Ροή δεδομένων γενικού σκοπού για ταχεία εκτέλεση βρόγχων και υλοποίηση σε Fpga

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τζιουβάρας Αθανάσιος

Βόλος | Σεπτέμβριος | 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Μηχανικού Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

……..

Τζιουβάρας Αθανάσιος
Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Copyright © Tziouvaras Athanasios, 2013
Με επιφύλαξη παντός δικαιώματος. All rights reserved.
Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό.
Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα.
Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Ευχαριστίες

Με την περάτωση αυτής της διπλωματικής εργασίας θα ήθελα να ευχαριστήσω τον επιβλέπον κ. Δημητρίου, συμβασιούχο καθηγητή του τμήματος για την άριστη συνεργασία που είχαμε, για τις συμβουλές του, την υπομονή του και την αμέριστη βοήθειά του καθ' όλη τη διάρκεια της εργασίας. Τον κ. Σταμούλη καθηγητή του τμήματος για τη αποδοχή και την υποστήριξη του. Τον κ. Μπέλλα αναπληρωτή καθηγητή του τμήματος για την παροχή του απαραίτητου τεχνολογικού εξοπλισμού και όλους του φίλους και συγγενείς που με την στήριξή τους η πορεία μου αυτά τα χρόνια θα ήταν πολύ πιο δύσκολη αν όχι αδύνατη.

Περιεχόμενα

1. Εισαγωγή

Με την εμφάνιση του αλγορίθμου tomasulo και την εκμετάλλευση του παραλληλισμού επιπέδου εντολής (ILP, instruction level parallelism) επεξεργαστές γενικού σκοπού κατάφεραν να αυξήσουν σημαντικά την απόδοσή τους. Η αύξηση αυτή όμως περιορίζεται από τις εξαρτήσεις δεδομένων μεταξύ των εντολών. Έτσι όταν οι εξαρτήσεις αυτές είναι πολύ πυκνές τότε οι αλγόριθμοι που εκμεταλλεύονται τον παραλληλισμό επιπέδου εντολής δεν έχουν τόσο υψηλή απόδοση. Το επόμενο βήμα για την αύξηση απόδοσης των επεξεργαστών γενικού σκοπού είναι η εμφάνιση των πολυνηματικών αρχιτεκτονικών (multithreaded architectures). Οι αρχιτεκτονικές αυτές υποστηρίζουν την παράλληλη εκτέλεση κώδικα διαφορετικών νημάτων. Έτσι εφαρμογές που μπορούν να παραλληλοποιηθούν έχουν σημαντική αύξηση της ταχύτητας εκτέλεσής τους. Από την άλλη η απόδοση της τεχνικής αυτής περιορίζεται από τον βαθμό που μπορεί να παραλληλοποιηθεί μία εφαρμογή. Αν ο κώδικας που μας ενδιαφέρει δεν μπορεί να παραλληλοποιηθεί τότε δεν έχουμε κάποια βελτίωση του χρόνου εκτέλεσης. Επιπλέον η τεχνική των παράλληλων νημάτων απαιτεί και πολύ περισσότερο υλικό καθώς όλα τα νήματα τρέχουν παράλληλα, το καθένα δεσμεύοντας δικούς του πόρους. Σε αυτό όμως λύση δίνει ο νόμος του Moore, ο οποίος λέει ότι ο αριθμός των transistor που χωράνε σε ένα ολοκληρωμένο κύκλωμα θα διπλασιάζεται κάθε 18 μήνες. Έτσι καθώς ο αριθμός των transistor ολοένα αυξάνεται οι αρχιτεκτονικές που υποστηρίζουν παράλληλα νήματα είναι πιο εύκολο να αναπτυχθούν.

Όμως με την συνεχόμενη αύξηση του αριθμού των transistor προκύπτει ένα πρόβλημα που πρωτοεμφανίστηκε την τελευταία δεκαετία, η μεγάλη κατανάλωση ενέργειας από ένα ψηφιακό ολοκληρωμένο κύκλωμα. Το πρόβλημα αυτό γίνεται όλο και πιο σημαντικό και πολλές ώρες έρευνας δαπανούνται για εύρεση τρόπων για τη μείωση κατανάλωσης ενέργειας. Λύσεις όπως ο loop stream buffer της Intel προτάθηκαν οι οποίες είχαν πολύ καλά αποτελέσματα. Η κατανάλωση ενέργειας όμως παραμένει ακόμα και σήμερα μείζον πρόβλημα στη σχεδίαση ενός επεξεργαστή.

Τελικό συμπέρασμα είναι ότι η ανάγκη για εμφάνιση νέων, διαφορετικών αρχιτεκτονικών είναι μεγάλη καθώς οι ήδη υπάρχοντες αδυνατούν να προσφέρουν

την απαιτούμενη απόδοση καθώς αυτό καθορίζεται αυστηρά από την φύση της κάθε εφαρμογής.

Σκοπός της εργασίας αυτής είναι η σχεδίαση και ανάπτυξη μίας αρχιτεκτονικής τύπου EDGE στη γλώσσα περιγραφής υλικού Verilog. Η αρχιτεκτονική αυτή έχει ως σκοπό την ταχεία εκτέλεση βρόγχων και χρησιμοποιώντας τα εργαλεία της Xilinx θα υλοποιηθεί σε μία SPARTAN 6 fpga.

2. Αρχιτεκτονικές Υπολογιστών

2.1Η μετάβαση από CISC σε RISC

Μέχρι τις αρχές του 1970 η αρχιτεκτονική τύπου CISC(Complex Instruction Set Computing) χρησιμοποιούταν ευρέως στους ηλεκτρονικούς υπολογιστές. Η αρχιτεκτονική αυτή υποστήριζε σύνολα εντολών με πολύ περίπλοκες λειτουργίες ανά εντολή. Έτσι κάθε εντολή εκτελούσε πολλές σύνθετες λειτουργίες. Αυτό είχε ως αποτέλεσμα την μεγάλη πολυπλοκότητα σχεδιασμού του συστήματος καθώς και την επιβάρυνση πολλών κύκλων μηχανής ανά εντολή. Επομένως η συχνότητα του ρολογιού όφειλε να είναι χαμηλή σε σχέση με τις σημερινές και οι προγραμματιστές δυσκολευόταν αρκετά λόγο της περιπλοκότητας του ISA (Instruction Set Architecture) του επεξεργαστή.

Στις αρχές του 1970 προτάθηκε μία διαφορετική προσέγγιση στην διάρθωση της αρχιτεκτονικής των υπολογιστών γνωστή και ως RISC (Reduced Instruction Set Architecture). Το σύνολο εντολών της αρχιτεκτονικής αυτής περιλάμβανε εντολές με πολύ πιο απλές λειτουργίες από το αντίστοιχο σύνολο CISC, απλοποιώντας πολύ τον σχεδιασμό ενός τέτοιου συστήματος και τον προγραμματισμό σε αυτό. Έτσι οι κύκλοι που χρειάζεται κάθε εντολή έγιναν σημαντικά λιγότεροι και οι συχνότητα του ρολογιού αυξήθηκε έχοντας σαν αποτέλεσμα πιο γρήγορους και ευέλικτους επεξεργαστές. Τέλος οι RISC επεξεργαστές χρησιμοποιούνε κατά κόρον τις επικαλυπτόμενες εντολές αυξάνοντας σημαντικά την απόδοση (throughput) του συστήματος.

Ο PDP-11 της DEC είναι ένας CISC επεξεργαστής

Ο Ultra SPARC της SUN είναι ένας RISC επεξεργαστής

2.2Επεξεργαστές τύπου VLIW

Οι επεξεργαστές VLIW (Very Large Instruction Word) αναπτύχθηκαν στις αρχές του 1980 από μηχανικούς της Intel και της HHewlett Packard. Η λογική σε μία τέτοια αρχιτεκτονική είναι η ενοποίηση πολλών RISC εντολών σε μια μεγάλη VLIW εντολή. Η ενοποίηση αυτή γίνεται στατικά από τον compiler και όχι από το υλικό. Η νέα ενοποιημένη εντολή εκτελείται σε υλικό επικαλυπτόμενης λογικής που υποστηρίζει στατική εκτέλεση εντολών. Για την εκμετάλλευση του παραλληλισμού επιπέδου εντολής ,στις αρχιτεκτονικές αυτές, πολύ βάρος πέφτει στον compiler. Έτσι το υλικό είναι σχετικά απλό ενώ ο μεταγλωττιστής, που κάνει την περισσότερη δουλειά, έχει υψηλή περιπλοκότητα και είναι σχεδιασμένος έτσι ώστε να εκτελεί περίπλοκες βελτιστοποιήσεις όπως αναδίπλωση βρόγχου και επικάλυψη λογισμικού. Επεξεργαστές της αρχιτεκτονικής αυτής όπως ο Itanium χρησιμοποιούνται σήμερα κυρίως σε servers.

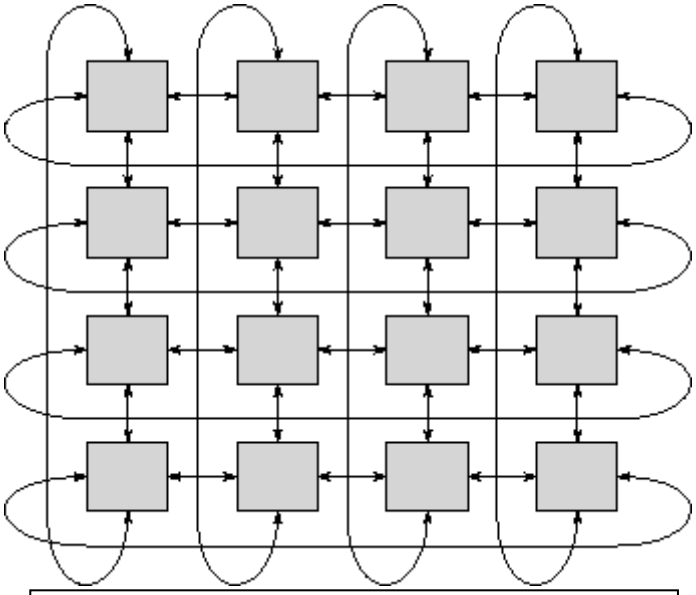
Ο Itanium 2 της Intel, από τους πιο γνωστούς VLIW επεξεργαστές

2.3 Επεξεργαστές τύπου EDGE

Οι επεξεργαστές τύπου EDGE (Explicit Data Graph Execution) δημιουργήθηκαν έτσι ώστε να εκμεταλλεύονται τον παραλληλισμό επιπέδου ροής δεδομένων των εφαρμογών. Έτσι ενώ σαν έννοια ο παραλληλισμός επιπέδου ροής δεδομένων είναι υπαρκτός από το 1970, το πρώτο δείγμα επεξεργαστή EDGE εμφανίστηκε μόλις την δεκαετία του 2000-2010.Η ανάγκη για την δημιουργία των επεξεργαστών EDGE δημιουργήθηκε καθώς οι RISC επεξεργαστές αδυνατούσαν να προσφέρουν την απαιτούμενη απόδοση και δεν μπορούσαν να έχουν την συνεχόμενη αύξηση ταχύτητας που ήταν απαραίτητη. Επιπλέον η τεχνολογία των Vlsi (Very large scale integration) πλέον καθιστά δυνατή την τοποθέτηση πάρα πολλών transistors σε μικρό χώρο κάτι που είναι απαραίτητο να εκμεταλευτεί από τους σχεδιαστές επεξεργαστών.

Οι επεξεργαστές αυτοί συνδυάζουν στοιχεία από RISC και από VLIW επεξεργαστές. Συγκεκριμένα ομαδοποιούνε πολλές απλές εντολές σε ομάδες που ονομάζονται hyperblocks οι οποίες μπορούνε να τρέχουνε παράλληλα. Ενώ λοιπόν η μεταγλώττιση που γίνεται από τον compiler μοιάζει πολύ με την μεταγλώττιση για VLIW επεξεργαστές, η εκτέλεση των εντολών γίνεται εκτός σειράς όπως σε έναν υπερβαθμωτό RISC επεξεργαστή. Πιο γενικά ο μεταγλωττιστής χωρίζει τον κώδικα προς εκτέλεση σε μικρότερα κομμάτια τα οποία δρομολογούνται σε διαφορετικά blocks υπομονάδων του επεξεργαστή.

Οι επεξεργαστές EDGE δημιουργήθηκαν με σκοπό να προσφέρουν την απόδοση που οι RISC επεξεργαστές δεν μπορούσαν να προσφέρουν. Το πιο σύγχρονο παράδειγμα είναι ο επεξεργαστής TRIPS ,που δημιουργήθηκε στο πανεπιστήμιο Austin του Texas της Αμερικής. Το 2012 , μάλιστα έσπασε τα 1 Tflops κάνοντας την αρχιτεκτονική του πολλά υποσχόμενη για μελλοντικά αποτελέσματα. Ο TRIPS χρησιμοποιεί ένα δίκτυο από λειτουργικές μονάδες ενωμένες σε διάταξη torus αυξάνοντας κατά πολύ την εκμετάλλευση του παραλληλισμού επιπέδου ροής δεδομένων.



Οι λειτουργικές μονάδες του TRIPS σε διάταξη torus

2.4 Είδη παραλληλισμού

Για την βελτίωση της ταχύτητας εκτέλεσης μίας εφαρμογής είναι καλό να γνωρίζουμε εκ των προτέρων πώς μπορούμε να την παραλληλοποιήσουμε, δηλαδή ποιά είδη παραλληλισμού υποστηρίζονται από την εφαρμογή. Έτσι και για την σχεδίαση ενός επεξεργαστή, πρέπει να στοχεύσουμε σε κάποια είδη παραλληλισμού του το σύστημά μας θα εκμεταλλεύεται εύκολα. Γενικά υπάρχουν 3 είδη παραλληλισμού που χρησιμοποιούνται συχνά, ο παραλληλισμός επιπέδου εντολής (ILP, instruction level parallelism), παραλληλισμός επιπέδου νήματος (thread level parallelism) και παραλληλισμός επιπέδου ροής δεδομένων (dataflow parallelism).

Ο παραλληλισμός επιπέδου εντολής εμφανίζεται σε μία εφαρμογή κυρίως όταν πολλές εντολές μπορούνε να εκτελούνται ταυτόχρονα. Περιορίζεται ιδιαίτερα όταν υπάρχουν πολλές πυκνές εξαρτήσεις δεδομένων μεταξύ εντολών ενώ όταν υπάρχει προσφέρει πολύ γρήγορη ταχύτητα εκτέλεσης εάν εκμεταλλευτεί κατάλληλα. Η έννοια του ILP εισάχθηκε το 1967 μαζί με τον αλγόριθμο tomasulo, έναν τρόπο για την εκτέλεση εντολών εκτός σειράς. Άλλες γνωστές τεχνικές για εκμετάλλευση του παραλληλισμού επιπέδου εντολής είναι οι επικαλυπτόμενες λειτουργίες, οι υπερβαθμωτοί επεξεργαστές, η εκτέλεση εκτός σειράς, η μετονομασία καταχωρητών, η υποθετική εκτέλεση εντολών και η πρόβλεψη διακλαδώσεων.

Ο παραλληλισμός επιπέδου νημάτων ή αλλιώς παραλληλισμός επιπέδου διεργασίας εκμεταλλεύεται κυρίως από πολυεπεξεργαστικά συστήματα ή από συστήματα που υποστηρίζουν παράλληλη εκτέλεση πολλών νημάτων (π.χ. hyper threaded processors της Intel). Για την εκμετάλλευση του παραλληλισμού αυτού μία διεργασία χωρίζεται σε μικρότερες ανεξάρτητες διεργασίες και η κάθε μία εκτελείται σαν διαφορετικό νήμα σε διαφορετικό επεξεργαστή. Αυτό προηποθέτει ότι το κάθε νήμα είναι ανεξάρτητο από τα άλλα ώστε να επιτυγχάνεται η μέγιστη απόδοση. Ο χωρισμός της διεργασίας σε πολλές μικρότερες γίνεται από τον compiler και είναι μία δύσκολη εργασία με πολλές φορές αμφίβολα αποτελέσματα. Ο παραλληλισμός αυτός μπορεί να αυξήσει την ταχύτητα εκτέλεσης μίας εφαρμογής κατά τάξεις μεγέθους εάν η εφαρμογή τον υποστηρίζει ενώ μπορεί ακόμα και να την μειώσει εάν η εφαρμογή δεν ενδείκνυται για κάτι τέτοιο.

Ο παραλληλισμός επιπέδου ροής δεδομένων χρησιμοποιείται με πολύ επιτυχία σε εφαρμογές με βαριές υπολογιστικές ρουτίνες. Επίκεντρο του παραλληλισμού αυτού

είναι το πώς οι μεμονωμένες εντολές συνδέονται μεταξύ τους και όχι τόσο τι εξαρτήσεις έχουν. Συχνά η απεικόνιση του παραλληλισμού αυτού γίνεται μέσω ενός γράφου που δείχνει την ροή των δεδομένων από την αρχή μέχρι το τέλος της εφαρμογής. Η μελέτη της ροής των δεδομένων και όχι τόσο των εξαρτήσεων μεταξύ των εντολών συχνά μας δίνει πληροφορίες και τρόπους για την βελτίωση του χρόνου εκτέλεσης της εφαρμογής ακόμα και αν υπάρχουν πυκνές εξαρτήσεις δεδομένων και ακόμα και αν η εφαρμογή δεν παραλληλοποιείται.

Ο κάθε τρόπος παραλληλισμού υποστηρίζεται συνήθως από διαφορετικές εφαρμογές και οι χρόνοι εκτέλεσής τους επιταχύνονται κατά πάρα πολύ εάν μπορούμε να τους εκμεταλλευτούμε σε κάθε περίσταση. Γενικά ένα σύστημα θεωρείται ολοκληρωμένο εάν μπορεί να προσφέρει βελτιωμένους χρόνους εκτέλεσης σε διαφορετικές διεργασίες που υποστηρίζουν διαφορετικούς τρόπους παραλληλισμού.

3. Το πρόβλημα

Η τεχνολογία των υπολογιστών είναι μία διαρκώς εξελισσόμενη αγορά με ισχυρές ανταγωνιστικές εταιρίες οι οποίες διακρίνονται για την διαρκώς αυξανόμενη ποιότητα των προσφερόμενων προϊόντων τους. Όμως όσο η εξέλιξη αυτή συνεχίζεται τόσο μεγαλώνουν και οι απαιτήσεις της αγοράς. Πλέον η απόδοση δεν είναι το επικρατέστερο μέτρο σύγκρισης επεξεργαστών καθώς την τελευταία δεκαετία η κατανάλωση ισχύος έγινε σοβαρό πρόβλημα. Αυτό συνέβη κυρίως γιατί η τεχνολογία των Vlsi έχει προχωρήσει τόσο ώστε να είναι δυνατόν 2 δισεκατομμύρια transistors να παραβρίσκονται σε έναν επεξεργαστή. Έτσι λόγο του αριθμού των transistors η κατανάλωση ισχύος του κυκλώματος γίνεται πολύ υψηλή.

Τεχνολογίες και τεχνικές αναπτύχθηκαν ώστε να περιοριστεί η κατανάλωση ισχύος σε ένα ψηφιακό ολοκληρωμένο κύκλωμα. Η πιο γνωστή τεχνολογία που βοήθησε σε αυτό είναι η ανάπτυξη transistors μεγέθους 22 nm από την Intel. Τα transistor αυτά παρέχουν μέχρι και 33% μικρότερη κατανάλωση ενέργειας από τα υπόλοιπα Cmos transistors. Εκτός όμως από τεχνολογίες αναπτύχθηκαν και τεχνικές για την μείωση της κατανάλωσης ισχύος. Το φαινόμενο γνωστό ως "σκοτεινή σιλικόνη (dark silicon)" υπάρχει σε όλους τους σύγχρονους επεξεργαστές. Όταν ένα μέρος του κυκλώματος δεν χρησιμοποιείται τότε η τάση τροφοδοσίας των transistors εκείνων μειώνεται κατά πολύ για εξοικονόμηση ενέργειας. Άλλη γνωστή τεχνική είναι ο loop stream buffer της Intel. Ο buffer αυτός τοποθετείται πλέον ακριβώς μετά την κρυφή μνήμη εντολών και πριν το κύκλωμα αποκωδικοποίησης. Όταν κάποιος βρόγχος στον κώδικα ανιχνεύεται τότε οι εντολές του αποκωδικοποιούνται μία μόνο φορά και εκδίδονται για εκτέλεση. Με τον τρόπο αυτό το κύκλωμα αποκωδικοποίησης εντολών καθώς και η κρυφή μνήμη εντολών χρησιμοποιούνται μόνο μία φορά ανά βρόγχο αντί για το καθιερωμένο "μία φορά ανά επανάληψη" μειώνοντας κατά πολύ το κόστος κάθε βρόγχου σε ισχύ.

Η χαμηλή κατανάλωση ισχύος σαν νεοεισαχθέν πρόβλημα απασχόλησε πολλούς σχεδιαστές επεξεργαστών. Όμως και η απόδοση ενός επεξεργαστή είναι ακόμη ένα βασικό μέτρο σύγκρισης και αντικείμενο έρευνας. Πλέον ανταγωνιστικός, ως προς την απόδοσή του, ένας σύγχρονος επεξεργαστής δεν θεωρείται αν απλά είναι ταχύτερος από μία προηγούμενη έκδοση του. Είναι σχεδόν απαραίτητο η απόδοση του νέου επεξεργαστή να είναι πολλαπλάσια αυτής του προγενέστερου μοντέλου του.

Έτσι επιβάλλεται μία εκθετική αύξηση στην ταχύτητα των επεξεργαστών γενικού σκοπού, αύξηση η οποία όπως φαίνεται δεν είναι εύκολο να διατηρηθεί αν δεν υπάρχει καινοτομία και σκέψη εκτός του καθιερωμένου τρόπου από τους σχεδιαστές. Με τον παραλληλισμό επιπέδου εντολής να έχει φτάσει στα όρια του από πλευράς εκμετάλλευσης και τον παραλληλισμό επιπέδου νημάτων να μην υποστηρίζεται από πολλές εφαρμογές είναι επιτακτική η ανάγκη για την δημιουργία νέων αρχιτεκτονικών, αρκετά διαφορετικών από τις ήδη υπάρχουσες, ο οποίες να μπορούνε να προσφέρουν εύκολα την απαιτούμενη απόδοση κρατώντας παράλληλα την κατανάλωση ισχύος σε χαμηλά επίπεδα.

4. Πρόταση νέας αρχιτεκτονικής

Σαν λύση στο παραπάνω πρόβλημα προτείνουμε μία νέα αρχιτεκτονική που σχεδιάστηκε με τους παραπάνω στόχους και πετυχαίνει πολύ γρήγορη εκτέλεση βρόγχων. Καθώς σε μία εφαρμογή ο περισσότερος χρόνος καταναλώνεται στην εκτέλεση βρόγχων, η επιτάχυνση αυτών συντελεί στην ταχύτερη εκτέλεση όλης της εφαρμογής. Η αρχιτεκτονική αυτή έχει στοιχεία από την αρχιτεκτονική του TRIPS γνωστή ως EDGE και εκμεταλλεύεται τον παραλληλισμό επιπέδου εντολής και τον παραλληλισμό επιπέδου ροής δεδομένων ενώ μπορεί με ορισμένες αλλαγές να εκμεταλλευτεί και τον παραλληλισμό επιπέδου νημάτων. Η δομή της είναι τέτοια ώστε να επιτρέπει την συνεχόμενη γραμμική αύξηση της απόδοσης της με την πρόσθεση περισσότερων λειτουργικών μονάδων ενώ επηρεάζεται ελάχιστα από τις εξαρτήσεις δεδομένων μεταξύ των εντολών. Επιπλέον υποστηρίζεται και η χαμηλή κατανάλωση ισχύος με τεχνικές που χρησιμοποιούνται ευρέως στην βιομηχανία σε σύγχρονους επεξεργαστές γενικού σκοπού.

Για την υλοποίησή της σχεδιάσαμε κυρίως το back end, δηλαδή το δίκτυο των λειτουργικών μονάδων όπου γίνεται και η εκτέλεση των εντολών αλλά και το front end, δηλαδή το κύκλωμα αποκωδικοποίησης των εντολών. Το όλο σύστημα γράφτηκε με την γλώσσα περιγραφής υλικού Verilog, προσομοιώθηκε με το εργαλείο Modelsim και υλοποιήθηκε στην Spartan 6 fpga της Xilinx. Τα πειραματικά αποτελέσματα που πήραμε είναι ιδιαίτερα ενθαρρυντικά καθώς ήταν όμοια με αυτά της προσομοίωσης. Σύμφωνα με αυτά κάθε επανάληψη του βρόγχου ξεκινάει σε κάθε κύκλο μηχανής, επικαλύπτοντας την προηγούμενη επανάληψη. Από την σκοπιά του compiler αυτό μοιάζει αρκετά με την επικάλυψη λογισμικού, μία τεχνική που χρησιμοποιείται πολύ σε επεξεργαστές VLIW.

Την ανάλυση της μικροαρχιτεκτονικής αυτής καθώς και των ειδικών εντολών της θα την παρουσιάσουμε παρακάτω, όπου και θα ακολουθήσει παρουσίαση των πειραματικών αποτελεσμάτων.

5. Προσομοίωση με γλώσσα περιγραφής υλικού

Η ιδέα πίσω από την αρχιτεκτονική αυτή είναι η ύπαρξη ενός αριθμού κόμβων (στην περίπτωσή μας 16) οι οποίοι θα περιέχουν μία λειτουργική μονάδα ο καθένας. Ο κάθε κόμβος θα μπορεί να εκτελεί μία εντολή ανά πάσα στιγμή, επωμένος το πολύ 16 εντολές μπορούνε να εκτελούνται ταυτόχρονα. Οι εντολές αυτές θα είναι το κυρίως σώμα ενός βρόγχου, άρα είναι λογικό να έχουν εξαρτήσεις από δεδομένα μεταξύ τους. Η παροχέτευση αποτελεσμάτων μεταξύ των κόμβων θα γίνεται μέσω απ' ευθείας συνδέσμων που θα ενώνουν τους κόμβους μεταξύ τους. Έτσι εάν οι λειτουργικές μονάδες είναι επικαλυπτόμενες θα είναι δυνατό να αρχίζει την εκτέλεσή της μία επανάληψη ανά κύκλο μηχανής αφού στον κάθε κόμβο ανατίθεται μία εντολή. Η ακριβής λειτουργία του συστήματος θα περιγραφεί παρακάτω καθώς προηγείται μία περιγραφή των λειτουργικών μονάδων και των εντολών που υποστηρίζονται από την αρχιτεκτονική (ISA).

5.1Αρχιτεκτονική συνόλου εντολών

Παρακάτω δίνεται ένας πίνακας με της εντολές που υποστηρίζονται από την νέα αρχιτεκτονική καθώς και το αν υποστηρίζονται για αριθμούς κινητής ή σταθερής υποδιαστολής. Είναι σαφές ότι οι εντολές αυτές είναι όμοιες με αυτές ενός RISC συστήματος και συγκεκριμένα του επεξεργαστή Mips. Επιλέχθηκαν έτσι ώστε να καλύπτουν ένα ευρύ σύνολο λειτουργιών και ταυτόχρονα να είναι κατανοητές για ακαδημαϊκούς λόγους. Έκτος όμως από τις κλασικές εντολές RISC υπάρχουν και μερικές λειτουργίες που είναι περισσότερο σύνθετες και μοιάζουν πιο πολύ με εντολές CISC. Ο λόγος που επιλέχθηκαν να υποστηρίζονται εξηγείται παρά κάτω. Το σύμβολο "--" υποδυκνύει ότι η εντολή δεν υποστηρίζεται για την εκάστοτε λειτουργία αλλά και ότι η εντολή δεν έχει νόημα να υποστηριχτεί για την λειτουργία αυτή καθώς κανένα σύνολο εντολών δεν την υποστηρίζει.

Εντολή σε συμβολική γλώσσα	Επεξήγηση εντολής	Υποστήριξη για ακεραίους	Υποστήριξη για αριθμούς κινητής
Add $1,$2,$3	$1=$2+$3	Ναι	Ναι
Addi $1,$2,int	$1=$2+int	Ναι	--
Sub $1,$2,$3	$1=$2-$3	Ναι	Ναι
Slt $1,$2,$3	If $2<=$3 then $1=1, else $1=0	Ναι	Όχι
Slti $1,$2,int	If $2<=int then $1=1, else $1=0	Ναι	Όχι
Or $1,$2,$3	$1=$2 or $3	Ναι	--
Ori $1,$2,int	$1=$2 or int	Ναι	--
Not $1,$2	$1=not $2	Ναι	--
Nor $1,$2,$3	$1=$2 nor $3	Ναι	--
Xor $1,$2,$3	$1=$2 xor $3	Ναι	--
Sll $1,$2,$3	$1=$2 << $3	Ναι	--
Srl $1,$2,$3	$1=$2 >> $3	Ναι	--
Sra $1,$2,$3	$1=$2 >>> $3	Ναι	--
Bne $1,$2,Label	If $1 != $2, goto Label	Ναι	Ναι
Beq $1,$2,Label	If $1 == $2, goto Label	Ναι	Ναι
Beqi1 $1,$2,Label	If $1 == $2, goto Label, $1++	Ναι	Ναι
Beqi4 $1,$2,Label	If $1 == $2, goto Label, $1=$1+4	Ναι	Ναι
Beqd1 $1,$2,Label	If $1 == $2, goto Label, $1--	Ναι	Ναι
Beqd4 $1,$2,Label	If $1 == $2, goto Label, $1=$1-4	Ναι	Ναι
Bnei1 $1,$2,Label	If $1 != $2, goto Label, $1++	Ναι	Ναι
Bnei4 $1,$2,Label	If $1 != $2, goto Label,$1=$1+4	Ναι	Ναι
Bned1	If $1 != $2, goto Label, $1--	Ναι	Ναι

$1,$2,Label			
Bned4 $1,$2,Label	If $1 != $2, goto Label,$1=$1-4	Ναι	Ναι
Mull $1,$2,$3	$1=$2*$3	Ναι	Ναι
Mulli $1,$2,int	$1=$2*int	Ναι	--
Lw $1,off($2)	$1=MEM[$2+off]	Ναι	Ναι
Lwi1 $1,off($2)	$1=MEM[$2+off],off++	Ναι	Ναι
Lwi4 $1,off($2)	$1=MEM[$2+off], off=off+4	Ναι	Ναι
Lwd1 $1,off($2)	$1=MEM[$2+off],off--	Ναι	Ναι
Lwd4 $1,off($2)	$1=MEM[$2+off]off=off-4	Ναι	Ναι
Sw $1,off($2)	MEM[$2+off]=$1	Ναι	Ναι
Swi1 $1,off($2)	MEM[$2+off]=$1,off++	Ναι	Ναι
Swi4 $1,off($2)	MEM[$2+off]=$1,off=off+4	Ναι	Ναι
Swd1 $1,off($2)	MEM[$2+off]=$1,off--	Ναι	Ναι
Swd4 $1,off($2)	MEM[$2+off]=$1,off=off-4	Ναι	Ναι
J label	PC=PC+4+label	Ναι	--

Καθώς οι κόμβοι (και άρα οι λειτουργικές μονάδες) για την εκτέλεση εντολών είναι περιορισμένοι, ορισμένες λειτουργίες που συμβαίνουν πολύ συχνά σε βρόγχους θα κατανάλωναν πάνω από έναν κόμβο για να εκτελεστούνε. Τέτοιες είναι η φόρτωση από την μνήμη ενός πίνακα, ή η αποθήκευσή του σε αυτή, ή ακόμα και μία αρχική συνθήκη για την εκτέλεση του βρόγχου. Γενικά λειτουργίες που παρουσιάζουν συγκεκριμένα μοτίβα προσαύξησης καταχωρητών που χρησιμοποιούνται για έλεγχο συνθηκών ή για πρόσβαση στη μνήμη υπάγονται στην κατηγορία αυτή. Σε αυτές τις περιπτώσεις μία απλή λειτουργία αποτελείται από 2 εντολές RISC. Επομένως σύμφωνα με την αρχική μας ιδέα θα έπρεπε να εκτελεστεί σε 2 κόμβους. Αυτό όμως είναι ασύμφορο καθώς ο αριθμός των κόμβων είναι πολύ περιορισμένος για να ξοδεύουμε 2 για μία τόσο συχνή λειτουργία. Έτσι καταλήξαμε ότι συγκεκριμένοι κόμβοι θα μπορούνε να εκτελούνε πιο σύνθετες εντολές έτσι ώστε να υποστηρίζουν τις παραπάνω περιπτώσεις χωρίς να είναι απαραίτητο να δεσμεύσουμε παραπάνω πόρους. Έτσι για παράδειγμα ένας απλός βρόγχος όπως ο παρακάτω :

```
For (i=0;i<100;i++)
{
A[i]=a+b;
}
```

Μεταφράζεται στον παρακάτω κώδικα:

```
Addi end,0,100
Addi i,0,-1

Loop:
Beqi1 I,end,exit
Add tmp,a,b
Sw tmp,off(A)
J loop
Exit:
```

Αυτό που κερδίζουμε είναι το γεγονός ότι η αύξηση της μεταβλητής “i” γίνεται μαζί με την εντολή beqi1 και δεν απαιτείται άλλος κόμβος για να κάνει την πράξη addi i,i,1. Για τον ίδιο λόγο προσθέσαμε και επιπλέον λειτουργικότητες στις εντολές που μεταφέρουν δεδομένα από και προς την μνήμη (lw, sw). Η εξοικονόμηση κόμβων μας επιτρέπει την εκτέλεση μεγαλύτερων βρόγχων με ένα μικρό σχεδιαστικό trade off σε ότι αφορά την πολυπλοκότητα του κυκλώματος. Να σημειωθεί εδώ ότι ο αντίστοιχος loop stream buffer της Intel υποστηρίζει μέχρι 21 εντολές, αριθμός πολύ ικανοποιητικός αφού οι περισσότεροι βρόγχοι σε ένα κώδικα αποτελούνται από 10 έως 25 εντολές. Αναλυτικότερα στο θέμα αυτό θα αναφερθούμε παρά κάτω.

5.2 Η μικροαρχιτεκτονική του συστήματος

Για να κατανοήσουμε καλύτερα την αρχιτεκτονική του επεξεργαστή θα ήταν καλό να δούμε ξανά τις σχεδιαστικές αρχές και περιορισμούς που θέσαμε από την αρχή. Το επεξεργαστικό σύστημα αυτό θα πρέπει να πληροί τις παρακάτω προϋποθέσεις :

1) Η εκτέλεση κάθε επανάληψης να ξεκινάει ανά κύκλο μηχανής όταν οι βαθμίδες επικάλυψης είναι γεμάτες, δηλαδή όταν φτάσει στο λεγόμενο steady state (σταθερή κατάσταση).
2) Θα υπάρχουν 16 λειτουργικές μονάδες- κόμβοι που θα είναι υπεύθυνες για την εκτέλεση απλών αλλά και πιο σύνθετων εντολών.
3) Οι κάθε εντολή θα πρέπει να εκτελείται ξανά και ξανά στον ίδιο κόμβο ανά επανάληψη.
4) Ο κάθε κόμβος θα πρέπει να χρησιμοποιεί καταχωρητές επικάλυψης έτσι ώστε να πετυχαίνει ρυθμό ολοκλήρωσης εντολών ανά κύκλο ίσο με ένα. Η επικάλυψη εντολών θα πρέπει να είναι τέτοια ώστε ο κάθε κόμβος να έχει εντολές από διαφορετικές επαναλήψεις σε κάθε του φάση επικάλυψης.
5) Ο κάθε κόμβος θα πρέπει να ενώνεται με τους υπόλοιπους άμεσα καθώς ένα εξωτερικό κύκλωμα διοχέτευσης θα καθυστερήσει το κύκλωμα και δεν θα μπορούμε να έχουμε τον ρυθμό ολοκλήρωσης εντολών που θέλουμε.
6) Αν όμως κάθε κόμβος συνδέεται με όλους τους άλλους τότε θα υπάρχουν τουλάχιστον 16 είσοδοι δεδομένων σε κάθε κόμβο και η επιλογή δύο από αυτές μέσω πολυπλεκτών θα επιβαρύνει το κύκλωμα με επιπλέον καθυστέρηση, κάτι το οποίο δεν θέλουμε. Επομένως θα πρέπει να βρούμε μία βέλτιστη τοπολογία για την συνδεσμολογία των κόμβων.
7) Μετά τη πρώτη επανάληψη το κύκλωμα του φακέλου καταχωρητών, το κύκλωμα αποκωδικοποίησης εντολών καθώς και η μνήμη εντολών δεν μας χρειάζονται, οπότε η τάση τροφοδοσίας των transistors που συμμετέχουν στα στοιχεία αυτά μπορεί να μειωθεί κατά πολύ για εξοικονόμηση ενέργειας.
8) Για λειτουργία του κυκλώματος σε υψηλές συχνότητες θα πρέπει να χρησιμοποιήσουμε πολλούς καταχωρητές επικάλυψης για τις λειτουργικές μονάδες που υποστηρίζουν ακριβές σε καθυστέρηση πράξεις.

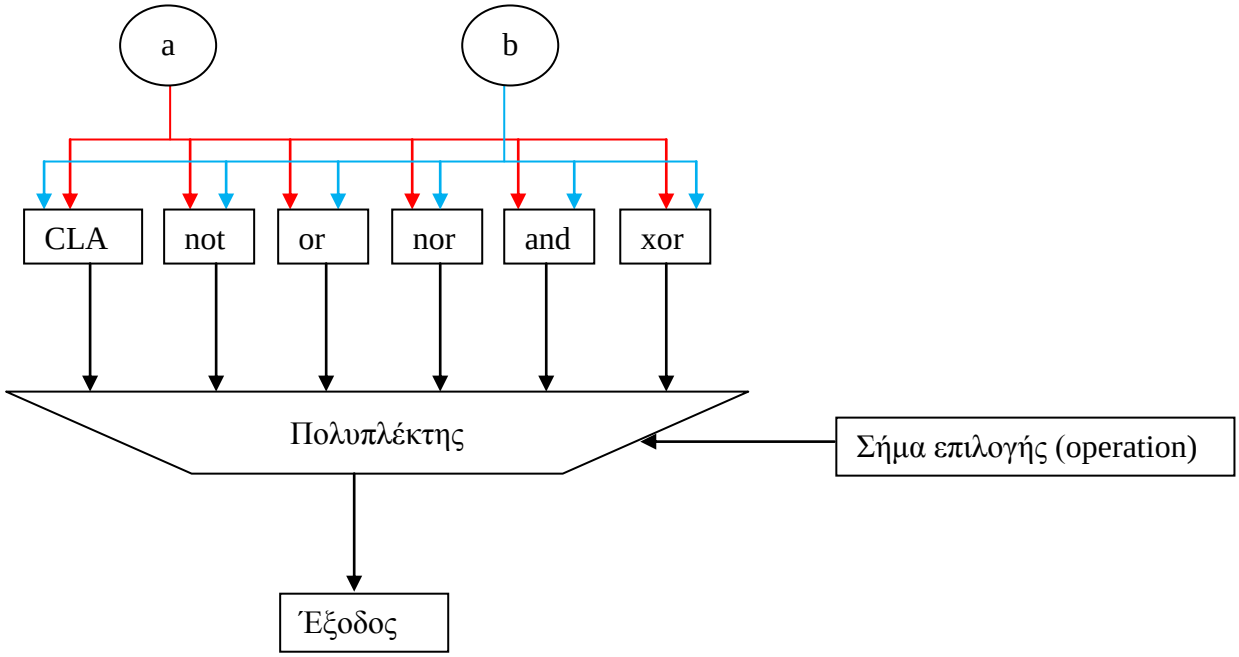
Με την γνώση των παραπάνω σχεδιάσαμε αρχικά τις λειτουργικές μονάδες του συστήματος.

5.2.1. Λειτουργικές μονάδες

Οι λειτουργικές μονάδες που δημιουργήθηκαν είναι οι εξείς : Αριθμητική και λογική μονάδα (alu), μονάδα πρόσθεσης αριθμών κινητής υποδιαστολής(fpalu), μονάδα πολλαπλασιασμού αριθμών σταθερής υποδιαστολής(mull), μονάδα πολλαπλασιασμού αριθμών κινητής υποδιαστολής(fpmull), μονάδα σύγκρισης(cmp), μονάδα μνήμης(mem), μονάδα ολίσθησης και μονάδα μετατροπής αριθμών από κινητή σε σταθερή υποδιαστολή και από σταθερή σε κινητή υποδιαστολή(convert). Για την δημιουργία όλων των μονάδων πήραμε υπ όψη μας τους πιο γρήγορους αλγορίθμους υλοποίησης της εκάστοτε πράξης και κάναμε βελτιστοποιήσεις σε επίπεδοο λογικών πυλών.

Αριθμητική και λογική μονάδα (Alu): Η μονάδα αυτή αποτελείται από ένα κύκλωμα πρόσθεσης και αφαίρεσης και από πύλες που εκτελούνε της λογικές πράξεις and,or,not,nor,xor. Από τις εξόδους αυτών των κυκλωμάτων επιλέγεται η μία έξοδος της alu μέσω ενός πολυπλέκτη που έχει ως σήμα επιλογής την λειτουργία (operation) του κυκλώματος. Το κύκλωμα της πρόσθεσης/αφαίρεσης υλοποιήθηκε σε διάταξη αθροιστή πρόβλεψης κρατουμένου (carry look ahead adder) 32 bit και ολοκληρώνει μία πράξη σε ένα κύκλο μηχανής. Η λογική του carry look ahead adder βασίζεται στον διαχωρισμό του κάθε τελεσταίου σε 8 ομάδες των $4^{ων}$ bit. Η κάθε ομάδα προσθέτει τα 4 bit αυτά και παράγει ένα κρατούμενο εξόδου και ένα αποτέλεσμα $4^{ων}$ bit, ενώ δέχεται και σαν είσοδο και ένα κρατούμενο εισόδου. Οι 8 αυτές πράξεις εκτελούνται ταυτόχρονα. Η ταχύτητα όμως του carry look ahead adder πηγάζει όχι μόνο από τον παραλληλισμό του πρώτου επιπέδου (παράλληλη πρόσθεση των 8 ομάδων $4^{ων}$ bit) αλλά και από το γεγονός ότι όλο το τελικό αποτέλεσμα παράγεται μετά από πολύ λιγότερες λογικές πύλες σε σύγκριση με έναν κλασικό προσθετή διάδοση κρατουμένου. Για να επιτευχθεί αυτό ο carry look ahead adder χωρίζεται σε επίπεδα τα οποία υπολογίζουν ορισμένα κρατούμενα και τα διαδίδουν στο αμέσως

πάνω επίπεδό τους. Με τον τρόπο αυτό όταν ο υπολογισμός φτάσει στο τελευταίο επίπεδο, τα παραγόμενα αποτελέσματα στέλνονται ξανά στο $1^{ο}$ επίπεδο για να υπολογιστεί το τελικό αποτέλεσμα. Η πράξη της αφαίρεσης ανάγεται σε πρόσθεση του ενός τελεσταίου με το συμπλήρωμα ως προς 2 του άλλου, καθιστώντας την δυνατή να υλοποιηθεί σε ένα κύκλωμα πρόσθεσης. Ο αθροιστής πρόβλεψης κρατουμένου χρησιμοποιείται πολύ σε σύγχρονους επεξεργαστές αλλά και σε ενσωματωμένα συστήματα όταν η ταχύτητα του κυκλώματος είναι πρωταρχικός στόχος.

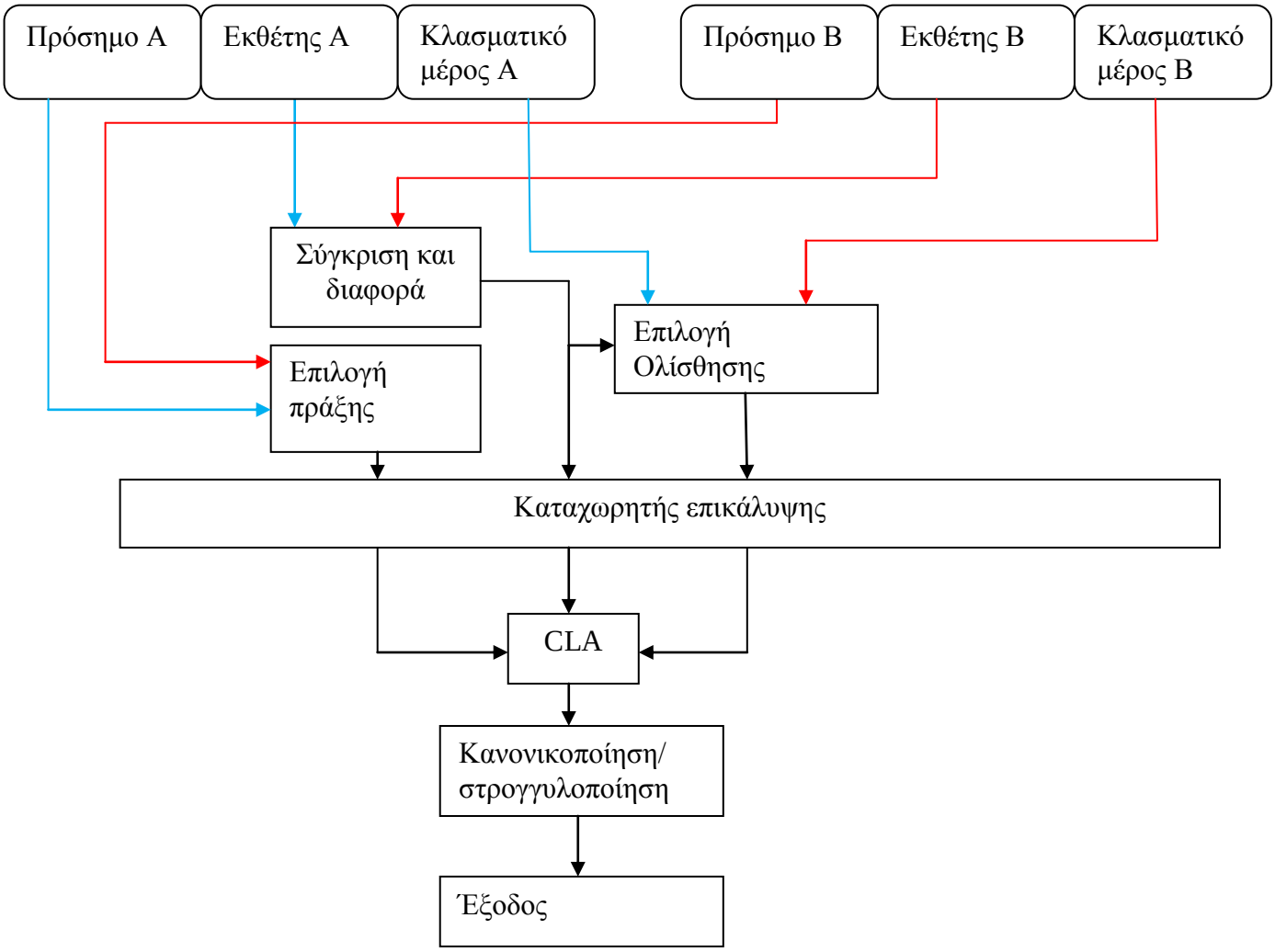


Το διάγραμμα της αριθμητικής και λογικής μονάδας

Μονάδα πρόσθεσης κινητής υποδιαστολής : Το κύκλωμα πρόσθεσης δύο αριθμών κινητής υποδιαστολής είναι αρκετά περίπλοκο. Ο σχεδιαστής θα πρέπει να τοποθετήσει αρκετούς καταχωρητές επικάλυψης έτσι ώστε να πάρει την κατάλληλη απόδοση. Για την απεικόνιση των αριθμών κινητής υποδιαστολής χρησιμοποιήθηκε το πρότυπο της ieee, 754. Κατά το πρότυπο αυτό, ο κάθε αριθμός είναι κανονικοποιημένος και εύρους 32 bit. Κανονικοποιημένος είναι ο αριθμός που έχει ένα μόνο ψηφίο αριστερότερα της υποδιαστολής και ο εκθέτης του είναι κατάλληλα προσαρμοσμένος.

Το πιο σημαντικό bit υποδηλώνει το πρόσημο του αριθμού ενώ τα bit 30 έως 23 είναι ο λεγόμενος πολωμένος εκθέτης. Ο πολωμένος εκθέτης είναι ο κανονικός εκθέτης του κανονικοποιημένου αριθμού προσαυξημένος με την σταθερά 127, ενώ τα bits 22 έως 0 είναι το κλασματικό μέρος του αριθμού.

Για να γίνει η πράξη της πρόσθεσης 2 αριθμών κινητής υποδιαστολής η παρακάτω διαδικασία ακολουθείται. Γίνεται σύγκριση των 2 εκθετών και το κλασματικό μέρος του αριθμού με τον μικρότερο εκθέτη ολισθένεται δεξιά τροποποιώντας έτσι τον εκθέτη του αριθμού, μέχρις ότου οι 2 εκθέτες να γίνουν ίσοι. Στη συνέχεια γίνεται πρόσθεση ή αφαίρεση (ανάλογα με τα πρόσημα) των δύο κλασματικών μερών ενώ για τον τελικό αριθμό κρατάμε τον μεγαλύτερο εκθέτη. Τέλος ακλουθούν τα στάδια της κανονικοποιήσης και στρογγυλοποίησης του τελικού αριθμού ώστε να σιγουρευτούμε ότι η αναπαράστασή του θα γίνει βάση του προτύπου ieee 754. Η πράξη ολοκληρώνεται σε 2 κύκλους μηχανής και είναι πλήρως επικαλυπτόμενη έτσι ώστε να αυξηθεί η απόδοση (throughput) της μονάδας.

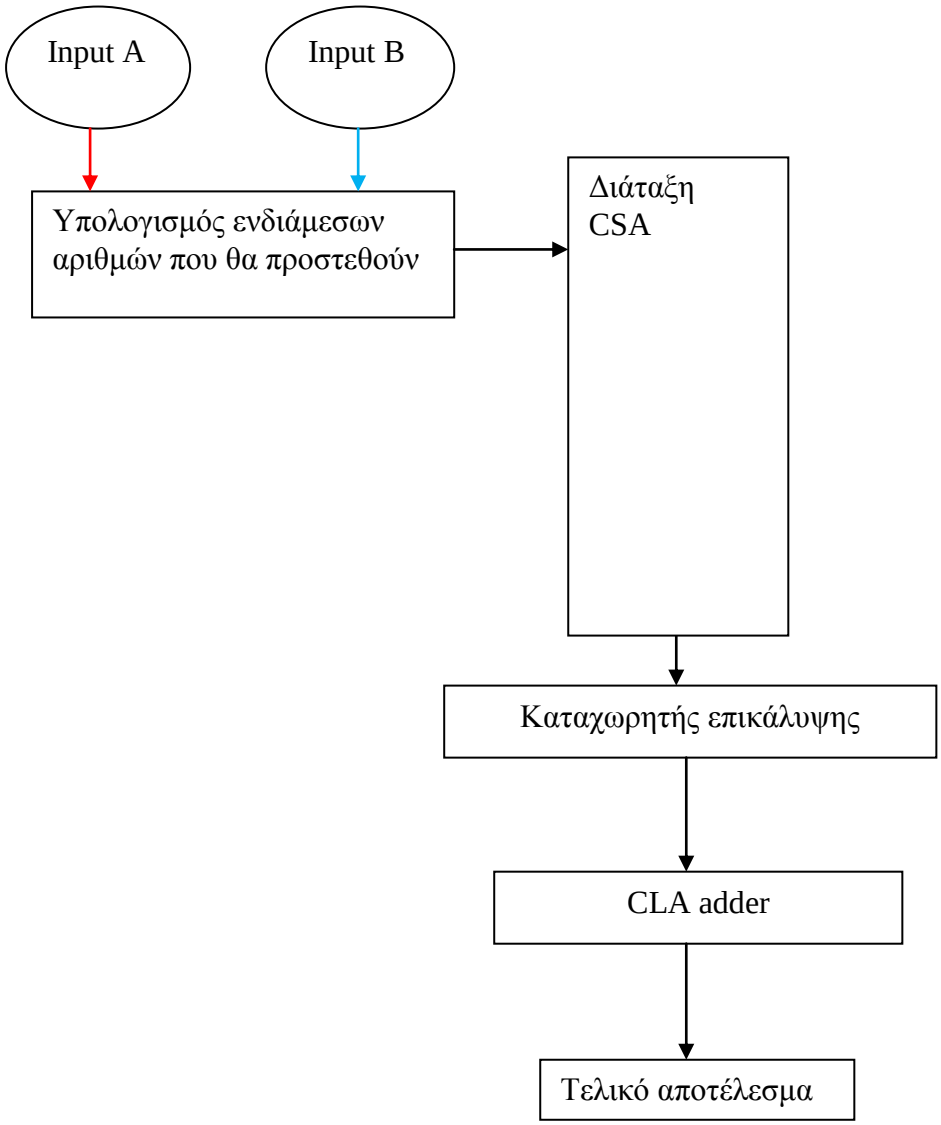


Το διάγραμμα της μονάδας πρόσθεσης αριθμών κινητής υποδιαστολής

Μονάδα πολλαπλασιασμού αριθμών σταθερής υποδιαστολής : Για τον σχεδιασμό της μονάδας αυτής επιλέξαμε να υλοποιήσουμε τον μη επαναληπτικό αλγόριθμο booth που εξετάζει 3 ψηφία κάθε φορά. Ο αλγόριθμος αυτός χρειάζεται μόλις 2 κύκλου μηχανής για να ολοκληρωθεί σε αντίθεση με τον επαναληπτικό booth που χρειάζεται 16. Επιπλέον η επικάληψη της μονάδας είναι πολύ πιο εύκολη με την υλοποίηση του αλγορίθμου αυτού. Το σχεδιαστικό trade off είναι η μεγάλη περιπλοκότητά του και το γεγονός ότι το τελικό κύκλωμα καταλαμβάνει αρκετά περισσότερο χώρο.

Ο τρόπος λειτουργίας του συνοψίζεται παρακάτω. Ο κλασσικός booth υπολογίζει τους αριθμούς που θα προστεθούνε στο μερικό γινόμενο με ρυθμό έναν ανά κύκλο

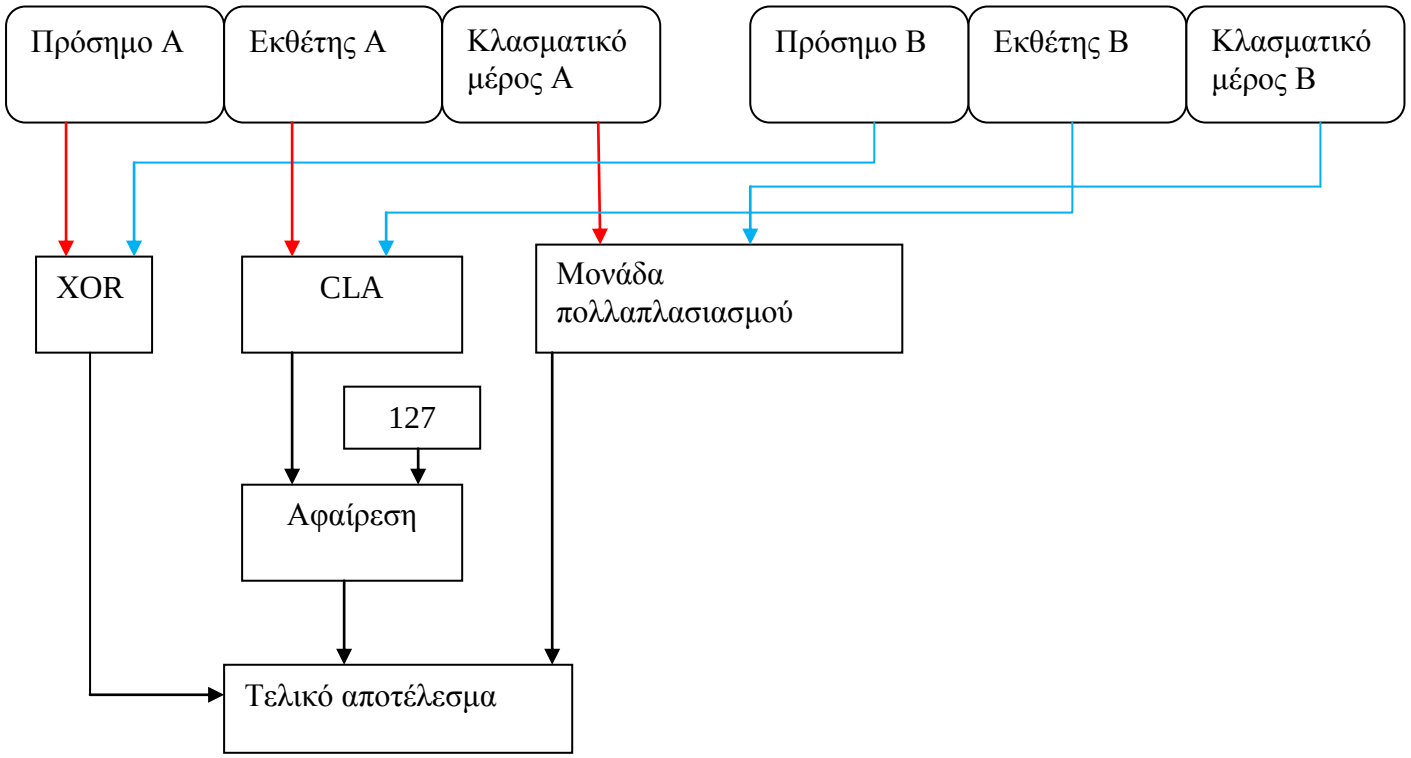
μηχανής. Όμως οι αριθμοί αυτοί μπορούνε να υπολογιστούν απ' ευθείας από τους αρχικού τελεστέους. Έτσι αρχικά υπολογίζουμε αυτούς τους 16 αριθμούς και στη συνέχεια τους προσθέτουμε μεταξύ τους ανά τρεις σε μία διάταξη αθροιστών διατήρησης κρατουμένων (carry save adders, CSA) ελαχιστοποιώντας παράλληλα τον αριθμό επιπέδων που χρειαζόμαστε. Στον $2^{ο}$ κύκλο μηχανής παίρνουμε τα 2 αποτελέσματα που παράγονται από την διάταξη carry save adder και τα προσθέτουμε σε έναν αθροιστή πρόβλεψης κρατουμένου για να πάρουμε το τελικό αποτέλεσμα. Να σημειώσουμε ότι μεταξύ των διαδοχικών προσθέσεων του $1^{ου}$ κύκλου μηχανής γίνονται και οι απαραίτητες ολισθήσεις των ενδιάμεσων αποτελεσμάτων για την παραγωγή του σωστού τελικού αποτελέσματος. Οι ολισθήσεις αυτές δεν γίνονται μέσο κάποιου κυκλώματος αλλά απλά μετακινώντας τα bits που μας ενδιαφέρουν.



Διάγραμμα μονάδας πολλαπλασιασμού σταθερής υποδιαστολής

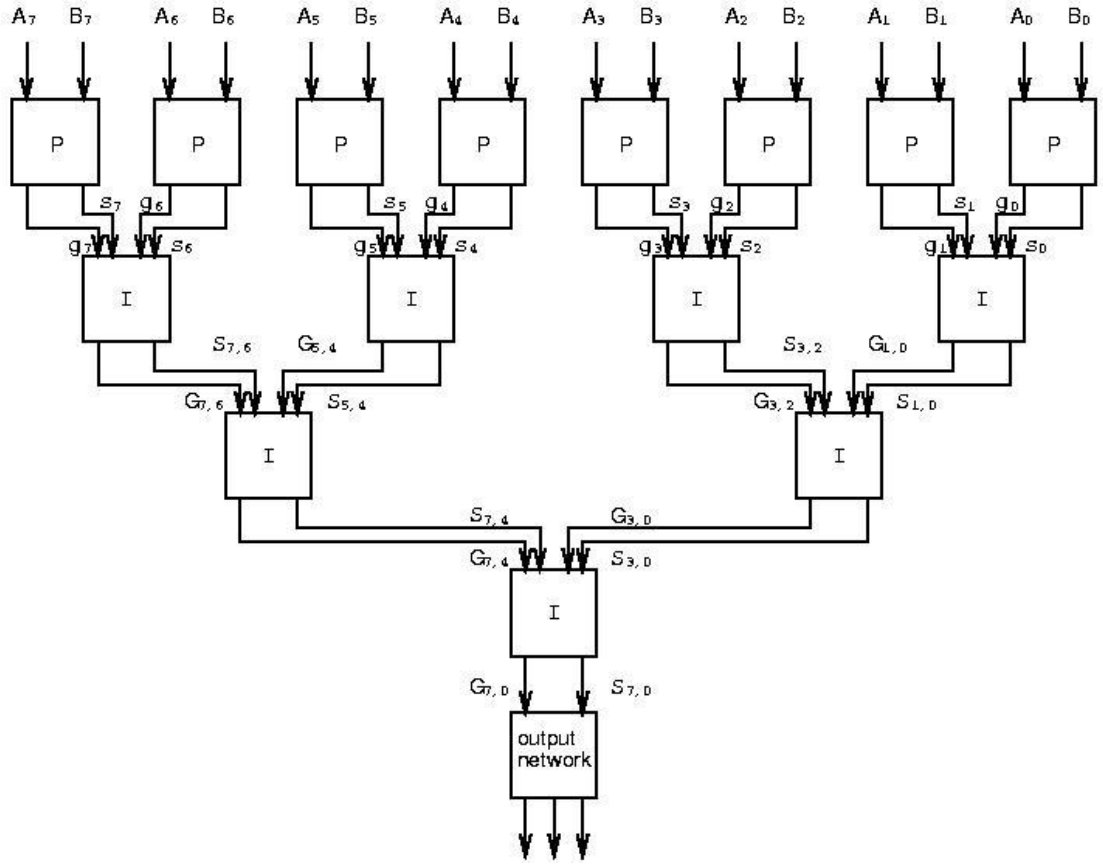
Μονάδα πολλαπλασιασμού κινητής υποδιαστολής: Ο πολλαπλασιασμός κινητής υποδιαστολής είναι γενικά μία δαπανηρή πράξη. Στην συγκεκριμένη περίπτωση ακολουθούμε την κλασσική υλοποίηση. Τα κλασματικά μέρη των αριθμών πολλαπλασιάζονται στην μονάδα πολλαπλασιασμού σταθερής υποδιαστολής που παρουσιάσαμε παρά πάνω. Οι εκθέτες των αριθμών προσθέτονται σε έναν αθροιστή πρόβλεψης κρατουμένου ενώ στη συνέχεια αφαιρούμε από το αποτέλεσμα τον αριθμό 127, δηλαδή την πόλωση ώστε ο τελικός εκθέτης να είναι μία φορά πολωμένος και όχι δύο. Τέλος το πρόσημο του τελικού αριθμού βγαίνει απλά από μία λογική πράξη XOR μεταξύ των δύο προσήμων των αρχικών αριθμών.

Οι δύο αριθμοί ακολουθούνε την δομή του προτύπου ieee 754 για την κινητή υποδιαστολή και η πράξη ολοκληρώνεται σε 3 κύκλους μηχανής.



Διάγραμμα μονάδας πολλαπλασιασμού κινητής υποδιαστολής

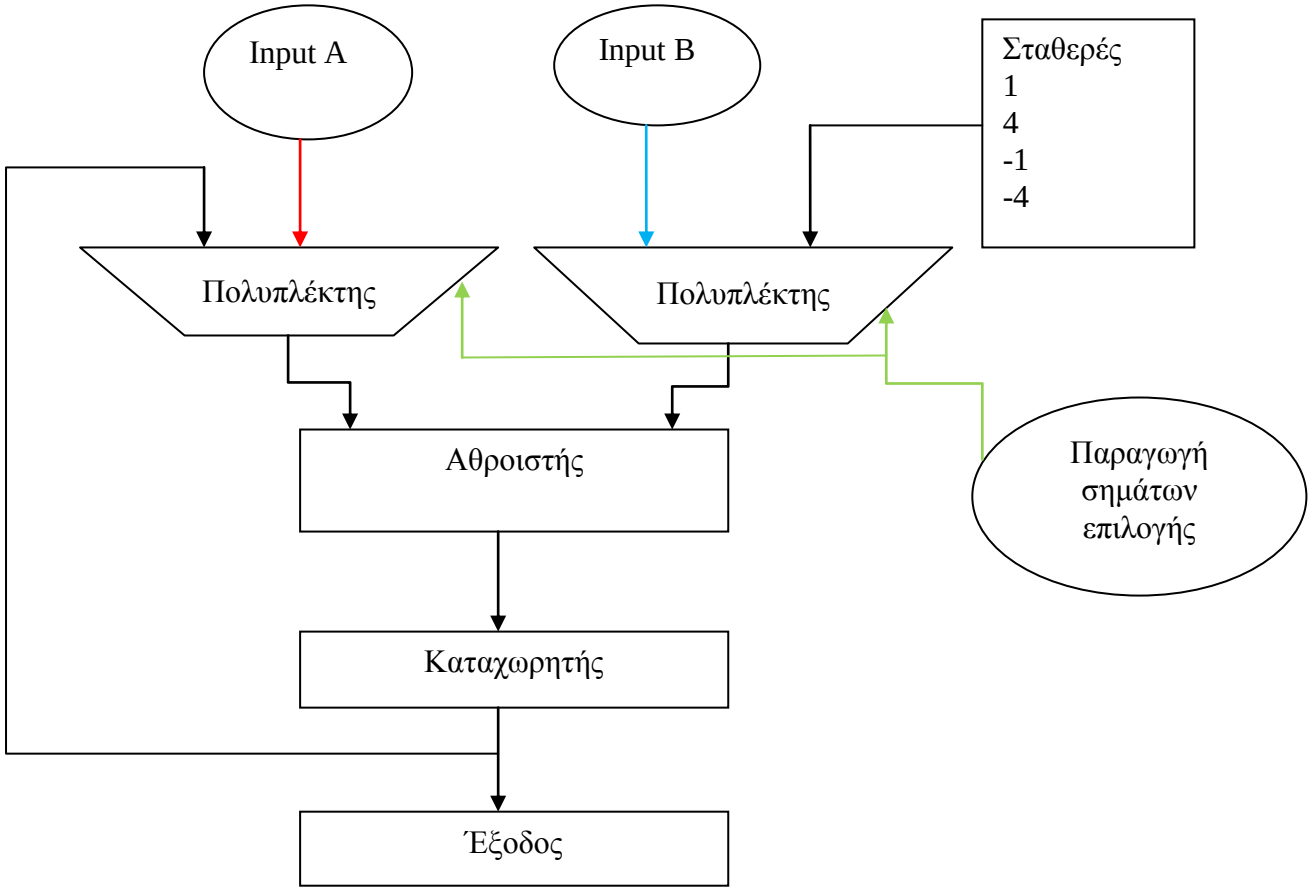
Μονάδα σύγκρισης: Η λειτουργική μονάδα αυτή συγκρίνει 2 αριθμούς 32 bits σε έναν κύκλο μηχανής. Για την σύγκριση αυτή χρησιμοποιούμε έναν κύκλωμα βασισμένο σε συγκριτή πρόβλεψης (look ahead prediction) ο οποίος υλοποιεί λογικές πράξεις μεταξύ των bits των δύο αριθμών για να βγάλει το τελικό αποτέλεσμα. Η παραγωγή του αποτελέσματος γίνεται σε έναν κύκλο μηχανής.



Διάγραμμα μονάδας σύγκρισης χρησιμοποιώντας το look ahead comperation

Μονάδα μνήμης: Στη μονάδα μνήμης γίνεται ο υπολογισμός της τελικής διεύθυνσης προσπέλασης, η οποία προσπέλαση γίνεται σε μία block ram που βρίσκεται εκτός του κυλώματος αυτού. Έτσι για τον υπολογισμό αυτό χρησιμοποιούμε έναν αθροιστή πρόβλεψης κρατουμένου που υπολογίζει το αποτέλεσμα σε ένα κύκλο μηχανής. Όμως η μονάδα αυτή υποστηρίζει και τις επιπλέων λειτουργίες (όπως η lwi1,lwd4 κτλπ) οι οποίες προσαυξάνουν τη διεύθυνση που υπολογίστηκε κατά μία σταθερά σε

κάθε κύκλο μηχανής. Για τη υποστήριξη των εντολών αυτών χρησιμοποιούμε τον ίδιο αθροιστή με την βοήθεια ενός καταχωρητή επικάλυψης για την αποθήκευση του αποτελέσματος και πολυπλεκτών για την επιλογή εισόδων του αθροιστή. Με τον τρόπο αυτό εάν θέλουμε απλά να υπολογίσουμε την τελική διεύθυνση προσπέλασης μνήμης βάζουμε σαν εισόδους στον αθροιστή τους δύο τελεστέους ενώ αν θέλουμε να χρησιμοποιήσουμε την περίπλοκη λειτουργία της αυτόματης προσαύξησης, ανακατευθύνουμε την σταθερά που μας ενδιαφέρει και την έξοδο του καταχωρητή επικάληψης στον αθροιστή.



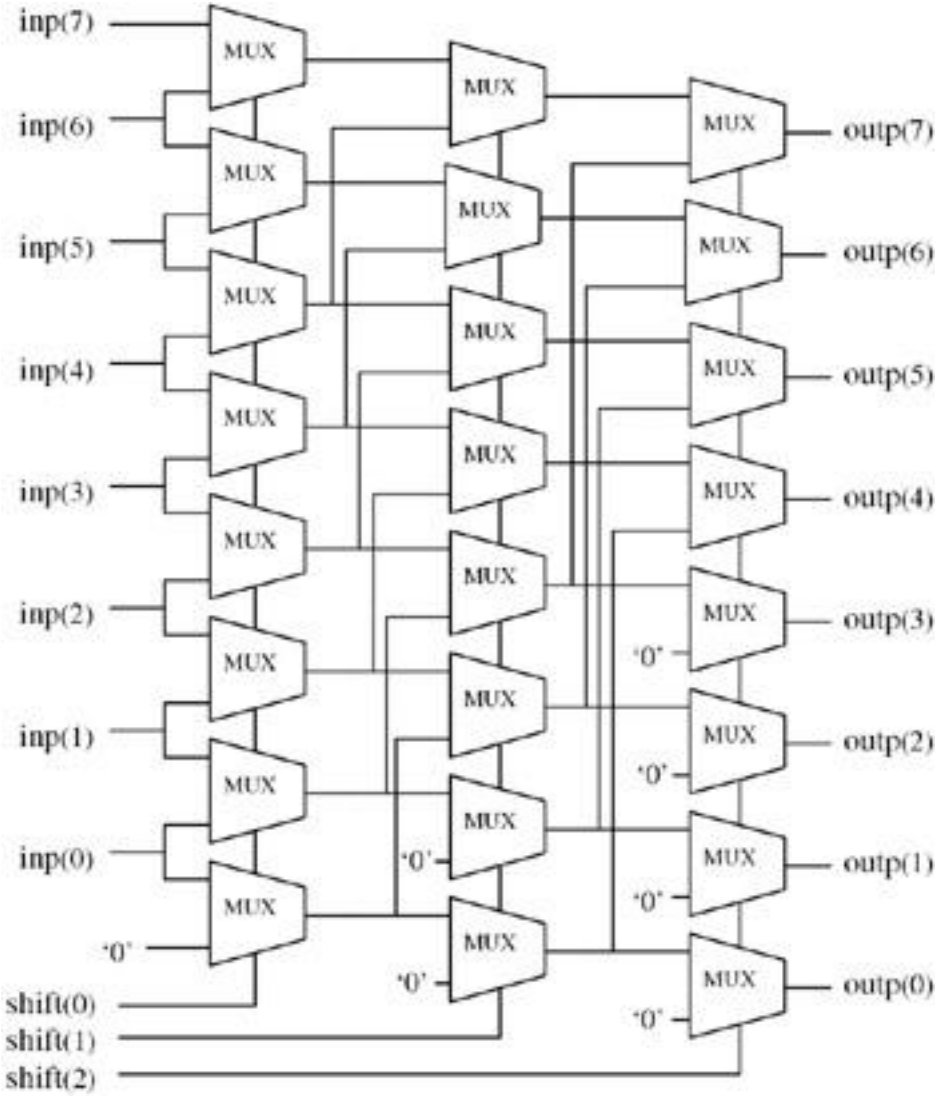
Διάγραμμα μονάδας μνήμης

Μονάδα μετατροπής αριθμού: Η μονάδα αυτή μετατρέπει έναν αριθμό κινητής υποδιαστολής σε ακέραιο αριθμό και αντίστροφα. Λόγο της δομής της μονάδας αυτής είναι πολύ εύκολη η διάσπασή της σε 2 διαφορετικές λειτουργικές μονάδες που η κάθε μία εκτελεί την μία από τις παρά πάνω λειτουργίες.

Η μετατροπή του αριθμού γίνεται χρησιμοποιώντας κυρίως hardwired σήματα (σήματα που έχουν μόνιμα σταθερή τάση) στου αλγορίθμους, δίνοντας μία πιο στατική και πιο γρήγορη υλοποίηση.

Μονάδα ολίσθησης : Η μονάδα αυτή αποτελείται από 3 μικρότερες λειτουργικές μονάδες : Μία μονάδα για την αριστερή ολίσθηση, μία για την δεξιά λογική ολίσθηση (το πρόσημο του αριθμού δεν διατηρείται) και μία για τη δεξιά αριθμητική ολίσθηση (το πρόσημο του αριθμού διατηρείται). Στο τέλος τα τρία αυτά αποτελέσματα οδηγούνται μέσω ενός πολυπλέκτη στην έξοδο. Ο πολυπλέκτης αυτός έχει σαν σήμα επιλογής τον κωδικό της πράξης που πρόκειται να γίνει.

Η ολίσθηση του κάθε αριθμού δεν γίνεται μέσω ενός καταχωρητή ολίσθησης καθώς αυτό παίρνει πάνω από έναν κύκλο μηχανής. Αντί αυτού μία διάταξη από πολυπλέκτεες χρησιμοποιείται. Στις εισόδους του κάθε πολυπλέκτη οδηγούνται δύο bits. Το πρώτο bit είναι το bit του αριθμού εάν δεν ολισθηθεί ενώ το 2ο bit είναι το bit του αριθμού εάν αυτός ολισθηθεί. Το σήμα επιλογής έχει τέτοια τιμή ώστε να υποστηρίζει τη πράξη της ολίσθησης. Να σημειωθεί ότι το τελικό αποτέλεσμα βγαίνει σε έναν κύκλο μηχανής αλλά σε πολλά επίπεδα. Συγκεκριμένα στο πρώτο επίπεδο πολυπλεκτών ο αριθμός ολισθένεται κατά 1 bit. Στο $2^{ο}$ επίπεδο ο αριθμός ολισθένεται κατά 2 bits. Στο $3^{ο}$ επίπεδο κατά 4, στο $4^{ο}$ επίπεδο κατά 8 και στο $5^{ο}$ επίπεδο κατά 16. Παρακάτω δίνεται ένα διάγραμμα ενός ολισθητή (barrel shifter) των 8 bits που μπορεί να ολισθήσει αριστερά έναν αριθμό.



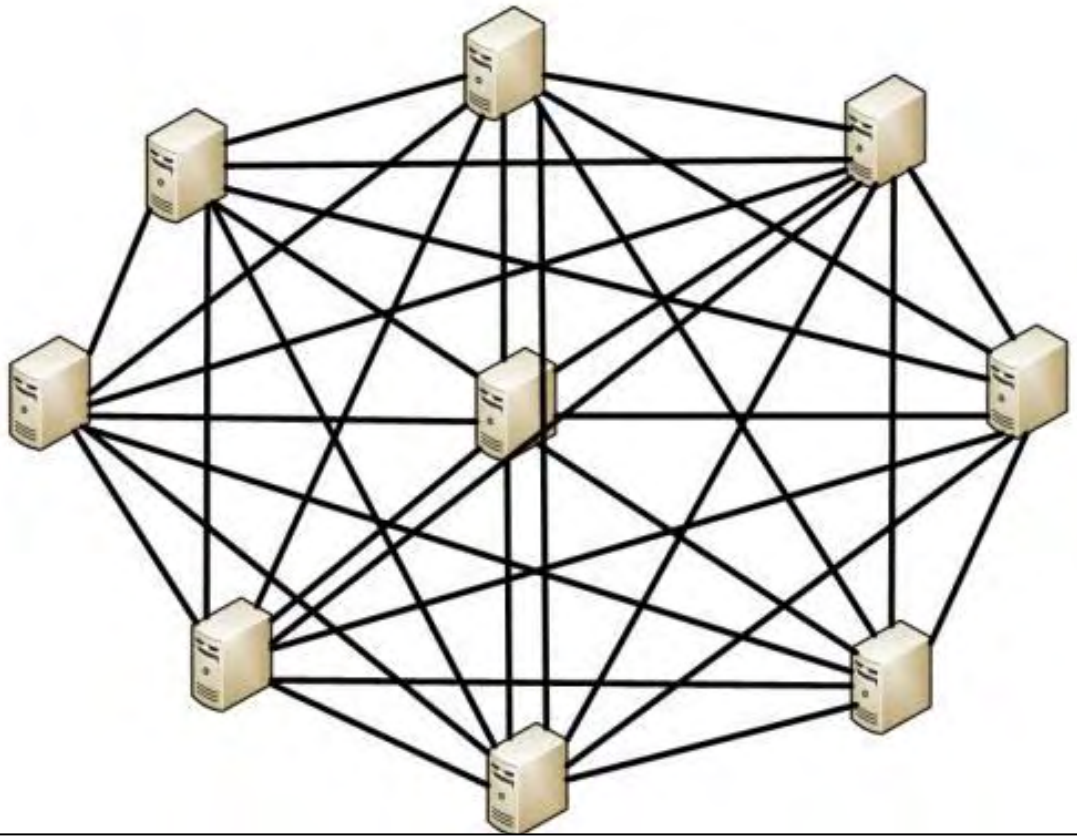
Το διάγραμμα ενός barrel shifter 8 bits για αριστερή ολίσθηση ενός αριθμού

5.2.2 Η τοπολογία των κόμβων και η οργάνωσή τους

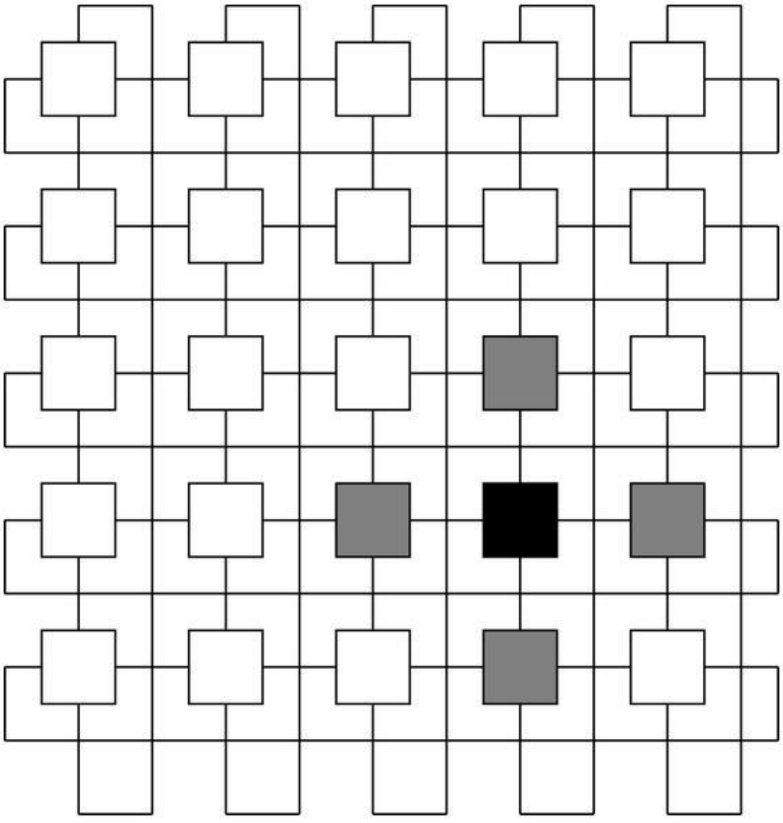
Έχοντας τελειώσει την σχεδίαση των λειτουργικών μονάδων, πρέπει να επιλέξουμε την οργάνωση των κόμβων καθώς και την τοπολογία τους. Οι κόμβοι είναι υπεύθυνοι για την εκτέλεση εντολών του βρόγχου. Ο κάθε κόμβος θα εκτελεί συνεχώς μία εντολή χωρίς να αναλαμβάνει άλλη λειτουργία κατά τη διάρκεια της εκτέλεσης. Επιπλέον θέλουμε η ολοκλήρωση κάθε εντολής να γίνεται με ρυθμό μίας εντολής ανά κύκλο μηχανής όπου θα τελειώνει και μία επανάληψη του βρόγχου. Έτσι ο κάθε κόμβος θα πρέπει να είναι ικανός να εκτελεί μία εντολή κατά την εκτέλεση μίας εφαρμογής αλλά πάνω από μία εντολή γενικά, ώστε να υπάρχει η δυνατότητα της εύκολης δρομολόγησης των εντολών (scheduling).Η τοπολογία των κόμβων θα πρέπει να είναι τέτοια ώστε να υποστηρίζονται όλες η δυνατές ροές δεδομένων από τον έναν κόμβο στον άλλο. Οι κόμβοι μεταξύ τους συνδέονται με άμεσες συνδέσεις, οπότε θα πρέπει να διασφαλίσουμε ότι το σύστημά μας θα μπορεί να παροχετεύει τα ενδιάμεσα παραγόμενα αποτελέσματα στους επιμέρους κόμβους ανεξάρτητα με τις εξαρτήσεις εντολών και τη ροή δεδομένων της εφαρμογής.

Για να πλήρη τα παραπάνω το σύστημά μας θα πρέπει να υποστηρίζει την κατάλληλη τοπολογία κόμβων. Είναι σαφές ότι η “ένα προς ένα” ,γνωστή και ως mesh τοπολογία των κόμβων θα κατέληγε σε εξαιρετικά περίπλοκο και αργό σύστημα. Η διάταξη torus που υποστηρίζεται και από τον trips απορρίφτηκε γιατί ο αριθμός των κόμβων είναι σχετικά μικρός για να προσθέσουμε την καθυστέρηση που προσθέτει η διάταξη αυτή. Παρακάτω δίνεται ένα παράδειγμα της διάταξης torus. Η διάταξη αυτή αν και καταλήγει σε σχετικά απλό σχεδιασμό των κόμβων, ο αριθμός των hops (μεταπηδήσεων) των δεδομένων για να φτάσουν στο επιθυμητό προορισμό είναι σχετικά μεγάλος και άρα η καθυστέρησή του το ίδιο.

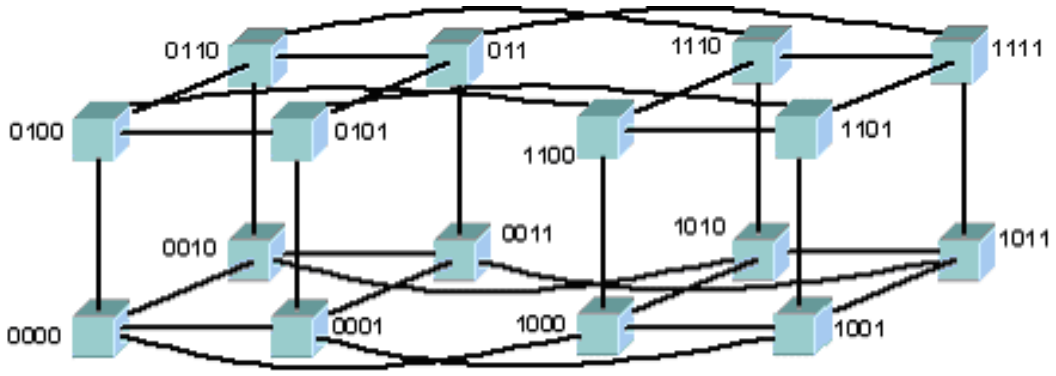
Η τοπολογία που ελαχιστοποιεί τον αριθμό των hops που απαιτούνται για την μετάδοση της πληροφορίας διαμέσου των κόμβων της είναι η τοπολογία hypercube. Εμείς επιλέξαμε μία ελαφρώς αλλαγμένη μορφής της, την enhanced hypercube καθώς πληρεί όλα τα παραπάνω χαρακτηριστικά που θέσαμε. Η enhanced hypercube τοπολογία χρησιμοποιείται κυρίως σε δίκτυα υπολογιστών και σε συστήματα υψηλών επιδόσεων με μικρό αριθμό κόμβων ή λειτουργικών μονάδων.



Η τοπολογία mesh



Η τοπολογία torus



Η τοπολογία hypercube

Έχοντας επιλέξει την τοπολογία κατά την οποία θα τοποθετηθούνε οι επιμέρους κόμβοι, πρέπει στη συνέχεια να επιλέξουμε τον τύπο και αριθμό των λειτουργικών μονάδων που ο κάθε κόμβος θα φιλοξενεί. Για την απόφαση αυτή θα πρέπει να λάβουμε υπόψη μας την σημαντικότητα της κάθε λειτουργικής μονάδας ως προς την εμφάνιση κάθε πράξης. Για παράδειγμα η πράξη της πρόσθεσης θα πρέπει να υποστηρίζεται σε κάθε κόμβο ενώ ο πολλαπλασιασμός, σαν πράξη λιγότερα συχνά εμφανιζόμενη, θα υποστηρίζεται από λιγότερους κόμβους.

Έτσι καταλήξαμε στην δημιουργία των παρακάτω κόμβων :

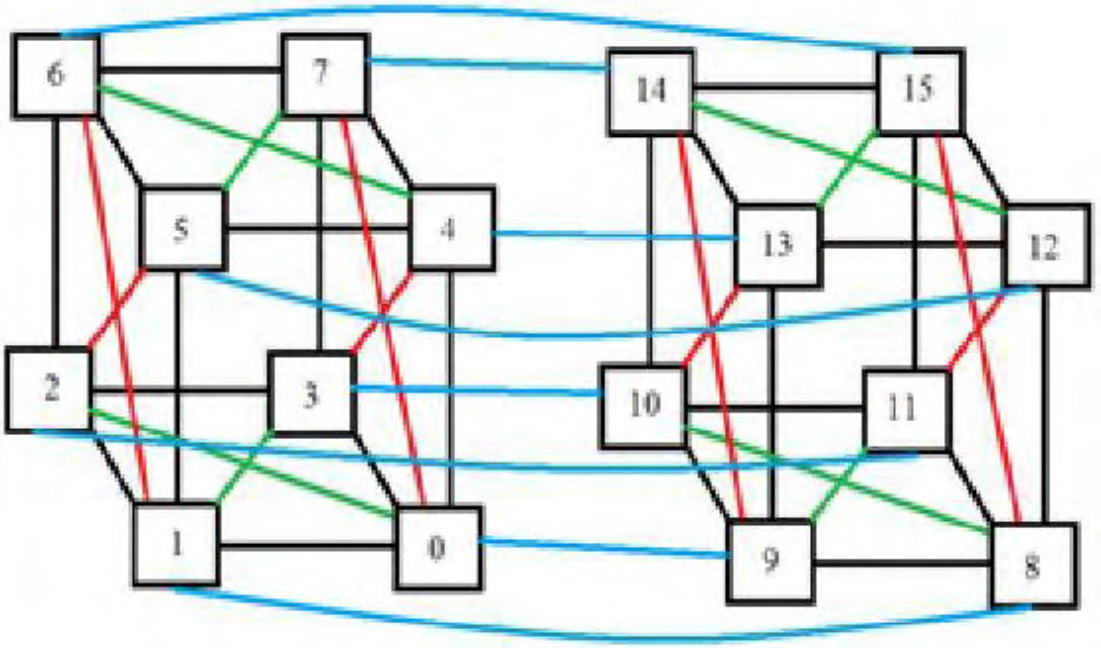
Κόμβος alu: Ο κόμβος αυτός περιλαμβάνει μία alu (αριθμητική και λογική μονάδα ακαιρέων) καθώς και μία μονάδα ολίσθησης ώστε να μπορεί να εκτελεί τις αντίστοιχες πράξεις.

Κόμβος πρόσθεσης κινητής υποδιαστολής: Ο κόμβος αυτός περιλαμβάνει μία αριθμητική και λογική μονάδα, μία μονάδα πρόσθεσης ακαίρεων αριθμών, μία μονάδα μετατροπής αριθμών σε ακεραίους και μία μονάδα πρόσθεσης ακεραίων αριθμών κινητής υποδιαστολής. Η επιλογή αυτή δεν είναι τυχαία καθώς και οι δύο μονάδες χρησιμοποιούνε αθροιστές πρόβλεψης κρατουμένου, οπότε ένα αθροιστής μπορεί να χρησιμοποιηθεί και για τις δύο λειτουργικές μονάδες για οικονομία χώρου.

Κόμβος πολλαπλασιασμού: Ο κόμβος αυτός περιέχει μία μονάδα πολλαπλασιασμού ακεραίων αριθμών, μία μονάδα πολλαπλασιασμού κινητής υποδιαστολής, μία μονάδα μετατροπής αριθμών σε αριθμούς κινητής υποδιαστολής και μία μονάδα πρόσθεσης

ακαιρέων αριθμών. Για εξοικονόμηση χώρου χρησιμοποιούμε τον ίδιο αθροιστή πρόβλεψης κρατουμένου και τον ίδιο πολλαπλασιαστή όπου αυτός χρησιμοποιείται.
Κόμβος μνήμης : Ο κόμβος αυτός υποστηρίζει εντολές πρόσθεσης ακεραίων και προσπέλασης μνήμης. Επομένως εμπεριέχει την μονάδα μνήμης και έναν αθροιστή πρόβλεψης κρατουμένου. Η ίδια στρατηγική χρησιμοποιείται κι εδώ μιας και ο αθροιστής πρόβλεψης κρατουμένου είναι απαραίτητος και στη μονάδα μνήμης και στη μονάδα πρόσθεσης ακεραίων. Έτσι τον τοποθετούμε μία μόνο φορά καθώς η μονάδα θα κάνει μία από τις δύο παραπάνω πράξεις κάθε φορά.
Κόμβος διακλάδωσης : Ο κόμβος αυτός έχει μία μονάδα πρόσθεσης και μία μονάδα σύγκρισης για τον υπολογισμό συνθηκών διακλαδώσεων.

Για τη τοποθέτηση των κόμβων στην τοπολογία που επιλέξαμε θέσαμε ορισμένα κριτήρια για να διευκολυνθούμε στην επίλυση του προβλήματος. Τα κριτήρια αυτά υποδιαιρούνται σε δύο βασικές κατηγορίες: Οι κόμβοι που σχετίζονται με την προσπέλαση μνήμης θα πρέπει να βρίσκονται σχετικά κοντά και για την διευκόλυνση της άμεσης ένωση τους με την μνήμη και για την βέλτιστη τοποθέτησή τους χωρικά σε σχέση με την κρυφή μνήμη (cache). Άλλο κριτήριο είναι η κάλυψη όλων των δυνατών συνδυασμών των εντολών. Θα πρέπει το σύστημα να μπορεί να εκτελεί όλους τους βρόγχους ανεξάρτητα από τον τύπο εντολών , αρκεί ο αριθμός τους να είναι 16 και κάτω. Έτσι οι ενώσεις μεταξύ των κόμβων θα πρέπει να γίνουν έτσι ώστε τα αποτελέσματα να μπορούν να παροχετεύονται και να καλύπτουμε μία ευρεία περιοχή των κόμβων.
Τελικώς καταλήξαμε στην παρακάτω διάταξη και συνδεσμολογία των κόμβων.



Η τελική τοποθέτηση των κόμβων. Οι κόμβοι 1,3,12,14 είναι κόμβοι πρόσθεσης κινητής υποδιαστολής, οι κόμβοι 2,5,8 είναι κόμβοι πολλαπλασιασμού, οι κόμβοι 0,13,15 είναι κόμβοι αριθμητικών και λογικών πράξεων, οι κόμβοι 4,7,9,10 είναι κόμβοι προσπέλασης μνήμης και οι κόμβοι 6,11 είναι κόμβοι διακλάδωσης.

Αριθμός κόμβου	Τύπος κόμβου	Συνδέσεις
0	Αριθμητικές και λογικές πράξεις	1,2,3,4,7,9
1	Πρόσθεση κινητής υποδιαστολής	0,2,3,5,6,8
2	Πολλαπλασιασμός	0,1,3,5,6,11
3	Πρόσθεση κινητής υποδιαστολής	0,1,2,4,7,10
4	Προσπέλαση μνήμης	0,3,5,6,7,14
5	Πολλαπλασιασμός	1,2,4,6,7,12
6	Διακλάδωση	1,2,4,5,7,15
7	Προσπέλαση μνήμης	0,3,4,5,6,14
8	Πολλαπλασιασμός	1,9,10,11,12,15
9	Προσπέλαση μνήμης	0,8,10,11,13,14
10	Προσπέλαση μνήμης	3,8,9,11,13,14
11	Διακλάδωση	2,8,9,10,12,15

12	Πρόσθεση κινητής υποδιαστολής	5,8,11,13,14,15
13	Αριθμητικές και λογικές πράξεις	4,9,10,12,14,15
14	Πρόσθεση κινητής υποδιαστολής	7,9,10,12,13,15
15	Αριθμητικές και λογικές πράξεις	6,8,11,12,13,14

Σύνοψη των κόμβων, των λειτουργιών τους και το πώς συνδέονται μεταξύ τους

Καθώς όμως ο κάθε κόμβος συνδέεται με παραπάνω από έναν κόμβους, προκύπτουν προβλήματα για την αποθήκευση των ενδιάμεσων αποτελεσμάτων καθώς και για την επιλογή εισόδων στις λειτουργικές μονάδες. Τα προβλήματα αυτά λύνονται με την τοποθέτηση πολυπλεκτών για την επιλογή εισόδων και μίας ουράς τύπου fifo για την αποθήκευση τον προσωρινών αποτελεσμάτων.

Η ουρά θα πρέπει να έχει τόσες θέσεις όσα ενδιάμεσα αποτελέσματα μπορούν να παραχθούν στην χειρότερη περίπτωση μέχρι να χρησιμοποιηθούν από τον κόμβο. Η χειρότερη περίπτωση ορίζεται από πολύ πυκνές εξάρτησης από δεδομένα μεταξύ διαδοχικών εντολών, ειδικά όταν αυτές οι εντολές έχουν χρόνο ολοκλήρωσης πάνω από έναν κύκλο μηχανής. Η ανάλυση χειρότερης περίπτωσης δείχνεται στον παρακάτω πίνακα. Στην περίπτωση αυτή υπάρχει μία αλυσίδα εξαρτήσεων των εντολών, με αποτέλεσμα την καθυστέρηση κάθε μίας από αυτές.

Αριθμός κόμβου	Πράξη	Καθυστέρηση
0	Πρόσθεση ακεραίων	1
1	Πρόσθεση κινητής υποδιαστολής	2
2	Πολλαπλασιασμός κινητής υποδιαστολής	3
3	Πρόσθεση κινητής υποδιαστολής	2

4	Προσπέλαση μνήμης	1
5	Πολλαπλασιασμός κινητής υποδιαστολής	3
6	Πρόσθεση ακεραίων	1
7	Προσπέλαση μνήμης	1
8	Πολλαπλασιασμός κινητής υποδιαστολής	3
9	Προσπέλαση μνήμης	1
10	Προσπέλαση μνήμης	1
11	Πρόσθεση ακεραίων	1
12	Πρόσθεση κινητής υποδιαστολής	2
13	Πρόσθεση ακεραίων	1
14	Πρόσθεση κινητής υποδιαστολής	2
15	Πρόσθεση ακεραίων	1
Συνολική καθυστέρηση		26

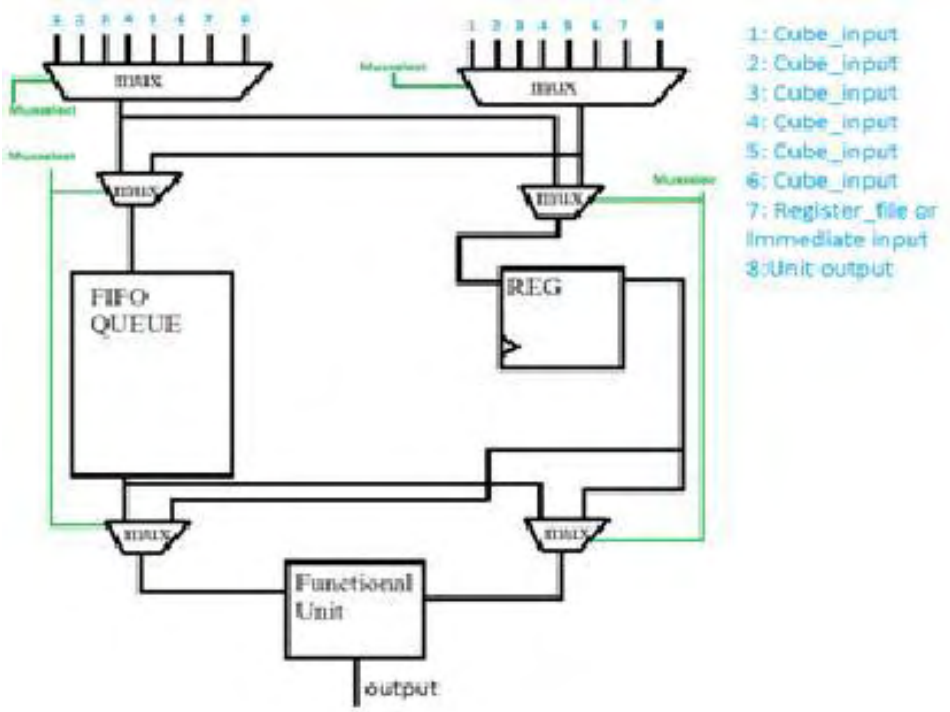
Η καθυστέρηση χειρότερης περίπτωσης

Επομένως η ουρά θα πρέπει να είναι 25 θέσεων (για την καθυστέρηση συμβάλουν 14 κόμβοι, όχι 15 αφού ο ένας περιμένει τα δεδομένα).

Η κάθε λειτουργική μονάδα έχει δύο τελεστέους σαν είσοδο, όμως στην ουρά θα αποθηκεύεται μόνο ο ένας από τους δύο (ο πρώτος) κατά την άφιξή του. Όταν παραχθεί και ο δεύτερος τότε η πράξη μπορεί να αρχίσει την εκτέλεση της οπότε ένας απλός καταχωρητής είναι αρκετός για την αποθήκευση ενός κύκλου της δεύτερης εισόδου.

Αυτό που πρέπει να προσέξουμε είναι η ανακατεύθυνση της πρώτης εισόδου του κόμβου να γίνεται στην ουρά ενώ της δεύτερης στον καταχωρητή. Για τον λόγο αυτό χρησιμοποιούμε πολυπλέκτες που μας απλουστεύουν τον διαχωρισμό αυτό. Επιπλέον θα πρέπει στις μη προσεταιριστικές πράξεις να τειρείται η σειρά των τελεστέων καθώς η πράξη “a-b” είναι διαφορετική από την “b-a”. Επομένως με όποια σειρά και να έρθουν οι τελεστέοι και όπου και αν αποθηκευτούν τελικώς (στην ουρά ή στον

καταχωρητή) θα πρέπει να οδηγηθούν σε συγκεκριμένες εισόδους της λειτουργικής μονάδας.



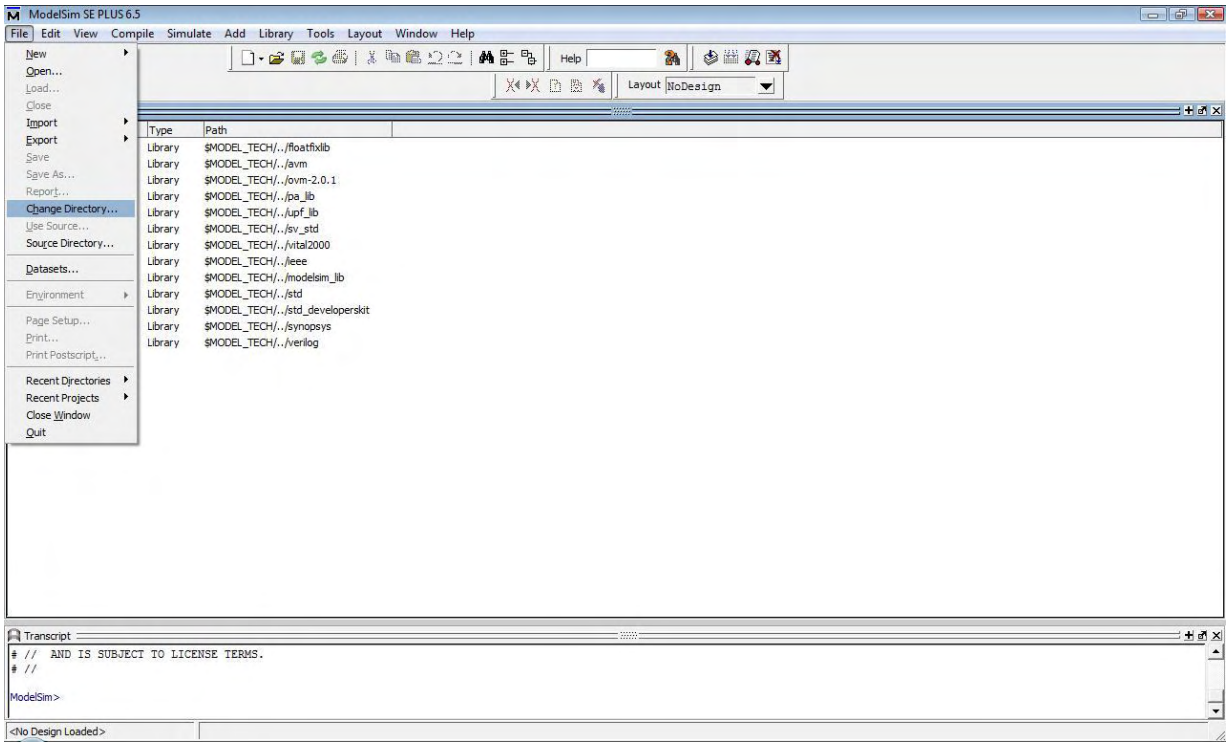
Το εσωτερικό διάγραμμα ενός κόμβου

5.3 Προσομοίωση με το εργαλείο modelsim

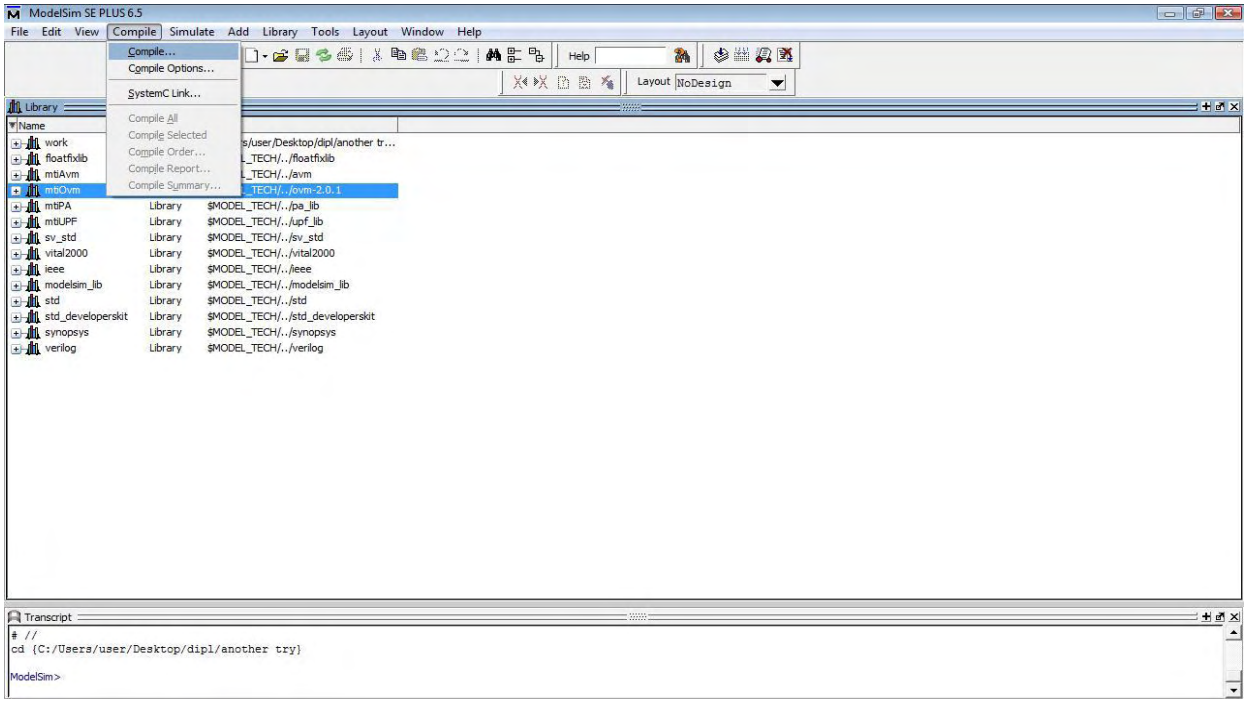
Μετά το τέλος της συγγραφής του συστήματος με τη γλώσσα περιγραφής υλικού Verilog επόμενο βήμα είναι η προσομοίωση της λειτουργίας του. Η διαδικασία της προσομοίωσης είναι η οδήγηση συγκεκριμένων εισόδων στο κύκλωμα με σκοπό την επαλήθευση των εξόδων του. Στο στάδιο αυτό ελέγχεται μόνο η λογική συμπεριφορά του κυκλώματος και αγνοούνται όλες οι καθυστερήσεις των λογικών πυλών. Το ακολουθιακό μέρος του κυκλώματος ελέγχεται και αυτό αγνοώντας καθυστερήσεις σημάτων και άλλους παράγοντες όπως το set up time και hold time των flip flops που χρησιμοποιούνται στους καταχωρητές. Είναι ένα πρώτο βήμα για να σιγουρευτεί ο σχεδιαστής ότι το κύκλωμα που δημιούργησε έχει σωστή λογική συμπεριφορά.

Για την σωστή προσομοίωση του κυκλώματος δημιουργήσαμε ένα κύκλωμα αποκωδικοποίησης των εντολών το οποίο στέλνει τις εντολές και του κωδικούς λειτουργίας των λειτουργικών μονάδων στους επιμέρους κόμβους. Παρακάτω περιγράφουμε βήμα προς βήμα την διαδικασία της προσομοίωσης.

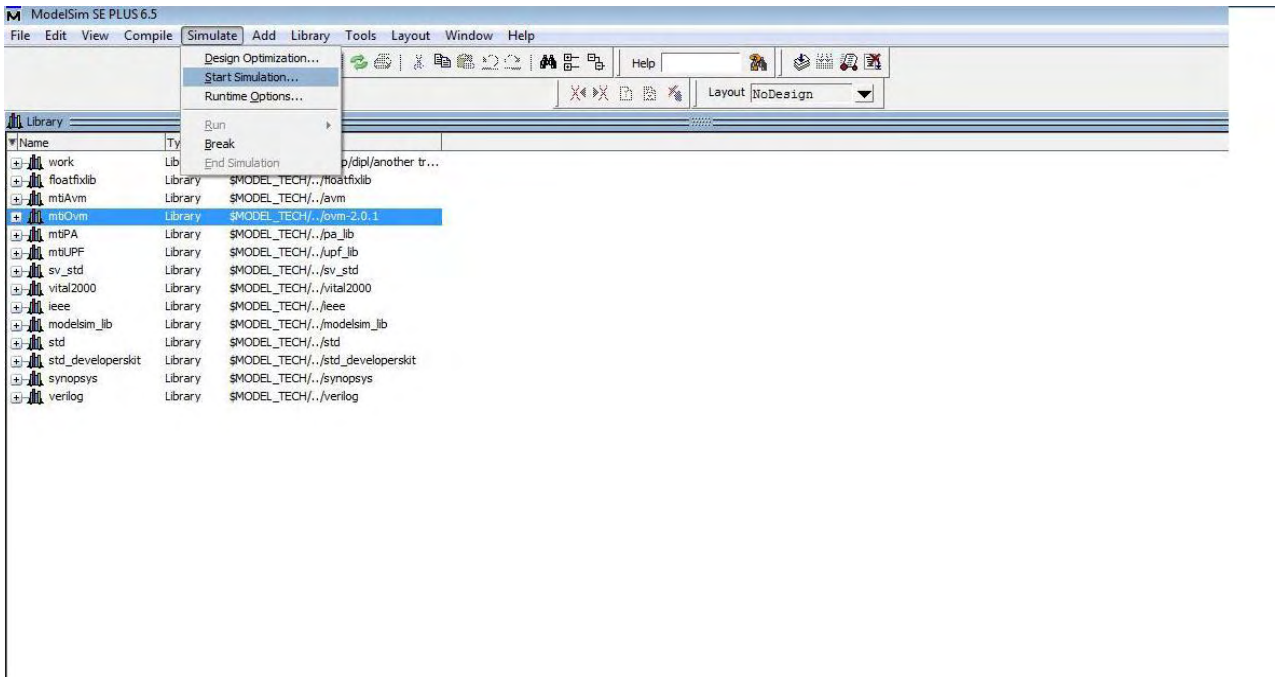
Αρχικά ανοίγουμε το εργαλείο modelsim. Στη συνέχεια επιλέγουμε “file -> change directory” και επιλέγουμε τον φάκελο στον οποίο έχουμε ερχεία Verilog ή Vhdl που θέλουμε να προσομοιώσουμε.

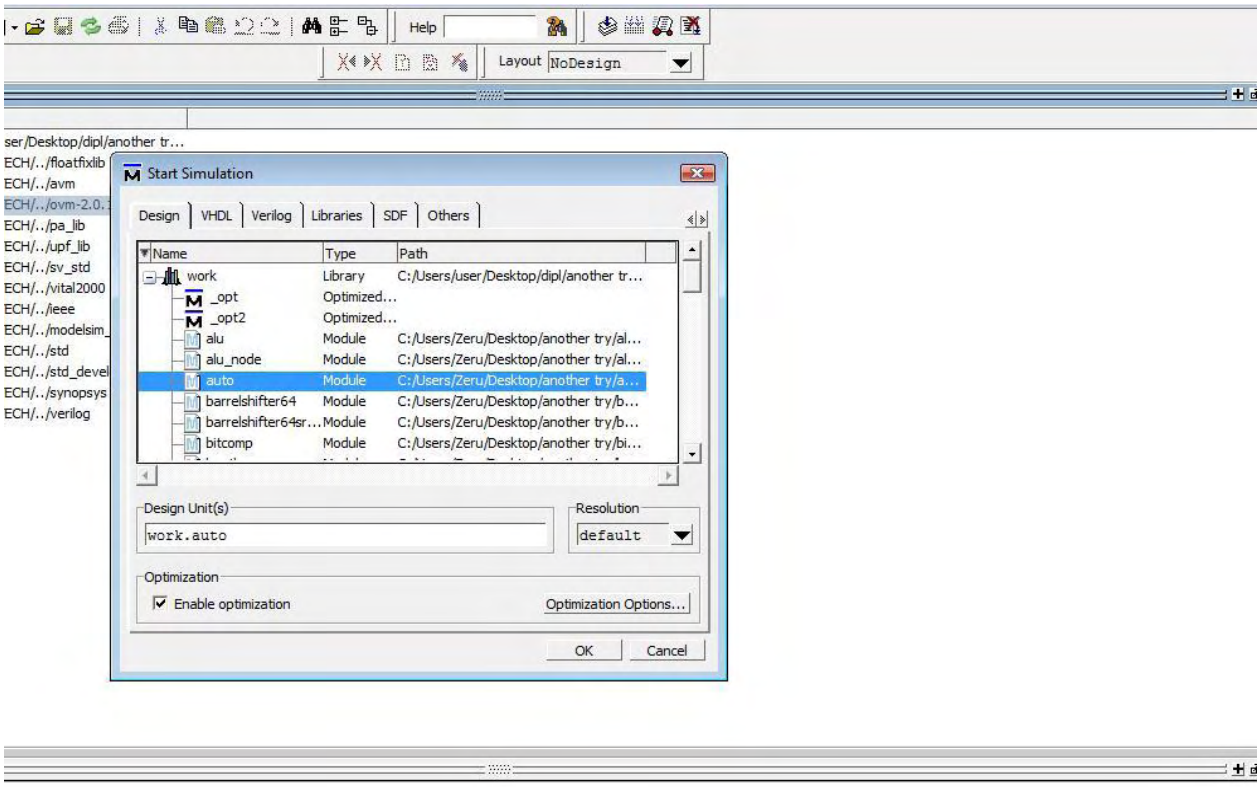


Στη συνέχεια επιλέγουμε “Compile -> compile” και στο παράθυρο που εμφανίζεται επιλέγουμε τα αρχεία πηγαίου κώδικα που θέλουμε να μεταγλωττιστούν και πατάμε “compile”. Εάν η μεταγλώττιση ολοκληρωθεί χωρίς λάθη θα εμφανιστεί αντίστοιχο μήνυμα. Σε διαφορετική περίπτωση ο compiler επισημαίνει τα συντακτικά λάθη του προγράμματος.



Αφού τελειώσει η διαδικασία της μεταγλώττισης θα πρέπει να εκκινήσουμε την προσομοίωση του κυκλώματος. Επιλέγουμε “simulate -> start simulation” και στη συνέχεια το υψηλότερο σε ιεραρχία αρχείο του συστήματός μας (από την βιβλιοθήκη work) και επιλέγουμε “ok".

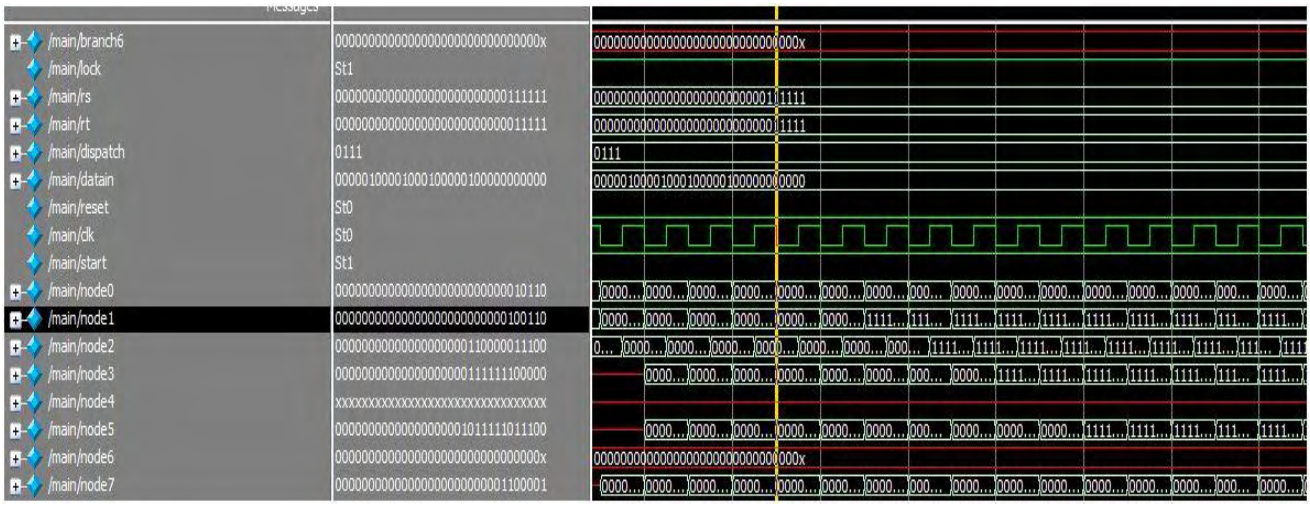




Στη συνέχεια το περιβάλλον του modelsim θα αλλάξει και θα μας εμφανιστεί το περιβάλλον προσομοίωσης. Τώρα έχουμε τρεις διαφορετικούς τρόπους για να προχωρήσουμε στη προσομοίωση του συστήματός μας.

Μπορούμε να εργαστούμε μέσω κονσόλας, κάτι που δεν ενδείκνυται για μεγάλους κώδικες. Με εντολές από εκεί μπορούμε να αρχικοποιήσουμε και να δώσουμε τιμές στα σήματα εισόδου (force <signal_name> <time>) και να αρχίσουμε την προσομοίωση από ένα χρονικό σημείο και μετά (run <time>). Ο άλλος τρόπος είναι να δημιουργήσουμε ένα “.do” αρχείο. Το “.do” αρχείο είναι ένα αρχείο το οποίο λειτουργεί σαν bus script και ουσιαστικά έχει όλες τις εντολές για compile και simulate που θα πληκτρολογούσαμε στη κονσόλα. Αρκεί η εντολή “do <name>.do” για να τρέξει ένα τέτοιο αρχείο το οποίο διευκολύνει και απλουστεύει πολύ την διαδικασία της προσομοίωσης. Εναλλακτικά μπορεί να χρησιμοποιηθεί ένα testbech. Testbench ονομάζεται ένα απλό κύκλωμα που δημιουργείται με σκοπό να τροφοδοτήσει με εισόδους το κύκλωμα που θέλουμε να εξετάσουμε. Στην περίπτωση μας η τεχνική του testbench χρησιμοποιήθηκε. Παρακάτω υπάρχει ένα στιγμιότυπο της προσομοίωσης του κυκλώματος. Παρατηρούμε ότι ο κάθε κόμβος παράγει ένα

αποτέλεσμα ανά κύκλο μηχανής όταν οι βαθμίδες επικάλυψης της αντίστοιχης λειτουργικής μονάδας γεμίσουν.



Στιγμιότυπο της προσομοίωσης. Ο κάθε κόμβος παράγει ένα αποτέλεσμα ανά κύκλο μηχανής

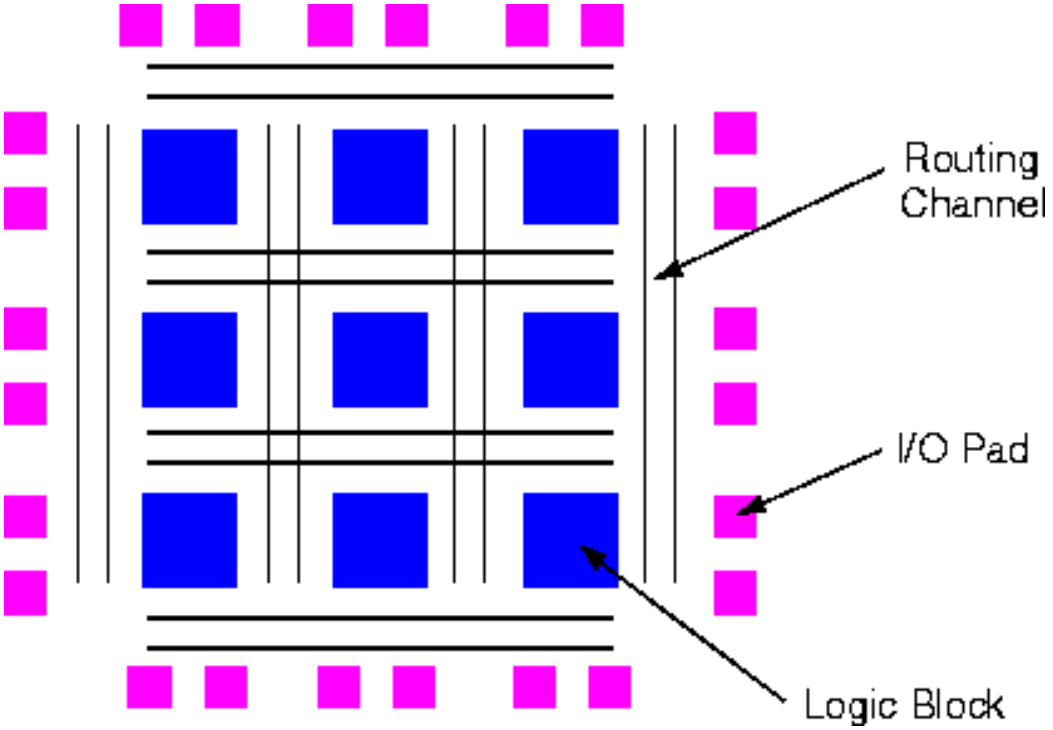
6. Υλοποίηση σε Fpga

Η επιτυχής προσομοίωση ενός συστήματος δεν δίνει πολλές πληροφορίες για το αν το σύστημα αυτό θα έχει την ίδια συμπεριφορά σε ένα πραγματικό κύκλωμα. Όμως επιβεβαιώνει την σωστή συμπεριφορά του κυκλώματος όσον αφορά την Boolean λογική του. Για να χρησιμοποιήσουμε μία fpga για την υλοποίηση του κυκλώματος θα πρέπει να περάσουμε το κύκλωμα από ένα άλλο εργαλείο για να γίνει η διαδικασία γνωστή ως σύνθεση. Προτού όμως επεκταθούμε στις λεπτομέρειες τις διαδικασίας αυτής καλό θα ήταν να εισάγουμε τον αναγνώστη στις βασικές έννοιες μίας fpga.

6.1 Εισαγωγή στις Fpga

Οι Fpgas (Field-programmable gate arrays) είναι προκατασκευασμένα ολοκληρωμένα ψηφιακά κυκλώματα τα οποία μπορούνε να προγραμματιστούνε έτσι ώστε να υλοποιούνε διάφορες λειτουργίες. Ο προγραμματισμός αυτός γίνεται δυναμικά από τον χρήστη όσες φορές θέλει αυτός.

Η κάθε Fpga έχει τα λεγόμενα μπλοκ λογικής, που είναι στοιχεία που περιέχουν πολλές λογικές πύλες. Κάθε φορά που ο χρήστης προγραμματίζει την fpga ορισμένες λογικές πύλες ενεργοποιούνται και ορισμένες όχι. Στη συνέχεια το μπλοκ λογικής συνδέονται μεταξύ τους με τέτοιο τρόπο ώστε να πληρείται η λειτουργικότητα που θέλει ο χρήστης. Έτσι σχηματίζεται ένα πραγματικό ολοκληρωμένο κύκλωμα που κάνει μία συγκεκριμένη λειτουργία. Λόγο της πολύ μεγάλης παραμετροποίησης που μπορεί κάποιος να πετύχει σε μία fpga καθώς και λόγο του γεγονότος ότι μπορούνε να αναπρογραμματιστούνε (σε αντίθεση με τα ASICs) πολύ συχνά οι fpgas χρησιμοποιούνται για εφαρμογές υψηλών απαιτήσεων και για κυκλώματα υψηλών επιδόσεων. Στη συγκεκριμένη περίπτωση χρησιμοποιήσαμε μία SPARTAN-6 της Xilinx η οποία αν και εύχρηστη χρησιμοποιείται κυρίως για ακαδημαϊκούς σκοπούς.



Η οργάνωση μίας fpga. Οι μπλε περιοχές είναι τα μπλοκ λογικής

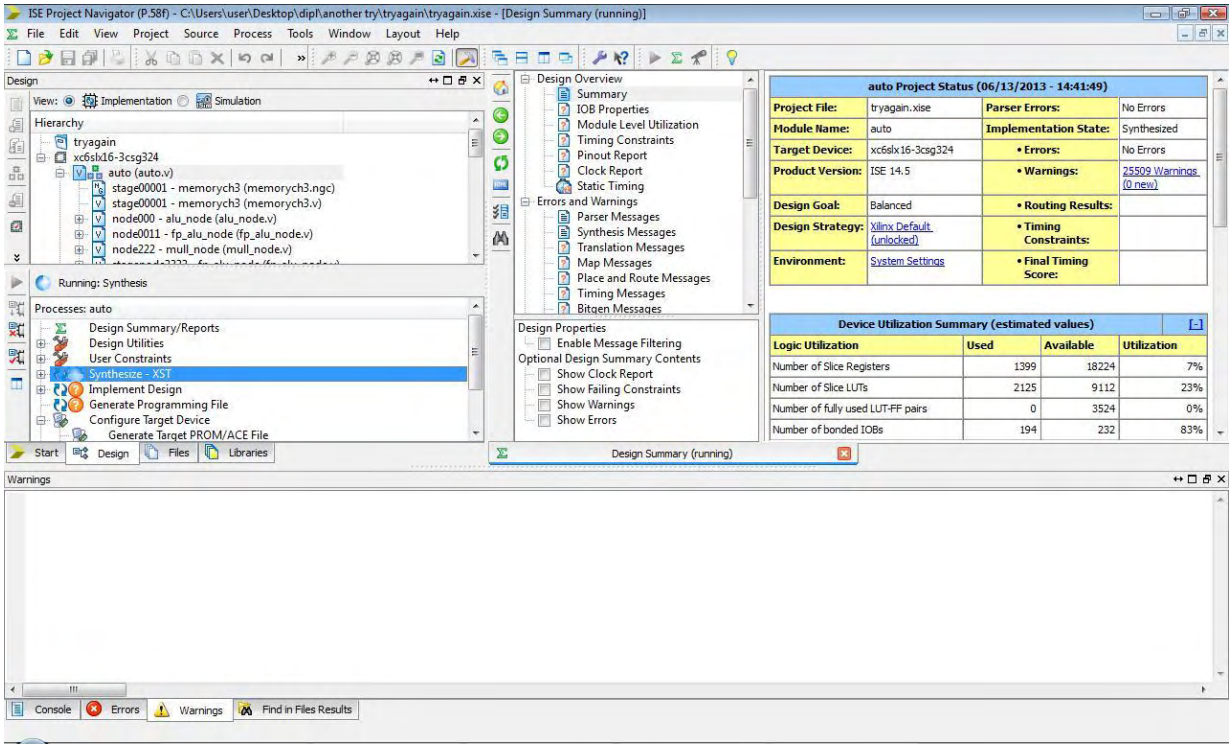
Η Spartan-6 ,η Fpga στη οποία υλοποιήσαμε το σύστημά μας

6.2 Σύνθεση

Η διαδικασία της σύνθεσης είναι υπεύθυνη για τον συντακτικό έλεγχο του κώδικα και τη δημιουργία του λεγόμενου “post-synthsis simulation”. Η διαδικασία αυτή καλείται μέσω του ISE εργαλείου του Xilinx. Αφού τελειώσει, αν το εργαλείο δεν βρει συντακτικά λάθη στον κώδικα τότε παρέχει στον χρήστη σημαντικές πληροφορίες για το κύκλωμα του. Οι πληροφορίες αυτές έχουν σχέση με τον χρονισμό του κυλώματος, το critical path delay και τον αριθμό των πόρων που χρησιμοποιεί το κύκλωμα από την fpga.

Κατά την διαδικασία της σύνθεσης ο μεταγλωττιστής ενημερώνει τον χρήστη για ορισμένα warnings τα οποία δεν υπάγονται στα συντακτικά λάθη αλλά είναι παραλήψεις που μπορούν να οδηγήσουν στην δημιουργία ατελές κυκλώματος. Γνωστά warnings είναι σήματα που δεν χρησιμοποιούνται, ασύνδετα μέρη από εξόδους ενός κυκλώματος και μη ολοκληρωμένη “if-else” λογική. Όλα τα παραπάνω δεν υπάγονται στα συντακτικά λάθη όμως μία τέτοια παράληψη είναι ικανή να οδηγήσει σε λανθασμένη έξοδο του κυκλώματος.

Τέλος κατά τη διαδικασία της σύνθεσης παράγεται και το αρχείο net list του κυκλώματος μας. Το αρχείο αυτό είναι ένας μεγάλος κατάλογος ο οποίος περιλαμβάνει όλες της συνδεσμολογίας μεταξύ των λογικών πυλών του κυκλώματος σε μία συγκεκριμένη δομή.



Αρχίζοντας την διαδικασία της σύνθεσης στο εργαλείο ISE

6.3 Υλοποίηση

Η διαδικασία της υλοποίησης είναι το προτελευταίο στάδιο για τον προγραμματισμός μίας fpga. Η διαδικασία αυτή αποτελείται από τρία μέρη: το ”translation”, το “mapping” και το “place and route”.

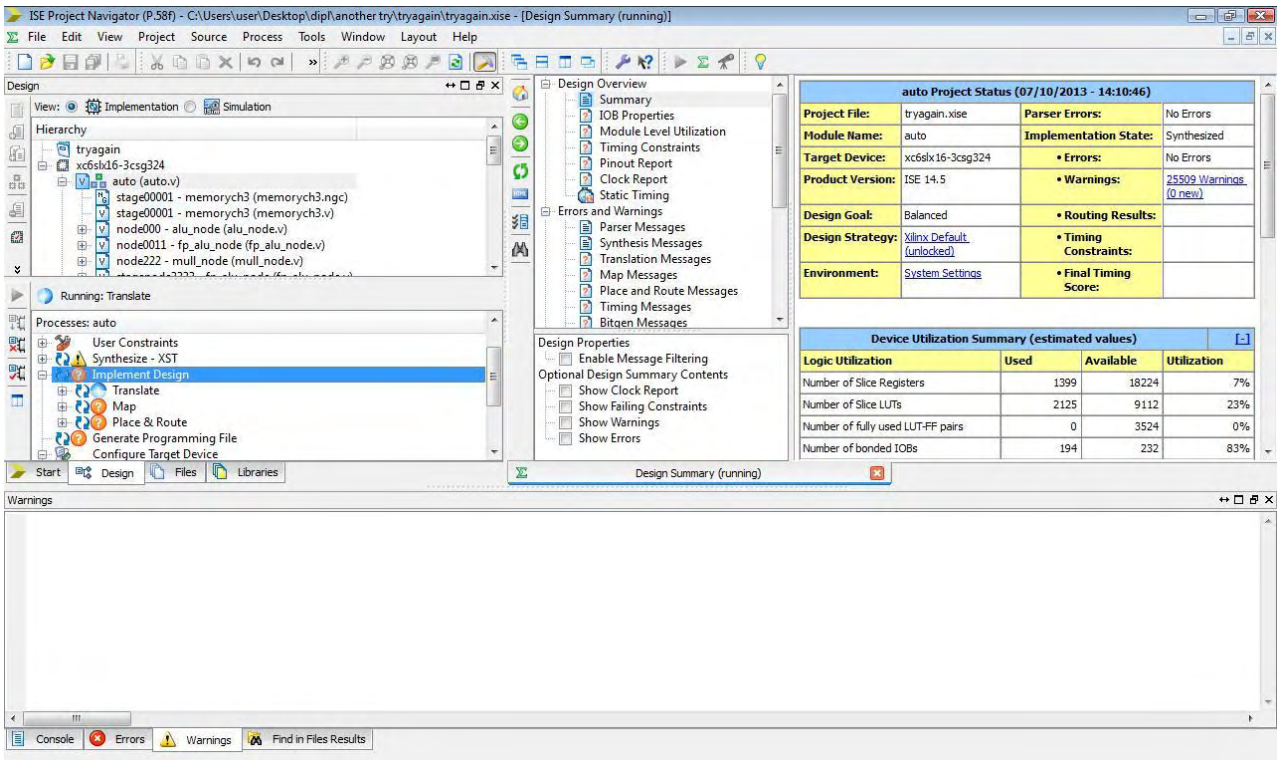
Translation είναι η διαδικασία κατά την οποία το αρχείο net list που έχει παραχθέι στην σύνθεση μετατρέπεται σε λογικές πύλες συμβατές με την fpga που θα χρησιμοποιήσουμε. Κατά την διαδικασία αυτή λαμβάνεται υπ όψη και το .ucf (“user constrains file”) που δημιουργήσαμε. Το αρχείο αυτό θέτει χωρικούς και χρονικούς περιορισμούς στο κύκλωμα μας και καθορίζει την συχνότητα του ρολογιού. Επιπλέον ενημερώνει τον μεταφραστή για τυχόν εξόδους ή εισόδους του κυκλώματος που ανακατευθύνονται σε κάποιο περιφερικό της fpga.

Στη συνέχεια ακολουθεί η διαδικασία του mapping κατά την οποία υπολογίζεται ο συνολικός αριθμός πόρων που θα χρησιμοποιηθούνε από την fpga και γίνεται η

δέσμευσή τους. Στην συνέχεια γίνονται βελτιστοποιήσεις όσον αφορά την περιοχή που καλύπτει το κύκλωμα ώστε να απελευθερωθούνε πόροι.

Στο τέλος λαμβάνει χώρα το "place and route". Η διαδικασία αυτή είναι υπεύθυνη για την απεικόνιση των στοιχείων του κυκλώματος σε φυσικές θέσεις της fpga. Επίσης είναι υπεύθυνη και για την δρομολόγηση των λογικών blocks της fpga με τέτοιο τρόπο ώστε να δημιουργείται η μικρότερη δυνατή καθυστέρηση.

Με το τέλος της διαδικασίας της υλοποίησης δημιουργείται το αρχείο bitstream που είναι ένα αρχείο σε δυαδική μορφή και περιέχει όλες τις απαραίτητες πληροφορίες για τον προγραμματισμό της fpga με το κύκλωμα που σχεδιάστηκε.



Η διαδικασία της υλοποίησης στο εργαλείο ISE

6.4 Προγραμματισμός της fpga και επαλήθευση του κυκλώματος

Για να προγραματίσουμε την fpga χρησιμοποιούμε το εργαλείο Impact της Xilinx το οποίο παίρνει σαν είσοδο το bitstream αρχείο και βάση αυτού προγραμματίζει την fpga. Παρακάτω εξηγούμε συνοπτικά την διαδικασία που πρέπει να ακολουθήσει ο χρήστης για το επιτύχει.

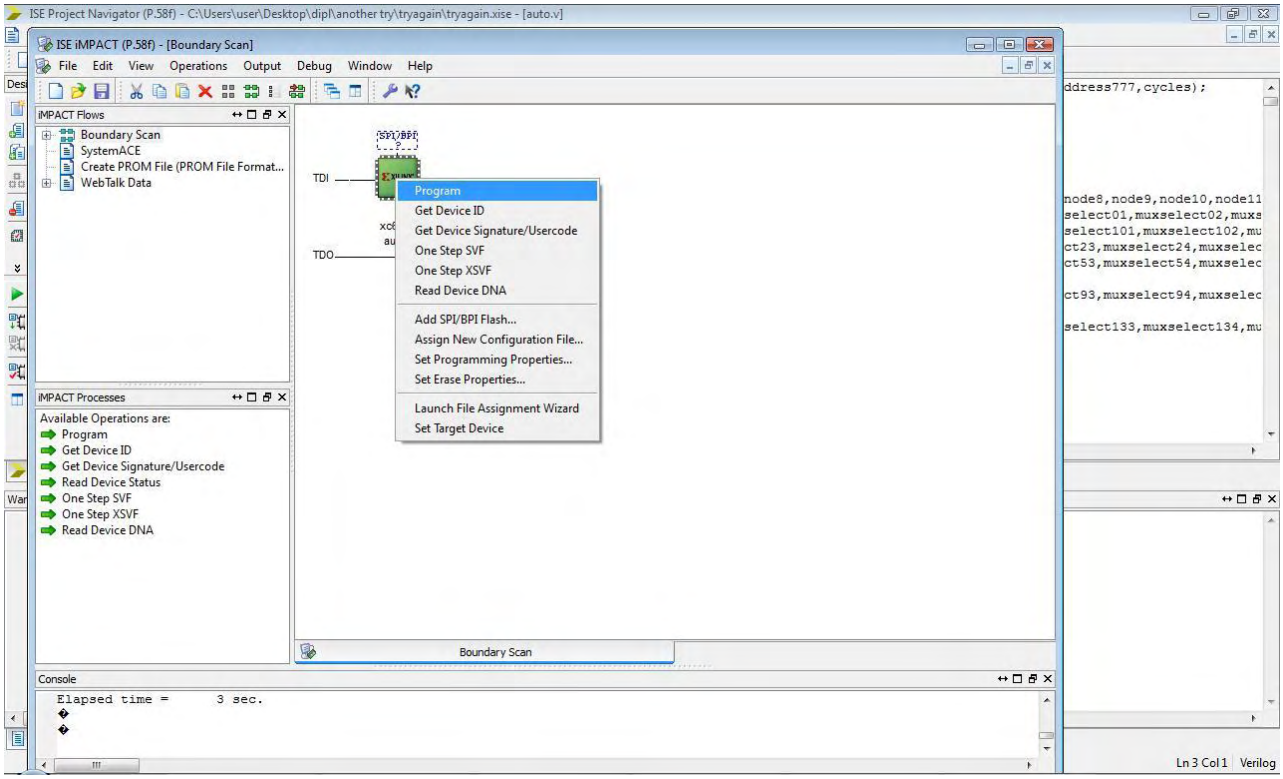
Αρχικά συνδέουμε την fpga με τον υπολογιστή είτε μέσο παράλληλου καλωδίου είτε μέσο usb, ανάλογα με το τι υποστηρίζεται.

Στη συνέχεια αφού τελειώσει η παραγωγή του bitstream, επιλέγουμε “configure target device” και μετά “Manage configuration project (Impact)” στο εργαλείο ISE. Οι επιλογές αυτές βρίσκονται κάτω από την επιλογή για την παραγωγή του bitstream στο πεδίο “processes” του εργαλείου.

Ένα νέο παράθυρο θα ανοίξει με το γραφικό περιβάλλον του Impact. Επιλέγουμε “boundary scan” (διπλό κλικ) στο πεδίο “iMPACT flows” και στη συνέχεια “file -> initialize chain”. Στη ερώτηση εάν θέλουμε να εισάγουμε κάποιο αρχείο στην fpga επιλέγουμε “yes” και επιλέγουμε το αρχείο bitstream (<όνομα αρχείου>.bit) που παράχθηκε προηγουμένως και στη συνέχεια “open”.

Στη συνέχεια μας δίνεται η επιλογή να δημιουργήσουμε ένα αρχείο μνήμης flash στο οποίο θα αποθηκεύσουμε το bitstream ώστε να αποφεύγουμε την παραπάνω διαδικασία (σύνθεση, υλοποίηση, προγραμματισμός) κάθε φορά που θέλουμε να τρέξουμε το κύκλωμα μας.

Το τελευταίο στάδιο είναι να κάνουμε δεξί κλικ στο τετράγωνο που δείχνουμε στη παρακάτω εικόνα το οποίο έχει το όνομα της fpga και να επιλέξουμε “program”. Μετά από κάποιο χρονικό διάστημα ο η fpga θα λειτουργεί με το κύκλωμα του σχεδιάσαμε.



Το εργαλείο iMPACT της Xilinx

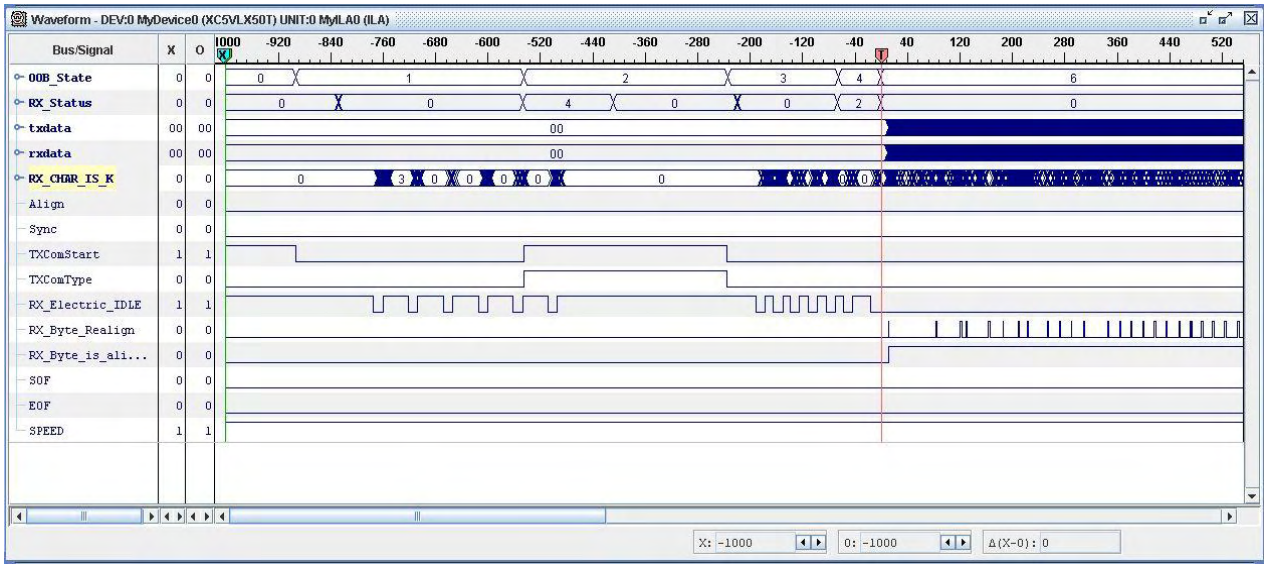
Για την επαλήθευση του κυκλώματος που υλοποιήσαμε στην fpga υπάρχουν διάφορες τεχνικές. Παρακάτω θα εξηγήσουμε ορισμένες από αυτές καθώς και ποιές επιλέξαμε εμείς.

Η πιο εύκολη και συνηθισμένη τακτική είναι η ανακατεύθυνση των εξόδων που μας ενδιαφέρουν στα leds ή στην Lcd οθόνη της fpga. Έτσι μπορούμε να βλέπουμε τα αποτελέσματα σε πραγματικό χρόνο και να ερευνήσουμε την συμπεριφορά του κυκλώματος. Μεταξύ άλλων επιλέξαμε και αυτόν τον τρόπο για να βεβαιωθούμε ότι το σύστημα μας έχει σωστή συμπεριφορά.

Άλλη τεχνική που χρησιμοποιείται είναι η δημιουργία μίας μνήμης δεδομένων στην fpga στην οποία αποθηκεύουμε τα αποτελέσματα που μας ενδιαφέρουν. Μετά τον τερματισμό του προγράμματος μπορούμε να κάνουμε προσπέλαση στη μνήμη αυτή και να δούμε τα περιεχόμενά της επιβεβαιώνοντας ή καταρρίπτοντας την σωστή συμπεριφορά του κυκλώματος.

Η επόμενη τεχνική είναι αυτή που χρησιμοποιήσαμε περισσότερο και περιλαμβάνει την μελέτη σε πραγματικό χρόνο του κυκλώματος με το εργαλείο ChipscopePro της Xilinx. Το εργαλείο αυτό μας δίνει την δυνατότητα να εισάγουμε στο κύκλωμα έναν “Ila core” κατά τη διάρκεια της διαδικασίας την υλοποίησης. Ο “Ila core” είναι ένα

κύκλωμα που παρέχεται από τη Xilinx και μπορεί να απομονώνει ορισμένα σήματα του κυκλώματος που θέλουμε εμείς. Τα σήματα αυτά μελετούνται από το ChipscopePro κατά την ανοδική ακμή του παλμού του ρολογιού και ανακατευθύνονται στην οθόνη του υπολογιστή σε διάταξη τύπου waveform του Modelsim. Το θετικό με τον τρόπο αυτό είναι ότι μας δίνεται η δυνατότητα να μελετήσουμε όχι μόνο τις εξόδους του κυκλώματος αλλά και τα ενδιάμεσα σήματα που επηρεάζουν τις εξόδους, καθιστώντας έτσι το εργαλείο αυτό ιδανικό για debugging. Αυτός είναι ο βασικός λόγος για τον οποίο επιλέξαμε να χρησιμοποιήσουμε αυτόν τον τρόπο επαλήθευσης του κυκλώματος. Το αρνητικό του εργαλείου αυτού είναι η αστάθεια του και το γεγονός ότι η ανάλυση των σημάτων γίνεται μόνο κατά την θετική ακμή του παλμού του ρολογιού, γεγονός περιοριστικό για ορισμένες περιπτώσεις.

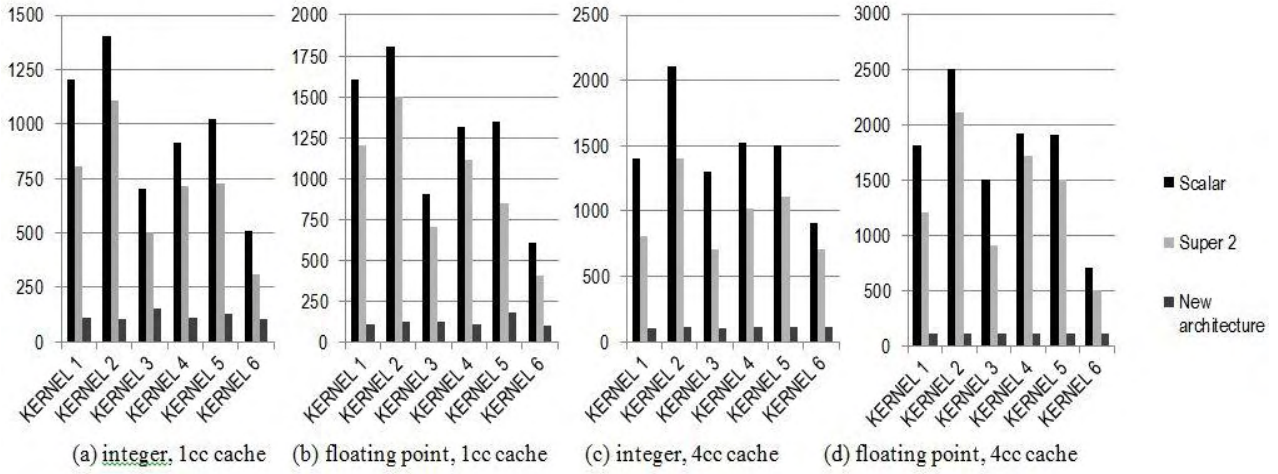


Ανάλυση συμπεριφοράς κυλώματος με το chipscopePro

7. Πειραματικά αποτελέσματα

Αφού υλοποιήσαμε το κύκλωμα και προγραμματίσαμε την fpga, πρέπει να κάνουμε μετρήσεις του χρόνου εκτέλεσης διαφόρων βρόγχων για να προσδιορίσουμε αν όντως το σύστημά μας επιτυγχάνει υψηλή απόδοση. Τα αποτελέσματα που πήραμε είναι πολύ ενθαρρυντικά καθώς ο πρωταρχικός μας στόχος επιτεύχθηκε. Η κάθε επανάληψη του βρόγχου ξεκινάει ανά ένα κύκλο μηχανής, εφ όσον οι καταχωρητές επικάληψης γεμίσουν, ανεξάρτητα από τις εξαρτήσεις από δεδομένα μεταξύ των εντολών.

Αυτό προσφέρει σημαντική επιτάχυνση σε σχέση με έναν υπερβαθμωτό επεξεργαστή τύπου Risc. Για να συγκρίνουμε καλύτερα την βελτίωση της απόδοσης υπολογίσαμε τον χρόνο εκτέλεσης γνωστών βρόγχων σε υπερβαθμωτούς επεξεργαστές και στο δικό μας σύστημα. Τα αποτελέσματα παρουσιάζονται παρακάτω.



Συγκριτικά αποτελέσματα αρχιτεκτονικών με βάση την απόδοσή τους

Οι επεξεργαστές που συγκρίνονται είναι ένας βαθμωτός Risc , ένας υπερβαθμωτός βαθμού 2 Risc και η δικιά μας αρχιτεκτονική (New architecture). Ο χρόνος εκτέλεσης μετριέται σε κύκλους μηχανής και ως μετροπρογράματα χρησιμοποιούνται γνωστοί βρόγχοι.

Παρατηρούμε ότι αυξάνοντας την καθυστέρηση της μνήμης cache η επιτάχυνση του συστήματός μας μεγαλώνει. Αυτό είναι πολύ θετικό καθώς σε πραγματικά συστήματα η μνήμη cache πολλές φορές έχει καθυστέρηση μεγαλύτερη από έναν κύκλο μηχανής.

Kernel 1	Hydro Fragment
Kernel 2	ICCG Excerpt
Kernel 3	Inner Product
Kernel 4	Banded Linear Equations – the inner loop only
Kernel 5	Equation of State Fragment
Kernel 6	First Sum. Partial Sum

Οι βρόγχοι που χρησιμοποιήθηκαν ως μετροπρογράματα

Τελικώς μπορούμε να συμπεράνουμε ότι η αρχιτεκτονική που προτείναμε παρουσιάζει πολύ αυξημένη απόδοση σε βρόγχους με 16 ή λιγότερες εντολές και είμαστε αισιόδοξοι ότι μπορεί να ανταγωνιστεί τους σύγχρονους επεξεργαστές στην ταχύτητα εκτέλεσης εφαρμογών.

8. Μελλοντική ανάπτυξη

Η εργασία αυτή ήταν ένα πολύ φιλόδοξο εγχείρημα με πολύ θετικά πειραματικά αποτελέσματα. Η ενασχόληση μου με αυτή ήταν κάτι νέο για μένα καθώς δεν υπάρχουν μαθήματα hardware design στη σχολή. Απέκτησα πολλές γνώσεις πάνω στα πραγματικά ψηφιακά ηλεκτρονικά κυκλώματα και χρησιμοποίησα πράγματα που ήξερα από άλλα μαθήματα όπως αρχιτεκτονική υπολογιστών, ενσωματωμένα συστήματα και ψηφιακή σχεδίαση.

Αν και το αρχικό ζητούμενο (η υλοποίηση σε fpga) ολοκληρώθηκε επιτυχώς υπάρχουν μερικά θέματα που η λύση τους θα οδηγούσε στην περαιτέρω βελτίωση της αρχιτεκτονικής. Αρχικά το σύστημά μας δεν υποστηρίζει εκτέλεση βρόγχων με εξαρτήσεις εντολών από δεδομένα προηγούμενων επαναλήψεων. Δηλαδή αν μία εντολή παροχετεύει ένα αποτέλεσμα σε μία άλλη εντολή προηγούμενης επανάληψης, τότε δεν υποστηρίζεται. Αυτό θα πρέπει να αλλάξει εάν θέλουμε το σύστημά μας να υποστηρίζει όλων των ειδών τους βρόγχους. Επιπλέον η δρομολόγηση των εντολών στους κόμβους θεωρούμε ότι γίνεται από τον compiler και όχι από το υλικό. Στόχος μας είναι να υλοποιήσουμε τη δρομολόγηση εντολών σε κόμβους με δικό μας αλγόριθμο που θα υλοποιείται στο υλικό. Η διαδικασία αυτή αρχικά φαίνεται δύσκολη και επίπονη αλλά κάνει το σύστημα πιο ολοκληρωμένο. Ένα άλλο θέμα που χρίζει προσοχής είναι ο αριθμός των κόμβων. Ο τωρινός αριθμός είναι 16, κάτι που σημαίνει ότι μόνο μέχρι 16 εντολές μπορούνε να υποστηριχθούν σε ένα σώμα βρόγχου. Ο αντίστοιχος μέγιστος αριθμός εντολών που υποστηρίζει ο loop stream buffer της Intel είναι 22, ένας αριθμός που καλό θα ήταν να υποστηρίξουμε κι εμείς.

Τέλος, η συχνότητα του το κυκλώματος που σχεδιάσαμε είναι 300MHz , η οποία περιορίζεται όχι από τον σχεδιασμό του κυκλώματος αλλά από τα τεχνικά χαρακτηριστικά της Spartan-6 fpga. Αυτό σημαίνει ότι η υλοποίηση το ίδιου κυκλώματος σε κάποια fpga που υποστηρίζει υψηλότερες συχνότητες ρολογιού θα επιφέρει αύξηση της απόδοσης. Επιπλέον η Spartan 6 χρησιμοποιεί 45 nm transistors τα οποία απέχουν πολύ από τα σύγχρονα transistors που χρησιμοποιούνται στην αγορά (22nm) κάνοντας έτσι την υλοποίηση του συστήματος σε μία πιο σύγχρονη και υψηλών επιδόσεων fpga περισσότερο κερδοφόρα στον τομέα της απόδοσης.

9. Βιβλιογραφία

[1] Agarwal, V., M. Hrishikesh, S. Keckler D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *Proc. of the 27th Int. Symp. on Comp. Arch. (ISCA'00)*, 248-259, 2000.

[2] Blake, G, R.G. Dreslinski, T. Mudge, K. Flautner. Evolution of Thread-Level Parallelism in Desktop Applications. In *Proc. of the 37th Int. Symp. on Comp. Arch. (ISCA'10)*, 302-313, 2010.

[3] Burger, D., and S.W. Keckler.. 19.5 Breaking the GOP/Watt Barrier with EDGE Architectures. *GOMACTech Intelligent Technologies Conference*, 2005.

[4] Burger, D., S.W. Keckler, K.S. McKinley, M. Dahlin, L.K. John, C. Lin, C.R. Moore, J. Burrill, R.G. McDonald, W. Yoder, and the TRIPS Team. Scaling to the End of Silicon with EDGE Architectures. In *Journal Computer Archives* Volume 37 Issue 7, 44-55, July 2004.

[5] Clark, N., A.Hormati, S.Mahlke. VEAL: Virtualized Execution Accelerator for Loops. In *Proc. ISCA'08*, 2008.

[6] Gebhart, M., B.A. Maher, K.E. Coons, J. Diamond, P. Gratz, M. Marino, N. Ranganathan, B. Robatmili, A. Smith, J. Burrill, S.W. Keckler, D. Burger, K.S. McKinley. An Evaluation of the TRIPS Computer System. In *Proc. of the 14th Int. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS XIV)*, 1-12, 2009.

[7] Gupta, S., S.Feng, A.Ansari, S.Mahlke. Erasing Core Boundaries for Robust and Configurable Performance. In *Proc. Int. Symp. on Microarch. (MICRO'10)*, 2010.

[8] Mathew, B., A. Davis. A Loop Accelerator for Low Power Embedded VLIW Processors. In *Proc. of the 2nd Int. Conf. on CODES+ISSS'04*, 2004.

[9] Paschalis, Antonis M. An effective BIST architecture for fast multiplier cores. *Design, Automation and Test in Europe Conference and Exhibition*, 1999.

[10] Rajagopalan D., S. Sethumadhavan, D. Burger, S.W. Keckler. Scalable Selective Re-Execution for EDGE Architectures. In *Proc. of the 11th Int. Conf. on Arch.*

Support for Programming Languages and Operating Systems (ASPLOS XI), 120-132, 2004.

[11] Sankaralingam, K., R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *Proc. of the 30th Int. Symp. on Comp. Arch. (ISCA'03)*, 422-433, 2003.

[12] Shee, S., Sri Parameswaran, Newton Cheung. Novel Architecture for Loop Acceleration: A Case Study. In *Proc. of the 3rd Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05)*, 297-302, 2005.

[13] Smith, A., J. Burrill, J. Gibson, B. Maher, N. Nethercote, B. Yoder, D. Burger, K. S. McKinley. Compiling for EDGE Architectures. In *Proc. of the Int. Symp. on Code Generation and Optimization (CGO'06)*, 185-195, 2006.

[14] Veeramachaneni, Sreehari. Efficient Design of 32-bit Comparators Using Carry Look-ahead Logic. Conference Publications, Montreal, Que., 2007.

[15] Yu, J., Guy Lemieux, Christopher Eagleston. Vector Processing as a Soft-core CPU Accelerator. In *Proc. of the 16th Int. Symp. on Field Programmable Gate Arrays (FPGA'08)*, 222-232, 2008.

[16] Risc processors,https://en.wikipedia.org/wiki/Reduced_instruction_set_computing

[17] Cisc processors, https://en.wikipedia.org/wiki/Complex_instruction_set_computing

[18] Spartan 6 overview, http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[19] Spartan 6 user guide, http://www.xilinx.com/support/documentation/user_guides/ug380.pdf

[20] Spartan 6 reference manual, http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf

[21] Chipscope pro user guide, http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/chipscope_pro_sw_cores_ug029.pdf

[22] Core generator guide, http://homepages.cae.wisc.edu/~ece554/website/Xilinx/Coregen_user_guide.pdf

[23] John K. Hennessy, David A. Patterson, Computer Architecture: A Quantitive Approach, 4th Edition

[24] Xilinx ISE manual, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf

[25] Xilinx ISE tutorial, http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/ise_tutorial_ug695.pdf

[26] Computer architecture techniques for reduced power consumption, Intel loop buffer, http://books.google.gr/books?id=HHyog7fjSMC&pg=PA115&lpg=PA115&dq=intel+loop+buffer&source=bl&ots=E9YtZGcMTX&sig=vKULPA623kLrzP55enA4LQvvCUM&hl=el&sa=X&ei=3TbfUdXsCe6V0QXD04GQDQ&ved=0CEUQ6AEwAw#v=onepage&q=intel%20loop%20buffer&f=false

[27] Evaluation of Trips computer system, ftp://ftp.cs.utexas.edu/pub/dburger/papers/ASPLOS09.pdf

[28] Network topology, http://en.wikipedia.org/wiki/Network_topology

[29] Nian-Feng Tzeng, Sizheng Wei. Enhanced Hypercubes. In Journal IEEE Transactions on Computers archive, Volume 40 Issue 3, March 1991, Page 284-294.