Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Υλοποίηση του αλγορίθμου ευθυγράμμισης ακολουθίας Smith-Waterman σε ενσωματωμένη πλατφόρμα FPGA**
Ονοματεπώνυμο Φοιτητή	**Τζότα Γεωργία**
Πατρώνυμο	**Παναγιώτης**
Αριθμός Μητρώου	**ΜΠΣΠ/ 09039**
Επιβλέπων	**Μιχάλης Ψαράκης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Απρίλιος 2013**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Μιχάλης Ψαράκης Επίκουρος Καθηγητής	Άγγελος Πικράκης Λέκτορας	Δημήτρης Γκιζόπουλος Αναπληρωτής Καθηγητής

Περίληψη

Η ευθυγράμμιση αλληλουχίας (sequence alignment), όρος που προέρχεται από τον κλάδο της βιοπληροφορικής, ασχολείται με την εύρεση κοινών ή παρόμοιων περιοχών ομοιότητας μεταξύ δύο αλληλουχιών DNA ή πρωτεϊνών. Για την υλοποιήση της διαδικασίας της ευθυγράμμισης ακολουθίας έχουν προταθεί πολλαπλοί αλγόριθμοι με την πάροδο των χρόνων, ωστόσο ο μεγάλος όγκος των δεδομένων υπο επεξεργασία και η αδυναμία εκματάλλευσης της παράλληλης επεξεργασίας των δεδομένων αυτών, καθιστούν τις μεθόδους αυτές εξαιρετικά χρονοβόρες αν εκτελεστούν αποκλειστικά με τη χρήση λογισμικού.

Για τους λόγους αυτούς, δημιουργείται η ανάγκη επιτάχυνσης αυτών των αλγορίθμων με χρήση υλικού και συγκεκριμένα, μέσω της τεχνολογίας των FPGAs (Field Programmable Gate Arrays) και των ενσωματωμένων συστημάτων. Τα FPGAs, όντας επαναπρογραμματιζόμενες μονάδες λογικής, μας δίνουν τη δυνατότητα να μετατρέψουμε ένα πολύπλοκο πρόβλημα σε ένα ηλεκτρονικό κύκλωμα εύκολα και χωρίς κόστη πρωτοτύπου, εκμεταλλευόμενοι οποία δυνατότητα παραλληλίας δεδομένων δεν μπορούσε να εκμεταλλευτεί το λογισμικό. Με αυτόν τον τρόπο, μπορεί να κατασκευαστεί ένας επιταχυντής υπολογισμών για την υλοποίηση του αλγορίθμου δυναμικού προγραμματισμού, ο οποίος σε συνδυασμό με άλλα πλεονεκτήματα των ενσωματωμένων συστημάτων, επιταχύνει αισθητά το χρόνο εκτέλεσης του αλγορίθμου.

Στην παρούσα διπλωματική εργασία καλούμαστε να υλοποιήσουμε έναν συγκεκριμένο αλγόριθμο ευθυγράμμισης ακολουθίας, τον αλγόριθμο Smith-Waterman σε ένα ολοκληρωμένο ενσωματωμένο σύστημα. Μελετώντας τα χαρακτηριστικά του αλγορίθμου, επιλέγουμε να υλοποιήσουμε έναν επιταχυντή υπολογισμών στο υλικό με τη μέθοδο του συστολικού πίνακα, ο οποίος προστίθεται στο ενσωματωμένο σύστημα σαν επιπλέον περιφερειακό και αλληλεπιδρά με τον ενσωματωμένο επεξεργαστή που εκτελεί την υπόλοιπη εφαρμογή. Στόχος μας είναι η δημιουργία μίας ενσωματωμένης και κατ' επέκταση φορητής, αποδοτικής μηχανής ανίχνευσης ηχητικών εφέ μέσα σε ροές ηχητικών δεδομένων, όπως είναι μία ταινία. Το σύστημα ανίχνευσης ηχητικών εφέ αποτελείται, αρχικά, από μια εφαρμογή, που εκτελείται στον ενσωματωμένο επεξεργαστή και διαχειρίζεται την προ-επεξεργασία των αρχείων που αποτελούν την ταινία και το προς ανίχνευση ηχητικό εφέ. Επιπρόσθετα, το σύστημα αποτελείται από το περιφερειακό που αποτελεί τον επιταχυντή των υπολογισμών, καθώς και τη μεταξύ τους διασύνδεση μέσω κοινών μνημών και καταχωρητών. Συγκεκριμένα, η γραμμένη σε γλώσσα προγραμματισμού C εφαρμογή, που εκτελείται στον ενσωματωμένο επεξεργαστή, αναλαμβάνει την εξαγωγή των χαρακτηριστικών από τα αρχεία εισόδου, ενημερώνει τις μνήμες του επιταχυντή με τα χαρακτηριστικά αυτά, ενώ παράλληλα μέσω εγγραφής σε κοινούς καταχωρητές τον θέτει σε εκκίνηση. Ο επιταχυντής, αυτός, αναπτύσσεται μετατρέποντας τον S-W αλγόριθμο σε μία συστολική συστοιχία από λογικά στοιχεία (systolic array), όπου το καθένα λειτουργεί όμοια, ανεξάρτητα και παράλληλα, μέσα σε μία αρχιτεκτονική σχεδίασης εκμεταλλευόμενη την παραλληλία των δεδομένων. Με τον τρόπο αυτό, το υλικό, εκμεταλλευόμενο την παραλληλία επεξεργασίας, επιτυγχάνει να επιταχύνει τους χρόνους εκτέλεσης κατά ένα σημαντικό ποσοστό, σε σύγκριση με άλλες υλοποιήσεις του ίδιου αλγορίθμου εκτελεσμένες μόνο σε λογισμικό.

Abstract

The sequence alignment, term derived from the field of bioinformatics, is concerned with finding common or similar regions of similarity between two sequences of DNA or proteins. To implement the process of sequence alignment, multiple algorithms over the years have been proposed, however the large volume of data being processed and the inability to exploit the parallel processing of such data, makes these methods extremely time consuming, if performed solely by computer software .

Therefore, there is a need for acceleration these algorithms using hardware, namely, through the technology of FPGAs (Field Programmable Gate Arrays) and embedded systems.

The FPGAs, as being reprogrammable logic units, enables us to transform a complex problem in an electronic circuit, easily and without prototyping's cost, exploiting whichever possibility of data parallelism that software could not exploit. In this way, a calculations' accelerator can be designed for the implementation of a dynamic programming algorithm, which in combination with other advantages of embedded systems, accelerates, considerably, the execution time of the algorithm.

In this thesis, we are about to implement a specific sequence alignment algorithm, the Smith-Waterman algorithm, into a FPGA embedded system. By examining the characteristics of the algorithm, we choose to implement a calculations' accelerator in hardware by using the method of the systolic array, which is, then, added to the embedded system as an additional peripheral and interacts with the embedded processor, which executes the rest of the application. Our goal is to create an embedded and thus portable, efficient sound effects' detection engine, for audio data streams, such as a movie. The sound effects' detection system consists, primarily, of an application running on the embedded processor, which handles the pre-processing of the files consisting of the film and the to-be-detected sound effect. Furthermore, the system consists of the peripheral assuming the role of the calculations' accelerator, as of the interconnection with each other through shared memory and registers. Specifically, the written in C programming language application, which runs on the embedded processor, is responsible for exporting the sound characteristics of the input files, updates the memories of the accelerator with these features, while making the accelerator to initiate by writing to their common registers. The accelerator is, thereby, developed by converting the S-W algorithm in a systolic array of logic elements (systolic array), each and every one of which elements functions identically, independently and in parallel, in a design architecture taking advantage of the data parallelism. In this way, the hardware taking advantage of the parallel processing, achieves acceleration in execution time by a considerable amount, in comparison with other implementations of the same algorithm designed to run only in software.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω θερμά
τον επιβλέποντα καθηγητή μου κ. Μιχάλη Ψαράκη
για την ευκαιρία που μου έδωσε να ασχοληθώ
με ένα τόσο εξαιρετικά ενδιαφέρον ερευνητικό κομμάτι,
καθώς και για
την αδιάκοπη υποστήριξη και συνεργασία,
καθ' όλη τη διάρκεια εκπόνησης
της παρούσας διατριβής.

1. Εισαγωγή

Πολύ συχνά, τα σύγχρονα υπολογιστικά συστήματα αδυνατούν να τηρήσουν τις προδιαγραφές εκτέλεσης πολύπλοκων προβλημάτων. Τόσο οι σύγχρονοι επεξεργαστές, αν και προηγμένης τεχνολογίας, όσο και το λογισμικό που τρέχει σε αυτούς, αντιμετωπίζουν προβλήματα συμφόρησης, όταν πρόκειται για εκτελέσεις υπολογισμών που αφορούν μεγάλο όγκο και εξαρτήσεις δεδομένων, στο λιγότερο δυνατό χρόνο εκτέλεσης. Για τους παραπάνω λόγους, η επιστημονική κοινότητα, σε διάφορους τομείς δραστηριότητας, επιλέγει την χρήση του υλικού και των ενσωματωμένων συστημάτων για την υλοποίηση ολόκληρου ή μέρους πολύπλοκων και χρονοβόρων εφαρμογών. Αναλυτικότερα, οι σύγχρονες πλατφόρμες ενσωματωμένων συστημάτων επιτρέπουν ένα ευρύ φάσμα εφαρμογών σε πολλαπλούς τομείς της αγοράς, συμπεριλαμβανομένων την αυτοματοποίηση, την ιατρική απεικόνιση, τα όργανα ελέγχου και μετρήσεων, μέχρι και διάφορες στρατιωτικές εφαρμογές.

Τα ενσωματωμένα συστήματα είναι ειδικού σκοπού συστήματα, εξυπηρετώντας τις απαιτήσεις της εφαρμογής που καλείται να εκτελέσει κάθε σύστημα. Ο σχεδιασμός σε ενσωματωμένο σύστημα δημιουργεί βαθύτερα επίπεδα ολοκλήρωσης και αξιοποίησης της επίδοσης παράλληλης επεξεργασίας, ενώ ταυτόχρονα συμβάλλει στην επαναχρησιμοποίηση του υλικού. Οι πλατφόρμες που χρησιμοποιούνται για την υλοποίηση των ενσωματωμένων συστημάτων ονομάζονται “Προγραμματιζόμενες διατάξεις λογικής” (Field Programmable Gate Arrays - FPGA) και προσφέρουν πλήθος πλεονεκτημάτων στο σχεδιαστή του συστήματος, αφού τα στοιχεία λογικής που περιέχουν, προγραμματίζονται επιτόπου με την εκάστοτε σχεδίαση, δίνοντας τη δυνατότητα στο σχεδιαστή να τροποποιεί το σύστημά του, μόλις διαπιστώσει στο υλικό ότι η σχεδίασή του δεν ικανοποιεί τις προδιαγραφές που της έχει θέσει. Αν και υπάρχουν πολλαπλές εφαρμογές χρήσης συσκευών FPGA σε επικοινωνία με κοινούς επεξεργαστές Η/Υ, τα οφέλη που μπορεί να επιφέρει η χρήση ενός ενσωματωμένου στο FPGA επεξεργαστή, μπορεί να είναι πολύ περισσότερα. Ο ενσωματωμένος στο FPGA επεξεργαστής υλοποιείται στα ίδια στοιχεία λογικής που υλοποιείται και η υπόλοιπη σχεδίαση, με αποτέλεσμα να είναι απόλυτα τροποποιήσιμος από το σχεδιαστή του ενσωματωμένου συστήματος. Επιπλέον, οι σύγχρονες πλατφόρμες σχεδίασης ενσωματωμένων συστημάτων δίνουν την δυνατότητα στο σχεδιαστή να επιλέγει τα συστατικά στοιχεία από τα οποία θα αποτελείται το σύστημα και να σχεδιάζει διαύλους για την επικοινωνία όλων των συστατικών αυτών σε ένα ενοποιημένο ενσωματωμένο σύστημα.

Η ικανότητα αυτή των ενσωματωμένων συστημάτων, να κάνουν εναλλαγές μεταξύ του υλικού και του λογισμικού, τα καθιστά ιδανικά για την μεγιστοποίηση της αποτελεσματικότητας και της απόδοσης του συστήματος. Εάν ένας αλγόριθμος έχει αναγνωριστεί να έχει σημεία συμφόρησης όταν εκτελείται σε λογισμικό, μπορεί να σχεδιαστεί ένας επιταχυντής υπολογισμών στο υλικό, η λειτουργία του οποίου θα ελέγχεται από καταχωρητές. Αυτός μπορεί να συνδεθεί με τον ενσωματωμένο επεξεργαστή, δίνοντας έτσι την δυνατότητα στη σχεδίαση να εκμεταλλευτεί την συνύπαρξη υλικού και λογισμικού, μεταφέροντας το πρόβλημα που αντιμετωπίζει το λογισμικό της συγκεκριμένης εφαρμογής στο υλικό. Η αύξηση της απόδοσης δεν οφείλεται μόνο στο γεγονός ότι έχουμε ειδικά διαμορφωμένο υλικό για έναν συγκεκριμένο αλγόριθμο, αλλά επίσης το υλικό από την φύση του μας βοηθά να εκμεταλλευτούμε στο έπακρο κάθε δυνατότητα παραλληλισμού που χαρακτηρίζει τον προς υλοποίηση αλγόριθμο, μειώνοντας δραστικά τους χρόνους εκτέλεσης μίας εφαρμογής.

Όσον αφορά το χώρο της βιοπληροφορικής, η αποτελεσματική ευθυγράμμιση ακολουθίας πρωτεϊνών ή DNA), αποτελεί ένα σημαντικό και δύσκολο έργο. Η διαδικασία της ευθυγράμμισης ακολουθίας είναι παρόμοια με το ταίριασμα συμβολοσειρών στο πλαίσιο των βιολογικών δεδομένων και χρησιμοποιείται για να συναχθεί η εξελικτική σχέση μεταξύ ενός συνόλου πρωτεϊνών ή αλληλουχιών DNA. Μια ακριβής ευθυγράμμιση μπορεί να παράσχει πολύτιμες πληροφορίες για τον πειραματισμό πάνω σε ακολουθίες που έχουν μόλις πρόσφατα βρεθεί. Πρόκειται για μία διαδικασία απαραίτητη τόσο στην έρευνα, καθώς και σε πρακτικές εφαρμογές, όπως η φαρμακευτική ανάπτυξη, η πρόληψη των ασθενειών και η εγκληματολογία. Στις περισσότερες περιπτώσεις η χρήση κλασικών μεθόδων σειριακού προγραμματισμού, εισάγει υπερβολικά μεγάλες καθυστερήσεις. Το πλήθος των αριθμητικών εντολών προς

εκτέλεση είναι τόσο μεγάλο που οι κλασικοί αλγόριθμοι δεν μπορούν να εξάγουν αποτελέσματα μέσα σε ένα ανεκτό χρονικό διάστημα.

Για την ανάπτυξη αποτελεσματικών και βέλτιστων λύσεων ευθυγράμμισης ακολουθιών, έχουν προταθεί αρκετοί αλγόριθμοι με το πέρασμα των χρόνων. Οι αλγόριθμοι δυναμικού προγραμματισμού λειτουργούν διασπώντας το αρχικό πρόβλημα, σε πολλά μικρότερα και απλούστερα υποπροβλήματα. Για κάθε ένα από αυτά τα υποπροβλήματα υπολογίζεται μία λύση, ενώ τα συνολικά αποτελέσματα που προκύπτουν συνδυάζονται, έτσι ώστε να διαμορφωθεί το τελικό αποτέλεσμα που αντιπροσωπεύει και την ολοκληρωμένη λύση. Λόγω του γεγονότος ότι τα υποπροβλήματα που προκύπτουν είναι συνήθως όμοια μεταξύ τους, μπορεί να επιτευχθεί κάποιο είδος παραλληλίας επεξεργασίας, και συνεπώς αρκετά μικρότεροι χρόνοι εκτέλεσης.

Στην παρούσα διατριβή, θα ασχοληθούμε εκτενέστερα με ένα τέτοιο αλγόριθμο, τον αλγόριθμο των Smith και Waterman. Θα παρουσιάσουμε τα οφέλη υλοποίησής του σε μια πλατφόρμα FPGA, καθώς και τους τρόπους με τους οποίους μπορεί να επιταχυνθεί σε μία σχεδίαση FPGA κατασκευάζοντας ένα επιταχυντή υπολογισμών στο υλικό. Τελικά, αυτό επιτυγχάνεται με την μετατροπή του S-W σε μία συστολική συστοιχία από λογικά στοιχεία (systolic array), όπου το καθένα λειτουργεί όμοια, ανεξάρτητα και παράλληλα, μέσα σε μία αρχιτεκτονική σχεδίασης εκμεταλλευόμενη την παραλληλία των δεδομένων. Στόχος της διατριβής αυτής, είναι η μεταφορά ολόκληρης της λογικής του αλγορίθμου σε μία ενσωματωμένη πλατφόρμα υλικού, μεταφέροντας και προσαρμόζοντας τόσο το λογισμικό, όσο και το υλικό, ώστε να λειτουργήσουν αποτελεσματικά σε ένα ενσωματωμένο σύστημα,

Βέβαια, η εφαρμογή του αλγορίθμου στην παρούσα διατριβή δεν προσανατολίζεται στον τομέα της βιοπληροφορικής, απ' όπου προέρχεται ο συγκεκριμένος αλγόριθμος, αλλά δανείζεται ιδέες από αυτόν για την εφαρμογή του στην ψηφιακή επεξεργασία σήματος. Ο στόχος είναι να δημιουργηθεί μία αποτελεσματική μηχανή ανίχνευσης γνώριμων ηχητικών γεγονότων μέσα σε μεγάλες ροές ηχητικών δεδομένων, όπως είναι μία ταινία. Βέβαια, για την προσαρμογή του αλγορίθμου στο δικό μας πλαίσιο έρευνας, χρειάστηκε να λάβουμε υπόψη ότι εδώ ο αλγόριθμος Smith-Waterman, δεν καλείται να βρει ομοιότητες μεταξύ ενός πεπερασμένου εύρους χαρακτηριστικών, όπως έχει μία αλυσίδα DNA, αλλά να αντιμετωπίσει ένα πολύ μεγαλύτερο εύρος και πιο πολύπλοκης αριθμητικής αναπαράστασης χαρακτηριστικά του ήχου.

Η συγκεκριμένη μεταπτυχιακή διατριβή στηρίζεται σε μεγάλο βαθμό σε προηγούμενη υλοποίηση του εν λόγω αλγορίθμου στα πλάισια της μεταπτυχιακής διατριβής [2]. Στην συγκεκριμένη διατριβή, έχει υλοποιηθεί ο αλγόριθμος Smith-Waterman, συνδυάζοντας τόσο λογισμικό, που εκτελείται σε Η/Υ γενικού σκοπού με λειτουργικό σύστημα linux, όσο και υλικό, που υλοποιεί τον επιταχυντή υπολογισμών σε FPGA. Η διασύνδεσή τους πραγματοποιείται μέσω της θύρας PCIe της αναπτυξιακής πλακέτας, μέσω της οποίας γίνεται η επικοινωνία και αποστολή δεδομένων μεταξύ των δύο συστατικών του συστήματος. Βασισμένοι σε αυτή την υλοποίηση, στην παρούσα διατριβή καλούμαστε να μεταφέρουμε αυτήν την εφαρμογή σε μια ολοκληρωμένη ενσωματωμένη πλατφόρμα, διευρευνώντας τις ιδιαιτερότητες και τα πλεονεκτήματα που αυτό προσφέρει.

Στο επόμενο τμήμα, ακολουθεί μία σύντομη αναφορά στα θέματα τα οποία πραγματεύεται η παρούσα διατριβή.

Στο κεφάλαιο 2 – Χρήση FPGA για επιτάχυνση υπολογισμών σε ένα ενσωματωμένο σύστημα, αφού περιγραφούν βασικές έννοιες της τεχνολογίας των ενσωματωμένων συστημάτων και των συσκευών FPGA, καθώς και η ολοένα και αυξανόμενη χρήση αυτών σε πολλαπλές πλευρές της σύγχρονης πραγματικότητας, αναπτύσσονται τα γενικότερα οφέλη χρησιμοποίησης των FPGA στην επιτάχυνση πολύπλοκων προβλημάτων. Συγκεκριμένα, παρουσιάζονται τα επιπλέον οφέλη χρησιμοποίησης ενσωματωμένων επεξεργαστών σε FPGA για την επίτευξη καλύτερων χρόνων εκτέλεσης πολύπλοκων αλγορίθμων, ενώ τέλος παρουσιάζεται η λύση της κατασκευής, στο υλικό, επιταχυντών αλγορίθμων, ικανών να προσδώσουν στο υπάρχον ενσωματωμένο σύστημα καλύτερη απόδοση και επιτάχυνση, σε σύγκριση με τους ίδιους αλγορίθμους εκτελεσμένους σε λογισμικό. Για πολλούς λόγους, είναι πιθανόν ένας αλγόριθμος γραμμένος για να εκτελεστεί αποκλειστικά με χρήση λογισμικού, να μην μπορεί να ικανοποιήσει τις χρονικές απαιτήσεις που θέτει ο προγραμματιστής/χρήστης,

ακόμα και εάν αυτός εκτελεστεί σε μια μηχανή υψηλών επιδόσεων. Ο ίδιος αλγόριθμος, ωστόσο, αν υλοποιηθεί ως υλικό (hardware) με χρήση επιταχυντών έχει πολύ καλύτερες προοπτικές ταχύτητας. Η τεχνολογία των FPGA συνοδεύεται από την ιδέα του επαναπρογραμματιζόμενου υλικού, γεγονός που συμβάλει στη βελτίωση των χρόνων εκτέλεσης.

Στο κεφάλαιο 3 - Ανάπτυξη αλγορίθμων ευθυγράμμισης ακολουθίας σε ενσωματωμένα συστήματα, γίνεται μία εισαγωγή στους αλγορίθμους ευθυγράμμισης ακολουθίας, του πεδίου εφαρμογής τους, της χρησιμότητάς τους, καθώς και των δυσκολιών εφαρμογής τους στο λογισμικό. Στη συνέχεια, μελετάμε τις διάφορες μεθόδους αλγορίθμων ευθυγράμμισης ακολουθίας, ενώ παράλληλα γίνεται μια παρουσίαση των πλατφορμών υλικού στις οποίες έχουν γίνει μέχρι τώρα προσπάθειες υλοποίησής τους. Μετά από σύγκριση των γενικότερων οφελών που προκύπτουν από την υλοποίηση σε κάθε τεχνολογία, εξηγούμε γιατί τελικά προτιμήσαμε να αναπτύξουμε ένα τέτοιο αλγόριθμο με τη χρήση της τεχνολογίας των FPGA. Αναλυτικότερα, οι υλοποιήσεις σε πλατφόρμες υλικού FPGA βασίζονται στην τεχνική δημιουργίας συστολικού πίνακα (systolic array) για την επίτευξη της επιτάχυνσης του υλικού. Ο συστολικός πίνακας αποτελείται από μία συστοιχία δομικών στοιχείων που ονομάζονται στοιχεία επεξεργασίας (Processing Elements – PE). Κάθε στοιχείο επεξεργασίας σε κάθε στάδιο λαμβάνει δεδομένα από έναν ή περισσότερους γείτονες, τα επεξεργάζεται και, στο επόμενο στάδιο, εξάγει τα αποτελέσματα προς τους απέναντι γείτονες, δίνοντας τη δυνατότητα στο συστολικό πίνακα να επεξεργάζεται τεράστιο όγκο δεδομένων παράλληλα. Τέλος, γίνεται μια λεπτομερής ανάλυση στη παράλληλη σχεδίαση του υλικού με χρήση συστολικού πίνακα, για να μπορέσει στη συνέχεια ο αναγνώστης να κατανοήσει τη δημιουργία του συστολικού πίνακα που χρησιμοποιήθηκε στο πλαίσιο της παρούσας διπλωματικής διατριβής για την υλοποίηση του αλγορίθμου Smith-Waterman.

Στο κεφάλαιο 4 – Ο αλγόριθμος δυναμικού προγραμματισμού Smith-Waterman, μελετούμε έναν συγκεκριμένο αλγόριθμο ευθυγράμμισης ακολουθίας, από το κλάδο της βιοπληροφορικής, αυτό των Smith-Waterman. Πρόκειται για έναν δυναμικό αλγόριθμο προγραμματισμού, αφού διασπά το αρχικό πρόβλημα σε πολλά μικρότερα και απλούστερα υποπροβλήματα, για καθένα από αυτά τα υποπροβλήματα υπολογίζεται μία λύση, ενώ το τελικό αποτέλεσμα προκύπτει από το συνδυασμό των επιμέρους λύσεων των προβλημάτων. Αρχικά, λοιπόν, παρουσιάζεται η απαραίτητη θεωρία για την κατανόηση του αλγορίθμου, ενώ αργότερα παρουσιάζεται πως μπορεί να υλοποιηθεί ο αλγόριθμος των Smith- Waterman, αξιοποιώντας την λογική του συστολικού πίνακα, ώστε να αναπτυχθεί ένας επιταχυντής υπολογισμών.

Στο κεφάλαιο 5 – Περιγραφή συστήματος – εφαρμογής, παρουσιάζονται όλα τα κομμάτια της υλοποίησης που αφορούν την εφαρμογή του αλγορίθμου S-W για την ανάλυση ηχητικού εφέ κατά μήκος μίας ταινίας. Εδώ, αρχικά περιγράφουμε την υλοποιημένη αρχιτεκτονική του S-W επιταχυντή υπολογισμών μέσω της FPGA τεχνολογίας, τα συστατικά στοιχεία από τα οποία αποτελείται η σχεδίασή μας, τόσο αναλύοντας την μονάδα επεξεργασίας (PE) του συστολικού πίνακα, όσο και τις διεργασίες του εκτελεί ο συστολικός πίνακας στο σύνολό του. Ο επιταχυντής υπολογισμών εντάσσεται ως περιφερειακό στο ενσωματωμένο σύστημα, το οποίο περιέχει ήδη έναν ενσωματωμένο επεξεργαστή, που επικοινωνεί με τον επιταχυντή μέσω ειδικού διαύλου επικοινωνίας. Επιπρόσθετα, για την ορθή και γρήγορη διασύνδεση των δύο παραπάνω συστατικών, υλοποιούμε τον επιταχυντή κάνοντας χρήση πλήθους καταχωρητών και μνημών, γνωστών και προσβάσιμων στον ενσωματωμένο επεξεργαστή, μειώνοντας τον χρόνο που θα απαιτούνταν για διαδικασίες εισαγωγής και εξαγωγής δεδομένων (I/O) μεταξύ των δύο, Όλα τα παραπάνω, αναλύονται διεξοδικά στην περιγραφή της ροής του τελικού συστήματος και την αλληλεπίδραση των συστατικών μεταξύ τους. Τέλος, περιγράφεται τόσο η πλατφόρμα υλικού FPGA, όσο και τα εργαλείου σχεδιασμού και ανάπτυξης, που χρησιμοποιήθηκαν για την ανάπτυξη της εν λόγω εφαρμογής.

Στο κεφάλαιο 6, αναλύονται τα συμπεράσματα που προκύπτουν από το σύνολο της παρούσας διατριβής, ενώ τέλος παρατίθενται δύο παραρτήματα, που αντίστοιχα αναλύουν την δημιουργία του περιφερειακού του επιταχυντή υπολογισμών στη σουίτα εργαλείων σχεδίασης που χρησιμοποιήθηκε, αλλά και του συστήματος αρχείων μνήμης (memory file system), στο οποίο αποθηκεύτηκαν και ‘κατέβηκαν’ στη συσκευή FPGA τα απαραίτητα για την υλοποίηση αρχεία της ταινίας και του ηχητικού εφέ προς σύγκριση, για την υλοποίηση του αλγορίθμου.

2. Χρήση FPGA για επιτάχυνση υπολογισμών σε ένα ενσωματωμένο σύστημα

2.1. Ενσωματωμένα Συστήματα

Ένα ενσωματωμένο σύστημα είναι ένα υπολογιστικό σύστημα ειδικού σκοπού, σχεδιασμένο έτσι ώστε να εκτελεί μια ή και περισσότερες συναρτήσεις, συνήθως σε σταθερές πραγματικού χρόνου. Είναι, συνήθως, ενσωματωμένο σαν ένα μέρος μιας ολοκληρωμένης συσκευής, περιλαμβάνοντας hardware, καθώς και μηχανικά μέρη. Σε αντίθεση, ένα υπολογιστικό σύστημα γενικού σκοπού, όπως ένας προσωπικός υπολογιστής, ανάλογα με τον προγραμματισμό που του έχει γίνει, μπορεί να εκτελέσει πολλά διαφορετικά καθήκοντα. Από τη στιγμή που ένα σύστημα είναι προσανατολισμένο σε συγκεκριμένα καθήκοντα, οι μηχανικοί σχεδίασης μπορούν να το βελτιστοποιήσουν, μειώνοντας το μέγεθος και το κόστος του.

Η ενσωματωμένη επεξεργασία με FPGA έχει γίνει αναπόσπαστο μέρος ενός ολοένα και αυξανόμενου αριθμού εφαρμογών, όπως η βιομηχανική δικτύωση και εφαρμογές βίντεο, καθώς και τα κλειστά συστήματα ελέγχου σε βιομηχανικές, αεροδιαστημικές και αμυντικές (A&D) αγορές. Ο σχεδιασμός σε ενσωματωμένο σύστημα δημιουργεί βαθύτερα επίπεδα ολοκλήρωσης και αξιοποίησης της επίδοσης παράλληλης επεξεργασίας, ενώ παράλληλα συμβάλλει στην επαναχρησιμοποίηση του υλικού. Επιπλέον, μετριάζονται οι κίνδυνοι σχεδιασμού και μειώνεται το κόστος, το βάρος, το μέγεθος και την κατανάλωση ενέργειας που απαιτείται για την υλοποίηση μίας εφαρμογής.

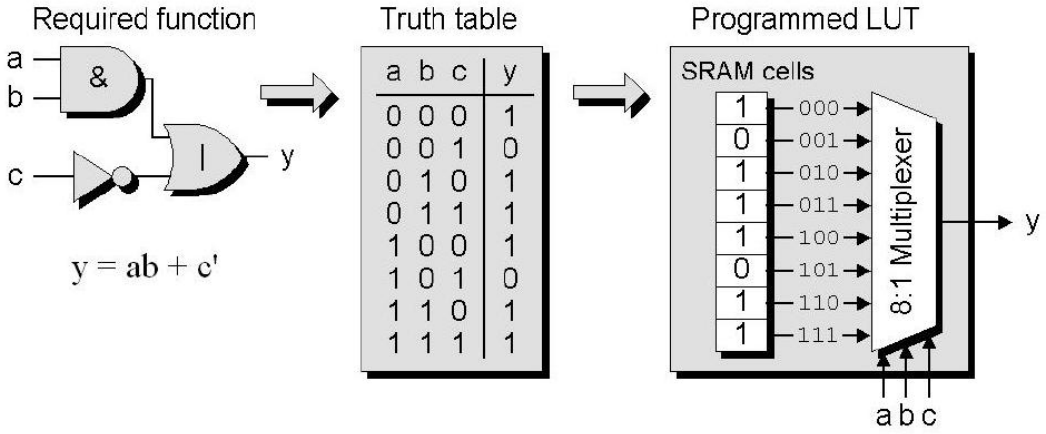
Οι πλατφόρμες ενσωματωμένων συστημάτων επιτρέπουν ένα ευρύ φάσμα εφαρμογών σε πολλαπλά τμήματα της αγοράς, συμπεριλαμβανομένων την αυτοματοποίηση, τη βιομηχανική δικτύωση, την ιατρική απεικόνιση, τα όργανα ελέγχου και μετρήσεων, τις στρατιωτικές υπηρεσίες, καθώς επίσης και το αυτοματοποιημένα συστήματα υποβοήθησης οδηγού αυτοκινήτου.

2.2. Συσκευές FPGA

Τα FPGAs (Επιτόπου Προγραμματιζόμενες Διατάξεις Πυλών - Field Programmable Gate Arrays) είναι διατάξεις προγραμματιζόμενης λογικής και υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων, ενώ ταυτόχρονα επιτρέπουν πλήρη ελευθερία όσον αφορά τη διαδικασία σχεδίασης. Οι συσκευές FPGAs είναι συσκευές γενικής χρήσης, οι οποίες εμπεριέχουν ένα μεγάλο αριθμό λογικών στοιχείων (LE, Logic Element), καλωδίων διασύνδεσης και διακοπτών.

Στη σημερινή αγορά, υπάρχει πληθώρα από διαφορετικές ποικιλίες των συσκευών αυτών, όπου ολοένα αναπτύσσονται και εξελίσσονται με γρήγορους ρυθμούς. Παράλληλα, βλέπουμε ένα αυξανόμενο ενδιαφέρον από ακαδημαϊκής πλευράς, στη μελέτη αυτής της τεχνολογίας μιας και αποτελεί τη σύγχρονη μορφή των ψηφιακών συστημάτων και της ψηφιακής σχεδίασης.

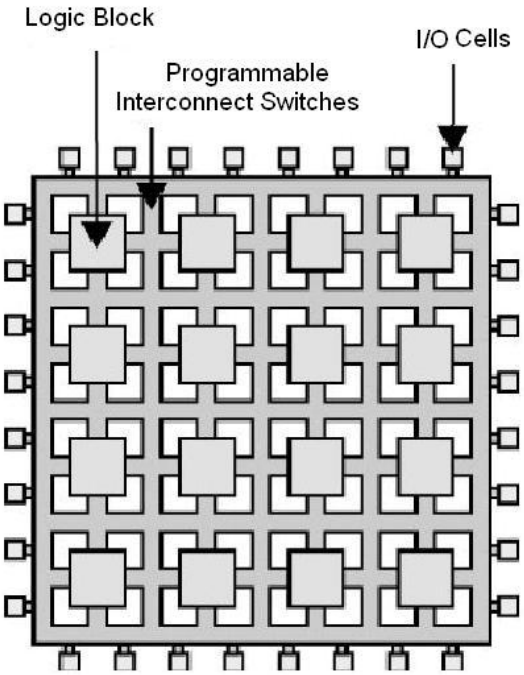
Η αρχιτεκτονική δομή των FPGAs διαφέρει σημαντικά από αυτή των διατάξεων SPLD και CPLD επειδή δεν περιέχουν πύλες AND και OR, αλλά περιέχει *λογικές βαθμίδες* για την υλοποίηση των ζητούμενων συναρτήσεων. Η πιο ευρέως χρησιμοποιημένη λογική βαθμίδα είναι ο *πίνακας αναζήτησης* (*LUT, Lookup Table*), Εικόνα 2-1, ο οποίος περιέχει *κυψέλες αποθήκευσης* (*storage cells*) που χρησιμοποιούνται για την υλοποίηση μιας μικρής συνάρτησης. Κάθε κυψέλη είναι μια μνήμη τύπου *SRAM* (*Static Random Access Memory*) και μπορεί να αποθηκεύσει μια λογική τιμή, 0 ή 1. Η αποθηκευμένη τιμή μεταφέρεται στην έξοδο της κυψέλης αποθήκευσης. Μπορούν να αναπτυχθούν πίνακες LUT σε διάφορα μεγέθη, όπου το μέγεθος ορίζεται από τον αριθμό των εισόδων. Ένα LUT n-εισόδων μπορεί να υλοποιήσει όλες τις πιθανές συνδυαστικές συναρτήσεις των n-εισόδων, προσθέτοντας μία ακόμη είσοδο είναι δυνατή η αναπαράσταση πιο σύνθετων συναρτήσεων. Μελέτες έχουν δείξει ότι τα LUT 4-εισόδων είναι μία “καλή” λύση.



Εικόνα 2-1: Πίνακας αναζήτησης (LUT, Lookup Table) αρχιτεκτονικής FPGA.

Οι κυψέλες αποθήκευσης των πινάκων LUT είναι *πτητικές* ή *μη μόνιμες* (*volatile*), αυτό συνεπάγεται ότι χάνουν τα δεδομένα που περιέχουν, εάν διακοπεί η τροφοδοσία του ολοκληρωμένου κυκλώματος. Δεδομένου αυτού του προβλήματος, θα έπρεπε το FPGA να επαναπρογραμματίζεται κάθε φορά που θα διακόπτοταν η τροφοδοσία στο κύκλωμα. Για την αποφυγή αυτού του προβλήματος, συμπεριλαμβάνεται συχνά μαζί με τη μητρική πλακέτα του FPGA και ένα μικρό ολοκληρωμένο κύκλωμα μη-πτητικής μνήμης. Αυτή η μνήμη είναι της μορφής PROM ή EPROM ή EEROM ή οτιδήποτε άλλο. Η μνήμη αυτή διατηρεί σε μόνιμη βάση τα δεδομένα της διαμόρφωσης, τα οποία έχουν εισέλθει μέσω της θύρας JTAG. Κάθε φορά που γίνεται τροφοδοσία στη συσκευή FPGA, διαβάζεται η μνήμη και διαμορφώνεται η συσκευή με τη συγκεκριμένη λειτουργία για την οποία σχεδιάστηκε.

Η γενική αρχιτεκτονική δομή των FPGAs είναι αυτή της Εικόνας 2-2, όπου μπορεί κανείς να παρατηρήσει ότι αποτελείται από τρεις χαρακτηριστικές ομάδες. Η πρώτη ομάδα αποτελείται από τους *ακροδέκτες εισόδου εξόδου* (*pins, I/O Cells*), οι οποίοι είναι και η διασύνδεση της συσκευής με τον έξω κόσμο, αποτελούν δηλαδή την είσοδο και την έξοδο του FPGA. Η δεύτερη ομάδα αποτελείται από τις *λογικές βαθμίδες* (*LB*, *Logic Block*), όπου η καθεμία από αυτές περιέχει ένα σύνολο από λογικά στοιχεία (ο αριθμός διαφέρει ανα αρχιτεκτονική). Και τέλος, η τρίτη ομάδα με τους *προγραμματιζόμενους διακόπτες διασύνδεσης* (*PIS*, *Programmable Interconnect Switches*) ή αλλιώς *κανάλια δρομολόγησης* (*Routing Channels*). Η τελευταία ομάδα είναι αυτή που διασύνδεει τις λογικές βαθμίδες (LB) μεταξύ τους, όπως και με τους ακροδέκτες (I/O). Η ποσότητα αυτών των χαρακτηριστικών (I/O, LE, LB,PIS κτλ) διαφέρει από αρχιτεκτονική σε αρχιτεκτονική, όμως η γενική δομή παραμένει η ίδια.

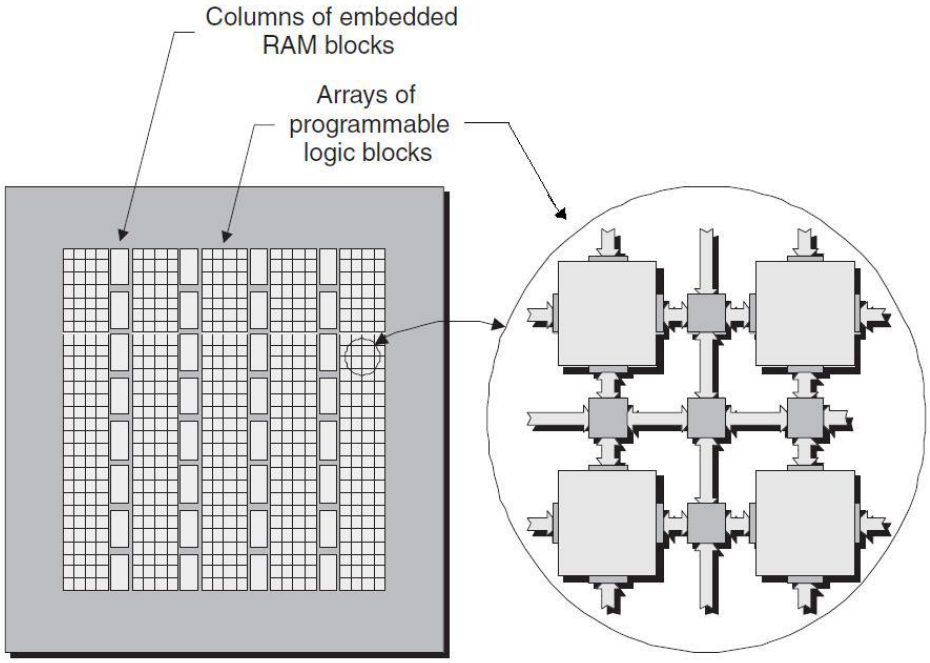


Εικόνα 2-2: Γενική αρχιτεκτονική δομή FPGA

Επιπλέον ενσωματωμένα χαρακτηριστικά, τα οποία δε φαίνονται στην παραπάνω εικόνα είναι τα ακόλουθα:

• Μνήμες (*Memories*) RAM

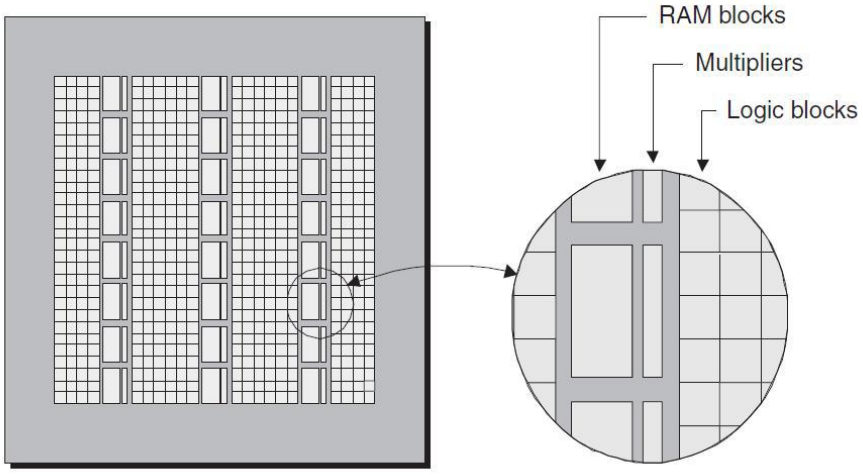
Μνήμες (*Memories*) ή *βαθμίδες μνημών* (*MB, Memory Block*), μπορεί να αποτελούν μέρος της αρχιτεκτονικής δομής, ανάμεσα σε λογικές βαθμίδες (LB) ή περιφερειακά της συσκευής FPGA (βλέπε Εικόνα 2-3). Αυτές οι βαθμίδες μνημών (MB) εξυπηρετούν τοπικές ανάγκες του κυκλώματος παρέχοντας επιπλέον μνήμη στις λογικές βαθμίδες. Είναι συνήθως μνήμες RAM και λόγω της ανάγκης για μεγάλη διάθεση μνήμης, πολλές από τις συσκευές FPGA ενσωματώνουν βαθμίδες από RAM (*RAM Block*). Κάθε βαθμίδα RAM μπορεί να χρησιμοποιηθεί ανεξάρτητα ή να συνδυαστούν πολλές μαζί, ώστε να υλοποιήσουν μία μεγαλύτερη μνήμη. Επιπρόσθετα. οι μνήμες αυτές μπορούν να χρησιμοποιηθούν σε ποικιλία εφαρμογών όπως, Single Port Ram, Dual Port Ram, FIFO κτλ.



Εικόνα 2-3: Ενσωματωμένες μνήμες RAM σε FPGA

• Αριθμητικές μονάδες (*Arithmetic Units*)

Κάποιες αριθμητικές συναρτήσεις, όπως ο πολλαπλασιασμός, είναι αργές όταν υλοποιηθούν από έναν αριθμό από λογικά στοιχειά. Για αυτό το λόγο, ενσωματώνονται αριθμητικές μονάδες στην αρχιτεκτονική δομή των FPGAs, έτσι ώστε να επιταχύνουν τη διαδικασία (Εικόνα 2-4). Πολλές διατάξεις FPGA ενσωματώνουν τέτοιες αριθμητικές μονάδες, όπως πολλαπλασιαστές (multipliers), αθροιστές (Adders), πολλαπλασιαστές - συσσωρευτές (MAC, Multiple-Accumulator) κτλ.

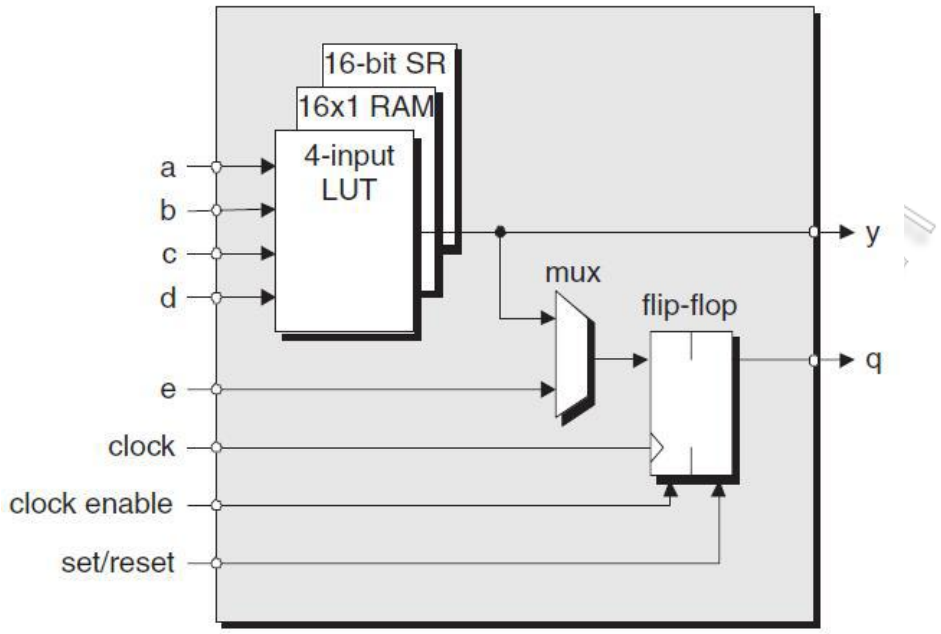


Εικόνα 2-4: Ενσωματωμένες αριθμητικές μονάδες σε FPGA

Οι ενσωματωμένες αριθμητικές μονάδες μαζί με τις ενσωματωμένες μνήμες κάνουν ιδανική τη χρήση των FPGAs σε εφαρμογές *ψηφιακής επεξεργασίας σήματος* (*DSP, Digital Signal Processing*).

• Λογικό στοιχείο (*LE, Logic Element*)

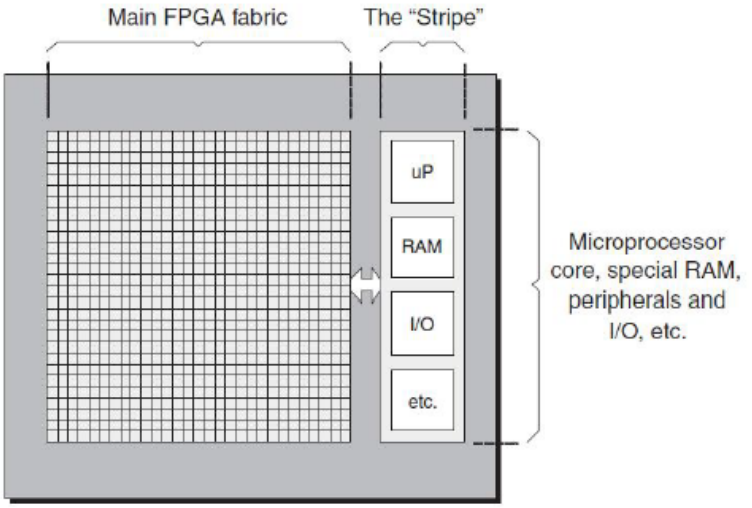
Το λογικό στοιχείο αποτελεί την ελάχιστη λογική μονάδα στην οποία μπορεί να αποθηκευτεί μια συνάρτηση ή ένα δεδομένο. Ένα λογικό στοιχείο αποτελείται από ένα πίνακα αναζήτησης (Lookup Table, LUT), ο οποίος όμως μπορεί να διαφέρει στον αριθμό των εισόδων (Εικόνα 2.11). Επιπλέον, ένα λογικό στοιχείο περιέχει πληθώρα από λογικές πύλες, κυκλώματα όπως και κύτταρα μνήμης flip flop (Εικόνα 2-5). Είναι αναμενόμενο, λοιπόν, η αρχιτεκτονική κάθε λογικού στοιχείου να διαφέρει από εταιρεία σε εταιρεία και από οικογένεια FPGA σε οικογένεια.



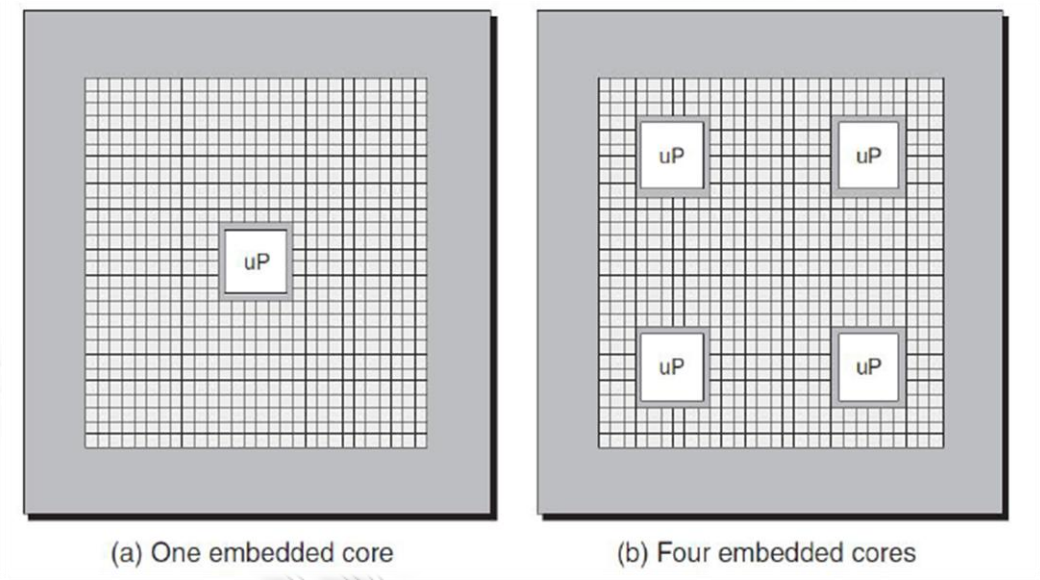
Εικόνα 2-5: Λογικό στοιχείο της εταιρείας Xilinx

2.3. Ενσωματωμένοι επεξεργαστές στο FPGA

Επεξεργαστές (Microprocessors) καλούνται οι πυρήνες επεξεργαστών που ενσωματώνουν οι προηγμένες διατάξεις FPGA, για χρήση σε πολύ απαιτητικές λειτουργίες, όπου χρειάζεται ένας γρήγορος επεξεργαστής. Υπάρχουν δύο τύποι ενσωματωμένων επεξεργαστών, οι σκληρού πυρήνα (hard core) και οι μαλακού πυρήνα (soft core). Οι επεξεργαστές μαλακού πυρήνα υλοποιούνται μέσω των λογικών στοιχειών της διάταξης του FPGA και μπορούν να τροποποιηθούν και να προσαρμοστούν ανάλογα με τις ανάγκες του σχεδιαστή υλικού. Οι σκληρού πυρήνα επεξεργαστές ενσωματώνονται εκ των προτέρων στην διάταξη της συσκευής και δεν μπορούν να τροποποιηθούν στη συνέχεια από το χρήστη. Τέλος, αξίζει να αναφέρουμε ότι ο αριθμός των επεξεργαστών που μπορούν να ενσωματωθούν σε μία συσκευή FPGA εξαρτάται από την εκάστοτε αρχιτεκτονική.



Εικόνα 2-6: Σκληρός πυρήνας (hard core) ενός FPGA έξω από την κύρια αρχιτεκτονική δομή του.



Εικόνα 2-7: Ενσωματωμένοι επεξεργαστές σε διαφορετικές αρχιτεκτονικές δομές.

2.4. Πλεονεκτήματα ενός ενσωματωμένου επεξεργαστή σε FPGA

Ένα σύστημα FPGA ενσωματωμένου επεξεργαστή προσφέρει πολλά πλεονεκτήματα σε σύγκριση με τους τυπικούς μικροεπεξεργαστές, όπως αναφέρονται τα παρακάτω:

1) παραμετροποίηση
2) μετριασμός απαρχαίωσης
3) μείωση συστατικών και κόστους και
4) την επιτάχυνση υλικού

2.4.1. Παραμετροποίηση

Ο σχεδιαστής ενός συστήματος ενσωματωμένου επεξεργαστή FPGA έχει πλήρη ευελιξία να επιλέξει οποιονδήποτε συνδυασμό περιφερειακών και ελεγκτών θεωρεί ότι ταιριάζει στην υλοποίησή του. Στην πραγματικότητα, ο σχεδιαστής έχει τη δυνατότητα να εφεύρει νέα, μοναδικά περιφερειακά που μπορεί να συνδεσεί απευθείας στον δίαυλο του επεξεργαστή. Εάν ένας σχεδιαστής έχει μια μη τυπική απαίτηση για ένα σετ περιφερειακών, η απαίτηση αυτή μπορεί εύκολα να πραγματοποιηθεί με ένα σύστημα FPGA ενσωματωμένου επεξεργαστή, καθιστώντας το ολοκληρωμένο ενσωματωμένο σύστημα σχεδόν απόλυτα προσαρμοσμένο στις ανάγκες του χρήστη. Για παράδειγμα, ένας σχεδιαστής δεν θα βρει εύκολα έναν επεξεργαστή με δέκα UARTs. Ωστόσο, σε ένα FPGA, αυτή η ρύθμιση επιτυγχάνεται πολύ εύκολα.

2.4.2. Μετριασμός της απαρχαίωσης του σχεδιασμού

Ορισμένες εταιρίες, ιδίως εκείνες που υποστηρίζουν στρατιωτικά συμβόλαια, έχουν τη σχεδιαστική απαίτηση να διασφαλιστεί η διάρκεια ζωής του προϊόντος, που είναι πολύ μεγαλύτερη από τη διάρκεια ζωής ενός προτύπου ηλεκτρονικών προϊόντων. Ο μετριασμός της απαρχαίωσης των συστατικών είναι ένα δύσκολο θέμα για ένα ολοκληρωμένο σύστημα. Οι FPGA soft-επεξεργαστές (‘soft’ καλούνται οι επεξεργαστές που χτίζονται με τη γενικής χρήσης λογική του FPGA, οι οποίοι σε αντίθεση με το ‘hard’ επεξεργαστή, πρέπει να συντεθούν και να αντιστοιχιστούν στην δομή του FPGA) είναι μια εξαιρετική λύση σε αυτή την περίπτωση. Λόγω του ότι οι ‘soft’ επεξεργαστές περιγράφονται συνήθως σε κάποια γλώσσα περιγραφής υλικού (HDL) ή λίστα σημάτων, αν ο σχεδιαστής/πελάτης εξασφαλίσει τα πηγαία αρχεία σχεδιασμού HDL του ‘soft’ επεξεργαστή εκπληρώνεται και η απαίτηση για εγγύηση διάρκειας ζωής του προϊόντος.

2.4.3. Μείωση συστατικών και κόστους

Με την ευελιξία του FPGA, προηγούμενα συστήματα που απαιτούσαν πολλαπλά εξαρτήματα μπορούν να αντικατασταθούν με ένα ενιαίο FPGA. Σίγουρα αυτό συμβαίνει, όταν ένα βοηθητικό I/O chip ή ένας συν-επεξεργαστής απαιτείται δίπλα σε έναν επεξεργαστή. Με τη μείωση του αριθμού των συστατικών σε ένα σχέδιο, μια εταιρεία μπορεί να μειώσει το μέγεθος της πλακέτας και τη διαχείριση των αποθεμάτων, τα οποία θα εξοικονομήσουν χρόνο και κόστος σχεδιασμού.

2.4.4. Επιτάχυνση υλικού

Ίσως ο πιο επιτακτικός λόγος για να επιλέξει κανείς έναν FPGA ενσωματωμένο επεξεργαστή είναι η ικανότητα να κάνει εναλλαγές (tradeoffs) μεταξύ του υλικού και του λογισμικού για να μεγιστοποιήσει την αποτελεσματικότητα και την απόδοση του συστήματος. Εάν ένας αλγόριθμος έχει αναγνωριστεί να έχει σημεία συμφόρησης όταν εκτελείται σε λογισμικό, ένας συν-επεξεργαστής (co-processor), ειδικός για τον εν λόγω αλγόριθμο, μπορεί να σχεδιαστεί στο FPGA. Ο συν-επεξεργαστής αυτός μπορεί να συνδεθεί με τον FPGA ενσωματωμένο επεξεργαστή μέσω ειδικών, χαμηλής χρονικής καθυστέρησης καναλιών, ενώ παράλληλα μπορούν να θεσπιστούν ειδικές εντολές για την εξάσκηση του συν-επεξεργαστή. Με την ίδια λογική, αντί του συν-επεξεργαστή, μπορεί να σχεδιαστεί ένας επιταχυντής υπολογισμών στο υλικό, η λειτουργία του οποίου θα ελέγχεται από καταχωρητές. Και αυτός, ωστόσο, μπορεί να συνδεθεί με τον ενσωματωμένο επεξεργαστή, δίνοντας έτσι την δυνατότητα στη σχεδίαση να εκμεταλλευτεί την συνύπαρξη υλικού και λογισμικού, μεταφέροντας το πρόβλημα που αντιμετωπίζει το λογισμικό, στην συγκεκριμένη εφαρμογή, στο υλικό. Με τα σύγχρονα FPGA εργαλεία σχεδιασμού υλικού, η μετάβαση της συμφόρησης λογισμικού από το λογισμικό στο υλικό έχει γίνει πολύ πιο εύκολη, δεδομένου ότι το λογισμικό σε γλώσσα προγραμματισμού C μπορεί εύκολα να προσαρμοστεί στο υλικό με μόνο μικρές αλλαγές στον κώδικα.

2.5. Υλοποίηση στο υλικό

Αν και τα πλεονεκτήματα της ενσωμάτωσης ενός επεξεργαστή στο FPGA ενσωματωμένο σύστημα, είναι πέραν από εμφανή, η επιτάχυνση του υλικού κ η μείωση του κόστους δεν επέρχονται μόνο και μόνο με τη χρήση του ενσωματωμένου επεξεργαστή. Σε αντίθεση με ένα έτοιμο επεξεργαστή, η πλατφόρμα υλικού για τον ενσωματωμένο επεξεργαστή πρέπει να σχεδιαστεί. Το γεγονός αυτό, αυξάνει την πολυπλοκότητα του σχεδιασμού, καθότι ο σχεδιασμός της πλατφόρμας υλικού απαιτεί την χρήση πολύπλοκων εργαλείων σχεδιασμού που ενσωματώνουν τόσο το υλικό που θα διαμορφώσει το FPGA, όσο και το λογισμικό που θα γραφεί για να εκτελεστεί από τον ενσωματωμένο επεξεργαστή, απαιτώντας πολλή μεγαλύτερη προσοχή από το σχεδιαστή του ενσωματωμένου συστήματος.

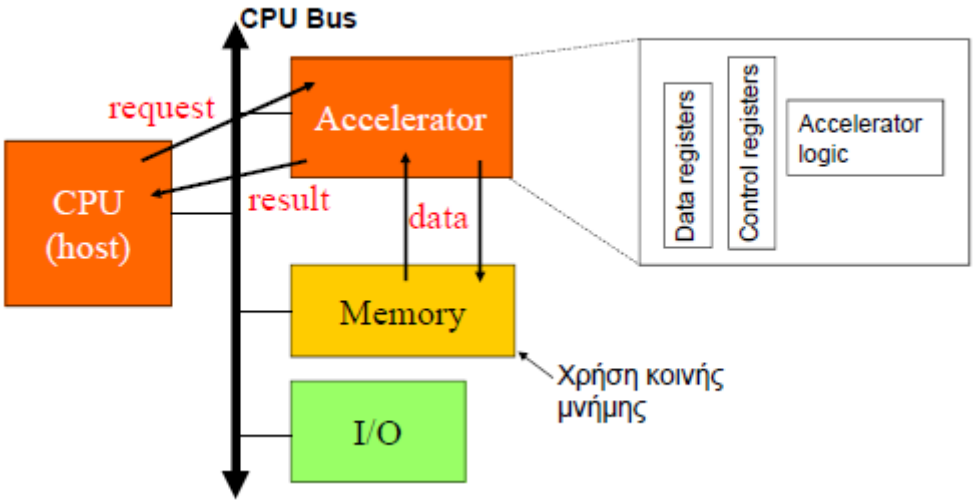
Συμπερασματικά, ωστόσο, μπορούμε να πούμε ότι ένας FPGA ενσωματωμένος επεξεργαστής έχει την ισχύ και την ικανότητα να επιφέρει μία προηγουμένως ακατόρθωτη απόδοση και ευελιξία στο σύστημα. Με τη χρήση του FPGA, ένας σχεδιαστής μπορεί να καθορίσει ακριβώς τα περιφερειακά που απαιτούνται για ένα δεδομένο σύστημα, ενώ ταυτόχρονα μπορεί να προσαρμόσει την εφαρμογή λογισμικού του (συνήθως γραμμένη σε γλώσσα C), έτσι ώστε να ξεπεράσει τυχόν προβλήματα συμφόρησης κατά την εκτέλεση των εντολών του προγράμματος δημιουργώντας, είτε μια επιπλέον προσαρμοσμένη hardware μονάδα επεξεργασίας, είτε έναν επιταχυντή υπολογισμών με τη μορφή ενός επιπλέον περιφερειακού που επικοινωνεί με τον ενσωματωμένο επεξεργαστή. Όλα τα παραπάνω πλεονεκτήματα μπορούν να εφαρμοστούν μόνο σε προγραμματιζόμενο υλικό (FPGA).

Αν, επιπρόσθετα με τον ενσωματωμένο επεξεργαστή, ενσωματώσουμε και στη σχεδίασή μας έναν επιταχυντή υπολογισμών κατασκευασμένο στο υλικό (με τη βοήθεια γλωσσών περιγραφής υλικού), ο οποίος θα επικοινωνεί με τον επεξεργαστή, τότε μπορούμε να φτάσουμε σε μεγάλης κλίμακας επιταχύνσεις.

2.5.1. Επιταχυντές

Επιταχυντή καλούμε μία πρόσθετη υπολογιστική μονάδα που προορίζεται για την εκτέλεση ορισμένων κρίσιμων λειτουργιών, μέσω της οποίας αξιοποιούμε την ταυτόχρονη χρήση υλικού και λογισμικού για την επίτευξη των προδιαγραφών του συστήματος. Οι επιταχυντές, σε αντίθεση με τους συν-επεξεργαστές (co-processors), δεν εκτελούν εντολές, αλλά η χρήση τους ελέγχεται από καταχωρητές δεδομένων και ελέγχου, παρόμοια με μια περιφερειακή συσκευή.

Πολλές υλοποιήσεις επιταχυντών έχουν κατασκευαστεί σε Ολοκληρωμένα Κυκλώματα Ειδικής Εφαρμογής (Application-Specific Integrated Circuits, ASICs), κυκλώματα σχεδιασμένα μόνο για συγκεκριμένες εφαρμογές και λειτουργίες, όπως αναφέρει και το όνομά τους, αυτά επιφέρουν πολύ μεγάλο κόστος παραγωγής, το οποίο καλύπτεται μόνο σε περιπτώσεις παραγωγής μεγάλου αριθμού προϊόντων. Ωστόσο, σημαντικές υλοποιήσεις έχουν αναπτυχθεί και σε FPGAs εκμεταλλευόμενοι τον επαναπρογραμματισμό του υλικού, καθώς και το χαμηλότερο κόστος υλοποίησης. Αν και η σχεδίαση ενός επιταχυντή απαιτεί επιπλέον προσπάθεια και διαφορετικές δεξιότητες σε σχέση με τη σχεδίαση ενσωματωμένου λογισμικού, η χρήση ενός επιταχυντή πολλές φορές διαφαίνεται ως η καλύτερη λύση σε εφαρμογές, όπου το λογισμικό αποτυγχάνει να εκτελέσει τις λειτουργίες που πρέπει στον επιθυμητό χρόνο και μας ενδιαφέρει η υλοποίηση μία λύσης με την καλύτερη σχέση κόστους/απόδοσης.



Εικόνα 2-8: Αρχιτεκτονική συστήματος με επιταχυντή.

Συγκεκριμένα, ένας επιταχυντής προτιμάται, καθότι εξαιτίας του γεγονότος ότι η σχεδίαση του είναι προσαρμοσμένη στην προς υλοποίηση εφαρμογή, ο επιταχυντής μπορεί να εκτελέσει γρηγορότερα κάποιες λειτουργίες από ότι ένας επεξεργαστής ίδιου κόστους. Επιπλέον, σε αντίθεση με ένα απλό επεξεργαστή, το κόστος του οποίου είναι μη-γραμμική συνάρτηση της απόδοσης, ο επιταχυντής προσφέρει καλύτερη απόδοση και λειτουργικότητα σε σχέση με το κόστος του. Τέλος, η σχεδίαση του επιταχυντή περιλαμβάνει το διαμερισμό της εφαρμογής σε περισσότερα επεξεργαστικά στοιχεία (Processing Elements - PE) χαμηλότερης απόδοσης και κόστους, γεγονός που καθιστά το κόστος συναρμολόγησης σε επίπεδο συστήματος χαμηλότερο. Σε αυτά τα λιγότερα επιβαρυμένα επεξεργαστικά στοιχεία ανατίθενται, είτε διαμοιρασμένες, είτε εκμεταλλευόμενοι κάποια παραλληλία δεδομένων, οι λειτουργίες που πρέπει να εκτελέσει στο σύνολό του ο επιταχυντής, προσφέροντας καλύτερη απόδοση σε πραγματικό χρόνο.

3. Ανάπτυξη αλγορίθμων ευθυγράμμισης ακολουθίας σε ενσωματωμένα συστήματα

3.1. Εισαγωγή

Στον κλάδο της βιοπληροφορικής, η ευθυγράμμιση ακολουθίας (sequence alignment) αποτελεί ένα τρόπο σύγκρισης δύο σειρών DNA, RNA ή πρωτεϊνών, έτσι ώστε να είναι δυνατή η ανίχνευση περιοχών ομοιότητας μεταξύ των σειρών. Θεωρούμε ότι κάθε τέτοια σειρά αντιστοιχίζεται σε μία ακολουθία χαρακτήρων. Συνεπώς, το πρόβλημα μέτρησης της ομοιότητας ανάμεσα σε δύο ακολουθίες χαρακτήρων μετατοπίζεται στην στοίχισή τους και στον υπολογισμό του κόστους μετατροπής από έναν χαρακτήρα σε έναν άλλο χαρακτήρα.

Πρακτικά, όταν έχουμε να συγκρίνουμε δύο συμβολοσειρές χαρακτήρων, δημιουργείται ένας πίνακας ομοιοτήτων M x N στοιχείων, όπου M αποτελεί το πλήθος χαρακτήρων της πρώτης συμβολοσειράς και N το πλήθος χαρακτήρων της δεύτερης. Συνεπώς, για κάθε πιθανό συνδυασμό χαρακτήρων μεταξύ των δύο συμβολοσειρών έχουμε και μία αντίστοιχη θέση στον παραγόμενο πίνακα . Σε κάθε μία από αυτές τις θέσεις, τρεις περιπτώσεις μπορούν να ισχύουν. Να έχουμε ταίριασμα (match) όταν ο ίδιος χαρακτήρας α είναι παρών και στις δύο συμβολοσειρές. Να μην έχουμε ταίριασμα (mismatch ή substitution), όταν έχουμε δύο διαφορετικούς χαρακτήρες α και β. Τέλος, να έχουμε κενό (gap), όταν στην συγκεκριμένη θέση υπάρχει ένας οποιοσδήποτε χαρακτήρας στη μία συμβολοσειρά (insertion), ενώ στην άλλη δεν υπάρχει κάποιος χαρακτήρας (deletion).

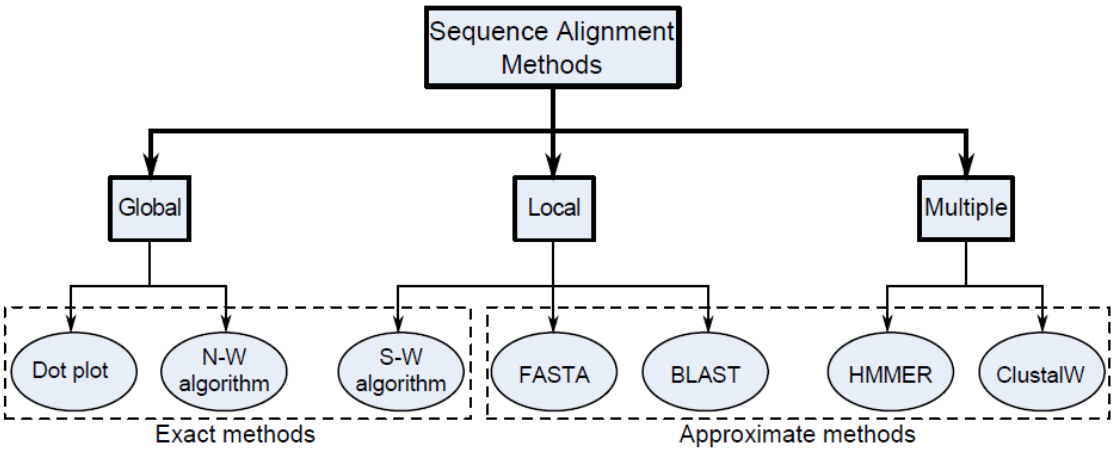
3.2. Μέθοδοι υλοποίησης αλγορίθμων ευθυγράμμισης ακολουθίας

3.2.1. Διαθέσιμες μέθοδοι

Όπως αναφέραμε και προηγουμένως, η ευθυγράμμιση αλληλουχίας (sequence alignment) στοχεύει στον προσδιορισμό περιοχών ομοιότητας μεταξύ δύο αλληλουχιών DNA ή πρωτεϊνών. Παραδοσιακά, οι μέθοδοι της κατά ζεύγη ευθυγράμμισης αλληλουχιών ταξινομούνται ως είτε καθολικές ή τοπικές, όπου η κατά ζεύγη ομοιότητα υπολογίζεται λαμβάνοντας υπόψη μόνο δύο ακολουθίες τη φορά.

Οι μέθοδοι καθολικής ευθυγράμμισης προσπαθούν να φτάσουν σε ταυτοποίηση με όσους περισσότερους χαρακτήρες γίνεται, από άκρη σε άκρη, ενώ οι τοπικές μέθοδοι στοχεύουν στον εντοπισμό μικρών τμημάτων ομοιότητας μεταξύ δύο αλληλουχιών. Ωστόσο, σε ορισμένες περιπτώσεις, μπορεί επίσης να χρειάζεται να ερευνηθούν ομοιότητες μεταξύ μιας ομάδας αλληλουχιών, ως εκ τούτου, να χρειάζεται η εισαγωγή μεθόδων πολλαπλών ευθυγραμμίσεων αλληλουχιών.

Στο σχήμα που ακολουθεί φαίνεται μία ταξινόμηση των διαφόρων διαθέσιμων μεθόδων ευθυγράμμισης ακολουθίας.



Εικόνα 3-1: Διάφορες μέθοδοι ευθυγράμμισης αλληλουχιών.

Οι μέθοδοι αυτές ταξινομούνται σε τρεις τύπους, καθολικής, τοπικής κ πολλαπλής κλίμακας. Επιπλέον, απεικονίζονται και οι αντίστοιχες μέθοδοι που ανήκουν σε κάθε τύπο.

3.2.2. Υλοποίηση σε πλατφόρμες υλικού

Εκμεταλλευόμενοι τα πλεονεκτήματα του υλικού (hardware) έχει σημειωθεί πρόοδος στην επιτάχυνση των μεθόδων ευθυγράμμισης ακολουθίας, με την εφαρμογή τους σε διάφορες διαθέσιμες πλατφόρμες hardware. Στην συνέχεια ακολουθεί μια σύντομη συζήτηση σχετικά με αυτές τις πλατφόρμες.

- CPUs

Οι επεξεργαστές είναι ευρέως γνωστές, ευέλικτες και επεκτάσιμες αρχιτεκτονικές. Με την αξιοποίηση του συνόλου εντολών ροής Streaming SIMD Extension (SSE) στους σύγχρονους επεξεργαστές, έχει επιτευχθεί μία σημαντική μείωση του χρόνου εκτέλεσης των αναλύσεων, καθιστώντας έτσι τις αναλύσεις απαιτητικών σε δεδομένα προβλημάτων, όπως είναι η ευθυγράμμιση ακολουθίας, εφικτές. Επίσης, οι αναδυόμενες τεχνολογίες CPU όπως το multi-core συνδυάζει δύο ή περισσότερους ανεξάρτητους επεξεργαστές σε ένα ενιαίο πακέτο. Η απλή εντολή Multiple Data-stream (SIMD) χρησιμοποιείται σε μεγάλο βαθμό σε αυτή την κατηγορία των επεξεργαστών, καθιστώντας την κατάλληλη για παράλληλες εφαρμογές δεδομένων όπως η ευθυγράμμιση ακολουθιών. Η SIMD περιγράφει επεξεργαστές με στοιχεία πολλαπλής επεξεργασίας που επιτελούν την ίδια λειτουργία σε πολλαπλά δεδομένα ταυτόχρονα. Συνεπώς, τέτοιες μηχανές αξιοποιούν στοιχεία για παραλληλισμό επιπέδου. Έτσι, αυξάνεται σημαντικά η απόδοση αφού ακριβώς οι ίδιες λειτουργίες θα πρέπει να πραγματοποιούνται σε πολλαπλά αντικείμενα δεδομένων, καθιστώντας έτσι την ευθυγράμμιση αλληλουχίας μια τυπική εφαρμογή.

- FPGAs

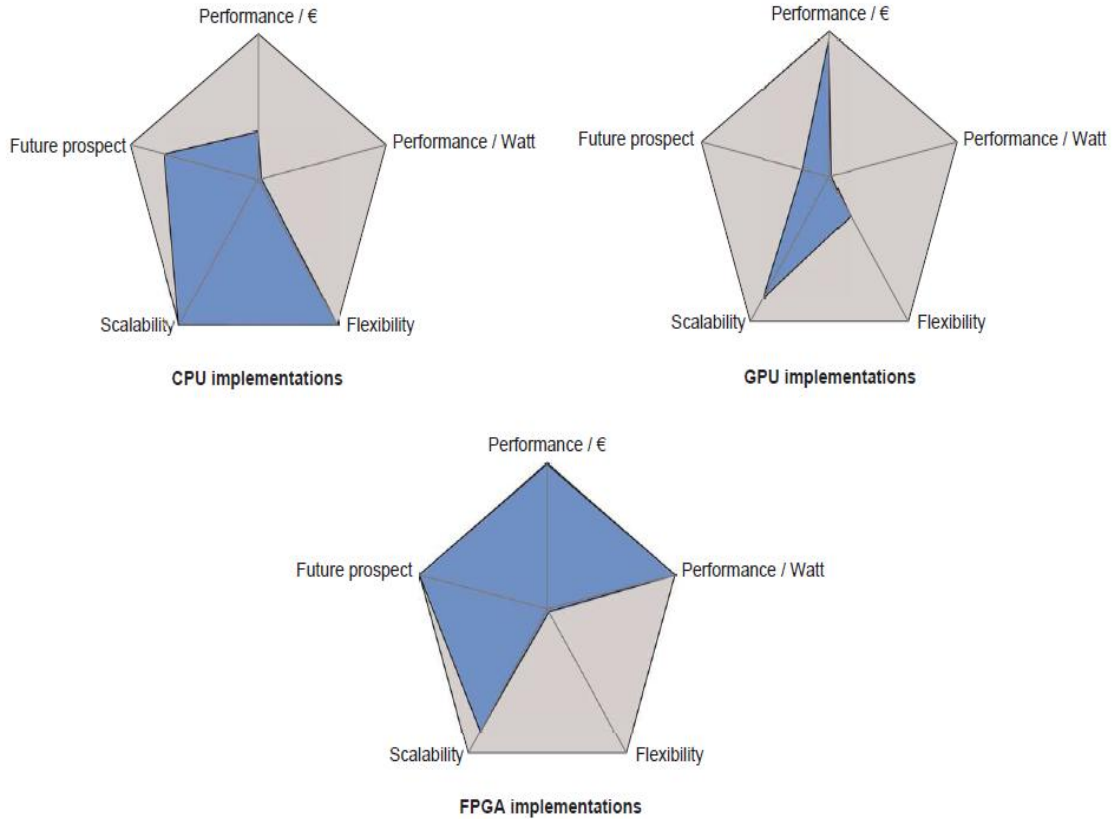
Τα FPGAs είναι ευπροσάρμοστες συσκευές επεξεργασίας δεδομένων επί των οποίων ένας αλγόριθμος απεικονίζεται απευθείας σε βασικά στοιχεία λογικής επεξεργασίας. Για να επωφεληθεί κανείς από τη χρήση ενός FPGA, πρέπει να εφαρμόσει μαζικά παράλληλους αλγορίθμους σε αυτή την αναδιαμορφωμένη συσκευή. Είναι, επομένως, κατάλληλη για ορισμένες κατηγορίες εφαρμογών βιοπληροφορικής, όπως η ευθυγράμμιση αλληλουχίας. Μέθοδοι όπως αυτές που βασίζονται στην εφαρμογή των συστολικών πινάκων χρησιμοποιούνται για να επιταχύνουν τέτοιες εφαρμογές. Στην εφαρμογή μίας τέτοιας υλοποίησης σε βάση το συστολικό πίνακα προχωρήσαμε και στο πλαίσιο αυτής της διατριβής, όπως θα αναλυθεί περαιτέρω και σε παρακάτω κεφάλαιο.

- GPUs

Αν και η δημιουργία τους, αρχικά, υποκινήθηκε από την ανάγκη για γραφικά πραγματικού χρόνου στα ηλεκτρονικά παιχνίδια, οι GPUs έχουν εξελιχθεί σε ισχυρούς και ευέλικτους επεξεργαστές διανυσμάτων, ιδανικούς για την επιτάχυνση ποικιλίας εφαρμογών παραλληλίας δεδομένων. Οι GPUs έχουν τα τελευταία δύο χρόνια έχουν εξελιχθεί από μία σταθερή μονάδα επεξεργασίας γραφικών λειτουργιών σε μια ευέλικτη πλατφόρμα που μπορεί να χρησιμοποιηθεί για πληροφορική υψηλών επιδόσεων (HPC). Εφαρμογές, όπως η ευθυγράμμιση ακολουθιών βιοπληροφορικής μπορεί να τρέξει πολύ αποτελεσματικά σε αυτές τις αρχιτεκτονικές.

3.2.3. Σύγκριση επιτάχυνσης υπολογισμών στις διάφορες πλατφόρμες υλικού

Στο κομμάτι αυτό συγκρίνεται η απόδοση των S-W υλοποιήσεων σε διαφορετικές πλατφόρμες, όπως CPUs, FPGAs και GPUs. Η σύγκριση βασίζεται σε παραμέτρους όπως το κόστος, η κατανάλωση ενέργειας, η ευελιξία (flexibility), η επεκτασιμότητα (scalability) και οι μελλοντικές προοπτικές. Ως CPU θεωρούμε έναν 4-way Opteron επεξεργαστή 43 πυρήνων. Για GPUs, ένα γρήγορο PC με 4 τελευταίας γενιάς κάρτες γραφικών, καθώς και για FPGAs αυτό της ταχύτερης γνωστής υλοποίησης του αλγορίθμου S-W, αυτό της υλοποίησης SciEngines (SciEngines, 2010). Τα αποτελέσματα της σύγκρισης αυτής παρουσιάζονται στο παρακάτω σχήμα ενώ μετά ακολουθεί μια σύντομη ανάλυση κάθε παραμέτρου σύγκρισης.



Εικόνα 3-2: Ανάλυση παραμέτρων σύγκρισης αλγορίθμου S-W σε διάφορες πλατφόρμες υλικού (Vermij,2011).

Σχέση Απόδοσης/ Euro

Τα FPGAs μπορούν να προσφέρουν το καλύτερο ποσό των GCUPS *(Giga Cell Updates Per Second - GCUPS)* ανά ευρώ, ακολουθούμενο από τα GPUs. Το χάσμα μεταξύ των

GPUs και CPUs μπορεί να εξηγηθεί από τα επιπλέον χρήματα που απαιτούνται για ένα σύστημα CPU 4 way, ενώ συνδέοντας 4 GPUs σε μια μητρική είναι δωρεάν. Το αποτέλεσμα αυτό εξηγεί γιατί τα FPGAs χρησιμοποιούνται για υπολογιστές υψηλών επιδόσεων. Αν και η επιλογή του αλγορίθμου S-W μπορεί να μην ενδείκνυται για να επιτύχει μία μεγάλη απόδοση ανά κέρδος ευρώ από τη χρήση FPGAs. Παρ 'όλα αυτά, δείχνει την τάση.

Σχέση Απόδοσης/Watt

Είναι σαφές ότι εδώ, σε αντίθεση με την προηγούμενη παράμετρο σύγκρισης, τα FPGAs είναι ο απόλυτης νικητής. Πλήρη συστήματα μπορούν να προσφέρουν χιλιάδες GCUPS για περίπου 1000 Watts. Αυτός είναι ένας άλλος σημαντικός λόγος για την προτίμηση χρήσης των FPGAs για την υλοποίηση της ευθυγράμμισης ακολουθιών. Σημειώστε ότι, ενώ δεν είναι ορατή στα γραφήματα, οι CPUs επιτυγχάνουν περίπου δύο φορές καλύτερα σκορ σε σχέση με τις GPUs.

Σχέση Ευελιξίας

Αυτή η παράμετρος σύγκρισης αντιπροσωπεύει την προσπάθεια που απαιτείται για να αλλάξει μια γρήγορη γραμμική gap εφαρμογή για να χρησιμοποιήσετε affine gap. Ένας έμπειρος μηχανικός θα καταφέρει να το κάνει αυτό σε μια μέρα για μια CPU υλοποίηση, σε μια-δυο μέρες για την GPU υλοποίηση, και πολλές εβδομάδες για την αντίστοιχη FPGA υλοποίησή τους.

Σχέση Επεκτασιμότητας

Εδώ θεωρούνται οι CPUs χρησιμοποιούνται ως η βάση. Λαμβάνοντας υπόψη ένα κατάλληλο πρόβλημα, οι CPUs είναι πολύ βαθμωτοί, δεδομένου ότι μπορούν να συνδέονται μεταξύ τους με τη χρήση τυποποιημένων λύσεων δικτύωσης. Με την ίδια λογική, οι GPUs είναι επίσης βαθμωτοί, δεδομένου ότι μπορούν μεν, να επωφεληθούν από την επεκτασιμότητα των CPUs, η οποία θα εισαγάγει δε, κάποια επιπλέον καθυστέρηση. Ως εκ τούτου επιτυγχάνει λίγο χαμηλότερα σκορ από ότι οι CPU. Ανάλογα με τη χρησιμοποιούμενη πλατφόρμα, και τα FPGAs μπορεί επίσης να γίνουν βαθμωτά.

Σχέση Μελλοντικής Προοπτικής

Τα τελευταία χρόνια, υπάρχει μια τάση για τους επεξεργαστές να αντικαταστήσουν τις GPUs σε συστήματα υψηλής ταχύτητας. Η τάση αυτή αναμένεται να συνεχιστεί, μειώνοντας έτσι το μερίδιο αγοράς των GPUs υπέρ των CPUs. Οι CPUs συνεπώς επιτυγχάνουν μεγάλα σκορ σε σχέση η μέτρηση, ενώ οι GPUs σκοράρουν μάλλον χαμηλή. Σε πολύ συγκεκριμένες περιοχές HPC, όμως, όπου οι απαιτήσεις bandwidth μνήμης είναι χαμηλές και το πρόβλημα είναι συνθέσιμο, τα FPGAs κατά πάσα πιθανότητα συνεχίζουν να είναι η καλύτερη επιλογή. Ο αλγόριθμος S-W έγκειται εν μέρει στην παρούσα κατηγορία.

3.3. Γιατί τελικά προτιμάμε τα FPGA;

Συμπερασματικά, από την ανάλυση των παραπάνω παραμέτρων σύγκρισης φτάνουμε στο συμπέρασμα να θεωρήσουμε τα FPGA ως μία πλέον κατάλληλη και αποτελεσματική πλατφόρμα υλικού ανάπτυξης και υλοποίησης τέτοιων πολύπλοκων αλγορίθμων

Στην πράξη, όντως τα FPGA χρησιμοποιούνται ευρέως για την επιτάχυνση εφαρμογών, όπως ο αλγόριθμος S-W, ο οποίος υλοποιείται στα πλαίσια της δικής μας εφαρμογής και θα αναλυθεί περαιτέρω στο Κεφάλαιο 4. Οι υλοποιήσεις βασίζονται στην ικανότητα των FPGA να επιταχύνουν το υλικό. Πιο συγκεκριμένα, βασίζονται στην τεχνική να δημιουργούν δομικά στοιχεία που ονομάζονται στοιχεία επεξεργασίας (Processing Elements – PE), που μπορούν να ενημερώσουν ένα κελί πίνακα σε κάθε κύκλο ρολογιού. Επιπλέον, πολλαπλά PE μπορούν να

συνδεθούν μεταξύ τους σε ένα δισδιάστατο ή γραμμικό συστολικό πίνακα/συστοιχία (systolic array) με σκοπό να επεξεργάζονται τεράστιο όγκο δεδομένων παράλληλα.

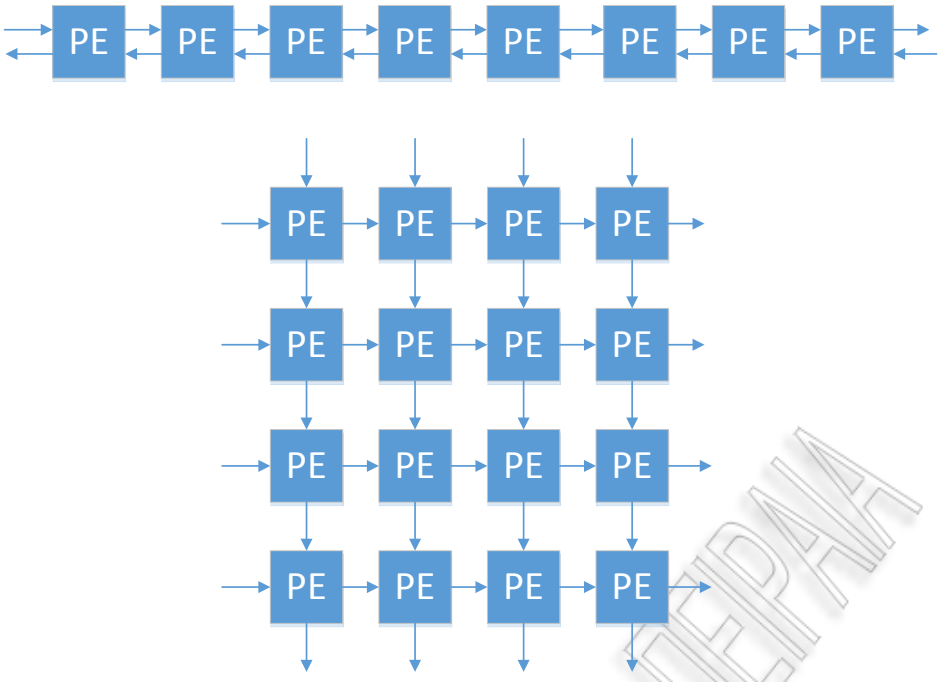
Συστολικό πίνακα καλούμε μια διαρρύθμιση επεξεργαστών σε πίνακα, όπου τα δεδομένα ρέουν συγχρονισμένα σε όλο τον πίνακα μεταξύ των γειτόνων, συνήθως με τα δεδομένα να ρέουν προς μια συγκεκριμένη κατεύθυνση (Kung & Leiserson, 1979), (Quinton & Robert, 1991).

Συνολικά, μπορούμε να πούμε ότι τα FPGA ταιριάζουν καλύτερα σε υλοποιήσεις υψηλών υπολογιστικών απαιτήσεων, όπως η βιολογική ευθυγράμμιση ακολουθιών. Το μειονέκτημα στο εγχείρημα αυτό, είναι ότι πρόκειται για έναν πολύ εξειδικευμένο τύπο επεξεργαστών, ο οποίος είναι περίπλοκος να υλοποιηθεί και να ‘οικοδομηθεί’ στο υλικό.

3.4. Σχεδίαση με χρήση συστολικού πίνακα

Η εξέλιξη των ηλεκτρονικών υπολογιστών και του Διαδικτύου έχει επιφέρει μεγάλη ζήτηση για ισχυρή και υψηλής ταχύτητας επεξεργασία δεδομένων, ωστόσο σε τέτοια σύνθετα περιβάλλοντα λίγες μέθοδοι μπορούν να παρέχουν την τέλεια λύση. Για την αποτελεσματική επεξεργασία μεγάλων ροών δεδομένων, προτείνεται η λύση της παράλληλης επεξεργασίας (parallel computing), χρησιμοποιώντας την αρχιτεκτονική του συστολικού πίνακα (systolic array) για συσκευές FPGAs (Επιτόπου Προγραμματιζόμενες Διατάξεις Πυλών).

Στην αρχιτεκτονική υπολογιστών, καλούμε συστολική αρχιτεκτονική μία διαρρύθμιση δικτύου διοχέτευσης (pipelined network) κάποιων στοιχείων επεξεργασίας (Processing Elements - PEs) που ονομάζονται ‘κύτταρα’ ή ‘κελιά’. Πρόκειται για μια εξειδικευμένη μορφή παράλληλης επεξεργασίας, όπου τα κελιά υπολογίζουν τα δεδομένα που τους έρχονται ως είσοδος και τα αποθηκεύουν ανεξάρτητα σε δικία τους μνήμη. Ουσιαστικά, μία συστολική αρχιτεκτονική είναι ένας πίνακας που αποτελείται από σειρές κύτταρων, που παρομοιάζονται με αυτές των μητρών. Εδώ, τα στοιχεία επεξεργασίας είναι παρόμοια με κεντρικές μονάδες επεξεργασίας (CPU) (εκτός από τη συνήθη έλλειψη ενός μετρητή προγράμματος, καταχωρητή εντολών, μονάδας ελέγχου κλπ., δεδομένου ότι η λειτουργία τους ενεργοποιείται με την μεταφορά σε αυτά των δεδομένων [13] ενώ κάθε κελί μοιράζεται τα δεδομένα του με τους γείτονές του, αμέσως μετά την επεξεργασία αυτών. Η συστολική συστοιχία είναι συχνά ορθογώνια, όπου τα δεδομένα ρέουν σε όλη τη συστοιχία δεδομένων μεταξύ των γειτονικών μονάδων επεξεργασίας, συχνά με διαφορετικά δεδομένα να ρέουν σε διαφορετικές κατευθύνσεις.



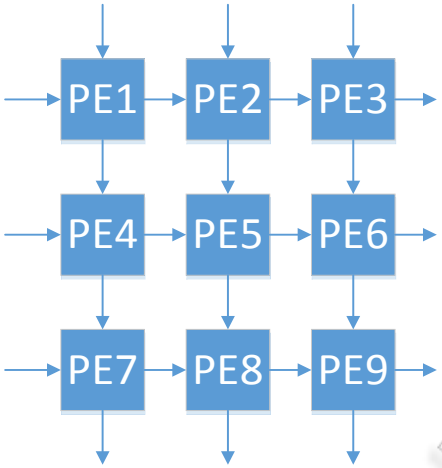
Εικόνα 3-3: Σχηματική αναπαράσταση αρχιτεκτονικών συστολικού πίνακα.

Ένα παράδειγμα ενός συστολικού αλγορίθμου αφορά τον πολλαπλασιασμό πινάκων. Ο ένας από τους 2 πίνακες τροφοδοτείται ως είσοδος σε μία σειρά, ξεκινώντας από την κορυφή του συστολικού πίνακα και καταλήγοντας στο κάτω μέρος του. Έπειτα ο άλλος πίνακας τροφοδοτείται και αυτός ως είσοδος σε μία στήλη, ξεκινώντας από το αριστερότερο PE και καταλήγοντας στο δεξιότερο. Στην συνέχεια προσθέτουμε στις εισόδους αδιάφορες τιμές έτσι ώστε κάθε PE να έχει δει μία ολόκληρη γραμμή και μία ολόκληρη στήλη. Στο τέλος, το αποτέλεσμα του πολλαπλασιασμού είναι αποθηκευμένο μέσα στο systolic array και μπορεί πλέον να αποτελέσει την έξοδο τους συστήματος η οποία μπορεί να γίνεται είτε ανά γραμμή είτε ανά στήλη.

Ας δούμε ένα παράδειγμα πολλαπλασιασμού πινάκων. Έστω ότι έχουμε τους πίνακες A και B. Ο κάθε πίνακας είναι δισδιάστατος NxN, όπου N=3. Θέλουμε να υπολογίσουμε τον πίνακα C ο οποίος είναι το γινόμενο των άλλων δύο. Με την κλασική σειριακή μέθοδο υπολογισμού έχουμε το παρακάτω πρόγραμμα:

```
for(i=1;i<=N;i++)
        for(j=1;j<=N;j++)
                for (k=1; k<=N; k++)
                        C[i,j] = C[i,j] + (A[j,k]* B[k,j])
```

Έχουμε διαδοχικά 3 βρόχους των N βημάτων ο καθένας. Ο τελικός πίνκας C συμπληρωμένος με τα αποτελέσματα των υπολογισμών μετά από χρόνο $N*N*N = N^3$, ενώ χρησιμοποιώντας ένα συστολικό πίνακα με NxN PEs οι υπολογισμοί μπορούν να ολοκληρωθούν μόνο σε χρόνο N.



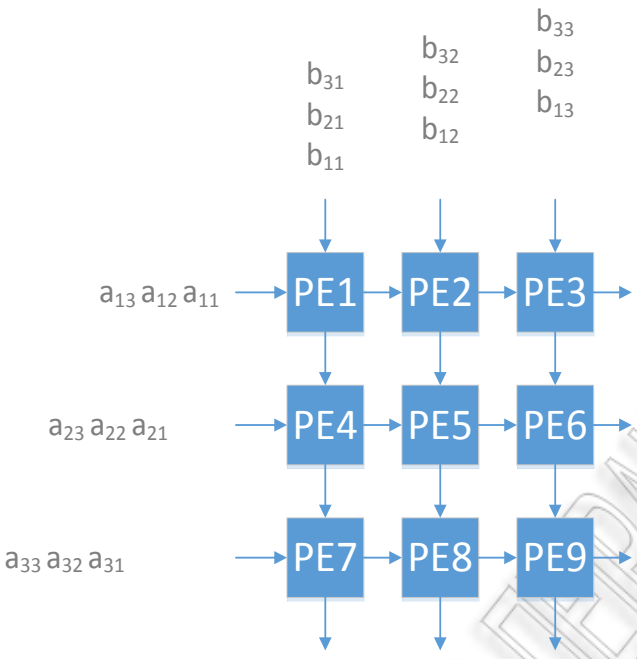
Εικόνα 3-4: Συστολικός πίνακας με 3x3 PEs για τον πολλαπλασιασμό πινάκων.

Αρχικά, θα πρέπει να κάνουμε μία αναδιάταξη των δεδομένων των πινάκων A και B προτού αποτελέσουν είσοδο για τον συστολικό πίνακα.

$$\begin{matrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{matrix} \xrightarrow{\text{Αναδιάταξη στηλών 1 \& 3}} \begin{matrix} a_{13} & a_{12} & a_{11} \\ a_{23} & a_{22} & a_{21} \\ a_{33} & a_{32} & a_{31} \end{matrix}$$

$$\begin{matrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{matrix} \xrightarrow{\text{Αναδιάταξη γραμμών 1 \& 3}} \begin{matrix} b_{31} & b_{32} & b_{33} \\ b_{21} & b_{22} & b_{23} \\ b_{11} & b_{12} & b_{13} \end{matrix}$$

Σε κάθε κύκλο ρολογιού, τα δεδομένα περνάνε στο κάθε PE από δύο διαφορετικές κατευθύνσεις, στην συνέχεια γίνεται ο πολλαπλασιασμός και τέλος το αποτέλεσμα αποθηκεύεται σε έναν καταχωρητή.

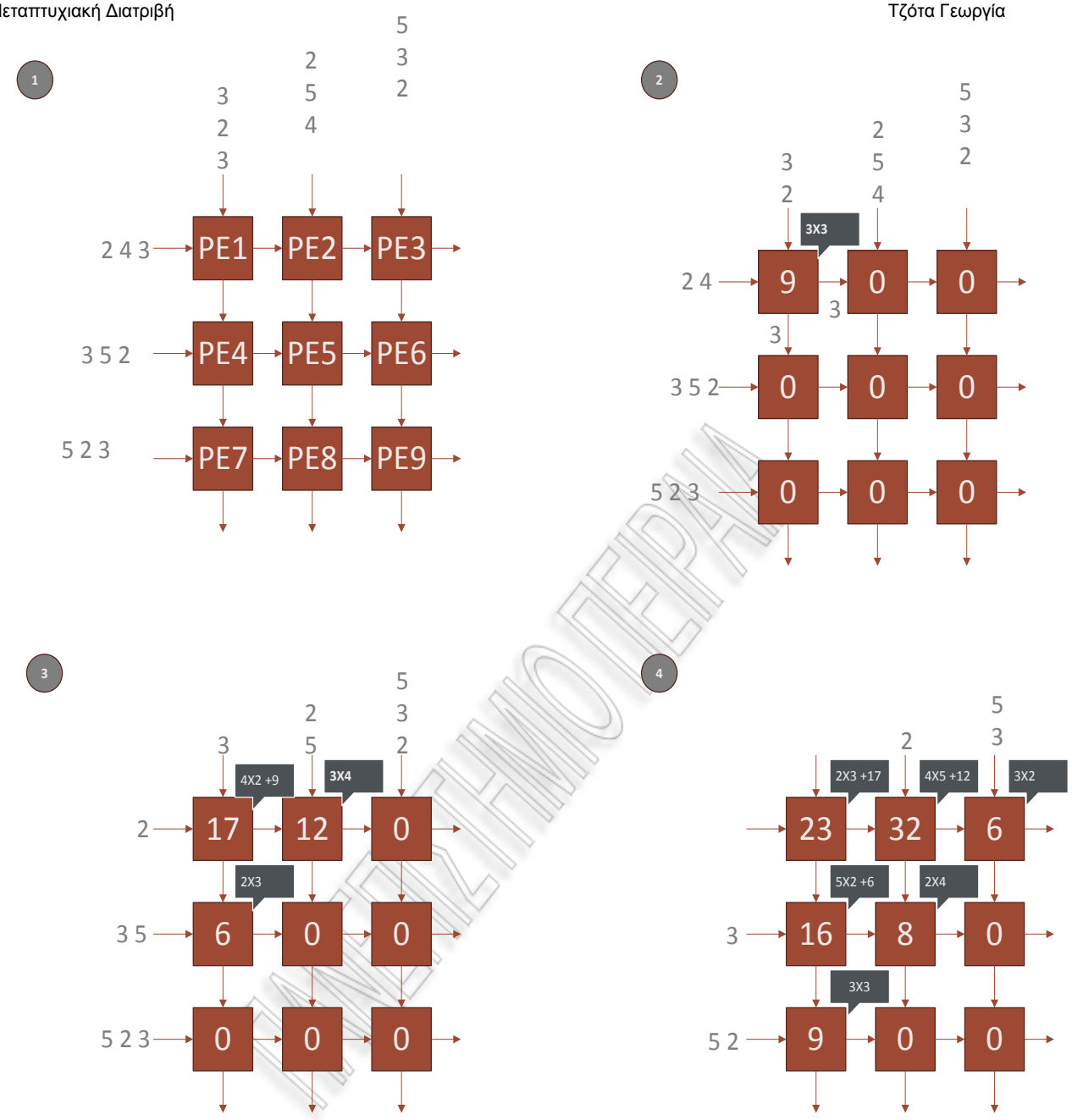


Εικόνα 3-5: Είσοδος δεδομένων στον συστολικό πίνακα.

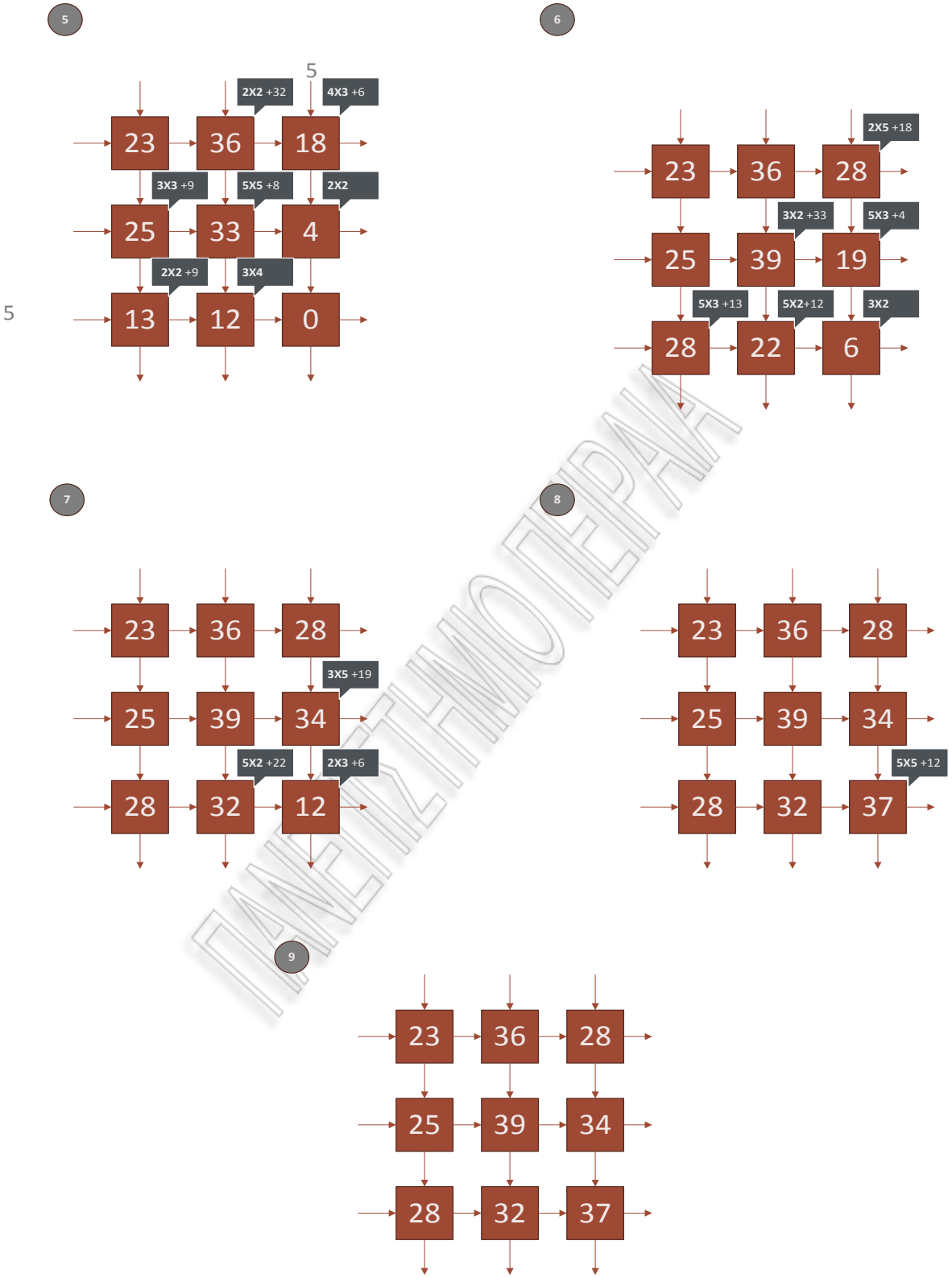
Ας δοκιμάσουμε τον παρακάτω πολλαπλασιασμό.

3 4 2		3 4 2		23 36 28
2 5 3	*	2 5 3	=	25 39 34
3 2 5		3 2 5		28 32 37

Στις εικόνες 3-6 και 3-7, παρατηρούμε τη ροή των δεδομένων μέσα στον συστολικό πίνακα σε κάθε κύκλο του ρολογιού. Κάθε PE έχει έναν δικό του καταχωρητή για να μπορεί να αποθηκεύει το αποτέλεσμα των υπολογισμών που παράγει. Μετά το πέρας τον υπολογισμών, κάθε PE θα αντιστοιχεί σε ένα από τα 9 κελιά του τελικού πίνακα C. Το αποτέλεσμα για το κάθε κελί θα βρίσκεται στον τοπικό καταχωρητή του εκάστοτε PE.



Εικόνα 3-6: Πολλαπλασισμός πινάκων στον συστολικό πίνακα (βήματα 1 έως 4)



Εικόνα 3-7: Πολλαπλασιασμός πινάκων στον συστολικό πίνακα (βήματα 5 έως 9)

4. Ο αλγόριθμος δυναμικού προγραμματισμού Smith Waterman

Στον κλάδο της βιοπληροφορικής, η ευθυγράμμιση ακολουθίας (sequence alignment) αποτελεί ένα τρόπο σύγκρισης δύο σειρών DNA, RNA ή πρωτεϊνών, έτσι ώστε να είναι δυνατή η ανίχνευση περιοχών ομοιότητας μεταξύ των σειρών. Θεωρούμε ότι κάθε τέτοια σειρά αντιστοιχίζεται σε μία ακολουθία χαρακτήρων. Συνεπώς, το πρόβλημα μέτρησης της ομοιότητας ανάμεσα σε δύο ακολουθίες χαρακτήρων μετατοπίζεται στην στοίχισή τους και στον υπολογισμό του κόστους μετατροπής από έναν χαρακτήρα σε έναν άλλο χαρακτήρα.

Για την επίλυση αυτών των προβλημάτων, προτάθηκε η χρήση μεθόδων δυναμικού προγραμματισμού. Με τον όρο δυναμικό προγραμματισμό (Dynamic Processing — DP), καλούμε έναν τρόπο επίλυσης πολύπλοκων προβλημάτων, όπου το αρχικό πρόβλημα διασπάται σε πολλά μικρότερα και απλούστερα προβλήματα. Στη συνέχεια, υπολογίζεται η λύση για κάθε ένα υποπρόβλημα, ενώ τελικά όλα τα αποτελέσματα που προκύπτουν συνδυάζονται έτσι ώστε να διαμορφωθεί το τελικό αποτέλεσμα που αντιπροσωπεύει και την ολοκληρωμένη λύση του προβλήματος.

Ένα χαρακτηριστικό παράδειγμα τέτοιου είδους αλγορίθμου είναι και ο Smith-Waterman (S-W), ο οποίος χρησιμοποιείται για την εύρεση παρόμοιων περιοχών σε αλυσίδες DNA, RNA ή πρωτεϊνών, με την χρήση της διαδικασία ευθυγράμμισης ακολουθίας.

4.1. Ο αλγόριθμος Smith - Waterman

Οι Smith και Waterman πρότειναν το 1981 τον αλγόριθμο που φέρει και το όνομά τους. Ο S-W αλγόριθμος αναζητά και βρίσκει δύο όμοιες υπό-ακολουθίες (local alignment — τοπική ευθυγράμμιση) μεταξύ δύο αλυσίδων DNA ή αλυσίδων πρωτεϊνών, υπολογίζοντας ένα κόστος ευθυγράμμισης, το οποίο μπορεί να μεταφραστεί και ως κόστος ομοιότητας (similarity cost).

Πρακτικά, όταν έχουμε να συγκρίνουμε δύο συμβολοσειρές χαρακτήρων, δημιουργείται ένας πίνακας ομοιοτήτων M x N στοιχείων, όπου M αποτελεί το πλήθος χαρακτήρων της πρώτης συμβολοσειράς και N το πλήθος χαρακτήρων της δεύτερης. Συνεπώς, για κάθε πιθανό συνδυασμό χαρακτήρων μεταξύ των δύο συμβολοσειρών έχουμε και μία αντίστοιχη θέση στον παραγόμενο πίνακα. Σε κάθε μία από αυτές τις θέσεις, τρεις περιπτώσεις μπορούν να ισχύουν. Να έχουμε ταίριασμα (match) όταν ο ίδιος χαρακτήρας α είναι παρών και στις δύο συμβολοσειρές. Να μην έχουμε ταίριασμα (mismatch ή substitution), όταν έχουμε δύο διαφορετικούς χαρακτήρες α και β. Τέλος, να έχουμε κενό (gap), όταν στην συγκεκριμένη θέση υπάρχει ένας οποιοσδήποτε χαρακτήρας στη μία συμβολοσειρά (insertion), ενώ στην άλλη δεν υπάρχει κάποιος χαρακτήρας (deletion).

Στις αλυσίδες πρωτεϊνών, το αν έχουμε ή δεν έχουμε ταιριάσματα, είναι αμφιβόλου σημασίας. Γι' αυτόν τον λόγο, αποδίδεται ένα σκορ σε κάθε ζεύγος αμινοξέων μέσα από μία συνάρτηση αντικατάστασης (substitution function) d(α, β), η οποία μετράει την ομοιότητα των στοιχείων α και β. Με παρόμοιο τρόπο, μπορούμε να αποδώσουμε μία ποινή gpenalty , για κάθε gap (κενό), το οποίο μπορούμε να το θεωρήσουμε ως μία ειδική περίπτωση της substitution function $d(\alpha,-) = g_{penalty}$ η $d(-,\beta) = g_{penalty}$.

Ας θεωρήσουμε ότι έχουμε δύο συμβολοσειρές για να συγκρίνουμε, που είναι τα $X = \{x_1, x_2, ..., x_m\}$ και $Y = \{y_1, y_2, ..., y_n,\}$. Το H(i, j) είναι το μέγιστο σκορ ομοιότητας ανάμεσα στα πιο όμοια ζεύγη των υπο-ακολουθιών που τερματίζουν στα στοιχεία x_i και y_j. Το H(i, j) μπορεί να υπολογιστεί αναδρομικά χρησιμοποιώντας την Εξίσωση που ακολουθεί:

$$\forall i: H\ i,0\ = H\ 0,j\ = 0$$

$$\forall i,j,ij\ \neq 0:$$

$$H\ i,j\ = \max \begin{cases} 0\ (\text{τοπική ευθυγράμμιση, ξεκινά πάντα με το μηδέν}) \\ H\ i-1,j-1\ + d\ x_i,y_j \quad (ταίριασμα\ ή\ αφαίρεση) \\ H\ i-1,j-1\ - g_{penalty} \quad (εισαγωγή) \\ H\ i-1,j-1\ - g_{penalty} \quad (διαγραφή) \end{cases}$$

Στο σχήμα που ακολουθεί, βλέπουμε ένα απλό παράδειγμα εφαρμογής της εξίσωσης S-W. Αν έχουμε ταίριασμα χαρακτήρων, η τιμή του σκορ ανεβαίνει κατά δύο, αν δεν έχουμε ταίριασμα μειώνεται κατά ένα και αν έχουμε κενό μειώνεται πάλι κατά ένα. Επειδή ο S-W δεν έχει αρνητικά κόστη σε κανένα κελί του πίνακα, σε αντίθεση με άλλους αλγορίθμους ευθυγράμμισης ακολουθίας, όλα τα αρχικά κόστη σε γραμμές και στήλες είναι μηδενικά. Αν κάποιος υπολογισμός δίνει αποτέλεσμα που να είναι μικρότερο του μηδενός, το αποτέλεσμα αυτομάτως μηδενίζεται.

		C	G	A	G	A	T	G	C	G	T	C	G	T
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	2	1	0	0	2	1	0	2
G	0	0	2	1	2	1	1	4	3	2	1	1	3	2
A	0	0	1	4	3	4	3	3	3	2	1	0	2	2
G	0	0	2	3	6	5	4	5	4	5	4	3	2	1
C	0	2	2	2	5	5	4	4	7	6	5	6	5	4
G	0	1	4	3	4	4	4	6	5	9	8	7	8	7
A	0	0	3	6	5	6	5	5	5	8	8	7	7	7
C	0	2	2	5	5	5	5	4	7	7	7	10	9	8
A	0	1	1	4	4	7	6	5	6	6	6	9	9	8

```
G A T G C G T C
| |   | | |   |
G A - G C G A C
```

Εικόνα 4-1: Παράδειγμα εφαρμογής S-W.

Συνεπώς, αντιμετωπίζοντας το πρόβλημα της ευθυγράμμισης ακολουθίας με τη μέθοδο δυναμικού προγραμματισμού ένα αρκετά πολύπλοκο πρόβλημα, μπορεί να αντικατασταθεί από ένα μικρό και επαναλαμβανόμενο σύνολο εντολών. ¨όπως είδαμε, μία ακολουθία X μπορεί να μετατραπεί σε μία ακολουθία Y με τρεις συγκεκριμένους τρόπους:

- Εισάγοντας το πρώτο στοιχείο της ακολουθίας Y και εκτελώντας μία βέλτιστη ευθυγράμμιση των X και των υπόλοιπων στοιχείων της ακολουθίας Y.
- Διαγράφοντας το πρώτο στοιχείο της ακολουθίας X και εκτελώντας βέλτιστη ευθυγράμμιση μεταξύ των Y και των υπολοίπων στοιχείων της X.
- Αντικαθιστώντας το πρώτο στοιχείο της ακολουθίας X με τον πρώτο στοιχείο της ακολουθίας Y και εκτελώντας βέλτιστη ευθυγράμμιση στα υπόλοιπα στοιχεία των X και Y.

Με αυτόν τον τρόπο, το αρχικό πολύπλοκο πρόβλημα της ανίχνευσης ομοιοτήτων ανάμεσα σε δύο ακολουθίες, μπορεί να αναπαρασταθεί με τρεις μόνο επαναλαμβανόμενες ενέργειες, που όπως θα δούμε και στο 5ο κεφάλαιο μπορούν πολύ εύκολα να παραλληλοποιηθούν.

4.1.1. Συναρτήσεις Gap

Όπως είδαμε, η συνάρτηση S-W που είδαμε, περιλαμβάνει μία τιμή $g_{penalty}$, η οποία είναι σταθερά. Μπορεί βέβαια να θεωρηθεί και ως μία γραμμική συνάρτηση $g(\lambda) = g_{penalty}$, όπου η παράμετρος λ αποτελεί το μήκος του εκάστοτε κενού. Παρ' όλα αυτά, μπορούμε να χρησιμοποιήσουμε πιο αντιπροσωπευτικές gap συναρτήσεις, που να ανταποκρίνονται καλύτερα στην βιολογική πραγματικότητα. Στις περισσότερες περιπτώσεις, το να επεκτείνουμε ένα gap έχει πολύ λιγότερο κόστος από το να ανοίξουμε ένα καινούργιο. Για να μπορέσουμε να εκμεταλλευτούμε αυτόν τον κανόνα στους δικούς μας υπολογισμούς δεν θα πρέπει να

χρησιμοποιούμε απλές γραμμικές συναρτήσεις για την απόδοση ποινής (αφαίρεση πόντων από το εκάστοτε υπολογιζόμενο σκορ), όπου υπάρχει κενό (gap). Μία συνάρτηση που χρησιμοποιείται συχνά για αυτόν τον σκοπό, είναι η affine συνάρτηση $g(\lambda) = g_{open} + \lambda g_{extend}$. Αυτήν την φορά έχουμε κόστος ανοίγματος και κόστος επέκτασης ενός gap, από τις αντίστοιχες μεταβλητές g_{open} και g_{extend} . Η affine συνάρτηση ποινής τιμωρεί την εισαγωγή και την διαγραφή των κενών χρησιμοποιώντας μία γραμμική συνάρτηση που αποτελείται από δύο μέρη. Το ένα μέρος δεν λαμβάνει υπόψη το μήκος του κενού ενώ το άλλο μέρος εξαρτάται από το μήκος αυτό. Μία τέτοια συνάρτηση "ενθαρρύνει" την επέκταση των κενών που υπάρχουν ήδη παρά την δημιουργία νέων.

$$\forall i,j,ij \neq 0:$$

$$H\ i,j = \max \begin{cases} 0\ (\text{τοπική ευθυγράμμιση, ξεκινά πάντα με το μηδέν}) \\ H\ i-1,j-1 + d\ x_i, y_j \quad (ταίριασμα\ ή\ αφαίρεση) \\ I\ i,j \quad (εισαγωγή) \\ D\ i,j \quad (διαγραφή) \end{cases}$$

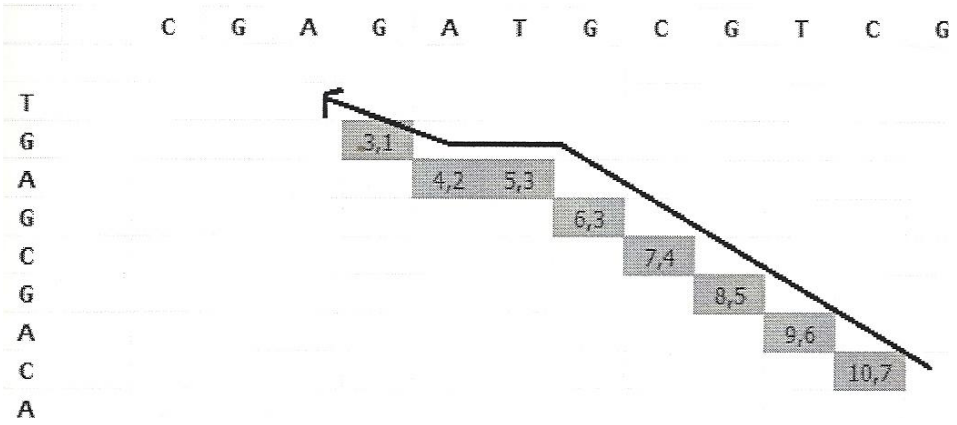
$$I\ i,j = \max \begin{cases} H\ i-1,j - g_{open} \quad (εισαγωγή) \\ I\ i-1,j - g_{extend} \quad (διαγραφή) \end{cases}$$

$$D\ i,j = \max \begin{cases} H\ i,j-1 - g_{open} \quad (εισαγωγή) \\ I\ i,j-1 - g_{extend} \quad (διαγραφή) \end{cases}$$

4.1.2. Back Tracking

Αν και το σκορ που δίνει η εξίσωση S-W είναι μία πολύ σημαντική πληροφορία για τις περισσότερες εφαρμογές (αν π.χ. αναζητούμε δεδομένα σε μία βάση δεδομένων χρησιμοποιώντας μικρά ερωτήματα με τα κατάλληλα φίλτρα), το βασικό πρόβλημα που έχουμε να αντιμετωπίσουμε, είναι η εύρεση της συγκεκριμένης ευθυγράμμισης που παρήγαγε το μέγιστο αποτέλεσμα.

Σε τέτοιες περιπτώσεις πρέπει να κρατάμε επιπλέον πληροφορία ιχνηλασίας. Με άλλα λόγια, θα πρέπει να έχουμε έναν πίνακα M x N, (όσο και ο αρχικός πίνακας ομοιότητας) που κάθε κελί θα έχει αποθηκευμένη ως τιμή τον πρόγονο από τον οποίο προήλθε ο υπολογισμός. Π.χ. στην Εικόνα 4-1 έχουμε ένα συμπληρωμένο S-W πίνακα ομοιότητας (similarity matrix) με μέγιστο σκορ = 10. Για να είμαστε σε θέση να κάνουμε back tracking και να δούμε όλη την διαδρομή που κάναμε για να καταλήξουμε στο 10, θα πρέπει να έχουμε αποθηκευμένα τα ίχνη προέλευσης του σκορ σε έναν άλλον πίνακα. Βλέπουμε ότι το προηγούμενο στοιχείο πριν το 10 βρίσκεται στην θέση (i-1,j-1). Οι πρόγονοι μπορούν να είναι μόνο τρεις για κάθε κελί. Μπορεί να είναι πάνω, αριστερά ή διαγώνια-πάνω-αριστερά. Δηλαδή, μπορεί να είναι μία από τις θέσεις (i,j-1), (i-1, j) και (i-1,j-1), όταν η τρέχουσα θέση είναι η (i, j).



Εικόνα 4-2: Πίνακας Backtracking

4.2. Υλοποίηση του S-W με χρήση συστολικού πίνακα

Κατα τη διάρκεια του υπολογισμού του πίνακα ομοιότητας του S-W αλγορίθμου, η τιμή του κάθε στοιχείου πίνακα V(i,j) εξαρτάται πάντα από την τιμή τριών άλλων στοιχείων:

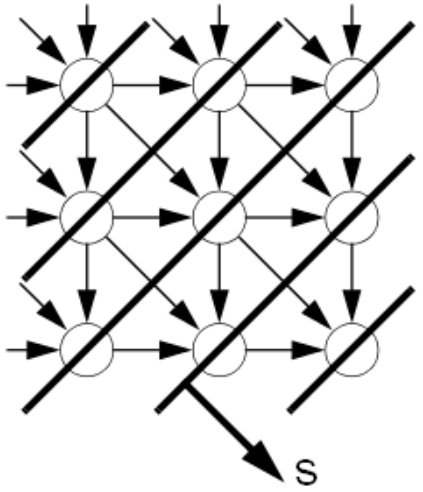
- Τον επάνω αριστερά γείτονα H(i-1, j-1)
- Τον αριστερό γείτονα H(i, j-1)
- Τον επάνω γείτονα H(i-1, j)

Ως εκ τούτου, η αλληλουχία υπολογισμών του πίνακα ομοιοτήτων, όπως φαίνεται στην ακόλουθη εικόνα, ξεκινά από το στοιχείο της αριστερής κορυφής προς το στοιχείο της κάτω δεξία γωνίας σύμφωνα με την κατεύθυνση που δείχνεται από το βέλος [14]. Ένας τέτοιος πίνακας με τιμές μπορεί να υλοποιηθεί ως S-W συστολικός πίνακας που περιέχει PEs.

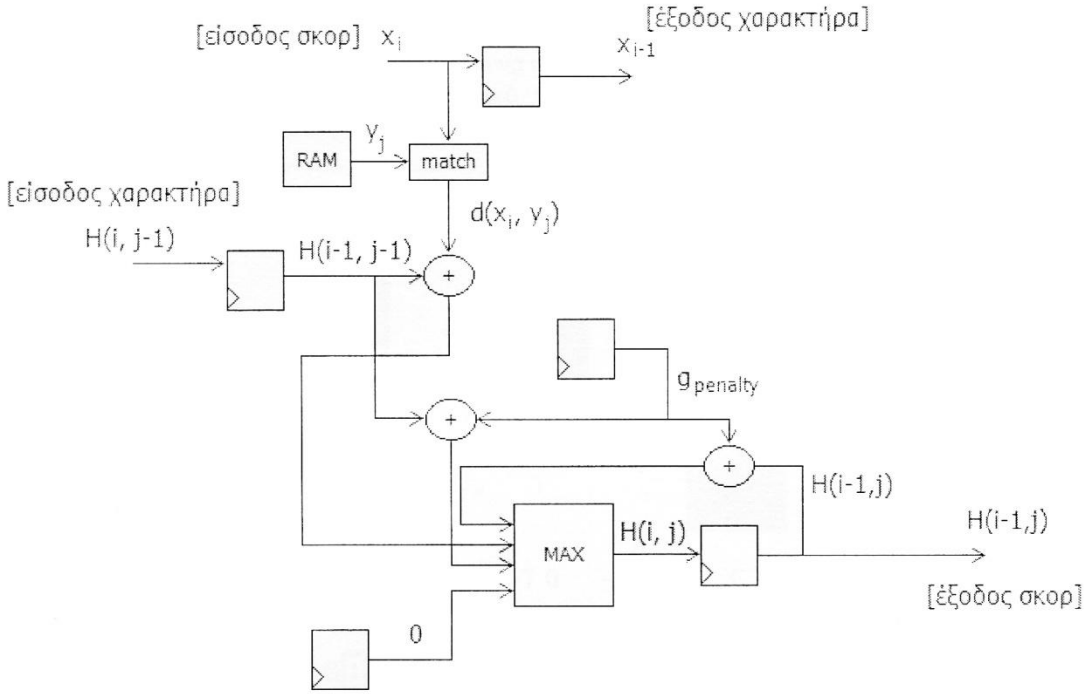
	-	S1	S2	S3	S4	S5	…
-	0	0	0	0	0	0	
T1	0	①	②	③	④	⑤	⑥
T2	0	②	③	④	⑤	⑥	⑦
T3	0	③	④	⑤	⑥	⑦	⑧
T4	0	④	⑤	⑥	⑦	⑧	⑨
T5	0	⑤	⑥	⑦	⑧	⑨	⑩
…		⑥	⑦	⑧	⑨	⑩	

Εικόνα 4-3: Τοπικότητα υπολογισμών ενός S-W PE [14].

Μέσα από την παρατήρηση της διαδιακασίας υπολογισμού του πίνακα ομοιοτήτων, βλέπουμε ότι για κάθε κύκλο ρολογιού, κάθε στοιχείο που βρίσκεται σε μία αντι-διαγώνια γραμμή και σημειωνέται με τον ίδιο αριθμό, μπορεί να ταυτόχρονα. Για παράδειγμα, στον πρώτο κύκλο, μόνο ένα στοιχείο που σημειώνεται με τον αριθμό (1) θα μπορούσε να υπολογιστεί, στο δεύτερο κύκλο δύο στοιχεία που χαρακτηρίζονται ως (2) μπορεί να υπολογιστούν, στον τρίτο κύκλο τρία στοιχεία που χαρακτηρίζονται ως (3) κ.ο.κ. , γεγονός που χαρακτηρίζει τον αλγόριθμο Smith-Waterman να έχει πολλές δυνατότητες παράληλης επεξεργασίας δεδομένων.



Εικόνα 4-4: Παραλληλή επεξεργασία PEs που βρίσκονται στη διαγώνιο του συστολικού πίνακα.



Εικόνα 4-5: Μορφή S-W PE [2].

Επιπρόσθετα, εξαιτίας του γεγονότος ότι ο S-W αλγόριθμος δεν περιλαμβάνει αρνητικά κόστη στους υπολογισμούς, η συνδυαστική λογική του S-W PE που επιστρέφει το max value, δέχεται μία επιπλέον είσοδο που παραμένει σταθερά στο λογικό μηδέν. Οι πράξεις που γίνονται για τον υπολογισμό των σκορ, χρειάζονται μόνο την τιμή $d(x_i,y_j)$ και όχι τους χαρακτήρες x_i και y_j (Εικόνα 4-5). Γι' αυτόν τον λόγο, η φάση ταιριάσματος (υπολογισμός του $d(x_i,y_j)$ και η φάση υπολογισμού των σκορ (αναδρομικός υπολογισμός του H(i,j)) είναι υλοποιημένες ξεχωριστά.

Κάθε PE υπολογίζει και αποθηκεύει τοπικά το επόμενο σκορ πάντα με βάση το προηγούμενο που έχει λάβει (από το προηγούμενο PE του συστολικού πίνακα), ενημερώνοντας την αντίστοιχη θέση της τοπικής του μνήμης. Όλες οι τοπικές θέσεις μνήμης χρησιμοποιούνται για την αποθήκευση της εκάστοτε στήλης του τελικού πίνακα ομοιοτήτων, αλλά επίσης χρησιμοποιούνται και ως τοπικές μεταβλητές οι οποίες ανακτώνται και ενδέχεται να αλλάξουν αρκετές φορές μέχρι την ολοκλήρωση των υπολογισμών. Επίσης, κάθε PE κρατάει τοπικά το μέγιστο σκορ το οποίο έχει υπολογίσει.

5. Περιγραφή συστήματος – εφαρμογής

5.1. Εισαγωγή

Στα πλαίσια της παρούσας μεταπτυχιακής διατριβής, χρησιμοποιήθηκε ο αλγόριθμος S-W για την ανάπτυξη μιας εφαρμογής η οποία θα αναγνωρίζει ηχητικά εφέ μέσα σε ψηφιακές ροές ήχου (ταινίες). Η εφαρμογή χρησιμοποιεί ένα προκαθορισμένο σύνολο ηχητικών εφέ, τα οποία, όπως και η ταινία, προσπελαύνονται ως ξεχωριστά αρχεία. Για κάθε ένα αρχείο ηχητικού εφέ, η εφαρμογή εκτελώντας τοπικές ευθυγραμμίσεις για όλο το μήκος της ταινίας παράγει τα αντίστοιχα αποτελέσματα για κάθε ζεύγος εφέ-ταινία.

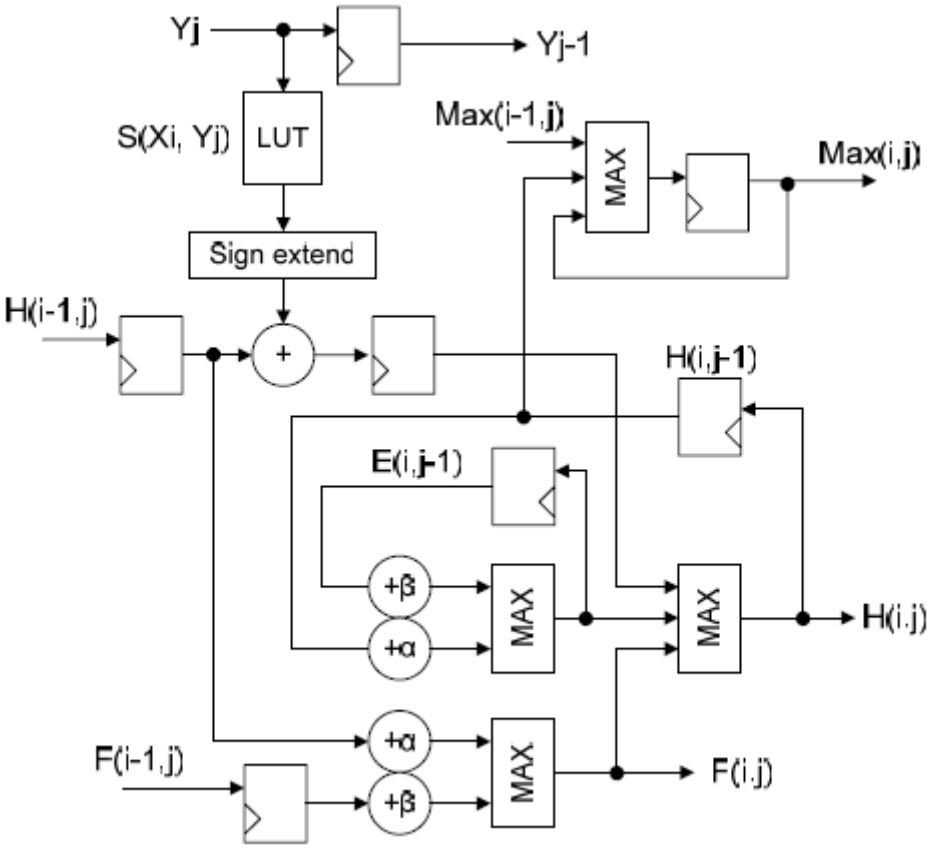
Ωστόσο, λόγω της ιδιαίτερης πολυπλοκότητας του αλγορίθμου καθώς και του γεγονότος ότι η εφαρμογή καλείται να υποστηρίξει ένα πολύ μεγάλο εύρος χαρακτηριστικών αποφασίστηκε να χρησιμοποιηθεί ένας S-W επιταχυντής που βασίζεται στην αρχιτεκτονική του συστολικού πίνακα. Ο επιταχυντής αυτός θα συμπεριληφθεί στο δεδομένο ενσωματωμένο σύστημα που εκτελεί την εφαρμογή ως ένα επιπλέον περιφερειακό που θα επικοινωνεί μέσω διαύλου με τον επεξεργαστή που τρέχει την εφαρμογή. Τέλος, αξίζει να τονιστεί ότι η σχεδίαση του συστολικού πίνακα έχει προσαρμοστεί με τέτοιον τρόπο ώστε να επιταχύνει τον αλγόριθμό S-W για τις ανάγκες της συγκεκριμένης εφαρμογής, καθώς επίσης έχουν γίνει και οι απαραίτητες τροποποιήσεις ώστε να μπορέσει να συνδεθεί ο επιταχυντής- περιφερειακό αποτελεσματικά μέσω διαύλου επικοινωνίας με τον ενσωματωμένο επεξεργαστή και την εφαρμογή.

Συνεπώς, το ολοκληρωμένο ενσωματωμένο σύστημα αποτελείται κυρίως από τον ενσωματωμένο επεξεργαστή που τρέχει την εφαρμογή του S-W (κώδικας σε γλώσσα προγραμματισμού C) και το περιφερειακό το οποίο υλοποιεί τον S-W με συστολικό πίνακα για την επιτάχυνση των τοπικών ευθυγραμμίσεων. Εκτός αυτών, έχουν προστεθεί και άλλα περιφερειακά, στοιχεία μνήμης (memory resources/cores) κ στοιχεία απασφαλμάτωσης (debug modules), απαραίτητα για την εκτέλεση της εφαρμογής, τα οποία θα αναφερθούν στη συνέχεια. Ο ενσωματωμένος επεξεργαστής και ο συστολικός πίνακας S-W επικοινωνούν μεταξύ τους μέσω του διαύλου επικοινωνίας του ενσωματωμένου επεξεργαστή, γλιτώνοντας έτσι τον επιπλέον χρόνο και πολυπλοκότητα που θα επέφερε συγκριτικά μία λύση εκτέλεσης ολόκληρης ή τμήματος της εφαρμογής από έναν κανονικό Η/Υ σε συνεργασία με ένα FPGA.

5.2. Περιγραφή αρχιτεκτονικής του S-W επιταχυντή στο FPGA

5.2.1. Περιγραφή της δομής S-W PE

Η ανάπτυξη του S-W επιταχυντή στο υλικό περιλαμβάνει την μετατροπή κομματιού της λογικής του αλγορίθμου σε ένα σύνολο από μονάδες επεξεργασίας, οι οποίες αποτελούν τον συστολικό πίνακα. Η αρχιτεκτονική που χρησιμοποιήθηκε για την ανάπτυξη του συστολικού πίνακα της εφαρμογής μας βασίζεται στην παρακάτω τυπική δομή μιας μονάδας επεξεργασίας PE.



$$H\ i,j\ = \max \begin{matrix} 0 \\ E(i,j) \\ F(i,j) \\ H\ i-1,j-1\ + S(X_i, Y_j) \end{matrix} \qquad \forall\ 1 \le i \le M\ , 1 \le j, N$$

$$E\ i,j\ = \max \begin{matrix} H\ i,j-1\ - a \\ E\ i,j-1\ - \beta \end{matrix}, \qquad \forall 0 \le i \le M, 1 \le j \le N$$

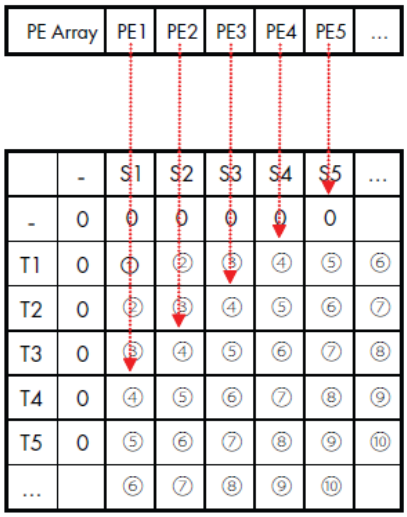
$$F\ i,j\ = \max \begin{matrix} H\ i,j-1\ - a \\ E\ i,j-1\ - \beta \end{matrix}, \qquad \forall 0 \le i \le M, 1 \le j \le N$$

Εικόνα 5-1: Το PE της S-W εφαρμογής.

Εδώ, το PE περιλαμβάνει μία τοπική μνήμη στην οποία αποθηκεύει τις τιμές H(i, j-1), H(i-1, j) και H(i-1, j-1) μαζί με ένα LUT (Lookup Table) για την αποθήκευση μίας στήλης για τα κόστη μετατροπής από τον ένα χαρακτήρα στον άλλο. Το κάθε PE αντιστοιχίζεται με ένα χαρακτήρα της μίας εκ των δύο ακολουθιών προς ευθυγράμμιση.

Αν θεωρήσουμε ένα PE ενός συστολικού πίνακα και πούμε ότι αντιστοιχίζουμε έναν χαρακτήρα A της πρώτης ακολουθίας χαρακτήρων στο PE, ενώ υποθέτουμε παράλληλα ότι η δεύτερη ακολουθία προς σύγκριση είναι η CAGGCT, τότε το LUT του PE θα περιλαμβάνει 6

εγγραφές κάθε μία από τις οποίες θα είναι και το αντίστοιχο προστιθέμενο σκορ $S(X_i, Y_j)$ του Similarity matrix, όπου το X_i = A και Y_j = {C,A,G,G,C,T} για κάθε j = {0,1,2,3,4,5}. Αν σε κάθε ταίριασμα προσθέτουμε τον αριθμό 4 και όταν δεν έχουμε ταίριασμα αφαιρούμε τον αριθμό 2 από το συνολικό σκορ, τότε τα τελικά δεδομένα του LUT, πριν την έναρξη της εκτέλεσης του S-W θα είναι (-2,4,-2,-2,-2,-2) [2]. Με αντίστοιχο τρόπο, θα γεμίσουν και τα LUTs των άλλων PEs ανάλογα με ποιον χαρακτήρα έχουν συσχετιστεί. Με αυτόν τον τρόπο, αν δούμε το PE μόνο ως στοιχείο μνήμης, το κάθε ένα αποτελεί μία στήλη για τον αρχικό πίνακα ομοιοτήτων (similarity matrix).



Εικόνα 5-2: Αντιστοίχιση του αλγορίθμου Smith Waterman σε ένα συστολικό πίνακα PE.

Αφού γεμίσουν τα LUTs των PEs με τα κόστη μετατροπής , ξεκινά η εκτέλεση του S-W. Αρχικά, αναζητείται στην LUT το κόστος μετάβασης από το αντίστοιχο προηγούμενο πάνω-αριστερά στοιχείο της διαγωνίου. Έπειτα, υπολογίζονται τα κόστη μετάβασης από τα δύο άλλα στοιχεία που βρίσκονται είτε πάνω, είτε αριστερά. Μετέπειτα, υπολογίζονται οι τιμές H(i-1, j-1) + S(X,, Y,), E(i, j) και F(i, j), βρίσκεται το μέγιστο μεταξύ των τριών διαφορετικών αυτών τιμών και του μηδενός (0). Το μέγιστο αποτέλεσμα αποτελεί το νέο σκορ H(i,j), το οποίο στην συνέχεια αποστέλλεται στο επόμενο PE. Όταν η εκτέλεση του S-W φτάσει στο τέλος της, κάθε PE θα έχει από μία μέγιστη τιμή αποθηκευμένη. Όλες οι μέγιστες τιμές θα συγκριθούν, έτσι ώστε να έχουμε μία και μόνο μέγιστη τιμή που θα αποτελεί και το μέγιστο σκορ της τοπικής ευθυγράμμισης S-W.

5.2.2. Περιγραφή της αρχιτεκτονικής του συστολικού πίνακα

Η αρχιτεκτονική του S-W επιταχυντή αποτελείται από ένα συστολικό πίνακα ο οποίος συντίθεται από μονάδες επεξεργασίας (PEs). Ο αριθμός των μονάδων επεξεργασίας καθορίστηκε να είναι 50, σύμφωνα με το πόσες καταφέραμε να χωρέσουμε στην αναπτυξιακή πλακέτα FPGA που χρησιμοποιούμε για την εφαρμογή. Ωστόσο, για την εφαρμογή θεωρούμε ότι ο αριθμός των PEs αντιστοιχεί σε μία μεταβλητή M (παράμετρος που καθορίζεται πριν την εκτέλεση της εφαρμογής), ενώ επίσης με την μεταβλητή N καθορίζουμε το μήκος της στήλης/ μέγεθος μνήμης του PE (μεταβλητή που μπορεί να φτάσει μέχρι την τιμή 127).

Συνεπώς, αν υποθέσουμε ότι έχουμε να αναλύσουμε μία ταινία X, αυτή τεμαχίζεται σε τμήματα μεγέθους M καρέ. Σε κάθε επανάληψη, τα M καρέ εισέρχονται στο systolic array και αντιστοιχίζονται σε κάθε ένα από τα M PEs. Αν υποθέσουμε, ότι κάθε PE έχει μία τοπική μνήμη N στοιχείων, το μέγεθος του εφέ που επιθυμούμε να αναγνωρίσουμε στην ταινία δεν θα πρέπει να ξεπερνά τα N καρέ. Έτσι, το εφέ αποθηκεύεται ολόκληρο στην τοπική μνήμη του κάθε PE. Στην πραγματικότητα, κάθε PE αποθηκεύει μία ολόκληρη γραμμή του πίνακα S (Simularity matrix) που παράγεται από το κομμάτι της εφαρμογής που τρέχει στον ενσωματωμένο

επεξεργαστή. Οι τιμές που αποθηκεύονται στα κελία του συστολικού πίνακα πριν την έναρξη της ευθυγράμμισης ακολουθιών, είναι τα κόστη αντικατάστασης (substitution costs) $S(X_k, Y_j)$ με $1 \leq j \leq N$. Το κόστος αντικατάστασης είναι μεταξύ του καρέ της ταινίας X_k και των καρέ Y_i έως Y_N που ανήκουν στο εφέ. Ο νέος πίνακας ομοιοτήτων φορτώνεται επαναληπτικά στο συστολικό πίνακα για κάθε νέο ζεύγος τιμών Χ και Υ.

Το PE υπολογίζει την τιμή H(i,j) όπως και την μέγιστη τιμή Max(i,j). Το μήκος των S-W σημάτων, E,F,H και Max(dw) καθορίζει το εύρος και την ακρίβεια των σκορ ομοιότητας (similarity scores) και άρα την ίδια την ακρίβεια της μεθόδου ανίχνευσης. Τα σκορ ομοιότητας S(i,j) που υπολογίζονται στο πρώτο βήμα (από το λογισμικό) έχουν εύρος τιμών [-1, 1], όπου ο άσσος σημαίνει ότι υπάρχει πλήρες ταίριασμα. Έτσι, τα σκορ ομοιότητας που προκύπτουν στρογγυλοποιούνται στα δύο δεκαδικά ψηφία και έπειτα πολλαπλασιάζονται με το 100. Με αυτόν τον τρόπο δεν έχουμε δεκαδικούς αριθμούς και οι τιμές που κατεβαίνουν στον S-W systolic array για ευθυγράμμιση ανήκουν στο εύρος [-100, 100]. Το εύρος των σκορ ομοιότητας H(i,j) εξαρτάται από το εύρος των αρχικών σκορ S(i,j) και φυσικά το μέγεθος του ηχητικού εφέ. Αν π.χ. έχουμε 50 PE στοιχεία (άρα χωρίζουμε την ταινία σε κομμάτια των 50 καρέ) και ένα εφέ προς αναζήτηση που αποτελείται από 30 καρέ, τότε το μέγιστο δυνατό σκορ που μπορούμε να επιτύχουμε είναι 30 * 50 = 1500 και επειδή ο S-W έχει ως αλγόριθμος τον αριθμό 0 ως το μικρότερο δυνατό κόστος, τότε το εύρος τιμών που μπορούν να έχουν τα σκορ είναι [0, 1500].

Οι δύο επιπλέον μνήμες RAM που έχουν τοποθετηθεί στο κάθε PE χρησιμεύουν για την αποθήκευση του αρχικού πίνακα ομοιοτήτων (S-RAM) που έλαβε το systolic array από τον ενσωματωμένο επεξεργαστή και την αποθήκευση του ενημερωμένου πίνακα ομοιοτήτων με τα νέα κόστη (H-RAM). Οι S μνήμες ενημερώνονται μέσα από μία αλυσίδα διαμόρφωσης, η οποία ολισθαίνει τις τιμές του αρχικού πίνακα ομοιοτήτων κατά μήκος του συστολικού πίνακα. Κατά την εκτέλεση του S-W αλγορίθμου, το κάθε PE διαβάζει τα δεδομένα που είναι αποθηκευμένα στην S-RAM και αφού γίνουν οι κατάλληλοι υπολογισμοί, αποθηκεύονται τα νέα κόστη στην H-RAM μνήμη. Στο τέλος της S-W εκτέλεσης τα αποτελέσματα των H μνημών ολισθαίνουν προς τα έξω μέσω της ίδιας αλυσίδας διαμόρφωσης.

Τα επιπλέον σήματα εισόδου reset, freeze και stop προσθέτουν επιπλέον λειτουργικότητα στο συστολικό πίνακα. Το reset σήμα καθαρίζει όλες τις τοπικές αποθηκευμένες τιμές E,F,H ενώ το σήμα freeze σταματάει προσωρινά την λειτουργία της μονάδας PE. Όταν το σήμα stop είναι ενεργό τότε το PE απενεργοποιείται σταματώντας την εκτέλεση της διαδικασίας ταιριάσματος και εφαρμόζοντας μόνο τις gap συναρτήσεις. Μία επιπλέον λειτουργία που κάνει το PE είναι να αποθηκεύει τον πρόγονο κάθε υπολογισμού. Ο πρόγονος αποθηκεύεται σε μορφή κατεύθυνσης. Τρεις είναι οι πιθανές κατευθύνσεις για τον πρόγονο: πάνω, αριστερά και διαγώνια-πάνω-αριστερά. Έτσι μπορούμε να χρησιμοποιήσουμε μία κωδικοποίηση των 2 bit για την αναπαράσταση των τριών πιθανών κατευθύνσεων — προγόνων.

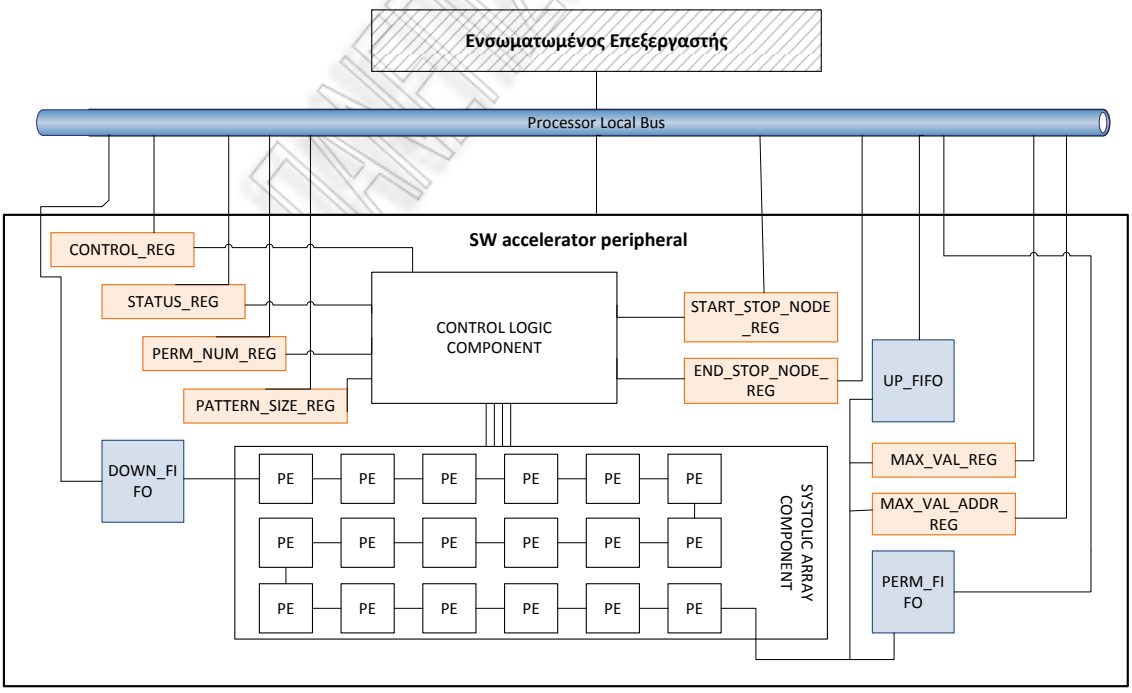
Εκτός όμως από το PE έχουμε και ένα άλλο κομμάτι λογικής που αποτελεί τον έλεγχο της εκτέλεσης του S-W μέσα στο FPGA. Η μονάδα ελέγχου (Control Logic Unit) συντονίζει την λειτουργία όλων των PE. Τα σήματα ελέγχου και εγγραφής των μνημών παράγονται από αυτό το συστατικό στοιχείο. Ο συστολικός πίνακας χαρακτηρίζεται από έξι καταστάσεις λειτουργίας (modes), κάθε μία από τις οποίες ενημερώνεται από την λογική ελέγχου:

- Process: Ξεκινά η εκτέλεση του S-W.
- Reset: Ενεργοποιεί το reset σήμα σε όλα τα PEs για το καθάρισμα των τοπικών μνημών των τιμών E, F, H.
- Wr_ram: Πραγματοποιείται η εγγραφή των τοπικών μνημών S με το αρχικό Similarity matrix που υπολογίστηκε από την εφαρμογή που τρέχει στον ενσωματωμένο επεξεργαστή
- Rd_ram: Διαβάζονται οι μνήμες H μέσα από το systolic array για προώθηση του αποτελέσματος προς τον ενσωματωμένο επεξεργαστή.
- Stop1 & stop2: Ορίζει τους ανενεργούς κόμβους σε περίπτωση που θέλουμε να κάνουμε ανίχνευση με πολλαπλά περάσματα.

Στην τελική αρχιτεκτονική του S-W επιταχυντή που απεικονίζεται στο παρακάτω σχήμα, έχουν, όπως αναφέραμε και προηγουμένως, προστεθεί τρεις μνήμες τύπου FIFO καθώς και

οκτώ καταχωρητές περιφερειακά του systolic array. Τόσο οι καταχωρητές όσο και οι μνήμες δημιουργήθηκαν με τέτοιο τρόπο ώστε να είναι άμεσα προσπελάσιμες από τον ενσωματωμένο επεξεργαστή μέσω του διαύλου επικοινωνίας (Παράρτημα Α).

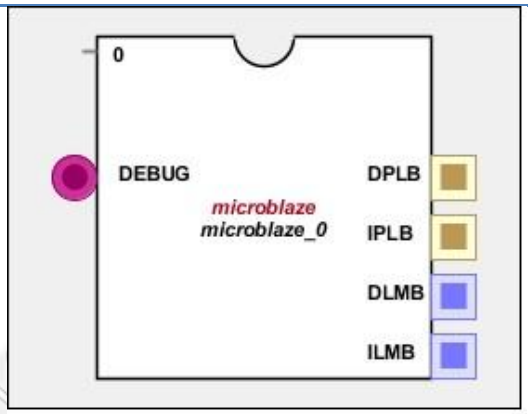
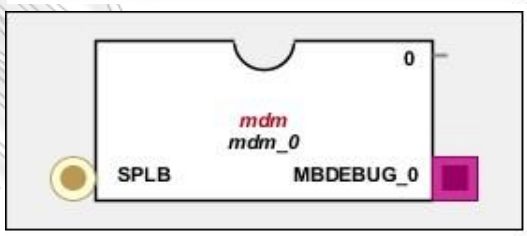
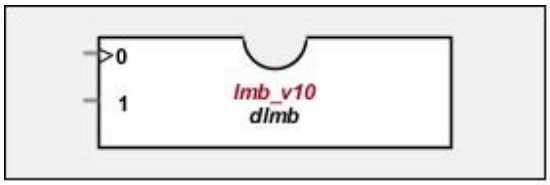
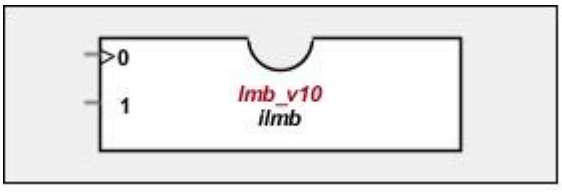
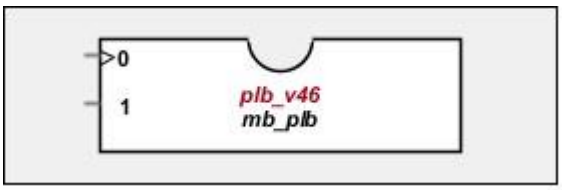
Οι καταχωρητές που χρησιμοποιούνται είναι απαραίτητο να γράφονται και να διαβάζονται συνήθως και από τα δύο μέρη του ενσωματωμένου συστήματος διασφαλίζοντας έτσι την ορθή λειτουργία του αλγορίθμου. Συγκεκριμένα, ο καταχωρητής CONTROL_REG χρησιμοποιείται από τον ενσωματωμένο επεξεργαστή για να καθορίσει πότε θα εκκινήσει η S-W εκτέλεση. Τίθεται στην τιμή 1 όταν ο Similarity matrix έχει ενημερώσει τη μνήμη DOWN_FIFO του S-W systolic array. Ο καταχωρητής STATUS_REG διαβάζεται συνεχώς από τον επεξεργαστή (polling) αμέσως μετά την ενεργοποίηση της S-W διαδικασίας (όταν θέσουμε τον CONTROL_REG στην τιμή 1). Ο S-W επιταχυντής θέτει τον STATUS_REG στην τιμή 1 μετά το πέρας της πρώτης S-W εκτέλεσης (δεν έχουν ολοκληρωθεί τα Permutations) και στην τιμή 3 όταν, αντίστοιχα, έχουν τελειώσει όλες οι εκτελέσεις του συστολικού πίνακα (αρχική και Permutations). Τις αλλαγές αυτές καταλαβαίνει ο ενσωματωμένος επεξεργαστής και σταματάει το συνεχές διάβασμα. Συγκεκριμένα, όταν ο STATUS_REG πάρει την τιμή 1, ο ενσωματωμένος επεξεργαστής καταλαβαίνει ότι μπορεί να διαβάσει τα δεδομένα της μνήμης UP_FIFO (η οποία ενημερώνεται με τα αποτελέσματα των υπολογισμών του συστολικού πίνακα). Επίσης, ο επεξεργαστής τώρα μπορεί να διαβάσει το περιεχόμενο των καταχωρητών MAX_VAL_REG και MAX_VAL_ADDR_REG. Ο καταχωρητής MAX_VAL_REG περιέχει το μεγαλύτερο σκορ που σημειώθηκε κατά τη διάρκεια της ευθυγράμμισης, ενώ ο καταχωρητής MAX_VAL_ADDR_REG περιέχει τη διεύθυνση στην οποία σημειώθηκε το μέγιστο αυτό σκορ. Ενώ τα δεδομένα αυτά διαβάζονται, παράλληλα εκτελούνται τα Permutations. Ο επεξεργαστής μετά το διάβασμα των δεδομένων της πρώτης εκτέλεσης, ξαναδιαβάζει τον καταχωρητή STATUS_REG μέχρι ο τελευταίος να έχει την τιμή 3. Όταν πάρει την τιμή 3 τότε ο επεξεργαστής θα μπορεί να διαβάσει την μνήμη PERM_FIFO η οποία θα έχει ήδη γεμίσει από το συστολικό πίνακα με τα μέγιστα σκορ. Επιπλέον, ο ενσωματωμένος επεξεργαστής ενημερώνει τον καταχωρητή PERM_NUM_REG με το πλήθος των Permutations, τον καταχωρητή PATTERN_SIZE_REG με το μέγεθος του ηχητικού εφέ προς ανίχνευση και τους καταχωρητές START_STOP_NODE_REG και END_STOP_NODE_REG, οι οποίοι κρατούν αντίστοιχα την αρχή και το τέλος των απενεργοποιημένων PEs και χρησιμοποιούνται σε περιπτώσεις όπου θέλουμε να εκτελέσουμε πολλαπλά περάσματα.

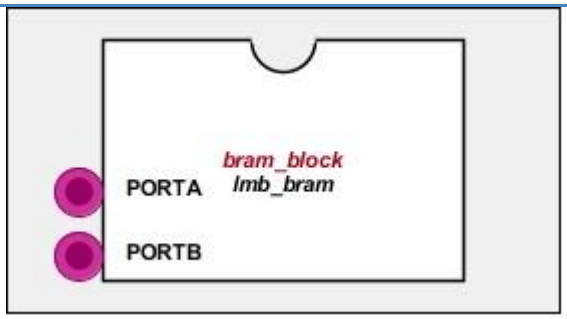
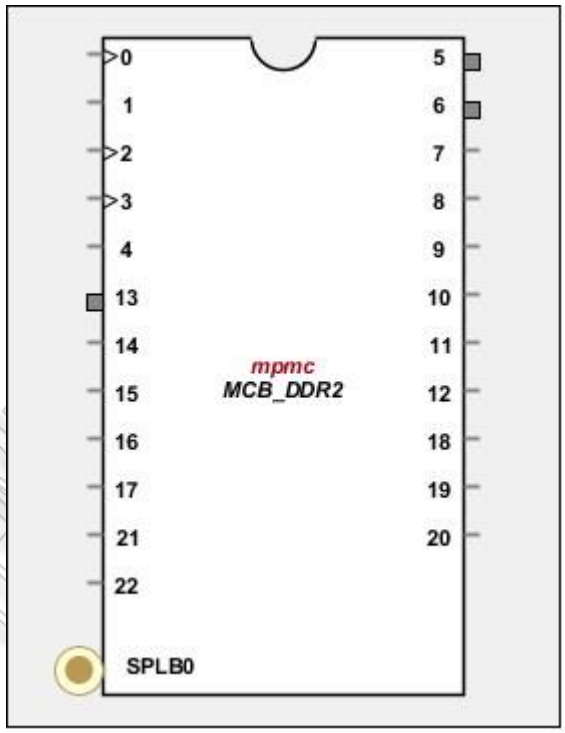
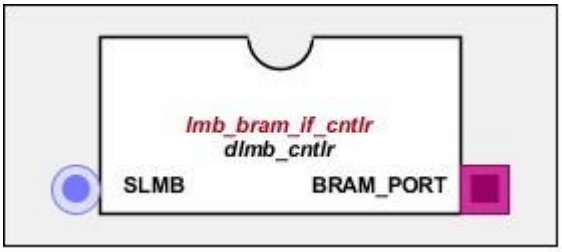
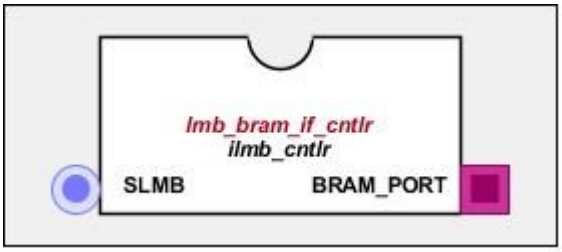


Εικόνα 5-3: Τελική αρχιτεκτονική του S-W accelerator peripheral

5.3. Περιγραφή του ενσωματωμένου συστήματος

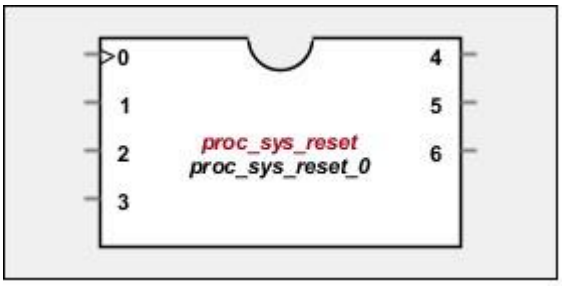
Όπως αναφέραμε και προηγουμένως, το ενσωματωμένο σύστημα που υλοποιεί την εφαρμογή μας αποτελείται κυρίως από ένα ενσωματωμένο επεξεργαστή Microblaze, στον οποίο τρέχει κομμάτι κώδικα σε γλώσσα C, που καλεί έναν S-W επιταχυντή φτιαγμένο στο υλικό και ενσωματωμένο στο σύστημα με το όνομα sw_accel_atlys_microblaze. Ωστόσο, για την αλληλεπίδραση αυτών των βασικών μονάδων του συστήματος, χρειάστηκαν να προστεθούν επιπλέον συστατικά τόσο για την σωστή λειτουργία της εφαρμογής, σύμφωνα με τις απαιτήσεις του συστήματος, όσο και τον έλεγχο/επιβεβαίωση της ορθότητάς της. Τα στοιχεία αυτά παρουσιάζονται αναλυτικά στον πίνακα που ακολουθεί:

1 MicroBlaze	
1 MicroBlaze Debug Module (MDM)	
1 Data Local Memory Bus (LMB) 1.0	
1 Instruction Local Memory Bus (LMB) 1.0	
1 Processor Local Bus (PLB) 4.6	

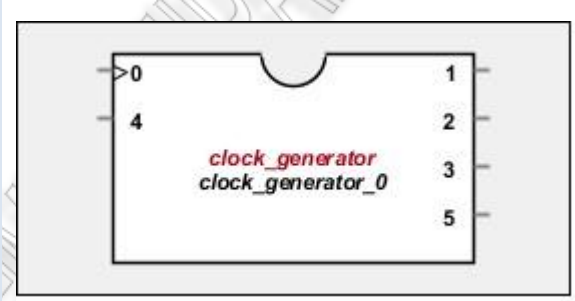
1 Block RAM (BRAM) Block	
1 Multi-Port Memory Controller(DDR/DDR2/SDRAM)	
1 Data LMB BRAM Controller	
1 Instruction LMB BRAM Controller	

1 XPS UART (Lite)

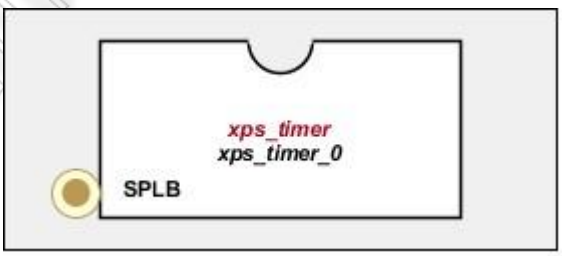
1 Processor System Reset Module



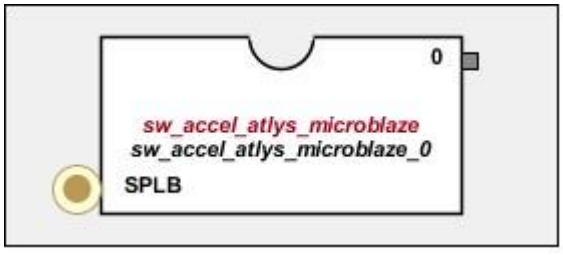
1 Clock Generator



1 XPS Timer/Counter



1 SW_ACCEL_ATLYS_MICROBLAZE



Παράμετροι του περιφερειακού **Sw_accel_atlys_microblaze**

Name	Value	Name	Value
C_BASEADDR	0x02000000	C_SPLB_CLK_PERIOD_PS	15000

C_HIGHADDR	0x0200FFFF	C_INCLUDE_DPHASE_TIMER	0
C_SPLB_AWIDTH	32	C_FAMILY	spartan6
C_SPLB_DWIDTH	32	C_MEM0_BASEADDR	0x00004000
C_SPLB_NUM_MASTERS	2	C_MEM0_HIGHADDR	0x00004FFF
C_SPLB_MID_WIDTH	1	C_MEM1_BASEADDR	0x00050000
C_SPLB_NATIVE_DWIDTH	32	C_MEM1_HIGHADDR	0x00051FFF
C_SPLB_P2P	0	C_MEM2_BASEADDR	0x00060000
C_SPLB_SUPPORT_BURSTS	0	C_MEM2_HIGHADDR	0x00061FFF
C_SPLB_SMALLEST_MASTER	32		

Συνολικά, λοιπόν, μπορούμε να πούμε ότι το ενσωματωμένο σύστημα αποτελείται κυρίως από τον ενσωματωμένο επεξεργαστή Microblaze, ο οποίος συνδέεται στο δίαυλο επικοινωνίας Processor Local Bus (PLB). Χρησιμοποιεί μία τοπικές μνήμες BRAM, η οποία διαχωρίζεται σε εντολές και δεδομένα του επεξεργαστή μεγέθους 64KB. Για την επικοινωνία των περιοχών της μνήμης που αφορούν τις εντολές και τα δεδομένα, με τον ενσωματωμένο επεξεργαστή Microblaze, χρειαζόμαστε δύο ξεχωριστούς ελεγκτές μνήμων (lmb bram controller). Αυτοί, αναλαμβάνουν την σύνδεση αυτών των μνημών μέσω δύο αντίστοιχων διαύλων τοπικής μνήμης (local memory bus - lmb), οι οποίοι στο σύστημα μας παίρνουν τα ονόματα ilmb και dlmb για τους διαύλους του χώρου μνήμης για τις εντολές και τα δεδομένα αντίστοιχα.

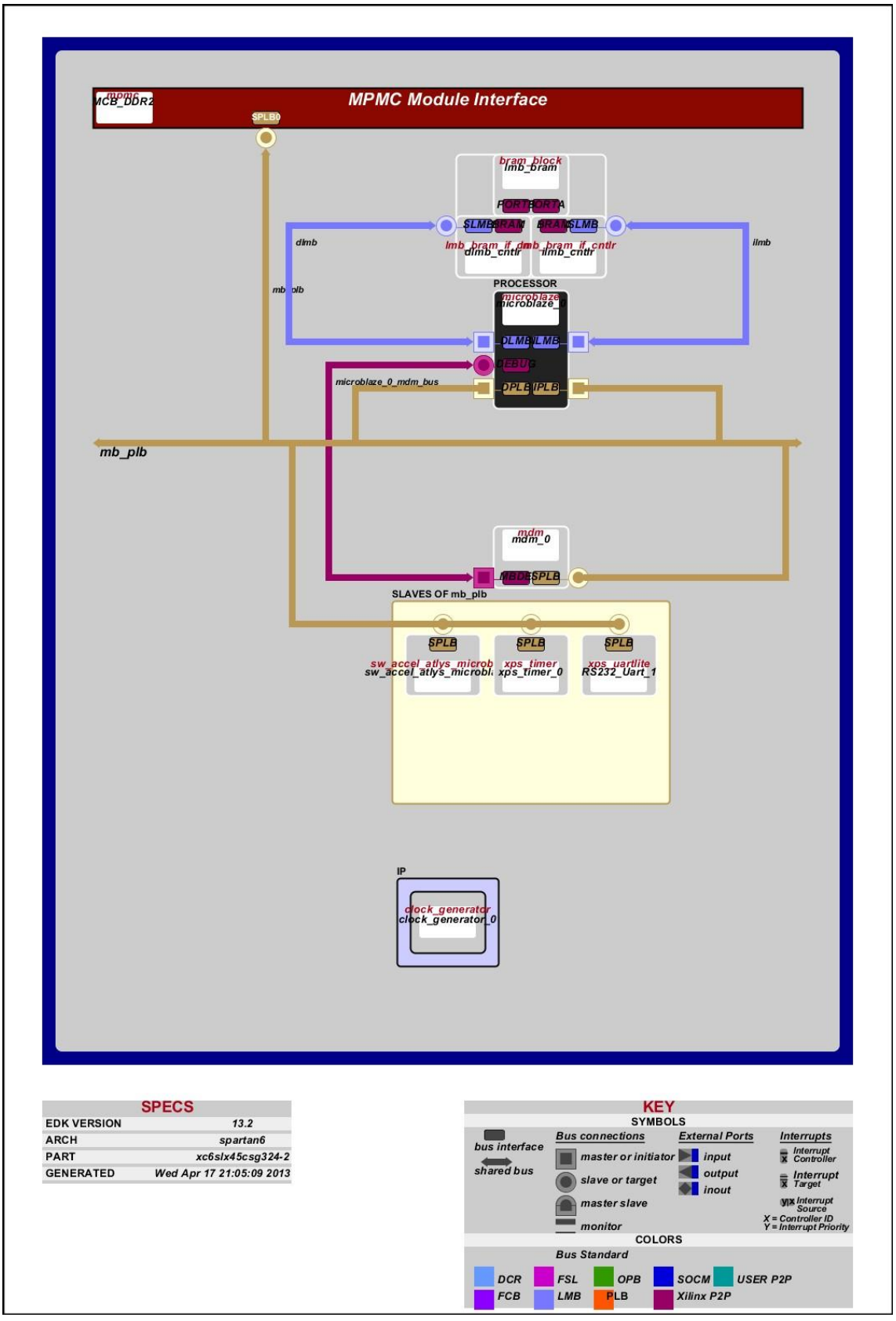
Εκτός της μνήμης του επεξεργαστή, έχει προστεθεί εξωτερική μνήμη DDR2 μεγέθους 128MB. Αυτό συνέβη, λόγω του ότι φτάσαμε στο συμπέρασμα, κατά τη διάρκεια προγραμματισμού της συσκευής, ότι η σχεδίαση μας έχει μεγάλες απαιτήσεις χωρητικότητας τόσο όσον αφορά το αρχείο προγραμματισμού του ενσωματωμένου επεξεργαστή (elf αρχείο), όσο και στα προαπαιτούμενα για να ξεκινήσει να τρέχει ο αλγόριθμος (διάβασμα δύο αρχείων ήχου τα οποία πρέπει να συγκριθούν, μεγέθους πάνω από 15 MB). Το γεγονός ότι η πλακέτα που χρησιμοποιούμε έχει ενσωματωμένο τσιπ 128 DDR έκανε την επιλογή μας πιο εύκολη.

Επιπλέον, για τον σωστό χρονισμό και επαναφορά του κυκλώματος έχουν προστεθεί τα περιφερειακά Clock Generator και Processor System Reset Module, ενώ για την μέτρηση του χρόνου εκτέλεσης του αλγορίθμου ενσωματώσαμε στο σύστημά μας έναν μετρητή XPS Timer/Counter.

Για τον έλεγχο της σωστής λειτουργίας του συστήματος έχουμε προσθέσει ένα περιφερειακό (XPS UART Lite) που ελέγχει τη θύρα USB UART της συσκευής, στην οποία αποστέλλουμε το stdout του κώδικα που τρέχει στον Microblaze. Επίσης, απαραίτητο για την αποσφαλμάτωση του κώδικα ήταν και ένα Microblaze Debug Module (mdm), το οποίο συνδέθηκε στο δίαυλο PLB μαζί με τα υπόλοιπα περιφερειακά για να επικοινωνεί μέσω αυτού με τον Microblaze.

Τελευταίο, έχουμε προσθέσει το περιφερειακό που δημιουργήσαμε και υλοποιεί το S-W επιταχυντή.

Όλα τα παραπάνω συστατικά που αποτελούν το ολοκληρωμένο ενσωματωμένο μας σύστημα, καθώς και οι μεταξύ τους συνδέσεις, απεικονίζονται στο σχήμα που ακολουθεί.



Εικόνα 5-4: Δομή του ολοκληρωμένου ενσωματωμένου συστήματος (XPS).

5.4. Περιγραφή διασύνδεσης μεταξύ S-W επιταχυντή και ενσωματωμένου επεξεργαστή

Όπως αναλύσαμε και σε προηγούμενη αναφορά μας στο ενσωματωμένο σύστημα της εφαρμογής, η υλοποίησή μας εκτελείται τμηματικά στο λογισμικό που τρέχει στον ενσωματωμένο επεξεργαστή και στο S-W επιταχυντή που προστίθεται σαν περιφερειακό στο σύστημα. Για να γίνει αυτό, θα πρέπει να υπάρχει επαρκής διασύνδεση μεταξύ των δύο (ενσωματωμένου επεξεργαστή και επιταχυντή), ώστε η επικοινωνία μεταξύ τους να είναι επιτυχής. Τη διασύνδεση αυτή προσφέρει εν μέρει ο δίαυλος επικοινωνίας PLB, μέσω του οποίου συνδέονται τα δύο περιφερειακά, καθώς και τα απαραίτητα σήματα που υποστηρίζουν τις λειτουργίες ανάγνωσης και εγγραφής δεδομένων από και προς το περιφερειακό επιταχυντή. Ωστόσο, για την επιτάχυνση της ροής δεδομένων του συστήματος αποφασίστηκε να δημιουργηθούν καταχωρητές και μνήμες στο περιφερειακό, η οποίες θα είναι άμεσα προσπελάσιμες από τον επεξεργαστή και κατ' επέκταση από το λογισμικό, εξαλείφοντας τυχόν χρόνο για λειτουργίες εισόδου/εξόδου ή αποστολής δεδομένων από και προς τον επιταχυντή.

Για το λόγο αυτό, κατά τη διαδικασία δημιουργίας του περιφερειακού στο εργαλείο Create and Import Peripheral Wizard επιλέχθηκαν οι λειτουργίες χρήσης καταχωρητών και περιοχών μνήμης (βλέπε σχετικά Παράρτημα Ι), για την επικοινωνία μεταξύ του λογισμικού και του επιταχυντή. Το ίδιο εργαλείο, παρήγαγε αυτόματα και τον οδηγό για το συγκεκριμένο περιφερειακό, γεννώντας συναρτήσεις, που μπορούν εύκολα να χρησιμοποιηθούν από κώδικα γραμμένα σε γλώσσα C, συμπεριλαμβάνοντας στον κώδικα την κατάλληλη βιβλιοθήκη (header file).

5.4.1. Χρήση οδηγού για την επικοινωνία με το περιφερειακό

Κατά τη δημιουργία του περιφερειακού, παράχθηκε αυτόματα και ο οδηγός μέσω του οποίου ορίζονται συναρτήσεις σε γλώσσα C για την προσπέλαση των καταχωρητών και των μνημών αυτού. Οι συναρτήσεις αυτές ορίζονται στα αρχεία sw_accel_atlys_microblaze.c και sw_accel_atlys_microblaze.h, τα οποία βρίσκονται στο φάκελο drivers του περιφερειακού.

Αρχικά, γίνονται ορισμένες δηλώσεις που αφορούν τις διευθύνσεις βάσης του επιταχυντή και των καταχωρητών, προτού δηλωθούν οι απαραίτητες συναρτήσεις για την εγγραφή και ανάγνωση από και προς το περιφερειακό. Παρακάτω βλέπουμε τα offset για τους καταχωρητές των 32 bits του περιφερειακού.

```
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG0_OFFSET  0x00000000
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG1_OFFSET  0x00000004
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG2_OFFSET  0x00000008
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG3_OFFSET  0x0000000C
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG4_OFFSET  0x00000010
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG5_OFFSET  0x00000014
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG6_OFFSET  0x00000018
#define SW_ACCEL_ATLYS_MICROBLAZE_SLV_REG7_OFFSET  0x0000001C
```

Για την ανάγνωση και εγγραφή δεδομένων από τους καταχωρητές και τις μνήμες που ορίστηκαν στον επιταχυντή, δηλώνονται οι ακόλουθες συναρτήσεις.

- `void SW_ACCEL_ATLYS_MICROBLAZE_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Data)`

Η συνάρτηση γράφει την τιμή Data (unsigned integer 32 bits) σε ένα καταχωρητή με offset *RegOffset* από τη βάση, για τη διεύθυνση βάσης *BaseAddress* του SW_ACCEL_ATLYS_MICROBLAZE περιφερειακού.

- `Xuint32 SW_ACCEL_ATLYS_MICROBLAZE_mReadReg(Xuint32 BaseAddress, unsigned RegOffset)`

Η συνάρτηση διαβάζει την τιμή *Data* (unsigned integer 32 bits) από ένα καταχωρητή με offset *RegOffset* από τη βάση, για τη διεύθυνση βάσης *BaseAddress* του SW_ACCEL_ATLYS_MICROBLAZE περιφερειακού.

- `void SW_ACCEL_ATLYS_MICROBLAZE_mWriteMemory(Xuint32 Address, Xuint32 Data)`

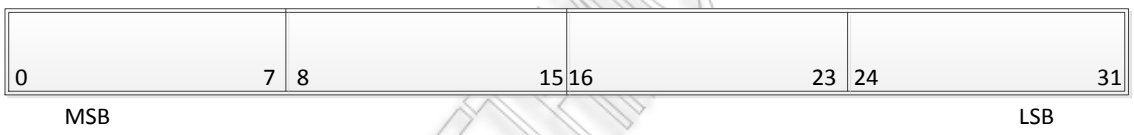
Η συνάρτηση γράφει την τιμή *Data* (unsigned integer 32 bits) στη διεύθυνση μνήμης *Address* του SW_ACCEL_ATLYS_MICROBLAZE.

- `Xuint32 SW_ACCEL_ATLYS_MICROBLAZE_mReadMemory(Xuint32 Address)`

Η συνάρτηση διαβάζει την τιμή *Data* (unsigned integer 32 bits) από τη διεύθυνση μνήμης *Address* του SW_ACCEL_ATLYS_MICROBLAZE.

5.4.2. Καταχωρητές

Συγκεκριμένα, οι καταχωρητές που ορίστηκαν για το περιφερειακό είναι μεγέθους των 32 bit, σε διάταξη big endian (διάταξη του Microblaze ενσωματωμένου επεξεργαστή στον οποίο εκτελείται η εφαρμογή), πράγμα που σημαίνει ότι το byte στην χαμηλότερη διεύθυνση μνήμης θα περιέχει και το πιο σημαντικό byte (most significant byte).



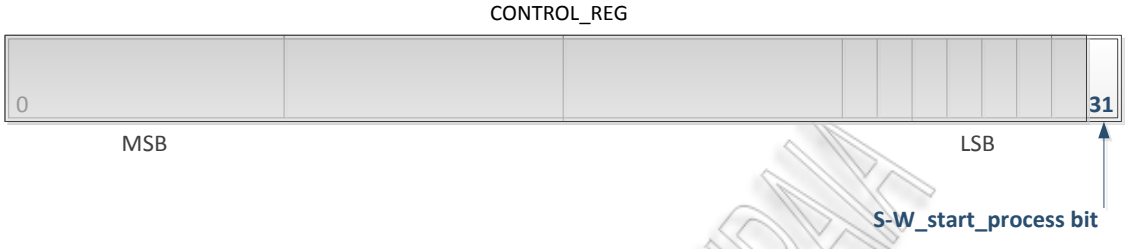
Εικόνα 5-5: Διάταξη καταχωρητών του S-W επιταχυντή.

Αυτοί οι καταχωρητές είναι οι εξής οκτώ:

CONTROL_REG	χρησιμοποιείται από τον ενσωματωμένο επεξεργαστή για να καθορίσει πότε θα εκκινήσει η S-W εκτέλεση
STATUS_REG	διαβάζεται συνεχώς από τον επεξεργαστή αμέσως μετά την ενεργοποίηση της S-W διαδικασίας και ενημερώνεται από τον επιταχυντή για το πέρας των εκτελέσεων
MAX_VAL_REG	περιέχει το μεγαλύτερο σκορ που σημειώθηκε κατά τη διάρκεια της ευθυγράμμισης
MAX_ADDR_REG	περιέχει τη διεύθυνση στην οποία σημειώθηκε το μέγιστο σκορ κατά τη διάρκεια της ευθυγράμμισης
PERM_NUM_REG	ενημερώνεται με το πλήθος των Permutations
PATTERN_SIZE_REG	ενημερώνεται με το μέγεθος του ηχητικού εφέ προς ανίχνευση
START_STOP_NODE_REG	περιέχει την αρχή των απενεργοποιημένων PEs
END_STOP_NODE_REG	περιέχει το τέλος των απενεργοποιημένων PEs

Καταχωρητής CONTROL_REG

Ο καταχωρητής CONTROL_REG χρησιμοποιείται από τον ενσωματωμένο επεξεργαστή για να καθορίσει πότε θα εκκινήσει η S-W εκτέλεση. Τίθεται στην τιμή 1, όταν ο Similarity matrix έχει ενημερώσει τη μνήμη DOWN_FIFO του S-W systolic array. Επομένως, το λογισμικό εξετάζει αν η τιμή του τελευταίου bit (bit 31) είναι ίση με '1' και ενεργοποιεί τον επιταχυντή.



Εικόνα 5-6: Καταχωρητής CONTROL_REG

Στο λογισμικό έχουμε:

```
ctrl_reg = (Xuint32)1;
…
SW_ACCEL_ATLYS_MICROBLAZE_mWriteReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_B
ASEADDR, CONTROL_REGISTER, ctrl_reg);
```

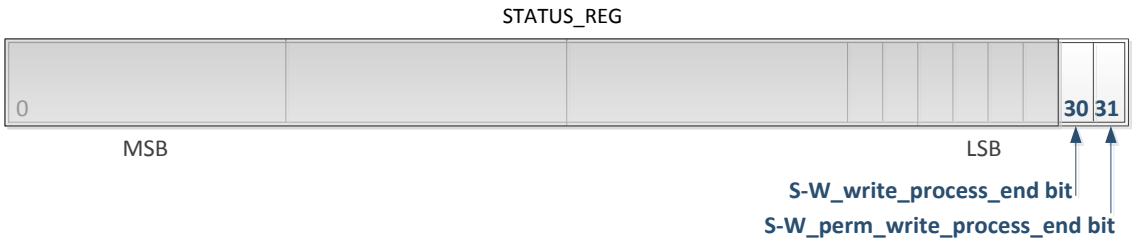
Ενώ στον επιταχυντή έχουμε:

```
usr_rd_start_o <= control_reg(31);
```

Καταχωρητής STATUS_REG

Ο καταχωρητής STATUS_REG διαβάζεται συνεχώς από τον επεξεργαστή αμέσως μετά την ενεργοποίηση της S-W διαδικασίας (όταν θέσουμε τον CONTROL_REG στην τιμή 1). Ενημερώνεται από τον S-W επιταχυντή και μπορεί να έχει τρεις διαφορετικές τιμές:

- Την τιμή '0', όταν δεν έχει ολοκληρωθεί καμία S-W εκτέλεση.
- Την τιμή '1', μετά το πέρας της πρώτης S-W εκτέλεσης (δεν έχουν ολοκληρωθεί τα Permutations). Τότε, ο ενσωματωμένος επεξεργαστής καταλαβαίνει ότι μπορεί να διαβάσει τα δεδομένα της μνήμης UP_FIFO (η οποία ενημερώνεται με τα αποτελέσματα των υπολογισμών του συστολικού πίνακα).
- Την τιμή '3' όταν, αντίστοιχα, έχουν τελειώσει όλες οι εκτελέσεις του συστολικού πίνακα (αρχική και Permutations). Τις αλλαγές αυτές καταλαβαίνει ο ενσωματωμένος επεξεργαστής και σταματάει το συνεχές διάβασμα. Όταν πάρει την τιμή 3 τότε ο επεξεργαστής θα μπορεί να διαβάσει την μνήμη PERM_FIFO η οποία θα έχει ήδη γεμίσει από το συστολικό πίνακα με τα μέγιστα σκορ.



Εικόνα 5-7: Καταχωρητής STATUS_REG

Στο λογισμικό έχουμε:

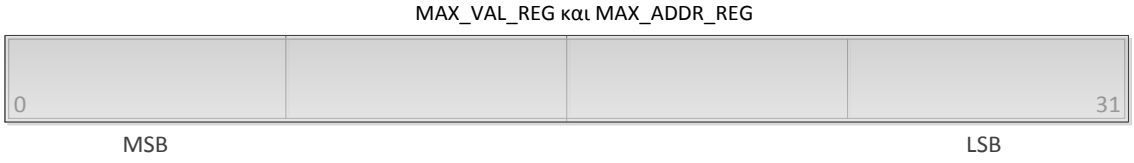
```
Reg32Value =
SW_ACCEL_ATLYS_MICROBLAZE_mReadReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_BA
SEADDR, STATUS_REGISTER);
…
while (Reg32Value != (Xuint32)0x03)
      {
      Reg32Value =
SW_ACCEL_ATLYS_MICROBLAZE_mReadReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_BA
SEADDR, STATUS_REGISTER);
      }
```

Ενώ στον επιταχυντή έχουμε:

```
if usr_wr_end_i = '1' then
         status_reg(31) <= '1';
elsif usr_perm_wr_end_i = '1' then
         status_reg(30) <= '1';
end if;
```

Καταχωρητές MAX_VAL_REG και MAX_ ADDR_REG

Ο καταχωρητής MAX_VAL_REG περιέχει το μεγαλύτερο σκορ που σημειώθηκε κατά τη διάρκεια της ευθυγράμμισης, ενώ ο καταχωρητής MAX_VAL_ADDR_REG περιέχει τη διεύθυνση στην οποία σημειώθηκε το μέγιστο αυτό σκορ. Οι καταχωρητές αυτές μπορούν να διαβαστούν από το λογισμικό όταν ο STATUS_REG πάρει την τιμή 3.



Εικόνα 5-8: Καταχωρητές MAX_VAL_REG και MAX_ADDR_REG

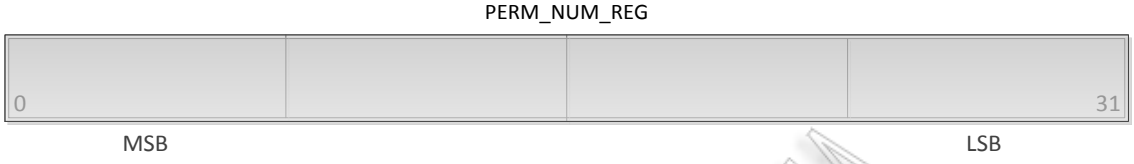
Στο λογισμικό έχουμε:

```
SW_ACCEL_ATLYS_MICROBLAZE_mReadReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_BA
SEADDR, MAX_VAL_REG);
…
```

```
SW_ACCEL_ATLYS_MICROBLAZE_mReadReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_BA
SEADDR, MAX_ADDR_REG);
```

Καταχωρητής PERM_NUM_REG

Ο ενσωματωμένος επεξεργαστής ενημερώνει τον καταχωρητή PERM_NUM_REG με το πλήθος των Permutations πριν από την εκκίνηση του επιταχυντή.



Εικόνα 5-9: Καταχωρητής PERM_NUM_REG

Στο λογισμικό έχουμε:

```
perm_num_reg = (Xuint32)NUM_OF_PERMUTATIONS;
…
SW_ACCEL_ATLYS_MICROBLAZE_mWriteReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_B
ASEADDR, PERMUTATIONS_NUMBER, perm_num_reg);
```

Καταχωρητής PATTERN_SIZE_REG

Ο ενσωματωμένος επεξεργαστής ενημερώνει τον καταχωρητή PATTERN_SIZE_REG με το με το μέγεθος του ηχητικού εφέ προς ανίχνευση πριν από την εκκίνηση του επιταχυντή.

Εικόνα 5-10: Καταχωρητής PATTERN_SIZE_REG

Στο λογισμικό έχουμε:

```
pattern_size_reg= (Xuint32)JJ;
…
SW_ACCEL_ATLYS_MICROBLAZE_mWriteReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_B
ASEADDR, PATTERN_SIZE_REGISTER, pattern_size_reg);
```

Καταχωρητές START_STOP_NODE_REG και END_STOP_NODE_REG

Οι καταχωρητές START_STOP_NODE_REG και END_STOP_NODE_REG κρατούν αντίστοιχα την αρχή και το τέλος των απενεργοποιημένων PEs και χρησιμοποιούνται σε περιπτώσεις όπου θέλουμε να εκτελέσουμε πολλαπλά περάσματα. Ενημερώνονται από τον ενσωματωμένο επεξεργαστή πριν από την εκκίνηση του επιταχυντή.

START_STOP_NODE_REG και END_STOP_NODE_REG

Εικόνα 5-11: Καταχωρητές START_STOP_NODE_REG και END_STOP_NODE_REG

Στο λογισμικό έχουμε:

```
start_node_reg = (Xuint32)0x0A;
…
SW_ACCEL_ATLYS_MICROBLAZE_mWriteReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_BASEADDR, START_STOP_NODE_REGISTER, start_node_reg);
…
stop_node_reg =(Xuint32)0x14;
…
SW_ACCEL_ATLYS_MICROBLAZE_mWriteReg(XPAR_SW_ACCEL_ATLYS_MICROBLAZE_0_BASEADDR, END_STOP_NODE_REGISTER, stop_node_reg);
```

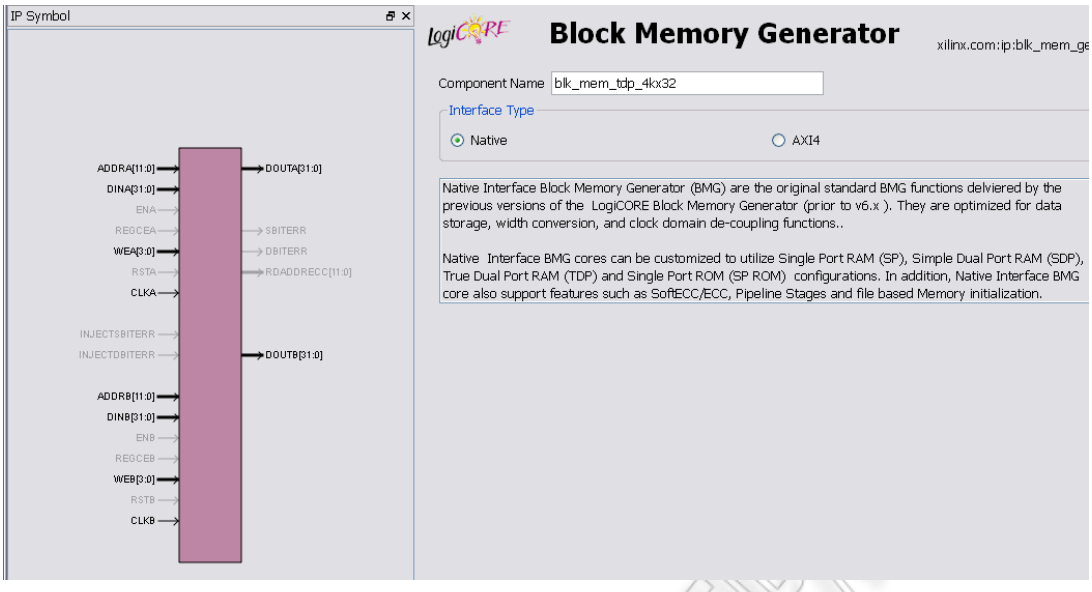
5.4.3. Μνήμες

Εκτός από τους καταχωρητές, στο περιφερειακό ορίζονται και οι παρακάτω μνήμες.

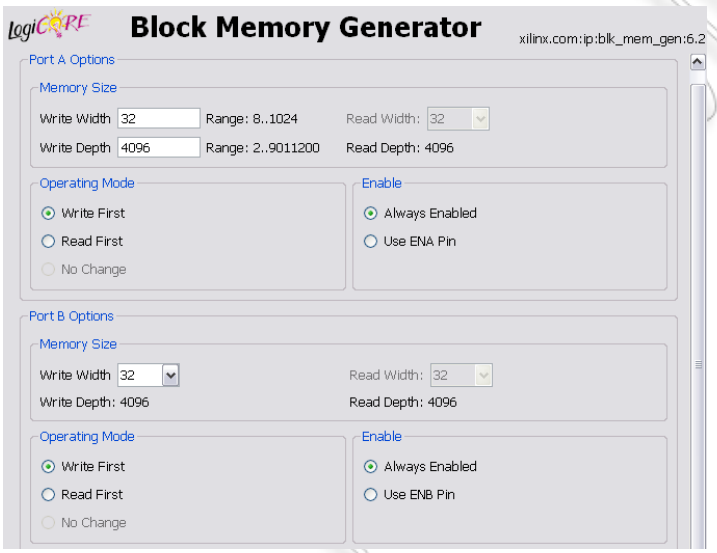
DOWN_FIFO	Ενημερώνεται από τον ενσωματωμένο επεξεργαστή με το αρχικό similarity matrix
UP_FIFO	Ενημερώνεται από το systolic array με τα αποτελέσματα των υπολογισμών
PERM_FIFO	Ενημερώνεται από το systolic array με τα μέγιστα σκορ κάθε permutation

Δηλώνοντας τη χρήση χώρου μνήμης για το περιφερειακό, το Create and Import Peripheral Wizard δημιουργεί τρεις ίδιες μνήμες των 256 θέσεων με μέγεθος δεδομένων 32 bits, καθώς και όλη τη λογική που χρειάζεται για να μεταφραστούν τα σήματα του PLB στο περιφερειακό. Συγκεκριμένα, για τη δική μας σχεδίαση έχουμε κρατήσει πολλά τα σήματα και τη λειτουργία τους, ενώ οι μνήμες αυτές έχουν αντικατασταθεί με μνήμες που έχουν γεννηθεί από το εργαλείο Core Generator της Xilinx.

Η DOWN_FIFO αντιστοιχίζεται στην blk_mem_tdp_4kx32, η οποία στην πράξη είναι μία Trual Dual Port RAM με βάθος 4096 στοιχείων και μέγεθος δεδομένων 32 bits.

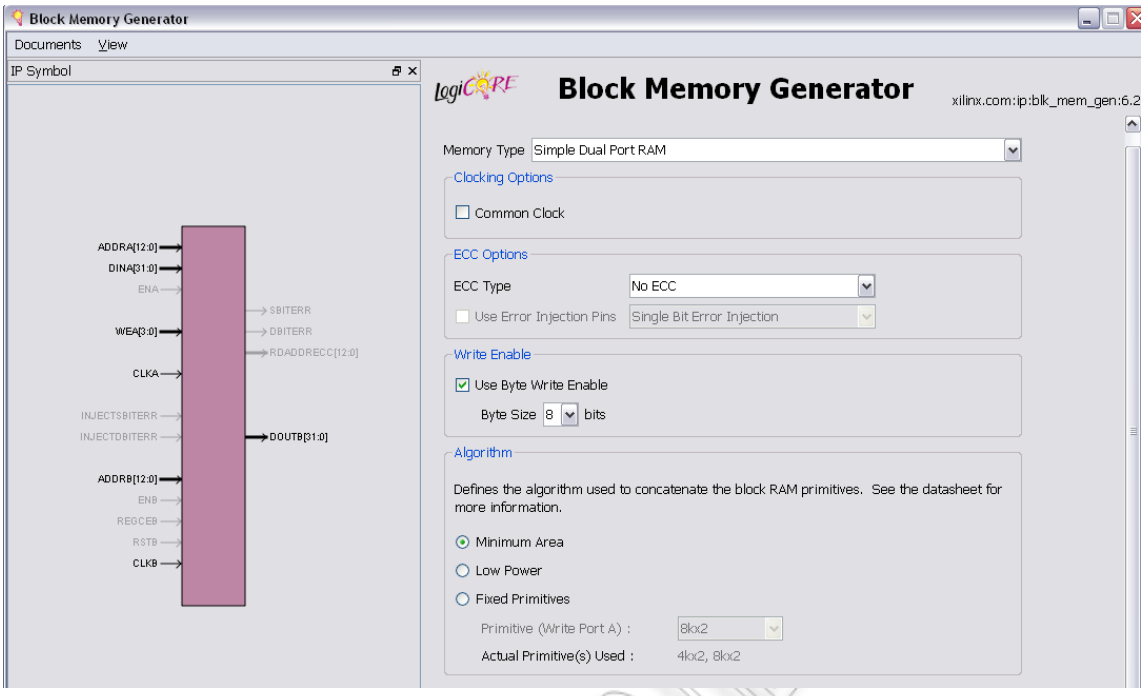


Εικόνα 5-12: CORE Generator για την Trual Dual Port RAM - DOWN FIFO (1/2)

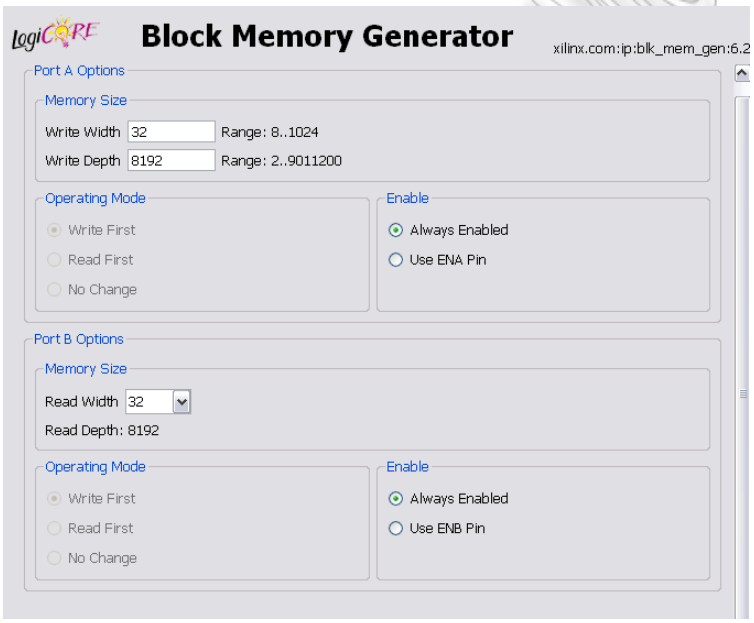


Εικόνα 5-13: CORE Generator για την Trual Dual Port RAM - DOWN FIFO (2/2)

Οι μνήμες UP_FIFO και PERM_FIFO αντιστοιχίζονται σε δύο ίδιες blk_mem_sdp_8kx32, οι οποίες αποτελούν δύο Simple Dual Port RAM με βάθος 8192 στοιχείων και μέγεθος δεδομένων 32 bits. Στο σημείο αυτό, αξίζει να σημειώσουμε ότι όλες οι μνήμες του συστήματος υποστηρίζουν εγγραφή και ανάγνωση από δυο θύρες (dual port rams), μία αποκλειστικά αφιερωμένη για την πρόσβαση του επεξεργαστή και μία για την πρόσβαση από τη μεριά του περιφερειακού.



Εικόνα 5-14: CORE Generator για την Simple Dual Port RAM - UP/PERM FIFO (1/2)



Εικόνα 5-15: CORE Generator για την Simple Dual Port RAM - UP/PERM FIFO (2/2)

Τέλος, αναφέρουμε ότι για το διάβασμα/γράψιμο σε καθεμία από αυτές τις μνήμες χρησιμοποιούμε και πάλι τις συναρτήσεις που ορίζονται στον οδηγό του περιφερειακού.

```
for ( Index = 0; Index < …; Index++ )
  {
```

```
    SW_ACCEL_ATLYS_MICROBLAZE_mWriteMemory(MemoryAddress+4*Index,
Data);
    Mem32Value=SW_ACCEL_ATLYS_MICROBLAZE_mReadMemory(MemoryAddress +
4 * Index);
  }
```

5.5. Ροή του τελικού συστήματος

Το software κομμάτι του συστήματος δέχεται αρχικά ως είσοδο 2 mat αρχεία. Τα αρχεία αυτά περιλαμβάνουν όλα τα απαραίτητα ηχητικά χαρακτηριστικά μίας ταινίας και ενός εφέ αντίστοιχα. Επίσης το λογισμικό δέχεται ως είσοδο τον επιθυμητό αριθμό των permutations, αλλά και το μέγεθος του εφέ (έως 127). Κατά την έναρξη του προγράμματος γίνονται κάποιες αρχικοποιήσεις, δηλώσεις πινάκων κ.τ.λ. Αρχικά, γράφονται οι καταχωρητές του FPGA που αφορούν τον αριθμό των permutations και το μέγεθος του εφέ. Οι δύο αυτές μεταβλητές έχουν δοθεί πριν το run time από τον χρήστη. Επίσης, διαβάζονται τα χαρακτηριστικά των δύο mat αρχείων και αποθηκεύονται σε 2 δισδιάστατους πίνακες. Εξ' αρχής γνωρίζουμε ότι ο αριθμός των PE του FPGA είναι 50. Ο λόγος που είναι 50, είναι επειδή αυτός είναι ο μέγιστος αριθμός PEs που καταφέραμε να χωρέσουμε μέσα στη σχεδίαση του LX45 Spartan-6 FPGA chip.

Από κει και πέρα ξεκινάει η κεντρική επανάληψη της εφαρμογής. Κάθε βρόχος εκτελείται για 50 καρέ της ταινίας. Έτσι λοιπόν έχουμε με την σειρά τα παρακάτω βήματα σε κάθε επανάληψη:

- Υπολογισμός του αρχικού πίνακα ομοιοτήτων με βάση τα φασματικά χαρακτηριστικά για 50 καρέ της ταινίας και ολόκληρου του εφέ. Αφού υπολογιστεί το αρχικό (δισδιάστατο) πίνακα ομοιοτήτων μετατρέπεται σε κατάλληλη μονοδιάστατη μορφή για να μπορέσει να αποτελέσει είσοδο για το συστολικό πίνακα.
- Όταν ο υπολογισμός και η μετατροπή του πίνακα ομοιοτήτων ολοκληρωθεί, το λογισμικό κατεβάζει τον μονοδιάστατο πίνακα στο FPGA γράφοντας τον στην μνήμη DOWN_FIFO, μέσω PCI-Express.
- Μόλις ολοκληρωθεί το κατέβασμα του πίνακα ομοιοτήτων στην πλακέτα, το λογισμικό γράφει την τιμή 1 στον καταχωρητή Control του FPGA. Με αυτόν τον τρόπο ενεργοποιεί το mode wr_ram με το οποίο η λογική ελέγχου της σχεδίασης ενημερώνει τις S-RAM των PE με τα στοιχεία που είναι αποθηκευμένα στην DOWN_FIFO, μέσω μίας αλυσίδας διαμόρφωσης. Χρησιμοποιείται ένας μετρητής (0 έως 49) για την απόδοση δεικτών (index) σε κάθε PE και ένας δεύτερος μετρητής (από 0 έως μέγεθος εφέ N — 1) για την παραγωγή διευθύνσεων αποθήκευσης για τις τοπικές S μνήμες. Το βήμα αυτό διαρκεί M x N κύκλους.
- Μετά την ενημέρωση όλων των τοπικών μνημών S ξεκινά η διαδικασία του S-W, η οποία εκτελείται μέσα στο συστολικό πίνακα. Τα νέα σκορ αλλά και οι πρόγονοι αποθηκεύονται μέσα στις μνήμες H. Το βήμα αυτό διαρκεί M + N — 1 κύκλους. Κατά την διάρκεια που το FPGA υπολογίζει τα κόστη, το λογισμικό εκτελεί polling καθώς διαβάζει διαρκώς τον καταχωρητή Status μέχρι να αλλάξει τιμή. Ο καταχωρητής Status αρχικά έχει την τιμή 0.
- Όταν ολοκληρωθούν οι υπολογισμοί (όλες οι μνήμες H είναι ενημερωμένες), τότε αλλάζει η κατάσταση λειτουργίας του συστολικού πίνακα σε rd_ram. Όταν γίνει η αλλαγή αυτή το περιεχόμενο των μνημών H ολισθαίνει προς τα έξω μέσω της αλυσίδας διαμόρφωσης και αποθηκεύεται στη μνήμη UP_FIFO. Επίσης ενημερώνεται ο καταχωρητής Max Value που κρατάει το μεγαλύτερο σκορ που σημειώθηκε κατά την διάρκεια της ευθυγράμμισης. Η διάρκεια του βήματος αυτού είναι ξανά M x N κύκλοι.
- Όταν ολοκληρωθεί η ενημέρωση της μνήμης UP_FIFO, το FPGA αλλάζει την τιμή του καταχωρητή Status από 0 σε 1. Με αυτόν τον τρόπο το λογισμικό σταματάει να είναι σε κατάσταση polling και ξεκινάει να διαβάζει την UP_FIFO μέσω του PCI-Express διαύλου. Μόλις όμως το λογισμικό διαβάσει όλα τα δεδομένα ξανακάνει έλεγχο του

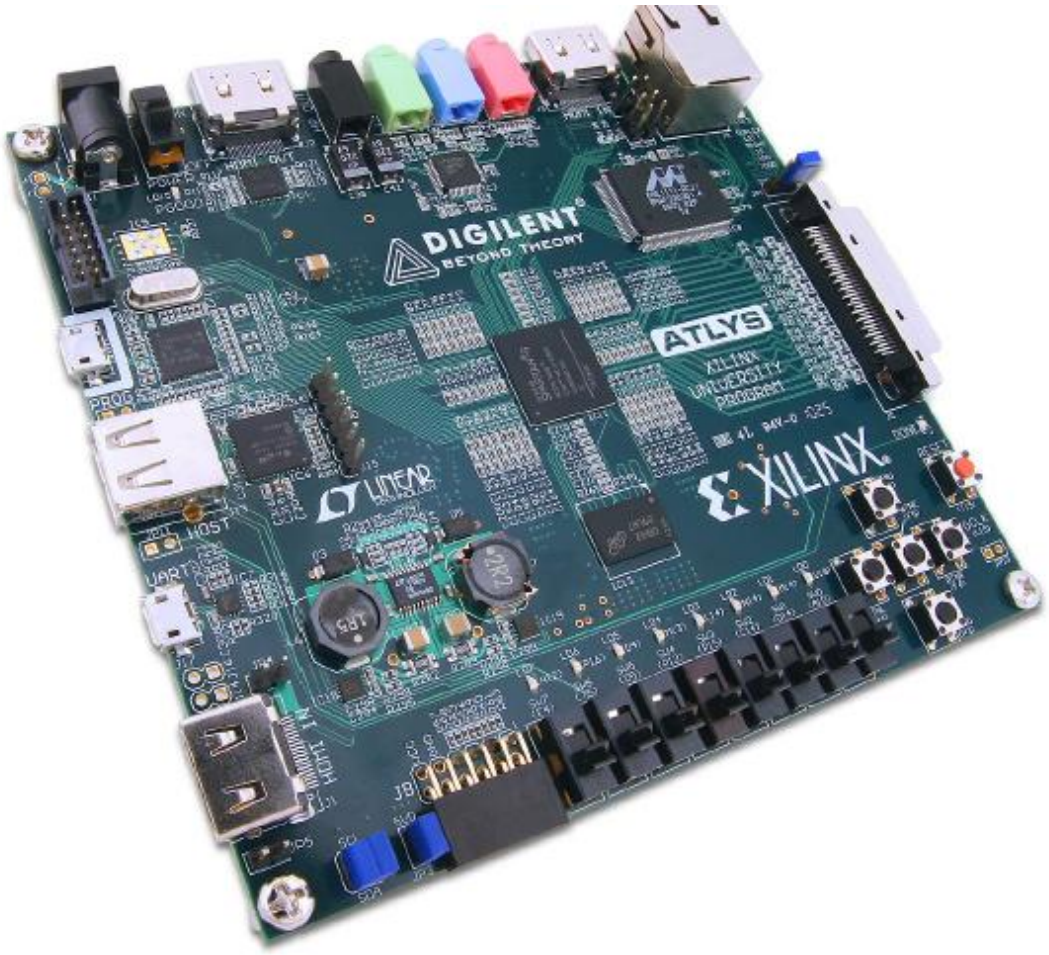
καταχωρητή Status, μέχρι αυτός να πάρει την τιμή 3, οπότε θα έχουν ολοκληρωθεί και τα permutations.

- Το FPGA όταν ολοκληρώσει την πρώτη εκτέλεση και ενημερωθούν μνήμη UP_FIFO και ο καταχωρητής Max Value, συνεχίζει αυτόματα με την εκτέλεση των permutations. Το πόσα permutations Θα εκτελεστούν εξαρτάται από την είσοδο που δώσαμε στην αρχή. Έστω ότι ορίσαμε αριθμό permutations K. Αυτό σημαίνει ότι η S-W διαδικασία θα τρέξει άλλες K φορές πριν πάμε στα επόμενα 49 καρέ ταινίας. Πριν όμως την κάθε permutation εκτέλεση γίνεται ένα ανακάτεμα της σειράς με την οποία είναι ταξινομημένα τα χαρακτηριστικά του ηχητικού εφέ. Οι S μνήμες παραμένουν ως έχουν. Αυτό που αλλάζει είναι η σειρά με την οποία θα γίνονται οι υπολογισμοί σε κάθε PE. Η σειρά αυτή δίνεται πάντα μέσω του Control Logic Component στο systolic συστολικό πίνακα. Επίσης, τα αποτελέσματα που μας ενδιαφέρουν από τα permutations είναι μόνο οι μέγιστες τιμές (max values). Ο χρόνος εκτέλεσης κάθε permutation ποικίλει, καθώς διαρκεί πάνω από M+N-1 κύκλους, καθώς συμβαίνουν αρκετά "freeze" σε κάθε εκτέλεση.
- Όταν ολοκληρωθούν και τα permutations, όλες οι μέγιστες τιμές είναι αποθηκευμένες στην μνήμη PERM_FIFO του FPGA. Επίσης ο καταχωρητής Status παίρνει την τιμή 3 που σημαίνει ότι το λογισμικό σταματάει το polling και διαβάζει όλα τα τυχαίες μέγιστες τιμές που βρίσκονται στην PERM_FIFO πάλι μέσω του PCI-Express διαύλου.
- Αν θέλουμε να τρέξουμε το σύστημα με πολλαπλά περάσματα τότε κάπου εδώ γίνεται και η χρήση των stop nodes. Τα stop nodes δίνονται παραμετρικά μέσα από το λογισμικό, ενημερώνοντας του αντίστοιχους καταχωρητές του FPGA.
- Τα βήματα εκτέλεσης επαναλαμβάνονται για τα επόμενα 50 καρέ της ταινίας.

5.6. Περιγραφή του υλικού και των εργαλείων σχεδίασης

5.6.1. Αναπτυξιακή Πλακέτα Digilent Atlys Spartan-6 LX45

Η Atlys πλακέτα κυκλώματος είναι μία πλήρης, έτοιμη προς χρήση αναπτυξιακή πλατφόρμα ψηφιακών κυκλωμάτων που βασίζεται σε αρχιτεκτονική FPGA Spartan-6 LX45 της Xilinx, -3 βαθμού ταχύτητα. Τόσο το μεγάλο σε μέγεθος FPGA, όσο και το σύνολο των τελευταίας τεχνολογίας περιφερειακών που βρίσκονται πάνω του, όπως η θύρα Gbit Ethernet, το HDMI Βίντεο, η 128MByte 16-bit DDR2 μνήμη, τα USB και οι θύρες ήχου κάνουν την πλακέτα Atlys ιδανική να φιλοξενήσει ένα ευρύ φάσμα ψηφιακών συστημάτων, συμπεριλαμβανομένων και υλοποιήσεων ενσωματωμένων επεξεργαστών βασισμένων στον Microblaze της Xilinx.



Εικόνα 5-16: Αναπτυξιακή πλακέτα Digilent Atlys.

Επιπλέον, η αναπτυξιακή πλακέτα Atlys είναι συμβατή με όλα τα CAD εργαλεία της Xilinx συμπεριλαμβανομένων του ChipScope, EDK, καθώς και του ISE, επομένως ο σχεδιασμός χρησιμοποιώντας αυτή την πλακέτα μπορεί να πραγματοποιηθεί χωρίς επιπλέον κόστος..

Το FPGA Spartan-6 LX45 έχει βελτιστοποιηθεί για λογική υψηλών αποδόσεων και προσφέρει:

• 6.822 slices, που το καθένα περιέχει τέσσερις 6 -LUTs εισόδου και οκτώ flip-flops

• 2.1Mbits γρήγορης μνήμης RAM

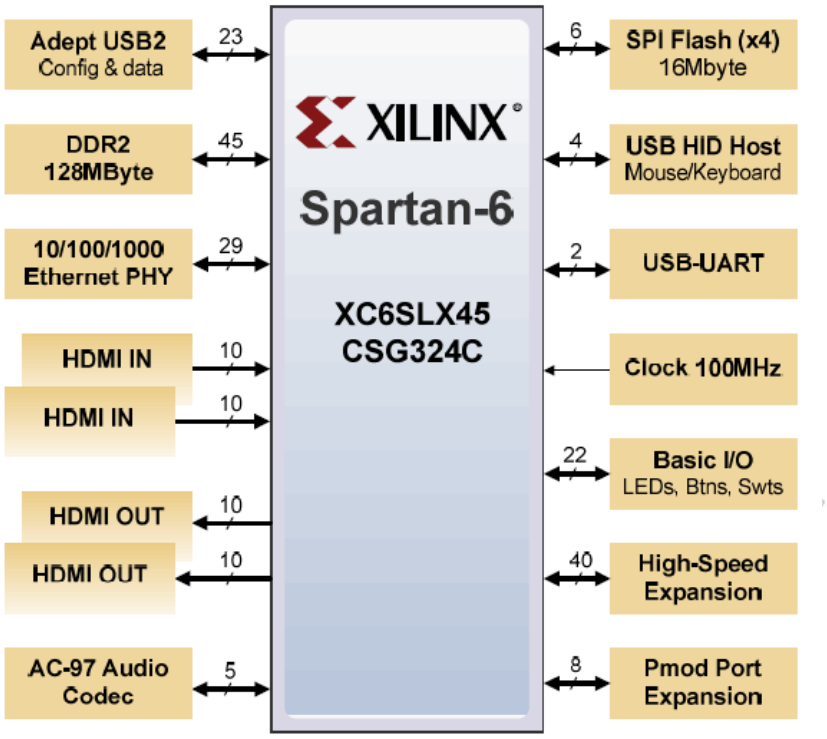
• τέσσερα πλακίδια ρολογιού (clock tiles) (οκτώ DCMS και τέσσερα PLLs)

• έξι φάση-κλειδωμένη βρόχο

• 58 slices DSP

• μεγαλύτερες από 500MHz ταχύτητες ρολογιού

Επιπλέον, στη συγκεκριμένη πλακέτα συμπεριλαμβάνεται και το τελευταίο σύστημα Adept USB2 της Digilent, το οποίο προσφέρει προγραμματισμό της συσκευής, παρακολούθηση σε πραγματικό χρόνο της παροχής ηλεκτρικού ρεύματος στην πλακέτα, αυτοματοποιημένες δοκιμές της συσκευής, εικονικά I/O, και απλοποιημένη διαδικασίες για τη μεταφορά δεδομένων του χρήστη.



Εικόνα 5-17: Διάγραμμα συστατικών για την πλακέτα Atlys XC6SLX45, πακέτο CSG324C.

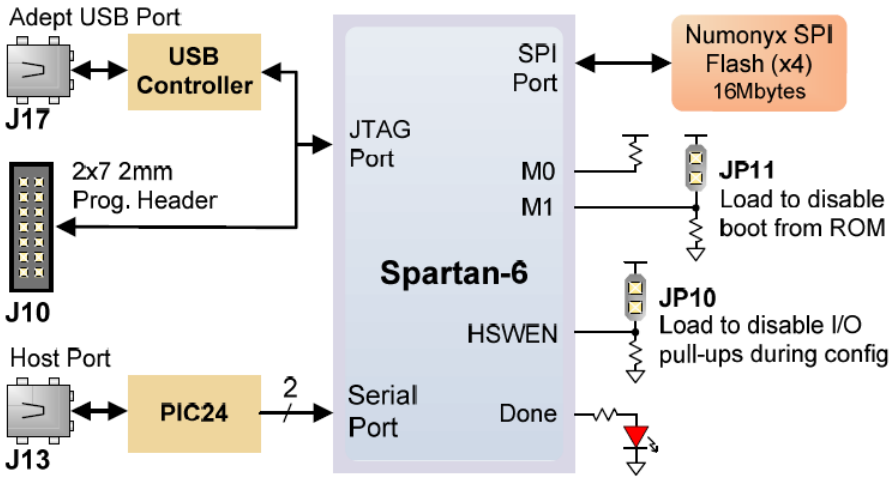
Χαρακτηριστικά:

- Xilinx Spartan-6 LX45 FPGA, πακέτο 324-pin BGA
- 128Mbyte DDR2 με 16-bit εύρος δεδομένων
- 10/100/1000 Ethernet PHY
- on-board USB2 θύρες για προγραμματισμό της συσκευής και μεταφορά δεδομένων
- USB-UART και USB-HID θύρα (για ποντίκι/πληκτρολόγιο)
- δύο HDMI video θύρες εισόδου και δύο HDMI θύρες εξόδου
- AC-97 Codec με line-in, line-out, μικρόφωνο και ακουστικό
- monitors ισχύος σε πραγματικό χρόνο
- 16Mbyte x4 SPI Flash για διαμόρφωση της συσκευής και αποθήκευση δεδομένων
- 100MHz CMOS ταλαντωτής
- 48 I/O κατευθυνόμενα προς υποδοχές επέκτασης (expansion connectors)
- GPIO συμπεριλαμβάνουν οκτώ LEDs, έξι buttons, και οκτώ διακόπτες
- Μέσα στο πακέτο περιλαμβάνονται μία 20W τροφοδοσία και καλώδιο USB

Διαμόρφωση:

Μετά την ενεργοποίηση, το FPGA πάνω στην Atlys πλακέτα του σκάφους πρέπει να διαμορφωθεί (ή να προγραμματιστεί), προκειμένου να εκτελέσει τις οποιεσδήποτε λειτουργίες.

Το FPGA μπορεί να διαμορφωθεί με τρεις τρόπους: ένα καλώδιο USB συνδεδεμένο σε υπολογιστή μπορεί να διαμορφώσει τη συσκευή χρησιμοποιώντας τη θύρα JTAG όταν η πλακέτα είναι ανοικτή, ένα αρχείο διαμόρφωσης αποθηκεύεται στη μνήμη SPI Flash ROM και μπορεί να μεταφερθεί αυτόματα στο FPGA με το άνοιγμά του, ή ένα αρχείο προγραμματισμού μπορεί να μεταφερθεί από ένα USB memory stick που είναι συνδεδεμένο στη θύρα USB HID της συσκευής.



Εικόνα 5-18: Επιλογές διαμόρμοφής στην πλακέτα Atlys.

Τόσο η Digilent όσο και η Xilinx διανέμουν ελεύθερα το λογισμικό που μπορεί να χρησιμοποιηθεί για να προγραμματίσει το FPGA και την ROM SPI. Τα αρχεία προγραμματισμού αποθηκεύονται εντός του FPGA σε κελιά μνήμης τύπου SRAM. Αυτά τα δεδομένα καθορίζουν τις συναρτήσεις λογικής του FPGA και τις συνδέσεις του κυκλώματος, οι οποίες εξακολουθούν να ισχύουν μέχρι να διαγραφούν με την αφαίρεση της τροφοδοσίας ή μέχρι να αντικατασταθεί από ένα νέο αρχείο διαμόρφωσης.

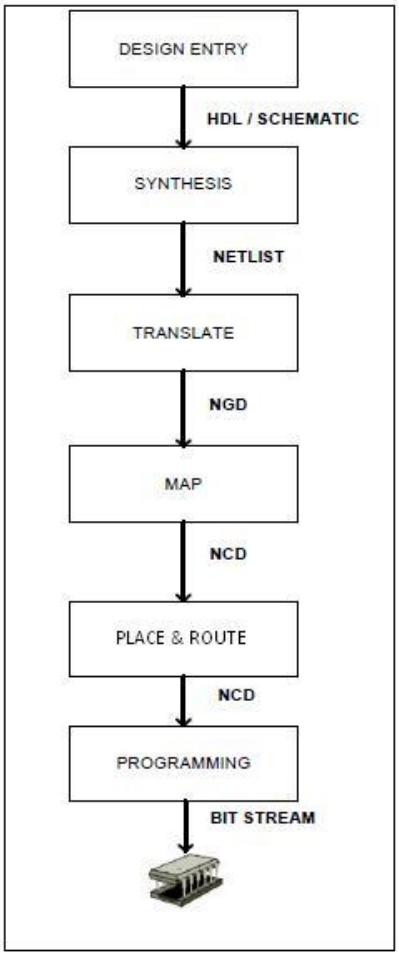
Τα αρχεία διαμόρφωσης του FPGA που μεταφέρονται μέσω της θύρας JTAG χρησιμοποιούν τους τύπους αρχείων .bin ή .svf, τα αρχεία τα οποία μεταφέρονται από ένα USB stick χρησιμοποιούν τους τύπους .bitfile, ενώ τα SPI αρχεία προγραμματισμού χρησιμοποιούν τους τύπους .bit, .bin, ή .mcs. Τα εργαλεία ISE και EDK της Xilinx μπορούν να δημιουργήσουν αρχεία τύπου .bit, .svf, .bin, ή αρχεία .mcs από πηγαία αρχεία VHDL, Verilog, ή Schematic-based. Το λογισμικό Adept της Digilent και το iMPACT της Xilinx μπορούν να χρησιμοποιηθούν για τον προγραμματισμό του FPGA ή της ROM χρησιμοποιώντας τη θύρα USB του Adept. Κατά τη διάρκεια του προγραμματισμού του FPGA, ένα .bit ή .svf αρχείο μεταφέρεται από τον υπολογιστή απευθείας στο FPGA χρησιμοποιώντας τη USB-JTAG θύρα.

Όταν προγραμματίζουμε την ROM, ένα .bit, .bin ή .mcs αρχείο μεταφέρεται στην ROM σε μια διαδικασία δύο σταδίων. Πρώτον, το FPGA προγραμματίζεται με ένα κύκλωμα που μπορεί να προγραμματίσει τη SPI ROM, και στη συνέχεια τα δεδομένα μεταφέρονται στην ROM μέσω του κυκλώματος FPGA (αυτή η πολυπλοκότητα είναι κρυμμένη από τον τελικό χρήστη και μόνο μία απλή διεπαφή "προγραμματισμός ROM" παρουσιάζεται στο χρήστη). Μετά τον προγραμματισμό της ROM, αυτή μπορεί να αυτόματα να διαμορφώσει το FPGA σε μεταγενέστερη ενεργοποίηση ή την επανεκκίνηση της συσκευής (εάν ο JP11 jumper δεν έχει φορτωθεί). Ένα αρχείο προγραμματισμού που είναι αποθηκευμένο στη SPI ROM θα παραμένει μέχρι να αντικατασταθεί, ανεξάρτητα από τη διακοπή ή όχι της τροφοδοσίας. Τέλος, το FPGA μπορεί να προγραμματιστεί από ένα memory stick που συνδέεται στη θύρα USB-HID, αν αυτό περιέχει ένα μοναδικό .bit αρχείο διαμόρφωσης στα περιεχόμενα του, το JP11 να έχει φορτωθεί (loaded), και η συσκευή να είναι σε ισχύ.

5.6.2. Διαδικασία σχεδιασμού πάνω σε FPGA

Η διαδικασία σχεδιασμού πάνω σε FPGA ακολουθεί 5 βήματα τα οποία φαίνονται στο παρακάτω σχήμα. Αυτά είναι τα εξής:

- Design Entry
- Simulation
- Synthesis
- Place& Route
- Programming



Εικόνα 5-19: Διαδικασία σχεδιασμού υλικού σε FPGA.

Design Entry: Σε αυτό το βήμα προχωρούμε στο σχεδιασμού του υλικού χρησιμοποιώντας μία γλώσσα περιγραφής υλικού (Hardware Description Language -HDL), όπως η VHDL ή Verilog. Τα αρχεία αυτά μπορεί να γραφτούν σε ένα απλό επεξεργαστή κειμένου ή σε έναν schematic editor της σουίτας εργαλείων της Xilinx. Σε καθένα από αυτά, ο σχεδιαστής έχει πλήρη ελευθερία χρήσης πόρων του συστήματος, δημιουργίας νέων περιφερειακών ή συστατικών της σχεδίασής του εφόσον τηρεί τους κανόνες ορισμού και δηλώσεων της γλώσσας περιγραφής στην οποία γράφει.

Synthesis: Σε αυτό το βήμα τα αρχεία σχεδιασμού που αποτελούν το υλικό μας (πάνω επίπεδο λογικής και τα συστατικά του) θα πρέπει να συνθέσουμε από τα αρχεία HDL ένα νέο αρχείο EDIF (Electronic Data Interchange Format). Στην περίπτωση μας, που χρησιμοποιούμε το εργαλείο Synthesis της Xilinx (Xilinx Synthesis Tool - XST), τότε μέσω αυτής της διαδικασίας

παράγεται ένα netlist αρχείο NGC (Netlist Constraints File) το οποίο μετατρέπεται αργότερα σε EDIF αρχείο. Και τα δύο αρχεία περιγράφουν την λίστα των σημάτων και των περιορισμών αυτών (constraints) μέσα στη λογική μας.

Translate: Η φάση αυτή είναι η πρώτη από τις οποίες αποτελείται η διαδικασία του Implementation της σχεδίασης. Σε αυτό το βήμα, συγχωνεύονται τόσο όλα τα netlists που περιγράφουν το κύκλωμα, όσο και οι περιορισμοί που έχουν τεθεί στη σχεδίαση (που έχουν αναδειχθεί από τις προηγούμενες διαδικασίες) και εξάγεται ένα νέου είδους αρχείου NGD (Xilinx Native Generic Database) κατάλληλο για εισαγωγή στην επόμενη φάση του Implementation που είναι το Map.

Map: Στη φάση Map, προσαρμόζεται η σχεδίαση στους διαθέσιμους πόρους (CLBs και IOBs) της συσκευής FPGA που έχουμε δηλώσει. Χρησιμοποιείται το αρχείο NGD που δημιουργήθηκε προηγουμένως, το οποίο ουσιαστικά εξετάζεται αν είναι συμβατό σύμφωνα με σχεδιαστικούς κανόνες (Design Rule Check). Η φάση αυτή παράγει ένα νέο είδους αρχείο NCD (Native Circuit Description), το οποίο δείχνει μία φυσική αναπαράσταση της σχεδίασης πάνω στο FPGA.

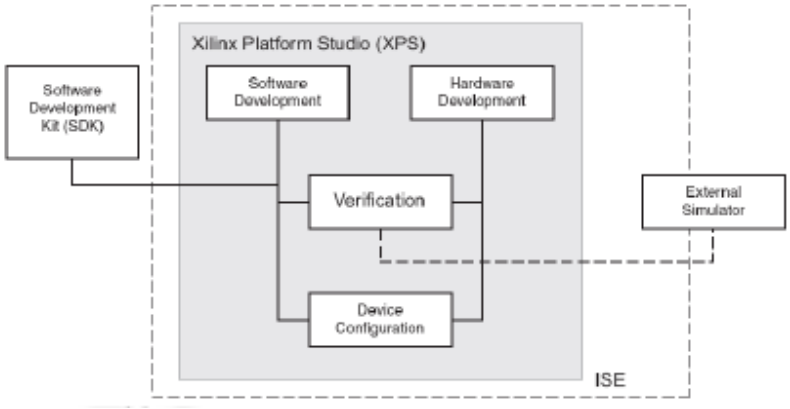
Place & Route: Σε αυτή τη φάση χρησιμοποιείται το αρχείο NCD που προέκυψε από τη προηγούμενη φάση και αφού ολοκληρωθεί παράγεται εκ νέου ανανεωμένο το NCD αρχείο, όπου περιέχει την πληροφορία για το που στο FPGA έχουν αντιστοιχιστεί τα στοιχεία του κυκλώματος σύμφωνα με τους διαθέσιμους πόρους του FPGA. Για την διαδικασία αυτή, προγραμματίζονται οι στοιχειώσεις διακόπτες του FPGA με τέτοιο τρόπο ώστε να δρομολογούνται τα σήματα μεταξύ των λογικών στοιχείων του κυκλώματος, τηρώντας τους χρονικούς περιορισμούς που έχουν τεθεί στο αρχείο NCD.

Programming: Αφού το design έχει περάσει επιτυχώς τη διαδικασία του Place and Route, πρέπει να διαμορφώσουμε τη συσκευή έτσι ώστε να εκτελέσει τη επιθυμητή λειτουργία. Αυτή η διαδικασία γίνεται χρησιμοποιώντας το αρχείο NCD που παρήγαγε η προηγούμενη φάση ως είσοδο στη διαδικασία Bitgen, από όπου παράγεται το αρχείο bitstream (binary BIT αρχείο) που θα διαμορφώσει τη συσκευή FPGA με το υλικό που σχεδιάσαμε.

Το αρχείο BIT περιέχει όλες τις ρυθμίσεις και τις πληροφορίες που προέρχονται από το αρχείο NCD, ορίζοντας την εσωτερική λογική και τις διασυνδέσεις του FPGA, καθώς επίσης και ειδικές παραμέτρους που αφορούν τη συσκευή. Το αρχείο BIT “κατεβαίνει” στη μνήμη του FPGA.

5.6.3. Σουίτα Σχεδίασης υλικού Xilinx Embedded Development Kit

Το ενσωματωμένο σύστημα σχεδιάστηκε και υλοποιήθηκε στη σουίτα σχεδίασης υλικού Embedded Development Kit (EDK) της εταιρείας Xilinx. Το EDK αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης ενσωματωμένων συστημάτων, καθώς περιλαμβάνει μία πλήρη συλλογή εργαλείων σχεδίασης, υλοποίησης, προγραμματισμού, αποσφαλμάτωσης και πρωτοτυποποίησης ενός ενσωματωμένου συστήματος σε μία συσκευή FPGA. Το περιβάλλον EDK διευκολύνει τον σχεδιαστή στην διαδικασία ανάπτυξης ενός συστήματος, δομώντας τη πάνω σε συγκεκριμένες σχεδιαστικές ροές, ανάλογα με τις προδιαγραφές του. Παρακάτω, αναφέρουμε, επιγραμματικά, την γενική χρήση των προγραμμάτων αυτών, σε ποιο κομμάτι της υλοποίησή μας χρησιμοποιήθηκαν και με ποιο σκοπό. Περισσότερες πληροφορίες, καθώς και παραδείγματα χρήσης των εργαλείων αυτών παρέχονται στην ιστοσελίδα της Xilinx, ενώ στα αντίστοιχα forums θα βρείτε μία αρκετά μεγάλη επιστημονική κοινότητα σχεδιαστών υλικού να σχολιάσει και να λύσει τυχόν απορίες και προβλήματα. Αναφέρουμε, τέλος, ότι η έκδοση της σουίτας εργαλείων Xilinx που χρησιμοποιήσαμε για την διεκπεραίωση αυτής της διατριβής είναι η 13.2.



Εικόνα 5-20: Βασική δομή του περιβάλλοντος EDK της Xilinx

Xilinx Platform Studio (XPS)

Η σχεδίαση ξεκινάει από το Xilinx Platform Studio, το οποίο είναι υπεύθυνο για την ανάπτυξη της βασικής πλατφόρμας του υλικού, πάνω στο οποίο θα χτιστεί το σύστημα. Χρησιμοποιώντας αυτό το πρόγραμμα, ο σχεδιαστής υλικού είναι σε θέση να δημιουργήσει ένα νέο ενσωματωμένο σύστημα ή να τροποποιήσει ένα ήδη υπάρχον, όπως αυτός επιθυμεί. Πρόκειται για το βασικότερο εργαλείο της σχεδίασης του ενσωματωμένου συστήματος, αφού εδώ ο σχεδιαστής έχει γενικότερη εποπτεία του συστήματος, ενώ ταυτόχρονα έχει τη δυνατότητα να πάρει τις αποφάσεις που χρειάζεται για τα συστατικά τα οποία θα περιέχει το σύστημα. Μπορεί να ενσωματώσει στο σχέδιο έναν επεξεργαστή, να καθορίσει τα περιφερειακά που θα επικοινωνούν μαζί του, τις θύρες και τις διευθύνσεις μνήμης τους, και να τα εντάξει στο σύστημα με μεγάλη ευκολία, αφού το πρόγραμμα παρέχει αυτοματοποιημένες λύσεις και παράγει αυτόματα τα αρχεία που συνθέτουν το σύστημα. Αρκεί από τον σχεδιαστή απλά να σύρει ένα περιφερειακό IP Core από την λίστα των διαθέσιμων IP στο σύστημα, και αυτό να το ενσωματώσει αμέσως στο σύστημα. Επιπλέον, εδώ ο σχεδιαστής έχει τη δυνατότητα να δημιουργήσει το τελικό αρχείο διαμόρφωσης το οποίο μπορεί να ‘κατεβάσει’ στην πλακέτα μέσω της θύρας JTAG της συσκευής. Σχετικά με την παρούσα διατριβή, το συγκεκριμένο πρόγραμμα αποτέλεσε και το μεγαλύτερο κομμάτι της υλοποίησης. Μέσω αυτού, κατασκευάσαμε το σύστημα με τον ενσωματωμένο επεξεργαστή Microblaze, τον οποίο και συνδέσαμε στο δίαυλο Processor Local Bus (PLB), επιλέξαμε και συνδέσαμε τα περιφερειακά πάνω σε αυτό το δίαυλο, ενώ παράλληλα ορίσαμε τις διευθύνσεις αυτών στη μνήμη. Επιπλέον, μέσω του εργαλείου Create and Import Peripheral Wizard, που θα αναλυθεί εκτενώς και στη συνέχεια, δημιουργήσαμε ένα νέο περιφερειακό IP Core, το οποίο αποτελεί την υλοποίηση του S-W επιταχυντή και το συνδέσαμε στον ίδιο δίαυλο επικοινωνίας.

Software Development Kit (SDK)

Όλη την παραπάνω πληροφορία για την σύσταση του συστήματος, τις θύρες, τις συνδέσεις και τις διευθύνσεις μνήμης του, μπορεί να εξαχθεί από το πρόγραμμα XPS και να εισαχθεί στο SDK. Εδώ, ο μηχανικός υλικού θα έχει την δυνατότητα να γράψει λογισμικό που θα τρέχει στον ενσωματωμένο επεξεργαστή του συστήματος. Για την επικοινωνία των συστατικών του συστήματος με τον κώδικα που τρέχει στον, Microblaze για εμάς, ενσωματωμένο επεξεργαστή, δημιουργούνται αυτόματα οδηγοί περιφερειακών, έτοιμοι προς χρήση από τον σχεδιαστή. Επιπλέον, αξίζει να αναφέρουμε πως το περιβάλλον αυτό δεν προσφέρεται μόνο για τη συγγραφή του κώδικα σε γλώσσα C, άλλα και για την απασφαλμάτωση αυτού, την επιλογή του προγραμματισμού του FPGA από τη γραμμή εντολών και την παρακολούθηση την μνήμης και των καταχωρητών του επεξεργαστή καθώς και των μεταβλητών του κώδικα σε πραγματικό χρόνο. Τέλος, από το συγκεκριμένο πρόγραμμα παράγεται το αρχείο προγραμματισμού του

ενσωματωμένου επεξεργαστή elf , το οποίο ‘κατεβαίνει’ στην συσκευή, αφού αυτή έχει ήδη διαμορφωθεί με τη σχεδίαση του υλικού από το αρχείο διαμόρφωσης .bit.

ISE Project Navigator

Με το συγκεκριμένο πρόγραμμα, πραγματοποιήθηκε η ανάπτυξη του S-W επιταχυντή. Ο σχεδιαστής έχει τη δυνατότητα να δημιουργεί και να τροποποιεί πηγαία αρχεία ή βιβλιοθήκες από τις οποίες αποτελείται το υλικό του. Η γλώσσα συγγραφής των αρχείων αυτών μπορεί να είναι οποιαδήποτε ή και συνδυασμός και των δύο από τις γλώσσες περιγραφής υλικού (HDL), VHDL και Verilog. Επιπλέον, εδώ ο σχεδιαστής έχει τη δυνατότητα να πραγματοποιήσει όλες τις διαδικασίες επικύρωσης και υλοποίησης του υλικού του (Synthesize, Translate, Map, Place & Route, Bitstream Genegation). Επομένως, ο σχεδιαστής ελέγχει το συντακτικό, τα παράγωγα της σύνθεσης του υλικού του, τη δημιουργία των σημάτων μέσα στη σχεδίαση του, καθώς και την αντιστοίχιση του στο FPGA, όλα από ένα εργαλείο. Οι διαδικασίες αυτές περιγράφονται αναλυτικά σε υποκεφάλαιο σχετικό με τη διαδικασία σχεδίασης πάνω σε FPGA.

ISE Core Generator

Ο σχεδιαστής, αν δεν θέλει να δημιουργήσει ένα core από την αρχή μέσα από το ISE Project Navigator, μπορεί να επιλέξει να ‘γεννήσει’ ένα ήδη υπάρχον IP Core, το οποίο αργότερα θα χρησιμοποιήσει μέσα στο σύστημά του, κάνοντας χρήση του εργαλείου Core Generator. Τέτοια cores μπορεί να είναι FIFO, Distributed RAM, PCI-Express κ.ά. Δίνεται, έτσι, η δυνατότητα στο χρήστη με ελάχιστο κόπο να δημιουργήσει έτοιμα πηγαία αρχεία που περιγράφουν τα cores αυτά, παρέχοντας μόνο στο interface του Core Generator wizard κάποιες παραμέτρους σχεδίασης τους. Τα αρχεία αυτά αποθηκεύονται στο υποφάκελο ip_cores του γενικότερου IP που τα περιλαμβάνει και από εκεί μπορούν να επιλεγούν να ενσωματωθούν στην ISE σχεδίαση, δηλώνοντας τα σαν συστατικά και συνδέοντας τα με σήματα της υπάρχουσας σχεδίασης (instantiate).

6. Συμπεράσματα – Περίληψη

Η παρούσα διπλωματική διατριβή καλύπτει την ανάπτυξη και υλοποίηση του αλγορίθμου Smith-Waterman για την ανίχνευση ηχητικών εφέ σε ταινίες αποκλειστικά σε ένα ενσωματωμένο σύστημα. Αφού, παρουσιάσαμε την αναγκαιότητα χρήσης του υλικού για την πολυπλοκότητα τόσο πολύπλοκων αλγορίθμων, καταλήξαμε ότι θα πρέπει η σχεδίαση μας να προσανατολιστεί στο υλικό και στην επιτάχυνση που αυτό μπορεί να επιφέρει στο χρόνο εκτέλεσης της εφαρμογής, με την ανάπτυξη ενός επιταχυντή υπολογισμών. Ωστόσο, εκτός από την μελέτη της επιτάχυνσης που μπορεί να επέλθει με τη δημιουργία και χρήση ενός επιταχυντή υπολογισμών, σε σχέση με την υλοποίηση στο λογισμικό, η παρούσα διατριβή στόχευε στη διερεύνηση και πραγματοποίηση της μεταφοράς του συνόλου της εφαρμογής αποκλειστικά σε μία ενσωματωμένη πλατφόρμα FPGA, εστιάζοντας στην φορητότητα της εφαρμογής σε σχέση με προηγούμενες υλοποιήσεις. Στην παρούσα μεταπτυχιακή διατριβή, αναπτύξαμε ένα πλήρες ενσωματωμένο σύστημα, τα κύρια συστατικά του οποίου θεωρούμε τον ενσωματωμένο επεξεργαστή που καλείται να τρέξει το λογισμικό της εφαρμογής και τον επιταχυντή υπολογισμών που εκτελεί τους υπολογισμούς του αλγορίθμου. Επιπρόσθετα με την χρήση του ενσωματωμένου επεξεργαστή, μπορέσαμε να επιλέξουμε ακριβώς τα συστατικά τα οποία απαιτούσε η σχεδίασή μας, να τα τροποποιήσουμε στο βαθμό που ήταν δυνατό και να ενσωματώσουμε στο σύστημα, ώστε να καλύψουμε τις απαιτήσεις της εφαρμογής.

Το γεγονός, ωστόσο, ότι ολόκληρη η εφαρμογή υλοποιείται αποκλειστικά σε ένα ενσωματωμένο σύστημα FPGA, μας περιόρισε χωρικά στο μέγεθος της υλοποίησης, αφού ολόκληρο το σύστημα (ενσωματωμένος επεξεργαστής, περιφερειακά, απαραίτητες μνήμες, μετρητές, συστολικός πίνακας) θα έπρεπε να χωράει στην αναπτυξιακή πλακέτα FPGA που επιλέχθηκε αρχικά για την ανάπτυξη της εφαρμογής. Αυτό, είχε σαν αποτέλεσμα να καταφέρουμε να χωρέσουμε μόνο 50 μονάδες επεξεργασίας στο συστολικό πίνακα του επιταχυντή. Σε περίπτωση, που καταφέρναμε μεγαλύτερο μέγεθος συστολικού πίνακα, είτε επιλέγοντας μεγαλύτερη αναπτυξιακή πλακέτα, είτε τροποποιώντας ή και αφαιρώντας κάποια συστατικά του συστήματος μέσω διαφορετικού λειτουργικού συστήματος (linux operating on microblaze), είμαστε σε θέση να γνωρίζουμε πως η σχεδίαση μας θα επιτύχαινε πολύ μεγαλύτερο ποσοστό επιτάχυνσης.

Συγκεκριμένα, η παρούσα διατριβή βασίστηκε σε ένα υπάρχον σύστημα ανίχνευσης ηχητικών εφέ σε ταινίες με χρήση του αλγορίθμου Smith-Waterman, τμήμα του οποίου υλοποιήθηκε στο πλαίσιο της μεταπτυχιακής διατριβής [2]. Το συγκεκριμένο σύστημα περιλαμβάνει υλοποίηση του S-W σε υλικό για την επιτάχυνση της διαδικασίας της ανίχνευσης των εφέ και υλοποίηση σε γλώσσα C του υπόλοιπου τμήματος της εφαρμογής. Το λογισμικό είναι προσαρμοσμένο να εκτελείται σε λειτουργικό σύστημα Linux,. Στην εφαρμογή αυτή, ο Η/Υ με λειτουργικό linux έκανε την προ-επεξεργασία των δύο αρχείων προς σύγκριση, εξήγαγε τα χαρακτηριστικά του ήχου και κατασκεύαζε τον πίνακα ομοιοτήτων για την έναρξη του S-W επιταχυντή. Την αποστολή αυτού του πίνακα στον επιταχυντή αναλάμβανε ένας οδηγός συσκευής, ο οποίος έπαιρνε τα στοιχεία από τον κώδικα C και τα έστελνε στην PCI Express θύρα της αναπτυξιακής πλακέτας. Από την άλλη, ο επιταχυντής περιελάμβανε εκτός από το κυρίως σώμα των υπολογισμών, και τη σχεδίαση της PCI Express, ώστε να μπορεί να δέχεται τα δεδομένα που φτάνουν στη θύρα αυτή και να εκτελεί μετέπειτα τους υπολογισμούς.

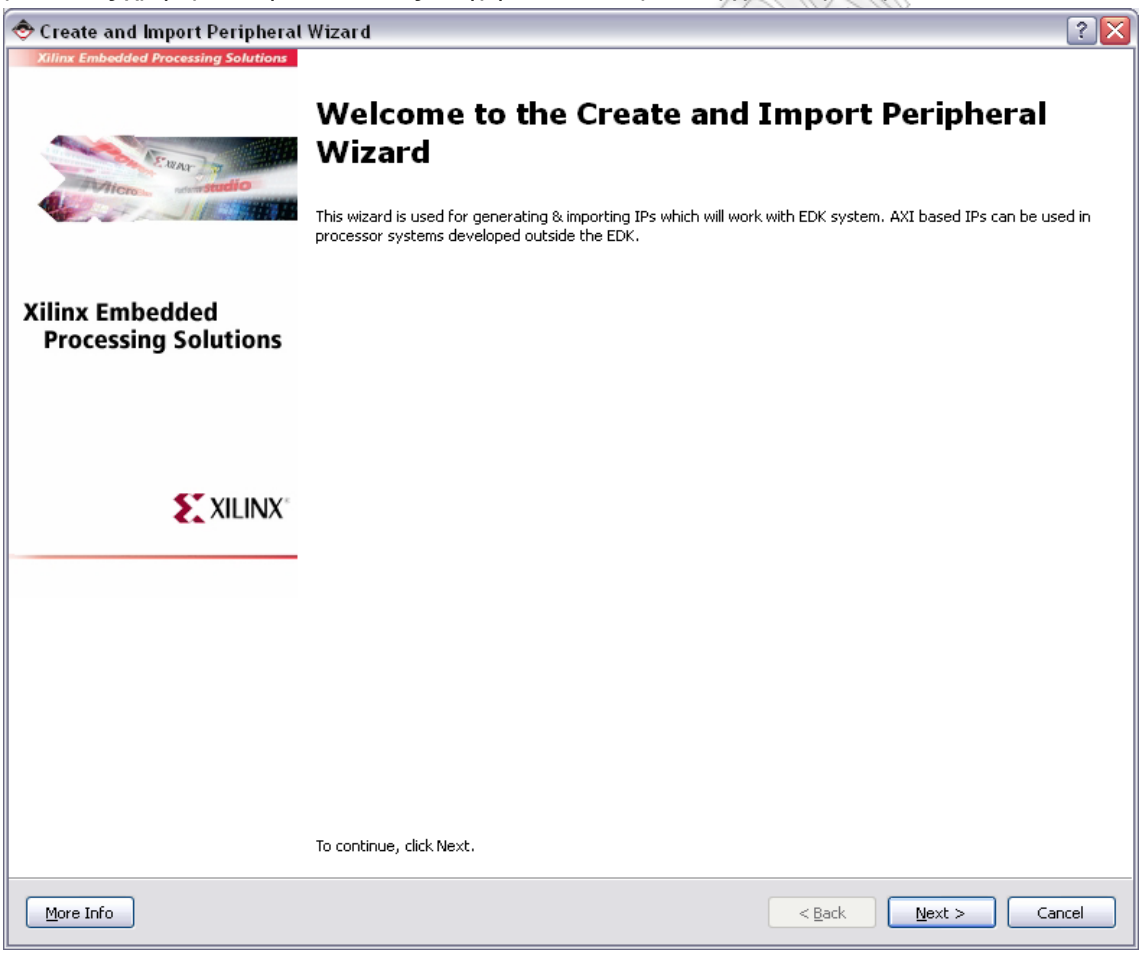
Βασισμένοι, λοιπόν, σε αυτή την υλοποίηση, στο πλαίσιο της παρούσας διατριβής εστιάσαμε στο να μεταφέρουμε το αρχικό σύστημα αποκλειστικά και μόνο σε μία ενσωματωμένη πλατφόρμα. Συνεπώς, η εργασία αναπτύχθηκε, αρχικά, εξαλείφοντας τα στοιχεία της PCIe από το περιφερειακό και την μεταφορά της λογικής του σε ένα νέο περιφερειακό που υλοποιεί μόνο τον επιταχυντή και είναι σε άμεση επικοινωνία με τον ενσωματωμένο επεξεργαστή, μέσω ειδικών σημάτων, καταχωρητών και μνημών. Επιπλέον, χρειάστηκε να μετατραπεί ο αλγόριθμος, ώστε να μπορεί να εκτελεστεί στον ενσωματωμένο επεξεργαστή και να επικοινωνήσει σωστά με το νέο επιταχυντή, ενώ τελευταίο, και ίσως πιο σημαντικό, χρειάστηκε ένα υλοποιηθεί ένα ενσωματωμένο σύστημα, το οποίο θα περιλαμβάνει τόσο τον επεξεργαστή όσο και τον επιταχυντή, και θα είναι ικανό να τηρήσει τις απαιτήσεις του χρήστη και να εκτελέσει την εφαρμογή στο σύνολό της (λογισμικό και υλικό).

Τέλος, διαπιστώνουμε ότι συνδυάζοντας διαφορετικά γνωστικά αντικείμενα από το χώρο της βιολογίας και της πληροφορικής, μπορούμε να υλοποιήσουμε γρήγορες και αποδοτικές εφαρμογές πολύπλοκων προβλημάτων, αποφεύγοντας παραδοσιακές λύσεις επίλυσης στο λογισμικό, κερδίζοντας μια σημαντική επιτάχυνση στο χρόνο εκτέλεσης, ενώ παράλληλα υλοποιώντας το σύνολο της εφαρμογής του αλγορίθμου σε μία ενσωματωμένη πλατφόρμα εξασφαλίζουμε τη φορητότητα της εφαρμογής. Συγκεκριμένα, στο πεδίο έρευνας μας, μία φορητή τεχνολογία που θα εντοπίζει ηχητικά εφέ στο μήκος μίας ταινίας, γρήγορα και αποτελεσματικά, μπορεί κάλλιστα να χρησιμοποιηθεί από νέας γενιάς media players, οι οποίοι θα έχουν την δυνατότητα να παράγουν αυτόματα υπότιτλους για την περιγραφή των ηχητικών εφέ (ιδιαίτερα χρήσιμο για ανθρώπους με προβλήματα ακοής). Ταυτόχρονα, η ενσωματωμένη εφαρμογή θα μπορούσε να συνδυαστεί με αισθητήρες (sensors) και συστήματα παρακολούθησης και ασφάλειας (surveillance and security systems), όπου διάφορα ανεπιθυμήτα ηχητικά γεγονότα (όπως είναι το σπάσιμο γυαλιών, κραυγές, πυροβολισμοί) θα αναγνωρίζονται κατά μήκος του αρχείου παρακολούθησης, και αυτόματα θα ειδοποιείται ο χρήστης ή κάποια ομάδα ασφάλειας.

Παράρτημα Α: Δημιουργία ενός νέου περιφερειακού με το Create and Import Peripheral της σουίτας ISE της Xilinx και ενσωμάτωση του στο σύστημα

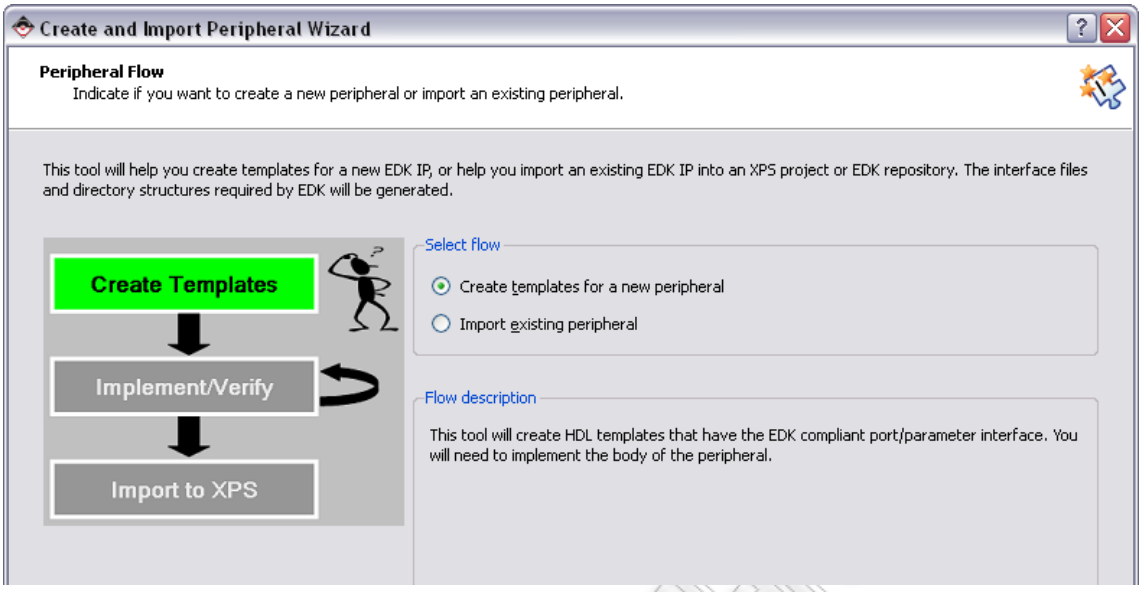
Στο πλαίσιο της παρούσας εργασίας, χρειάστηκε να αναπτύξουμε έναν δικό μας IP Core περιφερειακό, που θα αποτελούσε τον επιταχυντή υπολογισμών του αλγορίθμου Smith-Waterman. Το περιφερειακό αυτό, όπως αναφέραμε και σε προηγούμενα κεφάλαια, προστίθεται στο ενσωματωμένο σύστημα σαν ένα επιπλέον περιφερειακό και επικοινωνεί με τον ενσωματωμένο επεξεργαστή μέσω διαύλου επικοινωνίας. Στο σημείο αυτό, θεωρήσαμε σημαντικό να παραθέσουμε τις λεπτομέρειες ανάπτυξης του περιφερειακού αυτού, χρησιμοποιώντας το εργαλείο Create and Import Peripheral της Xilinx.

Το εργαλείο Create and Import Peripheral Wizard μπορεί να εκκινήσει είτε επιλέγοντάς το από τη σουίτα προγραμμάτων της Xilinx > EDK > Tools, είτε από το πρόγραμμα του XPS (Xilinx Platform Studio) > Hardware > Create and Import Peripheral Wizard. Όποια από τις δύο μεθόδους χρησιμοποιήσετε, θα σας οδηγήσει στο επόμενο αρχικό παράθυρο.

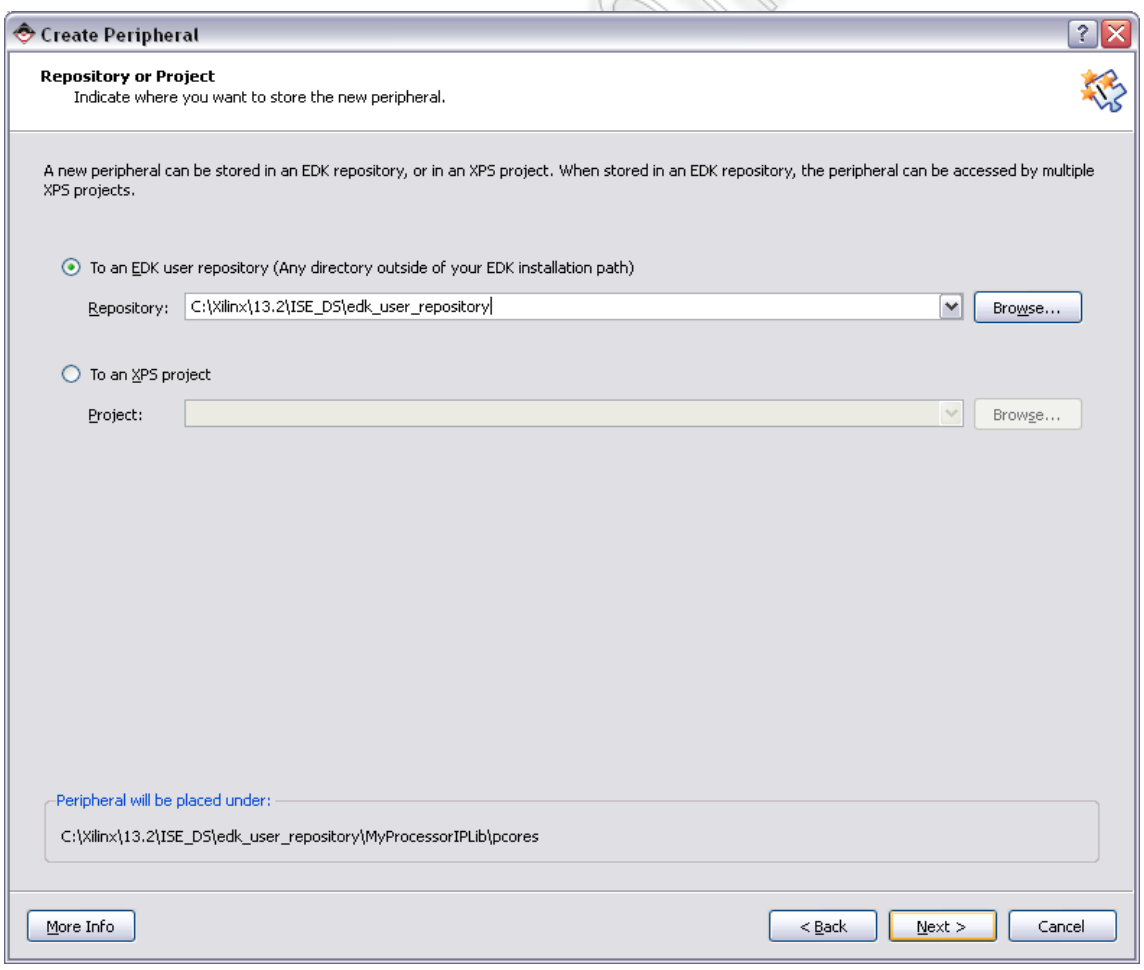


Παράρτημα Α - 1: Αρχικό παράθυρο του Create and Import Peripheral Wizard.

Στα επόμενα παράθυρα του εργαλείου, ζητείται από το χρήστη να καθορίσει αν επιθυμεί να δημιουργήσει ένα νέο περιφερειακό από την αρχή, ή αν επιθυμεί να εισάγει ένα περιφερειακό, που έχει πρότερα δημιουργήσει, σε ένα υπάρχον ενσωματωμένο σύστημα. Στη δική μας περίπτωση, επιλέγουμε το πρώτο, οπότε μεταφερόμαστε στην οθόνη (Παράρτημα Α-2).

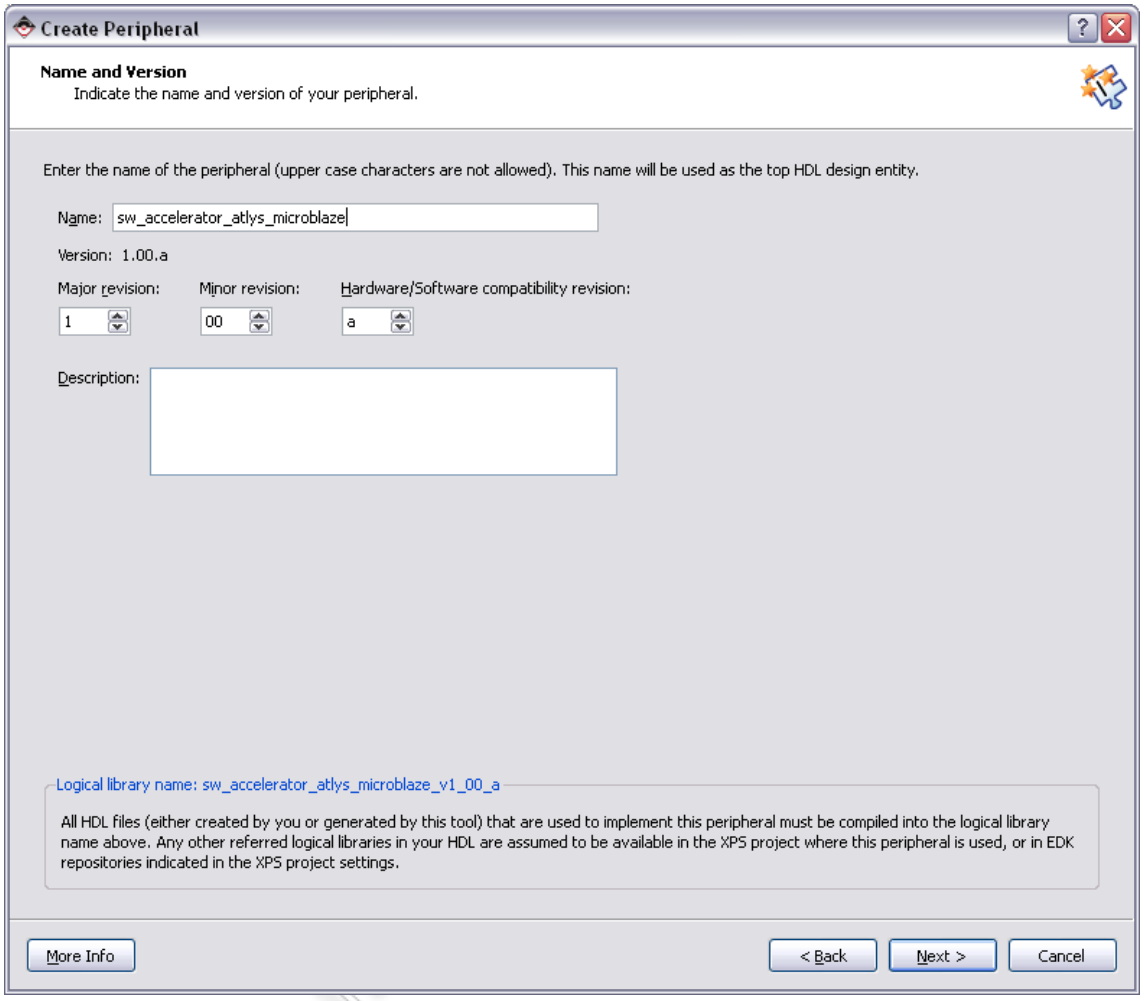


Παράρτημα Α - 2: Οθόνη επιλογής δημιουργίας νέου ή εισαγωγής περιφερειακού στο Create and Import Peripheral.



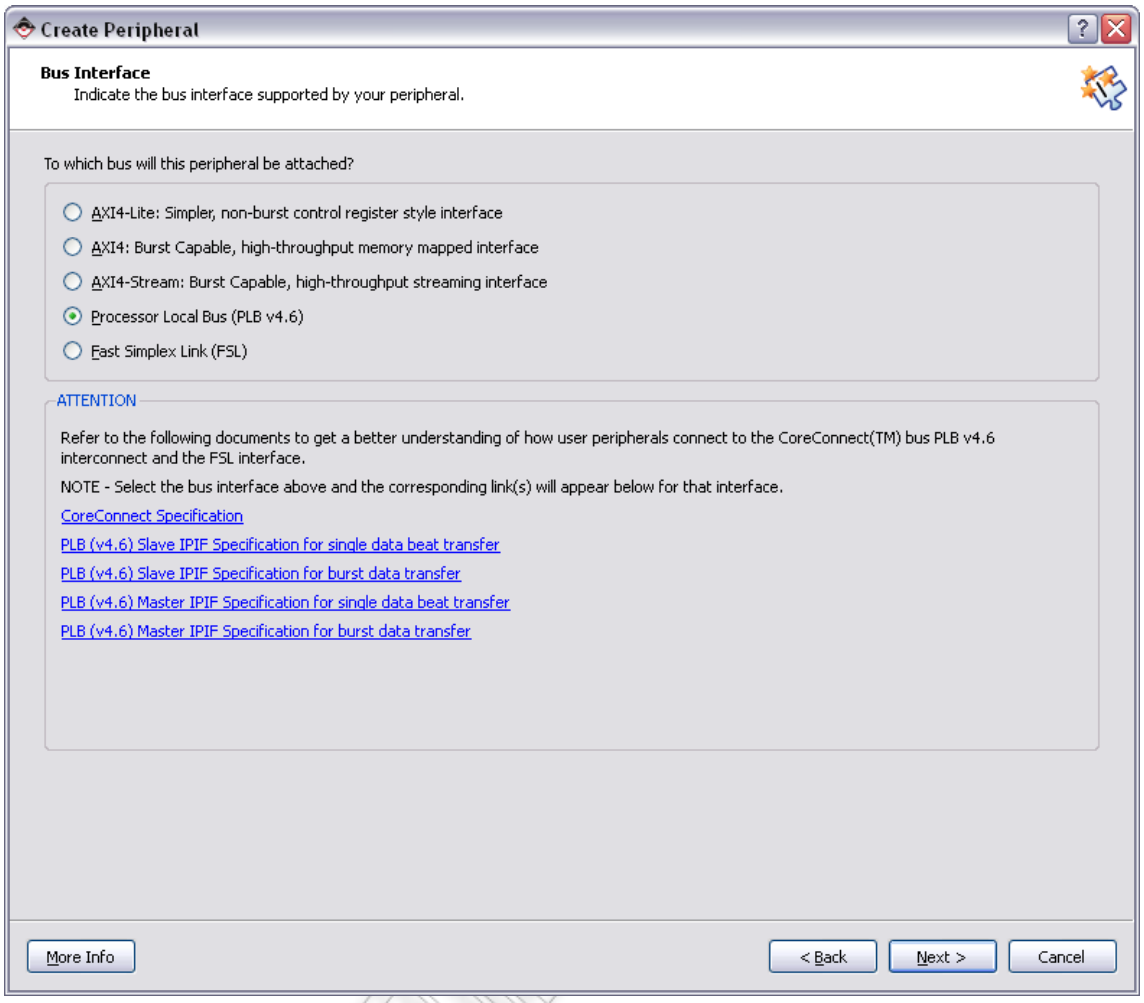
Παράρτημα Α - 3: Οθόνη επιλογής αποθήκευσης περιφερειακού στο Create and Import Peripheral.

Από εκεί, έχουμε τη δυνατότητα να επιλέξουμε που θα αποθηκευτεί το περιφερειακό που πρόκειται να κατασκευάσουμε. Έχουμε τη δυνατότητα να το αποθηκεύσουμε σε ένα repository, από όπου θα μπορεί να χρησιμοποιηθεί από όλα τα projects που δημιουργηθούν στο περιβάλλον του XPS, ή να αποθηκευθεί τοπικά σε ένα υπάρχον project (στον τοπικό κατάλογο pcores του project) και να είναι προσβάσιμο μόνο από αυτό (Παράρτημα Α-3). Επιπρόσθετα, συνεχίζοντας πρέπει να επιλέξουμε το όνομα και την έκδοση του περιφερειακού (Παράρτημα Α-4).



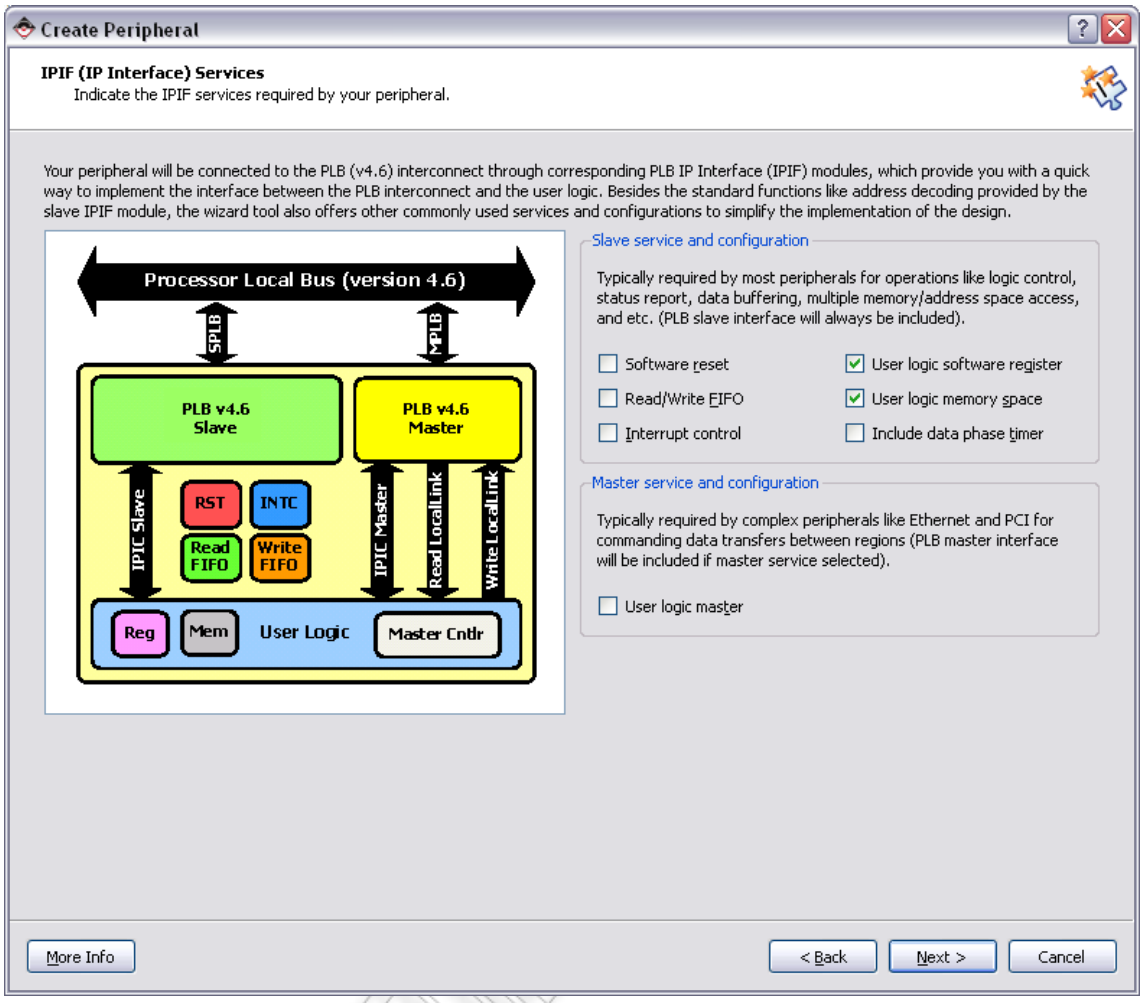
Παράρτημα Α - 4: Ονομασία νέου περιφερειακού.

Έπειτα, πρέπει να δηλώσουμε μέσω ποιου διαύλου θα συνδεθεί το νέο περιφερειακό στον ενσωματωμένο επεξεργαστή. Στην περίπτωσή μας (Παράρτημα Α-5), επιλέγουμε τον δίαυλο PLB (Processor Local Bus), τον οποίο έχουμε ήδη εισάγει στο ενσωματωμένο σύστημα, και είναι ο δίαυλος, στο οποίον 'κρέμονται' τα υπόλοιπα κύρια συστατικά του ενσωματωμένου συστήματος.



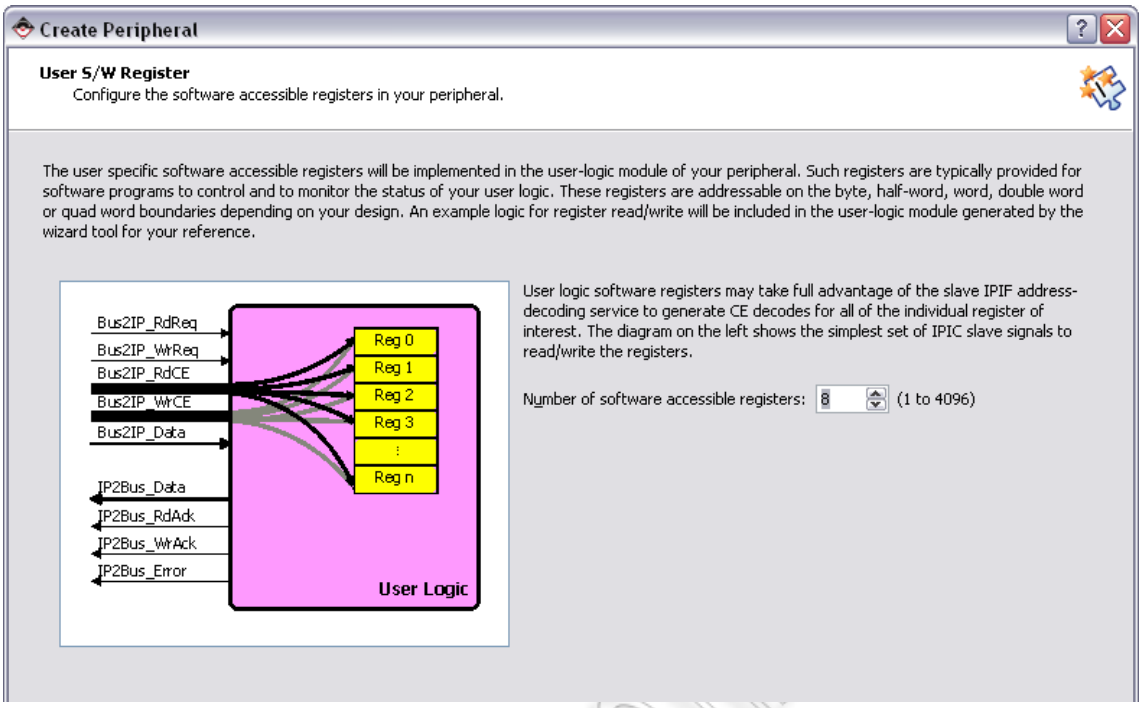
Παράρτημα Α - 5: Επιλογή διαύλου διασύνδεσης περιφερειακού με το υπόλοιπο σύστημα.

Συγκεκριμένα, η αρχιτεκτονική του Microblaze επεξεργαστή διαθέτει ένα δίαυλο, ο οποίος συμβολίζεται ως PLB (Processor Local Bus). Κάθε IP Core που συνδέεται με τον PLB έχει δύο στρώματα, εξίσου αναγκαία για να παρέχουν τη διεπαφή από το user logic (ο VHDL κώδικας που καθιστά λειτουργικό το core) προς τα σήματα του PLB, τα οποία είναι το IPIF (Διεπαφή IP προς τη διεπαφή του PLB) και το IPIC (Η IP ενδοδιασύνδεση που παρέχει διεπαφή μεταξύ λογικής χρήστη και IPIF). Η βασική λογική με τα cores, και δε με τα περιφερειακά κατασκευασμένα από το χρήστη, είναι ότι το μόνο με το οποίο χρειάζεται, κανείς, να ασχοληθεί είναι το user logic, συνήθως το αρχείο *user_logic.vhd* που τοποθετείται στον φάκελο του core.

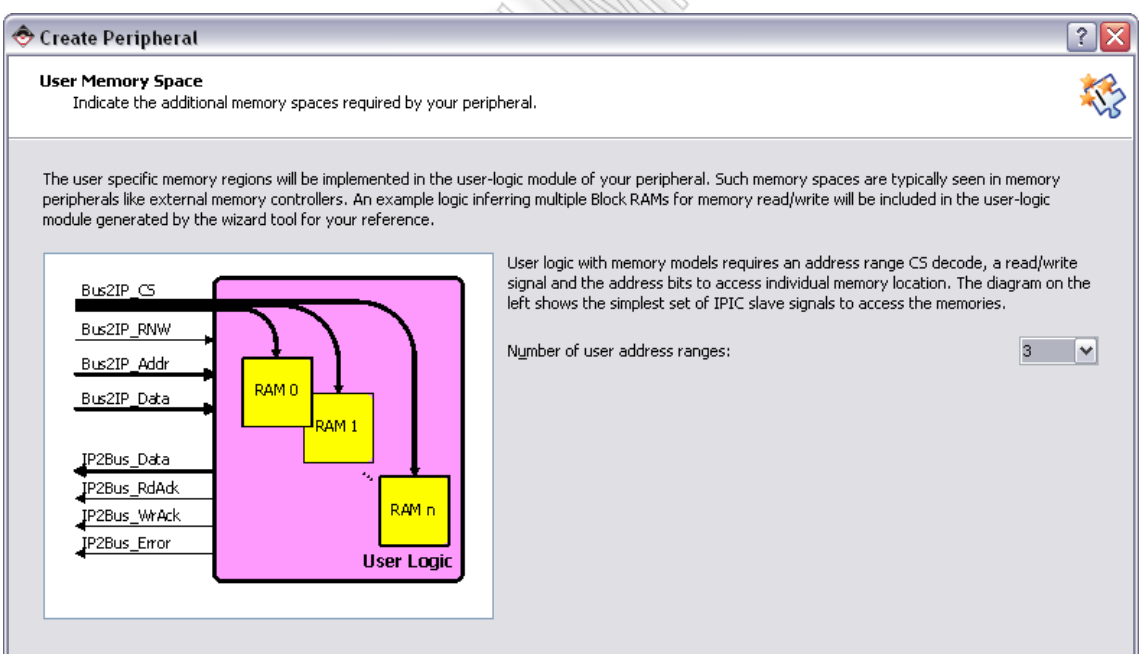


Παράρτημα Α - 6: Δήλωση λειτουργιών IPIF (IP διεπαφής) του user logic του περιφερειακού.

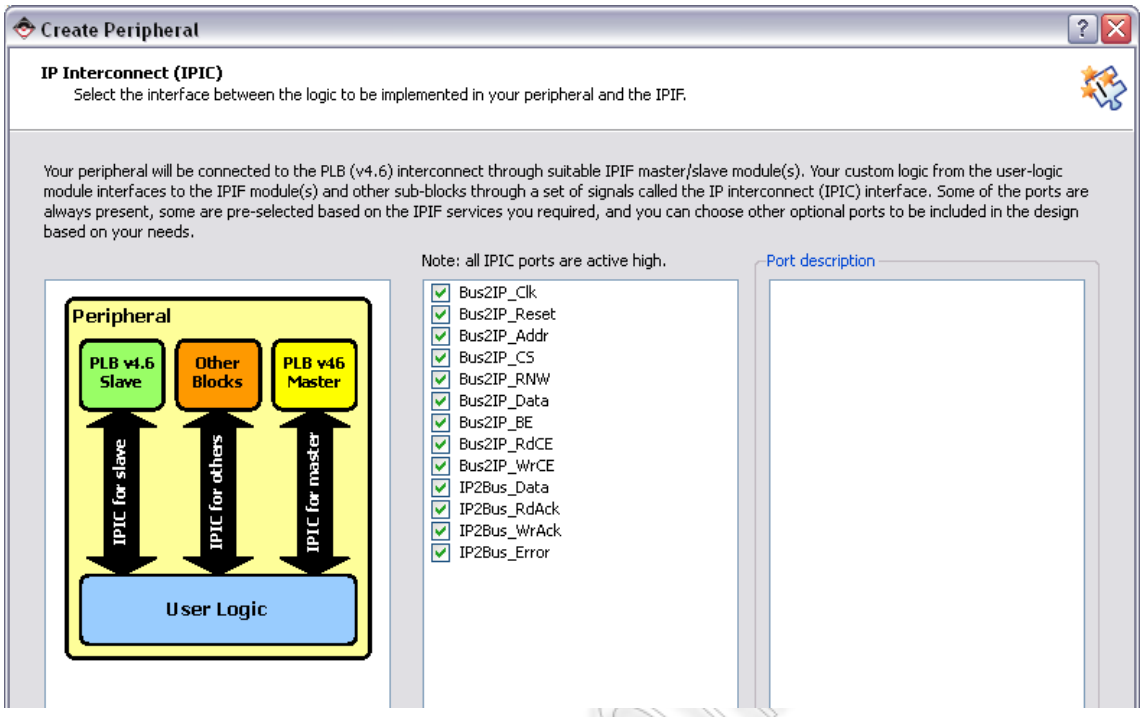
Επιπρόσθετα, η διεπαφή IPIF του PLB IP παρέχει στο χρήστη ορισμένες επιπλέον λειτουργίες, όπως είναι η επιλογή επαναφοράς του περιφερειακού στην αρχική του κατάσταση (reset), την επιβολή διακοπών στη λειτουργία του, τη χρήση καταχωρητών ή μνήμης προσβάσιμων από το λογισμικό κ.ά. (Παράρτημα Α-6). Για το περιφερειακό του επιταχυντή που εμείς κατασκευάζουμε, δηλώνουμε ότι χρειαζόμαστε την λειτουργία των καταχωρητών λογισμικού, καθώς και χώρου μνήμης για το user logic του περιφερειακού. Στις επόμενες οθόνες του wizard, δίνοντας το πλήθος των καταχωρητών (Παράρτημα Α-7) καθώς και των μνημών (Παράρτημα Α-8) που θα είναι προσπελάσιμοι από το λογισμικό, το wizard αναλαμβάνει να ενημερώσει τα αρχεία που συνθέτουν το core, με τα απαραίτητα σήματα για το διάβασμα και γράψιμο αυτών των στοιχείων μέσω του διαύλου PLB.



Παράρτημα Α - 7:Δήλωση του πλήθους των καταχωρητών του user logic, προσβάσιμων από το λογισμικό.

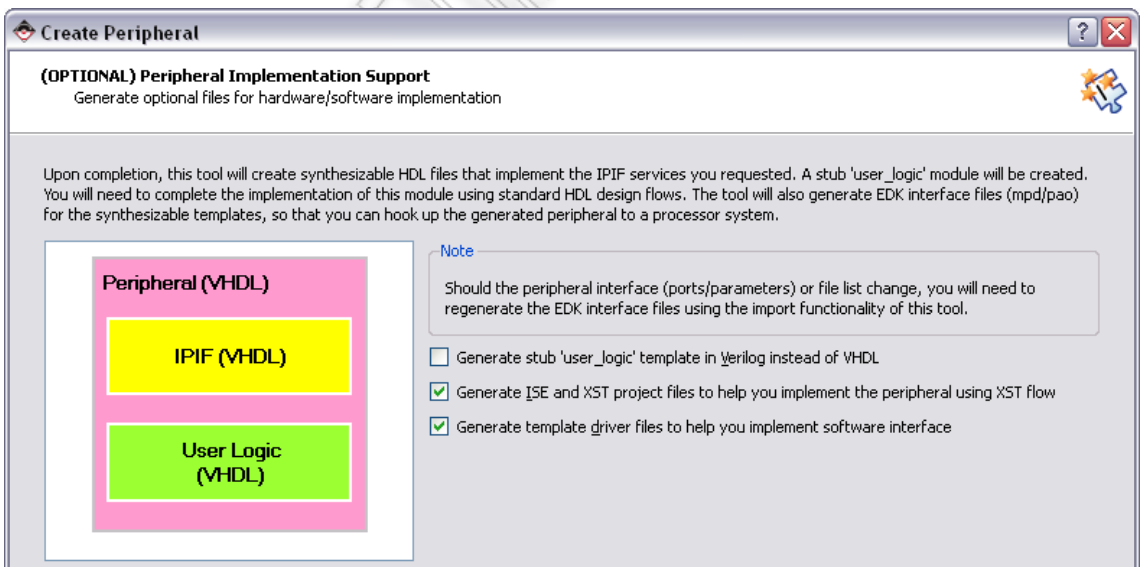


Παράρτημα Α - 8:Δήλωση του πλήθους των περιοχών μνήμης του user logic, προσβάσιμων από το λογισμικό.



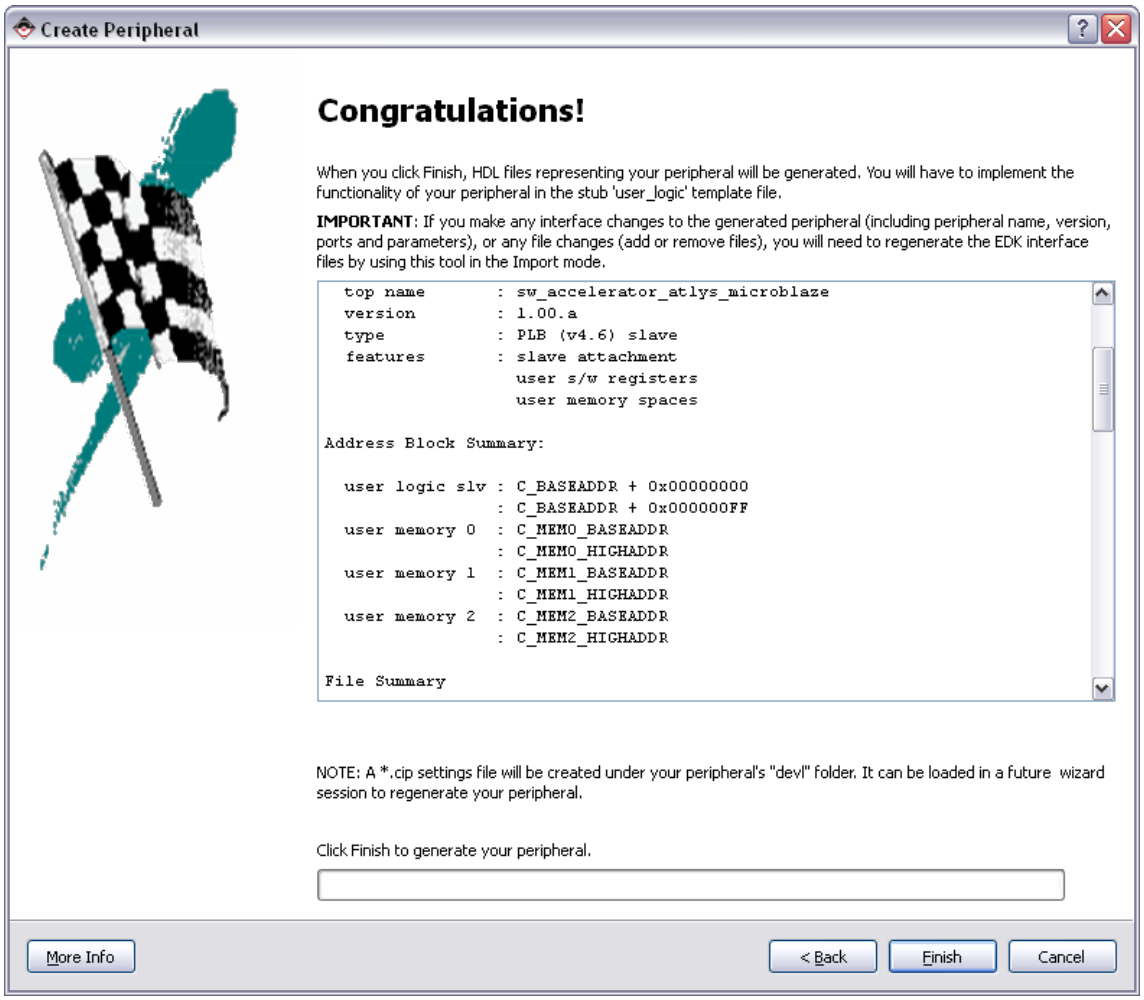
Παράρτημα Α - 9: Δήλωση σημάτων IPIC (IP ενδοδιασύνδεσης).

Έπειτα, δίνονται τα σήματα για τη διεπαφή της ενδοδιασύνδεσης (IPIC) μεταξύ του user logic και του IPIF (Παράρτημα Α-9). Τα σήματα, που ήδη έχουν κριθεί απαραίτητα να ενσωματωθούν στο σύστημα, από τις λειτουργίες που έχει δηλώσει ο σχεδιαστής ότι θέλει να υποστηρίζει το περιφερειακό, είναι ήδη επιλεγμένα στη λίστα, ενώ παράλληλα μπορούν να επιλεγούν και άλλα σύμφωνα με τις ανάγκες του χρήστη.



Παράρτημα Α - 10: Αυτόματη δημιουργία αρχείων για το εργαλείο ISE και γέννηση οδηγών για χειρισμό από το λογισμικό.

Τέλος, υπάρχει η δυνατότητα να δημιουργήσει το εργαλείο κατάλληλα αρχεία για την υλοποίηση του περιφερειακού από το εργαλείο ISE της Xilinx, σύμφωνα με την ροή υλοποίησης XST. Παράλληλα, μπορεί να δημιουργήσει αυτόματα οδηγούς (drivers) και βιβλιοθήκες (header files) για το χειρισμό του περιφερειακού από λογισμικό, γραμμένους σε γλώσσα C. Το αρχείο ISE, συγκεκριμένα, θα μας είναι ιδιαίτερα χρήσιμο για την ενημέρωση του user logic με την λογική που θέλουμε να εκτελεί το περιφερειακό, ώστε να επιτελεί τις ζητούμενες λειτουργίες. Όπως μπορεί κανείς να παρατηρήσει στα αποτελέσματα υλοποίησης από το εργαλείο, παρουσιάζεται η δομή και οι βασικές διευθύνσεις του χώρου μνήμης και των καταχωρητών του περιφερειακού, σε μορφή παραμέτρων.



Παράρτημα Α - 11: Ολοκλήρωση δημιουργίας νέου περιφερειακού.

Μετά την ολοκλήρωση της δημιουργίας του περιφερειακού από το εργαλείο, έχουν δημιουργηθεί όλα τα απαραίτητα αρχεία:

Αρχεία υλοποίησης σε VHDL κώδικα

- Αρχείο με το όνομα της top level οντότητας της σχεδίασης (για εμάς ίδιο με το όνομα του περιφερειακού), που περιέχει όλες τις απαραίτητες συνδέσεις με τα σήματα του PLB, ενώ ταυτόχρονα εμπεριέχει τη σύνδεση με την οντότητα του user logic.

- Αρχείο με το όνομα user_logic, που υλοποιεί τη λογική του περιφερειακού. Το αρχείο αυτό εμπλουτίζεται με επιπλέον θύρες και λειτουργίες και υλοποιείται μέσω της ροής του προγράμματος XPS, ώστε να υποστηρίζει τις επιθυμητές λειτουργίες του περιφερειακού.

```
- HDL source -

C:\Xilinx\13.2\ISE_DS\edk_user_repository\MyProcessorIPLib/pcores/sw_a
ccelerator_atlys_microblaze_v1_00_a/hdl
        top entity      : vhdl/sw_accelerator_atlys_microblaze.vhd
        user logic      : vhdl/user_logic.vhd
```

Αρχεία περιγραφής της διεπαφής του περιφερειακού για το πρόγραμμα XPS

- Το αρχείο MPD (Microprocessor Peripheral Definition) περιέχει τη διεπαφή του περιφερειακού, αναφέροντας λεπτομερώς τη λίστα των θυρών που χρησιμοποιούνται, τη προκαθορισμένη συνδεσιμότητα των διασυνδέσεων των διαύλων, καθώς και τις διεπαφές εισόδων/εξόδων (I/O) και τυχόν παραμέτρους, αν αυτές ορίζονται.
- Το αρχείο PAO (Peripheral Analyze Order) περιέχει τη λίστα των αρχείων Hardware Description Language (HDL), που χρειάζονται για τη σύνθεση του περιφερειακού και ορίζεται η σειρά ανάλυσης των αρχείων για την υλοποίησή του.

```
- XPS interface -

C:\Xilinx\13.2\ISE_DS\edk_user_repository\MyProcessorIPLib/pcores/sw_a
ccelerator_atlys_microblaze_v1_00_a/data
        mpd             : sw_accelerator_atlys_microblaze_v2_1_0.mpd
        pao             : sw_accelerator_atlys_microblaze_v2_1_0.pao
```

Αρχεία υλοποίησης μέσω του προγράμματος ISE και περιγραφή σύνθεσης του περιφερειακού για τη ροής υλοποίησης XST

- Το αρχείο ISE ανοίγει το πρόγραμμα ISE για την περαιτέρω υλοποίηση του περιφερειακού. Είναι σε άμεση σύνδεση με τα αρχεία περιγραφής σε κώδικα VHDL (top level, user logic), τα οποία μπορούν να τροποποιηθούν ή και να προστεθούν και άλλα, μέσα από το συγκεκριμένο περιβάλλον.
- Στο αρχείο TCL περιέχεται τη δέσμη ενεργειών (script) που πρέπει να εκτελεστεί για την δημιουργία του project στο περιβάλλον του ISE. Περιλαμβάνει κυρίως εντολές σχετικές με τα αρχεία από τα οποία αποτελείται το περιφερειακό, τις βιβλιοθήκες από τις οποίες εξαρτάται (τις οποίες και δημιουργεί) καθώς και πληροφορίες για την αναπτυξιακή πλακέτα στην οποία σκοπεύει να 'υλοποιηθεί το υλικό.
- Το αρχείο XST script δίνει τις απαραίτητες επιλογές (flags) στην διαδικασία σύνθεσης του XST, ώστε να προσαρμόσει το περιφερειακό στη δική μας σχεδίαση (τύπος πλακέτας, επιλεγμένος βαθμός βελτιστοποίησης, προσθήκη buffers σε I/O κ.ά.).
- Στο αρχείο XST project, αναφέρεται κάθε αρχείο που πρέπει να συμπεριληφθεί στη XST σύνθεση του περιφερειακού, τη βιβλιοθήκη στην οποία ανήκει και το μονοπάτι στο οποίο βρίσκεται στο δίσκο σε σχέση με το τρέχον αρχείο σύνθεσης.

```
 - ISE project -

C:\Xilinx\13.2\ISE_DS\edk_user_repository\MyProcessorIPLib/pcores/sw_a
ccelerator_atlys_microblaze_v1_00_a/devl/projnav
        ise project    : sw_accelerator_atlys_microblaze.xise
        tcl script     : sw_accelerator_atlys_microblaze.tcl

- XST synthesis -

C:\Xilinx\13.2\ISE_DS\edk_user_repository\MyProcessorIPLib/pcores/sw_a
ccelerator_atlys_microblaze_v1_00_a/devl/synthesis
        xst script     : sw_accelerator_atlys_microblaze_xst.scr
        xst project    : sw_accelerator_atlys_microblaze_xst.prj
```

Αρχεία οδηγού συσκευής (driver) του περιφερειακού

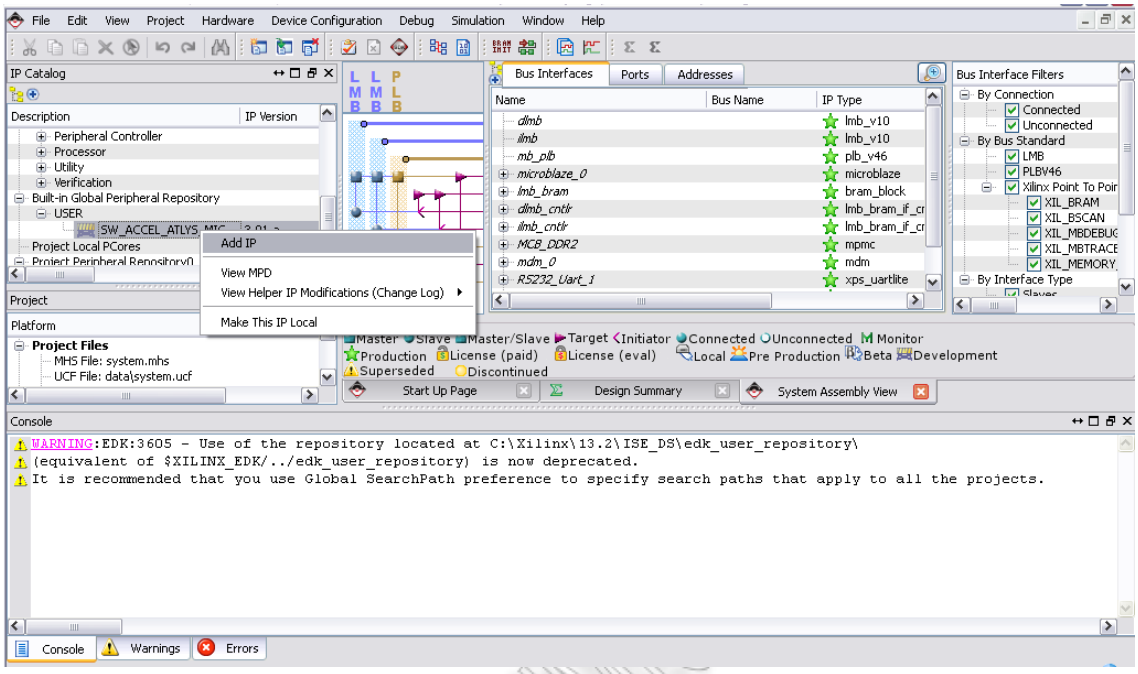
- Το αρχείο header τη δήλωση των συναρτήσεων του περιφερειακού και ενσωματώνεται (#include) στο λογισμικό, αν θέλουμε να εκτελέσουμε οποιεσδήποτε διαδικασίες εγγραφής και ανάγνωσης στο περιφερειακό. Συγκεκριμένα, για τη δική μας περίπτωση, εδώ ορίζονται οι συναρτήσεις με τις οποίες μπορούμε να γράψουμε και να διαβάσουμε από τους καταχωρητές και τη μνήμη του περιφερειακού μέσω του λογισμικού που θα τρέχει στον ενσωματωμένο επεξεργαστή.
- Το αρχείο source περιέχει την υλοποίηση των συναρτήσεων που αυτόματα γεννάει το εργαλείο για την ανάγνωση και εγγραφή των καταχωρητών/μνημών του περιφερειακού.
- Το selftest αρχείο δημιουργείται αυτόματα για τον διαγνωστικό έλεγχο ορθότητας των συναρτήσεων που ορίζονται στα παραπάνω αρχεία.

```
- Driver source -

C:\Xilinx\13.2\ISE_DS\edk_user_repository\MyProcessorIPLib/drivers/sw_
accelerator_atlys_microblaze_v1_00_a/src
        makefile       : Makefile
        header         : sw_accelerator_atlys_microblaze.h
        source         : sw_accelerator_atlys_microblaze.c
        selftest       : sw_accelerator_atlys_microblaze_selftest.c
```

Μετά την ολοκλήρωση της διαδικασίας δημιουργίας του περιφερειακού από το Create and Import Peripheral Wizard, μπορούμε να προχωρήσουμε στην ανάπτυξη του μέσα από τη ροή εργασιών του XST από το πρόγραμμα ISE. Όταν, πλέον, αυτό έχει περάσει επιτυχώς τη διαδικασία της σύνθεσης και υλοποίησης από το ISE, είναι έτοιμο να εισαχθεί σε ένα ενσωματωμένο σύστημα, χρησιμοποιώντας ξανά το Create and Import Peripheral Wizard, αυτή τη φορά για εισαγωγή. Επιλέγοντας το ίδιο όνομα και την ίδια έκδοση, θα μας εμφανιστεί ένα μήνυμα ότι το περιφερειακό υπάρχει ήδη, οπότε επιλέγουμε να συνεχίσουμε. Αφού έχει ολοκληρωθεί το wizard, θα πρέπει να ενσωματωθεί το core στο ενσωματωμένο σύστημα, χρησιμοποιώντας τη λίστα του IP Catalog στο πρόγραμμα XPS. Αν αυτό έχει αποθηκευτεί τοπικά στο συγκεκριμένο project (βρίσκεται στον κατάλογο pcores του project αυτού), το περιφερειακό φαίνεται στη λίστα των Project Local PCores, ενώ αν έχει αποθηκευτεί σε ένα

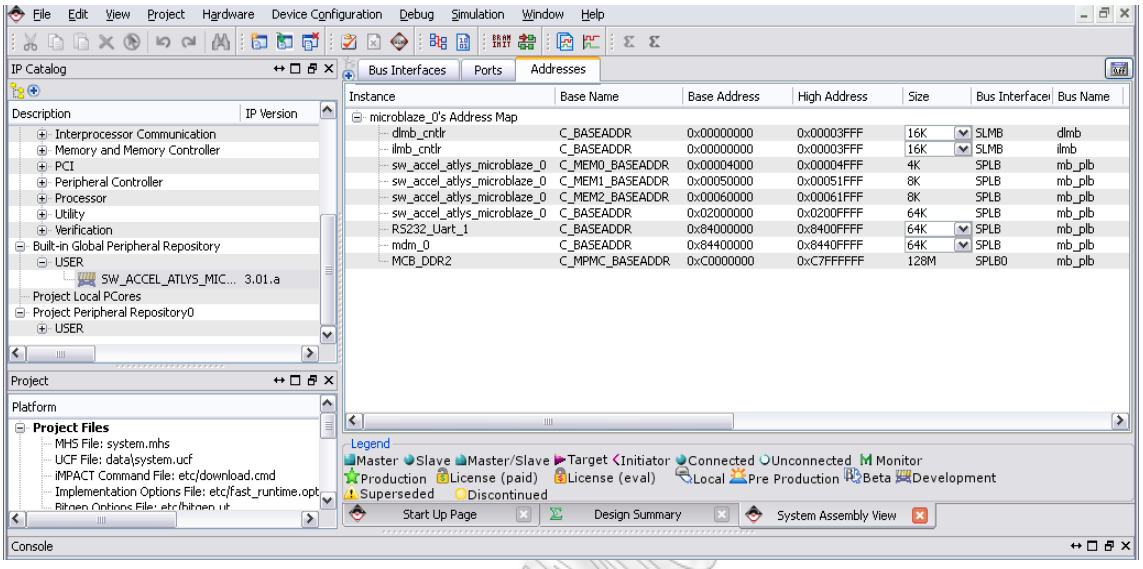
repository προσβάσιμο από όλα τα projects, θα εμφανίζεται στη λίστα των Build-In Global Peripheral Repository>USER. Σε κάθε περίπτωση, επιλέγοντας πάνω στο περιφερειακό και πατώντας την επιλογή ‘Add IP’ μπορούμε να το εισάγουμε στο ενσωματωμένο σύστημα (Παράρτημα Α-12).



Παράρτημα Α - 12: Εισαγωγή περιφερειακού στο ενσωματωμένο σύστημα.

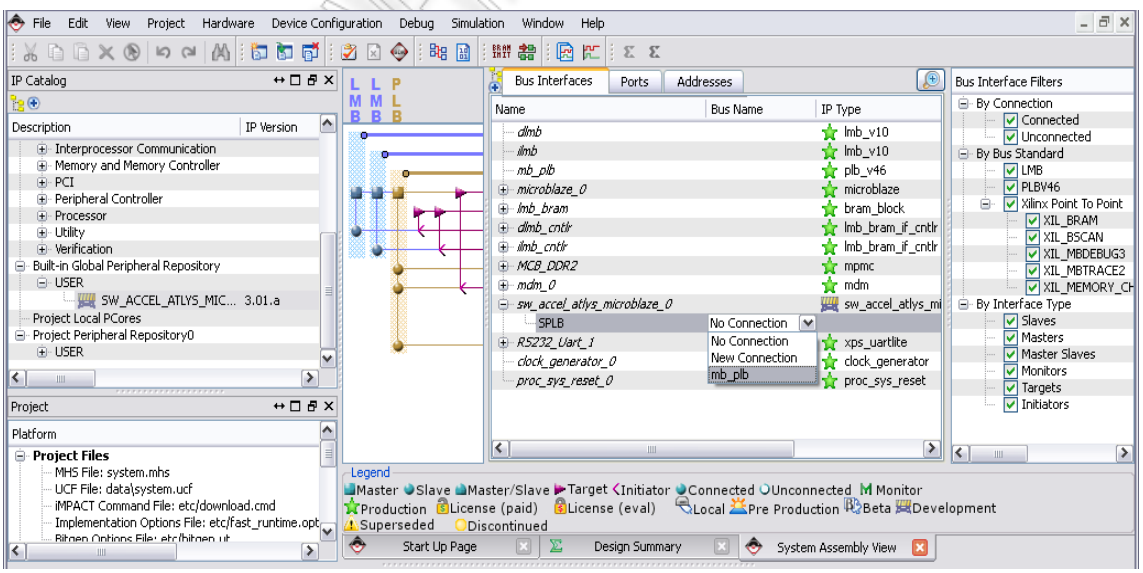
Παράρτημα Α - 13: Ενημέρωση των διευθύνσεων βάσεων περιφερειακού και μνήμης για την εισαγωγή του στο σύστημα.

Στη συνέχεια, εμφανίζεται ένα παράθυρο, όπου καλούμαστε να δηλώσουμε τις τιμές των παραμέτρων του περιφερειακού, στην δική μας περίπτωση αυτές είναι οι διευθύνσεις βάσης του περιφερειακού, καθώς και των τριών μνημών που χρησιμοποιεί (Παράρτημα Α-13). Μόλις το περιφερειακό ενημερωθεί με τις διευθύνσεις αυτές, ενημερώνεται πλήρως και το κομμάτι των διευθύνσεων (Addresses View) του ενσωματωμένο συστήματος, όπως απεικονίζεται και στο ακόλουθο παράθυρο.



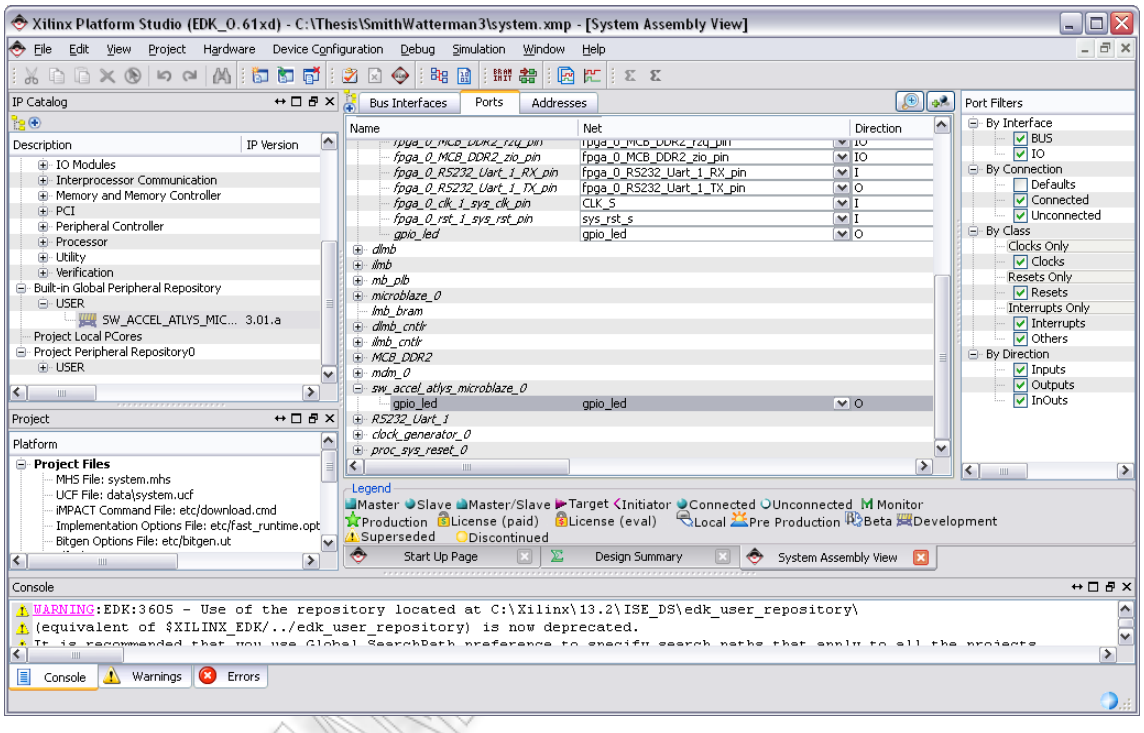
Παράρτημα Α - 14: Ενημέρωση διευθύνσεων ενσωματωμένου συστήματος.

Μετά από αυτό, θα πρέπει να επισημάνουμε τη σύνδεση του περιφερειακού στον δίαυλο PLB. Αυτό γίνεται όντας στο παράθυρο system assembly του XPS (bus interfaces view), επιλέξουμε το περιφερειακό και εισάγουμε τη σύνδεση του στον mb_plb (Microblaze PLB).



Παράρτημα Α - 15: Σύνδεση περιφερειακού στον δίαυλο PLB του Microblaze.

Στη συνέχεια, πηγαίνουμε στην άποψη των θυρών (Ports view) του συστήματος, ελέγχουμε αν οι συνδέσεις των θυρών έχουν γίνει σωστά και κάνουμε τις θύρες του περιφερειακού εξωτερικές. Αυτό σημαίνει ότι πρέπει να ενημερώσουμε το αρχείο UCF με τα pins στα οποία αναθέτουμε αυτές τις εξωτερικές θύρες. Γενικότερα, όμως, αν δεν υπάρχουν εξωτερικές θύρες για το περιφερειακό, τα αρχεία UCF, MHS και MSS, που δημιουργούνται αυτόματα από το σύστημα, δεν χρειάζονται καμία μετατροπή ή προσθήκη. Έτσι, το περιφερειακό έχει ενταχθεί πλήρως στο ενσωματωμένο σύστημα και είναι έτοιμο να φορτωθεί στην αναπτυξιακή πλακέτα, μόλις η διαδικασία παραγωγής του αρχείου διαμόρφωσης .bit ολοκληρωθεί επιτυχώς. Τέλος, χρησιμοποιώντας τον οδηγό του περιφερειακού είμαστε σε θέση να γράψουμε κώδικα που χειρίζεται το περιφερειακό από το λογισμικό που εκτελείται στον ενσωματωμένο επεξεργαστή του συστήματος.



Παράρτημα Α - 16: Σύνδεση θυρών του περιφερειακού.

Παράρτημα Β: Δημιουργία ενός Memory File System και αποστολή του στην FPGA συσκευή

Όπως αναφέραμε και κατά τη διάρκεια του κυρίου σώματος της εργασίας, η εφαρμογή που εκτελείται στον Microblaze ενσωματωμένο επεξεργαστή, είναι υπεύθυνη για την προ-επεξεργασία των αρχείων της ταινίας και του ηχητικού εφέ. Αυτό σημαίνει, εκ των πραγμάτων πως η εφαρμογή θα πρέπει να δέχεται σαν είσοδο τα δύο επεξεργασμένα αρχεία ήχου, από τα οποία πρέπει να εξάγει τα χαρακτηριστικά του ήχου και να αποθηκεύσει αυτά στην μνήμη του επιταχυντή. Για να γίνει αυτό, θα πρέπει τα αρχεία αυτά να είναι ήδη διαθέσιμα και αποθηκευμένα στην αναπτυξιακή πλακέτα, πριν αυτή διαμορφωθεί με το αρχείο προγραμματισμού .bit. Ως εκ τούτου, έπρεπε να δημιουργηθεί ένα τύπου σύστημα αρχείων μνήμης (memory file system), που θα περιείχε τα εν λόγω αρχεία, και να φορτωθεί σε μια διεύθυνση μνήμης, από την οποία η εφαρμογή θα μπορεί να επεξεργαστεί τα αρχεία αυτά. Όλα τα παραπάνω πραγματοποιήθηκαν με τη χρήση της ειδικής βιβλιοθήκης της Xilinx LibXil Memory File System (MFS) για τη δημιουργία ενός τέτοιου συστήματος αρχείων μνήμης.

Η βιβλιοθήκη **LibXil** παρέχει τη δυνατότητα διαχείρισης της μνήμης του προγράμματος με τη μορφή χειριστών αρχείων. Με τη χρήση της βιβλιοθήκης αυτής, μπορεί κανείς να δημιουργήσει καταλόγους και αρχεία μέσα σε κάθε κατάλογο. Το σύστημα αρχείων μπορεί να είναι προσβάσιμο από τη γλώσσα υψηλού επιπέδου C μέσω κλήσεων συναρτήσεων, ειδικών για το σύστημα αρχείων, οι οποίες θα παρουσιαστούν στην συνέχεια (Παράρτημα Β.2).

Παράρτημα Β.1. Δημιουργία συστήματος αρχείων μνήμης χρησιμοποιώντας το utility mfsgen

Μαζί με τη MFS βιβλιοθήκη παρέχεται και ένα πρόγραμμα επονομαζόμενο mfsgen, το οποίο μπορεί να χρησιμοποιηθεί για να δημιουργήσει μία MFS εικόνα μνήμης (memory image) σε ένα Η/Υ, η οποία μπορεί στη συνέχεια να αποσταλεί και μεταφορτωθεί στην μνήμη του ενσωματωμένου συστήματος. Το mfsgen συνδέεται αυτόματα με τη βιβλιοθήκη libXilMFS και και είναι προγραμματισμένο να τρέχει στο μηχάνημα και όχι στον ενσωματωμένο (MicroBlaze ή PowerPC) επεξεργαστή του συστήματος. Μια ολόκληρη ιεραρχία καταλόγων του host Η/Υ μπορεί να αντιγραφεί σε μια τοπική MFS εικόνα αρχείων χρησιμοποιώντας το πρόγραμμα mfsgen. Στη συνέχεια, αυτή η εικόνα αρχείων μπορεί να 'κατέβει' στη μνήμη του ενσωματωμένου συστήματος, δημιουργώντας ένα προ-φορτωμένο σύστημα αρχείων.

Το πρόγραμμα καλείται από τη γραμμή εντολών, αφού πλοηγηθούμε στον φάκελο του συστήματος του Η/Υ που περιέχει τα αρχεία/καταλόγους από τα οποία θα αποτελείται η εικόνα αρχείων που θα δημιουργήσουμε. Για εμάς, ο φάκελος αυτός είναι στο ακόλουθο μονοπάτι *C:\Thesis\SmithWaterman3\SDK\SDK_Export\MemoryFileSystem*, όπου *SmithWaterman3* είναι το project που περιέχει όλο το ενσωματωμένο σύστημα, και *SDK\SDK_Export* ο φάκελος που περιέχει το workspace της εφαρμογής που εκτελείται στον Microblaze ενσωματωμένο επεξεργαστή, ενώ το πρόγραμμα έχει την ακόλουθη χρήση:

```
mfsgen -txcvsbf [mfs_filename] [num_blocks] <filelist>
```

t: λίστα με τα αρχεία στην MFS εικόνα του συστήματος αρχείων

c: η δημιουργία μιας νέας MFS εικόνας συστήματος χρησιμοποιώντας τη λίστα των αρχείων/καταλόγων που αναφέρονται στη γραμμή εντολών (αν δηλωθούν κατάλογοι στη γραμμή εντολών, αυτοί επεξεργάζονται αναδρομικά)

x: εξαγωγή του MFS συστήματος αρχείων από την εικόνα στο host μηχάνημα

v: λεπτομερής λειτουργία

f: καθορισμός του ονόματος του αρχείου (mfs_filename), όπου το αρχείο εικόνας MFS είναι αποθηκευμένο

- Εάν η επιλογή f καθορίζεται, το όνομα του αρχείου MFS θα πρέπει να προσδιορίζεται
- Εάν η επιλογή f παραλείπεται, το προεπιλεγμένο όνομα αρχείου είναι filesystem.mfs

s: επιλογή endianness

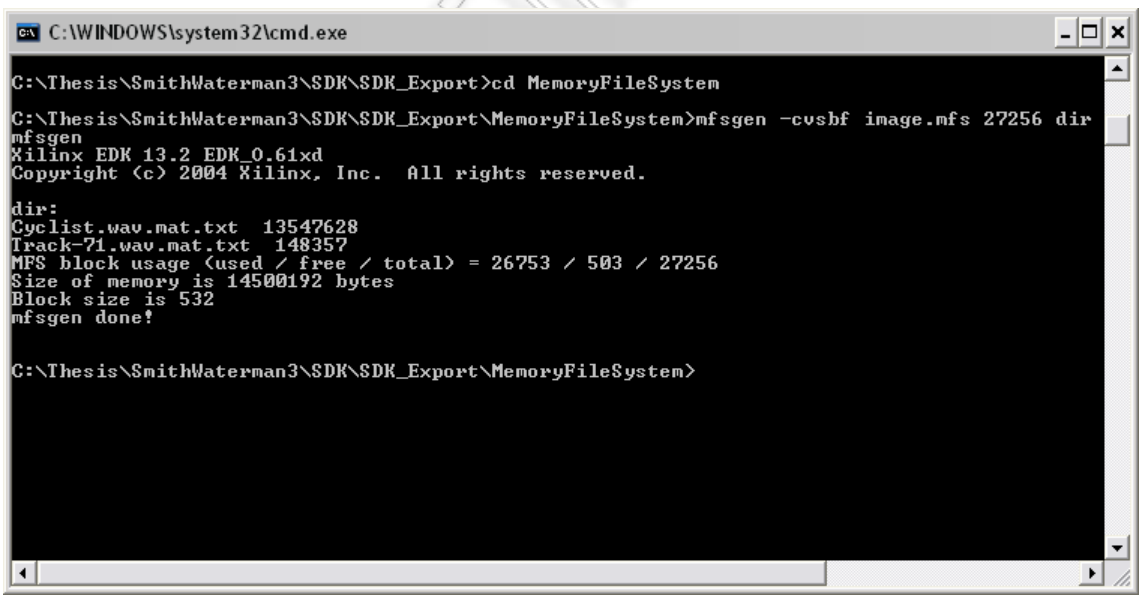
b: αριθμός των μπλοκ - θα πρέπει να είναι περισσότερα από 2

- Εάν η επιλογή b ορίζεται, η τιμή των num_blocks πρέπει να καθορίζεται
- Εάν η επιλογή b παραλείπεται, η προεπιλεγμένη τιμή των num_blocks είναι 5000
- Η επιλογή b έχει νόημα μόνον όταν χρησιμοποιείται σε συνδυασμό με την επιλογή c

Στην δική μας περίπτωση, η εντολή δημιουργίας της εικόνας συστήματος αρχείων ήταν η ακόλουθη:

C:\Thesis\SmithWaterman3\SDK\SDK_Export\MemoryFileSystem>mfsgen -cvsbf image.mfs 27256 dir

όπου καλούμε το πρόγραμμα mfsgen να παράγει μία εικόνα MFS στο φάκελο MemoryFileSystem, που θα περιέχει τη δομή αρχείων του καταλόγου dir (ο φάκελος περιέχει τα αρχεία *Cyclist.wav.mat* και *Track71-01.wav.mat*, σε μορφή *txt*, όπως αυτά προκύπτουν από προεπεξεργασία σε Η/Υ με λειτουργικό ΄συστημα linux με χρήση της βιβλιοθήκης *sndfile*), το μέγεθος σε μπλοκ της εικόνας θα είναι 27256 και θα έχει όνομα **image.mfs**. Στο screenshot που ακολουθεί, απεικονίζεται η εκτέλεση της παραπάνω εντολής από τη γραμμή εντολών, η οποία αναλύει τα περιεχόμενα της δομής που θα περιλαμβάνει η εικόνα του συστήματος αρχείων, πόσα bytes καταλαμβάνουν στο δίσκο καθένα από τα αρχεία αυτά, καθώς και αναλυτικές πληροφορίες για την αξιοποίηση των block της MFS εικόνας (πόσα block είναι κατειλημμένα και πόσα ελεύθερα). Τέλος, μας δίνεται το συνολικό μέγεθος της μνήμης σε bytes που καταναλώνει η εικόνα MFS που δημιουργείται, πληροφορία απαραίτητη για την σωστή παραμετροποίηση της βιβλιοθήκης στα εργαλεία προγραμματισμού της Xilinx, καθώς και την αρχικοποίηση του συστήματος αρχείων στη συνέχεια.



Παράρτημα Β - 1: Εκτέλεση του προγράμματος mfsgen για την δημιουργία μίας εικόνας συστήματος αρχείων (MFS).

Παράρτημα Β.2. Συναρτήσεις MFS

Στο σημείο αυτό, παρουσιάζουμε τις ειδικές συναρτήσεις της βιβλιοθήκης xilmfs, μέσω των οποίων μπορούμε να έχουμε πρόσβαση στο MFS σύστημα αρχείων που δημιουργήθηκε, από κώδικα σε γλώσσα C.

void mfs_init_fs (int numbytes, char *address, int init_type)
void mfs_init_genimage (int numbytes, char *address, int init_type)
int mfs_change_dir (char *newdir)
int mfs_create_dir (char *newdir)
int mfs_delete_dir (char *dirname)
int mfs_get_current_dir_name (char *dirname)
int mfs_delete_file (char *filename)
int mfs_rename_file (char *from_file, char *to_file)
int mfs_exists_file (char *filename)
int mfs_get_usage (int *num_blocks_used, int *num_blocks_free)
int mfs_dir_open (char *dirname)
int mfs_dir_close (int fd)
int mfs_dir_read (int fd, char **filename, int *filesize, int *filetype)
int mfs_file_open (char *filename, int mode)
int mfs_file_read (int fd, char *buf, int buflen)
int mfs_file_write (int fd, char *buf, int buflen)
int mfs_file_close (int fd)
long mfs_file_lseek (int fd, long offset, int whence)
int mfs_ls (void)
int mfs_ls_r (int recurse)
int mfs_cat (char *filename)
int mfs_copy_stdin_to_file (char *filename)
int mfs_file_copy (char *from_file, char *to_file)

Παράρτημα Β - 2: Συναρτήσεις της βιβλιοθήκης XilMFS.

Από τις συναρτήσεις αυτές, λίγες ήταν αυτές που χρειάστηκαν στο δικό μας κώδικα C για την αρχικοποίηση του συστήματος και το διάβασμα των απαραίτητων αρχείων. Παρακάτω, παραθέτουμε αναλυτικά ποιες συγκεκριμένες εντολές ήταν αυτές.

void **mfs_init_fs** (int numbytes, char *address, int init_type) ή

void **mfs_init_genimage** (int numbytes, char *address, int init_type)

numbytes είναι το πλήθος των bytes μνήμης που είναι διαθέσιμα για το σύστημα αρχείων (τίθεται ίσο με το αποτέλεσμα που μας έδωσε το πρόγραμμα mfsgen κατά τη δημιουργία της MFS εικόνας),

address είναι η διεύθυνση βάσης από την οποία ξεκινάει η μνήμη του συστήματος αρχείων,

init_type είναι μία από τις ακόλουθες επιλογές MFSINIT_NEW, MFSINIT_IMAGE ή

MFSINIT_ROM_IMAGE (MFSINIT_NEW για τη δημιουργία ενός νέου, άδειου συστήματος αρχείων για γράψιμο/διάβασμα, MFSINIT_IMAGE για την αρχικοποίηση ενός συστήματος αρχείων του οποίου τα δεδομένα έχουν φορτωθεί προηγουμένως στην μνήμη στην προκαθορισμένη διεύθυνση βάσης, MFSINIT_ROM_IMAGE για την αρχικοποίηση ενός συστήματος αρχείων μόνο για διάβασμα, τα δεδομένα του οποίου έχουν φορτωθεί προηγουμένως στην μνήμη στην προκαθορισμένη διεύθυνση βάσης)

Η συνάρτηση αυτή αρχικοποιεί τη μνήμη του συστήματος αρχείων και πρέπει να καλείται προτού γίνει οποιαδήποτε άλλη λειτουργία στο σύστημα αρχείων. Στην περίπτωση που για τη δημιουργία του συστήματος αρχείων χρησιμοποιήθηκε το πρόγραμμα mfsgen, τότε χρησιμοποιούμε την συνάρτηση mfs_init_genimage.

---

int **mfs_change_dir** (char **newdir*)

newdir είναι ο προορισμός της συνάρτησης chdir

Επιστρέφει 1, όταν είναι επιτυχημένη και 0 όταν αποτύχει

Εάν ο φάκελος newdir υπάρχει, τότε η συνάρτηση αυτή τον θέτει ως τον τρέχων κατάλογο του MFS. Σε περίπτωση αποτυχίας της συνάρτησης, ο τρέχων κατάλογος παραμένει ο ίδιος.

---

int **mfs_get_current_dir_name** (char *dirname)

dirname: Το όνομα του τρέχοντος καταλόγου επιστρέφεται σε αυτό το δείκτη. Επιστρέφει 1, όταν είναι επιτυχημένη και 0 όταν αποτύχει

Η συνάρτηση επιστρέφει το όνομα του τρέχοντος καταλόγου σε έναν προκαθορισμένο buffer dirname (τουλάχιστον 16 χαρακτήρες). Δεν επιστρέφει το όνομα του απόλυτου μονοπατιού του τρέχοντος καταλόγου, απλά το όνομά του.

---

int **mfs_dir_open** (char **dirname*)

dirname: Ο κατάλογος ο οποίος πρόκειται να ανοιχτεί προς ανάγνωση

Επιστρέφει το δείκτη του dirname στον πίνακα των ανοιχτών αρχείων ή -1 σε περίπτωση αποτυχίας

Η συνάρτηση ανοίγει ένα φάκελο dirname για διάβασμα. Το διάβασμα του καταλόγου πραγματοποιείται μέσω της συνάρτησης mfs_dir_read().

---

int **mfs_dir_close** (int fd)

fd: περιγραφέας του καταλόγου που επέστρεψε η συνάρτηση open

Επιστρέφει 1 σε περίπτωση επιτυχίας, 0 σε περίπτωση αποτυχίας

Η συνάρτηση κλείνει τον κατάλογο στον οποίο δείχνει ο fd. Το σύστημα αρχείων ανακτά τον δείκτη fd, τον οποίο μπορεί πλέον να τον χρησιμοποιήσει για νέα αρχεία/καταλόγους.

---

int **mfs_exists_file** (char *filename)

filename: αρχείο/κατάλογος που πρέπει να ελεγχθεί η ύπαρξή του

Επιστρέφει

0: αν το filename δεν υπάρχει

1: αν το filename υπάρχει και αντιπροσωπεύει αρχείο

2: αν το filename υπάρχει και αντιπροσωπεύει φάκελο

Η συνάρτηση ελέγχει την ύπαρξη αρχείου ή καταλόγου στον τρέχων κατάλογο του MFS.

---

int **mfs_file_open** (char *filename, int mode)

filename: το αρχείο που πρόκειται να ανοιχτεί

mode: λειτουργία Read/Write ή Create

Επιστρέφει το δείκτη του filename στον πίνακα των ανοιχτών αρχείων, ή 0 σε περίπτωση αποτυχίας

Η συγκεκριμένη συνάρτηση αναλαμβάνει να ανοίξει ένα αρχείο filename υπό συγκεκριμένη mode λειτουργία. Θα πρέπει να χρησιμοποιείται μόνο για αρχεία και όχι για καταλόγους.

---

int **mfs_file_read** (int fd, char *buf, int buflen)

fd: περιγραφέας του αρχείου που επέστρεψε η συνάρτηση open

buf: buffer προορισμού για το διάβασμα

buflen: μήκος του buffer

Επιστρέφει τον αριθμό των bytes που διαβάστηκαν σε περίπτωση επιτυχίας, και 0 σε περίπτωση αποτυχίας

Η συνάρτηση αυτή διαβάζει buflen πλήθος από bytes και τα τοποθετεί στον buf. Ο fd θα πρέπει να είναι ένα έγκυρος δείκτης στον πίνακα των ανοιχτών αρχείων, και να δείχνει σε αρχείο και όχι σε κατάλογο, ενώ ο buf θα πρέπει να είναι ήδη διαθέσιμος (μεγέθους buflen ή περισσότερο). Αν έχουν απομείνει λιγότεροι από buflen χαρακτήρες διαθέσιμοι, τότε μόνο τόσοι χαρακτήρες θα διαβαστούν.

---

int **mfs_file_close** (int fd)

fd: περιγραφέας του αρχείου που επέστρεψε η συνάρτηση open

Επιστρέφει 1 σε περίπτωση επιτυχίας, 0 σε περίπτωση αποτυχίας

Η συνάρτηση κλείνει το αρχείο στο οποίο δείχνει ο fd. Το σύστημα αρχείων ανακτά τον δείκτη fd, τον οποίο μπορεί πλέον να τον χρησιμοποιήσει για νέα αρχεία.

---

int **mfs_ls** (void)

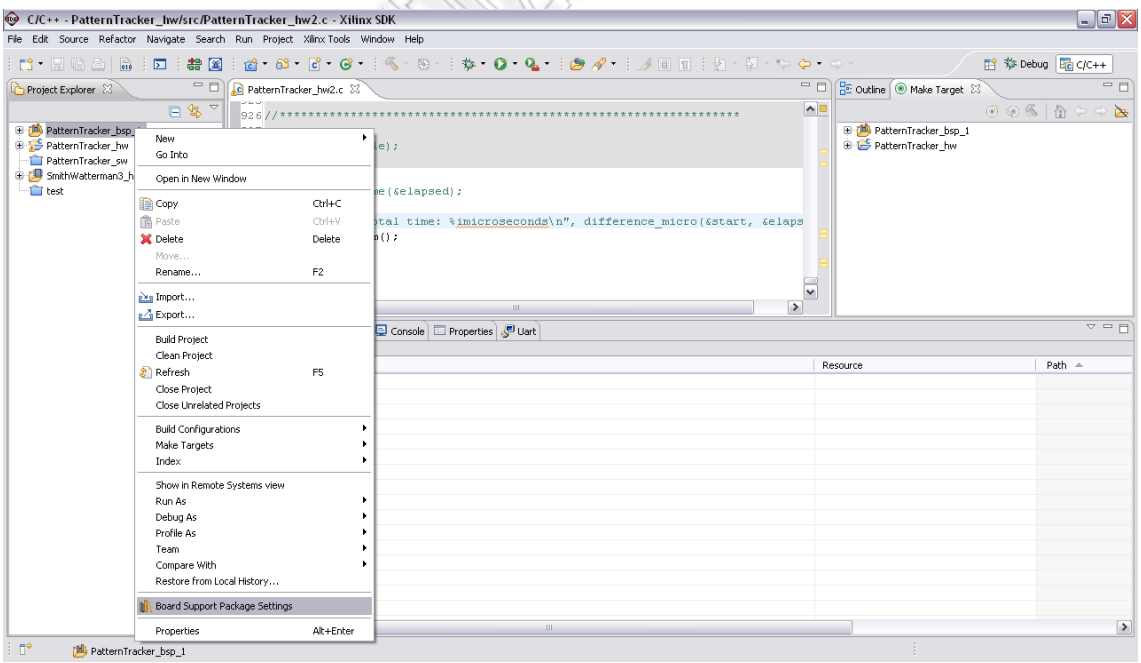
Επιστρέφει 1 σε περίπτωση επιτυχίας, 0 σε περίπτωση αποτυχίας

Η συνάρτηση αυτή παράγει μία λίστα με τα περιεχόμενα του τρέχοντος καταλόγου στο STDOUT.

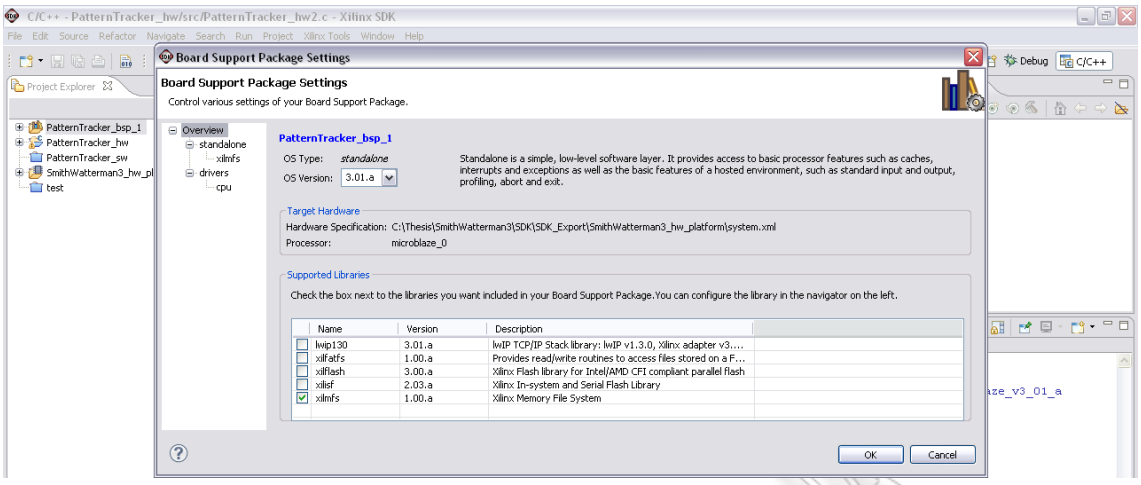
Παράρτημα Β.3. Ενσωμάτωση και τροποποίηση της βιβλιοθήκης Xilmfs στο SDK

Για να ενσωματώσουμε τη βιβλιοθήκη xilmfs στην εφαρμογή, θα πρέπει πρώτα να την δηλώσουμε στο λειτουργικό σύστημα της πλατφόρμας λογισμικού (board support package) πάνω στην οποία θέλουμε να αναπτύξουμε την εφαρμογή μας.

Στο εργαλείο SDK (Software Development Kit), επιλέγουμε το Board Support Package της εφαρμογής και επιλέγουμε στη συνέχεια τις ρυθμίσεις του.

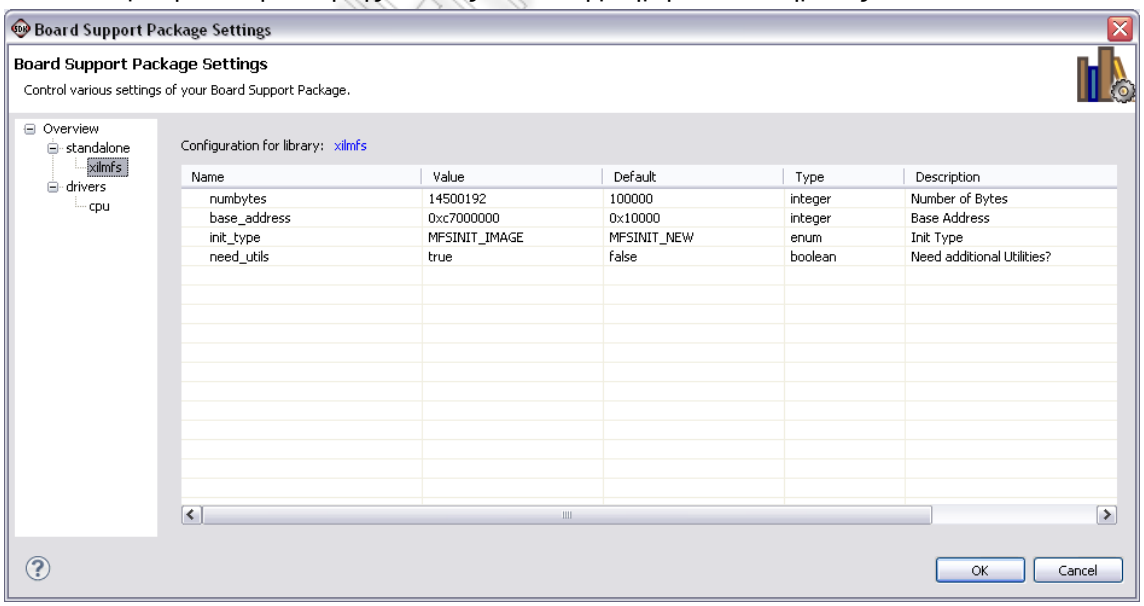


Παράρτημα Β - 3: Παραμετροποίηση Board Support Package (BSP) της εφαρμογής (1/3).



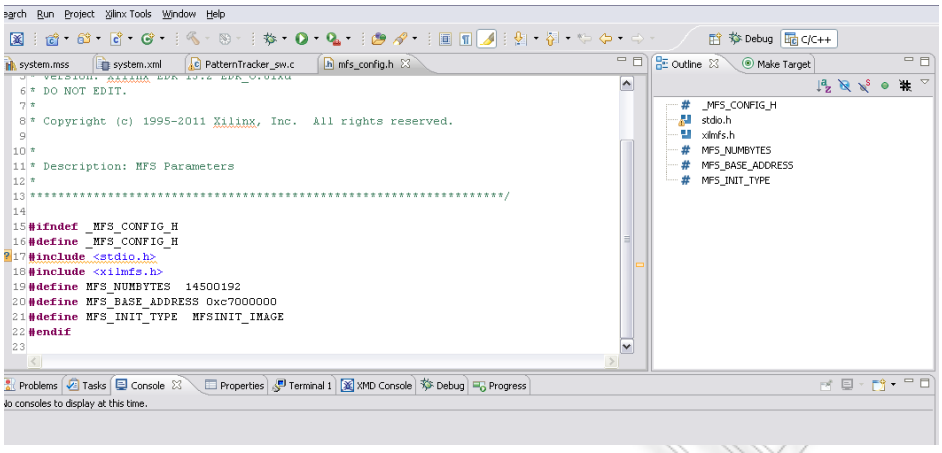
Παράρτημα Β - 4: Παραμετροποίηση Board Support Package (BSP) της εφαρμογής (2/3).

Στο παράθυρο που εμφανίζεται, βλέπουμε το λειτουργικό σύστημα που χρησιμοποιούμε (standalone), καθώς και τις ειδικές βιβλιοθήκες που παρέχει η Xilinx. Για την ενσωμάτωση της βιβλιοθήκης XilMFS στο πακέτο λογισμικού επιλέγουμε τη βιβλιοθήκη (Παράρτημα Β- 1) και πατώντας πάνω στο όνομά της στο πλαίσιο πλοήγησης στα αριστερά του παραθύρου, τροποποιούμε τα απαραίτητα στοιχεία της βιβλιοθήκης (Παράρτημα Β- 2), έτσι ώστε να συμφωνεί με τα στοιχεία της εικόνας MFS που δημιουργήσαμε με το πρόγραμμα mfsgen. Αξίζει να σημειωθεί, ότι η τιμή των numbytes τίθεται ίση με την τιμή των bytes που καταλαμβάνει η εικόνα MFS στο mfsgen, ενώ ως base address επιλέξαμε μία διεύθυνση εντός του εύρους διευθύνσεων (0xC0000000 – 0xC7FFFFFF) της DDR μνήμης του συστήματος, αρκετά ‘μακριά’ ωστόσο από τη base address της DDR, ώστε να εξασφαλιστεί ότι δεν θα συμπέσει με το προγραμματιστικό αρχείο elf του ενσωματωμένου επεξεργαστή, όταν αυτό ‘κατέβει’ στην πλακέτα και φορτωθεί στη DDR μνήμη από τη διεύθυνση μνήμης 0xC0000000. Το μέγεθος του αρχείου elf δίνεται κατά τη δημιουργία του στην κονσόλα του SDK, οπότε είναι εύκολο να υπολογιστεί τι εύρος διευθύνσεων θα πρέπει να αφεθεί κενό, ώστε να ξεκινήσει από εκεί και πέρα η αποθήκευση της εικόνας MFS στη μνήμη του συστήματος.



Παράρτημα Β - 5: Παραμετροποίηση Board Support Package(BSP) της εφαρμογής (3/3) - εισαγωγή βιβλιοθήκης XilMFS.

Το SDK αναλαμβάνει, μετέπειτα, να δημιουργήσει τη βιβλιοθήκη XilMFS με τις παραμέτρους που ορίστηκαν παραπάνω χρησιμοποιώντας το εργαλείο libgen. Η βιβλιοθήκη ενσωματώνεται αυτόματα στο σύστημα και στο αρχείο mfs_config.h δηλώνονται οι σωστές παράμετροι της βιβλιοθήκης MFS. Τόσο η βιβλιοθήκη *xilmfs.h*, όσο και η *mfs_config.h* πρέπει να δηλωθούν (#include) στην εφαρμογή, ώστε να μπορεί να γίνει σωστή χρήση του MFS συστήματος αρχείων.



Παράρτημα Β - 6: Αρχείο παραμέτρων mfs_config.h της βιβλιοθήκης XilMFS.

Παράρτημα Β.4. Μεταφόρτωση στη μνήμη της πλακέτας μέσω XMD ή run configuration

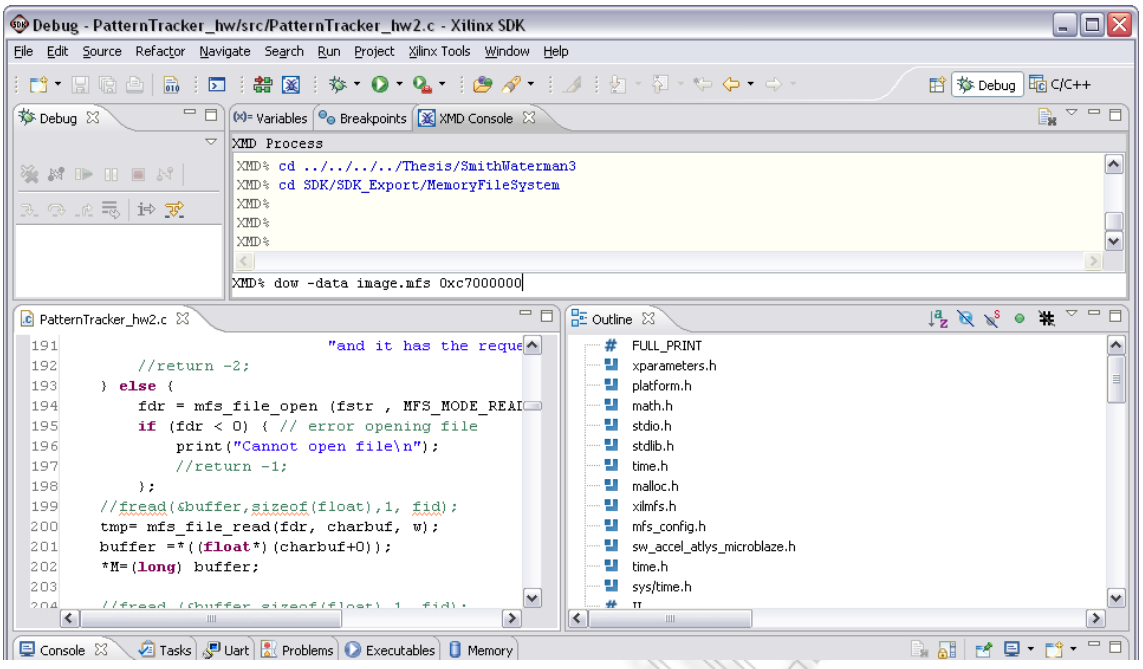
Αφού δημιουργηθεί η εικόνα MFS και παραμετροποιηθεί η βιβλιοθήκη, ώστε να μπορεί να χρησιμοποιηθεί από το ενσωματωμένο σύστημα, θα πρέπει να μεταφορτώσουμε το αρχείο της εικόνας MFS στην αναπτυξιακή πλακέτα. Αυτό επιτυγχάνεται με δύο τρόπους:

- ✓ Είτε, μέσω της XMD κονσόλας του περιβάλλοντος SDK, και αφού έχουμε προηγουμένως πλοηγηθεί στο κατάλογο που βρίσκεται η εικόνα MFS, δίνοντας την εντολή

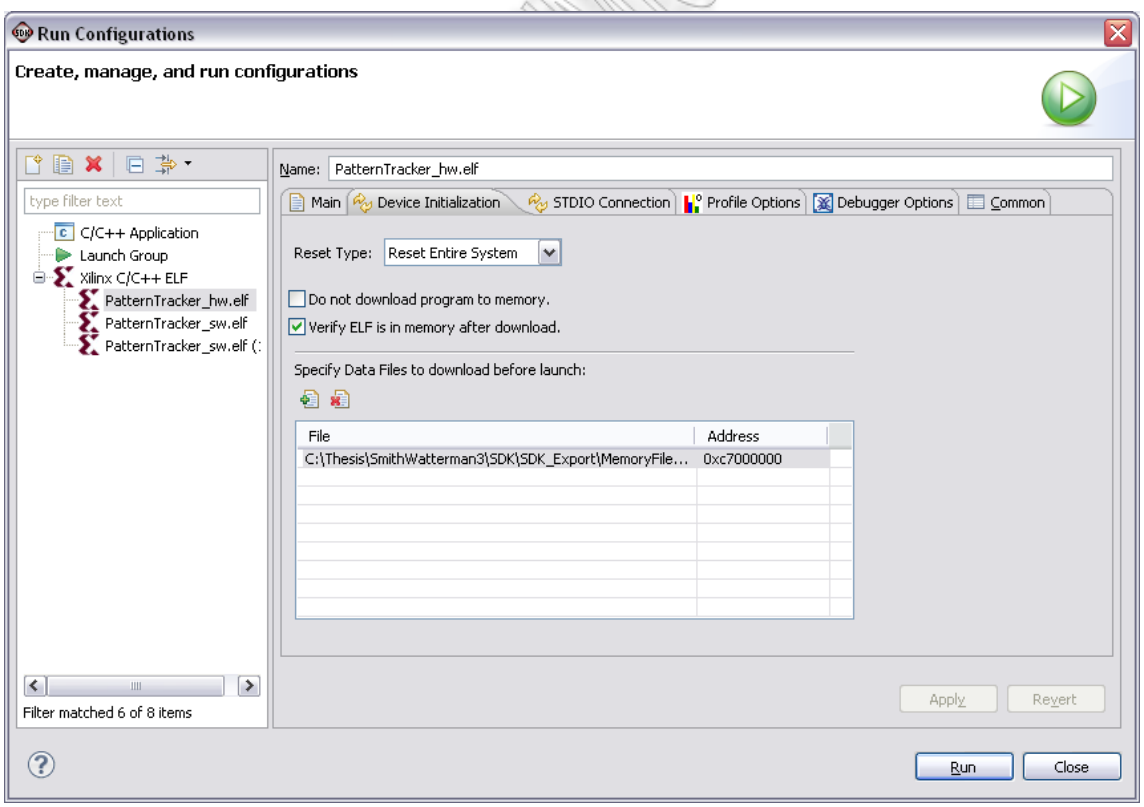
  XMD% dow -data image.mfs 0xC7000000

  Με την εντολή αυτή, ουσιαστικά, λέμε στην κονσόλα να 'κατεβάσει' ένα data αρχείο με όνομα image.mfs στην πλακέτα, στην διεύθυνση μνήμης που ορίζεται (Παράρτημα Β- 9). Αν η επιλογή data έλειπε, τότε η κονσόλα θα καταλάβαινε ότι αυτό που κατεβάζει στην πλακέτα είναι το αρχείο προγραμματισμού του επεξεργαστή elf.

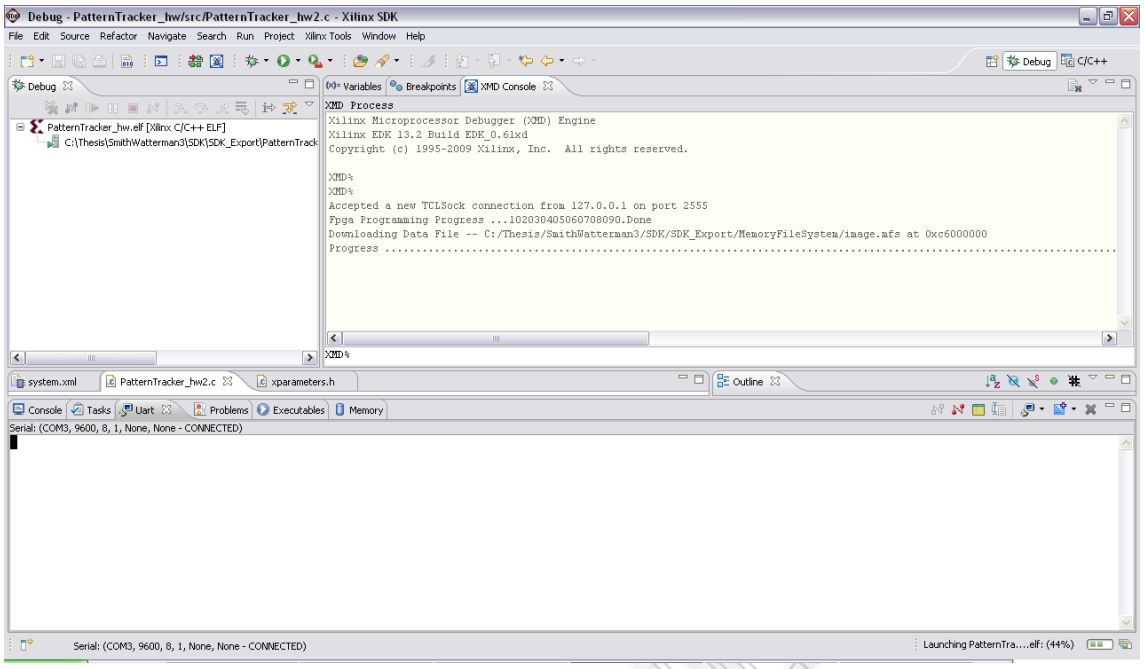
- ✓ Είτε δημιουργώντας ένα run configuration, επιλέγοντας την εφαρμογή και πατώντας δεξί κλικ ή από το tab run. Στο run configuration, ουσιαστικά, ορίζουμε τις παραμέτρους για να τρέξει η εφαρμογή μας (αρχείο elf) και αναλαμβάνει αυτό να κατεβάσει τα απαιτούμενα αρχεία στην πλακέτα. Αν παρατηρήσουμε την κονσόλα XMD, χρησιμοποιούνται οι ίδιες εντολές με την παραπάνω λύση, απλά αυτοματοποιείται η διαδικασία (Παράρτημα Β- 10, 11)



Παράρτημα Β - 7: Φόρτωση του αρχείου εικόνας MFS μέσω της XMD κονσόλας.



Παράρτημα Β - 8: Φόρτωση του αρχείου εικόνας MFS μέσω Run Configuration.



Παράρτημα Β - 9: Εκτέλεση του Run Configuration και 'κατέβασμα' του αρχείου εικόνας στην πλακέτα.

Βιβλιογραφία

[1]. Mihalis Psarakis, Aggelos Pikrakis, Giannis Dendrinos, “FPGA_based Acceleration for Tracking Audio Effects in Movies”, FCCM Annual International IEEE Symposium on Field Programmable Custom Computing Machines, April 2012

[2]. Δενδρινός Γιάννης, “Επιτάχυνση του αλγορίθμου δυναμικού προγραμματισμού Smith-Waterman με χρήση FPGA”, Πανεπιστήμιο Πειραία, Τμήμα Πληροφορικής, Οκτώβριος 2012

[3]. Bryan H. Fletcher, “FPGA Embedded Processors: Revealing True System Performance”, Embedded Training Program, Embedded Systems Conference ETP-367, San Francisco, 2005

[4]. http://www.xilinx.com, Xilinx official site

[5]. Laiq Hasan and Zaid Al-Ars, ”An Overview of Hardware-Based Acceleration of Biological Sequence Alignment”, Computational Biology and Applied Bioinformatics, InTech, The Netherlands, 2011

[6]. C.W. Yu, K.H. Kwong, K.H. Lee and P.H.W. Leong, “A SMITH-WATERMAN SYSTOLIC CELL”, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, HONG KONG

[7]. Xianyang /jiang, Xinchun Liu, Liu Xu, Peiheng Zhang, Ninghui Sun, “A Reconfigurable Accelerator for Simth-Waterman Algorithm”, IEEE TRANSACTIONS IN CIRCUITS AND SYSTEMS, December, 2007

[8]. Mihalis Psarakis, “Advanced Digital Design Notes”, Department of Informatics, University of Piraeus, Piraeus, 2012

[9]. Maya B. Gokhale, Paul S.Graham, “reconfigurable Computing, Accelerating computation with field-programmable gate arrays”, Springer, 2010

[10]. http://www.digilentinc.com , “Atlys Board Reference Manual”, 2011

[11]. Olaf O. Storaasli, Dave Strenski, “Exploring Accelerating Science Applications with FPGAs”, Laboratory Directed Research & Development Program of Oak Ridge National Laboratory, 2008

[12]. T. F. Smith and M. S.Waterman, “Identication of common molecular subsequence”, Journal of Molecular Biology, 147:196-197, 1981.

[13]. Mahendra Vucha, Arvind Rajawat, ‘Design and FPGA Implementation of Systolic Array Architecture for Matrix Multiplication’, International Journal of Computer Applications (0975 –8887) Volume 26–No.3, July 2011

[14]. Altera Corporation – White Paper, “Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform”, Altera Corporation

[15]. Xilinx, “LibXil Memory File System (MFS)”