Μελέτη Συσχέτισης της Κατανάλωσης Ισχύος με την Μεταβολή των Μετρητών Απόδοσης ενός Μικροεπεξεργαστή

Γεώργιος Κυμπαρίδης

Μεταπτυχιακή Εργασία Εξειδίκευσης

— ♦ —

Ιωάννινα, Ιούλιος 2022

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Μελέτη Συσχέτισης της Κατανάλωσης Ισχύος με την Μεταβολή των Μετρητών Απόδοσης ενός Μικροεπεξεργαστή

Η Μεταπτυχιακή Διπλωματική Εργασία

υποβάλλεται στην ορισθείσα

από τη Συνέλευση

του Τμήματος Μηχανικών Η/Υ και Πληροφορικής

Εξεταστική Επιτροπή

από τον

Γεώργιο Κυμπαρίδη

ως μέρος των υποχρεώσεων για την απόκτηση του

ΔΙΠΛΩΜΑΤΟΣ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΣΤΗ ΜΗΧΑΝΙΚΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕ ΕΙΔΙΚΕΥΣΗ

ΣΤΑ ΠΡΟΗΓΜΕΝΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Ιωάννινα 2022

Εξεταστική Επιτροπή:

- **Ευθυμίου Αριστείδης**, Επίκουρος Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων (Επιβλέπων)
- **Τσιατούχας Γεώργιος**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Τενέντες Βασίλειος**, Επίκουρος Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων

Περιεχομενα

Κατάλογος Σχημάτων **iii**

Κατάλογος Πινάκων **vi**

Περίληψη **vii**

Extended Abstract **ix**

1 Εισαγωγή **1**

1.1 Στόχοι 1

1.2 Δομή της Διατριβής 2

2 Υπόβαθρο **3**

2.1 Αρχιτεκτονική Συνόλου Εντολών RISC-V 4

2.1.1 Αρχιτεκτονική Συνόλου Εντολών 4

2.1.2 Μορφή Εντολών 5

2.1.3 Hardware Performance Counters 5

2.2 Ibex: An embedded 32 bit RISC-V CPU Core 6

2.2.1 Η Αρχιτεκτονική του Ibex 7

2.3 Stress Test και Θερμικοί Ιοί 10

2.3.1 GeST: Ένα Αυτόνομο Πλαίσιο Λογισμικού για την Δημιουργία CPU Stress-Tests 11

2.4 Εργαλεία Προσομοίωσης Επεξεργαστή 13

2.4.1 Fusesoc-Εργαλείο Διαχείρισης Πακέτων 13

2.4.2 Verilator-Προσομοιωτής Γλώσσας Περιγραφής Υλικού 13

2.4.3 Xilinx Vivado 14

2.5 FPGA-Field Programmable Gate Array 14

2.6 Νευρωνικά Δίκτυα και Βαθιά Μάθηση 14

2.6.1 Βαθιά Νευρωνικά Δίκτυα 15
2.6.2 Συνάρτηση Ενεργοποίησης 16
2.6.3 Υπερπαράμετροι 16
2.6.4 Βελτιστοποιητές 17
2.6.5 Υπερεκπαίδευση και Υποεκπαίδευση 17

3 Υλοποίηση **18**
3.1 Προσθήκη Επίπλεον Performance Counters 19
3.2 Διαμόρφωση του GeST 20
3.3 Σύνδεση του GeST και Προσομοίωση 23
3.4 Σύνδεση του GeST και Μετρήσεις Πραγματικών Τιμών με το Arty A7 24
3.4.1 Λήψη και Αποθήκευση της Τιμής του Ρεύματος Μέσω XADC Wizard 25
3.4.2 Αποθήκευση των Τιμών των Performance counters στην Μνήμη 27
3.4.3 Σειριακή Επικοινωνία Board-Υπολογιστή με Χρήση του Uart Protocol 27
3.5 Περιγραφή Νευρωνικού Δικτύου 33

4 Αποτελέσματα και Αξιολόγηση **36**
4.1 Αποτελέσματα Προσομοίωσης 37
4.2 Αποτελέσματα Μετρήσεων στο FPGA 38
4.3 Σύγκριση και Αξιολόγηση των Μεθόδων Μέτρησης 42
4.4 Συσχέτιση των Αποτελεσμάτων με Μετροπρογράμματα 47
4.5 Αποτελέσματα Νευρωνικού δικτύου 48

5 Συμπεράσματα **50**
5.1 Συμπεράσματα 50
5.2 Μελλοντικές Επεκτάσεις 51

Βιβλιογραφία **52**

A Σχήματα με Προγράμματα της εργασίας **54**

Κατάλογος Σχημάτων

2.1 Μορφές εντολών RISC-V 5
2.2 RISC-V Μετρητές Επίδοσης 6
2.3 Ibex,διοχέτευση δύο σταδίων 8
2.4 Γενετικός Αλγόριθμος 12
2.5 Συνολική Εικόνα του Gest 13
2.6 Επίπεδα Νευρωνικού Δικτύου 16

3.1 Ορισμός Εντολών RISC-V 21
3.2 Σύνδεση GeST-Ibex στα πλαίσια Προσομοίωσης 24
3.3 FPGA Core Supply παρακολούθηση ρεύματος 25
3.4 Διάγραμμα XADC 26
3.5 Απεικόνιση Πακέτου του Uart 28
3.6 Διάγραμμα Καταστάσεων Receiver 28
3.7 Διάγραμμα Καταστάσεων Transmitter 29
3.8 Διάγραμμα Καταστάσεων Split Bytes 30
3.9 Διάγραμμα Καταστάσεων Memory Controller 31
3.10 Διάγραμμα Καταστάσεων Controller 32
3.11 Σειρική Επικοινωνία Uart 33
3.12 Απεικόνιση Νευρωνικού Δικτύου 35

4.1 Η τιμή της κατανάλωσης Ισχύος για κάθε Individual ls=2.047,ls=8.188 38
4.2 Μέγιστη τιμή κατανάλωσης Ισχύος για κάθε population ls=2.047,ls=8.188 38
4.3 Πορεία των τιμών όλων των Performance Counters 39
4.4 Performance Counters με την μεγαλύτερη συσχέτιση στις μετρήσεις . . 40
4.5 Τιμές Ρεύματος για κάθε Individual ls=2.047,ls=1.000.000,ls=10.000.000 41

4.6 Μέγιστη τιμή Ρεύματος για κάθε population ls=2.047,ls=1.000.000,ls=10.000.000 42
4.7 Πίνακας Συσχέτισης ls=2.047,ls=8.188 45
4.8 Correlation Matrix ls=2.047,ls=1.000.000,ls=10.000.000 46
4.9 Σφάλμα του Νευρωνικού Δικτύου στο Test Set 49

Κατάλογος Σχημάτων Προγραμμάτων

3.1 Κομμάτι κώδικα main_original που απεικονίζει τον κενό βρόγχο . . . 23
3.2 Εντολή Assembly και pointer για αποθήκευση Performance counters στην μνήμη 27
3.3 Configuration for the serial port 32
A.1 Κώδικας main_original που απεικονίζει το κενό βρόγχο 54
A.2 Κώδικας γλώσσας C όπου φορτώνεται στο fpga 55
A.3 Κώδικας της κλάσης MeasurementHPerfCounters-Vivado - makeMeasurement-PerfCounters-Vivado script 57

Κατάλογος Πινάκων

2.1 Μετρητές Επίδοσης 10
2.2 Παράμετροι GeST 12

3.1 Ορισμός Εντολών RISC-V 22

4.1 Simulation Results 42
4.2 FPGA Results 42
4.3 Αποτελέσματα των μέγιστων τιμών ανά επανάληψη 42
4.4 Αποτελέσματα για διαφορετικό πλήθος επαναλήψεων 43
4.5 Αποτελέσματα μετροπρογραμμάτων 47
4.6 Αποτελέσματα Benchmark,Προσομοίωσης,FPGA 48
4.7 Αποτελέσματα Νευρωνικού Δικτύου 49
4.8 Πρόβλεψη Νευρωνικού για τις τιμές των Προγραμμάτων Benchmark . 49

Περιληψη

Γεώργιος Κυμπαρίδης, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2022.
Μελέτη Συσχέτισης της Κατανάλωσης Ισχύος με την Μεταβολή των Μετρητών Απόδοσης ενός Μικροεπεξεργαστή.
Επιβλέπων: Αριστείδης Ευθυμίου, Επίκουρος Καθηγητής.

Πρώτης τάξης αντικείμενο μελέτης και έρευνας, αλλά και βασικό κριτήριο αξιολόγησης ενός κυκλώματος, είναι η κατανάλωση ισχύος. Η ανεξέλεγκτη κατανάλωση ισχύος και η συνεπακόλουθη αύξηση της θερμοκρασίας ενός ολοκληρωμένου κυκλώματος, εγείρουν ένα μεγάλο φάσμα προβλημάτων αξιοπιστίας και ασφάλειας, από την μείωση της αναμενόμενης διάρκειας ζωής του κυκλώματος και την μείωση της επίδοσης, μέχρι και την καταστροφή του.

Αυξάνοντας το κατασκευαστικό κόστος, οι περισσότεροι επεξεργαστές προσθέτουν ενσωματωμένους αισθητήρες που μετρούν συνεχώς την ισχύ και την θερμοκρασία έτσι ώστε αν ξεπεραστούν κάποια όρια, περιορίζουν τον ρυθμό εκτέλεσης, ή σε ακραίες περιπτώσεις θέτουν το σύστημα εκτός λειτουργίας για να προστατευθεί η ακεραιότητά του. Ωστόσο, οι σύγχρονοι επεξεργαστές υλοποιούν και μετρητές επίδοσης (Hardware Performance Counters), που μετρούν συμβάντα που σχετίζονται με την λειτουργία τους όπως οι κύκλοι εκτέλεσης, οι εντολές προσκόμισης, κ.α. Η παρούσα διπλωματική εργασία μελετά τη συσχέτιση του ρυθμού αύξησης των ενσωματωμένων μετρητών με την κατανάλωση ισχύος ώστε να ανιχνεύεται υπέρβαση ορίων κατανάλωσης ισχύος χωρίς να είναι απαραίτητοι οι αισθητήρες. Ο απώτερος στόχος είναι η εξάλειψη των αισθητήρων από επεξεργαστές χαμηλού κόστους χωρίς να τίθεται σε κίνδυνο η ασφάλεια και η αξιοπιστία των επεξεργαστών.

Για τους σκοπούς της εργασίας υιοθετήθηκε ένας απλός επεξεργαστής αρχιτεκτονικής RISC-V και τροποποιήθηκε κατάλληλα ώστε να περιλαμβάνει επιπλέον

μετρητές επίδοσης. Για την πρόκληση μεγάλης κατανάλωσης ισχύος στον επεξεργαστή απαιτούνται ειδικά προγράμματα καταπόνησης που είναι γνωστά ως θερμικοί ιοί (power virus). Καθώς τέτοια προγράμματα δεν υπάρχουν διαθέσιμα στον RISC-V, τροποποιήθηκε ένα πλαίσιο λογισμικού, βασισμένο σε γενετικούς αλγορίθμους, και που εστίαζε στις αρχιτεκτονικές ARM και x86, ώστε να παράγει θερμικούς ιούς για RISC-V.

Αρχικά η ανάπτυξη και ο πειραματισμός έγιναν με προσομοίωση. Ο επεξεργαστής προσομοιώνεται σε επίπεδο συμπεριφοράς (behavioral simulation) για να εξαχθούν οι τιμές των μετρητών επίδοσης και στη συνέχεια σε δεύτερο στάδιο υλοποιείται μια λειτουργική προσομοίωση (functional simulation) για την καταγραφή της δραστηριότητας μεταγωγής. Αυτή η δραστηριότητα μεταγωγής χρησιμοποιείται στη συνέχεια για τη δημιουργία μιας λεπτομερούς αναφοράς της κατανάλωσης ισχύος. Επειδή η διαδικασία αυτή ήταν πολύ χρονοβόρα,ο πειραματισμός μεταφέρθηκε σε μετρήσεις σε πραγματικό χρόνο με χρήση πλακέτας FPGA. Προστέθηκε στην υλοποίηση του επεξεργαστή στην FPGA ένα κύκλωμα σειριακής επικοινωνίας (UART) καθώς και ένας μετατροπέας αναλογικού σε ψηφιακό σήμα (ADC) που μετράει το ρεύμα της πηγής τροφοδοσίας του επεξεργαστή. Έτσι ένας υπολογιστής συνδέεται με την πλακέτα FPGA του επεξεργαστή, στέλνει το πρόγραμμα προς εκτέλεση και, στο τέλος της εκτέλεσης παίρνει τους μετρητές και το μέσο ρεύμα κατανάλωσης μέσω της σειριακής θύρας.

Η επεξεργασία των μετρήσεων που πραγματοποιήθηκαν έδειξε ότι πράγματι κάποιοι μετρητές επίδοσης έχουν μεγάλη συσχέτιση με την συνολική κατανάλωση ισχύος του επεξεργαστή. Επιπλέον με έμπνευση από την βιβλιογραφία, υλοποιήθηκε ένα απλό νευρωνικό δίκτυο που με είσοδο τις κανονικοποιημένες τιμές των μετρητών επίδοσης, προβλέπει το μέσο ρεύμα κατανάλωσης με μεγάλη ακρίβεια.

Extended Abstract

Georgios Kymparidis, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2022.
Study of the Correlation between power consumption and the rise rate of performance counters of a microprocessor.
Advisor: Aristides Efthymioy, Assistant Professor.

Power-consumption in integrated circuits is a first-class research problem and a major evaluation metric. The uncontrolled consumption of power and its consequent temperature increase of an integrated circuit, raises a large spectrum of reliability and safety problems: from the reduction of the expected lifetime of the circuit and the reduction in performance, up to its destruction.

Increasing manufacturing costs, most processors add embedded sensors that continuously measure power and the temperature so that, if certain limits are exceeded, they can limit the execution rate, or in extreme cases shut the system down to protect its integrity. However, modern processors also implement hardware performance counters, which count events related to their operation such as execution cycles, instruction fetches, etc. This thesis studies the correlation of the growth rate of embedded performance counters with power consumption in order to detect when the power consumption is exceeding safe limits. The ultimate goal is to eliminate sensors from low-cost processors without compromising their safety and reliability.

For the purposes of this work, a simple, RISC-V processor was adopted and suitably modified to provide additional performance counters. In order to cause high power consumption on the processor, special stress-test programs known as power viruses are required. As such programs are not available for RISC-V, a genetic-algorithm-based software framework, which is originally focused on ARM and x86 architectures, was adapted to produce thermal viruses for RISC-V.

Initially the development and experimentation was done by simulation. The processor is simulated at the behavioral level to extract the performance counter values and at then a second functional-level simulation is performed to record the switching activity. This switching activity is then used to generate a detailed report of power consumption. Because this process was too time-consuming, the experimentation was transferred to real-time measurements using an FPGA board. A serial communications circuit (UART) was added to the implementation of the processor in the FPGA as well as an analog to digital converter (ADC) which measures the current at the processor's power source. So a computer is connected to the FPGA board, transmits the program to be executed and, at the end of the execution, receives the performance counters and the average current consumption through the serial port.

The analysis of the measurements showed that some of the performance counters have a high correlation with the overall processor power consumption. In addition, inspired by the bibliography, a simple neural network was implemented. Taking as input the normalized values of the performance counters, it predicts the average consumption current with great accuracy

Κεφαλαιο 1

Εισαγωγη

1.1 Στόχοι

1.2 Δομή της Διατριβής

1.1 Στόχοι

Στόχος της παρούσας εργασίας να γίνει κατανοητό αν υπάρχει συσχέτιση του ρυθμού αύξησης των μετρητών επίδοσης με την κατανάλωση ισχύος.Έτσι θα μπορεί να ανιχνεύεται τυχόν υπέρβαση στο ορίο κατανάλωσης ισχύος χωρίς να είναι απαραίτητοι ενσωματωμένοι αισθητήρες θερμότητας.

Οι επιμέρους στόχοι παρουσιάζονται παρακάτω:

- Να αναλυθεί η συμπεριφορά των μετρητών επίδοσης σε σχέση με την κατανάλωση ρεύματος και ισχύος.
- Η ανάπτυξη κατάλληλου κυκλώματος ώστε οι μετρήσεις να μπορούν να αντληθούν απο τον επεξεργαστή.
- Να κατηγοριοποιηθούν τα αποτελέσματα ώστε να είναι εμφανής τα προγράμματα που ορίζονται ως θερμικοί ιοί.
- Να επιβεβαιωθούν τα αποτελέσματα με προγράμματα που φτάνουν έναν επεξεργαστή στα όρια του χωρίς όμως να είναι κακόβουλα.

1.2 Δομή της Διατριβής

Η μεταπτυχιακή διπλωματική εργασία περιέχει συνολικά πέντε κεφάλαια. Το πρώτο κεφάλαιο περιέχει τους στόχους και την δομή της εργασίας. Στο δεύτερο κεφάλαιο αναφέρεται ό,τι είναι απαραίτητο να γνωρίζει ο αναγνώστης σχετικά με τον επεξεργαστή και την αρχιτεκτονική του. Τους θερμικούς ιούς και το εργαλείο που χρησιμοποιήθηκε και τροποποιήθηκε κατάλληλα για την παραγωγή τους. Επίσης γίνεται αναφορά στα εργαλεία που χρησιμοποιήθηκαν για τις μετρήσεις,και τις έννοιες για τα νευρωνικά δίκτυα. Το τρίτο κεφάλαιο, περιλαμβάνει όλες τις ενέργειες και τις υλοποιήσεις που έγιναν. Παρουσιάζονται οι αλλαγές στον επεξεργαστή που χρησιμοποιήθηκε,οι ενέργειες για την σύνδεση του με τις προσομοιώσεις αλλά και με το board για την άντληση των αποτελεσμάτων και τέλος η δημιουργία ενός απλού νευρωνικού δικτύου.

Το τέταρτο κεφάλαιο είναι αυτό που περιέχει όλα τα αποτελέσματα τόσο από τις προσομοιώσεις όσο και από τις μετρήσεις στο board. Την σύγκριση που έχουν με αποτελέσματα από μετροπρογράμματα που χρησιμοποιήθηκαν και τις προβλέψεις του νευρωνικού δικτύου .Το πέμπτο και τελευταίο κεφάλαιο περιέχει τα συμπεράσματα της συνολικής εργασίας και των υλοποιήσεων που έγιναν καθώς επίσης και τις μελλοντικές επεκτάσεις που μπορούν να γίνουν.

Κεφαλαιο 2

Υποβαθρο

2.1 Αρχιτεκτονική Συνόλου Εντολών RISC-V

2.2 Ibex: An embedded 32 bit RISC-V CPU Core

2.3 Stress Test και Θερμικοί Ιοί

2.4 Εργαλεία Προσομοίωσης Επεξεργαστή

2.5 FPGA-Field Programmable Gate Array

2.6 Νευρωνικά Δίκτυα και Βαθιά Μάθηση

Το συγκεκριμένο κεφάλαιο εισάγει τις γενικές πληροφορίες που είναι χρήσιμες για την κατανόηση του περιεχομένου και των εργαλείων που χρησιμοποιήθηκαν για την συγκεκριμένη εργασία. Στην πρώτη ενότητα παρουσιάζεται η αρχιτεκτονική συνόλου εντολών RISC-V. Η συγκεκριμένη αρχιτεκτονική επιλέχτηκε διότι είναι σύχρονη,ανοιχτή αρχιτεκτονική με αρκετές επεκτάσεις και με πολλές προοπτικές ερευνας και μελέτης. Παρουσιάζοντας τους βασικούς σχεδιαστικούς της στόχους και αρχές,την μορφή των εντολών και τους μηχανισμούς επεκτάσεων που έχει,θα δημιουργηθεί μία πλήρης είκόνα για την τόσο διαδεδομένη αρχιτεκτονική συνόλου εντολών. Στις επόμενες ενότητες παρουσιάζεται αρχικά ο επεξεργαστής που χρησιμοποιήθηκε καθώς επίσης και τα εργαλεία τόσο για την προσομοίωση του όσο και για τις μετρήσεις σε πραγματικό περιβάλον. Επιπλέον γίνεται εισαγωγή των εννοιών power virus και stress test. Για την δημιουργία τέτοιων ιών ψάχνοντας στην βιβλιογραφία βρέθηκε ο GeST,ένα ανοιχτού κώδικα εργαλείο παραγωγής θερμικών

ιών που τροποποιήθηκε ώστε να παράγει κώδικα σε αρχιτεκτονική συνόλου εντολών RISC-V. Τέλος γίνεται μια μικρή εισαγωγή για τα νευρωνικά δίκτυα που στα πλαίσια της εργασίας χρησιμοποιήθηκαν για την παραγωγή ενός μοντέλου πρόβλεψης κατανάλωσης ισχύος.

2.1 Αρχιτεκτονική Συνόλου Εντολών RISC-V

Το RISC-V είναι ένα νέο και ανοιχτό Instruction Set Architecture ISA, που αναπτύχθηκε αρχικά από το University of California Berkeley (UCB) και τώρα το διαχειρίζεται το ίδρυμα RISC-V [1].

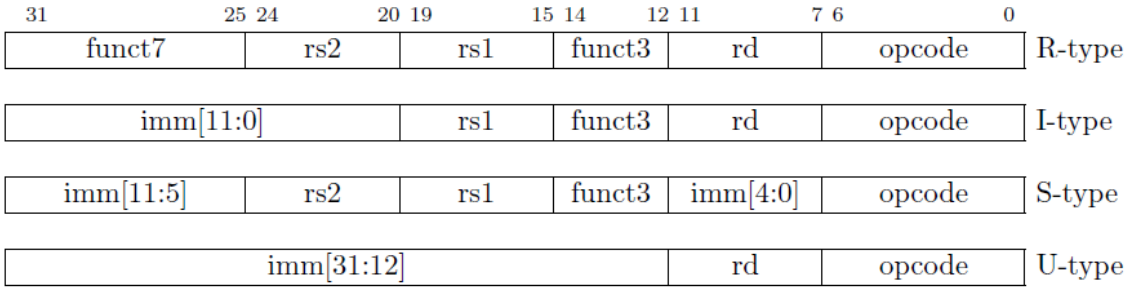
Βασισμένο σε Reduced Instruction Set Architecture RISC σχεδιάστηκε αρχικά για την έρευνα και την εκπαίδευση της αρχιτεκτονικής των υπολογιστών, αλλά στην συνέχεια στόχος έγινε ώστε να είναι μια δωρεάν και ανοιχτή αρχιτεκτονική για βιομηχανικές εφαρμογές. Το RISC-V εφιστά πλέον όλο και περισσότερο την προσοχή τόσο της βιομηχανίας όσο και του ακαδημαϊκού κόσμου, γιατί προσφέρει τη δυνατότητα να διευκολύνει τον σχεδιασμό των επεξεργαστών χωρίς να απαιτείται η προσφυγή σε ακριβά εμπορικά ISA.

2.1.1 Αρχιτεκτονική Συνόλου Εντολών

Στην αρχιτεκτονική συνόλου εντολών(Instructionn Set Architecture) RISC-V [2]. Το σύνολο εντολών χαρακτηρίζεται από το πλάτος των καταχωρητών και το αντίστοιχο μέγεθος του χώρου διευθύνσεων. Υπάρχουν δύο βασικές παραλλαγές εντολών ακεραίων, RV32I και RV64I,τα οποία παρέχουν χώρους διευθύνσεων 32-bit ή 64-bit αντίστοιχα. Επιπλέον η παραλλαγή υποσυνόλου RV32E του βασικού σετ εντολών RV32I, έχει το μισό αριθμό ακεραίων καταχωρητών και έχει προστεθεί για υποστήριξη μικροεπεξεργαστών.

Το σύνολο εντολών δέχεται και επεκτάσεις οι οποίες είναι προαιρετικές. Κάποιες από τα πιο συχνές επεκτάσεις είναι οι εξής:

- ”M” :Εντολές πολλαπλασιασμού και διαίρεσης ακεραίων
- ”A” :Εντολές για τον συγχρονισμό μεταξύ πολλαπλών RISC-V πυρήνων όπου κάνουν προσπελάσεις στην ίδια θέση μνήμης
- ”F” :Εντολές κινητής υποδιαστολής μονής ακρίβειας



Σχήμα 2.1: Μορφές εντολών RISC-V

- ”D” :Εντολές κινητής υποδιαστολής διπλής ακρίβειας
- ”C” :Συμπιεσμένες εντολές 16bit
- ”B” :Εντολές διαχείρισης των bit
- ”Zifencei”:Επέκταση συνέπειας μνήμης
- ”Zicsr”:Καταχωρητές ελέγχου και κατάστασης

2.1.2 Μορφή Εντολών

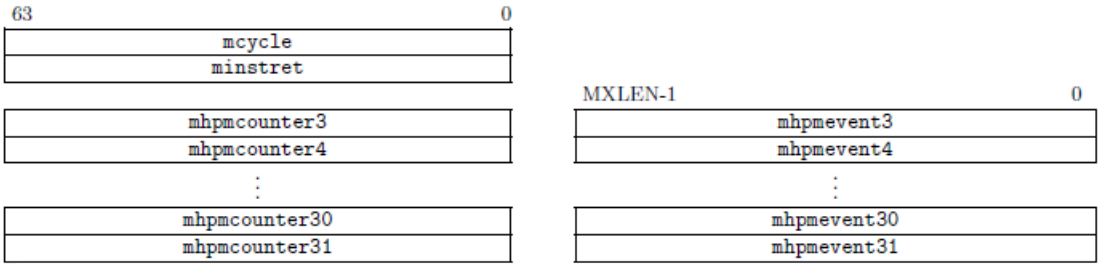
Στην βασική RV32I ISA, υπάρχουν τέσσερις διαφορετικές μορφές εντολών R-type ,I-type,S-type,U-type οπως φαίνονται στο Πίνακα 2.1.

Το RISC-V ISA διατηρεί τους καταχωρητές πηγής (rs1 και rs2) και τον καταχωρητή προορισμού (rd) στην ίδια θέση σε όλους τους τύπους εντολών για απλοποίηση της αποκωδικοποίησης της σταθεράς (immediate) είναι πάντα με επέκταση προσήμου. Το bit προσήμου για όλες τις σταθερές βρίσκεται πάντα στο bit 31 της εντολής.

2.1.3 Hardware Performance Counters

Στους σύγχρονους μικροεπεξεργαστές υπάρχει ένα σύνολο καταχωρητών ειδικού σκοπού που είναι ενσωματωμένοι για την αποθήκευση και καταμέτρηση δραστηριοτήτων που σχετίζονται με την λειτουργία τους. Αυτοί οι καταχωρητές ονομάζονται Hardware Performance Counters και μέσω αυτών μπορούν να βγουν συμπεράσματα για την λειτουργία του επεξεργαστή[3].

Στο RISC-V υπάρχουν 31 τέτοιοι καταχωρητές των 64 bit. Οι δύο από αυτούς mcycle,minstret είναι καθορισμένοι με συγκεκριμένες μετρήσεις,ο πρώτος καταχωρεί τους κύκλους ρολογιού που εκτελούνται σε έναν επεξεργαστή και ο δεύτερος τον αριθμό των εντολών που έχουν εκτελεστεί και ολοκληρωθεί. Για τους εναπομείναντες 29 καταχωρητές υπάρχει ένα άλλο σύνολο 29 καταχωρητών που ονομάζονται event-selector control-status-register(CSR) με ονομασία mhpmcounter3–mhpmcounter31 όπου καθορίζουν τι ακριβώς θα προσμετρά ο αντίστοιχος καταχωρητής(Σχήμα2.2).



Σχήμα 2.2: RISC-V Μετρητές Επίδοσης

2.2 Ibex: An embedded 32 bit RISC-V CPU Core

Ο Ibex CPU core [4],[5] είναι ανοιχτού κώδικα πυρήνας που συντηρείται και αναπτύσσεται από την lowRisc [6] είναι γραμμένος σε SystemVerilog και μπορεί να προσομοιωθεί χρησιμοποιώντας τα εργαλεία ανοιχτού κώδικα Fusesoc [7][8] και το Verilator[9][10],καθώς και με αρκετούς εμπορικούς προσομοιωτές από Cadence, Synopsys, Aldec.

Ένα πολύ σημαντικό χαρακτηριστικό της σχεδίασης του συγκεκριμένου πυρήνα είναι ότι σε μεγάλο βαθμό είναι παραμετροποιήσιμος. Ως προς την υποστηριζόμενη αρχιτεκτονική ο Ibex μπορεί να παραμετροποιηθεί ώστε να υποστηρίζει είτε RV32I είτε RV32E. Επιπλέον υποστηρίζει RISC-V ISA επεκτάσεις και συγκεκριμένα RV32IMC σύνολο εντολών πάντα σύμφωνα με το [1], από τις οποίες προεκτάσεις του συνόλου εντολών κάποιες είναι προαιρετικές και κάποιες πάντα ενεργοποιημένες: C (προαιρετικό) για τις compressed εντολές,M (προαιρετικό) πολλαπλασιασμό και διαίρεση ακεραίων,B (προαιρετικό) προέκταση για διαχείριση των bit,Zicsr (πά-

ντα ενεργοποιημένο)εντολές για Control-Status Registers,Zifencei (πάντα ενεργοποιημένο).

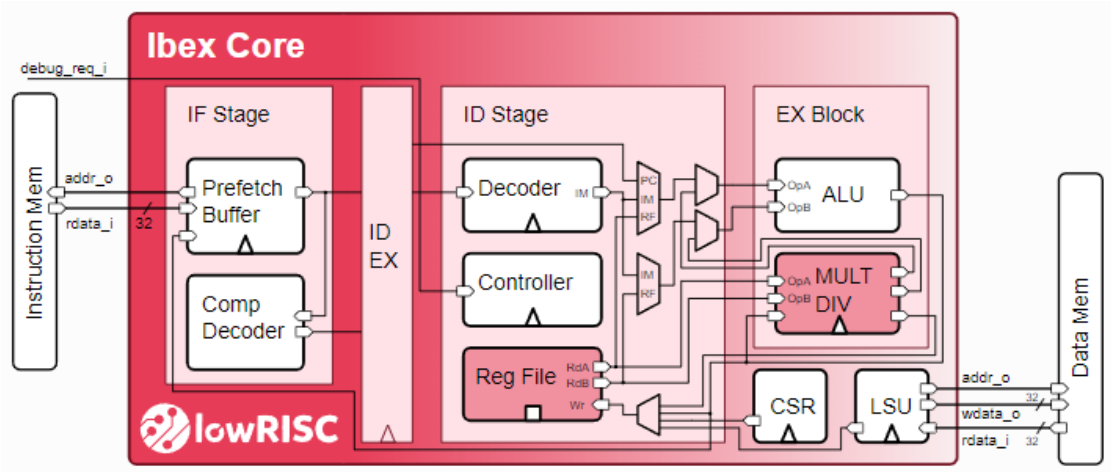
Κάποιες απο τις παραμέτρους είναι σε πειραματικό επίπεδο ενώ οι υπόλοιποι πλήρως υλοποιήσιμοι και επαληθευμένοι. Ένα παράδειγμα παραμέτρων είναι στην περίπτωση που υπάρχει ενεργοποιημένη η επέκταση M αυτή δηλαδή του πολλαπλασιασμού. Υπάρχουν δύο παράμετροι που καθορίζουν την υλοποίηση της. Η RV32MSingleCycle που είναι και η πρώτη επιλογή για ένα FPGA χρησιμοποιεί τρείς παράλληλες μονάδες πολλαπλασιαστή των 17x17 bit ενός κύκλου,συνεπώς το αποτέλεσμα ολοκληρώνεται σε ένα κύκλο. Η δεύτερη επιλογή είναι RV32MFast όπου είναι καλή επιλογή για σύνθεση ASIC,στην οποία το αποτέλεσμα ολοκληρώνεται σε τρείς κύκλους. Χρησιμοποιεί Multiplication and Accumulation (MAC), που είναι ένα εσωτερικό μπλοκ πολλαπλασιασμού και διαίρεσης που μπορεί να εκτελέσει 17×17 bit πολλαπλασιασμό με 34 bit accumulator.

Κάποιες ακόμα από τις παραμέτρους είναι η επιλογή Register File ανάμεσα σε RegFileFF το οποίο είναι default,RegFileLatch και RegFileFPGA. Η παράμετρος BranchTargetALU η οποία όταν είναι ενεργοποιημένη προβλέπει τη χρήση ξεχωριστής ALU για τον υπολογισμό του στόχου διακλάδωσης(branch target), επιτρέποντας τον επεξεργαστή να εξαλείψει τους κύκλους stall από τις διακλαδώσεις και έτσι να αυξήσει την απόδοση.

Μια ακόμα πολύ σημαντική παράμετρος είναι αυτή του Writeback ώστε να έχει και ένα τρίτο στάδιο διοχέτευσης το οποίο έχει σημαντικές επιπτώσεις στην απόδοση και στη συμπεριφορά των εντολών. Αυτή η διαμόρφωση τριών σταδίων του Ibex έχει υλοποιηθεί στην εργασία.

2.2.1 Η Αρχιτεκτονική του Ibex

Η κεντρική υλοποίηση του πυρήνα είναι διοχέτευση δύο σταδίων Instruction Fetch (IF) και Instruction Decode and Execute (ID/EX) όπως φαίνεται και στο παρακάτω σχήμα 2.3.



Σχήμα 2.3: Ibex,διοχέτευση δύο σταδίων

Το Instruction Fetch (IF) διαχειρίζεται τον prefetch buffer και παρέχει εντολές στο επόμενο στάδιο ID/EX. Ο prefetch buffer οποίος μπορεί να προσκομίσει μια εντολή ανά κύκλο από την μνήμη και έχει την δυνατότητα να προσκομίσει εντολές νωρίτερα μέχρι να γεμίσει.Σε περίπτωση διακλαδώσεων ,αλμάτων ή εξαιρέσεων πρέπει να αδειάσει και να ανακτηθούν εκ νέου εντολές.

Σε περίπτωση που διαχειρίζεται συμπιεσμένες εντολές ο Compressed Decoder τις μετατρέπει σε μη συμπιεσμένες εντολές. Ορίζοντας την παράμετρο ”BranchPrediction” σε 1,ο Ibex μπορεί να διαμορφωθεί να χρησιμοποιήσει πρόβλεψη στατικής διακλάδωσης(Static Branch Prediction). Αυτό αυξάνει την αποτελεσματικότητα προβλέποντας ότι οποιαδήποτε διακλάδωση με αρνητική μετατόπιση θα ακολουθήσει,ενώ με θετική μετατόπιση όχι. Αν η πρόβλεψη είναι λανθασμένη τότε έχουμε το λιγότερο έναν κύκλο ποινή. Η πρόβλεψη διακλάδωσης στον Ibex είναι σε πειραματικό στάδιο.

Στο επόμενο στάδιο το Instruction Decode and Execute αποκωδικοποιείται και εκτελείται η εντολή που ανακτήθηκε απο το Instruction Fetch. Οι εντολές με το που αποκωδικοποιούνται εκτελούνται αμέσως.Σε περίπτωση εντολών πολλαπλών κύκλων γίνετε καθυστέρηση (stall) του σταδίου μέχρι αυτές να ολοκληρωθούν. Αυτό δείχνει ότι το μέγιστο IPC (Instructions per Cycle) το οποίο μπορεί να επιτευχθεί είναι IPC=1 όταν δεν χρησιμοποιούνται εντολές που υλοποιούν stall. Το συγκεκριμένο στάδιο όπως φαίνεται και από το προηγούμενο σχήμα χωρίζεται σε πολλά τμήματα.

Στο πρώτο τμήμα αυτό του Instruction Decode υπάρχει ο Decoder,ο Controller και το Register File.

- ”Decoder” : Λαμβάνει τις εντολές και στέλνει τα κατάλληλα σήματα στα υπόλοιπα μπλοκ ώστε να εκτελεστεί κατάλληλα η εντολή.
- ”Controller” : Στον Controller βρίσκεται η μηχανή κατάστασης όπου ελέγχει την συνολική λειτουργία του επεξεργαστή. Κάποιες από τις λειτουργίες που εκτελεί είναι η εκκίνηση του επεξεργαστή ή η επαναφορά του ,η διαχείριση των εξαιρέσεων,η ρύθμιση του program counter σε περιπτώσεις διακλαδώσεων ή αλμάτων.
- ”Register File” :Ο Ibex διαθέτει τριάντα δύο καταχωρητές μεγέθους 32 bit,στην περίπτωση όμως που είναι ενεργοποιημένη η επέκταση RV32E τότε υπάρχουν δεκαέξι καταχωρητές μεγέθους 32 bit. Υπάρχουν δύο θύρες ανάγνωσης και μία θύρα εγγραφής, και σε περίπτωση αιτήματος ανάγνωσης τα δεδομένα είναι διαθέσιμα στον ίδιο κύκλο με το αίτημα.

Στο δεύτερο μπλοκ αυτού του σταδίου αυτό το Execute block υπάρχει η ALU και το μπλοκ που εκτελεί τον πολλαπλασιασμό και την διαίρεση(multiplier/divider block)

- ”Arithmetic Logic Unit (ALU)”:Υλοποιούνται οι βασικές λειτουργίες που απαιτούνται για τις ακέραιες πράξεις,πράξεις ολίσθησης ,δυαδικές πράξεις και πράξεις σύγκρισης.
- ”Multiplier/Divider Block (MULT/DIV)”:Είναι ένα μπλοκ πολλαπλασιασμού και διαίρεσης. Υπάρχουν δύο μοντέλα αυτό του multdiv_fast και του multdiv_slow.

Στο παραπάνω σχήμα της διοχέτευσης των δύο σταδίων του Ibex υπάρχουν και δύο ακόμα μπλοκ αυτό του Load Store Unit(LSU) και αυτό του Control Status Register(CSR). Το LSU είναι υπεύθυνο για την πρόσβαση στην μνήμη δεδομένων. Λέξεις 32,16,8 bit μπορούν να ανακτηθούν και να αποθηκευτούν. Οποιοδήποτε αίτημα για αποθήκευση δεδομένων ή ανάκτηση(store,load data)σταματάει το Instruction Decode and Execute στάδιο για ένα κύκλο.

Το CSR διαχειρίζεται όλες τις αναγνώσεις και τις εγγραφές που γίνονται στους Control Status Register και αποθηκεύει-τροποποιεί τους Performance Counters. Τα

δεδομένα αυτά μπορούν να διαβαστούν στον ίδιο κύκλο σε περίπτωση αιτήματος ανάγνωσης. Ο Ibex διαθέτει τους Performance Counters σύμφωνα με το RISC-V. Οι mcycle,minstret οι οποίοι είναι και οι βασικοί για την καταμέτρηση των κύκλων και των εντολών που εκτελέστηκαν καθώς επίσης και ακόμα δέκα οι οποίοι έχουν συγκεκριμένες λειτουργίες του επεξεργαστή που καταμετρούν. Στον παρακάτω πίνακα φαίνονται οι Performance Counters με τα αντίστοιχα γεγονότα που εκτελούν(Πίνακας. 2.1).

Πίνακας 2.1: Μετρητές Επίδοσης

Event ID	Performance Counter	Event Counter	Event Name	Event Description
0	mcycle	-	NumCycles	Number of cycles
1	-	-	-	-
2	minstret	-	NumInstrRet	Number of instructions retired
3	mhpmcounter [3]	mhpmevent [3]	NumCyclesLSU	Number of cycles waiting for data memory
4	mhpmcounter [4]	mhpmevent [4]	NumCyclesIF	Cycles waiting for instruction fetches
5	mhpmcounter [5]	mhpmevent [5]	NumLoads	Number of data memory loads
6	mhpmcounter [6]	mhpmevent [6]	NumStores	Number of data memory stores
7	mhpmcounter [7]	mhpmevent [7]	NumJumps	Number of unconditional jumps (j, jal, jr, jalr)
8	mhpmcounter [8]	mhpmevent [8]	NumBranches	Number of branches (conditional)
9	mhpmcounter [9]	mhpmevent [9]	NumBranchesTaken	Number of taken branches (conditional)
10	mhpmcounter [10]	mhpmevent [10]	NumInstrRetC	Number of compressed instructions retired
11	mhpmcounter [11]	mhpmevent [11]	NumCyclesMulWait	Cycles waiting for multiply to complete
12	mhpmcounter [12]	mhpmevent [12]	NumCyclesDivWait	Cycles waiting for divide to complete

2.3 Stress Test και Θερμικοί Ιοί

Τα Stress Test στοχεύουν σε συγκεκριμένες μετρικές ενός συστήματος και στόχος τους να ανακαλύψουν ελαττώματα που πιθανόν υπάρχουν. Σχετικά με τμήματα ενός επεξεργαστή μπορούν να χωριστούν σε τρεις κατηγορίες:α) stress test που μεγιστοποιούν συγκεκριμένες μετρήσεις της μικροαρχιτεκτονικής , ρυθμός μετάδοσης απο την μνήμη, IPC(instruction per cycle),αστοχίες κρυφής μνήμης β) stress tests που μεγιστοποιούν κατανάλωση ενέργειας και θερμοκρασία, που συνήθως αναφέρονται ως power virus και γ) stress test που μεγιστοποιούν το θόρυβο της τάσης(voltage noise), γνωστοί και ως voltage-noise viruses ή dI/dt-viruses [11] [12].

Οι Power-viruses μεγιστοποιούν την κατανάλωση ισχύος (power consumption) και την θερμότητα. Είναι χρήσιμοι ώστε να χαρακτηρίζουν την ισχύ και τα θερμικά περιθώρια ενός συστήματος καθώς και τα IR drop. Αυτό που κάνουν συνήθως εί-

ναι να αυξάνεται το IPC με αποτέλεσμα να μεγιστοποιείται η δραστηριότητα του επεξεργαστή.

2.3.1 GeST: Ένα Αυτόνομο Πλαίσιο Λογισμικού για την Δημιουργία CPU Stress-Tests

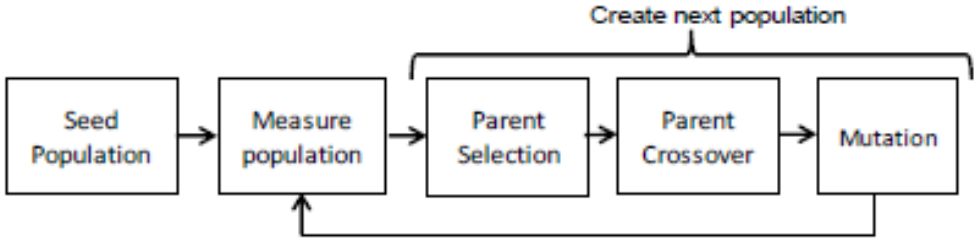
Για την δημιουργία των Stress-Tests υπάρχει το συγκεκριμένο framework όπου είναι βασιζόμενο στους γενετικούς αλγορίθμους για την παραγωγή των τέστ. Ο χρήστης μπορεί να καθορίσει λειτουργίες μέτρησης και φυσικής κατάστασης καθώς και τις εντολές της αρχιτεκτονικής του επεξεργαστή. Σαν είσοδο παίρνει αρχεία xml που καθορίζουν τις παραμέτρους και τις εντολές. Οι καθορισμένες αρχιτεκτονικές που υποστηρίζει είναι οι ARM και x86.[13]

Το Gest βασίζεται στους γενετικούς αλγορίθμους.Οι γενετικοί αλγόριθμοι είναι αλγόριθμοι οι οποίοι μεταβιβάζουν χαρακτηριστικά απο γενιά σε γενιά οδηγώντας στην εξέλιξη της μετρικής η οποία έχει οριστεί και είναι βασισμένοι στις αρχές της βιολογικής εξέλιξης.Βασικά σημεία τους είναι ο αρχικός πληθυσμός,η διαδικασία επιλογής,οι γενετικοί τελεστές και η διαδικασία τερματισμού. Είναι κατάλληλοι και αποτελεσματικοί για την παραγωγή Stress-Tests. Ο γενετικός αλγόριθμος βελτιστοποιεί την καθορισμένη μετρική και λειτουργεί όπως φαίνεται στο σχήμα 2.4. Το πρώτο βήμα Seed Population είναι να δημιουργηθεί ο αρχικός πληθυσμός. Ο πληθυσμός (population) χωρίζεται σε μοναδικά προγράμματα (individuals) όπου είναι ένα σύνολο από εντολές assembly της αντίστοιχης αρχιτεκτονικής. Ο πληθυσμός μπορεί να είναι είτε ένας νέος τυχαίος αρχικός πληθυσμός είτε πληθυσμός από προηγούμενη επανάληψη . Στην συνέχεια το στάδιο Measure Individuals είναι η μέτρηση για καθε individual, το παραγόμενο πρόγραμμα assembly του ώστε να εξαχθούν τα αποτελέσματα απο τις μετρήσεις που έχουνε οριστεί. Το τελευταίο βήμα αποτελείται απο τρία διαδοχικά στάδια τα Parent Selection ,Parent Crossover, Mutation. Τα συγκεκριμένα είναι υπεύθυνα για την δημιουργία της επόμενης γενιάς πληθυσμού όταν έχει τερματιστεί η μέτρηση όλων των individuals ξεχωριστά. Τα individuals με τα καλύτερα αποτελέσματα αλλάζουν κατάσταση και ονομάζονται parents(Selection) ,υλοποιείται το Crossover και το Mutation στο οποίο ανάλογα το ποσοστό που έχει καθοριστεί, μια εντολή έχει τόση πιθανότητα να αλλάξει. Οι προκαθορισμένες τιμές που παρουσιάζονται παρακάτω είναι και αυτές που σύμφωνα με την δημοσίευση είναι και τα επικρατέστερα για καλύτερο αποτέλεσμα(Πίνακας

2.2).

Parameter	***Default_Values***
population_ size	50
Individual Size (number of loop instructions)	15-50
crossover_operator	one point crossover
elitism (Best individual promoted to next generation)	TRUE
parent_selection_method	Tournament Selection
tournament_size	5

Πίνακας 2.2: Παράμετροι GeST

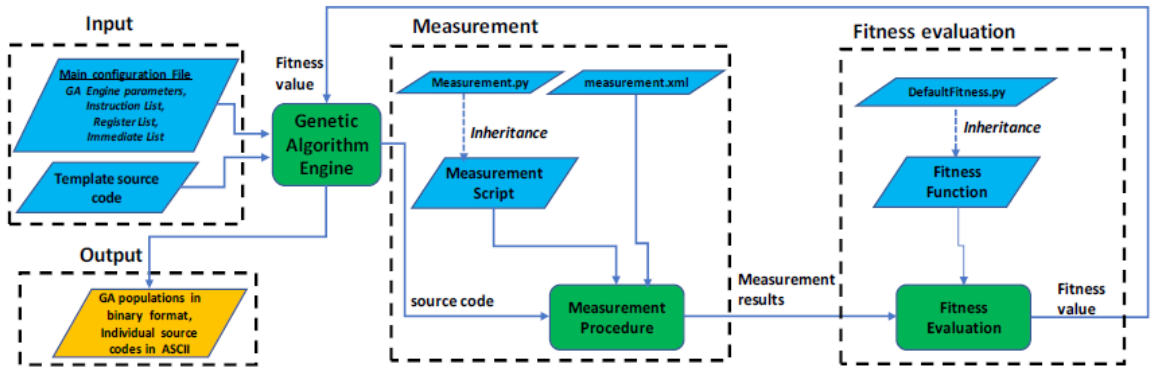


Σχήμα 2.4: Γενετικός Αλγόριθμος

Όσο αναφορά τα αρχεία εισόδου του GeST δέχεται το main_configuration_file το οποίο είναι ένα αρχείο xml το οποίο περιεχέι τις παραμέτρους ώστε να αρχικοποιηθεί και επίσης τις εντολές και τους τελεσταίους και τους καταχωρητές της αρχιτεκτονικής που θα παραχθεί ο τελικός κώδικας.

Σαν έξοδο σε κάθε επανάληψη δίνει ένα αρχείο assembly.Το αρχείο αυτό έχει αρχικοποιημένες τις τιμές των καταχωρητών και τις εντολές που έχουν παραχθεί και θα εκτελεστούν. Υπάρχει ένα σημείο μέσα στον κώδικα το #loop_code όπου τοποθετείται κάθε φορά,για τα διαφορετικά individuals,οι παραγόμενες εντολές.

Το συνολικό πλαίσιο λογισμικού φαίνεται στο σχήμα 2.5.



Σχήμα 2.5: Συνολική Εικόνα του Gest

2.4 Εργαλεία Προσομοίωσης Επεξεργαστή

2.4.1 Fusesoc-Εργαλείο Διαχείρισης Πακέτων

Το FuseSoC είναι ένας διαχειριστής πακέτων και ένα εργαλείο σύνθεσης κώδικα HDL (Hardware Description Language) [7][8]. Είναι γραμμένο σε python και τρέχει σε όλα τα λειτουργικα συστήματα.

Το εργαλείο δέχεται σαν είσοδο μεταβλητές που καθορίζουν την διαδικασία κατασκευής όπως τον στόχο κατασκευής αν είναι προσομοίωση, σύνθεση FPGA ή σύνθεση ASIC ή το εργαλείο για την σχεδίαση (EDA tool) που θα χρησιμοποιηθεί όπως Verilator,Icarus Verilog,Xilinx Vivado,Synopsys VCS,Altera Quartus,ModelSim και άλλα. Σαν έξοδο διαθέτει όλα εκείνα τα αρχεία και δεδομένα που χρειάζεται ένα τέτοιο εργαλείο EDA για την σωστή λειτουργία του.

2.4.2 Verilator-Προσομοιωτής Γλώσσας Περιγραφής Υλικού

Ο Verilator [9][10] είναι ένα εργαλείο ανοιχτού κώδικα όπου ,μετατρέπει κώδικα γραμμένο από Verilog-System Verilog σε cycle-accurate μοντέλο γλώσσας C++ ή SystemC. Ταυτόχρονα το βελτιστοποιεί και σε μία πιο γρήγορη μορφή του,με αποτέλεσμα να προσφέρει καλύτερη επίδοση σε σχέση με άλλους απλούς προσομοιωτές Verilog.

2.4.3 Xilinx Vivado

Το Vivado Design Suite είναι μια σουίτα λογισμικού της Xilinx για σύνθεση και ανάλυση σχεδίων με γλώσσα περιγραφής υλικού (HDL). Περιλαμβάνει ενσωματωμένο προσομοιωτή και μας δίνει την δυνατότητα εκτίμηση της ισχύος. Καταγράφει το switching activity και σύμφωνα με αυτό δημιουργεί αρχείο με την λεπτομερή αναφορά της ισχύος του συστήματος.

2.5 FPGA-Field Programmable Gate Array

Το FPGA είναι ένα ολοκληρωμένο κύκλωμα σχεδιασμένο ώστε μετά την κατασκευή του να διαμορφώνεται ανάλογα με την χρήση του. Η διαμόρφωση του γίνεται σε κάποια γλώσσα περιγραφής υλικού(Verilog,VHDL). Διαθέτουν configurable logic blocks-CLB ,κανάλια επικοινωνίας και block εισόδου/εξόδου. Το βασικό χαρακτηριστικό τους είναι τα CLB. Όταν αυτά συνδέονται μεταξύ τους μέσω των καναλιών επικοινωνίας μπορούν να εκτελέσουν διάφορες λειτουργίες. Τα CLB και αυτά διαθέτουν flip-flops ,Look up Tables και πολυπλέκτες. Τα block εισόδου/εξόδου περιβάλουν τα CLB και είναι για την διασύνδεση με εξωτερικές συσκευές.'Ενα από τα πλεονεκτήματα των FPGA είναι ότι σε μεγάλο πλήθος εφαρμογών προσφέρουν ταχύτητα για την επίλυση τους. Τα FPGA δεν διατηρούν τον προγραμματισμό τους αν διακοπεί η τροφοδοσία τους.

2.6 Νευρωνικά Δίκτυα και Βαθιά Μάθηση

Η μηχανική μάθηση(Machine Learning) είναι υποπεδίο της επιστήμης των υπολογιστών και διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. Χωρίζεται στις εξής κατηγορίες: την μάθηση με επίβλεψη (supervised learning), την μάθηση χωρίς επίβλεψη(unsupervised learning) και την ενισχυτική μάθηση(reinforcement learning).

Στην μάθηση με επίβλεψη στο πρόγραμμα δίνονται τόσο οι είσοδοι όσο και οι έξοδοι. Κλασσικά μοντέλα Μηχανικής Μάθησης είναι Linear Regression, Support-Vector Machines(SVM),Decision Trees και Neural Networks.

Σε αντίθεση, η μάθηση χωρίς επίβλεψη δεν παρέχει στο πρόγραμμα τις εξόδους.

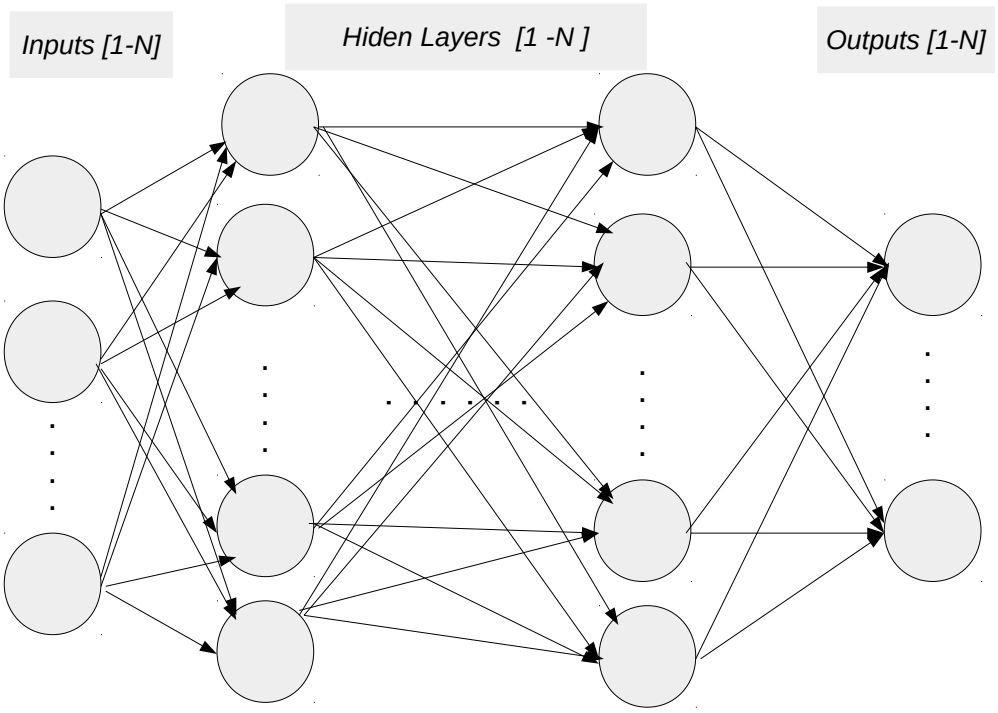
Προβλήματα όπως Clustering,Association,Dimensionality Reduction λύνονται με αυτή την τεχνική.

Η ενισχυτική μάθηση διαφέρει ριζικά από τις υπόλοιπες καθώς αντί τα μοντέλα της να εκπαιδεύονται από ένα συγκεκριμένο πλήθος δεδομένων το σύστημα μάθησης προσπαθεί να μάθει μέσα από την άμεση αλληλεπίδραση με το περιβάλλον.

2.6.1 Βαθιά Νευρωνικά Δίκτυα

Όπως προκύπτει και από την ίδια την έννοια ένα Βαθύ Νευρωνικό Δίκτυο (DNN) διαφέρει από ένα απλό Νευρωνικό λόγω της ύπαρξης του βάθους (Depth). Σαν βάθος ορίζουμε στα Νευρωνικά Δίκτυα τον αριθμό των κρυφών επιπέδων, και, επομένως, σαν DNN ένα Νευρωνικό με μεγαλύτερο από ένα αριθμό κρυφών επιπέδων(Σχήμα 2.6). Όσο πιο βαθύ είναι ένα επίπεδο τόσο πιο σύνθετα είναι τα χαρακτηριστικά πάνω στα οποία αυτό εκπαιδεύεται. Είναι ικανά να διαχειριστούν πολύ μεγάλα training sets και μάλιστα με πολύ καλές προβλέψεις.

Προσθέτοντας περισσότερα επίπεδα και περισσότερα στοιχεία σε κάθε επίπεδο το βαθύ νευρωνικό δίκτυο μπορεί να αναπαραστήσει όλο και πιο περίπλοκες συναρτήσεις. Για να προκύψουν τα επιθυμητά αποτέλεσμα θα πρέπει να οριστούν κατάλληλα πολλοί παράμετροι του δικτύου όπως ο αριθμός των επιπέδων,ο αριθμός των νευρώνων σε κάθε επίπεδο,κατάλληλες συναρτήσεις ενεργοποίησης και βελτιστοποιητές.



Σχήμα 2.6: Επίπεδα Νευρωνικού Δικτύου

2.6.2 Συνάρτηση Ενεργοποίησης

Οι συναρτήσεις ενεργοποίησης(Activation Function) είναι συναρτήσεις που χρησιμοποιούνται στα νευρωνικά δίκτυα για τον υπολογισμό του σταθμισμένου αθροίσματος εισόδου και bias,το οποίο χρησιμοποιείται για να αποφασιστεί εαν ο νευρώνας θα ενεργοποιηθεί ή οχι[14]. Χρησιμοποιούνται για τον έλεγχο των εξόδων του νευρωνικού και μπορεί να χωριστούν σε γραμμικές και μη. Κάποιες απο τις συναρτήσεις είναι η Rectified Linear Activation (ReLU) με τύπο σ(x)=max(0,x) όπου χρησιμοποιείται πιο συχνά,η σιγμοειδής(sigmoid-logistic) με τύπο σ(x)= 1/(1+exp^{-x}) και η Υπερβολική εφαπτομένη(tanh) σ(x) = e^{x} - e^{-x} / e^{x} + e^{-x}.

2.6.3 Υπερπαράμετροι

Υπερ-παράμετροι ονομάζονται οι παράμετροι όπου οι τιμές τους καθορίζουν την δομή του νευρωνικού δικτύου και την διαδικασία της μάθησης. Ο αριθμός τον κρυμμένων επιπέδων, η συνάρτηση ενεργοποίησης,ο ρυθμός μάθησης και ο αριθμός των επόχών είναι κάποιες απο τις υπερ-παραμέτρους όπου οι σωστές τίμες που θα τους

δοθούν καθορίζουν την αποτελεσματικότητα του νευρωνικού δικτύου.

2.6.4 Βελτιστοποιητές

Οι βελτιστοποιητές είναι αλγόριθμοι όπου προσπαθούν να αλλάξουν χαρακτηριστικά του δικτύου όπως ο ρυθμός μάθησης(learning rate) ώστε να ελαχιστοποιηθεί η συνάρτηση κόστους. Ο ρυθμός μάθησης είναι μία υπερ-παράμετρος που ελέγχει το μέγεθος του βήματος σε κάθε επανάληψη. Οι Adam,Gradient Descent,Stochastic Gradient Descent,Nesterov Momentum είναι κάποιοι απο τους πιο συχνούς βελτιστοποιητές.

2.6.5 Υπερεκπαίδευση και Υποεκπαίδευση

Στην μηχανική μάθηση ένα μεγάλο πρόβλημα που προκύπτει είναι πως θα αποφευχθεί τόσο η υπερεκπαιδευση(over-fitting) όσο και η υποεκπαίδευση(under-fitting). Τα μοντέλα Μηχανικής Μάθησης εκπαιδεύονται με τη χρήση ενός πλήθους διαθέσιμων δεδομένων και ο σκοπός είναι η εξαγωγή ιδιοτήτων από τα δεδομένα αυτά.

Η εκπαίδευση όμως πρέπει να γίνει σε τέτοιο βαθμό, που το μοντέλο να μπορεί μετά την εκπαίδευση να αναγνωρίζει τις ιδιότητες αυτές και σε δεδομένα τα οποία θα δει για πρώτη φορά. Εάν το μοντέλο εκπαιδευτεί περισσότερο από ότι πρέπει, θα αρχίσει να θεωρεί ως χρήσιμη πληροφορία τον θόρυβο των δεδομένων. Αυτό συμβαίνει, είτε επειδή είναι μεγάλος ο αριθμός των παραμέτρων της εκπαίδευσης είτε επειδή το μοντέλο έχει εκπαιδευτεί με τα ίδια δεδομένα αρκετές φόρες. Αντίθετα,εαν ένα μοντέλο δεν εκπαιδευτεί αρκετά, δεν θα μπορέσει να διακρίνει τις απαραίτητες ιδιότητες στα δεδομένα της εκπαίδευσης. Αυτό μπορεί να συμβεί όταν δεν υπάρχουν αρκετά δεδομένα διαθέσιμα ή όταν τα δεδομένα είναι χαμηλής ποιότητας.

Κεφαλαιο 3

Υλοποιηση

3.1 Προσθήκη Επίπλεον Performance Counters

3.2 Διαμόρφωση του GeST

3.3 Σύνδεση του GeST και Προσομοίωση

3.4 Σύνδεση του GeST και Μετρήσεις Πραγματικών Τιμών με το Arty A7

3.5 Περιγραφή Νευρωνικού Δικτύου

Στο παρόν κεφάλαιο θα παρουσιαστούν όλες οι υλοποιήσεις,και οι ενέργειες που έγιναν για την υλοποίηση της εργασίας.Ο Ibex που χρησιμοποιείται στα πλαίσια της υλοποίησης διαθέτει συνολικά δώδεκα μετρητές.Απο τους συγκεκριμένους μετρητές μπορούν να αντληθούν αρκετές πληροφορίες για την λειτουργία του ωστόσο, για την καλύτερη κατανόηση των αποτελεσμάτων και για να υπάρχει μεγαλύτερη συσχέτιση των μετρητών με τις μετρήσεις Ισχύος και Ρέυματος χρειάστηκε η προσθήκη δύο επιπλέον Performance Counters. Ακόμα,αφού ο Ibex είναι ενα RISC-V Core η διαμόρφωση του GeST ώστε ο κώδικας που παράγει να είναι εντολές RISC-V ήταν αναγκαία αφού εστιάζει μόνο στις αρχιτεκτονικές ARM και x86.
Αρχικά οι μετρήσεις έγιναν σε επίπεδο προσομοίωσης όπου ολοκληρώνεται σε δύο μέρη. Γίνεται μία προσομοίωση σε επίπεδο συμπεριφοράς για να εξαχθούν οι τιμές των performance counters και μία λειτουργική προσομοίωση για να εξαχθεί η τιμή της κατανάλωσης ισχύος.Εκτός απο μετρήσεις σε επίπεδο προσομοίωσης έγιναν και πραγματικές μετρήσεις σε ολοκληρωμένο κύκλωμα. Η χρήση ένος FPGA έλυσε ένα

βασικό πρόβλημα αφού βοήθησε στο σύνολο των δεδομένων καθώς οι μετρήσεις με προσομοίωση σε υπολογιστή ήταν πολύ αργές. Ετσι έγιναν κάποιες τροποποιήσεις στο board ώστε να γίνεται σειριακή επικοινωνία με τον υπολογιστή για τα δεδομένα σύμφωνα με το πρωτόκόλο (UART). Τα δεδομένα που συλλέγονται απο τις μετρήσεις στο board χωρίζονται στους μετρητές επίδοσης και σε μία τιμή ρεύματος που αντιστοιχεί σε αυτά.Στα πλαίσια της εργασίας υλοποιήθηκε ένα απλό νευρωνικό δίκτυο όπου το επίπεδο εισόδου δέχεται διάνυσμα που αποτελείται από δεκατέσσερα χαρακτηριστηκά, τους μετρητές επίδοσης, και την τιμή ρεύματος που αντιστοιχεί σε αυτά.

3.1 Προσθήκη Επίπλεον Performance Counters

Οι πληροφορίες που αντλούνται απο τους Performance counters παίζουν σημαντικό ρόλο στα συμπεράσματα για την λειτουργιά του επεξεργαστή. Ο Ibex όπως έχει ήδη αναφερθεί(Ενότητα 2.1) διαθέτει ένα συγκεκριμένο πλήθος από μετρητές:

- ”Cycles” : αριθμός των κύκλων
- ”Instructions Retired”: οι εντολές που ολοκληρώθηκαν
- ”LSU Busy”: αριθμός κύκλων περιμένοντας δεδομένα απο την μνήμη
- ”IFetch wait”: αριθμός κύκλων που αφιερώθηκαν περιμένοντας εντολή για προσκόμιση
- ”Loads”: ο αριθμός τον εντολών φόρτωσης
- ”Stores”: ο αριθμός των εντολών αποθήκευσης
- ”Jumps”: ο αριθμός των εντολών άλματων
- ”Branches”: αριθμός των διακλαδώσεων
- ”Taken Branches”: ο αριθμός τον διακλαδώσεων όπου ληφθηκαν
- ”Compressed Instructions”: ο αριθμός των συμπιεσμένων εντολών
- ”Multiply Wait”: αριθμός κύκλων που περιμένουν να ολοκληρωθούν οι πολλαπλασιασμοί

- ”Divide Wait”: αριθμός κύκλων που περιμένουν να ολοκληρωθούν οι διαίρεσεις

Για να δημιουργηθεί μια πιο ολοκληρωμένη εικόνα προστέθηκαν δυο επιπλέον Performance counters:

- ”Instruction_fetch”: αριθμός εντολών που έχουν προσκομιστεί συνολικά
- ”Total_stalls”: οι συνολικές καθυστερήσεις

Ο Ibex διαχειρίζεται τους Performance counters στο αρχείο ibex_cs_registers. Για τον ”instruction_fetch” στην είσοδο του ibex_cs_registers δίνεται σήμα απο το instruction fetch module όπου μετράει κάθε καινούργια εντολή που γίνεται fetch. Για τα stall απο το id stage υπάρχει σήμα εξόδου όπου δείχνει τα συνολικά stall. Τα σήματα stall που είναι υπεύθυνα για τις καθυστερήσεις είναι τα stall_ld_hz , stall_mem , stall_multdiv , stall_jump , stall_branch , stall_alu . Ο ibex_cs_registers για κάθε μετρητή καλεί τον ibex_counter που αυξάνει το κάθε μετρητή.

3.2 Διαμόρφωση του GeST

Το GeST είναι framework βασισμένο σε γενετικούς αλγορίθμους για την παραγωγή stress tests που μεγιστοποιούν διάφορες μετρικές του επεξεργαστή όπως την κατανάλωση ισχύος,τις εντολές που εκτελούνται ανα κύκλο(IPC) και τον ρυθμό μεταβολής του ρεύματος dI/dt. Το συγκεκριμένο εστίαζε στις αρχιτεκτονικές ARM και x86 επομένως τροποποιήθηκε ώστε να παράγει stress tests και για την αρχιτεκτονική RISC-V. Τα δύο βασικά αρχεία εισόδων είναι το configuration_RISC αρχείο xml και το main_original που είναι πρότυπο αρχείου assembly.

Το xml αρχείο είναι και αυτό που προσδιορίζει όλες τις βασικές παραμέτρους για τον γενετικό αλγόριθμο καθώς και τις εντολές όπου θα παραχθεί ο τελικός κώδικας. Κάποιες απο τις παραμέτρους είναι το πλήθος των εντολών που θα έχει ο τελικός κώδικας(individual size)<*loopSize*>,το πλήθος απο ξεχωριστά individual <*population_size*>,ο συντελεστής μετάλλαξης<*mutation_rate*> που δείχνει την πιθανότητα που έχει κάθε εντολή να αλλάξει,<*populations_to_run*> όπου δείχνει για πόσες γενιές θα τρέξει ο αλγόριθμος και άλλες που σχετίζονται με το τρόπο διασταύρωσης την αποθήκευση των αρχείων και την σύνδεση.
Το βασικό όμως κομμάτι του συγκεκριμένου αρχείου είναι ο καθορισμός των εντολών της αρχιτεκτονικής συνόλου εντολών που ακολουθείται. Οι εντολές,οι κατα-

χωρητές και οι τιμές immediate καθορίζονται στο συγκεκριμένο αρχείο. Στο GeST προϋπάρχουν τέτοια αρχεία για εντολές ARM και x86. Σαν αποτέλεσμα να δημιουργηθεί ένα τέτοιο αρχείο και για τις εντολές RISC-V. Οι εντολές που προστέθηκαν είναι της κατηγορίας RV32I και RV32M για τον πολλαπλασιασμό και την διαίρεση. Απο το RV32I δεν προστέθηκαν οι εντολές διακλαδώσεων και αλμάτων.Οι εντολές που παράγονται και εκτελούνται μέσα σε έναν βρόγχο θα πρέπει να έχουν μία συνέχεια μεταξύ τους.Σε περίπτωση διακλάδωσης ή άλματος χάνεται η σειρά εκτέλεσης των εντολών δημιουργούνται καθυστερήσεις και μειώνεται η δραστηριότητα μεταγωγής.Οι μόνες εντολές διακλάδωσης είναι αυτές για τον εξωτερικό βρόγχο.Στην παρακάτω εικόνα φαίνεται ένα παράδειγμα καθορισμού των εντολών ADD και ADDI και τον καταχωρητών που χρησιμοποιούν(Σχήμα 3.1). Για τις εντολές υπάρχουν επτά παράμετροι. Το όνομα της εντολής,το πλήθος των τελεστών όπου είναι οι καταχωρητές που εμπλέκονται στην εντολή ή και αριθμοί όπως στην περίπτωση της *"ADDI"*. Στις μεταβλητές τους καθορίζεται απο που θα λάβουν τις "τιμές" τους. Οι τελεστές καθορίζονται στο αρχείο ξεχωριστά όπως φαίνεται στο δεξί μέρος της εικόνας. Η μεταβλητή που προσδιορίζει τον τύπο της εντολής και τέλος η μεταβλητή που δείχνει την μορφή που θα έχει η εντολή στην έξοδο.

```
<instruction
    name="ADD"
    num_of_operands="3"
    type="shortLat"
    operand1="general_integer_register"
    operand2="general_integer_register"
    operand3="general_integer_register"
    format="add op1,op2,op3"
</instruction>
```

```
<operand
    id="general_integer_register"
    values="x6 x7 x8 x9 x13 x14 x15
            x16 x17 x18 x19 x20 x21 x22
            x23 x24 x25 x26 x27 x29 x30"
    type="register"
</operand>
```

```
<instruction
    name="ADDI"
    num_of_operands="3"
    type="shortLat"
    operand1="general_integer_register"
    operand2="general_integer_register"
    operand3="sign_extended_constant_value_immediate"
    format="addi op1,op2,op3"
</instruction>
```

```
<operand
    id="sign_extended_constant_value_immediate"
    min="-2047"
    max="2047"
    stride="1"
    type="constant"
</operand>
```

Σχήμα 3.1: Ορισμός Εντολών RISC-V

Πίνακας 3.1: Ορισμός Εντολών RISC-V

	Name	Format
1	***"ADD"*** Add	ADD rd,rs1,rs2
2	***"ADDI"*** Add Immediate	ADDI rd,rs1,imm
3	***"SUB"*** Subtract	SUB rd,rs1,rs2
4	***"AND"***	AND rd,rs1,rs2
5	***"OR"***	OR rd,rs1,rs2
6	***"XOR"***	XOR rd,rs1,rs2
7	***"ANDI"*** And Immediate	ANDI rd,rs1,imm
8	***"ORI"*** Or Immediate	ORI rd,rs1,imm
9	***"XORI"*** Xor Immediate	XORI rd,rs1,imm
10	***"SLL"*** Shift Left Logical	SLL rd,rs1,rs2
11	***"SRL"*** Shift Right Logical	SRL rd,rs1,rs2
12	***"SRA"*** Shift Right Arithmetic	SRA rd,rs1,rs2
13	***"SLLI"*** Shift Left Logical Immediate	SLLI rd,rs1,imm
14	***"SRLI"*** Shift Right Logical Immediate	SRLI rd,rs1,imm
15	***"SRAI"*** Shift Right Arithmetic Immediate	SRAI rd,rs1,imm
16	***"SLT"*** Set Less Than	SLT rd,rs1,rs2
17	***"SLTU"*** Set Less Than Unsigned	SLTU rd,rs1,rs2
18	***"SLTI"*** Set Less Than Immediate	SLTI rd,rs1,imm
19	***"SLTIU"*** Set Less Than Immediate Unsigned	SLTIU rd,rs1,imm
20	***"LUI"*** Load Upper Immediate	LUI rd,imm
21	***"MUL"*** Multiply	MUL rd,rs1,rs2
22	***"MULH"*** Multiply High Signed Signed	MUL rd,rs1,rs2
23	***"MULHU"*** Multiply High Unsigned Unsigned	MULHU rd,rs1,rs2
24	***"MULHSU"*** Multiply High Signed Unsigned	MULHSU rd,rs1,rs2
25	***"DIV"*** Divide Signed	DIV rd,rs1,rs2
26	***"DIVU"*** Divide Unsigned	DIVU rd,rs1,rs2
27	***"REM"*** Remainder Signed	REM rd,rs1,rs2
28	***"REMU"*** Remainder Unsigned	REMU rd,rs1,rs2
29	***"SW"*** Store Word	SW rs2,offset(rs1)
30	***"LW"*** Load Word	LW rd,offset(rs1)

Οι εντολές που προκύπτουν απο τον αλγόριθμο αποθηκεύονται σε ένα αρχείο assembly, το main_original. Το αρχείο διαθέτει την ετικέτα *#loop_code* μέσα σε έναν βρόγχο,η οποία θα αντικατασταθεί με τις ενολές RISC-V που προέκυψαν απο τον αλγορίθμο. Μέσα στον συγκεκριμένο αρχείο γράφεται και προκαθορισμένος κώδικας όπως για παράδειγμα η αρχικοποίηση των καταχωρητών,η οποία γίνεται με συγκεκριμένες τιμές οι οποίες βοηθούν στην αύξηση της εναλλαγής των bit[0x00000000,0x00000001,0xFFFFFFFF,0x555555550x33333333]. Παρακάτω φαίνεται το αρχείο,κομμάτι κώδικα assembly με τον άδειο βρόγχο όπου θα τοποθετηθούν οι εντολές. Το συνολικό αρχείο assembly βρίσκεται στο παράρτημα Α στο σχήμα Α.1.

```
    addi x31,x0,2047        #x31 = 2047
Start:

    #loop_code

    addi  x31,x31,-1
    bnez x31,Start
ret
```

Πρόγραμμα 3.1: Κομμάτι κώδικα main_original που απεικονίζει τον κενό βρόγχο

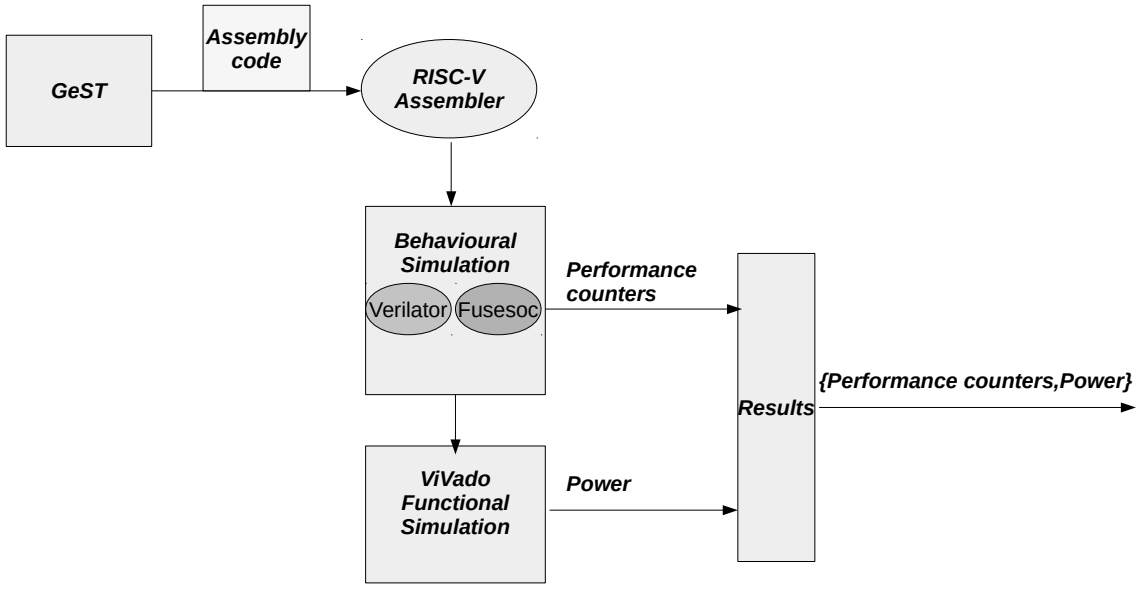
3.3 Σύνδεση του GeST και Προσομοίωση

Το GeST δίνει την δυνατότητα στο χρήστη να ορίζει την δική του διαδικασία για τις μετρήσεις που θέλει να κάνει. Υπάρχει ένα πρότυπο σε κώδικα python, η κλάση measurements η οποία μπορεί να χρησιμοποιηθεί για να δημιουργηθεί αντίστοιχη με τα στοιχεία που θέλει ο κάθε χρήστης. Βασική προυπόθεση στο configuration αρχείο στην μεταβλητή <measurementClass value=" "/> να οριστεί το όνομα της κλάσης που θα καλεί ο αλγόριθμος. Έτσι δημιουργήθηκε η κλάση MeasurementHPerfCounters_Vivado μαζί με ένα αρχείο script το makeMeasurement_PerfCounters_Vivado όπου είναι υπεύθυνα για την αντιγραφή του κώδικα στην μνήμη,την προσομοίωση σε επίπεδο συμπεριφοράς που γίνεται σε πρώτο στάδιο, την αντιγραφή των αποτελεσμάτων δηλαδή των Performance Counters όπου δίνονται σαν έξοδος και την έναρξη του δεύτερου σταδίου αυτό της λειτουργικής προσομοίωσης.Η συγκεκριμένη γίνεται για την καταγραφή της δραστηριότητας μεταγωγής στο vivado simulation με τον ίδιο κώδικα,όπου παράγει το αρχείο post_implementation_power_result ,και

λαμβάνεται η τιμή της κατανάλωσης ισχύος.Ο κώδικας και των δύο βρίσκεται στο Παράρτημα Α στο σχήμα (A.3). Η συγκεκριμένη τιμή της κατανάλωσης ισχύος δίνεται σαν είσοδο πίσω στον GeST για να συνεχίσει η λειτουργία του (Σχήμα 3.2). Έτσι με τις δύο προσομοιώσεις δημιουργείται ενα ζευγάρι απο Performance counters - power για ένα συγκεκριμένο κομμάτι εντολών RISC-V που έχει παράξει το GeST με την εξής μορφή :

[1039955 , 343941 , 0 , 16374 , 0 , 8195 , 5 , 8188 , 8187 , 24600 , 73692 , 597759 , 352149 , 679643 , 12527.565]. Οπου ο κάθε counter αντιστοιχεί :

[Cycles,Instructions Retired,LSU Busy,IFetch wait,Loads,Stores,Jumps,Branches,Taken Branches,Compressed Instructions,Multiply Wait,Divide Wait, instruction_fetch,total_stalls]



Σχήμα 3.2: Σύνδεση GeST-Ibex στα πλαίσια Προσομοίωσης

3.4 Σύνδεση του GeST και Μετρήσεις Πραγματικών Τιμών με το Arty A7

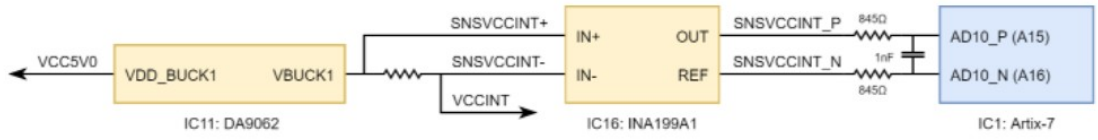
Όπως περιγράφηκε προηγουμένως το GeST παράγει ένα κώδικα σε γλώσσα μηχανής. Αφού μεταγλωττιστεί αποθηκεύεται στην μνήμη του board μέσω της σειριακής επικοινωνίας και του πρωτοκόλλου UART όπου παρουσιάζεται σε επόμενη ενότητα. Μόλις εκτελεστεί ο κώδικας αποθηκεύονται στην μνήμη οι performance counters

μαζί με την τιμή του XADC και στέλνονται μέσω της σειριακής επικοινωνίας πίσω στον υπολογιστή. Η τιμή του XADC περνάει σαν είσοδο στο GeST για να μπορέσει να συνεχιστεί ο αλγόριθμος για την παραγωγή του επόμενου κώδικα μηχανής.

Για την υλοποίηση χρησιμοποιήθηκε ο Arty A7-100T(xc7a100tcsg324-1), FPGA της Xilinx [15]. Κάποια κύρια χαρακτηριστικά του είναι ότι διαθέτει 101,440 Logic Cells,240 DSP Slices,4.860(Kbits) Memory ,USB-JTAG κύκλωμα προγραμματισμού και analog-to-digital converter (XADC) [16].

3.4.1 Λήψη και Αποθήκευση της Τιμής του Ρεύματος Μέσω XADC Wizard

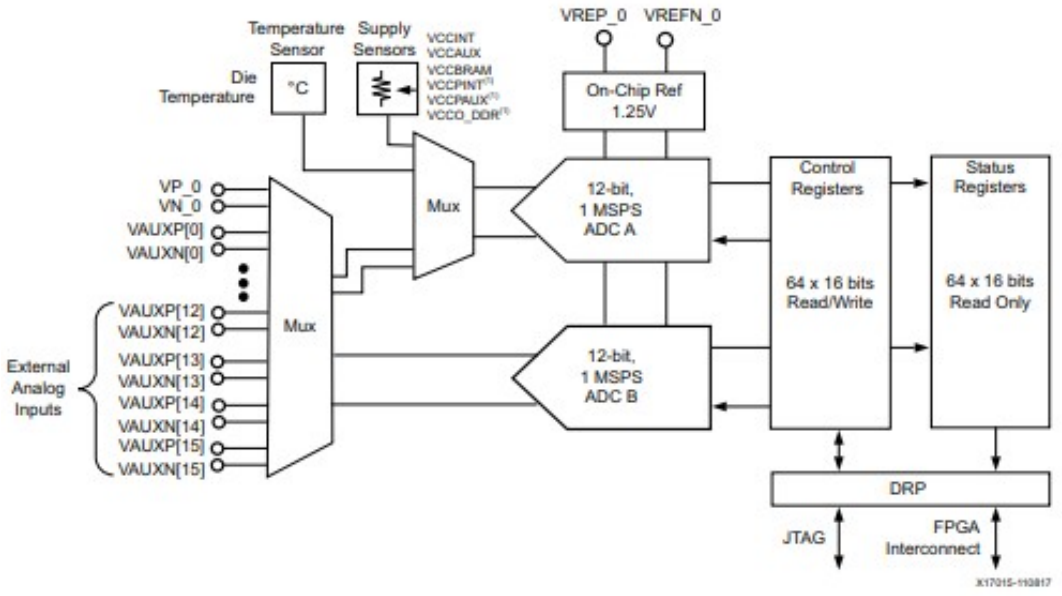
Το συγκεκριμένο board περιλαμβάνει κύκλωμα για την παρακολούθηση του ρεύματος που καταναλώνεται από τον πυρήνα. Το ρεύμα παρακολουθείται με μέτρηση της τάσης σε μια αντίσταση δέκα milliohm που τοποθετείται μεταξύ της εξόδου του συστήματος μετατροπέα DC-DC, και του δικτύου VCCINT. Κατά μήκος της αντίστασης είναι συνδεδεμένος ένας ενισχυτής αίσθησης ρεύματος και παράγει τάση εξόδου πεντακόσια millivolt ανά amp ρεύματος. Η έξοδος του ενισχυτή αίσθησης ρεύματος τροφοδοτείται στο βοηθητικό κανάλι 10 στο XADC του board. Αυτό το κύκλωμα αίσθησης ρεύματος μπορεί να μετρήσει ρεύμα μεταξύ μηδέν και δύο Amp. Στο παρακάτω σχήμα 3.3 παρουσιάζεται το συγκεκριμένο κύκλωμα.



Σχήμα 3.3: FPGA Core Supply παρακολούθηση ρεύματος

Το Arty A7 έχει το δικό του on chip ADC όπου ονομάζεται XADC είναι ένα dual channel ADC των 12 bit, με 17 auxiliary analog input channels,όπου μετατρέπει το αναλογικό σε ψηφιακό σήμα. Το XADC μπορεί να ρυθμιστεί ώστε να επιτρέπει τον υπολογισμό του μέσου ορου των δειγμάτων για την μείωση του θορύβου,ενα χαρακτηριστικό που χρησιμοποιήθηκε στην εργασία. Στο παρακάτω σχήμα 3.4 φαίνεται

το διάγραμμα του XADC όπως παρουσιάζετε στο [16].



Σχήμα 3.4: Διάγραμμα XADC

Για την προσθήκη του στην εργασία αρκεί να συμπεριληφθεί το wizard που παρέχει η Xilinx και διαμορφώνει το XADC.Το συγκεκριμένο κύκλωμα παίρνει συνεχόμενα δείγματα και δημιουργεί τον μέσο όρο των τιμών. Μεταξύ των ρυθμίσεων που διαθέτει,βρίσκεται και αυτή της επιλογής στον αριθμό των δειγμάτων που θα λαμβάνονται κάθε φορά για τον μέσο όρο.Στα πλαίσια της εργασίας επιλέχτηκε η επιλογή του μέγιστου πλήθους δειγμάτων με αριθμό samples=256.Επιλέχτηκε η μεγαλύτερη τιμή για να μην επηρεάζεται απο τον χρόνο εκτέλεσης των προγραμμάτων. Οι τιμές που δημιουργούνται αποθηκεύονται σε καταχωρητές στην μνήμη του XADC.Για την ανάγνωση αυτών των τιμών χρησιμοποιείται ένα 16bit interface,το Dynamic Reconfiguration Port(DRP).

Σε γλώσσα περιγραφής υλικού όταν η τιμή από το wizard ληφθεί τότε αποθηκεύεται σε συγκεκριμένη τιμή στην μνήμη ώστε να μεταδοθεί στον υπολογιστή μέσω της σειριακής επικοινωνίας σε επόμενο βήμα.

3.4.2 Αποθήκευση των Τιμών των Performance counters στην Μνήμη

Η αποθήκευση των τιμών των Performance Counters γίνεται στο software με την βοήθεια των εντολών *assembly[asm]*που δίνεται η δυνατότητα να χρησιμοποιηθούν στο πρόγραμμα γλώσσας C που τρέχει στο board.
Χρησιμοποιώντας την δεσμευμένη λέξη *asm* σε ένα πρόγραμμα C ,δίνεται η δυνατότητα να εισαχθούν εντολές assembly. Οι assembly εντολές είναι της μορφής : *asm [volatile] ("assembly instruction")* με την λέξη volatile που συμπληρώνεται να είναι προαιρετική ,και χρησιμοποιείται για να μην επιτραπούν βελτιστοποιήσεις. Παράδειγμα της εντολής που υπάρχει στο C πρόγραμμα είναι:

```
1 __asm__ volatile("csrr %0,  mcycle ;" : "=r"(perfC_value));
2  *performance_counters_memory = perfC_value;
```

Πρόγραμμα 3.2: Εντολή Assembly και pointer για αποθήκευση Performance counters στην μνήμη

Το συγκεκριμένο κομμάτι κώδικα διαβάζει την τιμή του Performance Counter όπου καταχωρείται στην μεταβλητή *[perfC_value]* και αποθηκεύεται στην μνήμη με χρήση του pointer*[*performance_counters_memory]*. Η συγκεκριμένη εντολή επαναλαμβάνεται για όλους τους μετρητές με την τιμή του δείκτη να αυξάνεται κάθε φορά για να αποθηκεύονται σε διαδοχικές θέσεις (Παράρτημα Α.1).

3.4.3 Σειριακή Επικοινωνία Board-Υπολογιστή με Χρήση του Uart Protocol

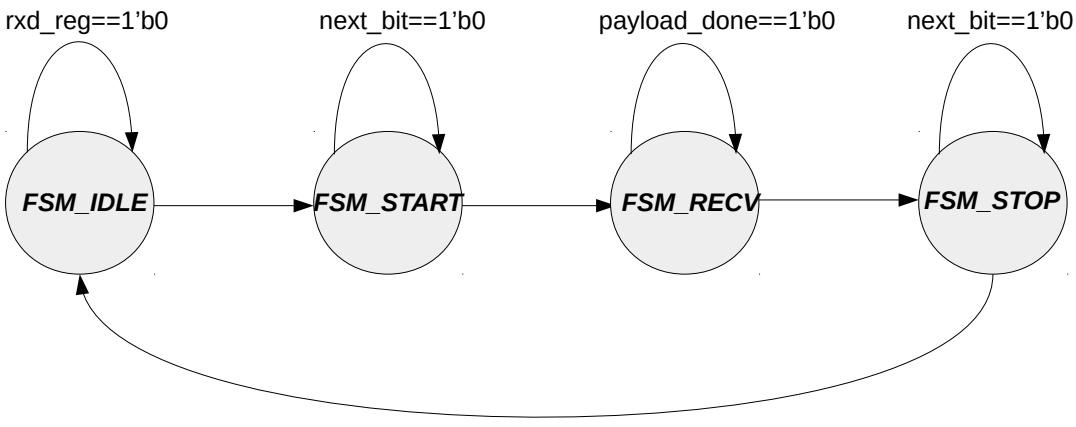
Για την σειριακή επικοινωνία χρησιμοποιείται ένα πρωτόκολλο επικοινωνίας μεταξύ δύο συσκευών το UART(Universal Asynchronous Receiver Transmitter),στην συγκεκριμένη περίπτωση του υπολογιστή με το board. Είναι υπεύθυνο για την μετάδοση και την λήψη σειριακών δεδομένων. Στην σειριακή επικοινωνία τα δεδομένα μεταφέρονται ανά bit χρησιμοποιώντας ένα καλώδιο.
Το UART περιλαμβάνει έναν πομπό(transmitter) και έναν δέκτη(receiver). Ο πομπός φορτώνει δεδομένα παράλληλα και τα μεταδίδει σειριακά,και αντίστοιχα ο δέκτης δέχεται τα δεδομένα σειριακά και τα μεταφέρει στην έξοδο του παράλληλα. Τα δεδομένα μεταδίδονται σε πακέτα. Το πακέτο αποτελείται από το start bit που βρίσκεται σε λογικό μηδέν στην διάρκεια μετάδοσης,τα data frame που είναι τα

πραγματικά δεδομένα που θα μεταδοθούν,τα parity bits για ανίχνευση σφάλματος ,stop bit που βρίσκεται σε λογικό ένα και τερματίζεται η μετάδοση του πακέτου. Η μορφή του πακέτου φαίνεται στο παρακάτω σχήμα 3.5.

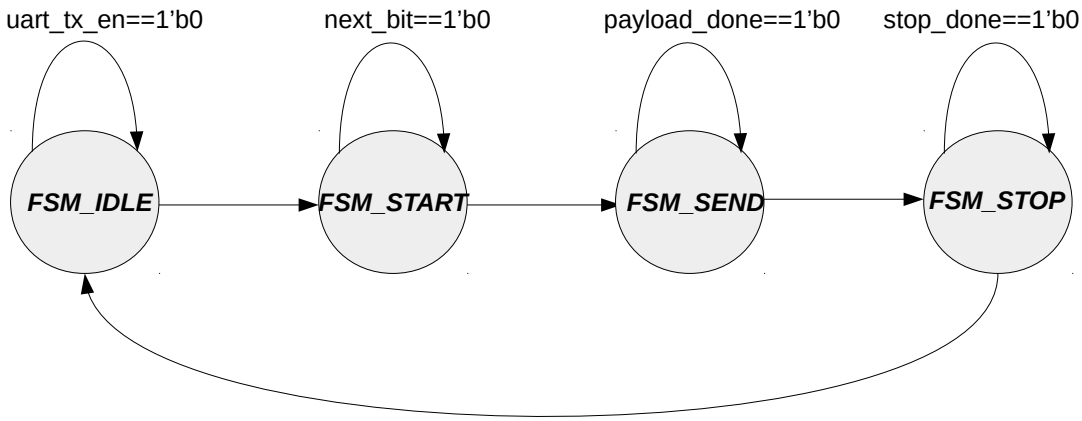
Start Bit (1 Bit)	Data Frame (5-9 Data Bits)	Parity Bits (0-1 Bit)	Stop Bit (1 Bit)

Σχήμα 3.5: Απεικόνιση Πακέτου του Uart

Για να υλοποιηθεί το πρωτόκολλο στην εργασία χρειάστηκαν δύο επιπλέον module ένα που υλοποιεί τον πομπό και έναν που υλοποιεί τον δέκτη. Τα δύο αυτά module βασίστηκαν στην υλοποίηση [17] όπου διαχειρίζονται πακέτα με δεδομένα των 8 bit. Τα διαγράμματα καταστάσεων τους φαίνονται παρακάτω σχήμα 3.6.

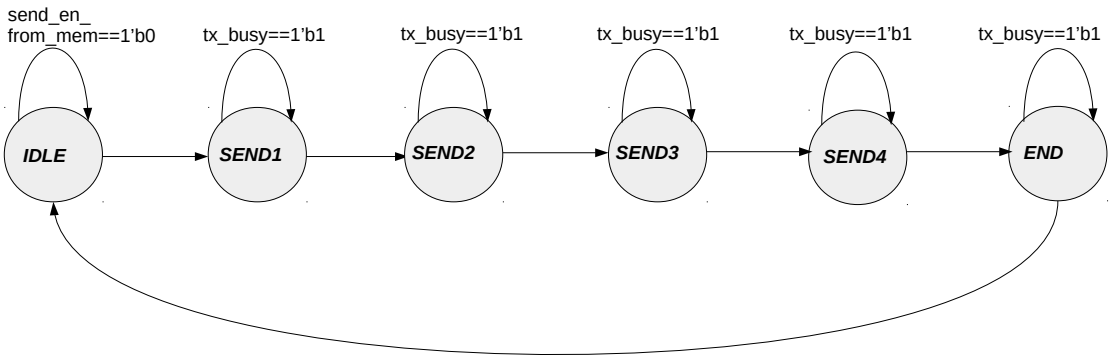


Σχήμα 3.6: Διάγραμμα Καταστάσεων Receiver



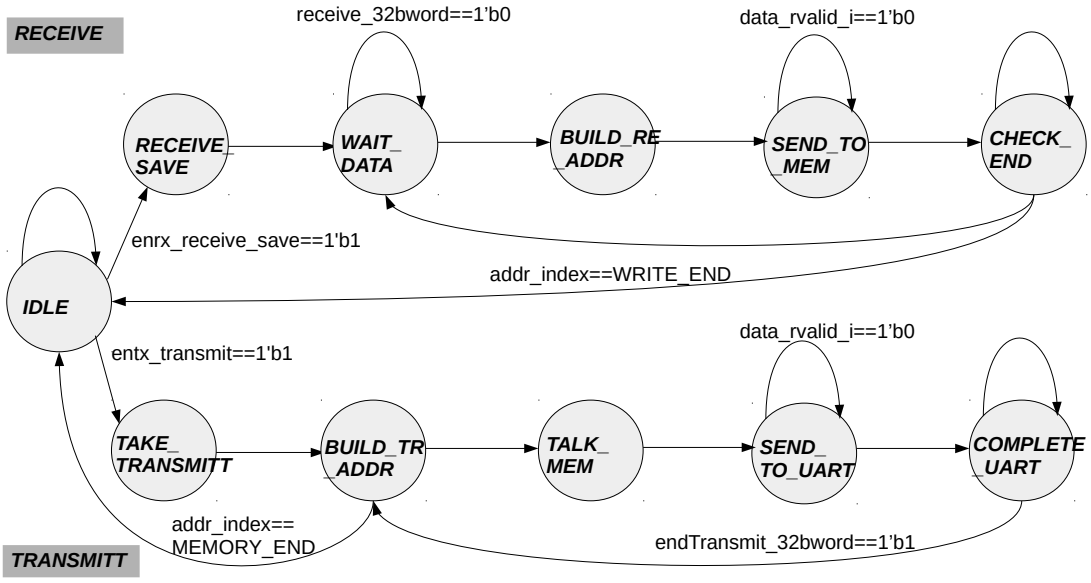
Σχήμα 3.7: Διάγραμμα Καταστάσεων Transmitter

Για να μπορέσουν να σταλούν οι λέξεις των 32 bit και αντίστοιχα να ληφθούν οι λέξεις των 32 bit,δημιουργήθηκαν δύο ακόμα module, αυτό που διαχωρίζει την λέξη για αποστολή από 32bit σε τέσσερις λέξεις των 8 bit το *split_bytes* και αυτό που δέχεται λέξεις των 8bit και τις συνθέτει σε μία των 32bit για να σταλεί στον υπολογιστή το *con_bytes*. Το module *con_bytes* δέχεται λέξεις από τον παραλήπτη *uart_rx*.Υπάρχει ένας μετρητής ώστε όταν λάβει τέσσερις λέξεις των 8 bit να προωθεί την συνολική λέξη μετά την ένωση στο module *memory_controller*. Το module *split_bytes* λαμβάνει από το *memory_controller* μία λέξη των 32bit και την χωρίζει σε τέσσερις λέξεις των 8bit. Υπάρχει ένα FSM όπου γίνεται ο διαχωρισμός της λέξης. Όταν η κάθε λέξη στέλνεται στο *uart_tx* περιμένει να λάβει σήμα ότι η λέξη στάλθηκε και δεν είναι πια απασχολημένο ώστε να μπορέσει να στείλει την επόμενη. Όταν στείλει και τα τέσσερα κομμάτια τις λέξεις στέλνει σήμα *memory_controller*. Παρακάτω υπάρχει εικόνα του διαγράμματος καταστάσεων 3.8.



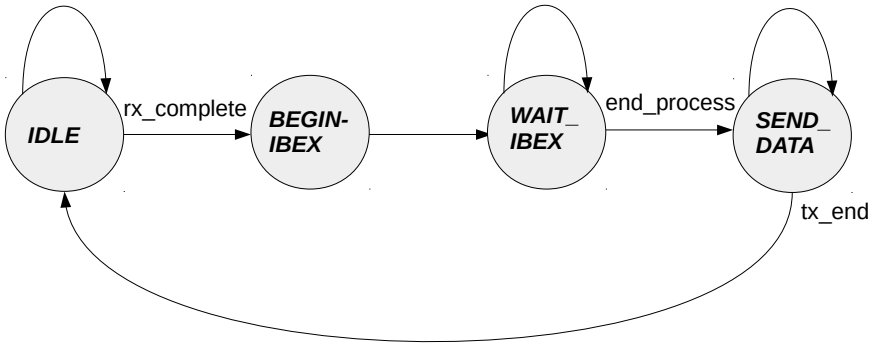
Σχήμα 3.8: Διάγραμμα Καταστάσεων Split Bytes

Η λειτουργία του *memory_controller* είναι η διαχείριση των λέξεων τόσο αυτών που λαμβάνονται από το board και αποθηκεύονται στην μνήμη,όσο και αυτών που στέλνονται σαν έξοδος από την μνήμη στον υπολογιστή. Υπάρχει Finite State Machine δύο διαδρομών (Σχήμα 3.9) όπου στην μία περίπτωση είναι υπεύθυνο για την λήψη των δεδομένων και στην άλλη για την αποστολή τους.



Σχήμα 3.9: Διάγραμμα Καταστάσεων Memory Controller

Υπάρχει ακόμα ενα βασικό για την συνολική λειτουργία module ο *controller*. Το συγκεκριμένο είναι υπευθυνο για την διαχείρηση των σημάτων εκκίνησης του ibex καθώς και την ενημέρωση του *memory_controller* ότι έχει τελειώσει η εκτέλεση του προγράμματος και μπορεί να ξεκινήσει η αποστολή των δεδομένων προς τον υπολογιστή. Υπάρχει FSM (Σχήμα 3.10) τεσσάρων καταστάσεων. Οταν ληφθεί το σήμα απο τον *memory _controller* ενεργοποιεί τον ibex μέσω του σήματος *reset* που διαθέτει αλλάζοντας την τιμή του. Ο ibex με την σειρά του διαθέτει σήμα(core_sleep) όπου ανάλογα την τιμή του δείχνει αν βρίσκεται σε λειτουργία ή οχι. Το σήμα core_sleep ενεργοποιείται απο τον ibex όταν τελείωσει η εκτέλεση του προγράμματος που υπάρχει στην μνήμη,το οποίο γίνεται μέσω της εντολής *asm volatile (" wfi ")* στον κώδικα C. Αυτο το σήμα στέλνεται στον *controller* και αλλάζοντας κατάσταση στέλνει σήμα στον *memory_controller* ώστε να αρχίσει η αποστολή των δεδομένων.



Σχήμα 3.10: Διάγραμμα Καταστάσεων Controller

Από την μεριά του υπολογιστή τρέχει ένα πρόγραμμα python με χρήση της βιβλιοθήκης PySerial[18], για σειριακές επικοινωνίες. Για την δημιουργία της σειριακής επικοινωνίας χρειάζεται ο προσδιορισμός της θύρας όπου γίνεται η μεταφορά των δεδομένων καθώς και κάποιες αρχικοποιήσεις σε μεταβλητές. Παράδειγμα κώδικα έναρξης την σύνδεσης φαίνεται στην παρακάτω εικόνα.

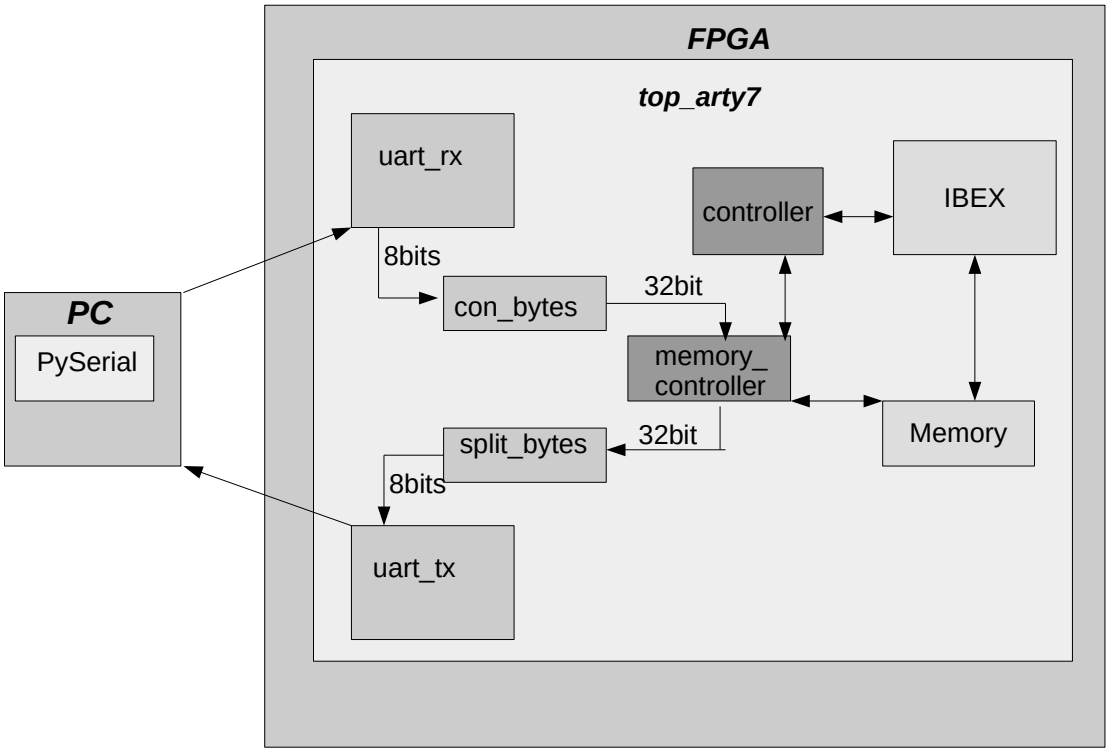
```
ser = serial.Serial(
        port='/dev/ttyUSB1',
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
    )
```

Πρόγραμμα 3.3: Configuration for the serial port

Στο κώδικα διαβάζεται αρχείο με δεκαεξαδικές λέξεις και με την εντολή *write()* μεταφέρονται στο board. Για την λήψη των δεδομένων το κανάλι ακούει συνέχεια σε πιθανά δεδομένα που θα του σταλούν με αποτέλεσμα κάθε φορά που λαμβάνει κάτι με την *read()* να το μετατρέπει σε δεκαεξαδική λέξη και να το αποθηκεύει σε αρχείο.

Στο παρακάτω σχήμα 3.11 βλέπουμε τα module που εμπλέκονται στην συνολική λειτουργία μιας μέτρησης καθώς και στην σειριακή επικοινωνία του board με τον

υπολογιστή.



Σχήμα 3.11: Σειρική Επικοινωνία Uart

3.5 Περιγραφή Νευρωνικού Δικτύου

Η δημιουργία του νευρωνικού δικτύου βασίστηκε στη υλοποίηση της δημοσίευσης [19].Στην συγκεκριμένη δημοσίευση έχει δημιουργηθεί νευρωνικό δίκτυο για την πρόβλεψη της κατανάλωσης ισχύος,εκπαιδευμένο χρησιμοποιώντας performance counters και του επεξεργαστή και της κάρτας γραφικών.Το μοντέλο έχει δύο κρυφά επίπεδα,με d=26 τα χαρακτηριστηκά που αποτελούνται απο 12 μετρητές απόδοσης του επεξεργαστή και 14 της κάρτας γραφικών και ο αριθμός των νευρώνων ανα κρυφό επίπεδο είναι H=50.

Συνολικά για την εκπαίδευση του νευρωνικού δικτύου χρησιμοποιήθηκαν m=423 δείγματα κάθε ένα εκ των οποίων αποτελείται από d=14 χαρακτηριστικά(Performance Counters) και μια τιμή Current που αντιστοιχεί σε αυτά. Το Νευρωνικό Δίκτυο που χρησιμοποιήθηκε στη συγκεκριμένη εργασία είναι ένα πολυεπίπεδο δίκτυο Percep-

tron πρόσθιας τροφοδότησης. Η αρχιτεκτονική του δικτύου είναι η εξής: αποτελείται από ένα διάνυσμα $x = (x_1, x_2, .., x_d)$ που αποτελεί το επίπεδο εισόδου, δύο κρυφά επίπεδα αποτελούμενα από H=50 νευρώνες το κάθε ένα και ένα επίπεδο υπολογιστικών κόμβων εξόδου, το οποίο επιστρέφει την τιμή του ρεύματος. Το βασικό χαρακτηριστικό ενός τέτοιου δικτύου είναι ότι οι νευρώνες οποιουδήποτε επιπέδου έχουν σαν είσοδο τα σήματα εξόδου μόνο του προηγούμενο επιπέδου. Οι νευρώνες των κρυφών επιπέδων και του επιπέδου εξόδου δίνονται ο κάθε ένας ως όρισμα σε μία μη γραμμική συνάρτηση ενεργοποίησης, όπου στη συγκεκριμένη εργασία είναι η σιγμοειδής συνάρτηση :

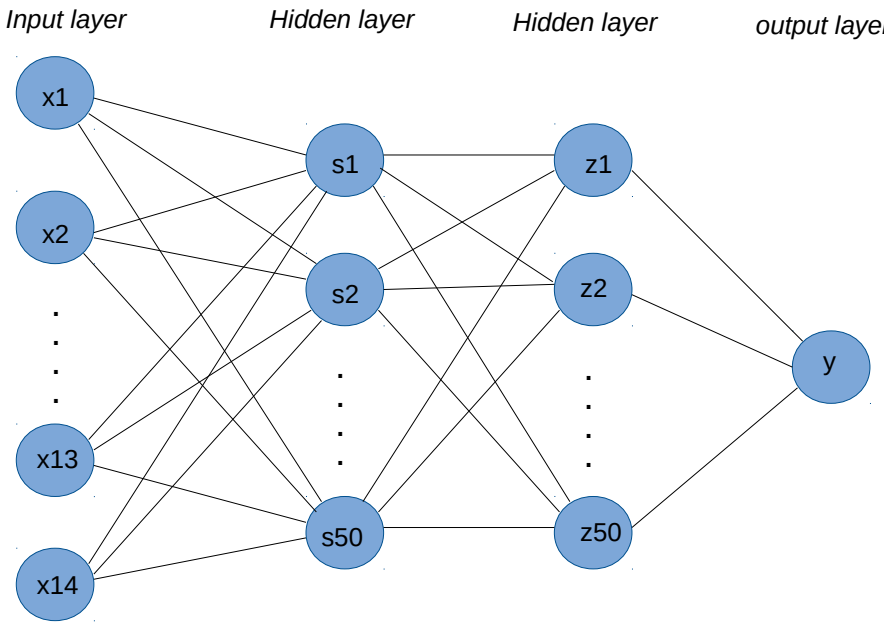
$$h : R \to R : h(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

Ο βασικός στόχος μας είναι η εκπαίδευση του Νευρωνικού δικτύου $N_\theta(x)$, δηλαδή η εύρεση των βέλτιστων παραμέτρων του $\theta = \{weights, biases\}$. Η συνάρτηση σφάλματος που θέλουμε να ελαχιστοποιήσουμε για αυτό το σκοπό δίνεται από τον τύπο:

$$E(\theta) = \sum_{i}^{m} \{N_\theta(x_\mathrm{i}) - y_\mathrm{i}\}^2 \tag{3.2}$$

όπου x_i είναι το διάνυσμα των d χαρακτηριστικών, $N_\theta(x_\mathrm{i}) \approx \tilde{y}_\mathrm{i}$ είναι το Current που προβλέπει το Νευρωνικό Δίκτυο, y_i είναι η πραγματική τιμή του Current και m είναι το πλήθος των δεδομένων.

Το Νευρωνικό Δίκτυο που περιγράφηκε παραπάνω έχει την εξής δομή (Σχήμα 3.12):



Σχήμα 3.12: Απεικόνιση Νευρωνικού Δικτύου

Η εκπαίδευση του δικτύου αποτελείται από δύο στάδια, το πέρασμα προς τα εμπρός και το πέρασμα προς τα πίσω. Στο πέρασμα προς τα εμπρός το διάνυσμα εισόδου x εφαρμόζεται στους νευρώνες του δικτύου, έχοντας σταθερές τιμές στις παραμέτρους και στο τέλος παράγεται η έξοδος η οποία είναι η απόκριση του δικτύου. Στο πέρασμα προς τα πίσω τα βάρη διορθώνονται σύμφωνα με τον κανόνα διόρθωσης που έχει οριστεί ακολουθώντας κάποια μέθοδο βελτιστοποίησης για την ελαχιστοποίηση του κριτηρίου κόστους (Συνάρτηση 3.2). Οι παράμετροι διορθώνονται με τέτοιο τρόπο ώστε η απόκριση του δικτύου να πλησιάζει την πραγματική τιμή. Στη συγκεκριμένη εκπαίδευση χρησιμοποιήθηκε ο αλγόριθμος back propagation.

Κεφαλαιο 4

Αποτελεσματα και Αξιολογηση

4.1 Αποτελέσματα Προσομοίωσης

4.2 Αποτελέσματα Μετρήσεων στο FPGA

4.3 Σύγκριση και Αξιολόγηση των Μεθόδων Μέτρησης

4.4 Συσχέτιση των Αποτελεσμάτων με Μετροπρογράμματα

4.5 Αποτελέσματα Νευρωνικού δικτύου

Στο παρόν κεφάλαιο θα παρουσιαστούν τα πειράματα και τα αποτελέσματα της εργασίας τόσο σε επίπεδο προσομοίωσης όσο και σε επίπεδο μετρήσεων στο FPGA. Στην συνέχεια παρουσιάζεται η σύγκριση των μεθοδολογιών των αποτελεσμάτων καθώς και η αξιολόγηση τους σύμφωνα με τα μετροπρογράμματα που χρησιμοποιήθηκαν σαν σημείο αναφοράς. Τέλος αναλύονται τα αποτελέσματα του νευρωνικού δικτύου.

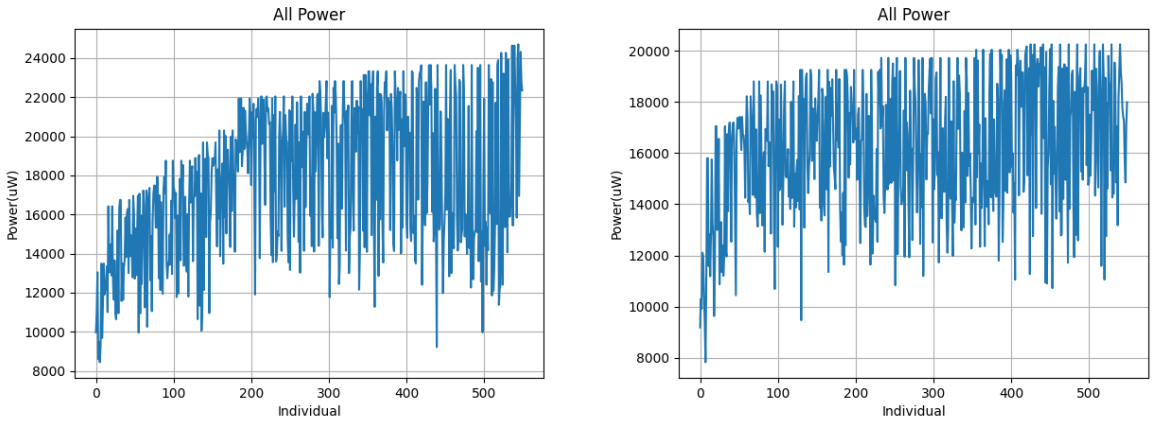
Στις παρακάτω μετρήσεις παραθέτονται διαγράμματα τριών διαφορετικών ομάδων. Κατά σειρά οι ομάδες διαγραμμάτων παρουσιάζουν, όλα τα αποτελέσματα μετρήσεων για κάθε individual,δηλαδή για κάθε ξεχωριστό πρόγραμμα assembly που δημιουργήθηκε από το GeST. Στην συνέχεια οι μέγιστες τιμές για κάθε population δηλαδή το πρόγραμμα εκείνο που παρήγαγε την μεγαλύτερη τιμή στην συγκεκριμένη ομάδα των προγραμμάτων assembly και τέλος το διάγραμμα με τις συσχετίσεις του κάθε μετρητή στο αποτέλεσμα.Στους πίνακες όπως επίσης σε κάποια διαγράμματα αλλά και στα δεδομένα για το νευρωνικό δικτύο οι τιμές των μετρητών επίδοσης έχουν διαιρεθεί με τον αριθμό των κύκλων εκτέλεσης και οι τιμές τους

βρίσκονται μεταξύ $[0, 1]$.Η αναλογία των τιμών σε σχέση με των αριθμό των κύκλων μας διευκολύνει τόσο στην κατανόηση των αποτελεσμάτων όσο και στην διαχείριση τους και μπορούν οι συγκρίσεις των αποτελεσμάτων και τα συμπεράσματα να αναδειχθούν ευκολότερα.

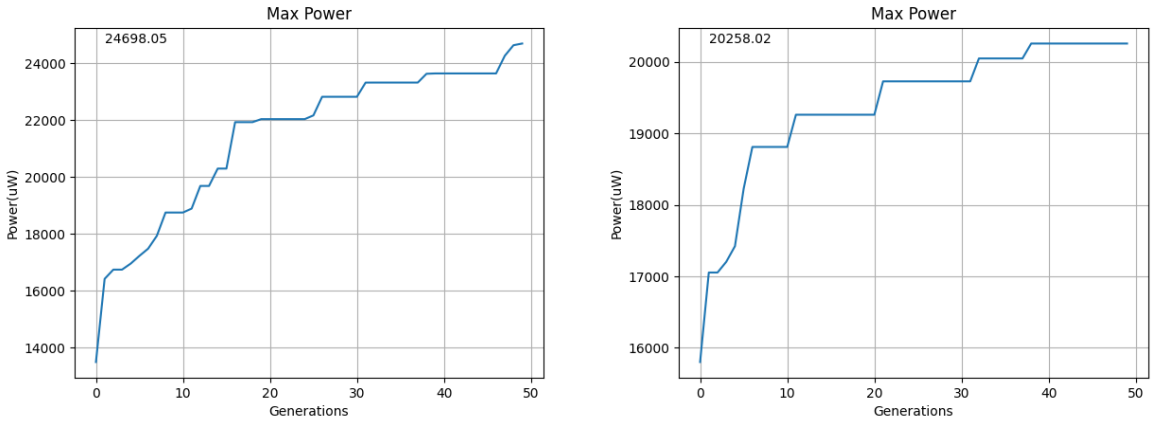
4.1 Αποτελέσματα Προσομοίωσης

Η πρώτη μέθοδο μετρήσεων,αυτή της προσομοίωσης, έγινε με την χρήση των εργαλείων Verilator,Fusesoc. Το Verilator παρουσιάζει στην έξοδο του τους Performance Counters και με την ενεργοποίηση συγκεκριμένης παραμέτρου δημιουργεί και αρχείο με την ανάλυση της Ισχύος. Παρακάτω βλέπουμε τα αποτελέσματα των προσομοιώσεων με αριθμό επαναλήψεων(loop size)ls=2.047 και ls=8.188.

Όπως αναφέρθηκε στο (Σχήμα 4.1) φαίνονται οι τιμές της Ισχύος για το κάθε πρόγραμμα assembly που παράγει ο GeST. Το μεγάλο εύρος στις τιμές είναι διότι οι αλλαγές που γίνονται κάθε φορά στο κώδικα μπορεί να μην έχουν τα αποτελέσματα που περιμένει ο αλγόριθμος.Με κάποιες αλλαγές θα προκύψει αύξηση της Ισχύος ενώ με κάποιες μείωση. Ωστόσο,για κάθε σύνολο απο αυτά τα μοναδικά προγράμματα assembly ο αλγόριθμος συγκρατεί την μέγιστη τιμή Ισχύος ,και το πρόγραμμα assembly που το παρήγαγε είναι και αυτό που ορίζεται σαν σημείο αναφοράς στο επόμενο σύνολο προγραμμάτων. Όπως φαίνεται λοιπόν στο (Σχήμα 4.2) οι μέγιστες τιμές έχουν αυξητική τάση. Στην πρώτη εικόνα η άνοδος της τιμής είναι πιο ομαλή ενώ στην δεύτερη πιο απότομη.Ο λόγος δεν είναι η διαφορά στον αριθμό των επαναλήψεων αλλά ότι το GeST είχε σαν έξοδο τυχαία γρήγορα ένα πρόγραμμα όπου παρήγαγε μεγάλη τιμή. Παρόλα αυτά στην πρώτη περίπτωση ο αλγόριθμος δημιούργησε κώδικα που σαν αποτέλεσμα είχε μεγαλύτερη τιμή ισχύος,από την δεύτερη περίπτωση που συγκλίνει νωρίτερα.



Σχήμα 4.1: Η τιμή της κατανάλωσης Ισχύος για κάθε Individual ls=2.047,ls=8.188



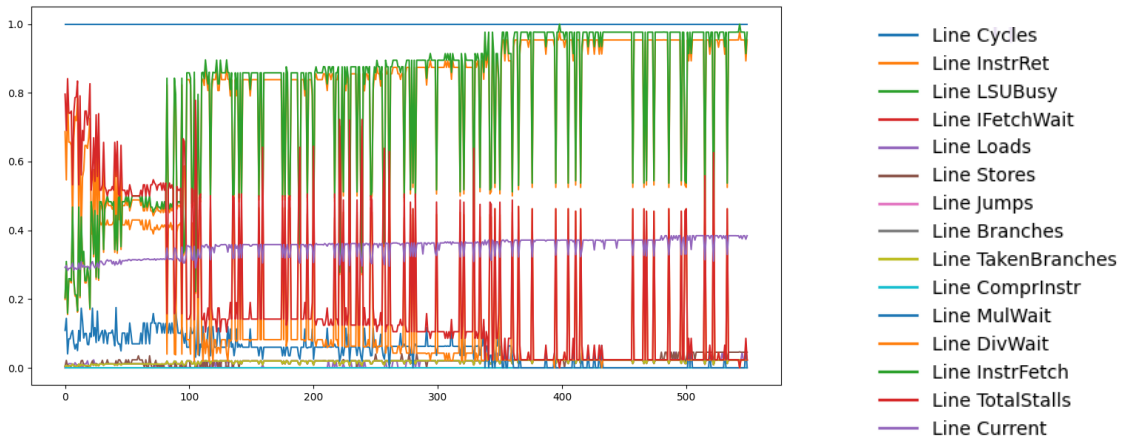
Σχήμα 4.2: Μέγιστη τιμή κατανάλωσης Ισχύος για κάθε population ls=2.047,ls=8.188

4.2 Αποτελέσματα Μετρήσεων στο FPGA

Η δεύτερη μέθοδος είναι αυτή των μετρήσεων στο FPGA. Με τους μηχανισμούς και την υλοποίηση που παρουσιάστηκε στο προηγούμενο κεφάλαιο αντλήθηκαν οι μετρητές επίδοσης και η μέτρηση του ρεύματος. Τα πειράματα έχουν χωριστεί με βάση το πλήθος των επαναλήψεων της εκτέλεσης του παραγόμενου προγράμματος assembly. Έτσι έχουμε το πείραμα όπου το πλήθος των επαναλήψεων είναι ls=2.047 ,το πείραμα όπου είναι ls=1.000.000 και όπου είναι ls=10.000.000

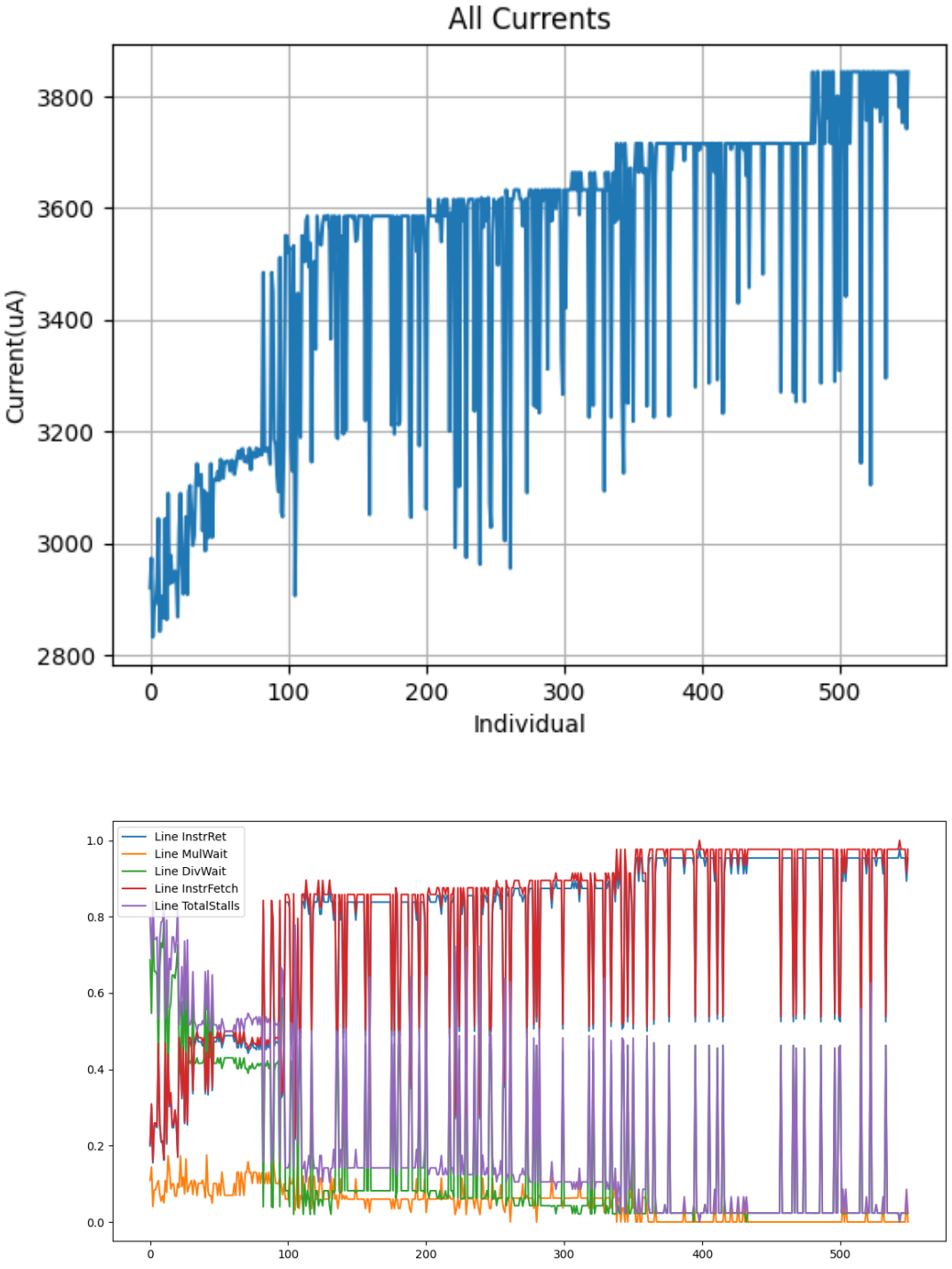
Στο πρώτο γράφημα (Σχήμα 4.3)απεικονίζεται η πορεία του κάθε μετρητή στην διάρκεια μιας μέτρησης. Παρατηρείται ότι η πορεία των μετρητών καθορίζει και την πορεία της τιμής του ρεύματος. Αυτοί,με τις πολύ χαμηλές τιμές,η πορεία τους είναι

σχεδόν ευθεία με αποτέλεσμα να μην επηρεάζουν την τιμή ρεύματος. Οι μετρητές MulWait,DivWait,TotalStalls στην πορεία του γραφήματος, φαίνονται να μειώνονται ενώ ανεβαίνει η κατεύθυνση της τιμής του ρεύματος.Έτσι δείχνουν ότι επηρεάζουν την τιμή της,συμπεραίνοντας ότι κάποιοι από τους μετρητές είναι σημαντικότεροι.



Σχήμα 4.3: Πορεία των τιμών όλων των Performance Counters

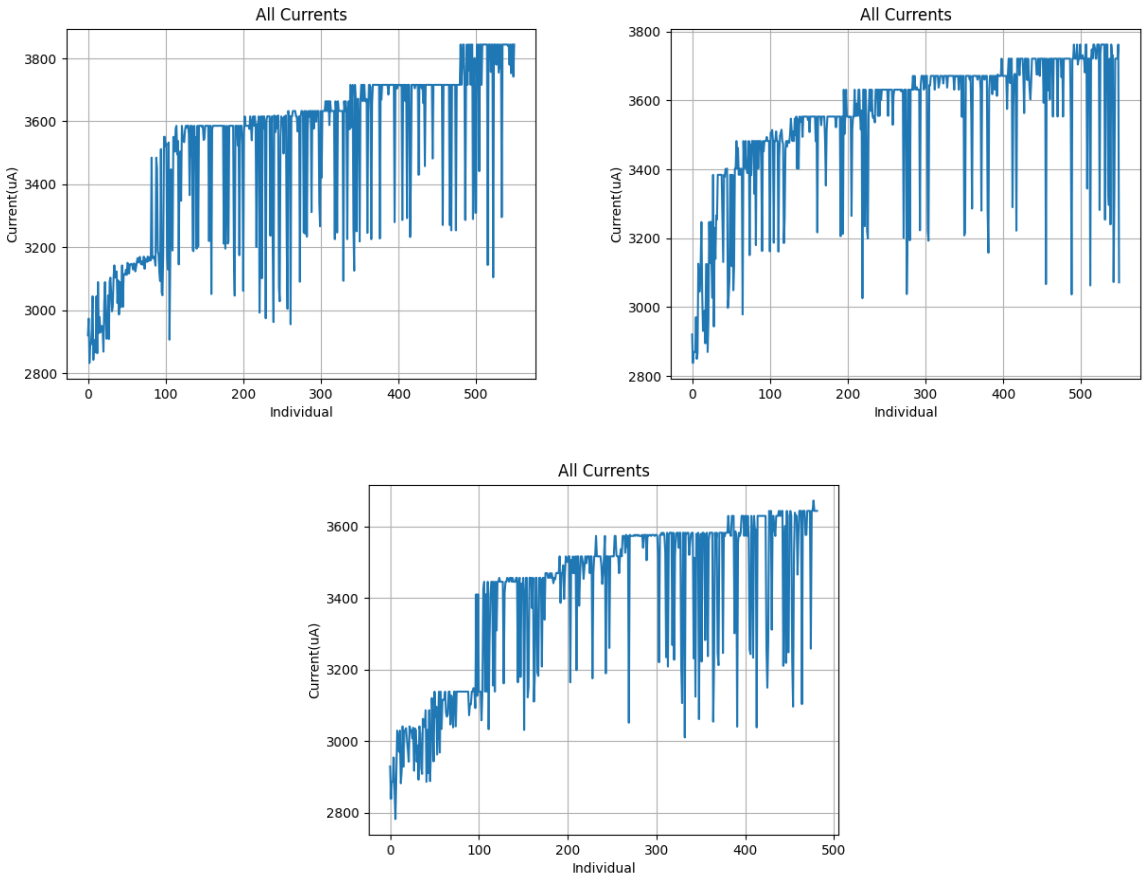
Στο (Σχήμα 4.4) φαίνονται οι μετρητές οι οποίοι έχουν την μεγαλύτερη συσχέτιση και η τιμή του ρεύματος.Οι μετρητές MulWait και DivWait έχουν άμεση σχέση με τον μετρητή TotalStalls ,αφού ο τελευταίος συμπεριλαμβάνει στις μετρήσεις τους τους κύκλους απο τις καθυστερήσεις του πολλαπλασιασμού και της διαίρεσης.Είναι ολοφάνερο απο το γράφημα ότι όταν ο μετρητής TotalStalls κατεβαίνει τότε η τιμή ρεύματος ανεβαίνει.Οπως επίσης φαίνεται ξεκάθαρα ότι οι μετρητές InstrFetch και InstrRet έχουν άμεση σχέση αφού ανεβαίνοντας ανεβαίνει και η τιμή του ρεύματος.



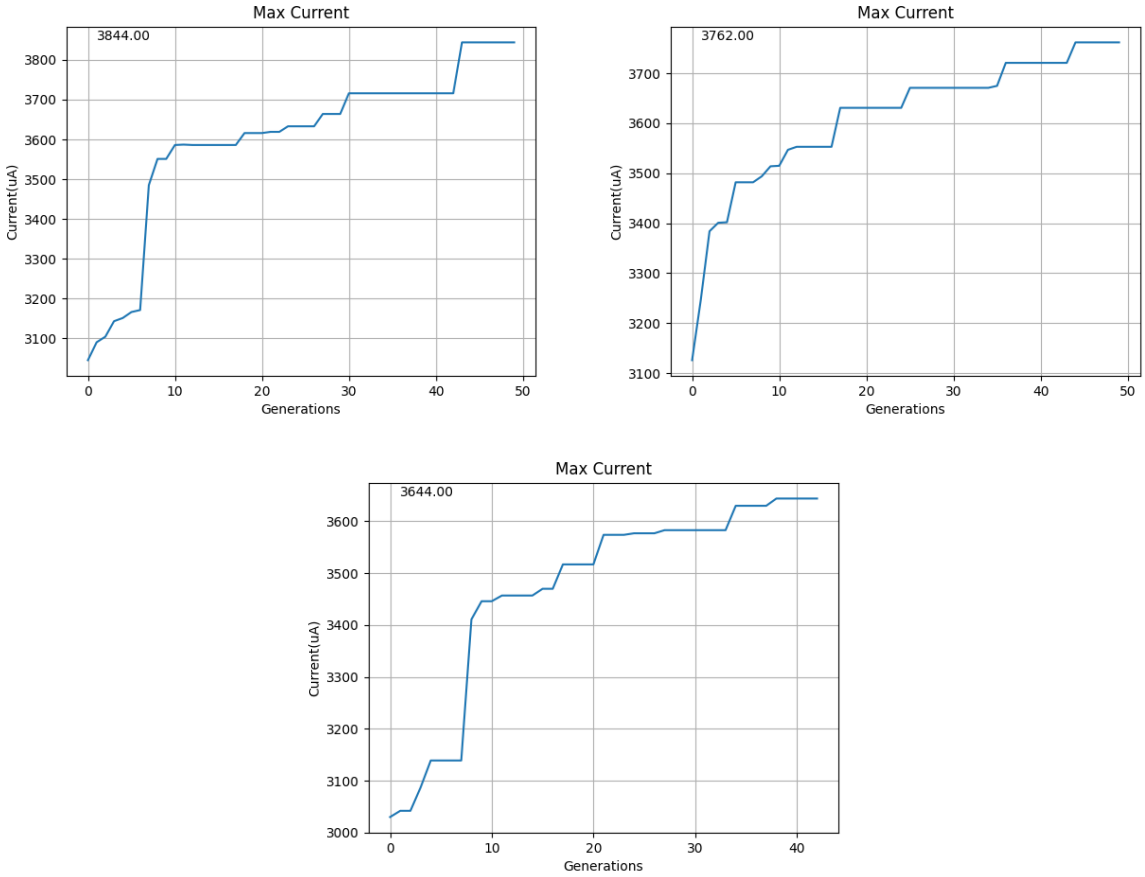
Σχήμα 4.4: Performance Counters με την μεγαλύτερη συσχέτιση στις μετρήσεις

Όπως και στα αποτελέσματα της προσομοίωσης ,υπάρχουν τα διαγράμματα για τις μετρήσεις με αριθμό επαναλήψεων ls=2.047,ls=1.000.000,ls=10.000.000. Τα πρώτα διαγράμματα (Σχήμα 4.5) απεικονίζουν τις τιμές ρεύματος για κάθε Indi-

vidual ενώ τα διαγράμματα (Σχήμα 4.6) απεικονίζουν τις μέγιστες τιμές για κάθε population. Στα διαγράμματα με τις μέγιστες τιμές φαίνεται ότι οι τιμές δεν έχουν ομαλή αύξηση και μπορεί η μέγιστη τιμή να διαφέρει αναλόγως τις αλλαγές.



Σχήμα 4.5: Τιμές Ρεύματος για κάθε Individual ls=2.047,ls=1.000.000,ls=10.000.000



Σχήμα 4.6: Μέγιστη τιμή Ρεύματος για κάθε population ls=2.047,ls=1.000.000,ls=10.000.000

4.3 Σύγκριση και Αξιολόγηση των Μεθόδων Μέτρησης

Τα αποτελέσματα τόσο των μετρήσεων της προσομοίωσης όσο και αυτά των μετρήσεων στο board δείχνουν ότι τα προγράμματα assembly που παράγει ο GeSt έχουν τα επιθυμητά αποτελέσματα των τιμών τόσο της Ισχύος όσο και του ρεύματος. Τα αποτελέσματα των μέγιστων τιμών συγκεντρωτικά φαίνονται στον παρακάτω πίνακα 4.3.

2047	8188
24698.05	20258.02

Πίνακας 4.1: Simulation Results

2047	1.000.000	10.000.000
3844.00	3762.00	3644.00

Πίνακας 4.2: FPGA Results

Πίνακας 4.3: Αποτελέσματα των μέγιστων τιμών ανά επανάληψη

Μία ακόμα διαφορετική προσέγγιση στις μετρήσεις όπου το πρόγραμμα assembly αυτό με την μέγιστη τιμή με πλήθος επαναλήψεων ls=2.047 δοκιμάστηκε και με ls=1.000.000 και ls=10.000.000

Performance Counters	*ls=2047*	*ls=1.000.000*	*ls=10.000.000*
Cycles	90171	44000104	440000104
Instructions Retired	0.95	0.95	0.95
LSU Busy	0	0	0
Fetch Wait	0.022	0.022	0.022
Loads	0.022	0.022	0.022
Stores	0.045	0.045	0.045
Jumps	0.0000022	0.0000022	0.0000022
Conditional Branches	0.022	0.022	0.022
Taken Conditional Branches	0.022	0.022	0.022
Compressed Instructions	0	0	0
Multiply Wait	0	0	0
Divide Wait	0.022	0.022	0.022
Instruction Fetch	0.97	0.97	0.97
Total Stalls	0.022	0.022	0.022
Current	**3844**	**3844**	**3844**

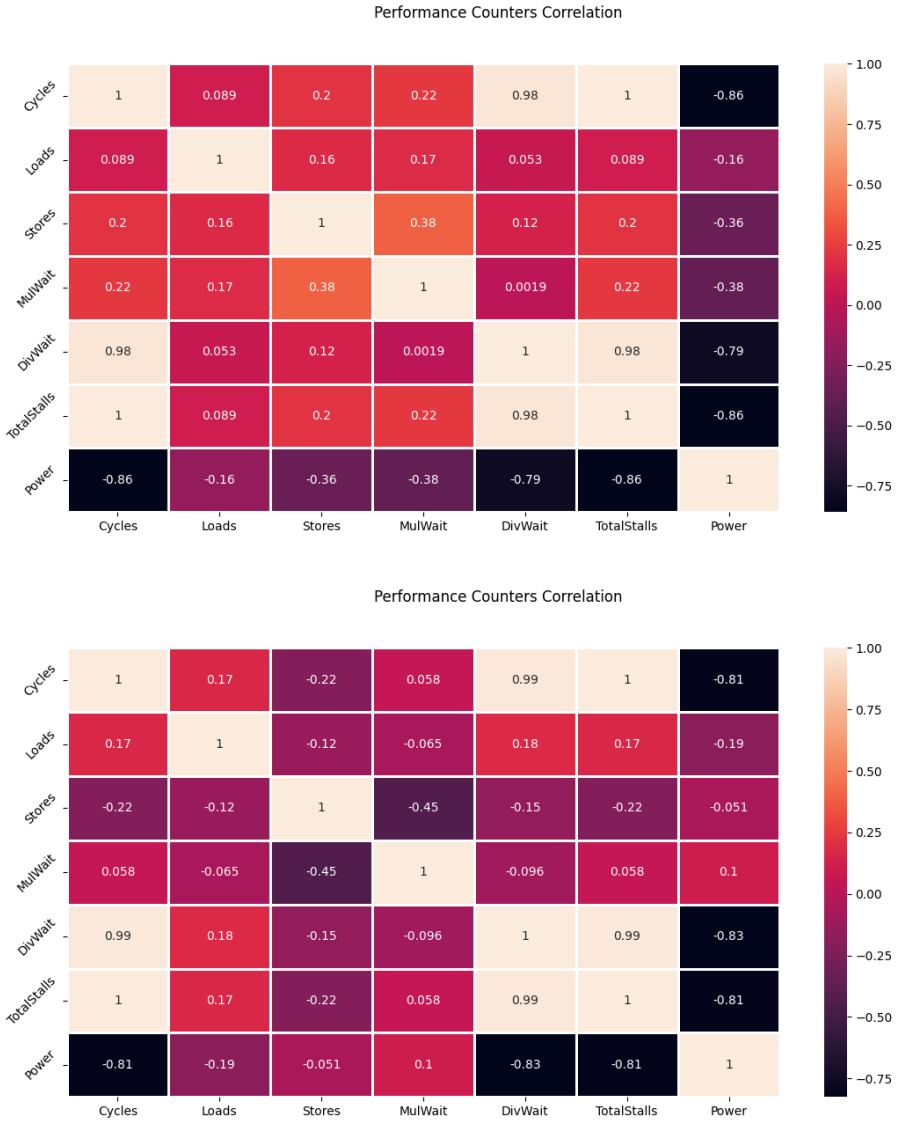
Πίνακας 4.4: Αποτελέσματα για διαφορετικό πλήθος επαναλήψεων

Το βασικό συμπέρασμα που προκύπτει τόσο απο τους παραπάνω πίνακες όσο και τα προηγούμενα διαγράμματα είναι ότι ο αριθμός των επαναλήψεων δεν παίζει σημαντικό ρόλο στην μέγιστη τιμή που θα προκύψει αλλα και στην απότομη αύξηση της τιμής. Περισσότερο πλήθος επαναλήψεων ισοδυναμεί με περισσότερο χρόνο,και η τιμή της κατανάλωσης ρευματος είναι ανάλογη. Ο συνδυασμός των τιμών των Performance Counters στο πρόγραμμα που εκτελείται είναι αυτός που καθορίζει την τιμή της ισχύος ή του ρεύματος που θα παραχθεί. Αν οι αλλαγές που προκύπτουν στο πρόγραμμα φέρουν απο νωρίς μια μεγάλη τιμή τότε προκύπτει η απότομη αύξηση και φέρει μεγάλες πιθανότητες να προκύψει και μεγαλύτερη μέγιστη τιμή

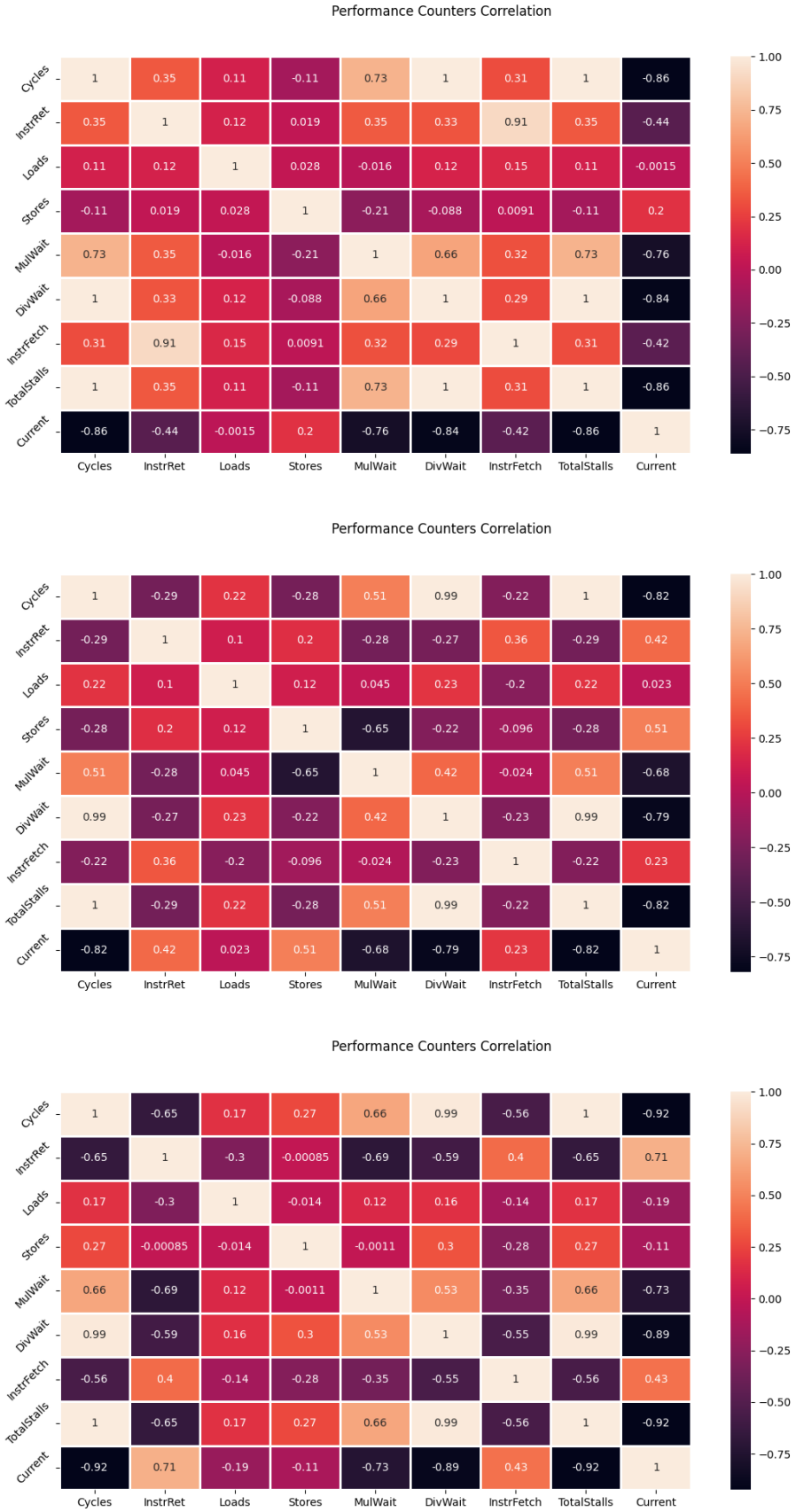
στο τέλος. Αν οι μεταβάσεις των τιμών είναι μικρές τότε η αύξηση είναι ομαλή και υπάρχουν λιγότερες πιθανότητες για μεγάλη μέγιστη τιμή.

Μία άλλη σημαντική μελέτη στα αποτελέσματα είναι αυτή της συσχέτισης των performance counters με τις μετρήσεις του ρεύματος και της κατανάλωσης ισχύος. Τα παρακάτω σχήματα δείχνουν τον συντελεστή συσχέτισης μεταξύ των μετρητών και της τιμής της κατανάλωσης ισχύος για την προσομοίωση (Σχήμα 4.7),και την συσχέτιση μεταξύ των μετρητών και της τιμής του ρευματος (Σχήμα 4.8). Οταν η τιμή είναι μεγαλύτερη ίση απο 0.7 ή μικρότερη ίση απο -0.7 τότε οι μεταβλητές έχουν ισχυρή συσχέτιση. Το πρόσημο μας πληροφορεί για το αν η αύξηση της μιάς μεταβλητής αντιστοιχεί σε αυξηση ή μείωση της άλλης. Αν το πρόσημο είναι αρνητικό τότε οι δύο μεταβλητές κινούνται με διαφορετική κατευθυνση δηλαδή όταν αυξάνεται η μία η άλλη μειώνεται. Με θετικό πρόσημο έχουν ίδια κατευθυνση.

Βλέποντας τα αποτελέσματα διακρίνεται αμέσως ότι κάποιοι μετρητες παίζουν σημαντικότερο ρόλο στην τιμή της ισχύος και του ρεύματος.Όπως διαπιστώθηκε και απο τα διαγράμματα της προηγούμενης ενότητας οι μετρητές: αριθμός των κύκλων,κύκλοι καθυστέρησης περιμένοντας να ολοκληρωθεί ο πολλαπλασιασμός ή η διαίρεση και τα συνολικά stalls είναι οι μετρητές με την μεγαλύτερη συσχέτιση. Τα συνολικά stalls μετράνε τους κύκλους αδράνειας για την πράξη του πολλαπλασιασμού και της διαίρεσης.Σαν αποτελέσμα ο αριθμός των κύκλων και ο μετρητής για τα συνολικά stalls να είναι τα πιο σημαντικά. Ακολουθούν με μέτρια συσχέτιση οι μετρητές των εντολών προσκόμισης και απόσυρσης(instrFetch,instrRet).



Σχήμα 4.7: Πίνακας Συσχέτισης ls=2.047,ls=8.188



Σχήμα 4.8: Correlation Matrix ls=2.047,ls=1.000.000,ls=10.000.000

4.4 Συσχέτιση των Αποτελεσμάτων με Μετροπρογράμματα

Για την καλύτερη εκτίμηση των αποτελεσμάτων και της σύγκρισης μεταξύ τους χρησιμοποιήθηκαν το μετροπρόγραμμα CoreMark και συγκεκριμένοι αλγόριθμοι αυτών των Bubblesort,Quicksort Mergesort και τον πολλαπλασιασμό πινάκων σαν σημείο αναφοράς για τα αποτελέσματα. Μέτα από πλήθος δοκιμών στο πλήθος των στοιχείων των πινάκων, το συμπέρασμα είναι ότι ανεξαρτήτως μεγέθους των πινάκων οι τιμές δεν αυξάνονται. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα 4.5.

Performance Counters	*CoreMark*	*BubbleSort*	*MergeSort*	*QuickSort*	*Multiplication*
Cycles	3565416	4252174	422996	182175	1390159
Instructions Retired	2750348	3499357	304116	157388	1015160
LSU Busy	194893	0	8684	0	0
Fetch Wait	187543	752817	74196	24787	124999
Loads	541082	999000	38143	25647	250000
Stores	143451	496360	28974	8300	1
Jumps	57169	999	28116	6440	0
Conditional Branches	523452	1000998	50742	35690	127550
Taken Conditional Branches	187543	751818	34100	18347	124999
Compressed Instructions	0	0	0	0	0
Multiply Wait	187920	0	0	0	250000
Divide Wait	0	0	36000	0	0
Instruction Fetch	2937891	4252174	366332	182175	1140159
Total Stalls	627525	0	44684	0	250000
Power-Current	**12468.631**	**3505**	**3647**	**3510**	**3351**

Πίνακας 4.5: Αποτελέσματα μετροπρογραμμάτων

Performance Counters	*CoreMark*	*BubbleSort*	*MergeSort*	*QuickSort*	*Multiplication*	*ls=8.188 simulation*	*ls=2047 simulation*	*ls=2047 board*	*ls=1.000.000 board*	*ls=10.000.000 board*
Cycles	3565416	4252174	422996	1821175	1390159	376692	135143	90171	50000034	520000104
Instructions Retired	0.77	0.82	0.71	0.86	0.73	0.91	0.63	0.95	0.84	0.8
LSU Busy	0.05	0	0.02	0	0	0	0	0	0	0
Fetch Wait	0.05	0.17	0.17	0.13	0.08	0.043	0.03	0.022	0.02	0.02
Loads	0.15	0.23	0.09	0.14	0.17	0	0	0.022	0.02	0.02
Stores	0.04	0.11	0.06	0.04	0.0000007	0.0000018	0.03	0.045	0.04	0.02
Jumps	0.01	0.0002	0.06	0.035	0	0.0000013	0.0000036	0.0000022	0.000000004	0.0000000003
Conditional Branches	0.14	0.23	0.11	0.19	0.09	0.0217	0.015	0.022	0.019	0.019
Taken Conditional Branches	0.05	0.17	0.08	0.1	0.08	0.0217	0.015	0.022	0.019	0.019
Compressed Instructions	0	0	0	0	0	0	0	0	0	0
Multiply Wait	0.05	0	0	0	0.17	0	0.3	0	0.05	0.11
Divide Wait	0	0	0.085	0	0	0.0217	0.01	0.02	0.07	0.05
Instruction Fetch	0.82	1	0.86	1	0.82	0.93	0.65	0.97	0.86	0.82
Total Stalls	0.17	0	0.1	0	0.17	0.043	0.333	0.02	0.13	0.17
Power-Current	**12468.631**	**3505**	**3644**	**3510**	**3351**	**24698.05**	**20258.02**	**3844**	**3762**	**3644**

Πίνακας 4.6: Αποτελέσματα Benchmark,Προσομοίωσης,FPGA

Όπως φαίνεται στον προηγούμενο πίνακα όλες οι τιμές των μετρητών έχουν γίνει σε αναλογία με τον αριθμό των κύκλων,για την καλύτερη κατανόηση τους.Επίσης τα προγράμματα είναι όλα διαφορετικά μεταξύ τους στην προσομοίωση και στα αποτελέσματα στο board. Στην τελευταία γραμμή του πίνακα οι τιμές των μετρήσεων ξεπερνάνε τις τιμές απο τα μετροπρογράμματα,και σε κάποιες περιπτώσεις κατά πολύ. Δεδομένου αυτού,τα προγράμματα των εντολών assembly απο το framework με τις συγκεκριμένες τιμές στους μετρητές είναι αποτελεσματικοί.

4.5 Αποτελέσματα Νευρωνικού δικτύου

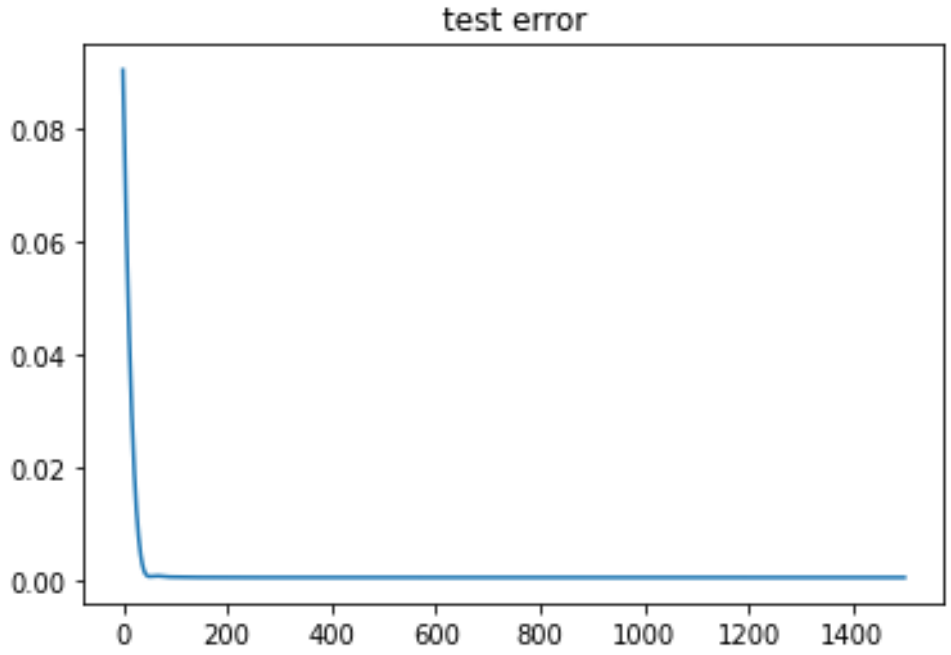
Όπως έχει ήδη αναφερθεί τα δεδομένα που αντλήθηκαν από τις μετρήσεις στο board χρησιμοποιήθηκαν σε Νευρωνικό δίκτυο το οποίο παρουσιάστηκε στο προηγούμενο κεφάλαιο.

Στη συγκεκριμένη υλοποίηση οι τιμές που χρησιμοποιήθηκαν για την εκπαίδευση του Νευρωνικού Δικτύου (μοντέλο 1) και τα αποτελέσματα που προέκυψαν δίνονται στον πίνακα 4.7. Για την εκπαίδευση χρησιμοποιήθηκαν m = 130 απο τα συνολικά 423 δεδομένα. Επιπλέον, γίνεται μία σύγκριση με τα αποτελέσματα του μοντέλου που παρουσιάζεται στο paper[19] (μοντέλο 2) και συγκεκριμένα με τα αποτελέσματα του *TABLE II: Different model configurations*, από τον οποίο χρησιμοποιούνται για σύγκριση οι παράμετροι που δίνουν καλύτερη απόδοση. Τα δύο μοντέλα είναι παρόμοιας αρχιτεκτονικής, αλλά υπάρχουν κάποιες μικρές διαφορές κυρίως του dataset εκπαίδευσης. Για το (2) χρησιμοποιήθηκε η βιβλιοθήκη Keras και η Tensorflow ενώ για το μοντέλο (1) χρησιμοποιήθηκε η Pytorch.

Μοντέλο	learning rate η	Χαρακτηριστικά δείγματος	Κρυμμένα επίπεδα	Νευρώνες	Test set(%)	Σφάλμα
Μοντέλο(1)	0.0004	d=14	2	H=50	30	**0.001305**
Μοντέλο(2)	0.0004	d=26	2	H=50	5	**0.042244**

Πίνακας 4.7: Αποτελέσματα Νευρωνικού Δικτύου

Βλέπουμε στον πίνακα 4.7 ότι η γενίκευση του μοντέλου (1) είναι καλύτερη και το σφάλμα είναι χαμηλότερο στο test set, παρόλο που για την εκπαίδευση του μοντέλου(2) χρησιμοποιήθηκαν 127.3k δεδομένα για εκπαίδευση και 6.7k δεδομένα ως test set. Στο σχήμα 4.9 φαίνεται το σφάλμα του νευρωνικού δικτύου του μοντέλου(1) που είναι σχεδόν μηδενικό.



Σχήμα 4.9: Σφάλμα του Νευρωνικού Δικτύου στο Test Set

Μια τελευταία δοκιμή που έγινε στο νευρωνικό δίκτυο,ότι δόθηκαν ως είσοδος οι μετρητές απο τα διάφορα μετροπρογράμματα που αναφέρθηκαν στη προηγούμενη ενότητα. Στο παρακάτω πίνακα 4.8 φαίνονται οι τιμές που προέβλεψε το νευρωνικό αλλα και οι τιμές που είχαν τα αποτελέσματα των μετρήσεων .

	BubbleSort	MergeSort	QuickSort	Multiplication
Πρόβλεψη Μοντέλου(1)	3444	3578	3462	3290
Μέτρηση	3505	3644	3510	3351

Πίνακας 4.8: Πρόβλεψη Νευρωνικού για τις τιμές των Προγραμμάτων Benchmark

Κεφαλαιο 5

Συμπερασματα

5.1 Συμπεράσματα

5.2 Μελλοντικές Επεκτάσεις

5.1 Συμπεράσματα

Η εργασία αυτή περιλαμβάνει την μελέτη και την υλοποίηση μηχανισμών για την ανάδειξη συμπερασμάτων σχετικά με την συσχέτιση των ενσωματωμένων μετρητών επίδοσης και την κατανάλωση ισχύος και ρεύματος ενός επεξεργαστή. Παρατηρήθηκε ότι τόσο σε επίπεδο προσομοίωσης όσο και σε πραγματικές μετρήσεις σε FPGA το εργαλείο που χρησιμοποιήθηκε και τροποποιήθηκε για την παραγωγή θερμικών ιών σε αρχιτεκτονική RISC-V είχε τα επιθυμητά αποτελέσματα. Τα προγράμματα που δημιούργησε αύξησαν την κατανάλωση ισχύος και ρεύματος σε επίπεδα πολύ μεγαλύτερα από τις τιμές των μετροπρογραμμάτων που χρησιμοποιήθηκαν σαν σημείο αναφοράς.

Οι τιμές των performance counters δείχνουν ότι έχουν άμεση συσχέτιση με τις τιμές της κατανάλωσης και μπορούν να αποτελέσουν μέθοδο αποφυγής κακόβουλων θερμικών ιών,χωρίς την ανάγκη ενσωματωμένων αισθητήρων. Ειδικότερα αρκεί ένα μόνο σύνολο από τους μετρητές για να μπορέσει να υλοποιηθεί η μέθοδος. Οι μετρητές των κύκλων εκτέλεσης ,των συνολικών καθυστερήσεων,των εντολών προσκόμισης και απόσυρσης είναι αρκετοί για την αποφυγή θερμικών ιών. Τέλος,ένα απλό νευρωνικό δίκτυο με είσοδο τους μετρητές επίδοσης είναι αρκετό για να προβλέπει την μέση κατανάλωση ρεύματος με μεγάλη ακρίβεια.

5.2 Μελλοντικές Επεκτάσεις

Μία πιθανή επέκταση θα ήταν αυτή των επιπλέον performance counters.Ο επεξεργαστής μετά και την προσθήκη των δύο μετρητών στα πλαίσια της εργασίας διαθέτει συνολικά δεκατέσσερις μετρητές. Θα μπορούσαν να προστεθούν και άλλοι μετρητές έτσι ώστε να δημιουργεί μεγαλύτερη συσχέτιση με τις μετρήσεις και να αντληθούν περισσότερα δεδομένα και καλύτερα για το νευρωνικό δίκτυο ώστε να κάνει καλύτερες αποτιμήσεις.

Ακόμα,θα μπορούσε να αποτελέσει προσθήκη ένας μηχανισμός έτσι ώστε όταν γίνεται ανίχνευση προγράμματος,με τιμή ρεύματος που παράγεται μεγαλύτερη από ένα όριο να προσθέτει εντολές nop ,με αποτέλεσμα να προκαλούνται καθυστερήσεις και να πέφτει ο φόρτος εργασίας και άρα και η τιμή της ισχύος και του ρεύματος. Ο Ibex διαθέτει ήδη έναν τέτοιο μηχανισμό που προσθέτει dummy εντολές που δεν έχουν επιπτώσεις στην λειτουργία του επεξεργαστή. Προσθέτουν αναδιοργάνωση στις εντολές που εκτελούνται ώστε να μην μπορούν να γίνουν προβλέψεις από κάποιον που κάνει επιθέσεις timing fault injection. Μία άλλη πιθανή επέκταση είναι να πραγματοποιηθεί παρόμοια υλοποίηση σε κάποιον άλλον επεξεργαστή είτε της ίδιας αρχιτεκτονικής είτε διαφορετικής ώστε να γίνει μελέτη και συγκριση αποτελεσμάτων.

Βιβλιογραφια

[1] Risc-v: The free and open risc instruction set architecture. [Online]. Available: https://riscv.org/

[2] A. Waterman, K. Asanovic, and SiFive Inc, "The risc-v instruction set manual volume i: Unprivileged isa," CS Division, EECS Department, University of California, Berkeley, Tech. Rep. Document Version 20191213, 2019. [Online]. Available: https://riscv.org/technical/specifications/

[3] A. Waterman, K. Asanovic, J. Hauser, and SiFive Inc, "The risc-v instruction set manual volume ii: Privileged architecture," CS Division, EECS Department, University of California, Berkeley, Tech. Rep. Document Version 20211203, 2021. [Online]. Available: https://riscv.org/technical/specifications/

[4] Ibex: An embedded 32 bit risc-v cpu core. [Online]. Available: https://ibex-core.readthedocs.io/en/latest/

[5] Github: Ibex risc-v core. [Online]. Available: https://github.com/lowRISC/ibex

[6] lowrisc. [Online]. Available: https://lowrisc.org/

[7] Fusesoc user guide. [Online]. Available: https://fusesoc.readthedocs.io/en/stable/user/

[8] Github: Fusesoc. [Online]. Available: https://github.com/olofk/fusesoc

[9] Verilator: the fastest verilog/systemverilog simulator. [Online]. Available: https://www.veripool.org/verilator/

[10] Verilator user's guide. [Online]. Available: https://verilator.org/guide/latest/index.html

[11] V. Tenentes, S. Das, D. Rossi, and B. M. Al-Hashimi, "Run-time protection of multi-core processors from power-noise denial-of-service attacks," *IEEE Transactions on Device and Materials Reliability*, vol. 20, no. 2, pp. 319–328, 2020.

[12] V. Tenentes, S. Das, D. Rossi, and B. M. Al-Hashimi, "Run-time detection and mitigation of power-noise viruses," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 275–280.

[13] Z. Hadjilambrou, S. Das, P. N. Whatmough, D. Bull, and Y. Sazeides, "Gest: An automatic framework for generating cpu stress-tests," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 1–10.

[14] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018.

[15] Arty a7 reference manual. [Online]. Available: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual

[16] 7series xadc. [Online]. Available: https://docs.xilinx.com/r/en-US/ug480_7Series_XADC

[17] Github: A very simple uart implementation, written in verilog. [Online]. Available: https://github.com/ben-marshall/uart

[18] pyserial's documentation. [Online]. Available: https://pyserial.readthedocs.io/en/latest/

[19] N. Mammeri, M. Neu, S. Lal, and B. Juurlink, "Performance counters based power modeling of mobile gpus using deep learning," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, 2019, pp. 193–200.

Παραρτημα Α

Σχηματα με Προγραμματα της εργασιας

```
  .data
  .text
  .align 2
  .global main_asm
  .type main_asm, @function

main_asm:

    mv x6,x0
    addi x7,x0, 1
    addi x8,x0,-1
    lui x9, 0x55555
    addi x9, x9, 0x555
    lui x10, 0x33333
    addi x10, x10, 0x333

    mv x11,x0
    mv x12,x6
    mv x13,x7
    mv x14,x8
    mv x15,x9
    mv x16,x10
    mv x17,x10
    mv x18,x9
    mv x19,x8
    mv x20,x7
    mv x21,x6
    mv x22,x0
    mv x23,x10
    mv x24,x9
    mv x25,x8
    mv x26,x7
```

```
    mv x27,x6
    mv x28,x0
    mv x29,x8
    mv x30,x9
    mv x31,x0

    lui x31, 0x000f4
    addi x31,x0,2047

Start:

    #loop_code

    addi  x31,x31,-1
    bnez x31,Start
ret
```

Πρόγραμμα Α.1: Κώδικας main_original που απεικονίζει το κενό βρόγχο

```
#include <stdio.h>

extern void main_asm(void);

int main(int argc, char **argv) {

  unsigned int inhibit_val2 = 0xFFFFFFFF;
  asm volatile("csrw  0x320, %0\n" : : "r"(inhibit_val2));

   asm volatile(
      "csrw minstret,       x0\n"
      "csrw mcycle,         x0\n"
      "csrw mhpmcounter3,   x0\n"
      "csrw mhpmcounter4,   x0\n"
      "csrw mhpmcounter5,   x0\n"
      "csrw mhpmcounter6,   x0\n"
      "csrw mhpmcounter7,   x0\n"
      "csrw mhpmcounter8,   x0\n"
      "csrw mhpmcounter9,   x0\n"
      "csrw mhpmcounter10,  x0\n"
      "csrw mhpmcounter11,  x0\n"
      "csrw mhpmcounter12,  x0\n"
      "csrw mhpmcounter13,  x0\n"
      "csrw mhpmcounter14,  x0\n"
      "csrw mhpmcounter15,  x0\n"
      "csrw mhpmcounter16,  x0\n"
      "csrw mhpmcounter17,  x0\n"
      "csrw mhpmcounter18,  x0\n"
      "csrw mhpmcounter19,  x0\n"
      "csrw mhpmcounter20,  x0\n"
      "csrw mhpmcounter21,  x0\n"
      "csrw mhpmcounter22,  x0\n"
      "csrw mhpmcounter23,  x0\n"
```

```
34        "csrw mhpmcounter24,  x0\n"
35        "csrw mhpmcounter25,  x0\n"
36        "csrw mhpmcounter26,  x0\n"
37        "csrw mhpmcounter27,  x0\n"
38        "csrw mhpmcounter28,  x0\n"
39        "csrw mhpmcounter29,  x0\n"
40        "csrw mhpmcounter30,  x0\n"
41        "csrw mhpmcounter31,  x0\n"
42        "csrw minstreth,      x0\n"
43        "csrw mcycleh,        x0\n"
44        "csrw mhpmcounter3h,  x0\n"
45        "csrw mhpmcounter4h,  x0\n"
46        "csrw mhpmcounter5h,  x0\n"
47        "csrw mhpmcounter6h,  x0\n"
48        "csrw mhpmcounter7h,  x0\n"
49        "csrw mhpmcounter8h,  x0\n"
50        "csrw mhpmcounter9h,  x0\n"
51        "csrw mhpmcounter10h, x0\n"
52        "csrw mhpmcounter11h, x0\n"
53        "csrw mhpmcounter12h, x0\n"
54        "csrw mhpmcounter13h, x0\n"
55        "csrw mhpmcounter14h, x0\n"
56        "csrw mhpmcounter15h, x0\n"
57        "csrw mhpmcounter16h, x0\n"
58        "csrw mhpmcounter17h, x0\n"
59        "csrw mhpmcounter18h, x0\n"
60        "csrw mhpmcounter19h, x0\n"
61        "csrw mhpmcounter20h, x0\n"
62        "csrw mhpmcounter21h, x0\n"
63        "csrw mhpmcounter22h, x0\n"
64        "csrw mhpmcounter23h, x0\n"
65        "csrw mhpmcounter24h, x0\n"
66        "csrw mhpmcounter25h, x0\n"
67        "csrw mhpmcounter26h, x0\n"
68        "csrw mhpmcounter27h, x0\n"
69        "csrw mhpmcounter28h, x0\n"
70        "csrw mhpmcounter29h, x0\n"
71        "csrw mhpmcounter30h, x0\n"
72        "csrw mhpmcounter31h, x0\n");
73
74
75    unsigned int inhibit_val1 = 0x0;
76    asm volatile("csrw  0x320, %0\n" : : "r"(inhibit_val1));
77
78    main_asm();
79
80    unsigned int inhibit_val3 = 0xFFFFFFFF;
81    asm volatile("csrw  0x320, %0\n" : : "r"(inhibit_val3));
82    uint32_t perfC_value;
83
84    volatile uint32_t *performance_counters_memory = (volatile uint32_t *) 0x00000AF0;
```

```
85    __asm__ volatile("csrr %0,  mcycle ;" : "=r"(perfC_value));
86    *performance_counters_memory = perfC_value;
87    performance_counters_memory+=1;
88    __asm__ volatile("csrr %0,  minstret;" : "=r"(perfC_value));
89    *performance_counters_memory = perfC_value;
90    performance_counters_memory+=1;
91    __asm__ volatile("csrr %0,  mhpmcounter3 ;" : "=r"(perfC_value));//lsu busy
92    *performance_counters_memory = perfC_value;
93    performance_counters_memory+=1;
94    __asm__ volatile("csrr %0,  mhpmcounter4;" : "=r"(perfC_value));//fetch wait
95    *performance_counters_memory = perfC_value;
96    performance_counters_memory+=1;
97    __asm__ volatile("csrr %0,  mhpmcounter5 ;" : "=r"(perfC_value));//loads
98    *performance_counters_memory = perfC_value;
99    performance_counters_memory+=1;
100   __asm__ volatile("csrr %0,  mhpmcounter6;" : "=r"(perfC_value));//stores
101   *performance_counters_memory = perfC_value;
102   performance_counters_memory+=1;
103   __asm__ volatile("csrr %0,  mhpmcounter7 ;" : "=r"(perfC_value));//jumps
104   *performance_counters_memory = perfC_value;
105   performance_counters_memory+=1;
106   __asm__ volatile("csrr %0,  mhpmcounter8;" : "=r"(perfC_value));//cond branch
107   *performance_counters_memory = perfC_value;
108   performance_counters_memory+=1;
109   __asm__ volatile("csrr %0,  mhpmcounter9 ;" : "=r"(perfC_value));//t cond branch
110   *performance_counters_memory = perfC_value;
111   performance_counters_memory+=1;
112   __asm__ volatile("csrr %0,  mhpmcounter10;" : "=r"(perfC_value));//compr instr
113   *performance_counters_memory = perfC_value;
114   performance_counters_memory+=1;
115   __asm__ volatile("csrr %0,  mhpmcounter11 ;" : "=r"(perfC_value));//mul wait
116   *performance_counters_memory = perfC_value;
117   performance_counters_memory+=1;
118   __asm__ volatile("csrr %0,  mhpmcounter12;" : "=r"(perfC_value));//div wait
119   *performance_counters_memory = perfC_value;
120   performance_counters_memory+=1;
121    __asm__ volatile("csrr %0,  mhpmcounter13 ;" : "=r"(perfC_value));//instr_fetch
122   *performance_counters_memory = perfC_value;
123   performance_counters_memory+=1;
124   __asm__ volatile("csrr %0,  mhpmcounter14;" : "=r"(perfC_value));//total stalls
125   *performance_counters_memory = perfC_value;
126   performance_counters_memory+=1;
127
128    asm volatile("wfi");
129
130   return 0;
131 }
```

Πρόγραμμα Α.2: Κώδικας γλώσσας C όπου φορτώνεται στο fpga

```
1
2
```

```
class MeasurementHPerfCounters_Vivado(Measurement):
    countTimes=0
    def __init__(self,confFile):
        super().__init__(confFile)

    def init(self):
        super().init()
        self.timeToMeasure = self.tryGetIntValue('time_to_measure')

    def measure(self):

        process_ibex=subprocess.run('/home/kympa/gk_myGA/makeMeasurement_PerfCounters_Vivado.sh')
        print(process_ibex.returncode)
        if int(process_ibex.returncode)==0:
            print("pass")
        else:
            print("dont pass")

        MeasurementHPerfCounters_Vivado.countTimes+=1
        from_dir = r"/home/kympa/ibex/ibex_system_log_dec.txt"
        new_name =str(MeasurementHPerfCounters_Vivado.countTimes)+"_ibex_system_log_dec_"+".txt"
        target_dir =r"/home/kympa/results_sim/"+new_name
        shutil.copyfile(from_dir,target_dir)
        from_dir = r"/home/kympa/ibex/build/lowrisc_ibex_top_artya7_0.1/synth-vivado/
    lowrisc_ibex_top_artya7_0.1.runs/impl_1/post_implementation_power_result.log"
        new_name =MeasurementHPerfCounters_Vivado.countTimes+"post_implementation_power_result_"+".
    txt"
        target_dir =r"/home/kympa/results_sim/"+new_name
        shutil.copyfile(from_dir,target_dir)
        infile = r"/home/kympa/ibex/build/lowrisc_ibex_top_artya7_0.1/synth-vivado/
    lowrisc_ibex_top_artya7_0.1.runs/impl_1/post_implementation_power_result.log"
        keep_phrases = []
        regex_num = ("\d+(\.\d+)?$")
        regex_word = ("\b|\s*u_top  ")

        with open(infile) as f:
            f = f.readlines()

        power =[]
        for line in f:
            for phrase in keep_phrases:
                if phrase in line:
                    for word in line.split():
                        if re.match(regex_num, word):
                            data = float(word)
                            power.append(data)
            if re.findall(regex_word, line):
                for word in line.split():
                    if re.match(regex_num, word):
                        data = float(word)
                        power.append(data)
```

```
51
52          return power;
53
54
55  ---------------------------------------------------------------
56
57
58  #!/bin/bash
59
60  echo " building verilator "
61  cd /home/gk/Desktop
62  pwd
63  unset VERILATOR_ROOT
64  cd verilator
65  autoconf
66  ./configure
67  make -j `nproc`
68  make install
69
70  cp /home/kympa/gk_myGA/assembly_compilation_RISCV/main.s /home/kympa/ibex/examples/sw/simple_system
        /GK_testing_ibex/testingCounters/main_asm.S
71  cd /home/kympa/ibex
72
73
74  fusesoc --cores-root=. run --target=sim --setup --build lowrisc:ibex:ibex_simple_system --RV32E=0 \
75                    --RV32M=ibex_pkg::RV32MFast  --WritebackStage=1
76
77  make -C examples/sw/simple_system/GK_testing_ibex/testingCounters
78
79  ./build/lowrisc_ibex_ibex_simple_system_0/sim-verilator/Vibex_simple_system -t \
80                  --meminit=ram,./examples/sw/simple_system/GK_testing_ibex/testingCounters/
        testCounters.elf
81
82  cd /home/kympa
83  python3 parserFile.py
84  cd /home/kympa/ibex/build/lowrisc_ibex_top_artya7_0.1/synth-vivado/lowrisc_ibex_top_artya7_0.1.runs
        /impl_1
85  vivado -mode batch -source ../../vivado_hook_write_bitstream_pre.tcl
```

Πρόγραμμα A.3: Κώδικας της κλάσης MeasurementHPerfCounters-Vivado - makeMeasurement-PerfCounters-Vivado script