ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Towards Performance Counter Based Power Modeling

RISC-V ISA Use Case

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γεώργιου Αλεξανδρή

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

Towards Performance Counter Based Power Modeling

RISC-V ISA Use Case

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γεώργιου Αλεξανδρή

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9^{η} Απριλίου, 2024.

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

Αθήνα, Απρίλιος 2024

....

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

στην οικογένεια μου

Περίληψη

Τα τελευταία χρόνια, οι ενεργειακές ανάγκες ενός τυπικού συστήματος γίνονται όλο και μεγαλύτερες. Σε αυτή την κατάσταση κλιμάκωσης, η ανάγκη διαχείρισης της ισχύος έχει καταστεί αναγκαιότητα, με σχεδόν όλες τις οικογένειες μικροελεγκτών να υλοποιούν ένα μοντέλο ισχύος που μπορεί να προβλέψει και, κατά συνέπεια, να διαχειριστεί επίσης την κατανάλωση ισχύος κατά την εκτέλεση μιας εφαρμογής. Σε αυτή τη διπλωματική εργασία προτείνουμε μια μέτρηση της απόδοσης ισχύος κοντά στο υλικό ενός SoC που βασίζεται σε RISC-V ISA και ονομάζεται RocketChip, το οποίο εξομοιώνεται στην αναπτυξιακή πλακέτα ZC706 FPGA, χρησιμοποιώντας το πρωτόκολλο επικοινωνίας με τον πυρήνα ARM του Zynq SoC και καταφέρνοντας να μεταφέρουμε τα δεδομένα απόδοσης χρησιμοποιώντας τις λίστες FIFO που υλοποιεί αυτό το πρωτόκολλο. Με τα δεδομένα που αποκτήθηκαν από αυτή την τεχνική μέτρησης χρησιμοποιούμε τη συσχέτιση Spearman και εκτελώντας στατική (τυπική) και διασταυρούμενη συσχέτιση με τα δεδομένα που αποκτήθηκαν από 10 διαφορετικά benchmarks, καθώς και δύο διαφορετικούς τρόπους αφαίρεσης θορύβου (Άθροισμα Κινούμενου Παραθύρου και Γκαουσιανό Φίλτρο), καταφέρνουμε να παρατηρήσουμε τους περιορισμούς που πρέπει να έχει ένα μοντέλο ισχύος, οι οποίοι είναι ένας ειδικός για την εφαρμογή χαρακτήρας, πρέπει να είναι διαδοχικό και να ανιχνεύει μη γραμμικές συμπεριφορές, να έχει μνήμη των προηγούμενων γεγονότων ισχύος-απόδοσης και να είναι κλιμακούμενο μεταξύ διαφορετικών διαμορφώσεων και συχνοτήτων. Τέλος, καταλήξαμε στο συμπέρασμα ότι οι ήδη υλοποιημένοι μετρητές απόδοσης του RocketChip πρέπει να επεκταθούν προκειμένου να καλύψουν περισσότερα δομικά στοιχεία του συστήματος SoC.

Λέξεις Κλειδιά — Εξαγωγή Χαρακτηριστικών, Μετρητές Απόδοσης, Μοντέλο Ενέργειας, RAPL, RISC-V, RocketChip, Χρονολογική Σειρά

Abstract

Over the last few years, the energy needs of a typical system are getting bigger. In this scaling situation the need of power management has become a necessity, with almost all the microcontroller families implementing a power model which can predict and, as a result, also manage the power consumption of a end application execution. In this thesis, we propose a close to hardware power-performance measurement of a RISC-V ISA based SoC called RocketChip, which is emulated in the ZC706 FPGA development board, by using the communication protocol with the ARM core of the Zynq SoC and managing to transfer the performance data using the FIFO lists which this protocol implements. With the data gained of this measurement technique we use Spearman correlation and performing static (typical) and cross-correlation with the data gained form 10 different benchmarks, as well as two different denoising algorithms (Rolling Average and Gaussian Filter), we manage to observe the constrains a power model should, which are an application-specific character, it needs to be sequential and to detect non-linear behaviors, to have memory of the past power-performance events and to be scalable among different configurations and frequencies. Finally we came to a conclusion that the already implemented performance counters of the RocketChip need to be expanded in order to cover more building blocks of the SoC system.

Keywords — Feature Extraction, Performance Counters, Power Modeling, RAPL, RISC-V, RocketChip, Time Series

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δημήτριο Σούντρη καθώς και τον καθηγητή Σωτήρη Ξύδη για την πολύτιμη καθοδήγηση τους κατά την διάρκεια της διπλωματικής μου. Θα ήθελα επίσης να ευχαριστήσω τους Υποψήφιο Διδάκτορα Χρήστο Λαμπράκο και τον καθηγητή Γιώργο Λεντάρη για την υποστήριξη και την προθυμία τους να βοηθήσουν όποτε ήταν απαραίτητο.

Επιπλέον θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη τους και την παρουσία της σε κάθε στάδιο της ζωής μου ως φοιτητής και την στήριξη τους σε κάθε μου απόφαση και σε κάθε μου κίνηση αυτά τα χρόνια.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου που μου προσέφεραν τα απαραίτητα εφόδια για να μπορέσω να ολοκληρώσω τόσο τις σπουδές μου όσο και την παρούσα διπλωματική εργασία. Τα εφόδια αυτά είναι η αγάπη, η υποστήριξη και η κατανόηση που μου προσέφεραν και μου προσφέρουν, δημιουργώντας ένα κλίμα μέσα στο οποίο ο κάθε άνθρωπος μπορεί να αναπτυχθεί και να εξελιχθεί, κυνηγώντας τα όνειρα του και κάνοντας αυτό που αγαπάει.

Γεώργιος Αλεξανδρής
Απρίλιος 2024

Contents

Figure List

Table List

Chapter 1

Εκτεταμένη Ελληνική Περίληψη

1.1 Εισαγωγή

Η έρευνα τον τελευταίο καιρό επικεντρώνεται στην εξοικονόμηση ενέργειας στα νευρωνικά δίκτυα λόγω της ενεργειακής κρίσης και των μεγάλων ενεργειακών απαιτήσεων των συσκευών μετά την εξέλιξη των νευρωνικών δικτύων[1]. Προτείνεται η παρακολούθηση και προσαρμογή μοντέλων για βέλτιστη χρήση σε διάφορες εφαρμογές, εκμεταλλευόμενη τις τεχνικές μάθησης και τη δυναμική παραμετροποίηση των πηγών ενέργειας, συμπεριλαμβανομένου του ρυθμού λειτουργίας του ρολογιού (DVFS)[2].

Στην ίδια κατεύθυνση πολλές αρχιτεκτονικές θέλουν να υλοποιήσουν κατασκευές βέλτιστης κατανάλωσης ενέργειας, ανάμεσα τους και η αρχιτεκτονική RISC-V η οποία πλέον έχει εισχωρήσει σε αρκετές εφαρμογές της ερευνητικής κοινότητας και έχει τροποποιηθεί για πολλές υλοποιήσεις[3][4][5]. Τα αποτελέσματα όταν συγκρίνονται με άλλους πυρήνες είναι πολύ ελπιδοφόρα[6].

Το γενικό όραμα στο οποίο βασίζεται η συγκεκριμένη έρευνα αποτελείται από την απόκτηση γνώση για την "συμπεριφορά" συγκεκριμένων building block του συστήματος μας με βάση την εποπτεία κατανάλωσης ισχύος αλλά και την παροχή δεδομένων από μετρητές απόδοσης του ίδιου του επεξεργαστή, έπειτα την κατασκευή μοντέλου πρόβλεψης της κατανάλωσης ισχύος με βάσει μετρήσεις που κρίθηκαν οι καλύτερες από την προηγούμενη ανάλυση και έπειτα βελτιστοποίηση του προβλεπτικού αυτού συστήματος. Αυτή η διπλωματική εργασία καλύπτει το πρώτο μέρος από το γενικό όραμα συσχετίζοντας την παρακολούθηση κατανάλωσης ισχύος του RocketChip RISC-V SoC, που εκτελείται σε μια FPGA πλακέτα ανάπτυξης ZC706, χρησιμοποιώντας τους μετρητές απόδοσης που δίνει ο επεξεργαστής. Μέσα από την διπλωματική εργασία θα προσπαθήσουμε να απαντήσουμε στις παρακάτω ερωτήσεις:

Ερώτηση1 : Μπορεί το RocketChip να παρέχει αρκετούς μετρητές επιδόσεων για να χρησιμοποιηθεί ως είσοδος;

Ερώτηση2 : Μπορεί η ανάλυση που πρόκειται να εκτελέσουμε να οδηγήσει σε πρόβλεψη με βάση την εφαρμογή;

Ερώτηση3 : Θα πρέπει να έχουμε μνήμη για μια ακριβή πρόβλεψη;

Ερώτηση4 : Είναι τα δεδομένα συσχέτισης ευαίσθητα στις αλλαγές διαμόρφωσης;

Ερώτηση5 : Μπορεί μια προεπεξεργασία δεδομένων με βάση την αποθορυβοποίηση να οδηγήσει σε καλύτερη ανάλυση δεδομένων ισχύος/απόδοσης;

Για να απαντηθούν τα παραπάνω έγινε μέτρηση σε διαφορετικά περιβάλλοντα και συχνότητες του Rocket-Core συστήματός μας, με επεξεργασία των δεδομένων μας με χρήση δύο διαφορετικών τεχνικών αφαίρεσης θορύβου και συσχέτιση μέσω των συντελεστών συσχέτισης Spearman αλλά και με διασταυρωμένη συσχέτιση.

1.2 Σχετική Βιβλιογραφία

1.2.1 x86 Αρχιτεκτονικές

Η παρούσα ανάλυση επικεντρώνεται στην πρόβλεψη και τον περιορισμό της ισχύος σε πραγματικό χρόνο μέσω της τεχνολογίας Running Average Power Limit (RAPL) της Intel, η οποία επιτρέπει την εξοικονόμηση ενέργειας σε διάφορα συστατικά του συστήματος όπως η CPU και η RAM, μέσω της χρήσης ενός χρονικού παραθύρου για την πρόβλεψη της ισχύος[7][8]. Έρευνες έχουν κάνει εκμετάλλευση των δεδομένων που παρέχει το RAPL για να έχουν περισσότερα περιθώρια αξιοπιστίας στην υλοποίηση ενός μοντέλου[9]. Επιπλέον, εξετάζονται άλλες προσεγγίσεις που βασίζονται στην εκμετάλλευση των μετρητών επιδόσεων και την εξαγωγή χαρακτηριστικών μέσω της συσχέτισης, είτε γραμμικής[10] είτε μη γραμμικής[11], με στόχο την ακριβέστερη πρόβλεψη της ισχύος. Αναφέρονται επίσης μέθοδοι που χρησιμοποιούν το μέσο σφάλμα και τη στατιστική κάλυψη ως κριτήρια για την επιλογή της καλύτερης μεθόδου επιλογής χαρακτηριστικών[12]. Τέλος, εξετάζονται τεχνικές όπως το Dynamic Voltage and Frequency Scaling (DVFS)[12] και η ανίχνευση φάσης εφαρμογής[13][14] για τη βελτιστοποίηση της πρόβλεψης ισχύος και τη μείωση της επιβάρυνσης.

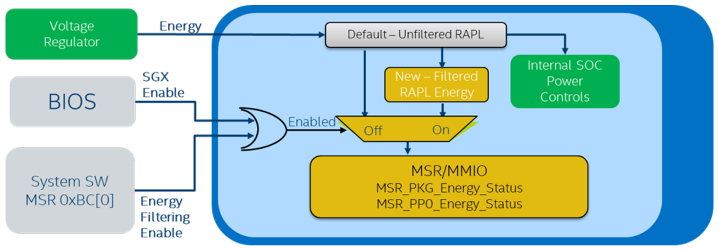


Figure 1.2.1: Η πλατρφόρμα Intel RAPL

1.2.2 ARM Αρχιτεκτονικές

Η έρευνα επικεντρώνεται στην εξαγωγή χαρακτηριστικών με βάση τη συσχέτιση από δεδομένα μετρητών επιδόσεων σε περιβάλλον ARM, χρησιμοποιώντας τον ARM Cortex-A15 και εφαρμόζοντας την προσέγγιση σε κινητές συσκευές[15]. Επιπλέον, μια άλλη μεθοδολογία[16] χρησιμοποίησε το Energy Delay Product (EDP) για την πρόβλεψη της ενέργειας και την εξαγωγή των βέλτιστων σημείων εκπαίδευσης μοντέλου, βασιζόμενη σε διαμόρφωση big.LITTLE με τους πυρήνες cortex-A15 και cortex-A7. Μία τρίτη προσέγγιση[17] χρησιμοποίησε την ίδια διαμόρφωση big.LITTLE σε περιβάλλον εξομοίωσης FPGA, εφαρμόζοντας μετρικές Clocks Per Instruction (CPI) για την εξαγωγή χαρακτηριστικών κατά τη διάρκεια συγκριτικής αξιολόγησης. Τέλος σε σχετική δουλειά [18] έχουν γίνει χρήση και δεδομένα μετρήσεων στο RTL ενός Neoverse A77 πυρήνα για να μπορέσει να φτιαχτεί μοντέλο που στο τέλος θα εκτελεί προβλέψεις σε πραγματικό υλικό.

1.2.3 Αρχιτεκτονικές RISC-V

Σύμφωνα με τα μοντέλα ανάλυσης ισχύος σε αρχιτεκτονικές βασισμένες σε RISC-V υπάρχουν 2 σημαντικά ερευνητικά έργα.

Το πρώτο είναι το MCPat-Calib[19], το οποίο επικεντρώνεται στη βαθμονόμηση των διαδικασιών παρακολούθησης των επιδόσεων σε ορισμένους μετρητές επιδόσεων και δείκτες αναφοράς στον πυρήνα RISC-V BOOM (η εκτός σειράς υλοποίηση του RocketChip) και στην πρόβλεψη της ανάλυσης ισχύος, επιτυγχάνοντας μείωση του μέσου απόλυτου ποσοστού σφάλματος κατά 3,64%-6,14%.

Το δεύτερο ερευνητικό έργο[20] επικεντρώνεται στα δεδομένα που συγκεντρώθηκαν μέσω της χρήσης του Nangate 45nm PDK για την ανάλυση ισχύος διαφόρων δομικών στοιχείων στον μεταγλωττιστή Synopsys Design ενός προσομοιωμένου πυρήνα βασισμένου στο PULP[21], γνωστού ως RI5CY[22]. Κατάφεραν να μειώσουν την ισχύ κατά 2,2% σε σχέση με την προβλεπόμενη μείωση με αυτή τη ρύθμιση μοντελοποίησης.

1.2.4 Μη CPU Αρχιτεκτονικές

Έρευνα πάνω σε GPU εστίασε στη συλλογή προφίλ ισχύος και επιδόσεων από 49 πυρήνες CUDA, χρησιμοποιώντας το CUDA SDK και τη σουίτα συγκριτικών δοκιμών Rodinia. Μέσω της χρήσης μετρητών επιδόσεων, συλλέχθηκαν δεδομένα για την απόδοση εντολών και μνήμης, τα οποία συσχετίστηκαν με την κατανάλωση ισχύος. Αναπτύχθηκε ένα μοντέλο γραμμικής παλινδρόμησης με τους μετρητές επιδόσεων ως ανεξάρτητες μεταβλητές και την κατανάλωση ισχύος ως εξαρτημένη μεταβλητή, επιτυγχάνοντας μέσο λάθος πρόβλεψης 4,7%. Ωστόσο, το μοντέλο αντιμετώπισε δυσκολίες στην ακριβή πρόβλεψη για πυρήνες με ανάγνωση υφής λόγω της έλλειψης σχετικών μετρητών επιδόσεων[23]. Σε άλλη έρευνα, η απόδοση και η κατανάλωση ενέργειας της Google Edge TPU μελετήθηκαν μέσω της ανάλυσης πάνω από 10.000 μοντέλων βαθιάς μάθησης. Η μελέτη αναδεικνύει τη μη γραμμική σχέση μεταξύ ενέργειας/απόδοσης και του αριθμού των πράξεων MAC, καθώς και τη σημασία της χρήσης μνήμης εντός/εκτός του chip για την ακριβή εκτίμηση της απόδοσης. Το πλαίσιο PETET χρησιμοποιεί μεθόδους μηχανικής μάθησης για να προβλέψει online την απόδοση και την ενέργεια της Edge TPU βάσει των ρυθμίσεων του μοντέλου DL, των υπολογιστικών φόρτων εργασίας και της χρήσης μνήμης[24].

1.3 Θεωρητικό Υπόβαθρο

1.3.1 Αρχιτεκτονική RISC-V

Η αρχιτεκτονική RISC-V αναδεικνύεται ως ένα πρωτοποριακό σύστημα εντολών ανοικτού κώδικα, που περιλαμβάνει τις αρχές της μειωμένης συλλογής εντολών (RISC). Αυτή η αρχιτεκτονική, αναπτυγμένη στο Πανεπιστήμιο της Καλιφόρνιας στο Μπέρκλεϊ το 2010, υποστηρίζει ένα ευρύ φάσμα υλοποιήσεων, από μικροελεγκτές με χαμηλές απαιτήσεις ισχύος έως ισχυρούς επεξεργαστές που απαιτούνται για υψηλές επιδόσεις υπολογιστικών περιβάλλονταν. Τεχνικά, το RISC-V προσφέρει ένα επεκτάσιμο σύνολο βασικών εντολών, δίνοντας τη δυνατότητα για εξειδίκευση σε διάφορες λειτουργίες, ενώ η διαχωριστική του αρχιτεκτονική φόρτωσης/αποθήκευσης επιτρέπει απλούστερες και πιο αποδοτικές σωληνώσεις εντολών, μειώνοντας το κόστος ισχύος σε σύγκριση με άλλες αρχιτεκτονικές. Με τη δυνατότητα επιλογής επεκτάσεων σύμφωνα με τις ανάγκες των εφαρμογών, οι προγραμματιστές μπορούν να προσαρμόσουν το RISC-V με ακρίβεια, ενσωματώνοντας μόνο τις απαραίτητες λειτουργίες, όπως floating-pont μονάδα (F), διπλής ακρίβειας floating-point (D), ή ατομικές λειτουργίες (A).

Η αρχιτεκτονική αυτή διαθέτει 4 λειτουργίες επίβλεψης:

- Λειτουργία μηχανής (M-mode) - η πιο προνομιακή λειτουργία, που χρησιμοποιείται κατά την εκκίνηση και σε πολύ συγκεκριμένες περιπτώσεις, οι οποίες απαιτούν τον πλήρη έλεγχο του πυρήνα.
- Λειτουργία hypervisor (H-mode) - μια λειτουργία που χρησιμοποιείται για το virtualization, η οποία δεν υπάρχει σε όλες τις RISC-V αρχιτεκτονικές.
- Λειτουργία επόπτη (S-mode) - μια λιγότερο προνομιούχα λειτουργία, που χρησιμοποιείται από τον πυρήνα του λειτουργικού συστήματος.
- Λειτουργία χρήστη (U-mode) - η λιγότερο προνομιούχα λειτουργία, που χρησιμοποιείται από το λογισμικό εφαρμογών.

1.3.2 RocketChip

Το RocketChip[25] είναι ένας πυρήνας βασισμένος σε RISC-V που ήταν επίσης το πρώτο παραγόμενο έργο περιγραφής υλικού του RISC-V ISA. Πρόκειται για μια εξαιρετικά παραμετροποιήσιμη γεννήτρια SoC ανοικτού κώδικα που μπορεί να ενσωματώσει μια ποικιλία διαφορετικών πυρήνων-επιταχυντών. Η περιγραφή υλικού ενός πυρήνα RocketChip (Rocket Core) είναι γραμμένη σε Chisel[26], μια ανοικτού κώδικα γλώσσα περιγραφής υλικού υψηλού επιπέδου που έχει γραφτεί σε Scala και αναπτύχθηκε στο Πανεπιστήμιο της Καλιφόρνιας στο Μπέρκλεϊ. Η διαμόρφωση του RocketCore είναι πολύ αρθρωτή, επιτρέποντας στον τελικό χρήστη να διαμορφώσει τους πυρήνες, τα δομικά στοιχεία και τις δυνατότητες του πυρήνα. Εκμεταλλευόμενοι αυτό το modularity, μπορούν να κατασκευαστούν διάφοροι επιταχυντές ειδικών εφαρμογών με διαφορετικές διαμορφώσεις. Η αρχιτεκτονική RocketChip μπορεί επίσης να παρέχει αναδιαμορφώσιμα μπλοκ επέκτασης εκτός του βασικού πυρήνα, όπως κρυφές μνήμες L1 και L2, μονάδα διαχείρισης μνήμης (MMU), μονάδα κινητής υποδιαστολής (FPU), μονάδα αποσφαλμάτωσης, μετρητές απόδοσης και ελεγκτή διακοπών, καθώς και επικοινωνία μεταξύ όλων αυτών των στοιχείων.

Το RocketTile αποτελεί το επόμενο επίπεδο διαμόρφωσης του RocketChip και μπορεί να ρυθμιστούν όλες οι πτυχές του πυρήνα, όπως κρυφές μνήμες L1 και L2, MMU, FPU, μονάδα εντοπισμού σφαλμάτων, μετρητές επιδόσεων και ελεγκτή διακοπών. Το RocketTile είναι το βασικό δομικό στοιχείο του RocketChip και χρησιμοποιείται για την κατασκευή του τελικού SoC. Μπορεί να διαμορφωθεί για έναν μονό πυρήνα, έναν multicore ή ακόμα και μια διαμόρφωση πολλαπλών πυρήνων πολλαπλών συστάδων. Οι κυριότερες διαμορφώσεις ενός Rocket Tile είναι οι εξής:

- **BigCore** : Ένας πυρήνας υψηλών επιδόσεων με 16 KiB, 4-way set-associative caches εντολών και δεδομένων που υποστηρίζει FPU by Default.
- **MediumCore** : Ένας πυρήνας με μικρότερες κρυφές μνήμες άμεσης αντιστοίχισης 4 KiB που δεν υποστηρίζει FPU από προεπιλογή.
- **SmallCore** : Ένας πυρήνας χαμηλής απόδοσης με πολύ περιορισμένη κρυφή μνήμη που δεν υποστηρίζει FPU εξ ορισμού.
- **TinyCore** : Μια όχι τόσο συχνά χρησιμοποιούμενη διαμόρφωση πυρήνα με υποστήριξη μόνο για αρχιτεκτονική 32-bit.

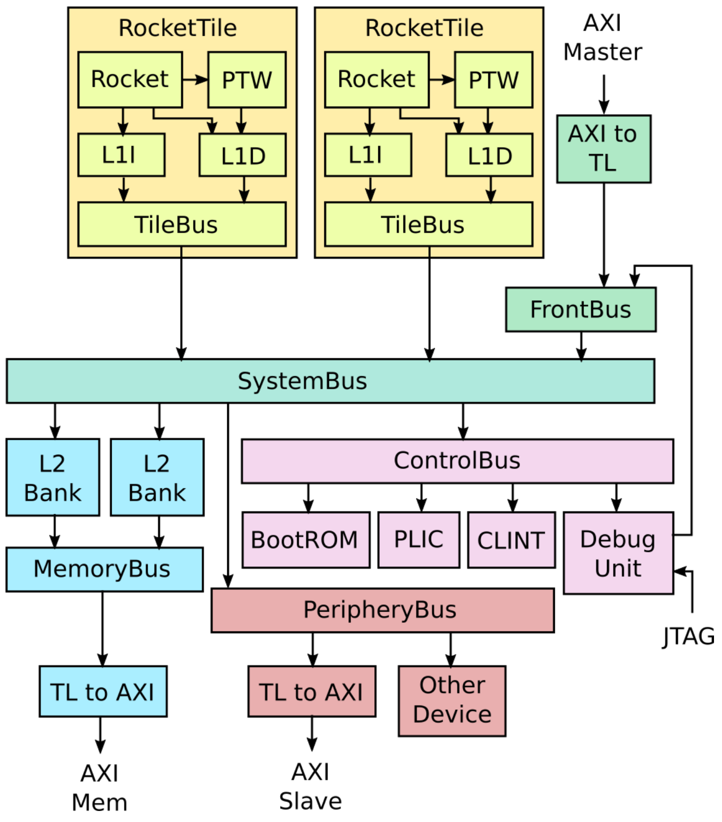


Figure 1.3.1: Παράδειγμα RocketChip Αρχιτεκτονικής.

1.3.3 Διεπαφές με το RocketChip

Front-End Server

Η χρήση του RocketChip σε περιβάλλον προσομοίωσης απαιτεί αποτελεσματική επικοινωνία με τον οικοδεσπότη υπολογιστή. Για τον έλεγχο του πυρήνα, απαιτείται η δημιουργία μιας χρήσιμης υποδομής, που περιλαμβάνει το Front End Server (fesvr)[27]. Το fesvr εκτελεί εκτελέσιμο πρόγραμμα στην αρχιτεκτονική του στόχου (φιλοξενούμενου) και παρέχει λειτουργίες επικοινωνίας για τον έλεγχο του RocketChip. Επιπλέον, μπορεί να χρησιμοποιηθεί για την εκτέλεση κώδικα bare metal και τη φόρτωση ενός bootloader πρώτου σταδίου. Το έργο παρέχει επίσης τη δυνατότητα αντιστοίχισης κλήσεων συστήματος μεταξύ του εξομοιωμένου συστήματος και της κεντρικής μηχανής. Το fesvr ανήκει στο νεότερο έργο Spike[28], αποτελώντας πλέον μέρος της επικοινωνίας κεντρικού στόχου του RocketChip.

Proxy Kernel

Ο Proxy Kernel[29] είναι ένας υψηλότερου επιπέδου από το Front-End Server πυρήνας ο οποίος επιτρέπει στην προς εκτέλεση εφαρμογή την πρόσβαση σε βασικές κλήσεις συστήματος για μία πιο ευέλικτη επικοινωνία με το κεντρικό σύστημα-οικοδεσπότη. Παρόλα αυτά ο Proxy Kernel δεν αντικαθιστά έναν Linux Kernel.

Host Target Interface

Η επικοινωνία μεταξύ του κεντρικού υπολογιστή και του πυρήνα RocketChip γίνεται μέσω μιας προσαρμοσμένης διασύνδεσης, γνωστής ως HTIF (Host Target Interface). Η HTIF αποτελεί το κύριο μέσο επικοινωνίας σε όλες τις πτυχές της πειραματικής διάταξης, υποστηρίζοντας λειτουργίες όπως η αποσφαλμάτωση, η ανίχνευση κατάστασης και η αντιστοίχιση κλήσεων συστήματος. Η επικοινωνία μέσω HTIF γίνεται μέσω δύο FIFO, μια για κάθε κατεύθυνση επικοινωνίας. Η κάθε μηχανή έχει πρόσβαση στο χώρο διευθύνσεων των FIFO για ανάγνωση και εγγραφή. Η εντολή HTIF αποτελείται από τρία βασικά μέρη: τη λειτουργία που ζητείται, το τμήμα δεδομένων που περιλαμβάνει τα ορίσματα και την εντολή εκτέλεσης. Η HTIF χρησιμοποιείται για τη μετάδοση εντολών συστήματος και δεδομένων μεταξύ του κεντρικού υπολογιστή και του πυρήνα RocketChip.

Η HTIF αποτελείται από τρία μέρη:

- **Συσκευή** (Device): Ένα συγκεκριμένο αναγνωριστικό για να υπάρχει γνώση ως προς ποια δομική μονάδα απευθύνεται ο οικοδεσπότης.
- **Εντολή** (Command): Η εντολή που θα εκτελεστεί από τον πυρήνα η οποία αντιπροσωπεύεται επίσης από το αναγνωριστικό της.
- **Δεδομένα** (Data): Τα δεδομένα που συνοδεύουν την εντολή και είναι απαραίτητα για την σωστή διαχείρισή της.

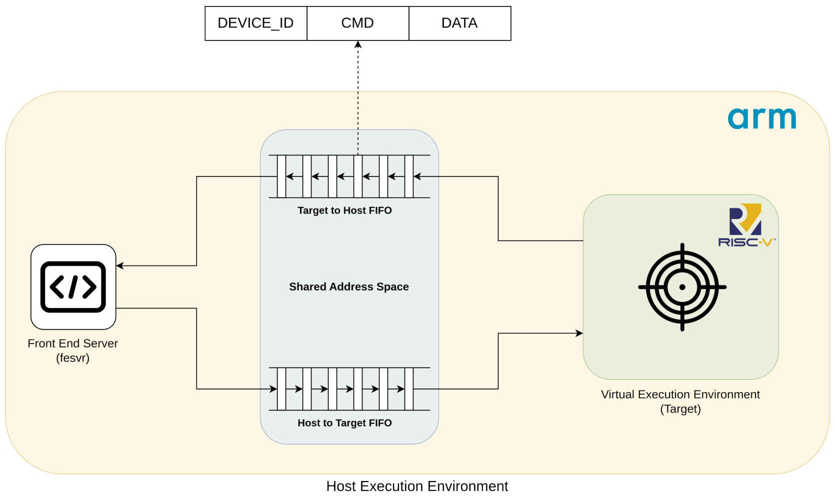


Figure 1.3.2: Επικοινωνία με χρήση της HTIF

1.3.4 Μετρητές Απόδοσης

Η αρχιτεκτονική του RocketChip αλλά και γενικότερα του RISC-V περιέχει μία σειρά από καταχωρητές γενικές χρήσης, που τους ονομάζει CSR (Control and Status Registers). Οι CSR χρησιμοποιούνται για να μπορεί να υπάρχει μία εποπτία σε διάφορα σημεία του συστήματος. Οι μετρητές απόδοσης είναι ένα υποσύνολο αυτών των καταχωρητών και έτσι μπορούν να γράφονται και να διαβάζονται με ίδιες εντολές μηχανής όπως και οι υπόλοιποι CSR. To RocketChip διαθέτει 29 γενικής χρήσης μετρητές οι οποίοι για να γίνουν configure πρέπει να γραφτεί στον register mhpmeventN ο κωδικός του προς μέτρηση γεγονότος και έτσι ο μετρητής mhpmcounterN θα ξεκινήσει να μετράει τον αριθμό των εμφανίσεων του γεγονότος. Η ανάθεση του γεγονότων μπορεί να γίνει μόνο σε λειτουργία μηχανής ενώ η ανάγνωση του μπορεί να γίνει σε όλες τις λειτουργίες. Στο περιβάλλον μας θα μετρήσουμε γεγονότα που αφορούν εντολές, την αρχιτεκτονική και τις μνήμες.

1.3.5 Πλακέτα Ανάπτυξης ZC706

Η πλακέτα ανάπτυξης ZC706 βασίζεται στο SoC Zynq το οποίο περιλαμβάνει ένα επεξεργαστή ARM Cortex-A9 και ένα FPGA. Εκτός από την αξιοποίησης του επεξεργαστή του συστήματος για το περιβάλλον μας και του AXI bus[30] που προσφέρεται σε όλες τις συσκευές τις Xilinx, ένας από από τους σημαντικότερους λόγους χρήσης αυτής της πλατφόρμας είναι η δυνατότητα μέτρησης της κατανάλωσης ισχύος του FPGA. Η ZC706 περιλαμβάνει τον μικροελεγκτή **UCD90120A**, ο οποίος είναι υπεύθυνος για την μέτρηση 5 ενεργειακών καναλιών της πλακέτας και την επικοινωνία με το SoC μέσω πρωτοκόλλου I2C για την αποστολή των δεδομένων μέτρησης.

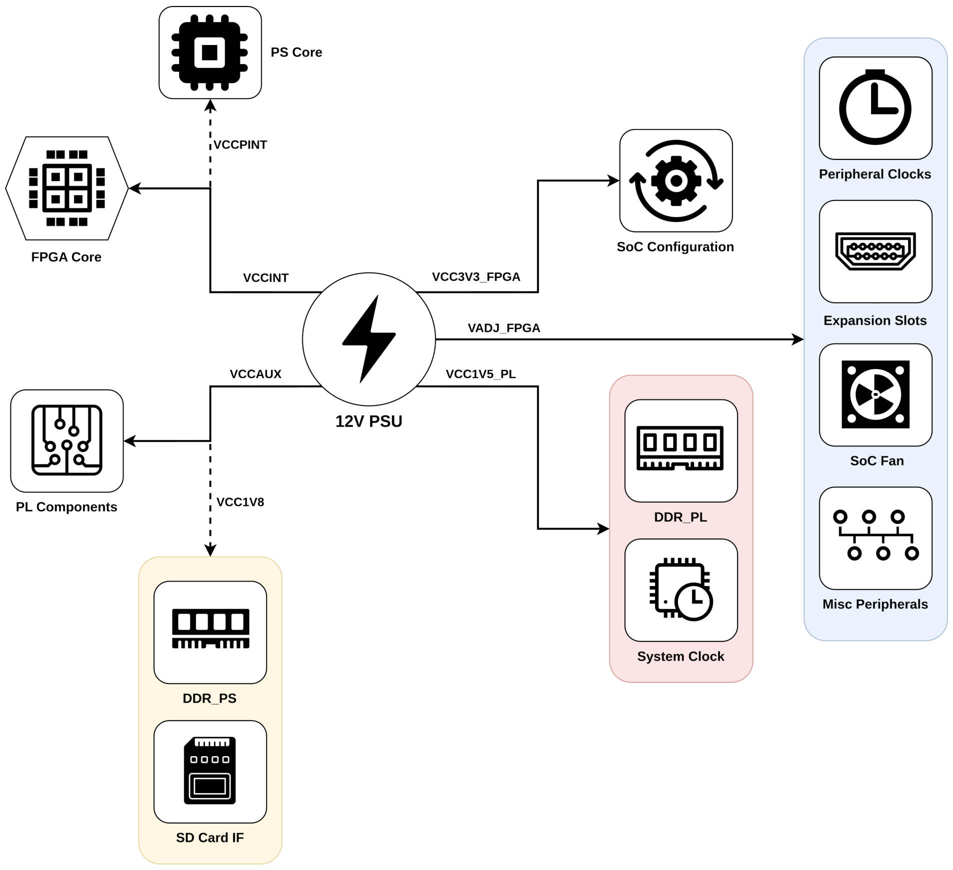


Figure 1.3.3: Κανάλια Ενέργειας στην Πλακέτα ZC706

Στην εικόνα 1.3.3 φαίνονται τα κανάλια ενέργειας στις περιοχές της πλακέτας ZC706.

1.3.6 Συσχέτιση Δεδομένων

Για την συσχέτιση των δεδομένων ενέργειας με των δεδομένων απόδοσης έγινε η χρήση του **συντελεστή συσχέτισης Spearman** καθώς δεν μας ενδιαφέρει μόνο η γραμμική σχέση ανάμεσα στους δύο τύπους δεδομένων. Επίσης για την συσχέτιση και ως προς τον χρόνο, παρατηρήσαμε ότι είναι χρήσιμο να γίνει και **διασταυρωμένη συσχέτιση** των δεδομένων, δηλαδή μετακίνηση τους στον χρόνο και υπολογισμός της νέας συσχέτισής τους. Τέλος, είναι χρήσιμο να αναφερθεί ότι για την επιβεβαίωση των δεδομένων μας εφαρμόσαμε το **κατώφλι σημαντικότητας p-value**[31] για την απόρριψη της μηδενικής υπόθεσης.

1.3.7 Αφαίρεση Θορύβου

Τα δεδομένα που συνήθως προέρχονται από μετρήσεις τάσης και ρεύματος εισάγουν κάποια ασυνήθιστα ή τυχαία συμπεριφορά, η οποία ονομάζεται θόρυβος. Αυτή η ανωμαλία των δεδομένων έχει πολλές πηγές, όπως η μέτρηση ο εξοπλισμός, η παροχή ρεύματος, το περιβάλλον και η ίδια η διαδικασία μέτρησης και αν η μέτρηση διάστημα μέτρησης είναι μικρό, τότε ο αντίκτυπος του θορύβου στα δεδομένα είναι πιο σημαντικός. Για το σκοπό αυτό, υπάρχουν στατιστικοί αλγόριθμοι που παρέχουν έναν τρόπο μείωσης του θορύβου του συνόλου δεδομένων, διατηρώντας την κύρια συμπεριφορά (π.χ. η κατανομή) όσο το δυνατόν πιο ανέπαφη. Δύο από τους αλγορίθμους που υπήρχαν στο out περιβάλλον αξιολόγησης ήταν ο **κυλιόμενος μέσος όρος** και το **Γκαουσιανό φίλτρο**.

1.4 Πειραματικό Περιβάλλον

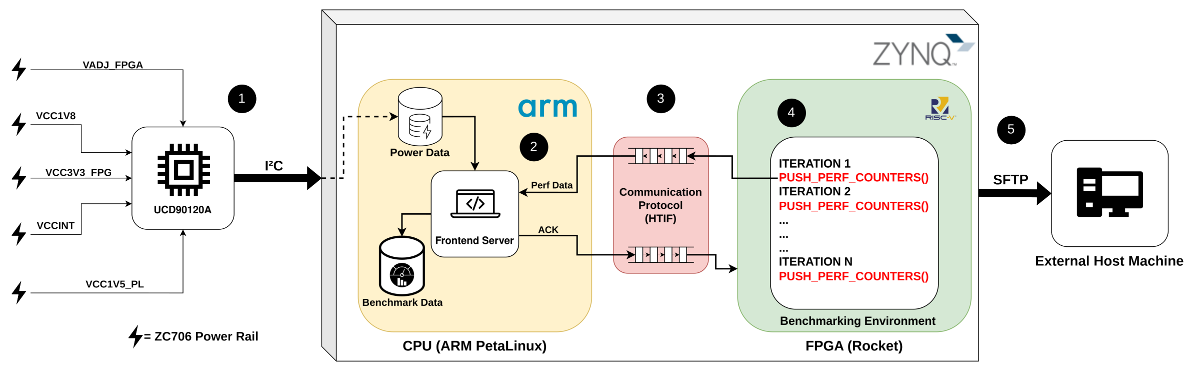


Figure 1.4.1: Πειραματικό Περιβάλλον

Στο πειραματικό μας περιβάλλον (εικόνα 1.4.1) έχουμε προσαρμόσει το RocketChip σύστημα στην πλακέτα ανάπτυξης μέσω wrapper και με την χρήση του λογισμικού Vivado HLx Editions 2018.3, με επικοινωνία με τον front-end server μέσω AXI bus και με ρολόι χρονισμού το οποίο παράγεται από το αρχικό ρολόι του συστήματος και διαιρείται μέσω του component MMCME2. Όλα τα εκτελέσιμα αρχεία που τρέχουν στον ARM εκτελούνται σε περιβάλλον PetaLinux και έχουν μεταγλωττιστεί με τον **arm-xilinx-linux-gnueabi** ενώ όλα τα εκτελέσιμα στον RocketChip έχουν μεταγλωττιστεί με τον **riscv64-unknown-elf**.

Κατά την διάρκεια των μετρήσεων τα δεδομένα από την εκτέλεση benchmark στο RocketChip (στάδιο **4**) μεταφέρονται μέσω μίας τροποποιημένης εκδοχής της HTIF (η οποία παρίσταται στην εικόνα 1.4.2) στον front-end server (στάδιο **3**), όπου εκεί μετρήσεις ενέργειας που ήδη έχουν πραγματοποιηθεί από και έχουν μεταφερθεί από τον μικροελεγκτή UCD90120A (στάδιο **1**) συνδέονται με τις λαμβανόμενες μετρήσεις απόδοσης και παράγεται ένα νέο data point στο σύνολο δεδομένων (στάδιο **2**). Τέλος τα παραγόμενα δεδομένα μεταφέρονται μέσω πρωτοκόλλου SFTP σε εξωτερικο σύστημα για την ανάλυσή τους (στάδιο **5**). Όλη η ανάλυση των δεδομένων έγινε με χρήση **Python 3.8.10**

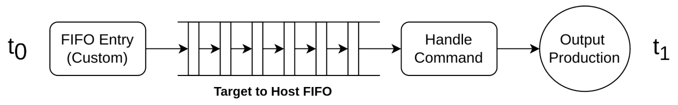


Figure 1.4.2: Προσαρμοσμένη Επικοινωνία με HTIF

Για το benchmarking των δεδομένων έγινε χρήση δύο διαφορετικών βιβλιοθηκών. Η πρώτη είναι η σουίτα **riscv-tests** η οποία περιέχει μικρά benchmark τα οποία τρέχουν έναν συγκεκριμένο αλγόριθμο. Η δεύτερη είναι η σουίτα benchmark **Coremark** v1.0 η οποία είναι προσαρμοσμένη για τις RISC-V αρχιτεκτονικές, στην οποία έχουμε διαχωρίσει 4 περιοχές μέτρησης οι οποίες αφορούν το mergesort, έυρεση σε λίστα, crc και πολλαπλασιασμό πινάκων.

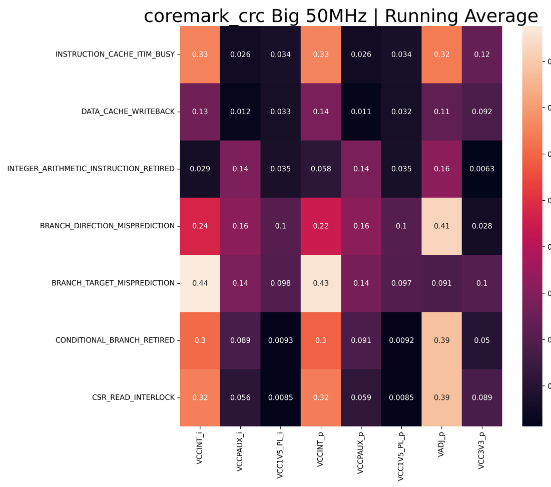
Τέλος, πρέπει να αναφερθεί ότι κατά την διάρκεια χτισίματος του περιβάλλοντος ήρθαμε αντιμέτωποι με διαφορετικά εμπόδια τα οποία κυρίως οφείλονταν στους χαμηλούς χρονισμούς ρολογιών, τον συγχρονισμό των δεδομένων μεταξύ RocketChip και front-end server, καθώς και στις περιορισμένες ιδιότητες του λειτουργικού που τρέχει στον ARM, το PetaLinux, και στο γεγονός ότι πάνω από ένα κανάλι ενέργειας ανήκει στην ίδια μέτρηση (συγχώνευση καναλιών).

1.5 Αποτελέσματα

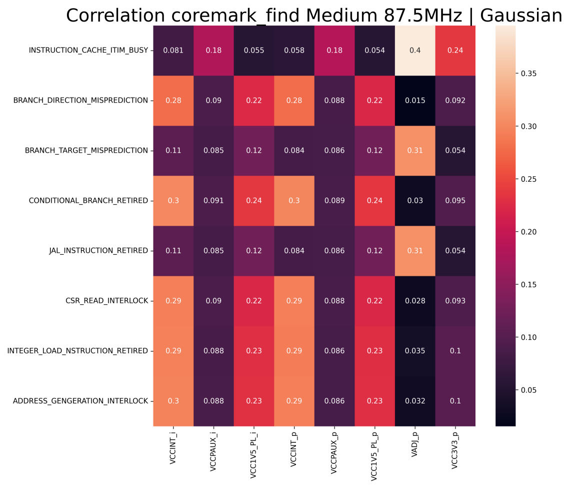
1.5.1 Αξιοποίηση Δομικών Στοιχείων Συστήματος

Από ανάλυση που έγινε σε όλα τα benchmark το αποτέλεσμα που προέκυψε ήταν ότι τα δομικά στοιχεία που όλα τα benchmark αφορούσαν κυρίως την αριθμητική και λογική μονάδα (ALU) με 72.86%, τον μηχανισμό κλάδων (BRANCH) με 10.11% καθώς και τον μηχανισμό παραλληλοποίησης 5-σταδίων (PIPELINE) με 16.28%. Αυτό είχε ως αποτέλεσμα την διαπίστωση ότι για αυτά τα δομικά στοιχεία **οι υλοποιημένοι μετρητές ισχύος αρκούν** για την ανάλυση των δεδομένων. Αντιθέτως για πολλά άλλα δομικά στοιχεία (μνήμες cache, floating-point μονάδες) οι μετρητές απόδοσης **δεν ήταν αρκετοί** για την ανάλυση των δεδομένων, και ίσως υπάρχει η **ανάγκη κατασκευής καινούργιων μετρητών** συμβατούς αποκλειστικά με τα στοιχεία που υπο-αντιπροσωπεύονται.

1.5.2 Στατική (Τυπική) Συσχέτιση



(a) Παράδειγμα Συσχέτισης Κυλιόμενου Μέσου Όρου



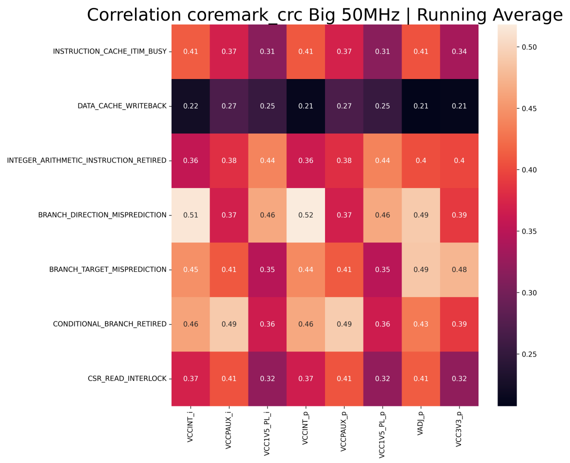
(b) Παράδειγμα Συσχέτισης Γκαουσιανού Φίλτρου

Figure 1.5.1: Παραδείγματα Στατικών Συσχετίσεων

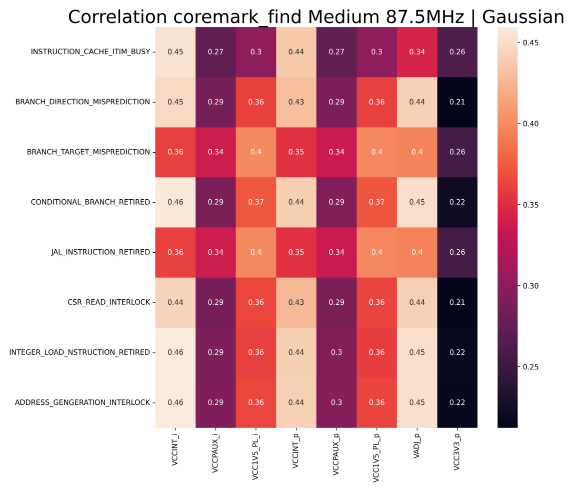
Στην εικόνα 1.5.1 παρατηρούμε παραδείγματα των στατικών συσχετίσεων για δύο συνδυασμούς συχνότητας-συστήματος καθώς και για τους δύο αλγόριθμους αφαίρεσης θορύβου.

Παρατηρώντας τα παραγόμενα αποτελέσματα μπορούμε να δούμε ότι το μοντέλο τείνει να είναι **πιο συγκεκριμένο για την εφαρμογή** όταν τα παρεχόμενα δεδομένα βασίζονται στους ήδη υλοποιημένους (παρέχονται από το ίδιο το RocketChip) μετρητές επιδόσεων και στα κανάλια ισχύος. Όπως μπορούμε να δούμε, υπάρχουν ορισμένες εξαρτήσεις στις συσχετίσεις που παράγονται, αλλά δεν υπάρχουν αλληλεπικαλυπτόμενες εξαρτήσεις σε κάθε σημείο αναφοράς. Έτσι, ακόμη και αν μπορεί να υπάρχουν εξαιρέσεις, δεν μπορούμε να παράγουμε μία τυποποιημένη ροή για όλα τα benchmarks που δοκιμάστηκαν.

1.5.3 Διασταυρωμένη Συσχέτιση



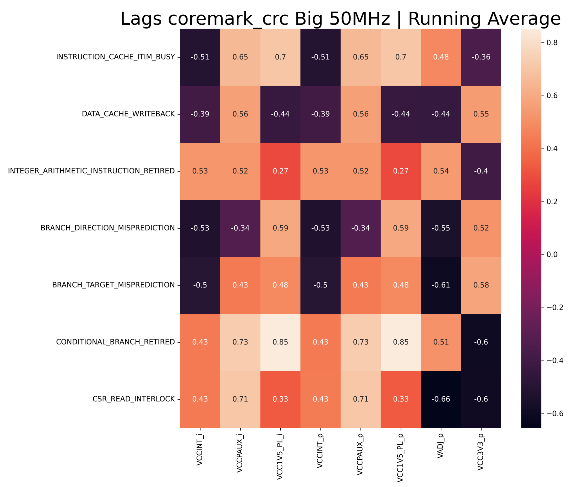
(a) Παράδειγμα Συσχέτισης Κυλιόμενου Μέσου Όρου



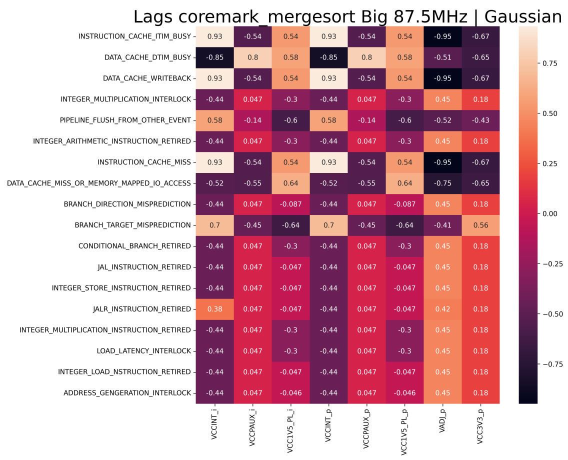
(b) Παράδειγμα Συσχέτισης Γκαουσιανού Φίλτρου

Figure 1.5.2: Παραδείγματα Διασταυρωμένων Συσχετίσεων

Στην εικόνα 1.5.2 φαίνονται ξανά οι διασταυρωμένες συσχετίσεις για δύο συνδυασμούς συχνότηταςσυστήματος καθώς και για τους δύο αλγόριθμους αφαίρεσης θορύβου, οι οποίες αντιπροσωπεύουν την μέγιστη απόλυτη τιμή συσχέτισης σε όλο το χρονικό διάστημα που δοκιμάστηκε η συσχέτιση. Από τα αποτελέσματα που παράχθηκαν, σαν συμπέρασμα μπορούμε να εξάγουμε ότι οι συντελεστές είναι μεγαλύτερη σε όλη την έκταση των αποτελεσμάτων, αλλά για περεταίρω ανάλυση πρέπει να γίνουν γνωστοί και οι χρόνοι μέγιστης συσχέτισης.



(a) Παράδειγμα Συσχέτισης Κυλιόμενου Μέσου Όρου



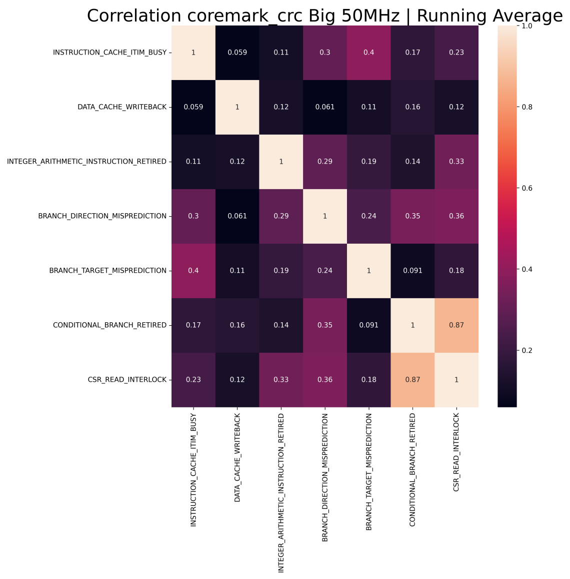
(b) Παράδειγμα Συσχέτισης Γκαουσιανού Φίλτρου

Figure 1.5.3: Παραδείγματα Χρόνων Μέγιστης Συσχέτισης

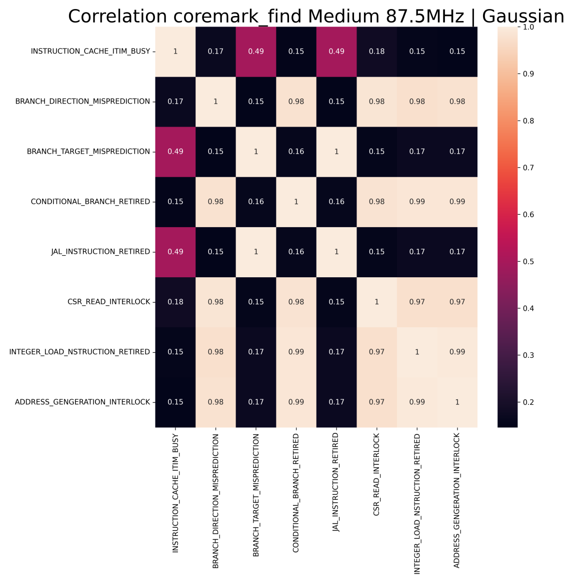
Στην εικόνα 1.5.3 φαίνονται και τα αντίστοιχα ποσοστά μετατόπισης χρόνου ως προς τις μετρήσεις τα οποία υπολογίστηκαν από την εξίσωση τιμή $= \frac{\text{Χρονική Μετατόπιση}}{N_{\text{Μετρήσεων}}}$.

Με την παρατήρηση και των παραγόμενων χρόνων, συμπεραίνουμε ότι στην ανάλυση διασταυρούμενης συσχέτισης ακόμη και αν οι τυποποιημένοι μετρητές που παρέχονται είναι αρκετοί για μια βασική αξιολόγηση χρειάζεται να έχουμε δύο επιπλέον διαδρομές ανάλυσης στη διαδικασία εξαγωγής χαρακτηριστικών. Η πρώτη είναι η ανάλυση χρονικής υστέρησης, η οποία μπορεί να δείξει την εξάρτηση της συσχέτισης στο ιστορικό των γεγονότων και όχι μόνο στην πραγματική στιγμή της μέτρησης. Αυτό σημαίνει ότι εκτός από την εφαρμογή συγκεκριμένο χαρακτήρα μιας πρόβλεψης χρειάζεται επίσης να έχει παράθυρο μνήμης των γεγονότων που συνέβησαν και τη χρονική περιοχή εντός της πιο ενδιαφέρουσας επιλογής του συντελεστή συσχέτισης. Η δεύτερη είναι η ψευδώς θετική ένδειξη, διότι ακόμη και αν ένα συμβάν του μετρητή επιδόσεων έχει υψηλότερο συντελεστή συσχέτισης, δεν σημαίνει απαραίτητα ότι το συμβάν αυτό είναι πιο σημαντικό. Η δεύτερη διαδρομή δεν είναι τόσο καλά καθορισμένη, διότι αγγίζει τα όρια μεταξύ της πρόβλεψης ενέργειας και της ανάλυσης της συμπεριφοράς του πυρήνα.

1.5.4 Συσχέτιση Μεταξύ Μετρητών Απόδοσης



(a) Παράδειγμα Συσχέτισης Κυλιόμενου Μέσου Όρου



(b) Παράδειγμα Συσχέτισης Γκαουσιανού Φίλτρου

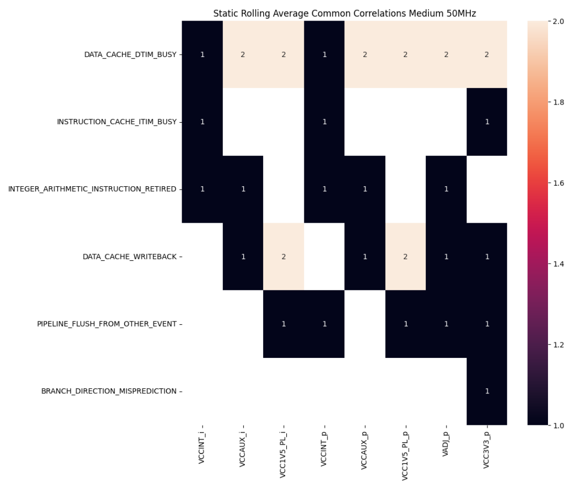
Figure 1.5.4: Παραδείγματα Συσχέτισης Μετρητών Απόδοσης

Για μία πιο ολοκληρωμένη ανάλυση , πρέπει να γίνει παρατήρηση των μετρητών που παρουσιάζουν παρόμοια συμπεριφορά. Το γεγονός αυτό ωθεί στην παρουσίαση των συσχετίσεων μεταξύ των ίδιων των μετρητών απόδοσης, παραδείγματα των οποίων φαίνονται στην εικόνα ??.

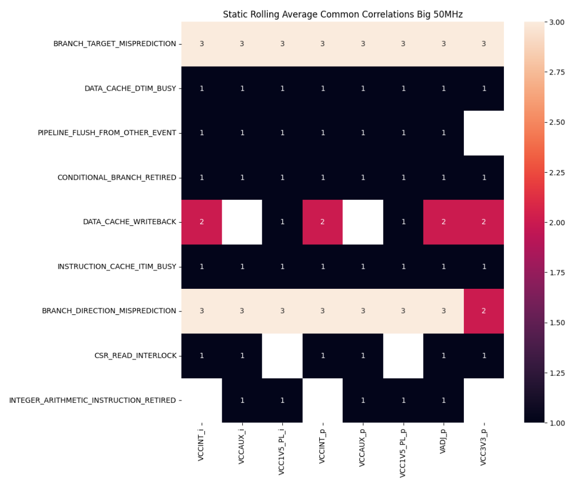
Συμπερασματικά, σε αυτό το τμήμα γίνεται κατανοητό ότι, εάν υπάρχουν μετρητές που συμπεριφέρονται με τον ίδιο τρόπο, η εστίαση της συμπεριφοράς πρέπει να είναι στην ομάδα των μετρητών και όχι στις συμπεριφορές των αντίστοιχων μετρητών. Επίσης, η μεθοδολογία αποθορυβοποίησης του φίλτρου Gauss παράγει περισσότερες ομάδες συσχετιζόμενων μετρητών, οι οποίες μπορούν να ακυρώσουν την ακεραιότητα των αποτελεσμάτων και να παράξουν ψευδώς θετικά αποτελέσματα στη ροή της εξαγωγής χαρακτηριστικών.

1.5.5 Αποτελέσματα Συσχέτισης στο Σύνολο των Benchmark

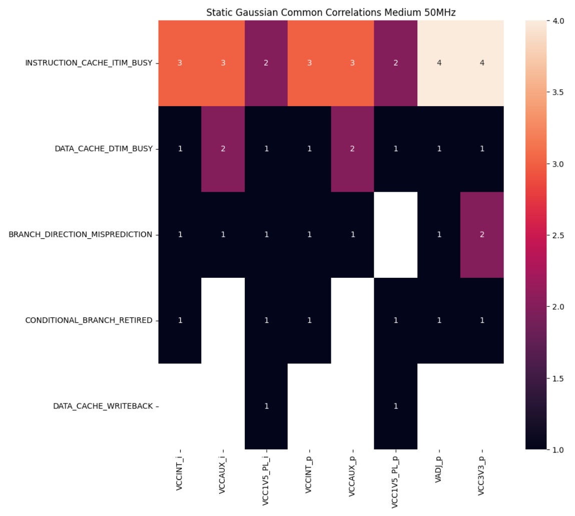
Στατική Συσχέτιση



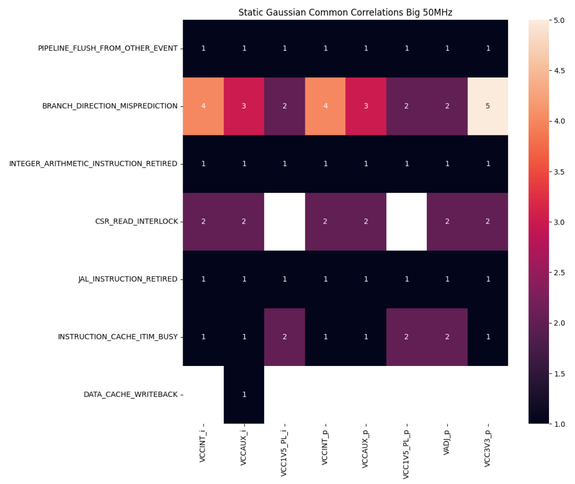
(a) Κυλιόμενος Μέσος Όρος Mid 50MHz



(b) Κυλιόμενος Μέσος Όρος Big 50MHz



(c) Γκαουσιανό Φίλτρο Mid 50MHz



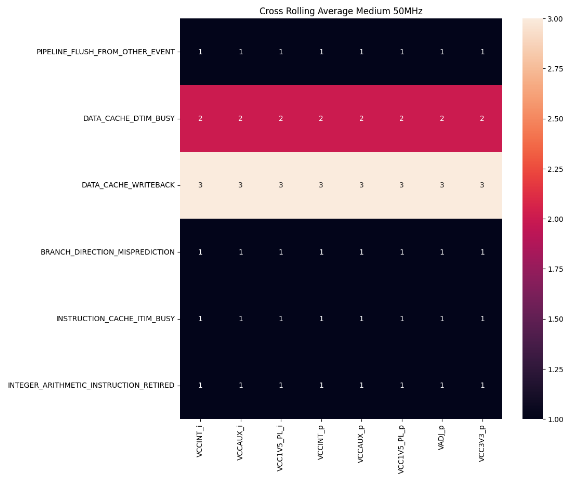
(d) Γκαουσιανό Φίλτρο Big 50MHz

Figure 1.5.5: Παραδείγματα Στατικής Συσχέτισης

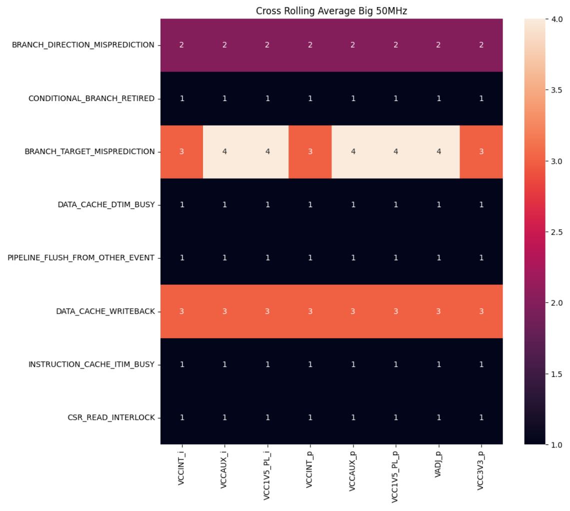
Στην εικόνα 1.5.5 φαίνονται αποτελέσματα από την ανάλυση μεταξύ όλων των benchmark για κάθε διαφορετικό συνδυασμό παραμέτρων. Οι αριθμοί που φαίνονται ανταποκρίνονται το ποσοστό στα 10 benchmark στα οποία συναντάμε τον ίδιο μετρητή απόδοσης μεταξύ του κορυφαίους 5 κάθε benchmark , οι οποίοι είναι πάνω από **κατώφλι σημαντικότητας** για $\alpha = 0.95$ και **δεν έχουν μεταξύ τους συσχέτιση** (υπάρχει ένα κατώφλι μεταξύ τους συσχέτισης **0.5**).

Από τα αποτελέσματα που παράγονται από αυτήν την ανάλυση φαίνεται ότι παρότι σε συγκεκριμένα σημεία των μετρήσεων μας **υπάρχει κάποια ομοιογένεια** (πχ στους μετρητές που αφορούν την L1 cache δεδομένων) στην γενική εικόνα **δεν μπορεί να είναι application agnostic** μία πρόβλεψη πάνω σε αυτά τα δεδομένα.

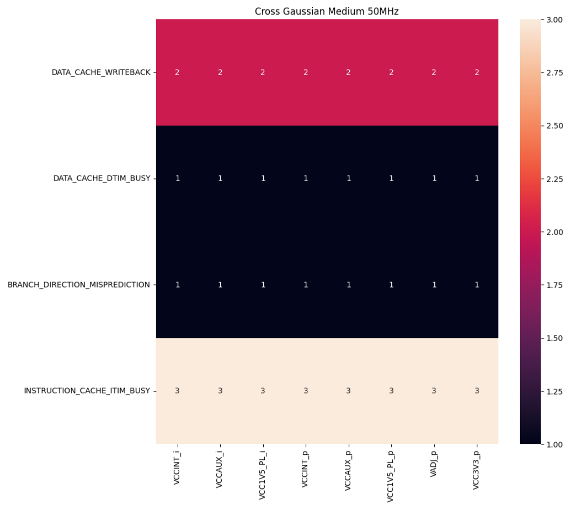
Διασταυρωμένη Συσχέτιση



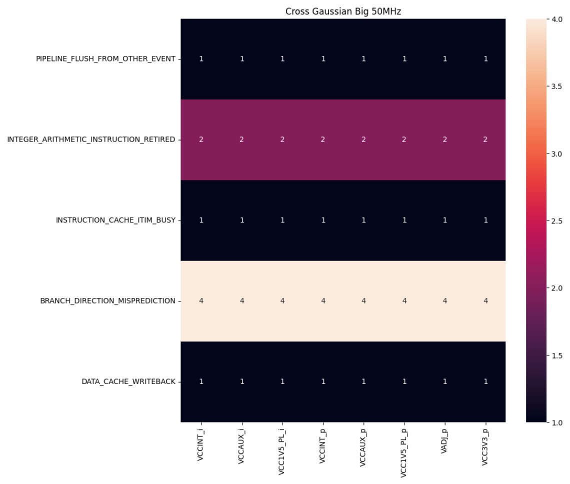
(a) Κυλιόμενος Μέσος Όρος Mid 50MHz



(b) Κυλιόμενος Μέσος Όρος Big 50MHz



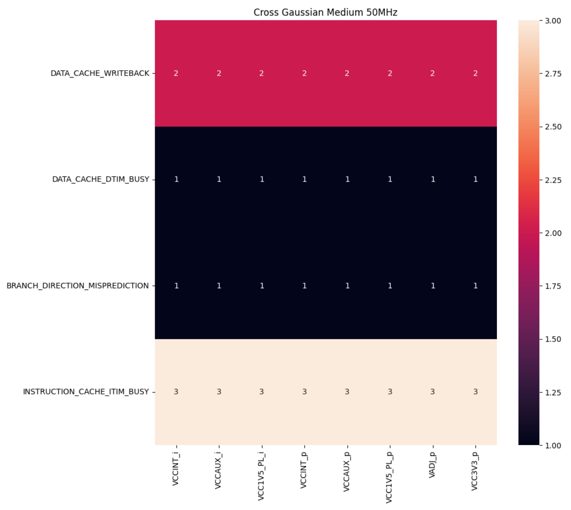
(c) Γκαουσιανό Φίλτρο Mid 50MHz



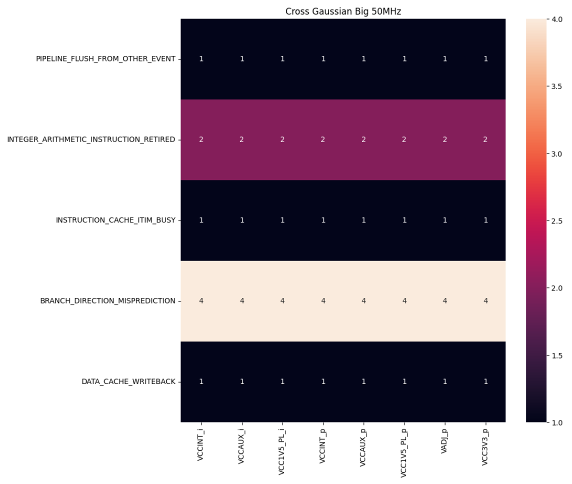
(d) Γκαουσιανό Φίλτρο Big 50MHz

Figure 1.5.6: Παραδείγματα Διασταυρωμένης Συσχέτισης

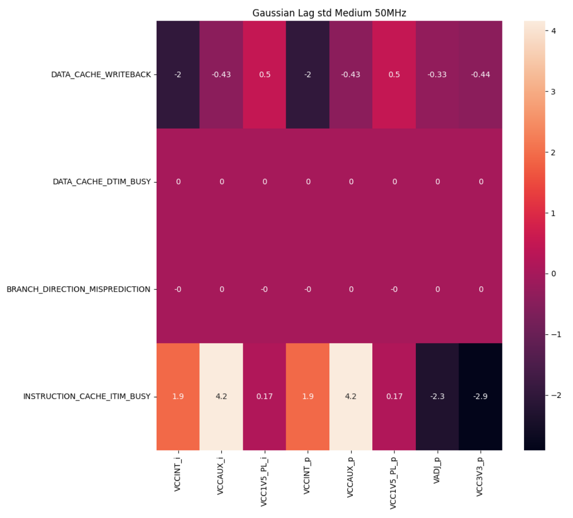
Στην εικόνα 1.5.6 φαίνονται αποτελέσματα από την ανάλυση μεταξύ όλων των benchmark για την διασταυρωμένη συσχέτιση, με τις ίδιες παραμέτρους με την στατική. Από την απλή παρατήρηση των αποτελεσμάτων φαίνεται ότι υπάρχουν ίδια συμπεράσματα σε σχέση και με την στατική ανάλυση, ωστόσο τα δεδομένα που παρουσιάζονται έχουν προκύψει από μέγιστες τιμές συσχέτισης, οπότε για την πλήρη εικόνα πρέπει να εξεταστούν και οι χρόνοι εμφάνισης των μέγιστων.



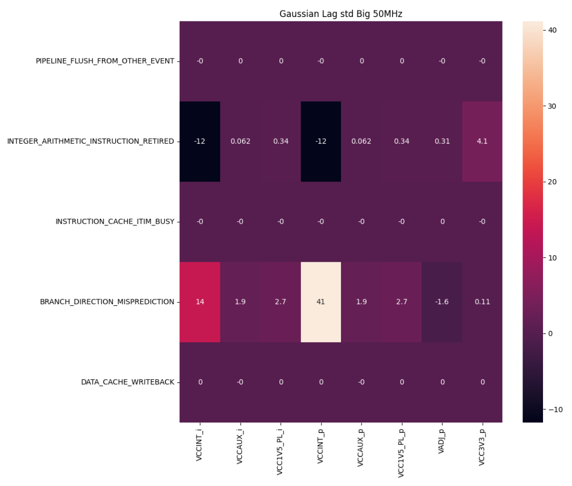
(a) Κυλιόμενος Μέσος Mid 50MHz



(b) Κυλιόμενος Μέσος Big 50MHz



(c) Γκαουσιανό Φίλτρο Mid 50MHz



(d) Γκαουσιανό Φίλτρο Big 50MHz

Figure 1.5.7: Παραδείγματα Ανάλυσης Χρόνων

Με την χρονική ανάλυση ως επόμενο βήμα, στην εικόνα 1.5.7 παρουσιάζονται οι συντελεστής διακύμανσης των χρονικών μεταβολών που είναι υπολογισμένοι ως $\frac{|\sigma|}{\mu}$. Έγινε χρήση της απόλυτης τιμής στην διασπορά για να υπάρχει σωστή αναπαράσταση του αν τα δεδομένα μας είναι οδηγούμενα από το παρελθόν ή το μέλλον.

Συμπερασματικά, από αυτά τα αποτελέσματα μπορούμε να δούμε ότι **δεν υπάρχει σε όλους τους δείκτες αναφοράς συνεπής συμπεριφορά** στα διασταυρωμένα συσχετισμένα δεδομένα, το οποίο είναι ένα λογικό αποτέλεσμα λόγω του γεγονότος ότι τα benchmarks στοχεύουν σε διαφορετικές ροές εκτέλεσης και ακόμη και αν αυτές οι ροές είναι παρόμοιες, δεν συμβαίνουν στον ίδιο χρόνο πλαίσιο. Αυτό το αποτέλεσμα ήταν αναμενόμενο και αποτελεί επίσης επικύρωση της μεθοδολογίας μας και σημείο εκκίνησης της απόδειξης ότι οι προβλέψεις δεν πρέπει να είναι **application-agnostic**.

1.6 Συμπεράσματα - Μελλοντική Εργασία

Από την παραπάνω ανάλυση προέκυψε ένα σύστημα μετρήσεων κοντά στο υλικό, καθώς και μία εκτενή ανάλυση στην συσχέτιση απόδοσης ενός RISC-V based συστήματος (RocketChip) από την οποία τα αποτελέσματα ήταν τα εξής:

Ερώτηση1 : Μπορεί το RocketChip να παρέχει αρκετούς μετρητές επιδόσεων για να χρησιμοποιηθεί ως είσοδος;

Απάντηση: Ναι, αν δοκιμάσουμε σε συγκεκριμένα building blocks, αλλά περισσότεροι (προσαρμοσμένοι) μετρητές θα παράγουν πιο ποιοτικά αποτελέσματα.

Ερώτηση2 : Μπορεί η ανάλυση που πρόκειται να εκτελέσουμε να οδηγήσει σε πρόβλεψη με βάση την εφαρμογή;

Απάντηση: Όχι βάσει αυτής της ανάλυσης, επειδή παρατηρήσαμε σημαντικές αλλαγές στη συμπεριφορά σε ορισμένα benchmarks.

Ερώτηση3 : Θα πρέπει να έχουμε μνήμη για μια ακριβή πρόβλεψη;

Απάντηση: Ναι, επειδή μπορούμε να δούμε ότι στο τμήμα της διασταυρούμενης συσχέτισης έχουμε σημαντικές αλλαγές τιμών και, ακόμη και αν αυτή η συμπεριφορά είναι λανθασμένη, η μνήμη θα μας βοηθήσει να διαγράψουμε αυτές τις περιπτώσεις.

Ερώτηση4 : Είναι τα δεδομένα συσχέτισης ευαίσθητα στις αλλαγές διαμόρφωσης;

Απάντηση: Ναι, καθώς μπορούμε να δούμε στα αποτελέσματα Across-Configurations.

Ερώτηση5 : Μπορεί μια προεπεξεργασία δεδομένων με βάση την αποθορυβοποίηση να οδηγήσει σε καλύτερη ανάλυση δεδομένων ισχύος/απόδοσης;

Απάντηση: Ο κυλιόμενος μέσος όρος έδωσε πιο συγκεκριμένα και "ξεκάθαρα" αποτελέσματα. Από την άλλη πλευρά το Gaussian Filter έδωσε πιο ομαλά μοτίβο το οποίο μπορεί εύκολα να οδηγήσει σε περισσότερες ψευδώς θετικές περιπτώσεις.

Η παραγόμενη έρευνα μελλοντικά μπορεί να επεκταθεί με τους εξής τρόπους:

- Απόκτηση δεδομένων ενεργοποίησης από περισσότερα building block και διασταυρούμενη επαλήθευση της υπόθεσής μας.
- Αξιολόγηση του συνόλου δεδομένων χρησιμοποιώντας τεχνικές DVFS.
- Σχεδιασμός συγκεκριμένων μετρητών επιδόσεων σε επίπεδο υλικού (στόχος σε συγκεκριμένα building blocks).
- Προσαρμογή αυτής της τεχνικής σε πυρήνες εκτός RocketChip.

Chapter 2

Introduction

The evolution of neural networks[1] and the emphasis on energy conservation in research in recent years combined with the wider energy crisis the planet is currently facing have made power consumption monitoring urgently necessary.It is advised that efforts be made in various industry contexts to process and correlate power consumption in various accelerator components in order to parameterize a model that can assess the best utilization for each application and take a step large enough to take advantage of both machine learning techniques and dynamic power source parameterization as well as various clock operating ranges (DVFS)[2].

Along with this direction, new microarchitecture technologies are being developed focusing in energy efficiency, performance and modularity having in mind all the applications mentioned above. The RISC-V architecture has been present in the research field and, due to its open source ISA has been modified for several state-of-the-art implementations[3][4][5]. Also the results when compared with several other cores are very promising[6].

The general vision of our research is to try and explore different parameters in a form of performance counters and power monitoring, in order to correctly and completely "understand" the behaviors of different building blocks of a RISC-V based system, then model (based on the best selected parameters from the previous step) the power consumption with the performance counters as an input and finally, optimize this system on runtime. In this diploma thesis the target is the first step of the vision explained , trying to make an **adaptive** and **open-source** system that can monitor the power consumption of the RocketChip RISC-V SoC, emulated in an ZC706 FPGA board, using the performance counters given by the processor.

The analysis presented in our research can also come as a form of answering the following research questions:

RQ1: Can RocketChip Provide Enough Performance Counters to be Used as Input?

RQ2: Can the Analysis we are Going to Perform Lead to an Application-Agnostic Prediction?

RQ3: Should we Have Memory for an Accurate Prediction?

RQ4: Are the Correlation Data Sensitive to Configuration Changes?

RQ5: Can a denoising-based data Preprocessing Lead to Better Power/Performance Data Analysis?

In order to answer the above, this exploration will include different frequencies and different RocketChip (targeted architecture) implementations, in order to observe different behaviors and make this parametric feature selection paradigm more general and adaptable to different and more complex architectures, but also will include data analysis based on Running Average and Gaussian Filter denoising

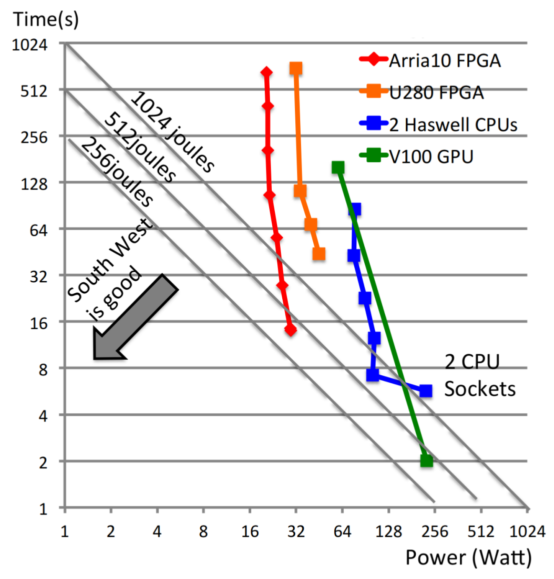


Figure 2.0.2: Versal FPGA energy consumption during BSW benchmarking.

techniques and also a correlation based analysis using Spearman's Rank Correlation Coefficient and cross-correlation.

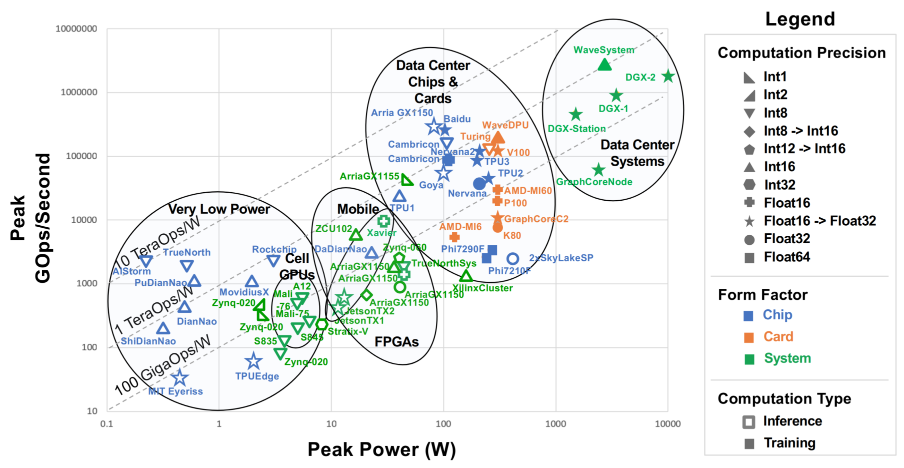


Figure 2.0.1: A complete overview of energy consumption (survey based) [32].

Chapter 3

Related Work

This section will be about feature selection methods and energy prediction techniques in other models and architectures. Because our analysis is based on RISC-V, we will primarily focus on CPU architectures to present the related work in these fields.

Architecture	Reference	Platform	Feature Extraction Type	Model Present	Open Source	Workload Agnostic	Evaluation Environment
	[33]	AMD Opteron	Correlation (OpenMP Data)	Yes	No	No	Physical CPU
	[10]	Intel Atom , Intel Nehalem	Pearson Correlation	Yes	No	No	Physical CPU
	[12]	Intel Core 2 Duo	Average Error and Coverage Techniques	Yes	No	Yes	Physical CPU
	[13]	Intel Xeon X3440	Phase Detection	No	No	No	Grid5000 Platform
x86	[34]	AMD Phenom	Spearman Correlation	Yes	No	No	Physical CPU
	[11]	Intel Core i7	Spearman Correlation (with DVFS Variations)	Yes	No	No	Physical CPU
	[35]	Intel Gainestown HotSniper	Pearson Correlation and Least Angle Regression	Yes	No	Yes	Physical System
	[14]	Intel Core 2 Duo	Power Phase Detection	Yes	No	Undefined	Thinkpad Kernel
	[9]	Intel Xeon version	RAPL inference with perf. events	Yes	Yes	Yes	Physical CPU
	[15]	Cortex A15 (Mobile Devices)	Correlation Metrics and Power Rail Metrics	Yes	No	No	Mobile Devices Evaluation
ARM	[16]	Cortex A15-A7 big.LITTLE	Energy Delay Product	Yes	No	No	Physical Cores
	[17]	Cortex A15-A7 big.LITTLE	CPI-Centric Feature Extraction	Yes	No	Undefined	FPGA Emulation
	[18]	Neoverse A77	RTL Traces From Simulation	Yes	No	Yes	RTL training - Physical Core Inference
	[19]	BOOM Core	Hyperparameter Tweaking on McPAT	Yes	Yes	Yes	Simulated
RISC-V	[20]	PULP-based RI5CY	RTL Activations Trigger	Yes	No	Yes	Simulated
	OurWork	**Rocket Core**	**Spearman Correlation and Cross Correlation**	**No**	**Yes**	**-**	**FPGA emulation**

Table 3.1: Related Work in CPU Architectures

3.1 x86 Architectures

In this field there is the most of the research output. The most famous tool that allowed the real time power analysis and prediction is Intel's RAPL (Running Average Power Limit)[7], using a time window and doing an prediction on the power, it allows to limit the output current to mach this prediction and save power in several building blocks of the execution system (CPU, RAM etc.). RAPL is also used also for energy measuring purposed in code paths[8] and as a data source for further power modeling[9].

Other approaches are also taking advantage of the performance counters of the different CPU architectures and do a feature extraction based on correlation. There are research approaches that try to define a linear behavior across the benchmarking environment (focusing on Pearson correlation)[10] but also to detect non-linear correlations on each system , either by focusing on Spearman correlation[11] or by manually applying non-linear transformations on the data and detecting the correlation drops[35]. Other approaches are using average error and statistical coverage in certain benchmarking criteria in order to define the best feature selection method[12].

In those fields except for the correlation approaches, there are outputs that define DVFS (Dynamic Voltage and Frequency Scaling)[12] and application phase detection[13][14] for more optimal and analytical power prediction (overhead reduction). In our case those implementations are not going to be used due to the limitations set by the close-to-hardware emulated environment and the bare-metal benchmarking.

Finally, a great observation among the most of the techniques presented above is that they are kernel dependant, meaning that there is no direct communication with the hardware during the analysis and evaluation phase. Another critical observation is that each research is triggering a specific CPU model (for example a specific Intel Core i7[35] or an AMD Opteron[33]), meaning that the results are limited almost exclusively to this CPU models only.

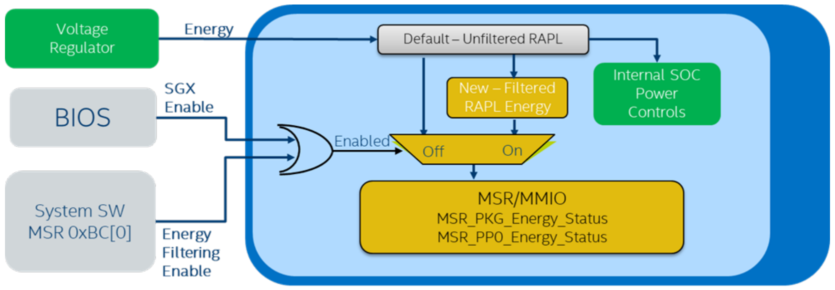


Figure 3.1.1: Intel's RAPL

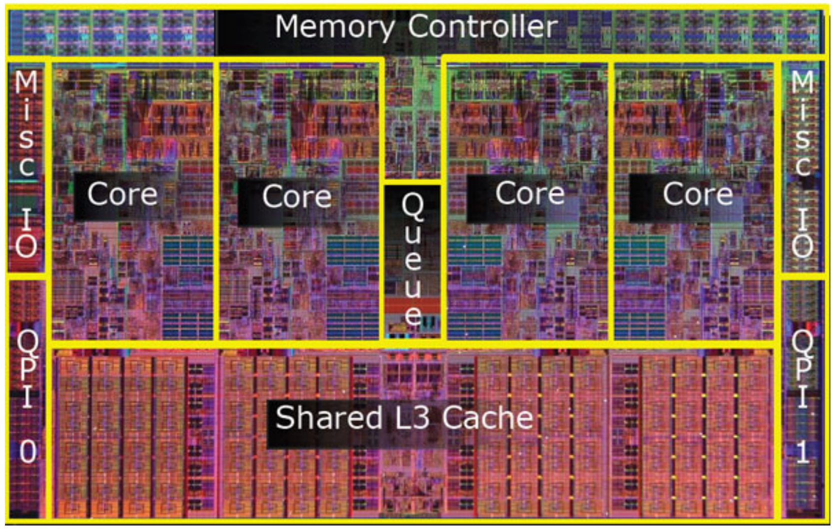


Figure 3.1.2: Intel Core i7

3.2 ARM Architectures

In a similar manner, the ARM perspective did a same correlation based feature extraction from performance counter data, based on the ARM Cortex-A15 and in a mobile device use case[15]. Another approach[16] used the EDP (Energy Delay Product) in order to properly predict energy end extract optimal points for model training, based again in cortex-A15 and cortex-A7 in big.LITTLE configuration. Another big.LITTLE configuration is utilizing an emulated environment (FPGA emulation) and used CPI (Clocks Per Instruction) metrics for extracting features in a certain benchmarking phases of the benchmarking approaches[17]. Finally, related work in ARM architectures also includes RTL traces in order to predict power consumption in a Neoverse A77 core[18].

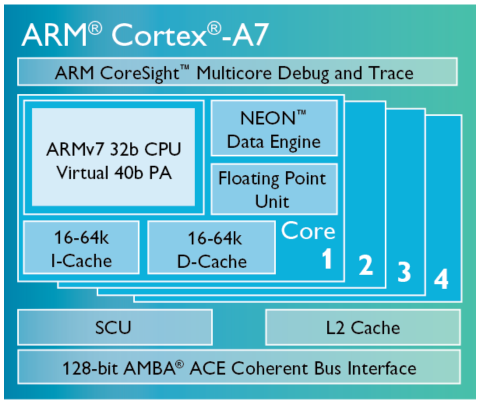

ARM® Cortex®-A7
ARM CoreSight™ Multicore Debug and Trace
ARMv7 32b CPU Virtual 40b PA
NEON™ Data Engine
Floating Point Unit
16-64k I-Cache
16-64k D-Cache
Core 1
2
3
4
SCU
L2 Cache
128-bit AMBA® ACE Coherent Bus Interface

Figure 3.2.1: ARM Cortex-A7

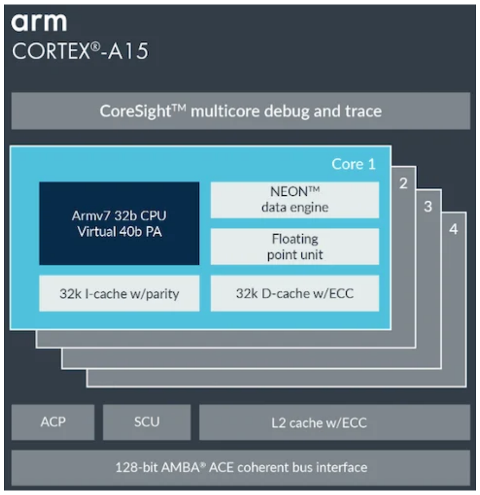


Figure 3.2.2: ARM Cortex-A15

3.3 RISC-V Architectures

According power analysis models in RISC-V based architectures there are 2 major research projects.

The first one is the MCPat-Calib[19], that focuses on calibrating performance monitoring procedures in certain performance counters and benchmarks in the RISC-V BOOM core (the out-of-order implementation of RocketChip) and predicting the power analysis, achieving mean absolute percentage error reduction of 3.64%–6.14%.

The second research project[20] focuses on the data gathered through the use of Nangate 45nm PDK for power analysis of various building blocks in the Synopsys Design compiler of a simulated PULP-based[21] core known as RI5CY[22]. They were able to reduce power by 2.2% over the projected reduction with this modeling setup.

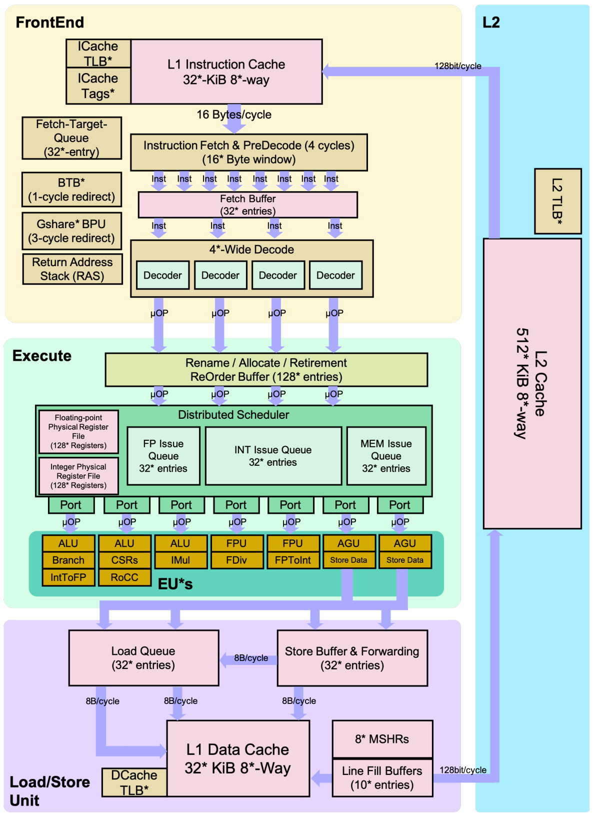

FrontEnd
L2
ICache TLB*
ICache Tags*
L1 Instruction Cache 32*-KiB 8*-way
128bit/cycle
16 Bytes/cycle
Fetch-Target-Queue (32*-entry)
Instruction Fetch & PreDecode (4 cycles) (16* Byte window)
BTB* (1-cycle redirect)
Fetch Buffer (32* entries)
Gshare* BPU (3-cycle redirect)
4*-Wide Decode
Return Address Stack (RAS)
Decoder
L2 TLB*
L2 Cache 512* KiB 8*-way
Execute
Rename / Allocate / Retirement ReOrder Buffer (128* entries)
Distributed Scheduler
Floating-point Physical Register File (128* Registers)
Integer Physical Register File (128* Registers)
FP Issue Queue 32* entries
INT Issue Queue 32* entries
MEM Issue Queue 32* entries
Port
ALU
Branch
IntToFP
CSRs
RoCC
IMul
FPU
FDiv
FPToInt
AGU
Store Data
EU*s
Load Queue (32* entries)
8B/cycle
Store Buffer & Forwarding (32* entries)
L1 Data Cache 32* KiB 8*-Way
8* MSHRs
Line Fill Buffers (10* entries)
DCache TLB*
Load/Store Unit

Figure 3.3.1: BOOM core

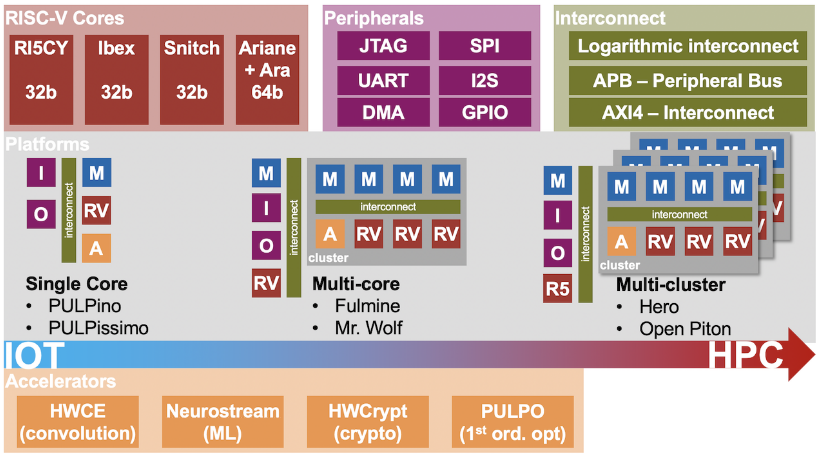


Figure 3.3.2: PULP core

3.4 Non CPU Targeted Architectures

A more different work[23] involved collecting power and performance profiles from 49 CUDA kernels in publicly available programs, specifically the CUDA SDK and the Rodinia benchmark suite. Performance counters were used to gather data on instruction and memory throughputs, which were found to be highly correlated with power consumption. A linear regression model was trained using these performance counters as independent variables and power consumption as the dependent variable. The model's estimation accuracies were evaluated using cross-validation, demonstrating an average error ratio of 4.7%. However, the model faced challenges in accurately estimating power consumption for kernels involving texture reads due to the absence of performance counters for monitoring texture accesses.

Another research[24] focuses on characterizing and modeling the performance and power consumption of the Google Edge TPU, a commercial deep learning accelerator. By extensively exploring over 10,000 DL models with various neural network settings and sizes, the study identifies key factors influencing inference time and power consumption. The research reveals a non-linear relationship between energy/performance and the number of MAC operations, indicating a stepped pattern as computation and DL model size increase. Factors such as on-chip/off-chip memory usage are crucial considerations for accurate performance estimation. The proposed PETET framework leverages machine learning techniques to provide online predictions for the performance and power of Edge TPU based on DL model settings, computational workloads, and memory usage.

Chapter 4

Theoretical Background

Before we proceed with the experimental setup and evaluation of the data, we must offer some basic instruction on the technologies and techniques employed in this thesis. This chapter will primarily explain the fundamentals of the RISC-V and RocketChip architectures, as well as the AMD-Xilinx design flow and the AMD-Xilinx ZC706 development board's power measuring capabilities, as well as all the data processing mechanisms used for our overall evaluation.

4.1 RISC-V ISA

RISC-V[36] is an open standard Instruction Set Architecture (ISA) that embodies the principles of reduced instruction set computing (RISC). It is unique in the landscape of computer architecture for being both open-source and highly modular, allowing for extensive customization and optimization in hardware design. Developed at the University of California, Berkeley, in 2010, RISC-V supports a wide range of implementations, from small, power-efficient microcontrollers to powerful processors capable of driving high-performance computing environments.

Technically, RISC-V defines a set of base integer instructions, and it is extendable through optional instruction sets for floating-point arithmetic, atomic operations, and various other functionalities. The ISA is designed around a load/store architecture, where memory operations are separate from arithmetic and logic operations, a hallmark of RISC principles. This separation facilitates simpler, more efficient instruction pipelines within processors, enhancing performance at lower power costs compared to complex instruction set computing (CISC) architectures.

RISC-V instruction set is divided into several subsets, known as "extensions," which are denoted by single-letter identifiers. This modular approach allows implementers to tailor the architecture to specific application needs by including only the relevant extensions. For example, the base integer ISA (RV32I, RV64I, or RV128I) can be augmented with extensions for single-precision floating-point (F), double-precision floating-point (D), atomic operations (A), and so on.
Additionally, the newest RISC-V ISAs introduce 4 operating modes

- **Machine mode (M-mode)** - the most privileged mode, used during startup and in very specific cases, which require the full core control.
- **Hypervisor mode (H-mode)** - a mode used for virtualization, which is not present in all RISC-V implementations.
- **Supervisor mode (S-mode)** - a less privileged mode, used by the operating system kernel.
- **User mode (U-mode)** - the least privileged mode, used by application software.

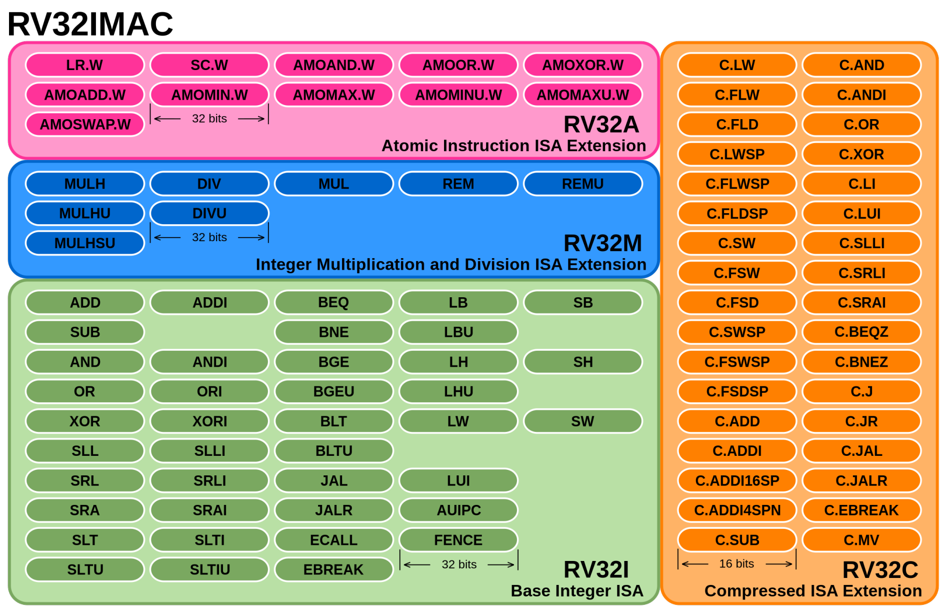


Figure 4.1.1: Basic RISC-V ISA.

4.2 RocketChip

4.2.1 General Overview

RocketChip[25] is a RISC-V based core that was also the first produced hardware description project of a the RISC-V ISA. It is a highly configurable, open-source SoC generator that can integrate a variety of different cores-accelerators. The hardware description of a RocketChip core (Rocket Core) it is written in Chisel[26] , which is an open source high level hardware description language written in Scala and developed at the University of California, Berkeley.
The configuration of the RocketCore is very modular. The end user can configure in a very high level manner the cores, the building blocks and also the core capabilities (there are different cores which are called small,medium and large for example).

Taking advantage this modularity several application specific accelerators can be made with different configurations. All this concepts can also be wrapped in the appropriate communication components (such as AXI interfaces in FPGAs) and be integrated in a more close-to-real environment.

Finally, the RocketChip architecture can also provide reconfigurable extension blocks outside the basic core, such as the L1 and L2 caches, the memory management unit (MMU),the Floating Point Unit(FPU), the debug module, the performance counters and the interrupt controller, as well as communication among all these components.

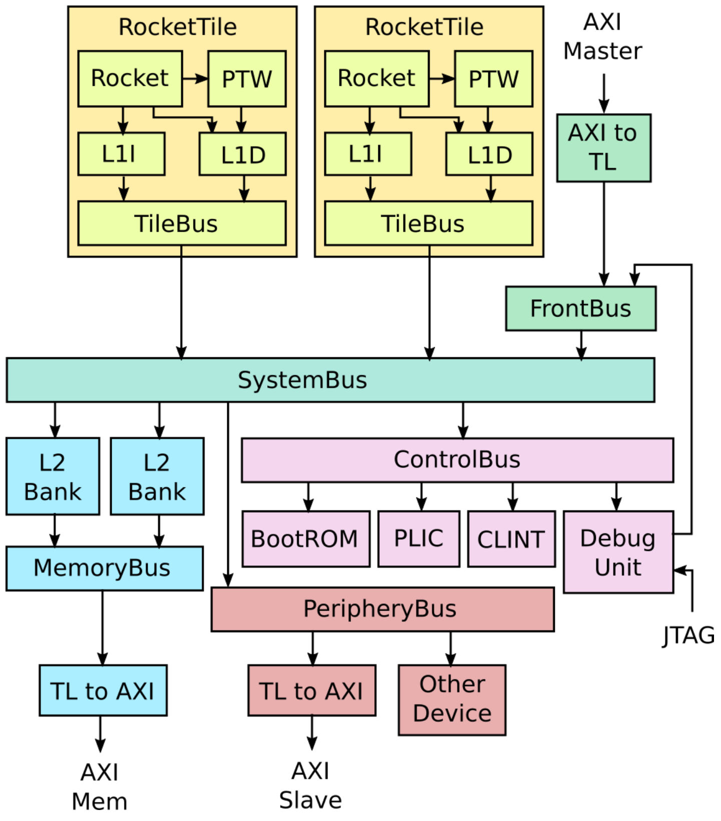


Figure 4.2.1: Example RocketChip architecture.

4.2.2 The RocketCore

Among all the peripheral devices the RocketChip implements and uses, the actual CPU architecture is in the implementation of the **RocketCore** building block . This block is a 5-stage in-order SoC generator that is responsible only for the ISA instruction mapping and the basic functional units, such as ALU, pipelining drivers and register files. The core can be externally configured for 32,64 or even 128 bit architecture mapping and also for some internal configuration (mul unroll factors, memory pages, etc).

4.2.3 The RocketTile

The RocketTile is the next configuration level of the RocketChip (it includes the RocketCore). In the RocketTile you can configure all the aspects of the core, such as the L1 and L2 caches, the MMU, the FPU, the debug module, the performance counters and the interrupt controller. The RocketTile is the basic building block of the RocketChip and it is used to build the final SoC. The RocketTile can be configured to have a single core, a multi-core or even a multi-cluster multi-core configuration. The basic configurations that are included in the RocketTile are the following:

- **BigCore** : A high performance core with 16 KiB, 4-way set-associative instruction and data caches that supports FPU by Default.
- **MediumCore** : A core with smaller 4 KiB direct-mapped caches that does not support FPU by Default.
- **SmallCore** : A low performance core with very limited cache that does not support FPU by Default.

- **TinyCore** : A not so common used core configuration with support for **only 32-bit architecture**.

All the above configurations can be mapped and used in a different configuration that can use any implementation of different caches, cores or even external accelerators.

4.2.4 Front End Server

If the RocketChip is used in a simulated/emulated environment, a communication with the host machine needs to be built in order to have a proper control over the core. This is achieved with the Front End Server[27] (fesvr). This contraption includes an executable which is compiled to run in the host machine architecture and it includes functions and communication protocols that can be used to control the RocketChip core.

A usefull feature of the fesvr is that it can be used to run bare metal code in the RocketChip core. It handles the elf decoding process ass well as the handling of the core state in different execution stages. It can also be used to load a first stage bootloader and boot an OS into the emulated RISC-V system, such as Linux, with support for a mountable file system.

Another feature that is implemented into this project and it is very usefull for the setup of this thesis, is the ability to map system calls from the emulated system to the host machine system calls. This is possible by invoking proper decoding the to-host commands to be interpreted as a higher level system calls by using a custom interface, called HTIF, which will be explained later.This allows for a better and more precise visualization of the core and also for build-in support of multiple system calls, which can support more complex execution environments.

The fesvr project, despite the fact that is used in this thesis due to compatibility issues, is an older version of the RocketChip host-target communication and now it is a part of the newest Spike project[28].

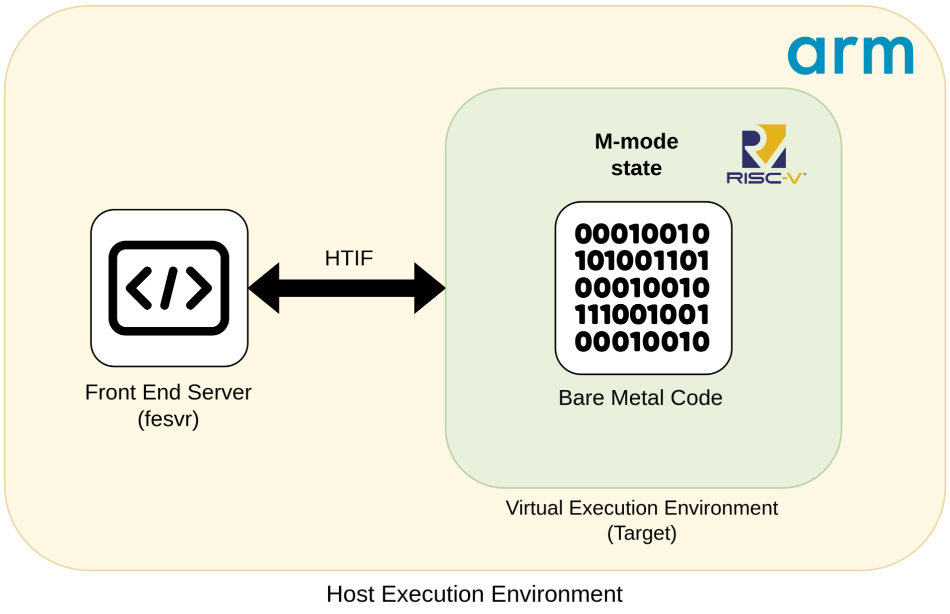


Figure 4.2.2: Front End Server Example Communication.

4.2.5 Proxy Kernel

The Proxy Kernel[29] is a very basic high level kernel written entirely in C and it is used to initialize basic system calls and drivers in order to have a basic environment. The functionality of this kernel is very limited compared to a full fledged bootloader or a Linux based kernel, but it covers enough aspects of core boot up operation that it is very useful for running "less bare metal"* applications with as little overhead as possible.

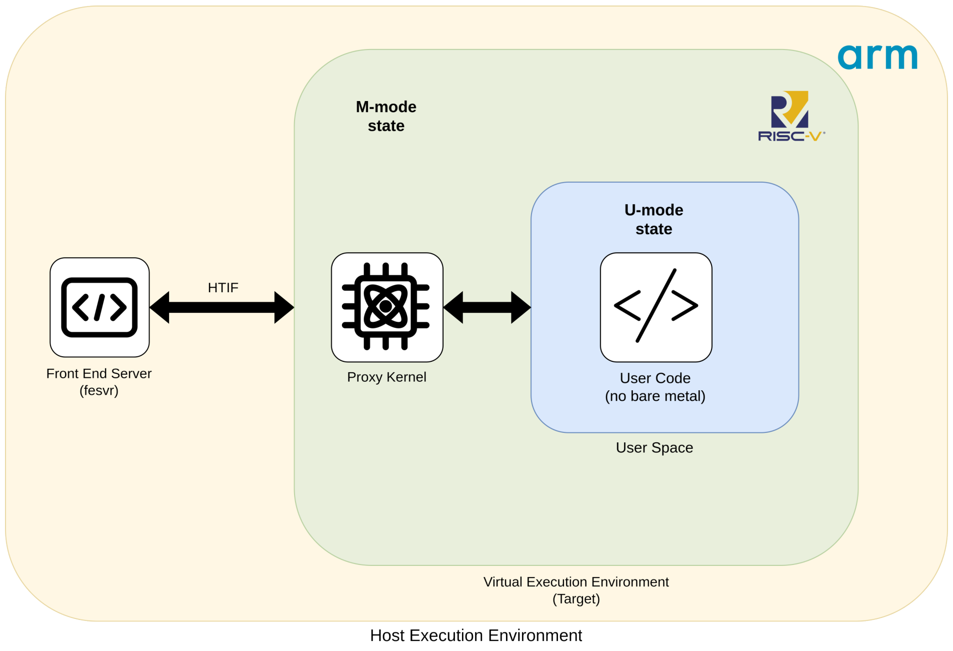

arm
M-mode state
RISC-V
U-mode state
HTIF
Front End Server (fesvr)
Proxy Kernel
User Code (no bare metal)
User Space
Virtual Execution Environment (Target)
Host Execution Environment

Figure 4.2.3: Proxy Kernel Example Communication.

4.2.6 Host Target Interface

For the communication between the host machine and the RocketChip core, a custom implemented interface is used, called HTIF(Host Target Interface).Almost every aspect of the experimental setup is based on this interface and the communication it provides. This interface is used primarily for the below reasons:

- Debugging and state tracing
- System call mapping
- Can be expanded to support more complex communication principals

The communication of the HTIF is being done by using two FIFOs (First In First Out lists), one for the host to target communication and one for the target to host communication. Both of the machines have access to the address space used by these FIFOs and can read and write to them using the appropriate pointers in the high level implementation.

The basic formation of an HTIF instruction have three basic parts.

Device

Each building block of the RocketChip can communicate with the target system. This communication is being established in the high level aspect by assigning an ID to each device and then using it to send the appropriate command to host. In the opposite direction, after the command is done processed, the host will send a response to the target device by using the same ID.

Command

The command consist of the actual operation that is requested from the host by the target. This is usually a system call code but can be expanded with the appropriate implementation in components such as the front end server and the proxy kernel.

Data

The data section is used to send the address space that includes the appropriate arguments that accompany the device and command instruction in order to proper execute the command in the host system (for example for a **printf** command, the target need to send the address of the string that needs to be printed).

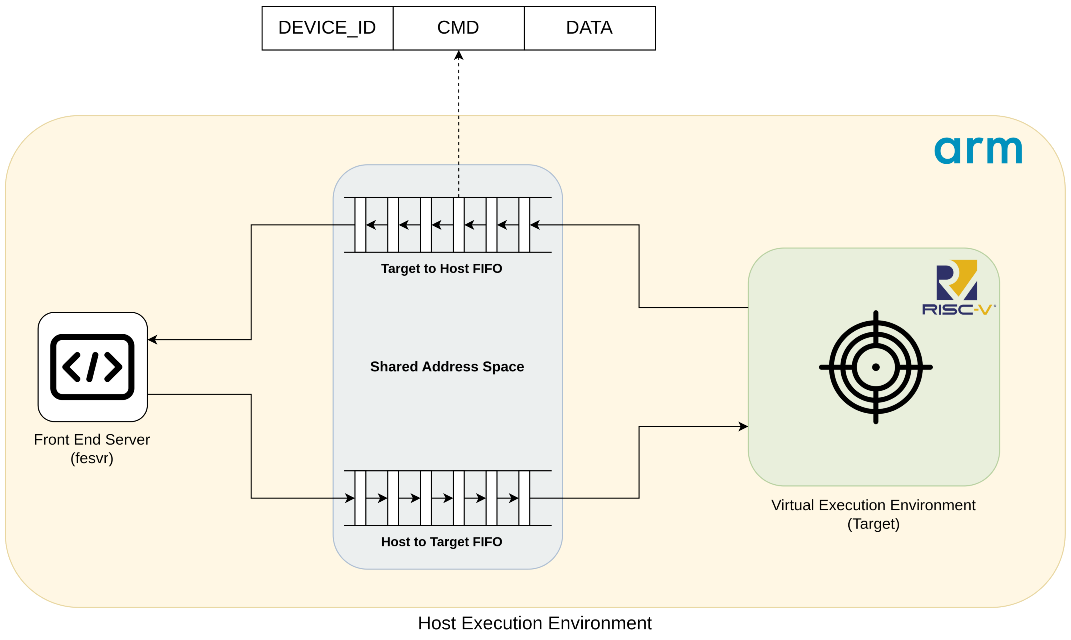


Figure 4.2.4: HTIF Communication

Finally, it is important to be addressed that the HTIF communication is present **only when an emulated environment is used**, because it only provides communication between a host machine.

4.3 Performance Monitoring

4.3.1 Control and Status Register (CSR)

The RISC-V ISA provides a set of CSR (Control and Status Registers) which are used for targeting and monitoring aspects of the core itself, as well as for the protection and privilege system. The total of the CSR registers are 4096 (12-bit space). Few of the CSRs are already defined by the ISA. The functionality of the defined CSRs are outside the scope of this thesis, so we will summarize the CSRs into the following categories:

- General Purpose CSRs (time, cycle, etc.)
- Exception Processing (status registets, interrupt enable, etc.)
- Trap Handling (temp registers, trap cause, etc.)
- Informational (ISA, version, vendor, etc.)
- Building block specific (floating points, cache, etc.)
- Debugging

- **Performance Monitoring**

In this thesis we will only focus int he Performance Monitoring CSRs.

ISA Instruction	Usage
csrrw	Read and Write a CSR
csrrs	Read and Set Selected Bits to 1
csrrc	Read and set selected bits to 0
csrrwi	Read and Write a CSR (from Immediate Value)
csrrsi	Read and Set Selected Bits to 1 (Using Immediate Mask)
csrrci	Read and Set Selected Bits to 0 (Using Immediate Mask)

Table 4.1: ISA Instructions for CSR Accessing

4.3.2 Performance Monitoring CSRs

The performance monitoring is well defined in the RISC-V ISA and can also be expanded accordingly for the specific needs of each core/application with respect to the ISA guidelines. The ISA defines 29 reconfigurable performance counters, which can be used to monitor various aspects of thr core or of its building blocks. Each core can define its own set of performance aspect that need to be measured. Also, if needed, the empty spaces of the CSR definitions can be used for measuring extra performance events.

Each performance counter have 4 CSR registers (3 for the 64-bit ISA) that define it which are presented in Table 4.2.

CSR	Usage	Core State	R/W
mhpmeventN	Event Selector	M-mode	R/W
hpmcounterN	Event Counter	Any	R
mhpmcounterN	Event Counter (M-mode)	M-mode	R/W
hpmcounterNh (32-bit Only)	Upper Half of Counter	Any	R
mhpmcounterNh (32-bit Only)	Upper Half of Counter (M-mode)	M-mode	R/W

Table 4.2: Performance Monitoring CSRs

From the overall configuration the most important registers are the **hpmcounterN** and **mhpmeventN**. The event that is going to be measured must be written to the event CSR and then the counter CSR will measure how many times this event was triggered. Each of the counters mentioned above have a unique register number and all the functionality of selecting an event and reading/writing performance data is defined in the ISA and it is presented in the Table 4.1.

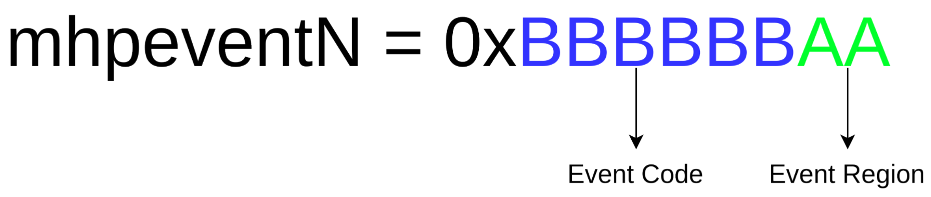


Figure 4.3.1: hpmevent Mapping Formation

4.3.3 Performance Monitoring in RocketChip

RocketChip, following the general ISA template, it is using the same counters and configuration architecture-wise. The difference is in the events that can support and in the way those counter system are being synthesized in the RTL level. The RocketChip has a separate Chisel implemented CSR file that include all the info about the CSRs that are being used , the performance events that can support (the default is 29) and how this events are mapped into the actual hardware implementation. This configuration can be expanded in the pre-synthesis state (Chisel) but for the scope of this thesis only the given events are going to be used because they can cover a mayor part of the system's building blocks and also because due to the emulated environment and not the physical presence of printed hardware, each new event will not be optimal mapped into the specific component and , as a result, the results will only have a valid meaning inside this specific emulated environment. Finally, during the configuration of the core that is going to be synthesized, the number of counters that are going to be used **exists as a parameter**. We will use all the 29 available counter slots.

Event Field	Event Name
Instruction Commit Events	Exception taken
	Integer load instruction retired
	Integer store instruction retired
	Atomic memory operation retired
	System instruction retired
	Integer arithmetic instruction retired
	Conditional branch retired
	JAL instruction retired
	JALR instruction retired
	Integer multiplication instruction retired
	Integer division instruction retired
	Floating-point load instruction retired
	Floating-point store instruction retired
	Floating-point addition retired
	Floating-point multiplication retired
	Floating-point fused multiply-add retired
	Floating-point division or square-root retired
	Other floating-point instruction retired
Microarchitectural Events	Load-use interlock
	Long-latency interlock
	CSR read interlock
	Instruction cache/ITIM busy
	Data cache/DTIM busy
	Branch direction misprediction
	Branch/jump target misprediction
	Pipeline flush from CSR write
	Pipeline flush from other event
	Integer multiplication interlock
	Floating-point interlock
Memory System Events	Instruction cache miss
	Data cache miss or memory-mapped I/O access
	Data cache writeback
	Instruction TLB miss
	Data TLB miss

Table 4.3: Performance Events

In the Table 4.3 are presented all the performance counters that our setup had used during the benchmarking process.

4.4 ZC706 Development Board

For the emulation environment we used the ZC706 development board from AMD-Xilinx. This development board used for our setup due to three crucial components

- **The Xilinx Zynq-7000 SoC**, which is a combination of a dual-core ARM Cortex-A9 processor and a Xilinx 7-series FPGA. This SoC is a perfect fit for our setup, as it allows us to run the Linux operating system on the ARM cores and use the FPGA for the hardware emulation of the RocketChip.
- **The UCD90120A** micro-controller, which is built by Texas Instruments and it measures Voltage, Current and Temperature. The measured data can be obtained through the I2C interface of the ZC706 board.

- **Interconnection Properties** that help the initialization and communication with the ARM core, FPGA and peripheral devices.

The technologies will be explained in the following sections.

4.4.1 Zynq-7000 SoC

The Zynq-7000 SoC is a combination of a dual-core ARM Cortex-A9 processor and a Xilinx 7-series FPGA. This was a very good match to have a higher level control over the emulated environment and handle all the peripheral control and the measured performance data processing away from the RocketChip core, in order to introduce as little overhead as possible to our benchmarks. Also the ARM core is capable of running a custom Linux kernel by Xilinx called PetaLinux, which makes the data processing of the evaluation system more configurable and manageable.

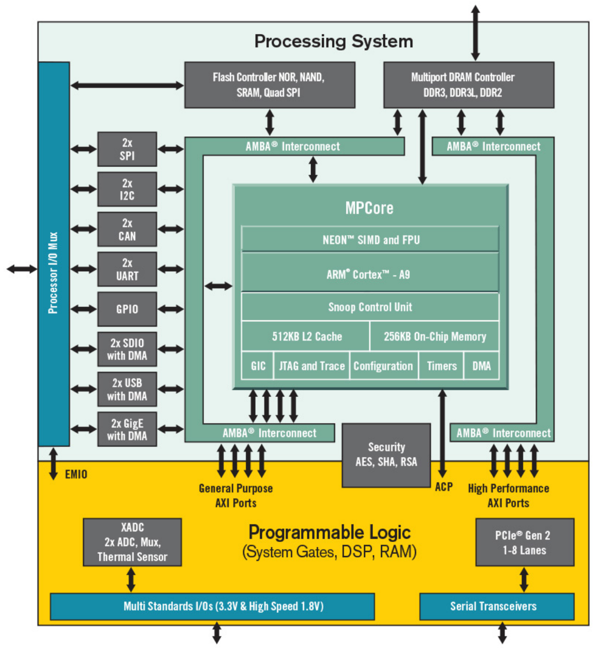


Figure 4.4.1: Zynq 7000 SoC

4.4.2 UCD90120A

The UCD90120A micro-controller offers an I2C communication with the central ARM core of the SoC, transmitting data for the Voltage, Current and Temperature of the board. This communication allows the dynamic monitoring of the power consumption of this specific power rails and thus the power consumption of the whole board. Also there is a specific driver for higher level communication with the Linux kernel, if any.

Another feature of this communication aspect is the ability to also **write Voltage** values to the rails, encouraging techniques like DVFS to be implemented in the high level.

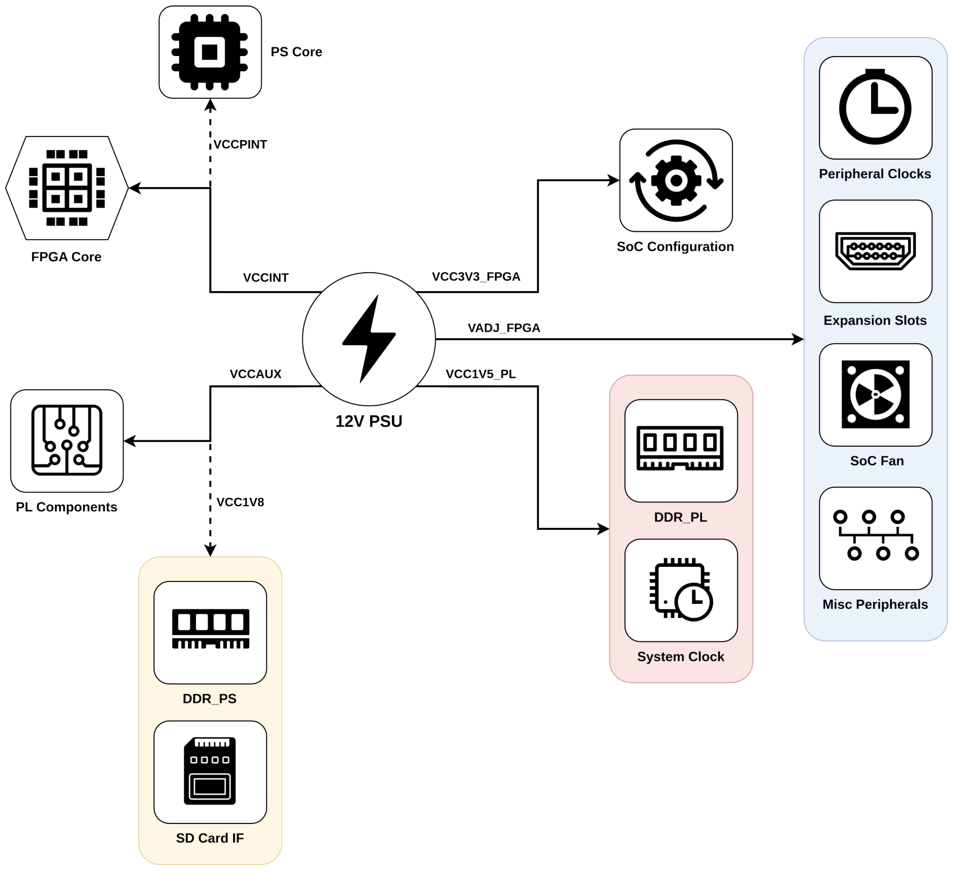


Figure 4.4.2: Power Rails Analysis in ZC706

The rails that are being monitored are presented in the Fig.4.4.2 as well as the corresponding SoC region that this rail powers.

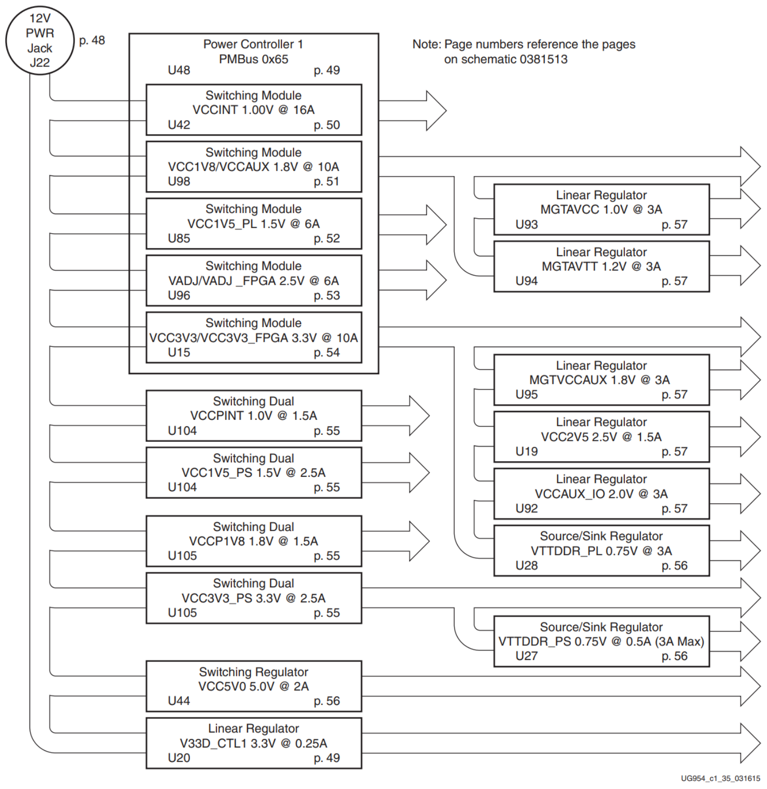

12V PWR Jack J22
p. 48
Power Controller 1 PMBus 0x65 U48 p. 49
Note: Page numbers reference the pages on schematic 0381513
Switching Module VCCINT 1.00V @ 16A U42 p. 50
Switching Module VCC1V8/VCCAUX 1.8V @ 10A U98 p. 51
Switching Module VCC1V5_PL 1.5V @ 6A U85 p. 52
Switching Module VADJ/VADJ _FPGA 2.5V @ 6A U96 p. 53
Switching Module VCC3V3/VCC3V3_FPGA 3.3V @ 10A U15 p. 54
Linear Regulator MGTAVCC 1.0V @ 3A U93 p. 57
Linear Regulator MGTAVTT 1.2V @ 3A U94 p. 57
Linear Regulator MGTVCCAUX 1.8V @ 3A U95 p. 57
Linear Regulator VCC2V5 2.5V @ 1.5A U19 p. 57
Linear Regulator VCCAUX_IO 2.0V @ 3A U92 p. 57
Source/Sink Regulator VTTDDR_PL 0.75V @ 3A U28 p. 56
Switching Dual VCCPINT 1.0V @ 1.5A U104 p. 55
Switching Dual VCC1V5_PS 1.5V @ 2.5A U104 p. 55
Switching Dual VCCP1V8 1.8V @ 1.5A U105 p. 55
Switching Dual VCC3V3_PS 3.3V @ 2.5A U105 p. 55
Source/Sink Regulator VTTDDR_PS 0.75V @ 0.5A (3A Max) U27 p. 56
Switching Regulator VCC5V0 5.0V @ 2A U44 p. 56
Linear Regulator V33D_CTL1 3.3V @ 0.25A U20 p. 49
UG954_c1_35_031615

Figure 4.4.3: Power Rail Configuration in ZC706

4.4.3 Interconnection Properties (AXI)

All the Xilinx FPGAs support the AXI-BUs[30] (Advanced eXtensible Interface) communication, which one of the most commonly used communication protocols for device connectivity. It uses a Master-Slave communication approach with appropriate wiring for Data-Address communication. This flow of intercommunicating supports the creation of the appropriate wrappers in order for the FPGA components (such as a RocketChip core) to communicate with the base ARM core of the SoC and also all the peripheral devices of the evaluation board.

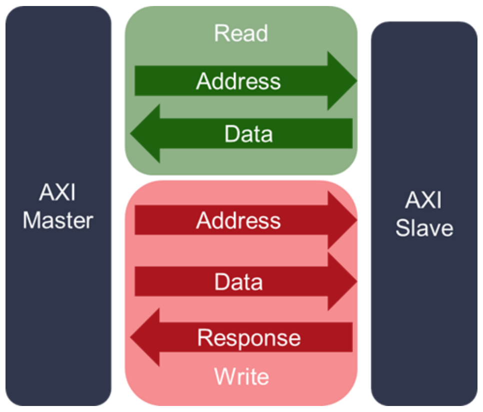


Figure 4.4.4: AXI-Bus Communication

In the Fig.4.4.4 is presented a very abstract implementation of the AXI protocol. The AXI bus has

3 different variations, called AXI4, AXI4-Lite and AXI4-Stream. The AXI4 is the most complex and the most used, as it supports burst transactions and out-of-order transactions. The AXI4-Lite is a simplified version of the AXI4, which is used for simpler peripherals and the AXI4-Stream is used for high speed data streaming. The deeper understanding of this protocol can be obtained from the Xilinx documentation, as it is very complex and out of the scope of this thesis.

4.5 Time Series

Time series analysis is a statistical technique used to analyze data points collected, recorded, or observed over a period of time. These data points are typically sequential and evenly spaced in time. Time series data can exhibit patterns such as trends, seasonality, and cyclicality, making it a powerful tool for understanding and forecasting phenomena in various fields including economics, finance, weather forecasting, and signal processing. By examining the temporal dependencies and fluctuations within the data, analysts can identify underlying patterns, detect anomalies, and make informed predictions about future trends or behaviors.

4.6 Data Correlation

The biggest part of all the data processing includes the correlation between the measured performance counter events and power rail measurements. In the experiments two types of correlation are introduced, the **static** and the **dynamic** correlation.

4.6.1 Static Correlation

The static correlation is the usage of the standard correlation metrics in order to determine how close the power metric data is to the performance metric data in the terms of the "behavior" along the time. With this concept as the main target and because we are not interested in a linear correlation, because such data may be incremented or decremented in a non-predictable way, we used the **Spearman's Rank Correlation Coefficient**.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Figure 4.6.1: Spearman's Rank Correlation Coefficient

In the provided equation in Fig.4.6.1, ρ is the Spearman's Rank Correlation Coefficient, d_i is the difference between the ranks of the two variables and n is the number of the samples. The ρ value is in the range of -1 to 1, where 1 is a perfect positive correlation, -1 is a perfect negative correlation and 0 is no correlation at all.

4.6.2 Cross Correlation

In the majority of the time series data (a great example will be the stock market) the data may be **historically correlated**. That means that a point in the time series A can be correlated with a point in the time series B, but not necessarily in the same time, meaning that a more statistically significant correlation can be found in a different time point. Also great observation can be made with the behavior of the different correlation coefficients in the different time lags(data shifts through time).

So the cross-correlation method is pretty straight forward. The n data samples are shifted through time in specific lags and then the static Spearman correlation is calculated. The result is $2*n$ **different**

correlation coefficients , n for feature correlations (positive lags) and n for past correlations (negative lags).

4.6.3 Significance Threshold

The correlation coefficient value by itself can show how wear or strong are two time series of data "behaving" in the same way, but it cannot provide enough information about how confident statistically we are that this correlation is real. For this reason the value called **Significance Threshold**[31] is introduced, in order to provide a correlation value, above which we can be confident that the correlation is not a random event.

The significance threshold is calculated by the following equation (for the Spearman correlation).

$$\rho_{\text{threshold}} = \sqrt{\frac{\alpha^2}{\alpha^2 + (n-2)}}$$

Figure 4.6.2: Significance Threshold

In the Fig.4.6.2, α is the significance level, which is usually set to 0.05 and n is the number of the samples.

4.7 Denoising

The data that usually come from voltage and current measurements introduce some unusual or random behavior, which is called **noise**. This anomaly of data has a lot of sources, like the measurement equipment, the power supply, the environment and the measurement process itself and if the measuring interval is small, then the noise impact on the data is more significant. For this purpose, there are statistical algorithms that provide a way to reduce the noise of the data set, keeping the main behavior (for example the distribution) as intact as possible. Two of the algorithms that were present in our evaluation environment were the **Rolling Average** and the **Gaussian Filter**.

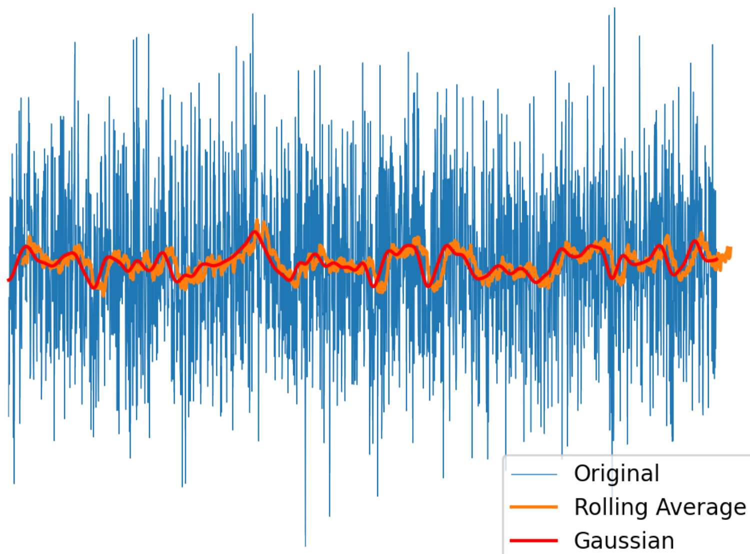


Figure 4.7.1: Abstract Denoising Representation

In Fig.4.7.1, the blue line represents the original data set, the orange one the Rolling Average technique and the red one the Gaussian Filter. The purpose of this representation is to abstractly show the behavior of the two algorithms in a noisy data set.

4.7.1 Rolling Average

The rolling average is a simple statistical algorithm that calculates the average of a specific number of data points in a specific window and then shifts the window through the data set. The result is a new data set with the same number of data points, but with less impact from the channel noise.

$$y_i = \frac{1}{n_{\text{window}}} \sum_{j=0}^{n-1} x_{i+j}$$

Figure 4.7.2: Rolling Average

In this smoothing situation is always a trade off between the quality loss of the information and thew noise reduction. This means that the window size n_{window} is a crucial parameter for the accuracy of the analysis.

4.7.2 Gaussian Filter

The Gaussian filter is a more complex algorithm that uses the Gaussian distribution to calculate the weighted average of the data points in the window. The result is a new data set with the same number of data points which has also less noise impact, but with a much more smooth behavior (lacks of "sharp edges"). This makes this algorithm less suitable for our cases, but can produce very beneficial results in the parts that this data behave in a more continuous way.

$$y_i = \frac{1}{\sqrt{2\pi}\sigma} \sum_{j=0}^{n-1} x_{i+j} e^{-\frac{(j-\mu)^2}{2\sigma^2}}$$

Figure 4.7.3: Gaussian Filter

In the Fig.4.7.3, σ is the standard deviation of the Gaussian distribution and μ is the mean of the Gaussian distribution. In the same way as the moving average, the σ value is a crucial parameter for the accuracy of the analysis.

Chapter 5

Modifying the HTIF

In the process of meticulously constructing the benchmarking environment, it became imperative to undertake a significant modification to the Host-Target Interface. This adjustment was essential for the efficient management of Voltage/Current measurements in conjunction with the collection of performance counter data. One of the most substantial challenges encountered during this endeavor was the synchronization between the host measuring process and the target performance metrics. The complexity of ensuring real-time alignment and accurate data correlation posed a significant overhead, necessitating the development of a custom solution, which ensures as little interference as possible with the target system's operation. This chapter will provide a detailed explanation of the modifications made to the HTIF, as well as the rationale behind these changes.

5.1 Timestamp Approach

The most naive approach that it must synchronize the data is to attach the same (or in worst case a very close) timestamp to the two different types of data measurements. The first obstacle of this approach is that due to the bare metal nature of the benchmarking environment the host-target timestamps are not automatically in sync, because the clock of the emulated system starts when the frontend opens. Although as described in Chapter 4, the HTIF provides a high level interface for system call handling, so the forwarding of the time system call can be used for this communication.

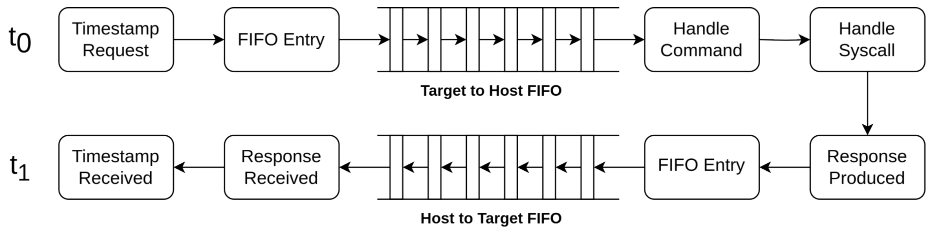


Figure 5.1.1: HTIF Timestamp Approach.

Algorithm 1 Timestamp Approach Algorithm (Host Side)

```
1: while TH_FIFO is not empty do
2:     $req \leftarrow$ pop(TH_FIFO)
3:     $res \leftarrow$ handle_syscall(req)
4:     push(res, HT_FIFO)
5: end while
```

Algorithm 2 Timestamp Approach Algorithm (Target Side)

```
 1: procedure FRONTEND_SYSCALL(code, data)
 2:     $req \leftarrow$ (code,data,device_id)
 3:     push(req, TH_FIFO)
 4:     wait until $res \leftarrow$ pop(HT_FIFO)
 5:     return res
 6: end procedure
 7: procedure BENCHMARKING
 8:     ...                                          ▷ Main Benchmarking Code
 9:     $res \leftarrow$ frontend_syscall(TIMESTAMP_SYSCALL,null)
10:     bind_data(res,PERF_DATA)
11: end procedure
```

As seen also in the Fig.5.1.1 from the time t_0 this timestamp will be requested, it will require **an extra communication and transmission overhead** and it finally produces the requested output in the time t_1. So this $\Delta t = t_1 - t_0$ is not a constant overhead in every timestamp request because **it depends on the FIFO contents**. This makes this approach not the most efficient for approaching the real-time synchronization of the data.

A more clean picture of the procedure is being analyzed in the Algorythms 1 and 2, where is we can see that both the target and the host are waiting for the transfer of the data, making a huge impact of the correctness of the data measured.

5.2 Custom System Call Approach

Another way of reducing the produced overhead by the timestamp request is to implement a custom system call that will transmit the requested data one time, so the one out of two FIFO transfers will be avoided. Thus the data processing will be more efficient to be in the ARM Core, it will be more optimal to implement the system call to transfer the performance data from the RISC-V core to the ARM core.

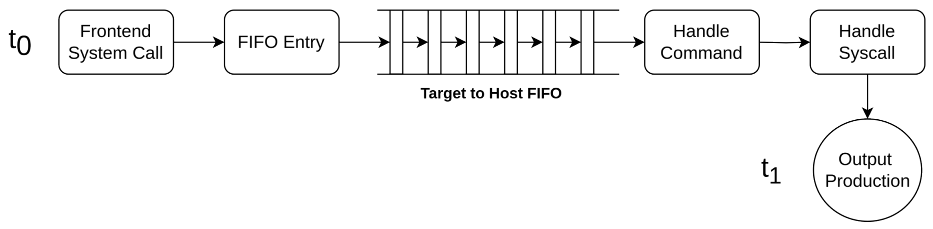


Figure 5.2.1: HTIF Custom System Call Approach.

Algorithm 3 Custom System Call Approach Algorithm (Target Side)

```
 1: procedure FRONTEND_SYSCALL(code, data)
 2:     $req \leftarrow$ (code,data,device_id)
 3:     push(req, TH_FIFO)
 4:     wait until $res \leftarrow$ pop(HT_FIFO)
 5:     return res
 6: end procedure
 7: procedure BENCHMARKING
 8:     ...                                          ▷ Main Benchmarking Code
 9:     frontend_syscall(CUSTOM_SYSCALL,HPC0_DATA,HPC1_DATA,...,HPCn_DATA)
10: end procedure
```

In this implementation which is shown in Fig.5.2.1, **the overhead is reduced** when compared to the timestamp implementation, but still **there are some steps before and after the FIFO implementation** that, although they are defined by the HTIF protocol as the optimal practice, they produce a not necessarily needed overhead.

In the Algorithm 3 we can see that the main change in this procedure is that the data will be transferred and produced in the host (which handles the system call the same way as described in the Algorithm 1).

5.3 Modified Data Transfer Approach

If we deconstruct the data transmission problem further we will come to a conclusion that is not the system call that matter in out implementation, but the data this system call will transfer. So the handling of the system call is not necessary for the data transfer.

Having this crucial aspect in mind, the HTIF was modified in order to support a custom command that if it send from the RISCV core, the data will be parsed **immediately** without any further high level handling of this command, which bypasses all the system call handling process. This approach is leaving **only the ctutial to functionality aspects of the HTIF**, such as the FIFO and the push and pop handling.

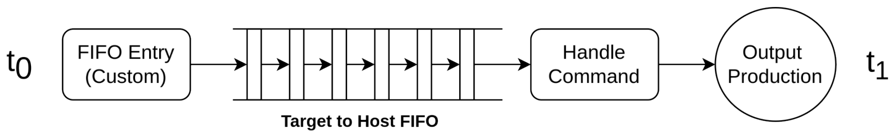


Figure 5.3.1: HTIF Modified Data Transfer Approach.

Algorithm 4 Modified Data Transfer Approach Algorithm (Host Side)

```
1: while TH_FIFO is not empty do
2:     req ← pop(TH_FIFO)
3:     if req[CMD] is a data transfer command then
4:         res ← parse_data(req)
5:         bind_data(res,POWER_DATA)
6:         push(null, HT_FIFO)                         ▷ Nothing to return back
7:         continue
8:     end if
9:     req ← pop(TH_FIFO)
10:    res ← handle_syscall(req)
11:    push(res, HT_FIFO)
12: end while
```

Algorithm 5 Modified Data Transfer Approach Algorithm (Target Side)

```
1: procedure BENCHMARKING
2:     ...                                             ▷ Main Benchmarking Code
3:     push(binded_req(DATA_TRANSFER_CMD,HPM_DATA), TH_FIFO)
4: end procedure
```

The implementation as stated in Fig.5.3.1 show the contraption of out implementation which so far **is the most efficient** in terms of computation overhead. This approach, as already discussed, have a constraint that **a high level co-processor must exists** so it cannot be reduced any further.

As well as the other approaches, the Algorithms 4 and 5 show the host and target side of the implementation, where we can see that there is no more system call wrapping in the data transfer and also the data processing starts immediately after fetching from FIFO.

Chapter 6

Experimental Setup

This chapter will describe all the experimental methodology that was followed in order to achieve the proper data collection and present the results of this thesis. The chapter will be divided into three main sections, the wrapping of the rocket chip in the Zynq FPGA, the data collection-processing and the benchmarks that were used for the validation of our results. Also in the end will be presented some challenges and compromises that needed to be addressed before the explained methodology.

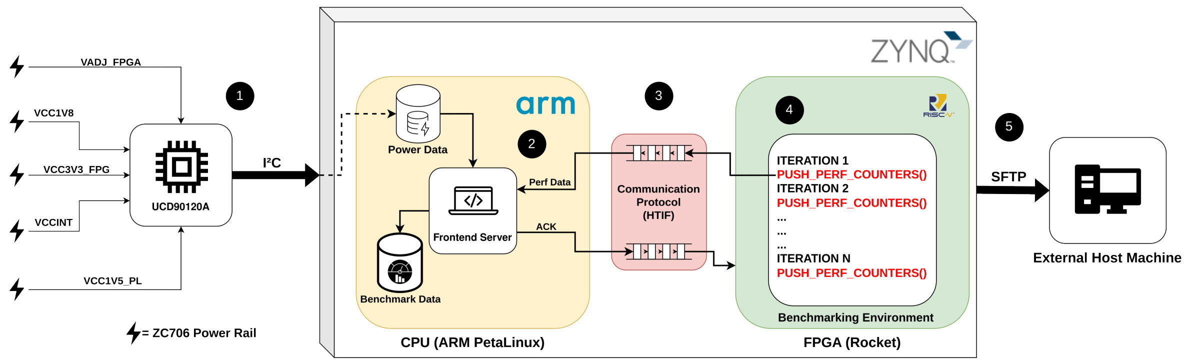


Figure 6.0.1: Experimental Flow

6.1 RocketChip in ZC706

Implementation

For adapting the RocketChip core to the ZC706 FPGA development board, we used a higher level of communication that introduced an **adapter** component to out RTL implementation. The adapter comes between the actual RocketChip implementation and the Processing System (ARM core and peripherals). The adapter component is responsible for the communication with the ARM core front end and then translating the AXI commands to instruction and data communication segments that are transferred to the RocketChip implementation. This wrapped module (adapter and Rocket) is loaded on the FPGA Core and communicate with the ARM core only through AXI as already mentioned.

External clocking is achieved be adapting the **Zynq system clock** with the **MMCME2** clock divider with the appropriate constrains that can control the clocking of the PS and all the components of the PL (which is mainly the RocketChip and helping peripheral devices).

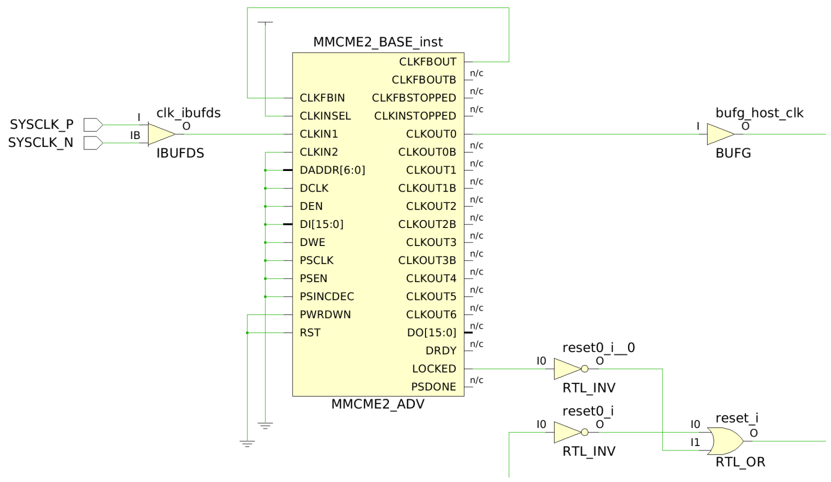


Figure 6.1.1: MMCME2 clocking divider RTL

In the Fig.6.1.1 is presented the clocking of the core along with the inputs and outputs, the only output is connected to the main RocketCore subsystem with clock rate calculated by the below equation.

$$f_{\text{RocketChip}} = \frac{1000}{T_{\text{ZYNQ}}} \times \frac{\text{RC\_CLK\_MULT}}{\text{RC\_CLK\_DIVIDE}}$$

Figure 6.1.2: MMCME2 clocking divider equation

From this equation (Fig.6.1.2) the parameter T_{ZYNQ} is the period of the Zynq system clock, and the parameters RC_CLK_MULT and RC_CLK_DIVIDE are the parameters that are set in the constrains file in post synthesis time.

The synthesis, implementation and bitstream generation parts were done using **Xilinx Vivado HLx Editions 2018.3**. The implementation was focused in a single core Rocket implementation of the **RV64IMAF architecture** in order to have focused execution environment and **to not include any scheduler overhead**, at least for this stage of exploration.

Utilization

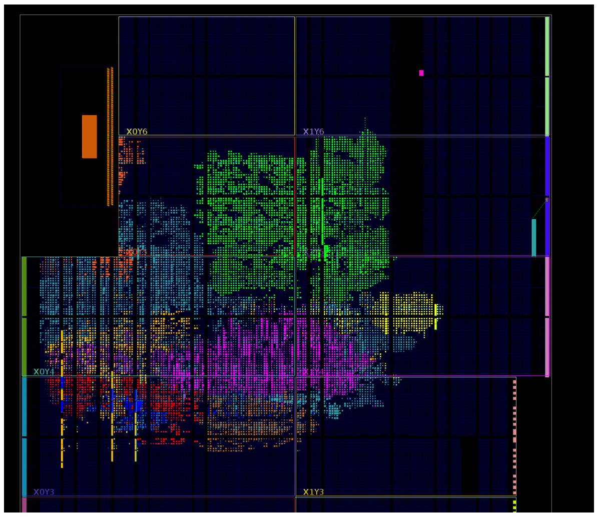


Figure 6.1.3: Floorplan of the Big Rocket Core

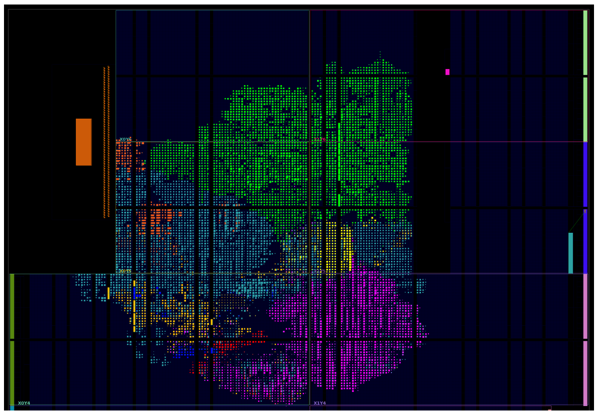


Figure 6.1.4: Floorplan of the Medium Rocket Core

Resource	Utilization	Available	Utilization %
LUT	35042	218600	16.03
LUTRAM	1121	70400	1.59
FF	18426	437200	4.21
BRAM	24	545	4.40
DSP	15	900	1.67
IO	2	362	0.55
BUFG	1	32	3.13
MMCM	1	8	12.50

Table 6.1: RocketChip Big Core Utilization

Resource	Utilization	Available	Utilization %
LUT	30312	218600	13.87
LUTRAM	1105	70400	1.57
FF	13195	437200	3.02
BRAM	6	545	1.10
DSP	11	900	1.22
IO	2	362	0.55
BUFG	1	32	3.13
MMCM	1	8	12.50

Table 6.2: RocketChip Medium Core Utilization

In the Fig. 6.1.3 and Fig. 6.1.4 is presented the floorplan of the RocketChip with one Big and one Medium core in the ZC706 FPGA. The utilization of the RocketChip in the ZC706 is also presented in the Tables 6.1 and 6.2. The utilization is calculated by the Vivado tool after the implementation of the RocketChip in the ZC706 FPGA. For the floorplaning analysis each color represents another building component of the system.

Power/Thermal Analysis

Furthermore in the FPGA floorplaning, in the post implementation step of the RocketChip in the ZC706, the power and thermal analysis was performed.

Configuration	Frequency (MHz)	Junction Temperature ($^{\circ}C$)	Max Power (W)
Big Rocket Core	12.5	28.7	2119
	50	28.7	2065
	87.5	28.5	2065
Medium Rocket Core	12.5	28.9	2207
	50	28.6	2052
	87.5	28.9	2207

Table 6.3: Post-Implementation Power/Thermal Analysis

6.2 Data Collection-Processing

For the proper communication with the frontend server, we introduced a customized implementation of the HTIF protocol as described in the Chapter 5 (Fig. 6.0.1 Section **3**). With this communication protocol, we were able to transfer 2 different performance counter data per benchmark run from the emulated environment to the host system. This data were then bind with the measured voltage and current from the FPGAs power rails (Fig. 6.0.1 Section **1**) in this time frame. Then this data were put in as an entry to a CSV file describing the benchmark run and the measured power consumption (Fig. 6.0.1 Section **2**). All the executables in the Processing System were compiled with the **arm-xilinx-linux-gnueabi** compiler and were executed in the PetaLinux (build 2018.3) environment using kernel version 4.14.

After the data were collected the produced dataset were transferred via SFTP from the development board to a host machine for the processing flow (Fig. 6.0.1 Section **5**). This flow includes:

- The production of the building block utilization of the RocketCore across benchmarks.
- The static (Spearman) correlation of the power rails and the performance counters using Running Average and Gaussian Filtering.
- The cross-correlation of the power rails and the performance counters using Running Average and Gaussian Filtering.
- The Spearman correlation between all the performance counters with themselves.
- The production of the best correlations per benchmark.
- The production of the best correlations across all benchmarks and environment configurations.

All the processing part was handled in a **Python 3.8.10** environment using the Pandas, Numpy, Scipy and Matplotlib libraries. The produced results were then presented in the next chapter.

6.3 Benchmarks

For the validation and the observation of our environment we used two types of benchmarking (Fig. 6.0.1 Section **4**). The micro-benchmarking technique in order to target individual and smaller building blocks of the RocketChip core in order to have more focused results and the utilization of an official benchmark suite which in our case was the CoreMark. All the benchmarks **were executed in the bare metal core**, in order to be zero interference from the operating system subprocesses and system calls, and compiled with the **riscv64-unknown-elf** compiler with cross-compilation with an x86_64 host machine.

Micro-Benchmarks

For the micro-benchmarking we used the implemented library of prebuilt benchmarks called **riscv-tests** and is developed by Berkeley University.

Benchmark	Description
mm	Matrix Multiplication
multiply	Multiplication Testing of Random Generated Data
qsort	Quicksort Algorithm
pmp	Memory Protection Test
towers	Hanoi Tower Algorithm (Recursion Test)

Table 6.4: Micro-Benchmarks

In the Table 6.4 are presented the five microbenchmarks that were used for this implementation. We also tested the **Dhrystone Benchmark** which was included in the same benchmarking suite but can be a more generic implementation than a conventional microbenchmark.

CoreMark

To have a more complete validation of the system running in terms of performance analysis we used the CoreMark benchmarking suite. The CoreMark is a widely used benchmarking suite for embedded systems and is used for the validation of the performance focused CPU and memory building blocks. For the build of the CoreMark we used the prebuilt **riscv-coremark** which is a CoreMark 1.0 port for the RocketChip core implemented also by the Berkeley University.

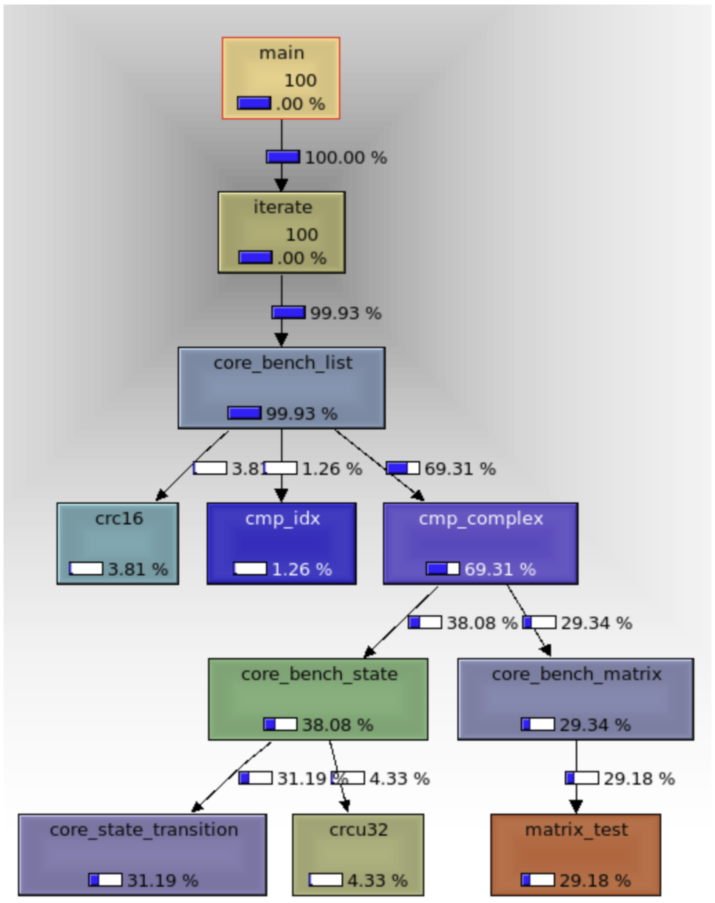


Figure 6.3.1: CoreMark Function Utilization

In order to enhance the data collection and also check the corresponding correlation, we based our research in the utilization scheme of CoreMark 1.0 presented in Fig.6.3.1 and we broke the data sources during the benchmarking in four main parts, which are also presented in the Table 6.5.

Benchmark	Description
coremark_matrix	Matrix Multiplication of Integers
coremark_mergesort	Mergesort Algorithm
coremark_find	Running a Finding Algorithm in a List
coremark_crc	Cyclic Redundancy Check Algorithm

Table 6.5: CoreMark Benchmark Breakdown

This breakout was necessary for the integrity of the data , because measuring the benchmark as a whole will create interference of different building block instances in one power measurement, which was not an optimal flow. For each metric we run a separate benchmark to reduce the overhead of data transfer.

6.4 Challenges

For the implementation of the RocketChip in the FPGA environment, there were a few challenges that need to be addressed. The biggest drawback were the clock critical paths of the implemented system. Many paths of the implemented system introduced a very wide critical path that picked around **10.7ns** meaning that the biggest frequency that the system can be tested is around **87.5MHz**. Everything above this will introduce timing violations and possible produce error in our measurement setup.

Another challenge was the AXI communication between the FPGA and the host system. This type of communication (FIFO based system) helped the overall procedure , making the ARM processor of the Zynq SoC the data collector and processor, but with a communication overhead. This challenge can be addressed (as will be presented in the *Feature Work* section) with the introduction of a RoCC accelerator that will address only this type of communication, with asynchronous timing and more low level communication.

Furthermore, based on the hard to debug environment of the emulated system and the configuration itself, it was not possible to test **riscv-torture** which performs a stress test in the CPU core. The source code of this benchmark was written in RISC-V Assembly and there were some low level incompatibilities with our custom system in the compilation stage.

Finally, in the Chapter 4 as we can see in the Fig. 4.4.2 and Fig. 4.4.3 some power rails power multiple components and also some rails are powered by other, closer to the PSU rails. This makes the data collection a bit more complex, because the outcome of each rail measurement is a combination of components and power rails so it is not so targeted at each individual component. Ideas and proposals for optimizing this flow will also be presented in the last section.

Chapter 7

Experimental Results - Evaluation

In this chapter, we will present the results of out experimental methodology. The type of the analysis was also analyzed in the Chapter 6. Due to the number of the overall outputs (about 500 plots for the correlations heatmaps) in some sections will present only some important results. The overall analysis can be found in the Appendix A.

Also in the produced correlation heatmaps the data were produced under this constrains which are valid for every plot that is presented in the following sections:

- In the **Y-axis** we have the **Performance Counters** that are measured.
- In the **X-axis** we have the **Power Rail Current and Power** that are measured. Voltage is in generally stable in every rail.
- When correlation coefficients are calculated, we present the **absolute value** of this coefficients, because we want to depict the **strength of the correlation** and not the **direction** (positive or negative).
- The **zero values are excluded** from the analysis, because they provide no useful information.
- In the same way, the **constant values of Power and Current** are also excluded.
- The **Rolling Window** is set to **200 values** when there are above 1000 measurements and **no window** when there are less.
- The **Gaussian filter** is set to a sigma of **15 values** when there are above 1000 measurements.

7.1 Benchmarks Utilization

In order to have a validation of the data measured, we need at first to be sure that the performance counters that are used are utilizing all the building blocks of the emulated system. For this, we presented the utilization across all benchmarks, as presented in the Figure 7.1. The results were collected from the **Big Core** configuration environment but the results are similar to the **Medium Core** so the re-run of all the benchmarks was not necessary.

Building Block	Utilization (%)
ALU	72.86
SYSTEM	0.01
BRANCH	10.11
FLOAT	0.01
I CACHE	0.02
D CACHE	0.70
PIPELINE	16.28

Table 7.1: Utilization of the building blocks across all benchmarks

As we can see from this analysis the **ALU** Performance Counters were the most utilized and with a huge difference with the other blocks. This is an expected behavior as the ALU handles the most simple number manipulation tasks (logic, arithmetic, shifting, etc.) and all the benchmarks include a part of this operations. So, even if we try to isolate all the other blocks always a basic ALU operation will be included in this frame of measurement.

Furthermore the **BRANCH** and **PIPELINE** blocks are also utilized in a very interesting amount. The branching is also well defined because of the **if-else**,**for** and **while** statements that are included in the benchmarks, which can also cannot be avoided. The pipeline events are also inevitable due to all the stalling and interlocking mechanisms in the classic 5-stage pipeline architecture that RocketChip implements.

In the other hand, **Data Cache** has a lower impact on the benchmarks. This is also expected because none of the benchmarks was memory bound and specific for targeting caches, so the impact was expected to be less than the core's main components. This impact is also visible in the **I Cache** utilization. In the same utilization percentage, **FLOAT** block does not have also impact, because this type of events occurred only on the **mm (matrix multiplication)** and **dhrystone** benchmarks, as well as the **SYSTEM**, which involves a very little set of system related counters, not directly related to benchmarks and core's performance.

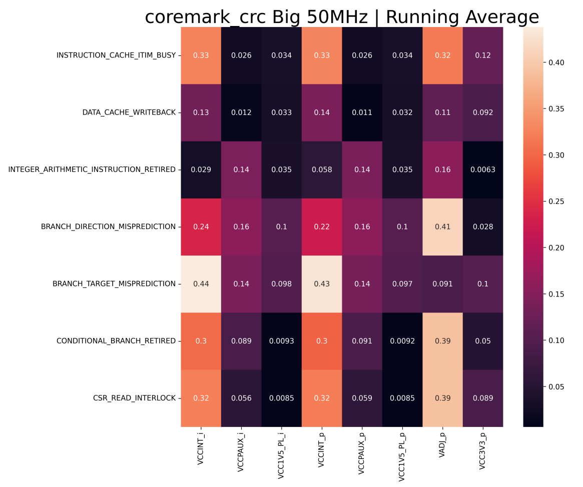
Finally, there are some system blocks that were not utilized at all, like the **EXCEPTION** handling events, which were utilized by 0.0002% of the overall system resources, something that is a very normal percentage, due to the lack of actual exceptions in the benchmarking process.

From the utilization we can come to a conclusion that besides the building blocks with the max utilization, as described above, the performance counters for the **underutilized** components cannot provide enough info for the complete analysis of those components. In cases as these, the creation of low-level (Chisel implemented) **custom performance event** is necessary, in order to have a complete picture of the system's behavior.

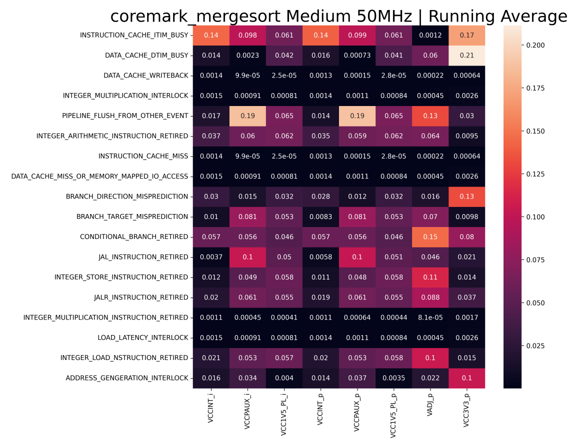
7.2 Static Correlation

In this section will be presented **important** results of the static Spearman correlation analysis and the descriptions across the total of the results.

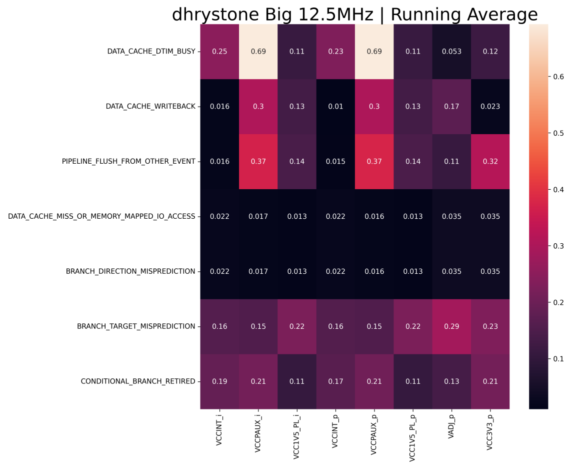
7.2.1 Rolling Window Correlation



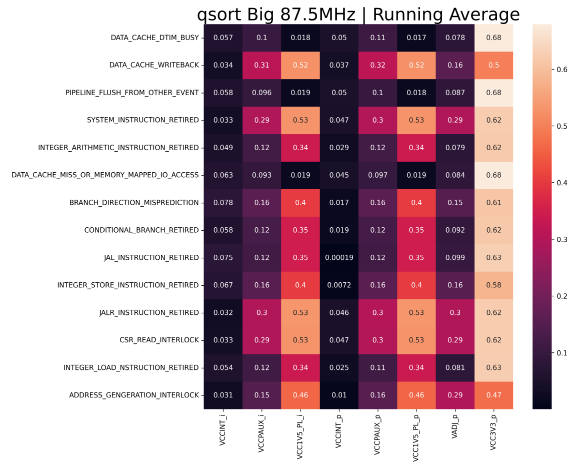
(a) Coremark Crc Big 50MHz



(b) Coremark Mergesort Mid 50MHz



(c) Dhrystone Big 12.5MHz



(d) Quicksort Big 87.5MHz

Figure 7.2.1: Rolling Window Correlations

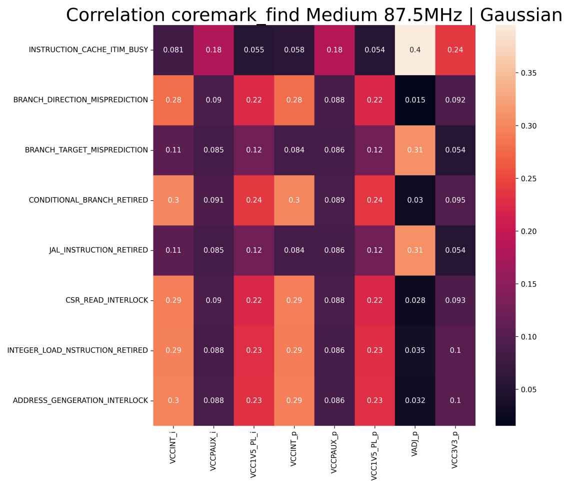
As we can see in the Fig.7.2.1 there were a lot of individual behaviors in each benchmark and each configuration under test. We can see that in cases such as Fig.7.2.1d the maximum correlations are in the **VCC1V5_PL**, which is an explainable behavior, due to the regions the VCC1V5 powers (Fig.4.4.2). We can also see that the **VCC3V3** has also a high value of correlation.

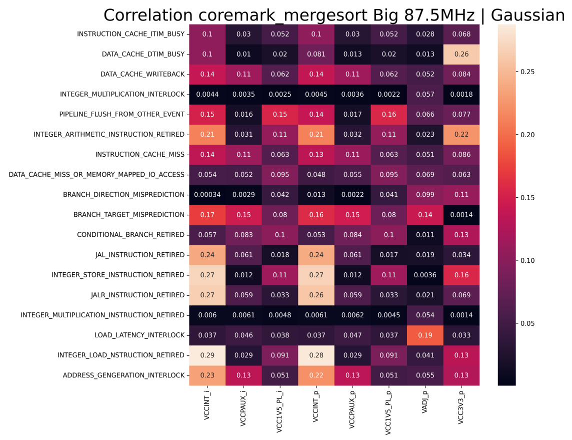
In the other hand, measurements as presented in the Fig.7.2.1b and Fig.7.2.1c have lower correlation coefficients and the picks are in rails that have different component utilization with PL, such as **VCCINT** and **VCCAUX**. That examples are not so clear, because the correlations are low and are in core components, so the results cannot be counted with a big weight in the feature extraction process.

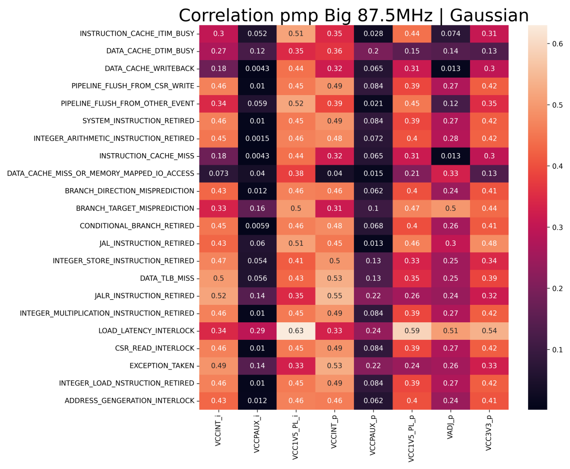
Furthermore, in examples such as Fig.7.2.1a we can see that there are high correlation values and in the rail that actively power the core (**VCCINT**).

Generally, we observed that the most correlated across all performance counters were the **VCCINT** and **VCC1V5_PL** rails, which are close to the logic (PL), as also described above. Also the benchmarks with generally high correlation values in important power rails were the **coremark_matrix** and the **dhrystone**.

7.2.2 Gaussian Filter Correlation


(a) Coremark Find Mid 87.5MHz


(b) Coremark Mergesort Big 87.5MHz


(c) PMP Big 87.5MHz

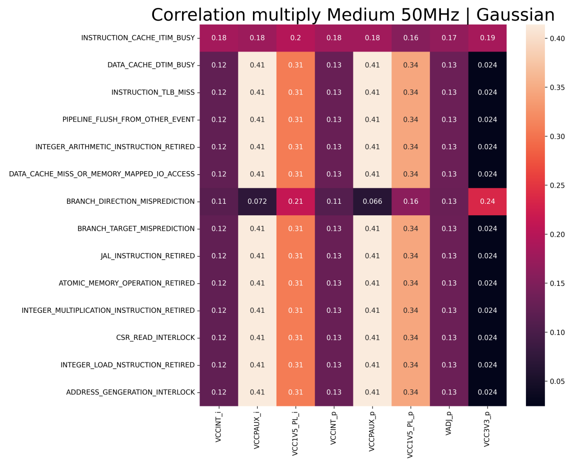

(d) Multiply Mid 50MHz

Figure 7.2.2: Gaussian Filter Correlations

In the Gaussian Filter Correlation analysis the overall results were more "blurry" than the rolling average, due to the nature of the filtering, which is more "soft" and "smooth" than the rolling window (it **cuts the spikes** of the dataset provided). The greatest correlation coefficient with scientific importance had a pick in the area of **0.65**.

We can see that in cases such as presented in Fig. 7.2.2b the there are same correlation coefficients across the almost all the counters/rails, this is a result that due to the lack of measurements or the behavior of the benchmarking events produced results that cannot be included in a model feature production and maybe more low-level measurement is needed for proper evaluation of this kind if behaviors.

In the other hand, more explainable behaviors can be found in cases such as Fig. 7.2.2a and Fig. 7.2.2c in which smaller correlation coefficients are produced but the picks were spotted around the **VCCINT** and **VCC1V5_PL** rails, which are important in the PL's power distribution.

Finally, in the case of Fig. 7.2.2c we can see that almost everywhere the coefficients are different and relatively high, with a pick correlation in **LOAD_LATENCY_INTERLOCK** in every rail. This is also a result that needs further investigation, due to the same behaving results in both of the axis.

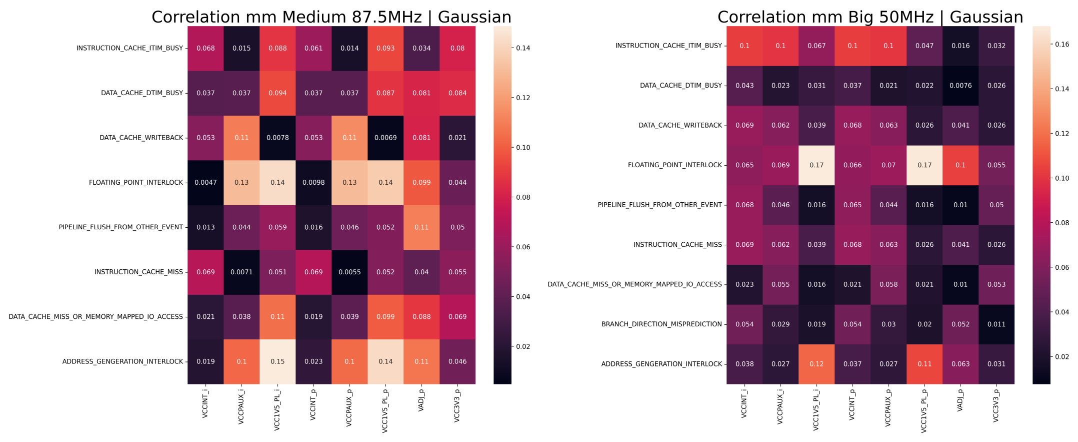


Figure 7.2.3: Case of Matrix Multiplication

In the Fig. 7.2.3 we have an interesting observation in the mm benchmark, because we can see that the **FLOATING_POINT_INTERLOCK** event is triggered with a high correlation (compared to the coefficients of the same benchmark) in the **VCCAUX** (in the Medium config) and **VCC1V5_PL** (in both configs) rails. The same behavior with hugh similarity is observed in 5 out of the 6 configurations (not observed in Big configuration with 87.5MHz frequency). This is a result that can be explained by the nature of the benchmark, which is a matrix multiplication and the floating point operations are the most important in this kind of operations. In the dhrystone benchmark, where we have also FPU presence, this behavior is not observed.

As analyzed in the beginning and in the description of the results, in this analysis there were not a standardized behavior of the performance counters and the power rails, so there are no so clear difference between the benchmarks and rail utilization except for the benchmarks of **multiply** and **pmp** which were producing very invalid results with behavior close to the Fig. 7.2.2d.

7.2.3 Feature Extraction in Static Correlation

Observing the produced results we can see that the model tend to be more **application specific** when the data provided are based in the "vanilla" performance counters and the power rails. As we can clearly see there are some heavy dependencies in the correlations that are produced, but there are no overlapping dependencies in each benchmark. So, even if may be exceptions, we cannot produce a standard flow for all the benchmarks tested.

7.3 Cross Correlation

In this section are presented the results of the cross-correlation between the performance counters and the power rails. In the presented results in the corresponding sections we will present the same benchmarks as in the static correlation analysis, so the visual comparison can be easier.

7.3.1 Cross Correlation Across Time

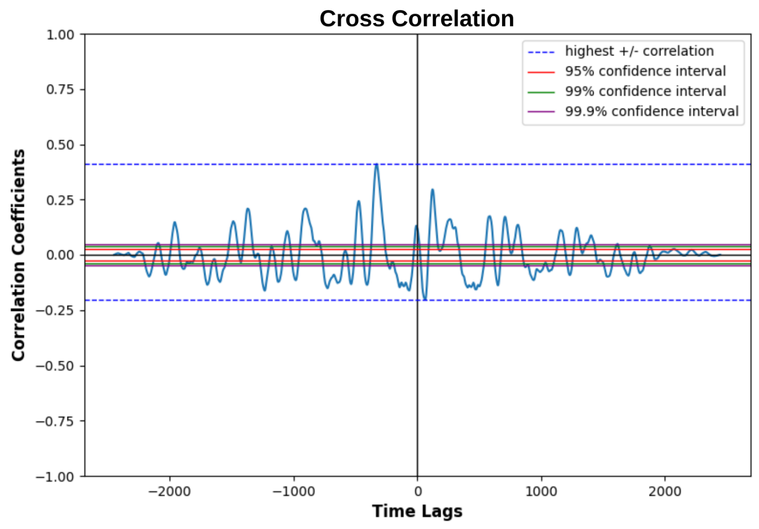


Figure 7.3.1: Cross Correlation Across Time Lags

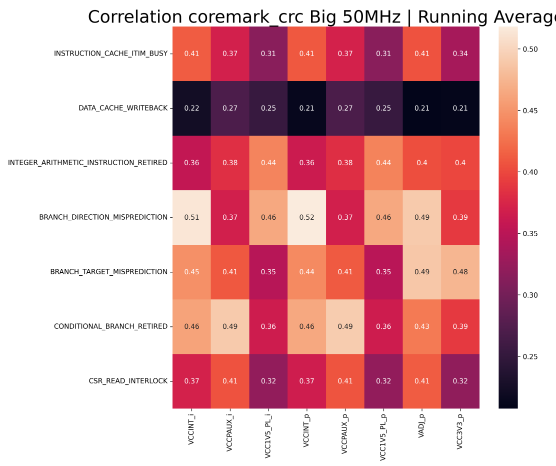
coremark_crc Big 50MHz VCC1V5_PL vs INSTRUCTION_CACHE_ITIM_BUSY

The Fig. 7.3.1 presents the cross correlations coefficients across different shifts in time, the time lags. We can see that in this example there is a periodic-like behavior of the correlation coefficients, which can be explained by the nature of those measurements because the crc routine can be called **multiple times** in the same time frame.

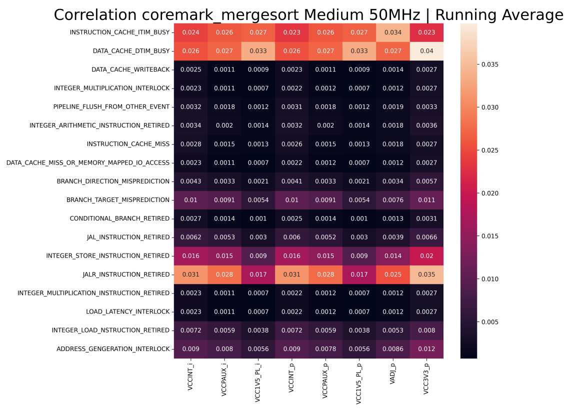
What we can also observe is the red, green and purple lines which represents the confidence intervals, above which the correlation (even if is not strong) is statistically significant. This formula is analyzed in Fig. 4.6.2 and as we can see is calculated with 3 different confidence levels, 0.95, 0.99 and 0.999.

If we run this analysis with each correlation then the amount of the produced plots will be in the area of **12000** plots. So to reduce this analysis overhead we will extract the **maximum absolute correlation coefficient** (highlighted by the blue dashed lines in Fig. 7.3.1) and the **time lag** that this coefficient was found. Then we will also consider the **confidence intervals** and the **significance** of the correlation in our final analysis.

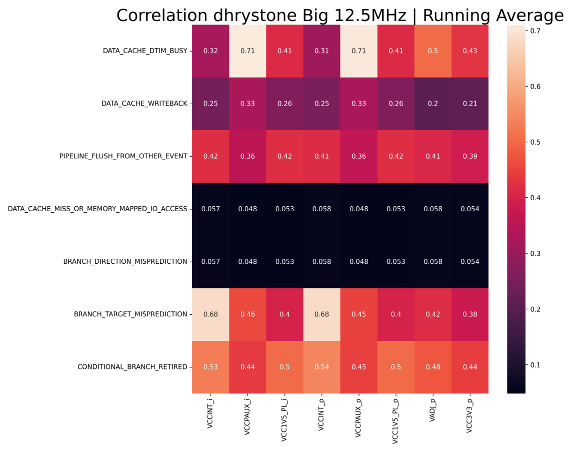
7.3.2 Rolling Window Cross Correlation



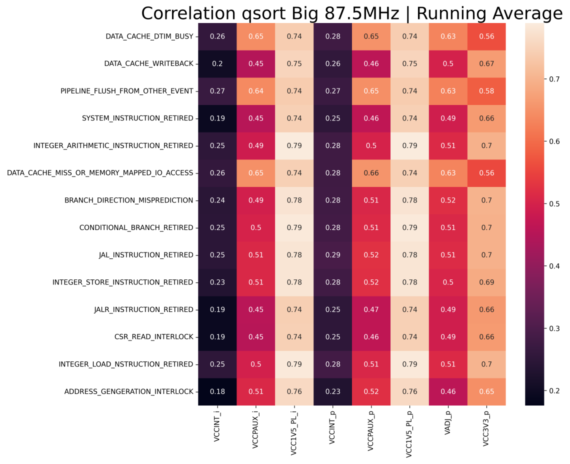
(a) Coremark Crc Big 50MHz



(b) Coremark Mergesort Mid 50MHz



(c) Dhrystone Big 12.5MHz



(d) Quicksort Big 87.5MHz

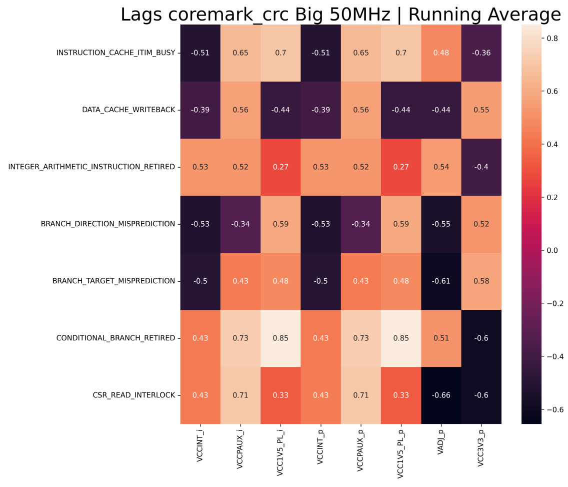
Figure 7.3.2: Rolling Window Correlations

As we can see in the Fig. 7.3.2 almost everywhere the results have higher correlation coefficients than the static analysis. For example in Fig. 7.3.2a we can see that the high correlation with the **VCCINT** remains but also there are a time dependant high correlation with the **VCCAUX** and also there are a same behavior in certain performance events such as **DATA_CACHE_WRITEBACK** which have higher coefficients but it is still the lowest in this benchmark. In the same flow are the benchmarks in Fig. 7.3.2c and Fig. 7.3.2b.

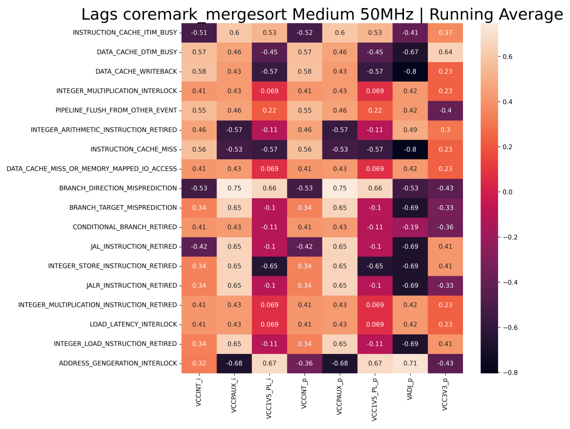
In the other hand int the results presented in Fig. 7.3.2d we can see that there are also in the same behaving scheme as the ones mentioned above, but in the X-axis in each rail we have almost the same correlation coefficients, which is a result that needs further investigation and timing analysis.

Futhermore, coming back to the **coremark_crc** example, the most of the events have the pick correlation in the same value area (0.38-0.5), which cannot be directly explained and we need to see the timing of this picks to obtain more information about the behavior of the benchmarks.

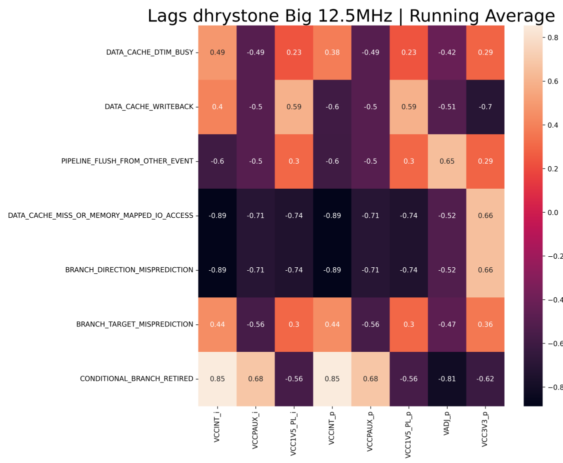
Concluding, we can see that in the majority of the benchmarks there is a **future**(all the values are positive) time dependency which tend to push the correlation higher than the static analysis. This result is not necessarily improving the correlation itself because it can heavily depend of the future behavior, which is not always predictable. This type of prediction and validation is an open topic for feature research, because it needs a heavy focus in lower level parts of the code (machine code level register and/or memory block accesses).



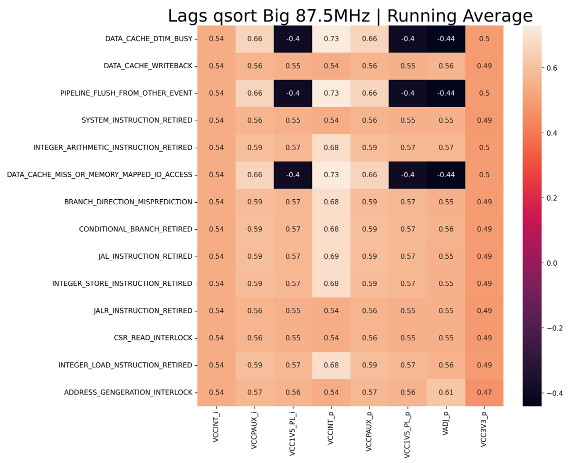
(a) Coremark Crc Big 50MHz



(b) Coremark Mergesort Mid 50MHz



(c) Dhrystone Big 12.5MHz



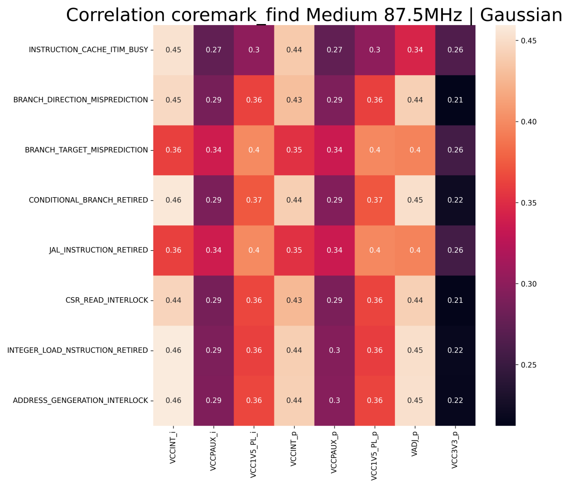
(d) Quicksort Big 87.5MHz

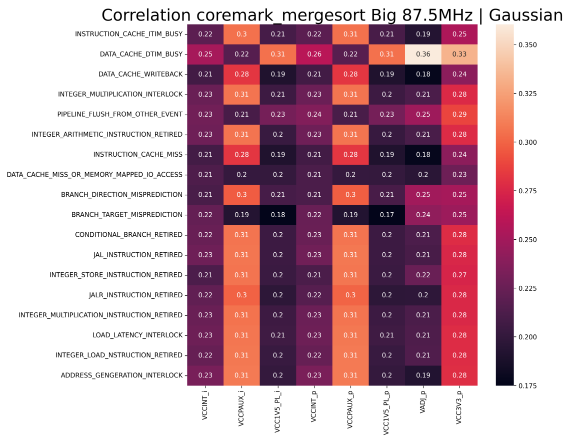
Figure 7.3.3: Rolling Window Cross Correlations Lags

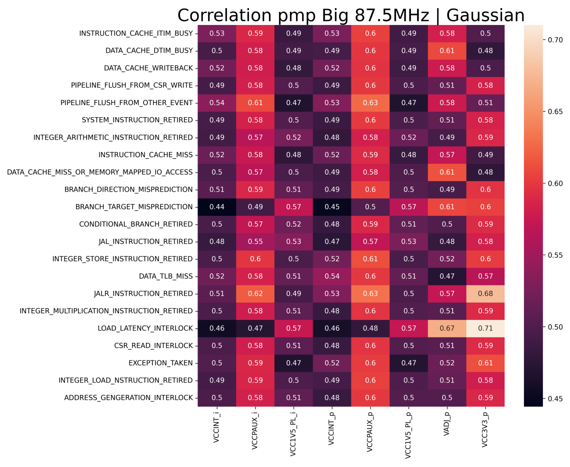
In Fig. 7.3.3 are presented the lags where the pick correlation was reached in the benchmarks. The form of the presentation of each lag is **the percentage of the distance from 0**, so the represented value is calculated as val $= \frac{\text{Lag}}{N_{\text{measurements}}}$.

From the presented results we can see the benchmark presented in Fig. 7.3.3d has the same time shifting in the pick correlation coefficients and this results in a low trust in this benchmark's results. In the other hand, the other three benchmarks mentioned in Fig. 7.3.3 have a more distributed timing lags, that have some similarities that mostly occur across the X-axis.

7.3.3 Gaussian Filter Cross Correlation


(a) Coremark Find Mid 87.5MHz


(b) Coremark Mergesort Big 87.5MHz


(c) PMP Big 87.5MHz

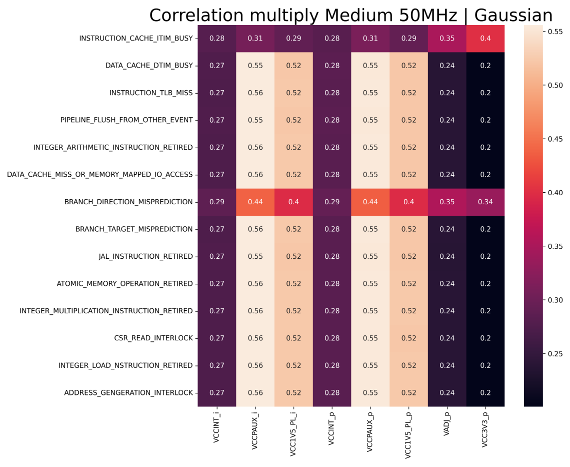
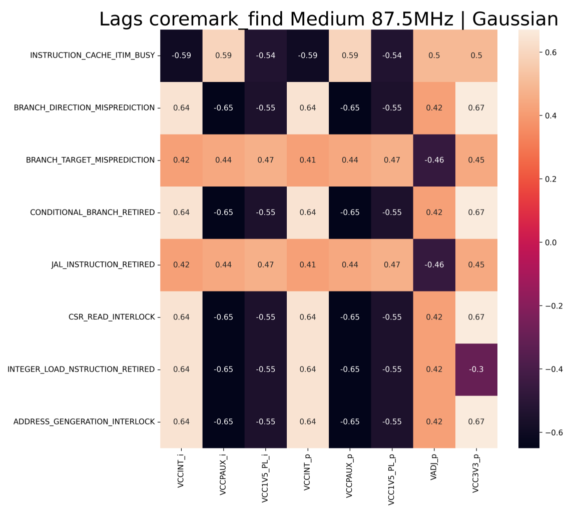
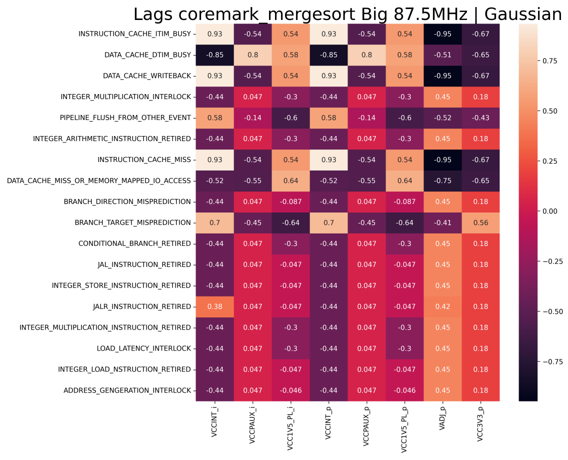

(d) Multiply Mid 50MHz

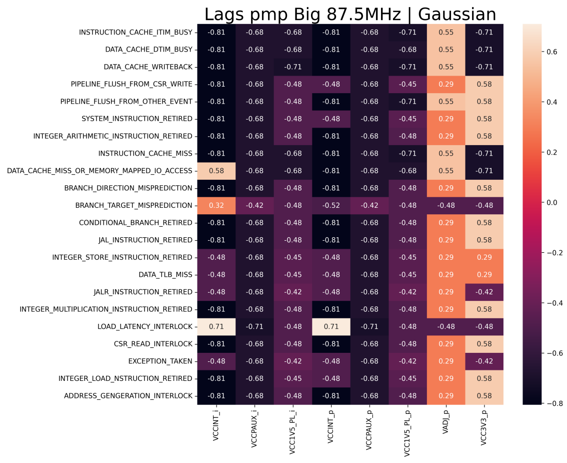
Figure 7.3.4: Gaussian Filter Cross Correlations

As discussed also in the rolling average result presentation, following the same flow, we can see that the correlation coefficients are relatively higher than the static analysis. So in this flow again the **multiply** benchmark (Fig. 7.3.4d) we again have results that arein a false positive case because we can see across two power rails same correlation coefficients.

In the other hand, the other benchmarks (and in general our solutions) have a higher coefficients in rails closer to PL such as **VCCINT** and **VCCAUX**. Again some false positive results can be found in cases such as Fig. 7.3.4a where also rail **VADJ** was utilized but this rail has no direct impact in PL nor PS core (is peripheral related).


(a) Coremark Find Mid 87.5MHz


(b) Coremark Mergesort Big 87.5MHz


(c) PMP Big 87.5MHz

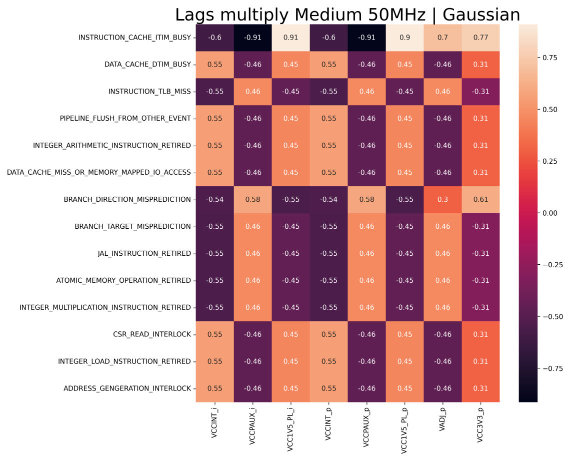

(d) Multiply Mid 50MHz

Figure 7.3.5: Gaussian Filter Cross Correlation Lags

Again, performing lag analysis we can see that again in the Fig. 7.3.5 the rails **VCC3V3** and **VADJ** have very out of scope correlation lags, which is normal due no direct impact on the powered be those rails regions. In the other hand, in the other rails the different counters have a pick in the same lag, which seems that there is a consistent behavior **per benchmark**.

Furthermore, as came as a result in previous sections also, benchmarks such as **multiply** (Fig. 7.3.5d) have a very similar behavior across the power rails, which is a result that needs further investigation and maybe another benchmarking target.

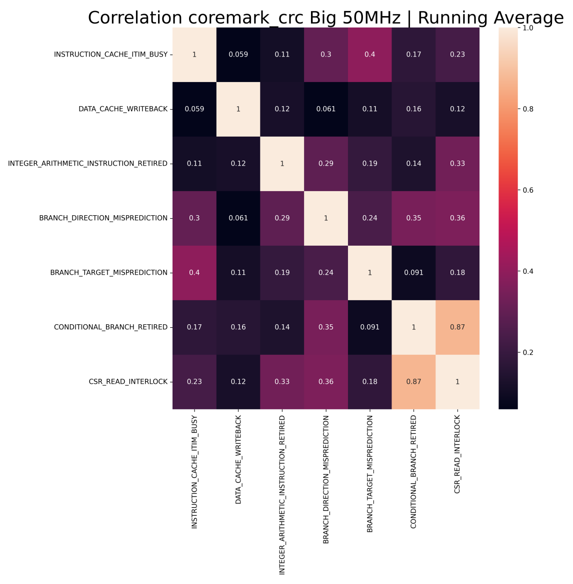
7.3.4 Feature Extraction in Cross Correlation

In the cross-correlation analysis is observed that even if the standard provided counters are enough for a basic evaluation we need to have two extra analysis tracks in the feature extraction process. The first one is the **time lag** analysis, which can show the dependance of the correlation in **history of events** and not only in the actual moment of measurement. That means that except the **applicaiton specific** character of a prediction it also needs to have **memory window** of the events that happened and the **time area** within the most interesting correlation coefficient pick. The second is the false-positive indication, because even if a performance counter event peaks in a higher correlation coefficient, does not necessarily means that this event is more important. The second track is not so well defined because it touches the boundaries between the energy prediction and the core behavior analysis.

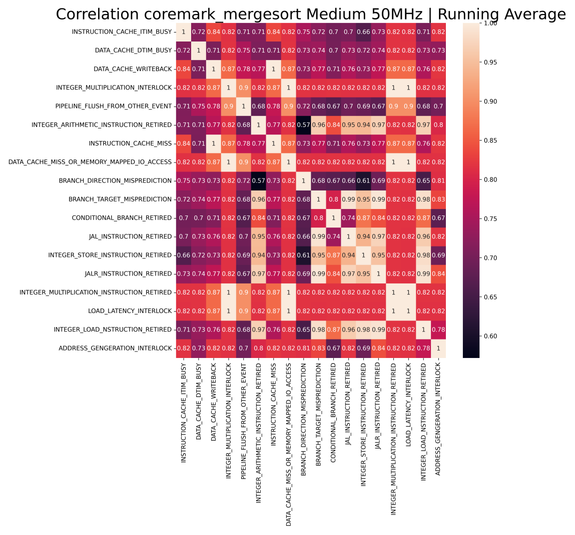
7.4 Performance Counter Inter-Correlation

In the previous sections were presented all the data from each individual benchmark and configuration with 2 different types of denoising filters and two correlation calculations (cross and static). In those results we observed that many counters in each individual rail behaved in the same way. This can certainly be a part of a benchmarking mechanism (if a comparison is made we can have events from **ALU** and **BRANCH** units fire in *almost* the same time). All this same firing events can also produce correlation in performance counters so when analysis is performed we can **focus** in the independent performance groups, each one with a different behavior (so no inter-correlation).

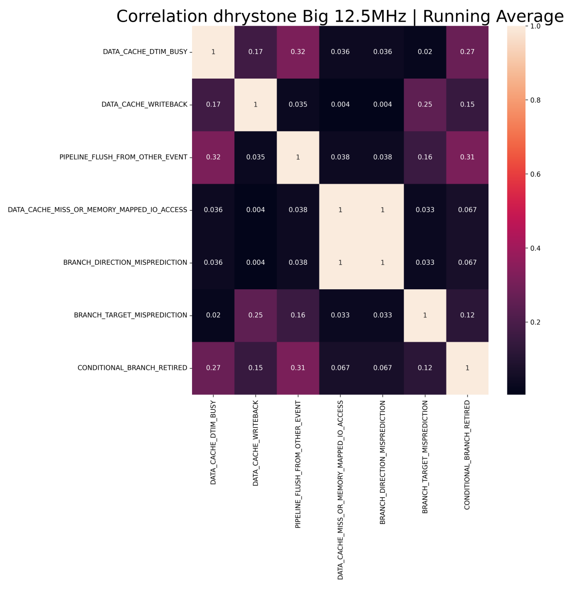
Before we proceed to the results, we first need to clarify that we did not calculated the cross-correlation between the performance counters, because we are investigating the property of two ore more performance counters "firing" together in the same time frame, which in our benchmarking system is included in each independent measurement.



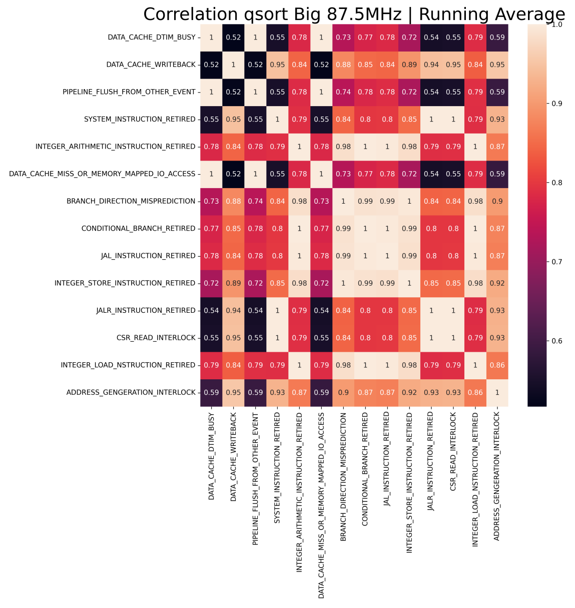
(a) Coremark Crc Big 50MHz



(b) Coremark Mergesort Mid 50MHz



(c) Dhrystone Big 12.5MHz



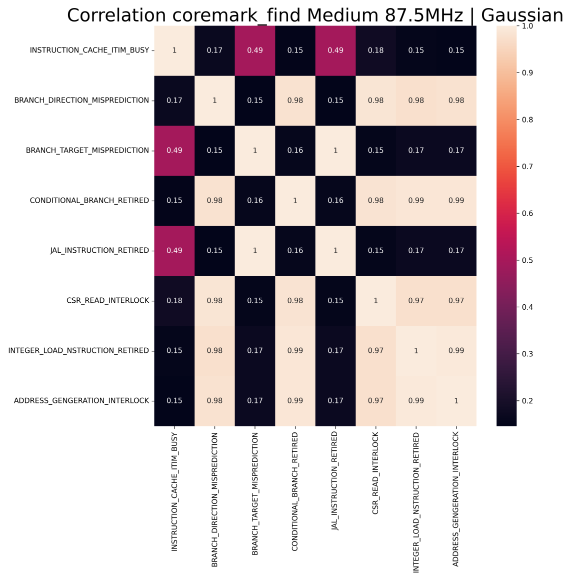
(d) Quicksort Big 87.5MHz

Figure 7.4.1: Rolling Window CSR Correlations

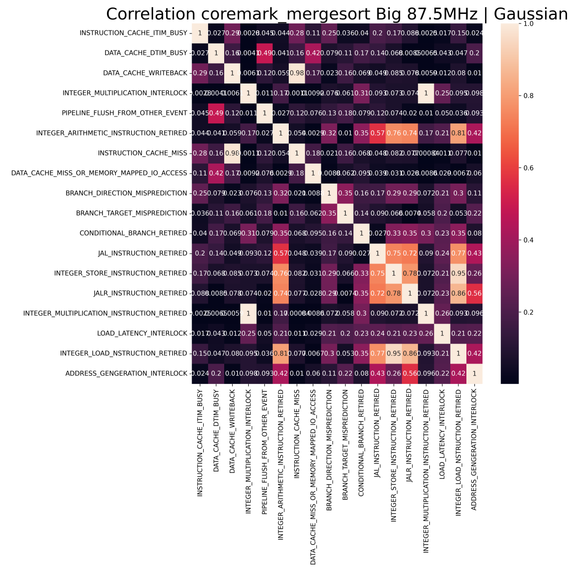
In the produced results (Fig. 7.4.1), observing benchmarks such as **coremark_crc** we can see that counters with same high correlation in static and cross analysis (Fig. 7.2.1a and Fig. 7.3.2a) have also high correlation in the CSR analysis (**CSR_READ_INTERLOCK** VS **CONDITIONAL_BRANCH_RETIRED**) but as we can also see in those benchmarks also other CSRs (**INTEGER_LOAD_INSTRUCTION_RETIRED**) do not have the same behavior.

Another example is the **coremark_mergesort** benchmark (Fig. 7.4.1b), where we can see that there are different groups of correlations between counters, but there are no important, because of the low correlation coefficients in benchmarking process.

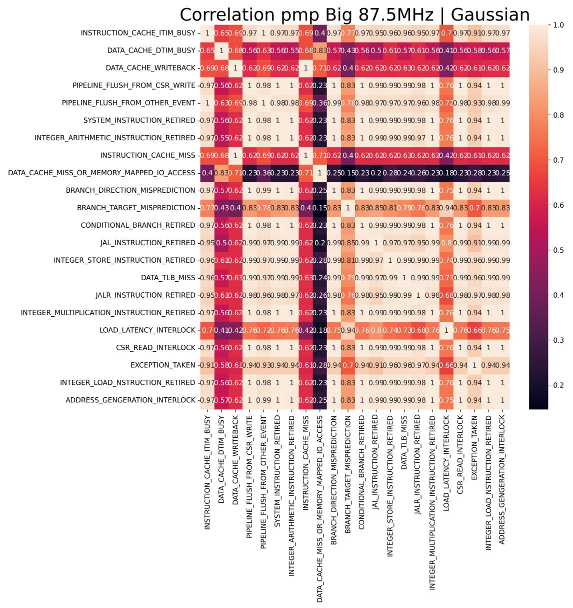
In the other hand, in examples such as **qsort** (Fig. 7.4.1d) we can see that the counters have a very similar behavior across the benchmarks, which makes bigger groups of same behaving counters, which reduces the feature extraction process and the data consistency.



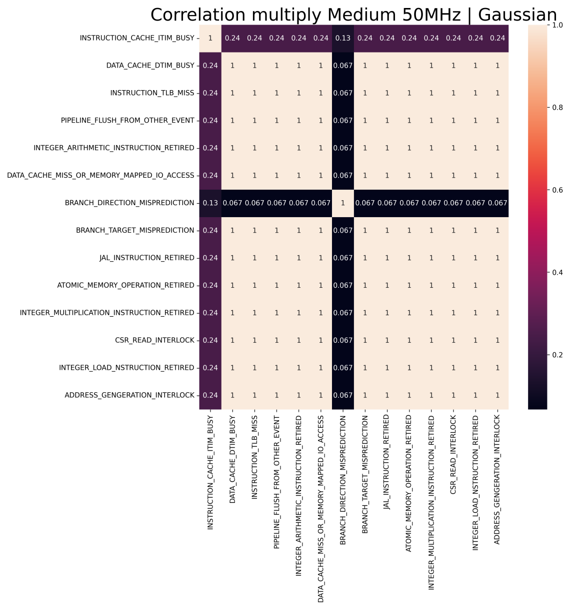
(a) Coremark Find Mid 87.5MHz



(b) Coremark Mergesort Big 87.5MHz



(c) PMP Big 87.5MHz



(d) Multiply Mid 50MHz

Figure 7.4.2: Gaussian Filter CSR Correlations

Using the Gaussian Analysis (Fig. 7.4.2) we can see that in general there are formed more groups of counters with same behavior. Benchmarks such as the **coremark_mergesort** (Fig. 7.4.2b) do not have large groups of correlated counters, but in the corresponding analysis they do not have also same correlation coefficients.

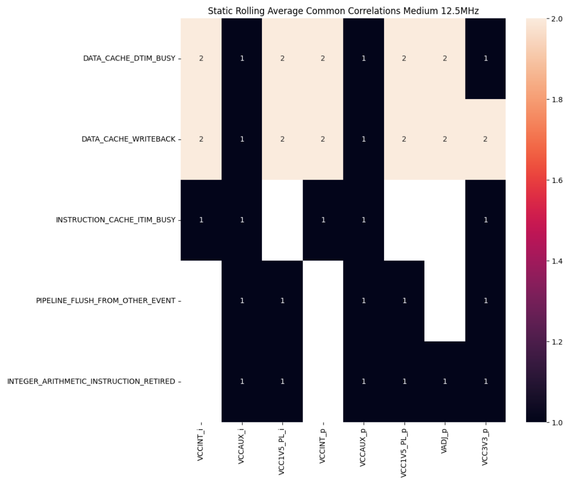
In a critical cases, such as the **multiply** benchmark (Fig. 7.4.2d) we can see that almost all the counters have the same behavior, which is no correct behavior as a result of the denoising methodology and the little number of the measurements we have in this specific benchmark. Almost the same behavior is observed in the **pmp** benchmark (Fig. 7.4.2c).

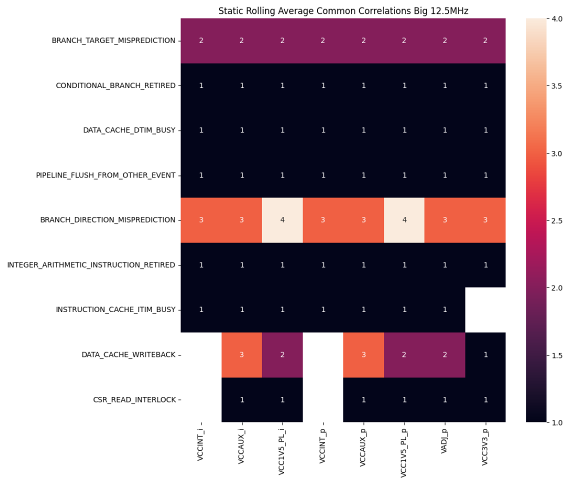
Concluding, in this section is understood that, if there are same behaving counters, the focus of the behavior must be in the **group of the counters** instead of the intendant counter behaviors. Also the denoising methodology of **Gaussian Filter** produces more groups of correlated counters, which can invalidate the integrity of the results and produce false positive results in the flow of feature extraction.

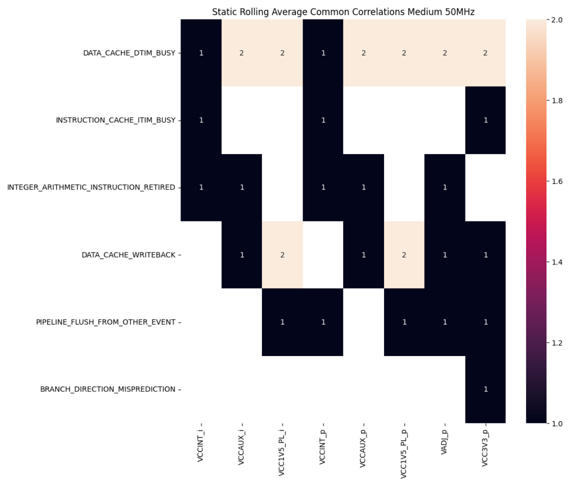
7.5 Across benchmark Analysis

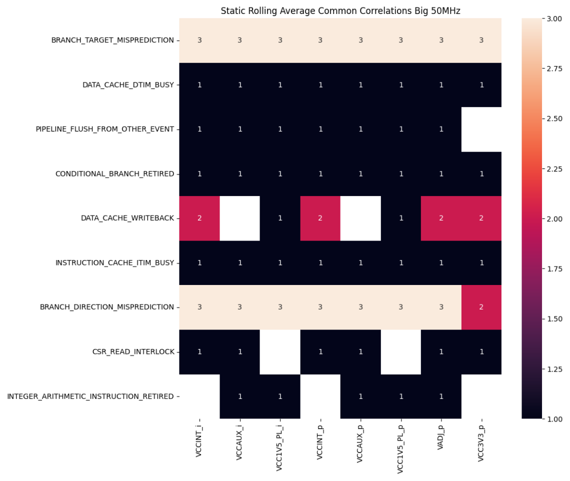
In this section we will present the results as a conclusion of all the above analysis process, in order to introduce a better view on **same correlation patterns** and **same behaving counters** across all the benchmarks. In all of the analysis done in the across-benchmark field **all the inter-correlated** counters are represented **by one counter** so we can be more focused on the **independent** results. Also, the correlations that are going to be presented are only the ones **above the significance level** with $a = 0.95$, so the correlation (even weak) will be statistically significant.

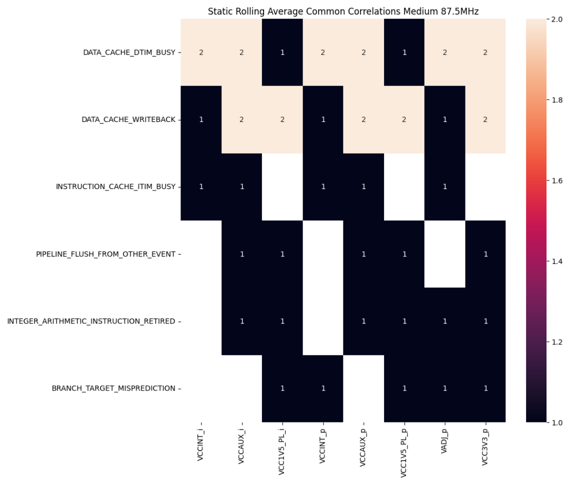
7.5.1 Static Rolling Window Analysis


(a) Mid 12.5MHz


(b) Big 12.5MHz


(c) Mid 50MHz


(d) Big 50MHz


(e) Mid 87.5MHz

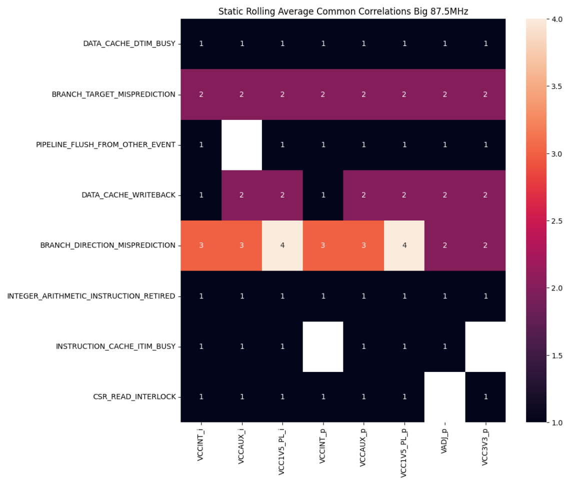

(f) Big 87.5MHz

Figure 7.5.1: Across Benchmarks Static Rolling Window Analysis

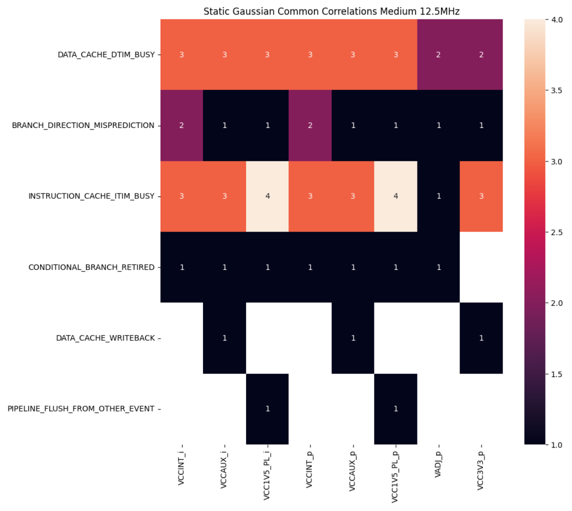
The results presented in 7.5.1 are showing the common counter-rail correlation pairs as chosen among the **top 5 common correlations** of each benchmark and with an **inter-correlation threshold of 0.5** (performance inter-correlated with a coefficient higher than 0.5 belong to the same group). Also we do not count the **VADJ** and **VCC3V3** in the final research output so the results that are correlated with those rails will be ignored. The rails are presented in the plots only for false positive reference. We can clearly see some common counter-rail correlations across all the frequencies that are presented in Table 7.2.

Configuration	Common Counter-Rail Behavior
Big	BRANCH_TARGET_MISPREDICTION
	DATA_CACHE_WRITEBACK
	BRANCH_DIRECTION_MISPREDICTION
Medium	DATA_CACHE_DTIM_BUSY
	DATA_CACHE_WRITEBACK

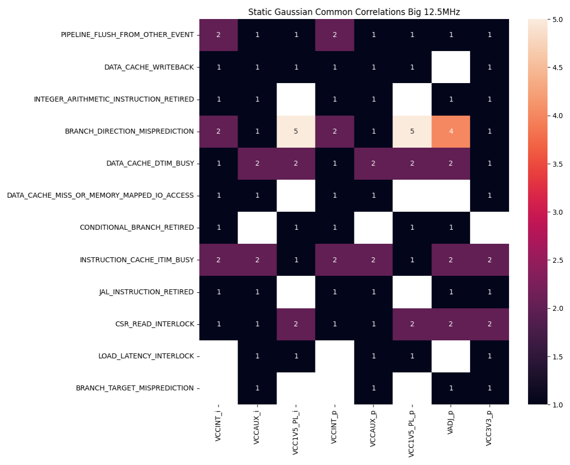
Table 7.2: Common Counter-Rail Behavior Across Benchmarks (Rolling)

In the produced outputs the **DATA_CACHE_WRITEBACK** is marked in blue because we can see that it appears in both of the configurations. We can also see that that in general the common correlation are not many. For example the pick is in the **BRANCH_TARGET_MISPREDICTION** in the Big configurations with 12.5MHz and 87.5MHz frequencies, which is the 4 out of 10 benchmarks. So this makes the behavior **not consistent** when the problem becomes generalized.

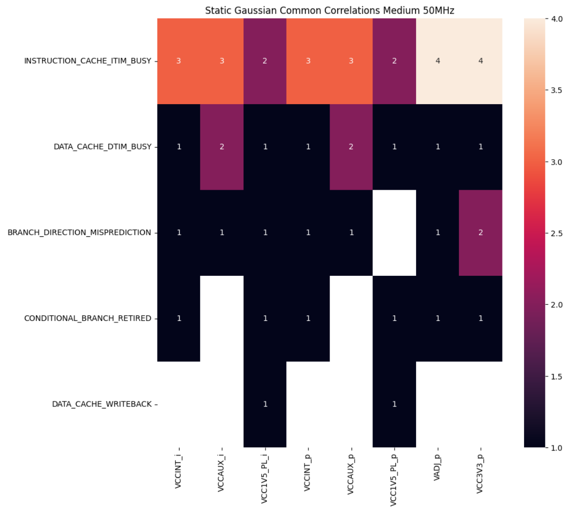
7.5.2 Static Gaussian Filter Analysis



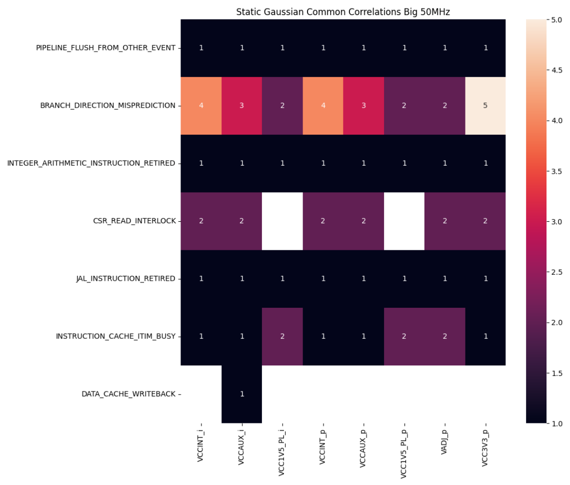
(a) Mid 12.5MHz



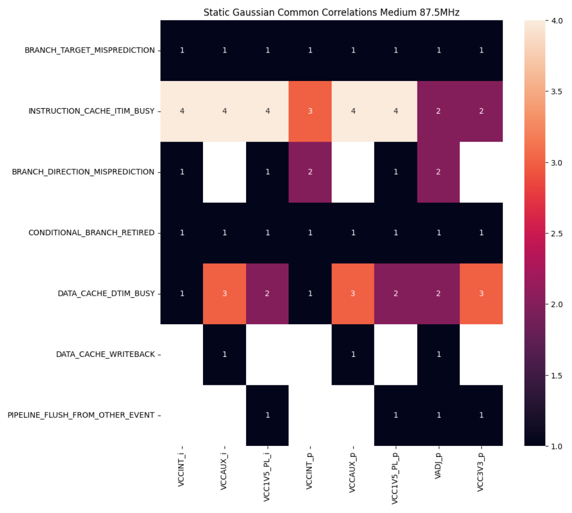
(b) Big 12.5MHz



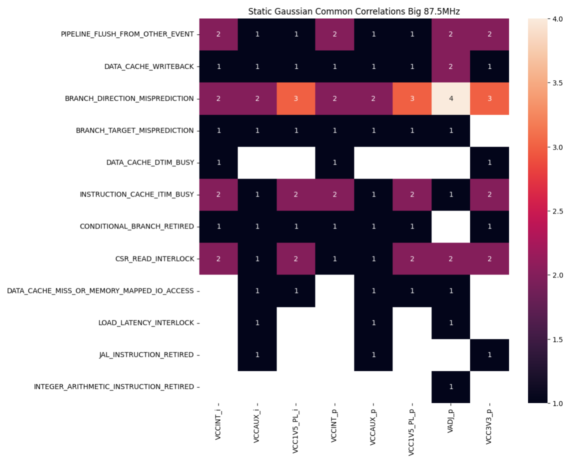
(c) Mid 50MHz



(d) Big 50MHz



(e) Mid 87.5MHz



(f) Big 87.5MHz

Figure 7.5.2: Across Benchmarks Static Gaussian Filter Analysis

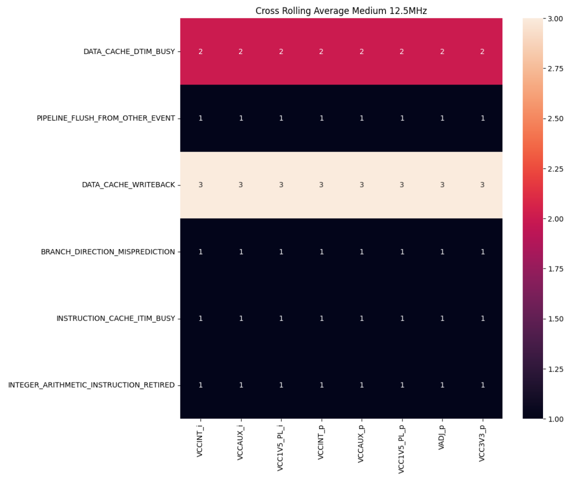
In Fig. 7.5.2 we present the across benchmark results for each configuration, which come as a result with the same parameters as the static rolling window analysis. In this analysis we can see that is a **less consistent** behavior across each configuration, with the same behaving counters presented in Table 7.3.

Configuration	Common Counter-Rail Behavior
Big	PIPELINE_FLUSH_FROM_OTHER_EVENT
	BRANCH_DIRECTION_MISPREDICTION
	DATA_CACHE_DTIM_BUSY
	INSTRUCTION_CACHE_ITIM_BUSY
	CSR_READ_INTERLOCK
Medium	**DATA_CACHE_DTIM_BUSY**
	BRANCH_DIRECTION_MISPREDICTION
	INSTRUCTION_CACHE_ITIM_BUSY

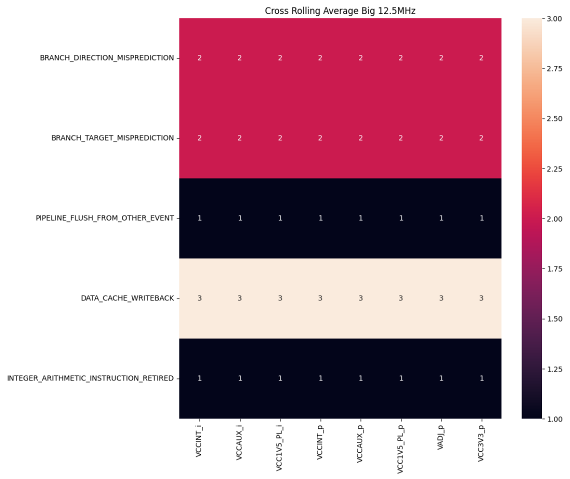
Table 7.3: Common Counter-Rail Behavior Across Benchmarks (Gaussian)

Again, from the results produced in the Table 7.3 we can see that the behavior is more consistent among all the configurations and frequencies, with peaks that reach even 5 out of 10 benchmarks. The downside of this analysis is that this denoising methodology produces more false positive results, so the results must be taken with caution and to further be analyzed to be included **with confidence** in the feature extraction process.

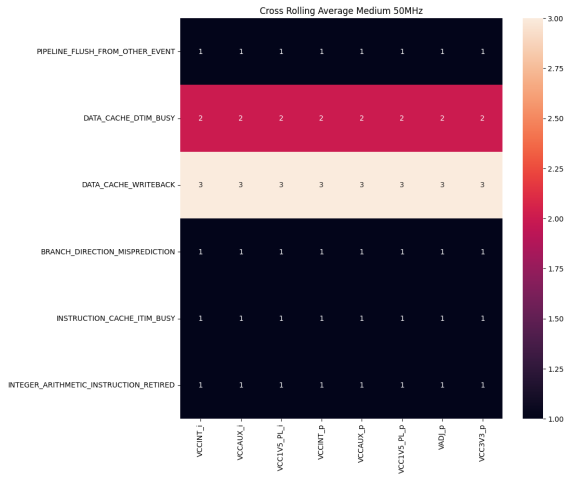
7.5.3 Cross Rolling Window Analysis



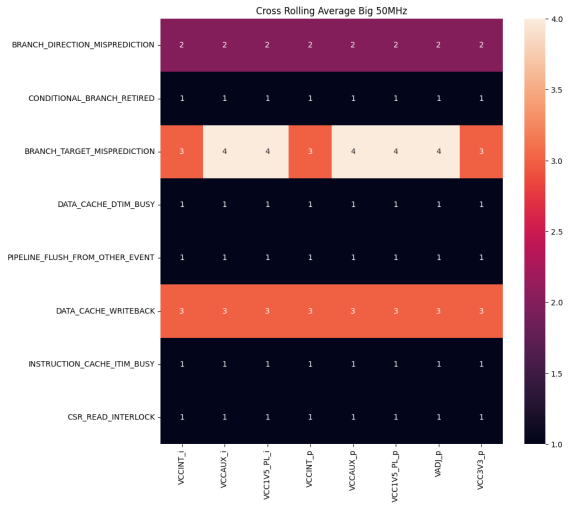
(a) Mid 12.5MHz



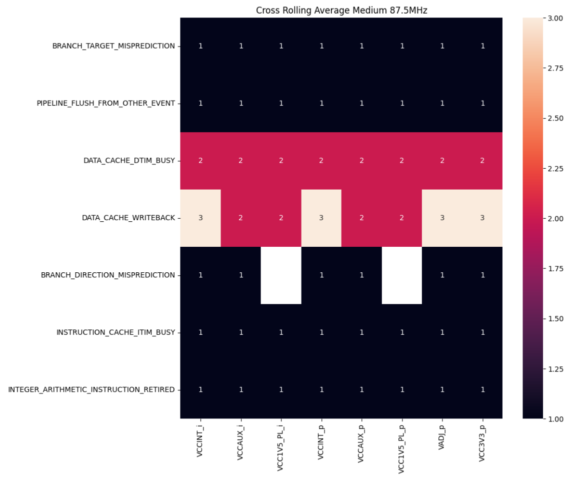
(b) Big 12.5MHz



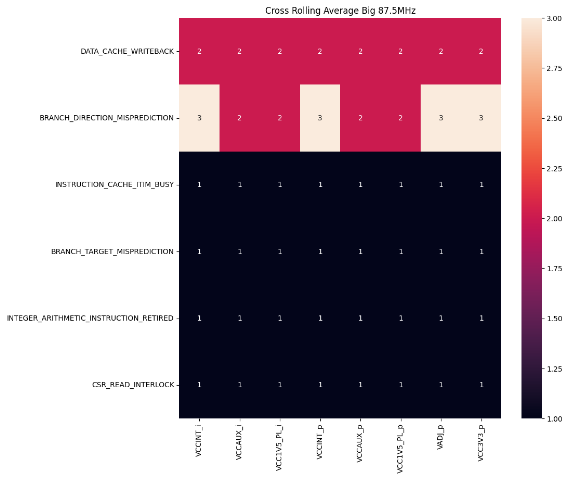
(c) Mid 50MHz



(d) Big 50MHz



(e) Mid 87.5MHz



(f) Big 87.5MHz

Figure 7.5.3: Across Benchmarks Cross Rolling Window Analysis

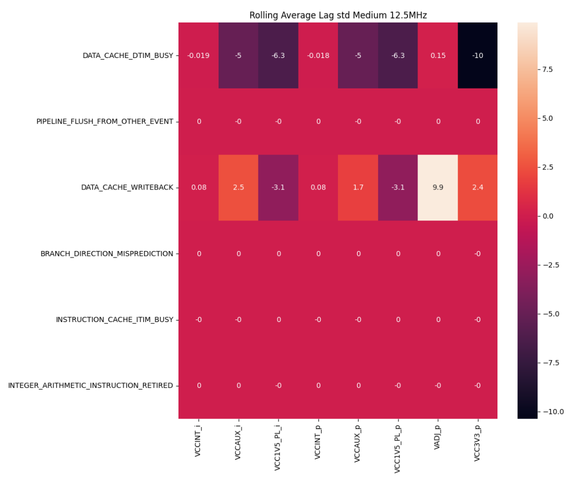
In the analysis performed in Fig. 7.5.3 are presented the results with the same constrains as in the static analysis except that instead of the top 5 common correlations we have the top 5 common peak correlations across the time lags. The results are presented in Table 7.4.

Configuration	Common Counter-Rail Behavior
Big	**DATA_CACHE_WRITEBACK**
	BRANCH_DIRECTION_MISPREDICTION
	BRANCH_TARGET_MISPREDICTION
Medium	**DATA_CACHE_WRITEBACK**
	DATA_CACHE_DTIM_BUSY

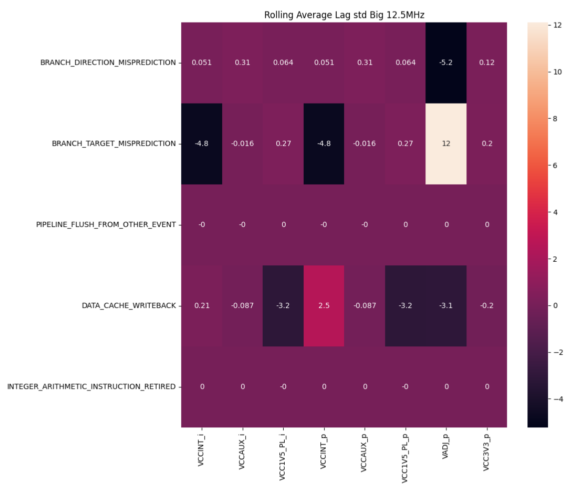
Table 7.4: Common Counter-Rail Behavior Across Benchmarks (Cross Rolling)

From the results in Table 7.4 we can see the common counters are the same in a great degree with the results in the static rolling window analysis (Table 7.2). This result is compatible with the data, because they were performed in the same power measurements but we can see that in almost every configuration the results per counter are the same in all the rails (even if the rails must not be corralated with the results such as **VADJ** and **VCC3V3**).

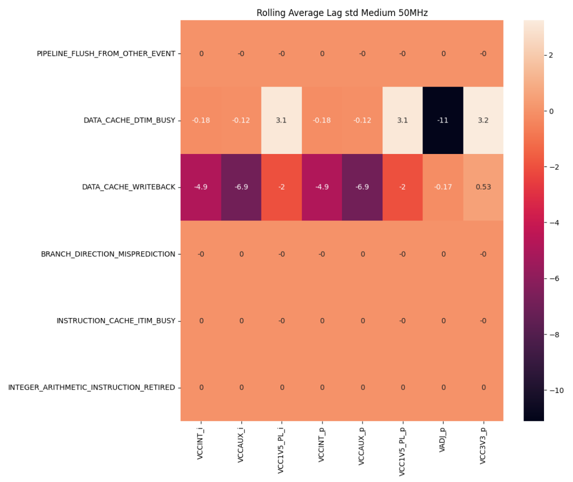
In order to analyze this homogenous behavior we need again to focus in the **the lags** of the peaks.



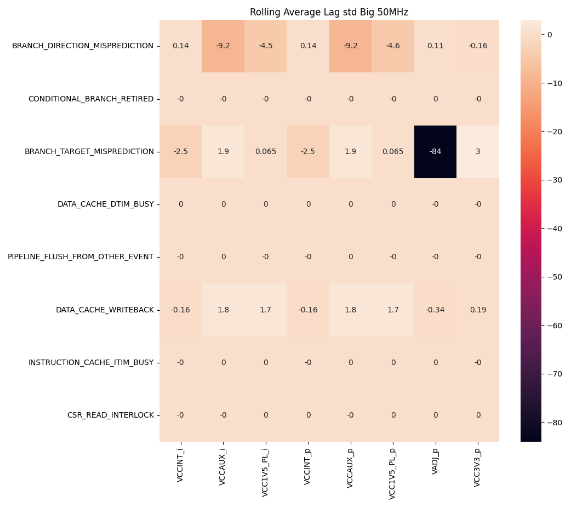
(a) Mid 12.5MHz



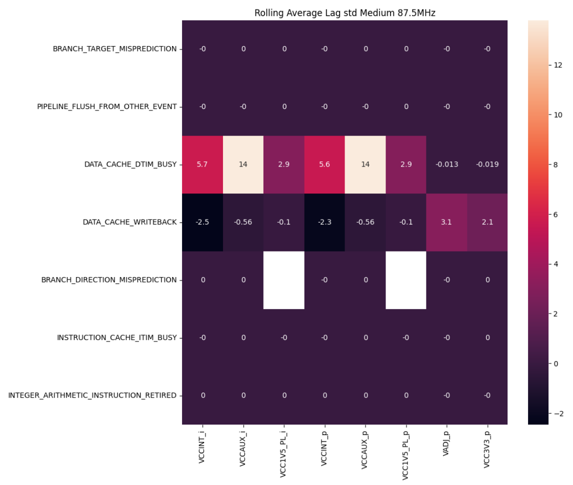
(b) Big 12.5MHz



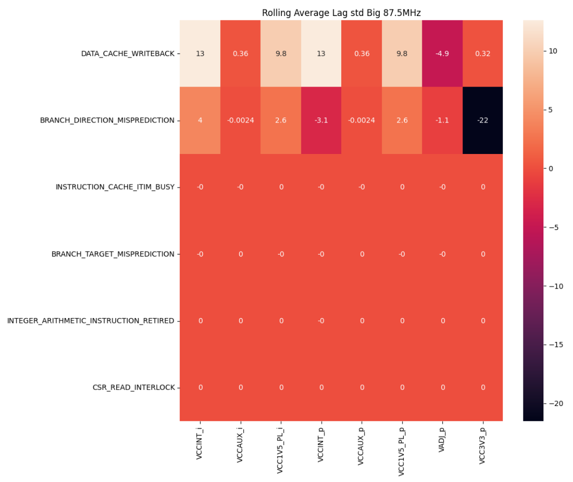
(c) Mid 50MHz



(d) Big 50MHz



(e) Mid 87.5MHz



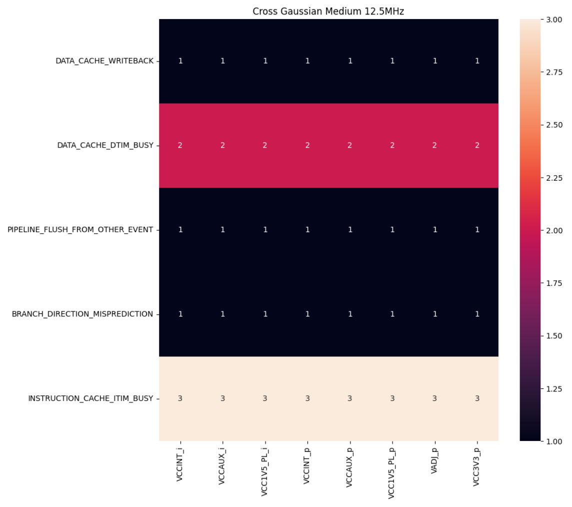
(f) Big 87.5MHz

Figure 7.5.4: Across Benchmarks Cross Rolling Window Time Lag Analysis

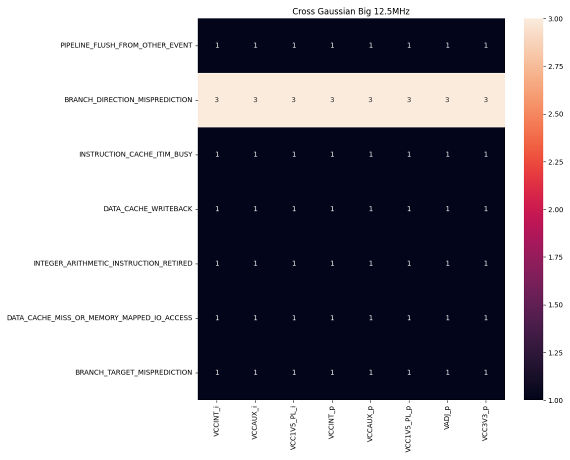
In the results presented in Fig. 7.5.4 we have calculated the **Coefficient of Variation (CV)** ($CV = \frac{|\sigma|}{\mu}$), from the across benchmark top 5 lag analysis. The CV is a standardized measure that evaluates the relative variability of data in relation to their means, independent of their scales or units and is calculated to validate the common lags. Also if a counter appears only on one benchmark, then std will be zero, so we will only focus in the cross benchmark common lags. We used absolute value in the variance part of the CV calculation in order to have the correct future/past orientation of the lags.

Concluding, from those results we can see that there is no across benchmark consistent behavior in the cross-correlated performance counters, which is a reasonable result due to the fact that the benchmarks target different execution flows and even if those flows are similar, the do not happen in the same time frame. This result was expected and is also validation of our methodology and a starting point of proof that **the predictions must not be application-agnostic**.

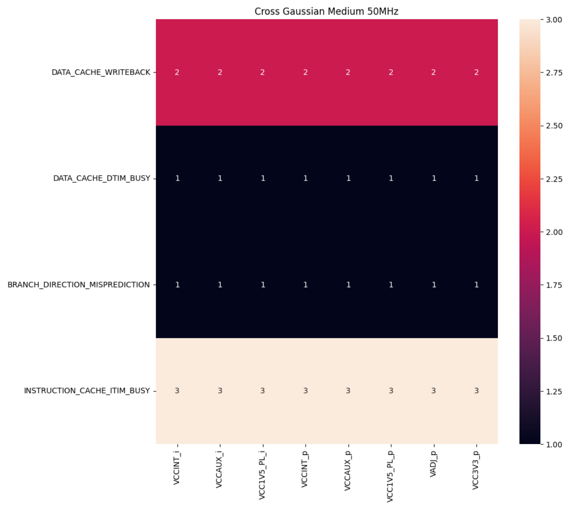
7.5.4 Cross Gaussian Filter Analysis



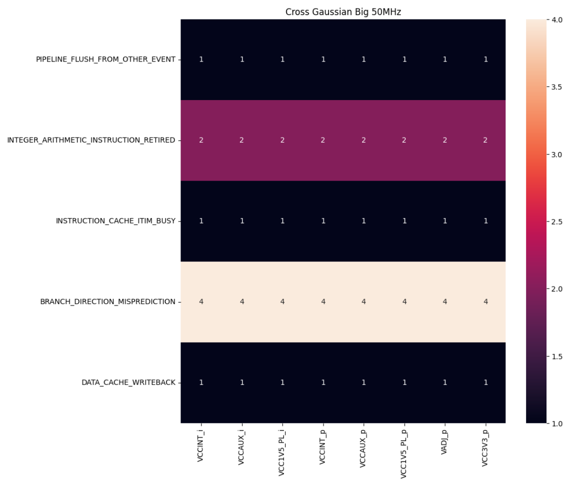
(a) Mid 12.5MHz



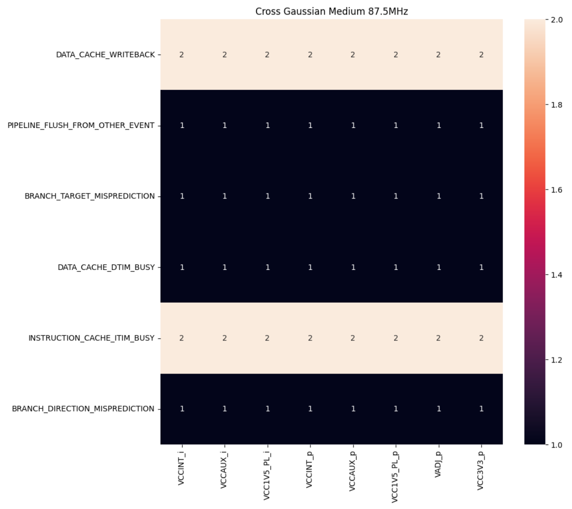
(b) Big 12.5MHz



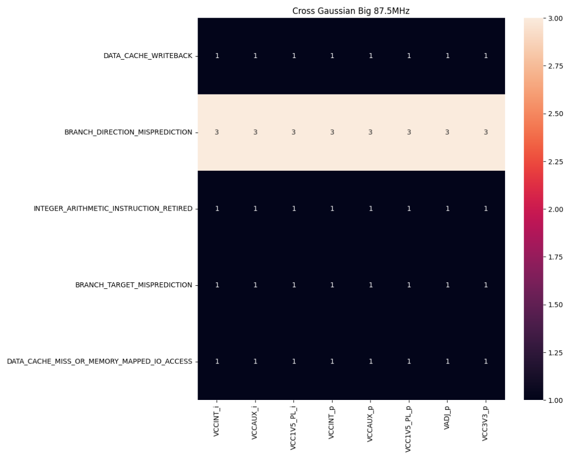
(c) Mid 50MHz



(d) Big 50MHz



(e) Mid 87.5MHz



(f) Big 87.5MHz

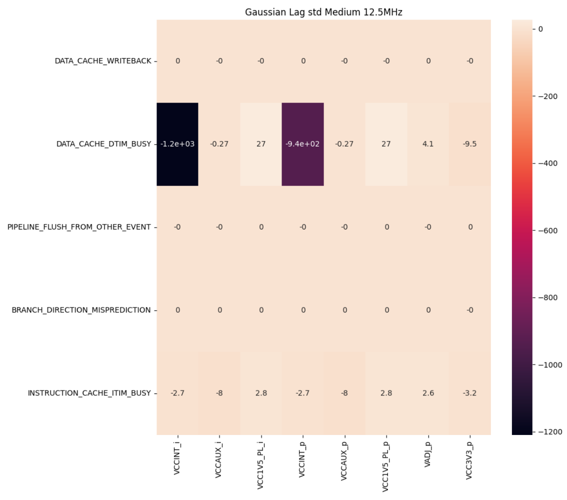
Figure 7.5.5: Across Benchmarks Cross Gaussian Analysis

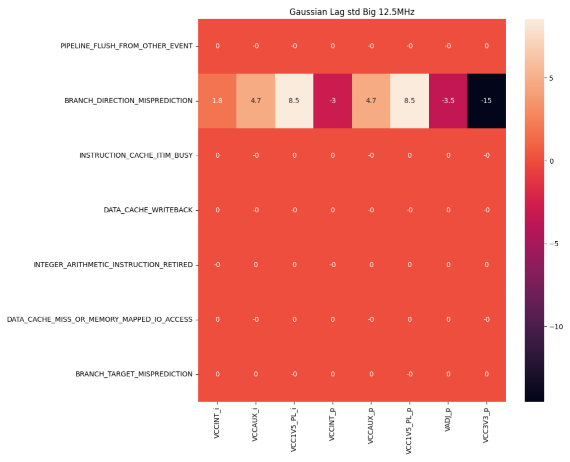
Performing analysis on the cross correlated data had the same processing constrains as the *Cross Rolling Window Analysis* and the results are presented in Table 7.5.

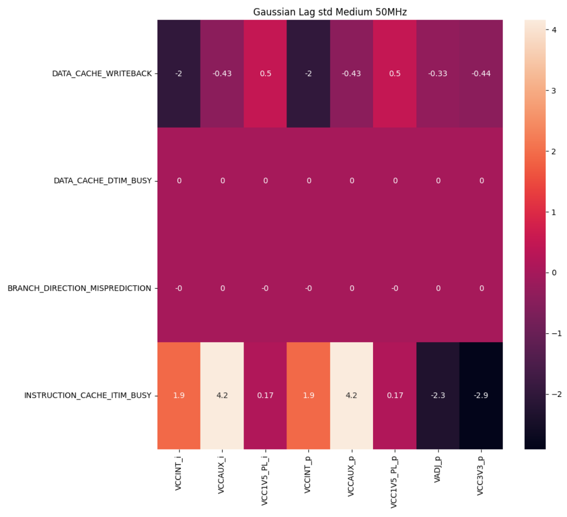
Configuration	Common Counter-Rail Behavior
Big	BRANCH_DIRECTION_MISPREDICTION
Medium	DATA_CACHE_WRITEBACK INSTRUCTION_CACHE_ITIM_BUSY

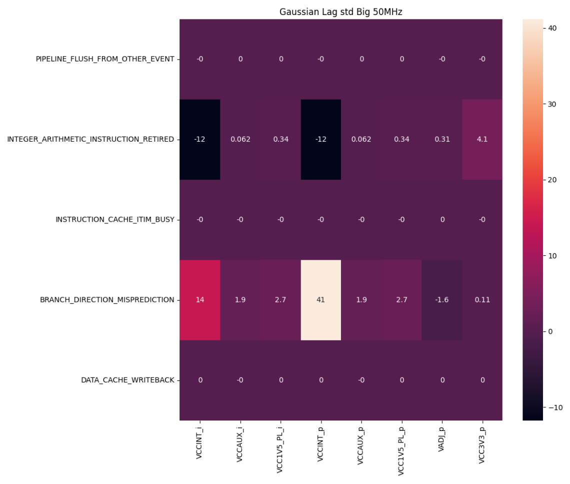
Table 7.5: Common Counter-Rail Behavior Across Benchmarks (Cross Gaussian)

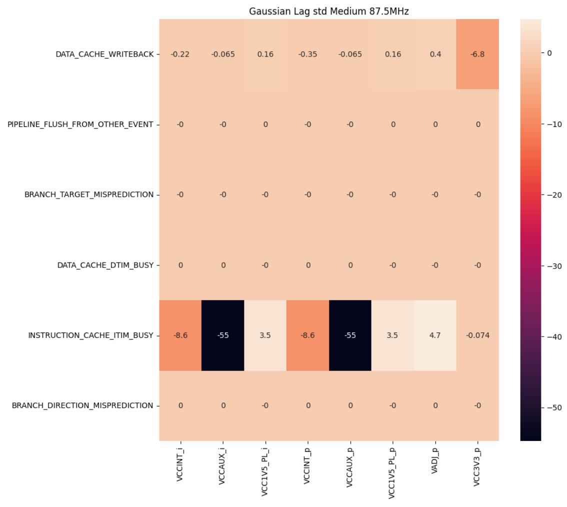
From the results in Table 7.5 we can see that there are **no common counters** across the configurations, although there are still common points with the static analysis performed in previous sections. Also we can see also that there are a **homogenous result** across all the rails in this analysis, so again we need to also focus in the **time lags** of the peaks.


(a) Mid 12.5MHz


(b) Big 12.5MHz


(c) Mid 50MHz


(d) Big 50MHz


(e) Mid 87.5MHz

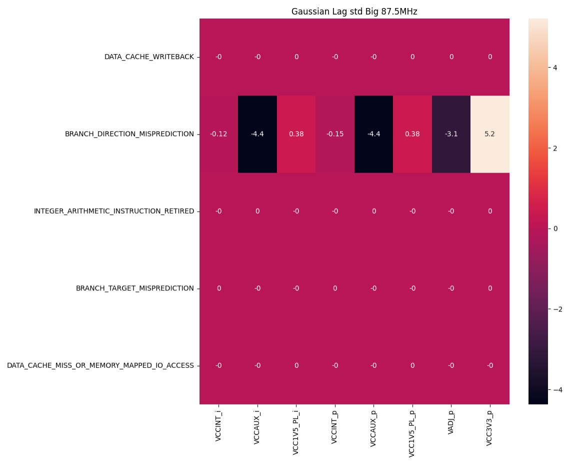

(f) Big 87.5MHz

Figure 7.5.6: Across Benchmarks Cross Gaussian Time Lag Analysis

In the results presented in Fig. 7.5.6 we have again calculating the **Coefficient of Variation (CV)** of the across benchmark correlations. We can see that again there are not a consistent behavior across benchmarks (high CV values) which is an expected result, which also enhances the need of a **benchmark specific** prediction and feature extraction process, as also mentioned in the previous sections with similar result generation.

Chapter 8

Conclusions - Future Work

8.1 Conclusions

In this research we managed to dive into the design space exploration of power analysis and correlation based feature extraction in a RISC-V ISA based SoC called RocketChip emulated in an ZC706 FPGA board and managed to:

- Implement an emulated adaptive power monitoring system, gathering performance counter data from the RocketChip SoC.
- Produce Data in a certain Real-Time Scenario.
- Analyze the Data Using Different Denoising Techniques.
- Analyze the Data Using Different Correlation Techniques.
- Gathering observations and analysis of the different behaviors across 3 frequencies and 2 RocketChip implementations.

Those implementations lead to answer the basic questions of the research around a power model for the RocketChip core. The results of the research are:

RQ1: Can RocketChip Provide Enough Performance Counters to be Used as Input?

Ans : Yes, if we test certain components but more (custom) counters will produce more building block specific measurements.

RQ2: Can the Analysis we are Going to Perform Lead to an Application-Agnostic Prediction?

Ans : No based on this analysis because we observed major behaviour changes in certain benchmarks.

RQ3: Should we Have Memory for an Accurate Prediction?

Ans : Yes, because we can see that in the cross-correlation section we have major value changes and, even if this behaviour is wrong, the memory will help us prune these cases.

RQ4: Are the Correlation Data Sensitive to Configuration Changes?

Ans : Yes, as we can see in the Across-Configurations results 7.

RQ5: Can a denoising-based data Preprocessing Lead to Better Power/Performance Data Analysis?

Ans : The Rolling Average gave more specific and "clear" results. In the other hand Gaussian Filter gave a smoother pattern which can easily lead to more false positive cases.

This first step into power modeling resulted as an interesting research topic that can be further explored in the future, because as already mentioned the parameters that need to be independently explored are many and very important for achieving power modeling in a RISC-V ISA environment.

8.2 Future Work

Based on the research the topics that needs to be explored in the future are:

- Gain Activation Data From Building Blocks and Cross-Validate our Assumption.
- Evaluate the Dataset Using DVFS Techniques.
- Design Specific Performance Counters (Target Specific Building Blocks).
- Adapt This Technique to Cores Outside RocketChip.

Appendix A

Correlation Data

All the correlation data that are used as a research output and presented in chapter 7 are available in the following link: data. The data are all in the form of heatmaps with titles indicating the target of each output.

Bibliography

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey,” en, *Heliyon*, vol. 4, no. 11, e00938, Nov. 2018, ISSN: 24058440. DOI: 10.1016/j.heliyon.2018.e00938. [Online]. Available: (visited on 03/24/2024).

[2] T. Kolpe, A. Zhai, and S. S. Sapatnekar, “Enabling improved power management in multicore processors through clustered dvfs,” in *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763052.

[3] S. Di Mascio, A. Menicucci, G. Furano, C. Monteleone, and M. Ottavi, “The Case for RISC-V in Space,” en, in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara and A. De Gloria, Eds., vol. 573, Series Title: Lecture Notes in Electrical Engineering, Cham: Springer International Publishing, 2019, pp. 319–325, ISBN: 978-3-030-11972-0 978-3-030-11973-7. DOI: 10.1007/978-3-030-11973-7_37. [Online]. Available: (visited on 02/09/2024).

[4] E. Cui, T. Li, and Q. Wei, “RISC-V Instruction Set Architecture Extensions: A Survey,” *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3246491. [Online]. Available: (visited on 03/01/2024).

[5] I. Elsadek and E. Y. Tawfik, “RISC-V Resource-Constrained Cores: A Survey and Energy Comparison,” in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, Toulon, France: IEEE, Jun. 2021, pp. 1–5, ISBN: 978-1-66542-429-5. DOI: 10.1109/NEWCAS50681.2021.9462781. [Online]. Available: (visited on 02/09/2024).

[6] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, “Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications,” in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Thessaloniki: IEEE, Sep. 2017, pp. 1–8, ISBN: 978-1-5090-6462-5. DOI: 10.1109/PATMOS.2017.8106976. [Online]. Available: (visited on 02/09/2024).

[7] J. Veiga, J. Enes, R. Expósito, and J. Touriño, “Bdev 3.0: Energy efficiency and microarchitectural characterization of big data processing frameworks,” *Future Generation Computer Systems*, vol. 86, Apr. 2018. DOI: 10.1016/j.future.2018.04.030.

[8] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, “RAPL in Action: Experiences in Using RAPL for Power Measurements,” en, *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 3, no. 2, pp. 1–26, Jun. 2018, ISSN: 2376-3639, 2376-3647. DOI: 10.1145/3177754. [Online]. Available: (visited on 02/10/2024).

[9] G. Fieni, R. Rouvoy, and L. Seiturier, “Selfwatts: On-the-fly selection of performance events to optimize software-defined power meters,” in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 324–333. DOI: 10.1109/CCGrid51090.2021.00042.

[10] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, “A Study on the Use of Performance Counters to Estimate Power in Microprocessors,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 882–886, Dec. 2013, ISSN: 1549-7747, 1558-3791. DOI: 10.1109/TCSII.2013.2285966. [Online]. Available: (visited on 03/04/2024).

[11] M. Y. Lim, A. Porterfield, and R. Fowler, “SoftPower: Fine-grain power estimations using performance counters,” en, in *Proceedings of the 19th ACM International Symposium on High Per-*

formance Distributed Computing, Chicago Illinois: ACM, Jun. 2010, pp. 308–311, ISBN: 978-1-60558-942-8. DOI: 10.1145/1851476.1851517. [Online]. Available: (visited on 02/06/2024).

[12] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Counter-Based Power Modeling Methods: Top-Down vs. Bottom-Up," en, *The Computer Journal*, vol. 56, no. 2, pp. 198–213, Feb. 2013, ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/bxs116. [Online]. Available: (visited on 03/04/2024).

[13] G. L. T. Chetsa, L. Lefevre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Beyond CPU Frequency Scaling for a Fine-grained Energy Control of HPC Systems," in *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, New York, NY, USA: IEEE, Oct. 2012, pp. 132–138, ISBN: 978-0-7695-4907-1 978-1-4673-4790-7. DOI: 10.1109/SBAC-PAD.2012.32. [Online]. Available: (visited on 02/06/2024).

[14] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*, ser. ICS '10, Tsukuba, Ibaraki, Japan: Association for Computing Machinery, 2010, pp. 147–158, ISBN: 9781450300186. DOI: 10.1145/1810085.1810108. [Online]. Available:

[15] S. Sankaran and R. Sridhar, "Energy modeling for mobile devices using performance counters," in *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Columbus, OH, USA: IEEE, Aug. 2013, pp. 441–444, ISBN: 978-1-4799-0066-4. DOI: 10.1109/MWSCAS.2013.6674680. [Online]. Available: (visited on 02/10/2024).

[16] E. Vasilakis, I. Sourdis, V. Papaefstathiou, A. Psathakis, and M. G. Katevenis, "Modeling energy-performance tradeoffs in ARM big.LITTLE architectures," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Thessaloniki: IEEE, Sep. 2017, pp. 1–8, ISBN: 978-1-5090-6462-5. DOI: 10.1109/PATMOS.2017.8106950. [Online]. Available: (visited on 02/10/2024).

[17] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Montreal, QC, Canada: IEEE, Sep. 2013, pp. 1–10, ISBN: 978-1-4799-1400-5. DOI: 10.1109/CASES.2013.6662519. [Online]. Available: (visited on 03/04/2024).

[18] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "Apollo: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21, Virtual Event, Greece: Association for Computing Machinery, 2021, pp. 1–14, ISBN: 9781450385572. DOI: 10.1145/3466752.3480064. [Online]. Available:

[19] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "McPAT-Calib: A RISC-V BOOM Microarchitecture Power Modeling Framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 243–256, Jan. 2023, ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2022.3169464. [Online]. Available: (visited on 02/29/2024).

[20] A. K. A. Kumar and A. Gerstlauer, "Learning-Based CPU Power Modeling," in *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, Canmore, AB, Canada: IEEE, Sep. 2019, pp. 1–6, ISBN: 978-1-72815-758-0. DOI: 10.1109/MLCAD48534.2019.9142100. [Online]. Available: (visited on 02/29/2024).

[21] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "PULP: A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision," en, *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 339–354, Sep. 2016, ISSN: 1939-8018, 1939-8115. DOI: 10.1007/s11265-015-1070-9. [Online]. Available: (visited on 02/29/2024).

[22] *Ri5cy*, https://github.com/openhwgroup/cv32e40p. [Online]. Available:

[23] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *International Conference on Green Computing*, Chicago, IL, USA: IEEE, Aug. 2010, pp. 115–122, ISBN: 978-1-4244-7612-1. DOI: 10.1109/GREENCOMP.2010.5598315. [Online]. Available: (visited on 02/08/2024).

[24] Y. Ni, Y. Kim, T. Rosing, and M. Imani, "Online Performance and Power Prediction for Edge TPU via Comprehensive Characterization," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium: IEEE, Mar. 2022, pp. 612–615, ISBN: 978-3-9819263-6-1. DOI: 10.23919/DATE54114.2022.9774764. [Online]. Available: (visited on 02/10/2024).

[25] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[26] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a Scala embedded language," en, in *Proceedings of the 49th Annual Design Automation Conference*, San Francisco California: ACM, Jun. 2012, pp. 1216–1225, ISBN: 978-1-4503-1199-1. DOI: 10.1145/2228360.2228584. [Online]. Available: (visited on 03/01/2024).

[27] *Fesvr*, https://github.com/riscvarchive/riscv-fesvr. [Online]. Available:

[28] *Spike*, https://github.com/riscv-software-src/riscv-isa-sim. [Online]. Available:

[29] *Risc-v proxy kernel*, https://github.com/riscv-software-src/riscv-pk. [Online]. Available:

[30] *Axi*, https://www.xilinx.com/products/intellectual-property/axi.html. [Online]. Available:

[31] D. G. Bonett and T. A. Wright, "Sample size requirements for estimating pearson, kendall and spearman correlations," en, *Psychometrika*, vol. 65, no. 1, pp. 23–28, Mar. 2000, ISSN: 0033-3123, 1860-0980. DOI: 10.1007/BF02294183. [Online]. Available: (visited on 02/07/2024).

[32] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA: IEEE, Sep. 2019, pp. 1–9, ISBN: 978-1-72815-020-8. DOI: 10.1109/HPEC.2019.8916327. [Online]. Available: (visited on 02/01/2024).

[33] R. Zamani and A. Afsahi, "A study of hardware performance monitoring counter selection in power modeling of computing systems," in *2012 International Green Computing Conference (IGCC)*, San Jose, CA, USA: IEEE, Jun. 2012, pp. 1–10, ISBN: 978-1-4673-2154-9 978-1-4673-2155-6 978-1-4673-2153-2. DOI: 10.1109/IGCC.2012.6322289. [Online]. Available: (visited on 03/04/2024).

[34] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," en, *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, May 2009, ISSN: 0163-5964. DOI: 10.1145/1577129.1577137. [Online]. Available: (visited on 02/06/2024).

[35] M. Sagi, N. A. V. Doan, M. Rapp, T. Wild, J. Henkel, and A. Herkersdorf, "A Lightweight Nonlinear Methodology to Accurately Model Multicore Processor Power," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3152–3164, Nov. 2020, ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2020.3013062. [Online]. Available: (visited on 03/04/2024).

[36] *Risc-v isa*, https://riscv.org/technical/specifications/. [Online]. Available: