Mapping the spectral algorithm to reconfigurable logic using DAE and memory HMC

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΤΡΑΚΗΣ ΠΑΥΛΟΣ

Μέλη Επιτροπής:

Καθηγητής Καλαϊτζάκης Κωνσταντίνος (Επιβλέπων)
Καθηγητής Πνευματικάτος Διονύσιος
Αν. Καθηγητής Κουτρούλης Ευτύχιος

Χανιά 2020

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Καλαϊτζάκη που μου έδωσε την ευκαιρία να συνεργαστώ μαζί του και να ασχοληθώ με νέα και ενδιαφέροντα πράγματα που συνάντησα στην παρούσα διπλωματική. Επίσης, θα ήθελα να ευχαριστήσω τα μέλη της επιτροπής .

Στη συνέχεια, θα ήθελα να ευχαριστήσω ιδιαίτερα τον κ. Βατσολάκη όπου με βοήθησε με τις
κατευθύνσεις που μου έδινε σε όλη τη διάρκεια της διπλωματικής και ήταν παρών σε κάθε απορία και πρόβλημα που αντιμετώπισα.

Ακόμα, δεν γίνεται να παραλείψω την οικογένεια μου, που έδειξαν ιδιαίτερη υπομονή και με βοήθησαν ψυχολογικά σε κάθε δυσκολία που αντιμετώπισα όλα αυτά τα χρόνια και τους ευχαριστώ πραγματικά για όλα.

Τέλος, θέλω να ευχαριστήσω τους φίλους μου για την στήριξη τους όλα αυτά τα χρόνια σε κάθε μου προσπάθεια.

Περίληψη

Τα τελευταία χρόνια η ανάγκη επεξεργασίας μεγάλου όγκου δεδομένων σε μικρό χρονικό διάστημα, έστρεψε το ενδιαφέρον στη δημιουργία προγραμμάτων όπου συνδυάζουν το software και το hardware με σκοπό την εκμετάλλευση των πλεονεκτημάτων που παρέχει το καθένα. Η ανάγκη αυτή οδήγησε το Phil Colela στην έμπνευση επτά αλγοριθμικών μεθόδων με μεγάλη φορητότητα, οι οποίες χρησιμοποιήθηκαν ως benchmarks σε διάφορες πλατφόρμες εκμεταλλευόμενες τα πλεονεκτήματα του παράλληλου προγραμματισμού. Στη συνέχεια οι μέθοδοι αυτοί επεκτάθηκαν σε δεκατρείς από ομάδα ερευνητών του Berkeley.

Παράλληλα, τα τελευταία χρόνια απλουστεύτηκε η απεικόνιση ενός αλγορίθμου στο hardware με τη βοήθεια του εργαλείου Vivado High Level Synthesis. Οι διαδικασίες έγιναν πιο αυτοματοποιημένες και η δημιουργία του RTL αρχείου αρκετά πιο εύκολη για το προγραμματιστή.

Ο στόχος λοιπόν αυτής της διπλωματικής, είναι η απεικόνιση ενός νάνου, συγκεκριμένα του αλγόριθμου Spectral, σε hardware με την βοήθεια της πλατφόρμας vivado HLS, στην συνέχεια με βάση την αρχιτεκτονική Decoupled Access/Execute προσπαθήσαμε να βελτιστοποιήσουμε τον συγκεκριμένο αλγόριθμο. Το συγκεκριμένο framework μετατρέπει το αρχικό instruction stream σε δυο μονάδες την fetch που είναι υπεύθυνη για την ανάκτηση των δεδομένων από την μνήμη και την process που είναι με την σειρά της υπεύθυνη για την επεξεργασία των δεδομένων. Για τον αλγόριθμο Spectal πραγματοποιήθηκαν τέσσερεις διαφορετικές υλοποιήσεις στο εργαλείο της Vivado HLS και μια υλοποίηση στο Hybrid Memory Cybe που μας παρέχει το Πολυτεχνείο Κρήτης μέσω του εργαλείου της vivado. Τέλος, πραγματοποιήθηκε σύγκριση στην απόδοση κάθε υλοποίησης με την αρχική βελτιστοποιημένη υλοποίηση σε software. Τα αποτελέσματα που εξάγαμε ήταν αρκετά ενθαρρυντικά δηλαδή περίπου 2 φορές αυξημένη απόδοση όσον αναφορά τη χρήση της αρχιτεκτονικής DAE και περίπου 4 φορές αυξημένη απόδοση όσο αναφορά της χρήση της πλατφόρμας της micron, HMC.

Abstract

In the latest years, the need to process large volumes of data in a short time period has shifted the interest in creating programs that combine software and hardware. This need led Phil Colela to the inspiration of seven algorithmic methods with great portability on various platforms that were used as benchmarks, exploiting the advantages of parallel programming. These methods were extended to thirteen by a Berkeley group of researchers.
Simultaneously, in the past few years, the visualization of an algorithm in hardware has been simplified with the help of the Vivado High Level Synthesis tool. As a result, procedures have become more automated and the creation of the RTL file has become easier for the developer, as well.

The aim of this Diploma Thesis is implement the Spectral algorithm in hardware according to the DAE architecture and for the optimization system performance. This algorithm fall into the 13 dwarfs a . The stages of converting an algorithm based on the above framework are simple and specific. For the Spectral algorithm we created four different implementations which were carried out on the Vivado HLS tool, and another one implementation in the HMC platform which Technical University of Crete provide us. Finally, both the times of the second and the third implementation are stated and compared with the initial optimized implementation in software.

Περιεχόμενα

Λίστα εικόνων

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή στους δεκατρείς νάνους

Αρχικά ο Phil Colela προσδιόρισε επτά αλγοριθμικές μεθόδους τις οποίες και ονόμασε νάνους. Πρωταρχικός του στόχος ήταν η ένταξη όλων των κοινών αλγορίθμων σε ένα μικρό σύνολο μεθόδων. Ισχυρίστηκε, ότι κάθε κατηγορία εμφανίζει κοινά πρότυπα υπολογισμού και επικοινωνίας. Στην ουσία κάθε κατηγορία νάνων παρουσιάζει οικογένειες αλγορίθμων, όπου εμφανίζουν όμοιες υπολογιστικές ιδιότητες. Οι παραπάνω αλγοριθμικές μέθοδοι δέχθηκαν και θα δεχθούν αρκετές αλλαγές με το πέρασμα των χρόνων, ωστόσο τα πρότυπα υπολογισμού παρέμειναν και θα παραμείνουν τα ίδια παίζοντας σπουδαίο ρόλο στο εγγύς μέλλον για την βελτίωση εφαρμογών σε διάφορους τομείς της επιστήμης και της μηχανικής. Ένα άλλο χαρακτηριστικό τους είναι ότι διακρίνονται για τα υψηλά επίπεδα αφαίρεσης, ώστε να βρίσκουν εφαρμογή σε ένα ευρύ φάσμα εφαρμογών. Εξαιτίας της μεγάλης αποδοτικότητας που τους διέπουν οι μέθοδοι αυτοί είναι ικανοί να εκτελεστούν οπουδήποτε, π.χ. CPU, GPU ακόμα και στις FPGAs κάτι το οποίο είναι αρκετά σημαντικό διότι γνωρίζουμε ότι είναι αρκετά επεκτάσιμες και είναι ιδιαίτερα χαμηλού κόστους.

Στη συνέχεια, μια ομάδα ερευνητών από το Berkeley επέκτεινε τους νάνους σε δεκατρείς εξετάζοντας και εφαρμόζοντας τους σε διάφορους τομείς. Απέδειξαν ότι οι νάνοι έχουν αρκετά καλή απόδοση, μειωμένη κατανάλωση ενέργειας ιδιαίτερα σε πολυπύρηνους επεξεργαστές με αποτέλεσμα να δημιουργηθούν νέα κριτήρια για το σχεδιασμό και την αξιολόγηση παράλληλων μοντέλων προγραμματισμού. Οι ερευνητές ισχυρίζονται ότι θα διατηρήσουν τις καλές αποδόσεις τους και σε μελλοντικές πλατφόρμες.

Τέλος οι νάνοι συνεισφέρουν στην διευκόλυνση της ζωής του προγραμματιστή καθιστώντας την υλοποίηση παράλληλων προγραμμάτων αρκετά πιο εύκολη. Εξαιτίας τις κατηγοριοποίησης των αλγορίθμων, ο προγραμματιστής μπορεί να επιλέξει εκ των προτέρων την κατάλληλη αλγοριθμική μέθοδο για την υλοποίηση του προγράμματος που επιθυμεί. Έτσι και η σχεδίαση σε παράλληλου προγραμματισμού γίνεται πιο εύκολη και πιο αποδοτική εφόσον μπορούμε να αποφασίσουμε εκ των προτέρων πότε θα χρησιμοποιήσουμε GPU και άλλους επιταχυντές και πότε CPU.

1.2 Εισαγωγή στην αρχιτεκτονική DAE

Στις μέρες μας στους περισσότερους μικροεπεξεργαστές υψηλών αποδόσεων χρησιμοποιείται δυναμικός προγραμματισμός. Ο επεξεργαστής δηλαδή εκτελεί τις εντολές ανάλογα με το ποιες είναι έτοιμες για εκτέλεση έτσι ώστε να εξαλείφονται τυχόν καθυστερήσεις στο πρόγραμμα. Η αρχιτεκτονική DAE είναι μια εκ των αρχιτεκτονικών που χρησιμοποιεί δυναμικό προγραμματισμό.

Η ιδέα της αρχιτεκτονικής Decoupled access/execute γεννήθηκε το 1982 από τον James E. Smith. Στην αρχιτεκτονική αυτή η ανάκτηση δεδομένων από την μνήμη απομονώνεται από την επεξεργασία των δεδομένων. Το αρχικό instruction stream χωρίζεται σε δυο μονάδες την fetch που είναι υπεύθυνη για την ανάκτηση των δεδομένων από την μνήμη και την process που είναι με την σειρά της υπεύθυνη για την επεξεργασία των δεδομένων οι δυο μονάδες επικοινωνούν μεταξύ του με buffer.

Με την αρχιτεκτονική αυτή επιτυγχάνονται υψηλά επίπεδα παραλληλισμού βελτίωση της απόδοσης του προγράμματος καθώς οι αστοχίες της μνήμης μειώνονται. Επίσης έχουμε βελτίωση της ενεργητικής αποδοτικότητας του συστήματος.

Θα αναφερθούμε αναλυτικότερα στην συγκεκριμένη αρχιτεκτονική στο κεφάλαιο 4.

1.3 Συνδυασμός DAE με τους δεκατρείς νάνους

Παραπάνω αναφέρθηκαν ο σκοπός της δημιουργίας της αρχιτεκτονικής DAE και η δημιουργία των δεκατριών αλγοριθμικών μεθόδων που ονομάστηκαν νάνοι. Αυτά τα δύο παρέχουν τη δυνατότητα της δημιουργίας ενός προγράμματος που διακρίνεται για την ευελιξία του και για την αποδοτικότητα του. Ακόμα, καθιστούν εύκολη τη ζωή του προγραμματιστή εφόσον πραγματοποιεί τετριμμένα βήματα για την δημιουργία της εφαρμογής που επιθυμεί με πρότυπα παράλληλου προγραμματισμού. Μπορούμε να αναλογιστούμε ότι η υλοποίηση μιας εκ των δεκατριών αλγοριθμικών μεθόδων στα πρότυπα της αρχιτεκτονική DAE θα εκτοξεύσει την απόδοση του συστήματος.

Εάν εξετάσουμε τους νάνους θα δούμε ότι οι αλγόριθμοι τους περιέχουν προσπελάσεις στη μνήμη και ανάκτηση μεγάλου όγκου δεδομένων. Με το διαχωρισμό του instruction stream σε δύο λειτουργικές μονάδες όπου η πρώτη είναι αρμόδια για την ανάκτηση των διευθύνσεων και η δεύτερη για την επεξεργασία των αποτελεσμάτων καταλαβαίνουμε ότι η αρχιτεκτονική DAE βελτιώνει την απόδοση των παραπάνω αλγοριθμικών μεθόδων εφόσον δεν χρειάζεται να έχει εκτελεστεί μία εντολή για να εκτελεστεί η επόμενη.

Στην συγκεκριμένη διπλωματική εργασία θα εξετάσουμε τον συνδυασμό της

αρχιτεκτονικής DAE με το spectral method που όπως θα δούμε παρακάτω είναι αλγόριθμος που ανήκει στους δεκατρείς νάνους.

1.4 Συνεισφορά διπλωματικής εργασίας

Συνοπτικά στην συγκεκριμένη διπλωματική υλοποιήσαμε τον spectral αλγόριθμο που υπάγεται στους δεκατρείς νάνους μέσω του εργαλείου Vivado High Level Synthesis (HLS) χρησιμοποιώντας την αρχιτεκτονική DAE για την βελτιστοποίηση της. Το εργαλείο της vivado παρέχει αυτοματοποιημένους επιταχυντές και dataflow directives όπου βελτίωσαν την απόδοση του προγράμματος. Πραγματοποιήσαμε 4 υλοποιήσεις που αφορούν το εργαλείο Vivado HLS και μια υλοποίηση στο Hybrid Memory Cube (HMC).

Η συνεισφορά της διπλωματικής συνοψίζεται στα παρακάτω:

- Απεικόνιση του αλγορίθμου spectral με τη χρήση το εργαλείου Vivado σε αναδιατασσόμενη λογική. Το εργαλείο αυτό παρέχει αυτοματοποιημένους τρόπους απεικόνισης εφαρμογών σε αναδιατασσόμενη λογική και μας βοηθάει να μετατρέψουμε πολύ εύκολα εφαρμογές από γλώσσα υψηλού προγραμματισμού όπως η C/C++ .
- Βελτιστοποίηση της αρχιτεκτονικής της παραπάνω εφαρμογής που πραγματοποιήθηκε στο Vivado HLS με τη χρήση του framework DAE(DAER) που αναλύεται στο κεφάλαιο 5. Το συγκεκριμένο framework βελτιστοποιεί την απόδοση διαφόρων εφαρμογών προσφέροντας streaming επεξεργασία και καλύπτοντας διάφορες αλγοριθμικές δυσκολίες όπως οι εξαρτήσεις δεδομένων κατά την επεξεργασία.
- Μέτρηση απόδοσης της απεικόνισης της παραπάνω εφαρμογής στην πλατφόρμα του HMC
- Εξαγωγή συμπερασμάτων όσο αναφορά την αρχιτεκτονική DAE αλλά και την πλατφόρμα HMC.
- Μελλοντικές προεκτάσεις της συγκεκριμένης διπλωματικής εργασίας

Κεφάλαιο 2

Παρόμοιες υλοποιήσεις

Επισκόπηση

Στο συγκεκριμένο κεφάλαιο αναφέρονται παρόμοιες υλοποιήσεις απόζευξης με τη decoupled access/execute που σκοπό είχαν την βελτίωση της παραπάνω αρχιτεκτονικής με κύριο στόχο την εξάλειψη των εξαρτήσεων των δεδομένων.

2.1 Υλοποιήσεις της χρήση DAE για βελτίωση της απόδοσης

Η αρχιτεκτονική απόζευξης δημιουργήθηκε με σκοπό την βελτίωση της απόδοσης των προγραμμάτων. Το μεγαλύτερο της πλεονέκτημα είναι ότι παρέχει τη δυνατότητα απόκρυψης καθυστέρησης της μνήμης με την βοήθεια της ιδιότητας του prefetching. Το κέρδος φυσικά είναι τα δεδομένα έρχονται την κατάλληλη στιγμή για επεξεργασία και αποφεύγονται τυχών αστοχίες της μνήμης. Ωστόσο, τελευταία χρόνια το ενδιαφέρον των αρχιτεκτονικών απόζευξης έχει συρρικνωθεί και αυτό γιατί ορισμένες φορές δεν είναι κατάλληλες σε γενικές εφαρμογές διότι δεν είναι δομημένες.

Παράλληλα βλέπουμε οι εφαρμογές των πολυμέσων να έχουν κυρίαρχο φόρτο εργασίας στους υπολογιστές. Σύγχρονες εφαρμογές όπως συμπίεση ήχου και βίντεο ,επεξεργασία εικόνας, αναγνώριση ομιλίας και φωνής είναι αρκετά δομημένες και προσφέρονται για αποσύνδεση. Έτσι ακολουθώντας την αρχιτεκτονική DAE δημιουργείται μία παρόμοια αρχιτεκτονική η οποία ονομάζεται MediaBreeze όπου και αυτή με τη σειρά της αποσυνδέει την κύρια εκτέλεση του προγράμματος σε δύο μέρη. Το πρώτο περιέχει χρήσιμους υπολογισμούς όπως απαιτείται από τον αλγόριθμο. Το δεύτερο μέρος αποτελεί την υποστηρικτική μονάδα και περιέχει λειτουργίες όπως δημιουργία και μετασχηματισμούς διευθύνσεων, αποθήκευση, φόρτωση και βρόγχους επανάληψης. Ο κύριος σκοπός της είναι να βοηθήσει στην εκτέλεση των χρήσιμων εντολών. Έτσι, εκμεταλλεύεται τις βασικές έννοιες της αρχιτεκτονικής DAE για την βελτίωση των προγραμμάτων. Αποτελέσματα έχουν δείξει ότι η βελτίωση της απόδοσης κυμαίνεται από 1,05% έως 16% περισσότερο.

Δεδομένου ότι άλλες αρχιτεκτονικές δεν προσφέρουν αποδοτικές αποκρύψεις

καθυστέρησης μνήμης προτείνεται η σχεδίαση της DAE αρχιτεκτονικής στο τομέα των GPUs για την βελτίωση της απόδοσης τους. Με την ικανότητα της να αποκρύπτει καθυστερήσεις στη μνήμη και να μειώνει τις αστοχίες της μνήμης cashe καθίσταται αρκετά αποδοτική σε αυτό το τομέα. Παρατηρήθηκαν αυξήσεις στην απόδοση έως και 40% και μειώθηκε αποτελεσματικά συγχρόνως και η κατανάλωση της ενέργειας κατά 30%.

Εν συνεχεία παρουσιάστηκε η αρχιτεκτονική MTDAE(Multithread Decoupled Access/Execute) με σκοπό την εκμετάλλευση της παραλληλίας. Υλοποιήθηκε αρχικά στο MARS-M computer το 1980.Στοχεύει στην συνολική ενίσχυση της αποδοτικότητας της μηχανής και ενός νήματος. Ο συνδυασμός των πολλαπλών νημάτων και της αρχιτεκτονικής DAE επιτρέπουν στον επεξεργαστή να εκμεταλλεύονται του παραλληλισμού με τις πολλαπλές μονάδες εκτέλεσης που ανατίθενται σε ένα νήμα. Κάθε λειτουργική μονάδα συνδέεται με ένα νήμα και έχουν την δυνατότητα να ανταλλάσσουν τα νήματα αυτά. Πολλές αρχιτεκτονικής χρησιμοποιούν MTDAE όπως HEP, MASA, Horizon για εναλλαγή μεταξύ των νημάτων σε κάθε κύκλο. Οι αποδόσεις αυτών των νημάτων βελτιώθηκαν από 30% μέχρι και 90%.

2.2 Υλοποιήσεις της χρήσης των νάνων για βελτίωση της απόδοσης

Με το πέρασμα των χρόνων αντιλαμβανόμενοι τη σπουδαιότητα των νάνων στο παράλληλο προγραμματισμό έγιναν προσπάθειες για εξάπλωση των νάνων σε ευρύτερες υπολογιστικές μεθόδους. Η αρχιτεκτονική με μορφή streaming που παρέχει η FPGA αποτέλεσε κύρια αρχιτεκτονική για βελτιστοποιήσεις των αποδόσεων σε τομείς όπως η οικονομία και η δυναμική των υγρών, τομείς που όπως θα δούμε και παρακάτω βρίσκουν εφαρμογή και οι δεκατρείς νάνοι. Ωστόσο, ο προγραμματισμός αυτών των πλακετών για την εκμετάλλευση του παράλληλου προγραμματισμού απαιτεί υψηλές γνώσεις γλώσσας προγραμματισμού χαμηλού επιπέδου. Για την αντιμετώπιση αυτού το προβλήματος μία καλή λύση είναι η χρησιμοποίηση της OPENCL η οποία παρέχει εύχρηστο και φορητό μοντέλο προγραμματισμού για CPUs,GPUs και τώρα και FPGAs.Για την βελτίωση της απόδοσης των OPENCL kernels στις FPGA δημιουργήθηκαν νέοι τρόποι κάτω από συγκεκριμένους περιορισμούς του υλικού. Στη συνέχεια ,αυτές οι τεχνικές αυτές εφαρμόστηκαν στο opendwarf benchmark suite όπου εμπεριέχονται πολλές και διάφορες μέθοδοι για την αξιολόγηση της βελτιστοποίησης ανάλογα με την απόδοση και τους πόρους που χρησιμοποιήθηκαν.

Ένα χαρακτηριστικό των δεκατριών νάνων είναι ότι μπορούν να βρούν εφαρμογή σε αρκετές πλατφόρμες. Για την ανάδειξη της εφαρμογής αυτής χρησιμοποιήθηκαν σαν benchmarks

σε αρκετές πλατφόρμες πολυπήρηνων επεξεργαστών της εταιρίας AMD, στη πλατφόρμα Intel Xeon Phi P1750 και τέλος χρησιμοποιήθηκε η FPGA, Xilinx Virtex-6 LX760[5]. Οι αλγοριθμικοί μέθοδοι που χρησιμοποιήθηκαν ως benchmarks ήταν N-Body Methods, Dynamic Programming, Structured Grid, Graph Traversal, Combinational Logic, Sparce Linear Algebra. Ειδικότερα η αλγοριθμική μέθοδος των δομημένων πλεγμάτων ήταν ιδιαίτερα αποδοτική στον επεξεργαστή της πλατφόρμας Intel Xeon Phi P1750. Στην FPGA αναλόγως τη χρήση των επιταχυντών άλλαζε και η απόδοση του συστήματος. Με τη χρήση πέντε επιταχυντών πραγματοποιήθηκε πολύ καλή απόδοση πλησιάζοντας την απόδοση της πλατφόρμας της Intel Xeon Phi.

Κεφάλαιο 3

Οι Δεκατρείς Νάνοι

Επισκόπηση

Σε αυτό το κεφάλαιο θα κάνουμε μια σύντομη περιγραφή των 13 αλγοριθμικών μεθόδων εστιάζοντας περισσότερο στον spectral τον οποίο εξετάζουμε σε αυτή την διπλωματική εργασία, και στην συνέχεια θα κάνουμε μια σύντομη περιγραφή των υλοποιήσεων που πραγματοποιήσαμε.

3.1 Οι δεκατρείς νάνοι

Όπως αναφέρεται και στο πρώτο κεφάλαιο ο αρχικός εμπνευστής αυτών των αλγοριθμικών μεθόδων είναι ο Phil Colella ο οποίος αρχικά παρουσίασε 7 αλγοριθμικές μεθόδους στην συνέχεια ο wu feng προσέθεσε άλλες 6.

Η κατηγοριοποίηση των μεθόδων είναι αρκετά σημαντική και δημιουργήθηκε για να καλύπτει τις ανάγκες όλων των κοινών αλγορίθμων. Επίσης, βοηθά το προγραμματιστή ανάλογα και με τις απαιτήσεις που έχει να επιλέξει και την κατάλληλη αλγοριθμική μέθοδο με σκοπό την αποτελεσματικότερη απόδοση της εφαρμογής.

Η τελική μορφή των δεκατριών νάνων αποτελείται από τους παρακάτω αλγοριθμικούς μεθόδους:

- Dense Linear Algebra
- Sparce Linear Algebra
- Spectral Methods
- N-Body Methods
- Structure Grids
- Unstructure Grids
- Map Reduce & Monte Carlo
- Combinational Logic
- Graph Traversal
- Dynamic Programming
- Backtrack and Branch + Bound
- Construct Graphical Models
- Finite State Machine

3.2 Περιγραφή των δεκατριών νάνων

3.2.1 Dense Linear Algebra

Οι εφαρμογές αυτές ,περιλαμβάνουν γραμμικές πράξεις διανυσμάτων και πινάκων. Υπάρχουν τρία διαφορετικά επίπεδα που περιλαμβάνουν πολλαπλασιασμούς μεταξύ διανυσμάτων ,πινάκων και πινάκων με διανυσμάτων. Στην ουσία ,αυτοί οι μέθοδοι είναι αρμόδιες για εφαρμογές που χρειάζονται προσβάσεις στην μνήμη και έχουν υψηλό βαθμό εξαρτήσεων δεδομένων. Λαμβάνουν δράση κυρίως σε τομείς της γραμμικής άλγεβρας όπως το LAPACK όπου είναι μια βιβλιοθήκη λογισμικού για την αριθμητική γραμμική άλγεβρα και περιέχει ρουτίνες για την επίλυση γραμμικών εξισώσεων ,συστημάτων και γραμμικών ελαχίστων τετραγώνων. Ένα άλλο πεδίο εφαρμογής είναι το data mining και κυρίως το stream cluster όπου βοηθά στην ομαδοποίηση μεγάλου όγκου δεδομένων σε τομείς όπως η οικονομία και κυρίως για οικονομικές συναλλαγές .

3.2.2 Sparce Linear Algebra

Οι εφαρμογές αυτές περιλαμβάνουν γραμμικές πράξεις διανυσμάτων και πινάκων όπως και στις dense linear. Ο νάνος αυτός χρησιμοποιείται σε περιπτώσεις όπου οι πίνακες που δέχεται ως είσοδο έχουν μεγάλο αριθμό μηδενικών στοιχείων.

3.2.3 N-body Methods

Οι εφαρμογές αυτές επικεντρώνονται στη λύση προβλημάτων ενός συστήματος όπου κάθε στοιχείο του συστήματος εξαρτάται αυστηρά από την κατάσταση κάθε άλλου στοιχείου του συστήματος. Οι μέθοδοι αυτοί βοηθούν στον υπολογισμό της συμπεριφοράς των μορίων και γενικά βρίσκουν εφαρμογή στην επιστήμη της αστρονομίας και της κοσμολογίας.

3.2.4 Map Reduce & Monte Carlo

Ο νάνος αυτός ονομαζόταν αρχικά "Monte Carlo", εξαιτίας της τεχνική της χρήσης στατιστικών μεθόδων βασισμένων σε επαναλαμβανόμενες τυχαίες δοκιμές. Τα μοτίβα που ορίζονται από το μοντέλο προγραμματισμού MapReduce είναι μια πιο γενική έκδοση της ίδιας ιδέας δηλαδή η επαναλαμβανόμενη ανεξάρτητη εκτέλεση μιας λειτουργίας, όπου μας βοηθά να καταλήξουμε σε κάποια συγκεντρωτικά αποτελέσματα στο τέλος. Σχεδόν δεν απαιτείται επικοινωνία μεταξύ των διαδικασιών. Εφαρμόζουν σε κατανεμημένες αναζητήσεις, τρόπος που χρησιμοποιείται και στις αναζητήσεις στο google και στη βιοπληροφορική όπου χρησιμοποιείται για την ευθυγράμμιση του DNA η των πρωτεϊνών για τον προσδιορισμό ομοιότητας των στοιχείων

3.2.5 Combinational Logic

Ο νάνος αυτός χρησιμοποιείται για τον παραλληλισμό αλγορίθμων που κατά βάση χρησιμοποιούν λογικές πράξεις με στόχο την αύξηση της απόδοσης του αλγορίθμου. Χρησιμοποιείται κυρίως σε τεχνικές κρυπτογράφησης και αποκρυπτογράφησης.

3.2.6 Dynamic Programming

Είναι μια αλγοριθμική τεχνική όπου υπολογίζει λύσεις με την επίλυση απλούστερων υπερκαλυπτόμενων υποπροβλημάτων. Η βασική ιδέα του δυναμικού προγραμματισμού είναι η χρήση της μνήμης για να αποθηκεύονται οι λύσεις των υποπροβλημάτων ώστε να μην είναι αναγκαία η επίλυσης τους πολλαπλές φορές. Ο δυναμικός προγραμματισμός βρίσκει εφαρμογή σε προβλήματα βελτιστοποίησης. Είναι γνωστή η χρησιμοποίηση του σε προβλήματα δικτιών όπως προβλήματα γράφων που υπολογίζει το βέλτιστο μονοπάτι.

3.2.7 Backtrack and Branch+Bound

Ο Branch and Bound είναι ένας αλγόριθμος που χρησιμοποιείται για την επίλυση προβλημάτων βελτιστοποίησης. Η βασική του ιδέα είναι η έξυπνη απαλοιφή τμημάτων του χώρου αναζήτησης στα οποία γνωρίζουμε ότι δεν μπορεί να υπάρχει λύση του προβλήματος. Ο αλγόριθμος έχει χώρο αναζήτησης στη μορφή δέντρου. Κάθε κόμβος αντιπροσωπεύει ένα σύνολο από λύσεις που μπορούν να χωριστούν σε αμοιβαίως αποκλειόμενα σύνολα. Κάθε υποσύνολο στην διαμέριση εκπροσωπείται απο ένα παιδί του αντίστοιχου κόμβου. Προβλήματα που μπορούν να επιλυθούν με αυτόν τον αλγόριθμο είναι το integer linear programming όπου είναι μια μαθηματική βελτιστοποίηση όπου οι τιμές του προγράμματος πρέπει να είναι αυστηρά ακέραιοι.

3.2.8 Finite State Machine

Ο αλγόριθμος αυτός χρησιμοποιεί ένα ντετερμινιστικό μοντέλο FSM για την αναζήτηση μοτίβων σε μια συμβολοσειρά. Εφαρμόζεται σε τομείς όπως αποκωδικοποιήσεις video και data mining δηλαδή μια μορφή υπολογισμού προτύπων με μεγάλο όγκο δεδομένων όπου χρησιμοποιούνται κλάδοι της στατιστικής και των βάσεων δεδομένων.

3.2.9 Graph Traversal

Ο νάνος αυτός επισκέπτεται πολλούς σε ένα γράφο ακολουθώντας διαδοχικές ακμές. Οι εφαρμογές αυτές περιλαμβάνουν έμμεσες αναζητήσεις και σχετικά μικρή ποσότητα υπολογισμών. Οι αλγόριθμοι αυτοί βρίσκουν εφαρμογή σε αναζητήσεις και ταξινομήσεις.

3.2.10 Construct Graphical Models

Είναι ένα γραφικό μοντέλο όπου οι κόμβοι αντιπροσωπεύουν μεταβλητές και οι άκρες αντιπροσωπεύουν υπό συνθήκη πιθανότητες. Χρησιμοποιούνται κυρίως σε τομείς όπως η βιολογία όπου με στατιστικές εκτιμήσεις μπορούν να μεταφέρουν χρήσιμες πληροφορίες σχετικά με το γενετικό υλικό. Ακόμα επιλύουν μοντέλα Markov όπου χρησιμοποιούνται για την χρονική αναγνώριση προτύπων όπως χειρόγραφά και ομιλίες.

3.2.11 Structure Grids

Οι εφαρμογές αυτές υπολογίζουν τις τιμές των κελιών σε ένα κανονικό πλέγμα. Το σχήμα των δομικών πλεγμάτων είναι στατικό και προσδιορίσιμο. Κάθε έξοδος υπολογίζεται απο την τιμή που περιέχει κάθε στοιχείο και την τιμή των πλησιέστερων γειτονικών σημείων. Οι εφαρμογές αυτές χρησιμοποιούνται στην επεξεργασία εικόνας όπου οι κόμβοι του πλέγματος συνδέονται με pixel, όπως και στην επίλυση ορισμένων διαφορικών εξισώσεων

3.2.12 Unstructure Grids

Τα μη δομικά πλέγματα διαθέτουν δομές δεδομένων συνδεδεμένες με μια λίστα με δείκτες όπου παρακολουθούν την τοποθεσία και την "γειτονιά" των στοιχείων που χρησιμοποιούνται για τον υπολογισμό των αριθμητικών πράξεων που απαιτούνται. Όπως και στη sparce γραμμική άλγεβρα, οι ενημερώσεις περιλαμβάνουν πολλαπλές αναφορές στην μνήμη .Για την ενημέρωση ενός σημείου πρώτα καθορίζεται μια λίστα με τα γειτονικά σημεία και μετά φορτώνει τις τιμές τους. Οι συγκεκριμένοι αλγόριθμοι χρησιμοποιούνται στον κλάδο της δυναμικής των ρευστών.

3.3 Spectral Methods

Είναι μέθοδοι ,όπου χρησιμοποιούνται στα εφαρμοσμένα μαθηματικά για την επίλυση διαφορικών εξισώσεων και περιλαμβάνουν τον μετασχηματισμού Fourrier. Οι μέθοδοι αυτοί συνδέονται με την ανάλυση πεπερασμένων στοιχείων που αναφέραμε προηγουμένως, με τη διαφορά ότι οι spectral μέθοδοι διατυπώνουν μια πιο σφαιρική προσέγγιση. Αυτό, τις καθιστά αρκετά σημαντικές για τον εντοπισμό σφάλματος και μάλιστα ορισμένες φορές είναι η μοναδική και η πιο γρήγορη μέθοδος για τον εντοπισμό των σφαλμάτων. Εφαρμόζεται σε τομείς όπως δυναμική των ρευστών όπου περιγράφει την ροή των υγρών. Ακόμα, στις προβλέψεις του καιρού και την κβαντομηχανική.

3.4 Περιγραφή αλγορίθμου Spectral

Στην διπλωματική αυτή θα ασχοληθούμε με τον συγκεκριμένο αλγόριθμο ο οποίος υπολογίζει το FFT ενός δισδιάστατου πίνακα. Για την λύση του 2D FFT ο αλγόριθμος λύνει ουσιαστικά δύο 1D Ffts ένα για τον χ άξονα και ένα για τον y.

Αρχικά οι είσοδοι που δέχεται ο αλγόριθμος είναι το μέγεθος του πίνακα που πρέπει να είναι σε δύναμη του 2 και τα περιεχόμενα του πίνακα που είναι μιγαδικοί αριθμοί όπου το πραγματικό μέρος χωρίζεται από το φανταστικό με το σύμβολο " " και κάθε μιγαδικός αριθμός χωρίζεται με το σύμβολο ";". Ο αλγόριθμος αρχικά υπολογίζει το 1D FFt για το x άξονα. Υπολογίζει στην συνέχεια τα βάρη όπως φαίνεται παρακάτω ξεχωριστά για το πραγματικό μέρος και για το φανταστικό

```
int bitwidth = (int)(log((double)length)/log(2.0));
  complex<double> w,t,u,v;
  //the complex array splits to real array and complex array

        if (W == NULL || Wlength != length)
        {
                Wlength = length;
                W = new complex<double>[length];

                //compute phase weights
                for (int i = 0; i < length; i++)
                {
                        W[i].real(cos(-((2* i * M_PI) / length)));
                        W[i].imag(sin(-((2* i * M_PI) / length)));
                }
        }
```

Έπειτα χρησιμοποιώντας την μέθοδο πεταλούδας λύνει το 1D Fft.

```
void Solver::FFT(complex<double> *complexLine)
{
  //variables for the FFT
  int bitwidth = (int)(log((double)length)/log(2.0));
  complex<double> w,t,u,v;
  //the complex array splits to real array and complex array

        if (W == NULL || Wlength != length)
        {
                Wlength = length;
                W = new complex<double>[length];

                //compute phase weights
                for (int i = 0; i < length; i++)
                {
                        W[i].real(cos(-((2* i * M_PI) / length)));
                        W[i].imag(sin(-((2* i * M_PI) / length)));
                }
        }
```

```
    //reorder for butterflies
    int firstBit = length >> 1;
    int index = 0;
    int reverseIndex = 0;
    for (;;) {
        if (reverseIndex < index) {
            w = complexLine[index];
            complexLine[index] = complexLine[reverseIndex];
            complexLine[reverseIndex] = w;
        }
        index++;
        if (index == length)
            // Break here, because the code below doesn't terminate for all 1's.
            break;

        // Add 1 to reverse index, going from left to right:
        int carry = firstBit;
        // If adding 1 changed the bit to 0, we have a carry:
        while (((reverseIndex ^= carry) & carry) == 0)
            carry >>= 1;
    }

          //combine and untangle applying phase weights
    //using successive butterflies of 2,4,...,bitwidth
    for (int i = 1; i <= bitwidth; i++)
    {
        int halfStep = (1 << (i - 1));
        for (int j = 0; j < halfStep; j++)
        {
            int tempShift = (j << (bitwidth - i));
            w = W[tempShift];
            for (int k = j; k < length; k += halfStep * 2)
            {
                u = complexLine[k + halfStep];
                v = complexLine[k];
                t = w * u;
                complexLine[k + halfStep] = v - t;
                complexLine[k] = v + t;
            }
        }
    }
}
```

και αφού τελειώσει με τον μετασχηματισμό όλων των γραμμών κάνει transpose τον πίνακα και με τον ίδιο τρόπο υπολογίζει και τις στήλες και τέλος κάνει πάλι transpose τον πίνακα για να επανέλθει πάλι στην κανονική του μορφή.

Κεφάλαιο 4
Αρχιτεκτονικές DAE και DAER

Επισκόπηση

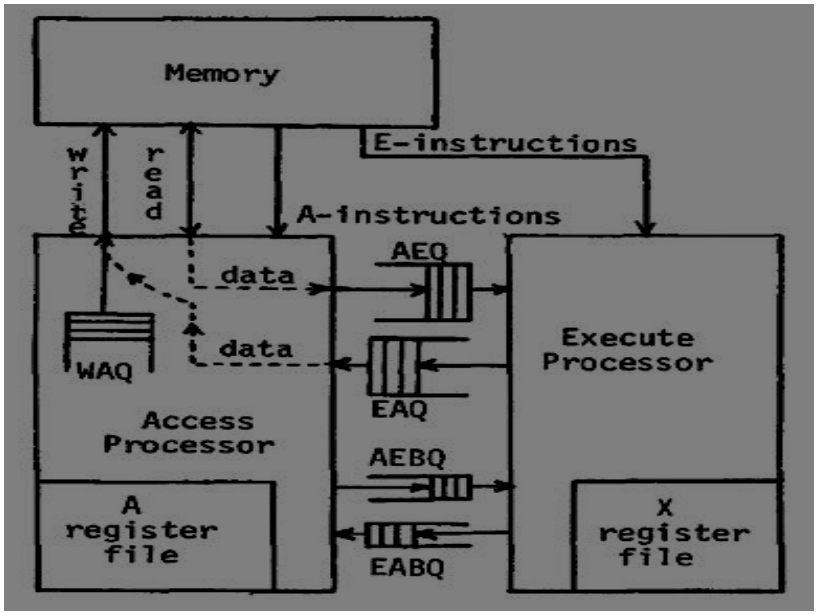
Στο συγκεκριμένο κεφάλαιο περιγράφεται η αρχιτεκτονική decoupled access/execute και μέθοδος με την οποία υλοποιείται όπως επίσης και η αρχιτεκτονική DEAR. Περιγράφεται επίσης η συμβολής της στην βελτίωση της απόδοσης και της κατανάλωσης ενέργειας μιας εφαρμογής.

4.1 Περιγραφή Αρχιτεκτονικής DAE

Τα τελευταία χρόνια οι απαιτήσεις για την επεξεργασία των δεδομένων αυξάνονται και μαζί τους και ο όγκος των δεδομένων. Για αυτό το λόγο, οι περισσότερες προσπάθειες επικεντρώνονται σε υπολογιστικά συστήματα όπου προσπαθούν να συνδυάσουν την ευελιξία του software με την παροχή της υψηλής απόδοσης από το hardware. Ωστόσο ,ορισμένες φορές ο μεγάλος όγκος δεδομένων μπορεί να προκαλέσει σοβαρές καθυστερήσεις και μειώνει την προσπάθεια για επιτάχυνση του προγράμματος από τους επιταχυντές του υλικού. Για την επίλυση αυτού του προβλήματος ο χρήστης θα πρέπει να βελτιστοποιήσει το σύστημα της ανακτήσεις των δεδομένων.

Έτσι γεννήθηκε η ιδέα της αρχιτεκτονικής decouple access/execute προήλθε από τον James E. Smith το 1982 . Στην προσέγγιση του οι μονάδες εκτέλεσης δεν περιείχαν υπολογισμούς διευθύνσεων και ήταν αρμόδιες μόνο για τις αριθμητικές και λογικές πράξεις. Στόχος της αρχιτεκτονικής αυτής ήταν η μείωση της εκτέλεσης του προγράμματος με ελάχιστη δυνατή επέμβαση από το χρήστη.

Είναι αρχιτεκτονική υψηλών αποδόσεων και βασίζεται στην μεγάλη απόζευξη της πρόσβασης και της επεξεργασίας δεδομένων. Το αρχικό instruction stream χωρίζεται σε δύο λειτουργικές μονάδες ,στη μονάδα ανάκτησης δεδομένων και στη μονάδα επεξεργασίας δεδομένων όπου επικοινωνούν μεταξύ τους με FIFO ουρές. Συγκρίνοντας την αρχιτεκτονική DAE με μία άλλη συμβατή αρχιτεκτονική αντιλαμβανόμαστε ότι υπάρχει μεγάλη διαφορά στο κέρδος απόδοσης ,διότι μειώνει την πολυπλοκότητα στη σχεδίαση και την επικοινωνία του επεξεργαστή με την κύρια μνήμη. Πειραματικά αποτελέσματα έχουν δείξει ότι οι βελτιστοποιήσεις μπορούν να κάνουν το πρόγραμμα 2,28 πιο γρήγορο και να μειώσουν την κατανάλωση ενέργειας κατά 15%.



Εικόνα 1: αρχιτεκτονική DAE

Κάθε μονάδα έχει το δικό της instruction stream, τον Access processor και τον Execute processor. Οι δύο αυτοί επεξεργαστές, εκτελούν ξεχωριστά προγράμματα με διαφορετικές λειτουργίες αλλά έχουν παρόμοιο διάγραμμα ροής. Έπειτα, το κάθε ένα instruction stream έχει το δικό τους αρχείο καταχωρητών που αποτελείται από 32 καταχωρητές ακεραίων. Στον επεξεργαστή A οι καταχωρητές δηλώνονται ως καταχωρητές A0,A1.. ενώ στον επεξεργαστή E ,ως X0,X1.. .Τα δύο stream είναι ανεξάρτητες λειτουργικές μονάδες όπου επικοινωνούν με την μνήμη μέσω ουρών.

Ο Access processor είναι υπεύθυνος, για την προσπέλαση της μνήμης εκτελώντας όλες τις απαραίτητες λειτουργίες για την μεταφορά δεδομένων από και προς την μνήμη. Δηλαδή υπολογίζει τις διευθύνσεις και εκτελεί όλες τις αιτήσεις ανάγνωσης και εγγραφής που είναι απαραίτητες. Για αυτό το λόγο αφαιρούμε στο κομμάτι αυτό ότι έχει να κάνει με αριθμητικές και λογικές πράξεις ,οι οποίες βρίσκονται μόνο στον execute processor. Έπειτα τα δεδομένα είτε χρησιμοποιούνται στον access είτε τοποθετούνται στην AEQ(Access to Execute) FIFO ουρά και αποστέλλονται στο execute processor. Είναι αναγκαίο ο access processor να εκτελεστεί πρώτα από τον execute processor ώστε να έχει γίνει η ανάκτηση των δεδομένων που θα χρειαστούν για την επεξεργασία. Επίσης, με την προ επεξεργασία των δεδομένων στον access processor, εξαλείφονται οι περισσότερες αστοχίες της μνήμης cashe και μετατρέπονται σε ευστοχίες βοηθώντας αρκετά τον execute processor. Έτσι ,στον execute processor όπου έχουν εξαλειφθεί οι περισσότερες αστοχίες μειώνονται αισθητά οι καθυστερήσεις και συγχρόνως η απόδοση του προγράμματος βελτιώνεται αισθητά.

Εκεί λοιπόν, εφόσον λάβει ο execute processor όλα τα απαραίτητα δεδομένα που έρχονται από την AEQ, τα επεξεργάζεται τα και τα χρησιμοποιεί για να εκτελέσει τις πράξεις που απαιτούνται

για την επίλυση του αλγορίθμου. Στη συνέχεια εισέρχονται στην EAQ(Execute to Access) FIFO ουρά όπου αποστέλλονται πίσω στον access processor. Τέλος, αφού εκδοθεί η εντολή αποθήκευσης αποστέλλονται στην μνήμη για την αποθήκευση των αποτελεσμάτων.

Ο υπολογισμός των διευθύνσεων απο τον access processor και τα αποτελέσματα που παράγει ο execute processor πραγματοποιούνται παράλληλα. Ο access processor παράγει τις διευθύνσεις όπου πρέπει να αποθηκευτούν τα αποτελέσματα χωρίς απαραίτητα να έχουν έρθει τα αποτελέσματα από τον execute processor. Οι διευθύνσεις αυτές κρατούνται παράλληλα στην WAQ(write address) και όταν φθάσουν τα δεδομένα σε συνδυασμό με την πρώτη διεύθυνση της WAQ αποστέλλοντα στη μνήμη. Αυτό ο συνδυασμός γίνεται αυτόματα με το που φθάσουν τα δεδομένα.

Αξίζει να σημειωθεί ότι υπάρχει μια τρίτη λειτουργική μονάδα ξεχωριστή από τον A και E επεξεργαστή όπου χειρίζεται τα δεδομένα και τις διευθύνσεις όπου χρειάζονται να γραφούν. Ο EAQ επίσης μπορεί να χρησιμοποιηθεί για την μεταφορά δεδομένων στον A processor τα όποια δεν πρόκειται να γραφούν στην μνήμη αλλά χρησιμοποιούνται για τον υπολογισμό διευθύνσεων. Σε κάποιες περιπτώσεις μπορεί να χρειαστεί διπλός υπολογισμός και στους δυο επεξεργαστές ώστε να αποφευχθεί το γεγονός να περιμένει ο A τα δεδομένα από τον E επεξεργαστή. Όταν πάμε να χρησιμοποιήσουμε την αρχιτεκτονική DAE σε συνδυασμό με το software πρέπει να είμαστε αρκετά προσεχτικοί και να συντονίζουμε κατάλληλα τα δεδομένα που επέρχονται από τους δύο επεξεργαστές έτσι ώστε να μεταδίδονται από τις ουρές με τη σωστή σειρά. Σε αρκετές περιπτώσεις το access stream προηγείται του execute με αποτέλεσμα να έχουμε σημαντικές μειώσεις όσο αναφορά την καθυστέρηση της ανάκτησης των δεδομένων από την μνήμης.

Όπως αναφέραμε ,ο υπολογισμός των διευθύνσεων μπορεί να υπολογισθεί πριν ακόμα τα δεδομένα είναι έτοιμα. Οι διευθύνσεις αυτές παραμένουν στο WAQ .Τα δεδομένα διαβιβάζονται από το EAQ στο WAQ πριν πάνε στην μνήμη. Η ανάκτηση των διευθύνσεων πριν ακόμα τα δεδομένα να είναι έτοιμα ,είναι σημαντικός παράγοντας για την βελτίωση της επίδοσης του προγράμματος διότι φορτώνει νέες εντολές χωρίς να έχει γίνει η αποθήκευση της προηγούμενης .Ωστόσο, δημιουργείται ένα σημαντικό πρόβλημα ορισμένες φορές, όπου η νέα εντολή που φορτώνεται χρησιμοποιεί την ίδια θέση μνήμης με την προηγούμενη. Γι αυτό το λόγο, ορισμένες φορές χρησιμοποιούμε interlocks όπου στην ουσία για να φορτώσουμε μια νέα εντολή περιμένουμε την αποθήκευση των δεδομένων της προηγούμενης. Μία άλλη επίσης λύση ,είναι να κάνουμε έναν έλεγχο όποιας καινούργιας διεύθυνσης φορτώνεται με τις διευθύνσεις που είναι αποθηκευμένες στο WAQ .Εάν υπάρχει ίδια,τότε η φόρτωση νέας εντολής πρέπει να περιμένει μέχρι να σταματήσει να ισχύει ο παραπάνω έλεγχος .Ωστόσο ,είναι μια πιο ακριβής λύση συγκριτικά με την προηγούμενη.

4.2 Αρχιτεκτονική DAER

Λαμβάνοντας υπόψιν την αρχιτεκτονική decouple access/execute γεννήθηκε η ιδέα της αποτύπωσης αυτής της αρχιτεκτονικής σε υψηλές αποδόσεις συστημάτων όπου εκμεταλλεύονται και το software και το hardware .Η προτεινόμενη αρχιτεκτονική απεικονίζεται στην παρακάτω εικόνα και μπορεί να εφαρμοστεί σε ένα ευρύ φάσμα εφαρμογών. Η χαρτογράφηση του αλγορίθμου με βάση αυτή τη δομή απαιτεί κάποια συγκεκριμένα βήματα. Το framework της παραπάνω αρχιτεκτονικής ακολουθεί τρία βασικά βήματα μετασχηματισμού. Αρχικά ο κώδικας χωρίζεται σε δύο λειτουργικές μονάδες. Η πρώτη μονάδα είναι αρμόδια για τις προσβάσεις στη μνήμη και η δεύτερη για την εκτέλεση του κύριου αλγορίθμου. Έπειτα, επιλύει όλες τις εξαρτήσεις μνήμης που μπορούν να προκύψουν και τέλος μετατρέπει τις δύο αυτές λειτουργικές μονάδες σε μηχανές ροής δεδομένων.

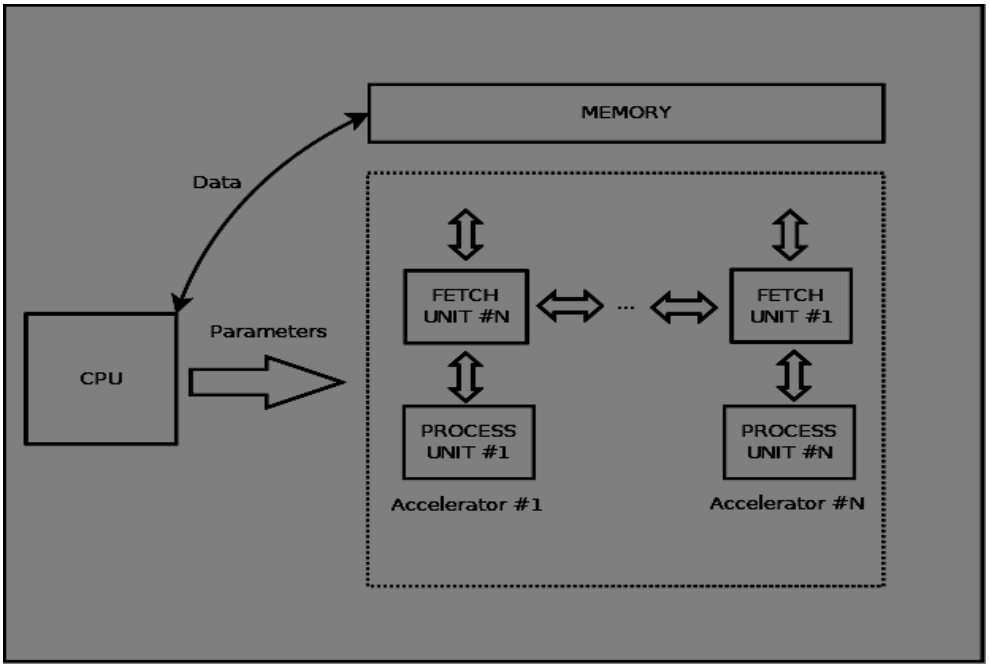
Η ερώτηση λοιπόν που προκύπτει είναι γιατί να προτιμήσουμε την χαρτογράφηση εφαρμογών με την αρχιτεκτονική DAER?

Αρχικά ,όπως και η αρχιτεκτονική DAE μας παρέχει μεγάλη ευελιξία στη χαρτογράφηση ποικίλων εφαρμογών. Εάν αναλογιστούμε δε και ότι οι νάνοι που επιθυμούμε να χαρτογραφήσουμε διακρίνονται για τα υψηλά επίπεδα αφαίρεσης τότε ο συνδυασμός αυτών των δύο παρέχουν ακόμα μεγαλύτερη ευελιξία στη χαρτογράφηση ποικίλων εφαρμογών. Έπειτα, η αρχιτεκτονική αυτή είναι αρκετά αποδοτική σε εφαρμογές που απαιτούν streams για την μεταφορά δεδομένων και είναι ικανή να επιλύσει τυχών αλληλεξαρτήσεις που μπορούν να προκύψουν κατά την διάρκεια της προσπέλασης στην μνήμη.

Πρώτα, χωρίζουμε το κώδικα μας σε δύο νέες λειτουργικές μονάδες όπου τις ονομάζουμε fetch και process αντίστοιχα. Η πρώτη μονάδα είναι αρμόδια για την προσπέλαση της κύριας μνήμης. Επίσης, λαμβάνει δεδομένα από την κύρια μνήμη και τέλος είναι αρμόδια για την αποθήκευση των αποτελεσμάτων σε αυτή. Ενώ η δεύτερη είναι αρμόδια για την επίλυση όλων των αριθμητικών πράξεων και για την κύρια υλοποίηση του αλγορίθμου. Χρησιμοποιούνται FIFO interface για την μεταφορά των δεδομένων από την μονάδα fetch με την κύρια μνήμη και από την fetch στην process μονάδα. Επομένως ,το πρόγραμμα γίνεται αρκετά ευέλικτο και καθιστά εύκολο των χειρισμό όλων των εφαρμογών που απαιτούν μεταφορές δεδομένων σε μορφή streaming.

Ακόμα η αρχιτεκτονική αυτή παρέχει την δυνατότητα χαρτογράφησης εφαρμογών που έιναι βασισμένες σε διεργασίες. Κάθε διεργασία μπορεί να χαρτογραφηθεί σε ένα επιταχυντή όπου ο κάθε επιταχυντής μπορεί να επικοινωνεί με τους υπόλοιπους μέσω της fetch unit. Επομένως, ορισμένες φορές η μονάδα fetch μπορεί να παρακάμψει να λάβει δεδομένα από την κύρια μνήμη και να λάβει άμεσα από τις υπόλοιπες μονάδες fetch.

Παρουσιάζοντας την παραπάνω αρχιτεκτονική συμπεραίνουμε ότι έχει πολλά να προσφέρει στην απόδοση μιας εφαρμογής .Ωστόσο, σε περιπτώσεις όπου ολόκληρη η εφαρμογή μπορεί να χαρτογραφηθεί σε ένα απλό επιταχυντή πρέπει να είμαστε προσεχτικοί και να επιλύσουμε τυχών προβλήματα που μπορούν να προκύψουν με τις εξαρτήσεις μνήμης κατά την διάρκεια της προσπέλασης της. Ένας τρόπος επίλυσης είναι η διαίρεση της μονάδας fetch σε υπομονάδες οι οποίες εκτελούνται παράλληλα σε μορφή pipiline. Επομένως, είναι αρκετά κατανοητό χωρίς να έχουμε παρουσιάσει ακόμα τα αποτελέσματα των χρόνων και της βελτίωσης που επέφερε αυτή η αρχιτεκτονική ότι μπορεί να παρέχει αρκετά υψηλά επίπεδα αποδοτικότητας.



Εικόνα 2: Αρχιτεκτονική DAER

Όπως γνωρίζουμε οι επιταγχυντές στο υλικό γίνονται ολοένα και πιο δημοφιλείς διότι παρέχουν την δυνατότητα της αύξησης απόδοσης του προγράμματος και συγχρώνως την μείωση της κατανάλωσης της ενέργειας.Ωστόσο,ο σχεδιασμός είναι αρκετά απαιτητικός,διότι η μεταφορά δεδομένων από την εξωτερική μνήμη στον επιταγχυντή πρέπει να γίνει αρκετά προσεχτικά.Όμως με την δημιουργία αυτοματοποιημένων εργαλίων όπως το vivado hls μπορούμε να υλοποιήσουμε τέτοιες εφαρμογές ευκολότερα.

Προβλέπεται ότι τα μελλοντικά υπολογιστικά συστήματα θα πρέπει να βασίζονται σε επιταγχυντές του υλικού για την γενική βελτίωση των εφαρμογών.Ήδη,πολλα κινητά χρησιμοποιούν σε μεγάλο βαθμό επιταγχυντές για την εκτέλεση εργασιών όπως επεξεργασία μουσικής και βίντεο.

Το HLS ειναι μια αρκετα ελπιδοφορα εφαρμογή για να δρομολογησει τον σχεδιασμο ενός πολύπλοκου προβλήματος,καθως επιτρέπει την αυτόματη δημιουργία επιταγχυντών υλικού. Δυστυχώς, ακόμα και στο HLS η παροχή των δεδομένων από την μνήμη πρέπει να γίνεται με προσεχτικό τρόπο και να συντονιζονται σωστα με χειροκινητες βελτιστοποιησεις προκειμενου να

επιτευχθει σωστη αποδοση στους επιταγχυντες υλικου.Ακόμα παρέχει την δυνατότητα χρησιμοποιήσης data flow directives για την βελτιστοποιήση της μονάδας επεξεργασίας οι οποίες θα αναλυθουν αργότερα.

Στην συγκεκριμένη διπλωματική θα χαρτογραφήσουμε τις αλγοριθμικές μεθόδους για το spectral method.Η χαρτογράφηση θα γίνει με την βοήθεια του εργαλείου vivado hls.

4.3 Ενεργειακή Αποδοτικότητα

Η ενεργειακή αποδοτικότητα έχει καταστεί ένας από τα πιο σημαντικούς παράγοντες όσο αναφορά τη σχεδίαση στο hardware λόγω του περιορισμένου χρόνου ζωής της μπαταρίας και του ενεργειακού κόστους. Το γεγονός ότι δεν μπορεί να παρέχεται σταθερή πυκνότητα ισχύος έχει ώς αποτέλεσμα την αδυναμία εξοικονόμηση τάσης και συχνότητα. Για αυτό πρέπει να βρούμε προγράμματα τα οποία ξοδεύουν το χρόνο τους περιμένοντας δεδομένα από την μνήμη και συνεπώς δεν είναι ευαίσθητα στις αλλαγές.

Όπως αναφέραμε παραπάνω η αρχιτεκτονική dae χωρίζεται σε δύο φάσεις την access και την execute. Και όπως θα δούμε είναι μια αρχιτεκτονική που βοηθάει αρκετά την ενεργειακή απόδοση του προγράμματος και μειώνει την κατανάλωση ισχύος. Πρώτα απ’ όλα χωρίζεται σε δύο τμήματα όπου χρησιμοποιούν την ίδια μνήμη και μας παρέχει την δυνατότητα να χρησιμοποιήσουμε την βέλτιστη συχνότητα ξεχωριστά. Η λειτουργική μονάδα access μεταβάλει τις περισσότερες αποτυχίες της cashe σε ευστοχίες πριν εισέλθει στην λειτουργική μονάδα execute. Επίσης ,δαπανά ένα μικρό μέρος του χρόνου στον υπολογισμό των διευθύνσεων και τον περισσότερο μέρος περιμένει να έρθουν τα δεδομένα από την μνήμη. Έτσι, έχει ως αποτέλεσμα να μην επηρεάζεται από την συχνότητα του συστήματος. Άρα, στη συγκεκριμένη μονάδα δεν απαιτείται και ούτε χρειάζεται μεγάλη τιμή η συχνότητα παρά μόνο η ελάχιστη δυνατή τιμή για την εκτέλεση αυτής της φάσης. Όλο αυτό συνδράμει στην εξοικονόμηση ενέργειας και στην αποφυγή της επιβάρυνσης της απόδοσης του προγράμματος.

Όσο αναφορά την execute μονάδα γνωρίζουμε ότι οι περισσότερες αστοχίες της cashe έχουν εξαλειφθεί από την προ επεξεργασία των δεδομένων στην access λειτουργική μονάδα. Επομένως μειώνοντας τις αστοχίες της μνήμης μειώνονται και οι καθυστερήσεις με αποτέλεσμα στη συγκεκριμένη μονάδα από άποψη μεταφορά δεδομένων η υψηλότερη συχνότητα συνίσταται ως καταλληλότερη.

Κεφάλαιο 5

Υλοποίηση

Επισκόπηση

Στο κεφάλαιο αυτό περιγράφονται οι υλοποιήσεις που σχεδιάστηκαν για την βελτιστοποίηση του αλγορίθμου spectral στο vivado hls καθώς και στο vidado μέσω του HMC.

5.1 Περιγραφή 1ης υλοποιήσης

Στην πρώτη υλοποίηση περάσαμε στο test bench του HLS τον αλγόριθμο spectral και φτιάξαμε κάποιες συναρτήσεις για να μπορούμε να δεχόμαστε την είσοδο απο αρχείο. Ο χρήστης αρχικά επιλέγει τις παραμέτρους του αλγορίθμου εισάγοντας τις στο αρχείο *inputfile*, πρώτα εισάγει το μέγεθος του πίνακα που θέλει να γίνει ο μετασχηματισμός fourier και στην συνέχεια τις τιμές του πίνακα οι οποίες είναι μιγαδικοί αριθμοί οι οποίοι χωρίζουν το πραγματικό από το φανταστικό μέρος με το σύμβολο “ “ και κάθε αριθμός χωρίζεται από τον επόμενο με το σύμβολο “;”.Για να μπορεί το πρόγραμμα να διαβάσει τις εισόδους από το αρχείο χρειάστηκε να δημιουργήσουμε τις κατάλληλες συναρτήσεις όπως η parseConfigFile η οποία δημιουργεί το αρχείο και διαβάζει μια μια τις γραμμές του αρχείου και η getnextLine η οποία ενημερώνει την parseconfig πότε αλλάζει η σειρά του αρχείου. Επίσης για να μπορέσουμε να μετρήσουμε την ταχύτητα εκτέλεσης του αλγορίθμου στο software δημιουργήσαμε δυο συναρτήσεις start και finish.Στην συνάρτηση start απλά ορίζουμε την αρχή της μέτρησης μας και την καλούμε ακριβώς πριν καλέσουμε την συνάρτηση solver η οποία είναι υπεύθυνη για την εκτέλεση του αλγορίθμου spectral.

```
void Settings_start(Settings * settings) {
        settings->startTime  = clock();
}
```

Στην συνάρτηση finish δηλώνουμε το τέλος της μέτρησης και μετράμε τον χρόνο εκτέλεσης

```
endTime = clock();
   elapsed = ((double)endTime – (double)settings->startTime)/(double)CLOCKS_PER_SEC;
```

Επίσης δημιουργούμε και το αρχείο outputResult στο οποίο αποθηκεύουμε τα αποτελέσματα του αλγορίθμου μας και ένα αρχείο outputprofile στο οποίο αποθηκεύουμε το μέγεθος του πίνακα που μετασχηματίσαμε καθώς και τον χρόνο που απαιτήθηκε για την εκτέλεση του σε second.
Στην συνέχεια δημιουργήσαμε τον accelerator μας. Στην συγκεκριμένη υλοποίηση δεν αλλάξαμε την κεντρική ιδέα του αλγορίθμου. Αρχικά προτού καλέσουμε τον accelerator από το testbench

μετατρέπουμε την είσοδο μας η οποία είναι ένας δισδιάστατος πίνακας σε μονοδιάστατο, και από complex_double δηλαδή μιγαδικός αριθμός τον μετατρέπουμε σε double. Ο τρόπος με τον οποίο έγινε αυτή η μετατροπή είναι ο εξής. Αν έχουμε έναν πίνακα μεγέθους length με μιγαδικούς αριθμούς δημιουργούμε έναν νέο πίνακα μεγέθους 2χlength όπου τα length πρώτα στοιχεία είναι το πραγματικό μέρος των αριθμών και τα επόμενα length στοιχεία είναι το φανταστικό μέρος των αριθμών.

```
for(i=0; i<solver->length; i++)
        {
                for(j=0; j<solver->length; j++)
                {
                        input[k]=__real__ solver->complexArray[i][j];
                        k++;
                }
        }

        for(i=0; i<solver->length; i++)
                {
                        for(j=0; j<solver->length; j++)
                        {
                                input[k]=__imag__ solver->complexArray[i][j];
                                //printf("temp[%d])= %f\n",k,temp[k]);
                                k++;
                        }
                }
```

σε όλες τις υλοποιήσεις οι μιγαδικοί θα αντιμετωπίζονται με τον ίδιο τρόπο έτσι ώστε να μπορούμε να έχουμε συγκρίσιμα αποτελέσματα.

Στην πρώτη υλοποίηση ο accelerator λειτουργεί όπως ακριβώς ο solver στο Software. Αρχικά ορίσαμε την είσοδο μας όπως και την έξοδο ως μια FIFO.

```
#pragma HLS INTERFACE ap_fifo depth=8196 port=input
#pragma HLS INTERFACE ap_fifo depth=8196 port=output
```

Στην συνέχεια απομονώσαμε μια γραμμή της εισόδου και καλέσαμε την συνάρτηση FFT η οποία είναι υπεύθυνη για τον μετασχηματισμό αυτής της γραμμής. Μετά την επιστροφή της συνάρτησης αποθηκεύουμε πίσω τα αποτελέσματα της FFT και στην συνέχεια καλούμε την συνάρτηση transpose η οποία κάνει μετατόπιση του πίνακα και πάλι μετά την συνάρτηση FFT. Στην συνάρτηση FFT και transpose οι αλλαγές που έγιναν αφορούν την μετατροπή από complex_double

σε double και από δυσδιάστατο πίνακα σε μονοδιάστατο.

Αφού ολοκληρώσαμε τον αλγόριθμο τρέξαμε το synthesis με κεντρική συνάρτηση την accelerator. Το synthesis μας δίνει πληροφορίες σχετικά με την κατανάλωση πόρων της FPGA. Τέλος τρέξαμε και το cosimulation test το οποίο παράγει το RTL και εμφανίζει τον χρόνο εκτέλεσης του αλγορίθμου σε κύκλους ρολογιού και επειδή η περίοδο είναι ορισμένη στα 4ns μπορούμε πολλαπλασιάζοντας να υπολογίσουμε τον χρόνο εκτέλεσης σε sec.

Τέλος θα πρέπει να γνωρίζουμε αν τα αποτελέσματα που εξάγει ο accelerator μας είναι ίδια με εκείνα που παράγει το software, για τον λόγω αυτό δημιουργήσαμε μια συνθήκη σύγκρισης των αποτελεσμάτων που παράγει το software με τα αποτελέσματα που θα παράξουμε στο hardware όπως φαίνεται και στο παρακάτω κώδικα.

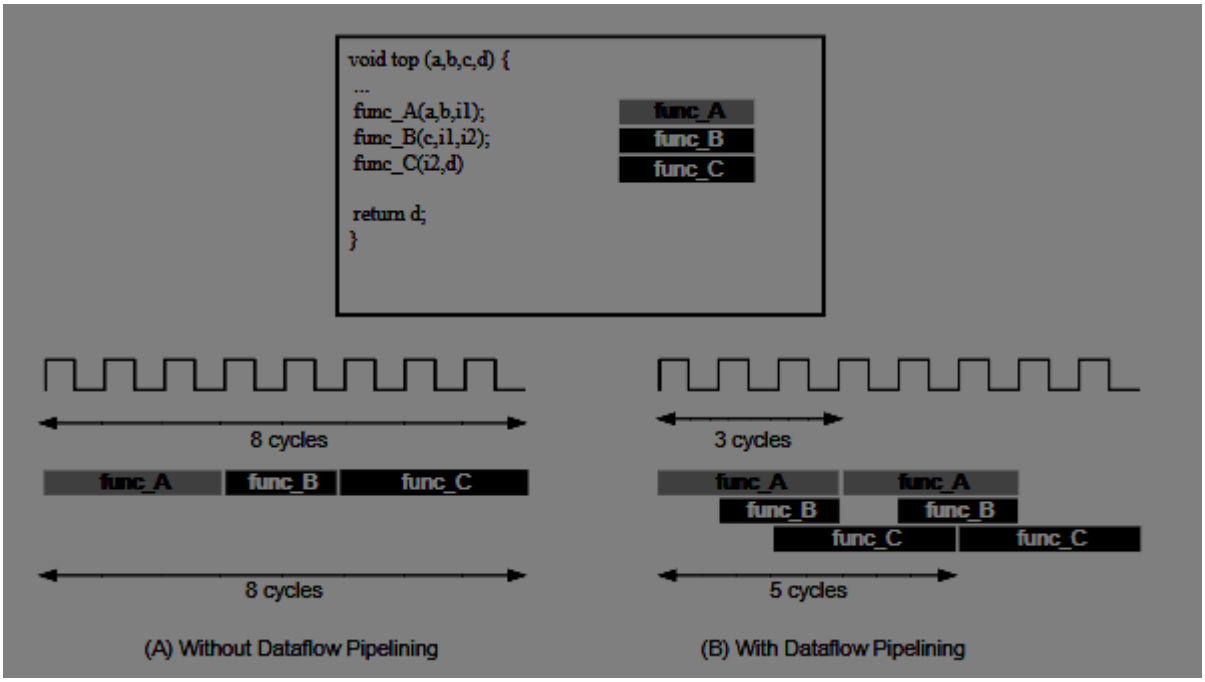
```
for(i=0; i<2*len; i++)
    {
        if(input[i]-Ctemp[i] < 0.000001 && Ctemp[i]-input[i] < 0.000001)
                p++;
    }

    if(p==2*len)
        printf("\n\n !!!!!The arrays are equal!!!!! \n\n");
    else
        printf("\n\n !!!!!The arrays are NOT equal!!!!! \n\n");
```

όπου input είναι τα αποτελέσματα που εξάγει το software και Ctemp είναι τα αποτελέσματα που εξάγει το hardware.

5.2 Περιγραφή 2ης υλοποίησης

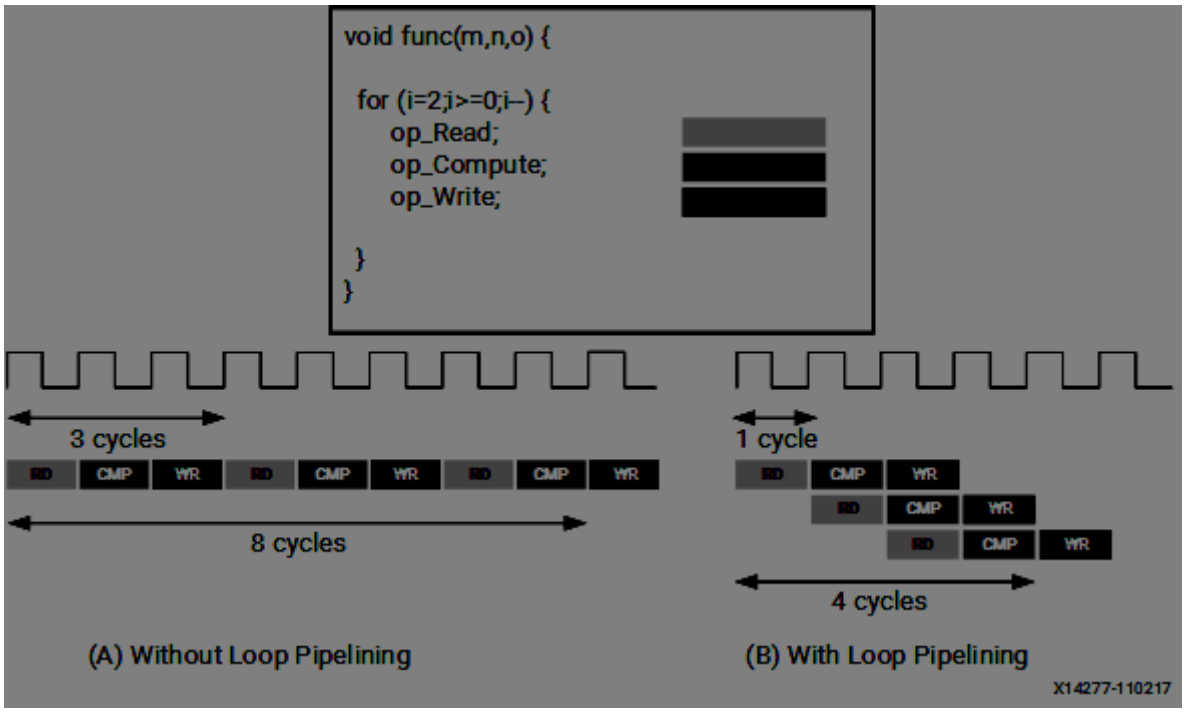
Στην δεύτερη υλοποίηση χρησιμοποιήσαμε τα εργαλεία βελτιστοποίησης του vivado hls για τον accelerator μας έτσι ώστε να καταφέρουμε μείωση του χρόνου εκτέλεσης του αλγορίθμου spectral. Αρχικά μετατρέψαμε του πίνακες σε FIFO ουρές. Για την δημιουργία FIFO ουρών χρησιμοποιήσαμε την εντολή *#pragma HLS INTERFACE ap_fifo depth=length port=output* όπου με το depth ορίζουμε το μέγεθος της ουράς μας και με το port ποιος είναι εκείνος ο πίνακας που θέλουμε να μετατρέψουμε σε ουρά FIFO. Επίσης για την βελτιστοποίηση του προγράμματος έγινε χρήση ορισμένων directives που μας παρέχει το vivado HLS, όπως την #pragma HLS dataflow η οποία επιτρέπει στην εφαρμογή διεργασιών να τρέχουν με μορφή Pipeline. Παρατηρούμε από την εικόνα, ότι στην πρώτη υλοποίηση όπου δεν χρησιμοποιούμε τη συγκεκριμένη directive ,οι τρεις συναρτήσεις υλοποιούνται σε 8 κύκλους. Ενώ ,στη δεύτερη υλοποίηση έπειτα από την χρησιμοποίηση της dataflow directive υλοποιείται σε 5 κύκλους. Αυτό γίνεται διότι η δεύτερη συνάρτηση δεν περιμένει να τελειώσει η πρώτη και η Τρίτη ομοίως για η δεύτερη. Άρα, μειώνοντας το latency της συνάρτησης αυξάνεται η απόδοση του αλγόριθμου και η μειώνεται η κατανάλωση ενέργειας. Η χρησιμοποίηση της dataflow είναι πολύ σημαντική στον συγκεκριμένο αλγόριθμο διότι αυτό που θέλουμε να πετύχουμε αργότερα είναι ο υπολογισμός των διευθύνσεων της συνάρτησης fetch και η παραγωγή αποτελεσμάτων από την process να γίνεται παράλληλα.



Εικόνα 3: Εφαρμογή dataflow directive

Χρησιμοποιήσαμε επίσης το directive **#pragma HLS pipeline.** Το δdirective αυτό μειώνει το

διάστημα έναρξης για μία συνάρτηση ή έναν βρόχο επανάληψης επιτρέποντας την ταυτόχρονη εκτέλεση των λειτουργιών. Μια τέτοια συνάρτηση η βρόχος μπορεί να επεξεργαστεί νέες εισόδους σε κάθε κύκλο ρολογιού. Το προεπιλεγμένο διάστημα έναρξης είναι II=1, το οποίο επεξεργάζεται μια νέα είσοδο σε κάθε κύκλο ρολογιού. Το pipeline επιτρέπει στις λειτουργίες των βρόχων να γίνονται παράλληλα όπως φαίνεται στο παρακάτω σχήμα. Στο σχήμα φαίνεται η σειριακή λειτουργία όπου χρειάζονται 3 κύκλοι μεταξύ κάθε ανάγνωσης κα συνολικά χρειάζονται 8κύκλοι για να ολοκληρωθεί και η τελευταία εγγραφή, ενώ με το pipeline οι κύκλοι αυτοί μειώνονται στους 4.



Εικόνα 4: Εφαρμογή pipeline directive

Τέλος χρησιμοποιήθηκε και η συνάρτηση **#pragma HLS LOOP_MERGE**. Αρκετά συχνά γίνεται χρήση πολλών διαδοχικών βρόχων κάτι το οποίο μπορεί να προκαλέσει αύξηση του latency δημιουργώντας περιττούς κύκλους ρολογιού. Το γεγονός αυτό αποτρέπει την προσπάθεια για βελτιστοποίηση. Ένας τρόπος επίλυσης είναι η συγκεκριμένη συνάρτηση όπου επιτυγχάνεται η συγχώνευση διαδοχικών βρόχων μειώνοντας την καθυστέρηση όπου μπορεί να προκύψει επιτυγχάνοντας τη βελτίωση της απόδοσης του αλγορίθμου.

5.3 Περιγραφή 3ης υλοποίησης

Η ιδέα της 3ης υλοποίησης είναι να δημιουργήσουμε μια μεγάλη BRAM από την οποία θα έχουμε πρόσβαση σε όλα τα απαραίτητα δεδομένα για την υλοποίηση του αλγορίθμου. Κάθε συνάρτηση χρησιμοποιεί ένα μέρος αυτού του πίνακα για να εκτελεστεί. Αρχικά χωρίσαμε αυτήν την μεγάλη BRAM σε μέρη έτσι ώστε να μπορούμε να πιο εύκολα να επιλέγουμε τα κομμάτια που χρειαζόμαστε για επεξεργασία. Το πρώτο κομμάτι [0,len] από την αρχή δηλαδή μέχρι και το μέγεθος του πίνακα που ορίσαμε στο τετράγωνο αντιπροσωπεύει το πραγματικό μέρος της εισόδου που δίνει ο χρήστης στο InputFile, το δεύτερο κομμάτι [len,2len] είναι το φανταστικό μέρος της εισόδου που δίνει ο χρήστης. Στην συνέχεια έχουμε το [2len,2len+length] και άλλο ένα ίδιο κομμάτι μνήμης το οποίο αντιπροσωπεύει το πραγματικό και το φανταστικό μέρος της γραμμής της εισόδου που θα μετασχηματιστεί από την συνάρτηση FFT. Η υλοποίηση αυτή όπως καταλαβαίνουμε χρησιμοποιεί πληθώρα αριθμό πόρων γιατί γίνονται πολλές πράξεις απλά για να περιγράψουμε ποιό σημείο της μνήμης χρειαζόμαστε κάθε φορά. Και όπως θα δούμε και στον επόμενο κεφάλαιο δεν βελτιστοποιεί τον αλγόριθμο μας ούτε είναι ενεργειακά αποδοτική.

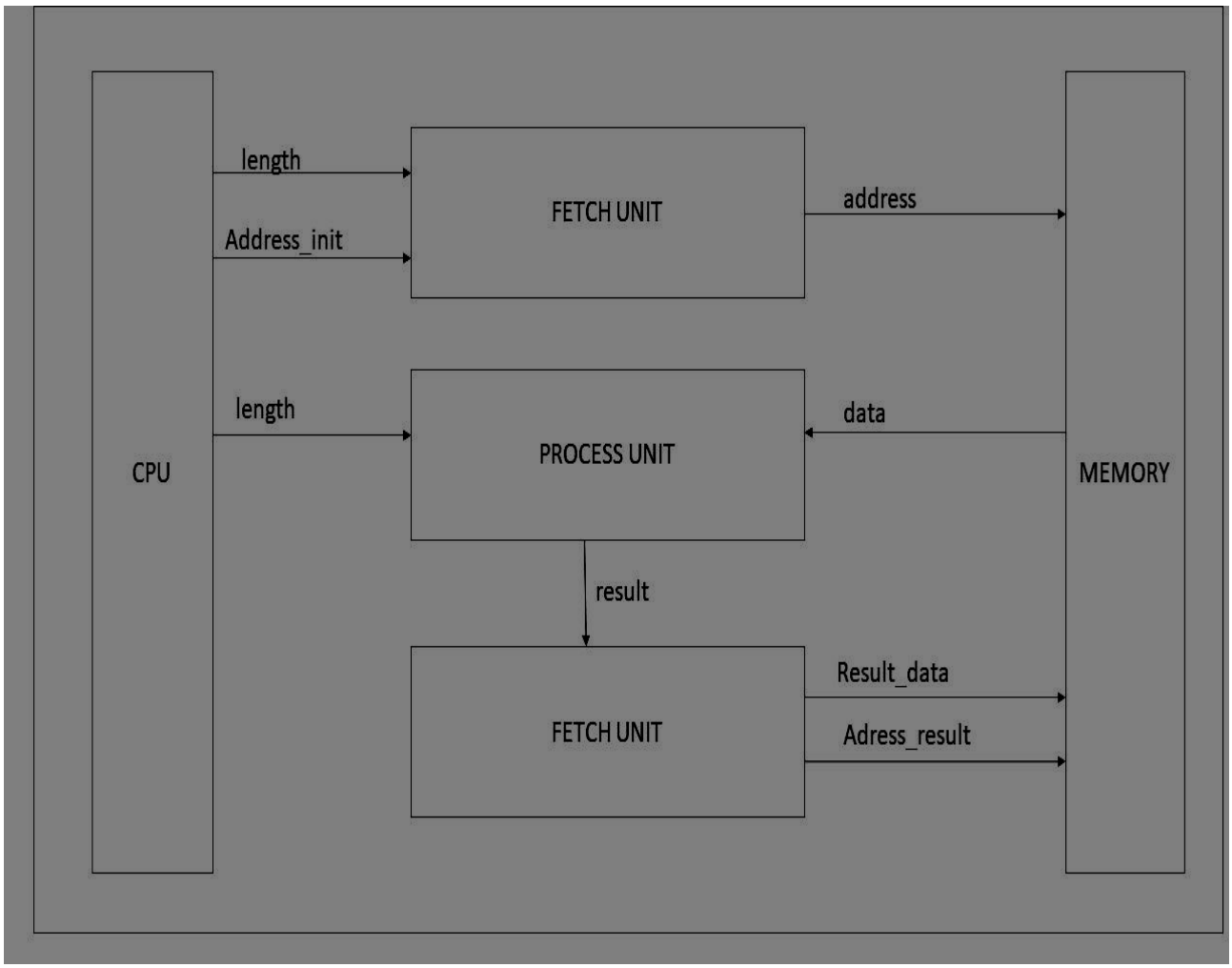
5.4 Περιγραφή 4ης υλοποίησης

Στην συγκεκριμένη υλοποίηση έγινε χρήση της αρχιτεκτονικής DEAR που είχαμε συζητήσει παραπάνω. Αρχικά αφού είδαμε ότι τα αποτελέσματα τις 3ης υλοποίησης δεν είναι ικανοποιητικά επιστρέψαμε στην αρχική λογική. Πρώτα δημιουργήσαμε την συνάρτηση fetch.Η fetch είναι η συνάρτηση που έχει ως έξοδο τις διευθύνσεις μνήμης με την σειρά που χρειάζεται έτσι ώστε να μπορούμε να έχουμε πρόσβαση στην μνήμη και να πάρουμε τα δεδομένα που χρειαζόμαστε στην συνέχεια για να γίνει η επεξεργασία των δεδομένων αυτόν. Η συνάρτηση Fetch έχει ως εισόδους το μέγεθος του πίνακα που ορίζει ο χρήστης μέσω του InputFile το init_complexline το οποίο είναι η αρχική διεύθυνση μνήμης στην οποία αναφερόμαστε και τέλος μια έξοδο μια FIFO με όλες τις διευθύνσεις μνήμης που φέρνει η συνάρτηση Fetch. Για την

αποθήκευση των διευθύνσεων χρησιμοποιήσαμε έναν πίνακα τον οποίο γεμίσαμε μέσω ενός for-loop εκκινώντας από το init_complexline και αυξάνοντας κάθε φορά κατά 8 καθώς τα δεδομένα μας είναι double αριθμοί. Στην συγκεκριμένη υλοποίηση δεν χρησιμοποιήσαμε tag καθώς δεν έχουμε να κάνουμε με πραγματική μνήμη αλλά την προσομοιώνουμε μέσω του test_bench. Στο τέλος της συνάρτησης αποθηκεύουμε τις διευθύνσεις από τον πίνακα στην FIFO.

Έπειτα υλοποιήσαμε την συνάρτηση process. Η συνάρτηση process είναι υπεύθυνη για την επεξεργασία των δεδομένων που επιστρέφονται από την μνήμη έτσι ώστε να επιτευχθεί ο μετασχηματισμός fourier του πίνακα εισόδου που είναι και ο σκοπός του αλγορίθμου μας. Η συνάρτηση process έχει ως εισόδους το μέγεθος του πίνακα εισόδου, το init_addressResult το οποίο είναι η αρχική διεύθυνση μνήμης στην οποία θα αποθηκευτούν τα τελικά αποτελέσματα,*compline είναι ο πίνακας που προέκυψε μετά από την ανάγνωση από την μνήμη είναι δηλαδή τα δεδομένα μας,*cline είναι ο πίνακας με τα αποτελέσματα τα οποία θα γίνουν στην συνέχεια εγγραφή στην μνήμη. Εσωτερικά η process είναι ουσιαστικά ο solver δηλαδή περιέχει τις συναρτήσεις FFT και transpose καθώς και τα κομμάτια κώδικα που απομονώνουν μια γραμμή από τον πίνακα καθώς όπως είπαμε νωρίτερα η ο μετασχηματισμός fourier τουλάχιστον στον συγκεκριμένο αλγόριθμο γίνεται γραμμή γραμμή.

Όπως αναφέραμε και παραπάνω στην συγκεκριμένη υλοποίηση τον ρόλο της μνήμης τον έπαιξε το softwate. Έτσι στο testbench μετά το κάλεσμα της συνάρτησης Fetch δημιουργήσαμε έναν αλγόριθμο μετατρέπει την έξοδο της Fetch τις διευθύνσεις δηλαδή μνήμης σε δεδομένα από τον πίνακα εισόδου. Με το τρόπο αυτό εξετάσαμε αν η συνάρτηση Fetch λειτουργεί με τον σωστό τρόπο και τέλος όπως και με τις προηγούμενες υλοποιήσεις μέσω της συνάρτησης επαλήθευσης ελέγξαμε αν η process επιστρέφει τα σωστά αποτελέσματα. Τέλος τρέξαμε το synthesis για να ελέγξουμε τους πόρους που χρησιμοποιεί ο αλγόριθμος μας καθώς και το RTL για να εξετάσουμε την ταχύτητα του.

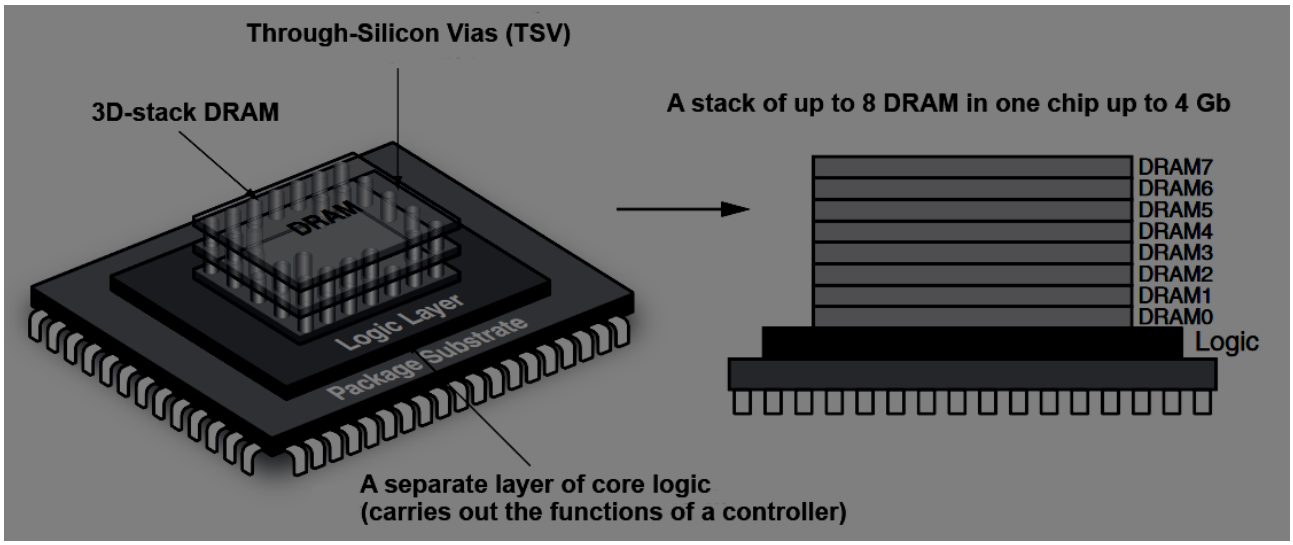


Εικόνα 5: Εφαρμογή της αρχιτεκτονικής DAER στον spectral αλγόριθμο

5.5 5η υλοποίηση

5.5.1 Hybrid Memory Cube (HMC)

Το HMC είναι μια τεχνολογία μνήμης η οποία αποτελείται από πολλές στρώσεις οι οποίες είναι ενωμένες μεταξύ τους χρησιμοποιώντας through-silicon via (TSV) τεχνολογία. Τα πάνω στρώματα αποτελούνται από Drams ενώ στο κάτω στρώμα βρίσκεται ένας controller που ελέγχει τα μεταφερόμενα δεδομένα. Η παρακάτω εικόνα δείχνει την εσωτερική δομή ενός HMC chip.

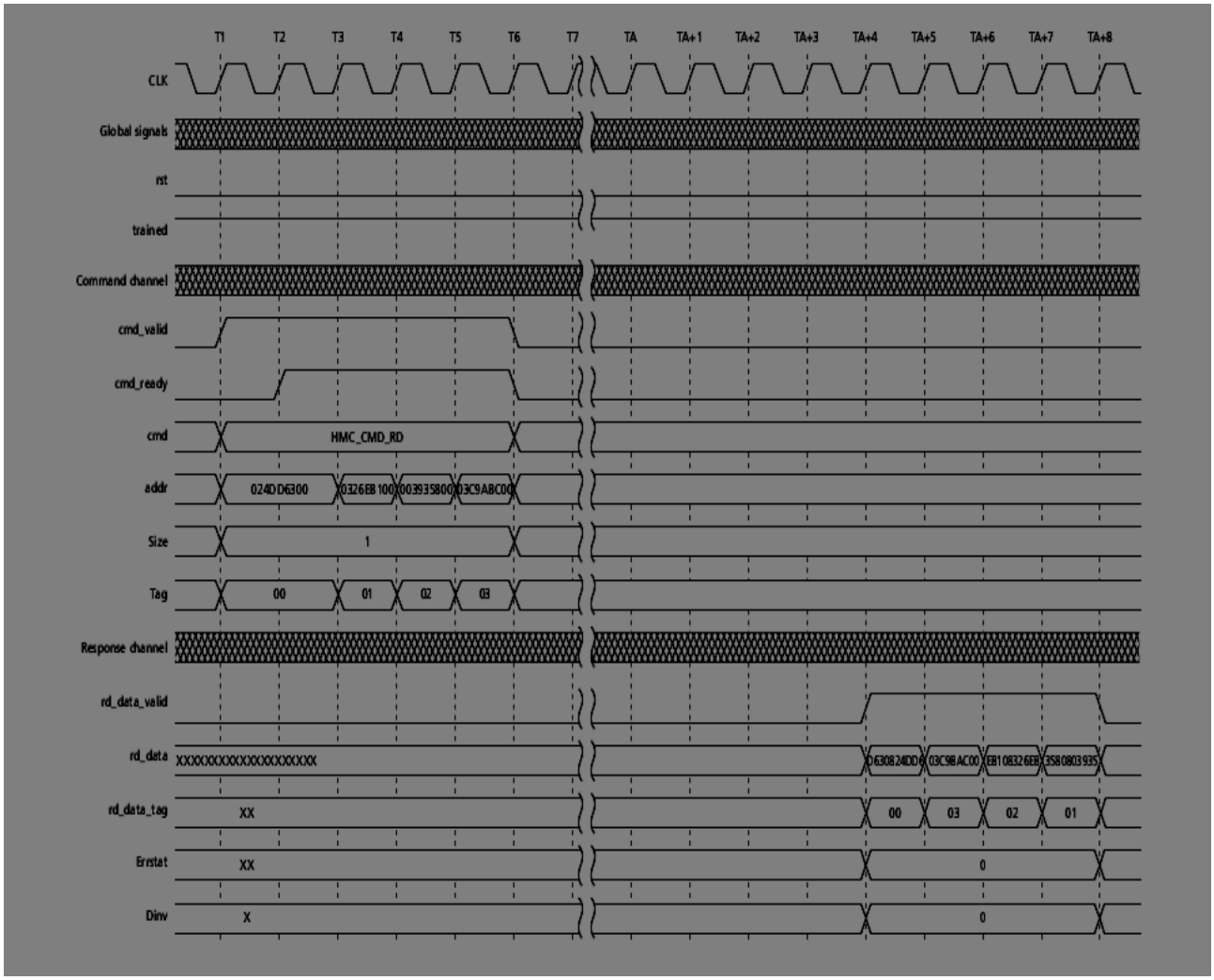


Εικόνα 6: Hybrid Memory Cybe

Το HMC χρησιμοποιείται εκεί όπου η ταχύτητα και ο μικρός αριθμός chip είναι αναγκαίος. Τα chips του HMC μπορούν να συνδυαστούν σε μια αλυσίδα έως και 8.Τα chips είναι διαθέσιμα σε χωρητικότητα των 2GB και 4GB.Τα δεδομένα μεταφέρονται μέσω σειριακής διεπαφής στην ταχύτητα των 15Gbits/s ανά γραμμή, ο συνολικός αριθμός των γραμμών μπορεί να είναι απο 32-64 έτσι το θεωρητικό bandwith που μπορεί να πιάσει το HMC είναι 240Gbits/s αλλά περιορίζεται από το bandwith της DRAM στα 160Gbits/s. Στον παρακάτω πίνακα φαίνεται η κατανάλωση ανά bit

Technology	VDD	IDD	BW GB/ s	Power (W)	mW/ GB/ s	pj/ bit	real pJ/ bit
SDRAM PC133 1GB Module	3.3	1.50	1.06	4.96	4664.97	583.12	762
DDR-333 1GB Module	2.5	2.19	2.66	5.48	2057.06	257.13	245
DDRII-667 2GB Module	1.8	2.88	5.34	5.18	971.51	121.44	139
DDR3-1333 2GB Module	1.5	3.68	10.66	5.52	517.63	64.70	52
DDR4-2667 4GB Module	1.2	5.50	21.34	6.60	309.34	38.67	39
HMC, 4 DRAM w/ Logic	1.2	9.23	128.00	11.08	86.53	10.82	13.7

Στο σημείο αυτό κρίνεται χρήσιμο να εξηγήσουμε τον τρόπο με τον οποίο το HMC αντιμετωπίζει τις αιτήσεις ανάγνωσης για να γίνει πιο κατανοητή η υλοποίηση μας. Κάθε αίτηση για ανάγνωση παράγει μία απάντηση ανάγνωσης.Το HMC αλλάζει την σειρά των αιτήσεων έτσι ώστε να είναι όσο το δυνατόν πιο αποδοτικό. Αυτό έχει ως αποτέλεσμα οι απαντήσεις ανάγνωσης να φτάσουν στην πόρτα ανάγνωσης σε διαφορετική σειρά από αυτήν την οποία αιτηθήκαμε. Για να λυθεί αυτό το πρόβλημα ο ελεγκτής συμπεριλαμβάνει το πεδίο tag το οποίο συνδέει τα δεδομένα προς ανάγνωση με τις αρχικές αιτήσεις. Για να ξεκινήσουμε μια εντολή ανάγνωσης θα πρέπει να δώσουμε στο HMC την διεύθυνση ανάγνωσης (16-byte alligned),το μέγεθος ανάγνωσης σε μονάδες των 16-byte λέξεων δηλαδή αν θέλουμε να κάνουμε ανάγνωση μιας λέξης 64bit το size θα πρέπει να είναι 4,καθώς και το μοναδικό tag που συνοδεύει την συγκεκριμένη διεύθυνση.



Εικόνα 7: Ανάγνωση στο Hybrid Memory Cube

Η παραπάνω εικόνα δείχνει μια σειρά από αιτήσεις ανάγνωσης καθώς και τα δεδομένα που επιστρέφονται. Μετά από κάποια καθυστέρηση η οποία εξαρτάται από το φόρτο εργασίας του HMC, ο ελεγκτής επιστρέφει τα δεδομένα από τις αιτήσεις ανάγνωσης. Όπως αναφέραμε λοιπόν παραπάνω ενώ στις αιτήσεις αναγνώσεις η σειρά των διευθύνσεων έχουν tag 0,1,2,3 τα δεδομένα που επιστρέφονται από το HMC έχουν tag 0,3,2,1.Όπως γίνεται λοιπόν κατανοητό όταν επιστρέφονται τα δεδομένα προς ανάγνωση θα πρέπει να τα επαναφέρουμε στην σειρά που ζητήθηκα για τον λόγω αυτό δημιουργήσαμε μια συνάρτηση την οποία θα συζητήσουμε αργότερα.

Με τον ίδιο τρόπο λειτουργεί και το write command στο HMC με την μόνη διαφορά ότι στο wrte θα πρέπει να δώσουμε στο πρόγραμμα και την τιμή που θέλουμε να γίνει εγγραφή στην διεύθυνση επιλογής μας.

5.5.2 Pico Computing Application Programming Interfaces

Για περαιτέρω λειτουργίες του αλγορίθμου μας εκτός από την μνήμη χρησιμοποιήσαμε το Pico Computing Application Programming Interfaces (Pico computing APIs) for linux platforms.To Pico computing Api προσφέρει μια σύνδεση μεταξύ την εφαρμογής software που τρέχει στον υπολογιστή και τον αλγόριθμο hardware που έχει υλοποιηθεί στην FPGA. Το pico api παρέχει στον χρήστη μια σειρά απο βιβλιοθήκες και συναρτήσεις έτσι ώστε να μπορεί να γίνεται έλεγχος της FPGA και της όλης διαδικασίας. Για τη παρούσα διπλωματική χρησιμοποιήσαμε κάποια μοντέλα από το Pico api

Stream model είναι μια μέθοδος επικοινωνίας του hardware με το software η δομή τους είναι βασισμένη στις FIFO. Το stream είναι μια ακολουθία από αριθμούς με έλεγχο της ροής. Το pico api σου δίνει την δυνατότητα να γνωρίζεις πότε το stream είναι διαθέσιμο για ανάγνωση πότε είναι διαθέσιμο να δεχτεί η να στείλει δεδομένα. Υπάρχουν δύο ειδών steams inputs στην FPGA και output στην FPGA. Όσο αναφορά το firmware το steam έχει τις ακόλουθες μεταβλητές

clk	*input*	free running clock for all the streams
rst	*input*	active-high reset signals for the streams

The following signals constitute stream #1 coming into the FPGA:

sli_valid	*input*	high when this stream has data to provide to your module
sli_rdy	*output*	drive this high when you latch the current sli_data value
sli_data[127:0]	*input*	the current data value. valid when sli_valid is high

And the signals for stream #1 going out of the FPGA:

slo_valid	*output*	drive this high when you have valid data on slo_data
slo_rdy	*input*	high when the stream is ready to accept
slo_data[127:0]	*output*	the value you're putting out on the stream

και όσο αναφορά τις συναρτήσεις του stream model στο software είναι
virtual ınt CreateStream (int streamNum) ; όπου είναι η συνάρτηση δημιουργίας του stream και εγγραφή και ανάγνωση του stream έχουμε τις συναρτήσεις

virtual int ReadStream (int streamHandle , void ∗ buf , int numBytes) ;

virtual int WriteStream (int streamHandle , const void ∗buf , int numBytes) ;

η πρώτη παράμετρος των δύο συναρτήσεων είναι η έξοδος της createStream, δεύτερη είναι ο πίνακας που θέλεις να αποθηκευτούν τα αποτελέσματα της εκάστοτε εντολής και η τρίτη παράμετρος είναι ο αριθμός των bytes που θέλεις να εγγραφούν ή να αναγνωστούν αντίστοιχα από το stream.

Το **picomemory** είναι μια εφαρμογή που βοηθάει τον χρήστη στην επικοινωνία με την μνήμη είδαμε παραπάνω πως γίνεται η εγγραφή και η ανάγνωση της μνήμης του HMC για να έχουμε όμως πρόσβαση στη μνήμη απο το software χρησιμοποιούμε τις συναρτήσεις που μασ παρέχει το picomemory

int WriteRam (uint64_t addr , const void \∗ buf , int size , int memID = PICO_DDR3_0)

η writeRam δέχεται σαν ορίσματα της διεύθυνση, έναν buffer από τον οποίο θα πάρει τα δεδομένα που είναι προς εγγραφή το μέγεθος της μνήμης και ένα memID το οποίο δεν είναι υποχρεωτικό και αν δεν μπει από τον χρήστη θέτεται DDR3 0.

int ReadRam (uint64_t addr , void \∗ buf , int size , int memID = PICO_DDR3_0)

αντίστοιχα με την WriteRam η ReadRam δέχεται σαν ορίσματα την διεύθυνση, έναν buffer στον οποίο θα αποθήκευση τα δεδομένα που είναι για ανάγνωση το μέγεθος και ένα προεραιτικό memID.

Κατά την διάρκεια της υλοποίησης χρησιμοποιήθηκαν και άλλες συναρτήσεις του pico appi που ήταν χρήσιμες στο debugging του αλγορίθμου όπως η getAvailiableBytes() που δίνει ως έξοδο την συγκεκριμένη χρονική στιγμή που καλείται πόσα Bytes είναι διαθέσιμα για διάβασμα από το κάποιο stream.

5.5.3 Περιγραφή 5ης υλοποίησης

Στην 5η υλοποίηση χρησιμοποιήσαμε HMC (hybrid memory cube) και ο προγραμματισμός του έγινε μέσω του εργαλείου vivado. Στην παρούσα διπλωματική χρησιμοποιήσαμε το HMC της micron το οποίο έχει στην κατοχή του το πολυτεχνείο Κρήτης. Η πρόσβαση σε αυτό έγινε απομακρυσμένα μέσω server του πολυτεχνείου στον οποίο είχαμε πρόσβαση.

Αρχικά το vivado μας δίνει την δυνατότητα να μεταφέρουμε τις συναρτήσεις που δημιουργήσαμε στο hls την fetch δηλαδή και την process.Η συνάρτηση fetch είναι η συνάρτηση η οποία μας φέρνει τις διευθύνσεις και τα tags των διευθύνσεων που χρειαζόμαστε για τις προσβάσεις στην μνήμη του HMC.Η συνάρτηση **Fetch** δέχεται ως είσοδο το μέγεθος του πίνακα που θέλουμε να μετασχηματίσουμε, την αρχική τιμή της διεύθυνσης μνήμης η οποία είναι μεγέθους 34bits, ένα πίνακα από αριθμούς των 40bit οι οποίοι εμπεριέχουν την διεύθυνση που θέλουμε να γίνει η ανάγνωση στα πρώτα 34bits και στα (39,34) το tag αυτής της διεύθυνσης, και τέλος ένα πίνακα με αριθμούς 6bit που είναι το tag. Στην συνάρτηση στην αρχή δημιουργούμε 2 FIFO την tagadr_complexline και την tagcomplexline που θα είναι και οι έξοδοι της συνάρτησης μας

```
#pragma HLS INTERFACE ap_fifo depth=1024 port= tagadr_complexLine
#pragma HLS INTERFACE ap_fifo depth=1024 port= tagcomplexline
```

και στην συνέχεια με την βοήθεια ενός for loop φτιάχνουμε τις διευθύνσεις μνήμης που θέλουμε να έχουμε πρόσβαση αυξάνοντας κατά 32 κάθε φορά για να τηρούμε το πρωτόκολλο του HMC που αναφέραμε παραπάνω. Επίσης αυξάνουμε κάθε φορά το tag κατά 1 έτσι ώστε να είναι εύκολο η εύρεση του αργότερα. Τέλος τα αποθηκεύουμε στις δύο FIFO που δημιουργήσαμε.

```
for(i=0; i<2*length*8; i +=32)
        {
        #pragma HLS pipeline II=1
                idx1 = init_complexLine + i;
                *tagcomplexline = tag;
                tagcomplexline++;
                tagadr_complexLine->range(39,34) = tag;
                tagadr_complexLine->range(33,0) = idx1;
                tagadr_complexLine++;
                tag++;
        }
```

Για του λόγους που αναφέραμε παραπάνω σχετικά με τον τρόπου που γίνεται το read στο HMC χρειάστηκε να φτιάξουμε μια συνάρτηση η οποία αφού επιστρέψουν οι τιμές απο την μνήμη του HMC θα μπορεί να τις βάλει στην σειρά με την οποία εμείς ζητήσαμε. Η συνάρτηση αυτή είναι η

fetch_reorder η οποία δέχεται σαν είσοδο το μέγεθος του πίνακα, έναν πίνακα με 134bit αριθμούς όπου τα (127,0)bit είναι τα δεδομένα που επιστρέφει το HMC και τα(133,128) είναι τα Bit του tag. Δέχεται επίσης έναν πίνακα με 6bit αριθμούς που είναι το tag των διευθύνσεων που ζητήσαμε για ανάγνωση δηλαδή με την σωστή σειρά και τέλος έναν πίνακα με 64Bit αριθμούς που είναι και τα τελικά δεδομένα μας με την σωστή σειρά και είναι έτοιμα για επεξεργασία από την process.
Στην αρχή της συνάρτησης όπως και σε κάθε συνάρτηση που υλοποιούμε στον accelerator μετατρέπουμε τους πίνακες σε FIFO

```
#pragma HLS INTERFACE ap_fifo depth=512 port=tagDataComplexline
#pragma HLS INTERFACE ap_fifo depth=512 port=tagComplexline
#pragma HLS INTERFACE ap_fifo depth=512 port=complexline_f
```

Ουσιαστικά σε αυτήν την συνάρτηση συγκρίνουμε την σειρά που επιστρέφει τα tag το HMC με την σειρά των tag που εμείς ζητήσαμε γνωρίζοντας ότι εμείς τα tag τα ζητάμε σειριακά, με αυτό τον τρόπο ανακατανέμουμε τα δεδομένα που επιστρέφει το HMC στην σωστή σειρά.

```
for(i =0; i <2*length; i+= 4)
        {
                ap_uint<6> incomingTag;
                ap_uint<8> Rindex;
                incomingTag = tagDataComplexline->range(133,128);
                Rindex = incomingTag;
                Rindex = Rindex <<2;
                dataBuffer[Rindex + 0] = tagDataComplexline->range(63,0);
                dataBuffer[Rindex + 1] = tagDataComplexline->range(127,64);
                tagDataComplexline++;
                dataBuffer[Rindex + 2] = tagDataComplexline->range(63,0);
                dataBuffer[Rindex + 3] = tagDataComplexline->range(127,64);
                tagDataComplexline++;
                /datavalid.bit(incomingTag) = 1;
                while (datavalid.bit(outgoingTag)){
                #pragma HLS PIPELINE
                        ap_uint<8> Windex = outgoingTag;
                        Windex = Windex<<2;
                        *complexline_f = dataBuffer[Windex + 0];
                        complexline_f++;
                        *complexline_f = dataBuffer[Windex + 1];
```

```
            complexline_f++;
            *complexline_f = dataBuffer[Windex + 2];
            complexline_f++;
            *complexline_f = dataBuffer[Windex + 3];
            complexline_f++;

            datavalid.bit(outgoingTag) = 0;
            outgoingTag=*tagComplexline;
            tagComplexline++;
        }
```

Τέλος αποθηκεύουμε στην FIFO τα δεδομένα με την σωστή σειρά τα οποία είναι πλέον έτοιμα για επεξεργασία από την συνάρτηση Process.

Η συνάρτηση **process** δεν έχει αλλάξει από την προηγούμενη υλοποίηση, σε αυτήν ουσιαστικά εμπεριέχονται οι συναρτήσεις **FFT** και **transpose** που είναι υπεύθυνες για τον μετασχηματισμό Fourier του πίνακα μας.
Αφού λοιπόν ολοκληρώσαμε την σύνταξη των συναρτήσεων στο vivado_hls και περάσουν όλες από το synthesis test τότε μπορούμε να τις μεταφέρουμε στο vivado αφού το hls μας δίνει αυτή την δυνατότητα. Αφού λοιπόν τις μεταφέραμε με επιτυχία στην συνέχεια δημιουργήσαμε τις απαραίτητες FIFO οι οποίες χρησιμοποιούνται από τις συγκεκριμένες συναρτήσεις είτε ως είσοδος είτε ως έξοδος. Στην συνέχεια όπως γίνεται κατανοητό θα πρέπει να ενώσουμε κατάλληλα αυτές τις συναρτήσεις έτσι ώστε να δημιουργήσουμε το core μας που θα είναι υπεύθυνο για την λύση του αλγορίθμου μας. Η σύνδεση αυτή έγινε μέσω του port map της vivado.

Στο δεύτερο κομμάτι της υλοποίησης στο Vidado δημιουργήσαμε αρχικά έναν ελεγκτή ο οποίος είναι υπεύθυνος για τον έλεγχο των σημάτων των συναρτήσεων fetch,fetch_reorder,process,τα σήματα ελέγχου αυτά είναι το ap_start το οποίο δηλώνει πότε η συνάρτηση εκκινεί,και το ap_done το οποίο δινώνει πότε η συνάρτηση έχει ολοκρηρωθεί. Ορίσαμε μια μεταβλητή global_start η οποία δηλώνει την εκκίνηση του συνολικού προγράμματος την οποία την ενεργοποιεί ο χρήστης μέσω του software. Δημιουργήσαμε επίσης έναν μετρητή για να μπορέσουμε με μετρήσουμε τον χρόνο εκτέλεσης του προγράμματος για να μπορέσουμε στο τέλος να συγκρίνουμε τις υλοποιήσεις μεταξύ τους. Ο μετρητής αυτός ξεκινάει να μετράει απο την στιγμή που το πρόγραμμα εκκινεί μέχρι που και ο τελευταίος done reg γίνει 1 που σημαίνει ότι το πρόγραμμα ολοκληρώθηκε.

```
always @(posedge pcie_clk) begin
    if (rst) begin
      clkCnt <=  #`dh 64'b0;
    end else begin
      if(doneReg == 3'b111)
        performance <=  #`dh clkCnt;
      else
        clkCnt <=  #`dh clkCnt + 64'b1;
    end
  end
```

Στο συγκεκριμένο module χρησιμοποιήσαμε επίσης ένα μοντέλο του Pico appi, πιο συγκεκριμένα χρησιμοποιήσαμε δυο stream ένα input stream και ένα Output stream. Το input stream το χρησιμοποιήσαμε για να μπορεί ο χρήστης να μας παρέχει μέσω του software τιμές για όλες τις απαραίτητες μεταβλητές για την εκτέλεση του προγράμματος. Το stream περιέχει τιμές των 128bit επειδή το πρόγραμμα δεν μπορεί να καταλάβει σε ποια μεταβλητή αναφερόμαστε χρησιμοποιήσαμε τα 4LSB ως κωδικό για να δηλώσουμε την μεταβλητή που θέλουμε, έτσι αν παραδείγματος χάριν θέλουμε να δώσουμε τιμή στο μέγεθος του πίνακα τα 4LSB θα ήταν 0001 αντίστοιχα και για τις υπόλοιπες μεταβλητές

```
if (s1i_valid) begin
        if(s1i_data[127:124] == 4'h1) begin
          length_r                <= #`dh s1i_data[31:0];
        end else if (s1i_data[127:124] == 4'h2) begin
          init_complexLine_V  <= #`dh s1i_data[97:64];
        end else if (s1i_data[127:124] == 4'h3) begin
          init_addressResult  <= #`dh s1i_data[63:0];
        end else if (s1i_data[127:124] == 4'h4) begin
          global_start        <= #`dh s1i_data[64];
```

Το output stream με την σειρά του παρέχει πληροφορίες στον χρήστη για την εξέλιξη του προγράμματος δηλαδή τι τιμή έχουν τα σήματα ελέγχου των συναρτήσεων την ώρα που ζητείται καθώς και για τον τελικό χρόνο εκτέλεσης.

```
        s1o_data[63:0] <= #`dh performance;
      s1o_data[64] <= #`dh ap_start_f;
        s1o_data[65] <= #`dh ap_ready_f;
        s1o_data[66] <= #`dh ap_idle_f;
```

```
s1o_data[67] <= #`dh ap_done_f;
s1o_data[68] <= #`dh ap_start_f_reorder;
s1o_data[69] <= #`dh ap_ready_f_reorder;
s1o_data[70] <= #`dh ap_idle_f_reorder;
s1o_data[71] <= #`dh ap_done_f_reorder;
s1o_data[72] <= #`dh ap_start_p;
s1o_data[73] <= #`dh ap_ready_p;
s1o_data[74] <= #`dh ap_idle_p;
s1o_data[75] <= #`dh ap_done_p;
s1o_data[76] <= #`dh global_start;
s1o_data[127:77] <= #`dh 31'b0;
```

Στο τελευταίο module (Top_level) αρχικά ενώσαμε τα δυο προηγούμενα Module μεταξύ τους, το core δηλαδή που περιέχει τις συναρτήσεις για την εκτέλεση του αλγορίθμου και τον ελεγκτή register_file για να μπορούν να επικοινωνούν μεταξύ τους. Στην συνέχεια ορίσαμε κάποια μεγέθη για το HMC πρώτα ορίσαμε το πλάτος της διεύθυνσης σε 34bit όπως δηλαδή το έχουμε ορίσει και στις προηγούμενες συναρτήσεις μας το μέγεθος που επιστρέφει σε 128bit πράγμα που σημαίνει αφού τα αποτελέσματα μας είναι double ότι θα επιστρέφει δυο λέξεις και θα πρέπει εμείς στο software να τις χωρίζουμε, και τέλος το πλάτος του tag=6.'Οπως είπαμε και παραπάνω η εγγραφή στην μνήμη αλλά και η ανάγνωση γίνεται μέσω του HMC. Σε αυτό το module συνδέσαμε κατάλληλα τα σήματα μας με τα σήματα του HMC έτσι ώστε να γίνεται σωστά η ανάγνωση και η εγγραφή από το HMC. Στην περίπτωση του read το σήμα *hmc_cmd* το ορίσαμε HMC_CMD_RD, το *hmc_addr* ίσο με τα 33MSB της εξόδου της συνάρτησης fetch, το *hmc_tag ίσο μετα 6LSB της εξόδου της συνάρτησης fetch,το hmc_valid το ορίσαμε ως το σήμα ~empty της FIFO,wr_data 0 αφού δεν θέλουμε να κάνουμε εγγραφή ομοίως και το wr_data_valid*

```
wire [39:0] tagadr_complexLine_V_dout;
wire  tagadr_complexLine_V_empty;
wire  tagadr_complexLine_V_read;
wire [133:0] tagDataComplexline_V_din;
wire  tagDataComplexline_V_write;
assign hmc_clk_p0 = hmc_tx_clk;
assign hmc_cmd_valid_p0 = ~tagadr_complexLine_V_empty;
assign tagadr_complexLine_V_read  =  hmc_cmd_ready_p0;
assign hmc_cmd_p0       = `HMC_CMD_RD;
assign hmc_addr_p0      = tagadr_complexLine_V_dout[33:0];
```

```
assign hmc_size_p0      = 4'h2;
assign hmc_tag_p0       = tagadr_complexLine_V_dout[39:34];
assign hmc_wr_data_p0   = 128'b0;
assign hmc_wr_data_valid_p0 = 1'b0;
assign tagDataComplexline_V_din = {hmc_rd_data_tag_p0, hmc_rd_data_p0};
assign tagDataComplexline_V_write = hmc_rd_data_valid_p0
```

ομοίως με την ανάγνωση από το HMC πράττουμε για την εγγραφή.

```
assign hmc_clk_p2 = hmc_tx_clk;
αssign hmc_cmd_valid_p2 = ~address_Result_empty;
assign address_Result_read = hmc_cmd_ready_p2;
assign hmc_cmd_p2 = `HMC_CMD_BW;
assign hmc_addr_p2 = address_Result_dout[33:0];
assign hmc_size_p2 = 4'h2;
assign hmc_tag_p2 = 6'b0;
assign hmc_wr_data_p2 = {64'b0, cLine_dout};
assign hmc_wr_data_valid_p2 = ~cLine_empty;
assign cLine_read = hmc_wr_data_ready_p2;
```

Αφού ολοκληρώσαμε το implementation στο vivado τρέξαμε το synthesis και στην συνέχεια κατεβάσαμε τον .bit αρχείο. Για να εξετάσουμε τα αποτελέσματα του αλγορίθμου μας έπρεπε να δημιουργήσουμε έναν κώδικα σε software σε γλώσσα c++.Όπως αναφέραμε και παραπάνω το pico appi μας παρέχει μια σειρά από βιβλιοθήκες με συναρτήσεις για τα μοντέλα που χρησιμοποιήσαμε στην υλοποίηση μας. Αρχικά χρησιμοποιήσαμε την συνάρτηση ***RunbitFile*** η οποία όπως είναι προφανές και από το όνομα της τρέχει το .bit αρχείο μας. Έπειτα ορίσαμε έναν πίνακα από 64Bits αριθμούς το γεμίσαμε με αριθμούς ο πίνακας αυτός είναι ο πίνακας με τα δεδομένα που θα εισάγουμε στο HMC για αυτό το λόγω τα δεδομένα δεν είναι τυχαία αλλά ίδια με εκείνα που είχαμε χρησιμοποιήσει στο HMC για να μπορούμε στην συνέχεια να συγκρίνουμε αν τα αποτελέσματα του αλγορίθμου μας είναι σωστά. Στην συνέχεια χρησιμοποιώντας την συνάρτηση ***CreateStream()*** δημιουργήσαμε το stream που χρησιμοποιήσαμε στον ελεγκτής μας για μεταφέρουμε χρήσιμες πληροφορίες. Αφού λοιπόν δημιουργήσαμε το stream θα κάνουμε εγγραφή μέσω της ***Writestream()*** για να δηλώσουμε τις μεταβλητές του προγράμματος μας

```
cfgVar[0] = length;
cfgVar[1] = ((uint64_t)1) << 60;
err = pico->WriteStream(stream1, cfgVar, 16);
cfgVar[0] = 0;
```

```
cfgVar[1] = ((uint64_t)2) << 60;
err = pico->WriteStream(stream1, cfgVar, 16);
cfgVar[0] = 0;
cfgVar[1] = ((uint64_t)3) << 60;
err = pico->WriteStream(stream1, cfgVar, 16);
```

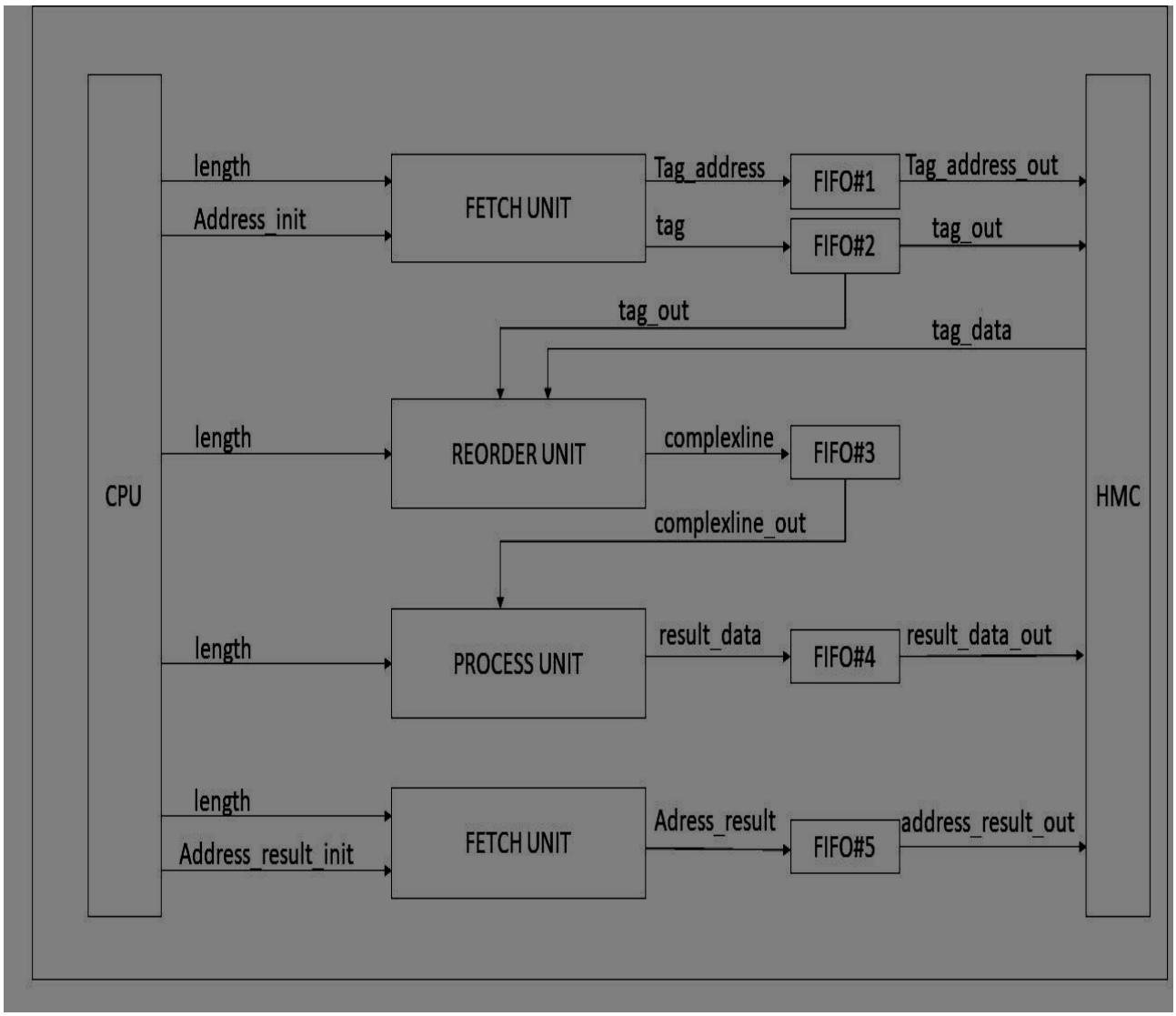
όπως αναφέραμε και στο implementation ανάλογα με τα 4LSB δηλώνουμε και την αντίστοιχη μεταβλητή. Στην συνέχεια χρησιμοποιούμε την συνάρτηση WriteRam() για να δώσουμε στην μνήμη μας τις τιμές που θέλουμε.

```
err = pico->WriteRam(0, Array_A, WRITE_DATA_SIZE, 0);
```

Αφού λοιπόν τελειώσαμε με τις αρχικοποιήσεις δίνουμε μέσω του stream στο global_start την τιμή 1 και το πρόγραμμα ξεκινάει.Όταν το πρόγραμμα τελειώσει ζητάμε μέσω της readstream να δουμε τον χρόνο εκτέλεσης καθώς και τα σήματα των συναρτήσεων μας,καθώς και μέσω της readram τα τελικά αποτελέσματα.

```
err = pico->ReadStream(stream1, cfgVar, 16);
err = pico->ReadRam(0, Array_A, WRITE_DATA_SIZE, 0);
```

Τέλος επειδή αυτές οι συναρτήσεις αποθηκεύουν τα αποτελέσματα στον πίνακα που έχουν τα ορίσματα τους κάνουμε εμφάνιση των πινάκων αυτών.



Εικόνα 8: Datapath 5ης υλοποίησης στο HMC

Κεφαλαιο 6

Αποτελέσματα

Επισκόπηση

Στο κεφάλαιο αυτό θα παρατεθούν οι χρόνοι εκτέλεσης τόσο στο hardware όσο και στο software. Σκοπός μας είναι η σύγκριση των χρόνων αυτών και να καθίσταται εμφανής η βελτίωση της απόδοσης του προγράμματος. Τέλος, θα παρατεθούν οι πόροι που χρησιμοποιήθηκαν σε κάθε αλγόριθμο.

6.1 Περιγραφή πλατφόρμων και εργαλείων

Το εργαλείο που χρησιμοποιήσαμε για την απεικόνιση των αλγορίθμων που υπάγονται στα δομημένα και μη πλέγματα είναι το Vivado HLS. Το εργαλείο αυτό αποτελεί αυτοματοποιημένη
μετατροπή μιας σχεδίασης από C, C++, SystemC σε RTL υλοποίηση η οποία με τη σειρά της γίνεται σύνθεση σε μια fpga. Αυτό, καθιστά τη ζωή του προγραμματιστή ευκολότερη, εφόσον η σχεδίαση του RTL αρχείου δεν γίνεται χειροκίνητα, κάτι το οποίο μάλιστα, ελαχιστοποιεί και την πιθανότητα των λαθών. Η σύνθεση υψηλού επιπέδου δέχεται σαν είσοδο μια συνάρτηση υλοποιημένη σε C/C++ καθώς και ένα αρχείο test bench το οποίο έχει δημιουργηθεί για την επαλήθευση της ορθής λειτουργίας του συστήματος. Το αρχείο αυτό αποθηκεύει όλα τα αποτελέσματα τα οποία συγκρίνονται με τα αποτελέσματα εξόδου της συνάρτησης. Μετά την επαλήθευση της σωστής λειτουργίας του προγράμματος ακολουθεί η διαδικασία της σύνθεσης σε μία FPGA. Στο στάδιο αυτό παρέχεται η δυνατότητα στο χρήστη να πειραματιστεί ανάλογα με τις προδιαγραφές που επιθυμεί να έχει το πρόγραμμα και να χρησιμοποιήσει και τις κατάλληλες directives. Τέλος, ελέγχεται η ορθότητα του αρχείου RTL.
Το εργαλείο της Vivado εμπεριέχει διάφορες FPGAs με διαφορετικά χαρακτηριστικά. Το rtl αρχείο σχετίζεται με την επιλογή της πλατφόρμας. Στη συγκεκριμένη διπλωματική, η FPGA που επιλέξαμε είναι η Kintex UltraScale ffva1156. Η πλατφόρμα εμπεριέχει 2760 DSP Slices, 331680 LUTs, 663360 Flip Flops, 2160 BRAMs. Η επιλογή αυτή έγινε με γνώμονα την εσωτερική μνήμη της συγκεκριμένης πλατφόρμας διότι, επιθυμούμε όσο το δυνατόν

μεγαλύτερη. Όσο το δυνατόν μεγαλύτερη εσωτερική μνήμη τόσο μεγαλύτερο και το dataset το οποίο θα χρησιμοποιηθεί.

Μερικά από τα πλεονεκτήματα αλλά και τα χαρακτηριστικά της συγκεκριμένης FGPA περιγράφονται στον παρακάτω πίνακα.

Value	Deliverables
Programmable System Integration	• Up to 1.5M System Logic Cells leveraging 2nd generation 3D IC • Multiple integrated PCI Express® Gen3 cores
Increased System Performance	• 8.2 TeraMACs of DSP compute performance • Up to two speed-grade improvement with high utilization • 16G backplane-capable transceivers, up to 64 per device • 2,400Mb/s DDR4 for robust operation over varying PVT
BOM Cost Reduction	• System integration reduces application BOM cost by up to 60% • 12.5Gb/s transceivers in slowest speed grade • 2,400Mb/s DDR4 in a mid-speed grade • VCXO integration reduces clocking component cost
Total Power Reduction	• Up to 40% lower power vs. previous generation • Fine granular clock gating with UltraScale devices ASIC-like clocking • Enhanced system logic cell packing reduces dynamic power
Accelerated Design Productivity	• Footprint compatibility with Virtex® UltraScale devices for scalability • Co-optimized with Vivado® Design Suite for rapid design closure

Εικόνα 9: Πλεονεκτήματα της Kintex UltraScale ffva1156

6.2 Χρόνοι Εκτέλεσης Του αλγορίθμου spectral

Στο κεφάλαιο αυτό θα παρατεθούν οι χρόνοι εκτέλεσης του αλγορίθμου. Αρχικά δόθηκαν διάφορα μεγέθη dataset όπου όπως καταλαβαίνουμε όσο μεγάλωνε το μέγεθος τόσο αυξανόταν και ο χρόνος εκτέλεσης. Οι χρόνοι παρατίθενται παρακάτω μαζί με ένα γράφημα στο οποίο φαίνεται η διαφορά στην απόδοση του αλγορίθμου. Αξίζει να σημειωθεί, ότι η χαρτογράφηση του αλγορίθμου στο hardware σύμφωνα με την αρχιτεκτονική DAE βελτίωσε την απόδοση του συστήματος έως και 2 φορές παραπάνω. Εν συνεχεία εκμεταλλευόμενοι την παραλληλία που παρέχεται από την αρχιτεκτονική αυτή και εκμεταλλευόμενοι και την δυνατότητα του εργαλείου της vivado να επιτρέπει στο πρόγραμμα να εκτελείται σε μορφή pipeline εκτόξευσε την απόδοση του συστήματος έως και 4,5 φορές παραπάνω. Συνολικά, κατά μέσο όρο είχε μία σταθερή βελτίωση της τάξεως του 4%.Σαφώς,η παράλληλη υλοποίηση δεν θα ήταν εφικτή εάν δεν επέτρεπε τέτοιου είδους υλοποιήσεις η fpga

Table size	**Software time**	**1η υλοποιηση**	**2η υλοποιηση**	**3η υλοποιηση**	**4η υλοποιηση**
8x8	155.000 ns	63.560 ns	32.980 ns	64.176 ns	15.289 ns
16x16	362.000 ns	298.444 ns	141.748 ns	305.640 ns	114.128 ns
32x32	835.000 ns	1.144.880 ns	831.244 ns	1.372.496 ns	426.822 ns
64x64	2.600.000 ns	4.759724 ns	2.986.455 ns	5.237.625 ns	1.496.638 ns

Εικόνα 10: Αποτελέσματα μετρήσεων στο HLS

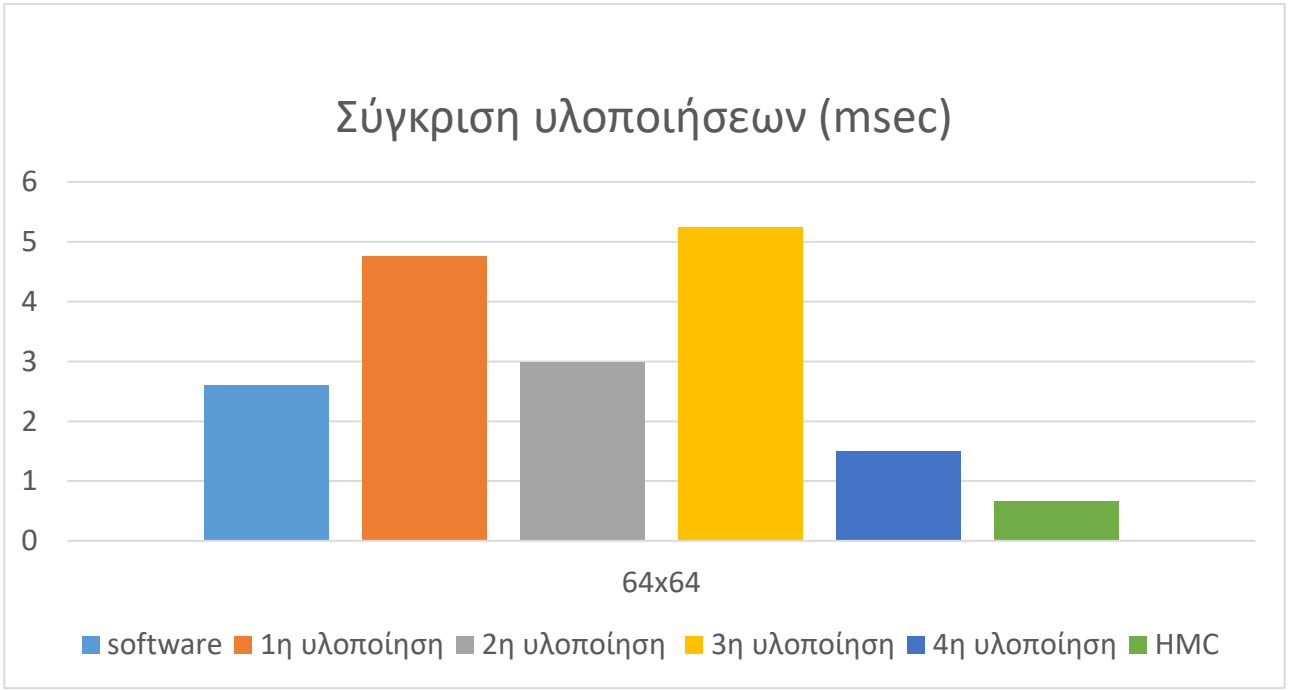
Παρατηρούμε λοιπόν ότι στην πρώτη υλοποίηση χωρίς δηλαδή περαιτέρω βελτιστοποιήσεις για μικρούς πίνακες υπάρχει αρκετά καλή βελτιστοποίηση αλλά όσο μεγαλώνουν τα datasets η διαφορά αυτή μικραίνει και στην συνέχεια το hardware γίνεται πιο αργό. Στην δεύτερη υλοποίηση όπου χρησιμοποιήσαμε directives για την βελτιστοποίηση του προγράμματος και πάλι υπάρχει βελτιστοποίηση στα μικρά datasets αλλά όσο μεγαλώνει ο όγκος των δεδομένων υπερισχύει πάλι το software.Η 3η υλοποίηση όπου χρησιμοποιήσαμε έναν ενιαίο πίνακα για τα δεδομένα μας φέρνει και τα χειρότερα αποτελέσματα ενώ όπως ήταν αναμενόμενο τα καλύτερα αποτελέσματα τα έχουμε με την χρησιμοποίηση του αλγορίθμου DAE.

6.3 Χρόνοι Εκτέλεσης Στο HMC

Στην ενότητα αυτή μετρήσαμε την απόδοση της υλοποίησης στο HMC με διαφορετικού μεγέθους dataset και συγκρίναμε τα αποτελέσματα με αυτά του software. Οι χρόνοι στο HMC έδειξαν τα αναμενόμενα αποτελέσματα, δηλαδή η υλοποίηση του αλγορίθμου με την αρχιτεκτονική DAE επέφερε βελτίωση στην απόδοση καθιστώντας την περίπου τέσσερις φορές πιο γρήγορη από εκείνη του software και από το δείγμα του μεγέθους 64x64 περίπου δύο φορές ταχύτερη από εκείνη του HLS. Πιο αναλυτικά παρακάτω παρατίθεται ο πίνακας σύγκρισης των αποδόσεων του software με το HMC.

Table size	Software time (msec)	HMC time(msec)
64x64	2,6	0,654
128x128	4,635	1,285
256x256	11,859	2,580
512x512	22,983	5,907
1024x1024	48,459	11,485

Εικόνα 11: Αποτελέσματα μετρήσεων στην πλατφόρμα HMC



6.4 Πόροι Συστήματος

Σε κάθε υλοποίηση ξεχωριστά χρησιμοποιήθηκε διαφορετικός αριθμός πόρων. Η χρήση λογικών πράξεων στη μονάδα process σε αντίθεση με τις μονάδες fetch είναι μια μεγάλη διαφορά στους πόρους όπου θα απαιτηθούν. Οι λογικές πράξεις απαιτούν την χρήση DSP. Τα δεδομένα που χρησιμοποιούνται είναι double 64 bit. Οι πράξεις του πολλαπλασιασμού και της πρόσθεσης χρειάζονται περισσότερες από μία DSP για των υπολογισμό τους. Στο παρακάτω γράφημα παρουσιάζεται το ποσοστό των πόρων που χρησιμοποιήθηκαν.

1η Υλοποίηση

	BRAM	**DSP**	**FF**	**LUT**
Total	117	161	24030	33874
Available	2160	2760	663360	331680
Utilization	5%	5%	3%	10%

Εικόνα 12: Πόροι συστήματος για την 1η υλοποίηση

2η Υλοποίηση

	BRAM	**DSP**	**FF**	**LUT**
Total	179	155	22966	32896
Available	2160	2760	663360	331680
Utilization	8%	5%	3%	9%

Εικόνα 13: Πόροι συστήματος για την 2η υλοποίηση

3η Υλοποίηση

	BRAM	DSP	FF	LUT
Total	27	177	4434	42256
Available	2160	2760	663360	331680
Utilization	1%	7%	6%	13%

Εικόνα 14: Πόροι συστήματος για την 3η υλοποίηση

4η Υλοποίηση

Fetch Unit

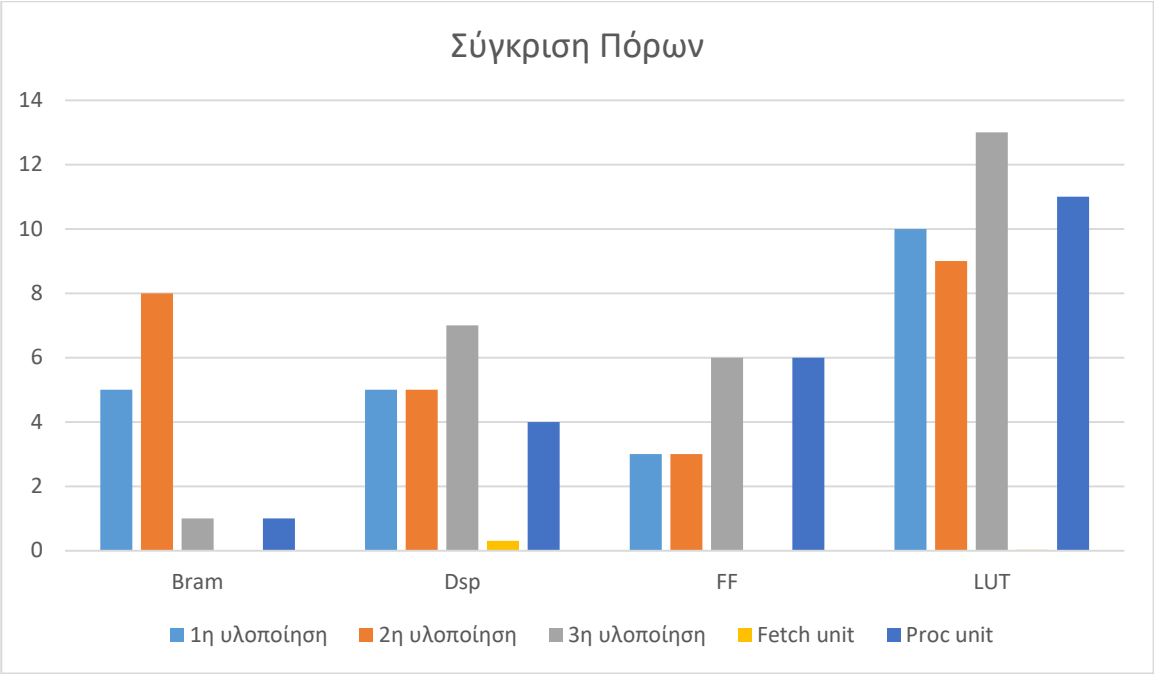
	BRAM	DSP	FF	LUT
Total	0	10	120	132
Available	2160	2760	663360	331680
Utilization	0%	0,3%	0,01%	0,03%

Εικόνα 15: Πόροι συστήματος για το Fetch unit

Process Unit

	BRAM	DSP	FF	LUT
Total	20	133	44828	35778
Available	2160	2760	663360	331680
Utilization	1%	4%	6%	11%

Εικόνα 16: Πόροι συστήματος για το Process unit


Σύγκριση Πόρων
14
12
10
8
6
4
2
0
Bram
Dsp
FF
LUT
1η υλοποίηση
2η υλοποίηση
3η υλοποίηση
Fetch unit
Proc unit

Κεφάλαιο 7

Τελικά Συμπεράσματα

7.1 Συμπεράσματα

Στη παρούσα διπλωματική παρουσιάστηκε η χρήση ενός εκ των δεκατριών νάνων συγκεκριμένα του αλγόριθμου spectral, ως benchmarks στη πλατφόρμα Kintex UltraScale ffva1156 και στο Hybrid Memory Cube του micron . Στον παραπάνω αλγόριθμο χρησιμοποιήθηκε το framework DAE. Το πρόγραμμα χωρίζεται σε δύο λειτουργικές μονάδες τη fetch και τη process. Η πρώτη είναι αρμόδια για την ανάκτηση των διευθύνσεων και η δεύτερη για το υπολογισμό του κύριου φόρτου εργασίας του αλγορίθμου. Οι διαδικασίες αυτές γίνονται παράλληλα. Η λειτουργική μονάδα process δεν περιμένει την ανάκτηση όλων των διευθύνσεων πάρα μόνο αυτών που απαιτούνται για τον υπολογισμό των λογικών πράξεων. Επομένως, παρατηρείται μεγάλη βελτίωση στην απόδοση εκμεταλλευόμενοι τα πλεονεκτήματα του παράλληλου προγραμματισμού. Προτού όμως χρησιμοποιήσουμε το DAE δημιουργήσαμε άλλες 3 υλοποιήσεις έτσι ώστε να μπορέσουμε τόσο να συγκρίνουμε τα αποτελέσματα με το software όσο και με τις άλλες υλοποιήσεις στο Hardware που δεν χρησιμοποιούν το framework DAE. Στην πρώτη υλοποίηση στην οποία δεν χρησιμοποιήσαμε κάποια μέθοδο βελτιστοποίησης παρατηρήσαμε ότι όσο μεγάλωνε το dataset τόσο πιο κοντά βρισκόταν το software με το hardware από άποψη απόδοσης στο τελευταίο datasheet μάλιστα παρατηρήσαμε ότι το software ήταν πιο αποδοτικό. Στην δεύτερη υλοποίηση χρησιμοποιήσαμε μεθόδους βελτιστοποίησης που μας παρέχει το εργαλείο vivado HLS, τα αποτελέσματα σε αυτήν την περίπτωση ήταν καλύτερα ναι μεν από την πρώτη αλλά και πάλι παρατηρούμε ότι στο τελευταίο datasheet το software είναι πιο αποδοτικό. Στην τρίτη υλοποίηση δημιουργήσαμε έναν ενιαίο πίνακα που περιείχε όλες τις παραμέτρους και τα δεδομένα που ήταν απαραίτητα για την επίλυση του συγκεκριμένου αλγορίθμου. Η υλοποίηση αυτή ήταν πιο αργή από όλες τις άλλες υλοποιήσεις καθώς και πιο ακριβή καθώς χρειάζονται πολλές πράξεις για να γνωρίζουμε πιο κομμάτι του πίνακα χρειαζόμαστε κάθε φορά. Η τέταρτη υλοποίηση είναι η υλοποίηση στην οποία χρησιμοποιήσαμε το framework DAE εδώ τα αποτέλεσμα ήταν όπως αναμενόταν είχαμε μέχρι και 2.5 φορές καλύτερη απόδοση από αυτήν του software. Τέλος στην υλοποίηση του HMC τα αποτελέσματα ήταν πολύ καλά καθώς πετύχαμε περίπου τέσσερεις φορές ταχύτερο αλγόριθμο από εκείνον του software. Οι υλοποιήσεις πραγματοποιήθηκαν στο εργαλείο της Vivado HLS. Παλαιότερα, η σχεδίαση RTL ήταν αρκετά δύσκολη και χρειαζόταν μεγάλη εξειδίκευση από τους προγραμματιστές. Ωστόσο, το εργαλείο Vivado HLS παρέχει μεγάλη ευελιξία

στη χαρτογράφηση των αλγορίθμων με τη προσφορά της αυτοματοποιημένης διαδικασίας της σύνθεσης του αρχείου rtl και του verification με αποτέλεσμα η υλοποίηση στο hardware να καθίσταται αρκετά πιο εύκολη. Επίσης, παρέχει directives που συμβάλλουν στην βελτίωση της απόδοσης.

Στο εγγύς μέλλον, οι υλοποιήσεις που συνδυάζουν το software και το hardware θα γίνουν ολοένα και περισσότερες και το εργαλείο Vivado HLS θα παίξει σημαντικό ρόλο στην βελτίωση της απόδοσης των συστημάτων

.

7.2 Μελλοντική εργασία

Όπως είδαμε παραπάνω η απεικόνιση των δύο αλγορίθμων στο hardware και η εκτέλεση των αλγορίθμων σε πλατφόρμες όπως FPGAs και HMC συντέλεσαν στη βελτίωση της απόδοσης τους. Αναλογιζόμαστε, ότι οι μελλοντικές εφαρμογές, οι οποίες θα αφορούν την επεξεργασία και τη μεταφορά μεγάλου όγκου δεδομένων θα στραφούν σε ανάλογες υλοποιήσεις με αυτές που εφαρμόσαμε στη παρούσα διπλωματική. Παρόλο που στο HMC και στον software η χρησιμοποίηση μεγάλων datasheet δεν ήταν επίπονη διαδικασία δεν μπορούμε να πούμε το ίδιο και για το RTL του HLS έτσι θα μπορούσαμε με ταχύτερα υπολογιστικά συστήματα να δοκιμάσουμε μεγαλύτερο όγκο δεδομένων έτσι ώστε να μπορούμε να βγάλουμε ασφαλέστερα αποτελέσματα. Επίσης λόγο αυτό πρέπει να πειραματιστούμε και με άλλες πλατφόρμες βγάζοντας πιο ολοκληρωμένα συμπεράσματα. Μία εναλλακτική χρήση πλατφόρμας είναι αυτή της Maxeler. Η πλατφόρμα αυτή δίνει τη δυνατότητα για βελτίωση στην απόδοση του συστήματος μέχρι και 30% σε σχέση με μία συμβατική CPU. Η υλοποίηση πραγματοποιείται από το συνδυασμό του hardware και του software, χρησιμοποιώντας γλώσσες όπως η C,C++,Java για την αύξηση της απόδοσης σε προγράμματα που απαιτούν τη γρήγορη ροή των δεδομένων.

Βιβλιογραφία

[1]K. Asanovic, P. Husbands, Plishker, J. Shalf, S. W. Williams and K. A. Yellick, The Landscape of Parallel Computing Reasearch: A view from Berkeley, Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley (Dec 2006)

[2]James E. Smith “Decoupled Access/Execute Computer Architectures”, Department of Electrical and Computer Engineering, University of Wisconsin-Madison,Madison,Wisconsin 53706

[3]Deependra Talla and Lizy K.John ‘‘A Decoupled Architecture for Accelerating Multimedia Applications’’Laboratory for Computer Architecture Department of Electrical and Computer Engineering The University of Texas, Austin, TX 78712

[4]Michael Sung, Ronny Krashinsky, and Krste Asanovic ‘‘Multithreading Decoupled Architectures for Complexity-Effective General Purpose Computing ’’

[5]Konstantinos Krommydas,Wu-chun Feng ,Christos D Antonopoulos,Nikolaos Bellas ‘‘Open Dwarfs Characterization of Dwarf-Based Benchmarks on Fixed Reconfigurable Architectures’’

[6]W. Feng, H. Lin, T. Scongland and J. Zhang “OpenCl and 13 Dwarfs: A Work in Progress”Department of Computer Science Virginia Tech Blacksburg, VA 24060, USA

[7]Mariza Ferro, Antonio R. Mury, Laion F. Manfroi, Bruno Schlze “High Performance Computing Evaluation A methodology based on Scientific Application Requirements”National Laboratory of Scientific Computing, Getulio Vargas 333, Petropolis, Rio de Janeiro

[8]E. M. Daoudi, A. Lakhouaja, and H. Outada “Study of the Parallel Block One-Sided Jacobi Method”University of Mohamed First, Faculty Sciences Department of Mathematics and Computer Science LaRI Laboratory, 60 000 Ouijda, Morocco

[9]Xilinx.com “kindex ultrascale documentation”

[10]Xilinx Inc. ¨Vivado Design Suite User Guide Getting Started UG910´´

[11]Xilinx Inc. ¨Vivado Design Design Suite Tutorial: High Level Synthesis UG871´´

[12]Xilinx Inc. ¨Vivado Design Suite User Guide: Synthesis UG901´´

[13]Xilinx. Inc ´´Improving Performance Vivado HLS 2013.3 Version´´

[14]George Charitopoulos, Charalampos Vatsolakis, Stefanos Sidiropoulos, Grigorios Chrysos and Dionisios N. Pnevmatikatos “A Decoupled Access-Execute Architecture for Reconfigurable Accelerators” School of Electrical and Computer Engineering Technical University of Crete, Chania, Greece 73100

[15]Prof. Gerhard Wellein, Dr Georg Hager “The seven dwarfs of HPC Possible projects” MuCoSim 27.04.2010 HPC Services, Regionales Rechenzentrum Erlangen (RRZE) Department fur Informatik

[16]Michael Bader “HPC-Algorithms and Applications Dwarf #5 Structured Grids”Winter 2012/2013 Technische Universitat Munchen

[17]Michael Bader “HPC-Algorithms and Applications Dwarf #6 Unstructured Grids”Winter 2012/2013 Technische Universitat Munchen

[18]micron.com ”hmc_controller_ip_user_guide”

[19]micron.com ”PicoAPI_userguide”

[20]Tao Chen and G. Edward Suh “Efficient Data Supply for Hardware Accelerators with Prefetching and Access/Execute Decoupling”Cornell University, Ithaca, NY 14850, USA

[20]Shaoyi Cheng and John Wawrynek “Architectural Synthesis of Computational Pipelines with Decoupled Memory Access” Department of EECS, UC Berkeley, California, USA 94720

[21]Chen-Han-Ho Sung Jin Kim Karthikeyan Sankaralingam Vertical Research Group “Memory Access Dataflow” Department of Computer Sciences, University of Winscosin-Madison

[22]A. Berrached, L. D. Coraor, P. T. Hulina “A decoupled access/execute architecture for efficient access of structured data” in Systems Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference

ΠΑΡΑΡΤΗΜΑ

Α Κώδικας Υλοποιήσεων

Α.1 1η Υλοποίηση

```
#include "accelerator.hpp"

int Wlength = 0;
double temp[8196];
double complex W[64];
int bitwidth;
void initW(int length){
	int i;
		Wlength = length;
		//compute phase weights
		for (i = 0; i < length; i++)
		{

			__real__ W[i] = cos(-((2.0* ((double)i) * M_PI) / ((double)length)));
			__imag__ W[i] = sin(-((2.0* ((double)i) * M_PI) / ((double)length)));
		}
		bitwidth = (int)(log((double)length)/log(2.0));
}


int length = 16;
void my_accelerator(double  *input,double *output)
{

#pragma HLS INTERFACE ap_fifo depth=8196 port=input
#pragma HLS INTERFACE ap_fifo depth=8196 port=output

//		#pragma HLS INTERFACE ap_stable port=length
   int len,i,j,k;
	len = length*length;

	double complex complexArray[4096];
		double complex A[4096];
		double complex B[4096];



	for (i=0; i<len; i++)
	{

		__real__ complexArray[i] = *input;
		input++;
	}
```

```
        for (i=0; i<len; i++)
                {

                        __imag__ complexArray[i] = *input;
                        input++;
                }


double complex complexLine[64];

for (i=0; i< 64; i++)
{

        complexLine[i] = 0;
}

initW(length);
for (j=0; j<length; j++)
   {
      for (i=0; i<length; i++)
      {

         complexLine[i] = complexArray[(j*length)+i];
      }
        FFT_H(complexLine,length);

      for (i=0; i< length; i++)
      {

         complexArray[(j*length)+i] = complexLine[i];

      }
    }

      for(i=0; i<len; i++)
      {

        B[i]=complexArray[i];

      }

   transpose_H(B,length);


   for(i=0; i<len; i++)
         {

                complexArray[i]=B[i];
         }

   for (j=0; j<length; j++)
   {
      for (i=0; i<length; i++)
      {

         complexLine[i] = complexArray[(j*length)+i];
      }
        FFT_H(complexLine,length);

      for (i=0; i< length; i++)
      {
```

```
            complexArray[(j*length)+i] = complexLine[i];
        }
    }

    for(i=0; i<len; i++)
    {

        B[i]=complexArray[i];
     }

    transpose_H(B,length);


        for(i=0; i<len; i++)
        {

            complexArray[i]=B[i];
        }


    for(i=0; i<len; i++)
    {

        temp[i] = __real__ complexArray[i];
        temp[i+len]= __imag__ complexArray[i];
    }

    for(i=0; i<2*len; i++)
        {

                *output = temp[i];
                output++;
        }



}

void FFT_H(double complex *complexLine,int length)
{

        int i,j,k;
          //variables for the FFT
          double complex w,t,u,v;
          //the complex array splits to real array and complex array


                //reorder for butterflies
              int firstBit = length >> 1;
              int index = 0;
              int reverseIndex = 0;
              for (;;) {

                if (reverseIndex < index) {
                  w = complexLine[index];
                  complexLine[index] = complexLine[reverseIndex];
                  complexLine[reverseIndex] = w;
                }
                index++;
                if (index == length)
                  // Break here, because the code below doesn't terminate for all 1's.
                  break;
```

```
            // Add 1 to reverse index, going from left to right:
            int carry = firstBit;
            // If adding 1 changed the bit to 0, we have a carry:
            while (((reverseIndex ^= carry) & carry) == 0)
                carry >>= 1;
        }

                //combine and untangle applying phase weights
        //using successive butterflies of 2,4,...,bitwidth
        for (i = 1; i <= bitwidth; i++)
        {

            int halfStep = (1 << (i - 1));

            for (j = 0; j < halfStep; j++)
            {

                double complex tempT[64];
                int tempShift = (j << (bitwidth - i));
                w = W[tempShift];


                for (k = j; k < length; k += halfStep * 2){

                                        u = complexLine[k + halfStep];
                                        tempT[k] = w * u;
                                }
                for (k = j; k < length; k += halfStep * 2)
                                {

                                        v = complexLine[k];
                                        complexLine[k + halfStep] = v - tempT[k];
                                        complexLine[k] = v + tempT[k];
                                }

            }
        }


}

void transpose_H(double complex *B,int length)
{
        double complex temp;
        int i,j;


        for (i = 0; i < length; i++)
                {
            for (j = i + 1; j < length; j++)
            {

                temp = B[(i*length)+j];
                B[(i*length)+j] = B[(j*length)+i];
                B[(j*length)+i] = temp;
            }
        }

}
```

A.2 2η Υλοποίηση

```
#include "accelerator.hpp"

int Wlength = 0;
double temp[8196];
double complex W[64];
int bitwidth;
void initW(int length){
	int i;
		Wlength = length;
		//compute phase weights
		for (i = 0; i < length; i++)
		{
#pragma HLS pipeline II=8
			__real__ W[i] = cos(-((2.0* ((double)i) * M_PI) / ((double)length)));
			__imag__ W[i] = sin(-((2.0* ((double)i) * M_PI) / ((double)length)));
		}
		bitwidth = (int)(log((double)length)/log(2.0));
}


int length = 16;
void my_accelerator(double  *input,double *output)
{

#pragma HLS INTERFACE ap_fifo depth=8196 port=input
#pragma HLS INTERFACE ap_fifo depth=8196 port=output
#pragma HLS DATAFLOW
//	#pragma HLS INTERFACE ap_stable port=length
   int len,i,j,k;
	len = length*length;

	double complex complexArray[4096];
		double complex A[4096];
		double complex B[4096];



	for (i=0; i<len; i++)
	{
#pragma HLS pipeline II=1
		__real__ complexArray[i] = *input;
		input++;
	}

	for (i=0; i<len; i++)
		{
#pragma HLS pipeline II=1
			__imag__ complexArray[i] = *input;
			input++;
		}


   double complex complexLine[64];

   for (i=0; i< 64; i++)
   {
#pragma HLS pipeline II=1
```

```
            complexLine[i] = 0;
    }

    initW(length);
    for (j=0; j<length; j++)
        {
           for (i=0; i<length; i++)
           {
#pragma HLS pipeline II=1
              complexLine[i] = complexArray[(j*length)+i];
           }
             FFT_H(complexLine,length);

           for (i=0; i< length; i++)
           {
#pragma HLS pipeline II=1
              complexArray[(j*length)+i] = complexLine[i];

           }
         }

           for(i=0; i<len; i++)
           {
#pragma HLS pipeline II=1
             B[i]=complexArray[i];

           }

        transpose_H(B,length);


        for(i=0; i<len; i++)
              {
#pragma HLS pipeline II=1
                        complexArray[i]=B[i];
              }

        for (j=0; j<length; j++)
        {
           for (i=0; i<length; i++)
           {
#pragma HLS pipeline II=1
              complexLine[i] = complexArray[(j*length)+i];
           }
             FFT_H(complexLine,length);

           for (i=0; i< length; i++)
           {
#pragma HLS pipeline II=1
              complexArray[(j*length)+i] = complexLine[i];
           }
        }

        for(i=0; i<len; i++)
        {
#pragma HLS pipeline II=1
              B[i]=complexArray[i];
         }

        transpose_H(B,length);


              for(i=0; i<len; i++)
```

```
            {
#pragma HLS pipeline II=1
                  complexArray[i]=B[i];
            }


      for(i=0; i<len; i++)
      {
#pragma HLS pipeline II=2
           temp[i] = __real__ complexArray[i];
           temp[i+len]= __imag__ complexArray[i];
      }

      for(i=0; i<2*len; i++)
            {
#pragma HLS pipeline II=1
                  *output = temp[i];
                  output++;
            }



}

void FFT_H(double complex *complexLine,int length)
{
#pragma HLS DATAFLOW
           int i,j,k;
              //variables for the FFT
              double complex w,t,u,v;
              //the complex array splits to real array and complex array


                     //reorder for butterflies
                        int firstBit = length >> 1;
                        int index = 0;
                        int reverseIndex = 0;
                        for (;;) {
#pragma HLS pipeline II=1
                           if (reverseIndex < index) {
                              w = complexLine[index];
                              complexLine[index] = complexLine[reverseIndex];
                              complexLine[reverseIndex] = w;
                           }
                           index++;
                           if (index == length)
                              // Break here, because the code below doesn't terminate for all 1's.
                              break;

                           // Add 1 to reverse index, going from left to right:
                           int carry = firstBit;
                           // If adding 1 changed the bit to 0, we have a carry:
                           while (((reverseIndex ^= carry) & carry) == 0)
                              carry >>= 1;
                        }

                              //combine and untangle applying phase weights
                        //using successive butterflies of 2,4,...,bitwidth
                        for (i = 1; i <= bitwidth; i++)
                        {
                                       #pragma HLS pipeline II=1
                           int halfStep = (1 << (i - 1));
```

```
                        for (j = 0; j < halfStep; j++)
                        {
                                #pragma HLS pipeline II=1
                          double complex tempT[64];
                          int tempShift = (j << (bitwidth - i));
                          w = W[tempShift];

                          /*for (k = j; k < length; k += halfStep * 2)
                          {
                                                #pragma HLS pipeline II=1

                            u = complexLine[k + halfStep];

                             v = complexLine[k];

                             t = w * u;

                             complexLine[k + halfStep] = v - t;

                             complexLine[k] = v + t;

                          }*/
                          for (k = j; k < length; k += halfStep * 2){
                                                #pragma HLS pipeline II=1
                                                u = complexLine[k + halfStep];
                                                tempT[k] = w * u;
                                        }
                          for (k = j; k < length; k += halfStep * 2)
                                        {
                                                //#pragma HLS pipeline II=1
                                                v = complexLine[k];
                                                complexLine[k + halfStep] = v - tempT[k];
                                                complexLine[k] = v + tempT[k];
                                        }

                        }
                      }


}

void transpose_H(double complex *B,int length)
{
        double complex temp;
        int i,j;


        for (i = 0; i < length; i++)
                        {
                          for (j = i + 1; j < length; j++)
                          {
#pragma HLS pipeline II=1
                            temp = B[(i*length)+j];
                            B[(i*length)+j] = B[(j*length)+i];
                            B[(j*length)+i] = temp;
                          }
                        }

}
```

A.3 3η Υλοποίηση

```
#include "litos.hpp"
/************************************************
realComplexarray --> bigbrother[i]
imagComplexarray --> bigbrother[len + i]
realComplexline --> bigbrother[2len + i]
imagComplexline --> bigbrother[2len +length + i]
realTempT       --> bigbrother[2len +2length + i]
imagTempT       --> bigbrother[2len +3length + i]
************************************************/
int Wlength = 0;

//double temp[8196];
double complex W[64];
int bitwidth;
void initW(int length)
{
                int i;
                Wlength = length;
                //compute phase weights
                for (i = 0; i < length; i++)
                {
                                #pragma HLS pipeline II=8
                                __real__ W[i] = cos(-((2.0* ((double)i) * M_PI) / ((double)length)));
                                __imag__ W[i] = sin(-((2.0* ((double)i) * M_PI) / ((double)length)));
                }
                bitwidth = (int)(log((double)length)/log(2.0));
}


int length = 8;
int len,len_t,len_t_p,len_t_pp,len_t_ppp;
len = 64;
len_t = 128; //2*glen;
len_t_p = 136; //len_t + length;
len_t_pp = 144; //len_t_p +length;
len_t_ppp = 152; //len_t_pp + length;

void my_accelerator(double  *input,double *output)
{

                #pragma HLS INTERFACE ap_fifo depth=8196 port=input
                #pragma HLS INTERFACE ap_fifo depth=8196 port=output
                //#pragma HLS DATAFLOW
                //          #pragma HLS INTERFACE ap_stable port=length
        int i,j,k;
        double  bigbrother[4096];



        for (i=0; i<len; i++)
        {
                                #pragma HLS pipeline II=1
                        bigbrother[i] = *input;
                        input++;
        }
```

```
    int index;
    index = len;
    for (i=0; i<len; i++)
            {
                    #pragma HLS pipeline II=1
                    bigbrother[index] = *input;
                    input++;
                    index++;
            }


    int index2,index3,index4;
    index = len_t;


    initW(length);
    for (j=0; j<length; j++)
    {
                for (i=0; i<length; i++)
                {
                                #pragma HLS pipeline II=1
                        bigbrother[len_t + i] = bigbrother[(j*length)+i];
                        bigbrother[len_t_p + i] = bigbrother[(j*length)+i+len];
                }

                FFT_H(bigbrother,length);

                for (i=0; i< length; i++)
                {
                                #pragma HLS pipeline II=1
                        bigbrother[(j*length)+i] = bigbrother[len_t + i];
                        bigbrother[(j*length)+i+len] = bigbrother[len_t_p + i];
                }
    }

transpose_H(bigbrother,length);


for (j=0; j<length; j++)
{
    for (i=0; i<length; i++)
    {
                                #pragma HLS pipeline II=1
                bigbrother[len_t + i] = bigbrother[(j*length)+i];
                bigbrother[len_t_p + i] = bigbrother[(j*length)+i+len];
    }

    FFT_H(bigbrother,length);

    for (i=0; i< length; i++)
    {
                                #pragma HLS pipeline II=1
                bigbrother[(j*length)+i] = bigbrother[len_t + i];
                bigbrother[(j*length)+i+len] = bigbrother[len_t_p + i];
    }
}

transpose_H(bigbrother,length);


for(i=0; i<len_t; i++)
```

```
	{
#pragma HLS pipeline II=1
		*output = bigbrother[i];
		output++;
	}

}

void FFT_H(double *bigbrother,int length)
{
#pragma HLS DATAFLOW
		int i,j,k;
	//variables for the FFT
	double complex w,t,u,v;
	//the complex array splits to real array and complex array

		//reorder for butterflies
		int firstBit = length >> 1;
		int index = 0;
		int reverseIndex = 0;
		for (;;)
		{
#pragma HLS pipeline II=1
			if (reverseIndex < index)
			{
				__real__ w = bigbrother[len_t + index];
				__imag__ w = bigbrother[len_t_p + index];
				bigbrother[len_t + index] = bigbrother[len_t + reverseIndex];
				bigbrother[len_t_p + index] = bigbrother[len_t_p + reverseIndex];
				bigbrother[len_t + reverseIndex] = __real__ w;
				bigbrother[len_t_p + reverseIndex] = __imag__ w;
			}
			index++;
			if (index == length)
			  // Break here, because the code below doesn't terminate for all 1's.
			    break;

			// Add 1 to reverse index, going from left to right:
			int carry = firstBit;
			// If adding 1 changed the bit to 0, we have a carry:
			while (((reverseIndex ^= carry) & carry) == 0)
			  carry >>= 1;
		}

		//combine and untangle applying phase weights
		//using successive butterflies of 2,4,...,bitwidth
		for (i = 1; i <= bitwidth; i++)
		{
			//#pragma HLS pipeline II=1
			int halfStep = (1 << (i - 1));

			for (j = 0; j < halfStep; j++)
			{
#pragma HLS pipeline II=1
			  int tempShift = (j << (bitwidth - i));
			  w = W[tempShift];

			  for (k = j; k < length; k += halfStep * 2){
#pragma HLS pipeline II=1
				__real__ u = bigbrother[len_t +k + halfStep];
				__imag__ u = bigbrother[len_t_p +k + halfStep];
				bigbrother[len_t_pp + k] = __real__ (w * u);
				bigbrother[len_t_ppp + k] = __imag__ (w * u);
```

```
                        }
                    for (k = j; k < length; k += halfStep * 2)
                    {
                                        #pragma HLS pipeline II=1
                                        __real__ v = bigbrother[len_t + k];
                                        __imag__ v = bigbrother[len_t_p + k];
                                        bigbrother[len_t + k + halfStep] = __real__ v -
bigbrother[len_t_pp + k];
                                        bigbrother[len_t_p + k + halfStep] = __imag__ v -
bigbrother[len_t_ppp + k];
                                        bigbrother[len_t + k] = __real__ v + bigbrother[len_t_pp + k];
                                        bigbrother[len_t_p + k] = __imag__ v + bigbrother[len_t_ppp +
k];
                                }

                }
}


}

void transpose_H(double *bigbrother,int length)
{
                double  temp;
                int i,j;


                for (i = 0; i < length; i++)
                {
                        for (j = i + 1; j < length; j++)
                        {
                                        #pragma HLS pipeline II=1
                            temp = bigbrother[(i*length)+j];
                            bigbrother[(i*length)+j] = bigbrother[(j*length)+i];
                            bigbrother[(j*length)+i] = temp;
                        }
                }

                for (i = 0; i < length; i++)
                {
                            for (j = i + 1; j < length; j++)
                            {
                                        #pragma HLS pipeline II=1
                                temp = bigbrother[len + ((i*length)+j)];
                                bigbrother[len + ((i*length)+j)] = bigbrother[len + ((j*length)+i)];
                                bigbrother[len + ((j*length)+i)] = temp;
                            }
                }

}
```

A.4 4η Υλοποίηση

```
void Fetch1(int length, ap_uint<34> init_complexLine, ap_uint<40> *tagadr_complexLine, ap_uint<6>
*tagcomplexline)
{
#pragma HLS INTERFACE ap_stable port=init_complexLine
#pragma HLS INTERFACE ap_fifo depth=1024 port= tagadr_complexLine
```

```
#pragma HLS INTERFACE ap_fifo depth=1024 port= tagcomplexline
#pragma HLS DATAFLOW

        int i;
        ap_uint<34> idx1;
        ap_uint<6> tag = 0;
        ap_uint<40> temp;

        for(i=0; i<2*length*8; i +=32)
        {
        #pragma HLS pipeline II=1
                idx1 = init_complexLine + i;
                *tagcomplexline = tag;
                tagcomplexline++;
                tagadr_complexLine->range(39,34) = tag;
                tagadr_complexLine->range(33,0) = idx1;
                tagadr_complexLine++;
                tag++;
        }

}


void Process2(int length,double init_addressResult,double *compLine,double *address_Result,double *cLine)

{

#pragma HLS INTERFACE ap_stable port=init_addressResult
#pragma HLS INTERFACE ap_fifo depth=1024 port=compLine
#pragma HLS INTERFACE ap_fifo depth=1024 port=cLine
#pragma HLS INTERFACE ap_fifo depth=1024 port=address_Result

#pragma HLS DATAFLOW
int i,j,k;
double complexLine[7][128];
double r_W[64];
double i_W[64];
int bitwidth;
//double compW[128];
double  r_w,r_t,r_u,r_v,i_w,i_t,i_u,i_v;

for(i=0; i<64; i++)
{
        r_W[i] = 0;
        i_W[i] = 0;
}
for(j=0; i<7; j++)
{
        for(i=0; i<128; i++)
        {
                complexLine[j][i] = 0;
        }
}

for(i=0; i<2*length; i++)
        {
#pragma HLS pipeline II=1
                complexLine[0][i] = *compLine;
                compLine++;
        }
```

```
        for(i=0; i<length; i++)
        {
#pragma HLS pipeline II=1
                r_W[i] =  cos(-((2.0* ((double)i) * M_PI) / ((double)length)));
                i_W[i] =  sin(-((2.0* ((double)i) * M_PI) / ((double)length)));
        }
        bitwidth = (int)(log((double)length)/log(2.0));


        int firstBit = length >> 1;
            int index = 0;
            int reverseIndex = 0;
            for (;;) {
              if (reverseIndex < index) {
                r_w = complexLine[0][index];
                i_w = complexLine[0][length+index];
                complexLine[0][index] = complexLine[0][reverseIndex];
                complexLine[0][length+index] = complexLine[0][length+reverseIndex];
                complexLine[0][reverseIndex] = r_w;
                complexLine[0][length+reverseIndex] = i_w;

              }
              index++;
              if (index == length)
                // Break here, because the code below doesn't terminate for all 1's.
                break;

              // Add 1 to reverse index, going from left to right:
              int carry = firstBit;
              // If adding 1 changed the bit to 0, we have a carry:
              while (((reverseIndex ^= carry) & carry) == 0)
                carry >>= 1;
            }


        for (i = 1; i <= bitwidth; i++)
            {
              int halfStep = (1 << (i - 1));

              for (j = 0; j < halfStep; j++)
              {
#pragma HLS pipeline II=1
                int tempShift = (j << (bitwidth - i));
                r_w = r_W[tempShift];
                i_w = i_W[tempShift];
                //printf("\nj=%d\n",j);
                for (k = j; k < length; k += halfStep * 2)
                {
#pragma HLS pipeline II=1
                  r_u = complexLine[i-1][k + halfStep];
                  i_u = complexLine[i-1][length+k+halfStep];
                  r_v = complexLine[i-1][k];
                  i_v = complexLine[i-1][length+k];

                  r_t = (r_w*r_u) - (i_w*i_u);
                  i_t = (r_w*i_u) + (i_w*r_u);

                  complexLine[i][k + halfStep] = r_v - r_t;
                  complexLine[i][length+k+halfStep] = i_v -i_t;

                    complexLine[i][k] = r_v + r_t;
                    complexLine[i][length+k] = i_v + i_t;
```

```
                    //  printf("\ntempShift = %d\n",length+k+halfStep);
                }
            }

        }


    for(i=0; i<2*length; i++)
    {
        *cLine = complexLine[6][i];
        cLine++;
    }

    for(i=0; i<2*length*8; i+=32)
    {
        *address_Result = init_addressResult + i;
        address_Result++;
    }

}
```

A.5 5η Υλοποίηση

```
void Fetch1(int length, ap_uint<34> init_complexLine, ap_uint<40> *tagadr_complexLine, ap_uint<6>
*tagcomplexline)
{
#pragma HLS INTERFACE ap_stable port=init_complexLine
#pragma HLS INTERFACE ap_fifo depth=1024 port= tagadr_complexLine
#pragma HLS INTERFACE ap_fifo depth=1024 port= tagcomplexline
#pragma HLS DATAFLOW

    int i;
    ap_uint<34> idx1;
    ap_uint<6> tag = 0;
    ap_uint<40> temp;

    for(i=0; i<2*length*8; i +=32)
    {
    #pragma HLS pipeline II=1
        idx1 = init_complexLine + i;
        *tagcomplexline = tag;
        tagcomplexline++;
        tagadr_complexLine->range(39,34) = tag;
        tagadr_complexLine->range(33,0) = idx1;
        tagadr_complexLine++;
        tag++;
    }

}
```

```
void fetch1_reorder(int length, ap_uint<134> *tagDataComplexline, ap_uint<6> *tagComplexline, ap_uint<64>
*complexline_f){

#pragma HLS CLOCK domain=default
#pragma HLS DATAFLOW

#pragma HLS INTERFACE ap_fifo depth=512 port=tagDataComplexline
#pragma HLS INTERFACE ap_fifo depth=512 port=tagComplexline
#pragma HLS INTERFACE ap_fifo depth=512 port=complexline_f

	ap_uint<64> datavalid =0;
	ap_uint<64> dataBuffer[256];
	int i,k;
	k=0;
	ap_uint<6> outgoingTag=*tagComplexline;
	tagComplexline++;
	for(i =0; i <2*length; i+= 4)
	{
		ap_uint<6> incomingTag;
		ap_uint<8> Rindex;
		incomingTag = tagDataComplexline->range(133,128);
		Rindex = incomingTag;
		Rindex = Rindex <<2;
		dataBuffer[Rindex + 0] = tagDataComplexline->range(63,0);
		dataBuffer[Rindex + 1] = tagDataComplexline->range(127,64);
		tagDataComplexline++;
		dataBuffer[Rindex + 2] = tagDataComplexline->range(63,0);
		dataBuffer[Rindex + 3] = tagDataComplexline->range(127,64);
		tagDataComplexline++;
		/*datavalid.bit(incomingTag) = 1;
		while (datavalid.bit(outgoingTag)){
		#pragma HLS PIPELINE
			ap_uint<8> Windex = outgoingTag;
			Windex = Windex<<2;
			*complexline_f = dataBuffer[Windex + 0];
			complexline_f++;
			*complexline_f = dataBuffer[Windex + 1];
			complexline_f++;
			*complexline_f = dataBuffer[Windex + 2];
			complexline_f++;
			*complexline_f = dataBuffer[Windex + 3];
			complexline_f++;

			datavalid.bit(outgoingTag) = 0;
			outgoingTag=*tagComplexline;
			tagComplexline++;
		}*/
	}
	for(i=0; i<2*length; i++)
	{
		*complexline_f = dataBuffer[i];
		complexline_f++;
	}

}



void Process2(int length,double init_addressResult,double *compLine,double *address_Result,double *cLine)

{
```

```
#pragma HLS INTERFACE ap_stable port=init_addressResult
#pragma HLS INTERFACE ap_fifo depth=1024 port=compLine
#pragma HLS INTERFACE ap_fifo depth=1024 port=cLine
#pragma HLS INTERFACE ap_fifo depth=1024 port=address_Result

#pragma HLS DATAFLOW
int i,j,k;
double complexLine[7][128];
double r_W[64];
double i_W[64];
int bitwidth;
//double compW[128];
double  r_w,r_t,r_u,r_v,i_w,i_t,i_u,i_v;

for(i=0; i<64; i++)
{
        r_W[i] = 0;
        i_W[i] = 0;
}
for(j=0; i<7; j++)
{
        for(i=0; i<128; i++)
        {
                complexLine[j][i] = 0;
        }
}

for(i=0; i<2*length; i++)
        {
#pragma HLS pipeline II=1
                complexLine[0][i] = *compLine;
                compLine++;
        }



        for(i=0; i<length; i++)
        {
#pragma HLS pipeline II=1
                r_W[i] =  cos(-((2.0* ((double)i) * M_PI) / ((double)length)));
                i_W[i] =  sin(-((2.0* ((double)i) * M_PI) / ((double)length)));
        }
        bitwidth = (int)(log((double)length)/log(2.0));


        int firstBit = length >> 1;
            int index = 0;
            int reverseIndex = 0;
            for (;;) {
              if (reverseIndex < index) {
                r_w = complexLine[0][index];
                i_w = complexLine[0][length+index];
                complexLine[0][index] = complexLine[0][reverseIndex];
                complexLine[0][length+index] = complexLine[0][length+reverseIndex];
                complexLine[0][reverseIndex] = r_w;
                complexLine[0][length+reverseIndex] = i_w;

              }
              index++;
              if (index == length)
                // Break here, because the code below doesn't terminate for all 1's.
                break;
```

```
                // Add 1 to reverse index, going from left to right:
                int carry = firstBit;
                // If adding 1 changed the bit to 0, we have a carry:
                while (((reverseIndex ^= carry) & carry) == 0)
                    carry >>= 1;
            }


        for (i = 1; i <= bitwidth; i++)
            {
                int halfStep = (1 << (i - 1));

                for (j = 0; j < halfStep; j++)
                {
#pragma HLS pipeline II=1
                    int tempShift = (j << (bitwidth - i));
                    r_w = r_W[tempShift];
                    i_w = i_W[tempShift];
                    //printf("\nj=%d\n",j);
                    for (k = j; k < length; k += halfStep * 2)
                    {
#pragma HLS pipeline II=1
                        r_u = complexLine[i-1][k + halfStep];
                        i_u = complexLine[i-1][length+k+halfStep];
                        r_v = complexLine[i-1][k];
                        i_v = complexLine[i-1][length+k];

                        r_t = (r_w*r_u) - (i_w*i_u);
                        i_t = (r_w*i_u) + (i_w*r_u);

                        complexLine[i][k + halfStep] = r_v - r_t;
                        complexLine[i][length+k+halfStep] = i_v -i_t;

                            complexLine[i][k] = r_v + r_t;
                            complexLine[i][length+k] = i_v + i_t;
                          //  printf("\ntempShift = %d\n",length+k+halfStep);
                    }
                }

            }


        for(i=0; i<2*length; i++)
        {
                *cLine = complexLine[6][i];
                cLine++;
        }

        for(i=0; i<2*length*8; i+=32)
        {
                *address_Result = init_addressResult + i;
                address_Result++;
        }

}
```

και τα modules του vivado

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 08/12/2018 04:55:58 PM
7  // Design Name:
8  // Module Name: ugrid_core
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 `define dh 0.5
22
23 module spectral_core(
24     input pcie_clk,
25     input hmc_clk,
26     input rst,
27
28     output  reg          s4o_valid,
29     input                s4o_rdy,
30     output  reg [127:0]  s4o_data,
31
32     output  reg          s5o_valid,
33     input                s5o_rdy,
34     output  reg [127:0]  s5o_data,
35
36     output  reg          s6o_valid,
37     input                s6o_rdy,
38     output  reg [127:0]  s6o_data,
39
40     output  reg          s7o_valid,
41     input                s7o_rdy,
42     output  reg [127:0]  s7o_data,
43
44     input [31:0] length_r,
45     input [33:0] init_complexLine_V,
46     input [63:0] bitwidth_V,
47     input [63:0] init_addressResult,
48
49     output [39:0] tagadr_complexLine_V_dout,
50     output tagadr_complexLine_V_empty,
51     input  tagadr_complexLine_V_read,
52
53     input [133:0] tagDataComplexline_V_din,
54     input  tagDataComplexline_V_write,
55
56
57     output [63:0] address_Result_dout,
```

```
58      output address_Result_empty,
59      input address_Result_read,
60      output [63:0] cLine_dout,
61      output cLine_empty,
62      input  cLine_read,
63
64
65      input  ap_start_f,
66      output  ap_done_f,
67      output  ap_ready_f,
68      output  ap_idle_f,
69      input  ap_start_f_r,
70      output  ap_done_f_r,
71      output  ap_ready_f_r,
72      output  ap_idle_f_r,
73      input  ap_start_p,
74      output  ap_done_p,
75      output  ap_ready_p,
76      output  ap_idle_p
77      );
78
79      wire [5:0] tagcomplexline_V_din;
80      wire  tagcomplexline_V_full;
81      wire  tagcomplexline_V_write;
82      wire [5:0] tagcomplexline_V_dout;
83      wire  tagcomplexline_V_empty;
84      wire  tagcomplexline_V_read;
85      FIFO_34x6_fwft_dclk_dist tagcomplexline_V(
86          .din(tagcomplexline_V_din),
87          .wr_en(tagcomplexline_V_write),
88          .full(tagcomplexline_V_full),
89          .dout(tagcomplexline_V_dout),
90          .rd_en(tagcomplexline_V_read),
91          .empty(tacomplexline_V_empty),
92          .wr_clk(hmc_clk),
93          .rd_clk(pcie_clk),
94          .rst(rst)
95      );
96
97
98
99      wire [39:0] tagadr_complexLine_V_din;
100     wire  tagadr_complexLine_V_full;
101     wire  tagadr_complexLine_V_write;
102     FIFO_33x40_fwft_dclk tagadr_complexLine(
103         .din(tagadr_complexLine_V_din),
104         .wr_en(tagadr_complexLine_V_write),
105         .full(tagadr_complexLine_V_full),
106         .dout(tagadr_complexLine_V_dout),
107         .rd_en(tagadr_complexLine_V_read),
108         .empty(tagadr_complexLine_V_empty),
109         .rd_clk(hmc_clk),
110         .wr_clk(pcie_clk),
111         .rst(rst)
112     );
113
114     always @(posedge pcie_clk) begin
```

```
114     always @(posedge pcie_clk) begin
115                 if(rst) begin
116                     s4o_valid           <= #`dh 1'b0;
117                     s4o_data            <= #`dh 128'b0;
118                 end else begin
119                             s4o_valid     <= #`dh tagadr_complexLine_V_write;
120                             s4o_data[127:40] <= #`dh 88'b0;
121                             s4o_data[39:0]   <= #`dh tagadr_complexLine_V_din;
122                 end
123         end
124 
125     Fetch1  fetch_unit (
126       .ap_clk(pcie_clk),
127       .ap_rst(rst),
128       .ap_start(ap_start_f),
129       .ap_done(ap_done_f),
130       .ap_ready(ap_ready_f),
131       .ap_idle(ap_idle_f0),
132       .length_r(length_r),
133       .init_complexLine_V(init_complexLine_V),
134       .tagadr_complexLine_V_din(tagadr_complexLine_V_din),
135       .tagadr_complexLine_V_full_n(~tagadr_complexLine_V_full),
136       .tagadr_complexLine_V_write(tagadr_complexLine_V_write),
137       .tagcomplexline_V_din(tagcomplexline_V_din),
138       .tagcomplexline_V_full_n(~tagcomplexline_V_full),
139       .tagcomplexline_V_write(tagcomplexline_V_write)
140     );
141 
142     wire [133:0] tagDataComplexline_V_dout;
143     wire  tagDataComplexline_V_empty;
144     wire  tagDataComplexline_V_read;
145     FIFO_512x134b_fwft_dclk tagDataComplexline_V (
146       .srst(rst),
147       .wr_clk(hmc_clk),
148       .rd_clk(pcie_clk),
149       .din(tagDataComplexline_V_din),
150       .wr_en(tagDataComplexline_V_write),
151       .rd_en(tagDataComplexline_V_read),
152       .dout(tagDataComplexline_V_dout),
153       .full(),
154       .empty(tagDataComplexline_V_empty)
155     );
156 
157 
158 
159 
160 
```

```
161     wire [63:0] complexline_f_din;
162     wire  complexline_f_full;
163     wire  complexline_f_write;
164     wire [63:0] complexline_f_dout;
165     wire  complexline_f_empty;
166     wire  complexline_f_read;
167     FIFO_512x64_sclk_fwft complexline_f(
168       .clk(pcie_clk),
169       .srst(rst),
170       .din(complexline_f_din),
171       .full(complexline_f_full),
172       .wr_en(complexline_f_write),
173       .rd_en(complexline_f_read),
174       .empty(complexline_f_empty),
175       .dout(complexline_f_dout)
176     );
177
178     always @(posedge pcie_clk) begin
179                         if(rst) begin
180                             s5o_valid           <= #`dh 1'b0;
181                             s5o_data            <= #`dh 128'b0;
182                         end else begin
183                                     s5o_valid     <= #`dh complexline_f_write;
184                                     s5o_data[127:64] <= #`dh 64'b0;
185                                     s5o_data[63:0]   <= #`dh complexline_f_din;
186                         end
187     end
188
189
190     fetch1_reorder  reorder (
191       .length_r(length_r),
192       .tagDataComplexline_V_dout(tagDataComplexline_V_dout),
193       .tagDataComplexline_V_empty_n(~tagDataComplexline_V_empty),
194       .tagDataComplexline_V_read(tagDataComplexline_V_read),
195       .tagComplexline_V_dout(tagcomplexline_V_dout),
196       .tagComplexline_V_empty_n(~tagcomplexline_V_empty),
197       .tagComplexline_V_read(tagcomplexline_V_read),
198       .complexline_f_V_din(complexline_f_din),
199       .complexline_f_V_full_n(~complexline_f_full),
200       .complexline_f_V_write(complexline_f_write),
201       .ap_clk(pcie_clk),
202       .ap_rst(rst),
203       .ap_start(ap_start_f_r),
204       .ap_done(ap_done_f_r),
205       .ap_ready(ap_ready_f_r),
206       .ap_idle(ap_idle_f_r)
207     );
208
209
210
```

```
211     wire [63:0] address_Result_din;
212     wire  address_Result_full;
213     wire  address_Result_write;
214     FIFO_33x64b_fwft_dclk address_Result (
215       .rst(rst),
216       .wr_clk(pcie_clk),
217       .rd_clk(pcie_clk),
218       .din(address_Result_din),
219       .wr_en(address_Result_write),
220       .rd_en(address_Result_read),
221       .dout(address_Result_dout),
222       .full(address_Result_full),
223       .empty(address_Result_empty)
224     );
225
226      always @(posedge pcie_clk) begin
227                                 if(rst) begin
228                                     s7o_valid           <= #`dh 1'b0;
229                                     s7o_data            <= #`dh 128'b0;
230                                 end else begin
231                                             s7o_valid     <= #`dh address_Result_write;
232                                             s7o_data[127:64] <= #`dh 64'b0;
233                                             s7o_data[63:0]   <= #`dh address_Result_din;
234                                 end
235                   end
236
237
238     wire [63:0] cLine_din;
239     wire  cLine_full;
240     wire  cLine_write;
241     FIFO_33x64b_fwft_dclk  cLine (
242       .rst(rst),
243       .wr_clk(pcie_clk),
244       .rd_clk(pcie_clk),
245       .din( cLine_din),
246       .wr_en( cLine_write),
247       .rd_en( cLine_read),
248       .dout( cLine_dout),
249       .full( cLine_full),
250       .empty( cLine_empty)
251     );
252
253          always @(posedge pcie_clk) begin
254                 if(rst) begin
255                     s6o_valid           <= #`dh 1'b0;
256                     s6o_data            <= #`dh 128'b0;
257                 end else begin
258                             s6o_valid     <= #`dh cLine_write;
259                             s6o_data[127:64] <= #`dh 64'b0;
260                             s6o_data[63:0]   <= #`dh cLine_din;
261                 end
262        end
263
264
265
```

```
    Process1 processUnit (
      .ap_clk(pcie_clk),
      .ap_rst(rst),
      .ap_done(ap_done_p),
      .ap_start(ap_start_p),
      .ap_ready(ap_ready_p),
      .ap_idle(ap_idle_p),
      .length_r(length_r),
      .init_addressResult(init_addressResult),
      .address_Result_din(address_Result_din),
      .address_Result_full_n(~address_Result_full),
      .address_Result_write(address_Result_write),
      .compLine_dout(complexline_f_dout),
      .compLine_empty_n(~complexline_f_empty),
      .compLine_read(complexline_f_read),
      .cLine_din(cLine_din),
      .cLine_full_n(~cLine_full),
      .cLine_write(cLine_write)
    );

endmodule
```

η registerfile

```
timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/22/2017 02:02:37 PM
// Design Name:
// Module Name: RFdae128
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
define dh 0.5

odule RFdae128(
    input               pcie_clk,
    input               rst,

    input               sli_valid,
    output reg          sli_rdy,
    input [127:0]       sli_data,
    output reg          slo_valid,
    input               slo_rdy,
    output reg [127:0]  slo_data,

    output ap_start_f,
    input  ap_done_f,
    input  ap_ready_f,
    input  ap_idle_f,
    output ap_start_f_reorder,
    input  ap_done_f_reorder,
    input  ap_ready_f_reorder,
    input  ap_idle_f_reorder,
    output ap_start_p,
    input  ap_done_p,
    input  ap_ready_p,
    input  ap_idle_p,

    output reg [31:0] length_r,
    output reg [33:0] init_complexLine_V,
   // output reg [33:0] init_rW_V,
    //output reg [33:0] init_iW_V,
    //output reg [63:0] bitwidth_V,
    output reg [63:0] init_addressResult

    );

```

```
56      reg global_start;
57      reg [2:0]doneReg;
58      reg ap_start_f_r;
59      reg ap_start_f_reorder_r;
60      reg ap_start_p_r;
61      reg [63:0] clkCnt;
62      reg [63:0] performance;
63
64      assign ap_start_f = ap_start_f_r & ~ap_done_f;
65      assign ap_start_f_reorder = ap_start_f_reorder_r & ~ap_done_f_reorder;
66      assign ap_start_p = ap_start_p_r & ~ap_done_p;
67
68      /*Start Register Signals*/
69      always @(posedge pcie_clk) begin
70          if (rst) begin
71              ap_start_f_r <=#`dh 1'b0;
72          end else begin
73              if(ap_done_f)
74                  ap_start_f_r <= #`dh 1'b0;
75              else if(doneReg == 3'b111 && global_start == 1'b1)
76                  ap_start_f_r <= #`dh 1'b1;
77              else
78                  ap_start_f_r <=#`dh ap_start_f_r;
79          end
80      end
81      always @(posedge pcie_clk) begin
82          if (rst) begin
83              ap_start_f_reorder_r <=#`dh 1'b0;
84          end else begin
85              if(ap_done_f_reorder)
86                  ap_start_f_reorder_r <= #`dh 1'b0;
87              else if(doneReg == 3'b111 && global_start == 1'b1)
88                  ap_start_f_reorder_r <= #`dh 1'b1;
89              else
90                  ap_start_f_reorder_r <=#`dh ap_start_f_reorder_r;
91          end
92      end
93
94
95      always @(posedge pcie_clk) begin
96          if (rst) begin
97              ap_start_p_r <=#`dh 1'b0;
98          end else begin
99              if(ap_done_p)
100                 ap_start_p_r <= #`dh 1'b0;
101             else if(doneReg == 3'b111 && global_start == 1'b1)
102                 ap_start_p_r <= #`dh 1'b1;
103             else
104                 ap_start_p_r <=#`dh ap_start_p_r;
105         end
106     end
107
```

```
108     /*Done collection Signals*/
109     always @(posedge pcie_clk) begin
110         if (rst) begin
111             doneReg[0] <=#`dh 1'b1;
112         end else begin
113             if(ap_done_f)
114                 doneReg[0] <=#`dh 1'b1;
115             else if(ap_start_f)
116                 doneReg[0] <=#`dh 1'b0;
117             else
118                 doneReg[0] <=#`dh doneReg[0];
119         end
120     end
121     always @(posedge pcie_clk) begin
122         if (rst) begin
123             doneReg[1] <=#`dh 1'b1;
124         end else begin
125             if(ap_done_f_reorder)
126                 doneReg[1] <=#`dh 1'b1;
127             else if(ap_start_f_reorder)
128                 doneReg[1] <=#`dh 1'b0;
129             else
130                 doneReg[1] <=#`dh doneReg[1];
131         end
132     end
133
134     always @(posedge pcie_clk) begin
135         if (rst) begin
136             doneReg[2] <=#`dh 1'b1;
137         end else begin
138             if(ap_done_p)
139                 doneReg[2] <=#`dh 1'b1;
140             else if(ap_start_p)
141                 doneReg[2] <=#`dh 1'b0;
142             else
143                 doneReg[2] <=#`dh doneReg[2];
144         end
145     end
146
147
148   /Performance counter
149     always @(posedge pcie_clk) begin
150         if (rst) begin
151             clkCnt <=  #`dh 64'b0;
152         end else begin
153             if(doneReg == 3'b111)
154                 performance <=  #`dh clkCnt;
155             else
156                 clkCnt <=  #`dh clkCnt + 64'b1;
157         end
158     end
159
```

```
/PCIe Register File
    always @(posedge pcie_clk) begin
        if(rst) begin
            sli_rdy             <= #`dh 1'b0;
            slo_valid           <= #`dh 1'b0;
            slo_data            <= #`dh 128'b0;
            global_start        <= #`dh 1'b0;

            length_r            <= #`dh 32'b0;
            init_complexLine_V  <= #`dh 34'b0;
            //init_rW_V           <= #`dh 64'b0;
            //init_iW_V           <= #`dh 64'b0;
            init_addressResult  <= #`dh 64'b0;
            //bitwidth_V          <= #`dh 64'b0;

        end else begin
            sli_rdy     <= #`dh 1'b1;
            if (sli_valid) begin
                if(sli_data[127:124] == 4'h1) begin
                    length_r                    <= #`dh sli_data[31:0];
                end else if (sli_data[127:124] == 4'h2) begin
                    init_complexLine_V  <= #`dh sli_data[97:64];
                end else if (sli_data[127:124] == 4'h3) begin
                    init_addressResult  <= #`dh sli_data[63:0];
                end else if (sli_data[127:124] == 4'h4) begin
                    global_start        <= #`dh sli_data[64];
                end else begin
                    slo_valid <= #`dh 1'b1;
                    slo_data[63:0] <= #`dh performance;
                    slo_data[64] <= #`dh ap_start_f;
                    slo_data[65] <= #`dh ap_ready_f;
                    slo_data[66] <= #`dh ap_idle_f;
                    slo_data[67] <= #`dh ap_done_f;
                    slo_data[68] <= #`dh ap_start_f_reorder;
                    slo_data[69] <= #`dh ap_ready_f_reorder;
                    slo_data[70] <= #`dh ap_idle_f_reorder;
                    slo_data[71] <= #`dh ap_done_f_reorder;
                    slo_data[72] <= #`dh ap_start_p;
                    slo_data[73] <= #`dh ap_ready_p;
                    slo_data[74] <= #`dh ap_idle_p;
                    slo_data[75] <= #`dh ap_done_p;
                    slo_data[76] <= #`dh global_start;
                    slo_data[127:77] <= #`dh 31'b0;
                end
            end else begin
                if(slo_valid & slo_rdy)begin
                    slo_valid <= #`dh 1'b0;
                    slo_data  <= #`dh 128'b0;
                end else begin
                    slo_valid <= #`dh slo_valid;
                    slo_data  <= #`dh slo_data;
                end

            end
        end
    end

```

και το top level

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 08/16/2018 01:49:44 PM
7  // Design Name:
8  // Module Name: unstructured_grid
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 `include "PicoDefines.v"
22 `include "hmc_def.vh"
23
24
25 module unstructured_grid#(
26         parameter SIZE_WIDTH = 4,
27         parameter ADDR_WIDTH = 34,
28         parameter DATA_WIDTH = 128,
29         parameter TAG_WIDTH = 6
30         ) (
31         input                       hmc_tx_clk,
32         input                       hmc_rx_clk,
33         input                       hmc_rst,
34         input                       hmc_trained,
35
36         output                      hmc_clk_p0,
37         output                      hmc_cmd_valid_p0,
38         input                       hmc_cmd_ready_p0,
39         output [3:0]                hmc_cmd_p0,
40         output [ADDR_WIDTH-1:0]     hmc_addr_p0,
41         output [SIZE_WIDTH-1:0]     hmc_size_p0,
42         output [TAG_WIDTH-1:0]      hmc_tag_p0,
43         output [DATA_WIDTH-1:0]     hmc_wr_data_p0,
44         output                      hmc_wr_data_valid_p0,
45         input                       hmc_wr_data_ready_p0,
46
47         input [DATA_WIDTH-1:0]      hmc_rd_data_p0,
48         input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p0,
49         input                       hmc_rd_data_valid_p0,
50         input [6:0]                 hmc_errstat_p0,
51         input                       hmc_dinv_p0,
52         output                      hmc_clk_p1,
53         output                      hmc_cmd_valid_p1,
54         input                       hmc_cmd_ready_p1,
55         output [3:0]                hmc_cmd_p1,
56         output [ADDR_WIDTH-1:0]     hmc_addr_p1,
57         output [SIZE_WIDTH-1:0]     hmc_size_p1,
```

```
58          output [TAG_WIDTH-1:0]      hmc_tag_p1,
59          output [DATA_WIDTH-1:0]     hmc_wr_data_p1,
60          output                      hmc_wr_data_valid_p1,
61          input                       hmc_wr_data_ready_p1,
62
63          input [DATA_WIDTH-1:0]      hmc_rd_data_p1,
64          input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p1,
65          input                       hmc_rd_data_valid_p1,
66          input [6:0]                 hmc_errstat_p1,
67          input                       hmc_dinv_p1,
68          output                      hmc_clk_p2,
69          output                      hmc_cmd_valid_p2,
70          input                       hmc_cmd_ready_p2,
71          output [3:0]                hmc_cmd_p2,
72          output [ADDR_WIDTH-1:0]     hmc_addr_p2,
73          output [SIZE_WIDTH-1:0]     hmc_size_p2,
74          output [TAG_WIDTH-1:0]      hmc_tag_p2,
75          output [DATA_WIDTH-1:0]     hmc_wr_data_p2,
76          output                      hmc_wr_data_valid_p2,
77          input                       hmc_wr_data_ready_p2,
78
79          input [DATA_WIDTH-1:0]      hmc_rd_data_p2,
80          input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p2,
81          input                       hmc_rd_data_valid_p2,
82          input [6:0]                 hmc_errstat_p2,
83          input                       hmc_dinv_p2,
84          output                      hmc_clk_p3,
85          output                      hmc_cmd_valid_p3,
86          input                       hmc_cmd_ready_p3,
87          output [3:0]                hmc_cmd_p3,
88          output [ADDR_WIDTH-1:0]     hmc_addr_p3,
89          output [SIZE_WIDTH-1:0]     hmc_size_p3,
90          output [TAG_WIDTH-1:0]      hmc_tag_p3,
91          output [DATA_WIDTH-1:0]     hmc_wr_data_p3,
92          output                      hmc_wr_data_valid_p3,
93          input                       hmc_wr_data_ready_p3,
94
95          input [DATA_WIDTH-1:0]      hmc_rd_data_p3,
96          input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p3,
97          input                       hmc_rd_data_valid_p3,
98          input [6:0]                 hmc_errstat_p3,
99          input                       hmc_dinv_p3,
100         output                      hmc_clk_p4,
101         output                      hmc_cmd_valid_p4,
102         input                       hmc_cmd_ready_p4,
103         output [3:0]                hmc_cmd_p4,
104         output [ADDR_WIDTH-1:0]     hmc_addr_p4,
105         output [SIZE_WIDTH-1:0]     hmc_size_p4,
106         output [TAG_WIDTH-1:0]      hmc_tag_p4,
107         output [DATA_WIDTH-1:0]     hmc_wr_data_p4,
108         output                      hmc_wr_data_valid_p4,
109         input                       hmc_wr_data_ready_p4,
110
111         input [DATA_WIDTH-1:0]      hmc_rd_data_p4,
112         input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p4,
113         input                       hmc_rd_data_valid_p4,
114         input [6:0]                 hmc_errstat_p4,
```

```
115         input                       hmc_dinv_p4,
116         output                      hmc_clk_p5,
117         output                      hmc_cmd_valid_p5,
118         input                       hmc_cmd_ready_p5,
119         output [3:0]                hmc_cmd_p5,
120         output [ADDR_WIDTH-1:0]     hmc_addr_p5,
(Ctrl+X)
121         output [SIZE_WIDTH-1:0]     hmc_size_p5,
122         output [TAG_WIDTH-1:0]      hmc_tag_p5,
123         output [DATA_WIDTH-1:0]     hmc_wr_data_p5,
124         output                      hmc_wr_data_valid_p5,
125         input                       hmc_wr_data_ready_p5,
126
127         input [DATA_WIDTH-1:0]      hmc_rd_data_p5,
128         input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p5,
129         input                       hmc_rd_data_valid_p5,
130         input [6:0]                 hmc_errstat_p5,
131         input                       hmc_dinv_p5,
132         output                      hmc_clk_p6,
133         output                      hmc_cmd_valid_p6,
134         input                       hmc_cmd_ready_p6,
135         output [3:0]                hmc_cmd_p6,
136         output [ADDR_WIDTH-1:0]     hmc_addr_p6,
137         output [SIZE_WIDTH-1:0]     hmc_size_p6,
138         output [TAG_WIDTH-1:0]      hmc_tag_p6,
139         output [DATA_WIDTH-1:0]     hmc_wr_data_p6,
140         output                      hmc_wr_data_valid_p6,
141         input                       hmc_wr_data_ready_p6,
142
143         input [DATA_WIDTH-1:0]      hmc_rd_data_p6,
144         input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p6,
145         input                       hmc_rd_data_valid_p6,
146         input [6:0]                 hmc_errstat_p6,
147         input                       hmc_dinv_p6,
148         output                      hmc_clk_p7,
149         output                      hmc_cmd_valid_p7,
150         input                       hmc_cmd_ready_p7,
151         output [3:0]                hmc_cmd_p7,
152         output [ADDR_WIDTH-1:0]     hmc_addr_p7,
153         output [SIZE_WIDTH-1:0]     hmc_size_p7,
154         output [TAG_WIDTH-1:0]      hmc_tag_p7,
155         output [DATA_WIDTH-1:0]     hmc_wr_data_p7,
156         output                      hmc_wr_data_valid_p7,
157         input                       hmc_wr_data_ready_p7,
158
159         input [DATA_WIDTH-1:0]      hmc_rd_data_p7,
160         input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p7,
161         input                       hmc_rd_data_valid_p7,
162         input [6:0]                 hmc_errstat_p7,
163         input                       hmc_dinv_p7,
164         output                      hmc_clk_p8,
165         output                      hmc_cmd_valid_p8,
166         input                       hmc_cmd_ready_p8,
167         output [3:0]                hmc_cmd_p8,
168         output [ADDR_WIDTH-1:0]     hmc_addr_p8,
169         output [SIZE_WIDTH-1:0]     hmc_size_p8,
170         output [TAG_WIDTH-1:0]      hmc_tag_p8,
171         output [DATA_WIDTH-1:0]     hmc_wr_data_p8,
```

```
172         output                      hmc_wr_data_valid_p8,
173         input                       hmc_wr_data_ready_p8,
174
175         input [DATA_WIDTH-1:0]      hmc_rd_data_p8,
176         input [TAG_WIDTH-1:0]       hmc_rd_data_tag_p8,
177         input                       hmc_rd_data_valid_p8,
178         input [6:0]                 hmc_errstat_p8,
179         input                       hmc_dinv_p8,
180
181
182         input                       clk,
183         input                       rst,
184
185         input                       sli_valid,
186         output                      sli_rdy,
187         input [127:0]               sli_data,
188
189         output                      slo_valid,
190         input                       slo_rdy,
191         output [127:0]              slo_data,
192
193         output                      s2o_valid,
194         input                       s2o_rdy,
195         output [127:0]              s2o_data,
196
197         output                      s3o_valid,
198         input                       s3o_rdy,
199         output [127:0]              s3o_data,
200
201         output                      s4o_valid,
202         input                       s4o_rdy,
203         output [127:0]              s4o_data,
204
205         output                      s5o_valid,
206         input                       s5o_rdy,
207         output [127:0]              s5o_data,
208
209         output                      s6o_valid,
210         input                       s6o_rdy,
211         output [127:0]              s6o_data,
212
213         output                      s7o_valid,
214         input                       s7o_rdy,
215         output [127:0]              s7o_data,
216
217         input                       PicoClk,
218         input                       PicoRst,
219         input  [31:0]               PicoAddr,
220         input  [31:0]               PicoDataIn,
221         output [31:0]               PicoDataOut,
222         input                       PicoRd,
223         input                       PicoWr
224     );
225
```

```
226     wire  ap_start_f;
227     wire  ap_done_f;
228     wire  ap_ready_f;
229     wire  ap_idle_f;
230     wire  ap_start_f_reorder;
231     wire  ap_done_f_reorder;
232     wire  ap_ready_f_reorder;
233     wire  ap_idle_f_reorder;
234     wire  ap_start_p;
235     wire  ap_done_p;
236     wire  ap_ready_p;
237     wire  ap_idle_p;
238
239     wire [31:0] length_r;
240     wire [33:0] init_complexLine_V;
241    // wire [33:0] init_rW_V;
242     //wire [33:0] init_iW_V;
243     //wire [63:0] bitwidth;
244     wire [63:0] init_addressResult;
245
246     wire [39:0] tagadr_complexLine_V_dout;
247     wire  tagadr_complexLine_V_empty;
248     wire  tagadr_complexLine_V_read;
249     wire [133:0] tagDataComplexline_V_din;
250     wire  tagDataComplexline_V_write;
251     assign hmc_clk_p0 = hmc_tx_clk;
252     assign hmc_cmd_valid_p0 = ~tagadr_complexLine_V_empty;
253     assign tagadr_complexLine_V_read  =  hmc_cmd_ready_p0;
254     assign hmc_cmd_p0       = `HMC_CMD_RD;
255     assign hmc_addr_p0      = tagadr_complexLine_V_dout[33:0];
256     assign hmc_size_p0      = 4'h2;
257     assign hmc_tag_p0       = tagadr_complexLine_V_dout[39:34];
258     assign hmc_wr_data_p0   = 128'b0;
259     assign hmc_wr_data_valid_p0 = 1'b0;
260     assign tagDataComplexline_V_din = {hmc_rd_data_tag_p0, hmc_rd_data_p0};
261     assign tagDataComplexline_V_write = hmc_rd_data_valid_p0;
262
263     wire [63:0] address_Result_dout;
264     wire  address_Result_empty;
265     wire  address_Result_read;
266     wire [63:0] cLine_dout;
267     wire  cLine_empty;
268     wire  cLine_read;
269     assign s2o_valid = ~address_Result_empty;
270     assign address_Result_read = s2o_rdy;
271     assign s2o_data[127:64] = 64'b0;
272     assign s2o_data[63:0]  = address_Result_dout;
273     assign s3o_valid = ~cLine_empty;
274     assign cLine_read = s3o_rdy;
275     assign s3o_data[127:64] = 64'b0;
276     assign s3o_data[63:0]  = cLine_dout;
277
278
```

```
291     RFdae128 RegisterFile(
292         .pcie_clk(clk),
293         .rst(rst),
294 
295         .sli_valid(sli_valid),
296         .sli_rdy(sli_rdy),
297         .sli_data(sli_data),
298         .slo_valid(slo_valid),
299         .slo_rdy(slo_rdy),
300         .slo_data(slo_data),
301 
302         .ap_start_f(ap_start_f),
303         .ap_done_f(ap_done_f),
304         .ap_ready_f(ap_ready_f),
305         .ap_idle_f(ap_idle_f),
306         .ap_start_f_reorder(ap_start_f_reorder),
307         .ap_done_f_reorder(ap_done_f_reorder),
308         .ap_ready_f_reorder(ap_ready_f_reorder),
309         .ap_idle_f_reorder(ap_idle_f_reorder),
310         .ap_start_p(ap_start_p),
311         .ap_done_p(ap_done_p),
312         .ap_ready_p(ap_ready_p),
313         .ap_idle_p(ap_idle_p),
314 
315         .length_r(length_r),
316         .init_complexLine_V(init_complexLine_V),
317         //.init_rW_V(init_rW_V),
318         //.init_iW_V(init_iW_V),
319         //.bitwidth_V(bitwidth_V),
320         .init_addressResult(init_addressResult)
321 
322         );
323 
324     spectral_core core(
325         .pcie_clk(clk),
326         .hmc_clk(hmc_tx_clk),
327         .rst(rst),
328 
329         .s4o_valid(s4o_valid),
330         .s4o_rdy(s4o_rdy),
331         .s4o_data(s4o_data),
332         .s5o_valid(s5o_valid),
333         .s5o_rdy(s5o_rdy),
334         .s5o_data(s5o_data),
335         .s6o_valid(s6o_valid),
336         .s6o_rdy(s6o_rdy),
337         .s6o_data(s6o_data),
338         .s7o_valid(s7o_valid),
339         .s7o_rdy(s7o_rdy),
340         .s7o_data(s7o_data),
341 
```

```
341
342         .length_r(length_r),
343         .init_complexLine_V(init_complexLine_V),
344         //.init_rW_V(init_rW_V),
345         //.init_iW_V(init_iW_V),
346         //.bitwidth_V(bitwidth_V),
347         .init_addressResult(init_addressResult),
348
349         .tagadr_complexLine_V_dout(tagadr_complexLine_V_dout),
350         .tagadr_complexLine_V_empty(tagadr_complexLine_V_empty),
351         .tagadr_complexLine_V_read(tagadr_complexLine_V_read),
352         .tagDataComplexline_V_din(tagDataComplexline_V_din),
353         .tagDataComplexline_V_write(tagDataComplexline_V_write),
354
355         //.W_V_din(W_V_din),
356         //.W_V_full(W_V_full),
357         //.W_V_write(W_V_write),
358         .address_Result_dout(address_Result_dout),
359         .address_Result_empty(address_Result_empty),
360         .address_Result_read(address_Result_read),
361         .cLine_dout(cLine_dout),
362         .cLine_empty(cLine_empty),
363         .cLine_read(cLine_read),
364
365         .ap_start_f(ap_start_f),
366         .ap_done_f(ap_done_f),
367         .ap_ready_f(ap_ready_f),
368         .ap_idle_f(ap_idle_f),
369         .ap_start_f_r(ap_start_f_reorder),
370         .ap_done_f_r(ap_done_f_reorder),
371         .ap_ready_f_r(ap_ready_f_reorder),
372         .ap_idle_f_r(ap_idle_f_reorder),
373         .ap_start_p(ap_start_p),
374         .ap_done_p(ap_done_p),
375         .ap_ready_p (ap_ready_p),
376         .ap_idle_p(ap_idle_p)
377     );
378
379 endmodule
380
```