Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Υλοποίηση RISC-V πυρήνα σε FPGA με Εφαρμογή Τεχνικών Προσεγγιστικού Υπολογισμού και Χαμηλής Κατανάλωσης

Διπλωματική Εργασία

του

Περδικούρη Ορφέα

Επιβλέπων : Δημήτριος Σούντρης

Καθηγητής Ε.Μ.Π

Εργ. Μικροϋπολογιστών & Ψηφιακών Συστημάτων (MicroLab)

Αθήνα, Οκτώβριος 2021

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Υλοποίηση RISC-V πυρήνα σε FPGA με Εφαρμογή Τεχνικών Προσεγγιστικού Υπολογισμού και Χαμηλής Κατανάλωσης

Διπλωματική Εργασία

του

Περδικούρη Ορφέα

Επιβλέπων : Δημήτριος Σούντρης

Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25η Οκτωβρίου 2021.

.....	.....	.....
Δημήτριος Σούντρης Καθηγητής Ε.Μ.Π.	Παναγιώτης Τσανάκας Καθηγητής Ε.Μ.Π.	Διονύσιος Πνευματικάτος Καθηγητής Ε.Μ.Π.

Εργ. Μικροϋπολογιστών & Ψηφιακών Συστημάτων (MicroLab)

Αθήνα, Οκτώβριος 2021

.....

Περδικούρης Ορφέας

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

Περίληψη

Τα FPGA (Field-Programmable Gate Arrays) είναι κυκλώματα ημιαγωγών που περιέχουν προγραμματιζόμενη λογική (logic blocks) και προγραμματιζόμενες διασυνδέσεις. Τα λογικά τμήματα μπορούν να εκτελέσουν λειτουργίες βασικών πυλών ή πιο περίπλοκες συναρτήσεις. Χρησιμοποιούνται ολοένα και περισσότερο σε εφαρμογές αναγνώρισης, εξόρυξης και αναζήτησης με σκοπό την πιο γρήγορη και ενεργειακά αποδοτική εκτέλεση τους. Τα κυκλώματα που υλοποιούνται ποικίλουν από απλά αριθμητικά (π.χ. πολλαπλασιαστες, αθροιστες) και χρησιμοποιούνται ως επιταχυντές υλικού (hardware accelerators) εως πιο πολύπλοκα όπως ολόκληρους πυρήνες (soft cores). Οι τελευταίοι παρόλο που λειτουργούν σε αρκετά χαμηλότερη ταχύτητα από τους hard core πυρήνες, παρέχουν μεγαλύτερη ευελιξία τροποποίησης ώστε να εξατομικεύονται στην εκάστοτε εφαρμογή.

Στην παρούσα διπλωματική θα χρησιμοποιηθεί ένας open source, soft core πυρήνας RISC-V υλοποιημένος σε γλώσσα περιγραφής υλικού VHDL.. Στόχος της εργασίας αποτελεί αρχικά η μελέτη και διόρθωση του ήδη υπάρχοντος κώδικα του επεξεργαστή και η δημιουργία ενός επιπλέον design το οποίο θα μπορεί να αρχικοποιεί τον επεξεργαστή πριν την εκτέλεση του εκάστοτε benchmark αλλά και να λαμβάνει τα αποτελέσματα της εκτέλεσης αυτής.Επιπλέον ακολουθεί η επέκταση του συνόλου εντολών με την προσθήκη ορισμένων νέων approximate εντολών(αριθμητικών και μνήμης) με σκοπό την μείωση της δυναμικής ισχύς (dynamic power) του επεξεργαστή.Για την υποστήριξη αυτών των εντολών δοκιμάστηκαν approximate αριμθητικά κυκλώματα από open source approximate βιβλιοθήκες πραγματοποιώντας ένα design space exploration αλλά και βελτιστοποιημένα κυκλώματα τελεστών(π.χ. Xilinx multiplier) σε συνδυασμό με τη μέθοδο αποκοπής bit (bit truncation).Τέλος για παιρετέρω βελτίωση της κατανάλωσης υλοποιήθηκε μηχανισμός αποκοπής bit από τη μνήμη (approximate load).

Συνοψίζοντας κεντρικός σκοπός της εργασίας αποτελεί η μελέτη κατανάλωσης ισχύος σε επίπεδο πυρήνα όταν χρησιμοποιούνται approximate components καθώς στην βιβλιογραφία κατά κύριο λόγο αναλύονται μεμονωμένα και όχι ενσωματωμένα σε ένα μεγαλύτερο design.

Λέξεις Κλειδιά

RISC-V, approximate computing, low power, fpga, soft core, VHDL

Abstract

The FPGA devices are semiconductor circuits with reconfigurable logic and interconnects. They consist of logic blocks that can implement any function from simple logic gates to more complicated designs. They are increasingly used in image recognition, data mining and search applications providing execution acceleration along with energy efficiency. In the majority of designs FPGAs are used for speeding up arithmetic operations as hardware accelerators. Recently, SoC design has added processors implemented in the fpga known as soft processors that are written in hardware description language and cooperate with hardcore processor of the chip. Operation frequency of soft cores is lower compared to classic hard processors but offer design flexibility and reduced production cost.

In this thesis, an open source, RISC-V soft core implemented in VHDL is studied and improved in order to fit in embedded devices with strict memory resources and time limitations. Furthermore, an input/output interface is developed for fast processor initialization and result retrieving. In addition, RISC-V basic instruction set is extended with new approximate arithmetic and load instructions and processor support for these instructions is added. Regarding approximate arithmetic operations, support implemented using approximate components from open source approximate libraries but also using accurate components generated from Xilinx operators along with bit truncation technique. In the experiments conducted, dynamic power consumption of the core was measured while running the same codes at accurate and approximate form. The goal is to determine the size of energy savings that an approximate component achieves as part of a bigger design and not as standalone component, like most of literature's studies.

Keywords:

RISC-V, approximate computing, low power, fpga, soft core, VHDL

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τους επιβλέποντες καθηγητές μου Δημήτριο Σούντρη ΕΜΠ και Κιαμάλ Πεκμεστζή ΕΜΠ, για την ευκαιρία που μου προσέφεραν να εργαστώ στο εργαστήριο μικροϋπολογιστών σε ένα ενδιαφέρον project.

Επίσης οφείλω ένα μεγάλο ευχαριστώ στο διδακτορικό Βασίλη Λέοντα για την καθοδήγηση και τις γνώσεις που μου προσέφερε σε όλη την πορεία της εργασίας. Ακόμα θα ήθελα να ευχαριστήσω τον μεταδιδακτορικό Γεώργιο Λεντάρη για τις χρήσιμες συμβουλές και την εμπειρία του σε αδιέξοδα που προέκυψαν κατα τη διάρκεια της ερευνητικής αυτής μελέτης καθώς και τον μεταπτυχιακό φοιτητή Αντρέα Έντρι Τάκα για την εξαιρετική συνεργασία στο σημείο που διασταυρώθηκαν το πεδίο ερευνητικού ενδιαφέροντός του με το θέμα της παρούσας διπλωματικής.

Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την αμέριστη στήριξη τους καθόλη τη διάρκεια της φοίτησης μου καθώς και τους φίλους μου που ήταν κοντά μου και με βοήθησαν.

Περδικούρης Ορφέας

2021

Περιεχόμενα

Κεφάλαιο 1

1.1. Εισαγωγή

Η ολοένα και αυξανόμενη εξάπλωση του Internet of Things (IOT) προϋποθέτει συσκευές χαμηλού κόστους και περιορισμένων πόρων να έχουν επιπλέον δυνατότητες σε σχέση με τα κλασσικά ενσωματωμένα συστήματα όπως αυτή της μάθησης (machine learning).Μέχρι τώρα η διαδικασία της μηχανικής μάθησης και τον πολύπλοκων υπολογισμών γινόταν στο cloud (υπολογιστές με μεγάλη υπολογιστική ισχύ που λαμβάνουν τα δεδομένα από την ενσωματωμένη συσκευή, πραγματοποιούν την επεξεργασία των δεδομένων και επιστρέφουν στη συσκευή τα αποτελέσματα).Τελευταία παρατηρείται μετακίνηση ενός τμήματος της επεξεργασίας των δεδομένων στην ίδια την ενσωματωμένη συσκευή (embedded machine learning).Εκτός από τους περιορισμούς σε πόρους και ενέργεια υπάρχει και η ανάγκη για πολύ χαμηλό χρόνο απόκρισης (ιδιαίτερα σε εφαρμογές πραγματικού χρόνου)εφόσον η συσκευή επικοινωνεί με άλλες συσκευές αλλά και τον άνθρωπο.Τα παραπάνω οδηγούν στην υιοθέτηση τεχνικών approximate computing στις αρχιτεκτονικές των ενσωματωμένων συστημάτων έτσι ώστε να μειωθεί η κατανάλωση ενέργειας και η καθυστέρηση (latency).Ιδιαίτερα οι εφαρμογές μηχανικής μάθησης μπορούν να υποστηριχθούν από approximate hardware με αξιοσημείωτα οφέλη. Μια μελέτη στη διαδικασία ταξινόμησης(classification) σε ενσωματωμένα συστήματα έδειξε πως οι εσφαλμένοι υπολογισμοί επιτρέπονται εφόσον τα δείγματα εμφανίζονται στην ίδια κλάση [1] .Επιπλέον μια άλλη έρευνα απέδειξε την ανοχή των deep neural networks στα αριθμητικά λάθη που προέκυψαν από approximate computing [2].

1.2. Βασικές έννοιες

1.2.1 Δομή και λειτουργία του FPGA

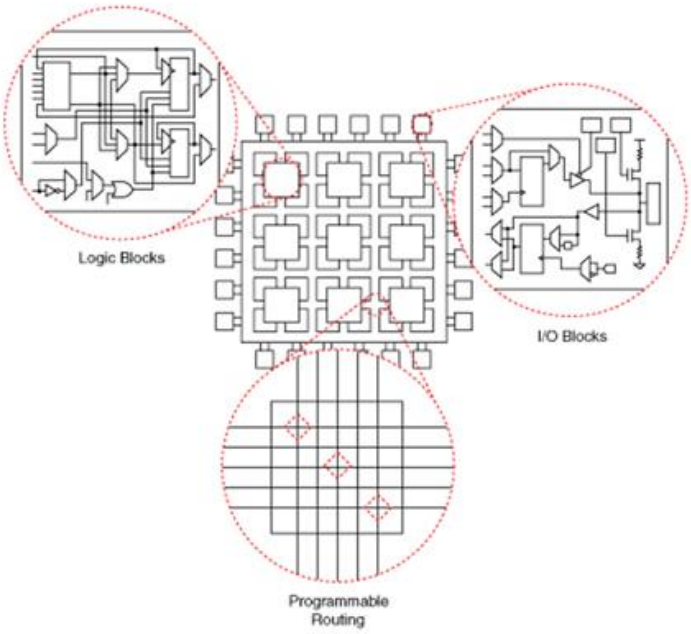
Το field-programmable gate array (FPGA) είναι ένα ολοκληρωμένο κύκλωμα το οποίο αποτελείται από εσωτερικά μπλοκ hardware με εσωτερικές διασυνδέσεις προγραμματιζόμενες από το χρήστη ώστε να

τροποποιείται ανάλογα με την εκάστοτε εφαρμογή. Οι διασυνδέσεις μπορούν εύκολα να επαναπρογραμματιστούν, επιτρέποντας στο FPGA να προσαρμόζεται στα διάφορα designs.

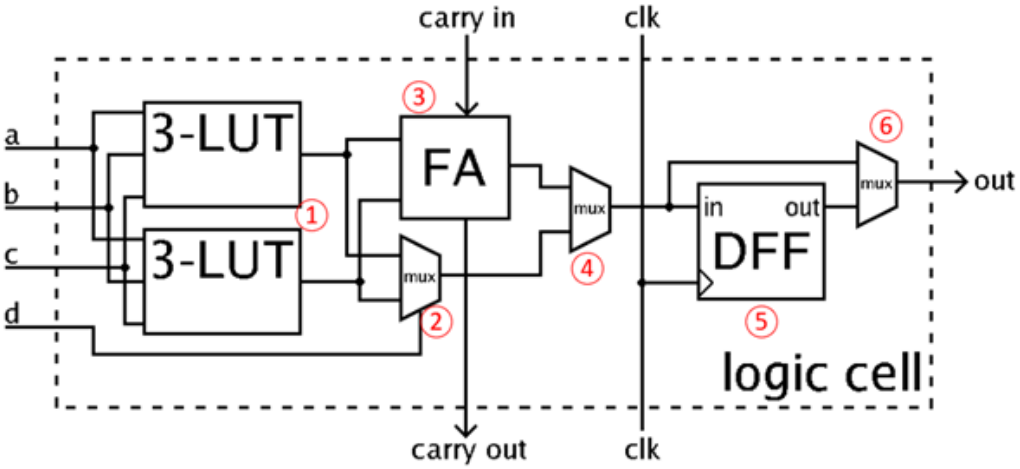
Το FPGA προέρχεται από προγενέστερες συσκευές όπως οι programmable read-only memories (PROMS) και οι programmable logic devices (PLDs).Αυτές οι συσκευές μπορούσαν να προγραμματιστούν είτε στο εργοστάσιο είτε εκτός αυτού αλλά δεν μπορούσαν να αλλάξουν αφού είχαν προγραμματιστεί διότι χρησιμοποιούσαν τεχνολογίες ασφαλειών. Αντίθετα, τα FPGA αποθηκεύουν τις πληροφορίες διαμόρφωσης τους σε σε ένα επαναπρογραμματιζόμενο μέσο όπως η static ram (SRAM) ή η flash memory.

Η βασική δομή του FPGA που φαίνεται στο *Σχήμα 1* αποτελείται χιλιάδες θεμελιώδη στοιχεία που ονομάζον ται configurable logic blocks(CLBs) τα οποία περιβάλλονται από ένα σύστημα προγραμματιζόμενων διασυνδέσεων.Ανάλογα με τον κατασκευαστή τα CLB μπορούν να αναφέρονται και ως logic block(LB), logic element(LE) ή logic cell (LC). Επίσης υπάρχουν διεπαφές εισόδου/εξόδου(I/O) για την επικοινωνία του FPGA με τις εξωτερικές συσκευές.

Η δομή ενός CLB φαίνεται στο *Σχήμα 2*.Αποτελείται από Lookup table (LUT) το οποίο είναι ένας πίνακας αληθείας δηλαδή μια προκαθορισμένη λίστα λογικών εξόδων για κάθε συνδυασμό εισόδων.(Αρκετά συνήθης είναι η χρήση LUT με αριθμό εισόδων από 4 εως 6).Ακόμα περιλαμβάνει πολυπλέκτες(mux), πλήρεις αθροιστές(FAs) και flip flops.



Σχήμα 1 : Θεμελιώδης δομή FPGA



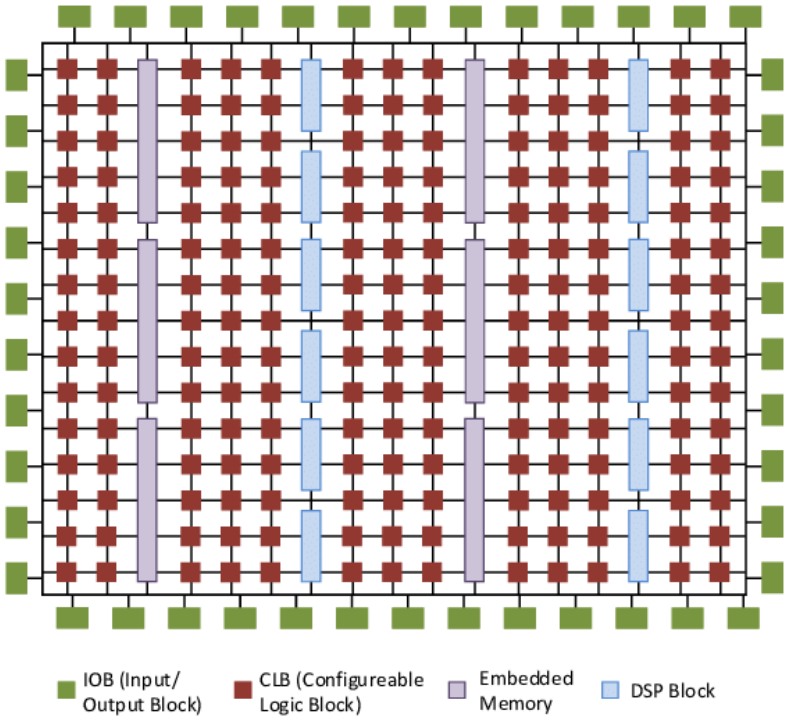
Σχήμα 2 : Απλουστευτική δομή ενός CLB

Ο αριθμός και η τοποθέτηση των στοιχείων του CLB διαφέρει ανάλογα με τη συσκευή.Στο Σχήμα 2 παρουσιάζεται μια απλουστευμένη μορφή του, η οποία περιλαμβάνει 2 LUT 3 εισόδων(1), ένα FA(3), ένα D – flip flop(5), και 3 πολυπλέκτες (2), (4), (6) , τα οποία διαμορφώνονται κατά τον προγραμματισμό του FPGA.

Αυτό το απλοποιημένο CLB έχει δύο τρόπους λειτουργίας. Στην κανονική λειτουργία, τα LUTs συνδυάζονται με τον Mux 2 ώστε να σχηματίσουν ένα LUT 4 εισόδων. Στην αριθμητική λειτουργία οι έξοδοι των LUT τροφοδοτούνται ως είσοδοι στο FA μαζί με την είσοδο του κρατουμένου από ένα άλλο CLB.Ο πολυπλέκτης 4 επιλέγει μεταξύ της εξόδου του FA και της εξόδου του LUT. Ο πολυπλέκτης 6 καθορίζει αν η λειτουργία θα γίνει ασύγχρονα ή σύγχρονα με το ρολόι του FPGA μέσω του D-flip flop.

Τα FPGA σύγχρονης γενιάς περιλαμβάνουν πιο σύνθετα CLBs ικανά να εκτελέσουν πολλαπλές λειτουργίες σε ένα μπλοκ. Τα CLB συνδυάζονται για πιο σύνθετες λειτουργίες όπως πολλαπλασιαστές, καταχωρητές, μετρητές εως και συναρτήσεις ψηφιακής επεξεργασίας σήματος (DSP).

Η παραπάνω παρουσίαση αφορά την θεμελιώδη αρχιτεκτονική των fpga η οποία όμως είναι υπεραπλουστευτική και είχε σκοπό την επεξήγηση στοιχειωδών λειτουργιών .Μια πιο σύγχρονη και ρεαλιστική απεικόνιση του FPGA φαίνεται στο *Σχήμα 3*.



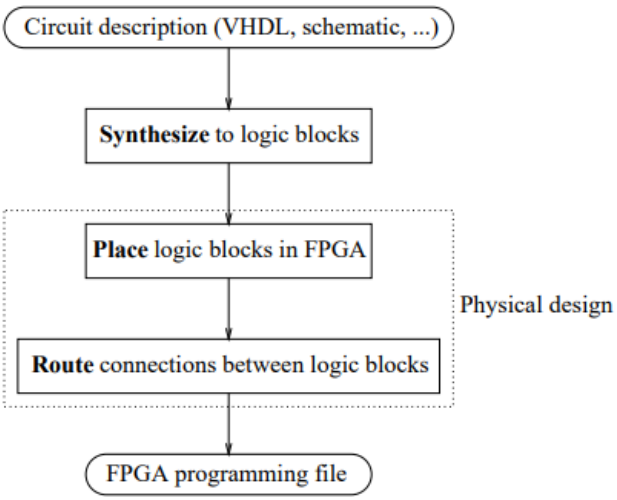
Σχήμα 3: Σύγχρονη δομή FPGA

Εδώ παρατηρούνται δυο νέα στοιχεία τα οποία είναι η *embedded memory* και τα *dsp block*. Η ενσωματωμένη μνήμη ονομάζεται **block ram** και αποτελείται από διακριτά μπλοκ προκαθορισμένο μεγέθους συνήθως 4,8,16,32 kb. Είναι μνήμη δύο θυρών (two port ram) και μπορεί να χρησιμοποιηθεί ως μνήμη μονής θύρας (single port ram), μνήμη διπλής θύρας(dual port ram) και fifo. Οι block ram χρησιμοποιούνται όταν απαιτείται μεγάλη μνήμη.Αν υλοποιηθεί μια μικρή μνήμη σε block ram τότε θα σπαταληθεί το μεγαλύτερο μέρος της και θα υπάρξει overhead στην ταχύτητα πρόσβασης στη μνήμη.Σε περίπτωση που υπάρχουν μικρές απαιτήσεις μνήμης χρησιμοποιείται η **distributed ram** η οποία είναι μνήμη υλοποιημένη στα CLBs και είναι ταχύτερη.Αντίστοιχα η distributed ram δεν μπορεί να χρησιμοποιηθεί για κατασκευή μεγάλων μνημών καθώς θα σπαταλήσει υπερβολικά μεγάλο αριθμό πόρων CLB.Το εργαλείο που θα μεταγλωττίσει τον κώδικα επιλέγει αυτόματα αν θα χρησιμοποιήσει block ram ή distributed ram ανάλογα με το μέγεθος της μνήμης που απαιτεί ο χρήστης αλλά μπορεί να καθοδηγηθεί ρητά ώστε να χρησιμοποιήσει μια από τις δύο μέσω attributes(εντολές καθοδήγησης του compiler) στον πηγαίο κώδικα Verilog ή VHDL.

Τα **dsp blocks** είναι ASIC κυκλώματα αριθμητικών πράξεων (π.χ. αθροιστές, πολλαπλασιαστές κ.ο.κ) τα οποία είναι βελτιστοποιημένα ως προς την ταχύτητα και την κατανάλωση ισχύος.Όπως αναφέρθηκε και για τις μνήμες παραπάνω το εργαλείο επιλέγει αν θα υλοποιήσει την πράξη σε dsp ή σε LUTs εκτός αν καθοδηγηθεί από το χρήστη μέσω attributes να χρησιμοποιήσει ένα από τα δυο.

1.2.2. Placement και Routing

Η διαδικασίες του place και route βρίσκονται στο τέλος της ροής για την εξαγωγή του δυαδικού αρχείου το οποίο θα φορτωθεί στο fpga.Η πλήρης ροή φαίνεται στο *Σχήμα 4*.



Σχήμα 4 : Ροή εξαγωγής bitstream

Αρχικά ο πηγαίος κώδικας περιγραφής υλικού μέσω της διαδικασίας synthesis μετατρέπεται σε μια απεικόνιση σε επίπεδο πυλών παράγοντας το netlist (λίστα καλωδιώσεων και λογικών πυλών σε ένα ενιαίο αρχείο).

Στη συνέχεια ακολουθεί η διαδικασία του **implementation** κατά την οποία αντιστοιχίζεται το netlist σε φυσικούς πόρους του fpga. Συγκεκριμένα το **placement** καθορίζει ποιό λογικό μπλοκ μέσα στο fpga θα υλοποιήσει κάθε λογικό τμήμα του κυκλώματος.Ο αλγόριθμος του placement επιδιώκει να ελαχιστοποιήσει το μήκος των διασυνδέσεων που συνδέουν τα διάφορα τμήματα του design(wire-length driven placement), να εξισορροπήσει την πυκνότητα καλωδιώσεων στην επιφάνεια του fpga(routability-driven placement) και να μεγιστοποιήσει την ταχύτητα του κυκλώματος(timing-driven placement).

Ακολουθεί η διαδικασία του **routing** κατα την οποία εφόσον έχουν επιλεχθεί οι τοποθεσίες όλων των λογικών μπλόκ, ένας δρομολογητής(router) αναθέτει τις διασυνδέσεις του κυκλώματος στις προγραμματιζόμενες διασυνδέσεις του fpga και καθορίζει ποιοι προγραμματιζόμενοι διακόπτες θα είναι ανοιχτοί ώστε να συνδεθούν τα λογικά μπλοκ.Εξαιτίας της μεγάλης πολυπλοκότητας του routing , συνήθως πραγματοποιείται σε δύο στάδια : το global routing και το detailed routing.Το global routing βρίσκει για κάθε δίκτυο ένα δέντρο δρομολόγησης επιλέγοντας ένα σετ από κανάλια δρομολόγησης,αλλά δεν επιλέγει κάποιο συγκεκριμένο μονοπάτι δρομολόγησης και διακοπτών για κάθε δίκτυο.Σκοπός και βασικό πεδίο βελτιστοποίησης είναι η

ελαχιστοποίηση της πυκνότητας των καναλιών.Το detailed routing αναθέτει κάθε δίκτυο σε συγκεκριμένα κανάλια δρομολόγησης τα οποία έχουν περιοριστεί από τον global router και εξαρτάται σε μεγάλο βαθμό από την αρχιτεκτονική διασυνδέσεων του fpga.

1.2.3. Approximate computing

Ο όρος approximate computing ή αλλιώς προσεγγιστικοί υπολογισμοί αναφέρεται σε τεχνικές υπολογισμών που παράγουν προσεγγιστικά ανακριβή αποτελέσματα αλλά επιτυγχάνουν βελτίωση στην καταναλισκόμενη ενέργεια ή/και στην επίδοση του design. Μπορούν να πραγματοποιηθούν μόνο σε εφαρμογές στις οποίες επιτρέπεται μειωμένη ακρίβεια και βρίσκουν εφαρμογή στα ενσωματωμένα συστήματα υπολογιστών (embedded devices) και το IOT(Internet of Things) στα οποία η εξοικονόμηση ενέργειας είναι καίρια.Η ιδέα δημιουργίας τους βασίστηκε στην παρατήρηση ότι ενώ η εκτέλεση ενός ακριβούς υπολογισμού απαιτεί μεγάλο όγκο πόρων, αν επιτραπεί περιορισμένη προσέγγιση μπορεί επιτύχει σημαντικότατα κέρδη σε ενέργεια και επίδοση με αποδεκτή ακρίβεια αποτελεσμάτων. Η βιβλιογραφία περιλαμβάνει αρκετές εργασίες όπου προτείνονται προσεγγιστικά αριθμητικά κυκλώματα, όπως αθροιστές [3], [4] , πολλαπλασιαστές [5]–[10], και διαιρέτες [11], [12]. Επιπλέον, τεχνικές προσεγγιστικού υπολογισμού εφαρμόζονται και στην σχεδίαση επιταχυντών υλικού [13]–[16] από τους τομείς της Τεχνητής Νοημοσύνης και της Ψηφιακής Επεξεργασίας Σήματος.

1.2.4. RISC-V ISA

Ο Risk-five (RISC-V)είναι μια νέα αρχιτεκτονική συνόλου εντολών (ISA) η οποία αρχικά σχεδιάστηκε για εκπαιδευτικούς και σκοπούς έρευνας στον τομέα της αρχιτεκτονικής υπολογιστών.Πλέον έχει μετατραπεί σε μία στάνταρ δωρεάν ανοιχτή αρχιτεκτονική με εφαρμογή στη βιομηχανία.Μπορεί να υλοποιηθεί απευθείας σε hardware και αποφεύγει την υπερβολική προσαρμογή σε καποια συγκεκριμένη αρχιτεκτονική (π.χ. microcoded, in-order, out-of-order) ή τεχνολογία υλοποίησης (π.χ. full-custom, fpga, asic) αλλά επιτρέπει την υλοποίηση σε όλα τα παραπάνω.Αποτελείται από ένα μικρό θεμελιώδες σετ εντολών ακεραίων και προαιρετικές επεκτάσεις (M, A, F, D, E) για να υποστηρίζει λογισμικό γενικού σκοπού.Υποστηρίζει διευθυνσιοδότηση 32, 64 και 128 bit και μπορεί να ενσωματωθεί σε πολυπύρηνες παράλληλες δομές(multiprocessors).Διαθέτει 3 επίπεδα προνομίων(Privilege Levels) τα οποία είναι το Machine (M-mode), το Supervisor(S-Mode) και το User(U-

mode) με διαφορετικές λειτουργικότητες, εντολές και καταχωρητές κατάστασης και ελέγχου[17].

1.2.5. Προσεγγιστικοί πυρήνες RISC-V στη βιβλιογραφία

Εδώ αξίζει να γίνει μια αναφορά σε σχετικές υλοποιήσεις προσεγγιστικών επεξεργαστών RISC-V καθώς και στις βελτιώσεις που επιτυγχάνουν. Στο [42] προτείνεται ένας πυρήνας RISC-V με αρκετές διαφορετικές μεθόδους προσέγγισης. Αρχικά προτείνεται λειτουργία δυναμικής μεταβολής μεγέθους των προσεγγιστικών μπλοκ ανάλογα με τον αριθμό των ενεργών bit κατά την εκτέλεση. Επίσης ως προς τα προσεγγιστικά μπλοκ εισάγεται προσεγγιστικός αθροιστής Sklansky και προσεγγιστικός πολλαπλασιαστής Wallace Tree με κωδικοποίηση Booth οι οποίο διαθέτουν διαφορετικά επίπεδα προσέγγισης που εναλλάσσονται δυναμικά. Πρώτα πραγματοποιήθηκαν πειράματα για εφαρμογή δυναμικού μεγέθους προσεγγιστικών μπλοκ ενώ το επίπεδο προσέγγισης ήταν 0(δηλαδή παρείχαν ακριβή αποτελέσματα).Δοκιμάστηκε η λειτουργία δυναμικού μεγέθους αθροιστή με κέρδος ισχύος 3.1%, 2.1% και 1.8% στους αλγορίθμους KNN, KM και ANN αντίστοιχα. Επίσης δοκιμάστηκε η λειτουργία δυναμικού πολλαπλασιαστή με κέρδος ισχύος 4.5%, 4% και 4.9% στους ίδιους αλγορίθμους. Τέλος δοκιμάστηκαν ταυτόχρονα τα δύο παραπάνω με κέρδος ισχύος 8.2%, 5.8% και 7.3% αντίστοιχα. Ακολούθως πραγματοποιήθηκαν πειράματα με επίπεδο προσέγγισης διάφορο του μηδενός(επίπεδο 2 για 8 bit σετ δεδομένων και επίπεδο 3 για 16 bit). Μελετήθηκαν εκτός από τον προτεινόμενο αθροιστή και πολλαπλασιαστή, 2 αθροιστές της βιβλιοθήκης DeMas και 2 πολλαπλασιαστές της βιβλιοθήκης SMApproxLib. Στο επίπεδο σύγκρισης αθροιστών καλύτερη επίδοση πέτυχε ο προτεινόμενος(Sklansky) με κέρδη ισχύος 9.8%, 9.3% και 11.7% στους αλγορίθμους KNN, KM και ANN αντίστοιχα. Στο επίπεδο σύγκρισης πολλαπλασιαστών υπερίσχυσε και πάλι ο προτεινόμενος (Wallace Tree Booth multiplier) με κέρδη ισχύος 13.1%, 14.1% και 14.7% για τους ίδιους αλγορίθμους. Τα κέρδη αυξήθηκαν περαιτέρω όταν χρησιμοποιήθηκαν ταυτόχρονα ο προσεγγιστικός αθροιστής και ο προσεγγιστικός πολλαπλασιαστής σε 22.1%, 21.5% και 23.4% αντίστοιχα.

Στο [43] πραγματοποιείται μελέτη του κέρδους που έχει ένας πυρήνας RISC-V με υποστήριξη αριθμητικών μονάδων και μονάδας μνήμης με μεταβλητό αριθμό bit. Τα πειράματα εκτελέστηκαν στους αλγορίθμους Sobel filter και forwardk2j(εφαρμογή ελέγχου τοποθεσίας ρομποτικού βραχίονα). Το κέρδος ισχύος προέκυψε εως και 7% στο Sobel filter με μεταβλητότητα bit μόνο στις υπολογιστικές εντολές και είναι εως 29% με επιπλέον μεταβλητότητα bit στη μνήμη. Για την εφαρμογή forwardk2j προέκυψε κέρδος έως και 13% για μεταβλητότητα bit στις αριθμητικές

μονάδες και εως 24% με προσθήκη μεταβλητότητας bit στις προσβάσεις στη μνήμη.

1.3 Κίνητρα και συνεισφορά της εργασίας

Παρόλο που οι softcore πυρήνες χρησιμοποιούνται ολοένα και περισσότερο σε FPGA SoC σε συνεργασία με παραδοσιακούς πυρήνες απουσιάζει από τη βιβλιογραφία επαρκής επεξήγηση της αρχιτεκτονικής και τον τεχνικών που έχουν χρησιμοποιηθεί για την κατασκευή τους (για ανοιχτού κώδικα soft processors). Η ανάγκη για έναν μινιμαλιστικό εύκολα τροποποιήσιμο soft processor πυρήνα RISC-V με πλήρη τεκμηρίωση της δομής του αποτέλεσε έναν από τους κύριους στόχους της εργασίας. Ο δεύτερος στόχος ήταν η δημιουργία ενός συστήματος που να ελαχιστοποιεί το χρόνο που απαιτεί ένας επεξεργαστής μέχρι να αρχίσει να εκτελεί τις εντολές του κυρίως προγράμματος και μπορεί να βρει εφαρμογή σε ενσωματωμένα συστήματα που απαιτούν ταχεία εκτέλεση μικρών και προκαθορισμένων τμημάτων κώδικα. Τέλος πραγματοποιήθηκε μελέτη αναφορικά με το κέρδος που μπορεί να επιτευχθεί σε επίπεδο soft core πυρήνα υλοποιημένο σε fpga όταν τοποθετούνται προσεγγιστικά κυκλώματα στην μονάδα αριθμητικών πράξεων. Με άλλα λόγια αποτελεί μια προσπάθεια για δημιουργία ενός προσεγγιστικού πυρήνα RISC-V σε fpga που βρίσκει εφαρμογή σε ενσωματωμένα συστήματα όπου υπάρχουν αυστηροί ενεργειακοί περιορισμοί.

Πιο αναλυτικά παρακάτω παρατίθενται οι βασικοί στόχοι που επιτεύχθηκαν:

- Δημιουργία documentation για τον soft processor πυρήνα Potato Processor ο οποίος είναι υλοποιημένος σε vhdl.
- Βελτίωση της μονάδας ALU.
- Προσθήκη εσωτερικής μνήμης εντολών και εσωτερικής μνήμης δεδομένων ώστε να επιτρέπεται λειτουργία απουσία εξωτερικής μνήμης και η εκτέλεση όλων των αιτήσεων στη μνήμη σε 1cc.
- Δημιουργία συστήματος ταχείας επικοινωνίας του πυρήνα με το χρήστη χωρίς τη χρήση περιφερειακών συσκευών και data-bus.
- Δημιουργία νέων προσεγγιστικών εντολών και προσθήκη τους στο riscv-gnu-toolchain.
- Ποσοτικοποίηση του κέρδους δυναμικής ισχύος του fpga και του αντίστοιχου λάθους που προκύπτει, όταν γίνεται χρήση προσεγγιστικών αριθμητικών κυκλωμάτων αλλά και χρήση της τεχνικής αποκοπής bit.

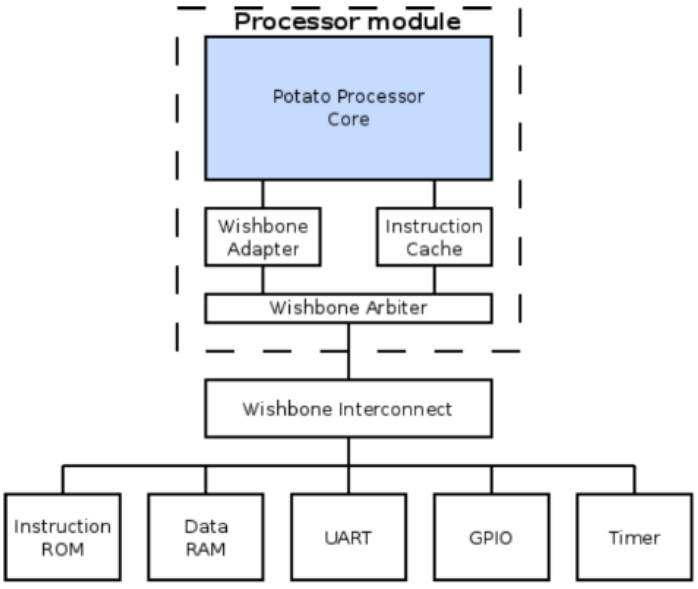
Κεφάλαιο 2

Αρχιτεκτονική του Potato Processor

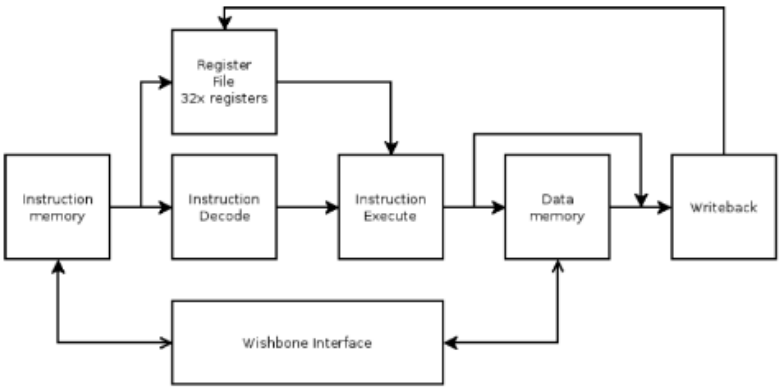
Ο **Potato Processor** είναι ένας open source soft core πυρήνας υλοποιημένος σε vhdl ο οποίος σύμφωνα με το documentation [18] διαθέτει :

- Πλήρη υποστήριξη του βασικού συνόλου εντολών ακεραίων 32 bit (RV32I) έκδοση 2.0
- Υποστήριξη σε λειτουργία μηχανής (machine mode) από την προνομιακη αρχιτεκτονική (RISC-V Privileged Architecture) έκδοση 1.10
- Μετρητή με ανάλυση us και διακοπές(interrupt) με σύγκριση
- Έως 8 IRQ εισόδους οι οποίες μπορούν να ενεργοποιηθούν μεμονωμένα
- Κλασική αρχιτεκτονική διοχέτευσης (pipeline) 5 επιπέδων RISC-V
- Διεπαφή Wishbone
- Προαιρετική cache εντολών

Παρακάτω στο *Σχήμα 5* φαίνεται η δομή του Potato SoC και στο *Σχήμα 6* φαίνεται η δομή του πυρήνα :



Εικόνα 5 : Potato SoC



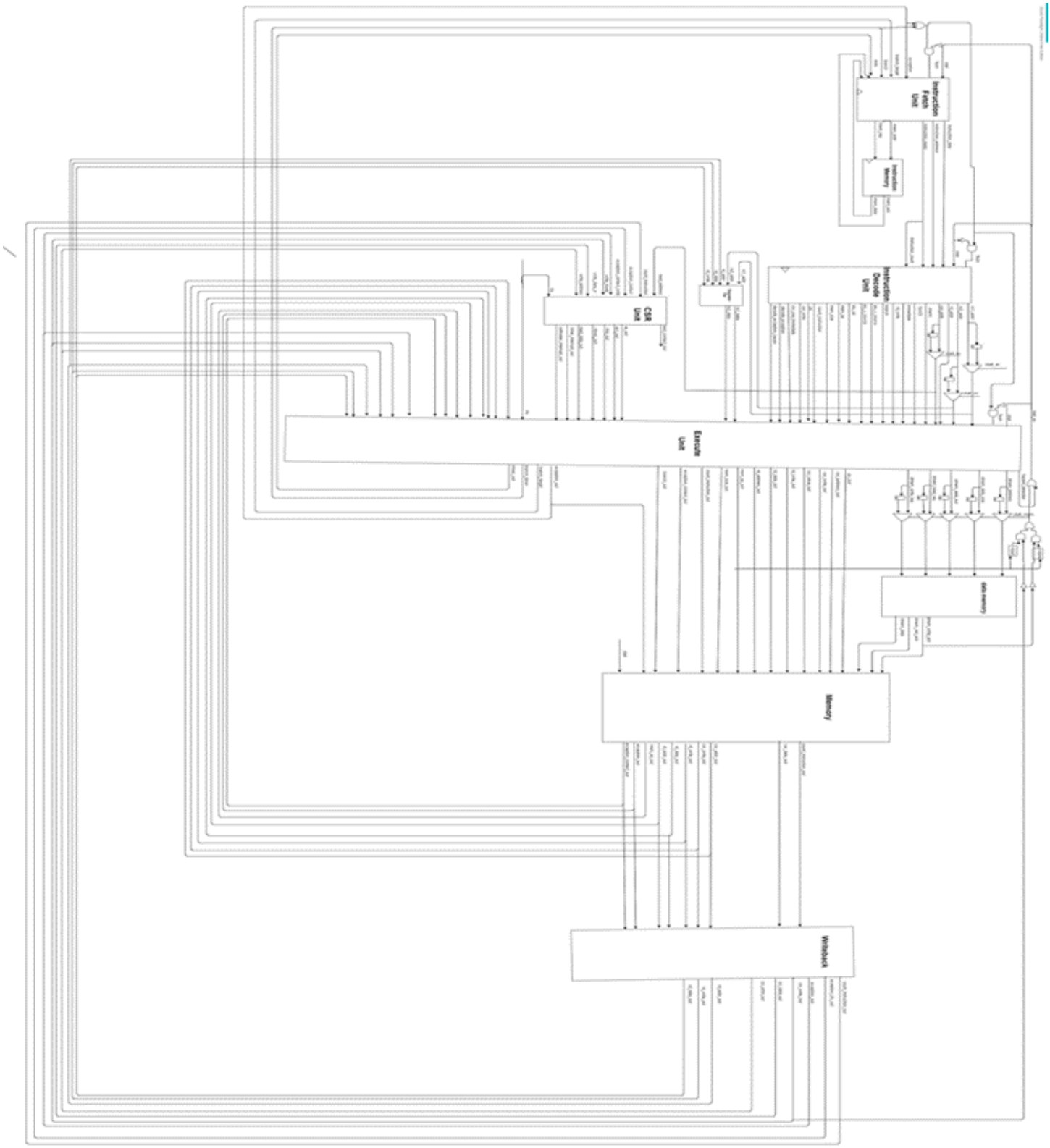
Εικόνα 6 : Στάδια πυρήνα potato

Οι αλλαγές που πραγματοποιήθηκαν για την ενσωμάτωση των προσεγγιστικών κυκλωμάτων αφορούν τον πυρήνα. Παρακάτω θα γίνει μια εκτενέστερη ανάλυση της αρχιτεκτονικής του και επεξήγηση των επιμέρους μονάδων από τις οποίες αποτελείται .

2.1. Δομική Περιγραφή

Στην *Εικόνα 7* βλέπουμε μια δομική απεικόνιση του η οποία περιλαμβάνει όλα τα σήματα και τις λογικές πύλες του ανώτερου επιπέδου ιεραρχίας.Τα λογικά μπλοκ από τα οποία αποτελείται ο potato είναι :

- **Μονάδα λήψης εντολών (Insturuction Fetch Unit)** η οποία είναι υπεύθυνη για να διαβάζει την επόμενη εντολή προς εκτέλεση από τη μνήμη εντολών και να την προωθεί στο επόμενο στάδιο.Σε περίπτωση άλματος με ή χωρίς συνθήκη διαβάζει και προωθεί την εντολή με διεύθυνση που έχει υπολογιστεί από το στάδιο εκτέλεσης (Execute).Ακόμα έχει τη δυνατότητα αναμονής(stall) σε περίπτωση που υπάρχει καθυστέρηση στη μνήμη.Να αναφερθεί εδώ πως προστέθηκε εσωτερική μνήμη δεδομένων ενώ ο αρχικός potato διέθετε μόνο εσωτερική μνήμη για τις εντολές και εξωτερική μνήμη για τα δεδομένα.Η προσθήκη αυτή επέτρεψε την εκτέλεση των εντολών μνήμης σε ένα κύκλο ρολογιού όπως και όλες οι υπόλοιπες εντολές.Τέλος σε περίπτωση που έχει ληφθεί λανθασμένα μια εντολή γίνεται εκκαθάριση του σταδίου(flush) (π.χ.στην περίπτωση που υπάρχει κάποιο άλμα μέχρι να υπολογιστεί η διεύθυνση προορισμού το στάδιο λήψης λαμβάνει κάθε φορά την επόμενη εντολή οδηγώντας σε λάθος προώθηση εντολών)



Εικόνα 7 : Δομική απεικόνιση potato core

- **Εσωτερική μνήμη εντολών(Instruction Memory)** η οποία διαβάζει την εντολή από τη διεύθυνση που υποδεικνύεται από το στάδιο

Fetch.Κατά την λειτουργία του πυρήνα λειτουργεί ως Read Only Memory εφόσον μόνο διαβάζει τις εντολές αλλά όπως θα δούμε παρακάτω χρησιμοποιήθηκε μνήμη ram για την υλοποίηση της για να μπορεί να υποστηρίξει τον μηχανισμό αρχικοποίησης που προστέθηκε.

- **Μονάδα αποκωδικοποίησης εντολών(Instruction Decode Unit)** η οποία αναλαμβάνει να διαμερίσει την εντολή στα αντίστοιχα τμήματα ανάλογα με το είδος της και να τα προωθήσει στις κατάλληλες μονάδες επεξεργασίας.Ενδεικτικά μερικά πεδία της εντολής 32 bit τα οποία εξάγει είναι οι διευθύνσεις των καταχωρητών πηγής(source registers) και προορισμού(destination register), το πεδίο shamt που χρησιμοποιείται στις πράξεις ολίσθησης και το πεδίο func3 το οποίο καθορίζει ποια σύγκριση θα γίνει στις εντολές διακλάδωσης. Ακόμα περιλαμβάνει μονάδα εξαγωγής άμεσης τιμής(immediate value) ανάλογα με την κατηγορία της εντολής (U, J, I, B, S) και μονάδα ελέγχου η οποία λαμβάνει ως είσοδο τα πεδία opcode, funct3, funct7, funct12 και παράγει σήματα όπως : εγγραφή σε καταχωρητή προορισμού, διακλάδωσης,εξαιρέσεις αποκωδικοποίησης(decode exceptions), σήματα ελέγχου και κατάστασης(csr), σήματα επιλογής ορισμάτων και λειτουργίας alu και σήματα επιλογής εντολής μνήμης(ανάγνωση ή εγγραφή , byte, halfword, word).Όπως και το στάδιο fetch διαθέτει εισόδους για αναμονή και εκκαθάριση.

- **Αρχείο καταχωρητών** το οποίο δέχεται ως εισόδους τις διευθύνσεις των καταχωρητών που λαμβάνει από το στάδιο της αποκωδικοποίησης και τροφοδοτεί το επόμενο στάδιο (εκτέλεση) με τις τιμές τους. Ακόμα διαθέτει είσοδο εγγραφής καταχωρητή προορισμού που πραγματοποιείται στο $5^{ο}$ στάδιο της διοχέτευσης που είναι η επανεγγραφή (writeback) σε περίπτωση που η εντολή διαθέτει εγγραφή σε καταχωρητή.

- **Καταχωρητές ελέγχου και κατάστασης (Control and Status Registers)** οι οποίοι χρησιμοποιούνται στη διαχείριση των εξαιρέσεων(exceptions) και των διακοπών(interrupts) και αποθηκεύουν την πρόσφατη κατάσταση του επεξεργαστή.Συνολικά υπάρχουν 4096 καταχωρητές ελέγχου κατάστασης σε ένα ξεχωριστό χώρο διευθύνσεων και εγγράφονται και διαβάζονται από τις csr εντολές [3].Εκτός από τις διεργασίες ανάγνωσης και εγγραφής των καταχωρητών υπάρχουν και ορισμένοι 64 bit μετρητές οι οποίοι

βρίσκονται μέσα στον χώρο διευθύνσεων που αναφέρθηκε παραπάνω και η πρόσβαση τους γίνεται σε τμήματα των 32 bit μέσω csr εντολών.

- **Μονάδα εκτέλεσης(Execution Unit)** η οποία πραγματοποιεί το σύνολο των υπολογισμών κάθε εντολής. Διαθέτει διεργασίες ελέγχου κακής ευθυγράμμισης δεδομένων(data misalign) και εντολών (instruction misalign), εύρεσης αριθμού διακοπής και εύρεσης αιτίας και διεύθυνσης εξαίρεσης(exception cause). Πραγματοποιείται υπολογισμός της διεύθυνσης άλματος υπό συνθήκη ή χωρίς και επιλέγεται μέσω πολυπλεκτών το όρισμα για κάθε μια από τις εισόδους της μονάδας αριθμητικών πράξεων(ALU). Τα πιθανά ορίσματα είναι η τιμή ενός καταχωρητή πηγής, η άμεση τιμή(immediate), η τιμή shamt, η τιμή pc και η τιμή csr.Προκειμένου να μην εισάγονται καθυστερήσεις όταν έχω κίνδυνο δεδομένων (π.χ. Read after Write) υπάρχει λογική προώθησης είτε από το στάδιο μνήμης(memory) είτε από το στάδιο επανεγγραφής(writeback).Τα σήματα που πραγματοποιούν την προώθηση είναι είτε έξοδοι του σταδίου execute που οδηγούνται στο στάδιο memory αλλά παράλληλα επανατροφοδοντούνται στην είσοδο του execute στις θύρες προώθησης από τη μνήμη, είτε έξοδοι του memory που οδηγούνται στο στάδιο writeback αλλά παράλληλα ανατροφοδοτούνται στην είσοδο του execute στις θύρες προώθησης από την επανεγγραφή. Πολυπλέκτες επιλέγουν την τιμή του καταχωρητή προορισμού που θα προωθήσουν στους πολυπλέκτες επιλογής ορισμάτων.Αν υπάρχει εξάρτηση δεδομένων θα επιλέξουν είτε την τιμή από την είσοδο προώθησης μνήμης είτε την τιμή από την είσοδο προώθησης από την επανεγγραφή ανάλογα αν η εξάρτηση υπάρχει μία ή δύο εντολές πριν αντίστοιχα.Αν δεν υπάρχει εξάρτηση θα επιλέξουν την τιμή από το αρχείο καταχωρητών.Επιπλέον διαθέτει μηχανισμό εντοπισμού κινδύνου ανάγνωσης από τη μνήμη όταν πραγματοποιείται ανάγνωση καταχωρητή ο οποίος στην προηγούμενη εντολή ανάγνωσης από μνήμη ήταν καταχωρητής προορισμού. Διαθέτει συγκριτή ο οποίος καθορίζει το αποτέλεσμα της διακλάδωσης βάσει του πεδίου funct3, αριθμητική μονάδα πράξεων και αριθμητική μονάδα πράξεων csr. Σημαντικό σημείο που πρέπει να αναφερθεί είναι η πρόσβαση στη μνήμη δεδομένων (data ram) η οποία δεν πραγματοποιείται στο επόμενο στάδιο memory όπως ίσως φαίνεται αλλά στο παρόν στάδιο του execute.Πέραν των καταχωρητών στην είσοδο του σταδίου execute υπάρχει συνδυαστική λογική μέχρι και την είσοδο της μνήμης δεδομένων. Η μονάδα αριθμητικών πράξεων είναι συνδυαστικό κύκλωμα και η έξοδος της δεν οδηγείται σε στοιχείο καθυστέρησης αλλά στην είσοδο της μνήμης σε περίπτωση που έχουμε εντολή πρόσβασης στη μνήμη. Συνεπώς γίνεται συγχώνευση των σταδίων execute και memory όπως αυτά ορίζονται στο κλασσικό pipeline 5 σταδίων με επακόλουθο περιορισμό της μέγιστης συχνότητας λειτουργίας.Το ερώτημα που προκύπτει είναι ποια η λειτουργία του σταδίου memory. Η απάντηση είναι πως περιλαμβάνει μόνο καθυστερήσεις για σήματα που οδηγούνται στο επόμενο στάδιο του writeback. Μερικά σήματα δε, μεταβιβάζονται από την είσοδο στην έξοδο χωρίς παρεμβολή στοιχείου καθυστέρησης.

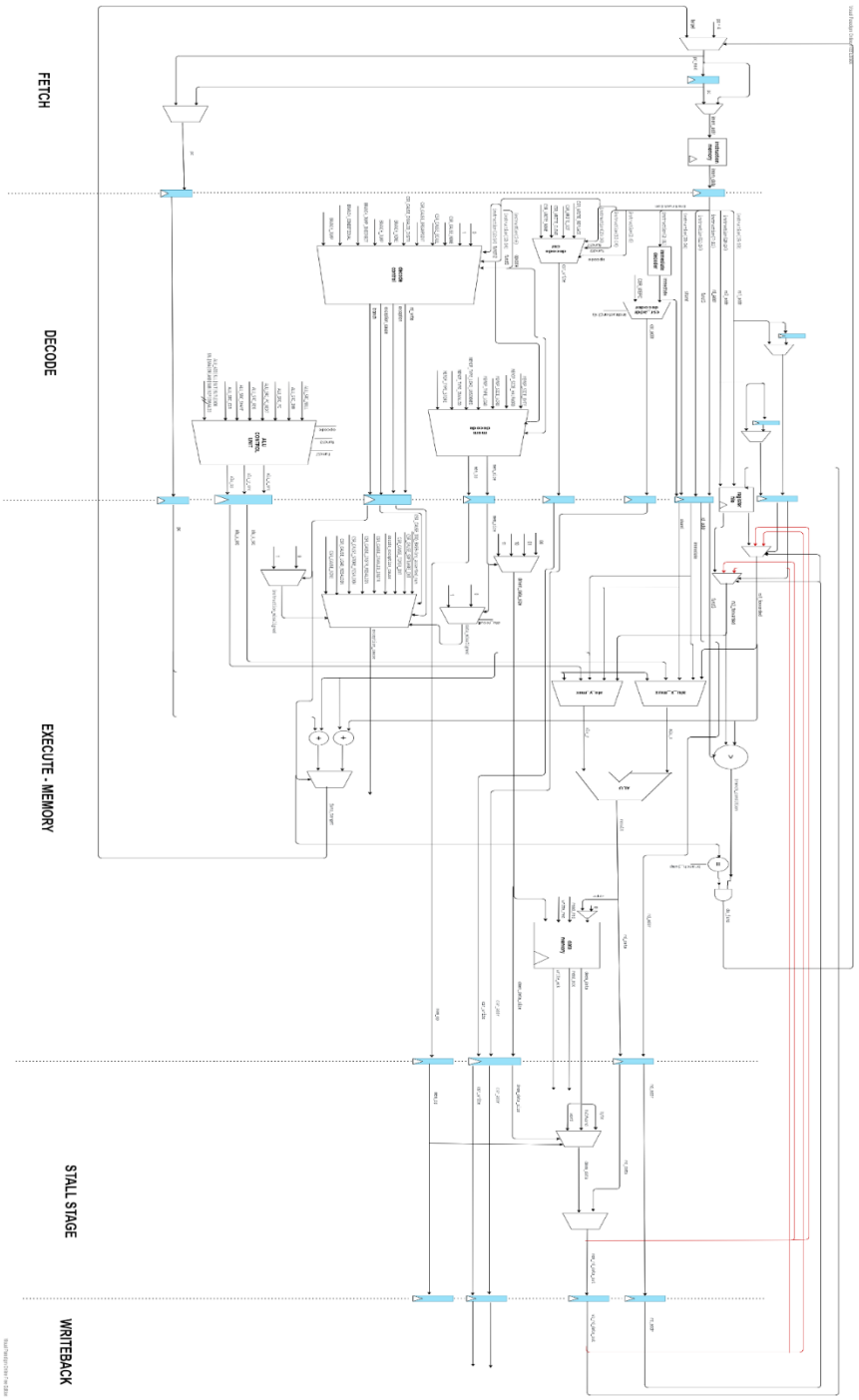
- **Μνήμη δεδομένων(Data memory)** η οποία περιλαμβάνει τα δεδομένα για την εκτέλεση του προγράμματος και αρχικοποιείται στην αρχή κάθε εκτέλεσης.Το instruction set υποστηρίζει προσβάσεις στη μνήμη με διαφορετικά μεγέθη (byte, ημι λέξη, λέξη) και η διευθυνσιοδότηση είναι ανα byte.Για να υποστηριχθούν τα διαφορετικά μεγέθη υπάρχει ως είσοδος σήμα επιλογής το οποίο καθορίζει αν θα γίνει ανάγνωση ή εγγραφή 8, 16 ή 32 bit.Η πρόκληση που παρουσιάζεται είναι πως αν επιλέξουμε ένα πίνακα σημάτων με πλάτος 8 bit ώστε να υποστηρίζει την διευθυνσιοδότηση τότε προκαλείται λάθος κατά το synthesis στο κομμάτι της εγγραφής διαφορετικών θέσεων μνήμης σε ένα κύκλο ρολογιού(όταν γίνεται εγγραφή λέξης ή ημιλέξης).Αν πάλι επιλεγεί πίνακας σημάτων με πλάτος 32 bit τότε θα πρέπει να εγγράφεται υποχρεωτικά μια λέξη (32 bit) ενώ δεν θα γίνεται δυνατή η εγγραφή ημιλέξης ή byte.Για να αποφευχθούν τα παραπάνω λάθη η μνήμη υλοποιήθηκε με 4 πίνακες πλάτους 8 bit.Η πρώτη θέση μνήμης των πινάκων αποθηκεύει τα byte της πρώτης λέξης (πίνακας 1 -> byte 0, πίνακας 2 -> byte 1, πίνακας 3 -> byte 2, πίνακας 4 -> byte 3) η δεύτερη θέση των πινάκων τα byte της δεύτερης λέξης κ.ο.κ. . Οι διευθύνσεις που λαμβάνει η μνήμη είναι ανα byte όπως αναφέρθηκε και παραπάνω συνεπώς στην είσοδο γίνεται μια δεξιά ολίσθηση κατά 2 ώστε να μετατραπούν σε διευθύνσεις ανα λέξη καθορίζοντας σε ποια θέση των πινάκων θα γίνει η ανάγνωση ή η εγγραφή.Έτσι αν ληφθεί διεύθυνση 0, 1, 2 ή 3 με την ολίσθηση κατά 2 θα μετατραπεί σε 0 αφού και τα 4 byte ανήκουν στη λέξη 0 και ανάλογα με το σήμα μεγέθους θα γίνει ανάγνωση ή εγγραφή και των 4 πινάκων(αν έχω λέξη), των 2 πινάκων (αν έχω ημιλέξη) ή μόνο ενός πίνακα(αν έχω byte).

- **Μονάδα καθυστέρησης(Memory)** η οποία λαμβάνει τα δεδομένα από τη μνήμη δεδομένων και τα προωθεί στο επόμενο στάδιο writeback αλλά και πίσω στην είσοδο προώθησης του execute και εισάγει καθυστερήσεις σε ορισμένα σήματα.

2.2. Απεικόνιση διοχέτευσης

Η παραπάνω ανάλυση αφορά τις βασικές λειτουργίες κάθε δομικής μονάδας του επεξεργαστή αλλά δεν δίνει πληροφορίες για τον χρονισμό δηλαδή σε

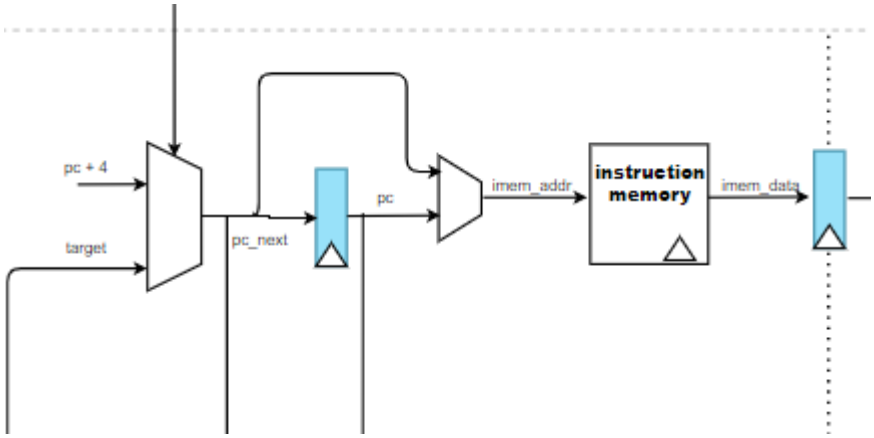
ποια στάδια διοχέτευσης έχει καταμεριστεί το datapath.Μια τέτοια ανάλυση θα πραγματοποιηθεί παρακάτω δίνοντας μια συνολική εικόνα για την πλήρη κατανόηση της αρχιτεκτονικής του Potato Processor.Στην *Εικόνα 8* φαίνεται η αρχιτεκτονική σε μορφή σταδίων.



Εικόνα 8 : Απεικόνιση διοχέτευσης potato processor

2.2.1. Στάδιο Fetch

Όπως παρατηρούμε τα στάδια fetch και decode πραγματοποιούν τις αναμενόμενες λειτουργίες.Κατα το fetch πραγματοποιείται επιλογή ανάμεσα στην επόμενη διεύθυνση από αυτή που βρίσκεται ο μετρητής προγράμματος(pc + 4) και στην διεύθυνση στόχου σε περίπτωση που έχουμε άλμα η οποία έχει υπολογιστεί από το στάδιο execute της προηγούμενης εντολής. Η διεύθυνση από αυτή την επιλογή οδηγείται στην είσοδο της μνήμης εντολών για να διαβαστεί η επόμενη εντολή προς εκτέλεση.

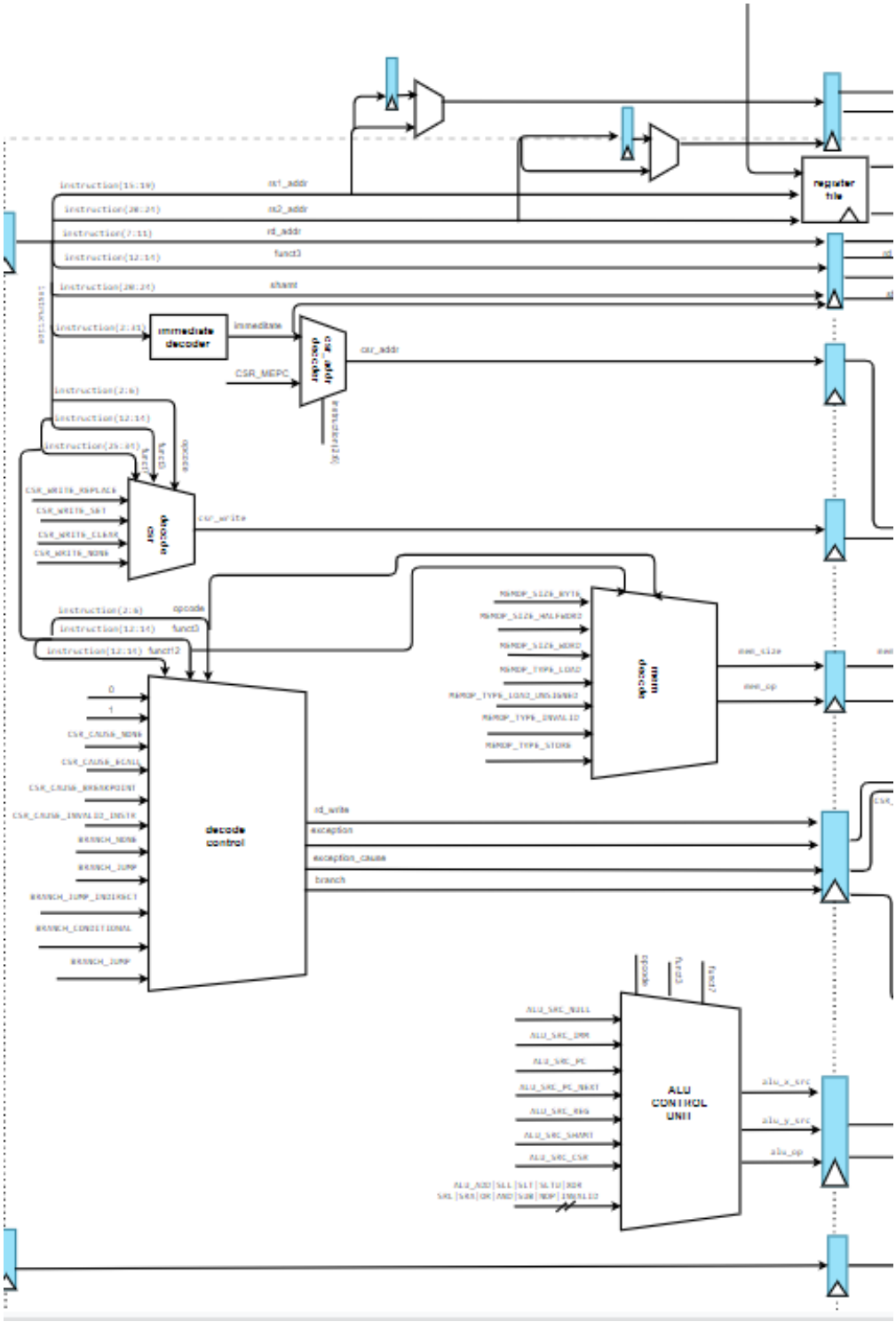


Εικόνα 9 : Στάδιο Fetch

Εκτός από τον καταχωρητή στο τέλος του σταδίου υπάρχει και ένας ενδιάμεσος καταχωρητής ο οποίος τοποθετήθηκε από τον κατασκευαστή για να αποθηκεύει την τελευταία εντολή σε περίπτωση κάποιας καθυστέρησης από τη μνήμη η οποία εισάγει καθυστερήσεις και στα προηγούμενα στάδια execute, decode, fetch.Έτσι όταν το σήμα για καθυστέρηση του σταδιου fetch ενεργοποιηθεί και για όσο διάστημα παραμένει ενεργό, η διεύθυνση επόμενης εντολής μένει σταθερή στην τιμή που έχει αποθηκεύσει ο καταχωρητής. Στην τροποποιημένη αρχιτεκτονική που προτείνεται και στην οποία πραγματοποιήθηκαν τα πειράματα δεν χρησιμοποιείται εξωτερική μνήμη αλλά τοποθετήθηκε εσωτερική μνήμη δεδομένων η οποία ικανοποιεί τα αιτήματα σε 1 κύκλο ρολογιού.Επομένως η καθυστέρηση αυτή δεν χρησιμοποιείται.

2.2.2. Στάδιο Decode

Στο επόμενο στάδιο (decode) όπως αναφέρθηκε και παραπάνω η εντολή καταμερίζεται σε τμήματα τα οποία θα τροφοδοτηθούν στην επόμενη μονάδα execute.



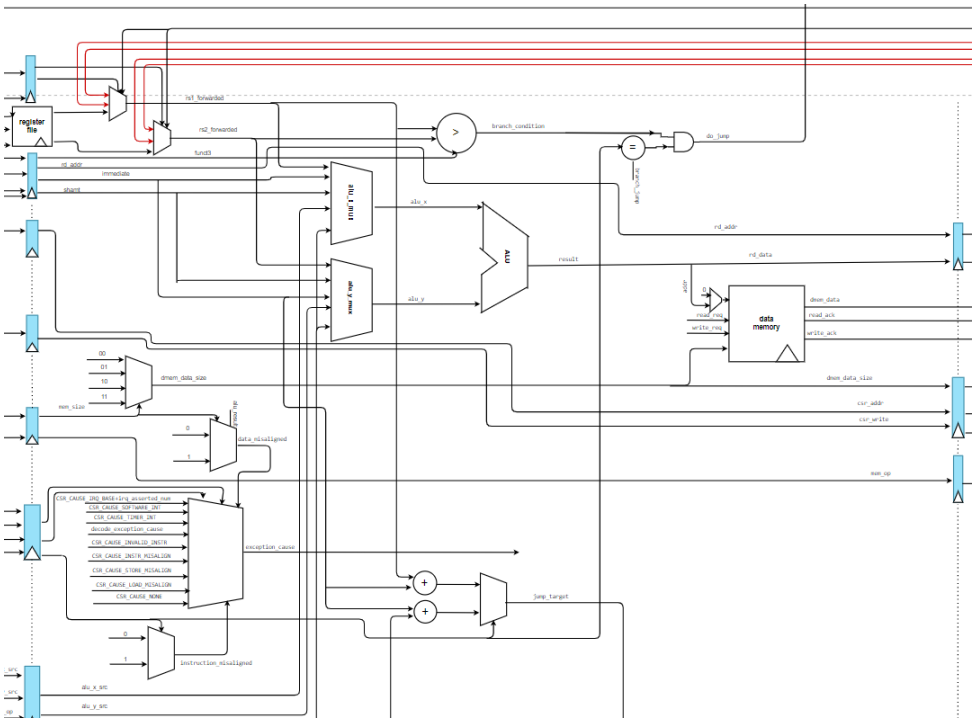
Εικόνα 10 : Στάδιο Decode

Αναλυτικά περιλαμβάνει μονάδα αποκωδικοποίησης σταθερής τιμής (immediate value), μονάδα αποκωδικοποίησης διεύθυνσης csr η οποία

λαμβάνει ως είσοδο την έξοδο της προηγούμενης μονάδας και εξάγει τη διεύθυνση csr, μονάδα αποκωδικοποίησης csr η οποία καθορίζει την τιμή του csr write, μονάδα αποκωδικοποίησης μνήμης η οποία εξάγει το είδος και το μέγεθος της λειτουργίας μνήμης (αν πρόκειται για τέτοια), μονάδα αποκωδικοποίησης ελέγχου η οποία εξάγει εξόδους σχετικές με εξαιρέσεις αλλά και πιθανή διακλάδωση και τέλος μονάδα ελέγχου της ALU η οποία εξάγει το είδος της αριμθητικής πράξης προς εκτέλεση αλλά και το είδος των ορισμάτων που θα προωθηθούν σε αυτή(register, immediate, csr, pc ,shamt, csr). Όπως και στο προηγούμενο στάδιο του fetch υπάρχουν ενδιάμεσοι καταχωρητές στα σήματα διευθύνσεων των καταχωρητών πηγής (rs1, rs2) που αποθηκεύουν τις τιμές τους σε περίπτωση καθυστέρησης. Όπως και πριν δεν χρησιμοποιούνται στη υλοποίηση που προτείνεται στην παρούσα εργασία. Ακόμα παρατηρείται πως το αρχείο καταχωρητών λειτουργεί ως καθυστέρηση που χωρίζει το στάδιο decode από αυτό του execute-memory επομένως δεν τοποθετούνται επιπλέον καθυστερήσεις όπως στα υπόλοιπα σήματα.

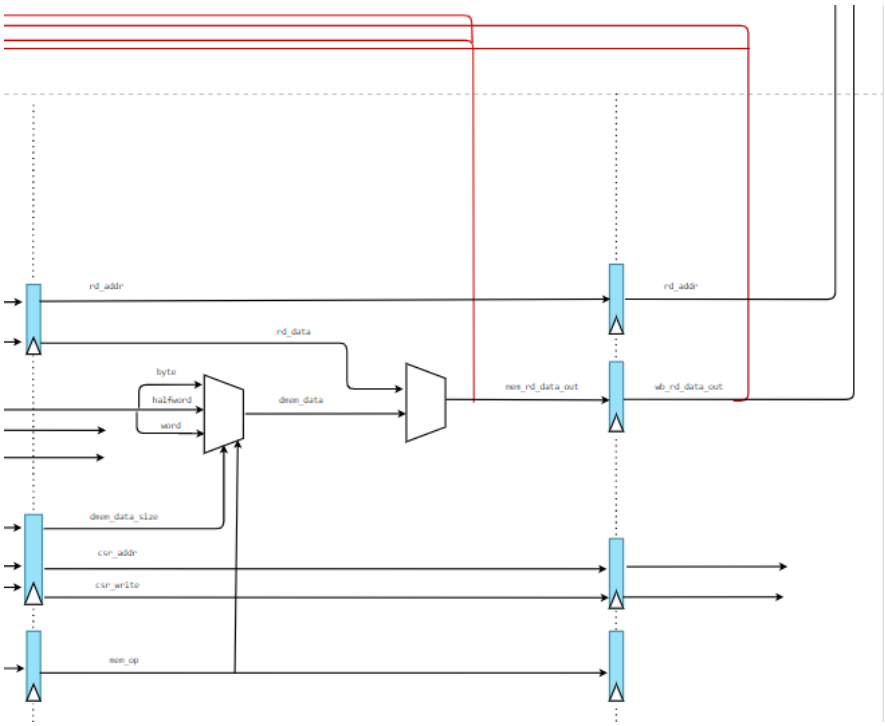
2.2.3. Στάδιο Execute-Memory

Ακολούθως φαίνεται το στάδιο execute-memory το οποίο ο κατασκευαστής ονομάζει execute λανθασμένα.



Εικόνα 11 : Στάδιο Execute-Memory

Αρχικά περιλαμβάνει μονάδες που σηματοδοτούν λανθασμένη ευθυγράμμιση εντολών και δεδομένων(instruction misalign και data misalign),μονάδα υπολογισμού διεύθυνσης προορισμού σε περίπτωση άλματος, συγκριτή που καθορίζει συνθήκες ανισο ισότητας και κατά συνέπεια αν θα πραγματοποιηθεί το άλμα υπό συνθήκη και μονάδα καθορισμού εξαιρέσεων. Με τα σήματα με κόκκινο απεικονίζονται οι τιμές καταχωρητών πηγής που προωθούνται από τα στάδια stall και writeback αντίστοιχα(χρησιμοποιούνται σε περίπτωση εξάρτησης δεδομένων) και ένας πολυπλέκτης επιλέγει ποια τιμή των rs1 και rs2 θα χρειαστεί στην αριθμητική πράξη της ALU.Οι έξοδοι των παραπάνω πολυπλεκτών οδηγούνται στην είσοδο των πολυπλεκτών που πραγματοποιούν την επιλογή του ορίσματος της ALU (register, immediate, csr, pc ,shamt, csr).Αφού γίνει η επιλογή τα ορίσματα οδηγούνται στην ALU η οποία αποτελεί συνδυαστικό κύκλωμα και η έξοδος της οδηγείται στην έξοδο του καταχωρητή προορισμού <u>αλλά και στην είσοδο διεύθυνσης της μνήμης δεδομένων σε περίπτωση που εκτελείται εντολή μνήμης</u>.Για αυτό και το στάδιο αυτό αναφέρεται ως execute-memory.



Εικόνα 12:Στάδιο Stall και στάδιο Writeback

2.2.4. Στάδια Stall και Writeback

Η έξοδος της μνήμης που περιλαμβάνει τα δεδομένα σε περίπτωση ανάγνωσης οδηγείται στο επόμενο στάδιο stall stage (ή memory όπως ονομάζεται λανθασμένα από τον κατασκευαστή) χωρίς να παρεμβάλλεται στοιχείο καθυστέρησης, στην είσοδο ενός πολυπλέκτη ο οποίος πραγματοποιεί επέκταση προσήμου προσαρμόζοντας τις αντίστοιχες τιμές σε μέγεθος 32 bit(π.χ. αν πρόκειται για byte επεκτείνεται με πρόσημο ή χωρίς κατά 24 bit, αν πρόκειται για ημιλέξη επεκτείνεται με πρόσημο ή χωρίς κατά 16 bit και αν πρόκειται για λέξη οδηγείται αυτούσιο στην έξοδο).Τέλος μέσω του δεύτερου πολυπλέκτη επιλέγεται αν θα οδηγηθεί η τιμή της ALU ή η τιμή από τη μνήμη στον καταχωρητή προορισμού(π.χ. αν έχω μια απλή αριθμητική πράξη τότε οδηγείται η τιμή της ALU στην έξοδο για τον καταχωρητή προορισμού ενώ αν έχω ανάγνωση από τη μνήμη οδηγείται η τιμή από τη πρόσβαση στη μνήμη) .Σε αυτό το σημείο αξίζει να τονιστεί ότι το κρίσιμο μονοπάτι (critical path) δηλαδή το πιο «αργό» μονοπάτι στον επεξεργαστή ξεκινά όπως φαίνεται από τους πολυπλέκτες της προώθησης των τιμών rs21, rs2 και καταλήγει στην έξοδο του πολυπλέκτη επιλογής της τιμής του καταχωρητή προορισμού.Αυτή η τιμή καθορίζει και τη μέγιστη συχνότητα λειτουργίας η οποία θα μπορούσε με ένα κλασσικό σχεδιασμό 5 σταδίων να ήταν ακόμα μεγαλύτερη.Παρόλα αυτά σε πολλούς soft processors πραγματοποιείται σύμπτυξη 2 σταδίων εφόσον η συχνότητα λειτουργίας δεν αποτελεί μείζον πρόβλημα. Πολλές φορές συνυπάρχουν soft και hard πυρήνες οι οποίοι επικοινωνούν επομένως διεργασίες που απαιτούν υψηλή απόδοση εκτελούνται στους hard πυρήνες.

Κεφάλαιο 3

Αρχιτεκτονική PMicroX

Με στόχο την αύξηση της απόδοσης και τη μείωση της δυναμικής ισχύς που καταναλώνεται στον Potato Processor αλλά και των πόρων που καταλαμβάνει, πραγματοποιήθηκαν αλλαγές σε ορισμένα σημεία της αρχιτεκτονικής του. Συγκεκριμένα βελτιώθηκε η λογική της ALU σε επίπεδο κατανάλωσης, προστέθηκε εσωτερική μνήμη δεδομένων και αφαιρέθηκε η εξωτερική μνήμη και τέλος δημιουργήθηκε σύστημα ταχείας αρχικοποίησης και λήψης δεδομένων ώστε να ελαχιστοποιηθεί ο όγκος των εντολών που εκτελούνται πριν και μετά την εκτέλεση του κυρίως προγράμματος αυξάνοντας την απόδοση. Ο PMicrox προσφέρει συγκριτικά με τον Potato processor:

- εκτέλεση εγγραφών και αναγνώσεων από τη μνήμη σε 1cc.
- ακριβέστερη μέτρηση της δυναμικής ισχύος που καταναλώνει το κυρίως πρόγραμμα μέσω της απομόνωσης από ρουτίνες αρχής και τέλους.
- λειτουργία χωρίς την παρουσία bootloader ο οποίος αντιγράφει τα δεδομένα της uart στη μνήμη δεδομένων
- παράλειψη των ρουτινών αντιγραφής δεδομένων από τη μνήμη δεδομένων στη στοίβα που ενδέχεται να είναι χρονοβόρες σε εφαρμογές με μεγάλες εισόδους, επιτρέποντας τη μείωση του χρόνου εκτέλεσης.
- εκτέλεση με χρήση της ελάχιστης δυνατής μνήμης (δεδομένων και εντολών)

3.1. Βελτιωμένη αριθμητική Μονάδα ALU

Παρατηρείται ότι στη αριθμητική μονάδα χρησιμοποιούνται τελεστές για να υλοποιήσουν τις πράξεις και έπειτα τα αποτελέσματα όλων

αυτών των πράξεων οδηγούνται σε έναν πολυπλέκτη ο οποίος επιλέγει ποιο αποτέλεσμα θα οδηγήσει στην έξοδο. Αυτό σημαίνει πως οι είσοδοι x, y οδηγούνται σε όλα τα επιμέρους κυκλώματα και κατά συνέπεια σε κάθε κύκλο εκτελούνται όλες οι πράξεις της ALU.Όπως γίνεται αντιληπτό κάτι τέτοιο είναι καταστροφικό ως προς την καταναλισκόμενη ισχύ.Επομένως όπως αναφέρεται στο [19] εφαρμόζεται η τεχνική operand gating για τα ορίσματα της ALU. Πριν από κάθε αριθμητική μονάδα τοποθετήθηκε ένας πολυπλέκτης ο οποίος για κάθε άλλη πράξη εκτός από τη αυτή της μονάδας που οδηγεί τροφοδοτεί μηδενικές εισόδους. Η νέα αρχιτεκτονική της ALU φαίνεται παρακάτω στο υποκεφάλαιο 5.1 στην Εικόνα 39. Πραγματοποιήθηκε σύγκριση της αρχικής και της νέας αρχιτεκτονικής, εκτελώντας τους αλγόριθμους fir filter, 2D convolution και matrix multiplication. Σε σύνολο 20 implementations το μέσο μέγεθος του κέρδους δυναμικής ισχύος υπολογίστηκε 28.4%, 29.6% και 29.2% αντίστοιχα.

3.2. Τροποποίηση σταδίου Execute-Memory

Όπως αναφέρθηκε και παραπάνω δεν χρησιμοποιήθηκε η εξωτερική μνήμη δεδομένων αλλά τοποθετήθηκε εσωτερική μνήμη με καθυστέρηση 1 κύκλο ρολογιού. Αυτό είχε ως συνέπεια να μην χρειάζεται η συνδυαστική λογική που χρησιμοποιούνταν για να πραγματοποιεί καθυστέρηση του σταδίου execute-memory συνεπώς αφαιρέθηκε.

3.3. Δημιουργία συστήματος ταχείας αρχικοποίησης μνημών και λήψης δεδομένων.

Έχοντας ως στόχο ενσωματωμένα συστήματα ειδικού σκοπού τα οποία εκτελούν συγκεκριμένα περιορισμένα τμήματα κώδικα σε μεγάλο αριθμό επαναλήψεων θεωρήθηκε ασύμφορο να εκτελούνται κώδικες κώδικες περιφερειακών και ρουτίνες αρχικοποιήσεων οι οποίες πολλές φορές υπερβαίνουν τον αριθμό εντολών του κυρίως προγράμματος. Έτσι έγινε μια προσπάθεια ελαχιστοποίησης των εντολών που εκτελούνται πριν και μετά το κυρίως πρόγραμμα μέσω της δημιουργίας ενός συστήματος το οποίο τοποθετεί τα δεδομένα εισόδου(εντολές, δεδομένα) που απαιτούνται για την εκάστοτε εφαρμογή απευθείας στη στοίβα ώστε να παραλείπονται περιττές αντιγραφές δεδομένων(ειδικά σε εφαρμογές με μεγάλα σετ δεδομένων η εκτέλεση με το προτεινόμενο σύστημα αρχικοποίησης-λήψης επιταχύνεται σημαντικά).

3.3.1 Μεταφορά και τοποθέτηση του εκτελέσιμου στη μνήμη

Η διαδικασία αρχικοποίησης ενός πυρήνα προκειμένου να εκτελέσει την εκάστοτε εφαρμογή είναι κοινή σε όλες τις αρχιτεκτονικές και περιλαμβάνει συγκεκριμένα διακριτά στάδια. Αρχικά παράγεται από τον πηγαίο κώδικα ένα εκτελέσιμο αρχείο μέσω του μεταγλωττιστή(compiler) ο οποίος στοχεύει την συγκεκριμένη αρχιτεκτονική (στην περίπτωση μας την αρχιτεκτονική RISC-V).Το αρχείο αυτό αποτελείται από ορισμένα τμήματα (θα αναλυθούν εκτενώς παρακάτω) τα οποία αντιστοιχίζονται σε συγκεκριμένες περιοχές διευθύνσεων της εξωτερικής μνήμης δεδομένων.Αποστέλλεται συνήθως στον επεξεργαστή μέσω σειριακής εισόδου (uart) όταν δεν υπάρχει λειτουργικό σύστημα να πλαισιώνει τον πυρήνα(bare metal εφαρμογές). Από την σειριακή είσοδο αντιγράφεται στην εξωτερική μνήμη RAM μέσω του bootloader ο οποίος είναι ένα τμήμα κώδικα που εκτελείται στην αρχή πριν κάθε εφαρμογή. Είναι αποθηκευμένο στη μνήμη ROM και αποτελεί μέρος του bitstream. Δηλαδή η μνήμη ROM είναι αρχικοποιημένη με αυτό τον κώδικα πριν αρχίσει η λήψη του εκτελέσιμου αρχείου από τη σειριακή είσοδο. Εφόσον τελειώσει η λήψη του αρχείου και η αντιγραφή του στη μνήμη το πρόγραμμα μεταβαίνει στη θέση 0 της μνήμης προκειμένου να αρχίσει η εκτέλεση του προγράμματος.

Στον Potato processor όπως αναγράφεται στις οδηγίες εκτέλεσης αρχικά προστίθεται block RAM IP η οποία χρησιμοποιείται ως ROM και αρχικοποιείται μέσω του αρχείου coefficient(με επέκταση .coef) το οποίο περιλαμβάνει τον bootloader.

3.3.2. Παραγωγή εκτελέσιμου του PMicroX χωρίς εξωτερική μνήμη

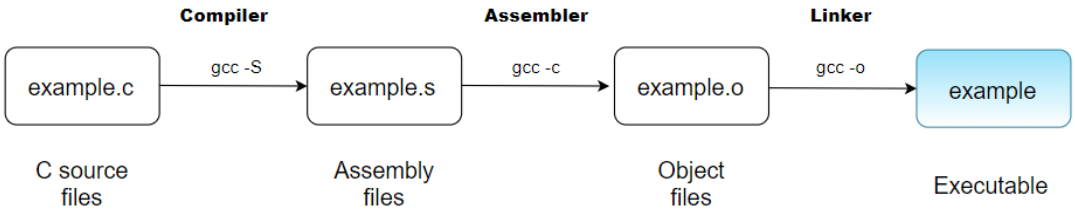
Στην προσπάθεια να κατασκευαστεί ένα κατά το δυνατόν μινιμαλιστικό design το οποίο να απομονώνει το τμήμα κώδικα της εφαρμογής από τμήματα κώδικα που εκτελούνται πριν και μετά από αυτό και ρόλος τους είναι η αρχικοποίηση μνημών και η αντιγραφή δεδομένων(π.χ. bootloader) σε διάφορες περιοχές της μνήμης, προτείνεται ένας εναλλακτικός τρόπος αρχικοποίησης δεδομένων και λήψης αποτελεσμάτων. Σύμφωνα με όσα αναγράφονται στην αρχειοθέτηση του potato processor για την επικοινωνία του πυρήνα με τον χρήστη διατίθεται η σειριακή θύρα η οποία εντάσσεται στην κατηγορία των περιφερειακών συσκευών και επομένως απαιτεί τη χρήση της διεπαφής

wishbone για να επικοινωνεί με τον πυρήνα. Κατ' αυτό τον τρόπο όμως το design γίνεται αρκετά συνθετότερο από το απλό datapath του RISC-V με αποτέλεσμα την δυσκολία απομόνωσης του και καταμέτρησης της δυναμικής ισχύς που καταναλώνει.

Οι εναλλακτικές μέθοδοι που προτείνονται είναι **δύο** και διακρίνονται ως προς τον τύπο του αρχείου που τροφοδοτούμε στον πυρήνα.

3.3.2.1. Ροή εξαγωγής εκτελέσιμου

Εδώ θα γίνει μια αναφορά στην ροή μεταγλώττισης ενός αρχείου πηγαίου κώδικα η οποία καταλήγει στην παραγωγή ενός εκτελέσιμου αρχείου. Όπως φαίνεται στην *Εικόνα 13* τα στάδια είναι compile, assemble και link.



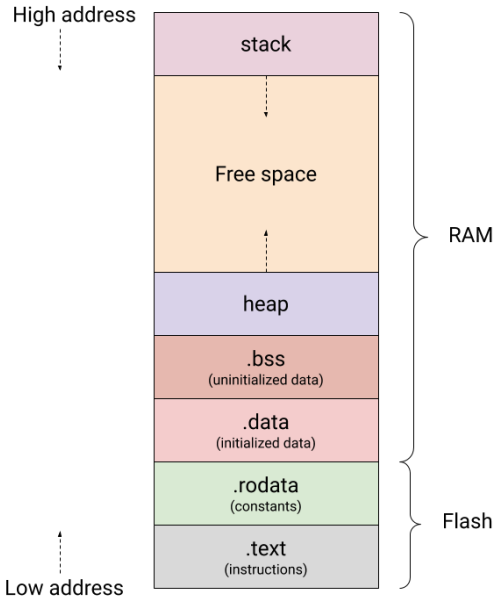
Εικόνα 13 : Ροή εξαγωγής εκτελέσιμου αρχείου

Κατά το 1ο στάδιο ο μεταγλωττιστής παράγει το αρχεία assembly τα οποία είναι το κατώτερο δυνατό αναγνώσιμο επίπεδο κώδικα πριν μετατραπεί σε κώδικα μηχανής (δυαδικό αρχείο). Το πηγαίο αρχείο περιλαμβάνει και βιβλιοθήκες οι οποίες μετατρέπονται και αυτές σε μορφή assembly. Στη συνέχεια, στο 2ο στάδιο, μετατρέπονται όλα τα assembly αρχεία σε object αρχεία δηλαδή δυαδικά αρχεία (γλώσσα μηχανής).Τέλος στο 3ο στάδιο ενώνονται όλα τα δυαδικά αρχεία (κυρίου κώδικα, βιβλιοθηκών, επιπρόσθετων τμημάτων κώδικα αρχής και τέλους) με σκοπό τη δημιουργία ενός ενιαίου δυαδικού αρχείου.

3.3.2.2. Δομή εκτελέσιμου στη μνήμη

Το αρχείο αυτό έχει συγκεκριμένη δομή η οποία φαίνεται στην *Εικόνα 14.* Ξεκινώντας από την θέση μνήμης 0 και αυξάνοντας προς μεγαλύτερες διευθύνσεις διακρίνονται τα εξής τμήματα [20] :

1. **.text** : Εδώ αποθηκεύονται οι εντολές του προγράμματος
2. **.data** : Εδώ αποθηκεύονται οι αρχικοποιημένες global μεταβλητές και οι αρχικοποιημένες static τοπικές μεταβλητές.
3. **.rodata** : Εδώ αποθηκεύονται οι σταθερές τιμές (const) που χρησιμεύουν μόνο για ανάγνωση. Τα περισσότερα λειτουργικά συστήματα δεν διαθέτουν τμήμα



Εικόνα 14 : Εικόνα εκτελέσιμου στη μνήμη

μόνο ανάγνωσης (read-only data segment) επομένως τα περιεχόμενα του .rodata μεταβιβάζονται στην περιοχή του κώδικα εντολών επειδή είναι read-only, ή στην περιοχή δεδομένων(data segment) επειδή αποτελούν δεδομένα.Εφόσον ο μεταγλωττιστής δεν γνωρίζει ποια πολιτική έχει υιοθετηθεί από το λειτουργικό σύστημα, δημιουργεί αυτό το τμήμα.

4. **.bss** : Εδώ αποθηκεύονται μη αρχικοποιημένες global μεταβλητές και μη αρχικοποιημένες static τοπικές μεταβλητές. Γενικά περιλαμβάνει μεταβλητές που πρέπει να αρχικοποιηθούν στο μηδέν.

Όλες οι τοπικές μεταβλητές (εντός της συνάρτησης main) τοποθετούνται στη στοίβα (stack) είται είναι αρχικοποιημένες είτε όχι. Αν η μεταβλητή είναι αρχικοποιμένη θα γραφτεί η νέα αρχικοποιημένη τιμή, ενώ αν όχι τότε αφήνεται όπως είναι χωρίς να πραγματοποιείται κάποια ενέργεια. Αυτό σημαίνει ότι σε αυτή τη θέση πιθανόν να υπάρχουν «άχρηστα δεδομένα» (memory trash). Για την εισαγωγή και εξαγωγή δεδομένων στη στοίβα χρησιμοποιείται ένας δείκτης (stack pointer) ο οποίος αρχικοποιείται στην μεγαλύτερη θέση μνήμης (π.χ. sp = mem_end)

και μειώνεται όταν εισάγονται δεδομένα ή αντίστοιχα αυξάνονται όταν αφαιρούνται.

Σε περίπτωση που πραγματοποιείται δυναμική εκχώρηση μνήμης (με χρήση του τελεστή new) τότε οι μεταβλητές (pointers) στις οποίες αποδίδονται οι αρχικές διευθύνσεις της δυναμικής εκχώρησης τοποθετούνται στο stack και δεσμεύονται τα αντίστοιχα τμήματα μνήμης στο σωρό (heap). Ο τελευταίος αναπτύσσεται από τιςς μικρές προς τις μεγάλες διευθύνσεις.

3.3.2.3. Τροποποίηση εικόνας μνήμης με χρήση του Linker script

Η εικόνα του εκτελέσιμου στην κύρια μνήμη όπως αναφέρθηκε παραπάνω είναι ενδεικτική.Οι τιμές διευθύνσεων στις οποίες αρχίζει και τελειώνει το κάθε τμήμα μπορούν να τροποποιηθούν μέσω οδηγιών που δίνονται στο linker για το μέγεθος και τον αριθμό των τμημάτων. Αυτές οι οδηγίες παρέχονται μέσω του αρχείο linker script (έχει επέκταση .ld) το οποίο μπορεί να δοθεί ως flag κατά το compile στο Makefile με τη μορφή :

```
-T my_linker_script.ld
```

Προκειμένου να γίνουν κατανοητές οι αλλαγές που έγιναν στη δομή του εκτελέσιμου το οποίο είναι προσαρμοσμένο στη νέα απλουστευμένη μορφή του potato processor ακολουθεί μια αναφορά στη δομή και τις βασικές ιδιότητες του κώδικα του linker script.

Κύριος σκοπός του είναι να περιγράψει τον τρόπο με τον οποίο τα διάφορα τμήματα των αρχείων εισόδου θα αντιστοιχιστούν στο αρχείο εξόδου, διαμορφώνοντας την εικόνα της μνήμης του προγράμματος.

Αν ο χρήστης δεν παρέχει κάποιο αρχείο κατα το link , ο compiler επιλέγει το δικό του προεπιλεγμένο αρχείο. Τα αρχεία εισόδου και εξόδου ονομάζονται object αρχεία αν και το αρχείο εξόδου συχνά αναφέρεται και ως εκτελέσιμο. Κάθε object αρχείο περιλαμβάνει μια λίστα από ενότητες [21] και κάθε μια από αυτές διαθέτει όνομα και μέγεθος. Οι περισσότερες ενότητες σχετίζονται με ένα μπλοκ δεδομένων, που περιλαμβάνει τα περιεχόμενα της ενότητας. Μια ενότητα μπορεί να χαρακτηριστεί ως φορτώσιμη (loadable), δηλαδή τα δεδομένα της πρέπει να φορτωθούν στη μνήμη όταν το αρχείο εξόδου εκτελείται. Αν δεν διαθέτει περιεχόμενα χαρακτηρίζεται ως κατανεμητέα (allocatable) που σημαίνει ότι ένα μια περιοχή της μνήμης πρέπει να δεσμευτεί, χωρίς όμως να αποθηκεύεται κάποιο δεδομένο. (Σε ορισμένες περιπτώσεις αυτό το τμήμα πρέπει να μηδενιστεί). Οι ενότητες που δεν ανήκουν σε καμία από τις παραπάνω κατηγορίες περιλαμβάνουν πληροφορίες αποσφαλμάτωσης.

3.3.2.3.1. Εικονική και φορτώσιμη διεύθυνση μνήμης

Κάθε φορτώσιμη ή κατανεμητέα ενότητα διαθέτει δύο διευθύνσεις. Η πρώτη ονομάζεται εικονική διεύθυνση μνήμης (VMA) και είναι η διεύθυνση που θα έχει αυτή η ενότητα όταν το αρχείο εξόδου εκτελείται. Η δεύτερη ονομάζεται διεύθυνση μνήμης φόρτωσης (LMA) και είναι η διεύθυνση στην οποία θα φορτωθεί η ενότητα. Στις περισσότερες περιπτώσεις οι δύο αυτές διευθύνσεις ταυτίζονται. Όμως γίνεται να διαφέρουν όπως για παράδειγμα όταν το κομμάτι δεδομένων (data section) φορτώνεται στη μνήμη ROM και έπειτα αντιγράφεται στη μνήμη RAM κατα την εκκίνηση του προγράμματος. (Αυτές οι τεχνικές χρησιμοποιούνται για να αρχικοποιηθούν global μεταβλητές σε ένα σύστημα βασισμένο σε ROM).

Κάθε object αρχείο διαθέτει επίσης μια λίστα συμβόλων, που αναφέρεται και ως πίνακας συμβόλων. Ένα σύμβολο μπορεί να είναι ορισμένο ή αόριστο και διαθέτει ένα όνομα. Τα ορισμένα σύμβολα διαθέτουν μεταξύ άλλων και μια διεύθυνση. Κατα τη μεταγλώττιση ενός πηγαίο κώδικα σε object αρχείο, οι ορισμένες μεταβλητές και συναρτήσεις αντιστοιχίζονται σε ορισμένα σύμβολα, ενώ οι απροσδιόριστες μεταβλητές και συναρτήσεις μετατρέπονται σε αόριστα σύμβολα.

3.3.2.3.2. Ενδεικτική δομή Linker script

Παρακάτω δίνεται η απλούστερη δυνατή μορφή ενός αρχείο linker [22]. Περιλαμβάνει μόνο την εντολή 'SECTIONS' η οποία περιγράφει το σχέδιο της μνήμης. Έστω ότι το πρόγραμμα αποτελείται μόνο από τις εντολές, αρχικοποιημένα και μη αρχικοποιημένα δεδομένα.Τα τμήματα που αντιστοιχίζονται είναι '.text ' , '.data' και '.bss' αντίστοιχα.Επίσης έστω ότι οι εντολές πρέπει να φορτωθούν στη διεύθυνση 0x10000, και τα δεδομένα πρέπει να ξεκινούν από τη διεύθυνση 0x8000000. Παρακάτω φαίνεται ένα σύντομο κομμάτι κώδικα που υλοποιεί όσα αναφέρθηκαν.

```
SECTIONS
{
  . = 0x10000;
  .text : { *(.text) }
  . = 0x8000000;
  .data : { *(.data) }
  .bss : { *(.bss) }
}
```

Η πρώτη γραμμή του παραπάνω κώδικα θέτει την τιμή του ειδικού συμβόλου '.' που είναι ο μετρητής τοποθεσίας. Αν δεν διευκρινιστεί η διεύθυνση της ενότητας εξόδου με κάποιο άλλο τρόπο(θα αναφερθούν και άλλοι τρόποι παρακάτω), τότε η διεύθυνση ορίζεται από την τρέχουσα τιμή του μετρητή τοποθεσίας και έπειτα αυξάνεται κατά το μέγεθος της ενότητας. Στην αρχή της εντολής 'SECTIONS' ο μετρητής τοποθεσίας είναι μηδέν.

Η δεύτερη γραμμή ορίζει την ενότητα εξόδου '.text'. Εσωτερικά των αγκυλών μετά το όνομα της ενότητας εξόδου , παρατίθεται η λίστα με τις ενότητες εισόδου που θα τοποθετηθούν στην ενότητα εξόδου. Το '*' υποδηλώνει ότι ταιριάζει σε οποιοδήποτε όνομα αρχείου και η έκφραση '*(.text)' σημαίνει όλες οι ενότητες εισόδου '.text' σε όλα τα αρχεία εισόδου. Εφόσον ο μετρητής τοποθεσίας είναι '0x10000' όταν ορίζεται η ενότητα εξόδου '.text' , ο linker θα θέσει τη διεύθυνση αυτής της ενότητας του αρχείο εξόδου σε '0x10000' .

Οι δύο εναπομείνασες γραμμές ορίζουν τις ενότητες εξόδου '.data' και '.bss' . Ο linker τοποθετεί την ενότητα '.data' στη διεύθυνση '0x8000000' και ο μετρητής τοποθεσίας αποκτά αυτή την τιμή συν το μέγεθος του τμήματος '.data'. Συνεπώς η ενότητα '.bss' τοποθετείται αμέσως μετά την ενότητα '.data'.

Εδώ αξίζει να αναφερθεί η έννοια της ευθυγράμμισης (alignment). Μια διεύθυνση μνήμης ονομάζεται ευθυγραμμισμένη κατα n-byte όταν είναι πολλαπλάσιο των n-byte και χρησιμοποιείται ώστε να αποφεύγονται οι περιττές φορτώσεις δεδομένων που καθυστερούν την εκτέλεση μειώνοντας την απόδοση. Ο linker διατηρεί την ευθυγράμμιση των ενοτήτων εξόδου, αυξάνοντας όπου χρειάζεται τον μετρητή τοποθεσίας, δημιουργώντας ένα μικρό κενό μεταξύ των ενοτήτων [23] .

Ο linker διαθέτει προεπιλεγμένη κατανομή της διαθέσιμης μνήμης στις διάφορες ενότητες. Η κατανομή αυτή τροποποιείται με χρήση της εντολής 'MEMORY' η οποία περιγράφει την τοποθεσία και το το μέγεθος των μπλοκ της εικόνας μνήμης εξόδου[24].Η εντολή χρησιμοποιείται για να καθορίσει ποιές ενότητες θα χρησιμοποιηθούν από τον linker και ποιες όχι, αναθέτοντας στη συνέχεια τις ενότητες αυτές σε συγκεκριμένες περιοχές της μνήμης. Ο linker θα θέσει τις διευθύνσεις των ενοτήτων βασιζόμενος στις διαθέσιμες περιοχές μνήμης και σηματοδοτεί τις περιπτώσεις στις οποίες οι ενότητες είναι εξαντλημένες σε χώρο.

Μόνο μια χρήση της εντολής 'MEMORY' επιτρέπεται σε ένα linker script, αλλά είναι να δυνατόν να οριστεί αυθαίρετος αριθμός από περιοχές μνήμης μέσα σε αυτή.
Η τυπική σύνταξη της εντολής φαίνεται ακολούθως :

```
MEMORY
  {
    name [(attr)] : ORIGIN = origin, LENGTH = len
    …
  }
```

Το πεδίο ‘ name ‘ χρησιμοποιείται για να αναφερθεί στο όνομα της περιοχής μνήμης (π.χ. ram, rom)

Το πεδίο ‘ attr ‘ είναι μια προαιρετική λίστα χαρακτηριστικών που καθορίζουν αν θα χρησιμοποιηθεί κάποια συγκεκριμένη περιοχή μνήμης για να τοποθετηθεί μία ενότητα εισόδου η οποία δεν έχει αντιστοιχιστεί σαφώς στο αρχείο του linker.

Αν δεν οριστεί όνομα ενότητας εξόδου για κάποια ενότητα εισόδου τότε θα δημιουργηθεί ενότητα εξόδου με το ίδιο όνομα με την ενότητα εισόδου. Αν οριστούν χαρακτηριστικά περιοχής, ο linker θα τα χρησιμοποιήσει για να επιλέξει την περιοχή μνήμης για κάθε ενότητα εξόδου που δημιουργεί. Οι τιμές που μπορεί να πάρει το πεδίο ‘ attr ‘ είναι :

‘R’ -> Ενότητα ανάγνωσης μόνο (Read-only)
‘w’ -> Ενότητα ανάγνωσης/εγγραφής (Read/write)
‘X’ -> Ενότητα εκτελέσιμη (Executable)
‘A’ -> Ενότητα κατανεμητέα (Allocatable)
‘I’ -> Ενότητα αρχικοποιημένη (Initialized)
‘L’ -> Ίδιο με το ‘I’
‘!’ -> Αντιστροφή της έννοιας όλων των χαρακτηριστικών που ακολουθούν

Το πεδίο ‘ origin ‘ είναι μια αριθμητική έκφραση (σταθερά) για την διεύθυνση αρχής της περιοχής.
Το πεδίο ‘ len ‘ είναι μια έκφραση για το μέγεθος της περιοχής σε byte.

Εφόσον ορίστηκαν οι δύο βασικές εντολές ‘ SECTIONS ‘ και ‘ MEMORY ‘ δίνεται ένα συνδυαστικό παράδειγμα χρήσης τους.

```
MEMORY { rom : ORIGIN = 0x1000, LENGTH = 0x1000 }
SECTIONS { ROM : { *(.text) } >rom }
```

Ορίζεται η περιοχή μνήμης rom με διεύθυνση αρχής 0x1000 και μέγεθος 0x1000 = 4096 bytes.

Στην επόμενη γραμμή ορίζεται η ενότητα εξόδου ROM η οποία περιλαμβάνει την ενότητα εισόδου ‘.text’ και τοποθετείται στην περιοχή μνήμης rom.

Παραπάνω έγινε αναφορά στην εικονική διεύθυνση και τη διεύθυνση φόρτωσης μιας ενότητας. Η εικονική διεύθυνση εφόσον έχει δημιουργηθεί περιοχή μνήμης που να ανήκει η ενότητα είναι η πρώτη ελεύθερη θέση μνήμης της περιοχής. Στο παραπάνω παράδειγμα η εικονικη διεύθυνση μνήμης της ενότητας ROM είναι η αρχή της περιοχής μνήμης rom, εφόσον δεν έχει εισαχθεί προηγουμένως κάποια άλλη ενότητα και ισούται με 0x1000.

Η διεύθυνση φόρτωσης ορίζεται από το πρόθεμα ‘ AT ‘ ή ‘ AT > ‘ και ο ορισμός της είναι προαιρετικός [25]. Χρησιμοποιείται για να διευκολύνει τη δημιουργία εικόνας ROM. Όταν για παράδειγμα η ενότητα ‘ .data ‘ που περιλαμβάνει αρχικοποιημένα δεδομένα έχει τοποθετηθεί αρχικά στη μνήμη ROM και κατα την εκτέλεση πρέπει να αντιγραφεί στη μνήμη RAM τότε χρησιμοποιείται το πρόθεμα αυτό στο linker script σε συνδυασμό με ένα τμήμα κώδικα στην αρχή, πριν την εκτέλεση του προγράμματος που πραγματοποιεί την αντιγραφή του τμήματος από την μια περιοχή της μνήμης στην άλλη εκμεταλλευόμενο τα σύμβολα που ορίζονται από τον linker.

3.3.2.3.3. Ενδεικτική δομή Linker script PMicroX απουσία εξωτερικής μνήμης

Στην αρχιτεκτονική του PMicroX η οποία δεν διαθέτει ενιαία εξωτερική μνήμη αλλά δύο εσωτερικές μνήμες (ram και rom) πρέπει να τροποποιηθεί το αρχείο του linker ώστε να προσαρμόζεται κατάλληλα. Παρακάτω δίνεται μια υπεραπλουστευμένη μορφή του αρχείου αυτού η οποία τονίζει τις βασικές αλλαγές σε σχέση με το ήδη υπάρχον κώδικα του linker.

```
MEMORY

{
  rom (rx) : ORIGIN = 0x00000000, LENGTH = 4096
  ram (rwx) : ORIGIN = 0x00000000, LENGTH = 4096
}


SECTIONS
{
 ...
        .text :
        {
          ...
        } > rom

        .rodata :
        {
          ...
        } > ram AT > rom

        .data
        {
         ...
        } > ram AT > rom
```

```
        .bss
        {
         …
        } > ram AT > rom
…
}
```

Αρχικά παρατηρούμε ότι οι τιμές με μπλε που καθορίζουν την αρχή της περιοχής μνήμης ram και την αρχή της περιοχής μνήμης rom είναι και οι δυο ίσες με μηδέν. Αυτό συμβαίνει διότι δεν αναφερόμαστε σε μια ενιαία μνήμη στην οποία το τμήμα rom είναι ένα κομμάτι αυτής και το τμήμα ram είναι ένα μεταγενέστερο κομμάτι αλλά αντιθέτως εδώ θέλουμε να αναφερθούμε σε δύο διαφορετικές μνήμες με διαφορετικές διευθύνσεις. Σκοπός είναι να τοποθετηθεί η ενότητα ‘ .text ‘ στην αρχή της μνήμης rom και οι ενότητες ‘ .rodata ‘ , ‘ .data ’ και ‘ .bss ‘ με αυτή τη σειρά στην μνήμη ram. Κατ’ αυτόν τον τρόπο δεν θα υπάρχουν αντιγραφές τμημάτων όταν ξεκινά η εκτέλεση αλλά αντιθέτως τα τμήματα θα βρίσκονται ήδη στις διευθύνσεις που περιμένει να τα βρει το πρόγραμμα για να εκτελεστεί. Αν στη συνέχεια κάθε ενότητα αντιστοιχιστεί ρητά είτε στο τμήμα rom είτε στο τμήμα ram τότε θα προκύψει σφάλμα επικάλυψης τμημάτων από τον μεταγλωττιστή κατά την διάρκεια του linking. Για αυτό το σκοπό χρησιμοποιείται η σύνταξη ram AT > rom η οποία πληροφορεί τον μεταγλωττιστή ότι η εικονική διεύθυνση είναι η 0x00000000 αλλά τα τμήματα αυτά θα φορτωθούν από τη rom. Κάτι τέτοιο δεν συμβαίνει στη λειτουργία που προτείνεται αλλά επιλύει το σφάλμα επικάλυψης τμημάτων.

3.3.2.4. Επιβεβαίωση εικόνας μνήμης με χρήση του εργαλείου objdump

Το ζητούμενο αποτέλεσμα επιβεβαιώνεται με χρήση των εργαλείων gnu binutils [26] τα οποία χρησιμεύουν στην εξαγωγή, διαχείριση και ανάλυση διαφόρων μορφών αρχείων όπως δυαδικών, object και assembly. Συγκεκριμένα με την βοήθεια του εργαλείου objdump λαμβάνονται πληροφορίες για ένα object αρχείο όπως το περιεχόμενο των ενοτήτων που διαθέτει και οι διευθύνσεις στις οποίες βρίσκεται κάθε μια. Στο παρόν εκτελέσιμο - object αρχείο που προέκυψε με τη χρήση του παραπάνω linker αρχείου, σύμφωνα με το objdump binutil η ενότητα ‘ .text ‘ έχει διεύθυνση 0x00000000 και όπως και η ενότητα ‘ .rodata ‘. Η ενότητα ‘ .data ‘ τοποθετείται ακριβώς μετά την ενότητα ‘ .rodata ‘. Ακολουθεί η ενότητα ‘ .bss ‘ και έπειτα τοποθετούνται ο σωρός που αναπτύσσεται προς τις μεγαλύτερες θέσεις μνήμης και η στοίβα που αναπτύσσεται προς τις μικρότερες θέσεις μνήμης.

Στην ενότητα ‘ .text ‘ δεν βρίσκονται μόνο οι εντολές της ‘ main ‘ συνάρτησης αλλά και διάφορες ρουτίνες αρχικοποίησης υπεύθυνες να

αντιγράψουν ή να αρχικοποιήσουν τμήματα της στοίβας πριν την εκτέλεση. (π.χ. περιεχόμενα της ενότητας ‘ .data ‘ ή ‘ .rodata ‘).

Στη συνέχεια μέσω σκριπτ απομονώνονται οι ενότητες του εκτελέσιμου και μετατρέπονται σε δυο δεκαεξαδικά αρχεία(dmem.hex και imem.hex για την αρχικοποίηση των εσωτερικών μνημών και δεδομένων.

3.3.2.5. Μεταγλώττιση bare metal εφαρμογών

Για την εξαγωγή του εκτελέσιμου χρησιμοποιείται Makefile το οποίο διαθέτει flags για μεταγλώττιση bare metal εφαρμογών. Σε ένα απλουστευτικό σύστημα χωρίς λειτουργικό όπως ο πυρήνας PMicroX ο μεταγλωττιστής πρέπει να καθοδηγηθεί ώστε να συνδέσει τις πλέον απαραίτητες βιβλιοθήκες ώστε να εξοικονομηθεί χώρος. Ακολουθεί επεξήγηση ορισμένων βασικών flags που χρησιμοποιήθηκαν κατα την μεταγλώττιση στο Makefile.

-ffreestanding : Ενημερώνει τον μεταγλωττιστή ότι η στάνταρ βιβλιοθήκη ενδέχεται να μην υπάρχει, επομένως να μην υποθέσει πως αυτή υπάρχει. Αυτό το flag δεν είναι απαραίτητο όταν εκτελείται η εφαρμογή σε φιλοξενούμενο περιβάλλον (με ύπαρξη λειτουργικού συστήματος)

-nostartfiles : Ενημερώνει τον μεταγλωττιστή ώστε να μην συνδέσει στάνταρ προεπιλεγμένα αρχεία εκκίνησης (όπως το αρχείο crt0 το οποίο είναι συνήθως γραμμένο σε assembly και περιλαμβάνει κυρίως ρουτίνες αρχικοποίησης).

-nostdlib : Ενημερώνει τον μεταγλωττιστή ώστε να μη χρησιμοποιήσει στάνταρ βιβλιοθήκες συστήματος.

-nodefaultlibs : Ενημερώνει τον μεταγλωττιστή ώστε να μην χρησιμοποιήσει προεπιλεγμένες συνδεόμενες βιβλιοθήκες.

Χρησιμοποιώντας το binutil size γίνεται σύγκριση του εκτελέσιμου με τις προεπιλεγμένες ρυθμίσεις μεταγλώττισης και σύνδεσης (αριστερά) και του εκτελέσιμου του PMicroX τα οποία παράγονται από τον ίδιο κώδικα C(δεξιά):

section	size	addr
.text	1956	65652
.rodata	100	67608
.eh_frame	4	71804
.init_array	8	71808
.fini_array	4	71816
.data	1064	71824
.sdata	12	72888
.bss	28	72900
.comment	18	0
.riscv.attributes	33	0
Total	3227	

section	size	addr
.text	784	0
.rodata	100	0
.data	0	100
.bss	0	100
.comment	18	0
.riscv.attributes	37	0
Total	939	

3.3.3. Τροφοδότηση αρχείου object αντι του εκτελέσιμου

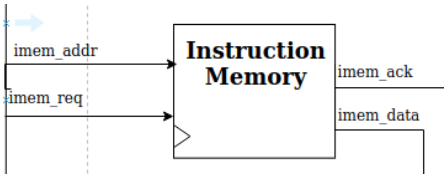
Η εναλλακτική μέθοδος που μπορεί να εφαρμοστεί παραλείπει τελείως το τελευταίο στάδιο του μεταγλωττιστή που είναι το linking. Με χρήση του flag -c εξάγεται το object αρχείο που περιλαμβάνει αποκλειστικά τον κώδικα της main. Σε αυτόν δεν έχει γίνει εκχώρηση διευθύνσεων αλλά όλες είναι σε μορφή σχετικής διεύθυνσης. Οι προσβάσεις στη μνήμη γίνονται στη στοίβα, θεωρώντας ότι τα απαραίτητα δεδομένα έχουν αντιγραφεί στις αντίστοιχες θέσεις στη στοίβα. Επειδή χρησιμοποιείται μόνο η ενότητα ‘ .text ‘, ο πηγαίος κώδικας σε αυτή τη μέθοδο δεν διαθέτει αρχικοποιήσεις πινάκων και μεταβλητών.Οι τιμές των μεταβλητών και πινάκων εισόδου δίνονται ως είσοδο πριν την εκκίνηση του προγράμματος και τοποθετούνται στις κατάλληλες διευθύνσεις στη στοίβα, στα σημεία που αναμένεται να βρεθούν από τον κώδικα. Προκειμένου να καθοριστούν οι διευθύνσεις στην εσωτερική μνήμη δεδομένων στα οποία θα μπει κάθε αρχική τιμή, δημιουργήθηκε το παρακάτω σύστημα αρχικοποίησης.

Βασικό πλεονέκτημα της μεθόδου αυτής είναι η σημαντική εξοικονόμηση μνήμης. Στη μνήμη εντολών αποθηκεύονται μόνο οι εντολές της συνάρτησης main() και όχι ρουτίνες αντιγραφής δεδομένων, ρουτίνες εξόδου και ρουτίνες construcors και destructors που δεν καλούνται ποτέ από τη main(). Η μνήμη δεδομένων πλεον αποτελείται μόνο από τη στοίβα το μέγεθος της οποίας μπορεί να προσαρμοστεί εύκολα (με χρήση generic μεγέθους μνήμης δεδομένων στο κορυφαίο ιεραρχικά αρχείο). Έτσι εξατομικεύονται οι απαιτήσεις μνήμης της κάθε εφαρμογής. Επίσης μειώνεται σημαντικά ο χρόνος εκτέλεσης της εφαρμογής εφόσον απουσιάζει η λογική του bootloader που αντιγράφει ολόκληρο το εκτελέσιμο από τη uart στη μνήμη όπως επίσης και οι ρουτίνες πριν τη main() για αντιγραφή δεδομένων από τη μνήμη ram στη στοίβα. Τέλος παρέχει τη δυνατότητα για έναν κώδικα να δοκιμαστούν πολλά σετ δεδομένων εξοικονομώντας χρόνο. Στις περισσότερες υλοποιήσεις soft processor οπως και στην αρχική μορφή του potato, πρέπει κάθε πρόγραμμα να μεταγλωττιστεί να και να σταλεί μέσω της σειριακής θύρας ενώ εκτελείται ο bootloader για να δοκιμάσουμε ένα σετ δεδομένων. Αντίθετα σε αυτή τη μέθοδο είναι δυνατόν να τροφοδοτούμε το hardware με νέες εισόδους δεδομένων διατηρώντας το ίδιο αρχείο εντολών σε διαστήματα που επιτρέπουν να έχει εκτελεστεί το προηγούμενο σετ και έχουν ληφθεί τα αποτελέσματα.Η λειτουργία αυτή δεν διαφέρει από αυτή ενός απλού αριθμητικού κυκλώματος (π.χ. πολλαπλασιαστής) στον οποίο τροφοδοτούμε καινούριες εισόδους ανα ένα κβάντο χρόνου. Αυτή η υλοποίηση εξυπηρετεί εφαρμογές κλιμάκωσης συχνότητας (frequency scaling) εφόσον με το κατάλληλο σύστημα

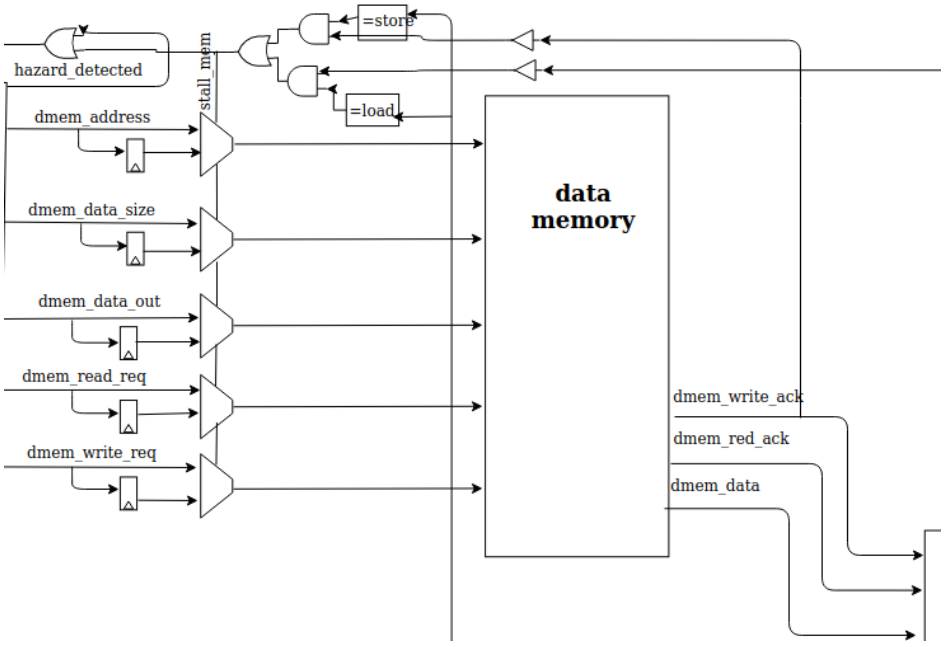
τροποποίησης της συχνότητας είναι δυνατή η δοκιμή διαφορετικών συχνοτήτων οι οποίες μπορούν να αλλάζουν ανα n σετ δεδομένων λαμβάνοντας μεγάλο όγκο αποτελεσμάτων με σημαντική εξοικονόμηση χρόνου.

3.3.4. Διεπαφή μνήμης εντολών και μνήμης δεδομένων με τον πυρήνα

Στις παρακάτω *Εικόνες 15* και *16* φαίνονται τα σήματα εισόδου και εξόδου των δυο εσωτερικών μνημών στην δομική απεικόνιση του επεξεργαστή.



Εικόνα 15 : Μνήμη εντολών

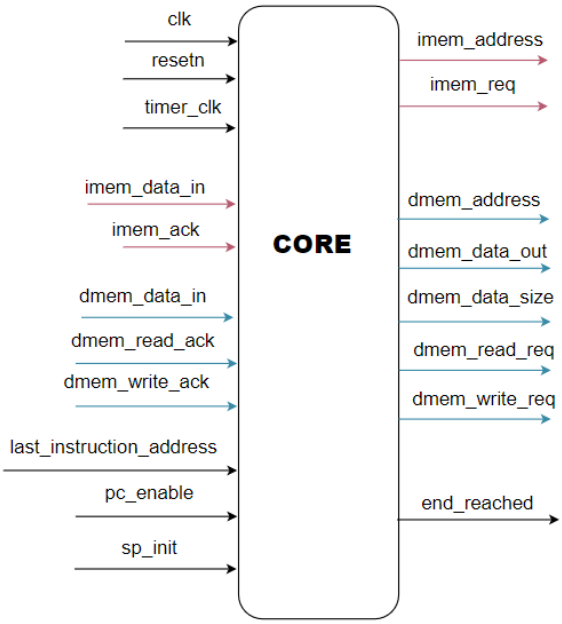


Εικόνα 16 : Μνήμη δεδομένων

Στη μνήμη εντολών υπάρχουν οι είσοδοι imem_addr και imem_req για τη διεύθυνση και την αίτηση για ανάγνωση και οι έξοδοι imem_ack και imem_data για την επιβεβαίωση της αίτησης και τα δεδομένα της εντολής προς εκτέλεση.

Στη μνήμη δεδομένων υπάρχουν οι είσοδοι dmem_address, dmem_data_size, dmem_data_out, dmem_read_req και dmem_write_req που που αναφέρονται στην διεύθυνση της εντολής προς ανάγνωση ή εγγραφή, το μέγεθος του στοιχείου προς εγγραφή ή ανάγνωση (byte, halfword, word) , τα δεδομένα προς εγγραφή, στην αίτηση για ανάγνωση και στην αίτηση για εγγραφή αντίστοιχα.

Στην δομική απεικόνιση του επεξεργαστή παραπάνω δεν αναφέρθηκε ότι το σύστημα του επεξεργαστή αποτελείται από 3 βασικές δομικές μονάδες. Τον πυρήνα , τη μνήμη εντολών και τη μνήμη δεδομένων. Ο πυρήνας περιλαμβάνει όλα τα στάδια της διοχέτευσης εκτός από τις μνήμες επομένως οι είσοδοι και έξοδοι κατά κύριο λόγο αφορούν τις διεπαφές με τις μνήμες. Παρακάτω στην *Εικόνα 17* φαίνεται το δομικό μπλοκ του πυρήνα.



Εικόνα 17: Πυρήνας PMicroX

Με κόκκινο απεικονίζονται τα σήματα που αποτελούν τη διεπαφή με τη μνήμη εντολών και με μπλε τα σήματα της διεπαφής με τη μνήμη δεδομένων. Τα επιπλέον σήματα με μαύρο χρώμα αφορούν λογική ελέγχου της διαδικασίας εισόδου-εξόδου από τον πυρήνα και θα εξηγηθούν παρακάτω.

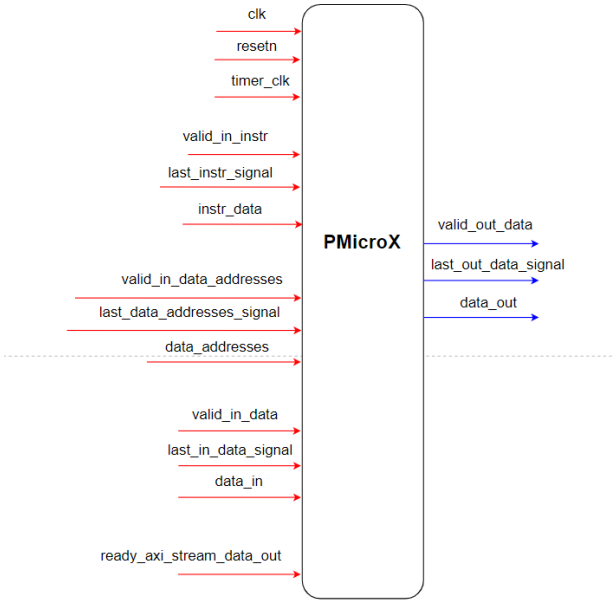
Στην αρχιτεκτονική του PMicroX που προτείνεται τοποθετούνται πολυπλέκτες στις διεπαφές των μνημών με τον πυρήνα. Οι καταστάσεις λειτουργίας που υπάρχουν στο σύστημα είναι 3 : λειτουργία αρχικοποίησης, λειτουργία εκτέλεσης , λειτουργία λήψης αποτελεσμάτων. Προκειμένου να υλοποιηθούν προστέθηκαν στο κορυφαίο επίπεδο ιεραρχίας εκτος από τον

πυρήνα και τις μνήμες, επιπλέον σήματα και λογική ελέγχου που πραγματοποιεί την είσοδο και έξοδο των δεδομένων.

Το σύνολο του πυρήνα PMicroX που πρέκυψε δεσμεύει 3.6% λιγότερους πόρους από το αρχικό σύστημα του Potato μαζί με τις περιφερειακές συσκευές για επικοινωνία με τον χρήστη.Επίσης μειώθηκε κατά περίπου 30% ο χρόνος εκτέλεσης του προγράμματος εφόσον μειώθηκαν οι κύκλοι ρολογιού για να ληφθού αποτελέσματα αλλά παρέμεινε σταθερό το critical path.

3.3.5. Διεπαφή PMicroX-AXI

Παρακάτω φαίνεται το κορυφαίο ιεραρχικά μπλοκ το οποίο περιλαμβάνει εισόδους και εξόδους για τα δεδομένα.Για να γίνουν κατανοητά τα σήματα εισόδου-εξόδου θα γίνει μια αναφορά στο πρωτόκολλο AXI με το οποίο έγινε η μεταφορά των δεδομένων από τη μνήμη DDR του fpga στον πυρήνα. Ο συγχρονισμός για την αποστολή δεδομένων γίνεται μέσω του πυρήνα arm που βρίσκεται στο soc του fpga. Παρότι οι μετρήσεις και τα πειράματα πραγματοποιήθηκαν σε επίπεδο προσομοίωσης , η λειτουργία του συστήματος επιβεβαιώθηκε στο hardware. Συγκεκριμένα χρησιμοποιήθηκε η ultrascale πλακέτα **Zynq UltraScale+ ZCU104 Evaluation Board (xczu7ev-ffvc1156-2-e)**. Μέσω του επεξεργαστή arm που βρίσκεται στο soc γίνεται η αποστολή των αρχείων στη διεπαφή axi η οποία συνδέεται με το σύστημα με τον soft core πυρήνα, στέλνοντας τα απαραίτητα δεδομένα προς εκτέλεση αλλά και λαμβάνοντας τα αποτελέσματα της εκτέλεσης αυτής .Η διεπαφή axi συνδέει δηλαδή τον πυρήνα arm με τον potato που βρίσκεται στο fpga



Εικόνα 18 : Κορυφαίο ιεραρχικά δομικό μπλοκ PMicroX processor

Στο πρωτόκολλο του AXI υπάρχουν οι διεπαφές master και slave. Διαθέτει 5 κανάλια τα οποία συνδέουν αυτές τις δύο πλευρές και διακρίνονται σε κανάλια εγγραφής και κανάλια ανάγνωσης. Με τα πρώτα μεταφέρονται δεδομένα από τον master στον slave και με τα δεύτερα το αντίστροφο [27]. Στο παρόν σύστημα ο arm αποτελεί τον master και ο potato τον slave. Επομένως όταν αποστέλλονται στον PMicroX τα δεδομένα αρχικοποίησης αξιοποιείται κάποιο κανάλι εγγραφής και όταν αποστέλλονται τα αποτελέσματα από τον potato στον arm, κάποιο κανάλι ανάγνωσης. Όλα τα κανάλια τα οποία είναι ανεξάρτητα μεταξύ τους διαθέτουν τα σήματα **VALID** και **READY** τα οποία υλοποιούν ένα αμφίδρομο μηχανισμό επικοινωνίας που αναφέρεται και ως «χειραψία» (handsake). Η πηγή των δεδομένων χρησιμοποιεί το σήμα **VALID** για να ενημερώσει όταν έγκυρα δεδομένα είναι διαθέσιμα. Ο προορισμός χρησιμοποιεί το σήμα **READY** για να ενημερώσει πότε είναι διαθέσιμος να δεχθεί δεδομένα. Επίσης σε όλα τα κανάλια χρησιμοποιείται το σήμα **LAST** για να υποδείξει την μεταφορά του τελευταία δεδομένου της συναλλαγής και διαθέτει τη μορφή παλμού.
Ο μηχανισμός «χειραψίας» επιτρέπει τόσο στον master όσο και στον slave να ελέγχουν τον ρυθμό με τον οποίο κινούνται τα δεδομένα μεταξύ τους. Όταν πραγματοποιείται μια ριπή εγγραφής, ο master ενώ έχει ενεργό το σήμα **WVALID** που σημαίνει ότι τα δεδομένα του είναι έγκυρα πρέπει να περιμένει να ενεργοποιηθεί το σήμα **WREADY** του slave ώστε να πραγματοποιηθεί ο χαιρετισμός.Το σήμα **WREADY** στο σύστημα του potato είναι μονίμως ενεργό και επομένως δεν φαίνεται ως είσοδος διότι ο potato μπορεί μονίμως να δεχθεί δεδομένα σε 1 κύκλο ρολογιού. Η ίδια διαδικασία ακολουθείται και όταν πραγματοποιείται ριπή ανάγνωσης. Όταν ο slave ενεργοποιεί το σήμα **ARVALID** που σημαίνει ότι στέλνει έγκυρα δεδομένα πρέπει να το διατηρήσει ενεργό μέχρι να ενεργοποιηθεί και το σήμα **ARREADY** που σηματοδοτεί πότε ο master είναι διαθέσιμος να λάβει δεδομένα. Στο σύστημα του potato ο master που είναι ο arm πιθανώς να μην είναι μόνιμα διαθέσιμος επομένως απαιτείται μια είσοδος που θα ενημερώνει τον potato πότε είναι διαθέσιμος για αποστολή δεδομένων(σήμα εισόδου **axi_stream_data_out**)

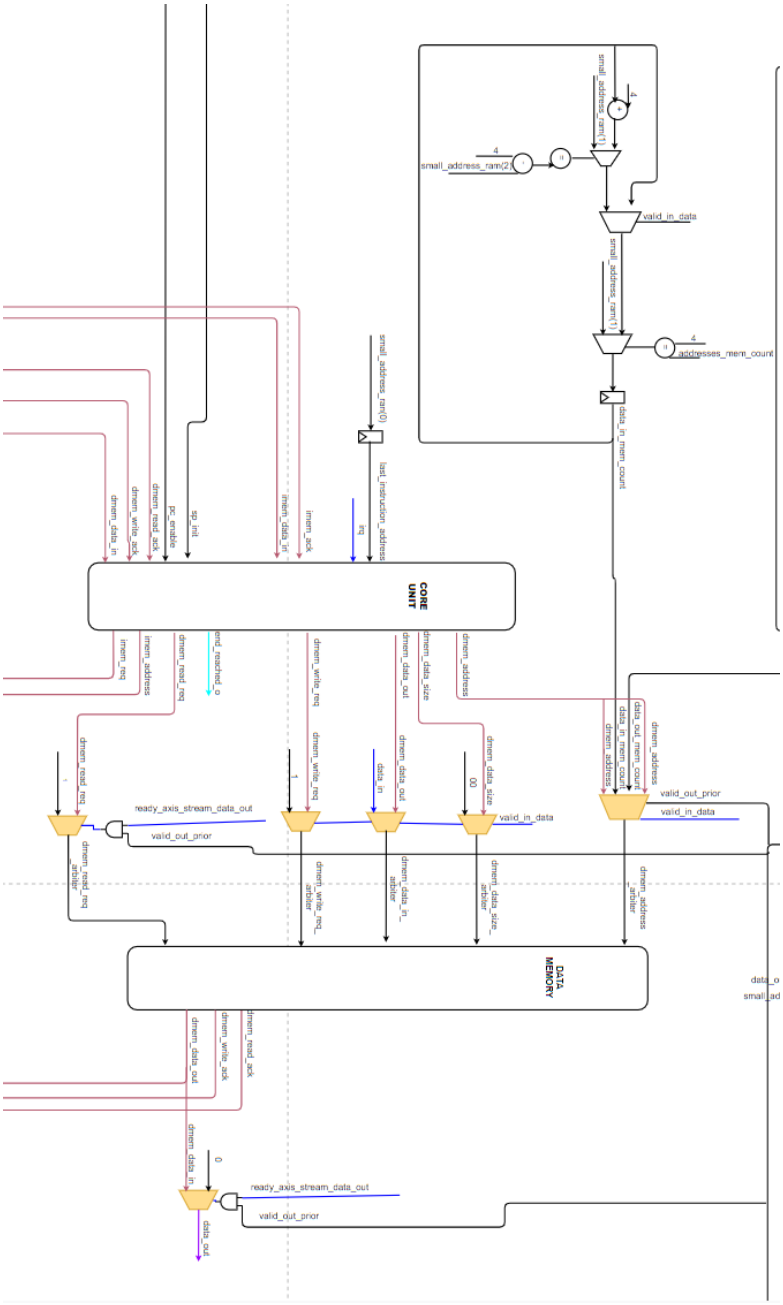
Στην εικόνα παραπάνω το σήμα valid_in_instruction , αφορά την έγκυρη λήψη του σετ εντολών, το σήμα last_instr_signal την τελευταία εντολή προς λήψη και το σήμα instr_data την είσοδο για τις εντολές οι οποίες αποθηκεύονται στη μνήμη εντολών (imem). Οι επόμενη τριάδα σημάτων εισόδου ακολουθεί την ίδια λογική για τα δεδομένα τα οποία αποθηκεύονται στη μνήμη δεδομένων. Η τελευταία τριάδα αφορά τις διευθύνσεις μνήμης στις οποίες θα καταχωρηθούν τα δεδομένα που ακολουθούν. Αυτό το σετ αποτελείται μόνο από 5 τιμές που όπως θα εξηγηθεί και παρακάτω υποδεικνύουν τις περιοχές διευθύνσεων των δεδομένων εισόδου και εξόδου στη στοίβα. Τέλος η τριάδα σημάτων εξόδου διαθέτει πανομοιότυπη λογική και χρησιμοποιείται για την αποστολή των αποτελεσμάτων στον arm.

Τα παραπάνω αφορούν τη διεπαφή του PMicroX με το εξωτερικό πυρήνα arm και επεξηγούν το είδος και τη λειτουργία των σημάτων εισόδου και εξόδου, απεικονίζοντας τον potato ως ένα κλειστό μπλοκ. Στη συνέχεια

θα πραγματοποιηθεί ανάλυση του εσωτερικού συστήματος το οποίο αρχικά περιλάμβανε μόνο τον core και τις 2 μνήμες. Επειδή η λογική είναι αρκετά εκτενής και πολύπλοκη θα καταμεριστεί σε επιμέρους τμήματα.

3.3.6. Τροποποιημένη διεπαφή core – data memory – instruction memory

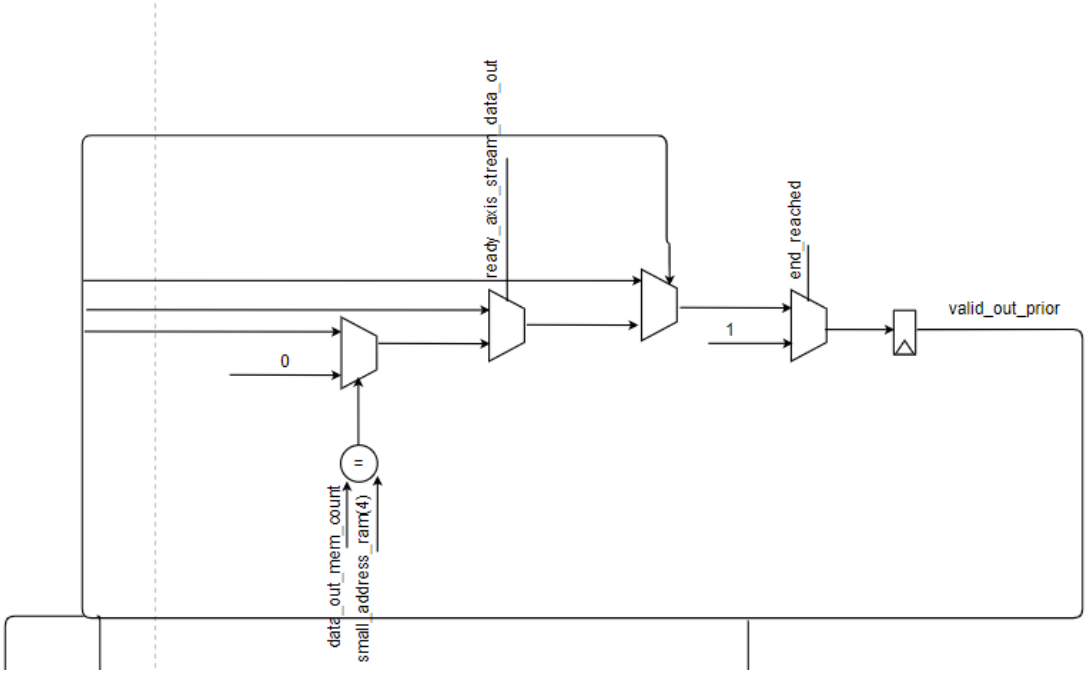
Ακολούθως φαίνεται το τμήμα της διεπαφής του core με την μνήμη δεδομένων και απεικονίζει την επιπλέον λογική που προστέθηκε σε σχέση με την *Εικόνα 16*.



Εικόνα 19 : Τροποποιημένη διεπαφή πυρήνα-μνήμης δεδομένων

Με κόκκινο χρώμα επισημαίνονται τα σήματα που συνδέουν τον πυρήνα με τη μνήμη και προϋπήρχαν στην αρχική υλοποίηση. Με μπλε χρώμα επισημαίνονται τα σήματα εισόδων και με μωβ τα σήματα εξόδων. Όπως διαπιστώνεται κύρια αλλαγή αποτελεί η προσθήκη πολυπλεκτών-διαιτητών (arbiters) ο οποίοι οδηγούν τα κατάλληλα σήματα ανάλογα με το στάδιο λειτουργίας. Έχει ήδη αναφερθεί πως αυτά είναι 3 : αρχικοποίηση, εκτέλεση, λήψη αποτελεσμάτων. Κατά την αρχικοποίηση οι πολυπλέκτες οδηγούν σήματα που επιτρέπουν εγγραφή δεδομένων από τις θύρες εισόδου στη μνήμη δεδομένων σε συγκεκριμένες θέσεις. Κατά την εκτέλεση οι πολυπλέκτες οδηγούν τα προυπάρχοντα σήματα ανάγνωσης και εγγραφής. Τέλος κατά την λήψη αποτελεσμάτων ο πολυπλέκτης εξόδου επιτρέπει την ανάγνωση δεδομένων από τη μνήμη δεδομένων στη θύρα εξόδου.

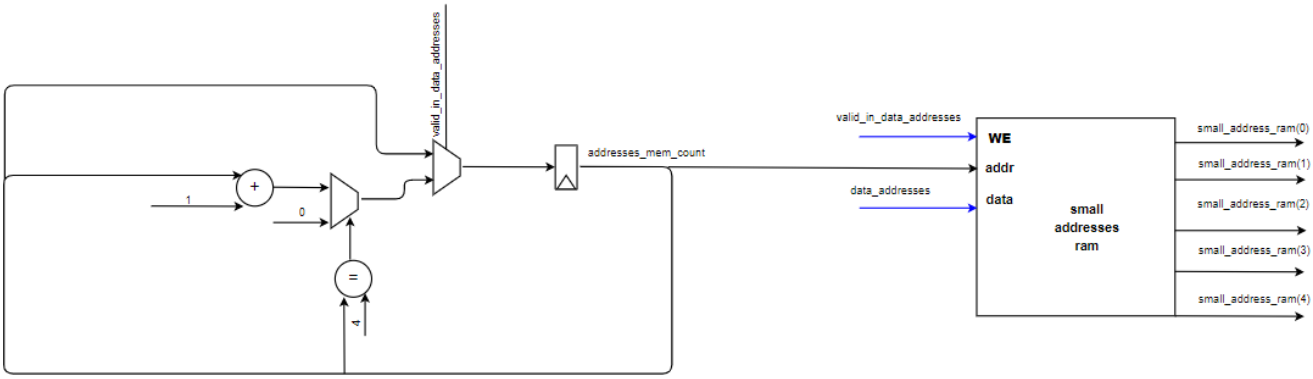
Ο πρώτος πολυπλέκτης στην είσοδο της μνήμης δεδομένων είναι «διαιτητής» διευθύνσεων. Τα σήματα επιλογής είναι τα **valid_data_in** και **valid_out_prior.** Το πρώτο είναι εξωτερικό σήμα εισόδου και είναι ενεργό μόνο κατά την αποστολή των δεδομένων αρχικοποίησης μνήμης δεδομένων(λειτουργία αρχικοποίησης) και το δεύτερο είναι εσωτερικό σήμα το οποίο ενεργοποιείται μετά τη λήξη της εκτέλεσης και σηματοδοτεί τη μετάβαση στη λειτουργία λήψης αποτελεσμάτων (λειτουργία λήψης αποτελεσμάτων).



Εικόνα 20 : Λογική σήματος valid_out_prior

Συγκεκριμένα για να ενεργοποιηθεί πρέπει να ενεργοποιηθεί το σήμα-παλμός **end_reached** το οποίο όπως θα αναλυθεί και παρακάτω σηματοδοτεί το τέλος της εκτέλεσης του προγράμματος. Αυτό επιτυγχάνεται με τη σύγκριση στο τέλος της διοχέτευσης της διεύθυνσης της κάθε εντολής που εκτελέστηκε με την διεύθυνση της τελευταίας εντολής στη μνήμη εντολών

(instruction memory) . Η διεύθυνση της τελευταίας εντολής (που σε ορισμένους κώδικες δεν ταυτίζεται με την εντολή στην μεγαλύτερη διεύθυνση της μνήμης) διαφέρει ανάλογα τον κώδικα της εφαρμογής. Επομένως τροφοδοτείται ως είσοδος στον core σε κάθε κύκλο (σήμα εισόδου **last_instruction_address** του core) ώστε να πραγματοποιείται η σύγκριση. Το σήμα **valid_out_prior** μηδενίζεται όταν η τιμή του σήματος **data_out_mem_count** που χρησιμοποιείται ως διεύθυνση ανάγνωσης δεδομένων εξόδου εξισωθεί με την τιμή της τελευταίας θέσης μνήμης του τμήματος εξόδου. Για να γίνει περισσότερο κατανοητό απαιτείται μια αναφορά στη χρησιμότητα της μικρής μνήμη ram που αποθηκεύει διευθύνσεις. Όπως είχε αναφερθεί παραπάνω η μνήμη αυτή διαθέτει 5 καταχωρήσεις.Αρχικοποιείται μέσω των εισόδων data_addresses αμέσως μετά την αρχικοποίηση της μνήμης εντολών. Οι καταχωρήσεις είναι κατά σειρά : 0 -> διεύθυνση τελευταίας εντολής, 1 -> διεύθυνση πρώτης θέσης μνήμης του μπλοκ δεδομένων εισόδου, 2 -> διεύθυνση τελευταίας θέσης μνήμης του μπλοκ δεδομένων εισόδου, 3 -> διεύθυνση πρώτης θέσης μνήμης του μπλοκ δεδομένων εξόδου, 4 -> διεύθυνση τελευταίας θέση μνήμης του μπλοκ δεδομένων εξόδου. Ως μπλοκ δεδομένων εισόδου/εξόδου αναφέρεται ένας πίνακας ή μια ακολουθία πινάκων εισόδου/εξόδου.Για παράδειγμα αν η εφαρμογή παίρνει ως είσοδο έναν πίνακα input[n] και δίνει ως έξοδο έναν πίνακα output[m] τότε η καταχώρηση 1 περιλαμβάνει την πρώτη διεύθυνση του πίνακα input, η καταχώρηση 2 την τελευταία διεύθυνση του πίνακα input, η καταχώρηση 3 την πρώτη διεύθυνση του πίνακα output και η καταχώρηση 4 την τελευταία διεύθυνση.Αυτές οι 5 διευθύνσεις παρέχονται από τον χρήστη και χρησιμοποιούνται στον προσδιορισμό των περιοχών και συνεπώς στον καθορισμό των σταδίων λειτουργίας. Επι της ουσίας η μικρή αυτή ram αποτελείται από 5 καταχωρητές αλλά απεικονίζεται ως μπλοκ για απλούστευση.



Εικόνα 21 : Λογική σήματος διεύθυνσης μνήμη ram διευθύνσεων

Η έξοδος των καταχωρητών **small_address_ram(0 – 4)** τροφοδοτεί ορισμένα από τα κυκλώματα ελέγχου παρέχοντας τις πληροφορίες για τα όρια των τμημάτων εισόδου και εξόδου.Ως είσοδο το αρχείο καταχωρητών

δέχεται το σήμα **valid_in_data_addresses** το οποίο είναι ενεργό για όσο διάστημα αποστέλλονται έγκυρες διευθύνσεις (συνολικά απαιτεί 5 κύκλους) και συνδέεται με το write enable των καταχωτρητών. Το σήμα που καθορίζει τις διευθύνσεις για την πρόσβαση στο αρχείο καταχωρητών είναι το **addresses_mem_cnt** το οποίο όσο το σήμα **valid_in_data_addresses** είναι ενεργό, αυξάνεται κατά ένα για να γίνει εγγραφή στο επόμενο στοιχείο της μνήμης διευθύνσεων. Μόλις αυτή η τιμή φτάσει το 4 δηλαδή εγγραφεί και το τελευταίο στοιχείο, μηδενίζεται για επόμενη εκτέλεση.(Δηλαδή μέχρι να ενεργοποιηθεί πάλι το σήμα **valid_in_data_addresses**). Τέλος είναι προφανές πως η είσοδος **data_addresses** παρέχει τις τιμές των διευθύνσεων προς εγγραφή στη μνήμη.

Επιστρέφοντας στην ανάλυση του σήματος **valid_out_prior** για να μηδενιστεί πρέπει η τιμή **data_out_mem_count** που χρησιμοποιείται ως διεύθυνση για την ανάγνωση του τμήματος εξόδου να εξισωθεί με την τιμή **small_address_ram(4)**.

Αφού αναλύθηκαν οι επιλογείς του διαιτητή διευθύνσεων μνήμης δεδομένων θα επεξηγηθούν οι είσοδοι. Με κόκκινο χρώμα φαίνεται το σήμα **dmem_addresses** το οποίο θα οδηγεί το σήμα **dmem_address_arbiter** όταν έχουμε λειτουργία εκτέλεσης (δηλαδή όταν τα σήματα **valid_out_prior** και **valid_in_data** είναι αφμότερα 0. Η περίπτωση να είναι αμφότερα 1 δεν υφίσταται παρόλα αυτά περιλαμβάνεται σαν επιλογή στο hardware).

Τα άλλα δύο σήματα εισόδου είναι το **data_in_mem_count** και το **data_out_mem_count**. Το σήμα **data_out_mem_count** λειτουργεί ως εξής : όταν ενεργοποιηθεί το σήμα παλμός end_reached τότε αρχικοποιείται με την τιμή **small_address_ram(3)** που περιλαμβάνει την αρχή του τμήματος εξόδου. Έπειτα εφόσον το σήμα valid_out_prior είναι ενεργό δηλαδή βρισκόμαστε σε λειτουργία λήψης δεδομένων και εφόσον το σήμα **ready_axi_stream_data** είναι επίσης ενεργό που σημαίνει ότι ο master μπορεί να δεχθεί δεδομένα τότε αυξάνεται κατα 4 (ο πίνακας εξόδου διαβάζεται ανα λέξη). Όταν το σήμα **valid_out_prior** μηδενιστεί που σημαίνει ότι διαβάστηκε όλο το τμήμα εξόδου το σήμα **data_out_mem_count** θα διατηρήσει την τιμή που βρισκόταν μέχρι να έρθει πάλι ένα σήμα παλμός **end_reached** και να αρχικοποιηθεί στη νέα αρχική διεύθυνση του τμήματος εξόδου.

Το σήμα data_in_mem_count λειτουργεί ως εξής : αρχικοποιείται στην τιμή **small_address_ram(1)** όταν η τιμή του σήματος **addresses_mem_cnt** που όπως αναφέρθηκε χρησιμοποιείται ως διεύθυνση του αρχείου καταχωρητών γίνει ίση με 4, δηλαδή όταν έχει ολοκληρωθεί η αρχικοποίηση του αρχείο καταχωρητών.Η λειτουργία αυτή στηρίζεται στην αλληλουχία με την οποία πρέπει να δωθούν τα δεδομένα στον potato. Πρώτα αποστέλλονται οι εντολές , μετά οι διευθύνσεις και τέλος τα δεδομένα .Άρα στο τέλος της αποστολής των διευθύνσεων γίνεται προετοιμασία υποδοχής των δεδομένων.Όταν ενεργοποιηθεί το σήμα valid_in_data η τιμή της διεύθυνσης **data_in_mem_count** ισούται με τη διεύθυνση αρχής του τμήματος εισόδου. Για όσους κύκλους παραμένει ενεργό το **valid_in_data** ,

το **data_in_mem_count** αυξάνεται κατά 4 μεταβαίνοντας στη επόμενη θέση προς αρχικοποίηση. Η τιμή αυτή θα σταθεροποιηθεί στην τελευταία διεύθυνση του τμήματος εισόδου και θα αρχικοποιηθεί πάλι σε νέα τιμή όταν το **addresses_mem_cnt** γίνει πάλι 4, σηματοδοτώντας την έλευση νέων διευθύνσεων ενός άλλου κώδικα.

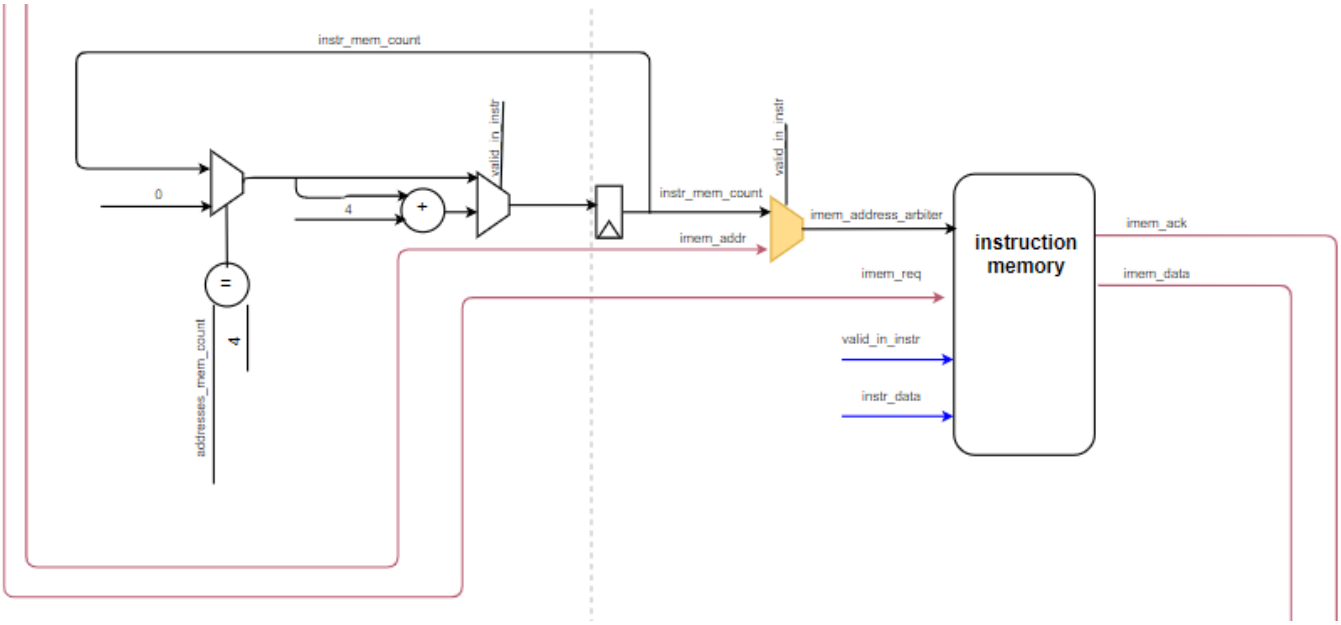
Επόμενος πολυπλέκτης-διαιτητής είναι ο dmem_size_arbiter ο οποίος καθορίζει τον αριθμό των bit που θα εγγραφούν ή θα φορτωθούν από τη μνήμη. Κατά την αρχικοποίηση ο αριθμός αυτός πρέπει να διατηρείται σταθερός στην τιμή 00 που σημαίνει ότι γίνεται εγγραφή λέξεων(32 bit) στην μνήμη. Το ίδιο και κατά την ανάγνωση. Άρα για την λειτουργία εκτέλεσης ο διαιτητής λαμβάνει την τιμή του σήματος διασύνδεσης του core με την μνήμη , εξυπηρετώντας οποιοδήποτε μέγεθος υποδεικνύει ο κώδικας, ενώ για τις άλλες δύο περιπτώσεις λειτουργίας η τιμή αυτή διατηρείται σταθερή στην τιμή 00.

Παρακάτω στην *Εικόνα 19* φαίνεται ο πολυπλέκτης data_in_arbiter ο οποίος καθορίζει τα δεδομένα προς εγγραφή στη μνήμη δεδομένων. Όταν το σήμα valid_in_data είναι ενεργό δηλαδή αρχικοποιείται η μνήμη τότε περνά στον πολυπλέκτη η είσοδος data_in ενώ όταν είναι μηδέν τότε μεταβιβάζεται το εσωτερικό σήμα διασύνδεσης το οποίο χρησιμοποιείται κατά την εκτέλεση του προγράμματος και μεταφέρει τα δεδομένα προς εγγραφή στη μνήμη (προερχόμενα από τον core).

Οι δύο τελευταίοι πολυπλέκτες στην είσοδο δεδομένων καθορίζουν τις εισόδους της μνήμης δεδομένων για τις αιτήσεις ανάγνωσης και εγγραφής. Το σήμα dmem_write_req_arbiter καθορίζει την τιμή της εισόδου για την αίτηση εγγραφής. Όταν το σήμα valid_in_data είναι ενεργό το σήμα εγγραφής που οδηγείται είναι σταθερά 1 εφόσον τα δεδομένα από την εξωτερική θύρα εισόδου πρέπει να εγγραφούν στη μνήμη και όταν το σήμα είναι 0 τότε θα έχω κανονική λειτουργία (ή λειτουργία λήψης δεδομένων) οπότε στον διαιτητή οδηγείται το εσωτερικό σήμα που διασυνδέει τον core με τη μνήμη και χρησιμοποιείται στις εγγραφές στην λειτουργία εκτέλεσης.

Το σήμα που οδηγεί την είσοδο για τις αιτήσεις ανάγνωσης διατηρεί σταθερά την τιμή 1 για όσο διάστημα αμφότερα τα σήματα ready_axi_stream_data_out και valid_out_prior είναι ενεργά. Το σήμα valid_out_prior σηματοδοτεί την έναρξη αποστολής έγκυρων αποτελεσμάτων στην έξοδο και το σήμα ready_axi_stream_data_out ενημερώνει για τη διαθεσιμότητα του master. Όταν λοιπόν είναι ενεργοποιημένα πραγματοποιούνται συνεχείς αναγνώσεις από τη μνήμη δεδομένων μέχρι να αποσταλεί στον master το σύνολο των αποτελεσμάτων της εκτέλεσης. Για αυτό το σήμα αυτό διατηρείται στο 1. Σε κάθε άλλη περίπτωση δηλαδή κατά την αρχικοποίηση και την κύρια λειτουργία εκτέλεσης στον πολυπλέκτη οδηγείται το εσωτερικό σήμα διασύνδεσης που εξυπηρετεί τις αιτήσεις για ανάγνωση όταν εκτελείται η εφαρμογή.

Παρατηρώντας τις εισόδους του core φαίνονται τα εξής σήματα. Η είσοδος last_instruction_address τροφοδοδοτείται από έναν καταχωρητή ο οποίος έχει αποθηκεύσει την τιμή small_address_ram(0) η οποία είναι η τιμή της τελευταίας εντολής του κώδικα προκειμένου να σηματοδοτήσει το τέλος της εκτέλεσης. Η εξωτερική είσοδος irq αφορά τις διακοπές (interrupts). Οι είσοδοι imem_ack και imem_data_in οδηγούνται από σήματα εξόδου της μνήμης εντολών και αφορούν την επιβεβαίωση ανάγνωσης μιας αίτησης από τη μνήμη εντολών και τα δεδομένα της εντολής αντίστοιχα.



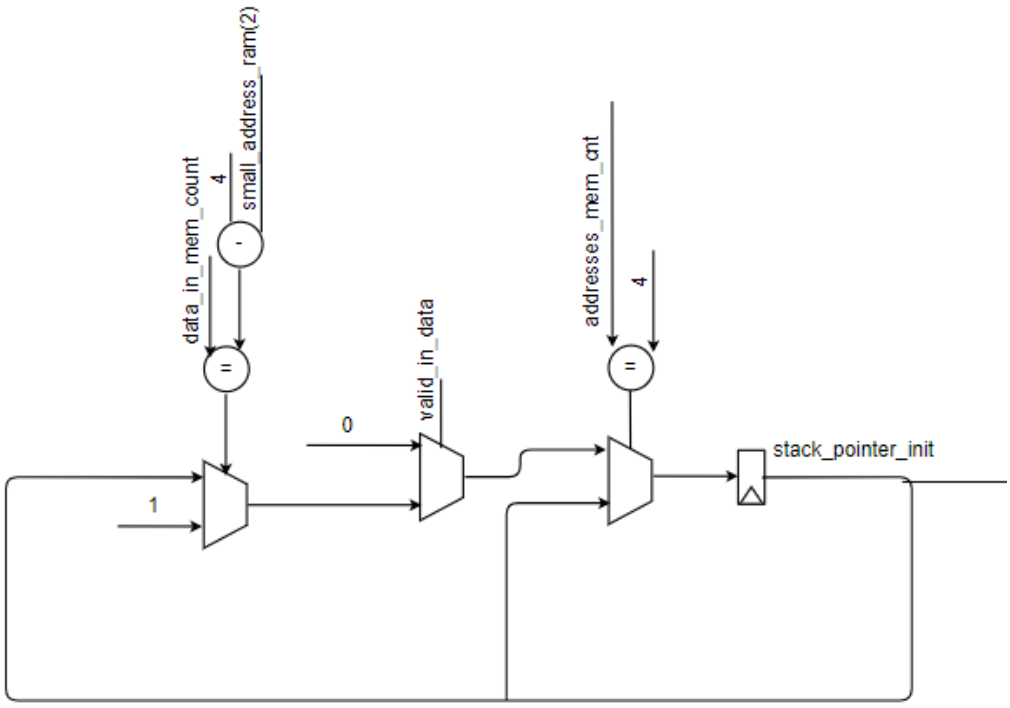
Εικόνα 22 : Τροποποιημένο σύστημα μνήμης εντολών και λογική σήματος imem_address_arbiter

Εδώ φαίνεται σχηματικά από που προέρχονται τα σήματα αυτά. Στην είσοδο της μνήμης εντολών υπάρχει ένα σήμα διαιτητής για τις διευθύνσεις των προσβάσεων. Στη μνήμη εντολών κατά την κανονική λειτουργία πραγματοποιούνται μόνο αναγνώσεις. Κατά την αρχικοποίηση της μνήμης

πραγματοποιούνται μόνο εγγραφές. Έτσι προστέθηκε στη μνήμη επιπλέον λογική ώστε να λειτουργεί ως μνήμη ram. Το επιπλέον στοιχείο που προστέθηκε στη λειτουργία της μνήμης είναι δυνατότητα εγγραφής.Το εξωτερικό σήμα εισόδου valid_in_instr οδηγεί τη θύρα επίτρεψης εγγραφής ώστε όταν λαμβάνονται έγκυρα δεδομένα εντολών να εγγράφονται απευθείας στη μνήμη και επίσης το εξωτερικό σήμα instr_data οδηγεί τη θύρα για τα δεδομένα εγγραφής. Το σήμα για την αίτηση ανάγνωσης δεν απαιτεί

πολυπλέκτη-διαιτητή εφόσον χρησιμοποιείται μόνο κατά τη λειτουργία εκτέλεσης επομένως συνδέεται απευθείας με την έξοδο του core που παράγει τις αιτήσεις για ανάγνωση από τη μνήμη εντολών. Το μοναδικό σήμα εισόδου που απαιτεί πολυπλέκτη είναι αυτό της διεύθυνσης. Όταν το σήμα valid_in_instr είναι 0 τότε διασυνδέεται με το εσωτερικό σήμα διευθύνσεων της μνήμης εντολών που προέρχεται από τον core και χρησιμοποιείται στη λειτουργία εκτέλεσης.Όταν όμως πραγματοποιείται αρχικοποίηση της μνήμης εντολών το σήμα αυτό είναι ενεργό, οπότε οδηγείται στο imem_address_arbiter το σήμα instr_mem_count. Αυτό διαθέτει παρόμοια λειτουργία με τα σήματα data_in_mem_cnt και data_out_mem_cnt που αναλύθηκαν παραπάνω. Κατά την πρώτη αρχικοποίηση το σήμα έχει αρχική τιμή 0 λόγω του reset. Όταν ενεργοποιηθεί το σήμα valid_in_instr και για όσους κύκλους είναι ενεργό αυξάνεται κατά 4 (γίνεται πρόσβαση ανα λέξη στη μνήμη εντολών). Όταν το valid_in_instr απενεργοποιηθεί το σήμα διατηρεί την τελευταία τιμή του και μηδενίζεται για επόμενη αποστολή δεδομένων όταν ολοκληρωθεί η αποστολή των διευθύνσεων. Αυτός ο έλεγχος πραγματοποιείται με σύγκριση του σήματος addresses_mem_count για τις διευθύνσεις στη μικρή μνήμη διευθύνσεων με την τιμή 4 (δηλαδή το τέλος της αρχικοποίησης της μικρής ram διευθύνσεων). Η επιλογή αυτή έγινε διότι ήταν από τα σήματα με λίγες εξαρτήσεις και δεν προέκυπτε κάποιο σφάλμα κατά την μεταγλώττιση.

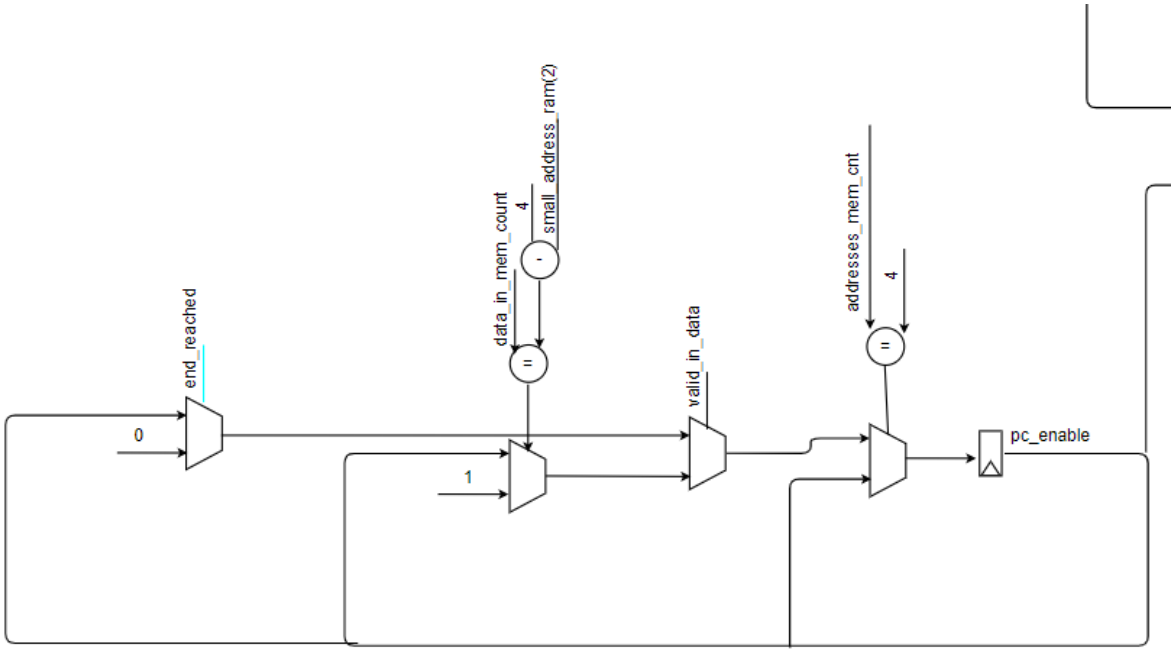
Επιστρέφοντας στην ανάλυση των εισόδων του core θα επεξηγηθούν τα σήματα sp_init και pc_enable. Το πρώτο αποτελεί ένα σήμα παλμό ο οποίος δίνεται πριν την εκκίνηση ώστε να αρχικοποιηθεί η τιμή του καταχωρητή δείκτη της στοίβας (sp : stack pointer , είναι ο 3ος κατά σειρά στο αρχείο καταχωρητών και αρχικοποιείται με την μεγαλύτερη τιμή της μνήμης.Αν για παράδειγμα η μνήμη έχει μέγεθος 16kB τότε πρέπει να πάρει την τιμή 16384). Στην *Εικόνα 23* φαίνεται η λειτουργία του σήματος *sp_init*. Το σήμα ενεργοποιείται κατα τον προτελευταίο κύκλο των έγκυρων δεδομένων της μνήμης δεδομένων. Έτσι όταν το σήμα valid_in_data είναι ενεργό και το σήμα data_in_mem_cnt για τη διευθυνσιοδότηση κατα την αρχικοποίηση είναι ίσο με τη τιμή του καταχωρητή small_address_ram(2) - 4, δηλαδή βρίσκεται στην προτελευταία λέξη της αρχικοποίησης,τότε ενεργοποιείται το σήμα stack_pointer_init.



Εικόνα 23 : Λογική σήματος stack_pointer_init

Στον επόμενο κύκλο θα εγγραφεί και το τελευταίο δεδομένο και το valid_in_data θα μηδενιστεί. Μαζί με αυτό μηδενίζεται και το stack_pointer_init το οποίο έχει αρχικοποιήσει το stack pointer ώστε να ξεκινήσει η εκτέλεση του κώδικα. Η αλλαγή που πραγματοποιήθηκε σχετικά με την αρχική έκδοση του potato είναι η προσθήκη εισόδου στο αρχείο καταχωρητών για το σήμα stack_pointer_init και επιπλέον λογικής στο reset ώστε να μηδενίζει όλες τις τιμές εκτός από αυτή του δείκτη στοίβας η οποία τίθεται σε μια συγκεκριμένη τιμή. Αυτή η τιμή είναι παραμετροποιημένη και τοποθετείται αυτόματα με την επιλογή μεγέθους μνήμης στο κορυφαίο ιεραρχικά αρχείο (PMicroX) με χρήση της δυνατότητας generic της vhdl. Η προσθήκη αυτή στο αρχείο καταχωρητών πρέπει να χρησιμοποιηθεί μόνο κατα τη δεύτερη μέθοδο εξαγωγής αρχείων αρχικοποίησης κατα την οποία δίνεται μόνο το object αρχείο που περιλαμβάνει τις εντολές και το αρχείο με τα δεδομένα. Αν χρησιμοποιηθεί η πρώτη μέθοδος κατα την οποία πραγματοποιείται το linking, στον κυρίως κώδικα του προγράμματος προστίθεται και κώδικας αρχικοποίησης (crt0.S ο οποίος περιλαμβάνει αρχικοποίηση δείκτη στοίβας), συνεπώς πρέπει να αφαιρεθεί η λογική που περιγράφηκε από το αρχείο καταχωρητών.

Το σήμα pc_enable οριοθετεί την λειτουργία εκτέλεσης του κώδικα. Οδηγείται στον core και συγκεκριμένα στον στάδιο instruction fetch όπου έχει προστεθεί επιπλέον λογική για να επιτρέπει την εκκίνηση εκτέλεσης μόνο όταν το σήμα pc_enable είναι ενεργό. Σε κάθε άλλη περίπτωση αναστέλλει τη διοχέτευση διακόπτοντας τη φόρτωση εντολών από την μνήμη εντολών.



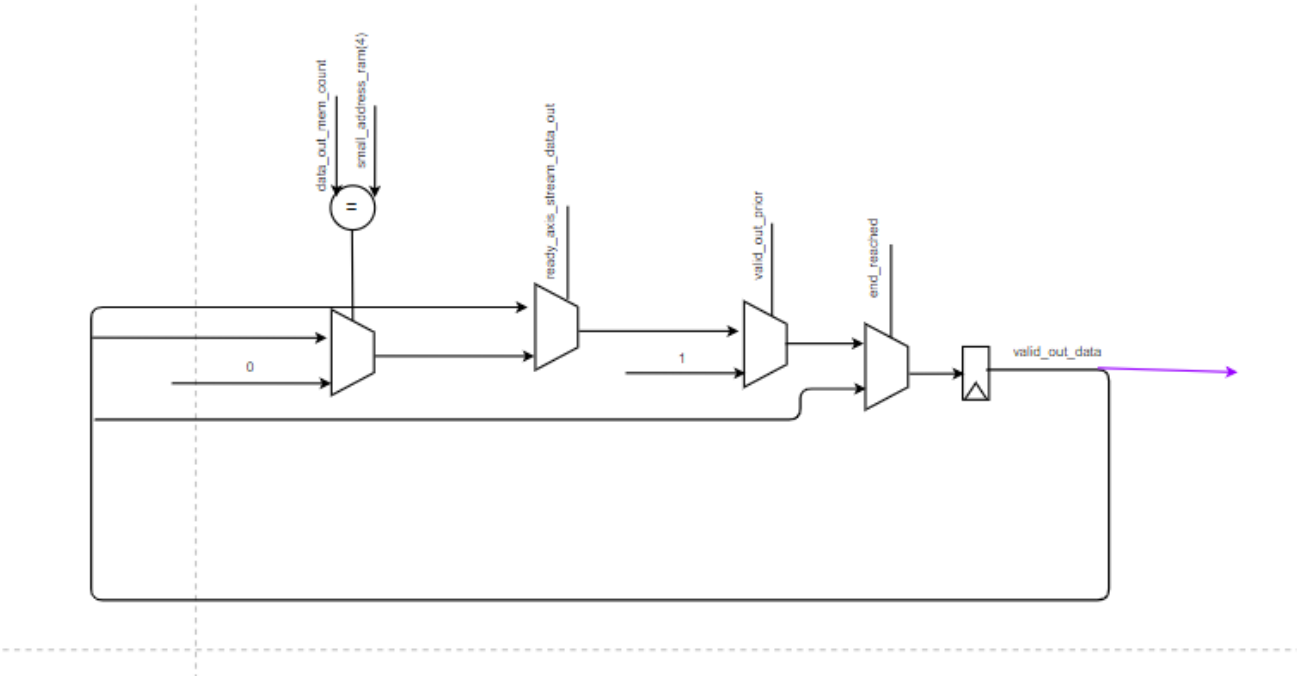
Εικόνα 24 : Λογική σήματος pc_enable

Ακολουθεί την ίδια λογική ενεργοποίησης με το stack_pointer_init δηλαδή ενεργοποιείται μετά τον τελευταίο κύκλο αποστολής έγκυρων δεδομένων και απενεργοποιείται όταν ενεργοποιηθεί το σήμα παλμός end_reached. Ο χρόνος που είναι ενεργό το σήμα είναι το διάστημα μεταξύ φόρτωσης της πρώτης εντολής από τη μνήμη εντολών στον επεξεργαστή εως και την εκτέλεση (στάδιο memory) της τελευταίας εντολής.

Τέλος όσον αφορά τις εισόδους του core τα 3 τελευταία σήματα αφορούν εσωτερικά σήματα που εξέρχονται από τη μνήμη δεδομένων και εισέρχονται στον core και είναι αντίστοιχα επιβεβαίωση αίτησης ανάγνωσης, επιβεβαίωση αίτησης εγγραφής και δεδομένα ανάγνωσης. Τα τελευταία εισέρχονται εκτός από τον core και σε πολυπλέκτη διαιτητή εξόδου ο οποίος τα οδηγεί στο θύρα εξόδου data_out όταν αμφότερα τα σήματα ready_axi_stream_data_out και valid_out_prior είναι ενεργά. Σε κάθε άλλη περίπτωση οδηγείται μηδενική τιμή στην έξοδο δεδομένων.

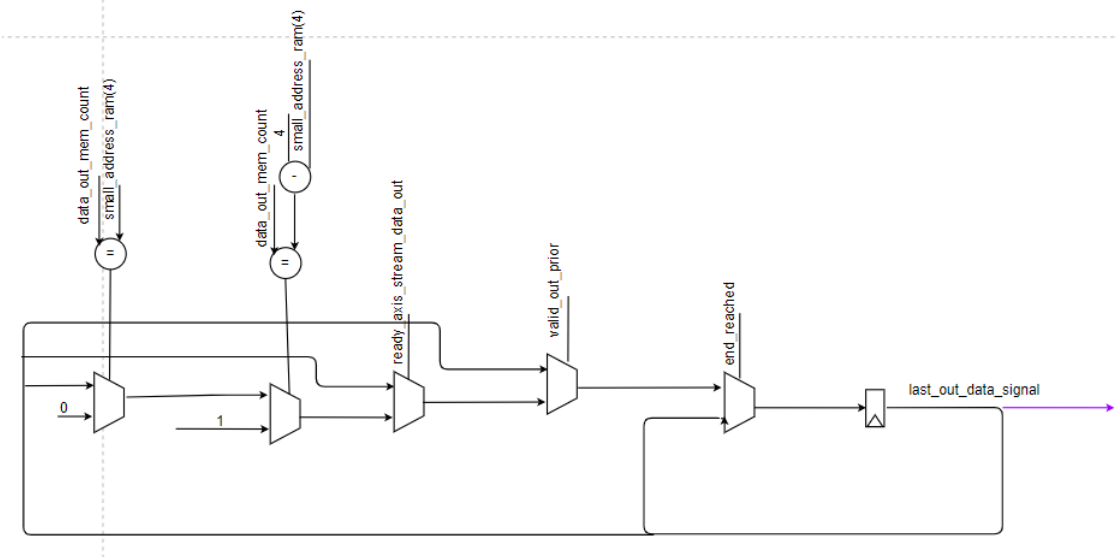
Η ανάλυση του συστήματος ολοκληρώνεται με τα σήματα εξόδου valid_out_data και last_out_data_signal. Το πρώτο ενεργοποιείται ένα κύκλο μετά από το σήμα valid_out_prior και απενεργοποιείται εφόσον το σήμα ready_axi_stream_data_out είναι ενεργό (δηλαδή ο master μπορεί να δεχθεί δεδομένα) και έχει γίνει ανάγνωση και του τελευταίου δεδομένου

εξόδου (σύγκριση του σήματος data_out_mem_cnt που πραγματοποιεί τις διαδοχικές αναγνώσεις με το σήμα small_address_ram(4) που διαθέτει την τελευταία διεύθυνση του τμήματος εξόδου).



Εικόνα 25 : Λογική σήματος valid_out_data

Το δεύτερο ενεργοποιείται κατά τον τελευταίο κύκλο ανάγνωσης δεδομένων εξόδου και απενεργοποιείται ταυτόχρονα με το σήμα valid_out_prior και το valid_out_data. Εξυπηρετεί την ίδια λογική με τα σήματα εισόδου last_intsrt_signal, last_data_addresses_signal και last_in_data_signal για να σηματοδοτήσει το τέλος αποστολής ενός σετ δεδομένων στον master.



Εικόνα 26 : Λογική σήματος last_out_data_signal

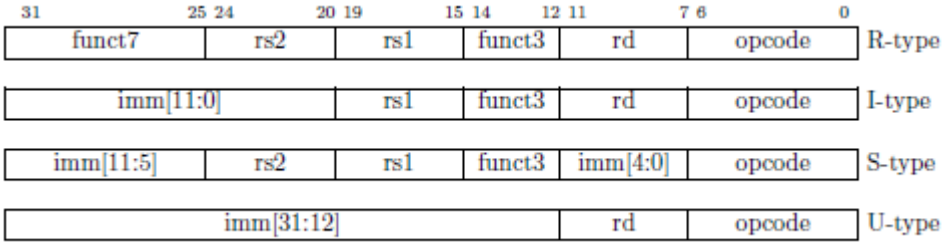
Κεφάλαιο 4

Επέκταση PMicroX RISC-V soft-processor για προσεγγιστικές εντολές

Αρχικά γίνεται αναφορά στα είδη των εντολών που διαθέτει το σύνολο εντολών του RISC-V και ακολουθεί η μεθοδολογία προσθήκης των νέων προσεγγιστικών εντολών στο σετ εντολών καθώς και η προσθήκη υλικού που απαιτείται για την υποστήριξη τους.

4.1 Είδη και δομή των εντολών

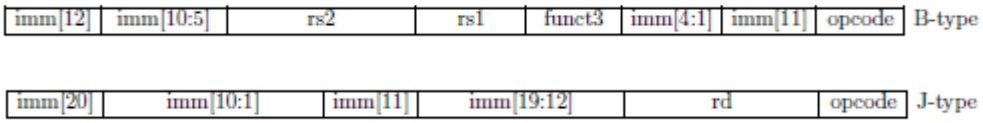
Στο θεμελιώδες σύνολο εντολών υπάρχουν τέσσερις βασικές μορφές εντολών όπως φαίνεται στο παρακάτω σχήμα [17].



Εικόνα 27 : Τύποι εντολών RISC-V

Διαθέτουν σταθερό μήκος 32 bit σε μήκος και πρέπει να είναι ευθυγραμμισμένες κατά 4 byte στη μνήμη. Στο σχήμα φαίνονται τα ονόματα των τμημάτων κάθε μορφής και τα bit τα οποία καταλαμβάνουν. Παρατηρείται πως η διεύθυνση των καταχωρητών πηγής και προορισμού βρίσκεται στην ίδια θέση σε όλες τις μορφές ώστε να απλοποιηθεί η αποκωδικοποίηση της εντολής. Εκτός από από τις άμεσες τιμές των 5 bit οι οποίες χρησιμοποιούνται από τις εντολές ελέγχου κατάστασης (CSR instructions), όλες οι υπόλοιπες άμεσες τιμές διαθέτουν επέκταση προσήμου και συνήθως τοποθετούνται στο αριστερό τμήμα των διαθέσιμων θέσεων ώστε να μειώσουν την πολυπλοκότητα του υλικού. Συγκεκριμένα το bit προσήμου για όλες τις άμεσες τιμές είναι το 31 ώστε να επιταχυνθεί το κύκλωμα επέκτασης προσήμου.

Υπάρχουν επιπλέον 2 παραλλαγές των μορφών των εντολών (B/J), οι οποίες διακρίνονται με βάση τη διαχείριση των άμεσων τιμών και παρατίθενται στην *Εικόνα 28*.



Εικόνα 28 : Επιπρόσθετες παραλλαγές εντολών RISC-V

Η μόνη διαφορά ανάμεσα στην μορφή S και την B είναι ότι η άμεση τιμή των 12 bit χρησιμοποιείται για να κωδικοποιήσει πολλαπλάσια του 2 όφσετ διακλαδώσεων στη B. Αντί της ολίσθησης όλων των bit της άμεσης τιμής αριστερά κατά μία θέση όπως γίνεταια συμβατικά, τα μεσαία bit (imm[10:1] και το bit προσήμου διατηρούνται σε σταθερές θέσεις, ενώ το χαμηλότερο bit στη μορφή S (inst[7]) κωδικοποιεί bit υψηλής τάξης στη μορφή B. Ομοίως, η μόνη διαφορά μεταξύ των μορών U και J είναι ότι τα 20 bit άμεσης τιμής ολισθαίνουν αριστερά κατά 12 θέσεις για να δημιουργήσουν άμεσες τιμές U και κατά 1 για άμεσες τιμές J. Η τοποθεσία των bit της άμεσης τιμής στις μορφές U και J επιλέχθηκε για να μεγιστοποιήσει την επικάλυψη με τις άλλες μορφές αλλά μεταξύ τους. Ακολουθεί εκτενέστερη ανάλυση των μορφών R και I στις οποίες στηρίχθηκαν οι νέες προσεγγιστικές εντολές.

4.2. Ακέραιες πράξεις καταχωρητή-καταχωρητή

Οι μορφή R χρησιμοποιείται για να κωδικοποιήσει αριθμητικές και λογικές πράξεις. Όλες οι εντολές αυτού του τύπου δέχονται τους καταχωρητές rs1 και rs2 ως ορίσματα πηγής και τοποθετούν το αποτέλεσμα στον καταχωρητή rd. Τα πεδία funct7 και funct3 επιλέγουν το είδος της πράξης. Παρακάτω φαίνεται το σύνολο των εντολών που διαθέτουν τη μορφή αυτή.

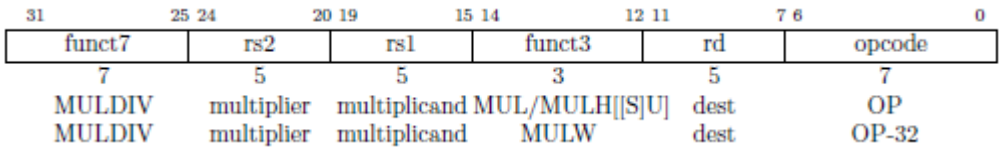
31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

Εικόνα 29 : Δομή εντολής τύπου R

Όπως γίνεται αντιληπτό οι ADD και SUB πραγματοποιούν πρόσθεση και αφαίρεση αντίστοιχα. Οι υπερχειλίσεις αγνοούνται και τα κατώτερα 32 bit του αποτελέσματος τοποθετούνται στον καταχωρητή προορισμού. Οι εντολές SLT και SLTU πραγματοποιούν προσημασμένη και απρόσημη σύγκριση αντίστοιχα, τοποθετώντας 1 στον προορισμό αν rs1 < rs2 και 0 διαφορετικά. Η δεύτερη σειρά απεικονίζει τις εντολές λογικών πράξεων AND, OR και XOR, οι οποίες πραγματοποιούνται ανά bit.Τέλος οι εντολές SLL, SRL και SRA πραγματοποιούν λογική ολίσθηση αριστερά, λογική ολίσθηση δεξιά και αριθμητική ολίσθηση δεξιά αντίστοιχα στην τιμή του καταχωρητή rs1 κατά τον αριθμό θέσεων που υποδεικνύουν τα 5 κατώτερα bit του καταχωρητή rs2.

4.3.1. Καθορισμός Επέκτασης «M» ακεραίων

Όπως βλέπουμε και παρακάτω η δομή των πράξεων πολλαπλασιασμού ακολουθεί τη μορφή R. Η εντολή MUL πραγματοποιεί πολλαπλασιασμό 32 x 32 bit και τοποθετεί στο καταχωρητή προορισμού τα 32 κατώτερα bit του αποτελέσματος. Οι MULH, MULHU, και MULSHSU πραγματοποιούν την ίδια πράξη αλλά τοποθετούν τα 32 υψηλότερα bit του αποτελέσματος στον καταχωρητή προορισμού.



Εικόνα 30 : Επέκταση «M» του RISC-V

Η διαφορά είναι πως γίνεται η πράξη με προσημασμένο x προσημασμένο, απρόσημο x απρόσημο και προσημασμένο x απρόσημο αντίστοιχα. Αν απαιτούνται τόσο τα υψηλότερα όσο και τα χαμηλότερα bits του αποτελέσματος των 64 bit τότε πρέπει να ακολουθεί η παρακάτω προτεινόμενη ακολουθία εντολών : MULH[[S]U] rdh, rs1, rs2; MUL rdl, rs1,rs2 .
Προκειμένου να εξεταστεί η συμπεριφορά προσεγγιστικών πολλαπλασιαστών στην αριθμητική ομάδα του PMicroX προστέθηκε υποστήριξη για την εντολή MUL η οποία έχει την ακόλουθη δομή :

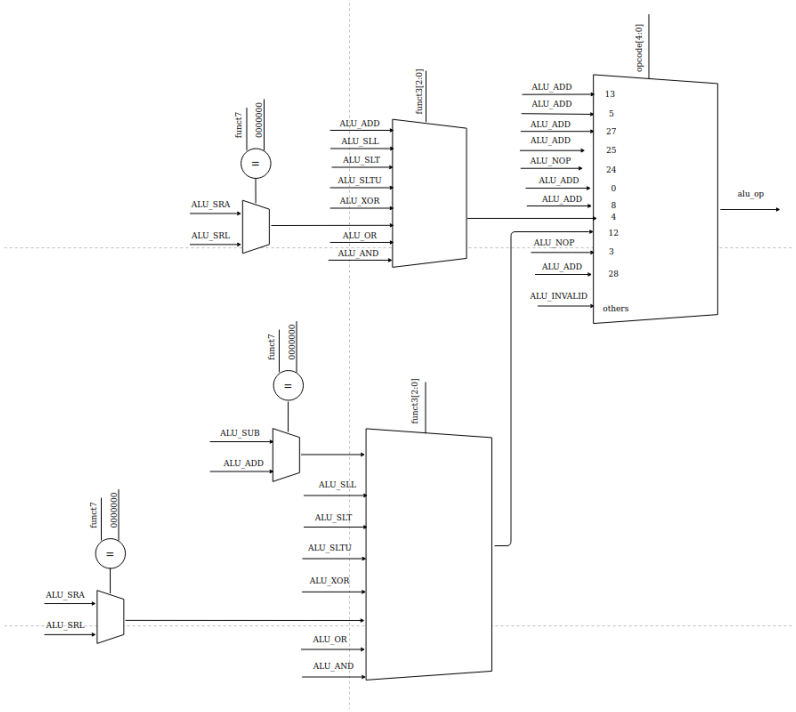
0000001	rs2	rs1	000	rd	0110011	MUL

Εικόνα 31 : Εντολή mul

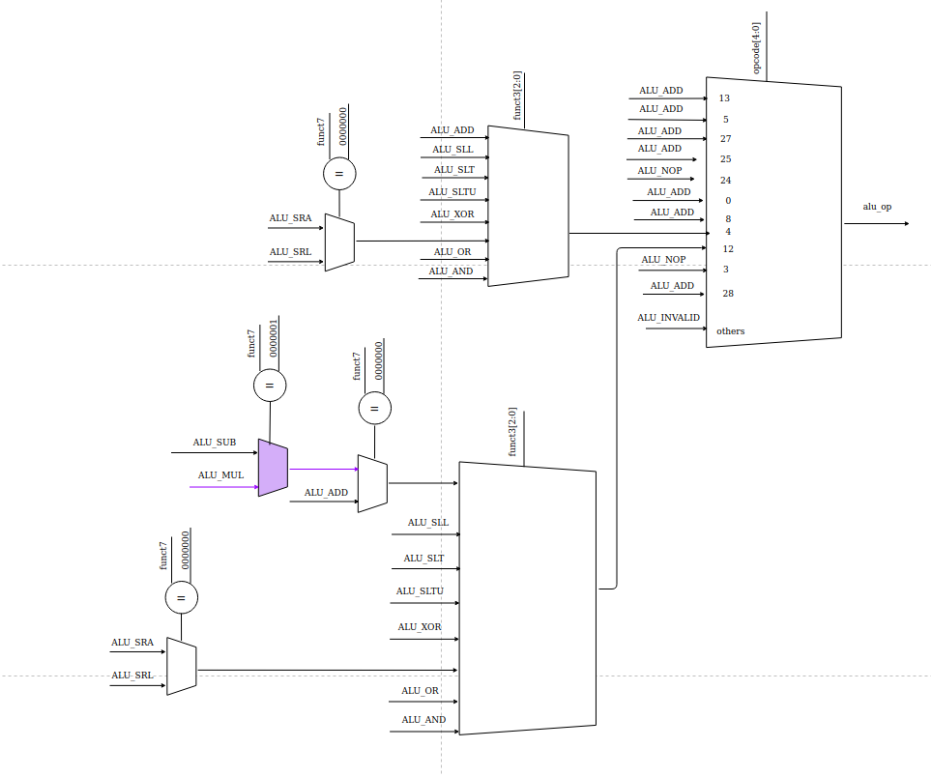
4.3.2 Υποστήριξη υλικού για την εντολή mul

Οι αλλαγές που πρέπει να πραγματοποιηθούν για να προστεθεί η εντολή mul είναι οι εξής:

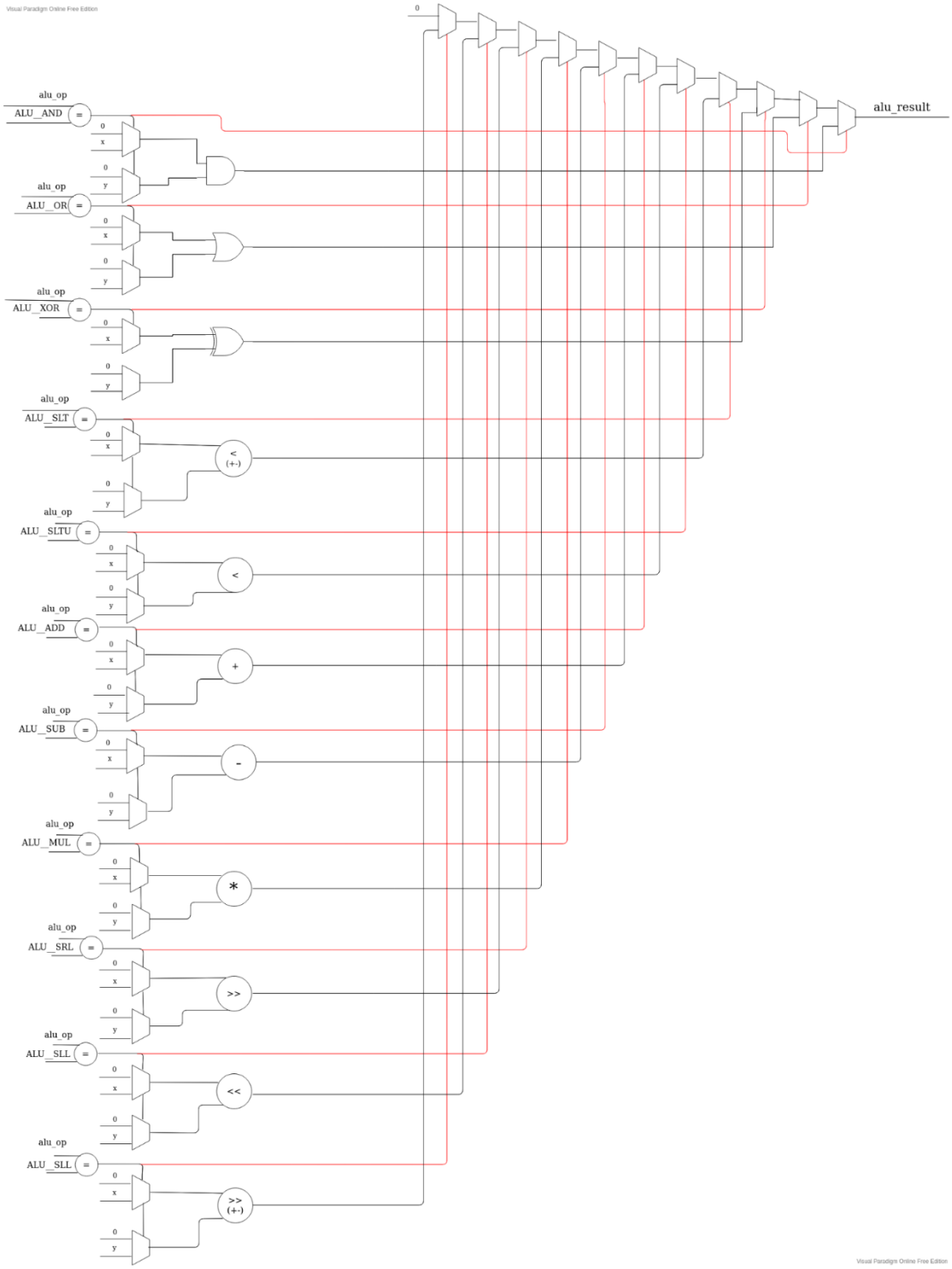
1. Στο στάδιο decode και συγκεκριμένα στο μπλοκ του alu_control unit προστίθεται μια επιπλέον επιλογή όσον αφορά την τιμή alu op. Στο σχήμα του pipeline που παρουσιάστηκε στο 1ο κεφάλαιο το μπλοκ παρουσιάζεται απλουστευμένα ως ένας μεγάλος πολυπλέκτης. Στην πραγματικότητα περιλαμβάνει 3 πολυπλέκτες οι οποίοι καθορίζουν τις τιμές των alu_x_src, alu_y_src και alu_op. Οι δύο πρώτες καθορίζουν τον τύπο των ορισμάτων (immediate, pc, register, shamt,csr,null) που εισάγονται στην αριθμητική μονάδα στο στάδιο execute. Η προσθήκη της εντολής mul δεν επηρεάζει τους πολυπλέκτες των alu_x_src και alu_y_src αλλά μόνο τον πολυπλέκτη του alu_op. Παρακάτω φαίνεται η αρχική μορφή του πολυπλέκτη και η μορφή μετά την προσθήκη υποστήριξης για την εντολή *mul*.
2. Στο στάδιο execute και συγκεκριμένα στο εσωτερικό της μονάδας αριθμητικών πράξεων(ALU) προστίθενται επιπλέον σήματα που οδηγούν τη μονάδα πολλαπλασιασμού. Για τον πολλαπλασιαστή επιλέχθηκε η προκαθορισμένη μονάδα της xilinx με χρήση του τελεστή '*'. Παρακάτω φαίνεται το εσωτερικό της μονάδας ALU με την προσθήκη σημάτων και μονάδας πολλαπλασιασμού (Στο σχήμα απεικονίζεται και η λογική μηδενισμού εισόδων που δεν χρησιμοποιούνται).



Εικόνα 37 : Αρχική μορφή συστήματος καθορισμού του είδους αριθμητικής πράξης που θα εκτελεστεί στην ALU (σήμα alu_op)



Εικόνα 38 : Σύστημα καθορισμού πράξης ALU μετά την προσθήκη υποστήριξης της εντολής mul



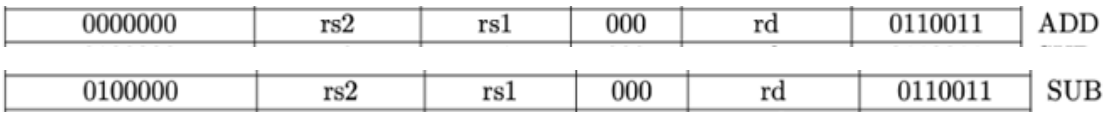
Εικόνα 39 : Αρχιτεκτονική αριθμητικής μονάδας ALU του PMicroX με υποστήριξη πολλαπλασιασμού

Στο σχήμα απεικονίζονται τα σήματα '< (+-)' και '>>(+-)'. Αυτά σημαίνουν προσημασμένη σύγκριση και προσημασμένη ολίσθηση προς τα δεξιά αντίστοιχα.Τα σύμβολα '<', '<<' και '>>' σημαίνουν απρόσημη σύγκριση, απρόσημη ολίσθηση προς τα αριστερά και απρόσημη ολίσθηση προς τα δεξιά αντίστοιχα.

Η αρχική υλοποίηση της ALU οδηγούσε τα σήματα x και y σε όλες τις εισόδους των επιμέρους κυκλωμάτων αριθμητικών και λογικών πράξεων και ένας πολυπλέκτης στο τέλος επέλεγε το σήμα που θα οδηγηθεί στην έξοδο.Εδώ εκτός από τους πολυπλέκτες εισόδου παρατηρείται αλλαγή και στην επιλογή εξόδου καθώς αντικαθίσταται ο ενιαίος πολυπλέκτης εξόδου με 11 μικρότερους πολυπλέκτες συνδεδεμένους σε αλυσίδα.

4.4.1 Καθορισμός δομής προσεγγιστικών εντολών xadd, xsub και xmul

Προκειμένου να δομηθούν οι νέες εντολές με την προσθήκη του ελάχιστου επιπλέον υλικού να τις υποστηρίζει πρέπει να βασιστούν σε ήδη υπάρχουσες εντολές και να τροποποιηθούν μόνο ορισμένα πεδία.



Εικόνα 32 : Εντολές add και sub

Οι εντολές add, sub και mul διαθέτουν ίδιο πεδίο *opcode = 0110011* και πεδίο *funct3 = 000* . Το πεδίο που αλλάζει και τις διακρίνει είναι το funct7 το οποίο είναι αντίστοιχα :

ADD -> funct7 = 0000000

SUB -> funct7 = 0100000

MUL -> funct7 = 0000001

Επιλέγεται οι νέες εντολές να έχουν funct7 ως εξής :

XADD -> funct7 = 0000010

XSUB -> funct7 = 0000011

XMUL -> funct7 = 0000100

Η τελική μορφή των τριων προσεγγιστικών εντολών φαίνεται παρακάτω:

0000010	rs2	rs1	000	rd	0110011	XADD
0000011	rs2	rs1	000	rd	0110011	XSUB
0000100	rs2	rs1	000	rd	0110011	XMUL

4.4.2. Ενσωμάτωση εντολών assembly σε κώδικα C

Προκειμένου να προστεθούν οι νέες εντολές στο σετ εντολών RISC-V θα πρέπει να γίνουν ορισμένες προσθήκες στα αρχεία του riscv toolchain. Συγκεκριμένα οι αλλαγές αφορούν τα binutils και πιο ειδικά τον assembler.Έχει προστεθεί επέκταση των binutils(patch) η οποία διαθέτει μια επιπλέον οδηγία (.insn) η οποία επιτρέπει την εγγραφή εντολών με διαφορετική μορφή ακριβώς όπως η οδηγία s/390's .insn. Ο σκοπός της οδηγίας αυτής είναι να διευκολύνει την προσθήκη νέων εντολών για πειραματικούς ελέγχους χωρίς να τροποποιείται ο κώδικας των binutils και είναι αρκετά πιο εύχρηστο από τη χρήση του .word ώστε να κωδικοποιηθεί μία εντολή [28]. Όπως περιγράφηκε και στο Κεφάλαιο 2 η κύρια διαφορά του compiler με τον assembler είναι ότι ο compiler μετατρέπει γλώσσα υψηλού επιπέδου σε γλώσσα μηχανής ενώ ο assembler μετατρέπει γλώσσα assembly σε γλώσσα μηχανής. Κατά τη διαδικασία μεταγλώττισης απαιτείται να μετατραπούν σε γλώσσα μηχανής τμήματα κώδικα υψηλού επιπέδου αλλά και τμήματα κώδικα assembly (crt0.S). Επίσης ο assembler χρησιμοποιείται όταν υπάρχουν τμήματα κώδικα assembly ενσωματωμένα σε κώδικα υψηλού επιπέδου. Γενικά η τεχνική αυτή χρησιμοποιείται όταν απαιτείται χειροκίνητη βελτιστοποίηση τμημάτων κώδικα με αυστηρούς χρονικούς περιορισμούς ή για να χρησιμοποιηθούν συγκεκριμένες εντολές οι οποίες δεν είναι διαθέσιμες στη γλώσσα υψηλού επιπέδου.

Προκειμένου να εισαχθεί μια ή παραπάνω εντολή assembly σε C κώδικα χρησιμοποιείται το πρόθεμα asm. Παρέχονται δύο είδη δηλώσεων asm, η βασική δήλωση asm η οποία δεν διαθέτει ορίσματα και η εκτενής δήλωση η οποία διαθέτει ένα ή περισσότερα ορίσματα. Με την δεύτερη επιτρέπεται η ανάγνωση και εγγραφή μεταβλητών της C από τον assembler αλλά και η υλοποίηση αλμάτων από τον κώδικα του assembler σε επιγραφές της C. Η σύνταξη του προθέματος είναι η εξής [29] :

```
asm asm-qualifiers ( AssemblerTemplate
                  : OutputOperands
                  [ : InputOperands
                  [ : Clobbers ] ])
```

```
asm asm-qualifiers ( AssemblerTemplate
                       : OutputOperands
                       : InputOperands
                       : Clobbers
                       : GotoLabels)
```

Το πεδίο qualifiers μπορεί 3 τιμές :
volatile: Η τυπική χρήση της εκτενής δήλωσης asm είναι να διαχειριστεί τιμές εισόδου ώστε να παράγει τιμές εξόδου. Ωστόσο η δήλωση αυτή είναι δυνατόν να προκαλέσει παράπλευρα αποτελέσματα. Σε αυτή την περίπτωση ο προσδιορισμός volatile απενεργοποιεί συγκεκριμένες βελτιστοποιήσεις που πιθανόν να γίνουν(όπως για παράδειγμα μετατόπιση κώδικα εκτός επαναληπτικού βρόχου όταν θεωρείται ότι ο νέος κώδικας καταλήγει σε ίδιο αποτέλεσμα).
inline: Το μέγεθος της δήλωσης asm επιλέγεται ως το μικρότερο δυνατό (Μερικές συσκευές απαιτούν από το GCC να υπολογίσει το μέγεθος της κάθε εντολής που χρησιμοποιείται προκειμένου να εξαχθεί σωστός κώδικας. Επειδή το τελικό μέγεθος του κώδικα που παράγεται από τη δήλωση asm είναι γνωστό μόνο από τον assembler, το GCC πρέπει να εκτιμήσει το μέγεθος του. Για αυτό προσμετρά τον αριθμό των εντολών στο εσωτερικό της δήλωσης asm και τον πολλαπλασιάζει με το μήκος της μεγαλύτερης εντολής που υποστηρίζεται από τον επεξεργαστή).Έτσι ο προσδιορισμός inline αγνοεί το πλήθος των εντολών που υπολόγισε το GCC και το τμήμα της δήλωσης υπολογίζεται ως το μικρότερο δυνατό.
goto: Επιτρέπει σε assembly κώδικα να μεταβαίνει σε επιγραφές της C.

Το πεδίο *Assembler Template* περιγράφει επακριβώς την εντολή ή τις εντολές που περιλαμβάνονται στη δήλωση asm.

Το πεδίο *Output Operands* αποτελεί μια λίστα μεταβλητών της C, διαχωρισμένη με κόμμα η οποία τροποποιείται από τις εντολές στο εσωτερικό του Assembler Template (επιτρέπεται κενή λίστα).

Το πεδίο *Input Operands* αποτελεί μια λίστα από εκφράσεις της C διαχωρισμένη με κόμμα, η οποίες διαβάζονται από τις εντολές στο εσωτερικό του Assembler Template (επιτρέπεται κενή λίστα).

Το πεδίο *Clobbers* αποτελεί μια διαχωρισμένη με κόμμα λίστα με καταχωρητές ή άλλες τιμές που τροποποιούνται από τις εντολές στο εσωτερικό του Assembler Template.

Ένα παράδειγμα χρήσης asm δηλώσεων στις οποίες να αναφέρονται οι 3 προσεγγιστικές εντολές που δημιουργήθηκαν φαίνεται παρακάτω :

```
int a,b,c;
 a = 1;
 b = 1;

 asm volatile
 (
  "xadd   %[z], %[x], %[y]\n\t"
  : [z] "=r" (c)
  : [x] "r" (a), [y] "r" (b)
 );

asm volatile
 (
  "xsub   %[z], %[x], %[y]\n\t"
  : [z] "=r" (c)
  : [x] "r" (a), [y] "r" (b)
 );

asm volatile
 (
  "xmul   %[z], %[x], %[y]\n\t"
  : [z] "=r" (c)
  : [x] "r" (a), [y] "r" (b)
 );
```

Χρησιμοποιώντας το binutil objdump πραγματοποιείται η αντίστροφη διαδικασίας του assemble η οποία ονομάζεται disassemble και λαμβάνει τον assembly κώδικα από γλώσσα μηχανής.

```
  c:   00100793                li      a5,1
 10:   fef42623                sw      a5,-20(s0)
 14:   00100793                li      a5,1
 18:   fef42423                sw      a5,-24(s0)
 1c:   fec42783                lw      a5,-20(s0)
 20:   fe842703                lw      a4,-24(s0)
 24:   04e787b3                xadd    a5,a5,a4
 28:   fef42223                sw      a5,-28(s0)
 2c:   fec42783                lw      a5,-20(s0)
 30:   fe842703                lw      a4,-24(s0)
 34:   06e787b3                xsub    a5,a5,a4
 38:   fef42223                sw      a5,-28(s0)
 3c:   fec42783                lw      a5,-20(s0)
 40:   fe842703                lw      a4,-24(s0)
 44:   08e787b3                xmul    a5,a5,a4
 48:   fef42223                sw      a5,-28(s0)
 4c:   00000793                li      a5,0
```

Εικόνα 33 : Κώδικας assembly επίδειξης νέων προσεγγιστικών εντολών

Αριστερά των προσεγγιστικών εντολών φαίνεται και ο δεκαεξαδικός αριθμός 8 ψηφίων που περιγράφει την εντολή.

4.4.3. Τροποποιήσεις αρχείων του riscv-toolchain για την προσθήκη προσεγγιστικών εντολών xadd,xsub και xmul

Παρατηρώντας το αρχείο *riscv-binutils/include/opcode/riscv-opc.h* εξάγεται η μορφή των πεδίων MASK και MATCH για τις προσεγγιστικές εντολές. Το πεδίο MASK προκύπτει αν αντικατασταθούν με '1' τα σταθερά πεδία της εντολής που είναι τα opcode, funct3 και funtc7 και με '0' τα μεταβλητά πεδία που είναι τα rs1, rs2 και rd. Το πεδίο MATCH προκύπτει αν αφεθούν ως έχουν τα σταθερά πεδία της εντολής και με '0' τα μεταβλητά πεδία. Με βάση τα παραπάνω προκύπτει ότι:

```
MASK_XADD = 0xfe00707f
MATCH_XADD=  0x4000033

MASK_XSUB = 0xfe00707f
MATCH_XSUB = 0x6000033

MASK_XMUL = 0xfe00707f
MATCH_XMUL = 0x8000033
```

Στο αρχείο *riscv-binutils/include/opcode/riscv-opc.h* προστίθενται τα ακόλουθα:

```
#define MATCH_XADD 0x4000033
#define MASK_XADD  0xfe00707f
#define MATCH_XSUB 0x6000033
#define MASK_XSUB  0xfe00707f
#define MATCH_XMUL 0x8000033
#define MASK_XMUL  0xfe00707f
. . .
DECLARE_INSN(xadd, MATCH_XADD, MASK_XADD)
DECLARE_INSN(xsub, MATCH_XSUB, MASK_XSUB)
DECLARE_INSN(xmul, MATCH_XMUL, MASK_XMUL)
```

Ακολούθως πραγματοποιείται τροποποίηση στον πίνακα κωδικοποίησης στη βιβλιοθήκη των opcode. Στο αρχείο *riscv-binutils/opcodes/riscv-opc.c* στο εσωτερικό της δομής *riscv_opcodes* παρατηρούμε τις δομές όλων των εντολών που υποστηρίζονται από το σύνολο εντολών και διαθέτουν την ακόλουθη γενική μορφή [30]:

{Name, Xlen, Instruction class, Instruction operands, Match, Mask, Match opcode, Pinfo}

Περιλαμβάνει τα εξής πεδία :

1. **Name** : Το όνομα της εντολής
2. **Xlen** : Αν η εντολή ανήκει μόνο στο σετ RV32 τότε είναι 32, αν ανήκει μόνο στο σετ RV64 τότε είναι 64 και αν ανήκει και στα δύο τότε είναι 0.
3. **Instruction class** : Η επέκταση στην οποία ανήκει η εντολή. Η λίστα με τις πιθανές κλάσεις βρίσκεται στο αρχείο *riscv-binutils/include/opcode/riscv.h* στην κλάση *riscv_insn_class* και είναι οι :

```
INSN_CLASS_NONE,
INSN_CLASS_I,
INSN_CLASS_C,
INSN_CLASS_A,
INSN_CLASS_M,
INSN_CLASS_F,
INSN_CLASS_D,
INSN_CLASS_D_AND_C,
INSN_CLASS_F_AND_C,
INSN_CLASS_Q,
```

4. **Instruction operands**: Καθορίζει τα ορίσματα της εντολής. Η αρχιτεκτονική RISC-V έχει ήδη ορίσει αρκετά διαφορετικά είδη ορισμάτων τα οποία βρίσκονται στο αρχείο *riscv-binutils/gas/config/tc-riscv.c* αλλά υπάρχει η δυνατότητα ορισμού νέων πλέον των ήδη υπάρχοντων με ορισμένες επιπλέον διαδικασίες. Εφόσον οι νέες προσεγγιστικές εντολές θα βασιστούν σε ήδη υπάρχουσες εντολές δεν υπάρχει ανάγκη δημιουργίας νέων.
5. **Match** : Αναλύθηκε παραπάνω
6. **Mask** : Ορίζεται όπως αναφέρθηκε στο αρχείο *riscv-binutils/include/opcode/riscv-opc.h* και χρησιμοποιείται για να διακρίνει την κάθε εντολή από τη ροή bit. Η μέθοδος διάκρισης της εντολής ορίζεται από το επόμενο πεδίο :
7. **Match opcode**: Η γενικευμένη συνάρτηση του RISC-V, match opcode εφαρμόζει τη λογική : encoding & MASK == MATCH
8. **Pinfo** : Αποτελεί μια σειρά από bit τα οποία περιγράφουν τη συνάρτηση και ιδιαίτερα πληροφορίες για κάποιο σχετικό κίνδυνο.

Βάσει των παραπάνω επιλέγεται και στις 3 εντολές το πεδίο Instruction class να είναι το *INSN_CLASS_I*, δηλαδή να ανήκουν στην βασική επέκταση του σετ εντολών. Ακόμα επιλέγεται και για τις 3 εντολές στο πεδίο Instruction operands την τιμή *"d,s,t"* η οποία είναι ίδια με τις εντολές add, sub, mul. Στο πεδίο Xlen τοποθετείται 0 ώστε να ανήκει τόσο στο σετ RV32 όσο και στο RV64 και στο πεδίο Pinfo επίσης τοποθετείται μηδέν εφόσον δεν χρειάζεται κάποια επιπλέον πληροφορία για την περιγραφή της εντολής (όπως και στις add, sub και mul).

Άρα η τελευταία προσθήκη γίνεται στο αρχείο *riscv-binutils/opcodes/riscv-opc.c* μέσα στη δομή *const struct riscv_opcode riscv_opcodes[] =* :

```
{"xadd", 0, INSN_CLASS_I, "d,s,t", MATCH_XADD, MASK_XADD, match_opcode, 0 },
{"xsub", 0, INSN_CLASS_I, "d,s,t", MATCH_XSUB, MASK_XSUB, match_opcode, 0 },
{"xmul", 0, INSN_CLASS_I, "d,s,t", MATCH_XMUL, MASK_XMUL, match_opcode, 0 },
```

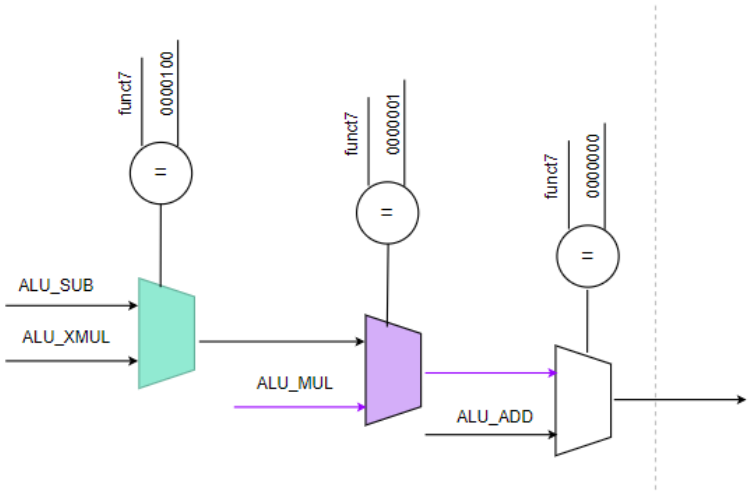
Σημείωση: Οι σειρές αυτές που περιγράφουν τη δομή των 3 νέων προσεγγιστικών εντολών δεν πρέπει να τοποθετηθούν ανάμεσα στις υποκατηγορίες αλλά στην αρχή.

Οι παραπάνω προσθήκες γίνονται και στο φάκελο *riscv-gdb* ο οποίος έχει τα ίδια περιεχόμενα και ονόματα αρχείων με τον *riscv-binutils* και έπειτα να χτιστεί εκ νέου η riscv-toolchain.

4.4.4. Υποστήριξη υλικού για την εντολή xmul

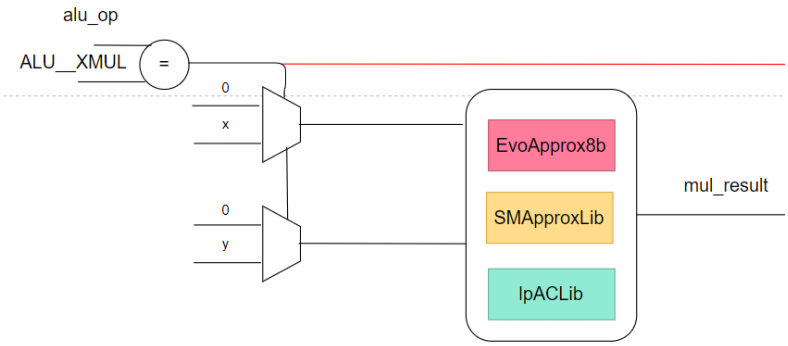
Οι προσθήκες που απαιτούνται για την υποστήριξη της εντολής xmul βρίσκονται στα ίδια σημεία με τις αλλαγές για την εντολή mul και είναι:

1. Στο σύστημα που καθορίζει την τιμή του σήματος alu_op αλλάζει μόνο το τμήμα πριν τον πολυπλέκτη που προστέθηκε για την εντολή mul (με μωβ χρώμα).

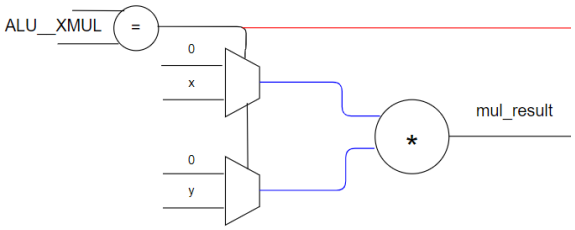


Εικόνα 40 : Προσθήκη υποστήριξης της εντολής xmul στο σύστημα καθορισμού του σήματος alu_op

2. Στην αριθμητική μονάδα ALU προστίθεται μια επιπλέον μονάδα που θα εκτελεί προσεγγιστικό πολλαπλασιασμό, δύο πολυπλέκτες που θα μηδενίζουν τις εισόδους για κάθε άλλη αριθμητική λειτουργία εκτός από αυτή και ένας πολυπλέκτης επιλογής αποτελέσματος στην σειρά των πολυπλεκτών εξόδου.Για την μονάδα πολλαπλασιασμού επιλέχθηκαν 2 μέθοδοι. Πρώτον η τοποθέτηση προσεγγιστικών πολλαπλασιαστών από τις βιβλιοθήκες ανοιχτού κώδικα EvoApprox8b [6], SMApproxLib [9], [31] και lpACLib [32] και δεύτερον η τοποθέτηση ακριβούς πολλαπλασιαστή (με χρήση του τελεστή '*' ο οποίος είναι βελτιστοποιημένος από την xilinx) με αποκοπή bit στις τιμές εισόδου. Στο δεύτερο σχήμα με μπλε χρώμα απεικονίζονται τα σήματα με αποκομμένα τα n λιγότερο σημαντικά bit.



Εικόνα 41 : Υποστήριξη xmul στην ALU με χρήση προσεγγιστικών πολλαπλασιαστών

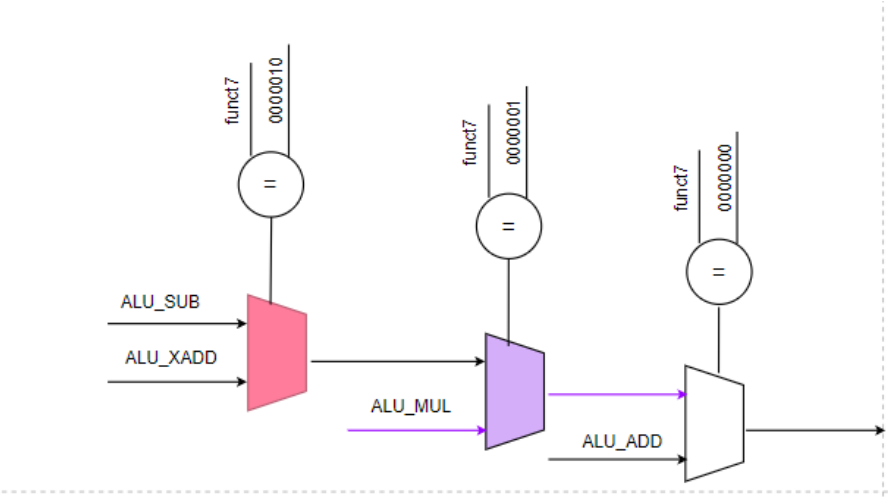


Εικόνα 42 : Υποστήριξη xmul στην ALU με χρήση ακριβούς πολλαπλασιαστή και αποκοπή bit εισόδου

4.4.5. Υποστήριξη υλικού για την εντολή xadd

Όμοια με τις παραπάνω προσεγγιστικές αριθμητικές εντολές απαιτούνται 2 προσθήκες στον πολυπλέκτη του σήματος alu_op και στην ALU.

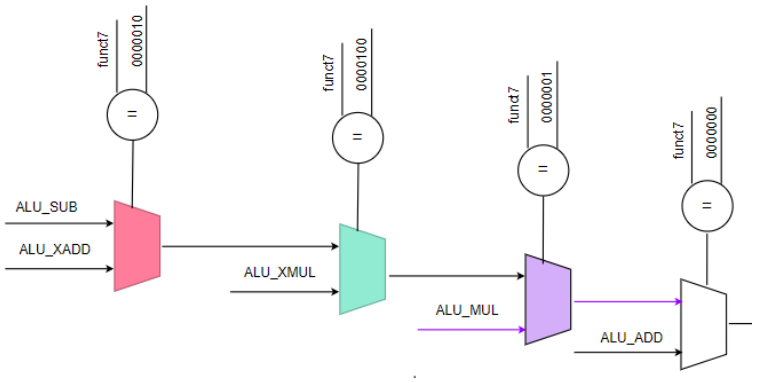
1. Αντι να τοποθετηθεί ο πολυπλέκτης του xmul πριν από τον πολυπλέκτη mul, τοποθετείται πολυπλέκτης με για την xadd.



Εικόνα 43 : Προσθήκη υποστήριξης της εντολής xadd στο σύστημα καθορισμού του σήματος alu_op

2. Στην ALU προστίθεται αντι της μονάδας προσεγγιστικού πολλαπλασιασμού μια μονάδα προσεγγιστικού αθροιστή. Όπως και πριν δοκιμάστηκαν τόσο προσεγγιστικοί αθροιστές ανοιχτού κώδικα από τις βιβλιοθήκες EvoApprox8b [33], DeMAS [34] και lpACLib [35] όσο και ακριβείς αθροιστές με αποκοπή bit εισόδου. Η αλλαγές στην ALU είναι πανομοιότυπες με αυτές της xmul οπότε δεν αναπαρίστανται σχηματικά.

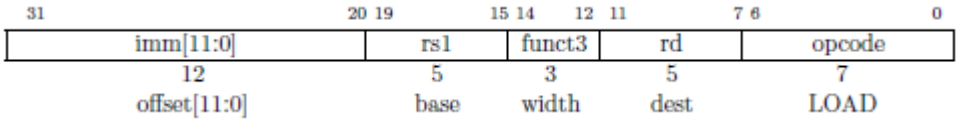
Σημείωση: Οι δύο παραπάνω εντολές xmul και xadd μπορούν να χρησιμοποποιηθούν είτε κάθε μια ξεχωριστά είτε ταυτόχρονα, δηλαδή να παρέχεται υποστήριξη και για τις δύο μαζί. Στην περίπτωση αυτή οι πολυπλέκτες στο αρχείο alu_control_unit θα έχουν την παρακάτω διάταξη και στην ALU θα προστεθούν τόσο μονάδες προσεγγιστικού πολλαπλασιασμού όσο και προσεγγιστικής πρόσθεσης καθώς και τα αντίστοιχα σήματα. Στις μονάδες προφανώς μπορεί να χρησιμοποιηθεί οποιοσδήποτε συνδυασμός προσεγγιστικών κυκλωμάτων-αποκοπής bit.



Εικόνα 44 : Προσθήκη υποστήριξης της εντολής xmul και xadd στο σύστημα καθορισμού του σήματος alu_op

4.5.1. Η εντολή Load

Η αρχιτεκτονική RV32I είναι φόρτωσης-αποθήκευσης που σημαίνει ότι μόνο οι εντολές load και store έχουν πρόσβαση στη μνήμη και οι αριθμητικές εντολές επηρεάζουν τους καταχωρητές. Η αρχιτεκτονική αυτή παρέχει 32 bit χώρο διευθύνσεων χρήστη ο οποίος είναι προσβάσιμος ανά byte και μικρού άκρου(δηλαδή το λιγότερο σημαντικό byte κάθε λέξης τοποθετείται στη μικρότερη θέση μνήμης και το περισσότερο σημαντικό στη μεγαλύτερη θέση μνήμης). Η εντολή load μεταφέρει τιμές από τη μνήμη στους καταχωρητές και κωδικοποιείται με τη I-μορφή.Η διεύθυνση αποκτάται από την πρόσθεση του καταχωρητή rs1 στην άμεση τιμή των 12 bit με επέκταση προσήμου.



Εικόνα 34 : Δομή εντολών τύπου LOAD

Η εντολή LW φορτώνει από τη μνήμη μια τιμή 32 bit στον καταχωρητή προορισμού rd.Η εντολή LH φορτώνει μια τιμή 16 bit από τη μνήμη και την επεκτείνει με πρόσημο στα 32 bit πριν την τοποθετήσει στον καταχωρητή rd. Η LHU φορτώνει μία τιμή 16 bit από τη μνήμη και την επεκτείνει με '0' στα 32 bit πριν την τοποθετήσει στον rd.Οι εντολές LB και LBU ορίζονται αντίστοιχα για τιμές 8 bit.

4.5.2. Καθορισμός δομής προσεγγιστικής εντολής xlw

Η προσεγγιστική εντολή xlw επιλέχθηκε να έχει ίδια δομή με την lw η οποία φαίνεται παρακάτω:

Εικόνα 35 : Εντολή LW

Διατηρείται το ίδιο **opcode = 0000011** και επιλέγεται η τιμή του πεδίου **funt3 = 011** ώστε να μην συμπίπτει με κάποια άλλη εντολή φόρτωσης I-μορφής.

Έτσι η μορφή της εντολής θα είναι :

imm[11:0]	rs1	011	rd	0000011	XLW

Όπως και οι προηγούμενες προσεγγιστικές αριθμητικές εντολές που δημιουργήθηκαν, ενσωματώνονται στον κώδικα C μέσω της δήλωσης asm.Ένα παράδειγμα αναφοράς της εντολής lw φαίνεται παρακάτω:

```
int source, destination;
 asm volatile
          (
            "xlw   %[z], 0( %[x])\n\t"
            : [z] "=r" (destination)
            : [x] "r" (&source)
          );
```

και η μορφή σε assembly αλλά και ο δεκαεξαδικός κωδικός 8 ψηφίων :

Εικόνα 36 : Κώδικας assembly επίδειξης προσεγγιστικής εντολής lw

4.5.3. Τροποποιήσεις αρχείων του riscv-toolchain για την προσθήκη της προσεγγιστικής εντολής xlw

Αρχικά γίνεται καθορισμός των πεδίων mask και match. Το πεδίο mask προκύπτει με αντικατάσταση των πεδίων opcode και funct3 τα οποία είναι σταθερά με την τιμή '1' και τα υπόλοιπα μεταβλητά πεδία που είναι τα immediate,rs1 και rd με '0'. Το πεδίο match προκύπτει με τοποθέτηση '0' στα immeditate, rs1 και rd και διατήρηση των opcode και funct3 ως έχουν.

```
MASK_XADD = 0x707f
MATCH_XADD=  0x3003
```

Άρα στο αρχείο *riscv-binutils/inlucde/opcode/riscv-opc.h* τοποθετούνται τα εξής:

```
#define MATCH_XLW 0x3003
#define MASK_XLW  0x707f
. . .
DECLARE_INSN(xlw, MATCH_XLW, MASK_XLW)
```

Επιπλέον στο αρχείο *riscv-binutils/opcodes/riscv-opc.c* προστέθηκε στο εσωτερικό της δομής *const struct riscv_opcode riscv_opcoded[] = :*

```
{"xlw", 0, INSN_CLASS_I, "d,o(s)", MATCH_XLW, MASK_XLW, match_opcode,
INSN_DREF|INSN_4_BYTE },
```

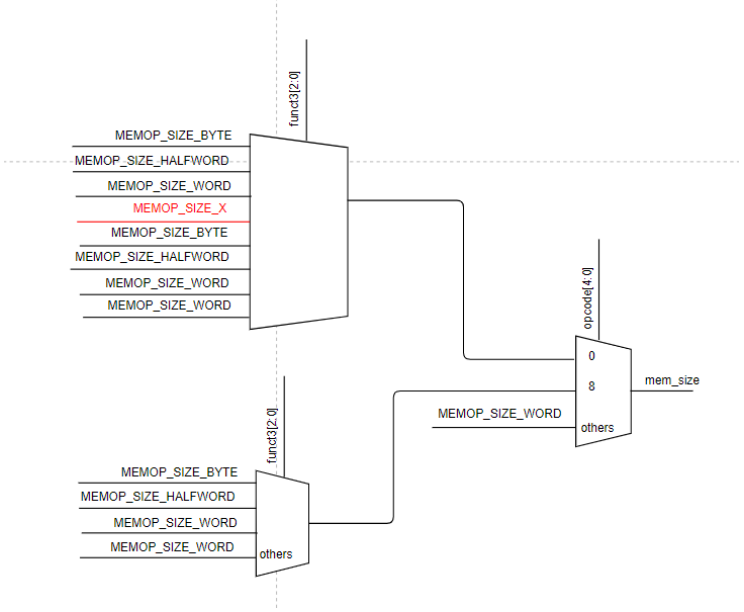
Όπως φαίνεται στο πεδίο **Xlen** επιλέχθηκε ο αριθμός 0 και στο πεδίο **Instruction class** η τιμή INSN_CLASS_I, για τους ίδιους λόγους με τις αριθμητικές προσεγγιστικές εντολές. Στο πεδίο Instruction operands επιλέχθηκε η τιμή "d,o(s)" η οποία προσθέτει μια άμεση τιμή στον καταχωρητή πηγής ώστε να σχηματίσει τη διεύθυνση και τοποθετεί το δεδομένο από την ανάγνωση της μνήμης στον καταχωρητή προορισμού rd. Τέλος στο πεδίο Pinfo αναγράφονται πληροφορίες για το πλήθος byte που φορτώνονται στην φόρτωση (επιλέχθηκε ίδιο με το αντίστοιχο πεδίο της εντολής lw η μορφή της οποίας βρίσκεται στη δομή *const struct riscv_opcode riscv_opcoded[]* όπως και όλες οι άλλες εντολές).

Σημείωση : Όλες οι προσθήκες που γίνονται στο εσωτερικό δομών πρέπει να τοποθετούνται στην αρχή και όχι ενδιάμεσα.

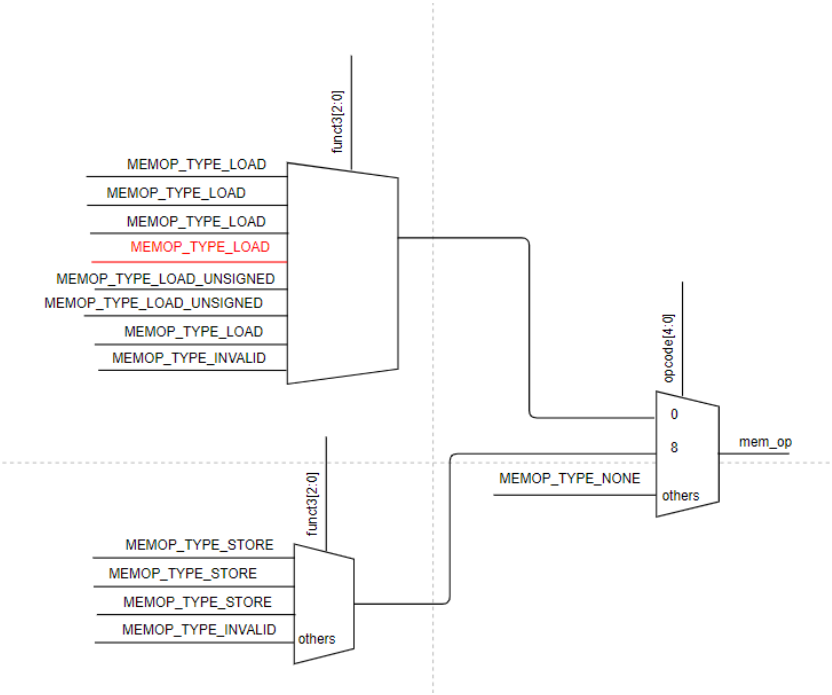
4.5.4. Υποστήριξη υλικού για την εντολή xlw

Οι προσθήκες σε hardware ώστε να προστεθεί υποστήριξη για την εντολή xlw είναι οι εξής:

1. Στο στάδιο *decode* της διοχέτευσης στον καταχωρητή *mem_decode* προστίθεται μια επιπλέον επιλογή. Ενώ απεικονίζεται ως ένας πολυπλέκτης χάριν απλούστευσης του σχήματος, στην ουσία αποτελείται από 2 πολυπλέκτες οι οποίοι οδηγούν τα σήματα mem_op και mem_size. Παρακάτω φαίνονται οι 2 πολυπλέκτες και με κόκκινο χρώμα υποδεικνύεται η προσθήκη.

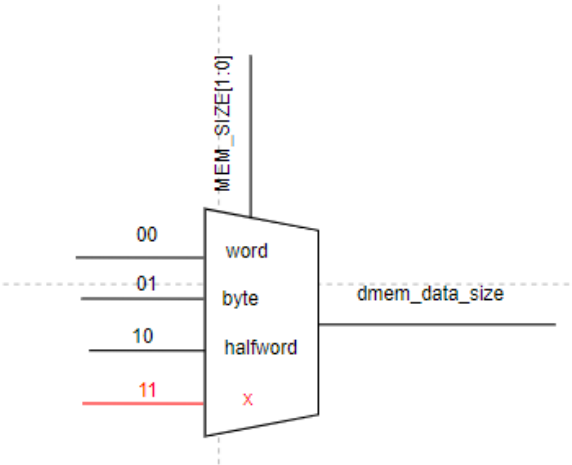


Εικόνα 45 : Σύστημα καθορισμού μεγέθους δεδομένου πρόσβασης στη μνήμη (σήμα mem_size)



Εικόνα 46 : Σύστημα καθορισμού είδους εντολής μνήμης (σήμα mem_op)

2. Στο στάδιο execute της διοχέτευσης ο πολυπλέκτης 4:1 που οδηγεί το σήμα dmem_data_size το οποίο είναι 2 bit και καθορίζει το μέγεθος της πρόσβασης στη μνήμη, αλλάζει ως εξής:



Εικόνα 47 : Πολυπλέκτης οδήγησης σήματος μεγέθους του αιτήματος στη μνήμη (dmem_data_size)

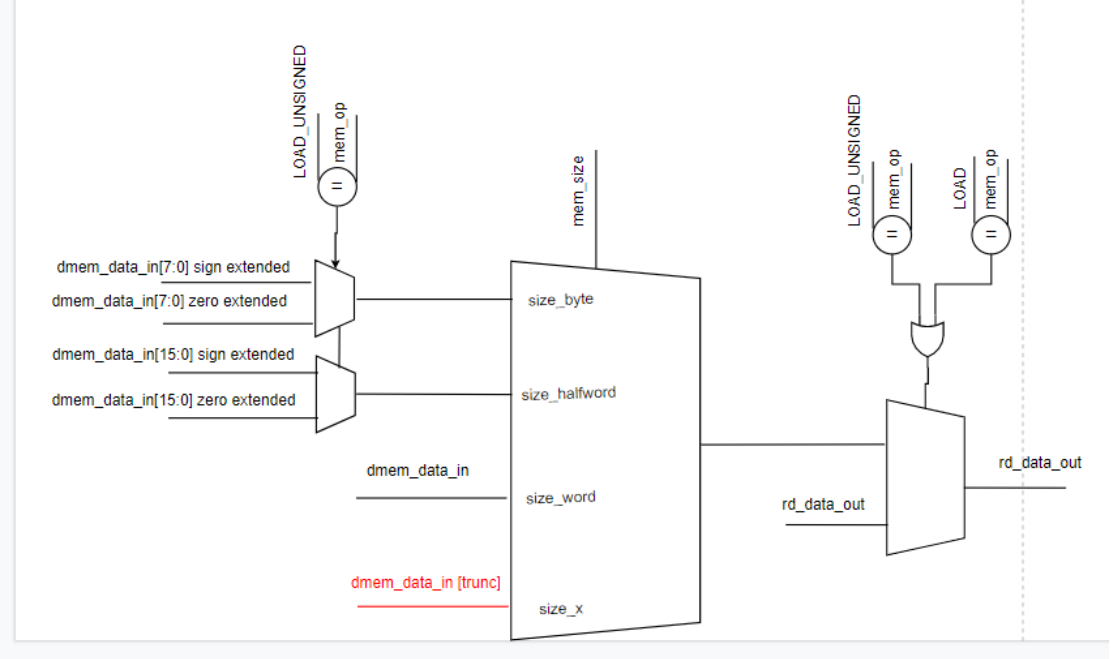
3. Στο στάδιο πρόσβασης στη μνήμη το οποίο είναι κοινό με το execute τροποποιείται λογική του enable της block ram που αποθηκεύει τα byte 0 των λέξεων(όπως είχε εξηγηθεί και στο Κεφάλαιο 1 στην περιγραφή της μνήμης δεδομένων χρησιμοποιούνται 4 block ram, μια για κάθε byte). Συγκεκριμένα στην αρχική μορφή όλα τα σήματα

enable των 4 block ram δέχονταν ως είσοδο την έξοδο μιας πύλης OR 2 εισόδων (των dmem_read_req και dmem_write_req). Αυτές οι είσοδοι αποτελούν τις αιτήσεις για ανάγνωση και εγγραφή και η λογική της or είναι να ελαχιστοποιήσει την κατανάλωση όταν δεν χρησιμοποιείται η μνήμη. Η αλλαγή αφορά την προσθήκη συνθήκης σε περίπτωση που το data_size είναι size_x. Πιο αναλυτικά η συνθήκη εκφράζεται ως εξής :

enable = (*dmem_read_req* or *dmem_write_req*) and not *dmem_size_x*

Η έκφραση αυτή σημαίνει ότι όποτε πρέπει κάθε φορά που έρχεται αίτημα στη μνήμη, το μέγεθος του δεν πρέπει να είναι size_x (με αυτόν το συμβολισμό απεικονίζεται ένα αυθαίρετο μέγεθος που καθορίζει τον αριθμό των αποκομμένων bit και ορίζεται από τον χρήστη). Με άλλα λόγια όταν έχω xlw η block ram του byte 0 απενεργοποιείται. Όπως γίνεται αντιληπτό η παραπάνω λογική μπορεί να εφαρμοστεί μόνο όταν ο αριθμός των αποκομμένων bit είναι μεγαλύτερος ή ίσος του 8. Για μικρότερες τιμές γίνεται κανονικά η φόρτωση και το byte αποκόπτεται στην πορεία.Όταν ο αριθμός bit αποκοπής γίνει μεγαλύτερος ή ίσος με 16 bit μπορεί να εφαρμοστεί και στη block ram του byte 1 και όταν γίνει μεγαλύτερος ή ίσος με 24 στη block ram του byte 2. Στο byte 3 δεν μπορεί να εφαρμοστεί ποτέ αφού απαιτείται αριθμός bit αποκοπής μεγαλύτερος ή ίσος με 32 bit.

4. Στο στάδιο stall (ή memory όπως ονομάζεται από τον κατασκευαστή), στον πολυπλέκτη ο οποίος οδηγεί την τιμή που θα προωθηθεί στο στάδιο writeback για να εγγραφεί στο αρχείο καταχωρητών. Αυτή η τιμή μπορεί να είναι είτε το αποτέλεσμα της ALU αν πρόκειται για αριθμητική πράξη, είτε τα δεδομένα φόρτωσης από τη μνήμη σε περίπτωση εντολής ανάγνωσης. Αν πρόκειται για δεδομένα φόρτωσης τότε ένας δεύτερος πολυπλέκτης καθορίζει πόσα bit θα ληφθούν για προώθηση (8 για φόρτωση byte ,16 για φόρτωση ημιλέξης και 32 για φόρτωση λέξης) και αν θα γίνει απρόσημη ή προσημασμένη επέκταση σε 32 bit. Με κόκκινο στο παρακάτω σχήμα τονίζεται το σήμα που προστέθηκε και πραγματοποιεί αποκοπή των n κατώτερων bit. Η τεχνική αυτή θα ήταν πιο αποδοτική αν υπήρχε δυνατότητα φόρτωσης αυθαίρετου αριθμού bit από τη μνήμη. Στο ήδη υπάρχον σύστημα μνήμης δεδομένων που έχει υλοποιηθεί υποστηρίζονται 3 διακριτές περιπτώσεις φόρτωσης που είναι 8, 16 και 32 bit. Άρα αναμένεται σημαντικότερη μείωση καταναλισκόμενης ενέργειας για bit αποκοπής μεγαλύτερα από 8, 16 και 24.Για αριθμό bit αποκοπής 1-7, 8-15, 16-23 και 24-32, αναμένεται να έχω παρόμοια συμπεριφορά με την αποκοπή bit στην είσοδο της ALU εφόσον φορτώνεται το σύνολο των μπιτ και αποκόπτονται στην συνέχεια.



Εικόνα 48 : Προσθήκη εισόδου στον πολυπλέκτη αποκομμένων bit από φόρτωση από τη μνήμη

Κεφάλαιο 5

Προσομοίωση της διεπαφής AXI-PMicroX και μέτρηση της δυναμικής ισχύς στο Vivado 2018.3

5.1. Κατασκευή Testbench προσομοίωσης της διεπαφής AXI

Το σύστημα του potato processor προσομοιώθηκε στο Vivado 2018.3 με στόχο την εκτίμηση της δυναμικής ισχύος που καταναλώνει και την συμπεριφορά όταν εισάγονται προσεγγιστικά κυκλώματα στην αριθμητική μονάδα επεξεργασίας ALU και στο στάδιο της μνήμης. Για την εκτέλεση των προσομοιώσεων κατασκευάστηκε testbench το οποίο αρχικά διαβάζει από δεκαεξαδικό αρχείο τις εντολές και τις οδηγεί στη θύρα instr_data διατηρώντας ενεργό το σήμα valid_in_instr για όσο διάστημα γίνεται η αποστολή των εντολών. Στο τελευταίο δεδομένο ενεργοποιεί για ένα κύκλο το σήμα last_instr_signal. Στη συνέχεια διαβάζει το αρχείο των διευθύνσεων το οποίο περιλαμβάνει 5 τιμές και το οδηγεί στη θύρα data_addresses ενώ διατηρεί ενεργό το σήμα valid_in_data_addresses κατά το διάστημα αποστολής των διευθύνσεων.Για τον τελευταίο κύκλο αποστολής διευθύνσεων ενεργοποιεί το σήμα last_data_addresses_signal . Τέλος διαβάζει τα δεδομένα του προγράμματος από δεκαεξαδικό αρχείο και τα οδηγεί στη θύρα data_in διατηρώντας για όσο διάστημα γίνεται η μεταφορά τους το σήμα valid_in_data ενεργό. Στο τελευταίο δεδομένο και μόνο ενεργοποιεί το σήμα last_in_data_signal. Έπειτα αναμένει για όσο χρονικό διάστημα εκτελείται το πρόγραμμα και όταν ενεργοποιηθεί το σήμα valid_out_data (το οποίο καθορίζεται από την εσωτερική λογική που περιγράφηκε στο Κεφάλαιο 2) διαβάζει την έξοδο data_out σε ένα δεκαεξαδικό αρχείο αποτελεσμάτων.

5.2. Εξαγωγή εκτίμησης για την δυναμική ισχύ με χρήση του Vivado Power Report

Το εργαλείο power analysis του Vivado πραγματοποιεί εκτιμήσεις της καταναλισκόμενης ισχύος μετά από κάθε στάδιο της ροής σχηματισμού του bitstream(post-synthesis, post-placement, post-routing). Η πιο ακριβής εκτίμηση λαμβάνεται στο στάδιο post-route διότι σε αυτό το στάδιο διαθέτει πληροφορίες για την ακριβή θέση των λογικών μπλοκ που υλοποιούν το κύκλωμα καθώς και των διασυνδέσεων[36]. Παρέχει εναλλακτικές μορφές εμφάνισης του Power όπως σε ιεραρχικό επίπεδο, ανα τύπο πόρων, και ανα τομέα ρολογιού οι οποίες παρουσιάζονται συγκεντρωτικά στο εξαγόμενο αρχείο power report μορφής 'txt'.

5.2.1. Σύντομη ορολογία ισχύος στο FPGA

Στο εργαλείο του vivado για εκτίμηση της ισχύος χρησιμοποιούνται οι εξής όροι:

Device Static Power : Η στατική ισχύς της συσκευής είναι η ισχύς από τις απώλειες των τρανζίστορ σε όλα τα διασυνδεδεμένα μονοπάτια τάσης και τα κυκλώματα που απαιτούνται για να λειτουργήσει η συσκευή. Μετράται συνήθως με τη φόρτωση ενός κενού bitstream στη συσκευή και είναι συνάρτηση της λειτουργίας, της τάσης και της θερμοκρασίας.

Design Power : Είναι η δυναμική ισχύς του συστήματος που ορίζει ο χρήστης και εξαρτάται από την είσοδο δεδομένων και την εσωτερική δραστηριότητα του συστήματος. Εξαρτάται από τα επίπεδα τάσης και τους πόρους λογικής και δρομολόγησης που χρησιμοποιούνται.

Total On-Chip Power : Είναι η ισχύς που καταναλώνεται εσωτερικά του FPGA και ισούται με το άθροισμα της στατικής ισχύς της συσκευής και της δυναμικής ισχύς (αναφέρεται και ως θερμική ισχύς).

Off-Chip Power: Προκύπτει εξαιτίας της ροής από την πηγή της τροφοδοσίας στις ακίδες ισχύος του FPGA και έπειτα στα I/O και διαχέεται σε εξωτερικά στοιχεία της συσκευής. Το ρεύμα που παρέχεται από το FPGA

καταναλώνεται σε στοιχείο εκτός chip όπως τα τερματικά I/O, τα LED, ή τα I/O buffers και επομένως δεν ανεβάζουν τη θερμοκρασία σύνδεσης.

5.2.2. Παράγοντες που επηρεάζουν την ακρίβεια εκτίμησης του Power Report

Η υλοποίηση μιας ακριβούς εκτίμησης είναι αρκετά απαιτητική καθώς ο αλγόριθμος της εφαρμογής πρέπει να υποθέσει αυθαίρετα πολλούς παράγοντες και παραμέτρους. Γι'αυτό όσο μεγαλύτερη είναι η καθοδήγηση που παρέχεται στο εργαλείο από το χρήστη ώστε να ελαχιστοποιηθούν αυτές οι υποθέσεις, τόσο πιο ακριβής θα είναι η εκτίμηση ισχύος.

Οι ακόλουθοι παράγοντες λαμβάνονται υπόψη για μια ακριβή ανάλυση ισχύος:

- Θερμικές ρυθμίσεις
- Ρυθμίσεις τροφοδοσίας ισχύος
- Προδιαγραφές ρολογιού
- Σήματα ελέγχου
- Πρωταρχικές Είσοδοι
- Μεμονωμένα στοιχεία

Θερμικές ρυθμίσεις : Ιδανικά η στατική ισχύς είναι το άθροισμα της διαρροής ισχύος από την πηγή στον απαγωγό και στην πύλη του τρανζίστορ. Η στατική ισχύς εξαρτάται άμεσα από τις θερμικές συνθήκες οι οποίες απαιτείται να δοθούν από το χρήστη για μια έγκυρη εκτίμηση.

Ρυθμίσεις τροφοδοσίας ισχύος : Η κλιμάκωση τάσης αποτελεί διακεκριμένη τεχνική εξοικονόμησης ισχύος στα FPGA. Υπάρχουν διαφορετικά μονοπάτια τάσης που τροφοδοτούν διαφορετικές λογικές και κυκλώματα. Οι τιμές της τάσης αυτής ορίζονται από τον κατασκευαστή και περιορίζονται σε εύρη αποδεκτών τιμών μεταξύ των οποίων εγγυάται η σωστή λειτουργία της συσκευής. Προφανώς κάθε τιμή σε αυτό το εύρος προκαλεί διαφορετική κατανάλωση ισχύος επομένως πρέπει να δοθεί συγκεκριμένη τιμή για το σύστημα.

Προδιαγραφές ρολογίου : Τα ρολόγια του συστήματος είναι βασικό στοιχείο στον υπολογισμό της ισχύος. Αν δεν οριστούν ρολόγια, η εκτίμηση της διακοπτικής δραστηριότητας (switching activity) θα είναι ανακριβής οπότε και η εκτίμηση της ισχύος.

Σήματα ελέγχου : Τέτοια σήματα αποτελούν γενικευμένα και τοπικά reset (global and regional resets) και γενικευμένα σήματα επίτρεψης ρολογιού των οποίων η δραστηριότητα επηρεάζει δραματικά την εκτίμηση ισχύος.

Πρωταρχικές Είσοδοι : Οι κοινοί κόμβοι προσδιορίζονται από τους παραπάνω παράγοντες. Υπάρχουν όμως κόμβοι που αποτελούν διεπαφές με εξωτερικές συσκευές που πραγματοποιούν ανταλλαγή δεδομένων (π.χ. πρωτόκολλα, διεπαφή μνήμης, θύρες δεδομένων) και πρέπει να ληφθούν υπόψη.

Μεμονωμένα στοιχεία: Η μέτρηση της διακοπτικής δραστηριότητας των θεμελιωδών στοιχείων του συστήματος όπως οι BRAMs, τα GTs και τα DSP.Το εργαλείο διαθέτει διεπαφή καθορισμού της δραστηριότητας εξόδου για καταχωρητές, καταχωρητές ολίσθησης , LUTs, RAMs, BRAMs, DSPs, και GTs.

5.2.3. Καθορισμός της διακοπτικής δραστηριότητας για την ανάλυση ισχύος

Το εργαλείο αντιστοιχίζει τα σήματα του σχεδιασμού με ονόματα στη λίστα σημάτων της προσομοίωσης. Τα αποτελέσματα της προσομοίωσης είναι μια λίστα σημάτων σε μορφή αρχείου SAIF(Switching Activity Interchange Format) [36]. Λαμβάνοντας υπόψη τη διακοπτική δραστηριότητα όλων των σημάτων και πιθανοτικές προσεγγίσεις εξάγει την εκτίμηση. Αν τα αποτελέσματα των προσομοιώσεων έχουν εξαχθεί σε πρώιμο στάδιο της ροής όπως για παράδειγμα πριν το synthesis ή το placement και routing, τότε είναι προτιμότερο να ληφθεί από τα αποτελέσματα της προσομοίωσης μόνο η δραστηριότητα των θυρών εισόδου/εξόδου και να αφεθεί στο vectorless μοντέλο ο υπολογισμός της δραστηριότητας των εσωτερικών κόμβων. Γενικά υπάρχουν δύο είδη εκτιμήσεων, η πιθανοτική(vectorless estimation) και αυτή που βασίζεται σε αρχείο saif (vector based estimation). Όταν η δραστηριότητα των κόμβων δεν παρέχεται ούτε από το χρήστη ούτε από αποτελέσματα προσομοίωσης το εργαλείο εφαρμόζει ευριστικούς αλγόριθμους με χρήση πιθανοτήτων ώστε να προβλέψει αυτή τη δραστηριότητα. Ο αλγόριθμος είναι ικανός να προσεγγίσει τον βαθμό glitching για όλους τους κόμβους του σχεδιασμού. Το φαινόμενο glitching περιγράφει την συνεχή εναλλαγή καταστάσεων ενός σήματος ή στοιχείου στο διάστημα μεταξύ ενεργών ακμών ρολογιού μέχρι να σταθεροποιηθεί σε μια τελική τιμή. Είναι υπεύθυνο σε μεγάλο βαθμό για την κατανάλωση δυναμικής ισχύος. Οι λειτουργικές προσομοιώσεις δεν καταγράφουν τη δραστηριότητα glitching [37], επομένως για ακριβέστερη εκτίμηση της ισχύος τροφοδοτούνται στο εργαλείο τα αποτελέσματα της post implementation timing προσομοίωσης. Όταν η δραστηριότητα παρέχεται

στο εργαλείο power report μέσω αρχείου SAIF το οποίο εξάγεται από την post implementation timing simulation, τότε παρέχεται η ακριβής δραστηριότητα κάθε κόμβου συμπεριλαμβανομένης και της glitching δραστηριότητας. Είναι η ακριβέστερη εκτίμηση που μπορεί να ληφθεί από το εργαλείο. Η μεθοδολογία που ακολουθήθηκε ώστε να καταγραφεί η διακοπτική δραστηριότητα είναι η εξής:

Εκτελείται functional simulation η οποία είναι η ταχύτερη ώστε να προσδιοριστούν τα όρια στα οποία το σήμα pc_enable είναι ενεργό. Το σήμα αυτό υποδεικνύει το διάστημα εκτέλεσης της εφαρμογής επομένως η μέτρηση της διακοπτικής δραστηριότητας πρέπει να περιοριστεί σε αυτό. Οι τιμές που καταγράφονται είναι η χρονική στιγμή της θετικής ακμής στην οποία ξεκινά η εκτέλεση του προγράμματος και η χρονική στιγμή της αρνητικής ακμής όπου τερματίζει η εκτέλεση. Τίθεται χρονικό διάστημα εκτέλεσης της προσομοίωσης ίσο με τη χρονική στιγμή της εκκίνησης. Έπειτα εκτελείται η *post implementation timing simulation* και όταν ολοκληρωθεί εισάγονται οι ακόλουθες tcl εντολές στο τερματικό του Vivado :

```
open_saif "saif_file.saif"
log_saif [get_objects –r *]
run run_time us
```

Δημιουργείται ένα αρχείο SAIF και συνδέονται όλα τα σήματα του συστήματος σε αυτό ώστε να καταγράφονται. Έπειτα εκτελείται η προσομοίωση για όσο διάστημα το σήμα pc_enable είναι ενεργό. Στο τέλος της προσομοίωσης με την εντολή *close_saif* κλείνεται το αρχείο και χρησιμοποιείται ως είσοδος στο power report για την εξαγωγή του αρχείου για την ισχύ.

Κεφάλαιο 6

Αποτελέσματα και μετρήσεις

6.1. Επιλογή αλγορίθμων

Όπως αναφέρεται στο [38] οι αλγόριθμοι μηχανικής μάθησης ωθούνται στο επίπεδο ομίχλης ώστε να επεξεργάζονται IOT δεδομένα. Απλές ικανότητες μάθησης όπως η ομαδοποίηση και η ταξινόμηση μπορούν να χρησιμοποιηθούν για ποικίλους σκοπούς όπως εντοπισμό ασυνήθιστης δραστηριότητας δικτύου ή ταυτοποίηση IOT botnet. Συνεπώς ο βαθμός εξοικονόμηση ενέργειας σε εφαρμογές μηχανικής μάθησης που επιτυγχάνεται με προσεγγιστικές τεχνικές αποτελεί ενδιαφέρον πεδίο μελέτης. Στον επεξεργαστή potato που μελετάται οι μετρήσεις ισχύος καθώς και τα αποτελέσματα εκτέλεσης λαμβάνονται από προσομοίωση του Vivado, συνεπώς ο χρόνος εκτέλεσης έστω και ενός πολύ απλού αλγορίθμου μηχανικής μάθησης είναι απαγορευτικός. Γι' αυτό επιλέχθηκαν αλγόριθμοι-τμήματα ενός αλγορίθμου μηχανικής μάθησης οι οποίοι επαναλαμβάνονται πολλές φορές όπως για παράδειγμα FIR φίλτρο, συνέλιξλη 2D, και πολλαπλασιασμός πινάκων. Στο [39] δίνεται η βασική δομή ενός συνελικτικού νευρωνικού δικτύου η οποία αποτελείται από αρκετά διαφορετικά επίπεδα. Υπάρχουν 3 είδη επιπέδων, το συνελικτικό επίπεδο (convolutional layer), το στρώμα συγκέντρωσης(pooling layer) και το πλήρως συνδεδεμένο επίπεδο (fully-connected layer). Το συνελικτικό επίπεδο αποτελείται από αρκετούς χάρτες χαρακτηριστικών καθένας από τους οποίος συνελίσεται με ένα εκπαιδευμένο πυρήνα ώστε να αποκτηθεί ένα νέο χαρακτηριστικό.

6.2. Μετρικές λάθους

Ως μετρική λάθους επιλέχθηκε το MRE(Mean Relative Error) το οποία ορίζεται ως [40]:

$$MRE = \sum_{i=1}^{N} \frac{RE_i}{N} \text{ , όπου } RE_i = |(\text{actual} - \text{predicted})/\text{actual}|$$

Επίσης επιλέχθηκε το PSNR(Peak signal-to-noise ratio) το οποίο ορίζεται ως[41]:

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\
&= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE)
\end{aligned}
$$

όπου MAX_I η μέγιστη τιμή πίξελ της εικόνας (για grayscale εικόνα των 8 bit είναι 255) και MSE(mean squared error) που ορίζεται από τον τύπο

$$
MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2
$$

όπου m,n οι διαστάσεις της εικόνας, I(i,j) η τιμή του ακριβούς πίξελ και K(i,j) η τιμή του αντίστοιχου προσεγγιστικού πίξελ.

6.3. Αποτελέσματα

Μέθοδος προσεγγιστικών αριθμητικών κυκλωμάτων

Πραγματοποιήθηκαν πειράματα με υποστήριξη των νέων εντολών xmul και xadd με προσεγγιστικά κυκλώματα από τις βιβλιοθήκες ανοιχτού κώδικα EvoApprox8b, SMApproxLib, DeMas και lpACLib.

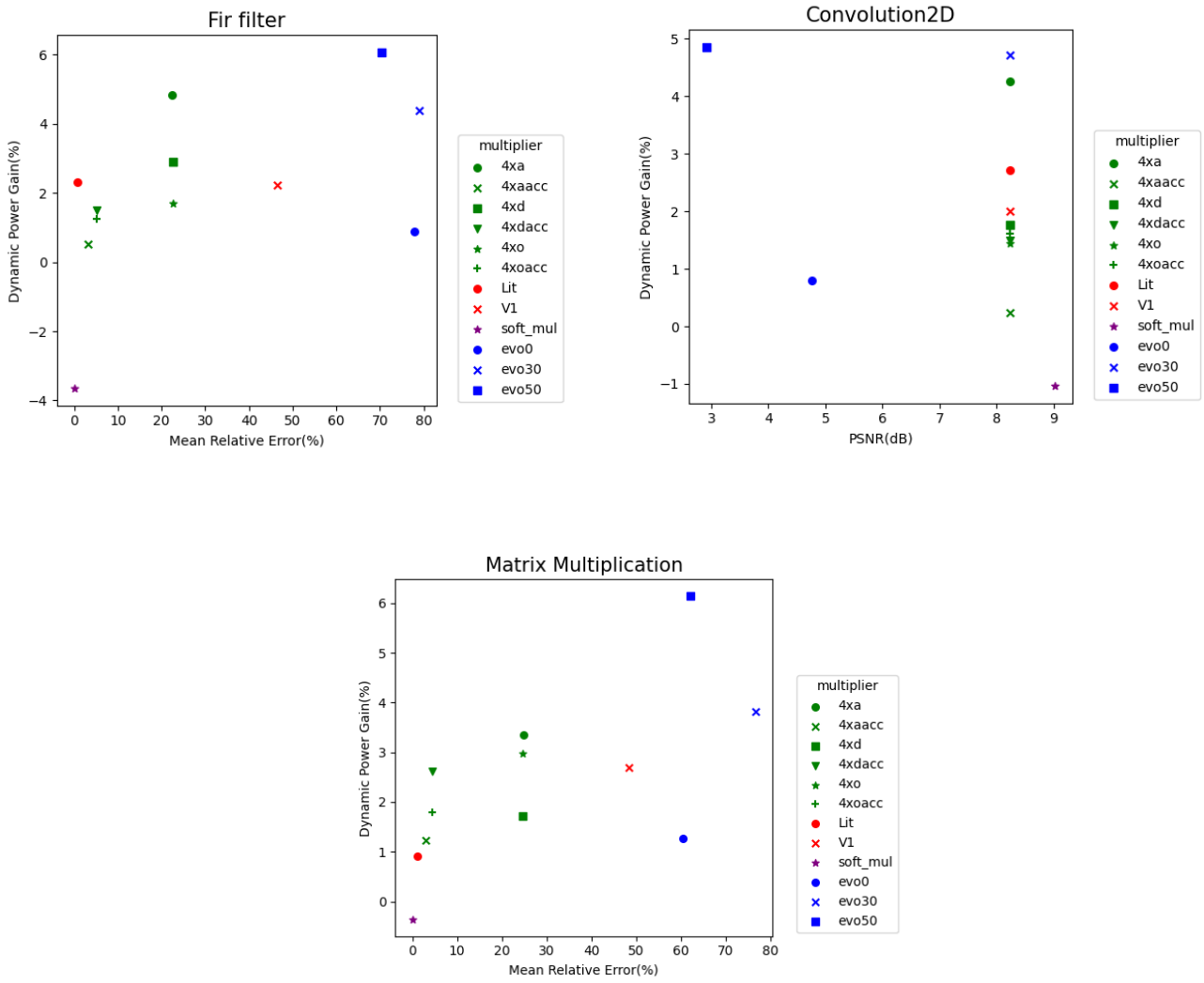
Προσεγγιστικοί πολλαπλασιαστές

Χρησιμοποιήθηκαν 6 προσεγγιστικοί πολλαπλασιαστές από τη βιβλιοθήκη SMApproxLib οι οποίοι κατασκευάστηκαν με τη βοήθεια του παρεχόμενου από τον κατασκευαστή python script(4xa, 4xaacc, 4xd, 4xdacc, 4xo, 4xoacc), 2 προσεγγιστικού πολλαπλασιαστές από τη βιβλιοθήκη IpACLib(Lit, V1), 3 προσεγγιστικοί πολλαπλασιαστές από τη βιβλιοθήκη EvoApprox8b(evo000, evo030, evo050) και τέλος ο προσεγγιστικός πολλαπλασιαστής προσαρμοσμένος για λειτουργία σε softcore επεξεργαστές.

fir filter				
Approximate multiplier	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
4xa	44.941	42.776	4.82	22.460
4xaacc	53.293	53.011	0.53	3.205
4xd	49.574	48.135	2.90	22.527
4xdacc	52.403	51.622	1.49	5.089
4xo	51.123	50.254	1.70	22.651
4xoacc	51.249	50.608	1.25	5.233
Lit	50.253	49.091	2.31	0.825
V1	54.215	53.00	2.24	46.508
softcore mul	55.259	57.284	-3.66	0.005
evo0	51.828	51.369	0.89	77.939
evo30	48.206	46.094	4.38	79.060
evo50	54.220	50.930	6.07	70.446

Convolution2D				
Approximate multiplier	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	PSNR(dB)
4xa	44.836	42.925	4.27	8.23
4xaacc	52.974	52.845	0.24	8.23
4xd	49.328	48.461	1.76	8.23
4xdacc	52.198	51.422	1.49	8.23
4xo	50.847	50.111	1.45	8.23
4xoacc	49.936	49.132	1.61	8.23
Lit	49.643	48.295	2.72	8.23
V1	53.944	52.863	2.00	8.23
softcore mul	54.727	55.295	-1.04	9.03
evo0	50.421	50.019	0.80	4.77
evo30	47.821	45.563	4.72	8.23
evo50	54.107	51.479	4.86	2.91

Matrix multiplication				
Approximate multiplier	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
4xa	43.560	42.102	3.35	24.773
4xaacc	53.823	53.159	1.23	2.973
4xd	49.199	48.354	1.72	24.706
4xdacc	50.671	49.348	2.61	4.494
4xo	48.986	47.530	2.97	24.724
4xoacc	49.188	48.303	1.80	4.431
Lit	47.394	46.964	0.91	1.063
V1	48.463	47.155	2.70	48.449
softcore mul	52.153	52.346	-0.37	0.004
evo0	49.538	48.910	1.27	60.553
evo30	47.166	45.364	3.82	76.688
evo50	51.495	48.327	6.15	62.061



Προσεγγιστικοί αθροιστές

Εδώ χρησιμοποιήθηκαν 3 προσεγγιστικοί αθροιστές της βιβλιοθήκης DeMas (AA1, AA3, AA5) και 1 προσεγγιστικός αθροιστής της βιβλιοθήκης IpACLib(IMPACTFirst). Όλοι οι παραπάνω αθροιστές διέθεταν επιλογή αριθμού προσεγγιστικών bit συνεπώς τα πειράματα έγιναν για τους αριθμούς 4,8,12,16,20,24,28 προσεγγιστικών bit.

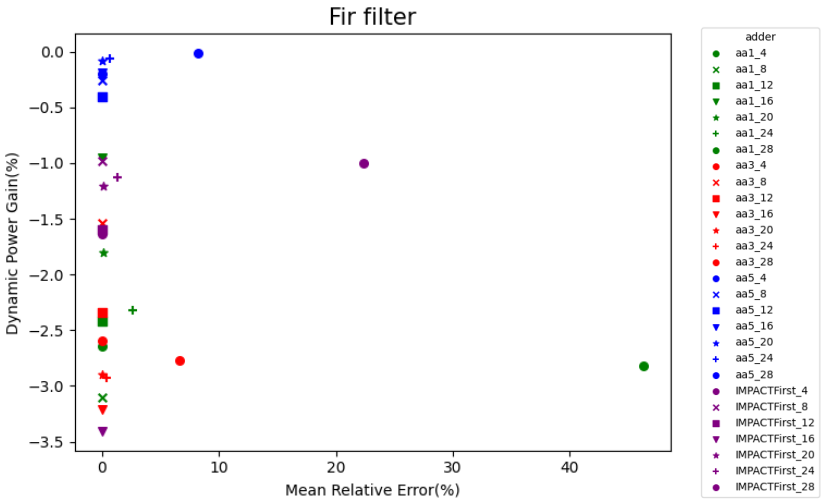
fir filter				
Approximate adders	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
aa1_4	52.372	53.753	-2.64	0.000
aa1_8	50.912	52.488	-3.10	0.000
aa1_12	51.547	52.792	-2.42	0.001
aa1_16	52.603	53.104	-0.95	0.010
aa1_20	51.382	52.309	-1.80	0.161

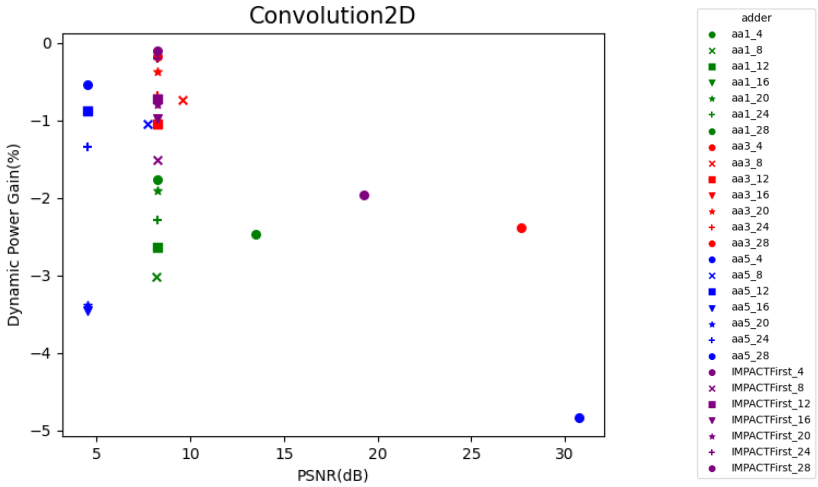
aa1_24	51.722	52.923	-2.32	2.646
aa1_28	51.340	52.788	-2.82	46.318
aa3_4	49.125	50.399	-2.59	0.000
aa3_8	49.731	50.498	-1.54	0.000
aa3_12	48.593	49.732	-2.34	0.000
aa3_16	49.054	50.630	-3.21	0.002
aa3_20	49.352	50.783	-2.90	0.024
aa3_24	48.179	49.591	-2.93	0.411
aa3_28	48.915	50.271	-2.77	6.650
aa5_4	18.383	18.419	-0.20	0.000
aa5_8	17.805	17.852	-0.26	0.000
aa5_12	17.951	18.023	-0.40	0.000
aa5_16	16.881	17.915	-0.19	0.003
aa5_20	18.426	18.441	-0.08	0.039
aa5_24	17.338	17.348	-0.06	0.650
aa5_28	17.947	17.949	-0.01	8.187
impactfirst_4	52.293	53.152	-1.64	0.000
impactfirst_8	53.943	54.474	-0.98	0.000
impactfirst_12	53.752	54.612	-1.60	0.000
impactfirst_16	54.148	55.996	-3.41	0.005
impactfirst_20	53.754	54.402	-1.21	0.077
impactfirst_24	53.645	53.344	-1.13	1.323
impactfirst_28	53.872	54.413	-1.00	22.395

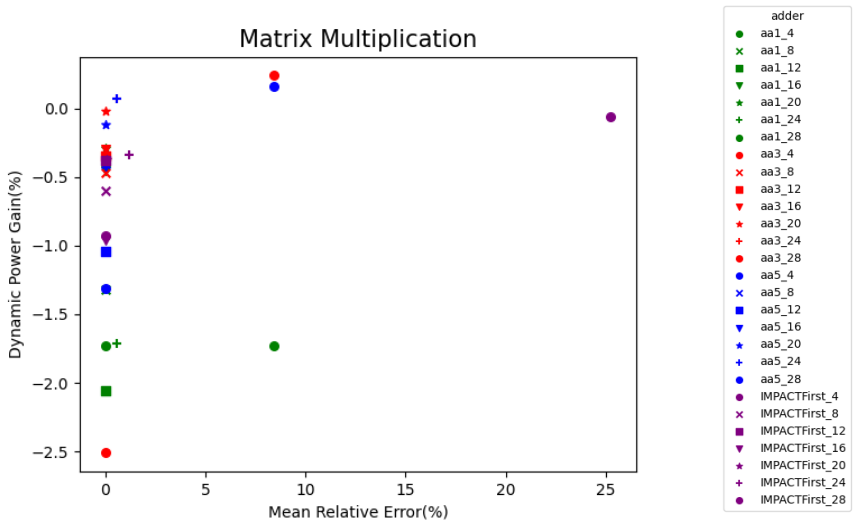
Convolution2D				
Approximate adders	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	PSNR(dB)
aa1_4	53.863	55.192	-2.47	13.49
aa1_8	53.359	54.973	-3.02	8.20
aa1_12	52.987	54.386	-2.64	8.23
aa1_16	52.639	54.027	-2.64	8.23
aa1_20	52.379	53.376	-1.90	8.23
aa1_24	51.768	52.951	-2.29	8.23
aa1_28	50.371	51.258	-1.76	8.23
aa3_4	48.452	49.612	-2.39	27.67
aa3_8	49.674	50.044	-0.74	9.58
aa3_12	48.591	49.103	-1.05	8.23
aa3_16	49.105	49.489	-0.78	8.23
aa3_20	49.673	49.857	-0.37	8.23
aa3_24	48.009	48.337	-0.68	8.23
aa3_28	47.911	47.993	-0.17	8.23
aa5_4	19.521	20.465	-4.84	30.78
aa5_8	18.439	18.632	-1.05	7.70
aa5_12	17.858	18.015	-0.88	4.48
aa5_16	18.146	18.773	-3.46	4.48
aa5_20	18.031	18.640	-3.38	4.48
aa5_24	19.338	19.599	-1.35	4.48
aa5_28	17.254	17.348	-0.54	4.48
impactfirst_4	53.425	54.471	-1.96	19.28
impactfirst_8	53.214	54.015	-1.51	8.23

impactfirst_12	54.521	54.914	-0.72	8.23
impactfirst_16	54.748	55.286	-0.98	8.23
impactfirst_20	53.255	53.681	-0.80	8.23
impactfirst_24	53.496	53.604	-0.20	8.23
impactfirst_28	53.851	53.912	-0.11	8.23

matrix multiplication				
Approximate adders	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
aa1_4	52.815	53.728	-1.73	0.000
aa1_8	54.258	54.973	-1.32	0.000
aa1_12	53.380	54.482	-2.06	0.000
aa1_16	52.720	52.957	-0.45	0.002
aa1_20	52.338	52.490	-0.29	0.045
aa1_24	51.537	52.420	-1.71	0.590
aa1_28	50.457	51.329	-1.73	8.444
aa3_4	48.326	49.537	-2.51	0.000
aa3_8	50.636	50.873	-0.47	0.000
aa3_12	48.823	48.995	-0.35	0.000
aa3_16	49.501	49.651	-0.30	0.002
aa3_20	49.582	49.590	-0.02	0.045
aa3_24	48.231	48.195	0.07	0.590
aa3_28	48.443	48.328	0.24	8.444
aa5_4	50.236	50.893	-1.31	0.000
aa5_8	51.750	51.958	-0.40	0.000
aa5_12	52.427	52.972	-1.04	0.000
aa5_16	50.531	50.741	-0.42	0.002
aa5_20	52.400	52.463	-0.12	0.045
aa5_24	51.135	51.098	0.07	0.599
aa5_28	52.665	52.579	0.16	8.444
impactfirst_4	53.517	54.013	-0.93	0.000
impactfirst_8	53.276	53.594	-0.60	0.000
impactfirst_12	54.383	54.587	-0.38	0.000
impactfirst_16	54.851	55.379	-0.96	0.004
impactfirst_20	53.578	53.776	-0.37	0.068
impactfirst_24	53.920	54.103	-0.34	1.174
impactfirst_28	53.583	53.617	-0.06	25.256


Fir filter
Dynamic Power Gain(%)
Mean Relative Error(%)
adder
aa1_4
aa1_8
aa1_12
aa1_16
aa1_20
aa1_24
aa1_28
aa3_4
aa3_8
aa3_12
aa3_16
aa3_20
aa3_24
aa3_28
aa5_4
aa5_8
aa5_12
aa5_16
aa5_20
aa5_24
aa5_28
IMPACTFirst_4
IMPACTFirst_8
IMPACTFirst_12
IMPACTFirst_16
IMPACTFirst_20
IMPACTFirst_24
IMPACTFirst_28


Convolution2D
Dynamic Power Gain(%)
PSNR(dB)
adder
aa1_4
aa1_8
aa1_12
aa1_16
aa1_20
aa1_24
aa1_28
aa3_4
aa3_8
aa3_12
aa3_16
aa3_20
aa3_24
aa3_28
aa5_4
aa5_8
aa5_12
aa5_16
aa5_20
aa5_24
aa5_28
IMPACTFirst_4
IMPACTFirst_8
IMPACTFirst_12
IMPACTFirst_16
IMPACTFirst_20
IMPACTFirst_24
IMPACTFirst_28


Matrix Multiplication
Dynamic Power Gain(%)
Mean Relative Error(%)
adder
aa1_4
aa1_8
aa1_12
aa1_16
aa1_20
aa1_24
aa1_28
aa3_4
aa3_8
aa3_12
aa3_16
aa3_20
aa3_24
aa3_28
aa5_4
aa5_8
aa5_12
aa5_16
aa5_20
aa5_24
aa5_28
IMPACTFirst_4
IMPACTFirst_8
IMPACTFirst_12
IMPACTFirst_16
IMPACTFirst_20
IMPACTFirst_24
IMPACTFirst_28

Μέθοδος αποκοπής bit

Χαρακτηριστικά συστήματος

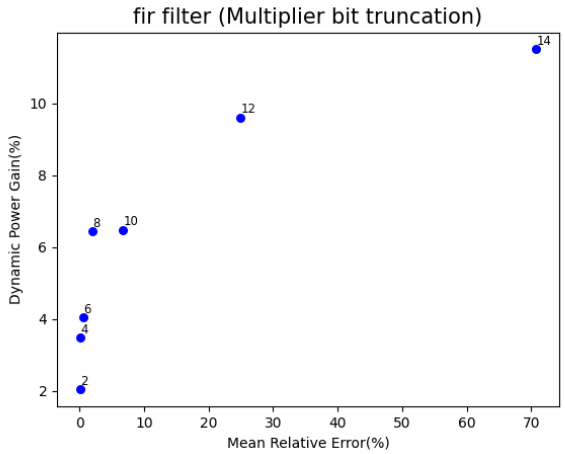
Μέγεθος μνήμης εντολών : 2 kB, μέγεθος μνήμης δεδομένων : 16 kB
Υλοποίηση αριθμητικής μονάδας ALU σε LUTs
Συχνότητα μετρήσεων : 50 MHz

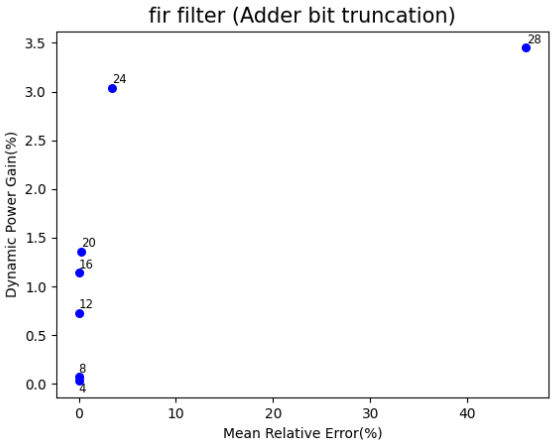
fir filter				
# of truncated MUL bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
2	45.469	44.539	2.04	0.023
4	44.079	42.539	3.49	0.108
6	43.292	41.538	4.05	0.531
8	44.214	41.364	6.45	2.046
10	41.318	38.640	6.48	6.751
12	43.895	39.679	9.61	24.880
14	43.054	38.105	11.49	70.823

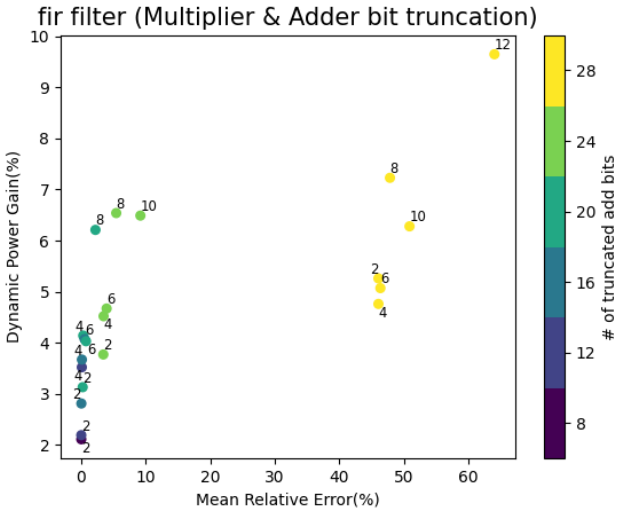
fir filter				
# of truncated ADD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
4	42.352	42.340	0.03	0.000
8	40.501	40.473	0.07	0.000
12	43.857	43.536	0.73	0.001
16	41.583	41.107	1.14	0.014
20	42.580	42.002	1.36	0.225
24	41.951	40.677	3.04	3.405
28	40.549	39.152	3.45	46.052

fir filter				
# of truncated MUL & ADD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
2_8	40.532	39.675	2.11	0.024
2_12	41.502	40.592	2.19	0.024
2_16	42.841	41.636	2.81	0.037
2_20	39.942	38.692	3.13	0.243
2_24	41.073	39.524	3.77	3.438
2_28	42.580	40.342	5.26	46.052
4_12	44.651	43.079	3.52	0.109
4_16	43.344	41.753	3.67	0.121
4_20	42.832	41.057	4.14	0.326
4_24	41.520	39.643	4.52	3.480
4_28	42.852	40.811	4.76	46.052
6_16	41.773	40.072	4.07	0.544
6_20	43.853	42.084	4.03	0.756

6_24	40.873	38.963	4.67	3.949
6_28	42.748	39.631	5.07	46.357
8_20	41.792	39.195	6.21	2.219
8_24	43.820	40.953	6.54	5.438
8_28	41.957	38.925	7.23	47.840
10_24	43.639	40.806	6.49	9.183
10_28	41.074	38.493	6.28	50.871
12_28	42.819	38.685	9.65	64.019



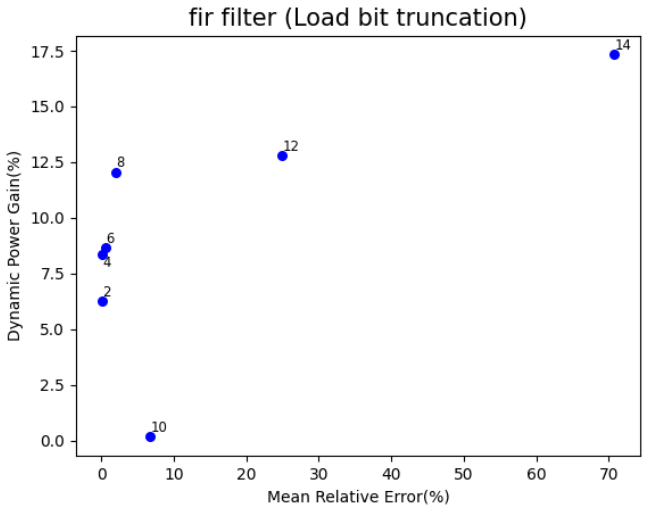




Οι αριθμοί στις κουκίδες υποδεικνύουν των αριθμό των αποκομμένων bit πολλαπλασιαστή και ο χρωματισμός τους των αριθμό των αποκομμένων bit αθροιστή

fir filter				
# of truncated LOAD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
2	41.733	39.124	6.25	0.023
4	40.272	36.902	8.37	0.108
6	42.277	38.621	8.65	0.531
8	43.116	37.917	12.06	2.046
8 (mem)	41.856	37.381	10.69	2.046
10	7.944	7.930	0.17	6.751
10 (mem)	46.119	40.822	11.48	6.751
12	39.990	34.868	12.81	24.880
12(mem)	42.495	37.395	12.00	24.880
14	45.278	37.432	17.33	70.823
14(mem)	45.074	38.868	13.77	70.823

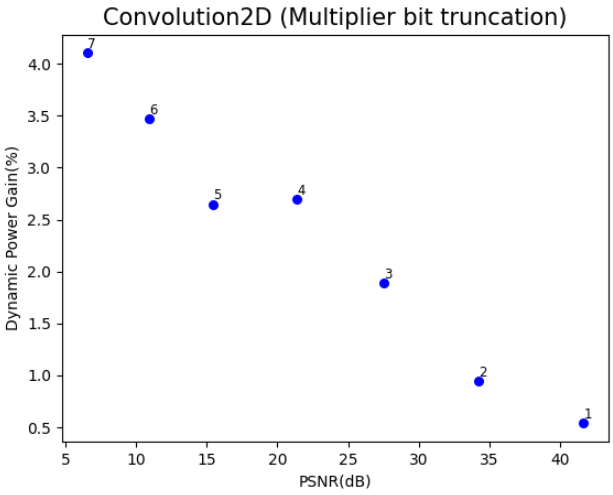
Σημείωση : Οι τιμές που τονίζονται με μπλε χρώμα (mem) δηλώνουν τη χρήση λογικής στο σήμα επίτρεψης της block ram του byte 0 ώστε να μην πραγματοποιεί φόρτωση δεδομένου. Όπως φαίνεται μόνο για αριθμό αποκομμένων bit μεγαλύτερο ή ίσο του 8 είναι δυνατόν να εφαρμοστεί αυτή η λογική. Όλες οι υπόλοιπες τιμές δηλώνουν ότι σε περίπτωση εντολής lw θα γίνεται η φόρτωση και μεταγενέστερα θα γίνεται η αποκοπή bit.

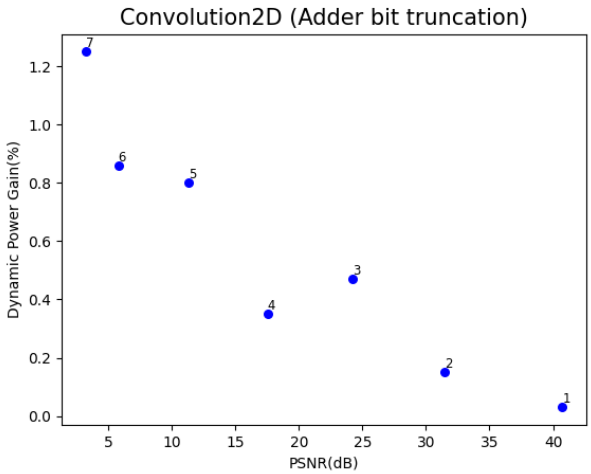


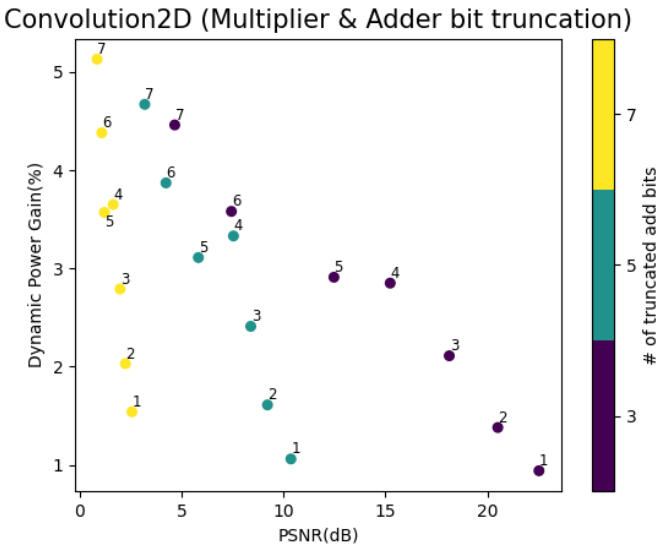
convolution2D				
# of truncated MUL bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	PSNR(dB)
1	37.425	37.223	0.54	41.67
2	38.338	37.976	0.95	34.25
3	37.147	36.444	1.89	27.53
4	36.896	35.901	2.69	21.39
5	37.175	36.190	2.65	15.45
6	40.008	38.621	3.47	10.96
7	38.880	37.284	4.10	6.53

convolution2D				
# of truncated ADD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	PSNR(dB)
1	37.652	37.640	0.03	40.74
2	37.537	37.481	0.15	31.43
3	38.473	38.293	0.47	24.25
4	37.715	37.584	0.35	17.53
5	38.825	38.514	0.80	11.30
6	40.534	40.185	0.86	5.79
7	38.527	38.047	1.25	3.21

convolution2D				
# of truncated MUL & ADD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	PSNR(dB)
1_3	41.345	40.956	0.94	22.53
1_5	40.525	40.094	1.06	10.36
1_7	42.503	41.848	1.54	2.56
2_3	42.852	42.262	1.38	20.51
2_5	40.591	39.936	1.61	9.21
2_7	38.938	38.147	2.03	2.24
3_3	39.741	38.901	2.11	18.13
3_5	38.684	37.753	2.41	8.39
3_7	42.974	41.773	2.79	1.98
4_3	43.020	41.794	2.85	15.23
4_5	42.677	41.256	3.33	7.54
4_7	39.819	38.365	3.65	1.64
5_3	40.551	39.372	2.91	12.47
5_5	43.094	41.753	3.11	5.82
5_7	41.419	39.942	3.57	1.21
6_3	42.631	41.104	3.58	7.44
6_5	40.588	39.017	3.87	4.23
6_7	40.052	38.297	4.38	1.08
7_3	38.247	36.541	4.46	4.66
7_5	40.885	38.974	4.67	3.19
7_7	41.376	39.252	5.13	0.85

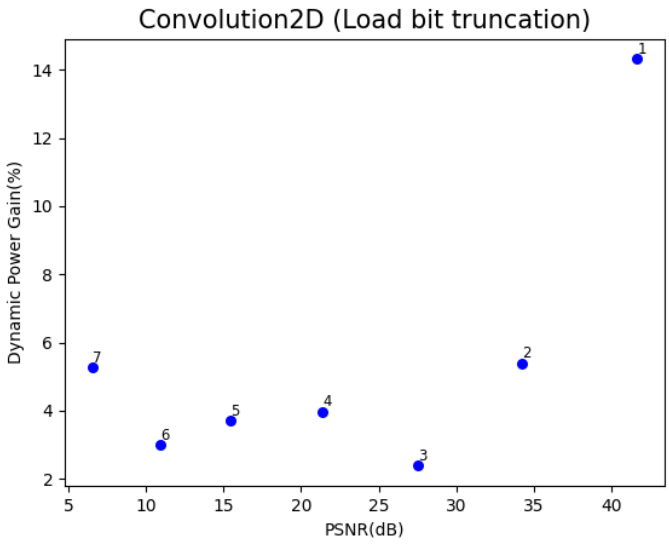






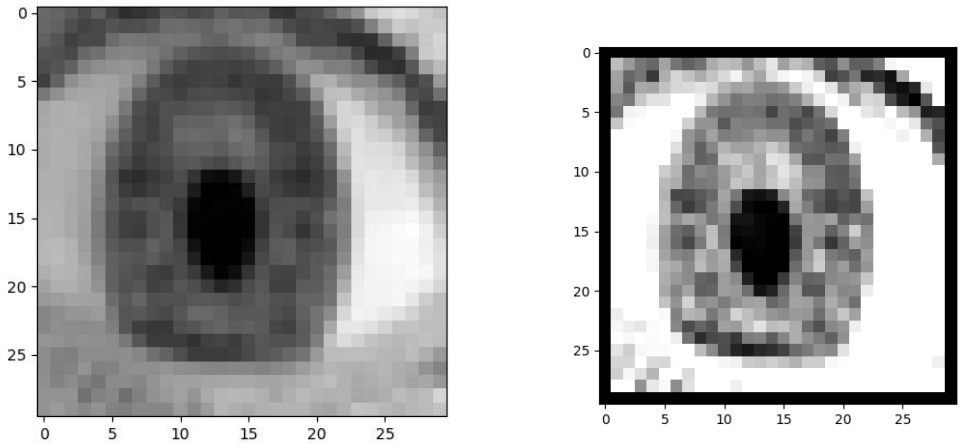
Οι αριθμοί στις κουκίδες υποδεικνύουν των αριθμό των αποκομμένων bit πολλαπλασιαστή και ο χρωματισμός τους των αριθμό των αποκομμένων bit αθροιστή

convolution2D				
# of truncated LOAD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	PSNR(dB)
1	34.712	29.744	14.31	41.67
2	31.367	29.677	5.39	34.25
3	37.671	36.774	2.38	27.53
4	37.013	35.544	3.97	21.39
5	38.520	37.091	3.71	15.45
6	34.943	33.898	2.99	10.96
7	37.668	35.688	5.26	6.53

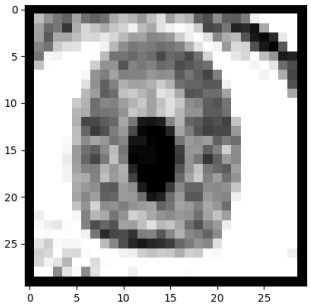


Ακολουθεί εφαρμογή του convolution2D σε grayscale εικόνα 30x30. Σε αυτή εφαρμόζεται πυρήνας 3x3 για sharpen.

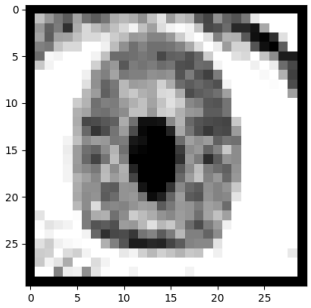
$$\text{Kernel} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \Rightarrow \text{sharpen}$$



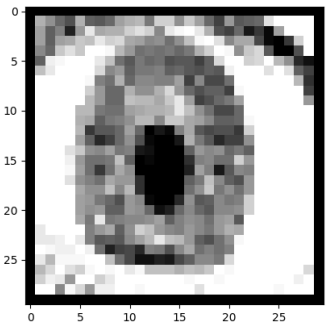
Εικόνα 49 : Πρωτότυπη εικόνα (αριστερά) και εικόνα μετά την εφαρμογή convolution2d με πυρήνα sharpen(δεξιά)



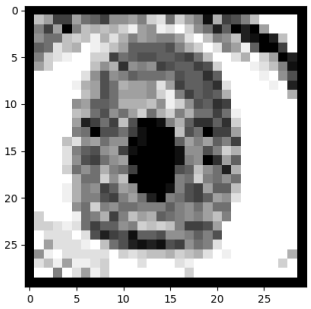
1 bit(41.67 dB)



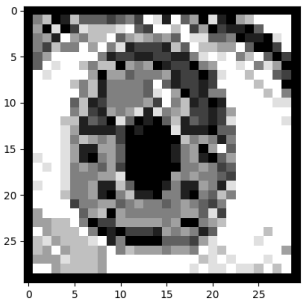
2 bit(34.25 dB)



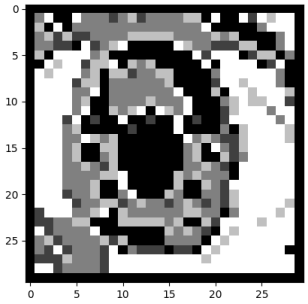
3 bit(27.53 dB)



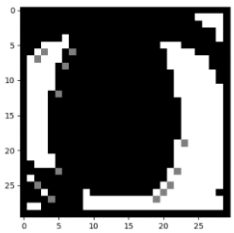
4 bit(21.39 dB)



5 bit(15.45 dB)



6 bit(10.96 dB)

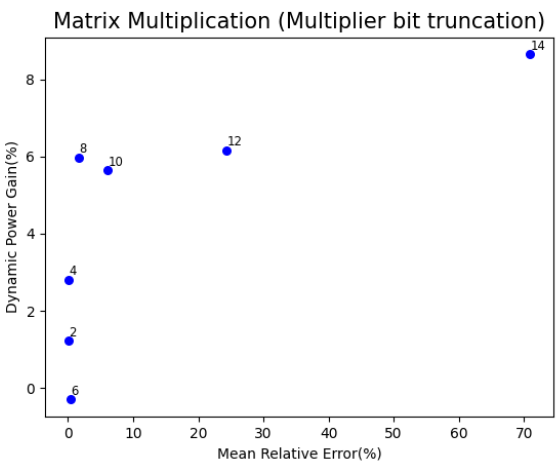


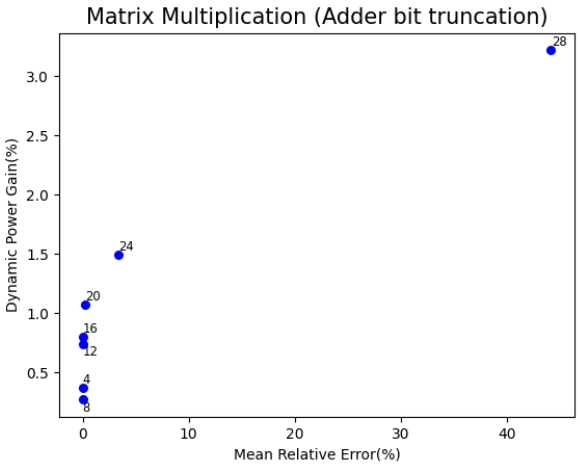
7 bit(6.53 dB)

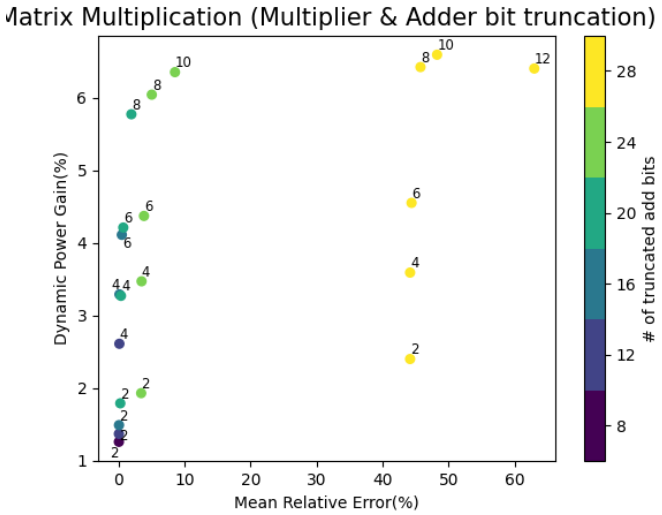
matrix multiplication				
# of truncated MUL bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
2	47.440	46.856	1.23	0.020
4	45.096	43.828	2.81	0.105
6	26.383	26.461	-0.30	0.471
8	45.868	43.127	5.98	1.708
10	45.177	42.628	5.64	6.143
12	45.162	42.381	6.16	24.379
14	46.606	42.578	8.64	70.969

matrix multiplication				
# of truncated ADD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
4	44.355	44.189	0.37	0.000
8	44.873	44.754	0.27	0.000
12	43.946	43.619	0.74	0.001
16	43.815	43.463	0.80	0.013
20	42.502	42.046	1.07	0.233
24	41.812	41.187	1.49	3.367
28	41.524	40.185	3.22	44.156

matrix multiplication				
# of truncated MUL & ADD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
2_8	43.842	43.289	1.26	0.038
2_12	42.690	42.106	1.37	0.041
2_16	42.294	41.663	1.49	0.056
2_20	44.531	43.734	1.79	0.251
2_24	41.671	40.867	1.93	3.413
2_28	41.864	40.861	2.40	44.156
4_12	43.713	42.573	2.61	0.111
4_16	43.572	42.139	3.29	0.125
4_20	42.991	41.586	3.27	0.350
4_24	42.647	41.166	3.47	3.474
4_28	43.853	42.280	3.59	44.156
6_16	44.479	42.650	4.11	0.486
6_20	42.382	40.598	4.21	0.704
6_24	42.721	40.856	4.37	3.841
6_28	41.645	39.752	4.55	44.377
8_20	42.738	40.274	5.77	1.928
8_24	42.456	39.892	6.04	5.024
8_28	40.622	38.013	6.42	45.730
10_24	39.705	37.185	6.35	8.537
10_28	43.039	40.201	6.59	48.243
12_28	44.427	41.583	6.40	62.972

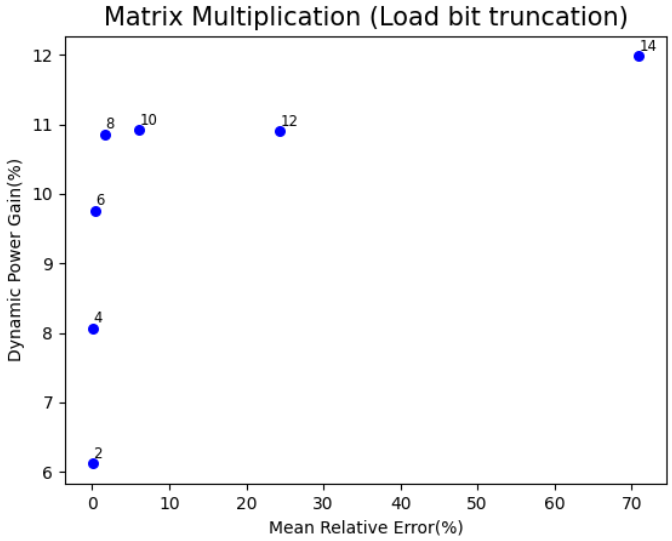






Οι αριθμοί στις κουκίδες υποδεικνύουν των αριθμό των αποκομμένων bit πολλαπλασιαστή και ο χρωματισμός τους των αριθμό των αποκομμένων bit αθροιστή

matrix multiplication				
# of truncated LOAD bits	Ref. Dynamic Power(mW)	Dynamic Power(mW)	Power Gain(%)	MRE(%)
2	41.384	38.850	6.12	0.020
4	42.053	38.658	8.07	0.105
6	41.549	37.493	9.76	0.471
8	40.585	36.177	10.86	1.708
10	40.956	36.485	10.92	6.143
12	45.133	40.210	10.91	24.379
14	43.371	38.175	11.98	70.969



6.4. Ανάλυση πειραματικών αποτελεσμάτων

Όσον αφορά τις μετρήσεις με χρήση προσεγγιστικών πολλαπλασιαστών από τις βιβλιοθήκες ανοιχτού κώδικα που αναφέρθηκαν παραπάνω τα σημεία Pareto προκύπτουν:

fir filter
evo50 *(κέρδος δυναμικής ισχύος: 6.07%* , mre: 60.446%)
4xa*(κέρδος δυναμικής ισχύος: 4.82%* , mre: 22.460%)
Lit*(κέρδος δυναμικής ισχύος: 2.31%* , mre: 0.825%)
Dresden softcore multiplier*(κέρδος δυναμικής ισχύος: -3.66%* , mre: 0.05%*)*

covvolution2D(grayscale)
evo50 *(κέρδος δυναμικής ισχύος: 4.86%* , PSNR: 2.91dB)
evo30 *(κέρδος δυναμικής ισχύος: 4.72%* , PSNR: 8.23dB)
Dresden softcore multiplier *(κέρδος δυναμικής ισχύος: -1.04%* , PSNR: 9.03)

matrix multiplication
evo50 *(κέρδος δυναμικής ισχύος: 6.15%* , mre: 62.061%)
4xa (*κέρδος δυναμικής ισχύος: 3.35%* , mre: 24.773%)
4xo (*κέρδος δυναμικής ισχύος: 2.97%* , mre: 24.724%)
4xdacc (*κέρδος δυναμικής ισχύος: 2.61%* , mre: 4.494%)
4xoacc (*κέρδος δυναμικής ισχύος: 1.80%* , mre: 4.431%)
4xaacc (*κέρδος δυναμικής ισχύος: 1.23%* , mre: 2.973%)
Lit*(κέρδος δυναμικής ισχύος: 0.91%* , mre: 1.063%)
Dresden softcore multiplier *(κέρδος δυναμικής ισχύος: 00.37%* , mre: 0.04%*)*

Άρα με την εφαρμογή προσεγγιστικών πολλαπλασιαστών επιτεύχθηκε εως και 6.07% μείωση ισχύος σε fir filter, εως και 4.86% σε convolution2D και εως 6.15% σε matrix multiplication. Παρατηρείται εντούτοις πως τα αντίστοιχα λάθη είναι αρκετά υψηλά.

Η χρήση αποκλειστικά προσεγγιστικών αθροιστών από τις βιβλιοθήκες DeMas και IpaCLib οδήγησε σε αρνητικά κέρδη δηλαδή αύξηση της κατανάλωσης δυναμικής ισχύος επομένως η χρήση τους καθίσταται ασύμφορη.

Όσον αφορά την τεχνική αποκοπής bit παρατηρείται σχεδόν σε όλα τα διαγράμματα πως η συνάρτηση κέρδους δυναμικής ισχύος – αριθμού αποκομμένων bit δεν είναι αύξουσα. Δηλαδή παρατηρείται το φαινόμενο μικρότεροι αριθμοί αποκομμένων bit να προκαλούν μεγαλύτερο κέρδος δυναμικής ισχύος σε σχέση με μεγαλύτερους αριθμούς. Η εξήγηση βρίσκεται στη φύση του fpga και συγκεκριμένα στις διαδικασίες placement και routing. Το εργαλείο vivado δίνει τη δυνατότητα παρατήρησης των πόρων που χρησιμοποιούνται μετά από κάθε εκτέλεση του implementation (χρωματίζει τα CLBs που χρησιμοποιούνται δίνοντας μια χωρική απεικόνιση του design στο fpga). Ο ίδιος κώδικας περιγραφής υλικού οδηγεί κάθε φορά σε διαφορετική τοποθέτηση των λογικών μπλοκ κατα το implement, η οποία καθορίζεται από τους πολύπλοκους αλγορίθμους place και route. Συνεπώς ανάλογα με την τοποθέτηση των στοιχείων κάθε φορά η μέθοδος αποκοπής bit μπορεί να έχει διαφορετική συμπεριφορά συνεισφέροντας λιγότερο ή περισσότερο στην εξοικονόμηση δυναμικής ισχύος της συσκευής. Όπως φαίνεται μπορεί να οδηγήσει ακόμα και σε αρνητικά κέρδη, δηλαδή η ισχύς εκτέλεσης προσεγγιστικής πράξης να υπερβαίνει αυτή της ακριβούς. Εντούτοις τα πειράματα δίνουν πληροφορίες για την τάξη μεγέθους του κέρδους δυναμικής ισχύος.

Ως προς την αποτελεσματικότητα αποκοπής bit:

στο **fir filter**: μείωση της δυναμικής ισχύος έως και 11.5% από την αποκοπή bit αποκλειστικά στον πολλαπλασιαστή και μείωση έως και 3.45% από την αποκοπή bit αποκλειστικά στον αθροιστή. Ο συνδυασμός αποκοπής bit τόσο σε πολλαπλασιαστή όσο και αθροιστή (όπως αναμενόταν για εφαρμογές πολλαπλασιασμού και συσσώρευσης (MAC)) συνετέλεσε κυρίως στη συμπλήρωση του φάσματος με περισσότερα σημεία κέρδους ισχύος-λάθους. Δηλαδή στο διάστημα 0-11.5% κέρδους υπάρχουν περισσότερα σημεία με διαφορετικά mre. Η μέθοδος προσεγγιστικής εντολής φόρτωσης (αποκοπής bit από τη μνήμη) συνετέλεσε σε εως και 17.3% μείωση δυναμικής ισχύος.

στο **convolution2D**: μείωση της δυναμικής ισχύος έως και 4.1% από την αποκοπή μόνο bit αποκλειστικά στον πολλαπλασιαστή και μείωση έως και 1.25% από την αποκοπή bit αποκλειστικά στον αθροιστή. Ο συνδυασμός αποκοπής bit σε πολλαπλασιαστή και αθροιστή συνετέλεσε σε μικρή περαιτέρω μείωση της δυναμικής ισχύς (5.13%). Η αποκοπή bit από τη μνήμη συνετέλεσε σε εως και 5.39% μείωση δυναμικής ισχύος.(Παρατηρείται ότι για

αποκοπή 1 bit αντιστοιχεί μείωση 14.31% το οποίο θεωρείται ιδιάζουσα περίπτωση εξαιτίας της τοποθέτησης των στοιχείων στο FPGA.)

στο **matrix multiplication**: μείωση της δυναμικής ισχύος έως και 8.64% από την αποκοπή μόνο bit αποκλειστικά στον πολλαπλασιαστή και μείωση έως και 3.22% από την αποκοπή bit αποκλειστικά στον αθροιστή. Η αποκοπή bit από τη μνήμη συνετέλεσε σε εως και 12% μείωση δυναμικής ισχύος.

Γενικά παρατηρείται μεγαλύτερη διακύμανση στις τιμές των κερδών ισχύος σε σχέση με τις αντίστοιχες από την αποκοπή bit στον πολλαπλασιαστή αλλά και μεγαλύτερος μέσος όρος κέρδους.

6.5. Μελλοντική έρευνα

Πιθανό πεδίο μελλοντικής μελέτης αποτελεί η εύρεση ενός αποδοτικού μηχανισμού δυναμικής προσαρμογής των αποκομμένων bit του οποίου η επιβάρυνση να μην ξεπερνά το κέρδος των στατικών αποκομμένων bit. Ακόμα ενδιαφέρουσα είναι η μελέτη της συμπεριφοράς του PMicroX σε κύκλωμα ASIC στο οποίο έχουμε υπεροχή της δυναμικής ισχύος έναντι της στατικής και επομένως αναμένεται μεγαλύτερη βελτίωση της ισχύος με εφαρμογή των τεχνικών που προτείνονται. Τέλος βελτίωση επιδέχεται και η αρχιτεκτονική της διοχέτευσης. Συγκεκριμένα η αναπροσαρμογή της ώστε να αφαιρεθεί το στάδιο stall το οποίο δεν έχει ουσιαστική σημασία και εντούτοις να μετατραπεί σε αρχιτεκτονική διοχέτευσης 4 σταδίων με τα στάδια εκτέλεσης και μνήμης συνεπτυγμένα.

Βιβλιογραφία

[1] J. Lee, M. Stanley, A. Spanias, and C. Tepedelenlioglu, "Integrating machine learning in embedded sensor systems for Internet-of-Things applications," *2016 IEEE Int. Symp. Signal Process. Inf. Technol. ISSPIT 2016*, pp. 290–294, 2017, doi: 10.1109/ISSPIT.2016.7886051.

[2] C. Y. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "Exploiting approximate computing for deep learning acceleration," *Proc. 2018 Des. Autom. Test Eur. Conf. Exhib. DATE 2018*, vol. 2018-Janua, pp. 821–826, 2018, doi: 10.23919/DATE.2018.8342119.

[3] M. Pashaeifar, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Approximate Reverse Carry Propagate Adder for Energy-Efficient DSP Applications," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 26, no. 11, pp. 2530–2541, 2018, doi: 10.1109/TVLSI.2018.2859939.

[4] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," *Proc. Int. Conf. …*, 2013, [Online]. Available: http://dl.acm.org/citation.cfm?id=2561854.

[5] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 26, no. 3, pp. 421–430, 2017, doi: 10.1109/TVLSI.2017.2767858.

[6] M. Ceska, J. Matyas, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished," *IEEE/ACM Int. Conf. Comput. Des. Dig. Tech. Pap. ICCAD*, vol. 2017-Novem, pp. 416–423, 2017, doi: 10.1109/ICCAD.2017.8203807.

[7] V. Leon, G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi, "Walking through the Energy-Error Pareto Frontier of Approximate Multipliers," *IEEE Micro*, vol. 38, no. 4, pp. 40–49, 2018, doi: 10.1109/MM.2018.043191124.

[8] V. Leon, K. Asimakopoulos, S. Xydis, D. Soudris, and K. Pekmestzi, "Cooperative arithmetic-aware approximation techniques for energy-efficient multipliers," *Proc. - Des. Autom. Conf.*, no. August 2021, 2019, doi: 10.1145/3316781.3317793.

[9] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGA-based Approximate Multipliers," *Proc. - Des. Autom. Conf.*, vol. Part F1377, pp. 1–6, 2018, doi: 10.1145/3195970.3196115.

[10] V. Leon, T. Paparouni, E. Petrongonas, D. Soudris, and K. Pekmestzi, “Improving Power of DSP and CNN Hardware Accelerators Using Approximate Floating-point Multipliers,” *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, 2021, doi: 10.1145/3448980.

[11] E. Adams, S. Venkatachalam, and S. B. Ko, “Approximate Restoring Dividers Using Inexact Cells and Estimation from Partial Remainders,” *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 468–474, 2020, doi: 10.1109/TC.2019.2953751.

[12] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, “TruncApp: A truncation-based approximate divider for energy efficient DSP applications,” *Proc. 2017 Des. Autom. Test Eur. DATE 2017*, no. 4, pp. 1635–1638, 2017, doi: 10.23919/DATE.2017.7927254.

[13] G. Lentaris, G. Chatzitsompanis, V. Leon, K. Pekmestzi, and D. Soudris, “Combining arithmetic approximation techniques for improved CNN circuit design,” *ICECS 2020 - 27th IEEE Int. Conf. Electron. Circuits Syst. Proc.*, pp. 3–6, 2020, doi: 10.1109/ICECS49266.2020.9294869.

[14] V. Leon, I. Stratakos, G. Armeniakos, G. Lentaris, and D. Soudris, “ApproxQAM: High-Order QAM Demodulation Circuits with Approximate Arithmetic,” *2021 10th Int. Conf. Mod. Circuits Syst. Technol. MOCAST 2021*, pp. 8–12, 2021, doi: 10.1109/MOCAST52088.2021.9493421.

[15] M. Masadeh, Y. Elderhalli, O. Hasan, and S. Tahar, “A Quality-assured Approximate Hardware Accelerators–based on Machine Learning and Dynamic Partial Reconfiguration,” *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 4, pp. 1–19, 2021, doi: 10.1145/3462329.

[16] V. Leon, K. Pekmestzi, and D. Soudris, “Exploiting the Potential of Approximate Arithmetic in DSP & AI Hardware Accelerators,” pp. 263–264, 2021, doi: 10.1109/FPL53798.2021.00049.

[17] V. I. U. Isa, “The RISC-V Instruction Set Manual,” vol. I, 2017.

[18] HEC-HMS, “Technical Reference Manual,” *US Army Corps Eng.*, no. March, p. 155, 2000, [Online]. Available: http://www.hec.usace.army.mil/software/hec-hms/documentation/HEC-HMS_Technical Reference Manual_(CPD-74B).pdf.

[19] R. Samanth, C. V. S. Chaitanya, and G. S. Nayak, “Power Reduction of a Functional unit using RT-Level Clock-Gating and Operand Isolation,” *2019 IEEE Int. Conf. Distrib. Comput. VLSI, Electr. Circuits Robot. Discov. 2019 - Proc.*, pp. 1–4, 2019, doi: 10.1109/DISCOVER47552.2019.9008025.

[20] “Where is My Variable?” https://lisha.ufsc.br/teaching/os/exercise/where_is_my_variable.html (accessed Aug. 14, 2021).

[21] “Basic Script Concepts (LD).” https://sourceware.org/binutils/docs/ld/Basic-Script-Concepts.html (accessed Aug. 21, 2021).

[22] “Simple Example (LD).” https://sourceware.org/binutils/docs/ld/Simple-Example.html (accessed Aug. 21, 2021).

[23] “Data structure alignment - Wikipedia.” https://en.wikipedia.org/wiki/Data_structure_alignment (accessed Aug. 21, 2021).

[24] “MEMORY (LD).” https://sourceware.org/binutils/docs/ld/MEMORY.html (accessed Aug. 21, 2021).

[25] “Output Section LMA (LD).” https://sourceware.org/binutils/docs/ld/Output-Section-LMA.html (accessed Aug. 21, 2021).

[26] “Binutils - GNU Project - Free Software Foundation.” https://www.gnu.org/software/binutils/ (accessed Aug. 21, 2021).

[27] A. R. M. Limited, “AXI spec,” 2011, [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf.

[28] D. Elsner and J. Fenlason, “Using as,” *Computer (Long. Beach. Calif).*, no. January, 1994.

[29] “Extended Asm (Using the GNU Compiler Collection (GCC)).” https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html#Extended-Asm (accessed Sep. 06, 2021).

[30] “Adding an Instruction to the GNU Assembler – Open Source Specialist Group.” https://ossg.bcs.org/blog/2020/10/29/adding-an-instruction-to-the-gnu-assembler/ (accessed Sep. 06, 2021).

[31] S. Ullah, H. Schmidl, S. S. Sahoo, S. Rehman, and A. Kumar, “Area-Optimized Accurate and Approximate Softcore Signed Multiplier Architectures,” *IEEE Trans. Comput.*, vol. 70, no. 3, pp. 384–392, 2021, doi: 10.1109/TC.2020.2988404.

[32] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, “A low latency generic accuracy configurable adder,” *Proc. - Des. Autom. Conf.*, vol. 2015-July, 2015, doi: 10.1145/2744769.2744778.

[33] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, “EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” *Proc. 2017 Des. Autom. Test Eur. DATE 2017*, pp. 258–261, 2017, doi: 10.23919/DATE.2017.7926993.

[34] B. S. Prabakaran *et al.*, “DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems,” *Proc. 2018 Des. Autom. Test Eur. Conf. Exhib. DATE 2018*, vol. 2018-

Janua, pp. 917–920, 2018, doi: 10.23919/DATE.2018.8342140.

[35] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," *Proc. - Des. Autom. Conf.*, vol. 2016-Augus, 2016, doi: 10.1145/2897937.2905008.

[36] P. Analysis, "Vivado Design Suite User Guide Power Analysis and," vol. 907, pp. 1–156, 2015.

[37] Xilinx, "Power Methodology Guide - UG786," vol. 786, no. v2017.2, p. 54, 2017.

[38] Q. D. La, M. V. Ngo, T. Q. Dinh, T. Q. S. Quek, and H. Shin, "Enabling intelligence in fog computing to achieve energy and latency reduction," *Digit. Commun. Networks*, vol. 5, no. 1, pp. 3–9, 2019, doi: 10.1016/j.dcan.2018.10.008.

[39] Institute of Electrical and Electronics Engineers, "2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA 2017) : March 10-12, 2017, Beijing, China.," pp. 721–724, 2017.

[40] "Mean relative error - DEBwiki." http://www.debtheory.org/wiki/index.php?title=Mean_relative_error (accessed Sep. 15, 2021).

[41] "Peak signal-to-noise ratio - Wikipedia." https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio (accessed Sep. 13, 2021).

[42] İ. Taştan, M. Karaca, and A. Yurdakul, "Approximate CPU design for iot end-devices with learning capabilities," *Electron.*, vol. 9, no. 1, 2020, doi: 10.3390/electronics9010125.

[43] G. Ndour, T. T. Jost, A. Molnos, Y. Durand, and A. Tisserand, "Evaluation of Variable Bit-Width Units in a RISC-V Processor for Approximate Computing," *ACM Int. Conf. Comput. Front. 2019, CF 2019 - Proc.*, pp. 344–349, 2019, doi: 10.1145/3310273.3323159.