι 0.0 ή 1.0
            -- Θέτω το isOneorNull σε true, αναθέτω στις μεταβλητές εξόδου
            -- κατάλληλες τιμές και μεταβαίνω στην τελευταία
            -- κατάσταση. (Βελτιστοποίηση).
              if(A(30 downto 0) = C_ONE) then
                exponentOut := ('0' & B(30 downto 23));
                signifOut := ("01" & B(22 downto 0) & "000");
                isOneorNull := true;
              elsif(B(30 downto 0) = C_ONE) then
                exponentOut := ('0' & A(30 downto 23));
                signifOut := ("01" & A(22 downto 0) & "000");
                isOneorNull := true;
              elsif((A(30 downto 0) = C_ZERO) or (B(30 downto 0) = C_ZERO)) then
                exponentOut := (others => '0');
                signifOut(27 downto 26) := "01";
                signifOut(25 downto 0) := (others => '0');
                isOneorNull := true;
              else
                normalized := false;
                isRounded := false;
                exponentA := ('0' & A(30 downto 23));
```

```
    exponentB := ('0' & B(30 downto 23));
    signifA := ('1' & A(22 downto 0));-- 24 bits.
    signifB := ('1' & B(22 downto 0));-- 24 bits.
    isOneorNull := false;
    VO <= '0';
   end if;
   if(VI = '0') then
    spfp := zero;
   else
    if(isOneorNull = true) then
     spfp := six;
    else
     spfp := one;
    end if;
   end if;

-- Προσθέτω τους εκθέτες και αφαιρώ την πόλωση.
when one =>
  exponentOut := (exponentA + exponentB) - BIAS;
  spfp := two;

-- Πολλαπλασιάζω τα σημαντικά των αριθμών και
-- παίρνω μόνο τα 28 σημαντικότερα Bits που
-- είναι χρήσιμα και τα καταχωρώ στην μεταβλητή signifOut.
when two =>
  tmpMul := signifA * signifB;
  signifOut := tmpMul(47 downto 20);
  spfp := three;

-- Ελέγχω το τελικό σημαντικό της εξόδου που είναι αποθηκευμένο
-- στην μεταβλητή signifOut εάν χρειάζεται κανονικοποίηση
-- και θέτω την μεταβλητή normalized σε true ή false ανάλογα.
when three =>
  if(signifOut(27 downto 26) = "01") then
   normalized := true;
   if(isRounded = true) then
    spfp := six;
   else
    spfp := five;
   end if;
  else
   normalized := false;
   spfp := four;
  end if;

-- Κάνω κανονικοποίηση του αποτελέσματος με μια ολίσθηση
-- προς τα δεξιά και αυξάνω τον τελικό εκθέτη κατά 1.
when four =>
  b_signifOut := To_bitvector(signifOut);
  b_signifOut := b_signifOut srl 1;
  signifOut := To_StdLogicVector(b_signifOut);
  exponentOut := exponentOut + 1;
  spfp := three;

-- Κάνω στρογγυλοποίηση προσθέτοντας +1 στο αποτέλεσμα και
-- θέτω το isRounded σε true για να δείξω ότι έχει γίνει
-- στρογγυλοποίηση έστω και μια φορά. Η στρογγυλοποίηση
-- γίνεται εάν τα 3 τελευταία Bits της μεταβλητής signifOut
-- που αντιστοιχούν σε guard, round και sticky bits έχουν τιμή
-- μεγαλύτερη από "011".
when five =>
  if(signifOut(2 downto 0) > "011") then
   signifOut(27 downto 3) := signifOut(27 downto 3) + 1;
  end if;
  isRounded := true;
  spfp := three;
```

```
        -- Στην κατάσταση αυτή απλά εξάγω από την μεταβλητή signifOut
        -- την ωφέλιμη πληροφορία στην έξοδο και επίσης θέτω
        -- την έξοδο Valid Out (VO) σε '1'.
        when six =>
          PRODUCT(30 downto 23) <= exponentOut(7 downto 0);
          PRODUCT(22 downto 0) <= signifOut(25 downto 3);
          -- Εξάγω το πρόσημο της εξόδου με μια πράξη XOR ανάμεσα στα πρόσημα εισόδων.
          PRODUCT(31) <= A(31) xor B(31);
          -- Ελέγχω για υπερχείλιση ή ανεπάρκεια και θέτω την έξοδο OVFL  σε '0' ή '1'
          -- αντίστοιχα.
          if( (exponentOut >= "00000001" and exponentOut <= "11111110") or
          (isOneorNull = true and exponentOut = "00000000") ) then
            OVFL <= '0';
          else
            OVFL <= '1';
          end if;
          VO <= '1';
          spfp := zero;
       end case;
      end if;
    end if;
  s_spfp <= spfp;
  end process;
end Behavioral;
```