Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

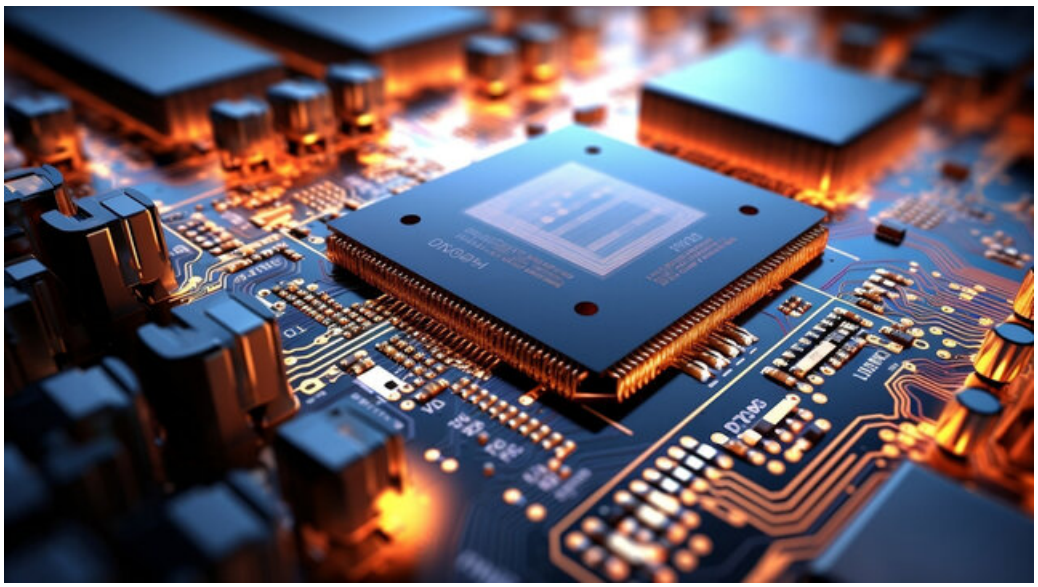
Υλοποίηση Αλγορίθμου Συμπίεσης Διευρύνοντας το ISA RISC-V σε FPGA για Τηλεπικοινωνιακές Εφαρμογές Χαμηλής Καθυστέρησης

Μελέτη και υλοποίηση

Διπλωματικη Εργασια

του

ΠΡΑΣΙΝΟΥ ΔΑΥΙΔ



Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Υλοποίηση Αλγορίθμου Συμπίεσης Διευρύνοντας το ISA RISC-V σε FPGA για Τηλεπικοινωνιακές Εφαρμογές Χαμηλής Καθυστέρησης

Μελέτη και υλοποίηση

Διπλωματικη Εργασια

του

ΠΡΑΣΙΝΟΥ ΔΑΥΙΔ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Ιουλίου 2025.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Γεώργιος Λεντάρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής ΠΑ.ΔΑ.

Αθήνα, Ιούλιος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
ΠΡΑΣΙΝΟΣ ΔΑΥΙΔ

4 Ιουλίου 2025
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών, ΕΜΠ

στους γονείς μου

Περίληψη

Η παρούσα διπλωματική εργασία αφορά την μελέτη και υλοποίηση ενός έξυπνου αλγορίθμου συμπίεσης δεδομένων, βελτιστοποιημένου για χρήση σε τηλεπικοινωνιακά περιβάλλοντα που απαιτούν μεγάλες ταχύτητες μεταφοράς πληροφορίας. Σκοπός είναι η αξιοποίηση των προσαρμοσμένων εντολών (custom instructions) του RISC-V ISA και η χρησιμοποίηση τους σε FPGA, με στόχο τη μείωση της του χρόνου συμπίεσης και αποσυμπίεσης δεδομένων σε Ethernet πακέτα. Η εργασία περιλαμβάνει τη σύγκριση απόδοσης διαφορετικών, πιθανώς κατάλληλων, αλγορίθμων συμπίεσης, τον εντοπισμό των bottleneck σημείων στον θεωρητικά βέλτιστο αλγόριθμο και τέλος την επιτάχυνση των συγκεκριμένων λειτουργιών, δημιουργώντας κατάλληλα custom instructions και υλοποιώντας τα σε FPGA μέσω VHDL. Τελικώς γίνεται η εισαωγή ενός software side traffic manager ο οποίος μας δίνει τη δυνατότητα να αναγνωρίζουμε σε πραγματικό χρόνο αν ένα πακέτο είναι ήδη συμπιεσμένο ή κρυπτογραφημένο, και αν δεν είναι, να αποφασίζουμε αν είναι αποδοτικό να συμπιεστεί, βελτιστοποιώντας έτσι το throughput και τη συνολική ταχύτητα μεταφοράς. Η εφαρμογή αυτή αποσκοπεί στη βελτίωση της απόδοσης σε εφαρμογές τηλεπικοινωνιών, όπου η χαμηλή καθυστέρηση και η υψηλή απόδοση είναι κρίσιμης σημασίας.

Λέξεις Κλειδιά

Αλγόριθμοι Συμπίεσης, VHDL, FPGA, RISC-V ISA, Custom Instruction

Abstract

This thesis investigates and implements an intelligent data compression algorithm aimed for telecommunications environments that demand very high throughput. Its aim is to exploit RISC-V ISA custom instructions on an FPGA to shorten the compression and decompression time of data carried in Ethernet packets. The work includes a performance study of several candidate compression algorithms comparing and identifying the theoritical best, the identification of bottleneck stages in the theoretically optimal algorithm and finally the acceleration of those stages by designing dedicated custom instructions in VHDL on the FPGA. Lastly, a software-side traffic manager is introduced, that can determine in real time, whether a packet is already compressed or encrypted and, if not, decide whether compression will be advantageous, maximizing throughput and minimizing latency. The overall solution targets performance gains in telecommunications applications where low delay and high efficiency are critical.

Keywords

Compression Algorithms, VHDL, FPGA, RISC-V ISA, Custom Instruction

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. Σούντρη για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να την εκπονήσω στο εργαστήριο Τεχνολογίας Πληροφορικής και Υπολογιστών. Επίσης ευχαριστώ ιδιαίτερα τον κ. Λεντάρη για την καθοδήγησή του και την εξαιρετική συνεργασία που είχαμε καθ' όλη την διάρκεια της εκπόνησης της διπλωματικής μου. Θέλω επίσης να ευχαριστήσω τους κ. Κορατζίνο και κ. Μπουσιόπουλο απο την INRACOM TELECOM που με την καθοδήγηση και τις ιδέες τους μα βοήθησαν να ξεπεράσω τις προκλήσεις που έβρισκα μπροστά μου. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την καθοδήγηση και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια.

Αθήνα, Ιούλιος 2025

ΠΡΑΣΙΝΟΣ ΔΑΥΙΔ

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο των σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο και πραγματοποιήθηκε στο εργατήριο του Τομέα Τεχνολογίας Πληροφορικής και Υπολογιστών. Η έρευνα και η ανάπτυξη του έργου πραγματοποιήθηκαν με τη συνεργασία της INTRACOM TELECOM, παρέχοντας πρόσβαση σε υποδομές και τεχνογνωσία απαραίτητες για την ολοκλήρωση της εργασίας.

Η εκπόνηση αυτής της εργασίας αποτέλεσε μια μοναδική ευκαιρία να εμβαθύνω στην αρχιτεκτονική των RISC-V επεξεργαστών και στην υλοποίηση αλγορίθμων συμπίεσης σε FPGA, ενισχύοντας τις γνώσεις μου στο πεδίο των ενσωματωμένων συστημάτων.

Κεφάλαιο 1

Εισαγωγή

Η ταχύτατη ανάπτυξη των τηλεπικοινωνιακών συστημάτων και η εκθετική αύξηση στον όγκο των δεδομένων που διακινούνται μέσω διαδικτύου έχουν καταστήσει επιτακτική την ανάγκη για οσο το δυνατόν αποδοτικότερες τεχνικές διαχείρισης δεδομένων. Λόγω αυτού, η συμπίεση δεδομένων, βρίσκεται πάντα στο επίκεντρο, καθώς επιτρέπει τη μείωση του όγκου των πακέτων που μεταφέρονται, βελτιώνοντας σημαντικά τη ροή (throughput) και τις επιδόσεις του δικτύου. Ωστόσο, όσο οι ταχύτητες δικτύου αυξάνονται, προκύπτει η απαίτηση η συμπίεση να πραγματοποιείται με εξαιρετικά χαμηλή καθυστέρηση (latency), ειδικά σε εφαρμογές πραγματικού χρόνου ή όπου η χρονική απόκριση είναι κρίσιμη.

Σε αυτό το πλαίσιο, αξίζει να σημειωθεί ότι έρευνες [1] δείχνουν πως περίπου το 90% της διαδικτυακής κίνησης είναι ήδη συμπιεσμένη ή/και κρυπτογραφημένη, αφήνοντας ένα 10% δεδομένων που παραμένουν «ανοικτά» σε περαιτέρω βελτιστοποίηση. Για το συγκεκριμένο 10%, θα μπορούσε να χρησιμοποιηθεί ένας εξελιγμένος traffic manager ικανός να αναγνωρίζει δυναμικά αν το πακέτο είναι ήδη συμπιεσμένο ή κρυπτογραφημένο και, σε αντίθετη περίπτωση, να αποφασίζει αν συμφέρει να εφαρμοστεί συμπίεση, κερδίζοντας έτσι σε throughput και ταχύτητα μεταφοράς.

Το ενδιαφέρον για τις αρχιτεκτονικές RISC-V οφείλεται στην ανοικτή (open source) και επεκτάσιμη φύση τους. Παράλληλα παρέχουν την δυνατότητα δημιουργίας προσαρμοσμένων εντολών (custom instructions) που μπορούν να επιταχύνουν συγκεκριμένες λειτουργίες, όπως είναι η συμπίεση. Η υλοποίηση τέτοιων αρχιτεκτονικών πάνω σε FPGA προσφέρει επιπλέον ευελιξία, καθώς μπορούμε να ενσωματώσουμε συγκεκριμένες λογικές μονάδες (accelerators) ακριβώς εκεί όπου χρειάζονται, βελτιώνοντας σημαντικά την απόδοση.

1.1 Σκοπός και Κίνητρο της Διπλωματικής

Η παρούσα διπλωματική εργασία αποσκοπεί στην αντιμετώπιση μιας θεμελιώδους πρόκλησης στα σύγχρονα τηλεπικοινωνιακά συστήματα, τη διαχείριση του συνεχώς αυξανόμενου όγκου δεδομένων, με τη χρήση καινοτόμων μεθόδων συμπίεσης που μπορούν να επιτευχθούν σε περιβάλλοντα χαμηλής καθυστέρησης (latency). Η επιλογή της συγκεκριμένης θεματικής πηγάζει από τη διαρκώς αυξανόμενη απαίτηση για υψηλές επιδόσεις σε πραγματικό χρόνο, ιδιαίτερα σε τηλεπικοινωνιακές εφαρμογές 5G, cloud computing, και Internet of Things (IoT), όπου ακόμη και μικρές καθυστερήσεις στη μεταφορά δεδομένων, αθροίζονται, έχοντας σημαντικό αντίκτυπο στην ποιότητα υπηρεσιών (Quality of Service - QoS) αλλά και

στην εμπειρία του τελικού χρήστη [2].

Βασικό κομμάτι της εργασίας είναι η υιοθέτηση της open source αρχιτεκτονικής RISC-V σε συνδυασμό με την ευελιξία που προσφέρουν τα FPGAs. Η αρχιτεκτονική RISC-V, χάρη στην ανοικτή και επεκτάσιμη φύση της, επιτρέπει την εισαγωγή προσαρμοσμένων εντολών που επιταχύνουν τις κρίσιμες λειτουργίες των αλγορίθμων συμπίεσης. Η χρήση των FPGAs παρέχει την απαραίτητη ευελιξία και δυνατότητα παράλληλης εκτέλεσης, κάτι που είναι δύσκολο να επιτευχθεί με παραδοσιακούς επεξεργαστές γενικού σκοπού [3].

Παρότι μια πλήρως hardware προσέγγιση θα μπορούσε θεωρητικά να επιτύχει ακόμα χαμηλότερους χρόνους επεξεργασίας, στην εργασία αυτή επιλέγεται η λύση που μας δίνει το RISC-V ISA με τα custom instructions, ώστε να επιτευχθεί η καλύτερη δυνατή ισορροπία μεταξύ ελαχιστοποίησης της καθυστέρησης αλλα και της ευελιξίας που μας παρέχει το software. Με αυτόν τον τρόπο, επιδιώκεται μια αρχιτεκτονική που θα προσφέρει ναι μεν την απαιτούμενη απόδοση, επιταχύνοντας κρίσιμες λειτουργίες στο υλικό, παράλληλα όμως θα μπορεί να τροποποιηθεί και να επεκταθεί ευκολότερα σε σχέση με μια αντίστοιχη υλοποίηση αποκλειστικά σε υλικό.

Ένα ακόμη κίνητρο προκύπτει από τη διαπίστωση ότι, παρόλο που ένα μεγάλο ποσοστό (περίπου 90%) των δεδομένων που κυκλοφορούν στο Διαδίκτυο είναι ήδη συμπιεσμένα ή κρυπτογραφημένα [1], υπάρχει ακόμη ένα σημαντικό μέρος δεδομένων (το υπόλοιπο 10%) που μπορεί να συμπιεστεί περαιτέρω. Αυτό το ποσοστό δεδομένων, αποτελεί μια αξιόλογη ευκαιρία για συμπίεση και ταυτόχρονα έξυπνης διαχείρισης της ροής, με στόχο την αύξηση της συνολικής απόδοσης του συστήματος.

Η υλοποίηση αυτής της προσέγγισης σε δικτυακό περιβάλλον αποτελεί ουσιαστική συμβολή στην εξέλιξη των τηλεπικοινωνιακών υπηρεσιών, εξασφαλίζοντας ταχύτατη και αποδοτικότερη διαχείριση των δεδομένων, παράλληλα με τη βέλτιστη εκμετάλλευση των διαθέσιμων πόρων του συστήματος. Αυτό καθιστά την παρούσα εργασία ιδιαίτερα επίκαιρη και αναγκαία, θέτοντας τις βάσεις για περαιτέρω έρευνα και ανάπτυξη στον τομέα των έξυπνων συστημάτων συμπίεσης δεδομένων με ελάχιστο χρόνο απόκρισης.

Συνοπτικά παρότι μια αμιγώς hardware προσέγγιση θα μπορούσε θεωρητικά να επιτύχει ακόμη μικρότερους χρόνους καθυστέρησης, η παρούσα εργασία επιλέγει τη λύση των custom instructions του RISC-V, επειδή μας προσφέρει ισορροπία μεταξύ της ελαχιστοποίησης του latency και της ευελιξίας του λογισμικού.

1.2 Αντικείμενο της διπλωματικής

Βασικός στόχος της παρούσας διπλωματικής εργασίας είναι η μελέτη και υλοποίηση ενός «έξυπνου» αλγορίθμου συμπίεσης δεδομένων σε FPGA, αξιοποιώντας τις δυνατότητες των προσαρμοσμένων εντολών RISC-V. Λαμβάνοντας υπόψη ότι περίπου το 90% της κίνησης στο Διαδίκτυο είναι ήδη συμπιεσμένη ή/και κρυπτογραφημένη, η πρόκληση εστιάζεται κυρίως στο υπόλοιπο 10%. Στο πλαίσιο αυτό, η παρούσα εργασία στοχεύει:

1. **Διερεύνηση και Επιλογή Κατάλληλου Αλγορίθμου Συμπίεσης:** Να μελετηθούν υπάρχοντες αλγόριθμοι συμπίεσης και να επιλεγεί ο καταλληλότερος με γνώμονα την επιθυμητή απόδοση και τη χαμηλή καθυστέρηση.

2. **Σχεδίαση Προσαρμοσμένης Λογικής (Custom Instructions):** Να αναπτυχθούν blocks προσαρμοσμένων εντολών που επιταχύνουν τις καίριες λειτουργίες του αλγορίθμου στο περιβάλλον RISC-V.

3. **Υλοποίηση σε FPGA και Αξιολόγηση Επιδόσεων:** Να υλοποιηθεί η προτεινόμενη αρχιτεκτονική σε FPGA και να αξιολογηθεί ως προς την καθυστέρηση (latency).

4. **Σύγκριση με Άλλες Προσεγγίσεις:** Να τεκμηριωθεί η ωφέλεια της υλοποίησης σε σχέση με τις παραδοσιακές υλοποιήσεις καθαρά λογισμικού (software-only) σε επεξεργαστές γενικού σκοπού.

5. **Υλοποίηση Software Side Traffic Manager:** Να σχεδιαστεί και να υλοποιηθεί ένας traffic manager που θα αναγνωρίζει σε πραγματικό χρόνο αν κάποιο εισερχόμενο πακέτο είναι ήδη συμπιεσμένο ή κρυπτογραφημένο και, σε περίπτωση που δεν είναι, θα κρίνει αν είναι αποδοτικό να συμπιεστεί, βελτιστοποιώντας έτσι το throughput και τη συνολική ταχύτητα μεταφοράς.

Η συμπερίληψη ενός ευέλικτου traffic manager επιτρέπει τη στοχευμένη χρήση των πόρων του συστήματος μόνο για εκείνα τα πακέτα που μπορούν να ωφεληθούν από τη διαδικασία συμπίεσης, μεγιστοποιώντας έτσι την αποτελεσματικότητα της προτεινόμενης προσέγγισης.

1.3 Δομή της Εργασίας

Η εργασία χωρίζεται σε συνολικά **5** κεφάλαια, τα οποία παρουσιάζονται συνοπτικά παρακάτω:

1. **Εισαγωγή**
   Παρουσιάζεται η αναγκαιότητα συμπίεσης δεδομένων σε τηλεπικοινωνιακά συστήματα και η σημασία αξιοποίησης των FPGA και της αρχιτεκτονικής RISC-V για χαμηλή καθυστέρηση. Καθορίζονται οι στόχοι και η συνεισφορά της εργασίας.

2. **Θεωρητικό Υπόβαθρο**
   Παρατίθενται οι βασικές αρχές συμπίεσης δεδομένων, οι λόγοι επιλογής συγκεκριμένων αλγορίθμων, καθώς και η αρχιτεκτονική RISC-V με προσαρμοσμένες εντολές. Επιπλέον, γίνεται εισαγωγή στις τεχνολογίες FPGA, στα εργαλεία σχεδίασης και συζητούνται οι προκλήσεις που ανακύπτουν από την ανάγκη για χαμηλό latency.

3. **Προτεινόμενη Λύση**
   Παρουσιάζονται τα κριτήρια επιλογής αλγορίθμων συμπίεσης, η σχεδίαση των custom instructions, καθώς και η αρχιτεκτονική του συστήματος. Γίνεται, επίσης, αναφορά σε εργαλεία ανάπτυξης και βέλτιστες πρακτικές για την υλοποίηση.

4. **Πειραματικά Αποτελέσματα και Αξιολόγηση Υλοποίησης**
   Περιγράφεται η διαδικασία υλοποίησης και ολοκλήρωσης (integration) των επιμέρους hardware και software τμημάτων, καθώς και η διαμόρφωση των πειραμάτων. Αξιολογούνται τα κέρδη μέσω μετρήσεων απόδοσης.

5. **Συμπεράσματα και Μελλοντική Εργασία**
   Γίνεται συνολική αποτίμηση της εργασίας, αναδεικνύονται τα βασικά συμπεράσματα και προτείνονται ιδέες για περαιτέρω βελτίωση ή επέκταση της προσέγγισης.

Μέρος Ι

Θεωρητικό Μέρος

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά οι βασικές τεχνολογίες που έχουν σχέση με την εργασία αυτή.

2.1 Επισκόπηση Τεχνικών Συμπίεσης

Η συμπίεση δεδομένων (data compression) συνιστά μία εκ των πιο διαδεδομένων και αποδοτικών μεθόδων για τη μείωση του όγκου των πακέτων που διακινούνται σε δικτυακά συστήματα [4]. Γενικά, διακρίνεται σε δύο κύριες κατηγορίες:

1. **Απωλεστική (Lossy) Συμπίεση:** Η οποία θυσιάζει μέρος της πληροφορίας προκειμένου να επιτύχει υψηλότερους λόγους συμπίεσης (π.χ. τα περισσότερα format εικόνας ή βίντεο).

2. **Χωρίς Απώλειες (Lossless) Συμπίεση:** Όπου η διαδικασία είναι αναστρέψιμη, δηλαδή επιτρέπει την ανακατασκευή των αρχικών δεδομένων χωρίς καμία αλλοίωση. Συνηθέστερα παραδείγματα είναι οι Huffman, LZ77, LZ4, Snappy κ.α.

Στην παρούσα εργασία εστιάζουμε αποκλειστικά σε τεχνικές lossless συμπίεσης.

2.1.1 Θεωρία Πληροφορίας και Έννοια της Εντροπίας

Στη θεωρία πληροφορίας, ο όρος εντροπία (entropy) εισήχθη από τον Shannon (1948)[5, 6, 7, 8] και αποτυπώνει το κάτω φράγμα της μέσης πληροφορίας (σε bits) που απαιτείται για την κωδικοποίηση ενός συνόλου συμβόλων.

- **Ορισμός Entropy:** Αν έχουμε ένα σύνολο από n διαφορετικά σύμβολα $\{s_1, s_2, \ldots, s_n\}$ που εμφανίζονται με πιθανότητες $\{p_1, p_2, \ldots, p_n\}$ αντίστοιχα, τότε η εντροπία (entropy) του συνόλου ορίζεται ως:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{2.1}$$

  Η τιμή του $H(X)$ αντανακλά πόσο «απρόβλεπτα» είναι τα δεδομένα. Όσο υψηλότερη είναι η εντροπία, τόσο πιο δύσκολη καθίσταται η συμπίεση.

- **Θεωρητικό Όριο Συμπίεσης:** Η εντροπία καθορίζει το ελάχιστο μήκος κωδικοποίησης (bits per symbol) που μπορούμε να επιτύχουμε ασυμπτωτικά. Σε συστήματα

με χαμηλή εντροπία (π.χ. μεγάλες επαναλήψεις του ίδιου συμβόλου), οι σύγχρονες τεχνικές συμπίεσης μπορούν να πετύχουν υψηλούς λόγους συμπίεσης. Αντίθετα, αν η εντροπία πλησιάζει τη μέγιστη τιμή της, οι δυνατότητες συμπίεσης είναι περιορισμένες.

- **Σύγχρονοι Αλγόριθμοι Συμπίεσης:** Οι περισσότεροι lossless αλγόριθμοι (Huffman, LZ77, LZ4, Snappy) αξιοποιούν έμμεσα ή άμεσα την έννοια της εντροπίας. Επιδιώκουν να ανιχνεύσουν πλεονασμούς (redundancies) στα δεδομένα ή να κωδικοποιήσουν συχνά εμφανιζόμενα μοτίβα με τρόπο που απαιτεί λιγότερα bits. Στόχος είναι η προσέγγιση του θεωρητικού ορίου συμπίεσης που θέτει η εντροπία.

Συνεπώς, η εντροπία λειτουργεί ως μετρική για τις δυνατότητες συμπίεσης ενός συνόλου δεδομένων, υποδεικνύοντας πόση «πληροφορία» περιέχουν και πόσο εύκολα ή δύσκολα μπορούν να συμπιεστούν. Στις ενότητες που ακολουθούν, θα εξετάσουμε συγκεκριμένους lossless αλγόριθμους που εκμεταλλεύονται το redundancy, λαμβάνοντας υπόψη τις αρχές της θεωρίας πληροφορίας.

2.1.2 Γνωστοί Αλγόριθμοι Lossless Συμπίεσης

Huffman Coding

Ο κλασικός αλγόριθμος Huffman προτείνει τη βέλτιστη (ως προς το μέσο μήκος) *variable-length prefix* κωδικοποίηση σε ένα σύνολο συμβόλων [9].

- Υπολογίζει έναν frequency table και κατασκευάζει δυαδικό δέντρο έτσι ώστε όσο υψηλότερη η συχνότητα τόσο συντομότερος ο κωδικός.
- Αποδεικνύεται ότι το αποτέλεσμα ελαχιστοποιεί το αναμενόμενο μήκος κωδικού υπό τον περιορισμό prefix-free [6].
- Χρησιμοποιείται ως στάδιο entropy coding σε σχήματα τύπου DEFLATE [10].

LZ77

Ο αλγόριθμος LZ77 [11] εισήχθη από τους Ziv και Lempel το 1977 και βασίζεται σε μια *αυτοαναφορική* (dictionary-based) προσέγγιση:

- Χρησιμοποιεί ένα sliding window (συνήθως με σταθερό μέγεθος) που περιέχει ένα τμήμα των δεδομένων που έχουν ήδη επεξεργαστεί.
- Εάν ένα νέο μοτίβο (pattern) επαναλαμβάνει ακολουθία συμβόλων που υπάρχει εντός του παραθύρου, αντί να αποθηκευτεί εκ νέου, αποθηκεύεται μια *αναφορά* (back-reference) με τη μορφή (*offset, length*).
- Το offset υποδεικνύει πόσο πίσω βρίσκεται η επαναλαμβανόμενη ακολουθία και το length το μήκος της εν λόγω ακολουθίας.
- Η ιδέα αυτή αποτελεί τη βάση επιτυχημένων μοντέρνων κωδικοποιητών όπως το DEFLATE [10].

LZ4

Ο LZ4 [12] είναι μια πιο σύγχρονη υλοποίηση, εμπνευσμένη από την οικογένεια LZ77, με στόχο την *υψηλή ταχύτητα* τόσο στην συμπίεση όσο και στην αποσυμπίεση:

- Βασίζεται επίσης σε dictionary-based τεχνικές, αλλά χρησιμοποιεί αποδοτικότερες δομές δεδομένων για γρηγορότερη ανεύρεση επαναλήψεων χαρακτήρων.
- Παρέχει ταχύτητες >500 MB/s ανά πυρήνα για συμπίεση και πολλά GB/s για αποσυμπίεση [12].
- Συνήθως επιτυγχάνει *καλή* (αλλά όχι την καλύτερη δυνατή) αναλογία συμπίεσης, θυσιάζοντας λίγη απο την απόδοση υπέρ της ταχύτητας. Ενδεικτικά σε δοκιμές βλέπουμε compression ratio 2.08:1 [13].
- Λόγω απλότητας και σταθερού λεξικού είναι ιδιαίτερα δημοφιλής για real-time εφαρμογές σε ενσωματωμένα συστήματα.

Snappy

Ο Snappy (αρχικά γνωστός ως Zippy) [14], αναπτύχθηκε από την Google και στοχεύει σε ταχύτατη συμπίεση θυσιάζοντας και αυτός λίγο απο το compression ratio:

- Εφαρμόζει κι αυτό τεχνικές της LZ77 οικογένειας, με έμφαση στην ταχύτητα εύρεσης ομοιοτήτων.
- Επιτυγχάνει συμπίεση περίπου στα 250 MB/s και αποσυμπίεση κοντά στα 500 MB/s με έναν πυρήνα [14].
- Ενώ η αναλογία συμπίεσης μπορεί να είναι χαμηλότερη συγκριτικά με άλλους αλγορίθμους, η υψηλή του απόδοση σε throughput τον καθιστά δημοφιλή σε streaming ή big data εφαρμογές.
- Είναι ευρέως διαδεδομένος σε εσωτερικές υποδομές της Google για ανταλλαγή δεδομένων σε πραγματικό χρόνο.

LZO

Συνδυάζει τεχνικές LZ77 με μικρό λεξικό 64 KB και χωρίς στάδιο entropy coding [15]:

- Πετυχαίνει συμπίεση περίπου στα 650 MB/s ενώ αποσυμπίεση σχεδόν in place με λόγο κοντά στο 2:1 [16].
- Σύμφωνα με την βιβλιογραφία, βλέπουμε πως υλοποιημένη έκδοση του σε FPGA κατάφερε να μειώσει τους πόρους κατά 60 % χωρίς καμία απώλεια στην απόδοση [17].

Zstd–L1

Και εδώ έχουμε συνδιασμό του LZ77 με Finite State Entropy. Το επίπεδο 1 L1 δίνει προτεραιότητα στην ταχύτητα [18].

- Παρέχει λόγο συμπίεσης περίπου 2.9:1 με κοντά 500 MB/s συμπίεση και αποσυμπίεση πανώ απο 1 GB/s [19].
- Υπάρχουν συγχρονές υλοποιήσεις με 8.6 GB/s throughput και ως συνεπεξεργαστής RISC-V [20, 21].

Gzip–6 (DEFLATE)

Ενώνει τον LZ77 με Dynamic Huffman Coding ενώ το επίπεδο 6 προσφέρει ισορροπία όσων αφορά την ταχύτητα και το ratio [4, 10].

- Πετυχαίνει εντυπωσιακά υψηλό ratio 2–4:1 αλλά ταχύτητα συμπίεσης και αποσυμπίεσης περίπου μόλις 40 MB/s και 200–500 MB/s αντίστοιχα.
- Υλοποιήσεις του σε FPGA και ASIC φαίνεται να άρουν το CPU bottleneck σε δικτυακές ροές [22].

Παρατηρώντας τις λειτουργίες καθενός από τους παραπάνω αλγορίθμους, διαπιστώνεται ότι όλοι στοχεύουν στην αφαίρεση redundancy από τα δεδομένα, ακολουθώντας βέβαια, σε έναν βαθμό διαφορετικές προσεγγίσεις προσδίδοντας τους διαφορετικά πλεονεκτήματα και μειονεκτήματα ως προς την ταχύτητα και το ratio. Στις τηλεπικοινωνιακές εφαρμογές, όπου η χαμηλή καθυστέρηση (latency) είναι κρίσιμη, οι πιο πρόσφατες ή γρήγορες υλοποιήσεις όπως οι LZ4, LZO και Snappy συχνά προτιμώνται, παρότι μπορεί να μην επιτυγχάνουν το ίδιο ratio με προσεγγίσεις όπως αυτή του Gzip–6.

2.2 Εισαγωγή στην Αρχιτεκτονική RISC-V

Η αρχιτεκτονική RISC-V αποτελεί μια ανοικτή (open source) και επεκτάσιμη αρχιτεκτονική συνόλου εντολών (ISA)[3], η οποία έχει σχεδιαστεί με στόχο την απλότητα και την ευελιξία. Οι βασικές επεκτάσεις (extensions) περιλαμβάνουν λειτουργίες για ακέραιες πράξεις, υποστήριξη κινητής υποδιαστολής, διανυσματικές πράξεις (vector extensions), κρυπτογραφικές εντολές κ.α.

Οι προσαρμοσμένες εντολές (custom instructions) έχουν αποδειχθεί ιδιαίτερα αποτελεσματικές σε εφαρμογές συμπίεσης [23].

Προσαρμοσμένες Εντολές (Custom Instructions) Μία σημαντική καινοτομία της αρχιτεκτονικής RISC-V, είναι η δυνατότητα προσθήκης προσαρμοσμένων εντολών, επιτρέποντας την υλοποίηση εξειδικευμένων λειτουργιών στο υλικό. Για παράδειγμα, μπορούμε να σχεδιάσουμε εντολές που επιταχύνουν κρίσιμα τμήματα αλγορίθμων συμπίεσης (π.χ. την εύρεση επαναλαμβανόμενων ακολουθιών, την κωδικοποίηση συμβόλων, κ.α.). Η προσέγγιση αυτή οδηγεί σε:

- **Χαμηλότερη Καθυστέρηση:** Εφόσον οι πράξεις συμπίεσης πραγματοποιούνται άμεσα σε επίπεδο υλικού.
- **Μείωση Φορτίου στον Επεξεργαστή:** Καθώς οι συγκεκριμένες πράξεις δεν απαιτούν εκτέλεση σύνθετων ρουτινών στο λογισμικό.

Χρήση της αρχιτεκτοινκής RISC-V σε Ενσωματωμένα Συστήματα Η ευελιξία της RISC-V την καθιστά ιδιαιτέρως ελκυστική στα ενσωματωμένα συστήματα (embedded systems), αφού επιτρέπει τη δημιουργία προσαρμοσμένων πυρήνων (softcores) με τις *ελάχιστες* απαιτούμενες λειτουργίες (π.χ. RV32I, RV64I) αλλά και την προσθήκη εξειδικευμένων επεκτάσεων (extensions).

2.3 FPGA: Αρχιτεκτονική και Ροές Σχεδίασης

Τα FPGA (Field Programmable Gate Arrays) είναι ολοκληρωμένα κυκλώματα που περιλαμβάνουν ένα πλέγμα (grid) προγραμματιζόμενων λογικών κυψελών (logic cells / LUTs), μπλοκ μνήμης (BRAM), μπλοκ ψηφιακών πολυπλασιαστών (DSP blocks) και διαύλους διασύνδεσης [24]. Το χαρακτηριστικό τους πλεονέκτημα έγκειται στην ικανότητα παράλληλης εκτέλεσης πολλαπλών λειτουργιών ταυτόχρονα, καθιστώντας τες ιδανικές για εφαρμογές όπως η συμπίεση, όπου απαιτούνται γρήγορες επαναλαμβανόμενες πράξεις.

Γλώσσες και Εργαλεία Σχεδίασης Η παραδοσιακή σχεδίαση FPGA στηρίζεται κυρίως σε γλώσσες περιγραφής υλικού (HDL), όπως η VHDL και η Verilog. Παράλληλα, οι σύγχρονες ροές σχεδίασης High-Level Synthesis (HLS) επιτρέπουν την αυτόματη μετάφραση κώδικα υψηλού επιπέδου (C/C++) σε λογισμικό RTL, μειώνοντας δραστικά τον χρόνο ανάπτυξης. Ωστόσο, η HLS μπορεί να παρουσιάζει ορισμένους περιορισμούς στη βέλτιστη διαχείριση των πόρων του FPGA, ανάλογα με την πολυπλοκότητα του αλγορίθμου που υλοποιείται.

Σύγκριση HDL και HLS

- **HDL:** Δίνει πολύ καλή λεπτομερή έλεγχο του υλικού (hardware), βέλτιστη αξιοποίηση πόρων, όμως απαιτεί μεγαλύτερο χρονικό διάστημα ανάπτυξης και εξειδικευμένες γνώσεις.
- **HLS:** Επιταχύνει τον αρχικό κύκλο ανάπτυξης και debugging (ιδιαίτερα σε αλγοριθμικό επίπεδο), αλλά ενδέχεται να μην επιτυγχάνει πάντοτε την ίδια βέλτιστη χρήση πόρων με την παραδοσιακή σχεδίαση HDL.

2.4 Σύγκριση Αποκλειστικής Υλοποίησης σε Υλικό Έναντι Software/Hardware co-design

Σε εφαρμογές με απαιτήσεις χαμηλού **latency** και υψηλού **throughput**, όπως η real time συμπίεση δεδομένων σε τηλεπικοινωνιακά συστήματα, μπορούν να ακολουθηθούν δύο κύριες προσεγγίσεις αρχιτεκτονικής. Η πρώτη είναι η πλήρως hardware υλοποίηση, δηλαδή

μια λύση αποκλειστικά σχεδιασμένη στο υλικό χωρίς την ανάγκη επεξεργαστή. Η δεύτερη προσέγγιση είναι ο συνδυασμός υλικού/λογισμικού, όπου χρησιμοποιείται ένας επεξεργαστής (στην περίπτωση μας αυτός ειναι ενας RISC-V) σε συνεργασία με custom instruction blocks για την επιτάχυνση συγκεκριμένων λειτουργιών.

Full-hardware υλοποίηση

Μια υλοποίηση εξόλοκλήρου στο υλικό μπορεί να επιτύχει εξαιρετικά χαμηλή καθυστέρηση και πολύ υψηλό throughput. Αθτό συμβαίνει γιατί ολη η δουλειά πραγματοποιείται με πλήρως παραλληλοποιημένη λογική, δίχως την αναάγκη διαμεσολάβησης ενός επεξεργαστή [24, 25]. Η προσέγγιση αυτή είναι ιδανική για απαιτητικά real-time συστήματα, καθώς προσφέρει χαμηλούς-ντετερμινιστικούς χρόνους απόκρισης αλλά και δυνατότητα ταυτόχρονης επεξεργασίας πολλαπλών ροών δεδομένων [24]. Επιπλέον, τέτοιες υλοποιήσεις σε FPGA ή ASIC μπορούν να επιτύχουν θεαματικά χαμηλότερες καθυστερήσεις και υψηλότερη απόδοση έναντι λύσεων που βασίζονται μόνο σε λογισμικό [24].

Το τίμημα είναι η περιορισμένη **ευελιξία** και η αυξημένη πολυπλοκότητα. Μια τέτοια υλοποίηση στο υλικό είναι συνήθως βελτιστοποιημένη μόνο για μία συγκεκριμένη λειτουργία ενώ οποιαδήποτε αλλαγή στον αλγόριθμο ή στις απαιτήσεις της εφαρμογής, απαιτεί μια ενδεχομενώς, χρονοβόρα επανασχεδίαση σε HDL και νέα υλοποίηση στο FPGA/ASIC [26]. Επιπλέον, η πλήρης υλοποίηση σε υλικό μπορεί να οδηγήσει σε μεγαλύτερη κατανάλωση ψηφιακών πόρων, καθώς για να επιτευχθεί η μέγιστη δυνατή επίδοση δεσμεύονται περισσότερα logic blocks και μνήμη, κάτι που μπορεί να περιορίσει άλλες αναγκαίες λειτουργίες ή να αυξήσει το κόστος [24]. Συνολικά, μια τέτοια υλοποίηση μας προσφέρει εξαιρετικές επίδοσεις σε latency και throughput, αλλά υστερεί σε ευελιξία ενώ έχει σαφώς μεγαλύτερο χρόνο ανάπτυξης, λόγω της πολυπλοκότητας που έχει. [26].

Συνδυασμένη υλοποίηση υλικού/λογισμικού

Αυτή η προσέγγιση στοχεύει σε καλύτερη ισορροπία μεταξύ απόδοσης και ευελιξίας. Ένας επεξεργαστής αναλαμβάνει τον έλεγχο και τις λιγότερο κρίσιμες εργασίες στο λογισμικό, ενώ οι πιο κοστοβόρες λειτουργίες επιταχύνονται μέσω των custom blocks στο υλικό (στην περίπτωση μας τα custom instructions ενσωματωμένα στον επεξεργαστή). Έτσι, ο σχεδιασμός επωφελείται τόσο από την **ευελιξία** του λογισμικού όσο και από την **ταχύτητα** του υλικού [3].

Η βιβλιογραφία επιβεβαιώνει ότι μια τέτοια προσέγγιση επιτυγχάνει τη σωστή ισορροπία μεταξύ καθαρά υλοποιήσεις σε γενικού σκοπού επεξεργαστές και full scale accelerators. Οι επεκτάσεις του RISC-V ISA προσφέρουν ένα αρκετά ελκυστικό trade-off, αποφεύγοντας τον τεράστιο χρόνο ανάπτυξης ενός πλήρως ειδικού design στο υλικό, ενω παράλληλα κερδίζουν σημαντικά σε ταχύτητα. Για παράδειγμα, σε μια πρόσφατη εργασία που αφορούσε την επιτάχυνση αλγορίθμων μηχανικής μάθησης, προστέθηκαν προσαρμοσμένες εντολές στον RISC-V και αντίστοιχα hardware blocks για επιτάχυνση sparse DNN (Deep Neural Networks). Σε αυτή την προσέγγιση φαίνεται καθαρά η φιλοσοφία του HW/SW co-design, αποδίδοντας μάλιστα επιτάχυνση εως και 5× σε σχέση με την απλή προσέγγιση baseline επεξεργαστή.[27]

Σε γενικειες γραμμές η προσέγγιση όπου ενας επεξεργαστής RISC-V υλοποιείται εντός FPGA με Custom instructions για συγκεκριμένες διεργασίες, προσφέρει μια ιδιαίτερα χρήσιμη ευελιξία. Όταν ο ίδιος πυρήνας μπορεί να εκτελεί τόσο γενικές άλλα τόσο και επιταχυνόμενες με custom instructions διεργασίες, χωρίς να απαιτείται κάποια ξεχωριστή μονάδα, κάτι που σε real time ενδεχομένως να είναι εξαιρετικά ωφέλιμο. Σε κάθε περίπτωση η προγραμματιζόμενη αρχιτεκτονική αυτή, μας επιτρέπει εκ των υστέρων σχεδιασμό νέων απαιτήσεων, συνδυάζοντας την ευελιξία τόσο των FPGA όσο και του RISC-V ISA. [28]

Από πλευράς πόρων, αυτη η αρχιτεκτονική μπορεί να κάνει πιο αποδοτική την χρήση του FPGA, καθώς υλοποιεί στο hardware μόνο τα επιλεγμένα τμήματα του αλγορίθμου, ενώ αφήνει στον επεξεργαστεί να κάνει τα υπόλοιπα [24]. Σε επίπεδο απόδοσης, μια τέτοια λύση είναι το πιο πιθανό να υστερεί έναντι της full-hardware υλοποίησης, καθώς εξακολουθεί να υπάρχει καθυστέρηση από το λογισμικό και την επικοινωνία με την CPU[25]. Ωστόσο, σε καλά σχεδιασμένες υλοποιησείς, το τελικό latency και throughput τείνουν να πλησιάζουν τις επιδόσεις μιας full-hardware λύσης, ενώ παράλληλα διατηρείται η προγραμματιστική ευελιξία.

Συμπέρασμα: Για πολλές σύγχρονες εφαρμογές που απαιτούν ταυτόχρονα πολύ χαμηλή καθυστέρηση και υψηλό ρυθμό επεξεργασίας, αλλά και ευελιξία ανάπτυξης, ο συνδυασμός ενός επεξεργαστή με προσαρμοσμένες μονάδες υλικού προσφέρει συνήθως την πιο ισορροπημένη λύση.

2.5 Δίκτυα

2.5.1 Διαχείριση Κυκλοφορίας και Real-Time Αξιολόγηση Πακέτων

Η ιδέα ενός traffic manager ενσωματώνεται άριστα σε ένα τέτοιο περιβάλλον, καθώς το FPGA μπορεί να φιλοξενεί τόσο τον πυρήνα συμπίεσης όσο και τον μηχανισμό αναγνώρισης (detection engine) για την ταξινόμηση (classification) πακέτων σε ήδη συμπιεσμένα/κρυπτογραφημένα ή μη συμπιεσμένα. Έτσι επιτυγχάνεται άμεση απόφαση για:

1. **Παράκαμψη Συμπίεσης:** Αν το πακέτο δεν είναι δυνατό να υποστεί πρόσθετη συμπίεση, εξοικονομούνται χρόνος και πόροι.

2. **Συμπίεση σε πραγματικό χρόνο:** Αν το πακέτο πληροί τις προϋποθέσεις, τότε δρομολογείται ταχύτατα στον συμπιεστή (compression engine).

Με τον τρόπο αυτό επιτυγχάνονται καλύτερο throughput και χαμηλότερες καθυστερήσεις, κάτι κρίσιμο για εφαρμογές low-latency.

Υπολογισμός Εντροπίας με Packet-Δειγματοληψία

Το ερώτημα εάν αρκεί να εξετάσουμε μόνον ένα τμήμα του payload έχει απαντηθεί θετικά σε πρόσφατη βιβλιογραφία:

- Το HEDGE δείχνει ότι η τυχαιότητα του payload μπορεί να μετρηθεί αξιόπιστα σε **τυχαιοποιημένα υποσύνολα πακέτων**[29].

- Ο RT–ETD ανιχνευτής του Dorfinger υπολογίζει την εντροπία **μόνο στο πρώτο packet** κάθε ροής και επιτυγχάνει >94 % επιτυχία ταξινόμησης σε πραγματικό χρόνο[30].
- Σε θεωρητικό επίπεδο, ο Paninski αποδεικνύει ότι οι εκτιμητές εντροπίας είναι συνεπείς ακόμη και "in surprisingly small sample regimes", παρέχοντας άνω/κάτω φράγματα σφάλματος[31].

2.6 Σύνοψη Κεφαλαίου

Στο παρόν κεφάλαιο, παρουσιάστηκαν οι βασικές αρχές συμπίεσης δεδομένων, με έμφαση στις απωλεστικές και μη-απωλεστικές τεχνικές. Αναδείχθηκε η σημασία των προσαρμοσμένων (custom instructions) στην αρχιτεκτονική RISC-V και ο ρόλος των FPGA ως μια ευέλικτη πλατφόρμα που υποστηρίζει την παράλληλη επεξεργασία. Τέλος, συζητήθηκε η έννοια του traffic manager που αναγνωρίζει σε πραγματικό χρόνο τα είδη πακέτων και βελτιστοποιεί τη διαδικασία συμπίεσης, ιδιαίτερα για το 10% των δεδομένων που παραμένουν μη συμπιεσμένα ή μη κρυπτογραφημένα.

Οι έννοιες αυτές αποτελούν το θεωρητικό πλαίσιο στο οποίο θα βασιστεί η προτεινόμενη μεθοδολογία και η υλοποίηση που θα παρουσιαστούν στο επόμενο κεφάλαιο.

Μέρος II

Πρακτικό Μέρος

Κεφάλαιο 3

Προτεινόμενη Λύση

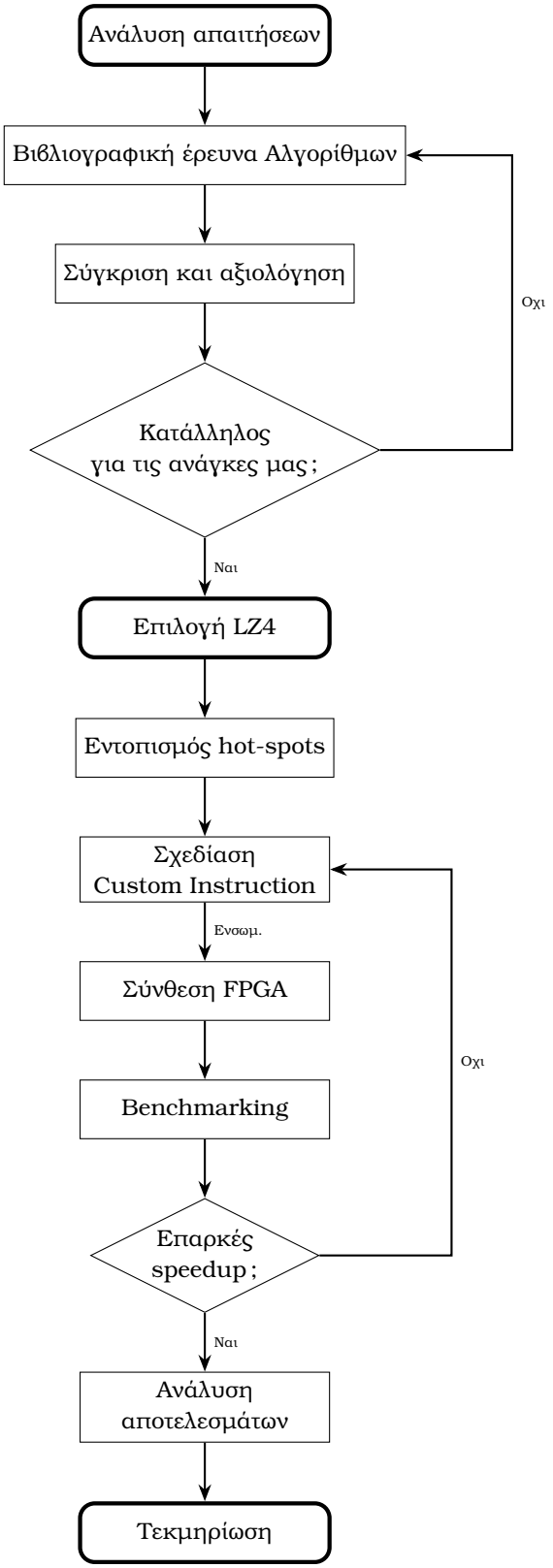
Εισαγωγή

Η παρούσα εργασία προτείνει μια αρχιτεκτονική συμπίεσης χαμηλής καθυστέρησης για ροές πακέτων Ethernet, στηριγμένη σε RISC-V επεξεργαστή που υλοποιείται σε FPGA και επωφελείται από τα custom instructions και την ευελιξία τους.

Η λύση μας, μετά απο έρευνα, επικεντρώνεται τελικώς αποκλειστικά στον αλγόριθμο LZ4. Μετά απο profiling καταλήγουμε σε 2 custom instructions το LZ4_COUNT και το LZ4_HASH, τα οποία ενσωματώνονται στον source κώδικα του lz4 ως macros. Παράλληλα, στον traffic manager ενσωματώνεται ενα ακόμα custom instruction το ENTROPY_CALC που, μέσω ενός Πίνακα Αναζήτησης (LUT), επιταχύνει σημαντικά τον υπολογισμό της εντροπίας.

Η υλοποίηση ξεκινά σε C πάνω σε Nios-V πυρήνα και τα custom instruction blocks είναι γραμμένα σε VHDL. Οι τρεις παραπάνω εντολές προστίθενται στο NIOS-V system που περιγράφουμε στα επόμενα κεφάλαια. Τα πειραματικά αποτελέσματα και η λεπτομερής περιγραφή των custom instructions παρουσιάζονται στις επόμενες ενότητες.

3.1 Μεθοδολογία



Σχήμα 3.1: *Διάγραμμα ροής της μεθοδολογίας: από τη βιβλιογραφική αξιολόγηση αλγορίθμων μέχρι την τελική τεκμηρίωση.*

3.1.1 Συγκριτική μελέτη αλγορίθμων

Μελετήθηκαν πέντε ευρέως χρησιμοποιούμενοι αλγόριθμοι (LZ4, Snappy, LZO, Zstd–L1, Gzip–6) σε τέσσερα μεγέθη/τάξεις Ethernet το 112 B (ελάχιστο + headers), τα 2 KB (τυπικό 1518 B + padding), τα 9.6 KB (*jumbo*) και ένα stress-case 983 KB (δοκιμή κλιμάκωσης). Για κάθε μέγεθος μετρήθηκαν

(i) ο λόγος συμπίεσης (ratio),

(ii) ο χρόνος συμπίεσης t_c,

(iii) ο χρόνος αποσυμπίεσης t_d

.

Πίνακας 3.1: *Latency (μs) και λόγος συμπίεσης για πακέτα 112 B.*

Αλγόριθμος	Ratio	t_c	t_d
LZ4	1.21	1.03	0.88
Snappy	1.03	2.00	1.86
LZO	1.08	1.54	0.32
Zstd–L1	0.79	4.28	2.50
Gzip–6	0.93	19.68	20.28

Πίνακας 3.2: *Latency (μs) και λόγος συμπίεσης για πακέτα 2 KB.*

Αλγόριθμος	Ratio	t_c	t_d
LZ4	0.77	4.24	1.67
Snappy	0.75	5.36	2.92
LZO	0.75	3.93	1.62
Zstd–L1	0.49	14.07	5.37
Gzip–6	0.49	64.79	37.51

Πίνακας 3.3: *Latency (μs) και λόγος συμπίεσης για jumbo πακέτα 9.6 KB.*

Αλγόριθμος	Ratio	t_c	t_d
LZ4	0.58	16.15	4.54
Snappy	0.55	23.65	10.72
LZO	0.55	21.71	16.59
Zstd–L1	0.38	56.68	17.51
Gzip–6	0.37	249.97	75.70

Πίνακας 3.4: *Latency (μs) και λόγος συμπίεσης για stress-case πακέτα 983 KB.*

Αλγόριθμος	Ratio	t_c	t_d
LZ4	0.47	2 878	426
Snappy	0.47	2 182	801
LZO	0.48	2 575	2 734
Zstd–L1	0.32	11 575	902
Gzip–6	0.30	56 607	4 051

Όλα τα τεστ εκτελέστηκαν με το *ίδιο* payload πακέτο, επαναλήφθηκαν 10 φορές για κάθε μέγεθος και οι χρόνοι προκύπτουν ως mean τιμές των μετρήσεων.

3.1.2 Γιατί επιλέχθηκε ο LZ4.

Με γνώμονα οτι μας ενδιαφέρει περισσότερο η ταχύτητα, με βάση τα αποτελέσματα καταλήγουμε σε έναν εκ των LZ4 ή LZO. Ο LZ4 υπερίσχυσε του LZO για τρεις κύριους λόγους.

(i) **Απόδοση.** Με ταχύτητα συμπίεσης πάνω 500 MB/s/core και αποσυμπίεσης που αγγίζει τα όρια του RAM bandwidth, ο LZ4 φτάνει σε επίπεδα κατάλληλα για τηλεπικοινωνιακά traffic-flows χωρίς να απαιτεί επιπλέον πυρήνες [12]. Πρακτικές υλοποιήσεις σε FPGA/ASIC καταγράφουν ακόμη υψηλότερο throughput, επιβεβαιώνοντας ότι ο αλγόριθμος κλιμακώνεται γραμμικά με την παραλληλοποίηση του υλικού [32].

(ii) **Βιομηχανική υιοθέτηση.** Ο LZ4 υποστηρίζεται πλέον στον Linux kernel από την έκδοση 3.11 (kernel image, Zswap, SquashFS), γεγονός που τον καθιέρωσε ως προεπιλογή σε πολλά open-source δίκτυα και embedded OS [33]. Ακόμη και σε carrier-grade πλατφόρμες, όπως το Cisco ASR 5 000 (Service Redundancy Protocol), προσφέρεται επιλογή LZ4 για τη συμπίεση μηνυμάτων SRP επειδή «κλιμακώνεται σχεδόν γραμμικά» σε πολυνηματικές εφαρμογές [34]. Η ευρεία χρήση ουσιαστικά μεταφράζεται σε ενεργή συντηρηση, παρέχοντας συχνά νέα patches και υποστήριξη σε toolchains.

(iii) **Άδεια χρήσης.** Η άδεια χρήσης της υλοποίησης του LZ4 είναι BSD 2-Clause [35] σε σχέση με την GPLv2 του LZO[15]. Αυτό επιτρέπει ενσωμάτωση σε ιδιόκτητα προϊόντα χωρίς περίπλοκους νομικούς περιορισμούς και licence barriers.

Συνοψίζοντας, ο LZ4 προσφέρει καλύτερη υλική κλιμάκωση και ένα open-source οικοσύστημα που συμβαδίζει με τις μακροχρόνιες απαιτήσεις συντήρησης τηλεπικοινωνιακών υποδομών, στοιχεία που τον καθιστούν προτιμότερο έναντι του πλέον γηραιότερου LZO.

3.1.3 Profiling επιλεγμένου αλγορίθμου LZ4

Για την αποδοτική επιτάχυνση του αλγορίθμου LZ4 σε υλικό, ήταν απαραίτητο να προηγηθεί λεπτομερής μελέτη της χρονικής συμπεριφοράς του μέσω profiling. Αρχικά επιχειρήθηκε η χρήση εργαλείων όπως το perf stat σε περιβάλλον Linux, με στόχο την καταγραφή μετρικών όπως CPU cycles, instructions retired, και cache misses. Ωστόσο, η χρήση τέτοιων εργαλείων αποδείχθηκε ανεπαρκής. Αυτό οφείλεται στο γεγονός ότι η υλοποίηση της LZ4 βιβλιοθήκης (όπως διατίθεται στο επίσημο GitHub) είναι έντονα βελτιστοποιημένη για εκτέλεση σε CPU. Συγκεκριμένα, πολλές κρίσιμες συναρτήσεις έχουν δηλωθεί ως force-inlined, ενώ επιπλέον η χρήση branchless constructs και η αποφυγή άσκοπων κλήσεων καθιστούν δύσκολη την ανάλυση μέσω εξωτερικού sampling.

Ως εναλλακτική λύση, εφαρμόστηκε εσωτερικός χρονισμός (instrumentation profiling) με χρήση της βιβλιοθήκης chrono της C++, μετρώτας την κατανάλωση χρόνου ανά τμήμα του αλγορίθμου. Η μέθοδος αυτή επέτρεψε τον εντοπισμό των σημαντικότερων περιοχών υπολογιστικού κόστους, με κύριες εξ αυτών να είναι η διαδικασία εύρεσης επαναλήψιμων

ακολουθιών (match finding) και ο υπολογισμός του token header και της αντιστοίχισης των literals μέσω hashing.

Ειδικότερα, η παραπάνω περιγραφή αντιστοιχεί στις συναρτήσεις LZ4_count και LZ4_-hash. Επιπλέον με αυτά, η εισαγωγή προσαρμοσμένης λογικής θα εξεταστεί για τον υπολογισμό της εντροπίας στον traffic manager.

3.1.4 Σχεδιασμός Custom Instructions

Η πορεία σχεδίασης ξεκίνησε με το profiling του software κώδικα LZ4 σε έναν βασικό general purpose Nios-V πυρήνα ώστε να εντοπιστούν τα σημεία συμφόρησης. Από αυτά επιλέξαμε δύο για υλοποίηση ως custom instructions, ενώ επιλέξαμε άλλο ένα custom instruction για τον traffic manager.

I. **`LZ4_COUNT` - Μέτρηση ίδιων bytes** Δέχεται δύο 32-bit λέξεις και επιστρέφει πόσα διαδοχικά leading bytes είναι ίδια (0–4). Κάνει XOR των δύο λέξεων και ελέγχει κάθε byte με συνδυαστική λογική αν το αποτέλεσμα diff=0, τότε όλα τα 4 bytes ταιριάζουν. Το αποτέλεσμα γράφεται στα κατώτερα 4 bits του result και το σήμα done γίνεται high στον ίδιο κύκλο, ώστε το pipeline να μην διαταράσσεται σημαντικά.

II. **`LZ4_HASH` - Υπολογισμός hash λεξικού** Υλοποιεί ακριβώς τον τύπο του LZ4:

$$\texttt{index} = (\texttt{seq} \times 0x9E3779B1) \gg \texttt{shift}$$

όπου seq είναι τα 4 bytes εισόδου, και shift δίνεται στα 5 LSBs του data1. Γίενται πολλαπλασιασμός 32×32 bit κρατώντας κρατάμε τα κατώτερα 32 bits και τα ολισθαίνουμε δεξιά, επιστρέφοντας τον 32-bit δείκτη λεξικού σε ένα ρολόι.

III. **`ENTROPY_CALC` - Υπολογισμός μέσω λογαριθμικής προσέγγισης** Δέχεται data0=συνολικό πλήθος συμβόλων και data1=σύνολο εμφάνισης συμβόλου. Αρχικά υπολογίζει την πιθανότητα $p = \frac{\texttt{data0}}{\texttt{data1}}$ σε μορφή Q16.16. Στην συνέχεια το p μετατοπίζεται ώστε το πιο σημαντικό bit να πάει στη θέση 23, σχηματίζοντας με αυτόν το τρόπο, δείκτη 8 bits προς ένα LUT 256 θέσεων που έχει αποθηκευμένη την προσέγγιση του $\log_2(p)$. Τέλος υπολογίζει το $-p \times \log_2(p)$ σε 64 bit, μετατοπίζει δεξιά για επιστροφή στην μορφή Q16.16 και αποθηκεύει το αποτέλεσμα στο result. Η υλοποίηση γίνεται μέσω ενός FSM 8 καταστάσεων.

Κάθε υπομονάδα ακολουθεί το πρωτόκολλο custom subordinate του Nios-V: είσοδοι ctrl, enable, δύο τελεστέων data0/1 και έξοδος result. Η σχεδίαση επιτρέπει το bypass του ALU όποτε η οδηγία δεν είναι ενεργή. Όλος ο κώδικας γράφτηκε σε VHDL για καλύτερο έλεγχο ενώ η επαλήθευση έγινε στο ModelSim.

Τα πειραματικά αποτελέσματα θα παρουσιαστούν στο επόμενο κεφάλαιο ώστε να αξιολογηθεί το κέρδος σε κύκλους.

3.2 Προτεινόμενη Αρχιτεκτονική

Επισκόπηση Συστήματος

Το προτεινόμενο σύστημα βασίζεται σε έναν μόνο Nios-V πυρήνα που τρέχει τον τροποποιημένο κώδικα του LZ4. Οι πιο αργές ρουτίνες του αλγορίθμου έχουν αντικατασταθεί από τα δυο προαναφερμένα LZ4_COUNT και LZ4_HASH custom instructions. Το `ENTROPY_CALC` προστίθεται στον traffic manager και βοηθάει στον ταχύτατο υπολογισμό της εντροπίας ενός εισερχόμενου πακέτου.

Ο traffic manager με βαση την μετρική της εντροπίας αποφασίζει αν το πακέτο θα δρομολογηθεί προς συμπίεση ή οχι.

3.3 Υλοποίηση και Περιβάλλον Ανάπτυξης

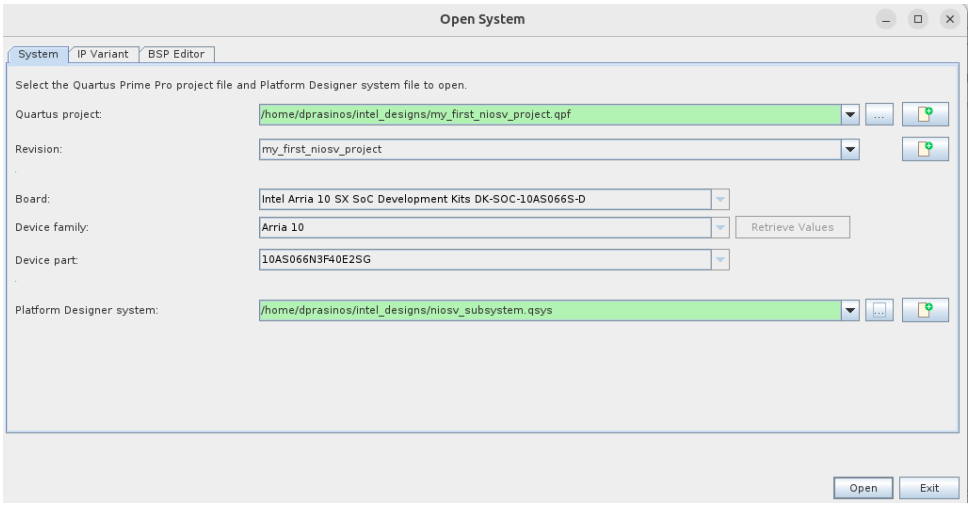
3.3.1 Εργαλεία και Εκδόσεις Λογισμικού

- **Intel Quartus Prime Pro 23.4**
- **Platform Designer (Qsys) 23.4**
- **Questa ModelSim FPGA EDITION 23.4**
- **Nios V Embedded Design Suite (EDS)**

3.3.2 Βήματα Διαδικασίας Ανάπτυξης

Ρύθμιση Project στο Quartus

1. Δημιουργία νέου project μέσω του New Project Wizard.
2. Επιλογή target FPGA
3. Προσθήκη VHDL αρχείων και ρυθμίσεων για το custom instruction.



Σχήμα 3.2: *Ρύθμιση νέου Project στον Quartus Prime Pro.*

Δημιουργία Συστήματος στον Platform Designer

1. Εισαγωγή CPU `Nios V/g`.

2. Προσθήκη αναγκαίων components όπως JTAG UART, On-Chip Memory II, Clock and Reset bridge.

3. Ορισμός master/slave interconnects.

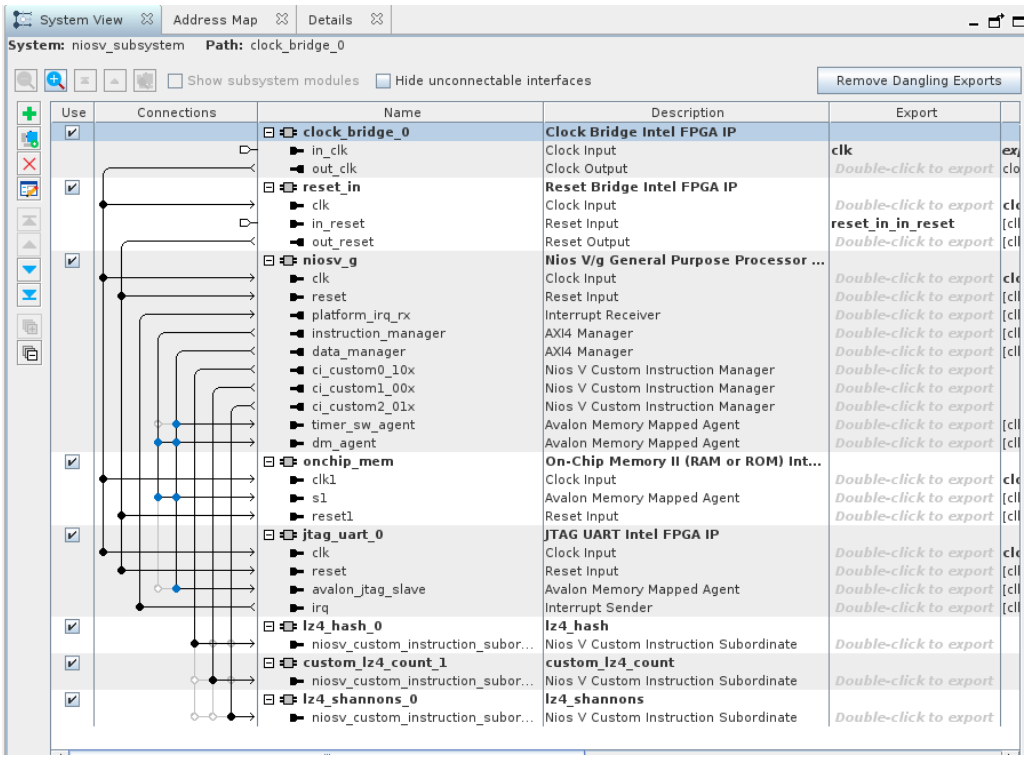
4. Δημιουργία custom instruction manager.

Ενσωμάτωση Custom Instruction

1. Δημιουργία νέου custom instruction component ως custom instruction subordinate μέσω του wizard και εισαγωγή αρχείου VHDL.

2. Προσθήκη στο σύστημα και σύνδεση με το αντίστοιχο custom instruction manager.

3. Παραγωγή bsp και αυτόματη δημιουργία custom instruction macro στο system.h με χρήση inline asm.



(α') *Τελικό System View*

```
/*
 * NiosV Custom instruction macros
 *
 */

#define lz4_count(VAL_1, VAL_2, F3, F7) ({ \
 int output; \
 asm volatile (".insn r 0x2B, %[FUNCT3], (0x0 | (%[FUNCT7]<<0)), %[out], %[input1], %[input2]" \
    : [out] "=r" (output) \
    : [input1] "r" (VAL_1), [input2] "r" (VAL_2), [FUNCT3] "i" (F3), [FUNCT7] "i" (F7)); \
    output;  \
})

#define lz4_hash(VAL_1, VAL_2, F3, F7) ({ \
 int output; \
 asm volatile (".insn r 0x0B, %[FUNCT3], (0x40 | (%[FUNCT7]<<0)), %[out], %[input1], %[input2]" \
    : [out] "=r" (output) \
    : [input1] "r" (VAL_1), [input2] "r" (VAL_2), [FUNCT3] "i" (F3), [FUNCT7] "i" (F7)); \
    output;  \
})

#define lz4_shannons(VAL_1, VAL_2, F3, F7) ({ \
 int output; \
 asm volatile (".insn r 0x5B, %[FUNCT3], (0x20 | (%[FUNCT7]<<0)), %[out], %[input1], %[input2]" \
    : [out] "=r" (output) \
    : [input1] "r" (VAL_1), [input2] "r" (VAL_2), [FUNCT3] "i" (F3), [FUNCT7] "i" (F7)); \
    output;  \
})
```

(β') *Τα παραγώμενα αυτόματα Custom Instruction Macros στο system.h*

Σχήμα 3.3: *Τελική εικόνα του Project στο Quartus και στο source code.*

Προσομοίωση στο Questa ModelSim

1. Δημιουργία testbench για τον πυρήνα.
2. Build and compilation του source κώδικα που θέλουμε να φορτώσουμε.
3. Αντιγραφή του .elf εκτελέσιμου στην μνήμη onchip_mem.
4. Εκκίνηση του Questa ModelSim τρέχοντας το παραγώμενο script msim_setup.do και κάνοντας ld για την φόρτωση του testbench.

5. Παρατήρηση σωστής λειτουργίας απο τις κυμματομορφές μέσω add wave και run.

Κεφάλαιο 4

Πειραματικά Αποτελέσματα και Αξιολόγηση Υλοποίησης

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα που προέκυψαν από τη μέτρηση της επίδοσης του τροποποιημένου LZ4 αλγορίθμου με χρήση custom instructions σε FPGA-βασισμένο RISC-V επεξεργαστή.

4.1 Πειραματικό Περιβάλλον

Για την αξιολόγηση της απόδοσης του αλγορίθμου συμπίεσης χρησιμοποιήθηκε το εκτελέσιμο αρχείο datagen από το επίσημο GitHub του LZ4. Το εργαλείο αυτό δημιουργεί τυχαία δεδομένα με την συμπιεσιμότητα που ορίζουμε εμείς. Χρησιμοποιήθηκαν μπλοκ δεδομένων μεγέθους 256, 512, 1024, 2048 και 4096 bytes.

Προκειμένου να εξεταστεί συστηματικά η συμπεριφορά του αλγορίθμου υπό διαφορετικά επίπεδα συμπιεσιμότητας, εισήχθησαν διακριτές ετικέτες εντροπίας (Entropy Labels) που χαρακτηρίζουν θεωρητικά την πληροφοριακή περιεκτικότητα των δεδομένων. Τα επίπεδα αυτά έχουν ως εξής:

- 0: Ασυμπίεστα δεδομένα (ομοιόμορφα τυχαία).
- 25, 50, 75: Ενδιάμεσου βαθμού συμπιεσιμότητα.
- 100: Υψηλής συμπιεσιμότητας δεδομένα.

Για λόγους αναφοράς και επικύρωσης, υπολογίστηκε η εντροπία Shannon για κάθε τιμή ετικέτας,σ όπως φαίνεται στον πίνακα 4.1:

Πίνακας 4.1: *Εντροπία Shannon ανά Entropy Label.*

Entropy Label	**Shannon's Entropy (bits/byte)**
0	7.5
25	5.4
50	4.5
75	3.8
100	0.0

Οι ίδιες τιμές εντροπίας χρησιμοποιήθηκαν σε όλα τα πειράματα, προκειμένου να διατηρείται η συγκρισιμότητα και η επαναληψιμότητα των αποτελεσμάτων. Οι επιδόσεις των

`LZ4_COUNT` και `LZ4_HASH` μετρήθηκαν υπό τα ίδια αυτά σενάρια για όλους τους παραπάνω συνδυασμούς μεγέθους δεδομένων και επιπέδου εντροπίας.

4.1.1 Ορισμός και Μεθοδολογία Μέτρησης *speed-up*

Η επιτάχυνση (***speed-up***) ορίζεται ως ο λόγος των κύκλων ρολογιού της software υλοποίησης ως προς το πλήθος κύκλων ρολογιού της υλοποίησης που ενσωματώνει τα custom instruction: [25]

$$S = \frac{C_{\mathrm{SW}}}{C_{\mathrm{CI}}}, \qquad S > 1 \implies \text{βελτίωση} \tag{4.1}$$

Βασική υλοποίηση (C_{SW}) Κώδικας αναφοράς `lz4.c` χωρίς καμία προσαρμοσμένη εντολή, μεταγλωττισμένος χωρίς optimization flags.

Υλοποίηση με προσαρμοσμένη εντολή (C_{CI}) Ίδιος κώδικας, με τις `LZ4_COUNT`, `LZ4_HASH` (ή και τις δύο μαζί) ενεργοποιημένες μέσω `volatile asm`.

4.2 Πλήρης Πίνακας Μετρήσεων

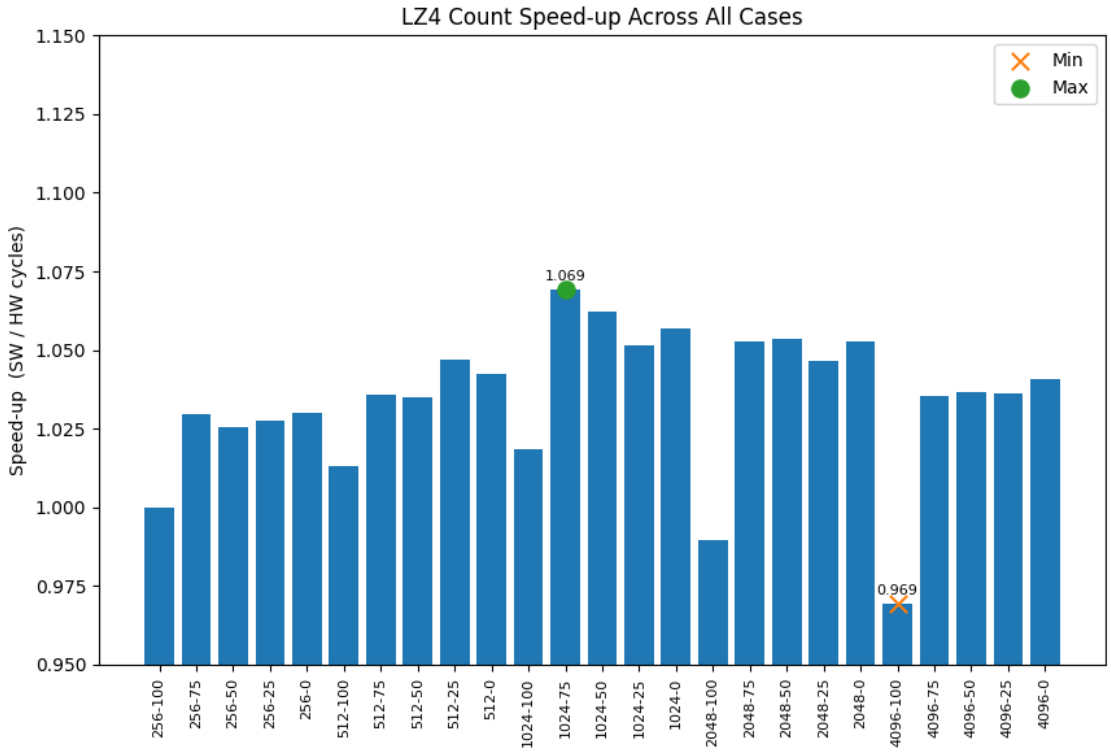
Πίνακας 4.2: *Πλήρες σύνολο πειραματικών μετρήσεων (25 δείγματα). Όλες οι τιμές δίνονται σε κύκλους ρολογιού.*

Μέγεθος (B)	Entropy Label	SW κύκλοι	`LZ4_COUNT` κύκλοι	`LZ4_HASH` κύκλοι	`COUNT+HASH` κύκλοι
256	100	38 824	38 835	38 397	38 502
256	75	102 254	99 309	100 786	97 788
256	50	126 103	122 962	125 295	120 903
256	25	124 776	121 407	123 611	119 314
256	0	102 757	99 752	102 936	99 101
512	100	46 862	46 250	45 882	45 384
512	75	111 954	108 075	109 255	106 491
512	50	186 878	180 562	184 330	177 967
512	25	225 349	215 278	220 318	212 316
512	0	145 902	139 960	144 612	138 796
1 024	100	65 104	63 918	62 620	63 795
1 024	75	227 232	212 514	215 285	207 536
1 024	50	427 211	402 260	413 839	397 224
1 024	25	341 933	325 175	333 519	320 992
1 024	0	209 675	198 356	205 277	197 144
2 048	100	98 677	99 696	96 976	99 215
2 048	75	611 954	581 340	588 339	569 882
2 048	50	581 999	552 395	566 108	544 185
2 048	25	600 538	573 785	589 921	565 694
2 048	0	297 810	282 895	292 647	281 225
4 096	100	167 298	172 579	166 172	173 020
4 096	75	974 675	941 397	940 724	911 903
4 096	50	1 118 161	1 078 560	1 089 529	1 054 846
4 096	25	984 157	949 866	964 147	934 057
4 096	0	426 668	409 892	419 306	406 877

4.3 Αποτίμηση Custom Instruction `LZ4_COUNT`

Η προσαρμοσμένη εντολή `LZ4_COUNT` με βάση τα πειραματικά δεδομένα διαπιστώθηκε ότι:

- Σε **22 από 25** σενάρια η εντολή μειώνει τους κύκλους εκτέλεσης (*speed-up* > 1).
- Ο μέσος συντελεστής επιτάχυνσης σε όλο το εύρος πακέτων/εντροπίας είναι $1.034\times$ ($\approx 3.4\,\%$ λιγότεροι κύκλοι).
- Η μέγιστη τιμή $1.069\times$ εμφανίζεται σε πακέτα $1024\,\mathrm{B}$ με entropy label 75, ενώ σε πλήρως redundant δεδομένα (entropy label 100) για μεγάλα πακέτα ($4096\,\mathrm{B}$) παρατηρείται ελαφριά επιβράδυνση ($0.97\times$).



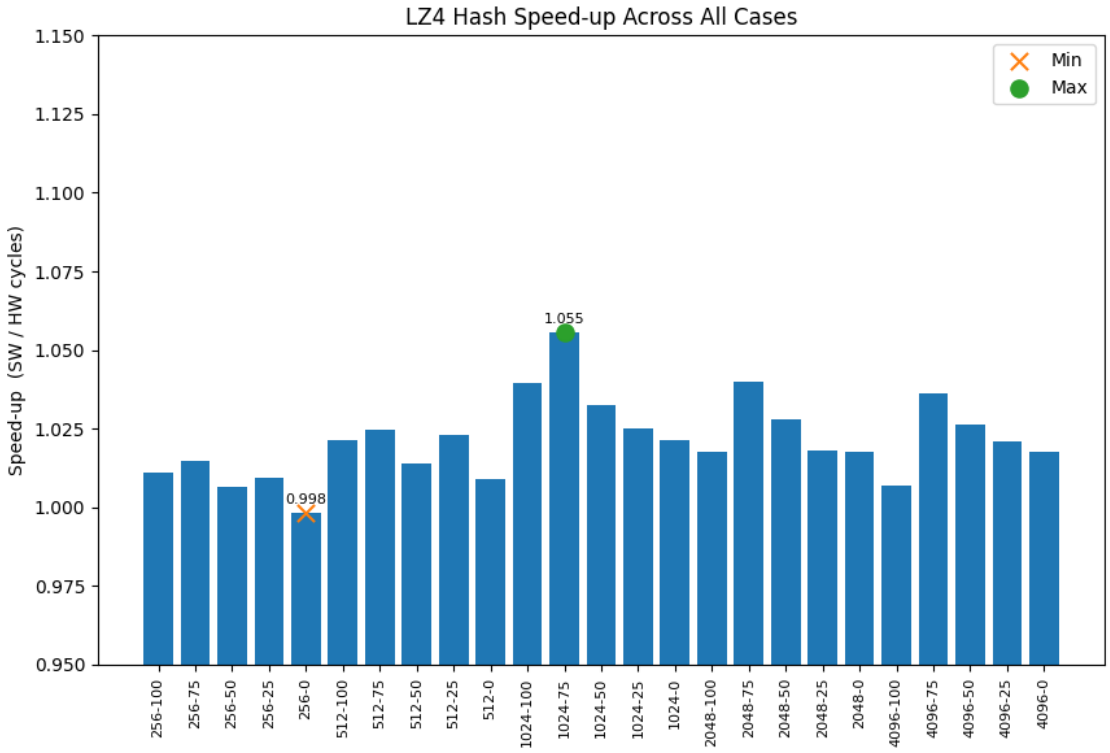
Σχήμα 4.1: *Λόγος κύκλων (software / `LZ4_COUNT`) ανα ζευγάρι πακέτου/εντροπίας.*

4.4 Αποτίμηση Custom Instruction `LZ4_HASH`

Με βάση τις μετρήσεις, η προτεινόμενη εντολή `LZ4_HASH` παρουσιάζει:

- Σε **24 από 25** σενάρια μειώνει τους κύκλους εκτέλεσης (*speed-up* > 1).
- Ο μέσος συντελεστής επιτάχυνσης σε όλο το εύρος πακέτων/εντροπίας είναι $1.021\times$ ($\approx 2.1\,\%$ λιγότεροι κύκλοι).

- Η μέγιστη τιμή 1.055× εμφανίζεται σε πακέτα 1024 B με entropy label 75, ενώ σε πλήρως συμπιεστά δεδομένα (label 0) για πολύ μικρά πακέτα (256 B) παρατηρείται η ελάχιστη τιμή 0.998×.

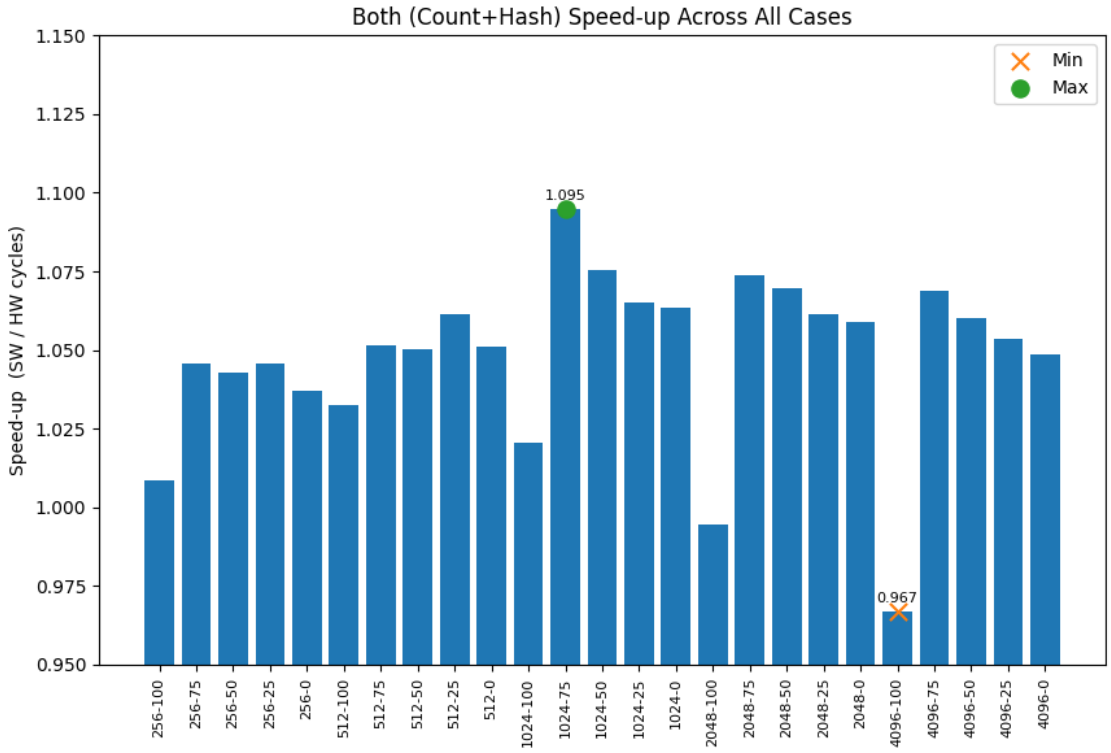


Σχήμα 4.2: *Λόγος κύκλων (software / `LZ4_HASH`) ανα ζευγάρι πακέτου/εντροπίας.*

4.5 Συνδυαστική Βελτιστοποίηση (`LZ4_COUNT` + `LZ4_HASH`)

Όταν και οι δύο εντολές χρησιμοποιούνται ταυτόχρονα (`both_cycles`), παρατηρούνται οι εξής τάσεις:

- **23 από 25** περιπτώσεις παρουσιάζουν βελτίωση.
- Ο μέσος συντελεστής επιτάχυνσης ανέρχεται σε 1.048× (≈4.8 %).
- Η μέγιστη τιμή 1.095× εμφανίζεται σε πακέτα 1024 B με entropy label 75, ενώ μικρό αρνητικό speedup (0.99×–0.97×) περιορίζεται σε πλήρως redundant δεδομένα, 2048 B και 4096 B.



Σχήμα 4.3: *Συνδυαστικό όφελος (LZ4_COUNT+LZ4_HASH).*

4.6 Ανάλυση Σεναρίων μη βελτίωσης

Παρότι η συντριπτική πλειοψηφία των μετρήσεων δείχνει καθαρή επιτάχυνση, εντοπίστηκαν λίγα σενάρια που δεν παρατηρήθηκε βελτίωση, αλλά ήπια επιβράδυνση. Τα σενάρια συνοψίζονται στον πίνακα 4.3.

Πίνακας 4.3: *Σενάρια με speed-up* ≤ 1

Εντολή	Μέγεθος pkt	Entropy Label	LZ4_COUNT Calls/pkt	LZ4_HASH Calls/pkt	Speed-up
LZ4_COUNT	256 B	100	3	-	0.99×
	2048 B	100	2	-	0.99×
	4096 B	100	2	-	0.97×
LZ4_HASH	256 B	0	-	150	0.998×
COUNT+HASH	2048 B	100	2	10	0.99×
	4096 B	100	2	10	0.97×

4.6.1 Αίτια και κοινά χαρακτηριστικά

1. **Σταθερό κόστος κλήσης.** Κάθε ενεργοποίηση προσαρμοσμένης εντολής φαίνεται να επιφέρει καθυστερήσεις στο pipeline της τάξης των 2–3 *kcycles*. Όταν οι κλήσεις της LZ4_COUNT είναι ≤ 5 ανά πακέτο, το κόστος αυτό δεν αποσβένεται (βλ. γραμμές 1–3 του Πιν. 4.3). Ενδεικτικά στην περίπτωση μέγιστου speedup 9.5% έχουμε 11 κλήσεις της LZ4_COUNT ανά πακέτο και 292 κλήσεις της LZ4_HASH ανά πακέτο. Προφανώς παρατηρείται συσχέτιση του αριθμού κλήσεων με το speed-up.

2. **Χαμηλή πυκνότητα κλήσεων σε πακέτα πολύ χαμηλής εντροπίας.** Σε δεδομένα υψηλού redundancy ο αλγόριθμος στο λογισμικό είναι βελτιστοποιημένος, ώστε να αποφεύγει περιττή δουλειά, με αυτό τον τρόπο ελαχιστοποιεί τις κλήσεις στο custom instruction. Σαφώς με αυτό τον τρόπο πληρώνει το λειτουργικό κόστος κλήσης (overhead), χωρίς να παίρνει πραγματικό όφελος.

3. **Απαιτούμενη πυκνότητα κλήσεων.** Η συσχέτιση speedup - custom instruction calls παραμένει θετική για όλο το υπόλοιπο φάσμα ήδη από ~ 10 κλήσεις/πακέτο για την LZ4_COUNT το καθαρό όφελος υπερβαίνει το 2 %, ενω όπως αναφέραμε πριν, για πακέτα 4096 B με 118 κλήσεις το κέρδος φτάνει 9.5 %.

Τα παραπάνω με την εισαγωγή ενός Traffic Manager αναμένονται να εξαληφθούν.

4.7 Συγκεντρωτικός Πίνακας Επιδόσεων

Ο Πίνακας 4.4 συνοψίζει τον μέσο, ελάχιστο και μέγιστο συντελεστή επιτάχυνσης που προέκυψε *αποκλειστικά* από το σύνολο μετρήσεων των πειραμάτων (25 δείγματα).

Πίνακας 4.4: *Σύνοψη επιταχύνσεων (25 δείγματα).*

Μετρική	**Μέση**	**Ελάχιστη**	**Μέγιστη**
`LZ_COUNT`	1.034×	0.969×	1.069×
`LZ_HASH`	1.021×	0.998×	1.055×
`COUNT+HASH`	1.048×	0.967×	1.095×

Τα αποτελέσματα δείχνουν ότι τα κέρδη σε κύκλους είναι μετριοπαθή, έως 9.5 % και εξαρτώνται έντονα από το μέγεθος πακέτου και τον βαθμό εντροπίας. Παρά τις περιορισμένες τιμές *speed-up*, η ISA επέκταση αποφεύγει σημαντικές καθυστερήσεις στις περισσότερες περιπτώσεις, χωρίς να έχουμε επιβράδυνση πέραν του 3.3 %.

4.7.1 Φιλτραρισμένο σύνολο με χρήση του Traffic Manager

Ο *traffic-manager* απορρίπτει πακέτα που θεωρούνται (α) πλήρως ασυμπίεστα (**entropy label 100**) και (β) πλήρως συμπιεστά (**entropy label 0**). Ουσιαστικά στην πράξη θεωρούμε σταθερό κατώφλι αποκοπής για τον traffic manager πακέτα με εντροπία πάνω απο 7 και κάτω απο 3. Τα πλήρως συμπιεστά πακέτα, όπως σχολιάσαμε και προηγουμένως, λόγω των βελτιστοποιήσεων και των guards του LZ4, συμπιέζονται πιο αποδοτικά στο software. Σε αυτή την περίπτωση αυτά τα πακέτα μπορούν *δυναμικά* να αποφασιστεί να ολοκληρωθεί η συμπίεση τους στο λογισμικό.

Αφαιρώντας τα συγκεκριμένα δείγματα (10 από τα 25) βελτιώνεται η συνοχή των μετρήσεων, ενώ εξαφανίζονται τα αρνητικά *speed-up* για το `LZ4_COUNT` αλλά και τον συνδυασμό των δύο εντολών.

Πίνακας 4.5: *Συντελεστής επιτάχυνσης (speed-up) μετά το φιλτράρισμα των entropy labels 0 και 100.*

Μετρική	Μέση	Ελάχιστη	Μέγιστη
LZ_COUNT	1.043×	1.026×	1.069×
LZ_HASH	1.025×	1.006×	1.055×
COUNT+HASH	1.061×	1.043×	1.095×

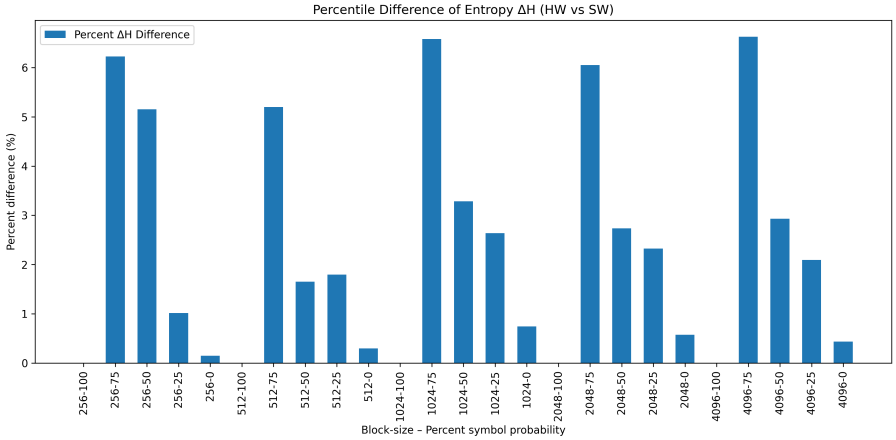
Η νέα ελάχιστη τιμή για τον συνδυασμό (**1.043**×) υποδηλώνει ότι η προσέγγιση αποφεύγει πρακτικά κάθε επιβράδυνση, ενώ το μέσο όφελος αυξάνεται στο 6.1 %.

4.8 Αποτίμηση Custom Instruction ENTROPY_CALC

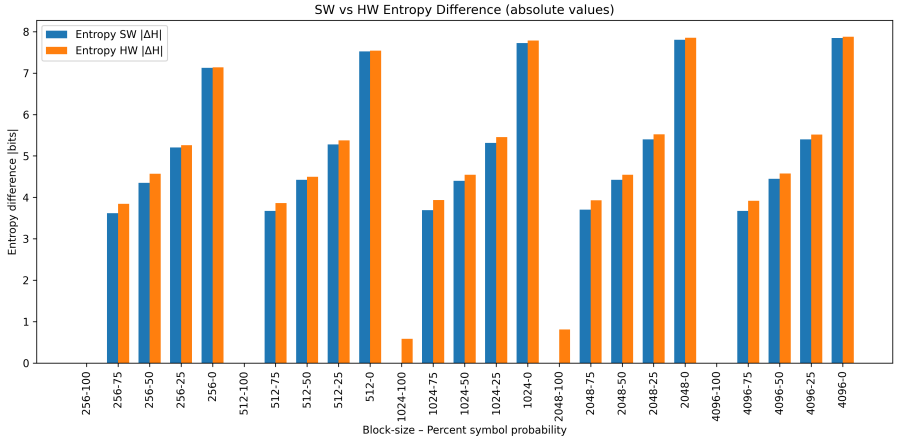
Η ENTROPY_CALC υλοποιεί μια lookup-table (LUT) προσέγγιση για τη στιγμιαία εκτίμηση της εντροπίας ενός πακέτου κατά τη ροή. Στόχος είναι η έγκαιρη απόφαση εάν ένα πακέτο αξίζει να συμπιεστεί, αποφεύγοντας περιττούς κύκλους σε μη συμπιεστά δεδομένα.

4.8.1 Αποτελέσματα ακρίβειας

Στα Σχ. 4.4 και 4.5 καταγράφεται η διαφοροποίηση της εκτιμώμενης εντροπίας HW έναντι SW. Η μέγιστη ποσοστιαία απόκλιση παραμένει κάτω από 6.6 % (μέγιστη τιμή για μπλοκ 4096 B με entropy label 75), ενώ ο μέσος όρος είναι ≈ 2.8 %. Η μικρή αυτή διαφορά θεωρείται αμελητέα για τον σκοπό λήψης απόφασης.



Σχήμα 4.4: *Ποσοστιαία διαφορά προσέγγισης εντροπίας, σε σχέση με την ακριβή τιμή.*

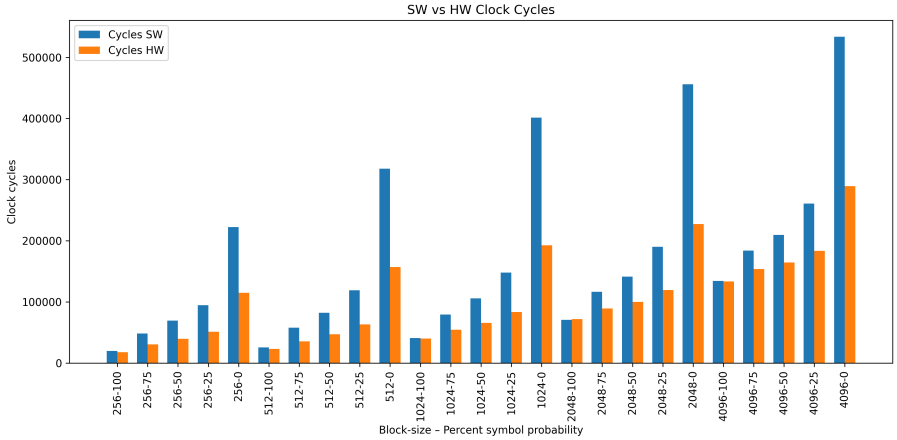


Σχήμα 4.5: *Διαφορά* $|\Delta H|$.

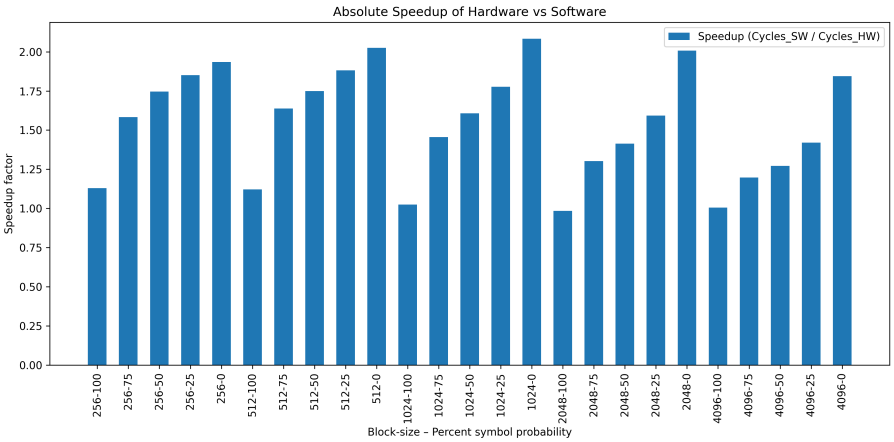
4.8.2 Αποτελέσματα απόδοσης

Το διάγραμμα κύκλων ρολογιού (Σχ. 4.6) δείχνει σταθερά λιγότερους κύκλους στην HW εκτέλεση ο λόγος Speed-up (Σχ. 4.7) κυμαίνεται από $1.1\times$ έως $2.1\times$, με μέσο όρο $\approx 1.6\times$, δηλαδή 40 % μείωση χρόνου εκτέλεσης. Έτσι η προσθήκη της εντολής απελευθερώνει πολύτιμους κύκλους για άλλες λειτουργίες.

Με βάση την βιβλιογράφια και όπως παρατέθηκε στο 2.5, η χρήση της LUT-based `ENTROPY_CALC` σε ένα μόνο υποσύνολο συμβόλων ή ακόμη σε λίγα packets είναι στατιστικά έγκυρη και πρακτικά αποδεδειγμένη μάλιστα μειώνει περαιτέρω τους απαιτούμενους κύκλους σε πακέτα μεγάλου μήκους.

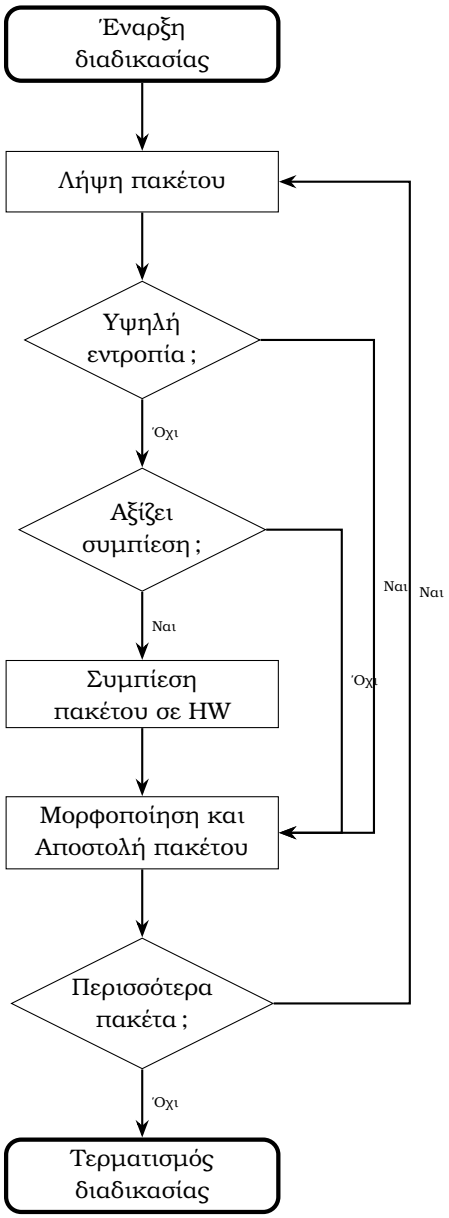


Σχήμα 4.6: *SW vs HW κύκλοι ρολογιού.*



Σχήμα 4.7: *Απόλυτη τιμή Speed-up.*

4.9 Διάγραμμα Ροής Λογικής Traffic Manager



Σχήμα 4.8: *Διάγραμμα ροής του Traffic Manager.*

Μέρος III

Επίλογος

Κεφάλαιο 5

Συμπεράσματα και Μελλοντική Εργασία

Στο κεφάλαιο αυτό αναφερόμαστε στα τελικά συμπεράσματα ενώ παράλληλα περιγράφονται πιθανές επεκτάσεις και βελτιώσεις που μπορούν να πραγματοποιηθούν πάνω στο προτεινόμενο σύστημα, με στόχο την περαιτέρω αύξηση των επιδόσεων, την ευελιξία στη χρήση και την κάλυψη πρόσθετων εφαρμογών.

5.1 Συμπεράσματα

Μελετώντας τον στόχο για όσο το δυνατόν ταχύτερο compression, η εργασία μας απέδειξε ότι η προτεινόμενη επέκταση του RISC-V ISA σε FPGA με τα custom instructions `LZ4_COUNT`, `LZ4_HASH`, `ENTROPY_CALC` μπορεί να επιταχύνει τα κρίσιμα τμήματα του αλγορίθμου LZ4 χωρίς να απαιτεί την πολυπλοκότητα μιας πλήρους σχεδίασης στο υλικό, απολαμβάνοντας με αυτόν τον τρόπο την ευελιξία του λογισμικού και την ταχύτητα του υλικού[25].

Με βάση την μεθοδολογία που ακολουθήθηκε (Σχήμα 3.1) και τελικώς την αρχιτεκτονική που σχεδιάστηκε (ενότητα 3.2), τα αποτελέσματα της εργασίας είναι ενθαρρυντικά. Με βάση τα 25 αντιπροσωπευτικά σενάρια που εξετάστηκαν (Πίνακα 4.2), παρατηρούμε πώς το μέσο speedup για την υλοποίηση με το `LZ4_COUNT` ήταν 1.034× με μέγιστη τιμή 1.69×, για το `LZ4_HASH` ήταν 1.021× με μέγιστη τιμή 1.055× ενώ τέλος για τον συνδυασμό των δύο ήταν 1.048× (Πίνακα 4.4) με μέγιστη τιμή 1.095×. Στην τελική προτεινόμενη αρχιτεκτονική με την εισαγωγή του traffic manager, πετύχαμε μέσο speedup 1.061× με μέγιστη τιμή 1.095× (Πίνακα 4.5).

Η `ENTROPY_CALC` μείωσε τον χρόνο λήψης απόφασης έως 2.1× (Σχήμα 4.7), επιτρέποντας στον *traffic manager* να αποκλείει τα πακέτα με υψηλή εντροπία με ακόμα μεγαλύτερη ταχύτητα.κεφάλαιο 3.

Συνοψίζοντας, το προτεινόμενο σύστημα αποδεδειγμένα επιτυγχάνει την απαιτούμενη ταχύτητα λόγω της φύσης του LZ4 ενώ προσφέρει βελτίωση απόδοσης της τάξης του 4 με 10% με σχεδόν μηδενική επιβάρυνση σε σχέση με την υπάρχουσα υλοποίηση του LZ4 στο λογισμικό.

5.2 Μελλοντικές επεκτάσεις

Τα αποτελέσματα, αν και ενθαρρυντικά, μας ανοίγουν νεα μονοπάτια ευρύτερης έρευνας όπως:

- **Πολυ-πύρηνη κλιμάκωση.** Η παράλληλη χρήση πολλαπλών πυρήνων *Nios-V* με παρουσία κάποιου κοινού *scratch-pad* θεωρητικά μας υπόσχεται γραμμική αύξηση σε throughput και ταχύτητα. Σε μια τέτοια προσέγγιση ωστόσο, απαιτείται πιο εκτενής διερεύνηση των μηχανισμών του LZ4, όπως αυτοί του hashing και του match finding, ώστε να αξιοποιηθεί αποδοτικά αυτή η παραλληλοποίηση.

- **HLS** Παρότι ο τρέχων σχεδιασμός των custom intruction σε vhdl μας παρείχε τον μέγιστο δυνατό έλεγχο στο υλικό, ενδεχομένως μία πειραματική υλοποίηση των custom instructions με εργαλεία hls, θα μας δείξει αν υφίσταται πιθανό κέρδος στην παραγωγικότητα και την πολυπλοκότητα έναντι απωλειών στην απόδοση.

- **Δυναμικό κατώφλι εντροπίας.** Το σταθερό κατώφλι που χρησιμοποίησαμε στον traffic manager (Σχήμα 4.8) θα μπορούσε να αντικατασταθεί από ένα δυναμικό σχήμα, ακόμα και από reinforcement learning, ώστε το σύστημα να «μαθαίνει» το βέλτιστο σημείο μεταξύ απόδοσης και compression ratio.

Τελικό συμπέρασμα: Η παρούσα εργασία απέδειξε πως τα RISC-V custom instructions είναι ενας βιώσιμος και αποδοτικός τρόπος για την μείωση της καθυστέρησης σε συνδιαστικά συστήματα που αξιοποιούν τα FPGA και RISC-V. Οι παραπάνω προτάσεις αποτελούν ενδεικτικές κατευθύνσεις για τη μελλοντική ανάπτυξη του προτεινόμενου συστήματος. Ανάλογα με τις απαιτήσεις της εκάστοτε εφαρμογής, η έμφαση μπορεί να δοθεί σε διαφορετικές περιοχές βελτίωσης. Η επεκτασιμότητα και η προσαρμοστικότητα του RISC-V ISA, σε συνδυασμό με την ευελιξία των FPGA, παρέχουν ένα γόνιμο έδαφος για καινοτόμες παρεμβάσεις στον χώρο της χαμηλής καθυστέρησης και της υψηλής απόδοσης στη συμπίεση δεδομένων.

Παραρτήματα

Παράρτημα Α΄

Αποσπάσματα Κώδικα σε VHDL και σε C

Listing A.1: Custom Instruction LZ4_COUNT

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY lz4_custom_inst IS
6      PORT (
7          clk : IN STD_LOGIC;
8          reset : IN STD_LOGIC;
9          ctrl : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
10         enable : IN STD_LOGIC;
11         data0 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
12         data1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
13         alu_result : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
14         result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
15         done : OUT STD_LOGIC
16     );
17 END lz4_custom_inst;
18
19 ARCHITECTURE behavioral OF lz4_custom_inst IS
20     CONSTANT ZERO32 : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
21 BEGIN
22     PROCESS (clk, reset)
23         VARIABLE diff : STD_LOGIC_VECTOR(31 DOWNTO 0);
24         VARIABLE byte_count : INTEGER RANGE 0 TO 4;
25         VARIABLE tmp_byte : STD_LOGIC_VECTOR(7 DOWNTO 0);
26     BEGIN
27         IF reset = '1' THEN
28             result <= (OTHERS => '0');
29             done <= '0';
30         ELSIF rising_edge(clk) THEN
31             -- both ctrl and enable must be 1
32             IF (ctrl(0) = '1') AND (enable = '1') THEN
33                 diff := data0 XOR data1;
34                 IF diff = ZERO32 THEN
35                     byte_count := 4;
36                 ELSE
37                     byte_count := 0;
38                     -- check each of the 4 bytes
```

```
39                  FOR i IN 0 TO 3 LOOP
40                      tmp_byte := diff((i + 1) * 8 - 1 DOWNTO i * 8);
41                      IF tmp_byte /= x"00" THEN
42                          EXIT; -- first mismatch byte stop
43                      ELSE
44                          byte_count := byte_count + 1;
45                      END IF;
46                  END LOOP;
47              END IF;
48              -- place the result in lower 4 bits into result. upper bits zero
49              result <= (OTHERS => '0');
50              result(3 DOWNTO 0) <= STD_LOGIC_VECTOR(to_unsigned(byte_count, 4));
51              done <= '1';
52          ELSE
53              -- pass through alu_result
54              result <= alu_result;
55              done <= '0';
56          END IF;
57      END IF;
58  END PROCESS;
59  END behavioral;
```

Listing A.2: Custom Instruction LZ4_HASH

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.numeric_std.ALL;
4
5   ENTITY lz4_hash_inst IS
6       PORT (
7           clk : IN STD_LOGIC;
8           reset : IN STD_LOGIC;
9           ctrl : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
10          enable : IN STD_LOGIC;
11          data0 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
12          data1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
13          alu_result : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
14          result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
15          done : OUT STD_LOGIC
16      );
17  END lz4_hash_inst;
18
19  ARCHITECTURE single_cycle OF lz4_hash_inst IS
20
21      CONSTANT LZ4_PRIME : unsigned(31 DOWNTO 0) := x"9E3779B1";
22
23  BEGIN
24
25      PROCESS (clk, reset)
26          VARIABLE product64 : unsigned(63 DOWNTO 0);
27          VARIABLE product32 : unsigned(31 DOWNTO 0);
28          VARIABLE shiftVal : NATURAL RANGE 0 TO 31;
29      BEGIN
```

```
        IF rising_edge(clk) THEN
            IF reset = '1' THEN
                result <= (OTHERS => '0');
                done <= '0';
            ELSE
                -- default pass through each cycle
                result <= alu_result;
                done <= '0';

                IF (ctrl(0) = '1') AND (enable = '1') THEN
                    -- 64bit multiply
                    product64 := unsigned(data0) * LZ4_PRIME;

                    -- low 32 bits overflow
                    product32 := product64(31 DOWNTO 0);

                    shiftVal := to_integer(unsigned(data1(4 DOWNTO 0)));
                    product32 := product32 SRL shiftVal;

                    result <= STD_LOGIC_VECTOR(product32);
                    done <= '1';
                END IF;
            END IF;
        END IF;
    END PROCESS;

END single_cycle;
```

Listing A.3: Custom Instruction ENTROPY_CALC

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY entropy_custom_instruction IS
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        ctrl : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        enable : IN STD_LOGIC;
        data0 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        data1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        alu_result : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        done_o : OUT STD_LOGIC
    );
END entropy_custom_instruction;

ARCHITECTURE rtl OF entropy_custom_instruction IS

    -- Q16.16 shift
    CONSTANT SHIFT : INTEGER := 16;

```

```
24      TYPE lut_array IS ARRAY (0 TO 255) OF STD_LOGIC_VECTOR(31 DOWNTO 0);
25      CONSTANT LOG2_LUT : lut_array := (
26          x"00000000",
27          x"00000171",
28               --
29               --
30               -- full LUT values here
31               --
32               --
33          x"0000FE8E",
34          x"0000FF47"
35
36      );
37
38      FUNCTION clz24(v : unsigned(23 DOWNTO 0)) RETURN unsigned IS
39          VARIABLE cnt : unsigned(4 DOWNTO 0) := (OTHERS => '0');
40      BEGIN
41          -- scan from msb to lsb
42          FOR i IN 23 DOWNTO 0 LOOP
43              IF v(i) = '1' THEN
44                  RETURN cnt; -- found the first 1
45              ELSE
46                  cnt := cnt + 1; -- one more leading zero
47              END IF;
48          END LOOP;
49          RETURN cnt;
50      END FUNCTION;
51
52      TYPE state_type IS (
53          S_IDLE,
54          S_COMPUTE,
55          S_LOOKUP,
56          S_LOOKUP2,
57          S_LUTOUT,
58          S_MULT,
59          S_SHIFT,
60          S_NEGATE,
61          S_DONE
62      );
63
64      SIGNAL r_state : state_type := S_IDLE;
65
66      SIGNAL s_data0 : unsigned(31 DOWNTO 0) := (OTHERS => '0');
67      SIGNAL s_data1 : unsigned(31 DOWNTO 0) := (OTHERS => '0');
68
69      SIGNAL p_fixed : unsigned(31 DOWNTO 0) := (OTHERS => '0');
70      SIGNAL lut_index : unsigned(7 DOWNTO 0) := (OTHERS => '0');
71
72      SIGNAL log2p_fixed : signed (31 DOWNTO 0) := (OTHERS => '0');
73      SIGNAL product128 : signed (127 DOWNTO 0) := (OTHERS => '0');
74      SIGNAL shifted_val : signed (31 DOWNTO 0) := (OTHERS => '0');
75      SIGNAL result_r : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
```

```
76
77      SIGNAL shifted_p : unsigned(31 DOWNTO 0);
78      SIGNAL shift_cnt : unsigned(4 DOWNTO 0);
79
80  BEGIN
81      result <= result_r;
82      done_o <= '1' WHEN r_state = S_DONE ELSE
83          '0';
84
85      PROCESS (clk, reset)
86      BEGIN
87          IF reset = '1' THEN
88              r_state <= S_IDLE;
89              result_r <= (OTHERS => '0');
90          ELSIF rising_edge(clk) THEN
91              CASE r_state IS
92                      ------------------------------------------------------------
93                  WHEN S_IDLE =>
94                      IF enable = '1' THEN
95                          s_data0 <= unsigned(data0);
96                          s_data1 <= unsigned(data1);
97                          r_state <= S_COMPUTE;
98                      END IF;
99
100                     ------------------------------------------------------------
101                 WHEN S_COMPUTE =>
102                     IF s_data1 /= 0 THEN
103                         p_fixed <= (s_data0 SLL SHIFT) / s_data1; -- Q16.16
104                     ELSE
105                         p_fixed <= (OTHERS => '0');
106                     END IF;
107                     r_state <= S_LOOKUP;
108
109                     ------------------------------------------------------------
110                 WHEN S_LOOKUP =>
111                     IF p_fixed = 0 THEN -- probability 0, term 0
112                         lut_index <= (OTHERS => '0');
113                         shift_cnt <= (OTHERS => '0');
114                         shifted_p <= (OTHERS => '0');
115                     ELSE
116                         -- 1) l e f t shift until bit 23 becomes 1
117                         shift_cnt <= clz24(p_fixed(23 DOWNTO 0));
118                         shifted_p <= resize(p_fixed, 32) SLL to_integer(shift_cnt);
119                         lut_index <= shifted_p(22 DOWNTO 15);
120
121                     END IF;
122                     r_state <= S_LOOKUP2;
123
124                     ------------------------------------------------------------
125
126                 WHEN S_LOOKUP2 => -- one idle cycle
127                     r_state <= S_LUTOUT;
```

```
128
129                 ------------------------------------------------------------
130             WHEN S_LUTOUT =>
131                 -- fraction part (Q16.16, negative)
132                 log2p_fixed <= signed(LOG2_LUT(to_integer(lut_index)))
133                     - ((resize(signed('0' & shift_cnt), 32) - to_signed(7, 32)) SLL 16);
134
135                 r_state <= S_MULT;
136
137                 ------------------------------------------------------------
138             WHEN S_MULT =>
139                 product128 <= resize(signed(p_fixed), 64) *
140                     resize(log2p_fixed, 64);
141                 r_state <= S_SHIFT;
142
143                 ------------------------------------------------------------
144             WHEN S_SHIFT =>
145                 shifted_val <= shift_right(product128(63 DOWNTO 0), SHIFT)(31 DOWNTO 0);
146                 r_state <= S_NEGATE;
147
148                 ------------------------------------------------------------
149             WHEN S_NEGATE =>
150                 result_r <= STD_LOGIC_VECTOR(shifted_val); --    plog2 (p)
151                 r_state <= S_DONE; --  done next clk
152
153                 ------------------------------------------------------------
154             WHEN S_DONE =>
155                 -- result & DONE are valid this cycle.
156                 -- if CPU issues start here accept it.
157                 IF enable = '1' THEN
158                     s_data0 <= unsigned(data0);
159                     s_data1 <= unsigned(data1);
160                     r_state <= S_COMPUTE; -- go straight into next calc
161                 ELSE
162                     r_state <= S_IDLE; -- otherwise return to idle
163                 END IF;
164         END CASE;
165     END IF;
166 END PROCESS;
167 END ARCHITECTURE rtl;
```

Listing A.4: Η εισαγωγή του custom instruction LZ4_COUNT στην αντίστοιχη ρουτίνα σε C

```
1 #define STEPSIZE sizeof(reg_t)
2 LZ4_FORCE_INLINE unsigned LZ4_count(const BYTE* pIn, const BYTE* pMatch, const BYTE* pInLimit) {
3     const BYTE* const pStart = pIn;
4
5     while (likely(pIn < pInLimit - (STEPSIZE-1))) {
6         uint32_t wordIn   = LZ4_read_ARCH(pIn);
7         uint32_t wordMatch= LZ4_read_ARCH(pMatch);
8
9         uint32_t count    = lz4_count(wordIn, wordMatch, 0, 0) & 0xF;
```

```
        if (count == 4) {
            pIn += 4;
            pMatch += 4;
        } else {
            pIn += count;
            return (unsigned)(pIn - pStart);
        }
    }
    while (likely(pIn < pInLimit && *pIn == *pMatch)) {
        pIn++;
        pMatch++;
    }
    return (unsigned)(pIn - pStart);
}


#define STEPSIZE sizeof(reg_t)
LZ4_FORCE_INLINE
unsigned LZ4_count_sw(const BYTE* pIn, const BYTE* pMatch, const BYTE* pInLimit)
{
    const BYTE* const pStart = pIn;
    if (likely(pIn < pInLimit-(STEPSIZE-1))) {
        reg_t const diff = LZ4_read_ARCH(pMatch) ^ LZ4_read_ARCH(pIn);
        if (!diff) {
            pIn+=STEPSIZE; pMatch+=STEPSIZE;
        } else {
            return LZ4_NbCommonBytes(diff);
    }   }

    while (likely(pIn < pInLimit-(STEPSIZE-1))) {
        reg_t const diff = LZ4_read_ARCH(pMatch) ^ LZ4_read_ARCH(pIn);
        if (!diff) { pIn+=STEPSIZE; pMatch+=STEPSIZE; continue; }
        pIn += LZ4_NbCommonBytes(diff);
        return (unsigned)(pIn - pStart);
    }

    if ((STEPSIZE==8) && (pIn<(pInLimit-3)) && (LZ4_read32(pMatch) == LZ4_read32(pIn))) { pIn+=4; pMatch+=4; }
    if ((pIn<(pInLimit-1)) && (LZ4_read16(pMatch) == LZ4_read16(pIn))) { pIn+=2; pMatch+=2; }
    if ((pIn<pInLimit) && (*pMatch == *pIn)) pIn++;
    return (unsigned)(pIn - pStart);
}
```

Listing A.5: Η εισαγωγή του custom instruction LZ4_HASH στην αντίστοιχη ρουτίνα σε C

```
LZ4_FORCE_INLINE U32 LZ4_hash4(U32 sequence, tableType_t const tableType)
{
    if (tableType == byU16) {
        return lz4_hash(sequence, (MINMATCH*8)-(LZ4_HASHLOG+1), 0, 0);
    } else {
        return lz4_hash(sequence, (MINMATCH*8)-LZ4_HASHLOG, 0, 0);
    }
}
```

```
9
10
11 LZ4_FORCE_INLINE U32 LZ4_hash4(U32 sequence, tableType_t const tableType)
12 {
13     if (tableType == byU16)
14         return ((sequence * 2654435761U) >> ((MINMATCH*8)-(LZ4_HASHLOG+1)));
15     else
16         return ((sequence * 2654435761U) >> ((MINMATCH*8)-LZ4_HASHLOG));
17 }
```

Παράρτημα Β΄

Παράδειγμα Δομής Εκτέλεσης Πειράματος εντός NIOS-V Shell

Listing B.1: *Βήματα δημιουργίας BSP, μεταγλώττισης εφαρμογών και προσομοίωσης*

```
# --- 1. Make Project Tree ---
mkdir -p sw/bsp
mkdir -p sw/app

# --- 2. Generate BSP ---
niosv-bsp -c -p=my_niosv_turbo.qpf -s=niosv_sub_system.qsys \
          -t=hal ./sw/bsp/settings.bsp

# --- 3. Generate Elf ---
niosv-app --bsp-dir=sw/bsp --app-dir=sw/app \
          --srcs=sw/app/hello.c --elf-name=myapp.elf

cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug \
      -B sw/app/debug -S sw/app
make -C sw/app/debug/

# --- 4. Produce hex file from elf and copy it to the simulation memory ---
elf2hex sw/app/debug/myapp.elf -b 0x0 -w 32 -e 0xfffff sw/app/debug/myapp.hex
cp sw/app/debug/myapp.hex niosv_sub_system_tb/niosv_sub_system_tb/sim/mentor/onchip_mem.hex

# --- 5. Start simulation in Questa ---
vsim -c -do "cd ~/intel_designs/niosv_sub_system_tb/niosv_sub_system_tb/sim/mentor; \
             do msim_setup.tcl; ld; run 40000 ms; quit"
```

Βιβλιογραφία

[1] W3Techs. *Usage Statistics of Compression for Websites*. https://w3techs.com/technologies/details/ce-compression, 2025. Snapshot 21 May 2025.

[2] Apeksha Telecom. *Analyzing Latency in 4G and 5G Networks(updated in 2024)*. https://www.telecomgurukul.com/post/analyzing-latency-in-4g-and-5g-networks-updated-in-2024, 2024. Snapshot 21 May 2025.

[3] Andrew Waterman και Krste Asanović. *The RISC-V Instruction Set Manual: Volume I, Unprivileged ISA*. Τεχνική Αναφορά με αριθμό 20191214, RISC-V Foundation, 2019.

[4] David Salomon και Giovanni Motta. *Handbook of Data Compression*. Springer, 5η έκδοση, 2010. DOI: https://doi.org/10.1007/978-1-84882-903-9.

[5] Claude E. Shannon. *A Mathematical Theory of Communication*. *Bell System Technical Journal*, 27(3–4):379–423, 1948. DOI: https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[6] Thomas M. Cover και Joy A. Thomas. *Elements of Information Theory*. Wiley, 2η έκδοση, 2006. ISBN: 978-0471241959.

[7] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. ISBN: 978-0521642989.

[8] E.T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003. ISBN: 978-0521592710.

[9] David A. Huffman. *A Method for the Construction of Minimum-Redundancy Codes*. *Proceedings of the IRE*, 40(9):1098–1101, 1952. DOI: https://doi.org/10.1109/JRPROC.1952.273898.

[10] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. Τεχνική Αναφορά με αριθμό 1951, IETF, 1996. URL: https://www.rfc-editor.org/rfc/rfc1951.

[11] Jacob Ziv και Abraham Lempel. *A Universal Algorithm for Sequential Data Compression*. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. DOI: https://doi.org/10.1109/TIT.1977.1055714.

[12] Y. Collet. *LZ4 - Extremely Fast Compression*. https://lz4.org/. Snashot 21 May 2025.

[13] Kyungsik Lee. *LZ4 Compression and Improving Boot Time*. *LinuxCon Japan*, 2013. URL: https://events.static.linuxfound.org/sites/events/files/lcjpcojp13_klee.pdf.

[14] Google LLC. *Snappy - A Fast Compressor/Decompressor*. https://github.com/google/snappy, 2024. Accessed: 13 Jun 2025.

[15] *Lempel-Ziv-Oberhumer (LZO) - Wikipedia*. URL: https://en.wikipedia.org/wiki/Lempel-Ziv-Oberhumer Accessed: 13 June 2025 .

[16] Jonghak Kim. *LZO data compression functions and improvements in Intel Integrated Performance Primitives*, 2017. Intel Developer Article ID 659816, URL: https://www.intel.com/content/www/us/en/developer/articles/technical/lzo-data-compression-functions-and-improvements-in-intel-integrated-performance-primitives.html.

[17] Xiaoling Lai, Jian Zhang, Yangming Guo, Ting Ju, Qi Zhu και Guochang Zhou. *An Improved LZO Compression Algorithm for FPGA Configuration Bitstream Files*. *Computers, Materials & Continua*, 82(2):3091–3109, 2025. DOI: https://doi.org/10.32604/cmc.2025.058688.

[18] Gregory Szorc. *Better Compression with Zstandard*, 2017. https://gregoryszorc.com/blog/2017/03/07/better-compression-with-zstandard/, Accessed: 13 June 2025.

[19] Danilo Piparo και others. *Compression Update: ZSTD & ZLIB*. *Proc. 19th HEPiX Workshop*, 2018. Presentation, CERN Indico, URL: https://indico.cern.ch/event/695984/contributions/2872933/.

[20] Jianyu Chen, Maurice Daverveldt και Zaid Al-Ars. *FPGA Acceleration of Zstd Compression Algorithm*. σελίδες 188–191, 2021. DOI: https://doi.org/10.1109/IPDPSW52791.2021.00035.

[21] Junsun Choi. *A Hardware Accelerator Generator for Zstandard Decompression*. Master's thesis, Univ. of California, Berkeley, EECS Department, 2024. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-236.pdf.

[22] CAST Inc. *Evaluating Lossless Data Compression Algorithms and Cores*. White Paper, CAST, Inc. (2023), Available: https://www.cast-inc.com/blog/white-paper-evaluating-lossless-data-compression-algorithms-and-cores.

[23] Qiang Dou Junzhe Huang και Li Shen. *Extending the RISC-V Instruction Set for High Performance Data Compression Hardware Acceleration*. σελίδες 131–132, 2024. DOI: https://doi.org/10.1109/ASAP61560.2024.00035.

[24] Ian Kuon, Russell Tessier και Jonathan Rose. *FPGA Architecture: Survey and Challenges*. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2008. DOI: https://doi.org/10.1561/1000000005.

[25] John L. Hennessy και David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 6η έκδοση, 2019. ISBN: 978-0128119051.

[26] Roberto Millon, Emmanuel Frati και Enzo Rucci. *A Comparative Study between HLS and HDL on SoC for Image Processing Applications*. https://arxiv.org/abs/2012.08320, 2020.

[27] Muhammad Sabih, Abrarul Karim, Jakob Wittmann, Frank Hannig και Jurgen Teich. *Hardware/Software Co-Design of RISC-V Extensions for Accelerating Sparse DNNs on FPGAs*. https://arxiv.org/pdf/2504.19659, 2024.

[28] Qiankun Liu και Sam Amiri. *Optimised Extension of an Ultra-Low-Power RISC-V Processor to Support Lightweight Neural Network Models*. *Chips*, 4(2), 2025. DOI: https://doi.org/10.3390/chips4020013.

[29] Fran Casino, Kim Kwang Raymond Choo και Constantinos Patsakis. *HEDGE: Efficient Traffic Classification of Encrypted and Compressed Packets*. σελίδες 1–6, 2019. URL: https://arxiv.org/abs/1905.11873.

[30] Peter Dorfinger, Georg Panholzer και Wolfgang John. *Entropy Estimation for Real-Time Encrypted Traffic Identification*. *Proc. Traffic Monitoring and Analysis Workshop (TMA 2011)*, τόμος 6613 στο *Lecture Notes in Computer Science*, σελίδες 164–171. Springer, 2011. DOI: https://doi.org/10.1007/978-3-642-20305-3_14.

[31] Liam Paninski. *Estimation of Entropy and Mutual Information*. *Neural Computation*, 15(6):1191–1253, 2003. DOI: https://doi.org/10.1162/089976603321780272.

[32] Jeehong Kim και Jundong Cho. *Hardware-Accelerated Fast Lossless Compression Based on LZ4 Algorithm*. *Proceedings of the 2019 3rd International Conference on Digital Signal Processing*, σελίδες 65–68. Association for Computing Machinery, 2019. DOI: https://doi.org/10.1145/3316551.3316564.

[33] M. Larabel. *Linux 3.11 Kernel Will Support LZ4 Compression*. https://www.phoronix.com/news/MTQwODY, 2013.

[34] Cisco Systems. *Ultra Gateway Platform System Administration Guide, Release 6.12*, 2020. URL: https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-20_6-14/UGP-Sys-Admin/6-14-ugp-sys-admin.pdf Section “Configuring LZ4 Compression Algorithm”.

[35] *LZ4 (compression algorithm) - Wikipedia*. https://en.wikipedia.org/wiki/LZ4_(compression_algorithm), 2025.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλπ	βλέπε
κ.α.	και άλλα
κ.λπ.	και λοιπά
κ.ο.κ.	και ούτω καθεξής
π.χ.	Παραδείγματος χάριν
Ε.Μ.Π.	Εθνικό Μετσόβιο Πολυτεχνείο
ΠΑ.ΔΑ.	Πανεπιστήμιο Δυτικής Αττικής
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
BRAM	Block Random Access Memory
CI	Custom Inctruction
CPU	Central Processing Unit
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
HLS	High-Level Synthesis
IoT	Interne of Things
ISA	Instruction Set Architecture
LUT	Lookup Table
QoS	Quality of Service
RISC-V	Reduced Instruction Set Computer V
RTL	Register Transfer Level
VHDL	VHSIC Hardware Description Language