Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Design and Evaluation of Bespoke Microprocessor Architectures for Flexible Devices

Σχεδιασμός και Αξιολόγηση Προσαρμοσμένων Αρχιτεκτονικών Μικροεπεξεργαστή για Ευέλικτες Συσκευές

Diploma Thesis

by

Panagiotis Chaidos

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Design and Evaluation of Bespoke Microprocessor Architectures for Flexible Devices

Σχεδιασμός και Αξιολόγηση Προσαρμοσμένων Αρχιτεκτονικών Μικροεπεξεργαστή για Ευέλικτες Συσκευές

Diploma Thesis

by

Panagiotis Chaidos

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Απριλίου, 2024.

.....	.....	.....
Δημήτριος Σούντρης	Σωτήριος Ξύδης	Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π

Αθήνα, Απρίλιος 2024

.....

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

Περίληψη

Με την άνοδο των εφαρμογών ML που απαιτούν πολύ ενέργεια, οι τυπωμένοι υπολογιστές ικανοποιούν τις απαιτήσεις κόστους και μη τοξικότητας, όπου οι τυπικοί υπολογιστές που βασίζονται σε πυρίτιο φαίνεται να υστερούν. Το χαμηλό κόστος κατασκευής και της δυνατότητας απόρριψης που παρέχουν οι τυπωμένες τεχνολογίες ταιριάζουν με τις ανάγκες εφαρμογών ML. Επιπλέον, η χρήση τυπωμένων επεξεργαστών επιτρέπει τον προγραμματισμό και την ευελιξία στις εφαρμογές που μπορούν να εκτελεστούν, σε σύγκριση με υλικό που είναι εξειδικευμένο ανα εφαρμογή και είναι πιο ενεργοβόρο. Οι κύριοι περιορισμοί για τις τυπωμένες τεχνολογίες είναι τα μεγάλα μεγέθη χαρακτηριστικών μαζί με την περιορισμένη υποστήριξη ισχύος από τυπωμένες μπαταρίες. Ως εκ τούτου, υπάρχει ανάγκη για βελτιώσεις σε επιφάνεια και ισχύ, προκειμένου να υλοποιηθούν πολύπλοκοι επεξεργαστές. Μια προσέγγιση σε αυτό το ζήτημα είναι οι τεχνικές μείωσης υλικού, οι οποίες έχουν αποδειχθεί γόνιμες και απαραίτητες όταν εξετάζονται τυπωμένοι επεξεργαστές που πρέπει να πληρούν περιορισμούς. Στην παρούσα διατριβή διερευνούμε τις δυνατότητες για βελτίωση της επιφάνειας και της ισχύος των τυπωμένων επεξεργαστών με τη χρήση της βιβλιοθήκης τυποποιημένων κελιών EGFET για την τεχνολογία εκτύπωσης χαμηλής τάσης, όσον αφορά εφαρμογές μηχανικής μάθησης και τυπωμένες εφαρμογές. Συνθέτουμε και αναλύουμε μετρήσεις υλικού για ένα σύνολο επεξεργαστών, εστιάζοντας κυρίως σε αρχιτεκτονικές χαμηλού αριθμού πυλών και χαμηλής ισχύος. Συνθέτουμε τις μετρήσεις αναφοράς και προσομοιώνουμε τους επεξεργαστές με προσομοιώσεις RTL και netlist για την εξαγωγή των ιχνών εκτέλεσης χρησιμοποιώντας τη σουίτα εργαλείων της Synopsys και τον προσομοιωτή Modelsim. Αναλύουμε τα ίχνη εκτέλεσης των εφαρμογών για να εντοπίσουμε και να αφαιρέσουμε αχρησιμοποίητα στοιχεία και συγκεκριμένες λογικές λειτουργίες του ISA των επεξεργαστών, με στόχο την κατασκευή εξειδικευμένων επεξεργαστών με βελτιωμένες προδιαγραφές. Στη συνέχεια, ενσωματώνουμε μονάδες MAC που βελτιώνουν αποτελεσματικά την απόδοση και την κατανάλωση ειδικά για φόρτους εργασίας ML, όπως MLP και SVM, με υψηλή χρήση MAC. Τέλος, διερευνούμε τα οφέλη με την εισαγωγή της κλιμάκωσης ακρίβειας στις νέες μονάδες MAC, μετρώντας το συμβιβασμό μεταξύ της ταχύτητας και της απώλειας ακρίβειας. Οι προτεινόμενες μονάδες μας και οι προσαρμοσμένες τροποποιήσεις επιτυγχάνουν από **22.2%**, **23.6%** και **33.79%** βελτιώσεις στην επιφάνεια, την ισχύ και την επιτάχυνση όταν δεν επιβάλλονται απώλειες ακρίβειας, μέχρι **29. 3%**, **28.7%** και **41.73%** κέρδη σε επιφάνεια, ισχύ και επιτάχυνση, με μόλις **0.5%** μείωση της μέσης ακρίβειας που εκτιμάται σε 3 σύνολα δεδομένων για τον κύριο Zero-Riscy επεξεργαστή.

Λέξεις-κλειδιά — Τυπωμένα Ηλεκτρονικά, Τυπωμένη Υπολογιστική, Μηχανική Μάθηση, EDA, Κλιμάκωση Ακρίβειας, Προσαρμοσμένοι Επεξεργαστές

Abstract

In recent years with the rise of power-hungry ML applications, Printed Computing serves to meet the requirements for cost, conformity, and non-toxicity where standard silicon-based computing seems to be lacking. The aspects of low manufacturing costs and disposability that printed technologies provide, fit well with a variety of ML applications' needs. On top of that, using printed microprocessors allows for programability and thus flexibility to the workloads that can be run, compared to application specific hardware that tends to be more power hungry. Limitations for printed technologies can be extremely large feature sizes along with limited power support from small printed batteries. Hence there is a rising need for improvements in the domains of area and power, in order to fit complex processors. One approach to this issue is hardware reduction techniques, that have shown to be fruitful and necessary when considering printed processors that need to meet constraints. In this thesis we explore the possibilities for area and power gains of printed microprocessors using the EGFET standard cell library for low voltage printing technology, regarding machine learning workloads and printed workloads. We synthesize and analyse hardware measurements for a set of examined processors, focusing mostly on low gate-count and low power architectures. We compile the benchmarks and simulate the processors with RTL and netlist simulations to extract the execution traces using the Synopsys EDA suite and Modelsim simulator. We analyse the execution traces of the workloads to locate and remove unused whole components and more specific logic functionalities of the ISA of our processors, with the aim of building bespoke processors with improved hardware specs. We then incorporate MAC units that efficiently improve the performance and consumption specifically for ML workloads like MLPs and SVMs with high MAC usage. Finally, we explore the benefits of introducing precision-scaling in our new MAC units, measuring the speedup and accuracy loss tradeoff. Our proposed units and bespoke modifications achieve from **22.2%**, **23.6%** and **33.79%** improvements in area, power and speedup when imposing no accuracy loss, up to **29.3%**, **28.7%** and **41.73%** gains in area, power and speedup, with just a **0.5%** decrease in average accuracy estimated over 3 datasets for the main Zero-Riscy Core.

Keywords — Printed Electronics, Printed Computing, Machine Learning, EDA, Precision Scaling, Bespoke Processors

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντά μου, κ. Δημήτριο Σούντρη για την ευκαιρία συνεργασίας και εκπόνησης αυτής της διπλωματικής εργασίας. Ευχαριστώ τον υποψήφιο διδάκτορα Γιώργο Αρμενιάκο για τη καθοδήγηση σε όλη τη διαρκεια της διπλωματικής. Ευχαριστώ τους φίλους μου με τους οποίους περάσαμε μαζί πολλά ευχάριστα και απαιτητικά σημεία της διαδικασίας των σπουδών. Τέλος, ευχαριστώ τους γονείς και τον αδερφό μου για την πολυετή υποστήριξη σε όλη τη διάρκεια της ζωής και εκπαίδευτικής μου πορείας.

Παναγιώτης Χάιδος, Απρίλιος 2024

Contents

List of Figures

Acronyms

ISA	**Instruction Set Architecture**
ASIC	**Application Specific Integrated Circuit**
ML	**Machine Learning**
MLP	**Multi-Layer Perceptron**
SVM	**Support Vector Machine**
DT	**Decision Tree**
HW	**Hardware**
SW	**Software**
MAC	**Multiply Accumulate**
PS	**Precision Scaling**
EDA	**Electronics Design Automation**
ALU	**Arithmetic Logical Unit**

Εκτεταμένη Ελληνική Περίληψη

Εισαγωγή

Οι φορετές, χαμηλού κόστους υπολογιστικές πλατφόρμες έχουν εξεταστεί πρόσφατα για πολλές περιπτώσεις χρήσης, ιδίως με την άνοδο του υπολογισμού στο άκρο και των εφαρμογών που λειτουργούν με μπαταρία. Εν μέσω των αυξανόμενων ανησυχιών για την κλιματική αλλαγή και τη βιωσιμότητα, η ιεράρχηση της μη τοξικής, χαμηλής ισχύος υπολογιστικής ευθυγραμμίζεται με ευρύτερες πρωτοβουλίες που αποσκοπούν στη μείωση των εκπομπών διοξειδίου του άνθρακα και στην προώθηση τεχνολογιών φιλικών προς το περιβάλλον. Τα τυπωμένα ηλεκτρονικά είναι συσκευές που εκτυπώνονται σε εύκαμπτα υλικά αντί των παραδοσιακών κυκλωμάτων πυριτίου, αποτελώντας μια πολύ χαμηλού κόστους και μη τοξική λύση στα προαναφερθέντα προβλήματα.

Επιπλέον, δεδομένου ότι οι φόρτοι εργασίας μηχανικής μάθησης φαίνεται να αποτελούν τον πυρήνα των περισσότερων σύγχρονων εφαρμογών, η προσαρμογή της τεχνολογίας εκτύπωσης προς την κατεύθυνση της χαμηλής κατανάλωσης ενέργειας και της αποδοτικής εκτέλεσης τέτοιων φόρτων εργασίας φαίνεται ελκυστική. Η συνήθης επιλογή υπολογιστικών παραδειγμάτων για την συγκεκριμένη περίπτωση μπορεί να είναι τα ASIC και οι πλήρεις επεξεργαστές, καθένα με τα δυνατά και τα αδύνατα σημεία του. Στην περίπτωσή μας, οι τυπωμένοι επεξεργαστές μπορούν να εξυπηρετήσουν πολλαπλούς σκοπούς, επιτρέπουν ευελιξία στις υποστηριζόμενες εφαρμογές και συνεπώς θεωρούνται καλύτεροι για εμπορική χρήση, σε σύγκριση με τα ASIC που εξειδικεύονται για ένα μόνο ζεύγος εφαρμογής-συνόλου δεδομένων, δεν μπορούν να επαναπρογραμματιστούν και έχουν υψηλότερο συνολικό κόστος παραγωγής και σχεδιασμού. Προκειμένου να αξιοποιηθούν όλα τα οφέλη των επεξεργαστών και να διατηρηθούν χαμηλοί πόροι υλικού, είναι υποχρεωτική η διερεύνηση των δυνατοτήτων μείωσης του υλικού, προσαρμόζοντας κάθε επεξεργαστή αυστηρά στις εφαρμογές που πρόκειται να εκτελεστούν.

Θεωρητικό Υπόβαθρο

Τυπωμένα Ηλεκτρονικά

Η τεχνολογία των τυπωμένων ηλεκτρονικών είναι ένας αναπτυσσόμενος τομέας της ηλεκτρονικής μηχανικής, που χρησιμοποιεί διαδικασίες προσθετικής κατασκευής χαμηλού κόστους, όπως η εκτύπωση με μελάνι, η οθόνη ή η φλεξογραφική εκτύπωση σε διάφορα υποστρώματα για τη δημιουργία ηλεκτρονικών συσκευών, για τη στόχευση διαφόρων εφαρμογών 0.0.1.

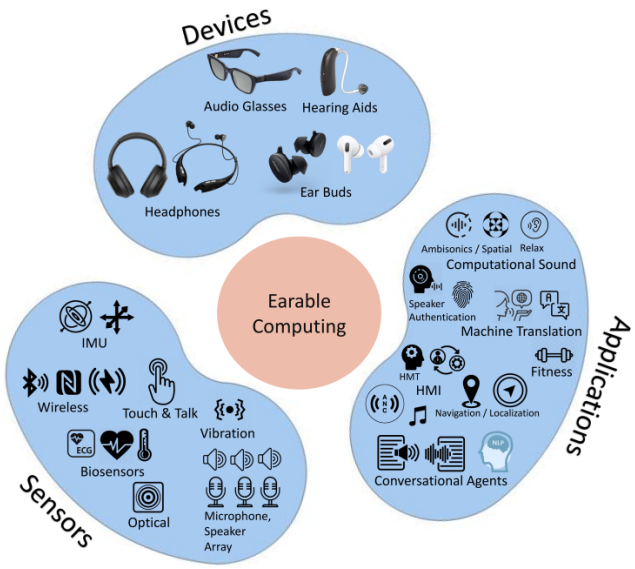

Figure 0.0.1: Πεδίο εφαρμογών και ικανοτήτων Τυπωμένων Ηλεκτρονικών

Σε αντίθεση με τις παραδοσιακές μεθόδους, τα τυπωμένα ηλεκτρονικά προσφέρουν ταχύτερη και φθηνότερη παραγωγή, αλλά μπορεί να οδηγήσουν σε πιο αργά κυκλώματα και μεγαλύτερα εξαρτήματα. Λειτουργικά υλικά όπως αγώγιμα, διηλεκτρικά και ημιαγωγικά μελάνια επιτρέπουν την ταχεία δημιουργία πρωτοτύπων και την προσαρμογή, καθιστώντας τα οικονομικά αποδοτικά και επεκτάσιμα σε σύγκριση με τα κυκλώματα που βασίζονται στο πυρίτιο. Παρά τα πλεονεκτήματα, τα μειονεκτήματα περιλαμβάνουν χαμηλότερες επιδόσεις, περιορισμένες επιλογές υλικών και προβλήματα σταθερότητας λόγω περιβαλλοντικών παραγόντων. Η βελτιστοποίηση των τυπωμένων ηλεκτρονικών για σταθερή απόδοση απαιτεί προσεκτική εξέταση των διαδικασιών εκτύπωσης 0.0.2, των συνθέσεων μελανιού και των ιδιοτήτων του υποστρώματος.

Τεχνολογία EGFET

Η τεχνολογία των τρανζίστορ πεδίου με ηλεκτρολυτική πύλη (EGFET) αποτελεί σημαντική πρόοδο στα εύκαμπτα ηλεκτρονικά, προσφέροντας χαμηλές τάσεις λειτουργίας και βελτιωμένη βιοσυμβατότητα. Σε αντίθεση με τα συμβατικά FETs, τα EGFETs χρησιμοποιούν ένα διάλυμα ηλεκτρολύτη ως μονωτή πύλης, επιτρέποντας τον δυναμικό έλεγχο της διάταξης μέσω της μετανάστευσης ιόντων. Είναι εξαιρετικά αγώγιμα και κατάλληλα για βιοηλεκτρονικές εφαρμογές, επιτρέποντας την άμεση διασύνδεση με βιολογικά συστήματα. Επιπλέον, το [8] εισάγει μια σχεδίαση μνήμης και μια βιβλιοθήκη για τυπωμένους μικροεπεξεργαστές που υλοποιούν αρχιτεκτονικές ROM και RAM οπως περιγράφονται στο σχήμα 0.0.3.

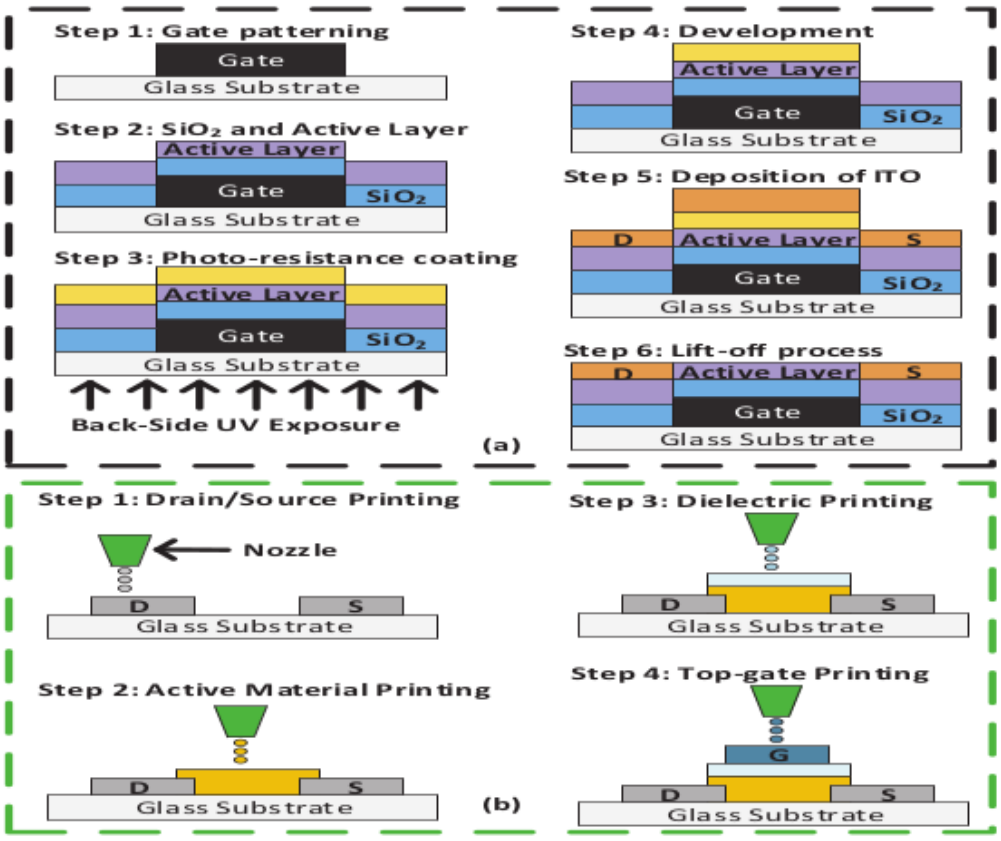

Figure 0.0.2: Διαδικασία Αφαιρετικής(a) και Προσθετικής(b) εκτύπωσης

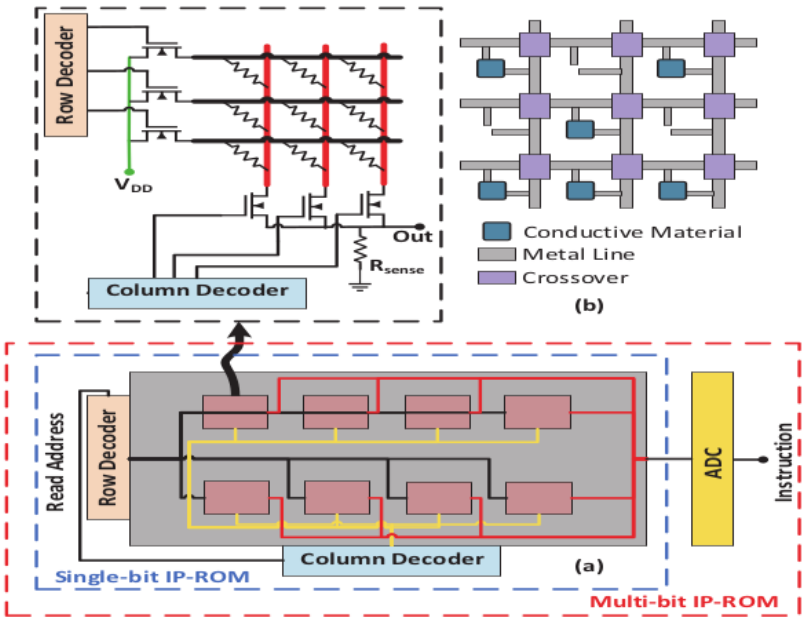

Figure 0.0.3: Μνήμες τεχνολογίας EGFET

Οι μνήμες ROM είναι ταχύτερες, μικρότερες και ενεργειακά αποδοτικότερες από τη RAM, καθιστώντας προτιμότερες τις σχεδιάσεις με λιγότερη RAM.Οι ROM πολλαπλών bit αυξάνουν την πυκνότητα μνήμης αποθηκεύοντας πολλαπλές τιμές με ένα μόνο αγώγιμο κελί, χρησιμοποιώντας διαφορετικές γεωμετρίες αγώγιμου υλικού για την κωδικοποίηση περισσότερων από δυαδικές τιμές. Ωστόσο, αυτό απαιτεί έναν πρόσθετο ADC για την ανάγνωση των αναλογικών επιπέδων τάσης.

Εργαλεία Σχεδίασης EDA

Τα εργαλεία αυτοματισμού ηλεκτρονικής σχεδίασης (EDA) βελτιώνουν τη σχεδίαση κυκλωμάτων προσφέροντας διαισθητικές διεπαφές και ισχυρούς αλγορίθμους για τη δημιουργία, την προσομοίωση και τη βελτιστοποίηση. Επιτρέπουν στους σχεδιαστές να διερευνήσουν εναλλακτικές λύσεις, να προβλέψουν με ακρίβεια την απόδοση του κυκλώματος και να ποσοτικοποιήσουν τις προδιαγραφές του υλικού. Τα εργαλεία EDA διευκολύνουν τον αποτελεσματικό σχεδιασμό, την ανάλυση και την επικύρωση ηλεκτρονικών συστημάτων, ενσωματώνοντας δυνατότητες προσομοίωσης και μέτρησης.

Οι τυποποιημένες βιβλιοθήκες κελιών είναι θεμελιώδεις στην EDA, παρέχοντας προχαρακτηρισμένα λογικά κελιά για σύνθεση 0.0.4. Μεταφράζουν περιγραφές υλικού υψηλού επιπέδου σε αναπαραστάσεις σε επίπεδο πύλης, εξασφαλίζοντας συμβατότητα και ικανοποιώντας περιορισμούς σχεδιασμού.

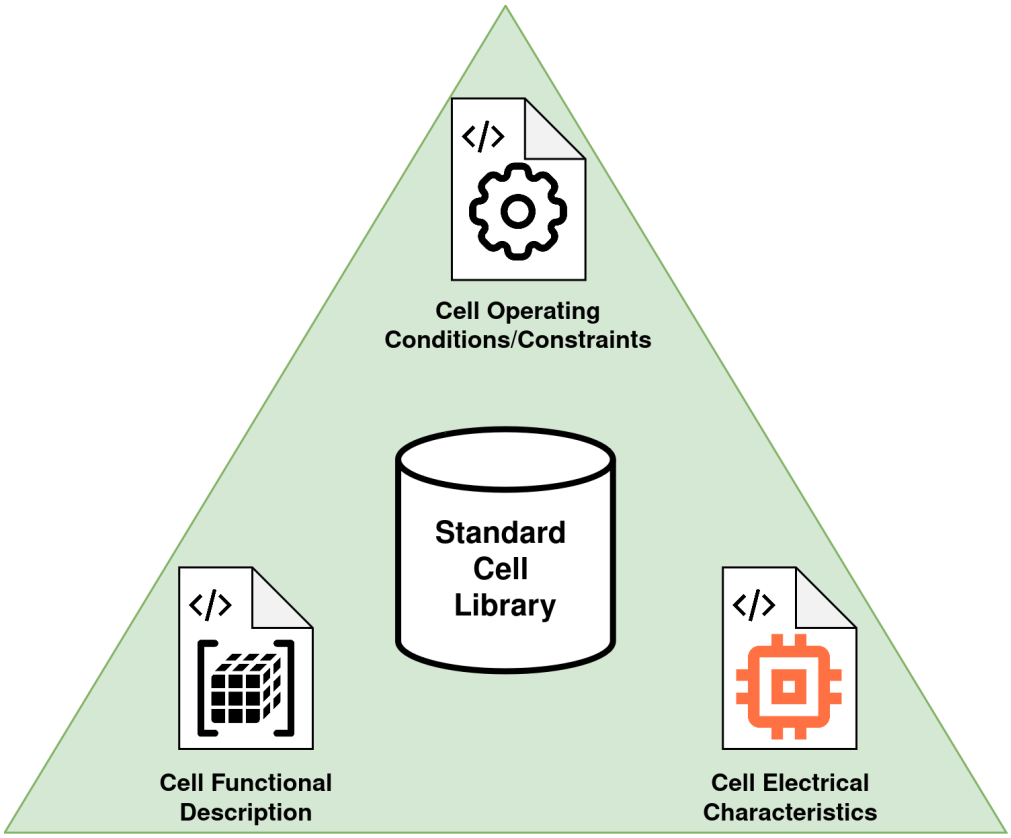

Figure 0.0.4: Βασικά στοιχεία Τυποποιημένων Βιβλιοθηκών Κελιών

Το Synopsys Design Compiler είναι ένα ευρέως χρησιμοποιούμενο εργαλείο σύνθεσης που

βελτιστοποιεί τις netlists για συγκεκριμένες τεχνολογίες-στόχους, προσφέροντας χαρακτηριστικά διερεύνησης σχεδίασης. Το Synopsys Power Compiler, ενσωματωμένο στο Design Compiler, ανταποκρίνεται στη ζήτηση για ηλεκτρονικά συστήματα χαμηλής κατανάλωσης ενέργειας. Αναλύει, βελτιστοποιεί και διαχειρίζεται την κατανάλωση ισχύος καθ' όλη τη διάρκεια της διαδικασίας σχεδίασης, βοηθώντας στη μείωση της κατανάλωσης, διατηρώντας παράλληλα την ακεραιότητα της σχεδίασης. Τα εργαλεία αυτά είναι ζωτικής σημασίας για την ανάπτυξη ενεργειακά αποδοτικών ηλεκτρονικών συστημάτων, ειδικά σε τυπωμένες εφαρμογές.

Εφαρμογές Μηχανικής Μάθησης

Η μηχανική μάθηση είναι ένας κλάδος της τεχνητής νοημοσύνης που επικεντρώνεται στην ανάπτυξη αλγορίθμων και στατιστικών μοντέλων που επιτρέπουν στους υπολογιστές να μαθαίνουν και να κάνουν προβλέψεις ή αποφάσεις με βάση δεδομένα, χωρίς να είναι ρητά προγραμματισμένοι για κάθε εργασία. Ο γενικός στόχος της μηχανικής μάθησης είναι να επιτρέψει στα συστήματα να βελτιώνουν αυτόματα την απόδοσή τους σε μια δεδομένη εργασία μέσω της εμπειρίας ή της έκθεσης σε δεδομένα. Η εκπαίδευση και η εξαγωγή συμπερασμάτων είναι δύο θεμελιώδεις διαδικασίες που συμβαίνουν κατά τη διάρκεια του κύκλου ζωής ενός μοντέλου.

Η εκπαίδευση περιλαμβάνει τη μάθηση του μοντέλου από τα ζεύγη εισόδου-εξόδου με την προσαρμογή εσωτερικών παραμέτρων όπως τα βάρη και τα bias για την ελαχιστοποίηση μιας προκαθορισμένης συνάρτησης απωλειών. Αυτή η επαναληπτική διαδικασία, που συχνά χρησιμοποιεί back propagation 0.0.1, ρυθμίζει τις παραμέτρους του μοντέλου για να βελτιστοποιήσει την απόδοση στα δεδομένα εκπαίδευσης, αποφεύγοντας παράλληλα την υπερπροσαρμογή. Η εξαγωγή συμπερασμάτων πραγματοποιείται μετά την εκπαίδευση, όπου το μοντέλο χρησιμοποιεί τις παραμέτρους που έχει μάθει για να κάνει προβλέψεις σε νέα δεδομένα χωρίς περαιτέρω προσαρμογές. Η αποτελεσματικότητα του μοντέλου αξιολογείται κατά τη διάρκεια της συμπερασματολογίας χρησιμοποιώντας μετρικές όπως η ακρίβεια ή το μέσο τετραγωνικό σφάλμα, δοκιμάζοντας την ικανότητα γενίκευσής του. Η εκπαίδευση και η συμπερασματολογία είναι συμπληρωματικές διαδικασίες που επιτρέπουν στα μοντέλα μηχανικής μάθησης να μαθαίνουν και να κάνουν προβλέψεις αποτελεσματικά.

$$\frac{\partial E}{\partial w_{ij}} = \delta_j y_i \quad \text{where} \quad \delta_j = \frac{\partial E}{\partial z_j} \cdot \sigma'(net_j) \tag{0.0.1}$$

Η ταξινόμηση και η παλινδρόμηση είναι διαδεδομένες εργασίες μηχανικής μάθησης. Η ταξινόμηση προβλέπει την κατηγορία της εισόδου βάσει χαρακτηριστικών, με μεταβλητές εξόδου που ανήκουν σε προκαθορισμένες κατηγορίες (π.χ. ανίχνευση ανεπιθύμητων μηνυμάτων, αναγνώριση εικόνων). Στην παλινδρόμηση, η μεταβλητή εξόδου είναι αριθμητική, επιτρέποντας διάφορες τιμές (π.χ. τιμές κατοικιών, πρόβλεψη θερμοκρασίας). Και οι δύο εργασίες περιλαμβάνουν την εκπαίδευση μοντέλων σε επισημασμένα δεδομένα για την εκμάθηση προτύπων. Οι μετρικές αξιολόγησης περιλαμβάνουν την ακρίβεια, την ακρίβεια, την ανάκληση, το F1-score (για ταξινόμηση), το μέσο τετραγωνικό σφάλμα και το μέσο απόλυτο σφάλμα (για παλινδρόμηση), μετρώντας τη γενίκευση του μοντέλου σε νέα δεδομένα. Και οι δύο εργασίες εξετάστηκαν στην πειραματική ενότητα.

Τα Multi Layer Perceptrons (MLPs) 0.0.5ίναι τεχνητά νευρωνικά δίκτυα που χρησιμοποιούν-

ται ευρέως στη μηχανική μάθηση και την αναγνώριση προτύπων. Διαθέτουν διασυνδεδεμένα στρώματα κόμβων, συμπεριλαμβανομένων των στρωμάτων εισόδου, των κρυφών στρωμάτων και των στρωμάτων εξόδου. Χρησιμοποιήσαμε μοντέλα με ένα ή δύο κρυφά στρώματα σε διάφορες εφαρμογές. Τα MLP χρησιμοποιούν την προς τα εμπρός διάδοση για την επεξεργασία των δεδομένων εισόδου, μετατρέποντάς τα σε προβλέψεις εξόδου μέσω των κρυφών στρωμάτων. Ο οπίσθιος πολλαπλασιασμός διευκολύνει την εκμάθηση σύνθετων μη γραμμικών σχέσεων κατά τη διάρκεια της εκπαίδευσης, προσαρμόζοντας τις εσωτερικές παραμέτρους. Υπολογιστικά, τα δεδομένα εισόδου περνούν από τα στρώματα, υπολογίζοντας σταθμισμένα αθροίσματα και εφαρμόζοντας συναρτήσεις ενεργοποίησης για μη γραμμικότητα. Οι προβλέψεις εξόδου βασίζονται σε μεγάλο βαθμό σε σταθμισμένα αθροίσματα, που περιλαμβάνουν πολλαπλασιασμούς και προσθέσεις (MAC), γεγονός που απαιτεί επιταχυντές με πολλαπλές μονάδες MAC για αποδοτικό παράλληλο υπολογισμό. Η εκπαίδευση περιλαμβάνει επαναληπτικές προσαρμογές παραμέτρων για την ελαχιστοποίηση του σφάλματος μεταξύ προβλεπόμενων και πραγματικών εξόδων μέσω αλγορίθμων βελτιστοποίησης.

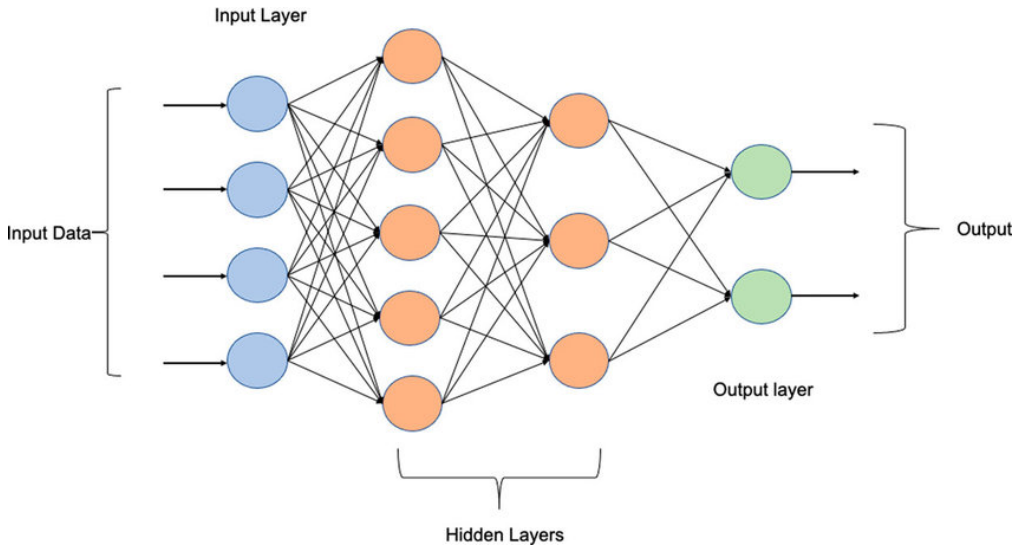

Figure 0.0.5: Απλή Αρχιτεκτονική MLP [1]

Οι μηχανές διανυσμάτων υποστήριξης (SVM) 0.0.6 είναι μοντέλα μάθησης με επίβλεψη για εργασίες ταξινόμησης και παλινδρόμησης. Στοχεύουν στην εύρεση του βέλτιστου υπερεπιπέδου που διαχωρίζει τις κλάσεις στο χώρο των δεδομένων εισόδου μεγιστοποιώντας το περιθώριο. Αυτό το υπερεπίπεδο προσδιορίζεται με την εύρεση διανυσμάτων υποστήριξης, τα οποία είναι σημεία δεδομένων που βρίσκονται πλησιέστερα στο όριο απόφασης. Οι SVM μετασχηματίζουν τα δεδομένα εισόδου σε έναν χώρο χαρακτηριστικών υψηλότερης διάστασης χρησιμοποιώντας συναρτήσεις πυρήνα, καθιστώντας τις κλάσεις πιο διαχωρίσιμες χωρίς ρητή αντιστοίχιση των σημείων δεδομένων. Η διαδικασία βελτιστοποίησης περιλαμβάνει την επίλυση ενός περιορισμένου προβλήματος για τον προσδιορισμό του βέλτιστου υπερεπιπέδου. Οι SVM χειρίζονται την ταξινόμηση αξιολογώντας τις θέσεις των νέων σημείων δεδομένων σε σχέση με το υπερεπίπεδο. Υπολογιστικά, τα SVM περιλαμβάνουν πολλαπλασιασμό και πρόσθεση εισόδων, bias και βαρών, παρόμοια με τα MLP, αλλά χωρίς συναρτήσεις ενεργοποίησης. Παρά τις διαφορές, οι SVM προσφέρουν πλεονεκτήματα όπως ο χειρισμός δεδομένων υψηλών διαστάσεων, η αντίσταση στην υπερπροσαρμογή και η αποτελεσματική αντιμετώπιση μη γραμμικών ορίων

απόφασης μέσω κατάλληλων πυρήνων.

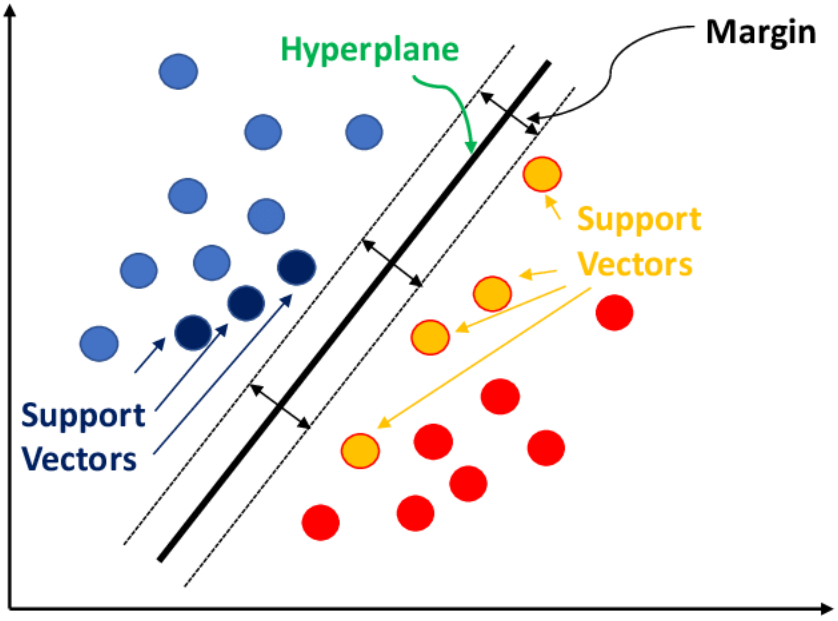

Figure 0.0.6: Οπτικοποίηση ενός SVM Classifier [23]

Κλιμάκωση Ακρίβειας

Η κλιμάκωση ακρίβειας, μια τεχνική προσέγγισης, προσαρμόζει την ακρίβεια των αριθμητικών αναπαραστάσεων ώστε να εξισορροπήσει την υπολογιστική αποδοτικότητα και την ακρίβεια. Χρησιμοποιείται ευρέως σε τομείς όπως η ψηφιακή επεξεργασία σήματος και η μηχανική μάθηση, βελτιστοποιεί την απόδοση και τη χρήση μνήμης μειώνοντας τα σημαντικά bits στους υπολογισμούς. Ωστόσο, εισάγει σφάλματα προσέγγισης λόγω αποκοπής ή στρογγυλοποίησης, επηρεάζοντας δυνητικά τη συνολική ακρίβεια. Η αξιολόγηση του συνολικού σφάλματος και των κερδών σε επιφάνεια, χρονισμό και ισχύ συνοδεύει τις αποφάσεις κλιμάκωσης ακρίβειας, καθώς η πολυπλοκότητα του υλικού και η κατανάλωση ενέργειας μειώνονται. Η κατανόηση των ειδικών απαιτήσεων και περιορισμών της εργασίας καθοδηγεί την απόφαση για την εφαρμογή της κλιμάκωσης ακριβείας, λαμβάνοντας υπόψη την αποδεκτή ανοχή σφάλματος. Στη μηχανική μάθηση, η κλιμάκωση ακριβείας μετατρέπει την υπερβολική ακρίβεια σε υπολογιστική αποδοτικότητα, ωφελώντας τα συστήματα πραγματικού χρόνου και τις εργασίες συμπίεσης που στοχεύουν σε χαμηλό χρόνο υπολογισμού και αποτύπωμα μνήμης. Τα κρίσιμα για την ασφάλεια και την ακρίβεια συστήματα, όπως οι ιατρικοί υπολογιστές και η κρυπτογραφία, απαιτούν υψηλή ακρίβεια και ακρίβεια παρά το αυξημένο υπολογιστικό κόστος.

Προσαρμοσμένη Ανάλυση και Μείωση Υλικού

Επεξεργαστές υπό Εξέταση

Η ροή κατασκευής και η σουίτα εφαρμογών δοκιμάστηκαν με διάφορους επεξεργαστές προσαρμοσμένους για τεχνολογίες τυπωμένων ηλεκτρονικών. Δύο επεξεργαστές αξιολογήθηκαν με βάση τα χαρακτηριστικά του υλικού και χρησιμοποιήθηκαν για προσομοιώσεις εφαρμογών.

- Ο Zero-riscy είναι μια αρχιτεκτονική RISC-V με αγωγό 2 σταδίων 32 bit, η οποία μπορεί να διαμορφωθεί με μειωμένο αρχείο καταχωρητών, χωρίς συμπιεσμένες εντολές και με ή χωρίς υποστήριξη εντολών mult-div. Παρά τις ελάχιστες διαμορφώσεις, ο Zero-riscy είναι σχετικά μεγάλος λόγω των πολλαπλών σταδίων pipeline και του πολύπλοκου ISA.

- Το OpenMSP430 είναι ένας διαμορφώσιμος μικροελεγκτής 16-bit ενός σταδίου με επιλογές για την προσαρμογή περιφερειακών συστημάτων και λειτουργικών μπλοκ. Χρησιμοποιείται ελάχιστη διαμόρφωση, εκτός από τη μονάδα πολλαπλασιαστή, η οποία μπορεί να μειώσει σημαντικά τους χρόνους εκτέλεσης σε εφαρμογές με μεγάλο βάρος σε πράξεις πολλαπλασιασμού.

- Ο ZPU_Small είναι ένας επεξεργαστής ISA 32-bit με βάση τη στοίβα, σχεδιασμένος για εφαρμογές χαμηλών πόρων, γραμμένος σε VHDL. Ελαχιστοποιεί τα διαδοχικά στοιχεία και προσφέρει μια συμπαγή λύση.

- Ο TP-ISA είναι μια προσαρμόσιμη αρχιτεκτονική σχεδιασμένη για την τεχνολογία EGFET, με παραμετροποιήσιμα αρχιτεκτονικά χαρακτηριστικά. Πρόκειται για μια συμπαγή αρχιτεκτονική RISC που έχει σχεδιαστεί για να ελαχιστοποιεί την πολυπλοκότητα και τα διαδοχικά στοιχεία, βελτιώνοντας την επιφάνεια και την κατανάλωση ενέργειας όταν στοχεύει στη βιβλιοθήκη EGFET. Στο σχήμα 0.0.7φαίνεται ολόκληρο το Σύνολο Εντολών του TP-ISA.

Instruction Format	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M-Type:	opcode				W	C	A	B	R	address1							S	address2							
ADD	OP-ADD				1	0	0	0	R	address1							S	address2							
ADC	OP-ADD				1	1	0	0	R	address1							S	address2							
SUB	OP-ADD				1	0	1	0	R	address1							S	address2							
CMP	OP-ADD				0	0	1	0	R	address1							S	address2							
SBB	OP-ADD				1	1	1	0	R	address1							S	address2							
AND	OP-AND				1	0	0	0	R	address1							S	address2							
TEST	OP-AND				0	0	0	0	R	address1							S	address2							
OR	OP-OR				1	0	0	0	R	address1							S	address2							
XOR	OP-XOR				1	0	0	0	R	address1							S	address2							
NOT	OP-NOT				1	0	0	0	R	address1							S	address2							
RL	OP-RL				1	0	0	0	R	address1							S	address2							
RLC	OP-RL				1	1	0	0	R	address1							S	address2							
RR	OP-RR				1	0	0	0	R	address1							S	address2							
RRC	OP-RR				1	1	0	0	R	address1							S	address2							
RRA	OP-RR				1	0	1	0	R	address1							S	address2							
S-Type:	opcode				W	3'b0			R	address1							immediate							
STORE	OP-STORE				1	0	0	0	R	address1							immediate							
SET-BAR	OP-BAR				0	0	0	0	x	ptr address							immediate							
B-Type:	opcode				4'b0001				R	address1							4'b0				bmask			
BR	OP-BR				0	0	0	1	R	address1							0	0	0	0	bmask			
BRN	OP-BR				0	0	1	1	R	address1							0	0	0	0	bmask			

Figure 0.0.7: Σύνολο Εντολών του TP-ISA

Ροή Εργασίας των Επεξεργαστών

Η διατριβή επικεντρώνεται κυρίως στη μέτρηση των χαρακτηριστικών του υλικού με τη χρήση εργαλείων EDA, στη διασταυρούμενη μεταγλώττιση εφαρμογών για διάφορους επεξεργαστές και στη διεξαγωγή προσομοιώσεων netlist-RTL. Τα κεντρικά στοιχεία αυτής της ροής εργασίας είναι το Synopsys Suite και ο προσομοιωτής Modelsim.

Για τον χαρακτηρισμό του υλικού, κάθε επεξεργαστής περιλαμβάνει ένα σύνολο αρχείων περιγραφής υλικού που χρησιμοποιούνται παράλληλα με τις τυποποιημένες βιβλιοθήκες κελιών στο εργαλείο EDA. Οι μεταγλωττιστές, όπως οι msp430-gcc-toolchain και zpugcc, είναι απαραίτητοι για τη δημιουργία εκτελέσιμων αρχείων από κώδικα C με στόχο συγκεκριμένες αρχιτεκτονικές. Οι εφαρμογές προσαρμόζονται για κάθε επεξεργαστή και μεταγλωττιστή, λαμβάνοντας υπόψη τις αρχιτεκτονικές και λειτουργικές διαφορές. Οι προσομοιώσεις Netlist και RTL πραγματοποιούνται με τη χρήση του Modelsim για το Pulpino και του Synopsys VCS για το OMSP430. Η συνολική διαδικασία ροής εργασιών απεικονίζεται στο Σχήμα 0.0.8, ξεκινώντας από την κατασκευή της εφαρμογής και προχωρώντας μέσω των ενοτήτων μείωσης υλικού και ανάλυσης. Λεπτομερείς επεξηγήσεις αυτών των ενοτήτων παρέχονται στις επόμενες ενότητες.

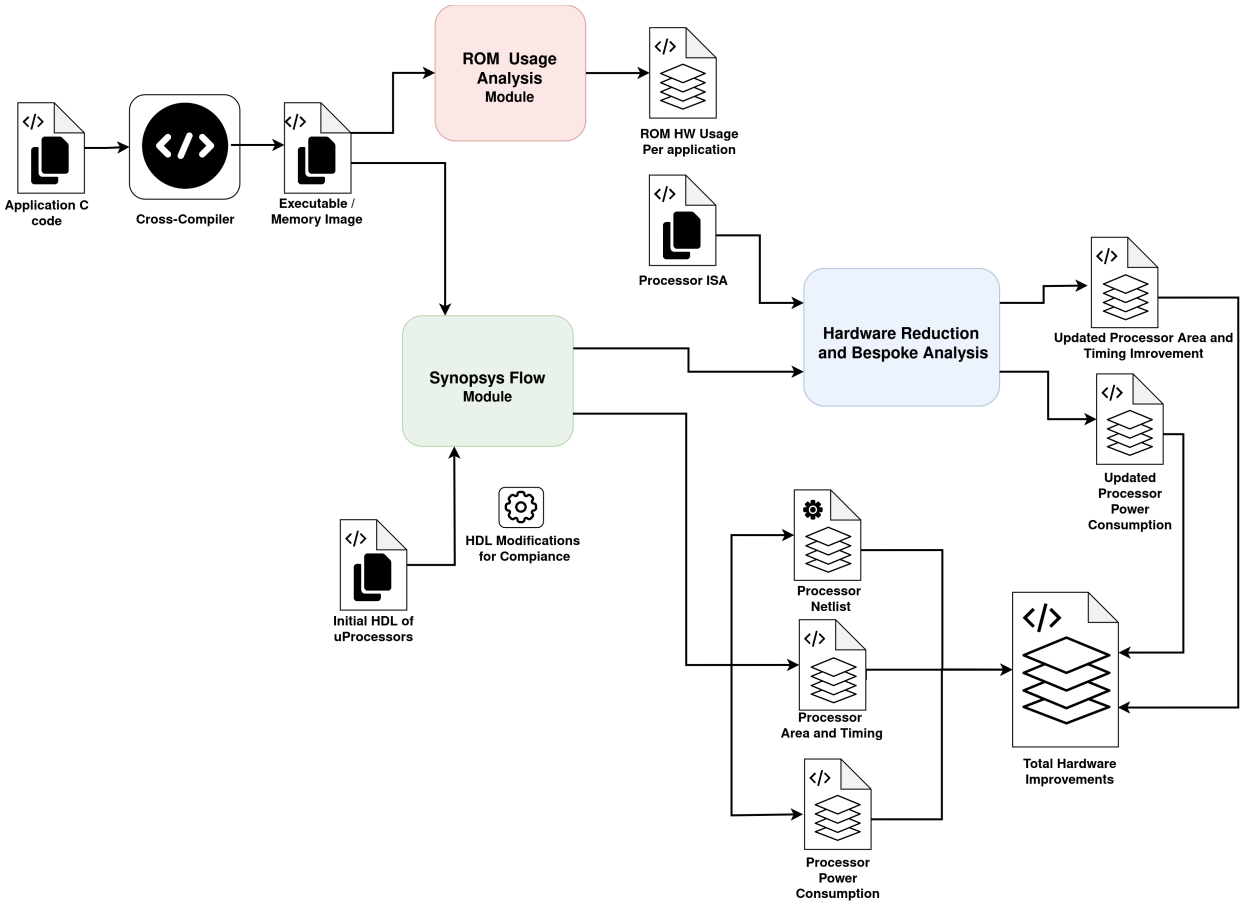

Figure 0.0.8: Συνολική ροή εργασίας

Ανάλυση Εφαρμογών και Επεξεργαστών

Ξεκινάμε με την ανάλυση υλικού χρησιμοποιώντας το Synopsys Flow Module, όπως φαίνεται στο σχήμα 0.0.11 για μετρήσεις των επεξεργαστών. Οι μεταγλωττιστές Synopsys και ο προσομοιωτής VCS χρησιμοποιούνται, μαζί με τη βιβλιοθήκη EGFET Standard Cell Library, για την παραγωγή βασικών πληροφοριών υλικού 0.0.9 και ιχνών εκτέλεσης με ακρίβεια κύκλου. Οι μετρήσεις περιλαμβάνουν επεξεργαστές με και χωρίς μονάδες πολλαπλασιαστή.

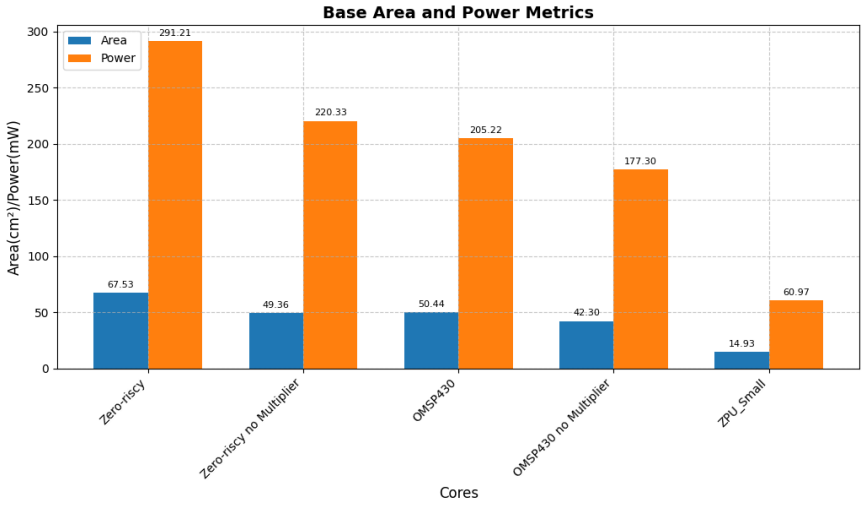

Figure 0.0.9: Μετρήσεις Επιφάνειας και Ισχύος για τους βασικούς επεξεργαστές

Οι λεπτομέρειες χρονισμού δείχνουν τη δυνατότητα αύξησης του ρολογιού χωρίς επιβάρυνση επιφάνειας. Στο σχήμα 0.0.10 βλέπουμε ποσοστιαίες μετρήσεις των των κύριων κομματιών για τους επεξεργαστές.

Η register file καταναλώνει σημαντική έκταση και πόρους ισχύος, ιδίως για τον OMSP430, ενώ οι πολλαπλασιαστές εισάγουν αξιοσημείωτη επιβάρυνση επιφάνειας και ισχύος. Τα χαρακτηριστικά υλικού είναι ευαίσθητα στις διάφορες ρυθμίσεις TP-ISA, ιδίως το πλάτος δεδομένων και το βάθος pipeline. Η επιφάνεια, η ισχύς και ο χρονισμός επηρεάζονται από τις αλλαγές στη διαμόρφωση, με το πλάτος δεδομένων να έχει σημαντικό αντίκτυπο καθώς το μικρότερο design πετυχαίνει ως και 3x λιγότερη επιφάνεια και κατανάλωση ισχύος έχοντας 24% ταχύτερο ρολόι.

Παρακάτω στον πίνακα 1 φαίνεται το σύνολο των εφαρμογών με το οποίο αξιολογούμε τους επεξεργαστές και τη ροή της δουλειάς.

Αφου εκτελέσουμε τις εφαρμογές στους επεξεργαστές με simulation λαμβάνουμε τους αρχικούς χρόνους εκτέλεσης και προχωράμε σε ανάλυση της ROM όπως φαίνεται στο σχήμα 0.0.13 . Χρησιμοποιώντας την EGFET τεχνολογία μνήμης για τυπωμένους μικροεπεξεργαστές [8], εκτιμούμε την επιβάρυνση υλικού της ROM λόγω μεγέθους προγράμματος, υπολογισμένο με την εξίσωση 0.0.1.

$$\begin{aligned} ROM_{Area} &= Core_{DW} * Instr * Cell_{Area} \\ ROM_{Power} &= Core_{DW} * Instr * (Cell_{AP} + Cell_{SP}) \end{aligned} \tag{0.0.1}$$

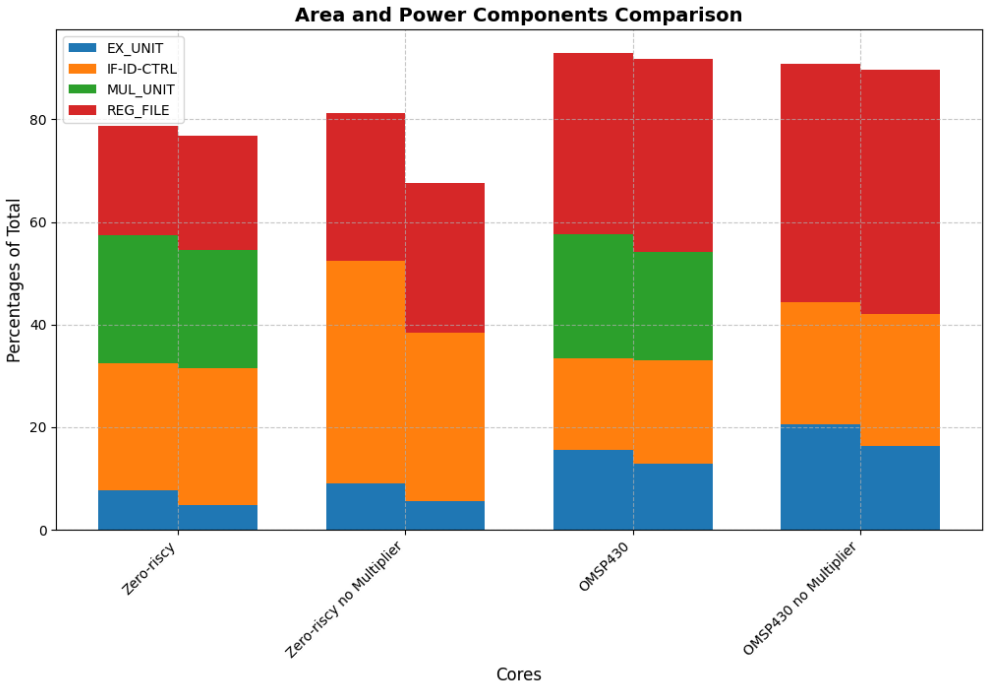

Figure 0.0.10: Ποσοστιαία ανάλυση καταναλώσεων Zero-riscy και OMSP430

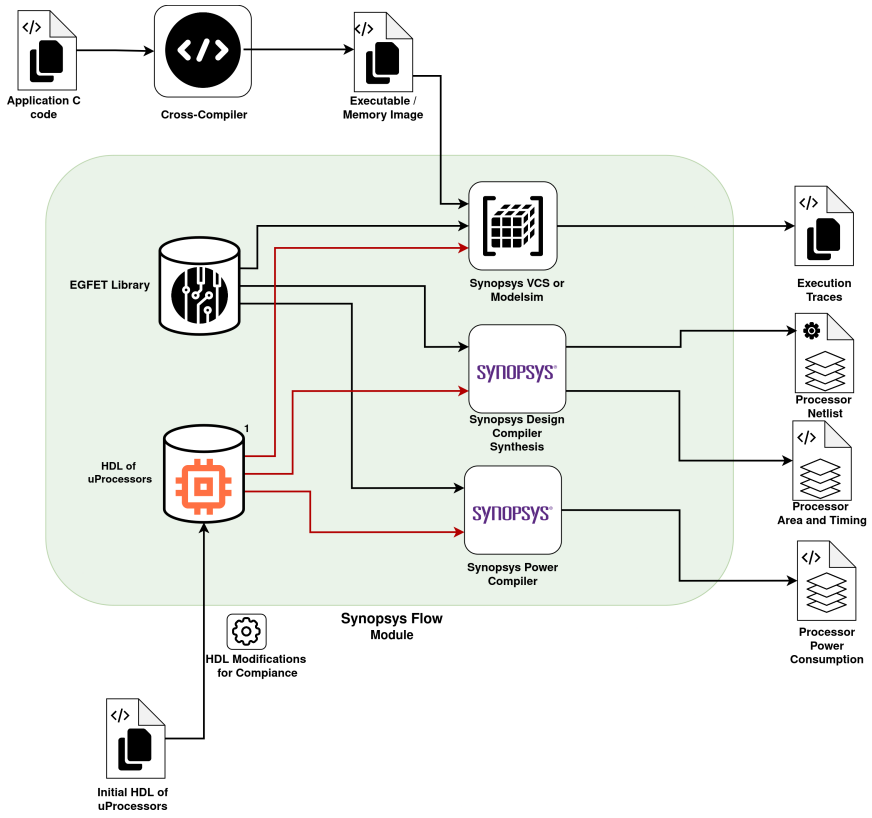

Figure 0.0.11: Αναλυτική περιγραφή του Synopsys Flow Module

Table 1: Περιγραφές των εφαρμογών

Benchmarks	**Description**
mult	Unsigned integer multiplication
div	Unsigned integer division
inSort	In-place Insertion Sort on array of size 16
intAvg	Signed integer average on array of size 16
crc8	Cyclic Redundancy Code for 16 byte array
tHold	Digital Threshold Detector on array of size 16 with hardcoded threshold
MLP	MLP with 3 Layers of size 4, 10 and 4 with relu activation function, run for 1 Inference
DTree	DTree of Depth 2 with hardcoded compare values, run for 1 Inference

Για την ανάλυση χρησιμοποιούνται τα χαρακτηριστικά των κελιών ROM. Η διαδικασία περιλαμβάνει τη συγγραφή κώδικα C και την παραγωγή μνήμης προγράμματος από έναν cross-compiler. Οι μετρήσεις περιλαμβάνουν τους τυπικούς επεξεργαστές και ZPU_Small, αναλύοντας με και χωρίς μονάδες πολλαπλασιαστή 0.0.12.

Η επιβάρυνση της ROM είναι σημαντική, ειδικά για πιο πολύπλοκες εφαρμογές και χωρίς μονάδα Mult. Οι ZPU_Small και Zero-riscy παρουσιάζουν μεγαλύτερες και πιο ενεργοβόρες ROM. Συγκριτικά, ο OMSP430 απαιτεί λιγότερα κελιά ROM ανά εντολή λόγω της αρχιτεκτονικής των 16 bit έναντι των 32 bit στους υπόλοιπους. Η επιβάρυνση της ROM για το Zero-riscy βελτιώνεται με την ενσωμάτωση του RV32M ISA, επωφελούμενη από τις πράξεις πολλαπλασιασμού. Αντίθετα, το μέγεθος ROM του OMSP430 βελτιώνεται οριακά με μια μονάδα πολλαπλασιαστή λόγω στρατηγικών βελτιστοποίησης του μεταγλωττιστή όπου κάνει unroll τον κώδικα με mult.

Μείωση Υλικού σε Χαμηλό και Υψηλό Επίπεδο

Με στόχο την βελτίωση των επεξεργαστών αφαιρόντας υλικό χρησιμοποιούμε πληροφορία από τα simulations με χρήση του Hardware Reduction Module 0.0.14.

Το πρώτο βήμα είναι ο εντοπισμός και αφαίρεση ολόκληρων στοιχείων των επεξεργαστών, το οποίο αποδεικνύεται ζωτικής σημασίας για σημαντική αύξηση της επιφάνειας και της ισχύος. Για τον OMSP430, ο έλεγχος για τα συμπεριλαμβανόμενα/αποκλειόμενα στοιχεία γίνεται μέσω ενός αρχείου ορισμών verilog. Βασικές λειτουργίες όπως το Frontend, η μονάδα εκτέλεσης και η μονάδα ρολογιού είναι απαραίτητες και διατηρούνται. Οι αχρησιμοποίητες λειτουργίες όπως οι λειτουργίες εντοπισμού σφαλμάτων, τα εξωτερικά πρωτόκολλα επικοινωνίας και η υποστήριξη DMA αφαιρούνται. Παρόμοια βήματα γίνονται για τον Zero-riscy, με προσοχή στη διατήρηση της λειτουργικής ισοδυναμίας καθώς αφαιρούνται οι μονάδες Debug Unit, Interrupt Controller και Compressed Decoder. Τα αποτελέσματα δείχνουν τη μειωμένη επιφάνεια και κατανάλωση ισχύος, με το R, στον πίνακα 2.

Στη συνέχεια, εντοπίζουμε και αφαιρούμε τις αχρησιμοποίητες εντολές στις εφαρμογές μας, χρησιμοποιώντας γνώση για το σύνολο εντολών κάθε επεξεργαστή. Αυτά τα αρχεία αναλύουν

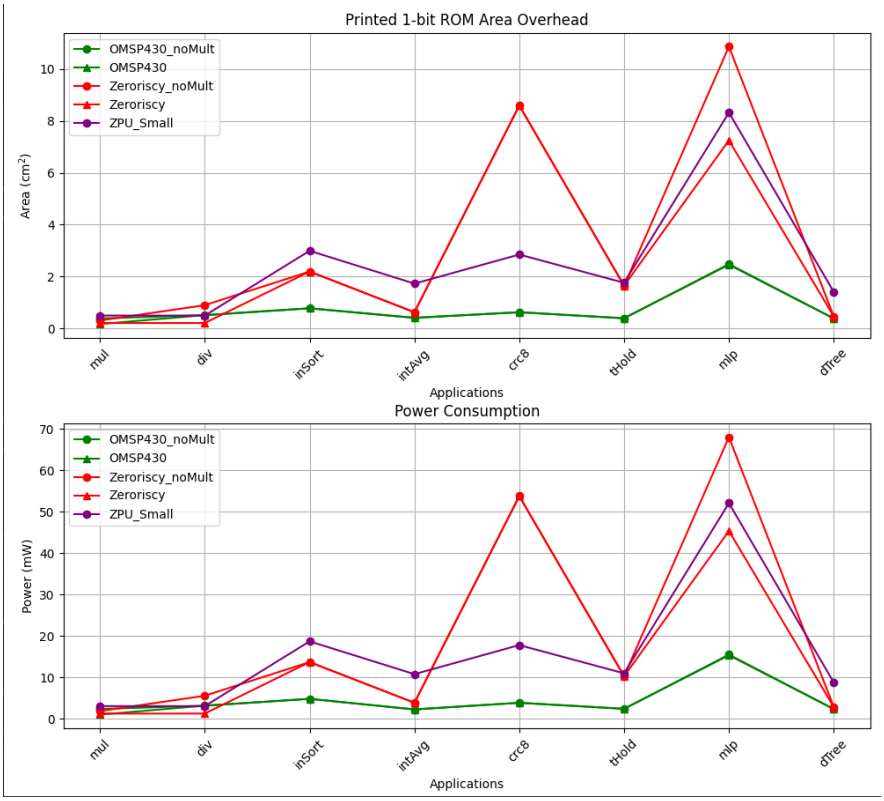

Figure 0.0.12: Επιβάρυνση σε επιφάνεια και ισχύ λόγω της ROM για κάθε εφαρμογή χρησιμοποιώντας τυπωμένη τεχνολογία 1-bit ROM κελιών

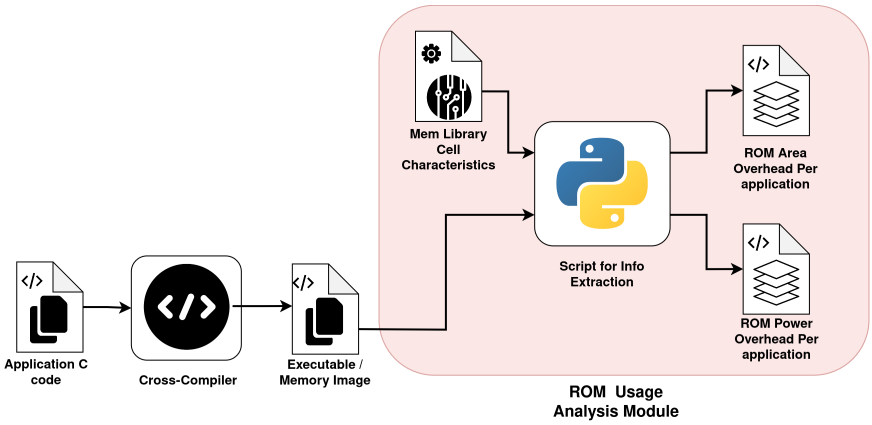

Figure 0.0.13: Αναλυτική Περιγραφή του ROM Analysis module

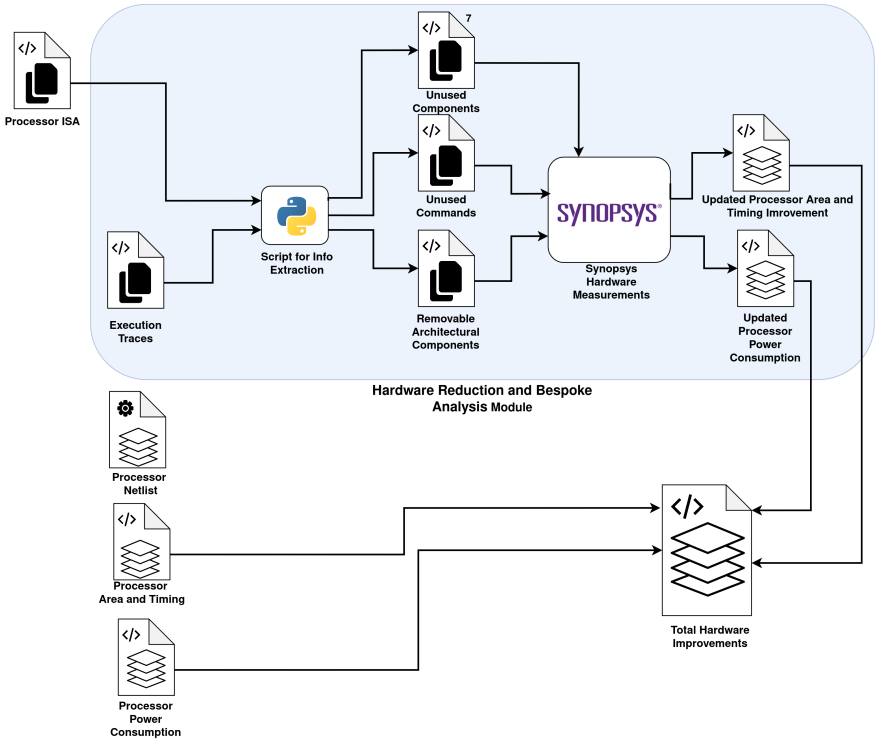

Figure 0.0.14: Αναλυτική περιγραφή του HW Reduction module

Table 2: Σύγκριση μεταξύ αρχικών και μειωμένων υλοποιήσεων

Cores	Area	Power	Max Timing	Synthesized Timing
Zeroriscy	67.53	291.21	14.49	5.91
Zeroriscy_R	61.82	264.31	14.59	5.91
Zeroriscy_noMul	49.36	220.33	14.49	5.91
Zeroriscy_noMul_R	43.08	190.98	14.87	5.91
OMSP430	50.44	205.22	4.25	4.07
OMSP430_R	33.21	132.31	4.94	4.07
OMSP430_noMul	42.30	177.30	4.25	4.07
OMSP430_noMul_R	25.07	104.19	4.94	4.07

τις εξόδους του ανιχνευτή για να εντοπίσουν τις αχρησιμοποίητες εντολές, οι οποίες στη συνέχεια αφαιρούνται από τον επεξεργαστή. Η αφαίρεση περιλαμβάνει σύνθετες διαδικασίες στις μονάδες αποκωδικοποίησης και εκτέλεσης, λαμβάνοντας υπόψη την κατανομή πόρων μεταξύ των εντολών. Για το Pulpino, οι αχρησιμοποίητες εντολές εντοπίζονται,οπως φαίνεται στον πίνακα 3, και ελέγχονται για δυνατότητες μείωσης του υλικού, με αποτέλεσμα ελάχιστα κέρδη.

Table 3: Αχρησιμοποίητες εντολές Zero-riscy

Type	**Instructions**
Set Less Than	slt, sltu, sltiu
Control Status Register	csrrs, csrrc, csrrwi, csrrsi, csrrci
Multiplication High Byte	mulh, mulhu, mulhsu
System Calls	ecall, ebreak, wfi

Ωστόσο, για τον OMSP430, οι περισσότερες αχρησιμοποίητες εντολές μοιράζονται υλικό με τις χρησιμοποιούμενες εντολές, περιορίζοντας τα σημαντικά κέρδη από αυτή την προσέγγιση. Τα συνολικά κέρδη από τη διαδικασία αφαίρεσης αχρησιμοποίητων εντολών φαίνονται στο πίνακα 4. Η λεπτομερής ανάλυση μπορεί να προσφέρει περαιτέρω βελτιώσεις, όπως συζητείται στην επόμενη ενότητα.

Table 4: Κέρδη υλικού από αχρησιμοποίητες εντολές Zero-riscy

Core	**Area(cm2)**	**Power(mW)**
Zero-riscy	67.53	291.21
Zero-riscy Red_isa	66.03	285.78
Zero-riscy noMul	49.36	220.33
Zero-riscy Red_isa noMul	48.49	217.80

Αφού εξαλείψαμε τις αχρησιμοποίητες εντολές και το βασικό μονοπάτι δεδομένων των εντολών, προχωράμε με τον εξορθολογισμό των λεπτότερων αρχιτεκτονικών στοιχείων για κάθε εφαρμογή. Αυτό περιλαμβάνει την αξιολόγηση τριών βασικών αρχιτεκτονικών πτυχών:

1. Αρχιτεκτονικές σημαίες του καταχωρητή κατάστασης: Αυτές εξετάζονται για να εκτιμηθεί η αναγκαιότητά τους.

2. Αριθμός αρχιτεκτονικών καταχωρητών γενικού σκοπού: Αξιολογείται η ανάγκη για καταχωρητές προσαρμοσμένους σε κάθε εφαρμογή.

3. Μέγεθος καταχωρητή μετρητή προγράμματος(PC) και διεύθυνσης βάσης(BAR): Αναλύονται οι απαιτήσεις μεγέθους τους.

Αυτή η στρατηγική βελτιστοποίησης αποσκοπεί στην προσαρμογή του επεξεργαστή στις ειδικές ανάγκες κάθε εφαρμογής, ενισχύοντας ενδεχομένως τη συνολική αποδοτικότητα της σχεδίασης. Οι αρχιτεκτονικές σημαίες, που χρησιμεύουν ως δυαδικοί δείκτες που επηρεάζουν τη συμπεριφορά του επεξεργαστή, είναι ζωτικής σημασίας. Ενώ ο επεξεργαστής Zero-riscy που βασίζεται σε RISCV παραμένει αναλλοίωτος λόγω της απουσίας αρχιτεκτονικών σημαιών εκτός του CSR

Module, οι τέσσερις σημαίες της αρχιτεκτονικής OMSP430 εξετάζονται εξονυχιστικά ανά εφαρμογή .

Table 5: OMSP430 Αρχιτεκτονικές σημαίες που χρησιμοποιούνται ανα εφαρμογή

Application	**Required flags**
Baseline	4(Z, V, N, C)
AllApps	4
mult	4
div	3(no Z flag)
inSort	3(no V flag)
intAvg	4
crc8	4
tHold	4
mlp	3(no Z flag)
dTree	4
AllApps w/ Mult Unit	4
mult w/ Mult Unit	4
mlp w/ Mult Unit	3(no Z flag)

Οι αρχιτεκτονικοί καταχωρητές είναι θεμελιώδη στοιχεία αποθήκευσης, απαραίτητα για την εκτέλεση εντολών και τη διαχείριση της κατάστασης του επεξεργαστή. Προσαρμόζουμε τα σχέδια αναλόγως, υπολογίζοντας τους εξειδικευμένους καταχωρητές που επιβάλλει το RISCV32E ISA, εξασφαλίζοντας τουλάχιστον πέντε αρχιτεκτονικούς καταχωρητές. Η χρησιμοποίηση καταχωρητών από τους επεξεργαστές για κάθε εφαρμογή φαίνεται στους πίνακες 6, 7. Όσον αφορά τους αρχιτεκτονικούς καταχωρητές, εστιάζουμε στη μείωση του αριθμού για βελτιώσεις HW, ιδίως για την εφαρμογή "mult", όπου δοκιμάζεται η μείωση του αριθμού των καταχωρητών από 15 σε 7. Επιπλέον, αξιολογούμε τα κέρδη από τη μείωση του πλάτους δεδομένων των καταχωρητών στο μισό από 32 σε 16 bits.

Ο PC και ο BAR, που απευθύνονται σε χώρους μνήμης, είναι κρίσιμοι αρχιτεκτονικοί καταχωρητές. Αξιολογούμε τις απαιτήσεις μεγέθους τους με βάση τα μεγέθη της μνήμης προγράμματος και δεδομένων. Η ανάλυση υποστηρίζεται από αρχεία που παράγουν τα απαιτούμενα πλάτη bit για τη διευθυνσιοδότηση. Όσον αφορά τα κέρδη σε επιφάνεια και ισχύ, το μέγεθος του PC εξετάζεται εξονυχιστικά, ιδιαίτερα για την εφαρμογή "mult", που απαιτεί διεύθυνση 5 bit για τη μνήμη προγράμματος. Παρατηρούμε κέρδη σε επιφάνεια και ισχύ 5.31cm2, 2cm2 και 24.89mW, 6mW για τους επεξεργαστές, για μείωση του PC σε πλάτος 5. Τα αποτελέσματα παρουσιάζονται αναλυτικά στο πίνακα 8.

Επιταχυντές Εφαρμογών Μηχανικής Μάθησης

Σε αυτό το κεφάλαιο παρουσιάζουμε υπολογιστικές μονάδες για την επιτάχυνση εφαρμογών Μηχανικής Μάθησης.

Table 6: Zero-riscy Αριθμός καταχωρητών γενικού σκοπού ανα εφαρμογή

Application	**# of required Registers**
Baseline	15
AllApps	14
mult	7
div	10
inSort	14
intAvg	7
crc8	8
tHold	8
mlp	12
dTree	8
AllApps w/ Mult Unit	14
mult w/ Mult Unit	6
div w/ Mult Unit	6
mlp w/ Mult Unit	14

Table 7: OMSP430 Αριθμός καταχωρητών γενικού σκοπού ανα εφαρμογή

Application	**# of required Registers**
Baseline	15
AllApps	10
mult	8
div	8
inSort	7
intAvg	6
crc8	5
tHold	6
mlp	10
dTree	5
AllApps w/ Mult Unit	9
mult w/ Mult Unit	6
mlp w/ Mult Unit	9

Table 8: Απαιτούμενο πλάτος καταχωρητών για διευθυνσιοδότηση μνημών

Application	**Pulpino PC**	**Pulpino BAR**	**OMSP430 PC**	**OMSP430 BAR**
Baseline	32	32	16	10
AllApps	10	8	9	8
mult	5	-	5	-
div	7	-	5	-
inSort	8	5	7	5
intAvg	7	-	6	-
crc8	7	-	7	-
tHold	10	5	6	5
mlp	10	8	9	8
dTree	5	-	6	-
AllApps w/ Mult Unit	10	8	9	8
mult w/ Mult Unit	4	-	5	-
div w/ Mult Unit	4	-	-	-
mlp w/ Mult Unit	9	8	9	8

Επιταχυντής πράξεων Νευρωνικών και Κλιμάκωση Ακρίβειας

Με τη γνώση ότι τα MLPs και SVMs αποτελούνται κυρίως από πράξεις πολλαπλασιασμού-συσσώρευσης (MAC), σχεδιάζουμε μία μονάδα που επιταχύνει αυτή τη πράξη. Ο επιταχυντής, υλοποιημένος σε Verilog HDL, επεκτείνει τους επεξεργαστές προσθέτοντας εντολές στο ISA, με πρόσβαση μέσω του αποκωδικοποιητή. Εκτελεί πράξεις MAC, αποθηκεύοντας τα αποτελέσματα σε έναν συσσωρευτή, και επαναφέρει/φορτώνει το bias. Οι λειτουργίες πραγματοποιούνται σε έναν μόνο κύκλο. Οι επεξεργαστές και η νευρωνική μονάδα χαρακτηρίζονται με τη χρήση της ροής εργαλείων Synopsys EDA για τη σύγκριση της επιφάνειας, της ισχύος και του χρονισμού έναντι της βασικής γραμμής. Η εισαγωγή του επιταχυντή μπορεί να εισάγει επιβάρυνση στις μετρήσεις υλικού, ενώ η αντικατάσταση μονάδων πολλαπλασιασμού που ήδη υπάρχουν παρουσιάζουμε τα tradeoffs.

Προχωράμε αυτή τη βασική μονάδα υλοποιόντας τεχνικές κλιμάκωσης ακρίβειας. Στόχος μας είναι να βελτιώσουμε την αποτελεσματικότητα της εκτέλεσης και να μειώσουμε την κατανάλωση ενέργειας σε έναν τυπωμένο επεξεργαστή για υπολογισμούς ML. Διερευνούμε την ισορροπία μεταξύ της ακρίβειας του μοντέλου και της υπολογιστικής επιτάχυνσης. Η βασική κλιμάκωση ακρίβειας αποδίδει εξοικονόμηση ισχύος, επιφάνειας και πιθανή εξοικονόμηση χρονισμού, αλλά δεν μειώνει τους κύκλους εκτέλεσης. Για να το αντιμετωπίσουμε αυτό, χρησιμοποιούμε το πλήρες εύρος bit του επεξεργαστή Zero-riscy και πολυπλέκουμε μέγεθος 32 bit για πολλαπλές πράξεις MAC χαμηλότερης ακρίβειας. Οι μειωμένες μονάδες MAC (16x16, 8x8, 4x4) επιτρέπουν τον ταυτόχρονο υπολογισμό 2, 4 και 8 μερικών παραγώγων σε έναν κύκλο, όπως περιγράφεται στην εξίσωση 0.0.1. Με τη σχεδίαση 16x16, η τιμή ενός νευρώνα μπορεί να υπολογιστεί στους μισούς κύκλους. Η περαιτέρω κλιμάκωση (8x8, 4x4) ακολουθεί παρόμοια προσέγγιση, απαιτώντας μια λειτουργία μεταξύ των επιπέδων του δικτύου για την αναδιοργάνωση των δεδομένων για αποδοτικό υπολογισμό. Το σχηματικό της μονάδας φαίνεται στο σχέδιο 0.0.15.

$$
\begin{aligned}
acc_1 &= (r1[7:0] * r2[7:0]) + acc_1; \\
acc_2 &= (r1[15:8] * r2[15:8]) + acc_2; \\
acc_3 &= (r1[23:16] * r2[23:16]) + acc_3; \\
acc_4 &= (r1[31:24] * r2[31:24]) + acc_4; \\
accum &= acc_4 + acc_3 + acc_2 + acc_1;
\end{aligned} \tag{0.0.1}
$$

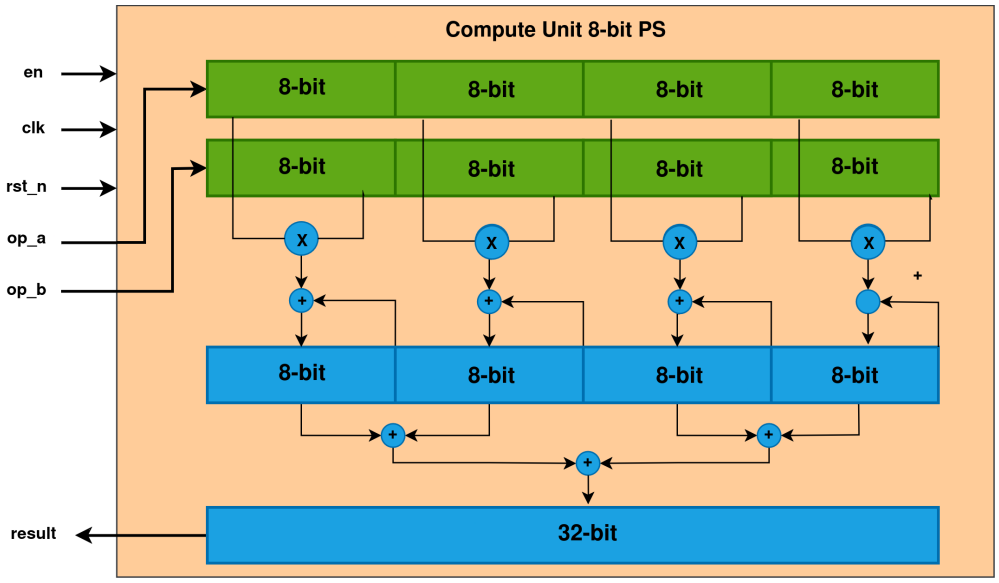

Figure 0.0.15: Υπολογιστική Μονάδα με κλιμάκωση ακρίβειας 8 bit

Πειραματικό Μέρος

Μοντέλα υπό Εξέταση

Το πειραματικό κομμάτι υλοποίησε τις παρακάτω εφαρμογές:

- Το σύνολο δεδομένων RedWine το οποίο επικεντρώνεται σε δείγματα ερυθρών οίνων από τη βόρεια Πορτογαλία, αποσκοπεί στη μοντελοποίηση της ποιότητας των οίνων με βάση 11 φυσικοχημικά χαρακτηριστικά. Η καταλληλότητά του για τις τυπωμένες τεχνολογίες έγκειται στην ευελιξία, το χαμηλό κόστος και τη δυνατότητα διάθεσης, επιτρέποντας την εφαρμογή του σε φιάλες κρασιού για την πρόβλεψη της ποιότητας.
- Το σύνολο δεδομένων WhiteWine ασχολείται με δείγματα λευκού κρασιού από την ίδια περιοχή, μοντελοποιώντας επίσης την ποιότητα του κρασιού με βάση 11 χαρακτηριστικά.
- Το σύνολο δεδομένων Cardio περιλαμβάνει μετρήσεις του εμβρυϊκού καρδιακού ρυθμού (FHR) και της συστολής της μήτρας (UC) από καρδιοτοκογραφήματα, ταξινομημένα από μαιευτήρες. Στόχος είναι η πρόβλεψη της κατάστασης του εμβρύου με τη χρήση 21 σχετικών χαρακτηριστικών

  Για κάθε μοντέλο αξιολογήθηκαν υλοποιήσεις σε MLP και SVM. Όλα τα παραπάνω μοντέλα ανήκουν στενά στο πεδίο των εφαρμογών για τυπωμένα ηλεκτρονικά.

Πειραματική Διάταξη

Στο σχήμα 0.0.16 φαίνεται συνολικά η πειραματική διάταξη που χρησιμοποιήθηκε.

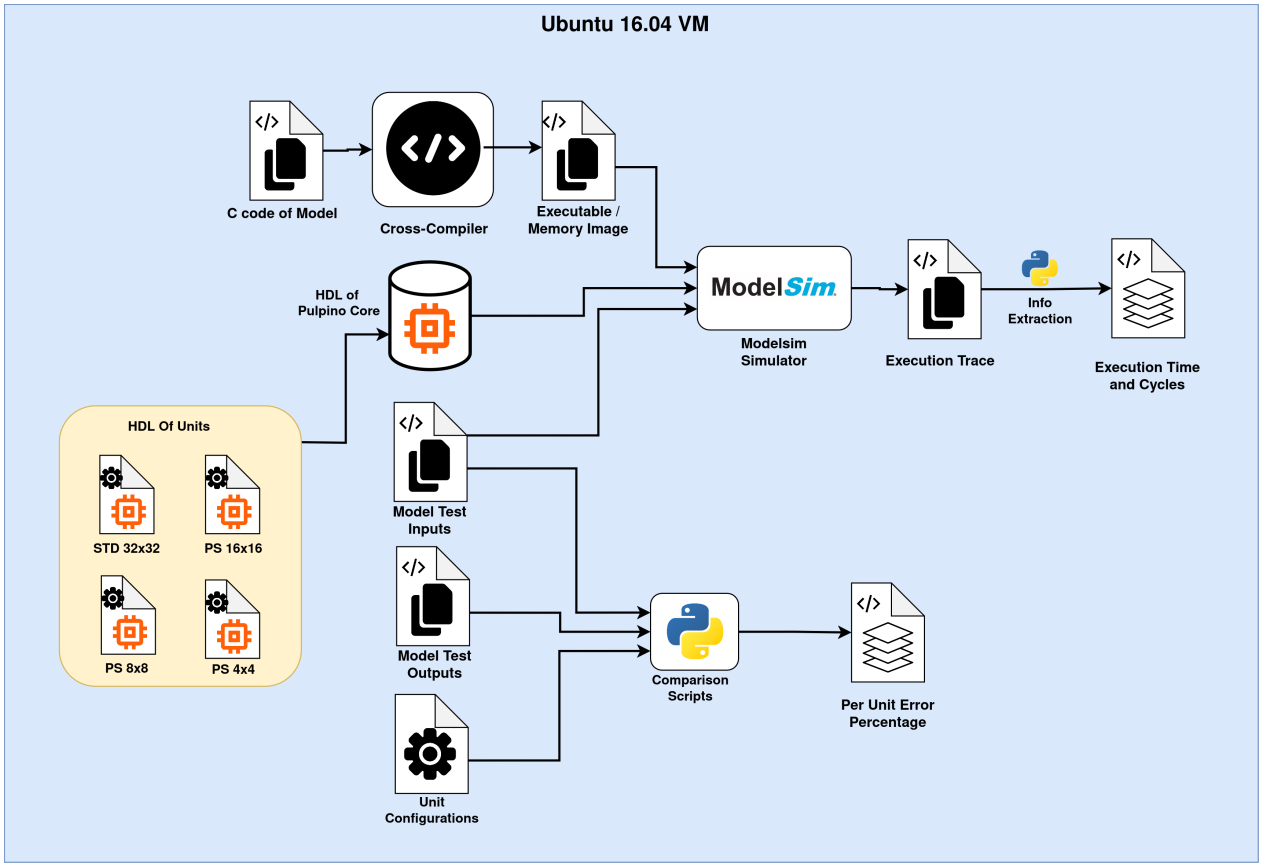

Figure 0.0.16: Λεπτομερές Σχηματικό Πειραματικής Διάταξης

Ποιό συνοπτικά, το περιβάλλον προσομοίωσης δημιουργήθηκε στο Ubuntu 16.04 σε ένα QEMU VM για την υποστήριξη των απαιτούμενων εργαλείων, συμπεριλαμβανομένου του Modelsim-Intel HDL Simulator για τις προσομοιώσεις rtl του pulpino SoC. Η μεταγλώττιση των εφαρμογών έγινε με τη χρήση του ri5cy-gnu-toolchain, δημιουργώντας κώδικα assembly. Αναπτύχθηκαν αρχεία Python για τον υπολογισμό των ακριβών αριθμών κύκλων από τον κώδικα assembly και τα ίχνη εκτέλεσης.

Διερευνήσαμε τη χρήση βαρών γραμμένων απευθείας στον κώδικα για τη μείωση της πρόσβασης στη RAM, αλλά επιλέξαμε την αποθήκευση στη RAM για τη μείωση του μεγέθους του προγράμματος και της ROM. Τα ποσοστά επιτάχυνσης εξήχθησαν εκτελώντας κάθε μοντέλο σε 50-100 εισόδους. Οι μετρήσεις υλικού περιλάμβαναν μεγέθη ROM και διερεύνηση των συμβιβασμών μεταξύ υλοποιήσεων, λαμβάνοντας υπόψη τις επιλογές κλιμάκωσης ακρίβειας. Ο TP-ISA θεωρήθηκε ελκυστική επιλογή, παρέχοντας χαρακτηριστικά υλικού και κύκλους εκτέλεσης με βάση την έξοδο ri5cy-gnu-toolchain. Η κλιμάκωση ακριβείας περιελάμβανε την τροποποίηση του κώδικα verilog και C για την ενσωμάτωση νέων μονάδων και τον χειρισμό της μετατόπισης τιμών μεταξύ των επιπέδων MLP. Ένα αρχείο Python υπολογίζει την απώλεια ακρίβειας συγκρίνοντας τις εξόδους με πλήρη ακρίβεια και κλιμακωτή ακρίβεια, παρουσιάζοντας το ποσοστό της απώλειας ακρίβειας λόγω σφαλμάτων ταξινόμησης ή παλινδρόμησης.

Αποτελέσματα

Εκτελούμε όλες τις διαμορφώσεις των υπολογιστικών μονάδων και επεξεργαστών στον Zero-riscy για να λάβουμε ίχνη εκτέλεσης. Αρχικά, προσδιορίζουμε το μέγεθος της ROM με βάση τον αριθμό των εντολών, ο οποίος είναι 1084 στη χειρότερη περίπτωση για τα μοντέλα μας, το οποίο συνιστά ελάχιστο μετρητή προγράμματος πλάτους 11-bit. Μετράμε όλες τις πιθανές διαμορφώσεις TP-ISA για pc=11, συμπεριλαμβανομένων των νευρωνικών μονάδων με διάφορες ακρίβειες.

Μετράμε τις προτεινόμενες υλοποιήσεις Zero-riscy αξιοποιόντας τις υπολογιστικές μας μονάδες και τις συγκρίνουμε με τη βασική Zero-riscy για τις προδιαγραφές υλικού στον πίνακα 9. Το Be σε κάθε όνομα σημαίνει Bespoke και είναι η προτεινόμενη Zero-riscy με εφαρμοσμένη μείωση υλικού και μείωση isa.

Core	Core Area	Power	Max Clock
Zeroriscy_Mul	67.53	291.21	14.49
Zeroriscy noMul	49.35	220.33	14.49
Zeroriscy_Be_Mult	60.32	257.80	14.59
Zeroriscy_Be_MAC 32x32	61.96	249.02	14.59
Zeroriscy_Be_MAC PS_16	52.47	222.27	15.02
Zeroriscy_Be_MAC PS_8	47.70	207.37	15.15
Zeroriscy_Be_MAC PS_4	42.86	191.70	15.65

Table 9: Μετρικές Υλικού για διαφερετικές ρυθμίσεις του Zero-riscy

Υπολογίζουμε την απώλεια ακρίβειας των μοντέλων λόγω της κλιμάκωσης της ακρίβειας, με την ακρίβεια 4-bit να αποδίδει δυνητικά σφάλμα έως και 26%, εκτός από το σύνολο δεδομένων Cardio όπου πέφτει στο 1,5%. Εκτιμούμε τέλος την επιτάχυνση με το TP-ISA, αν και δεν έχουμε πρόσβαση στον μεταγλωττιστή, ενσωματώνοντας τις μονάδες στη βασική σχεδίαση και εκτελώντας τη ροή μέτρησης στοιχείων υλικού, χρησιμοποιώντας ίχνη εκτέλεσης από το Zero-riscy για την εκτίμηση των κύκλων.

Στον πίνακα 10 παρουσιάζονται οι βελτιώσεις μαζί με το σφάλμα που επιφέρει η κάθε μία.

Table 10: Βελτιώσεις των προτεινόμενων αρχιτεκτωνικών έναντι του αρχικού Zero-riscy

Cores	**Area Gain**	**Power Gain**	**Avg Speedup**	**Error**
Zeroriscy-Bespoke	10.6%	11.4%	0%	0.0%
Zeroriscy-Bespoke_MAC32	8.2%	14.4%	23.93%	0.0%
Zeroriscy-Bespoke_MACQ16	22.2%	23.6%	33.79%	0.0%
Zeroriscy-Bespoke_MACQ8	29.3%	28.7%	41.73%	0.5%
Zeroriscy-Bespoke_MACQ4	**36.5%**	**34.1%**	**46.4%**	**15.66%**

Οι πίνακες 11 και 12 συγκρίνουν τις βασικές διαμορφώσεις TP-ISA με τις προτεινόμενες διαμορφώσεις TP-ISA που περιέχουν εκδόσεις της μονάδας μας όσον αφορά την έκταση, την ισχύ, την απώλεια ακρίβειας και την πρόχειρη εκτιμώμενη επιτάχυνση. Ο Πίνακας 11 περιέχει τη σύγκριση της βασικής ακριβούς υλοποίησης 32 bit TP-ISA με την προτεινόμενη ταχύτερη διαμόρφωση 32

bit με αποδεκτό σφάλμα (d32_pc11_MAC32_Q8) και ο Πίνακας 12 συγκρίνει τη βασική μικρή διαμόρφωση 8 bit TP-ISA με τη μικρότερη διαμόρφωσή μας (d8_pc11_MAC8). Λαμβάνουμε υπόψη ότι δεν υπάρχει μονάδα πολλαπλασιασμού σε κανέναν από τους βασικούς επεξεργαστές, οπότε όλες οι πράξεις MAC τροφοδοτούνται μέσω της ALU σε πολλαπλούς κύκλους.

Table 11: Σύγκριση της προτεινόμενης γρήγορης 32 bit υλοποίησης έναντι του αρχικου 32 bit TP-ISA με ακρίβεια

Configuration	**Prop_FAST**
Area Overhead	**x2.12**
Power Overhead	**x1.97**
Avg Err (Base is 0%)	0.5%
Estimated Speedup	**up to 88.5%**

Table 12: Σύγκριση της προτεινόμενης μικρής 8 bit υλοποίησης έναντι της αρχικης 8 bit TP-ISA

Configuration	**Prop_SMALL**
Area Overhead	x1.98
Power Overhead	x1.82
Avg Err (Base is 0.5%)	0.5%
Estimated Speedup	**up to 85.1%**

Δεδομένου ότι πρόκειται για έναν πολύ μικρό επεξεργαστή βελτιστοποιημένο για χαμηλούς πόρους και δεν μπορούμε να αφαιρέσουμε καθόλου από το υλικό, αναμένουμε φυσικά μια αύξηση των πόρων υλικού, την οποία αντισταθμίζουμε με αυξημένη εκτιμώμενη επιτάχυνση. Εάν ο χρήστης δεν εκτιμά πραγματικά κάθε ποσοστό ακρίβειας στο μοντέλο, μπορούμε να περιμένουμε από 85,1% έως 88,5% μέση εκτιμώμενη επιτάχυνση για τα μοντέλα μας με περίπου x2 και x1,9 επιβάρυνση σε επιφάνεια και ισχύ για την TP-ISA. Η κλιμάκωση της ακρίβειας με λιγότερα bits αποδίδει καλύτερα αποτελέσματα αλλά απαγορευτικό σφάλμα, στις περισσότερες περιπτώσεις.

Συμπεράσματα

Σε αυτή τη διατριβή, διερευνούμε την υλοποίηση προσαρμοσμένων μικροεπεξεργαστών σε τυπωμένη τεχνολογία. Πρώτον, επισημαίνουμε την ανάγκη για μεγάλη μείωση της επιφάνειας και της ισχύος σε τέτοια κυκλώματα, προκειμένου να καταστεί δυνατή η υλοποίηση εφαρμογών ML σε εξαιρετικά μικρούς επεξεργαστές. Για το σκοπό αυτό, συντονίζουμε και υλοποιούμε επεξεργαστές χαμηλού αριθμού πυλών, ικανούς να εκτελούν συγκεκριμένες εφαρμογές, αφαιρώντας όλη τη λογική που εγγυημένα δεν θα χρησιμοποιηθεί. Η αφαίρεση των πλήρως αχρησιμοποίητων στοιχείων αποδεικνύεται πολύ επωφελής και αποδοτική ως προς την προσπάθεια βελτίωσης χρόνου. Η αφαίρεση αρχιτεκτονικών στοιχείων και αχρησιμοποίητων εντολών μπορεί να είναι πιο χρονοβόρα, αλλά επίσης επωφελής, ιδίως όταν στοχεύει σε ακολουθιακά στοιχεία του επεξεργαστή.

Κατά την εξέταση των επιβαρύνσεων της μνήμης προγράμματος στη σχεδίαση παρατηρούμε ότι ακόμη και ελαφρώς πιο σύνθετες εφαρμογές μπορούν γρήγορα να προκαλέσουν την κατάληψη πόρων από τη ROM που είναι συγκρίσιμοι με κάποιους από τους μικρούς επεξεργαστές. Όσον αφορά τη χρήση της ROM, τα σχέδια με μικρότερα μήκη εντολών πλεονεκτούν επειδή απαιτούν λιγότερα κελιά ROM, ενώ οι πολλαπλασιαστές αποδεικνύεται ότι αξίζει να εξεταστούν αφού οι πολύπλοκες εντολές καταλαμβάνουν χώρο στη μνήμη όταν πρέπει να γραφτούν για μια ALU.

Αναπτύσσουμε μια μονάδα ενός κύκλου προκειμένου να βελτιώσουμε τις επιδόσεις για φόρτους εργασίας ML, εφαρμόζουμε κλιμάκωση ακρίβειας στις μονάδες μας και δοκιμάζουμε με 3 μοντέλα MLP και SVM. Για τα μοντέλα που δοκιμάσαμε, μπορούμε να αυξήσουμε σημαντικά την απόδοση της μονάδας με παραλληλισμό με χρήση της κλιμάκωσης ακρίβειας με αμελητέα απώλεια ακρίβειας.

Ωστόσο, η πρόσθετη επιβάρυνση από την κατάλληλη ρύθμιση των εισόδων για τη μονάδα κλιμάκωσης ακριβείας μέσω της συνένωσης και της μετατόπισης μπορεί να αποτελέσει τροχοπέδη. Όσον αφορά τη μελλοντική εργασία, υπάρχουν περιθώρια για επεκτάσεις τόσο στον τομέα των εφαρμογών ML όπως αυτές που διερευνήσαμε όσο και για βελτίωση σε άλλους τομείς. Η netlist και η συμβολική προσομοίωση μπορούν να αξιοποιηθούν για μια συγκεκριμένη εφαρμογή με τη διάδοση απροσδιόριστων σημάτων στην προσομοίωση και την εξαγωγή πληροφοριών σχετικά με το αμετάβλητο επίπεδο των λογικών πυλών. Αυτές οι πληροφορίες μπορούν να χρησιμοποιηθούν για την προσαρμογή του επεξεργαστή σε μεγαλύτερο βαθμό, λαμβάνοντας μεγαλύτερα οφέλη όσον αφορά τη χρήση του υλικού.

Chapter 1

Introduction

Wearable, low-cost computing platforms have been considered for many use cases recently, especially with the rise of edge computing and battery powered applications. Amidst growing concerns about climate change and sustainability, prioritizing non-toxic, low power computing aligns with broader initiatives aimed at reducing carbon emissions and promoting eco-friendly technologies. Printed electronics are devices printed on flexible materials instead of the traditional silicon circuits, constituting a very low cost and non-toxic solution to the problems mentioned above.

Furthermore, since machine learning workloads seem to be at the core of most modern applications, the adaptation of printed technology towards low power and efficient execution of such workloads seems appealing. The usual choice for computing paradigms for the task can be ASICs and Full Processors, each with its strengths and weaknesses. In our case, printed processors can serve multiple purposes, allow for flexibility in the applications supported and are thus considered better for commercial use, compared to ASICs that are specific for a single application-dataset pair, cannot be reprogrammed and have higher overall production and design cost. In order to get all the benefits of the processors and maintain low hardware resources, it is mandatory to explore hardware reduction capabilities, tailoring each processor strictly to the applications to be run.

1.1 Related Work

The development of low cost, flexible and disposable computing circuits has been the topic of much recent research. [10] start by creating and evaluating a suite of low demand applications for earable programmable platforms that closely relate to the field of wearable applications in Printed Computing and develop a CGRA-like architecture that targets performance on the gives suite while maintaining constraints.

The main work on Printed microprocessors [8] presents the first standard cell libraries for Printed Computing and examines whether candidate microprocessors fit the constraints of printed applications. The work develops a processor specific to tackle the main problems of printed computing and measures the specifications of bespoke versions of the core, tailored to different applications. This work also develops a standard cell library for CNT-TFT and EGFET technologies for synthesis and physical design implementation with EDA tools, allowing for exploration, testing of designs and building a better understanding of computing properties for such circuits. It is shown through testing that sequential circuits in such technologies tend to have prohibitive area and power characteristics, and that RAM cells are much more costly compared to ROM cell.

[9] develop and fabricate a core on flexible material and test it with a representative application set. The results of the evaluation are used to tune the ISA of the core to better suit the target applications. They measure process variation on Printed Microprocessors and perform an exploration on various design points and list benefits from each approach. [13] propose a symbolic methodology for tailoring a processor to a specific application by removing unused gates during gate-simulation of the application. They measure benefits for a set of applications and provide a methodology for support of multiple applications per bespoke core. [31] generalize symbolic simulation for tailoring processors, removing the need for custom simulators per core and extending the work to cover wide varieties of HW-SW pairs.

There are also works on ML with printed electronics [6], [5], [7]. Work [27], which is mainly concerned with printed implementations of Classifiers, comparing bespoke and lookup based circuits for calculations. They evaluate and compare both digital and analog solutions. [3] focus on approximation techniques on Printed solutions with application specific circuits. An automated cross-layer approximation framework is proposed, considering MLPs and SVMs, followed by evaluation of the accuracy-hardware specs tradeoff.

1.2 Thesis Objectives

The target of this thesis is to examine and optimize Printed Microprocessors performance and resource consumption on Printed kernels and machine learning workloads such as Multi Layer Perceptrons and Support Vector Machines. We perform hardware reduction, removing not utilized units, instructions from the ISA and architectural features that are not required for the benchmarks suite we use. Eliminating this logic that is guaranteed to not be used by an application, can produce a design tailored to the application - a bespoke processor - that has significantly lower area and power than the original microprocessor that targets an arbitrary application. We develop a hardware unit that executes MAC operations that targets ML

Classifiers and explore precision scaling for the benefits and accuracy costs it entails to the unit. As a proof-of-concept we examine MLP and SVM datasets for printed applications and measure tradeoffs in the hardware characteristics and execution time speedup for each workload.

1.3 Thesis Outline

This thesis is organized in a total of 5 chapters with the following structure:

1. Theoretical Background

   This chapter presents a brief mandatory theoretical background for Printed Electronics and the EDA tools that were used. We introduce basic ML concepts and the types of ML applications that were tested. Finally we explain Precision Scaling and its usability for our scope.

2. Bespoke Analysis and Hardware Reduction

   This chapter provides the workflow for the extraction of hardware specifications including rom usage, processor area, timing and power as well as the process of eliminating hardware. We explain in detail the complete workflow starting from hardware analysis, application writing, compilation, simulation and extracting the execution information. Furthermore, we present our process for measuring hardware utilization regarding our needs and applications. We remove non-utilized hardware first in the level of components and subsequently in the finer, architectural level given the information received from our flow.

3. Machine Learning Acceleration Units

   In this chapter we develop a computing unit that targets the performance of our cores when handling ML workloads that are MAC intensive. We discuss the functionality and the benefits of the unit over the standard ML executions on the cores that contain or don't contain a hardware Multiplier. Finally we present the work regarding the application of Precision Scaling on the unit, exploring the relevant tradeoffs for our task.

4. Experiments and Results

   In this chapter we present the models that we used for testing, the experiment setup, methodology, and the results for our cores and units when running a suite of Printed ML workloads. We measure area, power and timing for hardware, the respective execution time and cycles of the targeted models and compare to the baseline processors. We analyse the power-execution-accuracy tradeoffs of using precision scaling with our units.

5. Conclusions and Future Work

In this chapter we mention conclusions from the previously examined experimental results and provide possible future work regarding improvements on the hardware and application execution metrics.

Chapter 2

Theoretical Background

This chapter examines the features of Printed Electronics, Printed Computing and describes the function of the Synopsys EDA tool in synthesizing hardware and measuring hardware specifications. Furthermore, we provide information about the types of ML workloads that were examined and tested in the the thesis. We introduce Precision Scaling and potentials scopes of use.

2.1 Printed Computing

2.1.1 Printed Electronics

Printed electronics technology represents a burgeoning field within the domain of electronic engineering, offering novel methods for fabricating electronic devices using printing processes on various substrates. Unlike traditional electronic manufacturing techniques that rely on photolithography and vacuum deposition, printed electronics utilize low-cost additive manufacturing processes, such as inkjet printing, screen printing, or flexographic printing, to deposit functional materials in specific patterns to create electronic circuits and devices. The main categories for printing technologies are subtractive and additive. In subtractive printing, deposition and etching steps follow one another, making it expensive and time consuming but creating faster and smaller circuits [12], [11]. Opposite to that, additive printing comes with a much faster manufacturing process, requiring only deposition steps for all types of components [32]. The main drawbacks compared to subtractive technologies are slower circuits and bigger component sizes. The basic steps for realising a component with each one of the processes is described in Figure 2.1.1 as described in [8]. It is evident that steps in additive technologies are less and simpler.

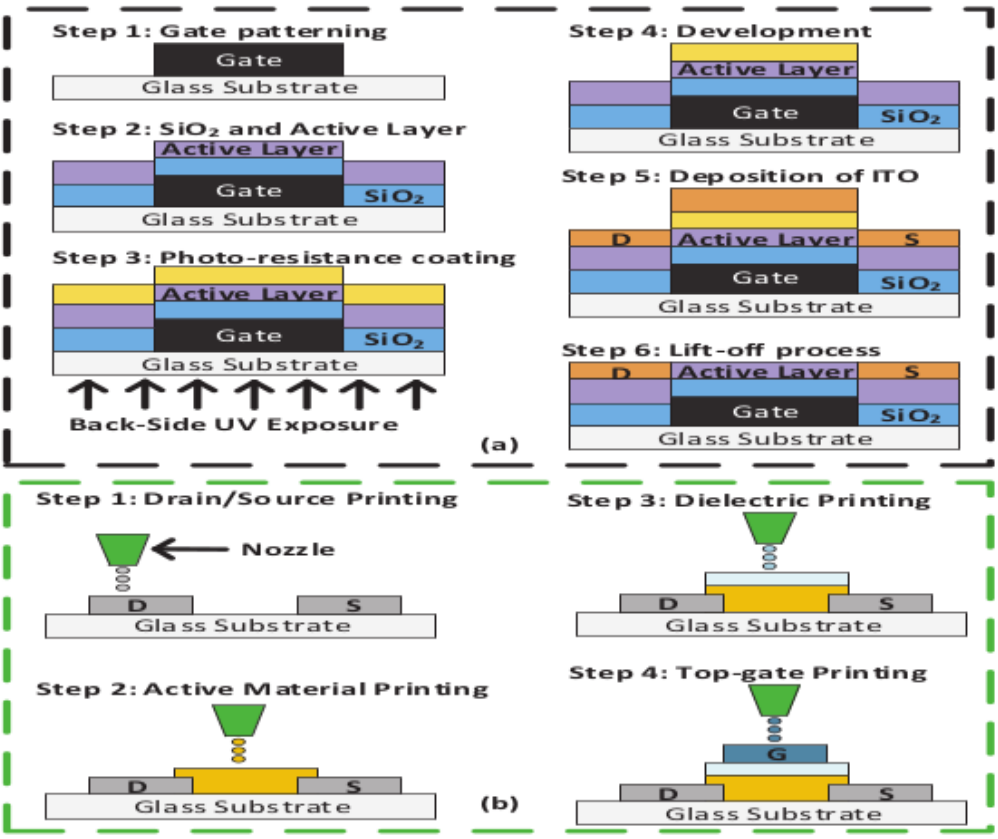

Figure 2.1.1: Steps required in Subtractive (a) and Additive (b) printing process

Printing functional materials typically includes conductive inks, dielectric inks, and semiconductor inks, among others, which are formulated to exhibit desired electrical and mechanical properties. This inherent flexibility in manufacturing processes enables rapid prototyping and

customization of electronic devices, facilitating iterative design cycles and reducing time-to-market for new products. Because of the facts mentioned, printed electronics appear to be more cost-effective and scalable in production compared to silicon-based circuits.

The versatility of printed electronics allows for the fabrication of lightweight, flexible, and even stretchable electronic devices, enabling applications in wearable electronics, biomedical sensors [18],[20], [34],[25], and food and agriculture [17], [35], among others . They are suitable for a wide variety of disposable and ultra-low cost applications since printed electronics can be manufactured using comparatively cheap printing equipment and materials on flexible substrates such as plastic or paper using non-toxic materials. The flexibility, and design versatility of printed electronics make them an attractive alternative to traditional silicon-based circuits for a variety of emerging applications in electronics, sensing, and beyond[14]. [10] also present a set of candidate sensors and applications for similar use cases with programmable wearable hardware in Figure 2.1.2, but by taking into account the disposable nature of printed computing, the range of uses becomes even greater.

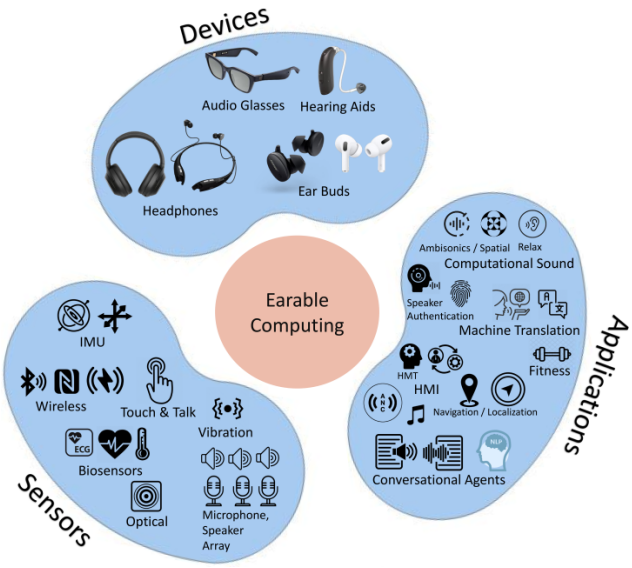

Figure 2.1.2: Capabilities and Applications of Earable and Printed Computing

Despite their numerous advantages, printed electronics also exhibit several drawbacks compared to silicon-based circuits. One significant limitation is the relatively lower performance and functionality of printed electronic devices, particularly in terms of speed, power consumption, and integration density. Silicon-based circuits, benefiting from mature semiconductor fabrication processes, offer superior performance characteristics, enabling high-speed operation and complex functionality suitable for a wide range of applications, including high-performance computing and advanced electronics. In contrast, printed electronics often suffer from higher resistivity and limited material choices, which can restrict their performance and functionality, especially in demanding applications requiring high-frequency operation or analog signal processing. The stability and reliability of printed electronic devices may pose challenges, as the properties of printed materials can be sensitive to environmental factors such as temperature, humidity, and mechanical stress, potentially leading to degradation or failure over time. The low cost manufacturing techniques for printed electronics produce

circuits with large feature sizes that cause area and power considerations. These can usually cause implementation issues with many applications, especially battery powered, and need to be handled by the hardware designers. Achieving consistent performance in printed electronics can be challenging due to variations in printing processes, ink formulations, and substrate properties, requiring careful optimization and quality control measures to ensure reliability and consistency.

2.1.2 EGFET

Electrolyte-gated field-effect transistor (EGFET) technology represents a promising advancement in the field of flexible electronics, offering novel means for achieving electronic devices with low operating voltages and enhanced biocompatibility [24]. EGFET presents important characteristics compared to other printed technologies such as CNT-TFT, namely high mobility and low supply voltage, making it suitable for the targeted applications mentioned [22], [29]. Unlike conventional FETs, which typically employ solid dielectrics as gate insulators, electrolyte-gated FETs utilize an electrolyte solution as the gate insulator, enabling dynamic control of the device properties through ion migration and electrochemical processes. Electrolyte-gated FETs exhibit high ionic conductivity, making them particularly suitable for integration with biological systems, such as bioelectronic devices and biosensors, where direct interfacing with biological fluids and tissues is required. EGFET technology holds great promise for advancing the field of flexible and bioelectronic devices, offering opportunities for innovative applications, mentioned in 2.1.1. Figure 2.1.3 is the design and actual printed hardware of an EGFET component, as printed in [8].

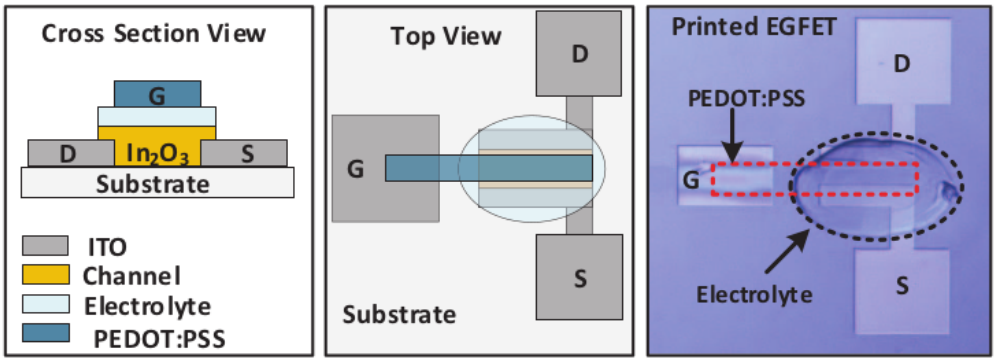

Figure 2.1.3: Cross Section View, Top View and Printed EGFET

In addition to that, [8] presents a memory design and library for printed microprocessors that implements ROM and RAM architectures. ROM memories appear to be faster, smaller and more energy efficient than RAM memories and so, designs with less RAM are usually more preferential. Figure 2.1.4 displays the architecture for single and multi bit ROM, using printed conductive material and sensing resistors to read the stored value.

These multi-bit ROMs allow for storing multiple values with use of a single conductive patch, thus increasing the memory density. Different geometries of the conductive material can output different conductivity levels and encode more than a binary value. This comes at the cost of an additional ADC required to read the analog voltage levels.

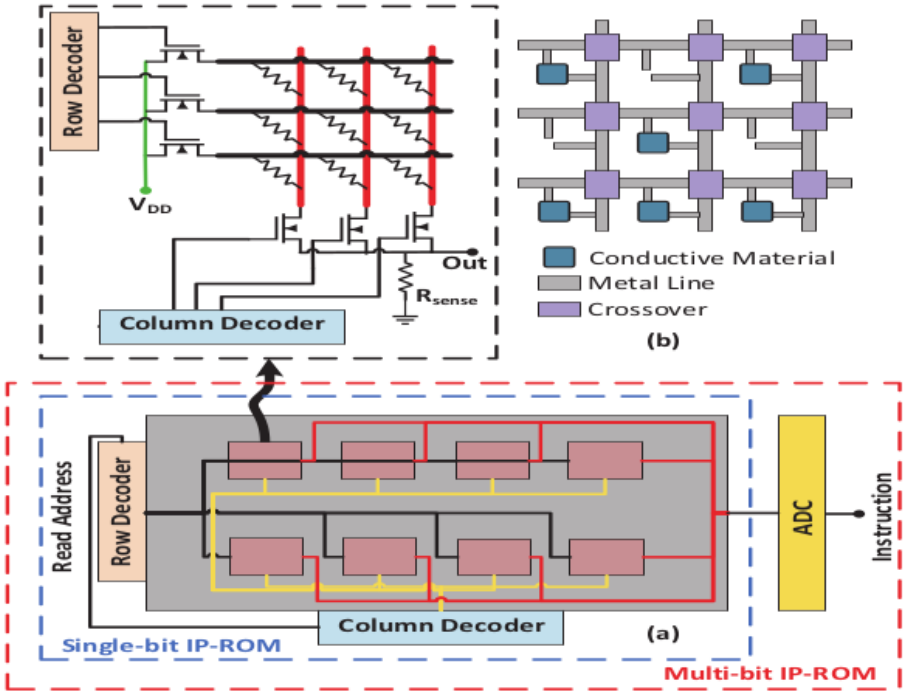

Figure 2.1.4: Example Image of egfet mem 0.8

2.2 EDA Tools

Electronics Design Automation tools represent a crucial component of electronic circuit design workflows, facilitating the creation, simulation, and optimization of intricate hardware systems. These software applications encompass a broad spectrum of functionalities, including netlist generation, simulation, and analysis. EDA tools streamline the design process by offering intuitive interfaces and powerful algorithms that expedite the exploration of various design alternatives and mitigate the need for extensive manual iterations. Moreover, simulation capabilities integrated into EDA tools empower designers to assess circuit performance under diverse operating conditions, predicting phenomena such as power consumption with remarkable accuracy.

In addition to facilitating circuit design, EDA tools play a pivotal role in quantifying hardware specifications and evaluating the performance metrics of electronic systems. Through sophisticated analysis modules, these tools enable engineers to conduct detailed characterization studies, examining crucial parameters such as signal propagation delays and power dissipation. By leveraging EDA tools for hardware measurement, designers can ascertain the efficacy of their designs and identify potential bottlenecks or design flaws early in the development cycle. Furthermore, these tools facilitate the generation of comprehensive reports and visualizations, providing actionable insights into the behavior and performance of their circuits. With the integration of simulation and measurement capabilities, EDA tools offer a unified platform for the design, analysis, and validation of electronic systems, allowing for designing efficiently in the rapidly evolving landscape of hardware design

2.2.1 Standard Cell Libraries

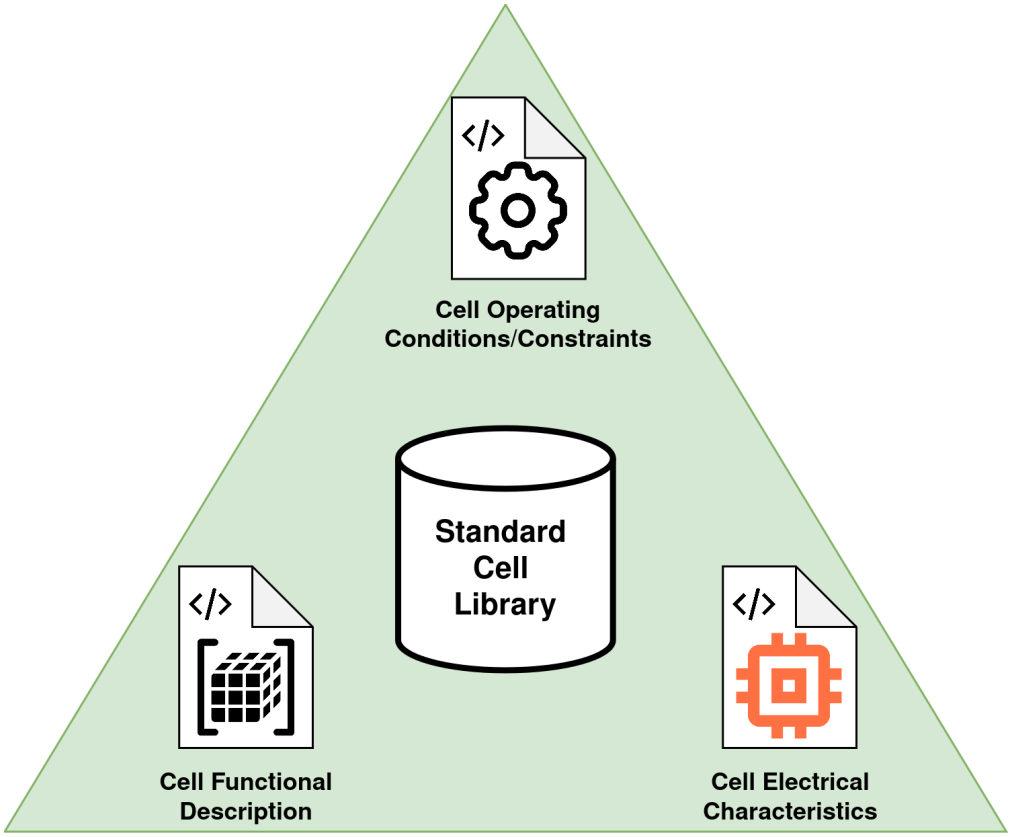

Figure 2.2.1: Basic components of Standard Cell Library

Standard cell libraries play a pivotal role in the synthesis process within the domain of electronic design automation (EDA). These libraries consist of pre-characterized and pre-optimized logic cells, each representing a specific combination of logic functionality and physical implementation. Standard cell libraries provide a comprehensive set of building blocks that make up essential components such as basic gates, flip-flops, and other complex functional units. Within the EDA environment, these libraries serve as fundamental resources for the translation of high-level hardware descriptions into gate-level representations. By utilizing standard cell libraries, EDA tools can efficiently map abstract descriptions of electronic circuits, expressed in hardware description languages like Verilog or VHDL, onto the available set of logic cells, ensuring compatibility with the target technology and meeting design constraints such as timing, area, and power consumption. Standard cell libraries facilitate design exploration and optimization by offering a diverse selection of cells with varying characteristics, enabling designers to make decisions regarding trade-offs between different design metrics. For this thesis we used a printed electronics standard cell library to synthesize our designs and measure specifications.

2.2.2 Synopsys Design Compiler

One prominent EDA tool widely used in the electronics industry is Synopsys Design Compiler. Design Compiler is a synthesis tool that translates high-level hardware descriptions, typically

specified in hardware description languages, into gate-level representations (netlists) suitable for implementation on specific target technologies. In our case the target technology is the EGFET library, that provides standard cells for printed electronics. Design Compiler employs advanced algorithms and optimization techniques to generate optimized netlists, aiming to meet design constraints such as timing and area. Additionally, the tool offers features for design exploration, allowing engineers to explore trade-offs between different design metrics and refine their designs. With its comprehensive synthesis capabilities, Design Compiler significantly streamlines the design process, enabling efficient realization of complex electronic systems while meeting the imposed performance requirements.

2.2.3 Synopsys Power Compiler

Synopsys Power Compiler is an advanced power optimization tool integrated within the Synopsys Design Compiler suite, tailored specifically to address the escalating demand for low-power electronic systems. It functions as a comprehensive solution for power-centric design, offering a range of features aimed at measuring both dynamic and static power consumption in electronic circuits. Leveraging sophisticated algorithms Power Compiler enables engineers to systematically analyze, optimize, and manage power consumption throughout the design process. Furthermore, it provides extensive power analysis capabilities, displaying power profiles and exposing potential optimization opportunities. Power Compiler aids monitoring and achieving reductions in power consumption while preserving design integrity and meeting performance requirements, thus facilitating the development of energy-efficient electronic systems crucial for the Printed applications we tackle.

2.3 Machine Learning Applications

2.3.1 Machine Learning

Machine Learning is a branch of artificial intelligence that focuses on the development of algorithms and statistical models that enable computers to learn from and make predictions or decisions based on data, without being explicitly programmed for every task. The overarching goal of machine learning is to enable systems to automatically improve their performance on a given task through experience or exposure to data. This is typically achieved by identifying patterns and relationships within the data, which are then used to make predictions or decisions on new, unseen data. Machine learning algorithms can be broadly categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning. Machine Learning applications are usually computation intensive mostly consisting of Multiply-Accumulate and Comparison operations especially when targeting MLPs, SMVs or DTs

2.3.2 Training and Inference

Training and Inference are two fundamental processes that occur during the lifecycle of a model.

Training

Training refers to the phase where the model learns from a dataset, typically consisting of input-output pairs. During training, the model adjusts its internal parameters, such as weights and biases in neural networks, based on the provided data in order to minimize a predefined loss function. This process involves iteratively feeding the training data through the model, computing the output, comparing it to the true labels, and updating the parameters using algorithms such as back propagation described in 2.3.1. Back propagation works backwards from the output layer of the network, tuning the weights of neurons inside it with the goal of minimizing loss 2.3.1.

$$\frac{\partial E}{\partial w_{ij}} = \delta_j y_i \quad \text{where} \quad \delta_j = \frac{\partial E}{\partial z_j} \cdot \sigma'(net_j) \tag{2.3.1}$$

The goal of training is to optimize the model's performance on the training data, enabling it to generalize well to unseen data, while avoiding pitfalls such as over fitting.

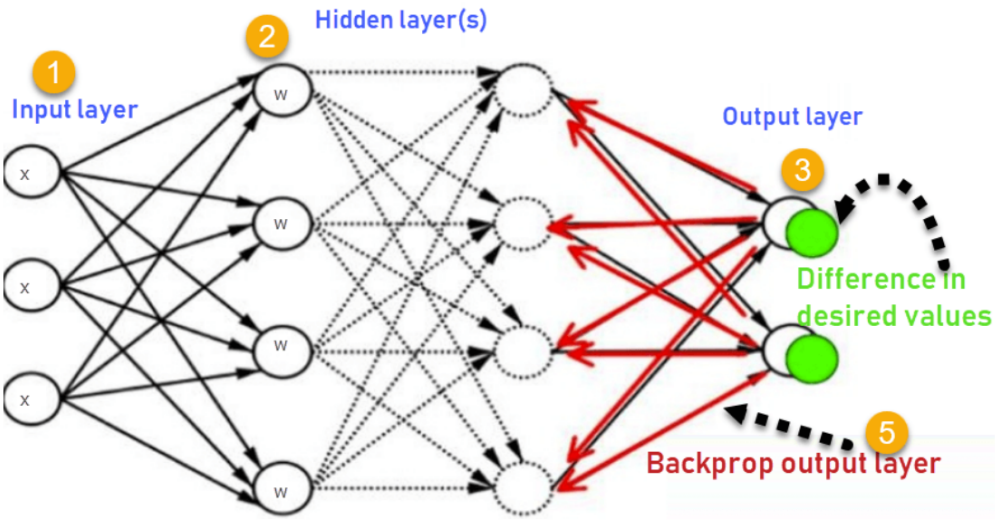

Figure 2.3.1: Back Propagation Basis

Inference

Inference on the other hand, occurs after the model has been trained and involves using the learned parameters to make predictions or decisions on new, unseen data. During inference, the trained model takes input data and processes it to produce an output prediction or decision, without further adjusting its parameters. Inference is the phase where the model's effectiveness and generalization capability are put to the test, as it encounters data that it did not see during training. The performance of the model during inference is evaluated based on various metrics, such as accuracy, precision, recall, or mean squared error, depending on the specific task the model is designed for. Overall, training and inference are complementary processes that together enable machine learning models to learn from data and make useful predictions or decisions. Our work focuses and is evaluated on the inference

of several machine learning models.

2.3.3 Classification and Regression

Classification and Regression are two of the most popular tasks in machine learning, and the ones that we examined.

Classification

Classification involves predicting the class or category of an input based on its features. In classification, the output variable is categorical, meaning it belongs to a finite set of predefined classes. Common examples include spam email detection, image recognition, and sentiment analysis.

Regression

In regression the output variable is numerical, allowing for a wide range of possible values. Typical applications of regression include predicting house prices, stock prices, and temperature forecasting.

Metrics

Both classification and regression involve training a model on labeled data, where the model learns the underlying patterns and relationships between the input features and the output variable. The performance of classification and regression models is evaluated using metrics such as accuracy, precision, recall, F1-score (for classification), and mean squared error, mean absolute error (for regression), among others, to assess how well the model generalizes to new, unseen data. Both classification and regression models were used in the experimental section.

2.3.4 Examined models

1. Multi Layer Perceptrons (MLPs)

   Multi Layer Perceptrons (MLPs) are a class of artificial neural networks widely used in machine learning and pattern recognition tasks. They consist of multiple layers of interconnected nodes, each layer comprising one or more neurons or units. The architecture typically consists of an input layer, one or more hidden layers, and an output layer. In our case, we used mostly models with one or two hidden layers, for a few different applications. In MLPs, information flows forward from the input layer through the hidden layers to the output layer, with each layer performing a series of computations to transform the input data into meaningful output predictions. The key feature of MLPs lies in their ability to learn complex non-linear relationships within the data through backpropagation that occurs during training, where the network adjusts its internal parameters (weights and biases) based on the difference between predicted

and actual outputs.

Regarding the computations needed for an MLP, firstly the input data is fed into the network through the input layer, where each node represents a feature or attribute of the input. Then, the weighted sum of inputs is calculated at each neuron in the subsequent hidden layers, followed by the application of an activation function to introduce non-linearity and enable the network to model complex relationships. This process is repeated for each layer, with the outputs of one layer serving as inputs to the next. The following is the calculation of a single neuron within an MLP, where W is the weight matrix, x in the input matrix, b is the bias of the neuron and f is the activation function which can be relu, step function etc.

$$\mathbf{y = f(WxT + b)} \tag{2.3.2}$$

The output layer produces the network's prediction based on the transformed representations learned through the hidden layers. It is evident that the prevailing computation when calculating the output of the MLP is the weighted sums, including multiplications and additions (MAC). These computations appear many times, since they are required for all neurons, for all layers. Accelerators for such applications usually contain many MAC units in order to speed up computation by calculation MAC oprations fast and in parallel. Finally, throughout training the network's parameters are iteratively adjusted using optimization algorithms, where the error between predicted and actual outputs is minimized by updating the weights and biases in a direction that reduces the loss function.

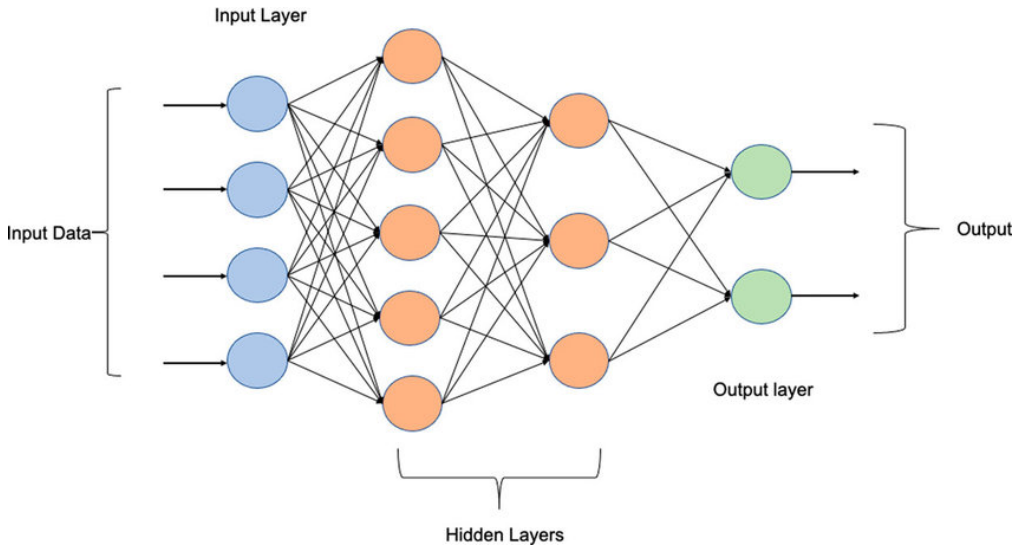

Figure 2.3.2: Simple MLP architecture where Circles are the Neurons of the Model [1]

2. Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are supervised learning models used for classification and regression tasks. At their core, SVMs aim to find the optimal hyperplane that separates different classes or groups in the input data space. This hyperplane is determined by maximizing the margin, which represents the distance between the hyperplane and the nearest data points of each class, known as support vectors. The key idea behind SVMs is to transform the input data into a higher-dimensional feature space where classes are more easily separable, typically achieved through the use of kernel functions. These functions compute the dot product between pairs of data points in the higher-dimensional space without explicitly mapping the data points into that space, thus avoiding the computational burden associated with working in high dimensions. SVMs then seek to find the hyperplane with the largest margin in this transformed space, effectively minimizing the classification error and improving generalization performance.

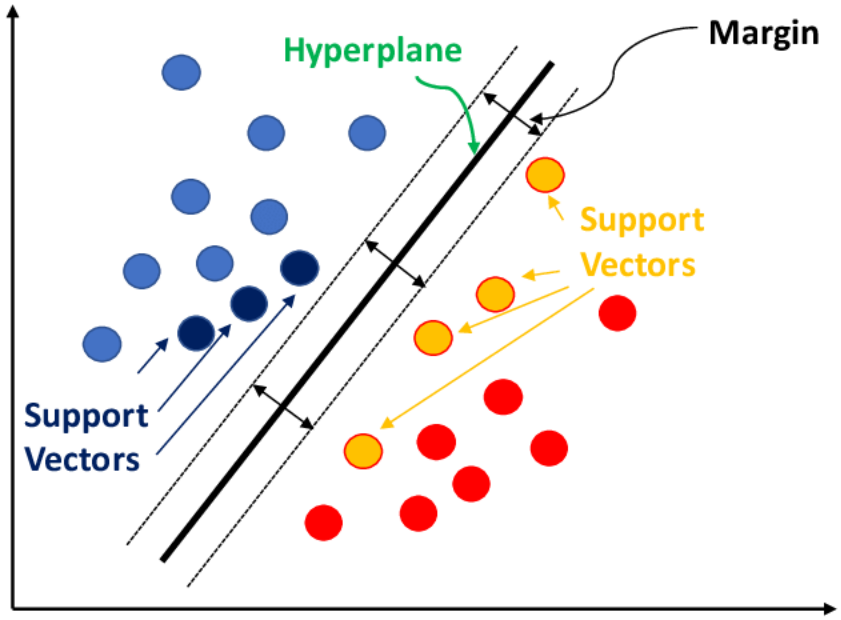

Figure 2.3.3: Visualization of SVM Classifier [23]

The process for computing SVMs involves several steps. In the beginning the input data is transformed into the higher-dimensional feature space using a kernel function, such as the radial basis function or polynomial kernel. Then, the optimal hyperplane that maximizes the margin between classes is determined by solving a constrained optimization problem, often using techniques such as quadratic programming or Lagrange multipliers. This optimization process involves finding the support vectors, which are the data points that lie closest to the decision boundary and thus have non-zero Lagrange multipliers. Once the optimal hyperplane is identified, new data points can be classified by evaluating their position relative to the hyperplane in the transformed feature space. Even though it may seem different from MLPs, SVMs have quite similar computation patterns to MLPs. Again, multiplication and addition of inputs, biases

and weights takes over as the biggest percentage of the calculations, even more so than the MLPs since there are no activation functions to be applied at the end of each node. Below is the calculation required for a hyperplane, where w is the weight matrix, x the input matrix and b the bias:

$$\mathbf{wTx} + \mathbf{b} \tag{2.3.3}$$

In general SVMs offer several advantages, including the ability to handle high-dimensional data, resistance to overfitting, and effectiveness in dealing with non-linear decision boundaries through the use of appropriate kernel functions.

3. Decision Trees

Decision Trees are popular supervised learning models used for both classification and regression tasks. They operate by recursively partitioning the input space into regions, guided by feature values, in order to make predictions. At each internal node of the tree, a decision is made based on the value of a specific feature, leading to one of several possible branches corresponding to different feature values. This process continues until a leaf node is reached, where a final prediction or decision is made. The key advantage of Decision Trees lies in their interpretability and ability to handle both numerical and categorical data. They can capture complex non-linear relationships within the data and are robust to outliers but they are susceptible to overfitting, especially when the trees are deep, which can be mitigated through techniques such as pruning or methods like Random Forests.

The computation required to build Decision Trees involves recursively partitioning the feature space based on the selected features. This process begins with the entire dataset at the root node, where the algorithm evaluates different splitting criteria, such as Gini impurity or entropy, to determine the best feature and value to split the data. This splitting criterion aims to maximize the homogeneity of the target variable within each resulting partition. Once a split is determined, the dataset is divided into subsets, and the process is repeated recursively for each subset until certain stopping criteria are met, such as reaching a maximum tree depth or having a minimum number of samples in each leaf node. Finally, the tree is pruned to prevent overfitting and improve generalization performance. During prediction, new data points traverse the tree, following the path defined by their feature values, until they reach a leaf node, where the predicted output is assigned based on the majority class or mean value of the training samples in that leaf node. In the processor level, Decision Trees are almost exclusively comprised of comparison operations.

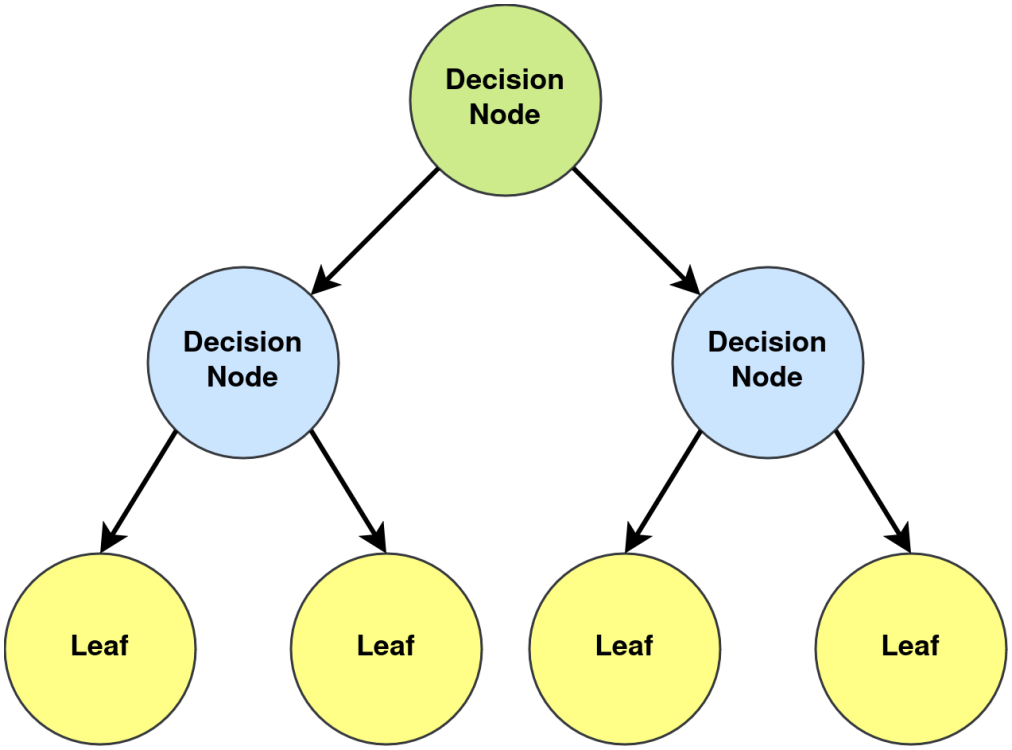

Figure 2.3.4: Binary DT schematic

2.4 Precision Scaling

Precision scaling is one of many approximation techniques [4], that involves adjusting the precision of numerical representations to strike a balance between computational efficiency and accuracy. In various fields such as digital signal processing, and mainly machine learning, where accuracy can be considered as another knob and not strictly required at 100% like most digital computation, precision scaling is usually employed to handle the tradeoff between precision and computational resources. By reducing the number of significant bits used in calculations, precision scaling techniques aim to optimize performance and memory usage while sacrificing as little accuracy as possible. However, the process of scaling precision inherently introduces approximation errors, as it involves truncating or rounding values. These errors can propagate through subsequent calculations, potentially impacting the overall accuracy of computational results. Thus, such changes are often paired with respective analysis on total error introduced to a system or machine learning model, as well as with overall gains in area, timing and power, since the hardware used for applications with scaled precision are simpler and consume less energy since much less computing components are required to accommodate the reduced precision.

The accuracy tradeoff inherent in precision scaling usually requires an understanding of the specific requirements and constraints of the computational task at hand. Depending on the task, there may be much or little room for accuracy loss. Consequently, the decision to employ precision scaling involves a careful evaluation of the acceptable level of error tolerance within the context of the application.

It is common in machine learning applications to easily achieve high accuracy for simple, non-critical problems, thus precision scaling can transform that excess accuracy to computational efficiency. Additionally, real-time systems and compression tasks, inherently benefit from reduced precision since they target low computation time, low memory footprint and thus the accuracy becomes a secondary concern. Tasks where precision scaling does not apply as favorably are safety-critical and accurate systems. For example, medical computing, scientific simulations and numerical analysis, mostly tied to cryptography, necessitate high precision and accuracy of result, despite the increased computational costs.

Chapter 3

Bespoke Analysis and Hardware Reduction

This chapter provides the workflow for the extraction of hardware specifications for the cores including area, timing and power with the EDA tools. We perform an analysis on a Printed Application Suite and present our process for measuring hardware utilization regarding our needs and applications. We consider ROM overhead for each application. Finally, we prune non-utilized hardware in the level of components and also in the architectural level, with the information from the previous analysis.

3.1 Examined Processors

The build flow and the application suite were tested with a variety of processors, all relatively small, in order to fit the needs for printed electronics technologies. All of the cores are evaluated on hardware characteristics based on our build-flow 3.2 and 2 of them were used for application simulations.

Zero-riscy

Zero-riscy is a 32 bit 2-stage pipeline RISCV architecture and part of the Pulpino SoC, developed by ETH [28]. The core can implement the RV32IMC ISA and is configurable with the ability to remove extensions. The extensions can configure the core to synthesize with 16 registers in the register file instead of the standard 32, whether or not to include multiplication and division commands using a mult-div unit, and whether to support compressed instructions. For this thesis we configured the core with the reduced register file, no inclusion of compressed instructions and tested both with and without a mult-div instructions support. Despite the minimal configurations, Zero-riscy is the biggest of the examined cores since its the only one with multiple pipeline stages and relatively complex ISA. Figure 3.1.1 showcases a block diagram of the core's components and functionalities.

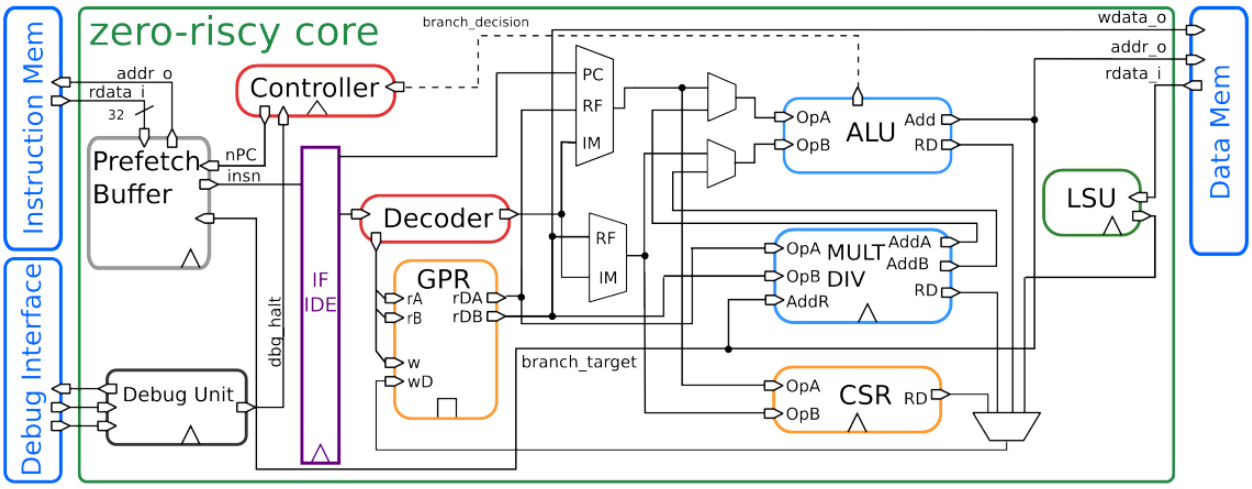

Figure 3.1.1: Zero-riscy core diagram

OMSP430

OpenMSP430[19] is a 16-bit single stage, low-power, register based microcontroller. OMSP430 is also a highly configurable core, with options for including or excluding components integrated in the core's Verilog description files. The configuration files allow for tweaking on all system peripherals like Clock Operation, Interrupts, DMA and Debugger functionality, Memory Sizes, certain Functional blocks removal and ASIC-FPGA specific configurations. We again use minimal configuration with the exception of the multiplier unit which can severely decrease execution times in Multiplication-Heavy applications. Figure 3.1.2 displays an overview of the core's components. Despite the low gate count and

attempts at low power, OMSP430 can still be very large in area when synthesized with EGFET technology as tested in [8].

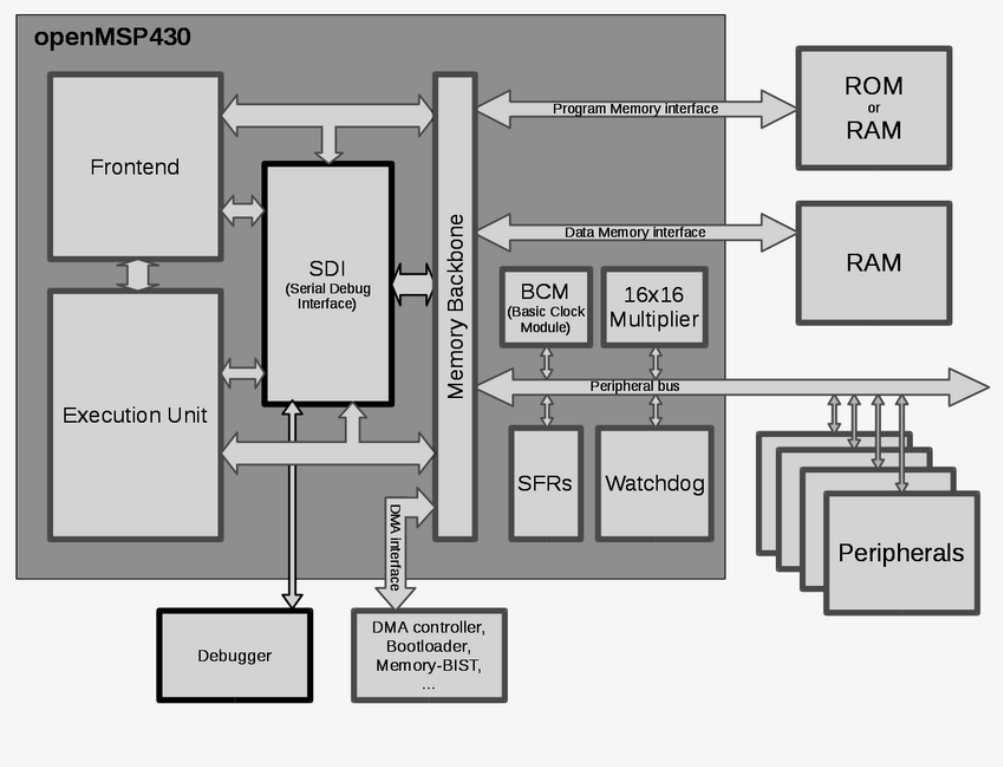

Figure 3.1.2: OpenMSP430 block diagram for core, capabilities and peripherals support [2]

ZPU_Small

ZPU_Small [21] is the small version of the configurable Zpu processor. It utilizes a stack-based 32 bit ISA, that is also very restrained when considering functionality. This aims in minimizing sequential components, and provides a good low resource solution for many of the applications examined. ZPU_Small is the only one of the cores tested that is written in VHDL instead of Verilog. A block diagram of ZPU_Small is shown in 3.1.3

TP-ISA

TP-ISA is an architecture designed in [8] with the intricacies of the EGFET technology and the commonly used applications in mind. It is highly customizable in terms of architectural characteristics such as PC size, BAR number and size, Pipeline Depth and Data Width, so analyses are done also for different core configurations. Figure 3.1.4 displays the complete ISA. It is a compact RISC architecture that minimizes complexity and use of sequential components, in order to improve area and power consumption when targeting the EGFET library.

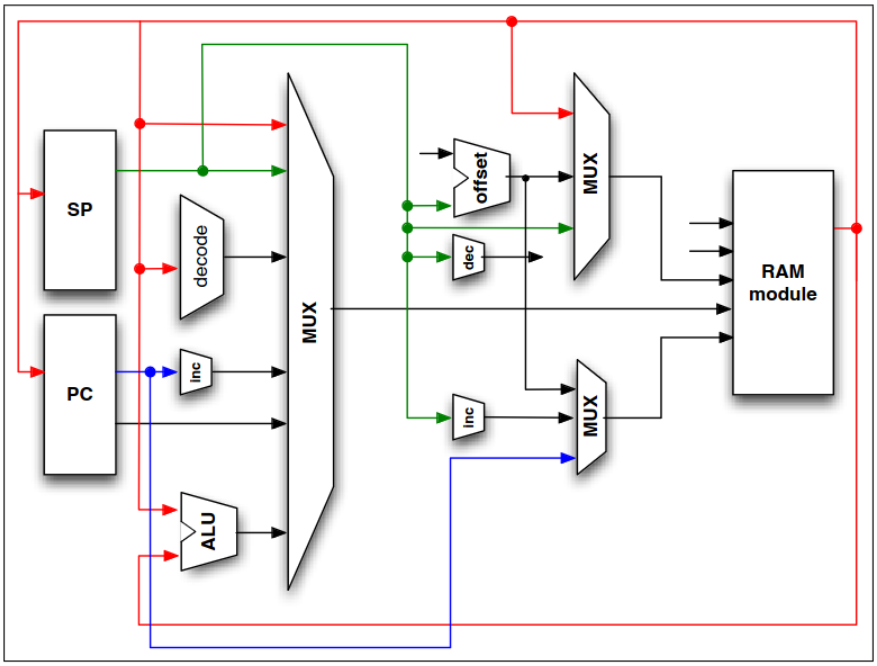

Figure 3.1.3: ZPU block diagram for core [16]

Instruction Format	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M-Type:	opcode				W	C	A	B	R	address1							S	address2						
ADD	OP-ADD				1	0	0	0	R	address1							S	address2						
ADC	OP-ADD				1	1	0	0	R	address1							S	address2						
SUB	OP-ADD				1	0	1	0	R	address1							S	address2						
CMP	OP-ADD				0	0	1	0	R	address1							S	address2						
SBB	OP-ADD				1	1	1	0	R	address1							S	address2						
AND	OP-AND				1	0	0	0	R	address1							S	address2						
TEST	OP-AND				0	0	0	0	R	address1							S	address2						
OR	OP-OR				1	0	0	0	R	address1							S	address2						
XOR	OP-XOR				1	0	0	0	R	address1							S	address2						
NOT	OP-NOT				1	0	0	0	R	address1							S	address2						
RL	OP-RL				1	0	0	0	R	address1							S	address2						
RLC	OP-RL				1	1	0	0	R	address1							S	address2						
RR	OP-RR				1	0	0	0	R	address1							S	address2						
RRC	OP-RR				1	1	0	0	R	address1							S	address2						
RRA	OP-RR				1	0	1	0	R	address1							S	address2						
S-Type:	opcode				W	3'b0			R	address1							immediate							
STORE	OP-STORE				1	0	0	0	R	address1							immediate							
SET-BAR	OP-BAR				0	0	0	0	x	ptr address							immediate							
B-Type:	opcode				4'b0001				R	address1							4'b0				bmask			
BR	OP-BR				0	0	0	1	R	address1							0	0	0	0	bmask			
BRN	OP-BR				0	0	1	1	R	address1							0	0	0	0	bmask			

Figure 3.1.4: The complete ISA of TP-ISA Core

3.2 Processors Build Flow

The work implemented in this thesis consists mostly of measurements of hardware characteristics through EDA tools, cross-compilers targeting the architectures of the cores examined, netlist-rtl simulations of processors and digital design to alter and write new hardware. At the center of the workflow lie the Synopsys Suite and the Modelsim simulator.

HDL files and Synopsys Compiler

Each core consists of a set of Hardware Description files, that the EDA tool will use along with the standard cell library to produce meaningful information about the hardware. Some of the tested cores can contain tens of files, scaling with the complexity of the core and the number of the units within. Many changes were required in the Verilog code of several cores, which is due to version mismatch with the tool, or in cases of unsupported operations when building a netlist from rtl. Many standards may also need to be applied for designs that are intended to run in simulations, such as timing information of units and proper description of tasks that can happen in simulation, like loading a memory with values. Also, in cases where a memory compiler or memory library is missing, memories cannot be built with the base standard cell library, since this leads to prohibitive area and power outputs. We did not compile memories for our experiments since a memory library is not provided for printed technologies.

For measuring hardware characteristics, firstly we import the processors' Verilog Description code in Synopsys EDA. Synopsys functions are called using tcl scripts that are used to tune environment parameters that relate to design constraints and general build guides, manipulate the building process and report on outputs. Furthermore, tcl scripts contain information about the structure of the processor's files, so for more complex cores the process requires more attention. Finally, with the core files, the parameter values, the script that guides the build process and the technology dependent standard cell library, which in our case is printed electronics EGFET, the relevant results are produced.

Cross-Compilers

Since we aim at running simulations of certain applications on the selected cores and strive to extract information from the executions of these, a cross-compiler is essential. Cross-Compilers allow the creation of an executable file based on an application written in high level code (C code in our case), for a specific target architecture, that is different from the machine running the compiler. The cross-compilers for OMSP430 and ZPUSmall, msp430-gcc-toolchain [26] and zpugcc[36] respectively, were build in Ubuntu 22.04 while the cross-compiler for Pulpino, along with the simulation flow was built in a Ubuntu 16.04 Virtual Machine since ri5cy-gnu-toolchain[30] and modelsim caused errors with newer OS versions.

With these compilers we are able to write any application in C code and have the compiler produce executable or memory images, in the case of OMSP430, that can be used to realize an application's memory requirements and to run the applications in simulation. Applications were rewritten for each core, taking into account the specifics of each core and compiler, including architectural differences, like scaling the bitwidths of the applications for cores of

different sizes, and functional differences. For example, the access to OMSP430 multiplier unit is implemented in c code by directly accessing the relevant registers to write inputs and read results. When it comes to the TP-ISA core, there is no compiler available and thus measurements were done mostly for hardware specifications and speedup estimations based on standard assembly code.

Netlist-Rtl Simulators

After having the rtl of the cores tuned to work with the tools and have the executable of our application produced by the cross-compiler we can run the relevant rtl simulations, or continue with the compilation of the rtl and run more accurate netlist simulations. For the Pulpino rtl simulations we mainly used the Modelsim simulator since the rtl description and testbench provided was tested to work directly with Modelsim. There are also makefiles provided that make the total processes of building an app with the cross-compiler, running the simulation and storing the output of the tracer module easier to perform, so this approach was preferred. For OMSP430 we used Synopsys VCS for rtl simulations, so the processes mentioned above were done manually.

A few tests were implemented with netlist simulation as well, yet netlist simulations take much more time to execute compared to the higher level rtl simulations. Since our analysis is focused on the commands executed, cycles required and execution time, the rtl simulation is enough for these purposes, thus we mainly use Modelsim and Synopsys VCS for rtl simulations. We specify that in all cases of simulations the memory was implemented in testbench and not built directly.

Total Work-flow

The total workflow process for this chapter is shown in Figure 3.2.1. The process starts at the Application Building, moving through the Synopsys Flow module that provides hardware measurements and input for Hardware Reduction and Bespoke Analysis through the simulations. The final outputs of the Hardware Reduction and Bespoke Analysis Module are compared to the initial measurements of the first Synopsys Flow. The ROM Analysis Module requires only the output from the cross-compiler to produce ROM Usage info.

Within each module, at the point were results have been produced, we develop a set of python and bash scripts to extract the more specific information that interests us for each of the different analyses that follow. Detailed explanations of the Analysis and Flow Modules is provides in the following sections.

3.3 Cores and Application Suite Analysis

3.3.1 Base Cores Measurements

We first run the cores through the Hardware Analysis flow with Synopsys as described in 3.2. Figure 3.3.1 contains a deeper look of the Synopsys Flow Module which was mainly used for these measurements. The Synopsys Compilers require the modified Verilog description as well as the EGFET Standard Cell Library, and produce our Base Hardware information. The VCS

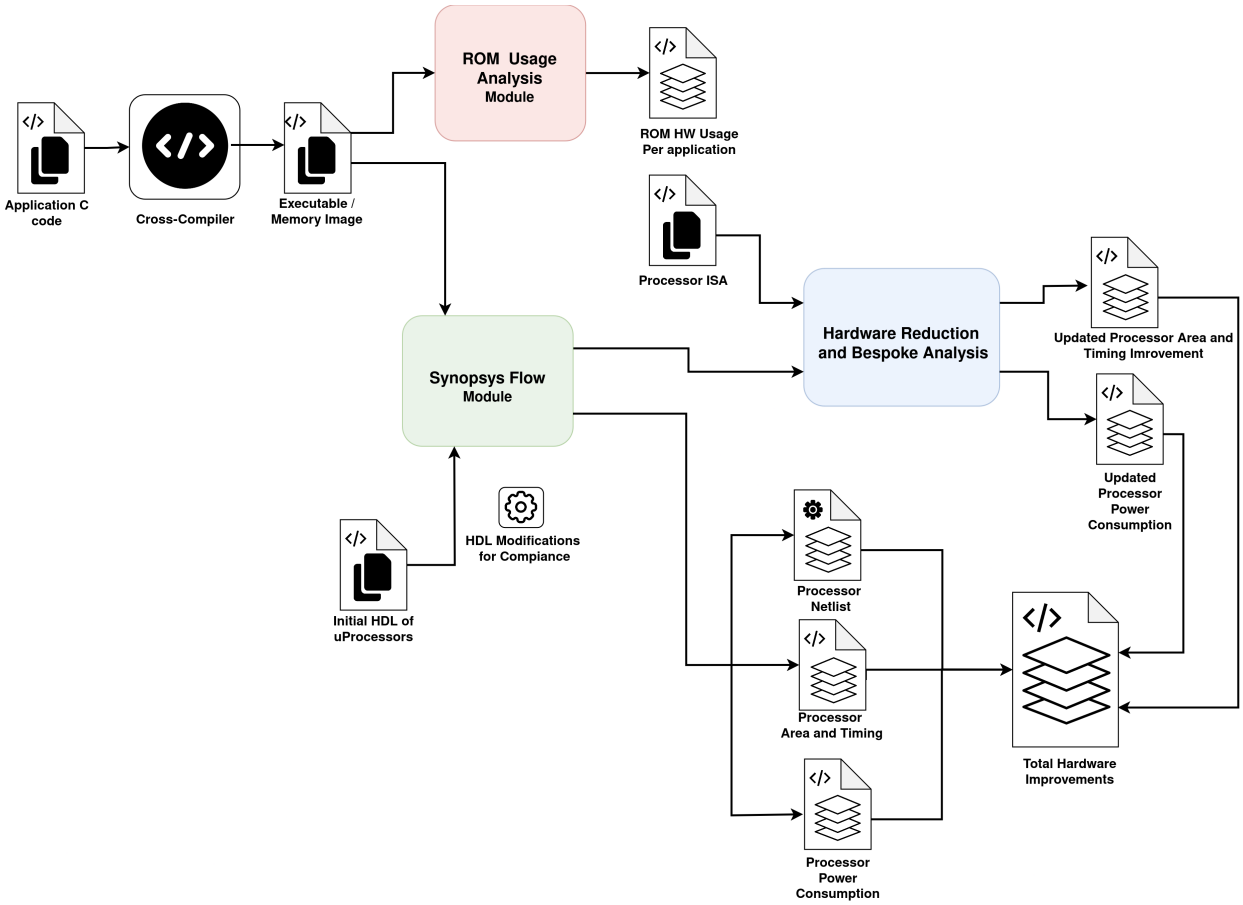

Figure 3.2.1: Total Workflow containing the Suite Analysis, Hardware Analysis and Hardware Modification Setup

Simulator similarly requires the same inputs as the Synopsys Compilers, with the addition of the executable of the application to be run. It produces the cycle accurate execution traces and these outputs are directly fed into the next Module for Reduction Analysis.

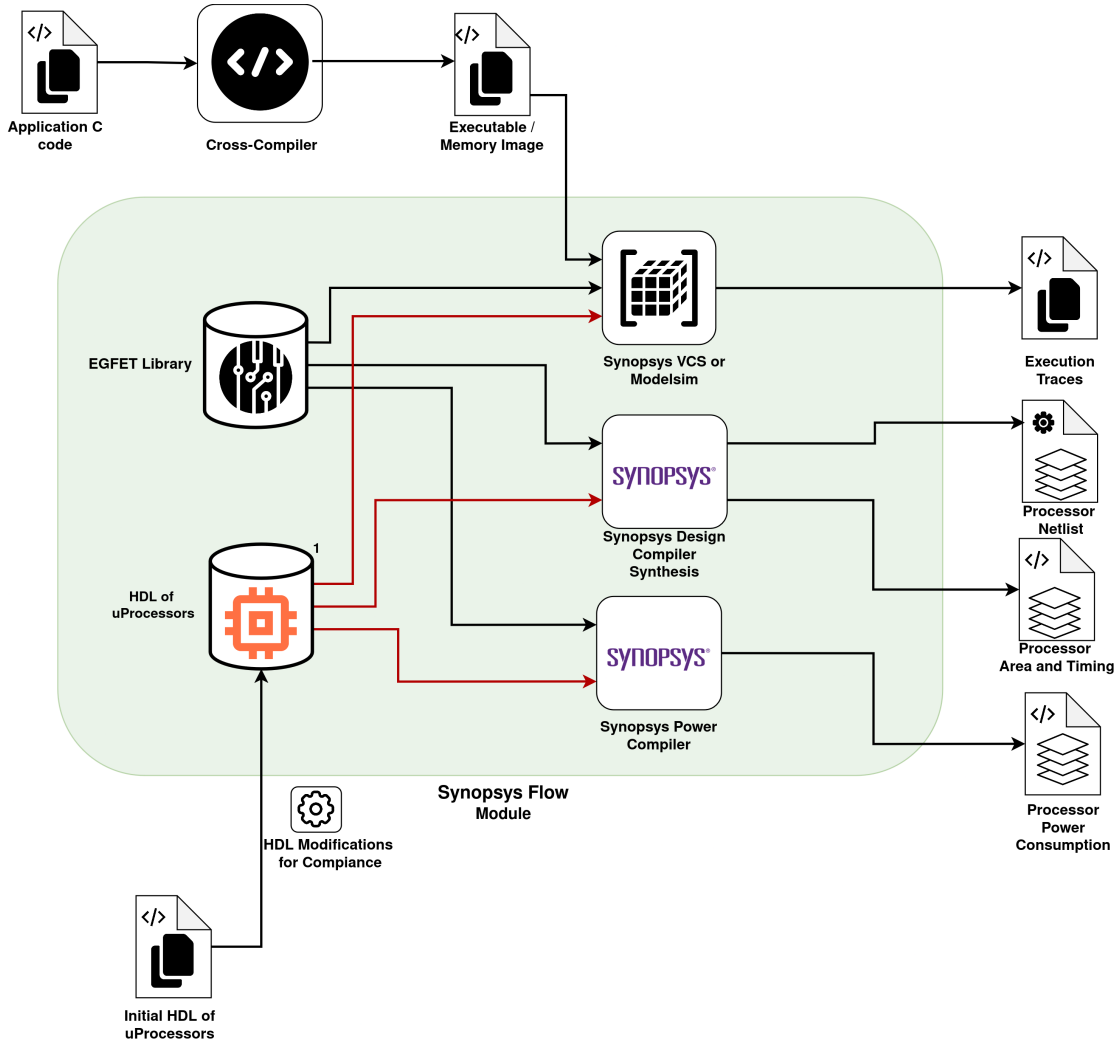

Figure 3.3.1: Detailed view of Synopsys Flow module

Plot 3.3.2 contains the base area and power measurements for Zero-riscy, OMSP430 and ZPU_Small. There are also measurements with and without the multiplier unit for the 2 cores that have Multipliers in their kits. Plot 3.3.3 shows timing details for the synthesized cores, allowing for a view in the possibility for increasing the clock without an area overhead. We observe that the max clock remains the same for designs with and without clock, which indicates that the multiplier unit in both cases is not on the critical path and thus does not restrict the timing period. Also, ZPU_Small appears to have the biggest margin for improvement, being the smallest of the three and able to reach 71Hz clock.

Next we measure the overhead of each of the basic functional units for the Zero-riscy and OMPS430 since they are the bigger cores and the ones that we simulate. The main blocks are Instruction Fetch, Instruction Decode and Controller together and the Execution Unit. Since the Multiplier Unit and Register File may be contained in other units, they are shown

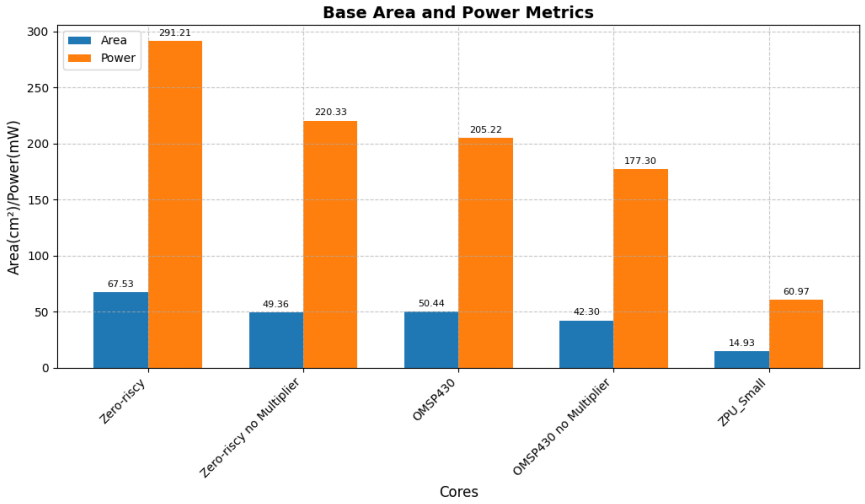

Figure 3.3.2: Area and Power Measurements for the Base Cores used

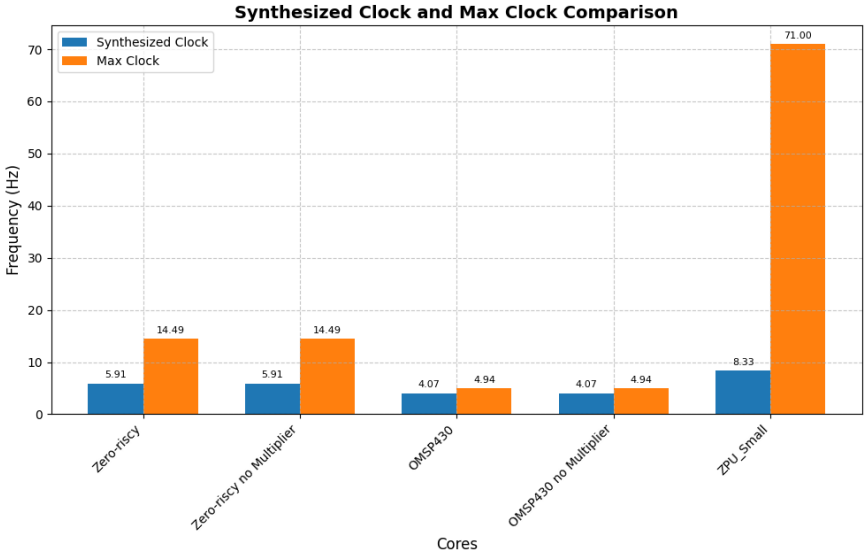

Figure 3.3.3: Timing details for the Base Cores

as separate percentages. Measurements are displayed in Figure 3.3.4 for designs with and without a multiplier.

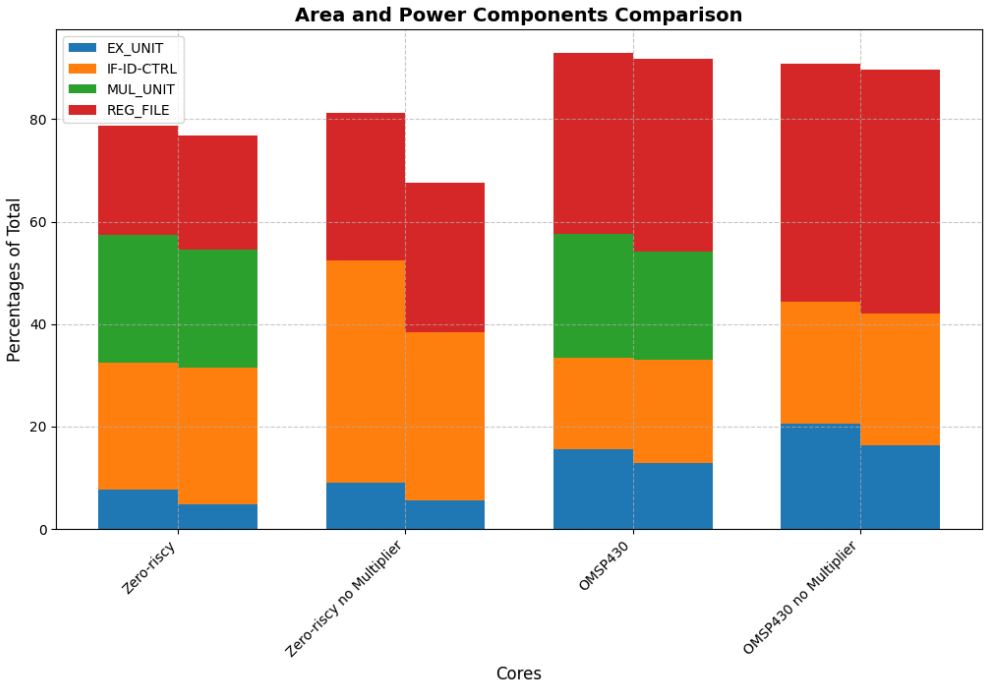

Figure 3.3.4: Component Percentages for Zero-riscy and OMSP430

The 2 smaller cores were not included in this hierarchical percentage analysis of components since their base components are merged to one or two large files, making it hard to distinguish where each logic part belongs. For the cores shown, the remaining percentages up to 100% of utilization are attributed mostly to logic that is not in some of the main components or logic from merged components.

For OMSP430 the Register file consumes more than a third of the area and power resources, while for Zero-riscy it is between 20-30%, even when considering that we used the RV32E architecture, that reduces the register file in half. Even with relatively small Register files, the EGFET technology heavily punishes the use of sequential components. The Multipliers introduce a 20-25% area and power overhead so it is definitely worth examining the tradeoff this with the reduced execution time offered by the base core.

Finally, we explore the hardware characteristics' sensitivity to the TP-ISA different configurations. The knobs that where used to produce the configurations are Datawidth, number of BARs and PC size. Pipeline Depth was also an available knob but since sequential components have a prohibitive cost in the EGFET technology, as shown in [8], we explore only single-stage pipeline configurations. Figure 3.3.5, 3.3.6, 3.3.7, 3.3.8 show the results in comparison.

Area and Power scale almost linearly for all configurations. Changes in DATAWIDTH and PC_WIDTH cause steep changes to area and power while BARs are less effective. Timing is also mostly dependent on DATAWIDTH, degrading rapidly with each doubling. The knobs

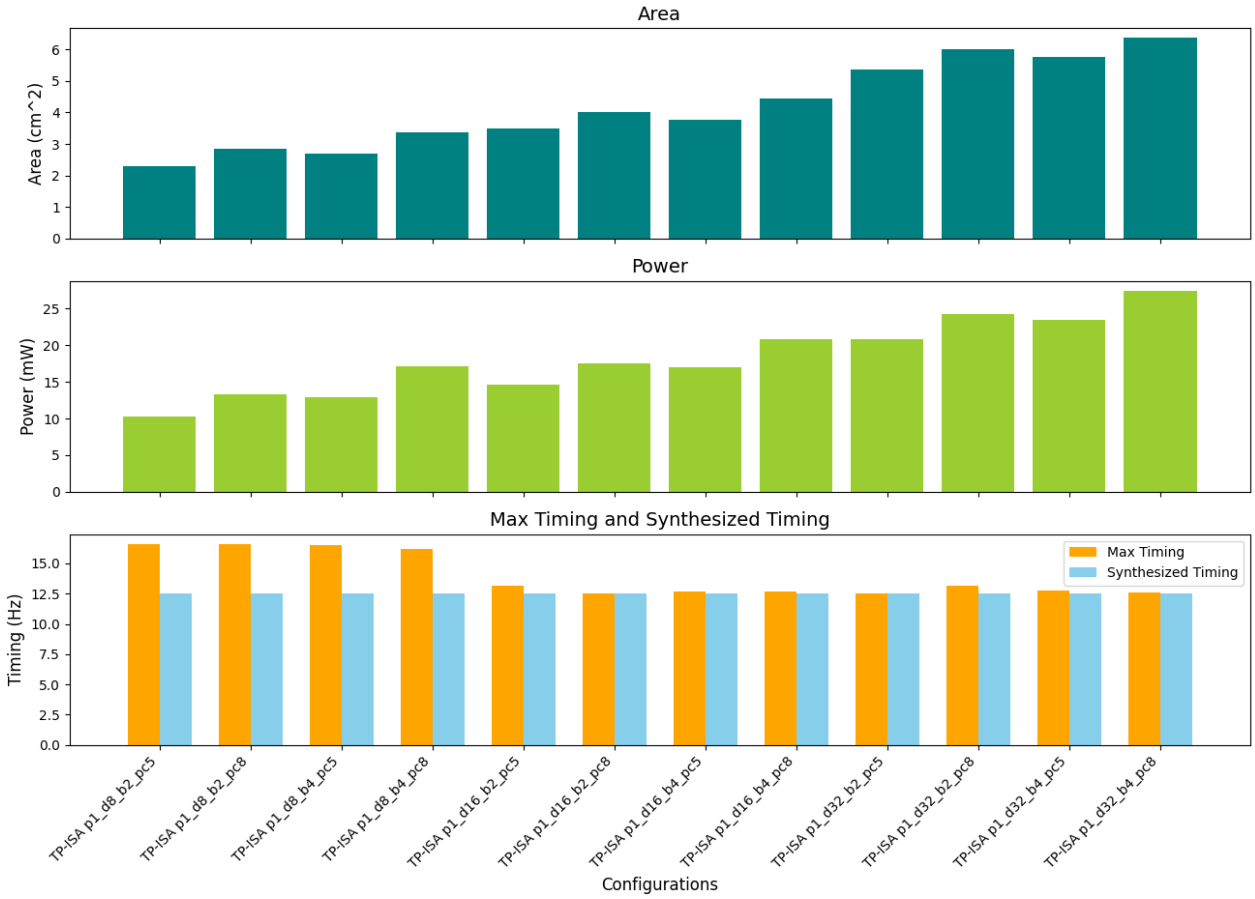

Figure 3.3.5: Area, Power and Timing Characteristics of several TP-ISA configurations

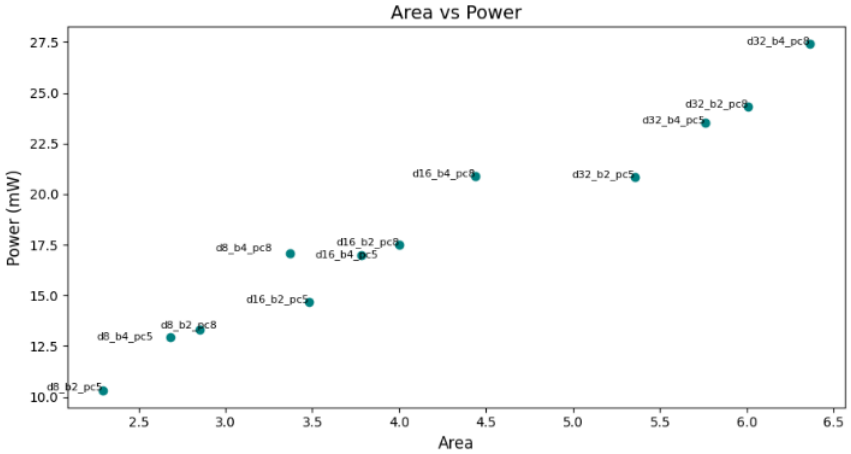

Figure 3.3.6: Plotted TP-ISA base configurations based on metrics from 3.3.5 for Area vs Power

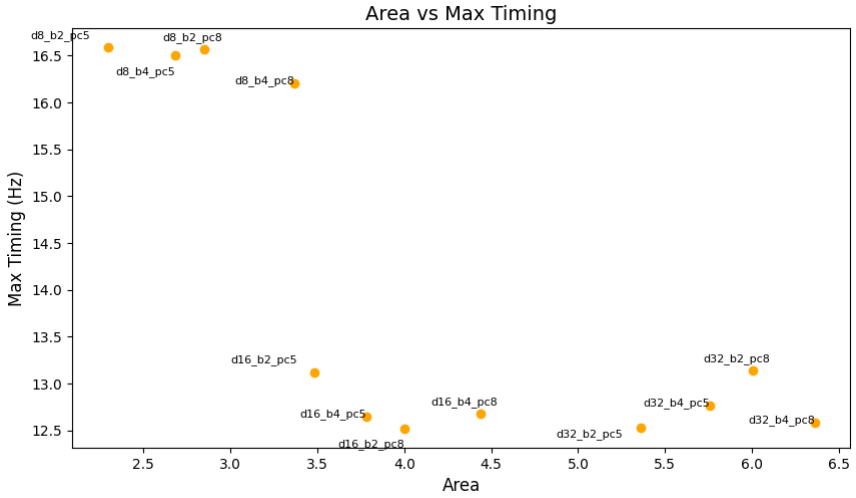

Figure 3.3.7: TP-ISA base configurations for Area vs Max Timing

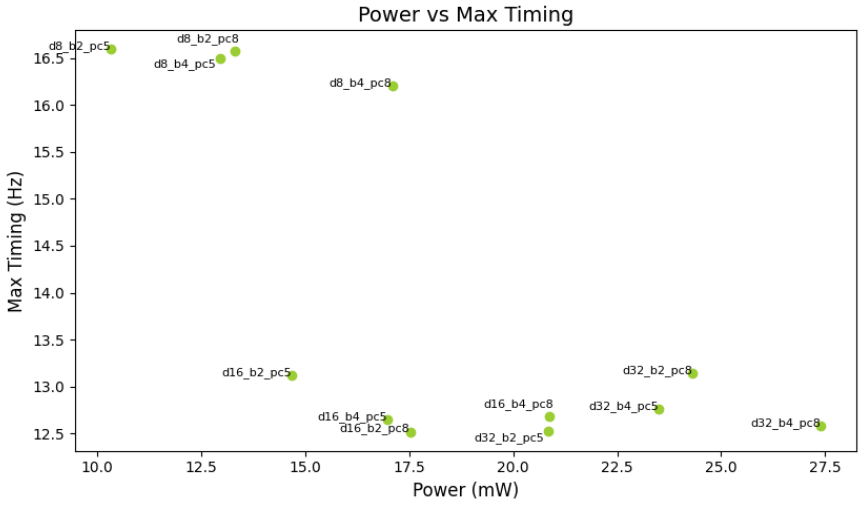

Figure 3.3.8: TP-ISA base configurations for Power vs Max Timing

chosen for the configurations play a massive role in gains, since the smallest of the designs takes up almost **3x** less space, **3x** lower power consumption and has a **24%** faster clock.

3.3.2 TestSuite Analysis

Since the setup for simulations was implemented for Pulpino and OMSP430, we evaluate these cores and workflows with a suite of printed applications also used in [8]. Table 3.1 contains the description for each of the applications. The suite is mostly comprised of common kernels/applications that are standard in printed applications [33], [8] and additionally a small Multi-Layer Perceptron and Decision Tree.

Table 3.1: Benchmark Descriptions

Benchmarks	**Description**
mult	Unsigned integer multiplication
div	Unsigned integer division
inSort	In-place Insertion Sort on array of size 16
intAvg	Signed integer average on array of size 16
crc8	Cyclic Redundancy Code for 16 byte array
tHold	Digital Threshold Detector on array of size 16 with hardcoded threshold
MLP	MLP with 3 Layers of size 4, 10 and 4 with relu activation function, run for 1 Inference
DTree	DTree of Depth 2 with hardcoded compare values, run for 1 Inference

Each of the apps was evaluated with the workflow described in 3.2 for multiplier and standard cores. It is important to evaluate these separately for designs with and without a multiplier since that heavily changes the execution time and assembly code produced for the application. Figure 3.3.9 shows the execution times of the 2 cores for each application.

OMSP430 performs consistently worse than the Zero-riscy core, due to a slower clock. We can also see that for multiplication intensive applications such as the mlp, inclusion of a Multiplier Unit drastically reduces execution times.

3.3.3 ROM Analysis

Using the memory design for printed microprocessors developed in [8] we can calculate the hardware overhead that is inserted with ROM because of program size. The library itself is not available, but the hardware characteristics of each cell are given in the paper. We calculate the overhead of the 1-bit ROM as shown in 3.3.1, where $Core_{DW}$ is each core's DataWidth, $Instr$ is the number of instructions in the program memory for the application, $Cell_{AP}$ is the active power of a 1 bit ROM cell in the EGFET memory library and $Cell_{SP}$ is the static power of a 1 bit ROM cell in the EGFET memory library.

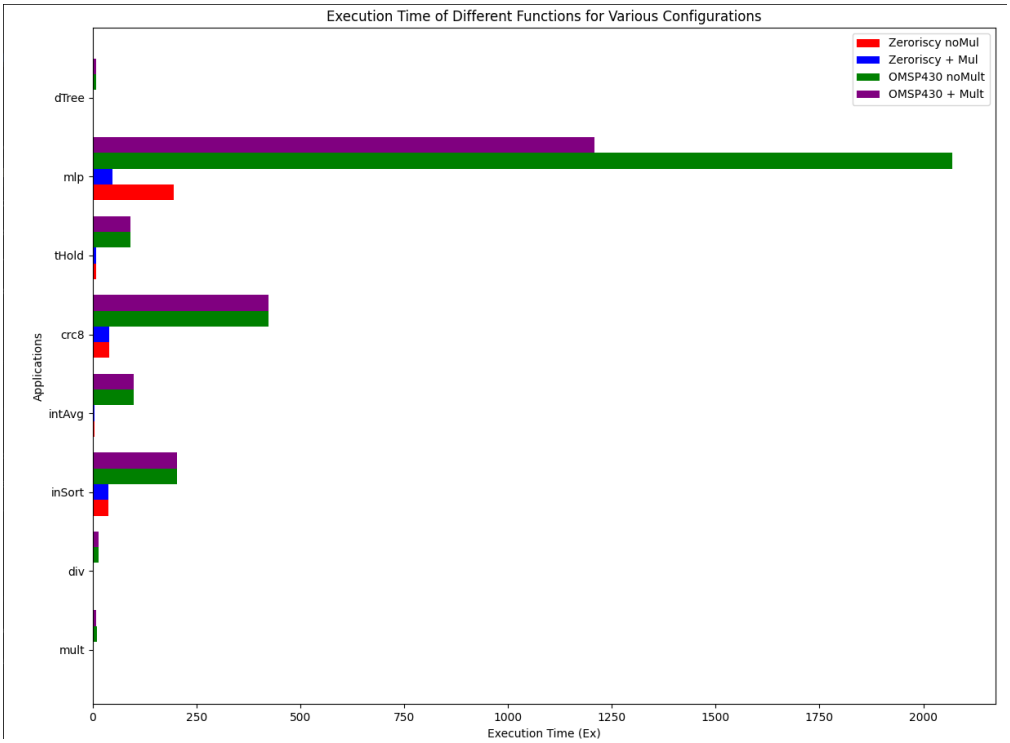

Figure 3.3.9: Execution times of Apps for the tested cores with and without a Mult Unit

$$\begin{aligned} ROM_{Area} &= Core_{DW} * Instr * Cell_{Area} \\ ROM_{Power} &= Core_{DW} * Instr * (Cell_{AP} + Cell_{SP}) \end{aligned} \tag{3.3.1}$$

The analysis of ROM sizes requires only the second stage of the build flow, namely, it requires writing the C code for each processor and the production of the program memory from the cross-compiler. Figure 3.3.10 illustrates this, showing the exact inputs and steps to get to the usage results.

We show measurements for the standard 2 cores and additionally for the ZPU_Small, since we have access to the zpugcc compiler we rewrite the applications for this as well and give the executable as input to the described flow. We also analyse for lack of multiplier units, since this can have a great effect in the program memory, as mentioned in the previous section. This means that we configure the cross-compiler to not use multiplier commands and re-do the flow to compare. Number of instructions per benchmark are shown in Table 3.2. Using 3.3.1 we extract Area overhead and Power consumption of ROM cells for all instances, displayed in Figure 3.3.11. For the more complex benchmarks and especially without a Mult Unit, we see that the ROM overheads can even become comparable to those of the core itself.

Overall, ZPU_Small and Zero-riscy have larger and more power hungry ROMs. Even though the number of instructions can be comparable in many benchmarks, OMSP430 is a 16 bit processor while ZPU_Small and Zero-riscy are 32-bit, thus requiring double the ROM cells per instruction. It is also shown, that benchmarks that do not use mult/div instructions like inSort, intAvg, crc8, tHold and dTree receive no benefits from using a configuration with a Multiplier unit.

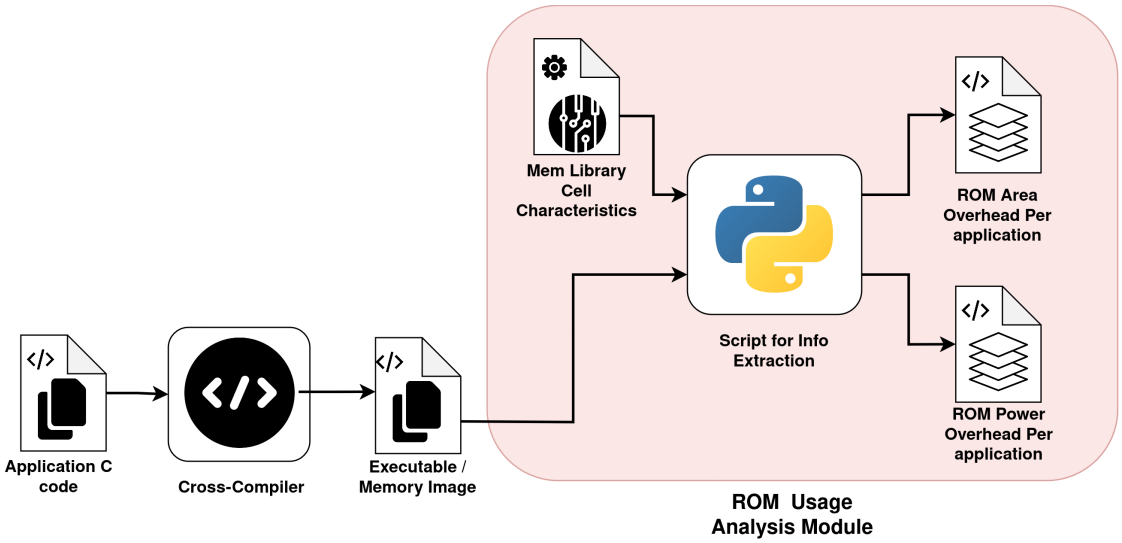

Figure 3.3.10: Detailed view of ROM Analysis module

Table 3.2: Instructions per Benchmark in each Processor

Processor	**Benchmark**							
	mul	**div**	**inSort**	**intAvg**	**crc8**	**tHold**	**mlp**	**dTree**
OMSP430 noMult	48	64	97	51	78	49	310	48
OMSP430	21	64	97	51	78	49	307	48
Zeroriscy noMult	19	56	137	39	573	103	679	29
Zeroriscy	13	13	137	39	573	103	453	29
ZPU_Small	31	31	187	108	178	110	520	88

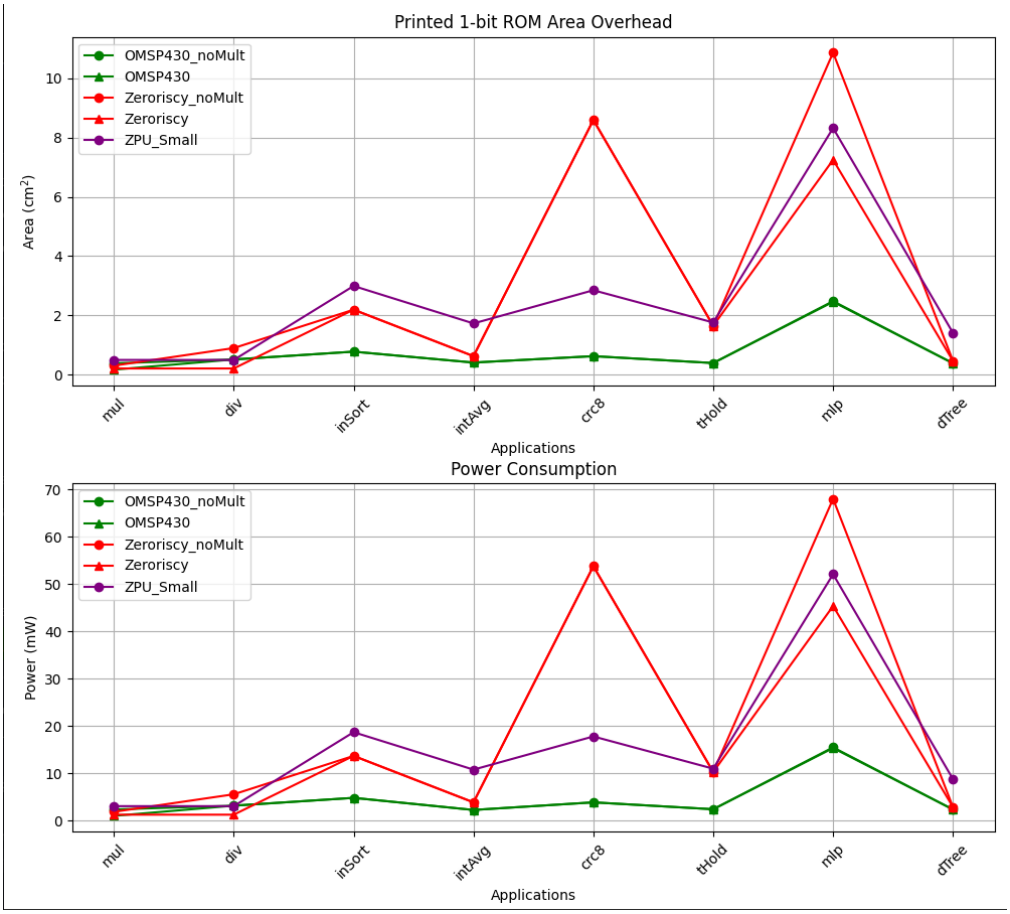

Figure 3.3.11: Area and Power overhead of ROM for each application using Printed Technology 1-bit ROM cells

As seen in Figure 3.3.11 for Zero-riscy, ROM overhead is much improved when extending to RV32M ISA including Multiply operations, both in simple mult and div, but especially for the mlp. This is because the compiler does not need to include the base function for multiplying using simple logical operations and shifts, instead using only the mult/div command. In contrast we can also see that the ROM size for OMSP430 improves only marginally with inclusion of a Multiplier unit like Zero-riscy. We observe that this is due to the msp430-gcc-toolchain unrolling the computations of the mlp neuron for loops. Since the access to the OMSP430 Multiplier happens by accessing specific registers and not with a standard "mul" command, the compiler can decide to instead unroll the code.

3.4 Coarse and Fine Grain Hardware Reduction

In order to begin optimizing the cores by reducing hardware we first need information about the commands executed by the core when running the application suite that we target. Using the build flow described in 3.2 for the Pulpino and OMSP430 cores, we produce both the assembly code and the traces required for extracting the needed information.

For the methods that follow, we can see that each core may receive more significant improvements from a method while the other does not. This, of course, is due to the different modules described in hardware, different architectures and the way each cross-compiler produces assembly code and utilizes the hardware.

The basis for this whole analysis is the Hardware Reduction Module, shown in Figure 3.4.1.

With the outputs of the previous module's traces, we can use each processors ISA to extract information about the utilization of each of the core's components and functionalities. Once these are recognized we modify the verilog descriptions accordingly and feed them back to Synopsys. A big part of this module is the hardware measurements part, also utilized in the Synopsys Flow Module. Finally the outputs of the 2 modules can be compared to produce the improvements offered by the total flow. Moving on, we present each of the removing processes in detail.

3.4.1 Module Removal

Locating and removing hardware at the component level of the core is the first step at receiving significant area and power gains. A module can serve several functionalities and commands of the cores, so we need to make sure that if none are needed in our application suite, we completely remove the module from the design. Some of the basic functionalities such as Fetch-Decode, Arithmetic-Logical operations, Register File and Clock functionality, if they exist as parts of modules, we can immediately conclude that these cannot be removed and so we continue by examining the remaining components.

OMSP430

For the OMSP430 core, the control for included and excluded components is configured via a verilog defines file, making the removal process simple. We firstly conclude that the Frontend, Execution Unit and Basic Clock Module components are needed for the operation of the core,

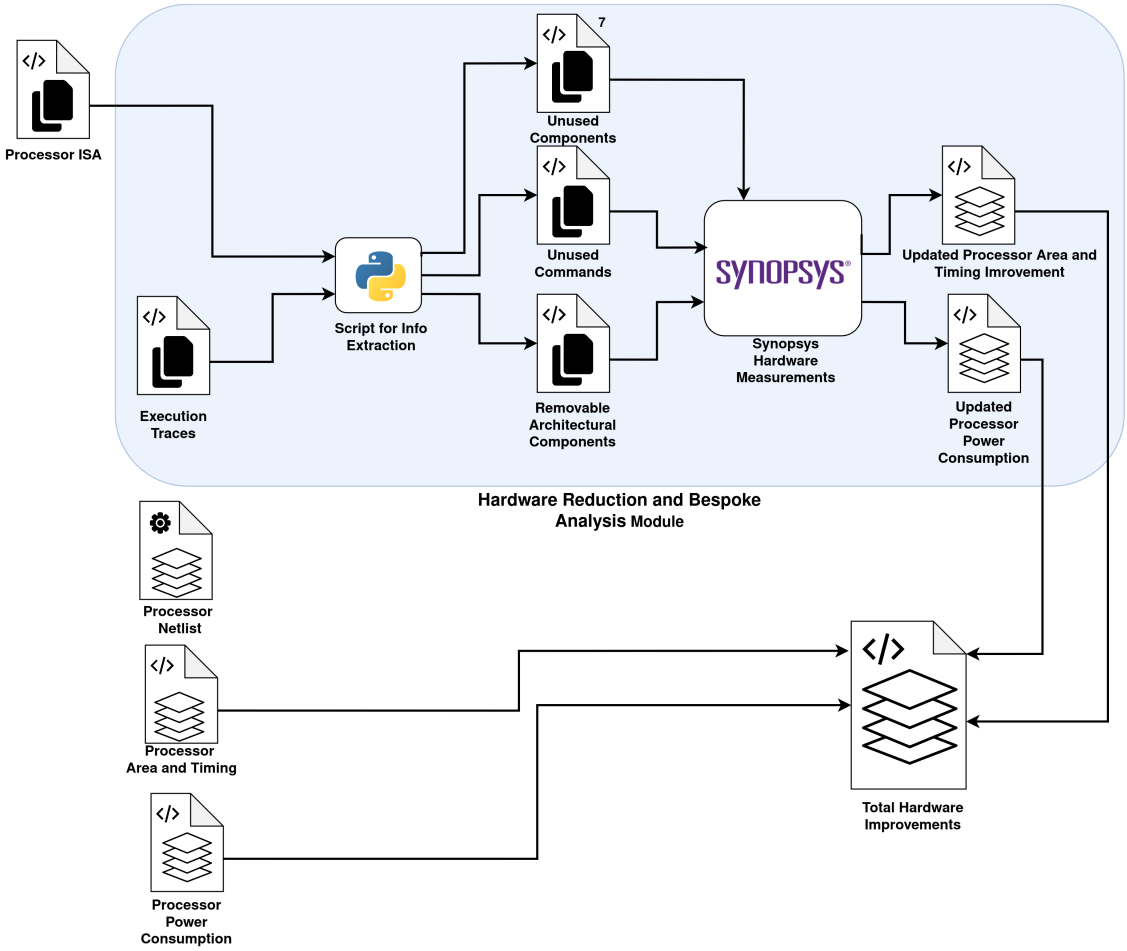

Figure 3.4.1: Detailed view of HW Reduction module

since all of them contain one or more of the basic functionalities mentioned in 3.4.1. In this particular core, there is a module responsible for communication with program and data memory, the Memory Backbone, which is needed for reading commands and data and is thus also needed.

After looking at the produced code by the cross-compiler, we can eliminate components from the unused functionalities. Firstly, we do not request any debugging operations, nor use any external communication protocols (the supported were UART and I2C) in our C code, which is also reflected in the final assembly. We also observe that the code does not include initialization and use of NMIs or Watchdog utilities during execution of any of the applications. Additionally, neither the C code nor the assembly code contain any commands that indicate need for DMA support.

With all of the above in mind we continue by undefining the Serial Debug Interface, Watchdog, all the extra clock options from the Basic Clock Module and the DMA option for the Memory Backbone module. We run the application suite 2 times, with and without the removal of the 16x16 Multiplier peripheral, in order to gather results and compare both implementations both on the hardware and software side which we analyse later.

Zero-riscy

For the Zero-riscy core, components need to be excluded manually, within the verilog description of the core, which requires more attention to the management of signals and maintaining functional equivalence of the rest of the core. In this case the high-level Execution, Fetch, Decode and Load-Store blocks include the essential functionalities for operation and thus are not considered for removal.

In similar fashion to the OMSP430 analysis, there are no debugging commands used in any of the applications, nor are there need for interrupts. Additionally, we did not intend on using any of the compressed commands, so they were disabled in the cross-compiler options and of course not used in the final assembly. For No further modules could be removed since many applications used direct control registers operations and the rest of the modules were all needed for greater modules to operate properly.

In the end, we remove the Debug Unit, Interrupt Controller and Compressed Decoder Units. After manually removing these units we rerun the applications to check for functional equivalence of the core. For the multipliers, the Pulpino has 2 implementations of Multipliers, a Slow and a Fast one. We only consider designs that implement either the Fast Multiplier or that implement no Multiplier, to realise the area-latency tradeoff.

The benefits both from the undefines of OMSP430 and from the changes on Zero-riscy are shown in Table 3.3 where R is reduced.

3.4.2 Unused Commands

The next step in hardware removal is locating the commands that are not used in our applications and removing the associated hardware.

Table 3.3: Comparison of Baseline and module reduced implementations

Cores	Area	Power	Max Timing	Synthesized Timing
Zeroriscy	67.53	291.21	14.49	5.91
Zeroriscy_R	61.82	264.31	14.59	5.91
Zeroriscy_noMul	49.36	220.33	14.49	5.91
Zeroriscy_noMul_R	43.08	190.98	14.87	5.91
OMSP430	50.44	205.22	4.25	4.07
OMSP430_R	33.21	132.31	4.94	4.07
OMSP430_noMul	42.30	177.30	4.25	4.07
OMSP430_noMul_R	25.07	104.19	4.94	4.07

For locating the commands we used the complete instruction set of each core and created a parser script for each one. These scripts receive as inputs the Instruction Set and the output of the tracer and output all the commands that were not found in the tracer from the Instruction Set as well as their type. We then call the script for all tracer files of our applications and get the complete list of unused commands for all of them. We can then look at the output and remove the commands one by one from the core.

The removal is a more complex process than simple module removal because it requires removing specific hardware in the Decoder module that recognizes when the command arrives and hardware in the Execution block that computes the command and communicates with the registers. Since many commands can reuse the same resources in a core, it is common that finding an unused commands can lead to no gain if the same resource is also occupied by a used command.

Zero-riscy

For the Zero-riscy core, we extract the unused commands with the method explained above, and we follow up by checking which of them can lead to reduction in hardware, mainly from the Decode and Execute blocks of the design. After this we have our final set of removable operations mentioned in Table 3.4, showing the type and the specific instructions.

Table 3.4: Zero-riscy Unused Commands

Type	**Instructions**
Set Less Than	slt, sltu, sltiu
Control Status Register	csrrs, csrrc, csrrwi, csrrsi, csrrci
Multiplication High Byte	mulh, mulhu, mulhsu
System Calls	ecall, ebreak, wfi

In Figure 3.4.2 we report the area, power and timing improvements, where Pulpino Reduced ISA is the core with the removed instructions. Area and Power gains are very small, between 1% and 2% and no timing gains.

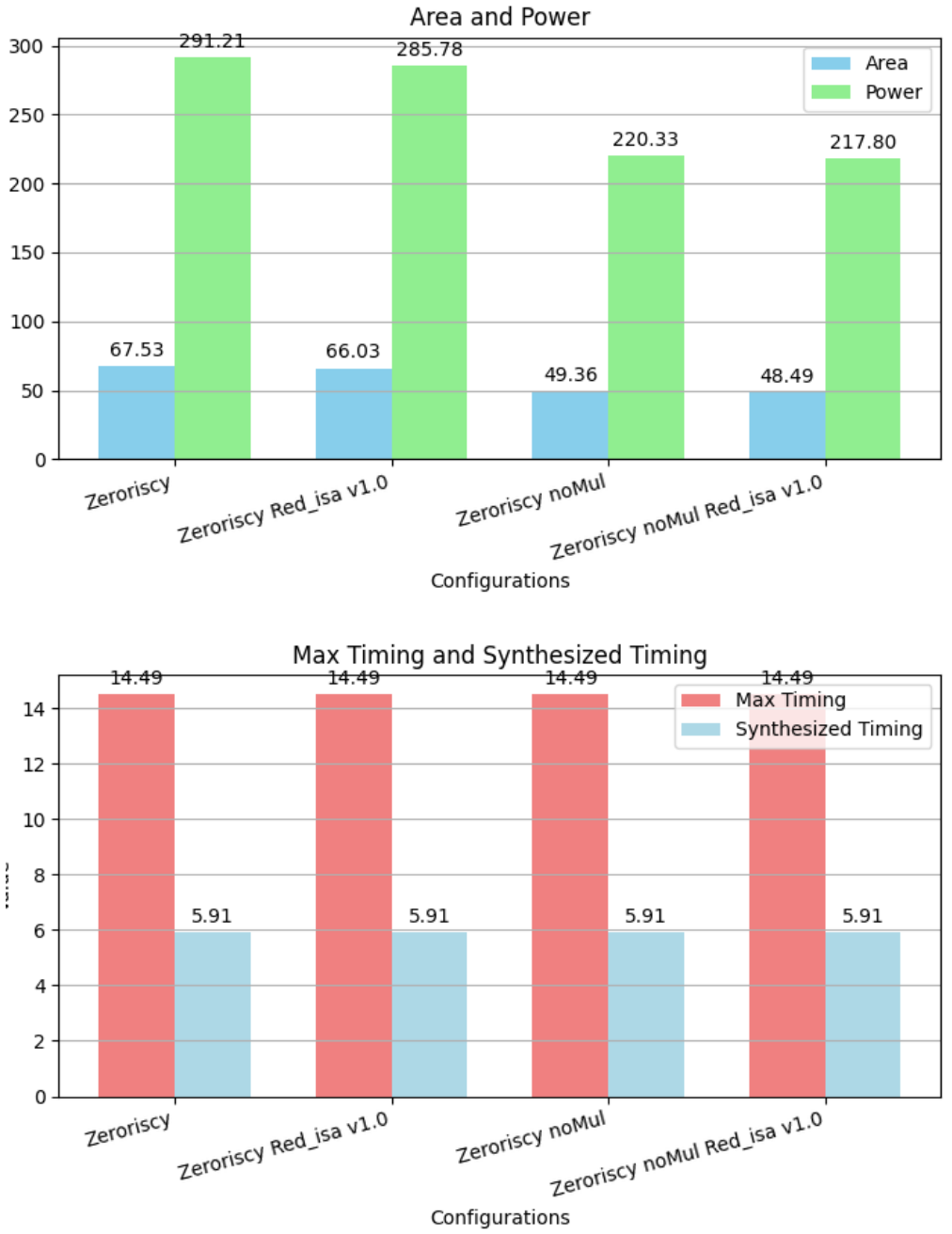

Figure 3.4.2: Area, Power and Timing Gains for the Zero-riscy Reduced ISA implementation

OMSP430

For the OMSP430 after extracting the unused commands for all the applications, we discover that most of them reuse the hardware of useful commands, as described above. This means that this approach does not produce meaningful gains for this particular application set. This however does not mean that finer levels of analysis cannot provide important improvements, as we well see in the following section.

3.4.3 Architectural Components

Since we now have the unused instructions per application and the basic instructions datapath responsible for them removed, we can continue by reducing the finer architectural components needed for each application. This subsection looks into the utilization of three basic architectural components:

1. Architectural Flags of the Status Register, where these exist
2. Number of Architectural Registers
3. Program Counter and Base Address Register size

This aims in seeing how tailoring the core to specific needs of each application can bring gains to our design. As mentioned in [8], registers in the EGFET printed technology library have a very significant cost for area and power compared to combinational circuits. This makes the choice of architectural components a good one, since all of them are implemented with registers and any excess sequential logic comes with a substantial overhead.

The removal and gains from this kind of tailoring is also examined in [8] for their specific TP-ISA core.

Architectural Flags of the Status Register

Architectural flags represent specific settings or indicators within a processor that govern its behavior or configuration. These flags serve as binary markers or variables that inform the processor about certain conditions or modes of operation, influencing its execution flow and outcomes. They are pivotal components of microarchitectures, providing mechanisms for tasks such as conditional branching, data manipulation, system state management and arithmetic/logical operations.

- For this stage, the Pulpino core remains unchanged since it is of RISCV architecture. The ISA of RISCV processors does not contain architectural flags and there is no gain here.
- As mentioned in the previous section, the OMSP430 had no unused commands for the whole set of applications, so this analysis is done per application. The OMSP430 architecture contains 4 flags in the Status Register:
  - Z flag: Is set to 1 when the result of a byte or word operation is 0 and cleared when the result is not 0

- V flag: Is set to 1 when the result of an arithmetic operation overflows the signed-variable range

- N flag: Is set to 1 when the result of a byte or word operation is negative and cleared when the result is not negative

- C flag: Is set to 1 when the result of a byte or word operation produced a carry and cleared when no carry occurred

After examining the ISA we create a document that contains the relations of each instruction to each architectural flag and modify the python script that locates the unused commands to finally produce the removable architectural flags. Table 3.5 shows the unused architectural flags per application tested, with and without use of a multiplier, where Baseline is the standard number of flags in the core and AllApps is the flags needed to be able to run all the apps in the suite. Table 3.6 presents the area and power overhead of each the removable flags when referring to each application.

Table 3.5: OMSP430 Architectural Flags Usage per Application

Application	**Required flags**
Baseline	4(Z, V, N, C)
AllApps	4
mult	4
div	3(no Z flag)
inSort	3(no V flag)
intAvg	4
crc8	4
tHold	4
mlp	3(no Z flag)
dTree	4
AllApps w/ Mult Unit	4
mult w/ Mult Unit	4
mlp w/ Mult Unit	3(no Z flag)

Table 3.6: OMSP430 Architectural Flags Area and Power Overhead

Flag Removed	**Area gain(cm2)**	**Power gain(mW)**
Z	0.09	0.4
V	0.11	0.06

The results derived from Table 3.6 show reductions that equate to 0.4% of the total area and power, which is too small of a benefit for these bigger cores. In comparison, smaller cores like TP-ISA [8] tend to benefit a lot more from such small changes.

Number of Architectural Registers

Architectural registers constitute a crucial component of a processor's internal structure, serving as storage elements directly specified by the processor's ISA. These registers facilitate the execution of instructions and managing the processor's state during program execution. Architectural registers typically encompass various types, including general-purpose registers, which hold operands and intermediate results during computation, and specialized registers such as program counters, instruction registers, and status registers. Architectural registers play a critical role in coordinating the cycle of the processor, facilitating efficient instruction execution and ensuring the proper handling of program flow and system state transitions.

- The Zero-riscy core uses specifically the RISCV32E ISA, which means that due to the specialized registers, we cannot have designs with less than 5 architectural registers, or we run into errors. Again we modify the python script to parse the tracer file and output all the needed registers for each application. Table 3.7 contains the needed registers for each application to run.

Table 3.7: Pulpino Number of Required Registers per Application

Application	**# of required Registers**
Baseline	15
AllApps	14
mult	7
div	10
inSort	14
intAvg	7
crc8	8
tHold	8
mlp	12
dTree	8
AllApps w/ Mult Unit	14
mult w/ Mult Unit	6
div w/ Mult Unit	6
mlp w/ Mult Unit	14

  We continue by removing the registers from HW and measuring the improvements. First we measure the gains for the mult application, by dropping the number of registers from 15 to 7. We also measure gains from dropping all register's Data Width to half, from 32-16, since none of our applications required 32 bit accuracy. The results are shown in Table 3.8.

- The OMSP430 core again contains 16 registers, out of which 4 are special purpose and 12 are general purpose, meaning we cannot have less than 5 registers needed for an application.

  With a similar methodology we extract the registers as shown in Table 3.9. For the register removal, we test gains from removing a single GPR and from reducing the data

Table 3.8: Pulpino Gains from Register removal

	Area gain(cm2)	**Power Gain(mW)**
15 to 7 Registers	7.41	32.36
32 to 16 Data Width	10.95	44.57

width of all GPRs by one. Results are shown in Table 3.10.

Table 3.9: OMSP430 Αριθμός καταχωρητών γενικού σκοπού ανα εφαρμογή

Application	**# of required Registers**
Baseline	15
AllApps	10
mult	8
div	8
inSort	7
intAvg	6
crc8	5
tHold	6
mlp	10
dTree	5
AllApps w/ Mult Unit	9
mult w/ Mult Unit	6
mlp w/ Mult Unit	9

Table 3.10: OMSP430 Gains from Register removal

	Area gain(cm2)	**Power Gain(mW)**
GPR Removal	1.27	5.31
16 to 15 Data Width	1.59	6.78

Program Counter and Base Address Register Size

The PC and BAR are important specific architectural registers with the purpose of addressing memory spaces. The PC is used to address the program memory and is the placeholder for the next instruction that is to be fetched, decoded and executed. On the contrary, the BAR addresses the data memory and IO spaces. Since we do not use other IO for either of the 2 cores when running these applications, the BAR is used to only address the Data Memory.

We can extract the amount that we can to reduce the PC by examining the size of the program memory of each application, which is produced after we compile the code. If the program memory contains N words then out PC need only be 2^N bits in order to address it. The same holds for the BAR when it comes to addressing the data memory. We create scripts that take the memory images, produced by cross-compiling, as input and generate the required bitwidth for addressing. Table 3.11 contains the minimum bitwidths for both processors and all applications.

For area and power gain we examine the PC in the case of the mult app that in both cases requires 5 bits for addressing the program memory. The improvements are shown in Table 3.12 In many cases for both processors, some applications do not require any space in data memory, this is shown in the table with a dash.

Table 3.11: Required bits for Memory Addressing

Application	**Pulpino PC**	**Pulpino BAR**	**OMSP430 PC**	**OMSP430 BAR**
Baseline	32	32	16	10
AllApps	10	8	9	8
mult	5	-	5	-
div	7	-	5	-
inSort	8	5	7	5
intAvg	7	-	6	-
crc8	7	-	7	-
tHold	10	5	6	5
mlp	10	8	9	8
dTree	5	-	6	-
AllApps w/ Mult Unit	10	8	9	8
mult w/ Mult Unit	4	-	5	-
div w/ Mult Unit	4	-	-	-
mlp w/ Mult Unit	9	8	9	8

Table 3.12: Gains from reduced memory addressing

	Area gain(cm2)	**Power Gain(mW)**
Puplino PC Drop from 32 to 5	5.31	24.89
OMSP430 PC Drop from 16 to 5	2	6

Chapter 4

Machine Learning Acceleration Units

In this chapter we present the computing units that target the performance of our cores when handling ML workloads. We discuss design choices for the neural operations unit. We present the work regarding the application of precision scaling on these units and a few design remarks.

4.1 Neural Operation Accelerator

As mentioned in the theoretical background, MLPs and SVMs are dominated by Multiply-Accumulate operations, that require multiplication of partial products, addition of a bias and addition to a total accumulator value. Accelerators for such operations often contain several MAC units that efficiently compute such operations. In a standard processor, one MAC operation would take at least 2 cycles to execute if the processor contains a one-cycle multiplier and several more cycles for other cases. Furthermore, full multipliers tend to be very power hungry and occupy a lot of area, so it is evident that improvements in this operation can cause major impact in circuit characteristics.

4.1.1 Design of Neural Unit

The accelerator unit was implemented in Verilog HDL and attached to each of the tested cores, being added in the ISA as commands, accessed through the decoder. The unit has an internal signal for accumulation and implements essentially 2 functionalities; executing the MAC operation with the result being stored back to the accumulator (Equations 4.1.1, where inp_1 is the input sample and inp_2 is the weight of the model) and resetting the accumulator/loading the new bias (Equation 4.1.2, where the bias is given in inp_2). Inside the unit, the accumulated value for the current neuron is always stored with dedicated registers inside the unit. The 1st operation takes as input 2 integers, performs multiplication, adds the result to the value that the accumulator registers hold currently and stores the result in the accumulator registers within the unit, in a single cycle. The 2nd operation takes as input 1 integer, clears all the accumulated contents and sets the integer as a new bias for the calculation of the next neuron, in one cycle.

$$accum = (inp\_1 * inp\_2) + accum \tag{4.1.1}$$

$$accum = inp\_2 \tag{4.1.2}$$

When testing for the results we print the outputs of the core and check for erroneous calculation. When running the unit with RV32M ISA included, we observe that the assembly from the ri5cy-gnu-toolchain compiler includes div instructions in the routine for printing the output and since we replace the mult-div unit with our new accelerator unit, we receive trash characters in the standard output. For testing purposes only, since the actual core applications do not use the uart functionality, we include a small module that implements division when it is needed for printing the output and debugging as shown in Figure 4.1.1 The division unit is removed for actual hardware measurements and measuring true program size and execution cycles.

We characterized the baseline cores and the neural unit and the core designs for the Zero-riscy and TP-ISA core using the Synopsys EDA toolflow with the EGFET standard cell library in order to extract information about area, power and timing and compare to baseline. In

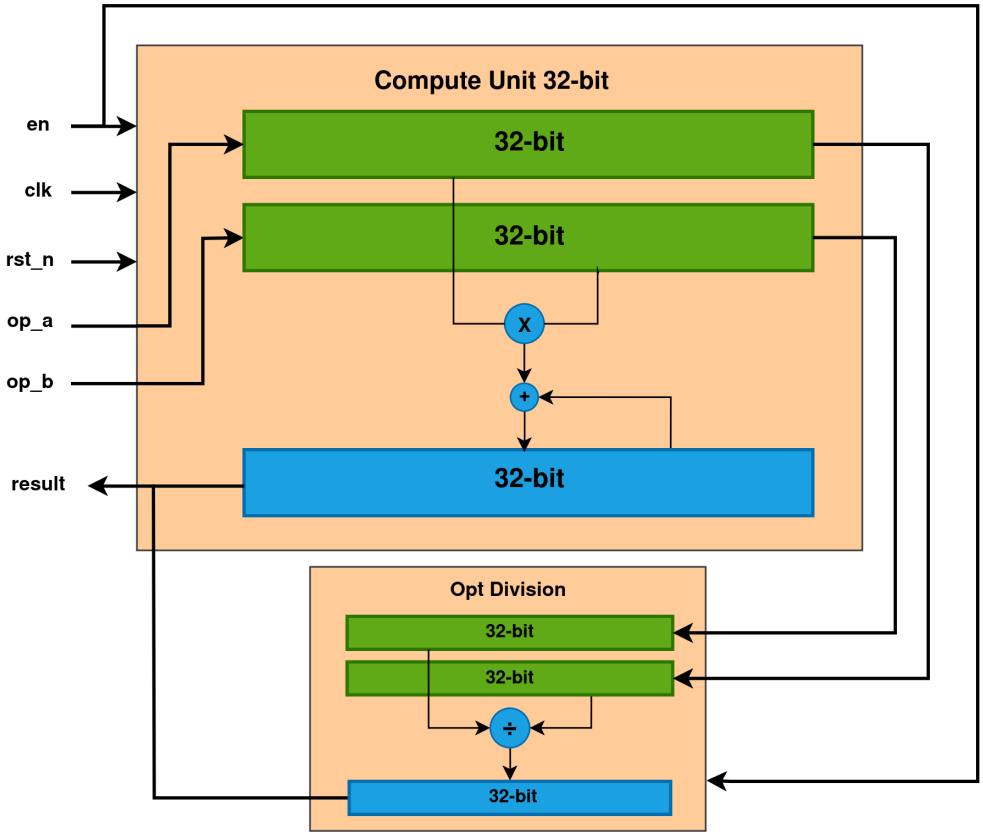

Figure 4.1.1: 32 bit neural unit containing a small division module used for Debugging and Testing

the cases that we had no multiplier previously, we expect to have a new overhead in the above metrics, caused by the insertion of additional hardware and in the cases where a unit is replaced to insert our new accelerator, we present the respective trade-offs.

4.1.2 MAC execution on cores

Zero-riscy Core

The Zero-riscy core, implements RV32M ISA and instructions include multiplication and division related instructions directly with the use of a 4-stage multiplication-division unit. There is also the option for not use of a multiplier unit which conserves power, area and timing but has greatly increases the execution time for workloads that are heavy on multiplication or division operations. The new unit was integrated in the place of the old multiplication/division unit and thus accessed through the multiplication and division commands in the software. The reset/bias setting is tied to the division command and the single stage MAC operation is tied to the multiplication command, since none of the original multiplication and division commands are needed for the calculation of our models.

Since the old multiplication and division commands were multicycle commands and caused the core to stall, additional changes to the core's HDL files were necessary in order to achieve a true one cycle MAC execution. The Instruction Fetch, Instruction Decode and Execution Block modules were modified in order to prevent the processor stall and start fetching the next command immediately in the following cycle. Additional modification of relevant control signals was also performed.

TP-ISA Core

The leading printed processor TP-ISA is a single cycle, configurable bitwidth core that does not contain a multiplication unit. As mentioned, this does conserve area and power from the design, but causes the execution cycles to skyrocket for applications heavy on the multiplication and increases the program memory that now needs to contain the code for multiplication using basic addition and shifting operations. The accelerator unit was implemented as a new command, performing operations similarly to the Pulpino implementation. The ISA of the core has space for encoding a test instruction, so with the appropriate modifications to the decoder and alu modules, the new command is recognised and usable.

4.2 Precision Scaling of Neural Units

Since our focus is to reduce execution time and consumption for a printed processor that calculates the computations of a ML model, we explore the tradeoff between the model's accuracy and the speedup in computations. When using basic precision scaling for the units, meaning that we replace each unit with a similar version, scaled down, we can expect power, area and possible timing savings, but not reduction in cycles needed for execution, since the code executed remains unchanged, only decreasing the bit precision. To tackle this, we take advantage of the full bit-width of the Zero-riscy core and multiplex the 32-bit size to fit multiple MAC operations of lower precision.

The base neural units contained a 32x32 bit Multiplier along with a 32 bit accumulator where the partial products are added for a neuron. We develop scaled down versions containing 16x16, 8x8 and 4x4 MAC units. This allows for simultaneous calculation of 2, 4 and 8 partial products in one cycle of operation. 4.2.1

In detail, the scaled down 16x16 units receive as input the two 32bit registers r1 and r2, but now instead of containing one 32bit parameter each, they contain two 16bit each. The unit contains two 16x16 Multipliers that execute the calculations and store the output to the accumulation register as shown below:

$$\begin{aligned} acc_1 &= (r1[7:0] * r2[7:0]) + acc_1; \\ acc_2 &= (r1[15:8] * r2[15:8]) + acc_2; \\ acc_3 &= (r1[23:16] * r2[23:16]) + acc_3; \\ acc_4 &= (r1[31:24] * r2[31:24]) + acc_4; \\ accum &= acc_4 + acc_3 + acc_2 + acc_1; \end{aligned} \tag{4.2.1}$$

Figure 4.2.1 showcases the schematic of the 32x32 bit unit with 8 bit precision scaling.

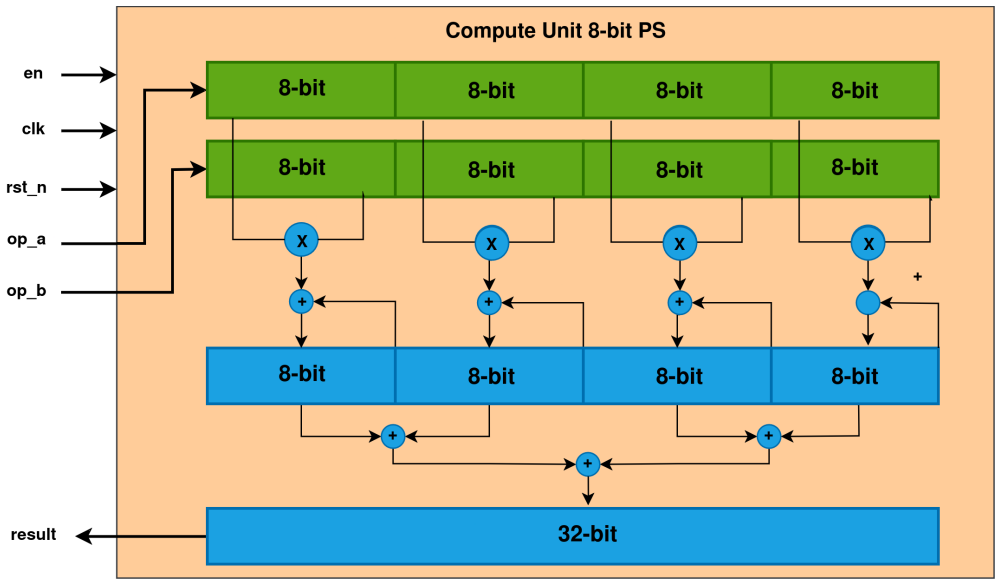

Figure 4.2.1: Overview of the 8 bit precision scaling unit

With the 16x16 design, it is evident that a neuron's value from partial products and accumulation can be calculated in half the cycles. The further scaled 8x8 and 4x4 units work similarly, with the exception of more MAC units and that the 32 bit registers hold even double the parameters. This approach requires an operation between layers of the network that reorganizes the previous layer exits to the format of the registers that we describe above. The operation is a concatenation in order to bring 2 two neuron outputs in one register and have the units work as mentioned.

Chapter 5

Experiments and Results

In this chapter we present the experiment setup and methodology, and the results for our cores and units when running the targeted workloads. We measure area, power and timing for hardware and the respective execution time and cycles of the targeted apps, and compare our proposed changes to the baseline processors.

5.1 Models Tested

The Neural acceleration unit was evaluated from 3 datasets for MLP and SVM. We tested for both classification and regression tasks in each model for each dataset [15]. The presentation and analysis of the models considers the base 32x32 MAC unit.

5.1.1 RedWine

This dataset is related to red wine samples from the north of Portugal. The goal is to model wine quality based on 11 features regarding physicochemical tests. This application greatly suits the benefits of printed technologies. The flexibility of the substrate, the low-cost and disposability allow for printed processors to be applied to wine bottles and predict the wine's quality.

MLP

The RedWine MLP Classifier consists of 2 fully connected layers with 2 input neurons and 6 output neurons, using relu as activation function for each neuron. This means that for a single sample of 11 features there are 34 MAC operations (34 multiplications and 34 accumulations) and 8 bias loads. For reference, standard Zero-riscy with multiplier requires 8 bias loads of 1 cycle, 34 multiplications of 4 cycles and 34 accumulations of 1 cycle, totaling 170 cycles. Comparing to our mac unit, it would require 8 bias loads of 1 cycle and 34 MAC operations of 1 cycle, totaling 42 cycles on neural operations.

The RedWine MLP Regressor consists of 2 fully connected layers with 2 input neurons and 1 output neuron, using relu as activation function for each neuron. So for 1 sample of 11 features the whole model performs 26 MAC operations and 3 bias loads.

SVM

For the calculation of the SVM Classifier it is required to calculate 15 sets of loading a bias and perform 11 MAC operations (as many as the features) for each inference input, to produce a classification result. This comes to 15 bias loads and 165 MAC operations, severely more than the MLP counterpart. This of course, coupled with the lack of needing to additionally calculate the relu activation functions, indicates a bigger room for speedup by optimizing MAC operations. Respectively, the Zero-riscy with multiplier requires a total of 840 cycles while our unit requires just 180 for neural operations.

The SVM regressor is a much smaller model, where instead of calculating 15 sets of feature MAC operations, we only calculate a single set of 1 bias load and 11 MAC operations, one for each feature. Since it is not as MAC dense, we would expect a smaller speedup, comparatively to the classifier.

5.1.2 WhiteWine

Similarly to the RedWine dataset, this dataset is also related to white wine samples from the north of Potrugal and the task is to model wine quality based on 11 features. This is also a well suited application for printed electronics, for the reasons mentioned previously.

MLP

The WhiteWine MLP Classifier consists of 2 fully connected layers with 4 input neurons and 7 output neurons, using relu as activation function for each neuron. So for a single sample of 11 features there are 72 MAC operations (72 multiplications and 72 accumulations) and 11 bias loads.

The WhiteWine MLP Regressor consists of 2 fully connected layers with 4 input neurons and 1 output neuron, using relu as activation function for each neuron. So for a single sample of 11 features there are 48 MAC operations and 5 bias loads.

SVM

For the calculation of the SVM Classifier it is now required to calculate 21 sets of loading a bias and perform 11 MAC operations for each inference input. This comes to 21 bias loads and 231 MAC operations, again more demanding than the MLP counterpart. This also indicates an increase in the complexity of the whitewine model compared to redwine, since the weights, biases and required calculations have increased.

Similarly to the RedWine dataset, for the Whitewine SVM regressor we only need to calculate a single set of 1 bias load and 11 MAC operations, one for each feature.

5.1.3 Cardio

The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians. The fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them. Classification was both with respect to a morphologic pattern (A, B, C. ...) and to a fetal state (N, S, P). The goal is to predict the fetal state based on 21 features from relevant measurements.

MLP

The Cardio MLP Classifier consists of 2 fully connected layers with 3 input neurons and 3 output neurons, using relu as activation function for each neuron. So for a single sample of 21 features there are 72 MAC operations (72 multiplications and 72 accumulations) and 6 bias loads.

The Cardio MLP Regressor consists of 2 fully connected layers with 3 input neurons and 1 output neuron, using relu as activation function for each neuron. So for a single sample of 21 features there are 66 MAC operations and 4 bias loads.

SVM

For the calculation of the SVM Classifier it is now required to calculate 3 sets of loading a bias and perform 21 MAC operations for each inference input, totaling 3 bias loads and 63 MAC operations, this time less demanding than the MLP counterpart.

Similarly to the RedWine dataset, for the Whitewine SVM regressor we only need to calculate a single set of 1 bias load and 11 MAC operations, one for each feature.

5.2 Setup

Figure 5.2.1 shows the complete flow for evaluating the units and the precision scaling technique with the MLP and SVM datasets chosen.

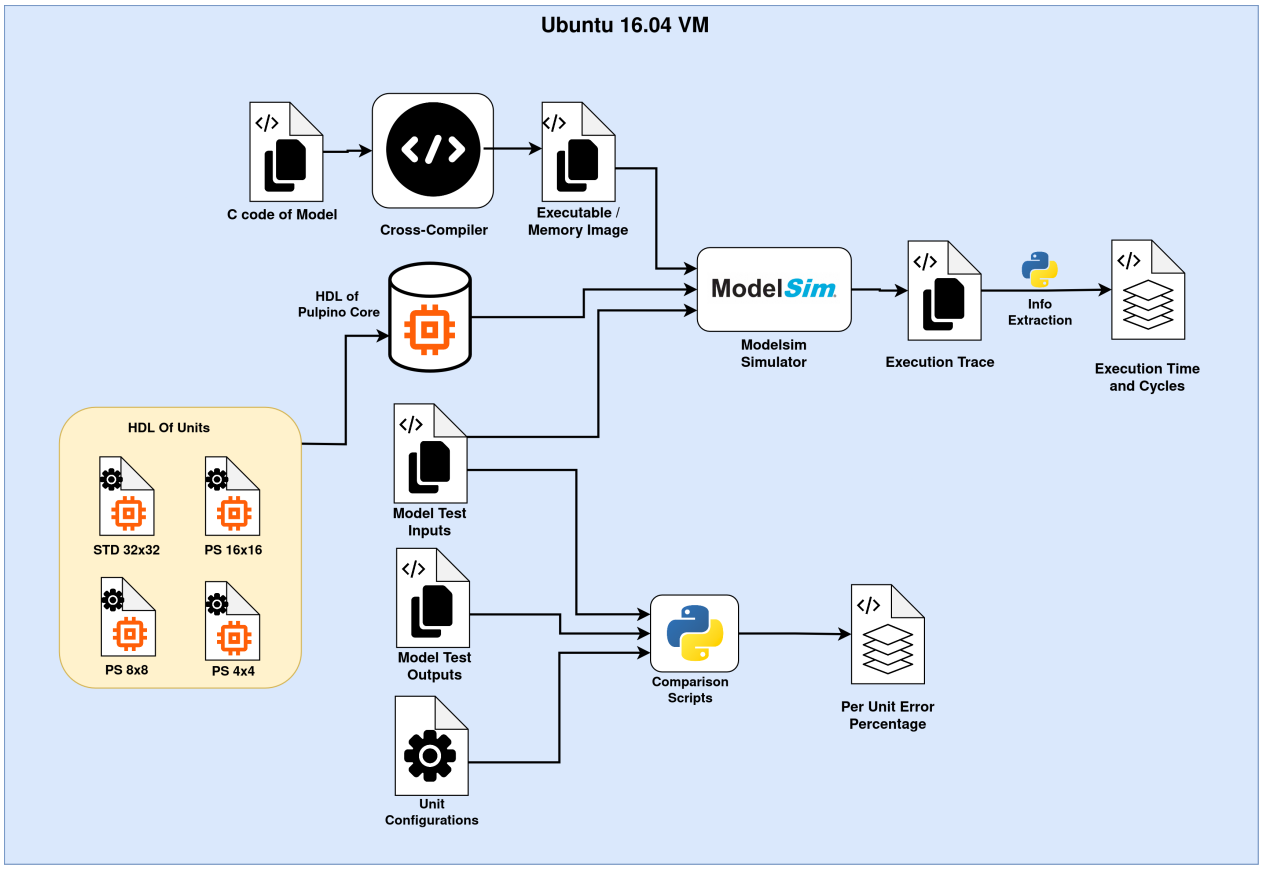

Figure 5.2.1: Experimental Setup

Simulation Environment

The whole environment for running the experiments and evaluating results was set up in Ubuntu 16.04 in a QEMU VM, in order to support all the tools needed, since some required old software versions. We use Modelsim-Intel HDL Simulator to perform rtl simulations of the whole pulpino SoC executing the models. For compiling the applications we setup the ri5cy-gnu-toolchain from the pulp repository and configure it for the zero-riscy architecture. This compiler also provides the corresponding assembly code.

All the models and applications were written in software c code for the baseline and the modified pulpino core version containing the customized commands that utilize the new accelerator units. We use the built in hardware tracer to extract exact timings, cycles and commands executed by the core when running the app. Since we have the assembly code

and the execution traces, we develop python scripts that compute the exact number of cycles by extracting the application commands part from the assembly code and locating the corresponding ones in the traces.

We explored the option of using hardcoded weights to minimise accesses to RAM. This would cause the code to not be able to be rolled into loops, thus creating prohibited program size for most of the large models. In the end the weights were stored into RAM in order to drastically reduce the program and consequently the ROM size.

To extract the speedup percentages, we run each model on 50-100 inputs of each dataset, as many as possible to get a representative sample that also fits in memory.

Hardware Measurements

We measure ROM sizes for our models. We take advantage of the TP-ISA high configurability and explore possible tradeoffs between implementations, keeping a steady PC since we already know the maximum size for our models. We measure our units' hardware resources for base 32x32 and Precision scaled implementations for 16, 8 and 4 bits. Since we intend on implementing the units with TP-ISA, we also measure smaller instances of our units, again with all available precision scaling options. We make a total exploration of all possible Unit-Core combination.

Even though the bulk of our work is done on the Zero-riscy core, TP-ISA stands as the most attractive option to address printed computing restrictions. We provide representative hardware characteristics based on the measurements and execution cycles/execution time basing on the program file generated by the ri5cy-gnu-toolchain compiler since we do not have access to the TP-ISA compiler.

Precision Scaling Tradeoff Calculation

After designing the mentioned units, we integrate them to our mentioned simulation environment, changing the verilog file that contains the neural unit and recompiling the rtl each time. The C code for each application is also modified to accommodate the new needs of the units, since the number of loops and operations need to be cut, in ratio to the scaling. There is also need to incorporate additional commands that handle shifting of values in the registers that carry the output from a layer of an MLP to the next, as described in Chapter 4.2.

We use a python script in order to extract the accuracy loss of each solution. Firstly, the calculations of the whole model for the entire dataset, are calculated with full precision and then with the respective precision of each unit. We then compare the outputs and present a percentage or accuracy loss, due to erroneous classification or regression outputs.

5.3 Results

Following the steps described in 5.2, we run for all possible configurations of units and cores on Zero-riscy to get the execution traces.

Firstly, using the outputs from the compiler we get the size of the ROM in regards to the number of instructions 5.1. The highest number of instructions is 1084, which is addressable by an 11 bit PC, since 11 bits can address 2048 spaces. Now we know that for our TP-ISA configurations we need exactly pc=11 to guarantee that all models run.

Model	ROM space used
Redwine_MLP_C	524
Redwine_MLP_R	448
Redwine_SVM_C	624
Redwine_SVM_R	608
Whitewine_MLP_C	524
Whitewine_MLP_R	448
Whitewine_SVM_C	624
Whitewine_SVM_R	608
Cardio_MLP_C	524
Cardio_MLP_R	504
Cardio_SVM_C	1476
Cardio_SVM_R	1084

Table 5.1: ROM space used by different models

Following we measure all the possible TP-ISA configurations for pc=11 in Figure 5.3.1. We include Neural Units with all available precisions and multiplexes. The lowest one for resources is the one with a datawidth of 4 and no unit attached with 1.38cm2, 6.58mW, 29.13Hz specs for area, power and max clock , while the most demanding one is the one with a datawidth of 32 and a full 32 bit unit with no precision scale and 23.15cm2, 74.43mW, 12.51Hz specs respectively. The 3 hardware metrics seem to have almost a linear relationship.

Now we measure the proposed bespoke Zero-riscy implementations and compare to the baseline Zero-riscy for hardware specs in Table 5.2. The Be in each names means Bespoke and is the standard Zero-riscy with applied hardware reduction and isa reduction as mentioned in chapter 3.4.

We then measure the error introduced with precision scaling. Figure 5.3.2 describes the accuracy loss on the tested models due the scaling precision of our units. We highlight that this applies both to standard smaller units and the larger scaled units regardless. The accuracy loss is calculated as the added error on top of the base model's accuracy. For 4-bit precision scale, the error can reach up to 26% which is strictly prohibited in most applications. However when considering the Cardio dataset the error drops to 1.5% making it viable. For 32 bits and 16 bits, all models have 0% added accuracy loss.

Tables 5.4, 5.5, 5.6 contain the set cycles for each option (a) no Multiplier Unit present, (b) standard Zero-riscy Multiplier Unit present, (c) use of Standard Neural Unit with no

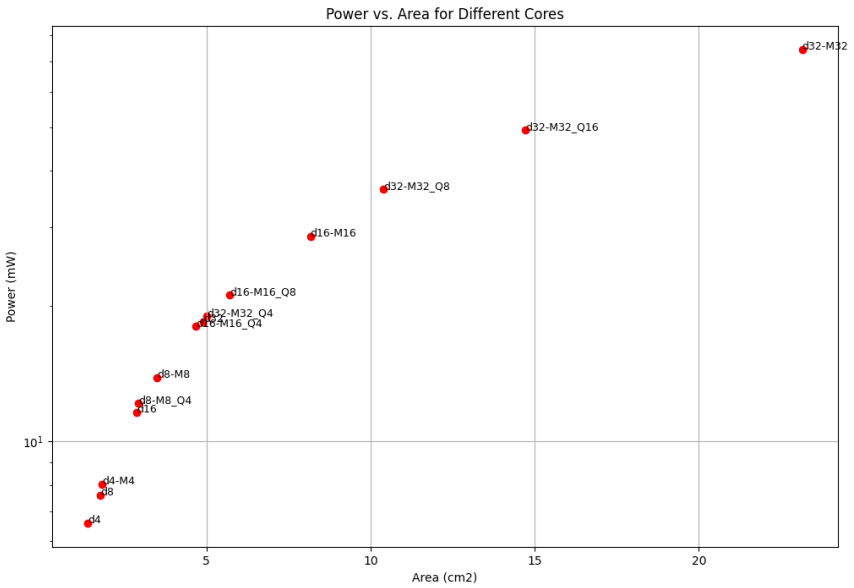

Figure 5.3.1: Total Area-Power diagram for all TP-ISA configs

Core	Core Area	Power	Max Clock
Zeroriscy_Mul	67.53	291.21	14.49
Zeroriscy noMul	49.35	220.33	14.49
Zeroriscy_Be_Mult	60.32	257.80	14.59
Zeroriscy_Be_MAC 32x32	61.96	249.02	14.59
Zeroriscy_Be_MAC PS_16	52.47	222.27	15.02
Zeroriscy_Be_MAC PS_8	47.70	207.37	15.15
Zeroriscy_Be_MAC PS_4	42.86	191.70	15.65

Table 5.2: Hardware metrics for different Zero-riscy configurations

Table 5.3: Speedups and gains of proposed architectures on Zero-riscy Core

Cores	**Area Gain**	**Power Gain**	**Avg Speedup**	**Error**
Zeroriscy-Bespoke	10.6%	11.4%	0%	0.0%
Zeroriscy-Bespoke_MAC32	8.2%	14.4%	23.93%	0.0%
Zeroriscy-Bespoke_MACQ16	22.2%	23.6%	33.79%	0.0%
Zeroriscy-Bespoke_MACQ8	29.3%	28.7%	41.73%	0.5%
Zeroriscy-Bespoke_MACQ4	**36.5%**	**34.1%**	**46.4%**	**15.66%**

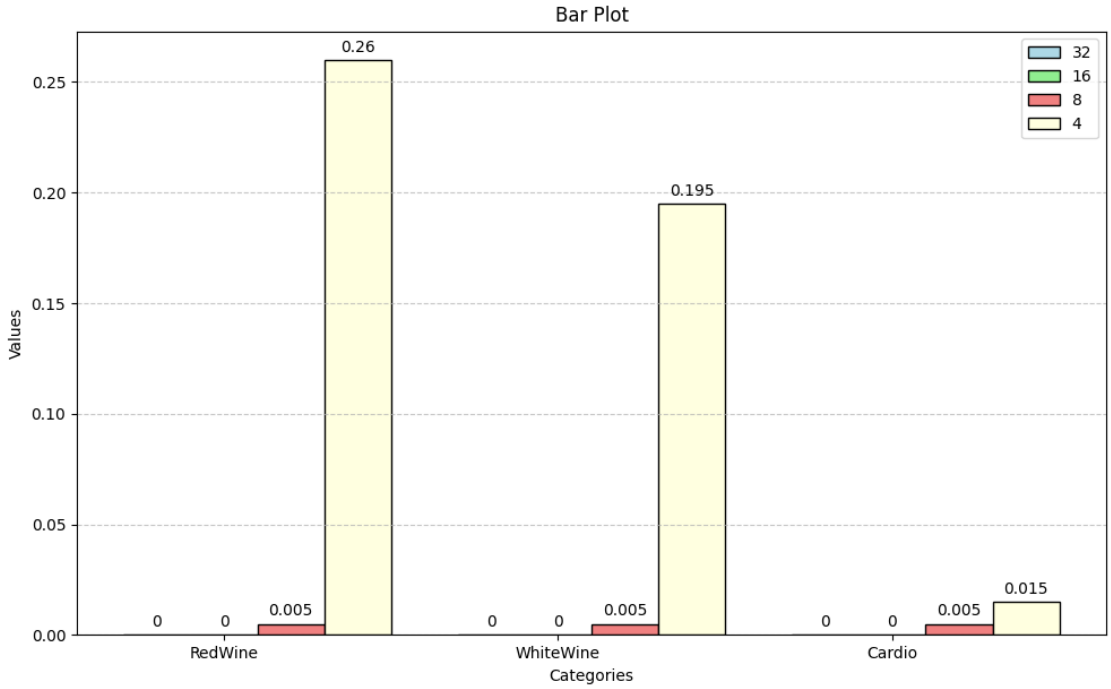

Figure 5.3.2: Error introduced in each model by the chosen precision

Register multiplexing, (d) use of Neural Unit with a x2 register multiplexing, (e) similar with x4 Multiplexing and (f) similar with x8 multiplexing. We group them in this way and not by the precision scaling group because units with low precision where the precision is equal to datawidth, for example TP-ISA with datawidth of 4 and neural unit of precision scale 4, will need the same amount of cycles as a unit with no precision scaling that is used by a bigger core, for example Zero-riscy with standard 32x32 Neural Unit. After we have this information we can use it along with the clock measurements of each configuration to extract the final execution times.

Table 5.4: Execution Cycles comparison for Redwine

Cycles	RWine_MLP_C	RWine_MLP_R	RWine_SVM_C	RWine_SVM_R
Baseline_noMul	750517	597740	2786184	118406
Baseline_Mul	163117	90040	529021	27321
MAC_noMux	140549	70460	386521	20021
MAC_2xMux	98375	59445	338521	19321
MAC_4xMux	93975	55045	289021	17521
MAC_8xMux	88375	49945	269521	16721

It is worth mentioning that for neurons with less connections, using extreme precision scaling for parallelizing mac executions yields less or no benefits benefits. For example, neurons with 4 or less inputs will receive no parallelization benefits by reducing the precision bellow 8 for the 32-bit unit.

Table 5.5: Execution Cycles comparison for Whitewine

Cycles	WWine_MLP_C	WWine_MLP_R	WWine_SVM_C	WWine_SVM_R
Baseline_noMul	1463281	945466	3904056	118716
Baseline_Mul	308081	173566	739621	27421
MAC_noMux	258903	133888	540121	19921
MAC_2xMux	201493	117957	472921	19221
MAC_4xMux	149093	103057	403621	17421
MAC_8xMux	138393	92957	376321	16621

Table 5.6: Execution Cycles comparison for Cardio

Cycles	Cardio_MLP_C	Cardio_MLP_R	Cardio_SVM_C	Cardio_SVM_R
Baseline_noMul	707033	627480	586174	121694
Baseline_Mul	132883	111530	99221	25521
MAC_noMux	102527	82965	71471	18271
MAC_2xMux	88663	72451	63221	18021
MAC_4xMux	72513	63101	56921	17171
MAC_8xMux	65358	56051	50171	15671

As mentioned earlier, in order to parallelize the unit with precision scaling it is necessary to first process the input signals and concatenate them to create the total 32 bit input. In our implementation, this operation happens in software and costs us a few execution cycles, preventing the speedup from reaching similar levels to that of the unit.

We also make an estimation for speedup with TP-ISA, being the SoTA for printed microprocessors. For proper gathering of information we would need the compiler for the core and execution traces to get cycle accurate results. Since we do not have access to the compiler, we are forced to estimate using the traces from the execution of Zero-Riscy. The units were added to the base TP-ISA design and implemented as extensions of the Instruction Set. We run the base and proposed configurations through our hardware flow and extract are, power and timing info. Since we do not have access to the TP-ISA compiler, we use the traces from Zero-riscy with ri5cy-gnu-toolchain to estimate the execution cycles on the core, since all instructions required for the execution of our models already belong to to the Instruction Set of TP-ISA.

Table 5.7 and 5.8 compare the base TP-ISA configurations with our proposed TP-ISA configurations containing versions of our unit with regards to area, power, accuracy loss and rough estimated speedup. Table 5.7 contains the comparison of the base accurate 32 bit TP-ISA implementation to our proposed 32-bit fastest configuration with acceptable error (d32_pc11_MAC32_Q8) and Table 5.8 compares the base small 8-bit TP-ISA with our smallest configuration (d8_pc11_MAC8). Keep in mind that there is no multiplication unit in any of the base processors, so all MAC operations are fed through the ALU in multiple cycles.

Table 5.7: Comparison of proposed Fast configuration against standard 32 bit Accurate TP-ISA

Configuration	**Prop_FAST**
Area Overhead	**x2.12**
Power Overhead	**x1.97**
Avg Err (Base is 0%)	0.5%
Estimated Speedup	**up to 88.5%**

Table 5.8: Comparison of proposed Small configuration against 8-bit TP-ISA

Configuration	**Prop_SMALL**
Area Overhead	x1.98
Power Overhead	x1.82
Avg Err (Base is 0.5%)	0.5%
Estimated Speedup	**up to 85.1%**

Since this is a very small core optimized for low resources and we cannot remove any of the hardware, we naturally expect an increase in hardware resources, that we make up for with increased estimated speedup. Unless the user really values every percentage of accuracy in the model, we can expect from 85.1% up to 88.5% average estimated speedup for our models with about x2 and x1.9 overhead in area and power for the TP-ISA. Precision scaling with less bits yields better results but prohibitive error, in most cases.

Chapter 6

Conclusion And Future Work

In this thesis, we explore the implementation of bespoke microprocessors in printed technology. Firstly, we highlight the need for high area and power reduction in such circuits in order to enable ML applications on ultra small cores. To this end, we tune and implement low gate-count processors, able to execute specific applications, by removing all the logic that is guaranteed not to be used. The removal of full unused components proves very beneficial and time efficient with regards to effort for returns. Removal of architectural components and unused instructions can be more time consuming, but also beneficial especially when targeting sequential components of the processor.

When considering program memory overheads to the design we observe that even slightly more complex benchmarks can quickly cause the ROM to take up resources comparable to some of the small cores. For ROM utilization, designs with smaller instruction lengths have an advantage because they require less ROM cells, while multipliers are proved to worth considering since complex instructions take up memory space when they need to be written for an ALU.

We develop a single-cycle unit in order to improve performance for MAC intensive ML workloads, we implement precision scaling to our units and test with 3 MLP and SVM models. For the models tested, we can greatly increase the unit's performance by parallelising with use of precision scaling for a negligible accuracy loss. However the added overhead from setting up the inputs properly for the precision scaled unit by concatenating and shifting can become a bottleneck.

As for future work, there is room for extensions both in the domain of ML applications like the ones we explored and also for improving in other domains. Netlist and symbolic simulation can be utilized for a given application by propagating undefined signals in the simulation and extracting information about unchanged level of logic gates. This information can be used to tailor the processor to a greater degree, receiving larger benefits in regards to hardware utilization.

Chapter 7

Bibliography

[1] Afan, H. et al. "Modeling the fluctuations of groundwater level by employing ensemble deep learning techniques". In: *Engineering Applications of Computational Fluid Mechanics* 15 (Sept. 2021), pp. 1420–1439. DOI: 10.1080/19942060.2021.1974093.

[2] Aiassa, S. "Low Power Architecture for Address-Event Processing Microcontroller". PhD thesis. July 2017. DOI: 10.13140/RG.2.2.12469.91365.

[3] Armeniakos, G. et al. "Cross-layer approximation for printed machine learning circuits". In: *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*. DATE '22. Antwerp, Belgium: European Design and Automation Association, 2022, pp. 190–195. ISBN: 9783981926361.

[4] Armeniakos, G. et al. "Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey". In: 55.4 (Nov. 2022). ISSN: 0360-0300. DOI: 10.1145/3527156. URL:

[5] Armeniakos, G. et al. "Co-Design of Approximate Multilayer Perceptron for Ultra-Resource Constrained Printed Circuits". In: *IEEE Transactions on Computers* (2023), pp. 1–8.

[6] Armeniakos, G. et al. "Model-to-Circuit Cross-Approximation For Printed Machine Learning Classifiers". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).

[7] Armeniakos, G. et al. *On-sensor Printed Machine Learning Classification via Bespoke ADC and Decision Tree Co-Design*. 2023. arXiv: 2312.01172 [cs.LG].

[8] Bleier, N. et al. "Printed Microprocessors". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 2020, pp. 213–226. DOI: 10.1109/ISCA45697.2020.00028.

[9] Bleier, N. et al. "FlexiCores: low footprint, high yield, field reprogrammable flexible microprocessors". In: *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ISCA '22. New York, New York: Association for Computing Machinery, 2022, pp. 831–846. ISBN: 9781450386104. DOI: 10.1145/3470496.3527410. URL:

[10] Bleier, N. et al. "Rethinking programmable earable processors". In: *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ISCA '22. New

York, New York: Association for Computing Machinery, 2022, pp. 454–467. ISBN: 9781450386104. DOI: 10.1145/3470496.3527396. URL:

[11] Chang, J., Facchetti, A., and Reuss, R. "A Circuits and Systems Perspective of Organic/Printed Electronics: Review, Challenges, and Contemporary and Emerging Design Approaches". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* PP (Mar. 2017), pp. 1–20. DOI: 10.1109/JETCAS.2017.2673863.

[12] Chang, J. et al. "Fully printed electronics on flexible substrates: High gain amplifiers and DAC". In: *Organic Electronics* 15 (Mar. 2014). DOI: 10.1016/j.orgel.2013.12.027.

[13] Cherupalli, H. et al. "Bespoke processors for applications with ultra-low area and power constraints". In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017, pp. 41–54. DOI: 10.1145/3079856.3080247.

[14] Costa, J. et al. "Flexible Sensors-From Materials to Applications". In: *Technologies* 7 (Apr. 2019), p. 35. DOI: 10.3390/technologies7020035.

[15] Dua, D. and Graff, C. *UCI machine learning repository*. http://archive.ics.uci.edu/ml. 2017.

[16] Eriksen, S. O. *Low-power Microcontroller Core*. 2009.

[17] Farabullini, F. et al. "Disposable electrochemical genosensor for the simultaneous analysis of different bacterial food contaminants". In: *Biosensors & bioelectronics* 22 (Mar. 2007), pp. 1544–9. DOI: 10.1016/j.bios.2006.06.001.

[18] Gao, W. et al. "Fully integrated wearable sensor arrays for multiplexed in situ perspiration analysis". In: *Nature* 529 (Jan. 2016), pp. 509–514. DOI: 10.1038/nature16521.

[19] Girard, O. *OPENMSP430*. https://opencores.org/projects/openmsp430.

[20] Gradl, S. et al. "Real-time ECG monitoring and arrhythmia detection using Android-based mobile devices". In: *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference* 2012 (Aug. 2012), pp. 2452–5. DOI: 10.1109/EMBC.2012.6346460.

[21] Harboe, Ø. *ZYLIN ZPU*. https://github.com/zylin/zpu.

[22] Lei, T. et al. "Low-voltage high-performance flexible digital and analog circuits based on ultrahigh-purity semiconducting carbon nanotubes". In: *Nature Communications* 10 (May 2019). DOI: 10.1038/s41467-019-10145-9.

[23] Manjrekar, O. and Dudukovi ć, M. "Identification of flow regime in a bubble column reactor with a combination of optical probe data and machine learning technique". In: *Chemical Engineering Science: X* 2 (Apr. 2019), p. 100023. DOI: 10.1016/j.cesx.2019.100023.

[24] Marques, G. et al. "Progress report on "from printed electrolyte-gated metal-oxide devices to cir-cuits"". In: *Advanced Materials* (2019).

[25] Mora, H. et al. "An IoT-Based Computational Framework for Healthcare Monitoring in Mobile Environments". In: *Sensors* 17 (Oct. 2017), p. 2302. DOI: 10.3390/s17102302.

[26] *MSP430_GCC COMPILER TOOLCHAIN*. http://www.ti.com/tool/MSP430-GCC-OPENSOURCE.

[27] Mubarik, M. H. et al. "Printed Machine Learning Classifiers". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2020, pp. 73–87. DOI: 10.1109/MICRO50266.2020.00019.

[28] *PULPino*. https://github.com/pulp-platform/pulpino.
[29] Rasheed, F. et al. "Predictive Modeling and Design Automation of Inorganic Printed Electronics". In: *Proceedings of the 2019 Design, Automation & Test in Europe (DATE), 25-29 March 2019, Florence, Italy*. Design, Automation and Test in Europe Conference and Exhibition. DATE 2019 (Florenz, Italien, Mar. 25–29, 2019). 43.22.03; LK 01. Institute of Electrical and Electronics Engineers (IEEE), 2019, pp. 30–35. ISBN: 978-3-9819263-2-3. DOI: 10.23919/DATE.2019.8715159.
[30] *RI5CY_GNU_TOOLCHAIN COMPILER TOOLCHAIN*. https://github.com/pulp-platform/ri5cy_gnu_toolchain.
[31] Sethumurugan, S. et al. "A scalable symbolic simulation tool for low power embedded systems". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. San Francisco, California: Association for Computing Machinery, 2022, pp. 175–180. ISBN: 9781450391429. DOI: 10.1145/3489517.3530433. URL:
[32] Subramanian, V. et al. "Printed Electronics For Low-Cost Electronic Systems: Technology Status and Application Development". In: Oct. 2008, pp. 17–24. DOI: 10.1109/ESSCIRC.2008.4681785.
[33] Zhai, B. et al. "Energy-Efficient Subthreshold Processor Design". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.8 (2009), pp. 1127–1137. DOI: 10.1109/TVLSI.2008.2007564.
[34] Zhang, G. et al. "SemanMedical: A kind of semantic medical monitoring system model based on the IoT sensors". In: Oct. 2012, pp. 238–243. ISBN: 978-1-4577-2039-0. DOI: 10.1109/HealthCom.2012.6379414.
[35] Zhao, J.-c. et al. "The study and application of the IOT technology in agriculture". In: vol. 2. Aug. 2010, pp. 462–465. DOI: 10.1109/ICCSIT.2010.5565120.
[36] *ZPU_GCC COMPILER TOOLCHAIN*. https://github.com/zylin/zpugcc.