ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σύνθεση Υψηλού Επιπέδου του Αλγορίθμου OpenSURF

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Φαλιάγκας

Επιβλέπων : Γεώργιος Οικονομάκος

Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σύνθεση Υψηλού Επιπέδου του Αλγορίθμου OpenSURF

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Φαλιάγκας

Επιβλέπων : Γεώργιος Οικονομάκος

Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή 22^{η} Ιουλίου 2013.

Αθήνα, Ιούλιος 2013

.....	.....	.....
Γεώργιος Οικονομάκος	Κιαμάλ Πεκμεστζή	Δημήτρης Σούντρης
Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

.....

Κωνσταντίνος Φαλιάγκας

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

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η σύνθεση υψηλού επιπέδου του αλγορίθμου OpenSURF. Το εργαλείο που χρησιμοποιούμε για τον σκοπό αυτό είναι το Vivado HLS της Xilinx. Αρχικά γίνεται μετατροπή της περιγραφής του αλγορίθμου από C++ σε C. Η περιγραφή σε C πρέπει να τροποποιηθεί με τέτοιο τρόπο ώστε να είναι κατάλληλη για να περάσει μέσα από το εργαλείο και να μας δώσει την RTL περιγραφή. Στη συνέχεια εφαρμόζουμε μετασχηματισμούς στην περιγραφή, προκειμένου να βελτιστοποιήσουμε την υλοποίηση και να πάρουμε το καλύτερο δυνατό αποτέλεσμα. Η τελική υλοποίηση κρίνεται ως προς την χρήση των πόρων υλικού που απαιτούνται, την κατανάλωση ισχύος και το latency.

Η διπλωματική εργασία χωρίζεται σε οχτώ κεφάλαια. Στα πρώτα κεφάλαια γίνεται μία εισαγωγή στα ενσωματωμένα συστήματα, μία σύνδεσή αυτών με την διπλωματική εργασία και περιγράφεται η δομή των FPGAs. Στα επόμενα κεφάλαια αναλύεται η διαδικασία της σύνθεσης υψηλού επιπέδου, ο επεξεργαστής Microblaze που χρησιμοποιείται για τη συσχεδίαση HW/SW καθώς και τα χαρακτηριστικά του εργαλείου σχεδίασης Vivado HLS που χρησιμοποείται στα πλαίσια της διπλωματικής εργασίας.Στη συνέχεια δίνεται η δομή του αλγορίθμου OpenSURF ,μαζί με κάποια στοιχεία για το επιστημονικό πεδίο της όρασης υπολογιστών, και γίνεται η υλοποίησή του. Στο τελευταίο κεφάλαιο γίνεται η σύνοψη των αποτελεσμάτων που προέκυψαν και παρατίθενται προτάσεις για μελλοντική εργασία.

Λέξεις Κλειδιά

Ενσωματωμένα συστήματα, Σύνθεση Υψηλού Επιπέδου, OpenSURF, Vivado HLS, EDK, Virtex-6, FPGA,Όραση Υπολογιστών, MicroBlaze.

Abstract

The aim of this thesis is the High-Level Synthesis of the OpenSURF algorithm. The tool we use for this purpose is the Xilinx Vivado HLS. Initially we convert the description of the algorithm from C++ to C. The description in C should be amended in such a way that it is suitable to go through the tool and give us the RTL description. We then apply transformations to the description in order to optimize the implementation and get the best possible result. The final implementation is in the use of hardware resources needed, the power consumption and latency.

The thesis is divided into eight chapters. The first chapters, are an introduction to embedded systems and a description of the structure of FPGAs. In the following chapters we analyze the process of high-level synthesis, the processor Microblaze which is used for HW/SW codesign and the characteristics of the Vivado HLS tool, used in this thesis. Next, the structure of the algorithm OpenSURF is given, together with some information about the scientific field of computer vision and the implementation. The last chapter is a summary of the results obtained and some recommendations for future work are presented.

KeyWords

Embedded Systems, High-Level Synthesis, OpenSURF, Vivado HLS, EDK, Virtex-6, FPGA, Computer Vision, MicroBlaze.

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω κατά κύριο λόγο τον επίκουρο καθηγητή κ. Γεώργιο Οικονομάκο, για τις συμβουλές και τις ιδέες τους σε όλη την διάρκεια της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Μικροϋπολογιστών και Ψηφιακών Συστημάτων, και ιδιαίτερα τον Διονύση Διαμαντόπουλο, του οποίου η συμβολή υπήρξε καθοριστική για την ολοκλήρωση της εργασίας. Τέλος θεωρώ χρέος μου να ευχαριστήσω την οικογένειά μου και το φιλικό μου περιβάλλον, καθώς στάθηκαν δίπλα μου, με στήριξαν, και συνέβαλαν τα μέγιστα τόσο στην ολοκλήρωση της εργασίας όσο και καθ' όλη τη διάρκεια των σπουδών μου.

Πίνακας Περιεχομένων

Κεφάλαιο 1: Ενσωματωμένα Συστήματα

Κεφάλαιο 2: Field Programming Gate Arrays (FPGA)

Κεφάλαιο 3: Σύνθεση υψηλού επιπέδου για σχεδίαση Hardware

Κεφάλαιο 4: Σχεδίαση με χρήση του συνεπεξεργαστή Microblaze™

Κεφάλαιο 5: Xilinx® Vivado HLS

Κεφάλαιο 6: Ο αλγόριθμος OpenSURF

Κεφάλαιο 7: High-Level Synthesis του αλγορίθμου OpenSURF

Ευρετήριο Σχημάτων και Πινάκων

Κεφάλαιο 1

Κεφάλαιο 2

Κεφάλαιο 3

Κεφάλαιο 4

Κεφάλαιο 5

Κεφάλαιο 6

Κεφάλαιο 7

Κεφάλαιο

Κεφάλαιο 1

Ενσωματωμένα συστήματα

Επισκόπηση

Στο συγκεκριμένο κεφάλαιο γίνεται μια εισαγωγή στα ενσωματωμένα συστήματα. Παρουσιάζονται τα εξαρτήματα από τα οποία αποτελείται ένα ενσωματωμένο σύστημα και γίνεται μια περιγραφή των χαρακτηριστικών τους. Τέλος, δίνονται παραδείγματα ενσωματωμένων συστημάτων και αναφέρονται σύγρονες τάσεις χρήσης τους.

1.1 Εισαγωγή

Ενσωματωμένο σύστημα είναι ένα ειδικού σκοπού υπολογιστικό σύστημα, που ενσωματώνεται πλήρως στη συσκευή που ελέγχει.Ένα ενσωματωμένο σύστημα έχει συγκεκριμένες απαιτήσεις και πραγματοποιεί προκαθορισμένες λειτουργίες σε αντίθεση με ένα γενικού σκοπού προσωπικό υπολογιστή. Τα ενσωματωμένα συστήματα είναι ουσιαστικά ένας συνδυασμός από hardware και software και πιθανώς μηχανικά και αλλα μέρη σχεδιασμένα να εκτελούν μια συγκεκριμένη συνάρτηση.Περιέχουν πυρήνες επεξεργασίας που είναι είτε μικροελεγκτές και επεξεργαστές ψηφιακού σήματος.

1.2 Εξαρτήματα

Τα κύρια εξαρτήματα από τα οποία αποτελούνται τα ενσωματωμένα συστήματα είναι:

- hardware: επεξεργαστή, χρονιστές, interrupt controller, i/o devices, μνήμες, θύρες

- main application software: μπορεί να εκτελέσει ταυτόχρονα μια σειρά εργασιών ή πολλαπλά καθήκοντα

- real time operating system: το RTOS καθορίζει τον τρόπο με τον οποίο θα δουλέψει το σύστημα.Είναι υπευθυνο για την επίβλεψη του application software και θέτει κανόνες κατα την διάρκεια εκτέλεσης της εφαρμογής.

  Να σημειωθεί ότι μικρής κλίμακας ενσωματωμένα συστήματα μπορεί να μην χρειάζονται RTOS.

1.3 Χαρακτηριστικά

Τα ενσωματωμένα συστήματα κατά την σχεδίαση τους μπορεί να είναι πολύ πιο απαιτητικά απ΄ ότι είναι ένας υπολογιστής γενικού σκοπού.Η λειτουργικότητα σε κάθε περίπτωση είναι πολύ σημαντική, ωστόσο οι ενσωματωμένες εφαρμογές πρέπει να πληρούν πολλά κριτήρια. Μερικά από τα βασικά χαρακτηριστικά και τις απαιτήσεις τους είναι:

- *κόστος* : Το κόστος αποτελείται από non recurring engineering κόστη όπως είναι το κόστος σχεδίασης και το κόστος παρασκευής πρωτοτύπου και recurring engineering κόστη όπως είναι το κόστος κατεργασίας, το πακετάρισμα του ολοκληρωμένου κυκλώματος και ο έλεγχος ορθής λειτουργίας του ενσωματωμένου συστήματος.Δεδομένου ότι χρησιμοποιούνται κυρίως σε ηλεκτρονικά είδη ευρείας κατανάλωσης, το κόστος τους θα πρέπει να είναι χαμηλό.

- *κατανάλωση ισχύος* : Η ισχύς που καταναλώνει το ενσωματωμένο σύστημα είναι μία από τις σημαντικότερες παραμέτρους που πρέπει να λαμβάνουμε υπόψιν κατά την σχεδίαση.Η ελαχιστοποίηση της ισχύος είναι σημαντική λόγω της περιορισμένης χωρητικότητας των μπαταριών που χρησιμοποιούν τα ενσωματωμένα συστήματα, της περιορισμένης διαθεσιμότητας της ενέργειας αλλά και του υψηλού κόστους της.

- *απόδοση*: Η απόδοση του ενσωματωμένου συστήματος είναι μια επίσης πολύ σημαντική παραμετρος και σχετίζεται με τη συχνότητα ρολογιού και τις εντολές που εκτελούνται ανα δευτερόλεπτο(τα οποία όμως δεν είναι καλα μεγέθη μέτρησης), το latency (ο χρόνος μεταξύ της έναρξης και του τέλους μιας εργασίας) και το throughput(ο αριθμός των εργασιών που εκτελούνται ανα δευτερόλεπτο)

- *πραγματικού χρόνου*: Πολλές φορές τα ενσωματωμένα συστήματα πρέπει να αναταποκρίνονται σε γεγονότα σε πραγματικό χρόνο, ενώ ταυτόχρονα ακολουθούν αυστηρούς περιορισμούς που επιβάλλονται από το περιβάλλον με το οποίο αλληλεπιδρούν. Πρέπει δηλαδή συνεχώς να αντιδρούν σε αλλαγές στο περιβάλλον του συστήματος και να υπολογίζουν συγκεκριμένα αποτελέσματα σε πραγματικό χρόνο, χωρίς καθυστέρηση

- *αξιοπιστία*: Τα ενσωματωμένα συστήματα πρέπει να είναι πολύ αξιόπιστα, δεδομένου

- επιτελούν κρίσιμες λειτουργίες. Για παράδειγμα, η αποτυχία ενός συστήματος που χρησιμοποιείται για έλεγχο πτήσης, μπορεί να έχει καταστροφικές συνέπειες

- *επεκτασιμότητα*: Η δυνατότητα επέκτασης ενός ενσωματωμένου συστήματος είναι ενα επίσης πλλυ σημαντικό χαρακτηριστικό του γιατί όταν

σχεδιάζεται το hardware για ένα προϊόν που πρόκειται να βγεί στην αγορά, το hardware αυτό αναμένεται να χρησιμοποιηθεί και στις αναβαθμίσεις αυτού του προϊόντος, με μικρές αλλαγές. Αν δεν ήταν επεκτάσιμο τότε κάθε φορά που έβγαινε μια επέκταση θα αυξανόταν πολύ ο χρόνος που απαιτείται για να βγει το προϊόν στην αγόρα, αλλά και το κόστος σχεδίασης του.

- *μέγεθος και βάρος*: τα χαρακτηριστικά αυτά μπορούν να ποικίλουν σε μεγάλο βαθμό ανάλογα με την εφαρμογή. Μια φορητή συσκευή μπορεί να έχει αυστηρές απαιτήσεις τόσο με το μέγεθος όσο και με το βάρος του τελικού συστήματος.

- single functioned: Η πλειοψηφία των ενσωματωμένων συστημάτων έχει σχεδιαστεί ώστε να εκτελούν μια συγκεκριμένη διεργασία επαναληπτικά.

1.4 Παραδείγματα και εφαρμογές

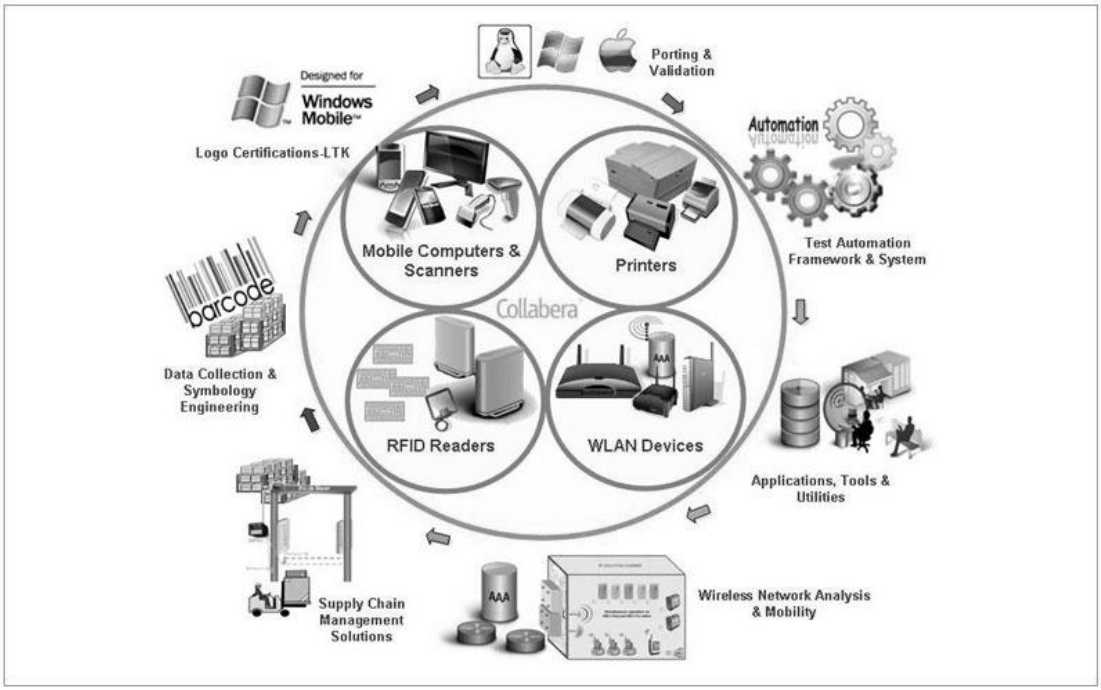
Ο τομέας των ενσωματωμένων συστημάτων είναι ιδιαίτερα αναπτυγμένος και συνεχίζει να αναπτύσσεται. Οι εφαρμογές τους καλύπτουν όλες τις πτυχές της καθημερινής ζωής και υπάρχουν πολλά παραδείγματα χρήσης τους. Μερικά τέτοια παραδείγματα συναντάμε σε:

- Συστήματα τηλεπικοινωνιών τα οποία χρησιμοποιούν ενσωματωμένα συστήματα σε διακόπτες τηλεφώνου, σε κινητά τηλέφωνα αλλά και στη δικτύωση των υπολογιστών με χρήση routers και γεφυρών δικτύου για τα δεδομένα της διαδρομής.

- Ηλεκτρονικά είδη ευρείας κατανάλωσης όπως είναι τα pda, mp3 players, κονσόλες βιντεοπαιχνιδιών, ψηφιακές φωτογραφικές μηχανές, συσκευές αναπαραγωγής DVD, δέκτες GPS, και εκτυπωτές. Επίσης πολλές οικιακές συσκευές, όπως φούρνοι μικροκυμάτων, πλυντήρια ρούχων και πιάτων, αλλά και συστήματα οικιακού αυτοματισμού περιλαμβάνουν ενσωματωμένα συστήματα προκειμένου να παρέχουν ευελιξία, αποτελεσματικότητα και αξιοπιστία.

- Τα συστήματα μεταφοράς χρησιμοποιούν όλο και περισσότερο ενσωματωμένα συστημάτα. Τα νέα αεροπλάνα περιλαμβάνουν προηγμένα ηλεκτρονικά συστήματα όπως συστήματα αδρανειακής καθοδήγησης και δέκτες GPS που έχουν επίσης σημαντικές απαιτήσεις ασφαλείας.Επίσης τα αυτοκίνητα και τα ηλεκτρικά και υβριδικά οχήματα χρησιμοποιούν όλο και περισσότερο τα ενσωματωμένα συστήματα για τη μεγιστοποίηση της αποδοτικότητας και τη μείωση της ρύπανσης. Άλλα συστήματα ασφαλείας αυτοκινήτου περιλαμβάνουν αντιεμπλοκής κατά την πέδηση (ABS), Ηλεκτρονικό Σύστημα Ευστάθειας (ESC / ESP), σύστημα ελέγχου πρόσφυσης (TCS) και αυτόματη τετρακίνηση.

- Με την εξέλιξη της τεχνολογίας στην Ιατρική τα σύχρονα ιατρικά μηχανήματα χρησιμοποιούν ενσωματωμένα συστήματα για την παρακολούθηση ζωτικών σημείων, υπάρχουν ηλεκτρονικά στηθοσκόπια για πολλαπλασιασμό των ήχων, καθώς και διάφορα μηχανήματα ιατρικής απεικόνισης (PET, SPECT, CT, MRI).



Σχημα 1.1 Εφαρμογές ενσωματωμένων συστημάτων

1.5 Σύγχρονες Τάσεις

Η έρευνα που γίνεται πάνω στα ενσωματωμένα συστήματα, για το 2013 ήταν επικεντρωμένη κυρίως στους τομείς:

- Ενσωματωμένα συστήματα πολυπύρηνων επεξεργαστών
- Λειτουργικά συστήματα πραγματικού χρόνου προσαρμοσμένα στις νέες αρχιτεκτονικές
- Κρυπτογραφία και ασφάλεια
- 3D ολοκλήρωση κυκλωμάτων
- Δίκτυο σε ψηφίδα (Network on Chip) και ασύρματη επικοινωνία μεταξύ πυρήνων

Κεφαλαιο 2

Field Programming Gate Arrays (FPGA)

Επισκόπηση

Σε αυτό το κεφάλαιο γίνεται αρχικά μια εισαγωγή στα FPGA. Στη συνέχεια, αναφέρονται κάποια βασικά χαρακτηριστικά τους και τέλος, αναλύεται η δομή τους.

2.1 Εισαγωγή

Το FPGA (ή συστοιχία επιτόπια προγραμματιζόμενων πυλών) είναι τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης και γεννήτριες PLL. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο στο τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία.

Ο κώδικας με τον οποίο προγραμματίζεται το FPGA γράφεται σε γλώσσες περιγραφής υλικού (VHDL, Verilog).

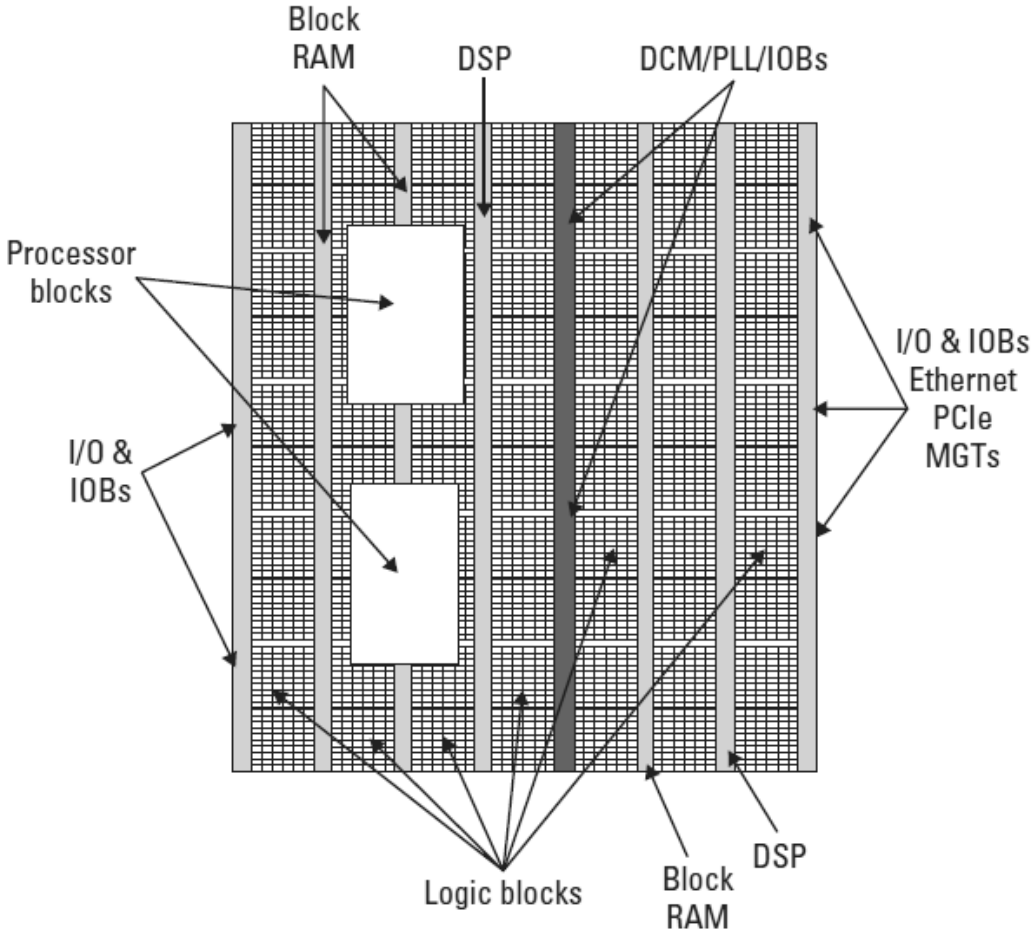
2.2 Χαρακτηριστικά

Μερικά από τα χαρακτηριστικά των FPGA είναι τα εξής:

- Το FPGA χάνει τον προγραμματισμό του κάθε φορά που διακόπτεται η τάση τροφοδοσίας. Επομένως απαιτεί εξωτερικό μικροεπεξεργαστή ή μνήμη με μόνιμη συγκράτηση δεδομένων (non-volatile memory) από τα οποία θα προγραμματίζεται, κάθε φορά που επανέρχεται η τάση τροφοδοσίας.
- Ο προγραμματισμός του FPGA μπορεί να αλλάζει κάθε φορά που τροποποιείται το λογισμικό του μικροεπεξεργαστή ή τα δεδομένα της μνήμης που το ελέγχει.
- Δεν υπάρχει όριο στο πόσες φορές μπορεί να επαναπρογραμματιστεί.
- Η κατανάλωση ισχύος είναι σημαντικά αυξημένη, σε σχέση με τα ASIC.

2.3 Δομή

Ένα FPGA, αποτελείται από έναν δισδιάστατο πίνακα από προγραμματιζόμενα λογικά blocks (CLBs) , από blocks σταθερής λειτουργίας και πηγές δρομολόγησης υλοποιημένες στη τεχνολογία CMOS. Κατά μήκος της περιμέτρου του FPGA υπάρχουν ειδικά λογικά blocks συνδεδεμένα με εξωτερικές συσκευασίες ακροδεκτών I/O. Τα λογικά blocks αποτελούνται από πολλαπλά λογικά κελιά, ενώ τα λογικά κελιά περιέχουν γεννήτριες συναρτήσεων και αποθηκευτικά στοιχεία.

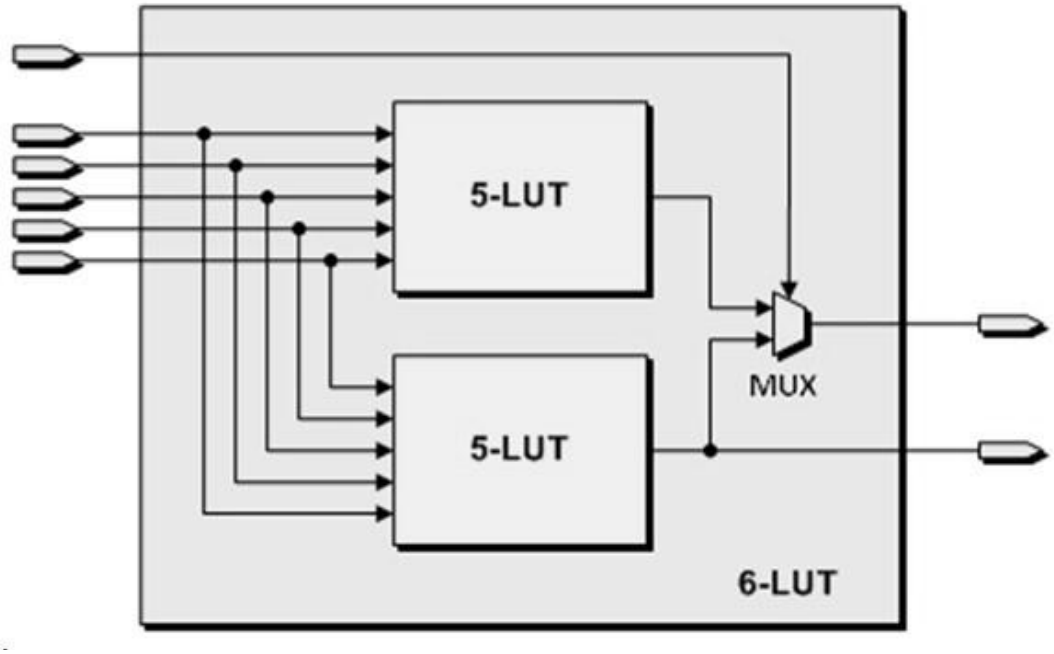


Σχήμα 2.1 Δομή FPGA

2.3.1 Γεννήτριες Συναρτήσεων (Look-Up Table)

Οι συσκευές FPGA χρησιμοποιούν γεννήτριες συναρτήσεων για την υλοποίηση της Boolean λογικής και όχι φυσικές πύλες σε αντίθεση με τις υπόλοιπες συσκευές προγραμματιζόμενης λογικής (PLAs, PALs, CPLDs). Για τις εισόδους μιας Boolean συνάρτησης δημιουργείται αρχικά ένας πίνακας αλήθειας. Για κάθε είσοδο ο πίνακας αλήθειας περιγράφει την τιμή της εξόδου της συνάρτησης. Κάθε bit της εξόδου της συνάρτησης αποθηκεύεται σε ξεχωριστό κελί μιας στατικής μνήμης. Τα κελιά αυτά συνδέονται ως είσοδοι σε έναν πολυπλέκτη όπου ανάλογα με την είσοδο επιλέγεται η κατάλληλη έξοδος. Το αποτέλεσμα είναι γνωστό ως **Look-Up**

Table (LUT). Τα FPGA της σειράς Virtex 7 δομούνται από LUTs 6 εισόδων. Αυτά τα LUTs των 6 εισόδων μπορούν να χρησιμοποιηθούν είτε ως άπλα 6-LUTs, είτε ως δυο 5-LUTs εφόσον τα τελευταία μοιράζονται τις ίδιες εισόδους. Αντίθετα με τα ψηφιακά κυκλώματα που υλοποιούνται με λογικές πύλες, η καθυστέρηση διάδοσης στα LUT είναι σταθερή. Αυτό σημαίνει ότι ανεξάρτητα από την πολυπλοκότητα του Boolean κυκλώματος, εφόσον "χωράει" σε ένα LUT, η καθυστέρηση διάδοσης είναι η ίδια. Αυτό επίσης ισχύει και για κυκλώματα που εκτείνονται σε περισσότερα LUT, αλλά εδώ η καθυστέρηση διάδοσης εξαρτάται και από των αριθμό των LUT που χρησιμοποιούνται.



Σχήμα 2.2 LUT 6 εισόδων

2.3.2 Στοιχεία Αποθήκευσης

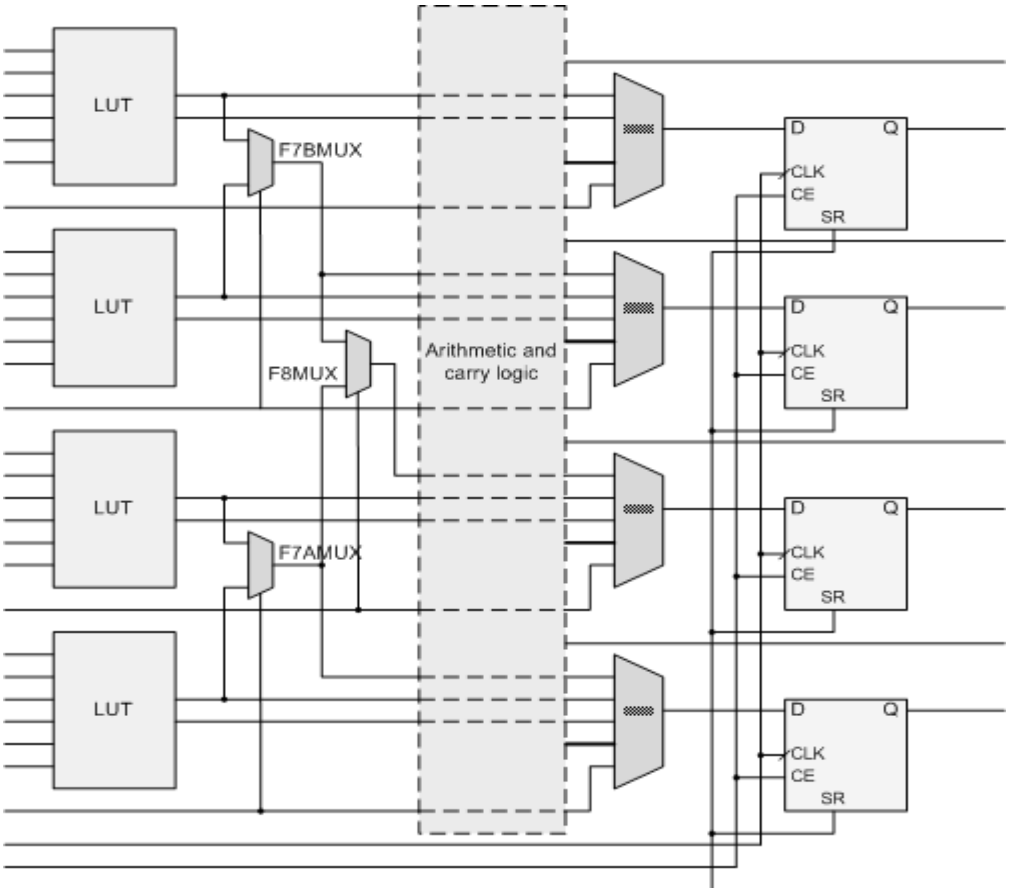
Ενώ οι γεννήτριες συναρτήσεων αποτελούν το βασικό block για την υλοποίηση συνδυαστικών κυκλωμάτων, τα FPGA διαθέτουν επιπρόσθετα στοιχεία παρέχοντας έτσι ένα πλούτο λειτουργιών. Όπως και στα block των PLA, D Flip-Flop ενσωματώνονται επίσης και στα FPGA. Τα Flip-Flops μπορούν να χρησιμοποιηθούν με διάφορους τρόπους, με πιο απλό, την αποθήκευση δεδομένων. Συνήθως, μια έξοδος του LUT συνδέεται με την είσοδο ενός Flip-Flop. Επίσης, το Flip-Flop μπορεί να χρησιμοποιηθεί και ως μανδαλωτής που λειτουργεί είτε σε θετική είτε σε αρνητική λογική.

2.3.3 Λογικά Κελιά

Συνδυάζοντας ένα LUT και ένα D Flip-Flop προκύπτει αυτό που αναφέρεται ως λογικό κελί. Τα λογικά κελιά αποτελούν το χαμηλότερου επιπέδου δομικό block σε ένα FPGA. Τόσο η συνδυαστική όσο και η ακολουθιακή λογική μπορούν να υλοποιηθούν από ένα λογικό κελί ή μια συλλογή από αυτά.

2.3.3.1 Slice

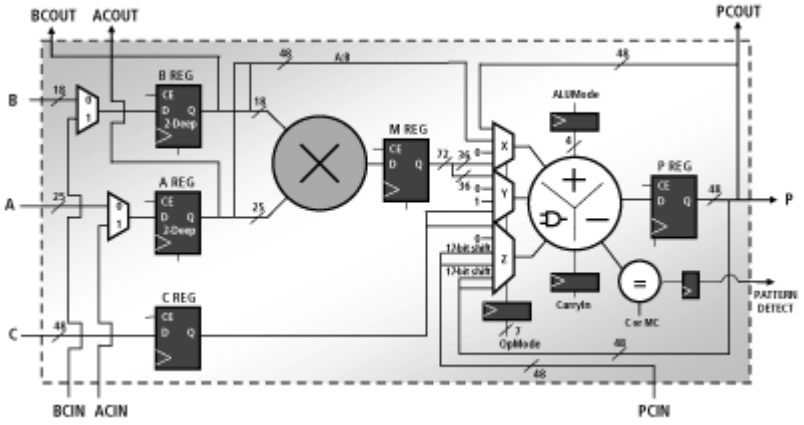
Τα Virtex 7 συνδυάζουν 4 λογικά κελιά, μαζί με τα flip-flops στις εξόδους των κελιών, για να δημιουργήσουν ένα slice. Με τέσσερα 6-LUTs και τέσσερα D Flip-Flops να βρίσκονται σε κοντινή απόσταση, είναι δυνατή η κατασκευή πιο σύνθετων σχεδίων (Σχήμα 2.3). Εκτός από τη Boolean λογική, ένα slice μπορεί να χρησιμοποιηθεί για αριθμητικές πράξεις καθώς και για την κατασκευή μνημών RAM και ROM. Μερικά slices συνδέονται με τέτοιο τρόπο ώστε μπορούν να χρησιμοποιηθούν για αποθήκευση δεδομένων, όπως διανεμημένες RAM και καταχωρητές ολίσθησης. Αυτό είναι δυνατό συνδυάζοντας πολλά LUTs σε ένα slice. Σε όλες αυτές τις δυνατές χρήσεις, τα D Flip-Flops μπορούν να χρησιμοποιηθούν για διαδικασία της σύγχρονης ανάγνωσης. Το Virtex-7 XC7VX485T διαθέτει συνολικά 75.900 slices.



Σχήμα 2.3 Slice

2.3.3.2 DSP Slices

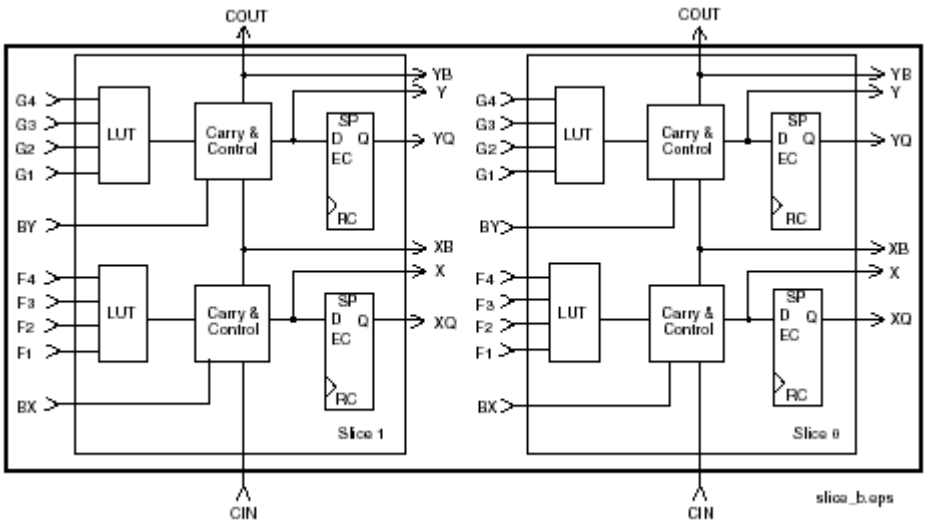
Μια κύρια χρήση των FPGA είναι η ψηφιακή επεξεργασία σήματος. Για την βελτίωση της απόδοσης, ορισμένα FPGA διαθέτουν ενσωματωμένα ειδικά DSP Blocks. Στα Virtex 7, τα DSP slices περιλαμβάνουν έναν 25×18 πολλαπλασιαστή συμπληρώματος του δύο, έναν συσσωρευτή, έναν αθροιστή/αφαιρέτη για πράξεις συνεχούς διοχέτευσης (pipelined) και λογικές πράξεις σε επίπεδο bit. Ενσωματώνοντας αυτή τη λειτουργία μέσα στα slices, μπορούμε να πετύχουμε σημαντική εξοικονόμηση των πόρων του FPGA, αφού η υλοποίηση του σχεδίου μόνο σε LUTs είναι αρκετά “ακριβή”. Το Virtex-7 XC7VX485T διαθέτει 2800 DSP Slices .



Σχήμα 2.4 DSP Slice

2.3.4 Λογικά Blocks

Ενώ τα λογικά κελιά μπορούν να θεωρηθούν ως τα βασικά δομικά blocks των εφαρμογών FPGA, στην πραγματικότητα είναι σύνηθες να ομαδοποιούμε μερικά λογικά κελιά σε blocks και να προσθέτουμε κυκλώματα ειδικού σκοπού με σκοπό να δημιουργήσουμε ένα λογικό block. Αυτό επιτρέπει σε μια ομάδα από λογικά κελιά, τα οποία βρίσκονται κοντά μεταξύ τους, να έχουν γρήγορα μονοπάτια επικοινωνίας, μειώνοντας έτσι την καθυστέρηση διάδοσης και βελτιώνοντας την υλοποίηση του σχεδίου. Για παράδειγμα, η οικογένεια Virtex-7 της Xilinx ομαδοποιεί τέσσερα λογικά κελιά σε ένα slice. Δυο slices και ένα carry-logic σχηματίζουν ένα προγραμματιζόμενο λογικό block (Configurable Logic Block, CLB). Τα λογικά κελιά, συνδέονται με ένα δίκτυο διασυνδέσεων ώστε να παρέχουν υποστήριξη για πιο πολύπλοκες υλοποιήσεις κυκλωμάτων. Αυτό το δίκτυο διασυνδέσεων αποτελείται από switch boxes. Ένα switch box χρησιμοποιείται για τη δρομολόγηση μεταξύ των εισόδων/εξόδων ενός λογικού κελιού με το γενικό δίκτυο δρομολόγησης του τσιπ. Είναι επίσης υπεύθυνο για τη διέλευση σημάτων από ένα τμήμα καλωδίωσης σε ένα άλλο. Τα τμήματα καλωδίωσης μπορεί να είναι είτε μικρά, συνδέοντας μόνο δυο λογικά κελιά είτε μεγάλα, διατρέχοντας όλη την επιφάνεια του τσιπ.



Σχήμα 2.5 Configurable Logic Block

2.3.5 Blocks Εισόδου/Εξόδου

Τα blocks εισόδου/εξόδου (Input/Output Blocks – IOBs) , που βρίσκονται περιμετρικά του τσιπ, συνδέουν τον πινάκα των λογικών blocks και τις πηγές δρομολογήσεων με τους εξωτερικούς ακροδέκτες της συσκευής.

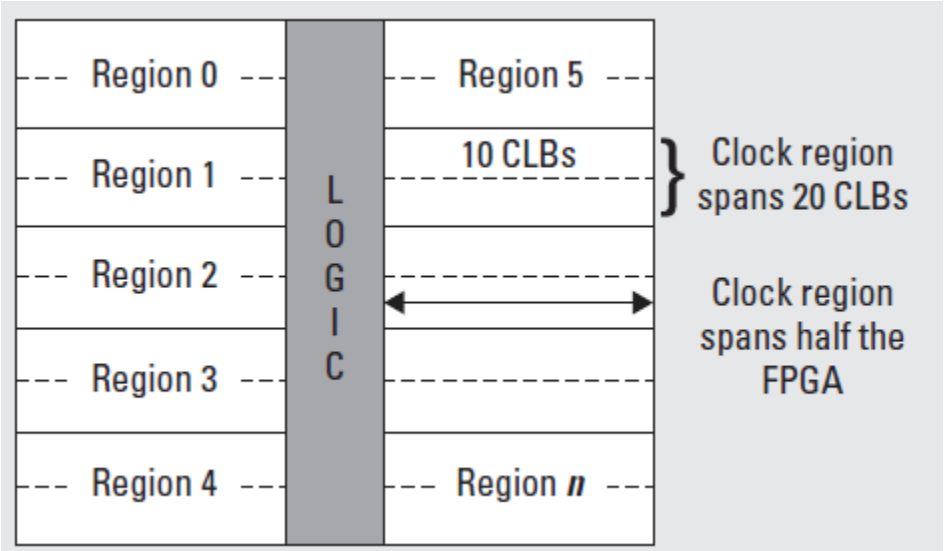
2.3.6 Blocks Ειδικού Σκοπού

Πολλά σχέδια απαιτούν την χρήση κάποιου ποσού της On-Chip μνήμης. Χρησιμοποιώντας λογικά κελιά είναι δυνατή η κατασκευή στοιχείων μνήμης διάφορων μεγεθών. Ωστόσο, όσο αυξάνονται οι απαιτήσεις για μνήμη τόσο μειώνονται οι διαθέσιμοι πόροι. Η λύση βρίσκεται στην ύπαρξη ενός σταθερού ποσού ενσωματωμένης on-chip μνήμης μέσα στο σώμα του FPGA που ονομάζεται **Block Ram** (BRAM). Τοπικές on-chip αποθηκευτικές μονάδες όπως μνήμες RAM, ROM ή Buffers μπορούν να κατασκευαστούν από BRAMs. Οι τελευταίες μπορούν να συνδυαστούν μεταξύ τους ώστε να σχηματίσουν μεγαλύτερες BRAMs. Είναι επίσης μνήμες dual-ported, επιτρέποντας την ανεξάρτητη ανάγνωση και εγγραφή από κάθε port συμπεριλαμβανομένων και των ανεξάρτητων ρολογιών. Πολλές συσκευές FPGA, θέλοντας να επιτρέπουν πιο σύνθετα σχέδια, συμπεριλαμβανομένης της ψηφιακής επεξεργασίας σήματος ή μιας γκάμας πολλαπλασιασμών, συμπεριλαμβάνουν στο σώμα τους Blocks Ψηφιακής Επεξεργασίας Σήματος (Digital Signal Processing Block, DSP). Όπως και με τις BRAMs, είναι δυνατό να υλοποιήσουμε αυτά τα στοιχεία χρησιμοποιώντας την προγραμματιζόμενη λογική, αλλά είναι πιο αποδοτικό από άποψη απόδοσης και κατανάλωσης ισχύος να ενσωματώσουμε πολλά από αυτά τα στοιχεία κατευθείαν στο σώμα του FPGA. Είναι δυνατό να συνδυάσουμε μεταξύ τους DSP Blocks ώστε να εκτελέσουμε μεγαλύτερες πράξεις όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση και τετραγωνική ρίζα floating point αριθμών απλής και διπλής ακρίβειας. Αναμφισβήτητα, μία από

τις πιο σημαντικές προσθήκες στο σώμα των FPGA είναι η προσθήκη ενσωματωμένων επεξεργαστών. Υπάρχει ένας μεγάλος αριθμός διεπαφών για τη διασύνδεση των επεξεργαστών με την προγραμματιζόμενη λογική των FPGA ώστε να είναι δυνατή η επικοινωνία με τους hardware πυρήνες. Τα περισσότερα συστήματα διαθέτουν ένα μόνο εξωτερικό ρολόι που παράγει μια σταθερή συχνότητα ρολογιού. Παρόλα αυτά, υπάρχουν διάφοροι λόγοι για τους οποίους ένας σχεδιαστής μπορεί να επιθυμεί οι hardware πυρήνες του να λειτουργούν σε διαφορετικές συχνότητες. Ο Manager Ψηφιακού Ρολογιού (Digital Clock Manager, DCM), επιτρέπει την παραγωγή διαφορετικών συχνοτήτων ρολογιού από ένα μόνο ρολόι αναφοράς. Είναι δυνατό να διαιρέσουμε το υπάρχων σήμα ρολογιού ώστε να πετύχουμε παλμούς χαμηλότερης συχνότητας. Το πλεονέκτημα της χρήσης των DCMs είναι ότι οι παραγόμενοι παλμοί θα έχουν μικρή απόκλιση από την επιθυμητή τιμή (Jitter) και μια προκαθορισμένη σχέση φάσης.

2.3.7 Ρολόγια

Είναι δυνατόν, διάφοροι hardware πυρήνες να λειτουργούν με διαφορετικές μεταξύ τους συχνότητες. Στον παραδοσιακό σχεδιασμό, κάθε ρολόι παράγεται εκτός του τσιπ και συνδέεται ως είσοδος στο σύστημα. Στις υλοποιήσεις με FPGA είναι δυνατό να δημιουργήσουμε διαφορετικούς ρυθμούς ρολογιών από ένα ή περισσότερα ρολόγια. Ο αριθμός των clock regions στα FPGA Virtex7 κυμαίνεται από 4 σε 24 (Σχήμα 2.2.6). Για να βοηθήσει στον σχεδιασμό, τη χρήση και τη διαχείριση αυτών των ρολογιών η Xilinx χρησιμοποιεί τους Μάνατζερ Ψηφιακού Ρολογιού (Digital Clock Managers, DCMs). Γενικά, ένας DCM παίρνει ένα ρολόι εισόδου και δημιουργεί ένα ρυθμιζόμενο ρολόι εξόδου.



Σχήμα 2.6 Clock regions

Κεφαλαιο 3

Σύνθεση υψηλού επιπέδου για σχεδίαση Hardware

Επισκόπηση

Στο συγκεκριμένο κεφάλαιο αναλύεται η μέθοδος της σύνθεσης υψηλού επιπέδου που χρησιμοποιείται για την σχεδίαση ψηφιακών ολοκληρωμένων κυκλωμάτων. Αρχικά, παρουσιάζονται τα βήματα που απαιτούνται για την πραγματοποίηση της σύνθεσης υψηλού επιπέδου. Στην συνέχεια καταγράφονται μερικές βασικές σχεδιαστικές προκλήσεις της HLS και τέλος αναφέρονται τα σημαντικότερα πλεονεκτήματα της.

3.1 Εισαγωγή

Σκοπός της σύνθεσης είναι να λάβει τις προδιαγραφές της συμπεριφοράς που απαιτείται να έχει το σύστημα καθώς και μια σειρά από περιορισμούς και στόχους που πρέπει να ικανοποιούνται και να βρει μια δομή που να υλοποιεί την επιθυμητή συμπεριφορά, ενώ ταυτόχρονα ικανοποιούνται οι στόχοι και οι περιορισμοί. Με τον όρο συμπεριφορά εννοούμε τον τρόπο με τον οποίο το σύστημα ή οι συνιστώσες του αλληλεπιδρούν με το περιβάλλον τους. Ο όρος δομή αναφέρεται στο σύνολο των αλληλένδετων συνιστωσών που συνθέτουν το σύστημα. Συνήθως υπάρχουν πολλές διαφορετικές δομές που μπορούν να χρησιμοποιηθούν για να υλοποιηθεί μια δεδομένη συμπεριφορά. Ένας από τα σκοπούς της σύνθεσης είναι να βρει τη δομή που ανταποκρίνεται καλύτερα στους περιορισμούς που υπάρχουν, όπως είναι ο περιορισμός στο χρόνο του κύκλου ρολογιού, στην περιοχή ή στην ισχύ που καταναλώνεται, ελαχιστοποιώντας παράλληλα άλλες δαπάνες.

Η σύνθεση ψηλού επιπέδου (High-Level Synthesis) ή διαφορετικά electronic system level synthesis(ESL) είναι μια αυτοματοποιημένη διαδικασία σχεδίασης κατά την οποία μια αλγοριθμική περιγραφή περνάει από το στάδιο της μετάφρασης προκειμένου να δημιουργηθεί hardware που υλοποιεί τον ίδιο αλγόριθμο.

Κατα τη σύνθεση υψηλού επιπέδου ως είσοδοι στο σύστημα δίνονται:

- ο behavioral προσδιορισμός
- σχεσιαστικοί περιορισμοί (κόστος, απόδοση, κατανάλωση ισχύος)
- μια βιβλιοθήκη που δείχνει το υλικό που έχουμε διαθέσιμο

και ως έξοδο παίρνουμε την RTL υλοποίηση καθώς και κάποιες πληροφορίες που αφορούν την γεωμετρία της σχεδίασης. Επίσης, στα σύγχρονα εργαλεία hls η σύνθεση γίνεται σε ANSI C/C++/SystemC.

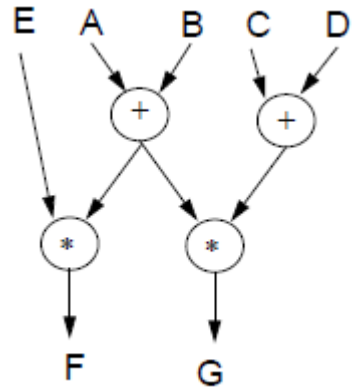
3.2 Διαδικασία HLS

Για την διαδικασία της σύνθεσης υψηλού επιπέδου απαιτούνται καποια βήματα. Αυτά είναι:

- Το πρώτο βήμα στην σύνθεση υψηλού επιπέδου είναι συνήθως η μετατροπή της γλώσσας, στην οποία γράφουμε, σε μια εσωτερική αναπαράσταση. Οι τύποι εσωτερικών αναπαραστάσεων που χρησιμοποιούνται γενικά είναι τα δέντρα αναλυσης και οι γράφοι. Οι περισσότερες προσεγγίσεις χρησιμοποιούν παραλλαγές των γράφων και περιέχουν τόσο τη ροή των δεδομένων όσο και τη ροή του ελέγχου που καθορίζεται από τις προδιαγραφές

```
PROCEDURE Test;
 VAR

A,B,C,D,E,F,G:integer;
 BEGIN
  Read(A,B,C,D,E);
  F := E*(A+B);
  G := (A+B)*(C+D);
  ...
 END;
```

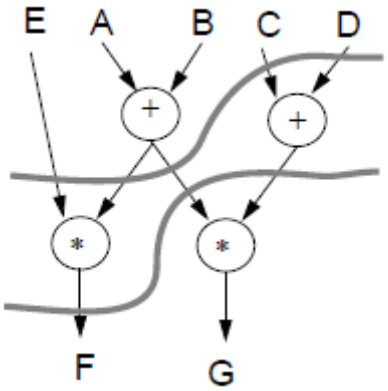


Σχήμα 3.1 Γράφος περιγραφής ροής δεδομένων

Δεδομένου ότι το πρόγραμμα έχει γραφτεί για να είναι πιο ευκολα αναγνώσιμο και όχι για την άμεση μετάφραση σε hardware, σε αυτό το στάδιο είναι σκόπιμο να γίνουν κάποιες βελτιστοποιήσεις της εσωτερικής αναπαράστασης. Αυτοί οι υψηλού επιπέδου μετασχηματισμοί περιλαμβάνουν τέτοιες βελτιστοποιήσεις όπως dead code elimination, constant propagation, common subexpression elimination, inline expansion of procedures και loop unrolling.

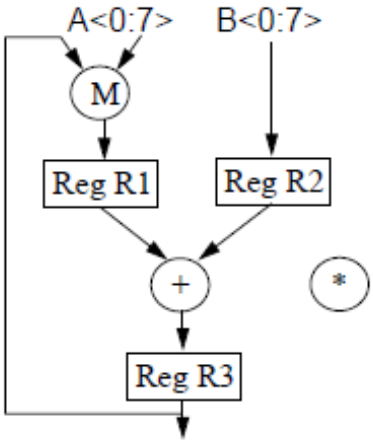
Τα επόμενα δύο βήματα της σύνθεσης είναι ο πυρήνας του μετασχηματισμού της behavioral περιγραφής σε δομική: η χρονοδρομολόγηση και η κατανομή των πόρων. Τα δύο αυτά βήματα είναι αλληλένδετα και αλληλοεξαρτώνται.

- Κατά την χρονοδρομολόγηση καθορίζεται ποιές εργασίες θα τρέξουν σε κάθε κύκλο ρολογιού. Στόχος είναι να ελαχιστοποιηθεί η ποσότητα του χρόνου ή του αριθμό των βημάτων ελέγχου που απαιτούνται για την ολοκλήρωση του προγράμματος, δεδομένων ορισμένων ορίων σχετικά με τους διαθέσιμους πόρους του υλικού.



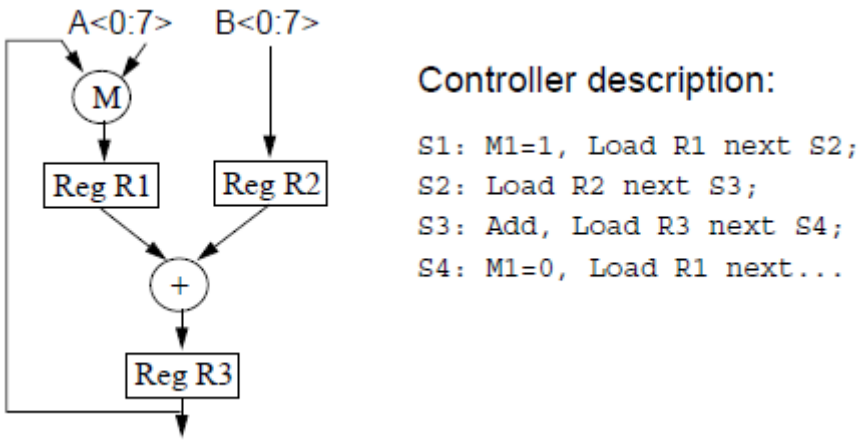
Σχήμα 3.2 Χρονοδρομολογημένος γράφος ροής δεδομένων

- Στην κατανομή των πόρων το πρόβλημα είναι η ελαχιστοποίηση της ποσότητα του hardware που απαιτείται. Το hardware αποτελείται ουσιαστικά από συναρτησιακές μονάδες, στοιχεία μνήμης και μονοπάτια επικοινωνίας(ο καθορισμός των οποίων γίνεται κατά το binding). Η ταυτόχρονη ελαχιστοποίησή τους είναι συνήθως πολύ περίπλοκη, έτσι τα περισσότερα συστήματα ελαχιστοποιούνται χωριστά.



Σχήμα 3.3 Κατανομή Πόρων

- Το επόμενο βήμα μετά την χρονοδρομολόγηση, την κατανομή των πόρων και των καθορισμό των ροών που θα ακολουθήσουν τα δεδομένα είναι η σύνθεση του ελεγκτή που θα οδηγήσει τα μονοπάτια δεδομένων που απαιτούνται από το πρόγραμμα.



Σχήμα 3.4 RTL δομή με περιγραφή ελεγκτή

- Τέλος, η σχεδίαση θα πρέπει να μετατραπεί σε πραγματικό hardware. Για τον σκοπό αυτό χρησιμοποιούνται χαμηλότερου επιπέδου εργαλεία όπως εργαλεία για λογική σύνθεση και σύνθεση του layout.

Συνοπτικά η διαδικασία της σύνθεσης υψηλού επιπέδου αποτελείται απο τα παρακάτω βήματα:

- Λεξική επεξεργασία
- Αλγοριθμική βελτιστοποίηση
- Dataflow ανάλυση
- Κατανομή των πόρων
- Scheduling
- Binding
- Επεξεργασία εξόδου

Τα διάφορα hls εργαλεία κάνουν τα βήματα αυτά με διαφορετική σειρά το καθένα, ανάλογα με τον αλγόριθμο που χρησιμοποιούν. Πολλές φορές μπορεί να γίνει συνδυασμός των παραπάνω βημάτων ή και επανάληψη τους προκειμένου τελικά να συγκλίνουμε στην επιθυμητή υλοποίηση.

3.3 Βασικά θέματα υλοποίησης

Σύμφωνα και με τα παραπάνω μερικά βασικά θέματα τα οποία αποτελούν και θεμελιώδης έννοιες της hls είναι:

- *Χρονοδρομολόγηση*. Κατά την χρονοδρομολόγηση καθορίζεται ποιές διεργασίες θα τρέξουν σε κάθε κύκλο.
- *Κατανομή πόρων.* Επιλογή των τύπων των στοιχείων υλικού και του αριθμού των στοιχείων για κάθε τύπο που πρέπει να περιλαμβάνονται στην τελική εφαρμογή.

- *Binding*, είναι η διαδικασία κατά την οποία καθορίζεται ποιός πόρος υλικού ή ποιός πυρήνας θα χρησιμοποιηθεί για κάθε χρονοδρομολογημένη διεργασία
- *Σύνθεση ελεγκτή*
- *Σχεδιασμός του τρόπου ελέγχου και χρονισμού*.
- *Δυνατότητα παραλληλισμού*

Λαμβάνοντας υπόψιν και τα παραπάνω θέματα, στόχος του High-Level Synthesis είναι η δημιουργία μιας RTL σχεδίασης, η οποία θα έχει την επιθυμητή συμπεριφορά, ενώ παράλληλα θα ικανοποιεί τους σχεδιαστικούς περιορισμούς που έχουν τεθεί και θα βελτιστοποιεί την δοσμένη συνάρτηση κόστους.

3.4 Πλεονεκτήματα

Τα τελευταία χρόνια υπάρχει μια τάση προς την αυτοματοποιημένη διαδικασία της σύνθεσης σε όλο και υψηλότερα επίπεδα της ιεραρχίας του σχεδιασμού. Το ενδιαφέρον αυτό για σύνθεση σε υψηλότερα επίπεδα οφείλεται στα πλεονεκτήματα που προσφέρει η High-Level Synthesis. Κάποια από αυτά είναι:

- *Μικρότερος κύκλος σχεδίασης*. Αν η διαδικασία σχεδίασης είναι αυτοματοποιημένη, τότε η εταιρεία μπορεί να βγάλει την υλοποίησή της στην αγορά πιο γρήγορα και έτσι να είναι περισσότερη ανταγωνιστική στην αγορά για το προϊόν αυτό. Επιπλέον, δεδομένου ότι μεγάλο μέρος του κόστους του τσιπ οφείλεται στη σχεδίαση, αυτοματοποίηση της διαδικασίας μπορεί να μειώσει το κόστος σημαντικά.

- *Λιγότερα λάθη*. Εάν η διαδικασία σύνθεσης μπορεί να ελεγχθεί ευκολότερα ότι είναι ορθή, τότε υπάρχει μεγαλύτερη σιγουριά ότι το τελικό σχέδιο θα ανταποκρίνεται στις αρχικές προδιαγραφές. Αυτό θα σημαίνει λιγότερα λάθη και λιγότερος χρόνος debugging για τα νέα τσιπ.

- *Δυνατότητα για διερεύνηση του χώρου σχεδίασης*. Ένα καλό σύστημα σύνθεσης μπορεί να παράγει διάφορα σχέδια για τις ίδιες προδιαγραφές σε ένα εύλογο χρονικό διάστημα. Αυτό επιτρέπει στον προγραμματιστή να διερευνήσει τα διάφορα trade-offs μεταξύ κόστους, ταχύτητας και κατανάλωσης ισχύος και έτσι , ή να λάβει ένα υπάρχον σχέδιο και να παράγει μια υλοποίηση με την ίδια λειτουργικότητα που είναι όμως ταχύτερη ή λιγότερο ακριβή.

- *Διαθεσιμότητα της τεχνολογίας των IC σε περισσότερους ανθρώπους*. Καθώς όλο και περισσότεροι εμπειρογνωμοσύνη του σχεδιασμού μετακινείται στο σύστημα σύνθεσης, γίνεται ευκολότερο για έναν μη ειδικό να σχεδιάσει ένα τσιπ που που θα έχει δεδομένο σύνολο προδιαγραφών.

ΚΕΦΑΛΑΙΟ 4

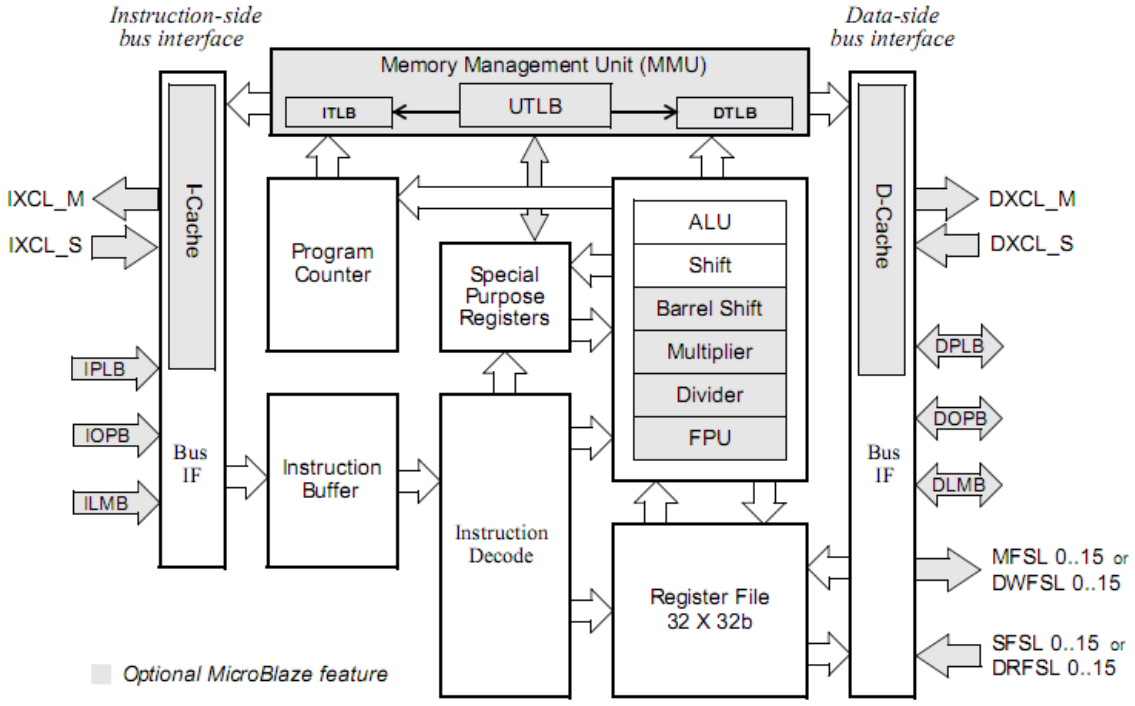
Σχεδίαση με χρήση του συνεπεξεργαστή Microblaze™

Επισκόπηση

Στο συγκεκριμένο κεφάλαιο αναλύεται ο software επεξεργαστής Microblaze που θα χρησιμοποιηθεί για το συσχεδιασμό HW/SW και περιγράφεται η αρχιτεκτονική της συσχεδίασης.

4.1 Ο επεξεργαστής Microblaze

Ο επεξεργαστής MicroBlaze ™ είναι ένας υπολογιστής μειωμένου instruction set (RISC), βελτιστοποιημένος για υλοποίηση σε Xilinx ® Field Programmable Gate Arrays (FPGAs). Το σχήμα δείχνει ένα λειτουργικό διάγραμμα του πυρήνα MicroBlaze.



Σχήμα 4.1 Μπλοκ διάγραμμα του επεξεργαστή Microblaze

4.2 Χαρακτηριστικά Microblaze

Ο soft core επεξεργαστής MicroBlaze είναι διαμορφώσιμος σε μεγάλο βαθμό, επιτρέποντας στον χρήστη να επιλέξει ανάμεσα σε ένα σύνολο χαρακτηριστικών που απαιτούνται για την σχεδίαση. Το σύνολο των χαρακτηριστικών αυτών περιλαμβάνει:

- Τριάντα δύο 32-bit γενικού σκοπού επεξεργαστές
- 32-bit λέξη εντολής με τρεις τελεστές και δύο τρόπους διευθυνσιοδότησης
- 32-bit δίαυλο διευθύνσεων
- single issue pipeline

4.3 Αρχιτεκτονική Μνήμης

Ο MicroBlaze υλοποιείται με αρχιτεκτονική μνήμης Harvard. Αυτό σημαίνει ότι η πρόσβαση σε εντολές και δεδομένα γίνεται σε ξεχωριστούς χώρους διευθύνσεων. Κάθε χώρος διευθύνσεων είναι 32-bit (δηλαδή, χειρίζεται μέχρι 4 GB για μνήμη εντολών και δεδομένων αντίστοιχα). Οι μνήμες εντολών και δεδομένων μπορεί να επικαλύπτονται με την απεικόνισή τους στην ίδια φυσική μνήμη. Αυτό είναι χρήσιμο για τον εντοπισμό σφαλμάτων του λογισμικού. Και οι δύο διεπαφές εντολών και δεδομένων στον MicroBlaze είναι πλάτους 32 bits και χρησιμοποιούν big endian, bit-reversed μορφή. Επίσης, υποστηρίζει προσβάσεις λέξης, μισή λέξης, και byte στα δεδομένα της μνήμης.

Ο MicroBlaze δεν διαχωρίζει τις προσβάσεις δεδομένων σε I / O και μνήμης (χρησιμοποιεί απεικονισμένα σε μνήμη I / O). Επίσης διαθέτει έως και τρεις διεπαφές για προσπελάσεις μνήμης:

- Local Memory Bus (LMB)
- Processor Local Bus (PLB) ή On-Chip Peripheral Bus (OPB)
- Xilinx CacheLink (XCL)

Η LMB περιοχή διευθύνσεων μνήμης δεν πρέπει να επικαλύπτεται με τις αντίστοιχες των PLB, OPB ή XCL. Ο MicroBlaze έχει latency ενός κύκλου για προσβάσεις στην τοπική μνήμη (LMB) και για cache read-hits, εκτός και αν είναι ενεργοποιημένη η βελτιστοποίηση ως προς την περιοχή που καταλαμβάνει η υλοποίηση όπου οι προσβάσεις δεδομένων και τα read hits στην cache χρειάζονται δύο κύκλους ρολογιού. Συνήθως ένα write στην data cache έχει latency δύο κύκλων ρολογιού.

Στον MicroBlaze οι κρυφές μνήμες εντολών και δεδομένων μπορούν να ρυθμιστεί ώστε να γίνεται χρήση 4 ή 8 λέξεων σε κάθε γραμμή cache. Όταν χρησιμοποιείται μεγαλύτερη γραμμή cache, φέρνουμε περισσότερα bytes, γεγονός που βελτιώνει γενικά την απόδοση για το λογισμικό όταν χρησιμοποιούνται πρότυπα

ακολουθιακών προσβάσεων. Ωστόσο, για λογισμικό με ένα πιο τυχαίο πρότυπο πρόσβασης στη μνήμη η απόδοση μπορεί να μειωθεί, για δεδομένο μέγεθος της cache. Αυτή η μείωση οφείλεται σε μειωμένο cache hit rate λόγω των λιγότερων διαθέσιμες γραμμών cache.

4.4 Δίαυλοι Επικοινωνίας

Ο πυρήνας MicroBlaze είναι οργανωμένος ως μια αρχιτεκτονική Harvard με ξεχωριστούς δίαυλους επικοινωνίας για τα δεδομένα και τις εντολές. Οι τρεις διεπαφές μνήμης που υποστηρίζονται είναι:

- Local Memory Bus (LMB)
- IBM Processor Local Bus (PLB) ή IBM On-chip Peripheral Bus (OPB)
- Xilinx® CacheLink (XCL)

Η LMB παρέχει ενός κύκλου πρόσβαση σε on-chip RAM διπλής θύρας. Οι διεπαφές PLB και OPB παρέχουν σύνδεση τόσο σε on-chip όσο και off-chip περιφερειακά και μνήμες. Η διεπαφή CacheLink προορίζεται για χρήση με εξειδικευμένους εξωτερικούς ελεγκτές μνήμης. Επίσης ο MicroBlaze υποστηρίζει έως 16 Fast Simplex Link (FSL) θύρες, κάθε μία με μία master και μία slave FSL διεπαφή.

4.4.1 Processor Local Bus (PLB)

Ο 128-bit δίαυλοςProcessor Local Bus (PLB) v4.6 παρέχει την υποδομή για το δίαυλο ώστε να συνδέονται PLB masters και slaves σε ένα γενικό σύστημα PLB. Αποτελείται από μια μονάδα ελέγχου bus, ένα timer, ξεχωριστή διεύθυνση, για να γράφει, και να διαβάζει τις μονάδες διόδου δεδομένων, καθώς και μια προαιρετική διεπαφή slave DCR (Device Control Register) για να παρέχει πρόσβαση σε καταχωρητές κατάστασης σφαλμάτων στο δίαυλο.

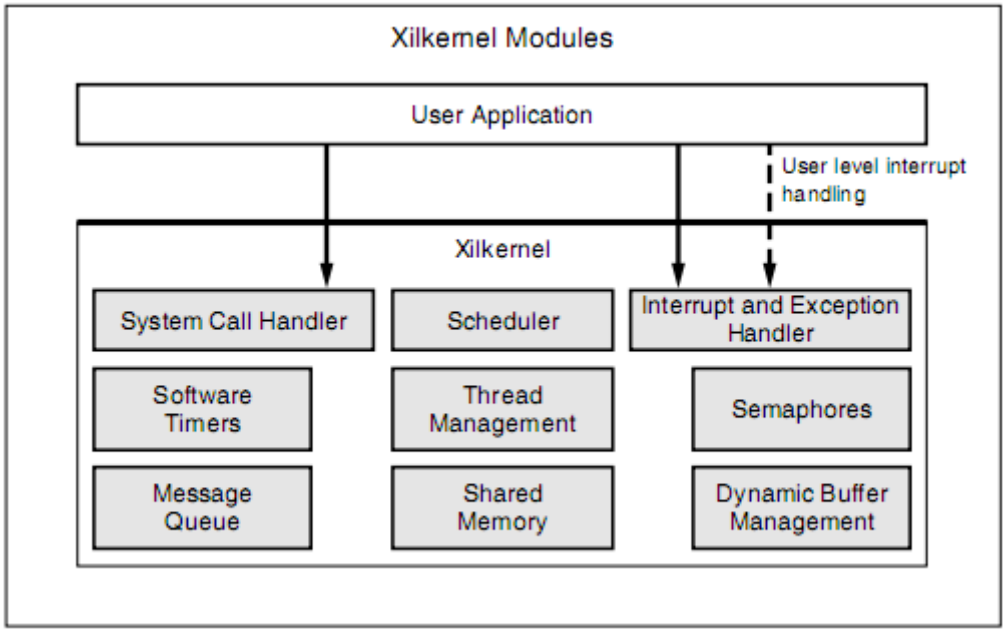
4.4.2 Local Memory Bus(LMB)

Ο LMB είναι ένας σύγχρονος δίαυλος που χρησιμοποιείται κυρίως για την πρόσβαση σε μπλοκ on-chip μνήμης RAM. Χρησιμοποιεί έναν ελάχιστο αριθμό σημάτων ελέγχου και ένα απλό πρωτόκολλο για να εξασφαλίσει ότι η πρόσβαση σε τοπικές RAM θα γίνει σε ένα κύκλο ρολογιού. Όλα τα LMB σήματα είναι ενεργά στο high.

4.4.3 Το λειτουργικό σύστημα Xilkernel

Ο Xilkernel είναι ένας μικρός και στιβαρός πυρήνας. Είναι σε μεγάλο βαθμό ενσωματωμένος με το Platform Studio framework και αποτελεί μια ελεύθερη βιβλιοθήκη λογισμικού που δίνεται μαζί με το Xilinx EDK. Επιτρέπει πολύ υψηλό βαθμό παραμετροποίησης, δίνοντας στους χρήστες την δυνατότητα να προσαρμόσουν τον πυρήνα σε ένα βέλτιστο επίπεδο, τόσο όσον αφορά το μέγεθος

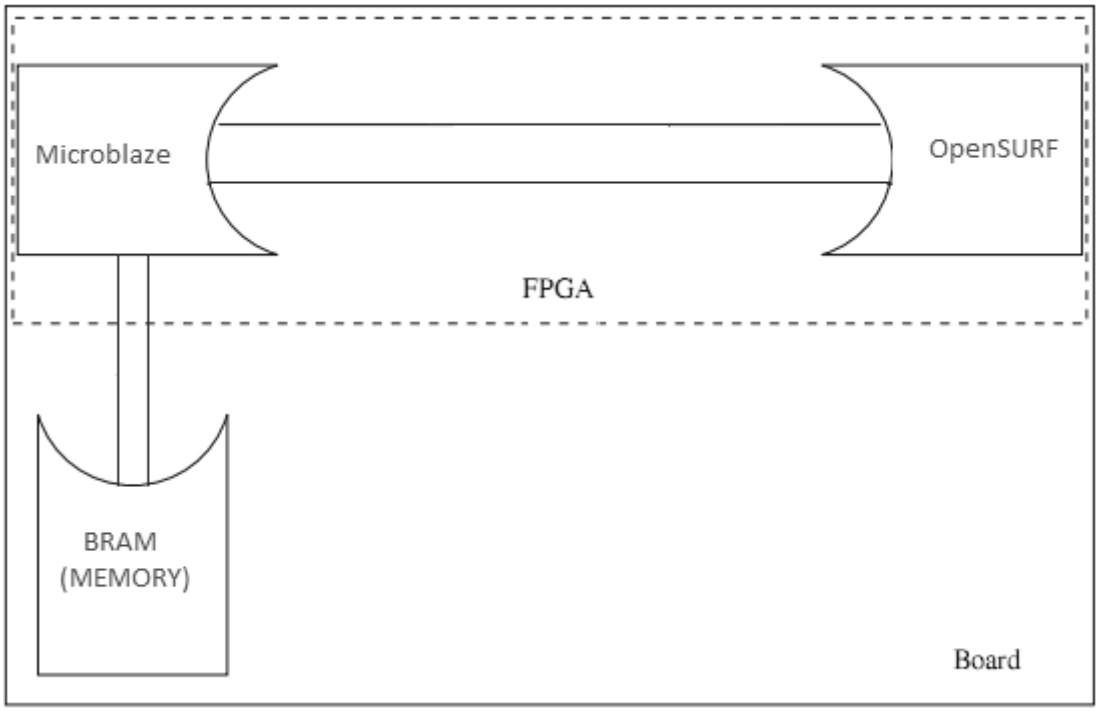
όσο και τη λειτουργικότητα. Υποστηρίζει τα βασικά χαρακτηριστικά που απαιτούνται σε ένα ενσωματωμένο soft πυρήνα, με ένα POSIX API. Ο Xilkernel λειτουργεί τόσο για MicroBlaze ™ και όσο και για PowerPC ™ 405 επεξεργαστές. Οι υπηρεσίες Xilkernel IPC μπορούν να χρησιμοποιηθούν για την υλοποίηση υπηρεσιών υψηλότερου επίπεδου (όπως δικτύωσης, βίντεο και ήχου) και στη συνέχεια μπορούμε να τρέξουμε τις εφαρμογές που χρησιμοποιούν αυτές τις υπηρεσίες. Στην παρακάτω εικόνα φαίνονται τα στοιχεία του Xilkernel:



Σχήμα 4.2 Xilinx kernel modules

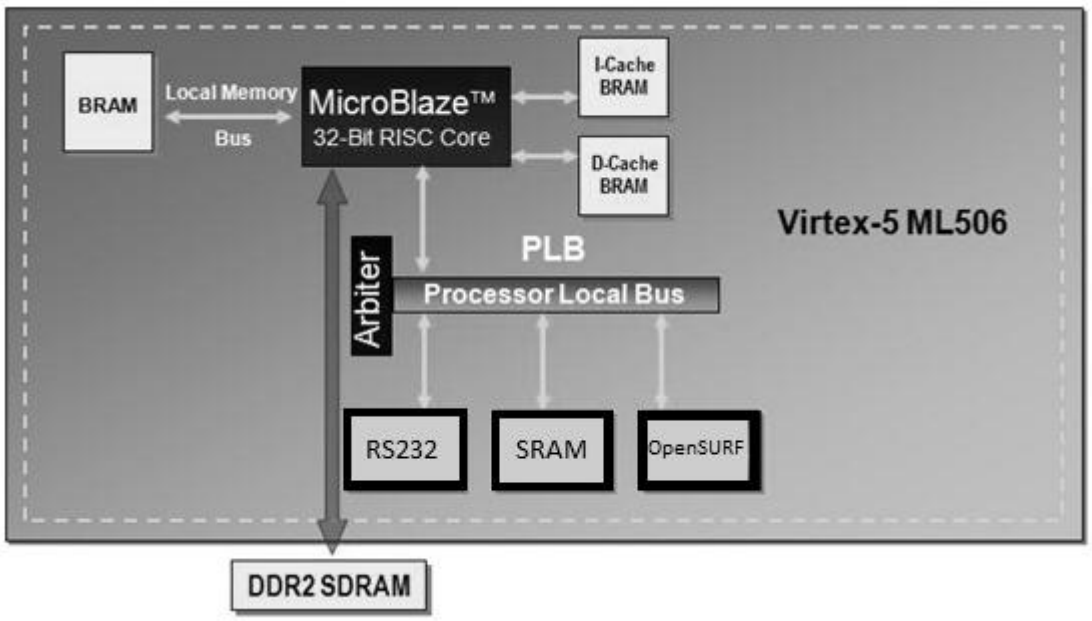
4.5 Αρχιτεκτονική Συσχεδίασης

Η αρχιτεκτονική του συστήματος αποφασίζεται μετά το διαχωρισμό HW / SW . Ο Microblaze επιλέγεται ως ο επεξεργαστήςπου θα τρέξει το software κομμάτι του αλγορίθμου OpenSURF και στο FPGA υλοποιούμε τον πυρήνα που θα τρέχει τις HW λειτουργίες. Το παρακάτω σχήμα απεικονίζει ένα μοντέλο αρχιτεκτονικής του συστήματος. Στο σχήμα δεν φαίνονται οι δίαυλοι επικοινωνίας:



Σχήμα 4.3 Μοντέλο αρχιτεκτονικής

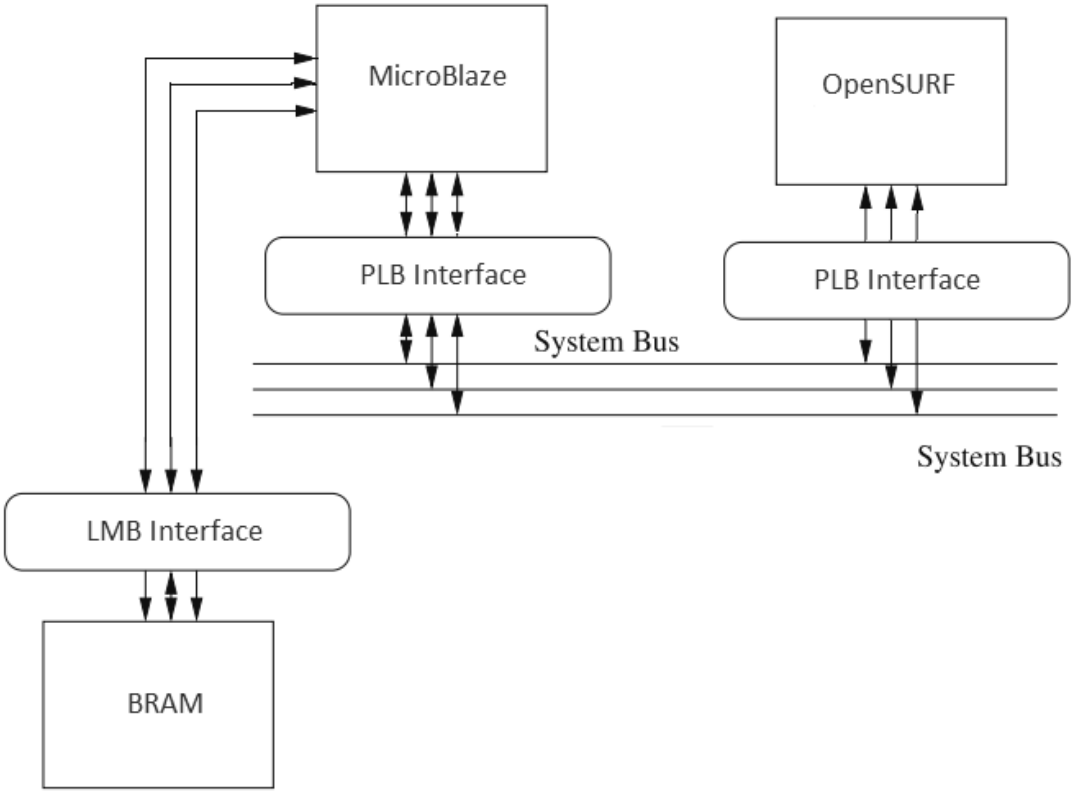
Το τελικό System on Chip (SoC) περιλαμβάνει μία SRAM για την αποθήκευση των εικόνων εισόδου και εξόδου, μία DDR2 SDRAM, προκειμένου να αποθηκεύεται το εκτελέσιμο αρχείο ELF (1,3 MB), και μια RS232 UART σειριακή θύρα για την αποστολή της εικόνας εξόδου στον υπολογιστή host. Το τελικό διάγραμμα είναι:



Σχήμα 4.4 τελικό SoC

4.5.1 Πρωτόκολλο Επικοινωνίας

Το σχήμα 4.4 δείχνει το μοντέλο επικοινωνίας μετά την εισαγωγή των πρωτοκόλλων επικοινωνίας. Τα πρωτόκολλα Local Memory Bus (LMB) and Processor Local Bus(PLB) πρωτόκολλα, που παρουσιάστηκαν προηγουμένως, χρησιμοποιούνται σε αυτή τη σχεδίαση. Ο Microblaze soft επεξεργαστής, ο πυρήνας που υλοποιήσαμε και η BRAM συνδέονται μέσω του διαύλου του συστήματος. Όλα τα εξαρτήματα που είναι συνδεδεμένα με τον ίδιο δίαυλο χρονίζονται με την ίδια ταχύτητα. Οι διασυνδέσεις παρεμβάλλονται μεταξύ του διαύλου και των στοιχείων του. Το PLB είναι το πρωτόκολλο επικοινωνίας μεταξύ του Microblaze και του διαύλου του συστήματος, ενώ η επικοινωνία μεταξύ BRAM και Microblaze γίνεται με το LMB. Γίνεται επικοινωνία μεταξύ των συνιστωσών του συστήματος για την διασφάλιση της επιτυχούς ολοκλήρωσης της μεταφοράς των δεδομένων.

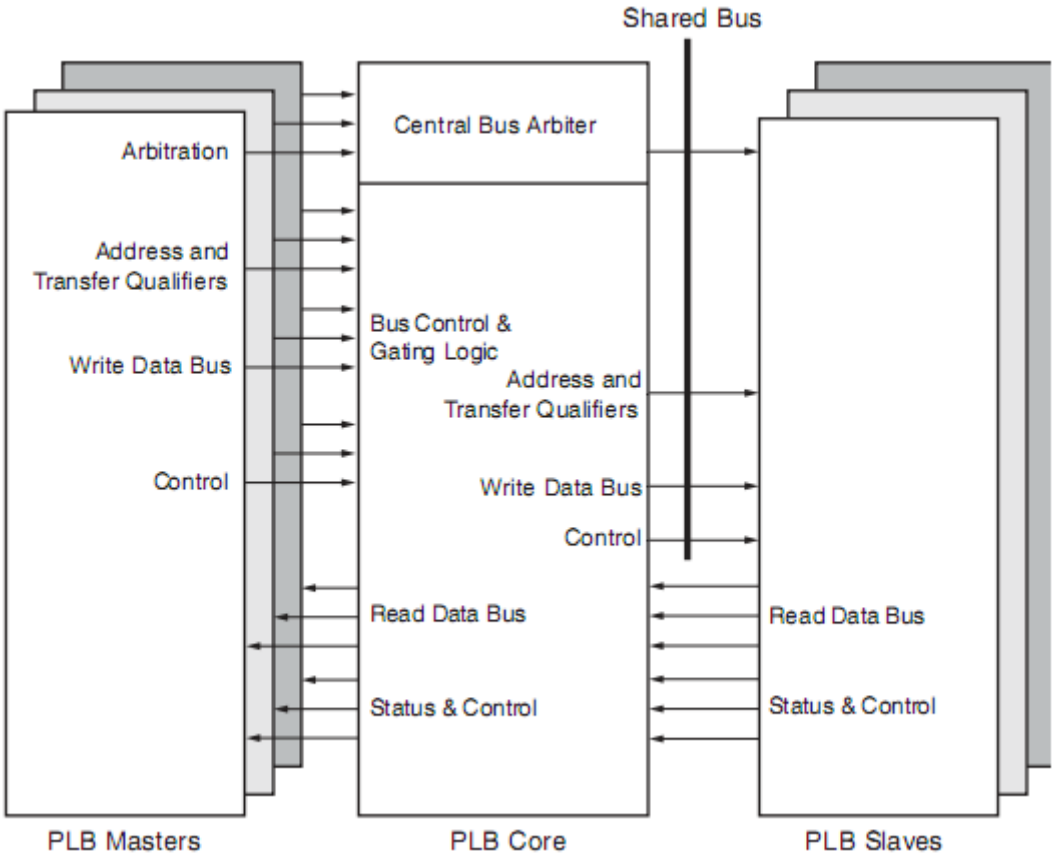


Σχήμα 4.5 Μοντέλο επικοινωνίας

4.5.2 Διεπαφή Microblaze/OpenSURF

Η επικοινωνία μεταξύ του Microblaze soft επεξεργαστή και του πυρήνα OpenSURF (η οποία είναι μια εξωτερική συσκευή) βασίζεται στο PLB πρωτόκολλο της πλατφόρμας Xilinx Virtex-6. Το πρωτόκολλο έχει 2 αναπαραστάσεις: master και slave πρωτόκολλα διαύλων. Τα περιφερειακά PLB δημιουργήθηκαν για να δουλεύουν ως master ή ως slave για το PLB. Ένα περιφερειακό συνδεδεμένο στις θύρες του master του PLB σπρώχνει τα δεδομένα και τα σήματα ελέγχου στο

δίαυλο, ενώ ένα περιφερειακό που συνδέεται με θύρες του slave διαβάζει και εξάγει δεδομένα και σήματα ελέγχου από το PLB. Η ιδέα πίσω από το σύστημα των διαύλων PLB φαίνεται στο Σχήμα 3.5 με ένα παράδειγμα PLB συνδέσεων σε ένα σύστημα με τρεις master και τρεις slave.



Σχήμα 4.6 PLB interface

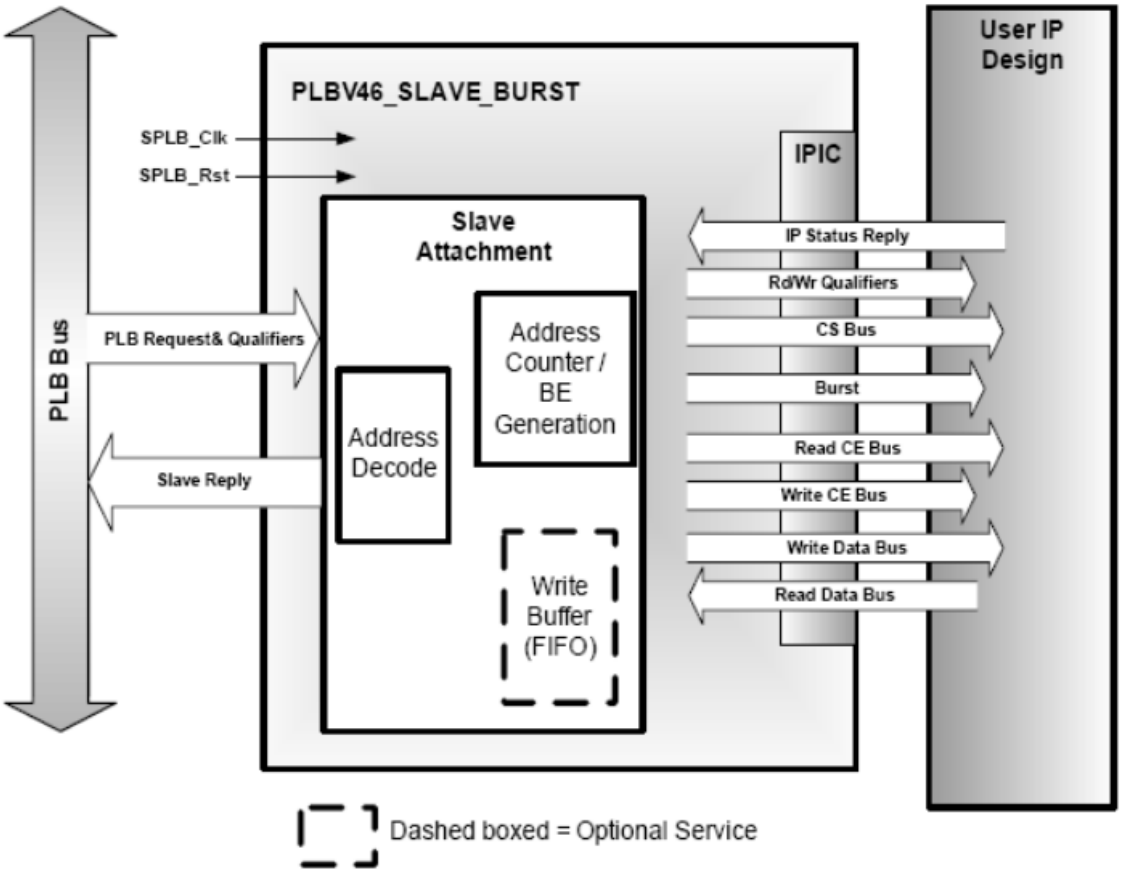
4.5.3 Πρωτόκολλο διαύλου για master

Το πρωτόκολλο PLB που χρησιμοποιήθηκε στα πλαίσια της εργασίας είναι πολύ περίπλοκο για να παρουσιαστεί ένας λεπτομερής κατάλογος των βασικών σημάτων διαύλου master που συνδέει το Microblaze επεξεργαστή στο διαύλο του συστήματος.

4.5.4 Πρωτόκολλο διαύλου για slave

Σε αντίθεση με τα σήματα διαύλου πλοίαρχο για Microblaze, ένα υποσύνολο των σημάτων διαύλου σκλάβος μπορεί να παρουσιαστεί και εξακολουθούν να παρέχουν μια σαφή και απλή άποψη για το πώς η εξωτερική συσκευή (IDWT) επικοινωνεί μέσω του PLB με Microblaze. Το EDK χρησιμοποιεί PLB slave και burst περιφερειακά για την υλοποίηση των κοινών λειτουργιών μεταξύ των διαφόρων περιφερειακών του επεξεργαστή. Ο PLB slave και τα περιφερειακά αυτά λειτουργούν ως δίαυλοι master ή δίαυλοι slave και παρέχουν ένα σύνολο απλουστευμένων πρωτοκόλλων

διαύλου. Στο σχήμα 3.6 απεικονίζεται η επικοινωνία μεταξύ του bus, ένός απλού slave PLB περιφερειακού και του υλοποιημένου πυρήνα IP.



Σχήμα 4.7 Συνιστώσα PLB slave

Χρησιμοποιώντας το πρωτόκολλο αυτό ο IP πυρήνας που υλοποιήσαμε είναι δυνατόν να ανταλλάζει δεδομένα με τον συνεπεξεργαστή Microblaze και να συνδέεται στα κατάλληλα σήματα ελέγχου με τα οποία ξεκινάει η επεξεργασία ή σηματοδοτείται το τέλος της επεξεργασίας από τον πυρήνα.

Κεφάλαιο 5

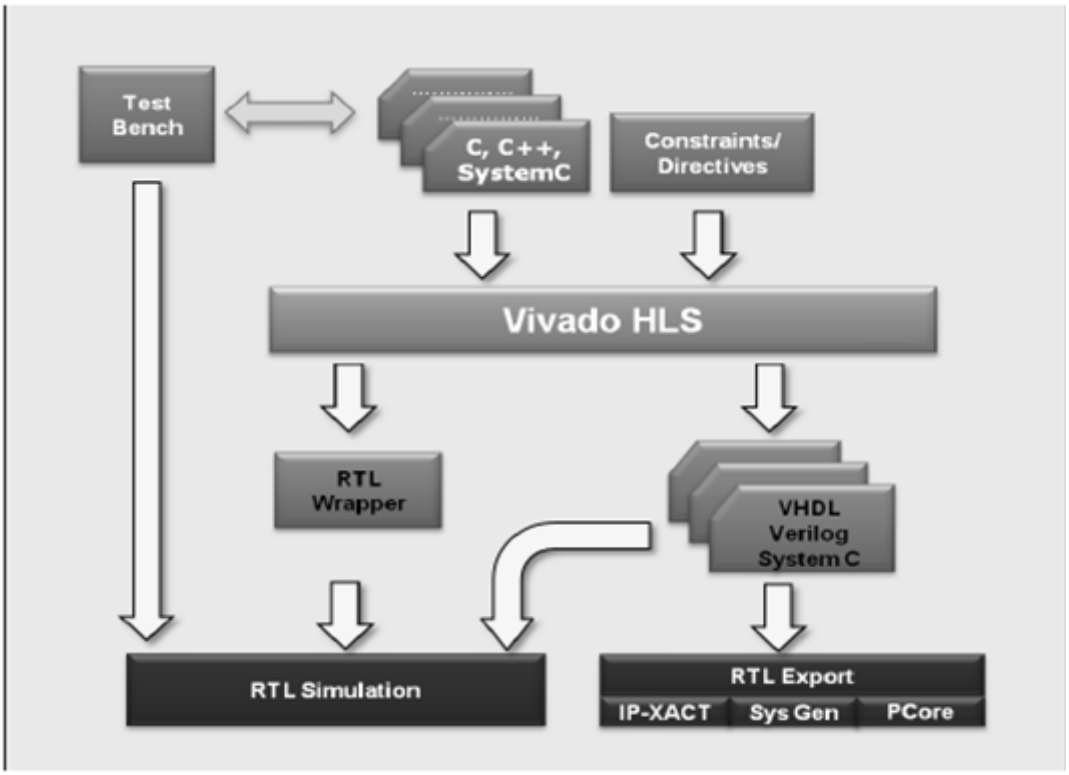
Xilinx® Vivado HLS

Επισκόπηση

Στο συγκεκριμένο κεφάλαιο αναλύεται το εργαλείο Vivado HLS της Xilinx, το οποίο χρησιμοποιείται στα πλαίσια της διπλωματικής εργασίας για την High-Level Synthesis του αλγορίθμου OpenSURF.

5.1 Εισαγωγή

High level synthesis είναι η αυτοματοποιημένη μετατροπή μιας σχεδίασης από C, C++ ή SystemC σε RTL υλοποίηση η οποία με τη σειρά της μπορεί να γίνει synthesized σ' ένα fpga. Στα πλαίσια της συγκεκριμένης διπλωματικής κάνουμε hls σε C , με χρήση του Xilinx Vivado HLS tool. Η σύνθεση υψηλού επιπέδου δέχεται ως είσοδο μια περιγραφή σχεδίασης σε C, μαζί με κάποιους περιορισμούς οι οποίοι επιλέγονται από τον χρήστη. Η έξοδος είναι RTL αρχεία σχεδίασης σε Verilog, VHDL ή SystemC. Η παρακάτω εικόνα δείχνει το μοντέλο της hls στο Vivado:



Σχήμα 5.1 Μοντέλο HLS στο Vivado

5.1.1 Επισκόπηση της HLS

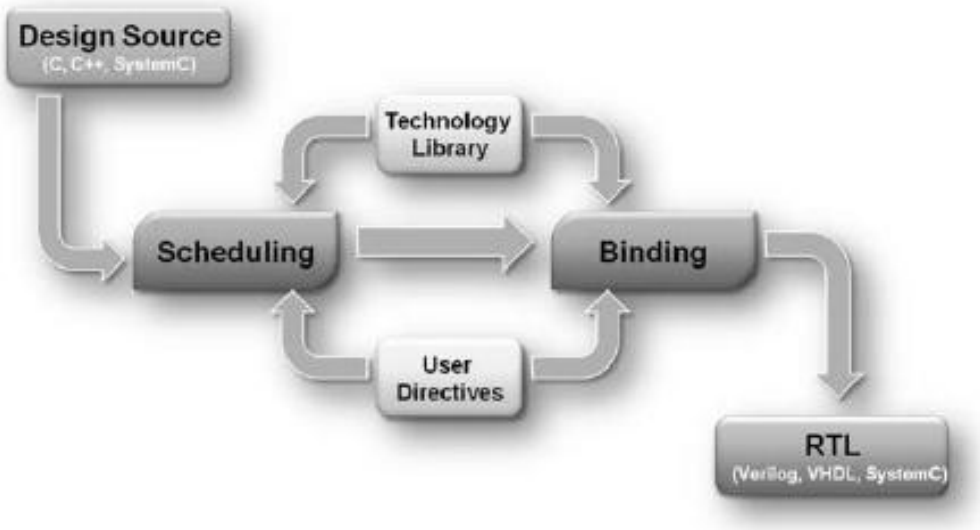
Όταν πραγματοποιείται High-Level Synthesis με το Vivado για τον μετασχηματισμό πηγαίου κώδικα από C σε RTL περιγραφή, η επεξεργασία του πηγαίου κώδικα γίνεται σε μια εσωτερική βάση δεδομένων η οποία περιλαμβάνει τελεστές. Με τους τελεστές αυτούς καλύπτεται όλο το εύρος πράξεων που μπορούμε να κάνουμε στη C, όπως προσθέσεις, shifts, πολλαπλασιασμούς, bit-slicing και πρόσβαση σε πίνακες. Το Vivado κάνει χρήση αυτής της εσωτερικής βάσης δεδομένων κατά την σύνθεση της σχεδίασης. Κατά την διαδικασία της σχεδίασης τα δύο βασικά βήματα είναι το scheduling και το binding.

- Scheduling: καθορίζει σε πόσους κύκλους θα πραγματοποιηθεί μια πράξη.

- Binding: Όταν ολοκληρωθεί το scheduling ,γίνεται το binding το οποίο είναι η διαδικασία κατά την οποία οι χρονοδρομολογημένες πράξεις υλοποιούνται στο διαθέσιμο hardware.

Κατα την διαδικασία του scheduling επηρεάζεται και η ποσότητα των πόρων που έχουμε διαθέσιμους για το binding, ώστε όταν κάνουμε το binding να μην χρειάζεται να επαναλάβουμε το scheduling.Η ποσότητα και ο τύπος των διαθέσιμων πόρων εξαρτάται από το device που επιλέγουμε.

Με βάση τα παραπανω η διαδικασία της σύνθεσης μπορεί να απεικονιστεί ως εξής:



Σχήμα 5.2 Διαδικασία HLS

5.2 Επαλήθευση της ορθής λειτουργίας της υλοποίησης

Η επαλήθευση της ορθής λειτουργίας μιας HLS ροής μπορεί να χωριστεί σε δύο κομμάτια. Στην pre-synthesis επαλήθευση (C προσομοίωση) όπου επαληθεύεται ότι το πρόγραμμα που έχουμε γράψει σε C έχει την επιθυμητή λειτουργία και στην post-synthesis επαλήθευση (RTL προσομοίωση) όπου επαληθεύεται η ορθή λειτουργία της RTL σχεδίασης.

Η επαλήθευση της ορθής λειτουργίας της συνάρτησης, που πρόκειται να γίνει synthesize και την οποία έχουμε γράψει σε C, γίνεται με την χρήση ενός test bench. Προκειμένου να υπάρχει καλός διαχωρισμός μεταξύ σύνθεσης και προσομοίωσης, είναι καλό το test bench και η συνάρτηση να βρίσκονται σε χωριστά αρχεία. Το αποτέλεσμα της σύνθεσης υψηλού επιπέδου είναι η rtl σχεδίαση. Η επαλήθευση της μπορεί να γίνει μέσω του Vivado hls αυτοματοποιημένα, κάνοντας χρήση του pre-synthesis test bench.

Πριν την σύνθεση, ο κώδικας C πρέπει να γίνει compile. Με hls δημιουργούμε το rtl αρχείο του οποίου η λειτουργικότητα θα πρέπει να είναι ίδια με αυτή του μεταγλωττισμένου προγράμματος που μας δίνει η έκδοση του compiler που χρησιμοποιούμε και την ίδια έκδοση χρησιμοποιούμε και για την προσομοίωση του rtl αρχείου με το test bench. Ως compiler μπορούμε να χρησιμοποιήσουμε διάφορες εκδόσεις του gcc, τον Microsoft Visual Studio Compiler ή τον apcc που είναι ενσωματωμένος στο Vivado hls και μας δίνει την δυνατότητα να χρησιμοποιήσουμε αυθαίρετης ακρίβειας ακεραίους.

5.3 Βελτιστοποιήσεις Κώδικα

Προκειμένου η τελική σχεδίαση να ικανοποιεί τους περιορισμούς που έχουμε θέσει, συνήθως χρειάζεται να κάνουμε κάποιες βελτιστοποιήσεις αυτές μπορεί να είναι:

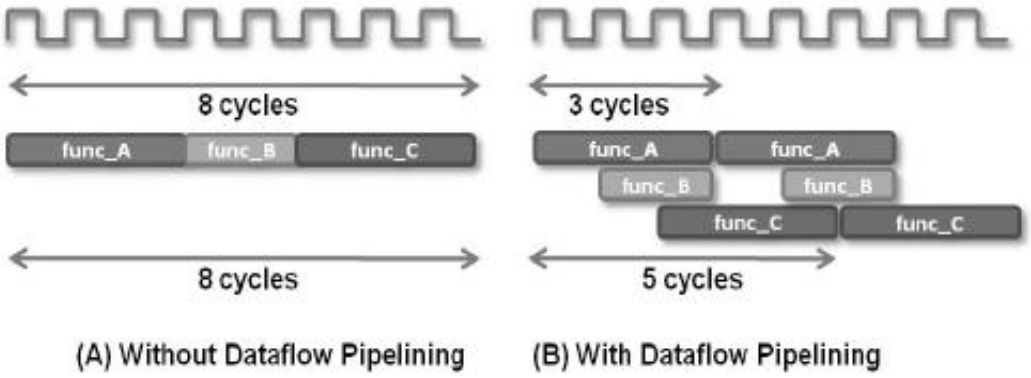
5.3.1 Συναρτησιακές Βελτιστοποιήσεις

Το εργαλείο Vivado hls μας δίνει την δυνατότητα για κάποια optimizations πάνω σε συναρτήσεις. Τα optimizations αυτά είναι:

- *Inline*: Υπάρχει επιβάρυνση σε κύκλους ρολογιού για την είσοδο και την έξοδο συναρτήσεων, οπότε κάνοντας άρση της ιεραρχίας των συναρτήσεων μπορεί να βελτιωθεί το latency και το throughput.

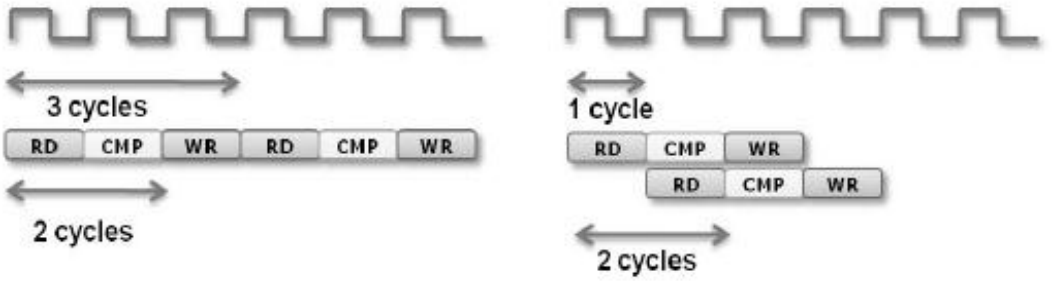
- *Instatiate*: Δίνει την δυνατότητα για βελτιστοποίηση των συναρτήσεων τοπικά.

- *Dataflow pipeline*: Αυτός ο μετασχηματισμός παίρνει μια ακολουθιακή περιγραφή συνάρτησης και δημιουργεί μια αρχιτεκτονική της η οποία έχει την δυνατότητα να τρέχει παράλληλα.Αυτός ο μετασχηματισμός αυξάνει συνήθως το hardware ωστόσο βελτιώνει το throughput και το latency της σχεδίασης.



Σχήμα 5.3 Dataflow pipeline συνάρτησης

- *Pipeline*: Επιτρέπει την παράλληλη εκτέλεση επιμέρους λειτουργιών της συνάρτησης.Και σε αυτήν την περίπτωση έχουμε βελτίωση του throughput.



Σχήμα 5.4 Pipeline Συνάρτησης

- *Latency*: Μας δίνεται η δυνατότητα να θέσουμε ένα ανώτατο και ένα κατώτατο όριο latency για την συνάρτηση.

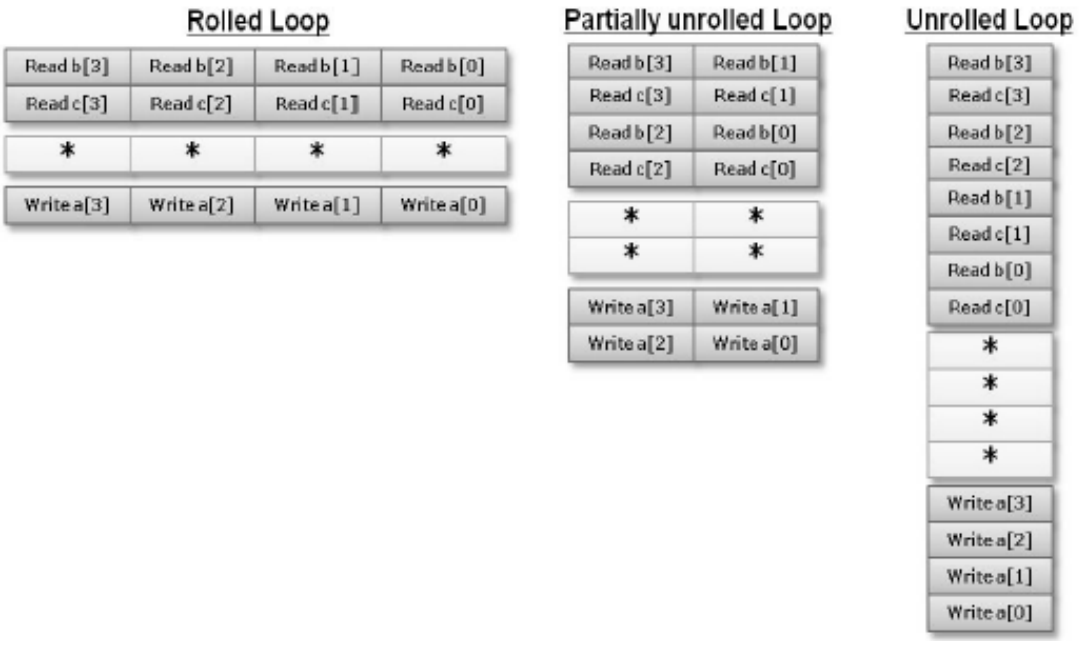
- *Interface*: Δίνεται η δυνατότητα μαζί με το synthesize και για αυτόματη δημιουργία πρωτοκόλλου διεπαφής ώστε να είναι δυνατή η πρόσβαση στους θύρες εισόδου και εξόδου της τελικής RTL σχεδίασης.

5.3.2 Βελτιστοποιήσεις Επαναληπτικών Βρόχων

Μέσα στις συναρτήσεις, οι περιγραφές σε γλώσσα C πολλές φορές υλοποιούνται ως ένα σύνολο επαναληπτικών βρόχων. Η κατανόηση του πώς υλοποιούνται οι επαναληπτικοί αυτοί βρόχοι σε hls, πώς βελτιστοποιούνται και της επίδρασης που έχει η ιεραρχία με την οποία γίνονται οι επαναλήψεις είναι πολύ σημαντική στο να πετύχουμε την βέλτιστη απόδοση σε επίπεδο rtl. Το Vivado hls μας δίνει την δυνατότητα να εφαρμόσουμε τους παρακάτω μετασχηματισμούς στους επαναληπτικούς βρόχους:

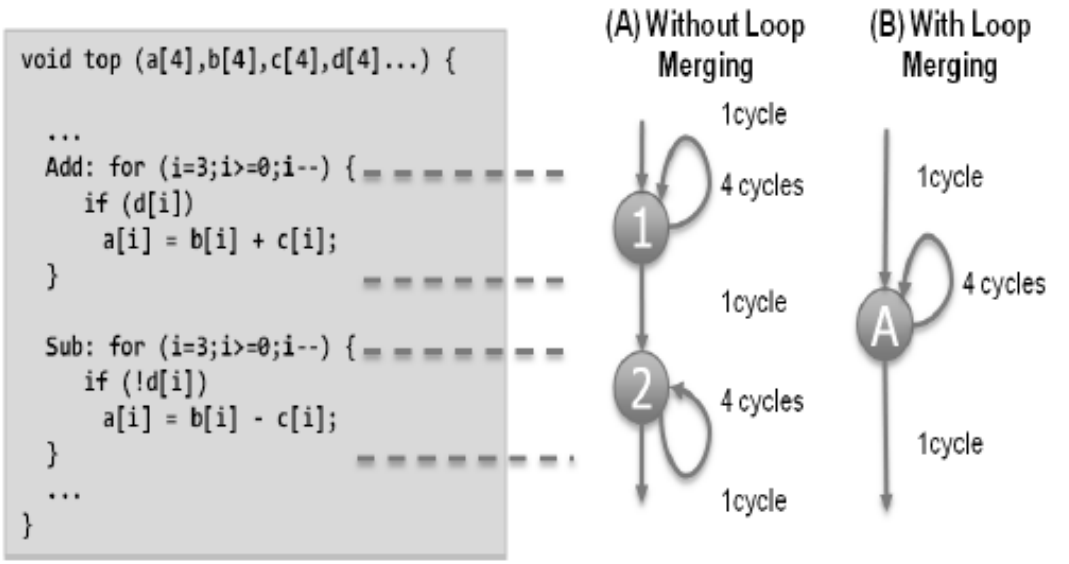
- *Unrolling*: Από προεπιλογή στο High-Level Synthesis οι βρόχοι αντιμετωπίζονται ως μια ενιαία οντότητα, δηλαδή όλες οι εργασίες στο

βρόχο υλοποιούνται χρησιμοποιώντας τους ίδιους πόρους υλικού για την επανάληψη του βρόχου. Το Vivado hls παρέχει τη δυνατότητα να "ξεδιπλώνονται" εν μέρει ή ολικά οι επαναληπτικοί βρόχοι, γεγονός που μπορεί να αυξήσει πολύ την απόδοση.



Σχήμα 5.5 Loop unrolling

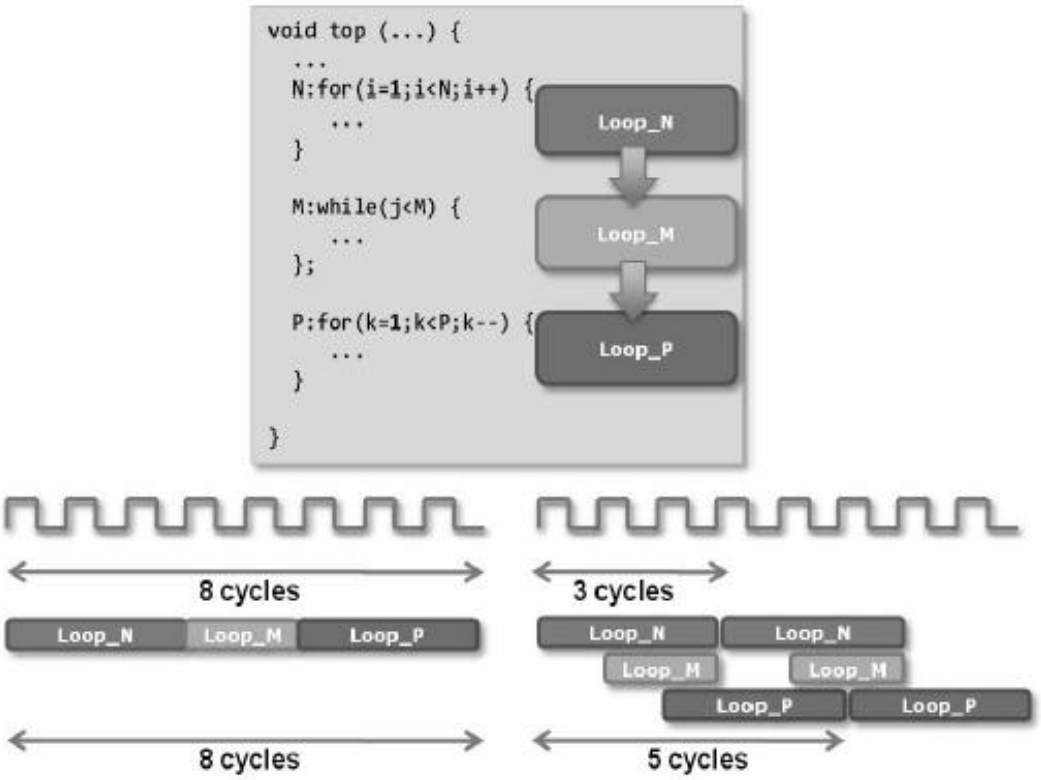
- *Merging*: Όταν υπάρχουν πολλοί διαδοχικοί βρόχοι αυτό μπορεί να δημιουργήσει μερικές φορές περιττούς κύκλους ρολογιού και αποτρέπει την περαιτέρω βελτιστοποιήση. Με τη συγχώνευση διαδοχικών βρόχων μπορούμε να μειώσουμε τη συνολική καθυστέρηση και πετύχουμε επιπλέον βελτίωση της απόδοσης.



Σχήμα 5.6 Loop merging

- *Flattening*: Επιτρέπει την δημιουργία ενός ενιαίου βρόχου από εμφωλευμένους επαναληπτικούς βρόχους, με αποτέλεσμα βελτιωμένο latency κια λογικές βελτιστοποιήσεις.

- *Dataflow pipelining*: Το dataflow pipelining μπορεί να εφαρμοστεί σε βρόχους με παρόμοιο τρόπο όπως μπορεί να εφαρμοστεί και σε συναρτήσεις. Επιτρέπει σε διαδοχικούς βρόχους τρέχουν ταυτόχρονα στο RTL. Το dataflow pipelining θα πρέπει να εφαρμοστεί σε μια συνάρτηση, βρόχο ή τμήμα του προγράμματος το οποίο περιέχει όλους τους βρόχους: δεν εφαρμόζεται σε ένα πεδίο που περιέχει ένα μίγμα των βρόχων και λειτουργιών.

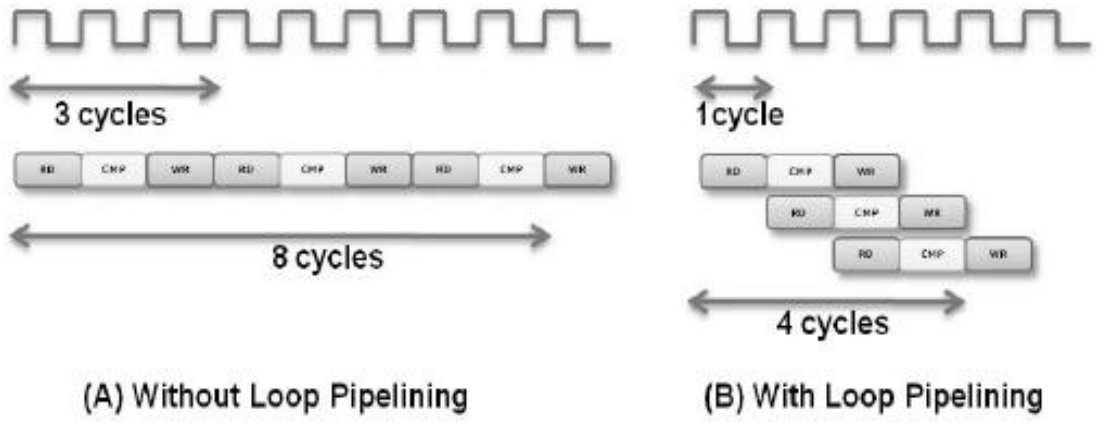


Σχήμα 5.7 Loop dataflow pipelining

Για να χρησιμοποιήσουμε dataflow pipelining οι μεταβλητές πρέπει να παράγονται από ένα βρόχο και καταναλώνονται από μόνο έναν άλλο βρόχο.

- *Pipelining*: Σε μια περιγραφή σε γλώσσα C οι λειτουργίες σε ένα βρόχο εκτελούνται διαδοχικά και η επόμενη επανάληψη του βρόχου μπορεί να αρχίσει μόνο όταν η τελευταία πράξη στο βρόχο έχει ολοκληρωθεί. Σε μια RTL σχεδίαση μπορούν να εκτελεστούν ταυτόχρονα πολλαπλές λειτουργίες και είναι συχνά επιθυμητό η σχεδίαση να υλοποιηθεί με τέτοιο τρόπο. Το

loop pipelining είναι ένας μετασχηματισμός που επιτρέπει τις εργασίες σε ένα βρόχο να εφαρμοστούν με παράλληλο τρόπο και έχει ως αποτέλεσμα την μείωση του latency και την αύξηση του throughput.



Σχήμα 5.8 Loop pipelining

Πολλές φορές το loop pipelining μπορεί να μην είναι δυνατόν να πραγματοποιηθεί λόγω ανεπάρκειας πόρων υλικού αλλά και λόγω εξαρτήσεων δεδομένων.

- *Dependence*: Η ύπαρξη εξαρτήσεων μπορεί να μην μας δίνει την δυνατότητα για pipelining του επαναληπτικού βρόχου. Ωστόσο μπορεί ο χρήστης, γνωρίζοντας εκ των προτέρων την τιμή κάποιων μεταβλητών, να ξέρει ότι υπάρχει δυνατότητα να γίνει το pipelining. Ένα τέτοιο παράδειγμα είναι

```
void foo(int A[3*N], int x)
{
   LF: for (i = 0; i < N; i++)
          A[i+x] = A[i] + i; // User knows that 2*N > x >= N
}
```

Σχήμα 5.9 παράδειγμα Dependence

Σε αυτήν την περίπτωση δίνεται η δυνατότητα μέσω του μετασχηματισμου αυτού να δηλώσουμε στο Vivado hls ότι οι εξαρτήσεις δεν επηρεάζουν το pipelining του loop.

- *Tripcount*: Πολλές φορές δεν είναι δυνατός ο υπολογισμός του latency γιατί ο αριθμός των επαναλήψεων εξαρτάται από μια μεταβλητή η οποία είναι όρισμα της συνάρτησης που υλοποιούμε και δεν είναι γνωστή η τιμή της. Δίνεται η δυνατότητα στον χρήστη να ορίσει ποιά είναι η μικρότερη και ποιά η μεγαλύτερη τιμή αυτής της μεταβλητής ώστε να είναι δυνατός ο υπολογισμός του ανώτερου και κατώτερου ορίου του latency.

- *Latency*: Δίνεται η δυνατότητα μέσω του tool να θέσουμε ένα ανώτατο και ένα κατώτατο όριο latency ως περιορισμό της υλοποίησης. Αυτό είναι

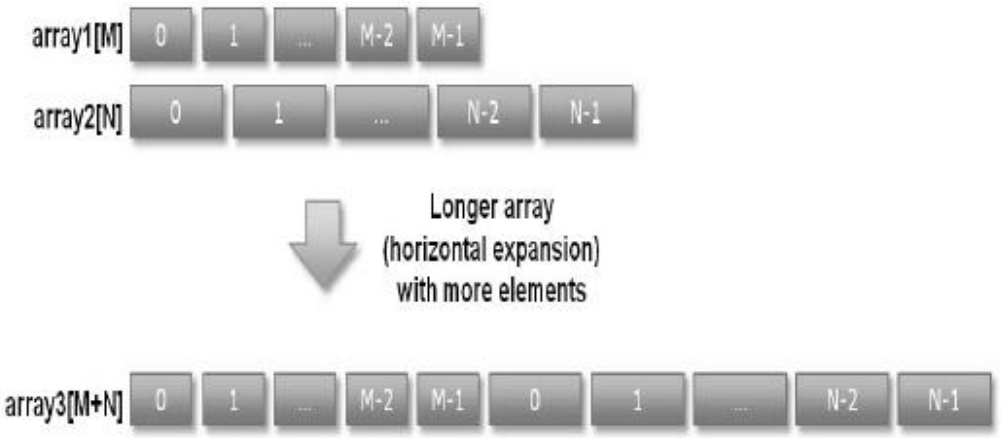
ουσιαστικά ένα μέσο για τη διασφάλιση της επίτευξης των στόχων απόδοσης της υλοποίησης και ένας ισχυρός τρόπος για να ελέγξουμε πώς χρησιμοποιούνται οι πόροι σε έναν βρόχο.

5.3.3 Βελτιστοποιήσεις Πινάκων

Οι πίνακες στη γλώσσα C συνήθως υλοποιούνται ως μνήμες σε επίπεδο RTL.Συνεπώς οι μετασχηματισμοί που εφαρμόζουμε στους πίνακες μπορούν να έχουν μεγάλη επίδραση τόσο στο εύρος όσο και στην απόδοση της υλοποίησης.Οι βελτιστοποίησεις που μπορούμε να κάνουμε είναι οι εξής:

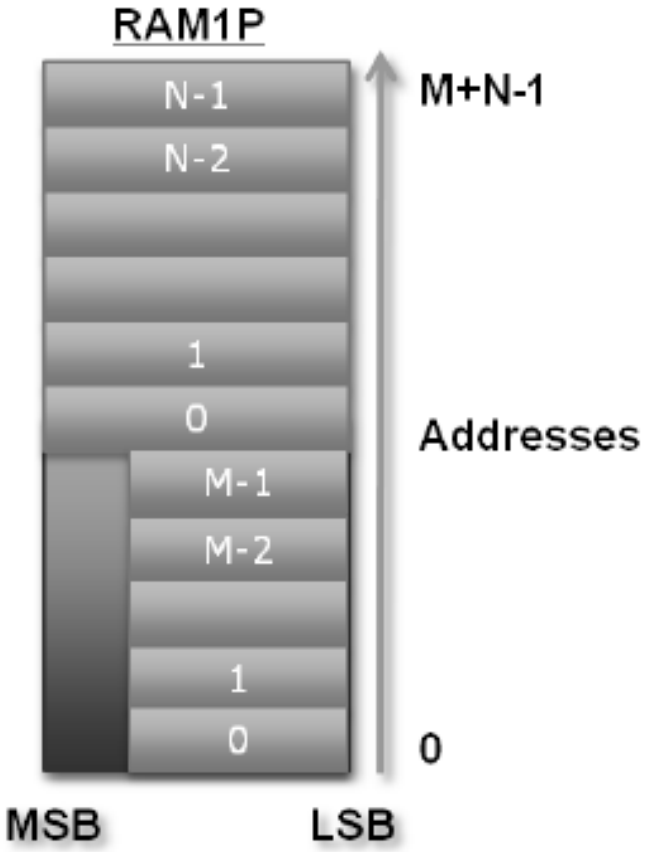
- *Resource*: Εάν δεν έχουν καθοριστεί οι πόροι μνήμης για έναν πίνακα, το High-Level Synthesis αυτόματα θα καθορίσει ποιοι (single-port, διπλής θύρας, κλπ.) θα πρέπει να χρησιμοποιηθούν. Το ίδιο εφαρμόζεται kai στους πίνακες που είναι ορίσματα συναρτήσεων στην top-level συνάρτηση. Πιο συγκεκριμένα το High-Level Synthesis μπορεί να δημιουργήσει μια διεπαφή σε μια διπλή-θύρα μνήμης γεγονός που επιτρέπει υψηλότερη απόδοση. Ωστόσο αυτό δεν είναι εγγυημένο ότι είναι η καλύτερη επιλογή και έτσι ο χρήστης θα πρέπει να ορίζει σε ποιους ακριβώς πόρους μνήμης κάθε πίνακας θα πρέπει να αντιστοιχίζεται.

- *Map*: Στις περισσότερες βιβλιοθήκες τα μοντέλα RAMS που παρέχονται έχουν προκαθορισμένα μεγέθη (π.χ. power-of-2 σε βάθος, με 1,8,16-bit λέξεις). Όταν υπάρχουν πολλοί μικροί πίνακες στα αρχικά ορίσματα, η απεικόνισή τους σε έναν ενιαίο μεγάλο πίνακα πριν από τον καθορισμό ενός πόρου υλικού μπορεί να μειώσει το hardware που χρησιμοποείται για αποθήκευση δεδομένων. Αν κάθε μικρός πίνακας παίρνει μια ξεχωριστή μνήμη, σπαταλάται πολύς χώρος μνήμης και ο τελικός σχεδιασμός θα είναι αδικαιολόγητα μεγάλος. Το Vivado hls υποστηρίζει δύο τρόπους απεικόνισης μικρών πινάκων σε ένα μεγαλύτερο:
  - Horizontal mapping που αντιστοιχεί στη δημιουργία ενός νέου πίνακα με συνένωση των αρχικών πινάκων. Ουσιαστικά υλοποιείται ως ένας ενιαίος πίνακας με περισσότερα στοιχεία. Για παράδειγμα



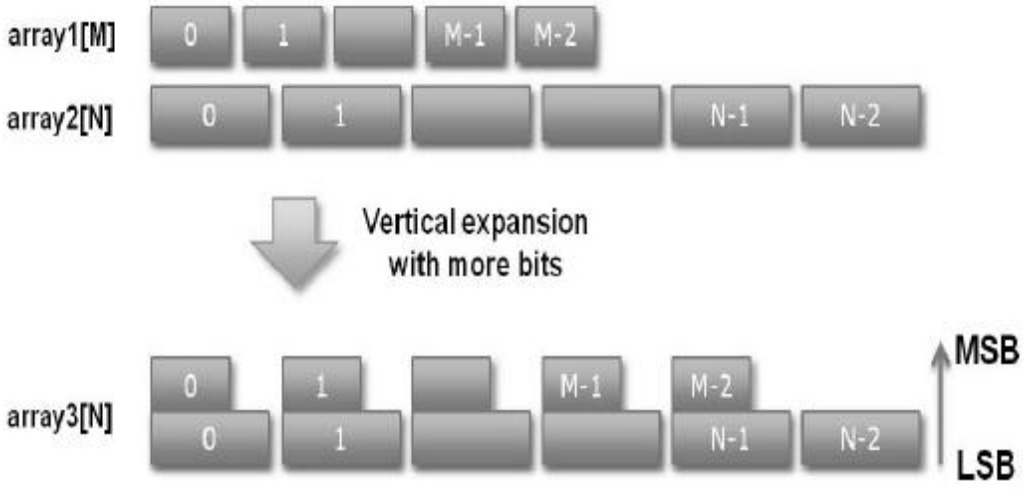
Σχήμα 5.10 horizontal mapping

και η αντίστοιχη RAM είναι



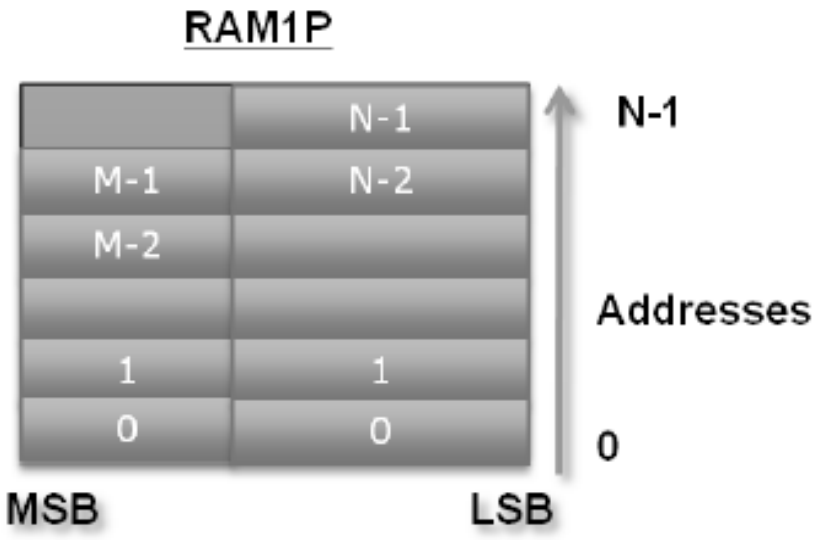
Σχήμα 5.11 horizontal mapping RAM

- Vertical mapping που αντιστοιχεί στη δημιουργία ενός νέου πίνακα με συνένωση των αρχικών λέξεων του πίνακα. Αυτό υλοποιείται με έναν πίνακα με μεγαλύτερο μήκος bit. Για παράδειγμα



Σχήμα 5.12 Vertical expansion

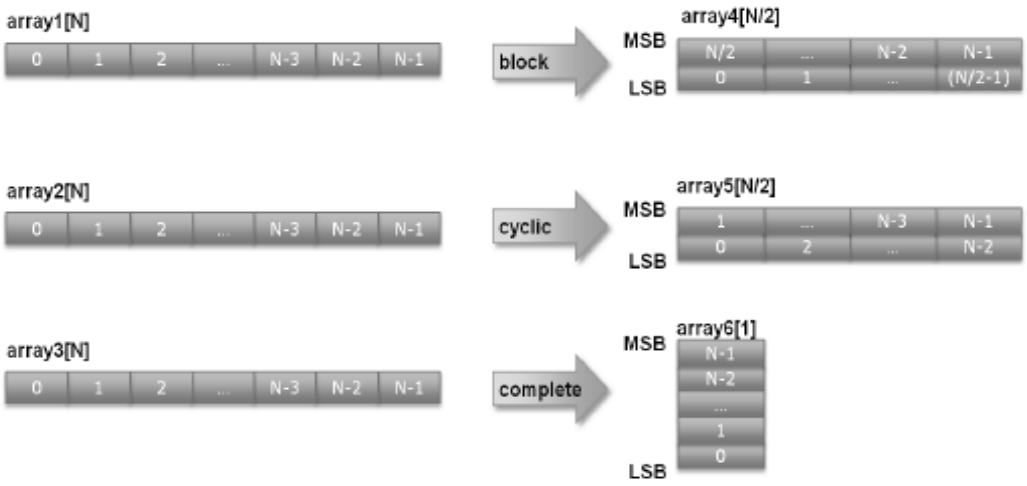
Και η αντίστοιχη RAM είναι:



Σχήμα 5.13 Vertical mapping RAM

- *Partition*: Οι πίνακες μπορούν να διαμεριστούν σε μικρότερους πίνακες. Οι μνήμες έχουν μόνο ένα περιορισμένο αριθμό θυρών ανάγνωσης και εγγραφής και αυτό μπορεί να περιορίσει τα load/stores σε έναν βαρύ για την μνήμη αλγόριθμο. Το εύρος ζώνης μπορεί μερικές φορές να βελτιωθεί με την διαμέριση του αρχικού πίνακα (ένας ενιαίος πόρους μνήμης) σε πολλούς μικρότερους πίνακες (πολλαπλές μνήμες), καθώς αυξάνεται ο αριθμός των θυρών. Κατά συνέπεια διαμέριση ενός μεγαλύτερου πίνακα σε μικρότερους πίνακες μπορεί να βελτιώσει την απόδοση της σχεδίασης.

- *Reshape*: Αυτός ο μετασχηματισμός συνδυάζει διαμέριση πίνακα με vertical mapping.



Σχήμα 5.14 array reshape

- *Stream*: Από προεπιλογή στο Vivado hls, όλοι οι πίνακες υλοποιούνται ως στοιχεία μνήμης, εκτός αν η πλήρης διαμέρησή τους τους μειώνει σε επιμέρους καταχωρητές. Αυτό σημαίνει ότι όλοι οι πίνακες απεικονίζονται σε μια RAM και είναι προσβάσιμοι με τα δεδομένα, τη διεύθυνση και τα σήματα ενεργοποίησης που καθορίζονται από την τεχνολογία της RAM. Στο tool δίνεται η επιλογή για χρήση FIFO interface, αντί για RAM, με τον καθορισμό του πίνακα ως streaming.

5.3.4 Βελτιστοποιήσεις Λογικών Δομών

Οι λογικές δομές που δημιουργούνται από την hls είναι πολύ μεγάλης σημασίας. Έχοντας ολοκληρώσει την βελτιστοποίηση των συναρτήσεων, των επαναληπτικών βρόχων και της απεικόνισης των πινάκων σε μνήμες, στην συνέχεια στρεφόμαστε στις λόγικές δομές. Τα παρακάτω στοιχεία υπαγορεύουν τον τύπο των λογικών δομών που υλοποιούνται σε μια σχεδίαση:

- *συχνότητα ρολογιού*

- *target device*

- *επιλογή αριθμητικών* τελεστών: Κατά τη διάρκεια του hls η σύνθεση επιλέγει υλοποιήσεις για τους τελεστές (+, -, *, /,%, κλπ.) από τη βιβλιοθήκη τεχνολογίας του device που έχουμε επιλέξει. Από προεπιλογή το hls επιλέγει τους τελεστές που δίνουν την καλύτερη ισορροπία μεταξύ χρονισμού και χώρου σχεδίασης. Μέσω του tool ο χρήστης μπορεί με τις κατάλληλες εντολές να καθορίσει ποιοί τελεστές θα χρησιμοποιηθούν αλλά και να ελαχιστοποιήσει τον αριθμό τους.

- *έλεγχος του hardware που θα χρησιμοποιήσουμε*: Οι πόροι υλικού που χρησιμοποιούνται για την υλοποίηση της RTL σχεδίασης μπορούν να καθοριστούν κατά τη διάρκεια της σύνθεσης ή μπορεί να τεθεί ένα γενικό όριο για τη σύνθεση των πόρων που επιτρέπεται να χρησιμοποιούνται. Αυτές οι τεχνικές μπορούν να χρησιμοποιηθούν τόσο για τη βελτίωση του χρονισμού (και ως εκ τούτου του latency και του throughput) όσο και τη μείωση της περιοχής σχεδίασης.

  Επίσης το hardware που χρησιμοποιείται για μια συγκεκριμένη λειτουργία μπορεί να είναι και αυτό καθορισμένο. Όταν πραγματοποιείται High-Level Synthesis χρησιμοποιούνται οι χρονικοί περιορισμοί που καθορίζονται από το ρολόι, οι καθυστερήσεις που καθορίζονται από το device που χρησιμοποιείται και τυχόν περιορισμοί που έχουν καθοριστεί από τον χρήστη σχετικά με το ποιός πυρήνας χρησιμοποιείται για την υλοποίηση των τελεστών. Για παράδειγμα θα μπορούσε να χρησιμοποιηθεί ο πυρήνας

"πολλαπλασιαστή" ή μπορεί να χρησιμοποιηθεί ένας pipelined πυρήνας πολλαπλασιαστή, όπως ο "Mul2S".

- *struct packing*: Το “πακετάρισμα” των στοιχείων μιας δομής σε μια ενιαία λέξη μπορεί να μειώσει το κόστος ελέγχου που συνδέεται με κάθε ένα από τα επιμέρους στοιχεία και αυτό μπορεί να έχει ως αποτέλεσμα μικρότερη και ταχύτερη σχεδίαση.

- *ισορροπία αριθμητικών εκφράσεων*: Κατά τη διάρκεια της σύνθεσης μια σειρά από βελτιστοποιήσεις, όπως η ελαχιστοποίηση του μήκους των bit, εκτελούνται αυτόματα. Στον κατάλογο των αυτόματων βελτιστοποιήσεων είναι και η εξισορρόπηση εκφράσεων. Αυτή η βελτιστοποίηση αναδιατάσσει τους τελεστές ώστε να κατασκευάσει ένα ισορροπημένο δέντρο και να μειώσει το latency. Η εξισορρόπηση εκφράσεων είναι ενεργοποιημένη ως προεπιλογή, αλλά δίνεται και η δυνατότητα να απενεργοποιηθεί γιατί συνήθως απαγορεύει την κοινή χρήση πόρων υλικού και μπορεί να οδηγήσει σε αυξημένη περιοχή σχεδίασης.

5.4 Αριθμητικοί τύποι δεδομένων

Το Vivado HLS υποστηρίζει τόσο integer όσο και fixed point arbitrary precision data types. Για περιγραφές σε γλώσσα C ισχύουν τα παρακάτω:

- *integer data types*: Δυνατότητα επιλογής μεγέθους N bits για integer data types, όπου το N παίρνει τιμές από 1 μέχρι 1024.
- *fixed point data types*: Για C το Vivado δεν δίνει την δυνατότητα για χρήση arbitrary precision data types
- Για *floating point αριθμητική* υποστηρίζονται 32 και 64 bit float.

5.4.1 Σχεδίαση floating point αριθμητικής

Η δυνατότητα για hardware υλοποίηση floating point αριθμητικής είναι ένα από τα μεγάλα πλεονεκτήματα του Vivado HLS.

Οι περισσότεροι σχεδιαστές χρησιμοποιούν fixed-point αριθμητική λογική για την εκτέλεση μαθηματικών συναρτήσεων στα σχέδιά τους, επειδή η μέθοδος δίνει καλά αποτελέσματα στην απόδοση της σχεδίασης. Ωστόσο, υπάρχουν πολλές περιπτώσεις όπου η εφαρμογή μαθηματικών υπολογισμών χρησιμοποιώντας μια floating-point αριθμητική μορφή είναι η καλύτερη επιλογή για να επιτευχθεί η βέλτιστη ποιότητα των αποτελεσμάτων.

Το Vivado hls βοηθάει την μετατροπή C / C++ περιγραφών σε RTL υλοποίηση στην οποία απαιτούνται floating-point υπολογισμοί. Το tool μειώνει δραστικά την προσπάθεια σχεδιασμού που χρειάζεται για να υλοποιηθούν floating-point αλγόριθμοι σε hardware.

Υποστηρίζονται C/C++ float και double τύποι δεδομένων, οι οποίοι βασίζονται σε μονής και διπλής ακρίβειας δυαδικές μορφές κινητής υποδιαστολής,

όπως ορίζονται από το πρότυπο IEEE-754. Επίσης παρέχεται υποστήριξη για υλοποίηση βασικών αριθμητικών πράξεων (+, -, *, /), σχεσιακών τελεστών (! ==, =, <, <=,>,> =) και μετατροπές τύπων δεδομένων(π.χ., ακέραιος σε float και float σε double) οι οποίες επιτυγχάνονται με την αντιστοίχιση αυτών των λειτουργιών στους Xilinx LogiCORE IP Floating-Point πυρήνες. Σε κατάλληλους πυρήνες floating point τελεστές αντιστοιχίζονται και κλήσεις από την οικογένεια συναρτήσεων sqrt() καθώς και κώδικας της μορφής 1.0/x και 1.0/sqrt(x). Γενικότερα παρέχεται δυνατότητα για υλοποίηση πολλών συναρτήσεων από την C(99)/C++ standard math βιβλιοθήκη. Οι συναρτήσεις αυτές υλοποιούνται ως πράξεις κινητής υποδιαστολής μονής και διπλής ακρίβειας. Εάν ο χρήστης καλέσει οποιαδήποτε από αυτές τις συναρτήσεις με ακέραιους ή fixed-point ορίσματα, το εργαλείο θα εφαρμόσει μετατροπές όπου χρειάζεται και θα κάνει τους υπολογισμούς σε κινητή υποδιαστολή.

Οι συναρτήσεις αυτές προορίζονται για να επεξεργάζονται (και να επιστρέφουν) double τύπους δεδομένων, για παράδειγμα, double sqrt (double).Μονής-ακρίβειας εκδόσεις των περισσότερων συναρτήσεων έχουν ένα "f" στο όνομα της συνάρτησης, για παράδειγμα, float sqrtf (float).

Δυνατότητα για παραλληλισμό

Επειδή οι πράξεις κινητής υποδιαστολής χρησιμοποιούν σημαντικούς πόρους υλικού σε σχέση με τις πράξεις ακεραίων ή fixed-point αριθμών, το Vivado HLS χρησιμοποιεί αυτούς τους πόρους όσο πιο αποτελεσματικά γίνεται. Το εργαλείο δίνει την δυνατότητα να μοιράζονται οι Floating-Point Operator πυρήνες μεταξύ πολλών κλήσεων των συναρτήσεων, όταν το επιτρέπουν οι εξαρτήσεις δεδομένων και οι περιορισμοί που έχουν τεθεί.

Κεφάλαιο 6

Ο αλγόριθμος OpenSURF

Επισκόπηση

Στο παρόν κεφάλαιο αρχικά γίνεται μια σύντομη αναφορά στο επιστημονικό πεδίο της όρασης υπολογιστών καθώς και στην βιβλιοθήκη λογισμικού OpenCV, που είναι η πιο διαδεδομένη για ανάπτυξη εφαρμογών όρασης υπολογιστών και η οποία χρησιμοποιείται για τον αλγόριθμο OpenSURF. Στη συνέχεια αναλύεται λεπτομερώς ο αλγόριθμος OpenSURF.

6.1 Εισαγωγή

Η ανθρώπινη όραση είναι ένας περίπλοκος συνδυασμός φυσικών, ψυχολογικών και νευρολογικών διαδικασιών που μας επιτρέπουν να αλληλεπιδρούμε με το περιβάλλον. Χρησιμοποιούμε την όραση για την πλοήγηση, την ανίχνευση, την αναγνώριση και την παρακολούθηση αντικειμένων. Ο στόχος της όρασης υπολογιστών είναι η σχεδίαση υπολογιστικών συστημάτων, ικανών να πραγματοποιήσουν τις παραπάνω διεργασίες σε πραγματικό χρόνο και κάτω από συγκεκριμένους περιορισμούς.

6.2 Όραση Υπολογιστών

Η όραση υπολογιστών είναι ένα πεδίο το οποίο περιλαμβάνει μεθόδους για επεξεργασία, ανάλυση και κατανόηση εικόνων προκειμένου να παραχθεί αριθμητική ή συμβολική πληροφορία(π.χ σε μορφή απόφασης). Βασίζεται κυρίως σε τομείς όπως είναι η γεωμετρία, η φυσική, η στατιστική, η θεωρία μάθησης και συνολοθεωρία.

Στην μηχανική όραση, χρησιμοποιούμε ψηφιακές εικόνες. Ως ψηφιακή εικόνα μπορούμε να θεωρήσουμε την διακριτή αναπαράσταση επεξεργασμένων δεδομένων που μας δίνουν χωρική και χρωματική πληροφορία και προκύπτει μέσω της δειγματοληψίας.

Μερικοί από τους τομείς με τους οποίους ασχολείται η όραση υπολογιστών είναι:

- Επεξεργασία ιστογράμματος εικόνας
- Φιλτράρισμα εικόνας
- Αποκατάσταση εικόνωννκαι βελτίωση ποιότητας
- Συμπίεση εικόνων
- Κατάτμηση εικόνων

- Ανίχνευση αντικειμένων
- Τρισδιάστατη αναπαράσταση αντικειμένων μέσω δισδιάστατων εικόνων
- Παρακολούθηση αντικειμένων
- Ταξινόμηση εικόνων
- Χρωματική αναπαράσταση
- Ανάλυση και ανίχνευση χαρακτηριστικών σημειών(ανίχνευση ακμών, ανίχνευση γωνιών)
- Αναπαράσταση υφής
- Ανίχνευση σκελετού αντικειμένων

Στις μέρες μας έχουν αναπτυχθεί πολλοί αλγόριθμοι όρασης μηχανής και κατα συνέπεια η όραση υπολογιστών βρίσκει εφαρμογή σε πολλούς τομείς.Επίσης, έχουν αναπτυχθεί πολλά προγράμματα για αλγόριθμους της όρασης υπολογιστών, με τα σημαντικότερα να αποτελούν πλέον κομμάτι της βιβλιοθήκης λογισμικού OpenCV.

6.3 Η βιβλιοθήκη OpenCV

Σχήμα 6.1 OpenCV

Η OpenCV (Open Source Computer Vision Library) είναι μια βιβλιοθήκη ελεύθερου λογισμικού για όραση υπολογιστών και εκμάθηση μηχανής. Η βιβλιοθήκη είναι γραμμένη κυρίως σε C και C++ αλλά πλέον έχουν αναπτυχθεί κάποιες συναρτήσεις και σε Python, C#, Java, Ruby και Matlab. Επίσης είναι διαθέσιμη για Linux, Windows, Android, iOS, Blackberry 10 και Mac OS, γεγονός που δίνει πληθώρα επιλογών στον προγραμματιστή.

Η OpenCV δημιουργήθηκε το 1999 στα εργαστήρια της Intel για να προσφέρει υπολογιστική αποδοτικότητα σε εφαρμογές επεξεργασίας και ανάλυσης εικόνας οι οποίες μπορούν να επιβαρύνουν πολύ την cpu και είναι επικεντρωμένη σε εφαρμογές πραγματικού χρόνου.

Ο κύριος στόχος του project της OpenCV είναι να προσφέρει μια απλή στη χρήση της υποδομή πάνω στην όραση υπολογιστών, ώστε να διευκολύνεται η γρήγορη ανάπτυξη εφαρμογών όρασης.Πλέον περιέχει τουλάχιστον 500 συναρτήσεις οι οποίες καλύπτουν μεγάλο εύρος εφαρμογών της όρασης υπολογιστών σε τομείς όπως:

- ιατρική επεξεργασία εικόνας
- ασφάλεια
- διεπαφές χρηστών
- στερεομετρία
- ρομποτική όραση
- επιθεώρηση προϊόντων σε εργοστάσια

6.4 OpenSURF

Ένας από τους πολλούς αλγορίθμους όρασης υπολογιστών που έχουν αναπτυχθεί τα τελευταία χρόνια και ο οποίος βασίζεται στην βιβλιοθήκη OpenCV είναι ο OpenSURF. Το έργο της εξεύρεσης σημείων αντιστοιχίας μεταξύ δύο εικόνων της ίδιας σκηνής ή του ίδιου αντικειμένου, αποτελεί αναπόσπαστο μέρος πολλών συστημάτων μηχανικής όρασης. Ο αλγόριθμος στοχεύει στην εξεύρεση των κυριότερων περιοχών σε εικόνες στις οποίες μπορεί να έχουμε εφαρμόσει μια ποικιλία μετασχηματισμών εικόνας. Αυτός είναι και ο λόγος που έχει αποτελέσει τη βάση πολλών διεργασιών όρασης υπολογιστών όπως αναγνώριση αντικειμένων, παρακολούθηση βίντεο, ιατρική απεικόνιση, επαυξημένη πραγματικότητα και ανάκτηση εικόνας. Ο αλγόριθμος αποτελείται από 3 βασικά κομμάτια:

6.4.1 Integral Images

Χάρη σε αυτήν την μορφή εικόνων αυξάνεται πολύ η απόδοση του αλγορίθμου. Δίνεται από την σχέση:

$$I_{\Sigma}=\sum_{i=0}^{i\le x}\sum_{j=0}^{j\le y} I(x,y)$$

Ουσιαστικά υπολογίζεται ως εξής:
Ξεκινάμε από την μικρότερη γραμμή και κάθε pixel της εικόνας Integral ισούται με το άθροισμα των προηγούμενων(της ίδιας γραμμής). Καθώς ανεβαίνουμε γραμμές, η τιμή σε κάθε pixel είναι ίση με το άθροισμα των πρηγούμενων (της αρχικής εικόνας) συν την τιμή του ακριβώς από κάτω pixel της integral εικόνας.

6.4.2 Fast-Hessian Detector

Hessian
Ο Surf detector βασίζεται στην ορίζουσα του πίνακα Hessian. Ο πίνακας αυτός είναι:

$$H(f(x,y))=\begin{bmatrix}\frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial xy}\\ \frac{\partial^2 f}{\partial xx} & \frac{\partial^2 f}{\partial y^2}\end{bmatrix}$$

Η τιμή της ορίζουσας χρησιμοποιείται για να βρούμε μέγιστα ή ελάχιστα, ανάλογα με την τιμή της δεύτερης παραγώγου. Αν η ορίζουσα είναι αρνητική τότε οι ιδιοτιμές έχουν διαφορετικά πρόσημα και επομένως το σημείο δεν είναι τοπικό ακρότατο.Αν είναι θετική, τότε οι ιδιοτιμές είναι ομόσημες και το σημείο είναι ακρότατο.

Για να μεταφέρουμε αυτήν την θεωρία και στις εικόνες, αρχικά αντί για την συνάρτηση f, χρησιμοποιούμε την εικόνα integral. Για τον υπολογισμόμ των παραγώγων χρησιμοποιούμε τον πολλαπλασιασμό με Gaussian φίλτρο. Με την χρήση του Gaussian μπορούμε να ελέγξουμε το smoothing, ώστε να υπολογίσουμε την ορίζουσα σε διάφορες κλίμακες. Επίσης η συνέλιξη μ' αυτόν τον πυρήνα επιτρέπει ανεξαρτησία από πιθανό rotation της εικόνας. Η Hessian για εικόνες τελικά ισούται με:

$$H(x,\sigma)=\begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix}$$

Όπου Lxx(x,σ) είναι η συνέλιξη της δεύτερης παραγώγου της Gaussian (ως προς x) με την εικόνα στο σημείο x = (x, y) . Οι παράγωγοι αυτοί είναι οι Laplacian of Gaussian. Μια υλοποίηση που αυξάνει πολύ την απόδοση είναι η προσέγγιση της Λαπλασιανής με box filter αναπαραστάσεις των αντίστοιχων πυρήνων. Αυτή η προσέγγιση της ορίζουσας ισούται με

$$det(H_{aprox})=D_{xx}\cdot D_{yy}-(0.9\cdot D_{xy})^2$$

Η αναζήτηση των τοπικών μεγίστων στο χώρο και στην κλίμακα, αποφέρει τα σημεία ενδιαφέροντος της εικόνας.

Κατασκευή του χώρου κλίμακας

Ο χώρος κλίμακας είναι μία συνεχή συνάρτηση που μπορεί να χρησιμοποιηθεί για την ευρεση ακρότατων σε όλες τις κλίμακες. Στην όραση υπολογιστών ο χώρος κλίμακας υλοποιείται ως μια πυραμίδα εικόνας όπου για να πάρουμε την κάθε κλίμακα πολλαπλασιάζουμε την πρηγούμενη με μια gaussian (αυτό γίνεται επαναληπτικά). Ωστόσο υτή η μέθοδος δεν είναι αποδοτική και δεν δίνει δυνατότητα για παράλληλη επεξεργασία.

Στην μέθοδο που χρησιμοποιείται στον αλγόριθμο OpenSURF η εικόνα μένει αναλοίωτη και για να πάρουμε την κάθε κλίμακα μεγαλώνουμε τα box filters με τα οποία την πολλαπλασιάζουμε. Το Dxy σε κάθε επίπεδο αυξάνεται κατά 6 pixels, ενώ τα Dxx και Dyy κατά 2. Καθώς ανεβαίνουμε σε κλίμακα πρέπει να αλλάζουμε και το σ, το οποίο δίνεται από την σχέση:

$$\sigma_{aprox}=CurrentFilterSize\cdot(\frac{1.2}{9})$$

Ακριβής εντοπισμός σημείων ενδιαφέροντος

Για τον εντοπισμό των σημείων ενδιαφέροντος της εικόνας ακολουθούμε 3 βήματα:

1. Κατωφλιοποιούμε την Hessian που υπολογίζουμε. Για όσα σημεία η ορίζουσα έχει τιμή μικρότερη από το κατώφλι, που ορίσαμε, τα απορρίπτουμε.
2. Όσα σημεία προκύπτουν μετά την κατωφλιοποίηση, τα συγκρίνουμε με τα γειτονικά τους.Συγκρίνουμε δηλαδή, με τα αντίστοιχα 9 της μεγαλύτερης

κλίμακας, με τα αντίστοιχα 9 της μικρότερης και με τα 8 γειτονικά της ίδιας κλίμακας. Κρατάμε το pixel μόνο αν η εικόνα σ' αυτό έχει τιμή μεγαλύτερη από τα υπόλοιπα 26.

3. Υπολογίζουμε προσεγγιστικά τη θέση των σημείων ως προς (x, y, σ) από την σχέση:

$$\hat{x} = -\frac{\partial^2 H^{-1}}{\partial x^2} \cdot \frac{\partial H}{\partial x}$$

Αν το x απέχει περισσότερο από 0.5 ως προς x, y ή σ τότε προσαρμόζουμε την θέση του και επαναλαμβάνουμε την διαδικασία μέχρι η διαφορά να γίνει μικρότερη του 0.5 σε όλες τις διαστάσεις ή μέχρι να ξεπεράσουμε τα προκαθορισμένα βήματα του interpolation.

6.4.3 Interest point Descriptor

Ο SURF descriptor περιγράφει πώς κατανέμονται οι "εντάσεις" των pixel, γύρω από μιας ανεξάρτητης κλίμακας γειτονιά των σημείων ενδιαφέροντος που ανιχνεύσαμε με την Fast-Hessian. Χρησιμοποιείται σε συνδυασμό με Haar wavelets για υπολογισμό gradient ως προς x και y. Ο υπολογισμός του descriptor περιλαμβάνει δύο βήματα:

1. Κάθε σημείο ενδιαφέροντος αντιστοιχίζεται σ΄έναν προσανατολισμό
2. Κατασκευάζουμε ένα παράθυρο που εξαρτάται από την κλίμακα, για την εξαγωγή ενός 64-dimensional διανύσματος.

Εκχώρηση προσανατολισμού

Για να πετύχουμε ανεξαρτησία από την περιστροφή της εικόνας, κάθε σημείο ενδιαφέροντος αντιστοιχίζεται σ΄έναν προσανατολισμό και η εξαγωγή των στοιχείων του descriptor γίνεται με βάση τον προσανατολισμό αυτό. Για να καθορίσουμε τον προσανατολισμό, υπολογίζουμε αποκρίσεις Haar wavelets μεγέθους 4σ για ένα σύνολο σημείων που βρίσκονται μέσα σε κύκλο ακτίνας 6σ γύρω από το σημείο ενδιαφέροντος. Το ακριβές σύνολο των pixels καθορίζεται δειγματολειπτώντας μέσα στον κύκλο με βήμα μεγέθους σ.

Κεφάλαιο 7

High-Level Synthesis του αλγορίθμου OpenSURF

Επισκόπηση

Στο κεφάλαιο 7 παρουσιάζονται με λεπτομέρεια τα αποτελέσματα της σχεδίασης του πυρήνα OpenSURF και το τελικό SoC που αναπτύχθηκε για την ανίχνευση των σημείων ενδιαφέροντος σε εικόνες. Τα αποτελέσματα αφορούν κατανάλωση ισχύος, latency και ευρος της περιοχής σχεδίασης.

Τα κομμάτια του αλγορίθμου OpenSURF που υλοποιούνται, είναι 2:

1. Integral Image
2. Fast-Hessian Detector

7.1 Εισαγωγικά

Αρχικά να αναφερθεί ότι το target device είναι της οικογένειας Virtex6 και πιο συγκεκριμένα το xc6vlx240t-2ff1156.

Οι στόχοι της υλοποίησης είναι:

- Να γράψουμε τον κώδικα από C++ σε C.
- Ο κώδικας να γραφτεί με τρόπο τέτοιο ώστε κομμάτια του προγράμματος που έχουν γραφτεί για να τρέχουν σε software , να αντικατασταθούν με άλλα που είναι κατάλληλα για την δημιουργία hardware.
- Εφαρμογή μετασχηματισμών ώστε να βελτιώσουμε την υλοποίηση.

Τέλος, για την υλοποίηση υποθέτουμε εικόνα μεγέθους 89x60.

7.2 Integral Image

Ο αρχικός κώδικας σε C++ της συνάρτησης ***Integral()*** είναι:

```
IplImage *Integral(IplImage *source)
{
  // convert the image to single channel 32f
  IplImage *img = getGray(source);
  IplImage *int_img = cvCreateImage(cvGetSize(img), IPL_DEPTH_32F, 1);

  // set up variables for data access
  int height = img->height;
  int width = img->width;
  int step = img->widthStep/sizeof(float);
  float *data   = (float *) img->imageData;
  float *i_data = (float *) int_img->imageData;
```

```
  // first row only
  float rs = 0.0f;
  for(int j=0; j<width; j++)
  {
    rs += data[j];
    i_data[j] = rs;
  }

  // remaining cells are sum above and to the left
  for(int i=1; i<height; ++i)
  {
    rs = 0.0f;
    for(int j=0; j<width; ++j)
    {
      rs += data[i*step+j];
      i_data[i*step+j] = rs + i_data[(i-1)*step+j];
    }
  }

  // release the gray image
  cvReleaseImage(&img);

  // return the integral image
  return int_img;
}
```

Αν την γράψουμε σε C τότε γίνεται:

```
#include "integral.h"

void integral (  float integral_data[89][60], float source_data[89][60] ) {

	float row_sum;
	int i,j;

	row_sum=0;

	for (j=0;j<60;j++){
		row_sum = row_sum + source_data[i][j] ;
		integral_data[0][j]=row_sum;
	}

	for (i=1;i<60;i++){
		row_sum=0;
		for (j=0;j<60;j++){
			row_sum = row_sum + source_data[i][j] ;
			integral_data[i][j] = integral_data[(i-1)][j] +
row_sum;
		}
	}

}
```

Κάνουμε synthesize και παίρνουμε τα αποτελέσματα:

Απόδοση

Performance Estimates

- Summary of timing analysis
  - Estimated clock period (ns): 7.20
- Summary of overall latency (clock cycles)
  - Best-case latency: 39480
  - Average-case latency: 39480
  - Worst-case latency: 39480
- Summary of loop latency (clock cycles)
  - Loop 1
  - Loop 2
    - # Trip count: 59
    - Latency: 39058
    - Loop 2.1

Περιοχή υλοποίησης

Area Estimates

Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	2	227	212	-
Expression	-	-	0	75	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	232	-
Register	-	-	287	-	-
ShiftMemory	-	-	-	-	-
Total	0	2	514	519	0
Available	832	768	301440	150720	37680
Utilization (%)	0	~0	~0	~0	0

Πίνακας 7.1

Από τον παραπάνω πίνακα βλέπουμε ότι για να μειώσουμε το latency θα πρέπει να εφαρμόσουμε μετασχηματισμούς στο loop 2. Εφαρμογή μετασχηματισμών στο loop1 θα φέρει μικρή αλλάγή στα αποτελέσματα, καθώς θα οδηγήσει σε μικρή μείωση του latency αλλά και σε μικρή αυξηση στο υλικό που χρησιμοποιούμε. Τελικά κάνουμε 2 φορές unroll το εξωτερικό loop ,4 φορές το εσωτερικό και loop pipelining. Εφαρμόζουμε τους μετασχηματισμούς με το χέρι, οπότε ο κώδικας γίνεται:

```
void integral (  float integral_data[89][60], float source_data[89][60] ) {

       float row_sum;
       int i,j;

       row_sum=0;
       for (j=0;j<60/2;j++){
              row_sum = row_sum + source_data[i][2*j] ;
              integral_data[0][2*j]=row_sum;
              row_sum = row_sum + source_data[i][2*j+1] ;
              integral_data[0][2*j+1]=row_sum;
       }


        Shift_Accum_Loop: for (i=1;i<60/2;i++){
              row_sum=0;

              fir_label0:for (j=0;j<60/4;j++){
```

```
                    row_sum = row_sum + source_data[i*2][4*j] ;
                    integral_data[2*i][4*j]=integral_data[2*(i-1)][4*j] + row_sum;
                    //----
                    row_sum = row_sum + source_data[i*2][4*j+1] ;
                    integral_data[2*i][4*j+1]=integral_data[2*(i-1)][4*j+1] + row_sum;
                    //----
                    row_sum = row_sum + source_data[i*2][4*j+2] ;
                    integral_data[2*i][4*j+2]=integral_data[2*(i-1)][4*j+2] + row_sum;
                    //----
                    row_sum = row_sum + source_data[i*2][4*j+3] ;
                    integral_data[2*i][4*j+3]=integral_data[2*(i-1)][4*j+3] + row_sum;
                    ////****
                    row_sum = row_sum + source_data[2*i+1][4*j] ;
                    integral_data[2*i+1][4*j]=integral_data[2*(i-1)+2][4*j] + row_sum;
                    //----
                    row_sum = row_sum + source_data[2*i+1][4*j+1] ;
                    integral_data[2*i+1][4*j+1]=integral_data[2*(i-1)+2][4*j+1] + row_sum;
                    //----
                    row_sum = row_sum + source_data[2*i+1][4*j+2] ;
                    integral_data[2*i+1][4*j+2]=integral_data[2*(i-1)+2][4*j+2] + row_sum;
                    //----
                    row_sum = row_sum + source_data[2*i+1][4*j+3] ;
                    integral_data[2*i+1][4*j+3]=integral_data[2*(i-1)+2][4*j+3] + row_sum;
            }
        }

}
```

Τα αποτελέσματα της σύνθεσης είναι:

Απόδοση	**Περιοχή υλοποίησης**	**Κατανάλωση ισχύος**

Performance Estimates

- **Summary of timing analysis**
  - Estimated clock period (ns): 7.20
- **Summary of overall latency (clock cycles)**
  - Best-case latency: 17355
  - Average-case latency: 17355
  - Worst-case latency: 17355
- **Summary of loop latency (clock cycles)**
  - Loop 1
  - Shift_Accum_Loop

Area Estimates

Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	4	454	424	-
Expression	-	-	0	212	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	362	-
Register	-	-	600	-	-
ShiftMemory	-	-	-	-	-
Total	0	4	1054	998	0
Available	832	768	301440	150720	37680
Utilization (%)	0	~0	~0	~0	0

Power Estimate

Summary

	Power
Component	87
Expression	21
FIFO	-
Memory	-
Multiplexer	36
Register	60
ShiftMemory	-
Total	204

Πίνακας 7.2

Παρατηρούμε ότι καταφέραμε να μειώσουμε το latency της υλοποίησης σε λιγότερο από το μισό. Αυτό είχε ως trade-off τον διπλασιασμό του υλικού που χρησιμοποιούμε, όμως εξακολουθεί να παραμένει σε πολύ χαμηλά επίπεδα βάσει του διαθέσιμου, οπότε δεν επηρεάζει την υλοποίηση.

7.3 Fast-Hessian Detector

Ο detector υλοποιείται από την συνάρτηση ***getIpoints()***. Αυτή αποτελείται από τις:

- buildResponseMap()
- isExtremum()
- interpolateExtremum()

Κάνουμε High-Level Synthesis κάθε μιας από αυτές τις συναρτήσεις χωριστά, ώστε να πετύχουμε την καλύτερη δυνατή υλοποίηση.

7.3.1 buildResponseMap()

Ο αρχικός κώδικας της συνάρτησης buildResponseMap() είναι:

```
void FastHessian::buildResponseMap()
{
  // Deallocate memory and clear any existing response layers
  for(unsigned int i = 0; i < responseMap.size(); ++i)
    delete responseMap[i];
  responseMap.clear();

  // Get image attributes
  int w = (i_width / init_sample);
  int h = (i_height / init_sample);
  int s = (init_sample);

  // Calculate approximated determinant of hessian values
  if (octaves >= 1)
  {
    responseMap.push_back(new ResponseLayer(w,   h,   s,   9));
    responseMap.push_back(new ResponseLayer(w,   h,   s,   15));
    responseMap.push_back(new ResponseLayer(w,   h,   s,   21));
    responseMap.push_back(new ResponseLayer(w,   h,   s,   27));
  }

  if (octaves >= 2)
  {
    responseMap.push_back(new ResponseLayer(w/2, h/2, s*2, 39));
    responseMap.push_back(new ResponseLayer(w/2, h/2, s*2, 51));
  }

  if (octaves >= 3)
  {
    responseMap.push_back(new ResponseLayer(w/4, h/4, s*4, 75));
    responseMap.push_back(new ResponseLayer(w/4, h/4, s*4, 99));
```

```
  }

  if (octaves >= 4)
  {
    responseMap.push_back(new ResponseLayer(w/8, h/8, s*8, 147));
    responseMap.push_back(new ResponseLayer(w/8, h/8, s*8, 195));
  }

  if (octaves >= 5)
  {
    responseMap.push_back(new ResponseLayer(w/16, h/16, s*16, 291));
    responseMap.push_back(new ResponseLayer(w/16, h/16, s*16, 387));
  }

  // Extract responses from the image
  for (unsigned int i = 0; i < responseMap.size(); ++i)
  {
    buildResponseLayer(responseMap[i]);
  }
}

//-------------------------------------------------------

//! Calculate DoH responses for supplied layer
void FastHessian::buildResponseLayer(ResponseLayer *rl)
{
  float *responses = rl->responses;
  unsigned char *laplacian = rl->laplacian;
  int step = rl->step;
  int b = (rl->filter - 1) / 2;
  int l = rl->filter / 3;
  int w = rl->filter;
  float inverse_area = 1.f/(w*w);
  float Dxx, Dyy, Dxy;

  for(int r, c, ar = 0, index = 0; ar < rl->height; ++ar)
  {
    for(int ac = 0; ac < rl->width; ++ac, index++)
    {
      // get the image coordinates
      r = ar * step;
      c = ac * step;

      // Compute response components
      Dxx = BoxIntegral(img, r - l + 1, c - b, 2*l - 1, w)
          - BoxIntegral(img, r - l + 1, c - l / 2, 2*l - 1, l)*3;
      Dyy = BoxIntegral(img, r - b, c - l + 1, w, 2*l - 1)
          - BoxIntegral(img, r - l / 2, c - l + 1, l, 2*l - 1)*3;
      Dxy = + BoxIntegral(img, r - l, c + 1, l, l)
            + BoxIntegral(img, r + 1, c - l, l, l)
            - BoxIntegral(img, r - l, c - l, l, l)
            - BoxIntegral(img, r + 1, c + 1, l, l);

      // Normalise the filter responses with respect to their size
      Dxx *= inverse_area;
      Dyy *= inverse_area;
      Dxy *= inverse_area;
```

```
      // Get the determinant of hessian response & laplacian sign
      responses[index] = (Dxx * Dyy - 0.81f * Dxy * Dxy);
      laplacian[index] = (Dxx + Dyy >= 0 ? 1 : 0);

#ifdef RL_DEBUG
      // create list of the image coords for each response
      rl->coords.push_back(std::make_pair<int,int>(r,c));
#endif
    }
  }
}

//-------------------------------------------------------
```

Στην αρχική συνάρτηση ***buildResponseMap()*** κάνουμε τις εξής αλλαγές ώστε το πρόγραμμα να μπορεί να γραφτεί σε C ,με στατική δέσμευση μνήμης:

- Αλλάζουμε τα ορίσματα της συνάρτησης και τις μεταβλητές int w,h,s τις παίρνουμε έτοιμες ως ορίσματα,ώστε να αποφύγουμε την πράξη της διαίρεσης η οποία απαιτεί πολύ hardware. Μπορούμε να υποθέσουμε ότι η πράξη γίνεται στον συνεπεξεργαστή που θα χρησιμοποιηθεί.

- Παραλείπουμε το σημείο όπου γίνεται memory deallocation γιατί έχουμε στατική δέσμευση μνήμης.

- Το struct responseMap είναι το output του πυρήνα που υλοποιούμε. Το array ***responseMap[]*** είναι αναγκαστικά μεγέθους 12 ,όσα και τα new στον αρχικό κώδικα γιατί εδώ έχουμε στατική μνήμη οπότε θα πρέπει να έχουμε καθορίσει εκ των προτέρων το μέγεθος του πίνακα. Επίσης, επειδή με το dynamic allocation κάποιες από αυτές τις θέσεις μπορεί να μην είχαν δημιουργηθεί,προσθέτουμε και ένα στοιχείο ***inUse*** στο struct το οποίο μας δείχνει αν το responseMap[i] χρησιμοποιείται.

- Μέσα στην συνάρτηση ***buildResponseLayer*** δημιουργήσαμε ένα πίνακα που περιείχε τις συντεταγμένες r,c. Τώρα κάνουμε κάτι αντίστοιχο αλλά με λίγο διαφορετικό τρόπο.Προσθέτουμε στο struct ResponseLayer τους πίνακες coord[1] και coord[2] όπου αποθηκευουμε τα r,c.

- Ως ***Iplimage*** ορίζουμε ένα struct παρόμοιας μορφής με αυτό της OpenCV αλλά πιο απλό, όπως καθορίζεται από τον τρόπο υλοποίησης του πυρήνα.

```
typedef struct IplImage {

    int width;
    int height;
    int widthStep;
    int inUse;
    float imageData[89][60];
```

```
} IplImage;
```

Επίσης τροποποιούμε το struct ***ResponseLayer*** και γίνεται:

```
typedef struct ResponseLayer {

    int width;
    int height;
    int step;
    int filter;
    int inUse;

       int coord1[89*60];
       int coord2[89*60];

       float responses[89*60];
       unsigned char laplacian[89*60];


} ResponseLayer;
```

- Κάνουμε inline την συνάρτηση ***buildResponseLayer()*** αντί να την καλούμε μέσα στο loop for (i = 0; i < 12; ++i) ώστε να αποφύγουμε το κόστος κλήσης συνάρτησης

Με βάση τα παραπάνω ο κώδικας της συνάρτησης γίνεται:

```
#include "responselayer.h"
#include <stdio.h>
#include "integral.h"


//
//! Build map of DoH responses
void buildResponseMap(ResponseLayer responseMap[12],int octaves,int w, int h,
int s, IplImage *img){

      int i;
      int step;
      int b ;
      int l;
      int w_2;
      float inverse_area;
      float Dxx, Dyy, Dxy;
      int r,c,ar,ac,index;

  for (i=0;i<12;i++) {
      responseMap[i].inUse = 0;
  }

  // Calculate approximated determinant of hessian values
  if (octaves >= 1) {
      for (i=0;i<4;i++){
            responseMap[i].width = w;
```

```
            responseMap[i].height = h;
            responseMap[i].step = s;
            responseMap[i].filter = 9+6*i;
            responseMap[i].inUse = 1;
        }
    }

    if (octaves >= 2) {
        responseMap[4].width = w/2;
        responseMap[4].height = h/2;
        responseMap[4].step = s/2;
        responseMap[4].filter = 39;
        responseMap[4].inUse = 1;
        responseMap[5].width = w/2;
        responseMap[5].height = h/2;
        responseMap[5].step = s/2;
        responseMap[5].filter = 51;
        responseMap[5].inUse = 1;
    }

    if (octaves >= 3){
        responseMap[6].width = w/4;
        responseMap[6].height = h/4;
        responseMap[6].step = s/4;
        responseMap[6].filter = 75;
        responseMap[6].inUse = 1;
        responseMap[7].width = w/4;
        responseMap[7].height = h/4;
        responseMap[7].step = s/4;
        responseMap[7].filter = 99;
        responseMap[7].inUse = 1;
    }

    if (octaves >= 4) {
        responseMap[8].width = w/8;
        responseMap[8].height = h/8;
        responseMap[8].step = s/8;
        responseMap[8].filter = 147;
        responseMap[8].inUse = 1;
        responseMap[9].width = w/8;
        responseMap[9].height = h/8;
        responseMap[9].step = s/8;
        responseMap[9].filter = 195;
        responseMap[9].inUse = 1;
    }

    if (octaves >= 5) {
        responseMap[10].width = w/16;
        responseMap[10].height = h/16;
        responseMap[10].step = s/16;
        responseMap[10].filter = 291;
        responseMap[10].inUse = 1;
        responseMap[11].width = w/16;
        responseMap[11].height = h/16;
        responseMap[11].step = s/16;
        responseMap[11].filter = 387;
        responseMap[11].inUse = 1;
    }
```

```
  // Extract responses from the image
      for ( i = 0; i < 12; ++i) {

            b = (responseMap[i].filter - 1) / 2;
            l = responseMap[i].filter / 3;
            w_2 = responseMap[i].filter;
            inverse_area = 1.f/(w_2*w_2);


            index=0;
            for( ar = 0; ar < 60; ++ar){
                  for( ac = 0; ac < 89; ++ac) {
                        // get the image coordinates
                        r = ar * responseMap[i].step;
                        c = ac * responseMap[i].step;

                        // Compute response components
                        Dxx = BoxIntegral(img, r - l + 1, c - b, 2*l - 1,
responseMap[i].filter)
                              - BoxIntegral(img, r - l + 1, c - l / 2, 2*l -
1, l)*3;
                        Dyy = BoxIntegral(img, r - b, c - l + 1,
responseMap[i].filter, 2*l - 1)
                              - BoxIntegral(img, r - l / 2, c - l + 1, l, 2*l
- 1)*3;
                        Dxy = + BoxIntegral(img, r - l, c + 1, l, l)
                              + BoxIntegral(img, r + 1, c - l, l, l)
                              - BoxIntegral(img, r - l, c - l, l, l)
                              - BoxIntegral(img, r + 1, c + 1, l, l);


                        // Normalise the filter responses with respect to
their size
                        Dxx *= inverse_area;
                        Dyy *= inverse_area;
                        Dxy *= inverse_area;

                        // Get the determinant of hessian response & laplacian
sign
                        responseMap[i].responses[index] = (Dxx * Dyy - 0.81f *
Dxy * Dxy);
                        if (Dxx+Dyy >=0)
                              responseMap[i].laplacian[index] = 1;
                        else
                              responseMap[i].laplacian[index] = 0;

                        responseMap[i].coord1[index]=r;
                        responseMap[i].coord2[index]=c;
                        index++;
                  }
            }
      }

}

float BoxIntegral(IplImage *img, int row, int col, int rows, int cols) {
      float data[N*N];
```

```
	float A,B,C,D;
	int step = img->widthStep/sizeof(float);
	int i;

	for (i=0;i<5340;i++){
		data[i]=img->imageData[i];
	}
  // The subtraction by one for row/col is because row/col is inclusive.
	int r1 = min(row,img->height) - 1;
	int c1 = min(col,img->width)  - 1;
	int r2 = min(row + rows,img->height) - 1;
	int c2 = min(col + cols,img->width)  - 1;

	A=0.0f;
	B=0.0f;
	C=0.0f;
	D=0.0f;

	if (r1 >= 0 && c1 >= 0) A = data[r1 * step + c1];
	if (r1 >= 0 && c2 >= 0) B = data[r1 * step + c2];
	if (r2 >= 0 && c1 >= 0) C = data[r2 * step + c1];
	if (r2 >= 0 && c2 >= 0) D = data[r2 * step + c2];


	if ((A - B - C + D)>0)
		return 0;
	else
		return (A - B - C + D);
}


int min(int a, int b) {

	if (a>b)
		return b;
	else
		return a;
}
```

Τα αποτελέσματα της σύνθεσης είναι:

Απόδοση	Περιοχή υλοποίησης
Performance Estimates Summary of timing analysis Estimated clock period (ns): 8.71 Summary of overall latency (clock cycles) Best-case latency: 21914464625 Average-case latency: 21914464628 Worst-case latency: 21914464630 Summary of loop latency (clock cycles) Loop 1 Loop 2 Loop 3 # Trip count: 12 Latency: 21914464608 ~ 21914464608 Loop 3.1 # Trip count: 60 Latency: 1826205360 Loop 3.1.1 # Trip count: 89 Latency: 30436754	Area Estimates Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	128	34	2234	4510	-
Expression	-	-	0	1088	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	574	-
Register	-	-	1416	-	-
ShiftMemory	-	-	-	-	-
Total	128	34	3650	6172	0
Available	270	240	126800	63400	15850
Utilization (%)	47	14	2	9	0

Πίνακας 7.3

Παρατηρούμε ότι το latency της υλοποίησης είναι πάρα πολύ μεγάλο, οπότε εφαρμόζουμε εκτός από το function inline, επιπλέον μετασχηματισμούς:

1. Η χρήση του 47% των διαθέσιμων BRAM οφείλοταν στην συνάρτηση ***BoxIntegral()***:

```
float BoxIntegral(IplImage *img, int row, int col, int rows, int cols) {
	float data[N*N];
	float A,B,C,D;
	int step = img->widthStep/sizeof(float);
	int i;

	for (i=0;i<5340;i++){
		data[i]=img->imageData[i];
	}
  // The subtraction by one for row/col is because row/col is inclusive.
	int r1 = min(row,img->height) - 1;
	int c1 = min(col,img->width)  - 1;
	int r2 = min(row + rows,img->height) - 1;
	int c2 = min(col + cols,img->width)  - 1;
```

```
	A=0.0f;
	B=0.0f;
	C=0.0f;
	D=0.0f;

	if (r1 >= 0 && c1 >= 0) A = data[r1 * step + c1];
	if (r1 >= 0 && c2 >= 0) B = data[r1 * step + c2];
	if (r2 >= 0 && c1 >= 0) C = data[r2 * step + c1];
	if (r2 >= 0 && c2 >= 0) D = data[r2 * step + c2];


	if ((A - B - C + D)>0)
		return 0;
	else
		return (A - B - C + D);
}
```

Και πιο συγκεκριμένα στον πίνακα ***data[]***, η χρήση του οποίου όχι μόνο απαιτεί επιπλέον χρήση μνήμης, αλλά απαιτεί και πράξεις που θα μπορούσαμε να αποφύγουμε και αυξάνουν το latency, λόγω του ότι γίνονται πολλές φορές (89*60*12). Η νέα συνάρτηση είναι:

```
float BoxIntegral(IplImage *img, int row, int col, int rows, int cols) {

	float A,B,C,D;
	int step = img->widthStep/sizeof(float);
	int i;


  // The subtraction by one for row/col is because row/col is inclusive.
	int r1 = min(row,img->height) - 1;
	int c1 = min(col,img->width)  - 1;
	int r2 = min(row + rows,img->height) - 1;
	int c2 = min(col + cols,img->width)  - 1;

	A=0.0f;
	B=0.0f;
	C=0.0f;
	D=0.0f;
	if (r1 >= 0 && c1 >= 0) A = img->imageData[r1 * step + c1];
	if (r1 >= 0 && c2 >= 0) B = img->imageData[r1 * step + c2];
	if (r2 >= 0 && c1 >= 0) C = img->imageData[r2 * step + c1];
	if (r2 >= 0 && c2 >= 0) D = img->imageData[r2 * step + c2];

	if ((A - B - C + D)>0)
		return 0;
	else
		return (A - B - C + D);
}
```

2. Κάνουμε unroll το εσωτερικό loop του τελευταίου επαναληπτικού βρόχου

Ξανακάνουμε Synthesize και παίρνουμε τα αποτελέσματα:

Απόδοση	Περιοχή υλοποίησης	Κατανάλωση ισχύος

Performance Estimates

- **Summary of timing analysis**
  - Estimated clock period (ns): 8.74
- **Summary of overall latency (clock cycles)**
  - Best-case latency: 10886633
  - Average-case latency: 10886636
  - Worst-case latency: 10886638
- **Summary of loop latency (clock cycles)**
  - Loop 1
  - Loop 2
  - buildResponseMap_label1
    - # Trip count: 12
    - Latency: 10886616
    - buildResponseMap_label3
      - # Trip count: 60
      - Latency: 907200
      - buildResponseMap_label2
        - # Trip count: 5
        - Latency: 15118

Area Estimates

Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	28	2203	3393	-
Expression	-	34	0	4788	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	2888	-
Register	-	-	5031	-	-
ShiftMemory	-	-	-	-	-
Total	0	62	7234	11069	0
Available	832	768	301440	150720	37680
Utilization (%)	0	8	2	7	0

Power Estimate

Summary

	Power
Component	355
Expression	481
FIFO	-
Memory	-
Multiplexer	288
Register	503
ShiftMemory	-
Total	1627

Πίνακας 7.4

Συγκρίνουμε τα αποτελέσματα και βλέπουμε ότι με μια μικρή αύξηση στο υλικό, πετύχαμε να μειώσουμε σχεδόν 1500 φορές το latency.

7.3.2 isExtremum()

Ο αρχικός κώδικας της συνάρτησης(χωρίς κάποιο μετασχηματισμό) είναι:

```
int isextremum(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer *b)
{
	double thresh=0.0004;
  // bounds check
  int layerBorder = (t->filter + 1) / (2 * t->step);
  if (r <= layerBorder || r >= t->height - layerBorder || c <= layerBorder || c >= t->width - layerBorder)
    return 0;

  // check the candidate point in the middle layer is above thresh
  int scale = m->width / t->width;
```

```
  float candidate = m->responses[(scale * r) * m->width + (scale * c)];
  if (candidate < thresh)
    return 0;

  int scale_b = b->width / t->width;
  for (int rr = -1; rr <=1; ++rr)
  {
    for (int cc = -1; cc <=1; ++cc)
    {
      // if any response in 3x3x3 is greater candidate not maximum
      if (
      t->responses[(r+rr) * t->width + c+cc] >= candidate ||
      ((rr != 0 || cc != 0) && m->responses[(scale * (r+rr)) * m->width + (scale *
      (c+cc))] >= candidate) ||
      b->responses[(scale_b * (r+rr)) * b->width + (scale_b * (c+cc))] >= candidate
      )
        return 0;
    }
  }

  return 1;
}
```

Να σημειωθεί ότι η γραμμή:

```
float candidate = m->getResponse(r,c,t)
```

Του αρχικού προγράμματος σε C++, αντικαταστάθηκε από τις:

```
int scale = m->width / t->width;
float candidate = m->responses[(scale * r) * m->width + (scale * c)];
```

Κάνουμε Synthesize και παίρνουμε τα αποτελέσματα:

Απόδοση	Περιοχή υλοποίησης
Performance Estimates Summary of timing analysis Estimated clock period (ns): 8.49 Summary of overall latency (clock cycles) Best-case latency: ? Average-case latency: ? Worst-case latency: ? Summary of loop latency (clock cycles) Loop 1	Area Estimates Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	0	6827	8311	-
Expression	-	7	0	403	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	372	-
Register	-	-	392	-	-
ShiftMemory	-	-	-	-	-
Total	0	7	7219	9086	0
Available	832	768	301440	150720	37680
Utilization (%)	0	~0	2	6	0

Πίνακας 7.5

Παρατηρούμε ότι με την μορφή που έχει η συνάρτηση, δεν μπορούμε να υπολογίσουμε το latency της υλοποίησης. Αυτό οφείλεται στο γεγονός ότι το πότε θα επιστρέψει το αποτέλεσμα, εξαρτάται από μεταβλητές που δέχεται σαν ορίσματα.

Στον παραπάνω κώδικα μπορούμε να κάνουμε τους εξής μετασχηματισμούς:

1. Συχγώνευση των if που έχουμε έξω από το loop, δηλαδή αντί για:

```
int layerBorder = (t->filter + 1) / (2 * t->step);
if (r <= layerBorder || r >= t->height - layerBorder || c <= layerBorder ||
c >= t->width - layerBorder)
   return 0;
// check the candidate point in the middle layer is above thresh
int scale = m->width / t->width;
float candidate = m->responses[(scale * r) * m->width + (scale * c)];
if (candidate < thresh)
   return 0;
```

Να έχουμε:

```
int layerBorder = (t->filter + 1) / (2 * t->step);
int scale = m->width / t->width;
float candidate = m->responses[(scale * r) * m->width + (scale * c)];
if (r <= layerBorder || r >= t->height - layerBorder || c <= layerBorder ||
c >= t->width - layerBorder || candidate < thresh )
   return 0;
```

2. Full unroll το loop:

```
int scale_b = b->width / t->width;

  //rr=-1, cc=-1
  if (
       t->responses[(r-1) * t->width + c-1] >= candidate ||
       m->responses[(scale * (r-1)) * m->width + (scale * (c-1))] >=
candidate ||
       b->responses[(scale_b * (r-1)) * b->width + (scale_b * (c-1))] >=
candidate
       )
       return 0;

  //rr=-1, cc=1
  if (
       t->responses[(r-1) * t->width + c+1] >= candidate ||
    m->responses[(scale * (r-1)) * m->width + (scale * (c+1))] >= candidate
||
       b->responses[(scale_b * (r-1)) * b->width + (scale_b * (c+1))] >=
candidate
       )
       return 0;

  //rr=1, cc=-1
  if (
       t->responses[(r+1) * t->width + c-1] >= candidate ||
    m->responses[(scale * (r+1)) * m->width + (scale * (c-1))] >= candidate
||
       b->responses[(scale_b * (r+1)) * b->width + (scale_b * (c-1))] >=
candidate
       )
       return 0;


  //rr=1, cc=1
  if (
       t->responses[(r+1) * t->width + c+1] >= candidate ||
       m->responses[(scale * (r+1)) * m->width + (scale * (c+1))] >=
candidate ||
       b->responses[(scale_b * (r+1)) * b->width + (scale_b * (c+1))] >=
candidate
       )
       return 0;
```

Τα αποτελέσματα που παίρνουμε από την σύνθεση είναι:

Απόδοση	Περιοχή υλοποίησης	Κατανάλωση ισχύος

Performance Estimates

Summary of timing analysis

Estimated clock period (ns): 8.65

Summary of overall latency (clock cycles)

Best-case latency: 38

Average-case latency: 98

Worst-case latency: 116

Area Estimates

Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	0	6827	8311	-
Expression	-	16	0	309	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	321	-
Register	-	-	358	-	-
ShiftMemory	-	-	-	-	-
Total	0	16	7185	8941	0
Available	832	768	301440	150720	37680
Utilization (%)	0	2	2	5	0

Power Estimate

Summary

	Power
Component	1513
Expression	31
FIFO	-
Memory	-
Multiplexer	32
Register	35
ShiftMemory	-
Total	1611

Πίνακας 7.6

Τελικά πετύχαμε όχι μόνο να μειώσουμε το υλικό,αλλά και να είναι δυνατός ο υπολογισμός του latency.

7.3.3 interpolateExtremum()

Η συνάρτηση σε C++ είναι:

```
//! Interpolate scale-space extrema to subpixel accuracy to form an image
feature.
void FastHessian::interpolateExtremum(int r, int c, ResponseLayer *t,
ResponseLayer *m, ResponseLayer *b)
{
  // get the step distance between filters
  // check the middle filter is mid way between top and bottom
  int filterStep = (m->filter - b->filter);
  assert(filterStep > 0 && t->filter - m->filter == m->filter - b->filter);

  // Get the offsets to the actual location of the extremum
  double xi = 0, xr = 0, xc = 0;
  interpolateStep(r, c, t, m, b, &xi, &xr, &xc );

  // If point is sufficiently close to the actual extremum
  if( fabs( xi ) < 0.5f  &&  fabs( xr ) < 0.5f  &&  fabs( xc ) < 0.5f )
  {
    Ipoint ipt;
    ipt.x = static_cast<float>((c + xc) * t->step);
    ipt.y = static_cast<float>((r + xr) * t->step);
```

```
    ipt.scale = static_cast<float>((0.1333f) * (m->filter + xi *
filterStep));
    ipt.laplacian = static_cast<int>(m->getLaplacian(r,c,t));
    ipts.push_back(ipt);
  }
}

//-------------------------------------------------------

//! Performs one step of extremum interpolation.
void FastHessian::interpolateStep(int r, int c, ResponseLayer *t,
ResponseLayer *m, ResponseLayer *b,
                                  double* xi, double* xr, double* xc )
{
  CvMat* dD, * H, * H_inv, X;
  double x[3] = { 0 };

  dD = deriv3D( r, c, t, m, b );
  H = hessian3D( r, c, t, m, b );
  H_inv = cvCreateMat( 3, 3, CV_64FC1 );
  cvInvert( H, H_inv, CV_SVD );
  cvInitMatHeader( &X, 3, 1, CV_64FC1, x, CV_AUTOSTEP );
  cvGEMM( H_inv, dD, -1, NULL, 0, &X, 0 );

  cvReleaseMat( &dD );
  cvReleaseMat( &H );
  cvReleaseMat( &H_inv );

  *xi = x[2];
  *xr = x[1];
  *xc = x[0];
}

//-------------------------------------------------------

//! Computes the partial derivatives in x, y, and scale of a pixel.
CvMat* FastHessian::deriv3D(int r, int c, ResponseLayer *t, ResponseLayer
*m, ResponseLayer *b)
{
  CvMat* dI;
  double dx, dy, ds;

  dx = (m->getResponse(r, c + 1, t) - m->getResponse(r, c - 1, t)) / 2.0;
  dy = (m->getResponse(r + 1, c, t) - m->getResponse(r - 1, c, t)) / 2.0;
  ds = (t->getResponse(r, c) - b->getResponse(r, c, t)) / 2.0;

  dI = cvCreateMat( 3, 1, CV_64FC1 );
  cvmSet( dI, 0, 0, dx );
  cvmSet( dI, 1, 0, dy );
  cvmSet( dI, 2, 0, ds );

  return dI;
}

//-------------------------------------------------------

//! Computes the 3D Hessian matrix for a pixel.
CvMat* FastHessian::hessian3D(int r, int c, ResponseLayer *t, ResponseLayer
```

```
*m, ResponseLayer *b)
{
  CvMat* H;
  double v, dxx, dyy, dss, dxy, dxs, dys;

  v = m->getResponse(r, c, t);
  dxx = m->getResponse(r, c + 1, t) + m->getResponse(r, c - 1, t) - 2 * v;
  dyy = m->getResponse(r + 1, c, t) + m->getResponse(r - 1, c, t) - 2 * v;
  dss = t->getResponse(r, c) + b->getResponse(r, c, t) - 2 * v;
  dxy = ( m->getResponse(r + 1, c + 1, t) - m->getResponse(r + 1, c - 1, t)
-
          m->getResponse(r - 1, c + 1, t) + m->getResponse(r - 1, c - 1, t)
) / 4.0;
  dxs = ( t->getResponse(r, c + 1) - t->getResponse(r, c - 1) -
          b->getResponse(r, c + 1, t) + b->getResponse(r, c - 1, t) ) /
4.0;
  dys = ( t->getResponse(r + 1, c) - t->getResponse(r - 1, c) -
          b->getResponse(r + 1, c, t) + b->getResponse(r - 1, c, t) ) /
4.0;

  H = cvCreateMat( 3, 3, CV_64FC1 );
  cvmSet( H, 0, 0, dxx );
  cvmSet( H, 0, 1, dxy );
  cvmSet( H, 0, 2, dxs );
  cvmSet( H, 1, 0, dxy );
  cvmSet( H, 1, 1, dyy );
  cvmSet( H, 1, 2, dys );
  cvmSet( H, 2, 0, dxs );
  cvmSet( H, 2, 1, dys );
  cvmSet( H, 2, 2, dss );

  return H;
}
```

Στον παραπάνω κώδικα κάνουμε τις εξής μετατροπές, ώστε να έχουμε μια αποδοτική περιγραφή σε C:

1. Υλοποιούμε την συνάρτηση ***CvInvert()*** της OpenCV. Ουσιαστικά γράφουμε σε C την μέθοδο lu decomposition.Ο αλγόριθμος εδώ εφαρμόζεται για την αντιστροφή ενός 3x3 πίνακα και είναι η απλή περίπτωση,δηλαδή χωρίς να γίνεται έλεγχος αν κάνουμε διαιρέσεις με το 0.

```
u[2][0]=0;
u[2][1]=0;
l[0][0]=1;
l[1][1]=1;
l[2][2]=1;
l[0][1]=0;
l[0][2]=0;
l[1][2]=0;
//lush susthmatos
u[0][0]=H.data[0][0];
u[0][1]=H.data[0][1];
u[0][2]=H.data[0][2];
l[1][0]=H.data[1][0]/u[0][0];
```

```
u[1][1]=H.data[1][1]-u[0][1]*l[1][0];
u[1][2]=H.data[1][2]-u[0][2]*l[1][0];
l[2][0]=H.data[2][0]/u[0][0];
l[2][1]=(H.data[2][1]-u[0][1]*l[2][0])/u[1][1];
u[2][2]=H.data[2][2]-(u[0][2]*l[2][0]+u[1][2]*l[2][1]);

//***1h sthlh
//bhma 1
z[0][0]=1;
z[1][0]= -l[1][0];
z[2][0]=-l[2][0]+l[2][1]*l[1][0];
//bhma 2
H_inv.data[2][0]=z[2][0]/u[2][2];
H_inv.data[1][0]=(z[1][0]-u[1][2]*(z[2][0]/u[2][2]))/u[1][1];
H_inv.data[0][0]=(1-u[0][1]*H_inv.data[1][0]-u[0][2]*H_inv.data[2]
[0])/u[0][0];

//*** 2h sthlh
//bhma 1
z[0][1]=0;
z[1][1]=1;
z[2][1]=-l[2][1];
//bhma 2
H_inv.data[2][1]=(-l[2][1])/u[2][2];
H_inv.data[1][1]=(1-u[1][2]*H_inv.data[2][1])/u[1][1];
H_inv.data[0][1]=-(u[0][1]*H_inv.data[1][1]+u[0][2]*H_inv.data[2]
[1])/u[0][0];

//*** 3h sthlh
//bhma 1
z[0][2]=0;
z[1][2]=0;
z[2][2]=1;
//bhma 2
H_inv.data[2][2]=1/u[2][2];
H_inv.data[1][2]=-u[1][2]*H_inv.data[2][2]/u[1][1];
H_inv.data[0][2]=-(u[0][1]*H_inv.data[1][2]+u[0][2]*H_inv.data[2]
[2])/u[0][0];
```

2. Έχοντας υπολογίσει τον αντίστροφο του H,πρέπει να κάνουμε τον πολλαπλασιασμό $X=-Hinv \cdot dD$, προκειμένου να πάρουμε τα x[0], x[1], x[2]. Για τον υπολογισμό χρησιμοποιήσαμε τον παρακάτω κώδικα:

```
for (i=0;i<3;i++) {
     x[i]=0;
     for (k=0;k<3;k++){
         x[i]=-H_inv.data[i][k]*dD.data[k][0];
     }
}

xi = x[2];
xr = x[1];
xc = x[0];
```

Ωστόσο επειδή οι πίνακες είναι μικρού μεγέθους,μπορούμε να γράψουμε τις πράξεις αναλυτικά.Τροποποιούμε δηλαδή το πρόγραμμα ώστε τελικα ο παραπάνω κώδικας να γίνει:

```
xi=-H_inv.data[2][0]*dD.data[0][0]-H_inv.data[2][1]*dD.data[1][0] H_inv.data[2][2]*dD.data[2][0];
xr=-H_inv.data[1][0]*dD.data[0][0]-H_inv.data[1][1]*dD.data[1][0]-H_inv.data[1][2]*dD.data[2][0];
xc=-H_inv.data[0][0]*dD.data[0][0]-H_inv.data[0][1]*dD.data[1][0]-H_inv.data[0][2]*dD.data[2][0];
```

3. Στα πλαίσια της εργασίας ορίζουμε την δομή CvMat της βιβλιοθήκης OpenCV, ως εξής:

```
typedef struct CvMat {
      int rows;
      int cols;
      int type;
      float data[3][3];
}CvMat;
```

Η τελική συνάρτηση είναι:

```
#include <math.h>
#include "ipoint.h"
#include "responselayer.h"
#include "interpolateextremum.h"

#include "ap_cint.h"


//! Interpolate scale-space extrema to subpixel accuracy to form an image feature.
void interpolateExtremum(Ipoint *ipt, int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b) {
  // get the step distance between filters
  // check the middle filter is mid way between top and bottom
   int filterStep = (m->filter - b->filter);
  //assert(filterStep > 0 && t->filter - m->filter == m->filter - b->filter);
  int i, j, k;
  int u[3][3],l[3][3], z[3][3];

  // Get the offsets to the actual location of the extremum
  double xi = 0, xr = 0, xc = 0;
  //interpolateStep(r, c, t, m, b, &xi, &xr, &xc );
  // Performs one step of extremum interpolation.
  CvMat dD ,  H, H_inv, X ;
  double x[3] = { 0 };

  //dD = deriv3D( r, c, t, m, b );
 //<-----edw 3ekinaei h deriv3D
 double dx, dy, ds;

  int scale = m->width / t->width;
  //edw to width anaferetai se kapoio global width h einai to m.width?
  dx = (m->responses[(scale * r) * m->width + (scale * (c+1))] - m->responses[(scale * r)
* m->width + (scale * (c-1))]) / 2.0;
  dy = (m->responses[(scale * (r+1)) * m->width + (scale * c)] - m->responses[(scale * (r-
```

```
1)) * m->width + (scale * c)]) / 2.0;
  int scale_2 = b->width / t->width;
  ds = (t->responses[r * t->width + c] - b->responses[(scale_2 * r) * b->width + (scale_2
* c)]) / 2.0;

  //dI = cvCreateMat( 3, 1, CV_64FC1 );  64-bit float single-channel matrix
  dD.rows = 3;
  dD.cols = 1;
  dD.data[0][0] = dx;
  dD.data[1][0] = dy;
  dD.data[2][0] = ds;
  //edw teleiwnei h deriv3D--->*/

  //H = hessian3D( r, c, t, m, b );
  //<----- edw 3ekinaei h hessian3D

  double v, dxx, dyy, dss, dxy, dxs, dys;

  v = m->responses[(scale * r) * m->width + (scale * c)];
  dxx = m->responses[(scale * r) * m->width + (scale * (c+1))] + m->responses[(scale * r)
* m->width + (scale * (c-1))] - 2 * v;
  dyy = m->responses[(scale * (r+1)) * m->width + (scale * c)] + m->responses[(scale * (r-
1)) * m->width + (scale * c)] - 2 * v;
  dss = t->responses[r * t->width + c] - b->responses[(scale_2 * r) * b->width + (scale_2
* c)] - 2 * v;
  dxy = ( m->responses[(scale * (r+1)) * m->width + (scale * (c+1))] - m->responses[(scale
* (r+1)) * m->width + (scale * (c-1))] -
          m->responses[(scale * (r-1)) * m->width + (scale * (c+1))] + m->responses[(scale
* (r-1)) * m->width + (scale * (c-1))] ) / 4.0;
  dxs = ( t->responses[r * t->width + (c+1)] - t->responses[r * t->width + (c-1)] -
          b->responses[(scale_2 * r) * b->width + (scale_2 * (c+1))] + b-
>responses[(scale_2 * r) * b->width + (scale_2 * (c-1))] ) / 4.0;
  dys = ( t->responses[(r+1) * t->width + c] - t->responses[(r-1) * t->width + c] -
          b->responses[(scale_2 * (r+1)) * b->width + (scale_2 * c)] + b-
>responses[(scale_2 * (r-1)) * b->width + (scale_2 * c)] ) / 4.0;

  H.rows = 3;
  H.cols = 3;
  H.data[0][0]= dxx;
  H.data[0][1]= dxy;
  H.data[0][2]= dxs;
  H.data[1][0]= dxy;
  H.data[1][1]= dyy;
  H.data[1][2]= dys;
  H.data[2][0]= dxs;
  H.data[2][1]= dys;
  H.data[2][2]= dss;

  //------->edw teleiwnei h hessian3D


  //H_inv = cvCreateMat( 3, 3, CV_64FC1 );
  H_inv.rows=3;
  H_inv.cols=3;

  //cvInvert( H, H_inv, CV_SVD );
  //lu decomposition
  u[1][0]=0;
```

```
  u[2][0]=0;
  u[2][1]=0;
  l[0][0]=1;
  l[1][1]=1;
  l[2][2]=1;
  l[0][1]=0;
  l[0][2]=0;
  l[1][2]=0;

  //lush susthmatos
  u[0][0]=H.data[0][0];
  u[0][1]=H.data[0][1];
  u[0][2]=H.data[0][2];
  l[1][0]=H.data[1][0]/u[0][0];
  u[1][1]=H.data[1][1]-u[0][1]*l[1][0];
  u[1][2]=H.data[1][2]-u[0][2]*l[1][0];
  l[2][0]=H.data[2][0]/u[0][0];
  l[2][1]=(H.data[2][1]-u[0][1]*l[2][0])/u[1][1];
  u[2][2]=H.data[2][2]-(u[0][2]*l[2][0]+u[1][2]*l[2][1]);


  //***1h sthlh
    //bhma 1
    z[0][0]=1;
    z[1][0]= -l[1][0];
    z[2][0]=-l[2][0]+l[2][1]*l[1][0];
    //bhma 2
    H_inv.data[2][0]=z[2][0]/u[2][2];
    H_inv.data[1][0]=(z[1][0]-u[1][2]*(z[2][0]/u[2][2]))/u[1][1];
    H_inv.data[0][0]=(1-u[0][1]*H_inv.data[1][0]-u[0][2]*H_inv.data[2][0])/u[0][0];

    //*** 2h sthlh
    //bhma 1
    z[0][1]=0;
    z[1][1]=1;
    z[2][1]=-l[2][1];
    //bhma 2
    H_inv.data[2][1]=(-l[2][1])/u[2][2];
    H_inv.data[1][1]=(1-u[1][2]*H_inv.data[2][1])/(u[1][1]);
    H_inv.data[0][1]=-(u[0][1]*H_inv.data[1][1]+u[0][2]*H_inv.data[2][1])/u[0][0];

    //*** 3h sthlh
    //bhma 1
    z[0][2]=0;
    z[1][2]=0;
    z[2][2]=1;
    //bhma 2
    H_inv.data[2][2]=1/u[2][2];
    H_inv.data[1][2]=-u[1][2]*H_inv.data[2][2]/u[1][1];
    H_inv.data[0][2]=-(u[0][1]*H_inv.data[1][2]+u[0][2]*H_inv.data[2][2])/u[0][0];

  xi=-H_inv.data[2][0]*dD.data[0][0]-H_inv.data[2][1]*dD.data[1][0]-
H_inv.data[2][2]*dD.data[2][0];
  xr=-H_inv.data[1][0]*dD.data[0][0]-H_inv.data[1][1]*dD.data[1][0]-
H_inv.data[1][2]*dD.data[2][0];
  xc=-H_inv.data[0][0]*dD.data[0][0]-H_inv.data[0][1]*dD.data[1][0]-
H_inv.data[0][2]*dD.data[2][0];
```

```
  // If point is sufficiently close to the actual extremum
  if( fabs( xi ) < 0.5f  &&  fabs( xr ) < 0.5f  &&  fabs( xc ) < 0.5f ) {
    ipt->x = (float)((c + xc) * t->step);
    ipt->y = (float)((r + xr) * t->step);
    ipt->scale = (float)((0.1333f) * (m->filter + xi * filterStep));
       scale = m->width / t->width;
       ipt->laplacian=m->laplacian[(scale * r) * m->width + (scale * c)];
  }

}
```

Και τα αποτελέσματα του synthesis:

Απόδοση	**Περιοχή υλοποίησης**	**Κατανάλωση ισχύος**

Performance Estimates

Summary of timing analysis

Estimated clock period (ns): 8.65

Summary of overall latency (clock cycles)

Best-case latency: 230

Average-case latency: 242

Worst-case latency: 252

Area Estimates

Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	80	23110	32052	-
Expression	-	23	0	598	-
FIFO	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	1817	-
Register	-	-	3868	-	-
ShiftMemory	-	-	-	-	-
Total	0	103	26978	34467	0
Available	832	768	301440	150720	37680
Utilization (%)	0	13	8	22	0

Power Estimate

Summary

	Power
Component	5524
Expression	61
FIFO	-
Memory	-
Multiplexer	181
Register	386
ShiftMemory	-
Total	6152

Πίνακας 7.7

Σημείωση:

Η χρήση παρενθέσεων δίνει προτεραιότητα σε υπολογισμούς και αυτο μπορεί να έχει ως αποτέλεσμα την μείωση του υλικού.Για παραδειγμα μπορούμε να γράψουμε την παρακάτω έκφραση με 2 τρόπους, ανάλογα με το που βάζουμε τις παρενθέσεις:

`H_inv.data[1][0]=(z[1][0]-u[1][2]*(z[2][0]/u[2][2]))/u[1][1];`	Solution 3
`H_inv.data[1][0]=(z[1][0]-(u[1][2]*z[2][0])/u[2][2])/u[1][1];`	Solution 4

Η σύγκριση των 2 solutions δίνει:

Overall performance (clock cycles):

	solution3	solution4
Throughput(II)	252	246
Latency	252	246

Resource Usage

	BRAM_18K		DSP48E		FF		LUT		SLICE	
	solution3	solution4	solution3	solution4	solution3	solution4	solution3	solution4	solution3	solution4
Component	-	-	80	80	23110	25287	32052	34738	-	-
Expression	-	-	23	23	0	0	598	598	-	-
FIFO	-	-	-	-	-	-	-	-	-	-
Memory	-	-	-	-	-	-	-	-	-	-
Multiplexer	-	-	-	-	-	-	1817	1809	-	-
Register	-	-	-	-	3868	3836	-	-	-	-
ShiftMemory	-	-	-	-	-	-	-	-	-	-
Total	0	0	103	103	26978	29123	34467	37145	0	0

Πίνακας 7.8

όπου βλεπουμε σημαντικη διαφορα στη χρήση LUT και FF.

7.3.4 getIpoints()

Η αρχική συνάρτηση ***getIpoints()*** σε C++ είναι:

```
//! Find the image features and write into vector of features
void FastHessian::getIpoints()
{
  // filter index map
  static const int filter_map [OCTAVES][INTERVALS] = {{0,1,2,3}, {1,3,4,5},
{3,5,6,7}, {5,7,8,9}, {7,9,10,11}};

  // Clear the vector of existing ipts
  ipts.clear();

  // Build the response map
  buildResponseMap();

  // Get the response layers
  ResponseLayer *b, *m, *t;
  for (int o = 0; o < octaves; ++o) for (int i = 0; i <= 1; ++i)
  {
    b = responseMap.at(filter_map[o][i]);
    m = responseMap.at(filter_map[o][i+1]);
    t = responseMap.at(filter_map[o][i+2]);

    // loop over middle response layer at density of the most
    // sparse layer (always top), to find maxima across scale and space
    for (int r = 0; r < t->height; ++r)
    {
      for (int c = 0; c < t->width; ++c)
      {
        if (isExtremum(r, c, t, m, b))
        {
```

```
          interpolateExtremum(r, c, t, m, b);
        }
      }
    }
  }
}
```

Έχουμε ήδη υλοποιήσει τις συναρτήσεις που καλούνται μέσα στην getIpoints(), οπότε:

1. εφαρμόζουμε τους μετασχηματισμούς που έχουν αναφερθεί
2. κάνουμε inline τις υλοποιημένες συναρτήσεις και τέλος
3. κάνουμε unroll 2 φορές το εσωτερικό loop και καταλήγουμε στη συνάρτηση

```
#include <math.h>
#include <stdio.h>
#include "responselayer.h"
#include "integral.h"
#include "isextremum.h"
#include "ipoint.h"
#include "interpolateextremum.h"



#include "ap_cint.h"

//define OCTAVES,INTERVALS

//-------------------------------------------------------

//! Find the image features and write into vector of features
int getIpoints( Ipoint *ipt,  IplImage *img ) {


      double thresh=0.0004;
      int octaves=5;
  ResponseLayer responseMap[12];
  int filter_map [5][4] = {{0,1,2,3}, {1,3,4,5}, {3,5,6,7}, {5,7,8,9},
{7,9,10,11}};


      int w =44;
      int h =30;
      int s =2;
      int i, o;
      int step;
      int b ;
      int l;
      int w_2;
      float inverse_area;
      float Dxx, Dyy, Dxy;
      int r,c,ar,ac,index;
      int r2, c2;

  for (i=0;i<12;i++) {
      responseMap[i].inUse = 0;
```

```
}

// Calculate approximated determinant of hessian values
if (octaves >= 1) {
     for (i=0;i<4;i++){
            responseMap[i].width = w;
            responseMap[i].height = h;
            responseMap[i].step = s;
            responseMap[i].filter = 9+6*i;
            responseMap[i].inUse = 1;
     }
}

if (octaves >= 2) {
     responseMap[4].width = w/2;
     responseMap[4].height = h/2;
     responseMap[4].step = s*2;
     responseMap[4].filter = 39;
     responseMap[4].inUse = 1;
     responseMap[5].width = w/2;
     responseMap[5].height = h/2;
     responseMap[5].step = s/2;
     responseMap[5].filter = 51;
     responseMap[5].inUse = 1;
}

if (octaves >= 3){
     responseMap[6].width = w/4;
     responseMap[6].height = h/4;
     responseMap[6].step = s*4;
     responseMap[6].filter = 75;
     responseMap[6].inUse = 1;
     responseMap[7].width = w/4;
     responseMap[7].height = h/4;
     responseMap[7].step = s*4;
     responseMap[7].filter = 99;
     responseMap[7].inUse = 1;
}

if (octaves >= 4) {
     responseMap[8].width = w/8;
     responseMap[8].height = h/8;
     responseMap[8].step = s*8;
     responseMap[8].filter = 147;
     responseMap[8].inUse = 1;
     responseMap[9].width = w/8;
     responseMap[9].height = h/8;
     responseMap[9].step = s*8;
     responseMap[9].filter = 195;
     responseMap[9].inUse = 1;
}

if (octaves >= 5) {
     responseMap[10].width = w/16;
     responseMap[10].height = h/16;
     responseMap[10].step = s*16;
     responseMap[10].filter = 291;
     responseMap[10].inUse = 1;
```

```
      responseMap[11].width = w/16;
      responseMap[11].height = h/16;
      responseMap[11].step = s*16;
      responseMap[11].filter = 387;
      responseMap[11].inUse = 1;
  }

buildResponseMap_label1:for ( i = 0; i < 12; ++i) {

            b = (responseMap[i].filter - 1) / 2;
            l = responseMap[i].filter / 3;
            w_2 = responseMap[i].filter;
            inverse_area = 1.f/(w_2*w_2);
            index=0;

buildResponseMap_label3:for( ar = 0; ar < 60; ++ar){
buildResponseMap_label2:for( ac = 0; ac < 89/2; ++ac) {
                        // get the image coordinates
                        r = ar * responseMap[i].step;
                        c = 2*ac * responseMap[i].step;

                        // Compute response components
                        Dxx = BoxIntegral(img, r - l + 1, c - b, 2*l - 1,
responseMap[i].filter)
                              - BoxIntegral(img, r - l + 1, c - l / 2,
2*l - 1, l)*3;
                        Dyy = BoxIntegral(img, r - b, c - l + 1,
responseMap[i].filter, 2*l - 1)
                              - BoxIntegral(img, r - l / 2, c - l + 1,
l, 2*l - 1)*3;
                        Dxy = + BoxIntegral(img, r - l, c + 1, l, l)
                              + BoxIntegral(img, r + 1, c - l, l, l)
                              - BoxIntegral(img, r - l, c - l, l, l)
                              - BoxIntegral(img, r + 1, c + 1, l, l);


                        // Normalise the filter responses with respect to
their size
                        Dxx *= inverse_area;
                        Dyy *= inverse_area;
                        Dxy *= inverse_area;

                        // Get the determinant of hessian response &
laplacian sign
                        responseMap[i].responses[index] = (Dxx * Dyy -
0.81f * Dxy * Dxy);
                        if (Dxx+Dyy >=0)
                              responseMap[i].laplacian[index] = 1;
                        else
                              responseMap[i].laplacian[index] = 0;

                        responseMap[i].coord1[index]=r;
                        responseMap[i].coord2[index]=c;
                        index++;

                        ////////////////////////////////////

                        c = (2*ac+1) * responseMap[i].step;
```

```
                        // Compute response components
                        Dxx = BoxIntegral(img, r - l + 1, c - b, 2*l - 1,
responseMap[i].filter)
                              - BoxIntegral(img, r - l + 1, c - l / 2,
2*l - 1, l)*3;
                        Dyy = BoxIntegral(img, r - b, c - l + 1,
responseMap[i].filter, 2*l - 1)
                              - BoxIntegral(img, r - l / 2, c - l + 1,
l, 2*l - 1)*3;
                        Dxy = + BoxIntegral(img, r - l, c + 1, l, l)
                              + BoxIntegral(img, r + 1, c - l, l, l)
                              - BoxIntegral(img, r - l, c - l, l, l)
                              - BoxIntegral(img, r + 1, c + 1, l, l);


                        // Normalise the filter responses with respect to
their size
                        Dxx *= inverse_area;
                        Dyy *= inverse_area;
                        Dxy *= inverse_area;

                        // Get the determinant of hessian response &
laplacian sign
                        responseMap[i].responses[index] = (Dxx * Dyy -
0.81f * Dxy * Dxy);
                        if (Dxx+Dyy >=0)
                              responseMap[i].laplacian[index] = 1;
                        else
                              responseMap[i].laplacian[index] = 0;

                        responseMap[i].coord1[index]=r;
                        responseMap[i].coord2[index]=c;
                        index++;
                  }
            }
      }
  //--------edw teleiwnei h buildResponseMap()

  // Get the response layers
  ResponseLayer *b2, *m, *t;
  for ( o = 0; o < octaves; ++o)
  {

            b2 = &responseMap[filter_map[o][0]];
            m = &responseMap[filter_map[o][1]];
            t = &responseMap[filter_map[o][2]];

            // loop over middle response layer at density of the most
            // sparse layer (always top), to find maxima across scale and
space
            for ( r2 = 0; r2 < t->height; ++r2)
            {

              for ( c2 = 0; c2 < t->width; ++c2)
              {
                    if (isExtremum(r2, c2, thresh, t, m, b2))
                    {
```

```
                        interpolateExtremum(ipt, r, c, t, m, b2);
                    }
                }
            }
            b2 = &responseMap[filter_map[o][1]];
            m = &responseMap[filter_map[o][2]];
            t = &responseMap[filter_map[o][3]];

            // loop over middle response layer at density of the most
            // sparse layer (always top), to find maxima across scale and
space
            for ( r2 = 0; r2 < t->height; ++r2)
            {

              for ( c2 = 0; c2 < t->width; ++c2)
              {
                   if (isExtremum(r2, c2, thresh, t, m, b2))
                   {
                     interpolateExtremum(ipt, r, c, t, m, b2);
                   }
              }
            }


  }

  return 0;
}



float BoxIntegral(IplImage *img, int row, int col, int rows, int cols) {
      float A,B,C,D;
      int step = (img->widthStep)/sizeof(float);
      int i;

  // The subtraction by one for row/col is because row/col is inclusive.
      int r1 = min(row,img->height) - 1;
      int c1 = min(col,img->width)  - 1;
      int r2 = min(row + rows,img->height) - 1;
      int c2 = min(col + cols,img->width)  - 1;

      A=0.0f;
      B=0.0f;
      C=0.0f;
      D=0.0f;
      if (r1 >= 0 && c1 >= 0) A = (float) img->imageData[r1 * step + c1];
      if (r1 >= 0 && c2 >= 0) B = (float) img->imageData[r1 * step + c2];
      if (r2 >= 0 && c1 >= 0) C = (float) img->imageData[r2 * step + c1];
      if (r2 >= 0 && c2 >= 0) D = (float) img->imageData[r2 * step + c2];

      if ((A - B - C + D)>0)
            return (A - B - C + D);
      else
            return 0;
}
```

```
int min(int a, int b) {

	if (a>b)
		return b;
	else
		return a;
}


int isExtremum(int r, int c, double thresh, ResponseLayer *t, ResponseLayer
*m, ResponseLayer *b) {

  int rr, cc;
  // bounds check+check the candidate point in the middle layer is above
thresh
  int layerBorder = (t->filter + 1)* (1 / (2 * t->step));
  int scale = m->width *(1/ t->width);
  float candidate = m->responses[(scale * r) * m->width + (scale * c)];
  if (r <= layerBorder || r >= t->height - layerBorder || c <= layerBorder
|| c >= t->width - layerBorder || candidate < thresh )
    return 0;


  int scale_b = b->width*(1 / t->width);

  //rr=-1, cc=-1
  if (
	t->responses[(r-1) * t->width + c-1] >= candidate ||
	m->responses[(scale * (r-1)) * m->width + (scale * (c-1))] >=
candidate ||
	b->responses[(scale_b * (r-1)) * b->width + (scale_b * (c-1))] >=
candidate
	)
	return 0;

  //rr=-1, cc=1
  if (
	t->responses[(r-1) * t->width + c+1] >= candidate ||
    m->responses[(scale * (r-1)) * m->width + (scale * (c+1))] >= candidate
||
	b->responses[(scale_b * (r-1)) * b->width + (scale_b * (c+1))] >=
candidate
	)
	return 0;

  //rr=1, cc=-1
  if (
	t->responses[(r+1) * t->width + c-1] >= candidate ||
    m->responses[(scale * (r+1)) * m->width + (scale * (c-1))] >= candidate
||
	b->responses[(scale_b * (r+1)) * b->width + (scale_b * (c-1))] >=
candidate
	)
	return 0;


  //rr=1, cc=1
  if (
```

```
      t->responses[(r+1) * t->width + c+1] >= candidate ||
      m->responses[(scale * (r+1)) * m->width + (scale * (c+1))] >=
candidate ||
      b->responses[(scale_b * (r+1)) * b->width + (scale_b * (c+1))] >=
candidate
      )
      return 0;

  return 1;
}


//! Interpolate scale-space extrema to subpixel accuracy to form an image
feature.
void interpolateExtremum(Ipoint *ipt, int r, int c, ResponseLayer *t,
ResponseLayer *m, ResponseLayer *b) {
  // get the step distance between filters
  // check the middle filter is mid way between top and bottom
   int filterStep = (m->filter - b->filter);
  int i, j, k;
  double u[3][3],l[3][3], z[3][3];

  // Get the offsets to the actual location of the extremum
  double xi = 0, xr = 0, xc = 0;
  //interpolateStep(r, c, t, m, b, &xi, &xr, &xc );
  // Performs one step of extremum interpolation.
  CvMat dD ,  H, H_inv, X ;
  double x[3] = { 0 };

  //dD = deriv3D( r, c, t, m, b );
 //<-----edw 3ekinaei h deriv3D
 double dx, dy, ds;

  int scale = m->width*(1 / t->width);
  //edw to width anaferetai se kapoio global width h einai to m.width?
  dx = (m->responses[(scale * r) * m->width + (scale * (c+1))] - m-
>responses[(scale * r) * m->width + (scale * (c-1))]) / 2.0;
  dy = (m->responses[(scale * (r+1)) * m->width + (scale * c)] - m-
>responses[(scale * (r-1)) * m->width + (scale * c)]) / 2.0;
  int scale_2 = b->width *(1/ t->width);
  ds = (t->responses[r * t->width + c] - b->responses[(scale_2 * r) * b-
>width + (scale_2 * c)]) / 2.0;

  dD.rows = 3;
  dD.cols = 1;
  dD.data[0][0] = dx;
  dD.data[1][0] = dy;
  dD.data[2][0] = ds;
  //edw teleiwnei h deriv3D--->*/

   //<----- edw 3ekinaei h hessian3D

  double v, dxx, dyy, dss, dxy, dxs, dys;

  v = m->responses[(scale * r) * m->width + (scale * c)];
  dxx = m->responses[(scale * r) * m->width + (scale * (c+1))] + m-
>responses[(scale * r) * m->width + (scale * (c-1))] - 2 * v;
  dyy = m->responses[(scale * (r+1)) * m->width + (scale * c)] + m-
```

```
>responses[(scale * (r-1)) * m->width + (scale * c)] - 2 * v;
  dss = t->responses[r * t->width + c] - b->responses[(scale_2 * r) * b-
>width + (scale_2 * c)] - 2 * v;
  dxy = ( m->responses[(scale * (r+1)) * m->width + (scale * (c+1))] - m-
>responses[(scale * (r+1)) * m->width + (scale * (c-1))] -
          m->responses[(scale * (r-1)) * m->width + (scale * (c+1))] + m-
>responses[(scale * (r-1)) * m->width + (scale * (c-1))] ) / 4.0;
  dxs = ( t->responses[r * t->width + (c+1)] - t->responses[r * t->width +
(c-1)] -
          b->responses[(scale_2 * r) * b->width + (scale_2 * (c+1))] + b-
>responses[(scale_2 * r) * b->width + (scale_2 * (c-1))] ) / 4.0;
  dys = ( t->responses[(r+1) * t->width + c] - t->responses[(r-1) * t-
>width + c] -
          b->responses[(scale_2 * (r+1)) * b->width + (scale_2 * c)] + b-
>responses[(scale_2 * (r-1)) * b->width + (scale_2 * c)] ) / 4.0;

  H.rows = 3;
  H.cols = 3;
  H.data[0][0]= dxx;
  H.data[0][1]= dxy;
  H.data[0][2]= dxs;
  H.data[1][0]= dxy;
  H.data[1][1]= dyy;
  H.data[1][2]= dys;
  H.data[2][0]= dxs;
  H.data[2][1]= dys;
  H.data[2][2]= dss;

  //------->edw teleiwnei h hessian3D


  //H_inv = cvCreateMat( 3, 3, CV_64FC1 );
  H_inv.rows=3;
  H_inv.cols=3;

    //lu decomposition
  u[1][0]=0;
  u[2][0]=0;
  u[2][1]=0;
  l[0][0]=1;
  l[1][1]=1;
  l[2][2]=1;
  l[0][1]=0;
  l[0][2]=0;
  l[1][2]=0;

  //lush susthmatos
  u[0][0]=H.data[0][0];
  u[0][1]=H.data[0][1];
  u[0][2]=H.data[0][2];
  l[1][0]=H.data[1][0]*(1/u[0][0]);
  u[1][1]=H.data[1][1]-u[0][1]*l[1][0];
  u[1][2]=H.data[1][2]-u[0][2]*l[1][0];
  l[2][0]=H.data[2][0]*(1/u[0][0]);
  l[2][1]=(H.data[2][1]-u[0][1]*l[2][0])*(1/u[1][1]);
  u[2][2]=H.data[2][2]-(u[0][2]*l[2][0]+u[1][2]*l[2][1]);

  //***1h sthlh
```

```
    //bhma 1
    z[0][0]=1;
    z[1][0]= -l[1][0];
    z[2][0]=-l[2][0]+l[2][1]*l[1][0];
    //bhma 2
    H_inv.data[2][0]=z[2][0]*(1/u[2][2]);
    H_inv.data[1][0]=(z[1][0]-u[1][2]*(z[2][0]/u[2][2]))*(1/u[1][1]);
    H_inv.data[0][0]=(1-u[0][1]*H_inv.data[1][0]-
u[0][2]*H_inv.data[2][0])*(1/u[0][0]);

    //*** 2h sthlh
    //bhma 1
    z[0][1]=0;
    z[1][1]=1;
    z[2][1]=-l[2][1];
    //bhma 2
    H_inv.data[2][1]=(-l[2][1])*(1/u[2][2]);
    H_inv.data[1][1]=(1-u[1][2]*H_inv.data[2][1])*(1/(u[1][1]));
    H_inv.data[0][1]=-
(u[0][1]*H_inv.data[1][1]+u[0][2]*H_inv.data[2][1])*(1/u[0][0]);

    //*** 3h sthlh
    //bhma 1
    z[0][2]=0;
    z[1][2]=0;
    z[2][2]=1;
    //bhma 2
    H_inv.data[2][2]=1/u[2][2];
    H_inv.data[1][2]=-u[1][2]*H_inv.data[2][2]*(1/u[1][1]);
    H_inv.data[0][2]=-
(u[0][1]*H_inv.data[1][2]+u[0][2]*H_inv.data[2][2])*(1/u[0][0]);


  xi=-H_inv.data[2][0]*dD.data[0][0]-H_inv.data[2][1]*dD.data[1][0]-
H_inv.data[2][2]*dD.data[2][0];
  xr=-H_inv.data[1][0]*dD.data[0][0]-H_inv.data[1][1]*dD.data[1][0]-
H_inv.data[1][2]*dD.data[2][0];
  xc=-H_inv.data[0][0]*dD.data[0][0]-H_inv.data[0][1]*dD.data[1][0]-
H_inv.data[0][2]*dD.data[2][0];


  // If point is sufficiently close to the actual extremum
  if( fabs( xi ) < 0.5f  &&  fabs( xr ) < 0.5f  &&  fabs( xc ) < 0.5f ) {
    ipt->x = (float)((c + xc) * t->step);
    ipt->y = (float)((r + xr) * t->step);
    ipt->scale = (float)((0.1333f) * (m->filter + xi * filterStep));
      scale = m->width / t->width;
      ipt->laplacian=m->laplacian[(scale * r) * m->width + (scale * c)];
  }

  }
```

Την οποία κάνουμε synthesize:

Απόδοση	Περιοχή υλοποίησης	Κατανάλωση ισχύος

Performance Estimates

Summary of timing analysis

Estimated clock period (ns): 8.74

Summary of overall latency (clock cycles)

Best-case latency: 12452323

Average-case latency: 12821473

Worst-case latency: 13122223

Area Estimates

Summary

	BRAM_18K	DSP48E	FF	LUT	SLICE
Component	-	150	17409	20281	-
Expression	-	5	0	396	-
FIFO	-	-	-	-	-
Memory	140	-	0	0	-
Multiplexer	-	-	-	704	-
Register	-	-	1274	-	-
ShiftMemory	-	-	-	-	-
Total	140	155	18683	21381	0
Available	832	768	301440	150720	37680
Utilization (%)	16	20	6	14	0

Power Estimate

Summary

	Power
Component	3585
Expression	39
FIFO	-
Memory	14
Multiplexer	70
Register	127
ShiftMemory	-
Total	3835

Πίνακας 7.9

Και αυτό είναι το αποτέλεσμα της τελικής υλοποίησης του Detector, όσον αφορά το latency, το hardware που θα χρησιμοποιήσουμε και την ισχύ που καταναλώνεται.

7.4 Σύνοψη

Συνολικά στην παραπάνω υλοποίηση εφαρμόστηκαν οι ακόλουθοι μετασχηματισμοί:

- function inline
- loop unrolling
- χρήση παρενθέσεων για να δωθεί προτεραιότητα σε πράξεις για τις οποίες το Vivado hls μας παρέχει έτοιμους πυρήνες
- εξάλειψη νεκρού κώδικα (dead code elimination),που μπορεί να προέκυπτε από άλλους μετασχηματισμούς
- pipiline
- merging εντολών συνθήκης
- απλοποίηση εκφράσεων
- εξάλειψη variable propagation

Επίσης πρέπει να αναφερθεί το γεγονός ότι η rtl σχεδίαση με το χέρι δίνει καλύτερα αποτελέσματα από την σύνθεση υψηλού επιπέδου και αυτό επιβεβαιώνεται και από την παραπάνω υλοποίηση. Αυτό συμβαίνει γιατί:

1) πολλοί μετασχηματισμοί στην πράξη είναι δύσκολο να εφαρμοστούν. Οι πραγματικές εφαρμογές, όπως στην συγκεκριμένη περίπτωση ο αλγόριθμος OpenSURF, περιέχουν πολλές εξαρτήσεις δεδομένων. Οπότε μέσω των σύγχρονων εργαλείων σύνθεσης υψηλού επιπέδου πολλές φορές δεν μπορούμε να εφαρμόσουμε αυτούς τους μετασχηματισμούς, γεγονός που περιορίζει τις δυνατότητες βελτιστοποίησης της απόδοσης.

2) κατά την σύνθεση με το Vivado HLS, στην rtl που παράγεται υλοποιούνται και κάποια σήματα ελέγχου. Έτσι μεγάλο μέρος του hardware που παράγεται οφείλεται σε πράξεις που γίνονται για την δημιουργία αυτών των σημάτων.

Κεφάλαιο 8

Συμπεράσματα και μελλοντική έρευνα

Επισκόπηση

Στο κεφάλαιο αυτό παρουσιάζονται τα συμπεράσματα στα οποία καταλήξαμε κάνοντας χρήση του εργαλείου Vivado HLS για την υλοποίηση του αλγορίθμου OpenSURF και προτείνεται περαιτέρω έρευνα που θα μπορούσε να γίνει για την βελτίωση της συγκεκριμένης εργασίας.

8.1 Συμπεράσματα

Σε αυτή την διπλωματική εργασία παρουσιάζεται η υλοποίηση του αλγορίθμου OpeSURF σε πλατφόρμα ανάπτυξης Xilinx Virtex-6. Η όλη διαδικασία, από τα πρώτα στάδια της ανάλυσης των προδιαγραφών και του διαχωρισμού του hardware / software μέρους του αλγορίθμου, μέχρι τα τελευταία στάδια της υλοποίησης και της επαλήθευσης της ορθής λειτουργίας του, παρείχε μια πρώτη κατανόηση των δυνατοτήτων που δίνονται στον σχεδιαστή μέσω των σύγχρονων εργαλείων της High-Level Synthesis, αλλά και τις πολλές δυνατότητες για περαιτέρω ανάπτυξή τους.

Το εργαλείο Vivado HLS που χρησιμοποιήθηκε για το στάδιο της υλοποίησης αυτής της διατριβής, προσφέρει μεγάλη αυτοματοποίηση της διαδικασίας που απαιτείται για την rtl σύνθεση αλλά και το verification. Αυτό φέρνει το rapid prototyping σε άλλο επίπεδο και επιτρέπει να επιτευχθούν σύντομες time-to-market προθεσμίες και με μεγαλύτερη αποτελεσματικότητα. Ταυτόχρονα συμβάλλει στη μείωση πολλών λαθών που γίνονται κατά την σχεδίαση, γιατί δίνει την δυνατότητα η υλοποίηση να γίνει μέσω περιγραφής σε γλώσσα υψηλού επιπέδου. Αυτό καθιστά την σύνθεση υψηλού επιπέδου διαθέσιμη και σε μη ειδικούς. Επίσης, πολύ σημαντικό χαρακτηριστικό του Vivado HLS είναι η δυνατότητα που δίνει για υλοποίηση floating point τύπων, αλλά και ακεραίων μεταβλητού μήκους.

Ωστόσο, εξακολουθούν να υπάρχουν μεγάλες προκλήσεις για τον σχεδιαστή. Η περιγραφή σε C, στην συγκεκριμένη εργασία, πρέπει να περάσει από αρκετούς μετασχηματισμούς για να μας δώσει ικανοποιητικά αποτελέσματα. Κάποιοι από αυτούς τους μετασχηματισμούς εφαρμόζονται καλύτερα όταν γίνονται από τον ίδιο τον σχεδιαστή με το χέρι, παρά όταν γίνονται αυτόματα μέσω του Vivado HLS.

8.2 Μελλοντική Εργασία

Πάνω στην συγκεκριμένη διπλωματική εργασία θα μπορούσε να γίνει περαιτέρω δουλειά ώστε να βελτιωθεί. Μια ιδέα είναι η εξής:

Λόγω της στατικής δέσμευσης μνήμης δεν μπορούμε να υλοποιήσουμε τον αλγόριθμο OpenSURF με το Vivado HLS για μεγάλες εικόνες, όπως για μία εικόνα 500x400. Αυτό οφείλεται στους περιορισμούς που έχουμε σε πόρους υλικού. Οπότε αναγκαζόμαστε είτε να κάνουμε την υλοποίηση μόνο για μικρότερες εικόνες, είτε αν την κάνουμε για επιθυμητό το μέγεθος, να δίνουμε λιγότερα bit σε μεταβλητές μεγάλου μήκους και κατά συνέπεια να χάνουμε σε ακρίβεια. Μία ιδέα είναι η μεγάλη εικόνα να χωρίζεται σε μικρότερες και ο αλγόριθμος να εφαρμόζεται σε κάθε μία από αυτές χωριστά. Άρα θα μπορούσε να μελετηθεί πόσο χάνει σε ακρίβεια ο αλγόριθμος όταν εφαρμόζεται σε εικόνα τμηματικά, με ποιόν τρόπο μπορεί να επιτευχθεί βελτίωση των αποτελεσμάτων, με τι κριτήρια θα διαχωρίζεται η εικόνα ,αν αυτός ο διαχωρισμός θα γίνεται στο software ή στο hardware, καθώς και η απόδοση αυτής της υλοποίησης.

Κεφάλαιο 9

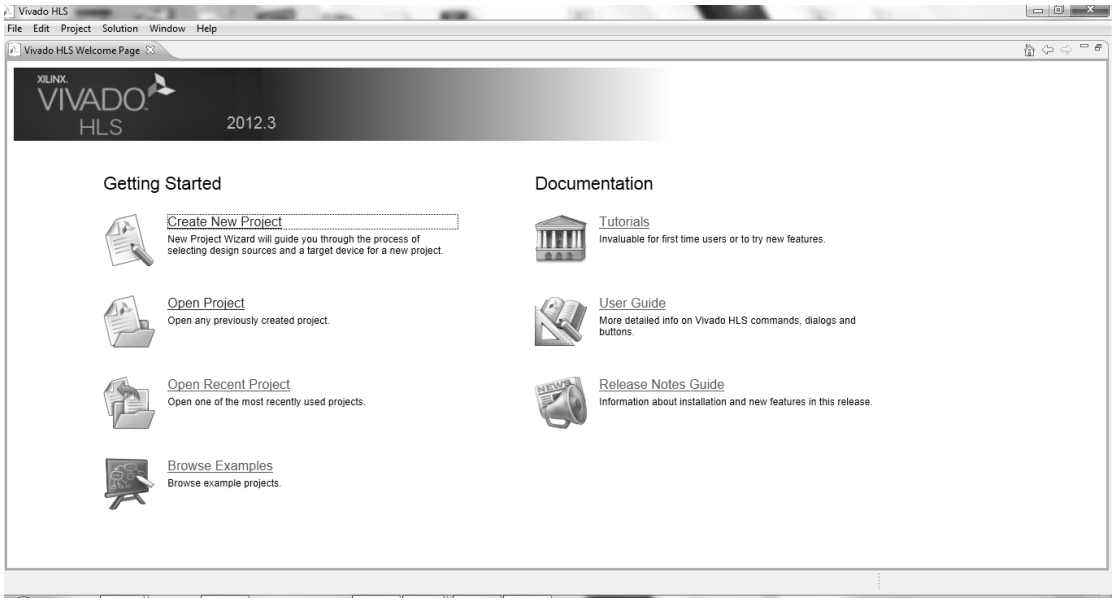
Παράρτημα

Επισκόπηση

Στο κεφάλαιο αυτό θα γίνει περιγραφή της διαδικασίας που ακολουθήθηκε μέσω των εργαλείων Vivado HLS και XPS για την σύνθεση υψηλού επιπέδου, την δημιουργία του Pcore και την ένωση του με τον software επεξεργαστή. Η συνάρτηση που υλοποιείται εδώ, αποτελεί απλό παράδειγμα, έτσι ώστε να είναι ευκολότερη η κατανόηση της διαδικασίας.

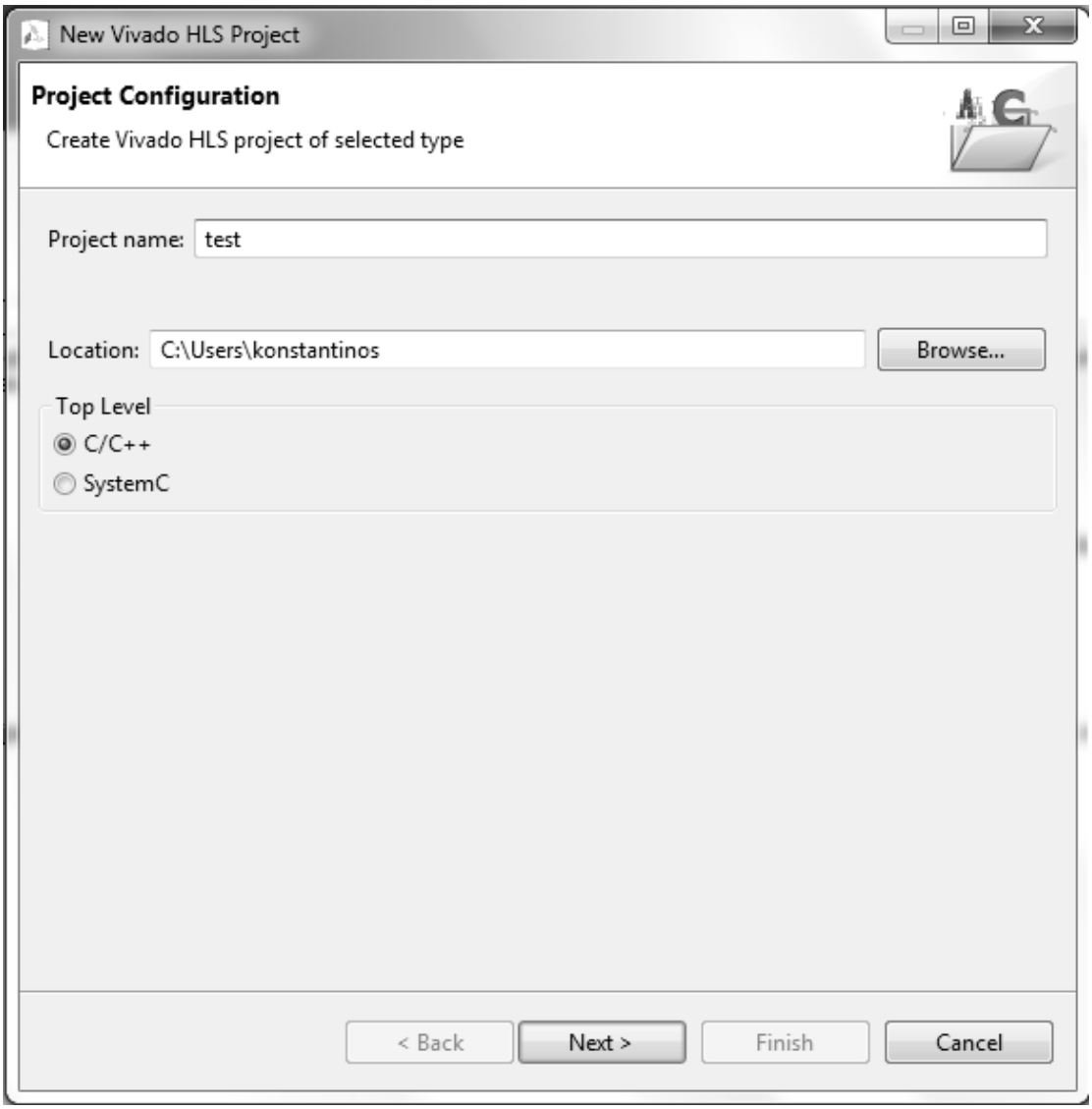
9.1 Δημιουργία νέου project

Αρχικά, αφού ανοίξουμε το Vivado HLS, επιλέγουμε create new project για να αρχίσει η δημιουργία του δικού μας project.



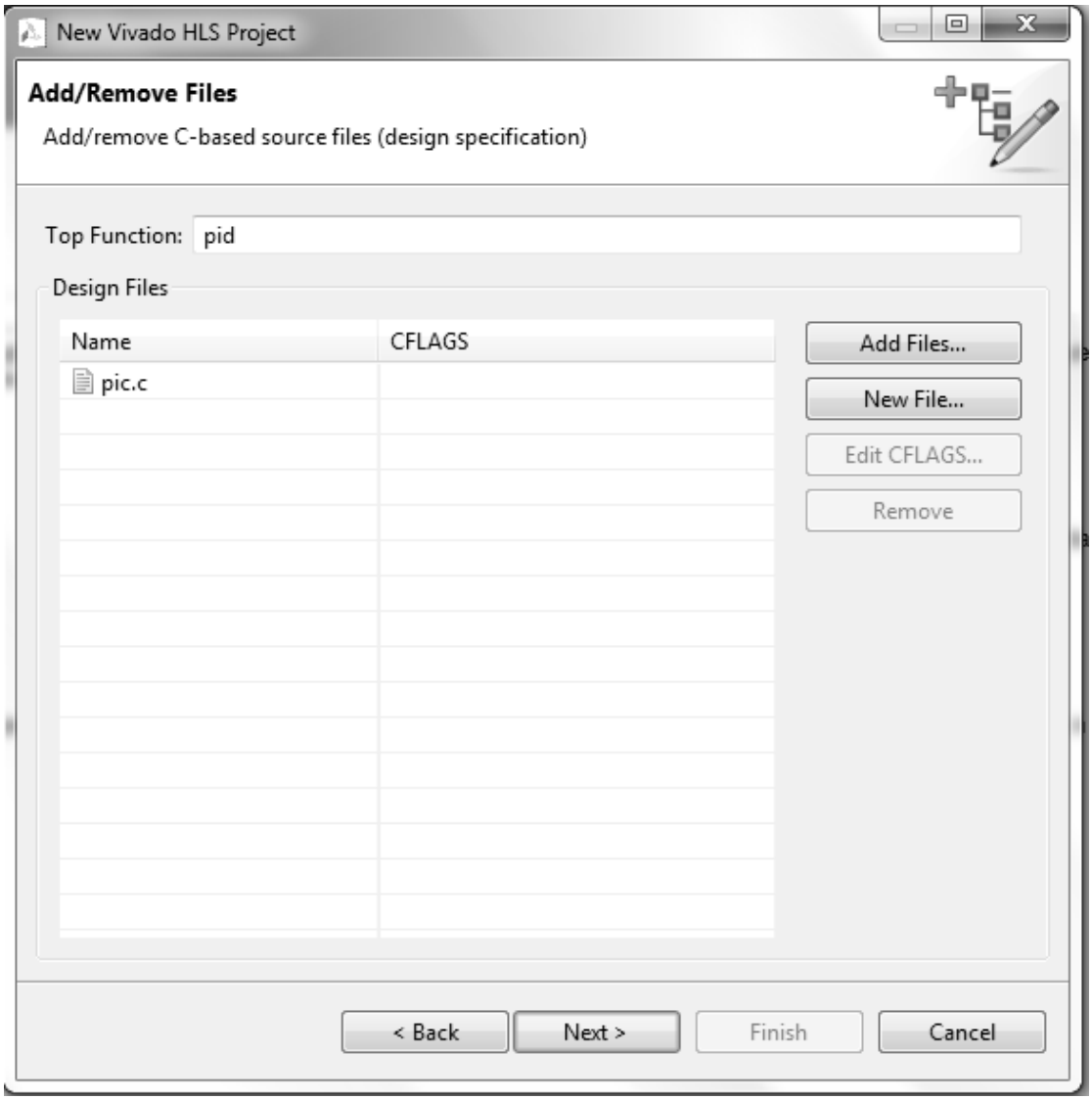
Σχήμα 9.1

Στην συνέχεια, αφού ονομάσουμε το project μας, πατάμε Browse και διαλέγουμε την τοποθεσία όπου θα αποθηκεύονται τα αρχεία του. Επίσης, επιλέγουμε C/C++ αφού ο κώδικας είναι γραμμένος σε C.


New Vivado HLS Project
Project Configuration
Create Vivado HLS project of selected type
Project name: test
Location: C:\Users\konstantinos
Browse...
Top Level
C/C++
SystemC
< Back
Next >
Finish
Cancel

Σχήμα 9.2

Πατώντας Next φτάνουμε στο σημείο που πρέπει να καθορίσουμε τα αρχεία που θα χρησιμοποιήσουμε για τη σύνθεση.



Σχήμα 9.3

Η συνάρτηση pid που βρίσκεται στο αρχείο pic.c είναι:

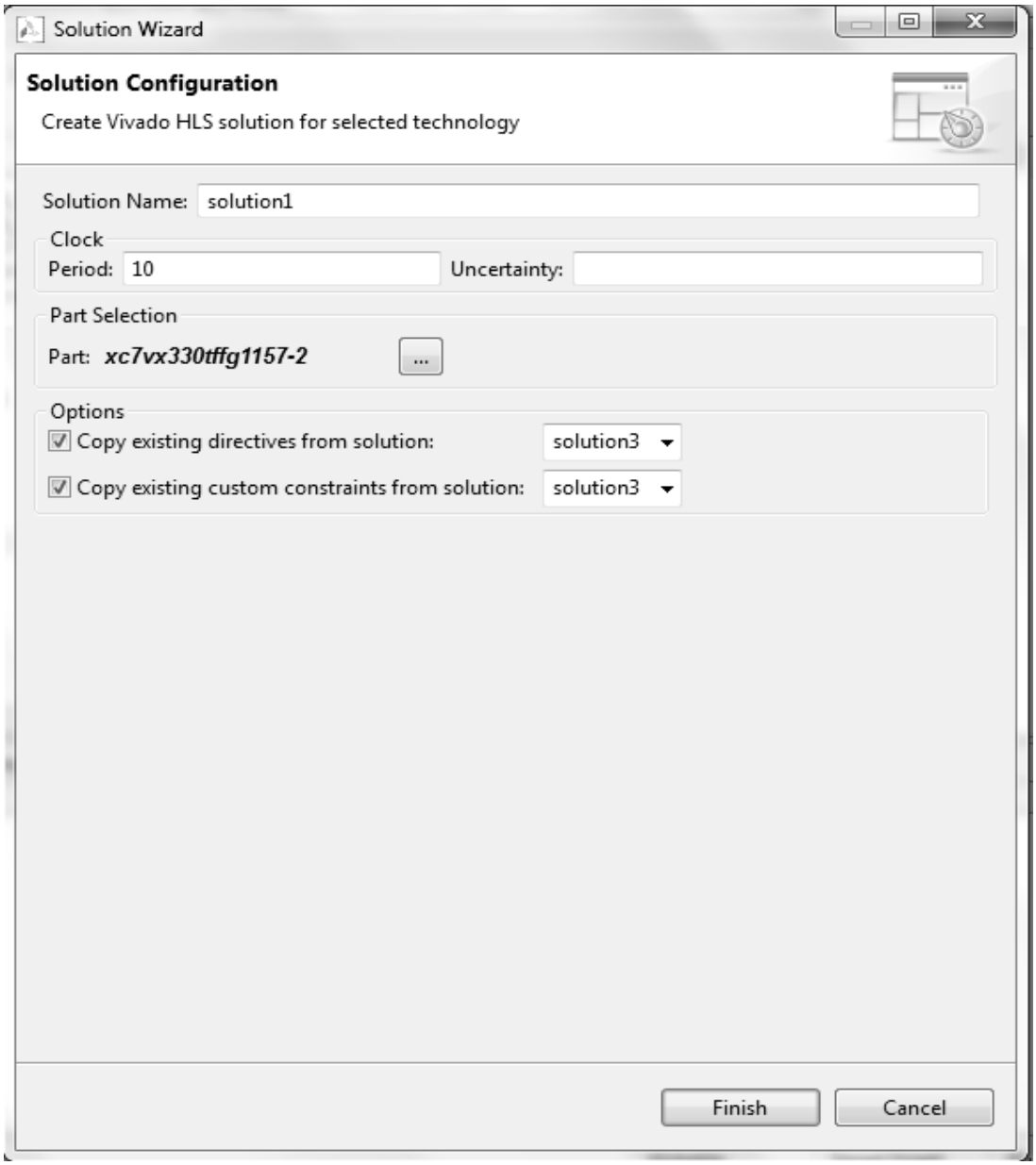
```
int pid(float y, float kp, float ki, float kd, float yp, float *u)
{
	float e, p, i, d;
	static float laste = 0;
	static float sum = 0;

	e = y - yp;
	p = kp * e;
	sum = sum + e;
	i = ki * sum;
	d = kd * (e - laste);
	laste = e;
	*u = p + i + d;
	return 0;
```

```
}
```

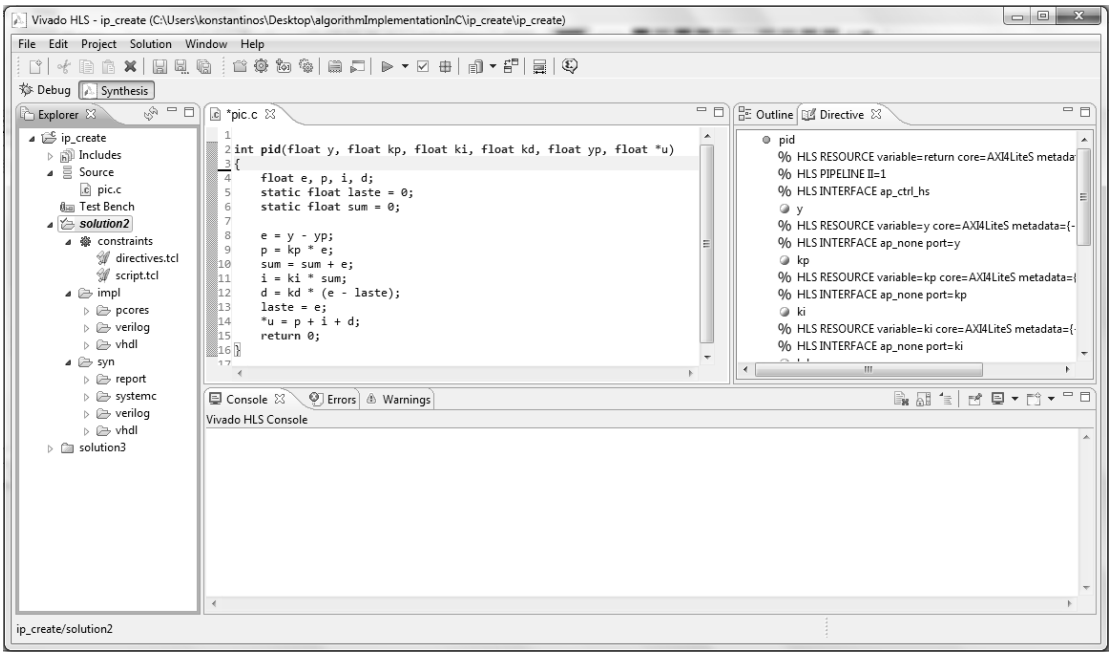
Αφού καθορίσουμε την top function που θα συνθέσουμε μετά, αν υπάρχουν, μπορούμε να προσθέσουμε αρχεία TestBench τα οποία ουσιαστικά ελέγχουν εάν ο κώδικας που κάναμε σύνθεση δουλεύει σωστά. Εμείς δεν χρησιμοποιούμε testbench, συνεπώς το συγκεκριμένο πεδίο παραμένει άδειο.

Τέλος, ονομάζουμε την λύση που θα προκύψει με συγκεκριμένο όνομα, καθώς στη συνέχεια μπορεί να προκύψουν και άλλες λύσεις του ίδιου project με διαφορετικό FPGA, περιορισμούς, ταχύτητα ή άλλα τεχνικά χαρακτηριστικά. Επίσης επιλέγουμε το FPGA που θα χρησιμοποιήσουμε, στην προκειμένη περίπτωση το Virtex6. Πατώντας Finish, η δημιουργία του project ολοκληρώνεται και ανοίγει το περιβάλλον του Vivado HLS μαζί με όλες τις πληροφορίες και τα αρχεία που ορίσαμε κατά τη δημιουργία του.



Σχήμα 9.4

Πατάμε Finish και πηγαίνουμε στην παρακάτω οθόνη



Σχήμα 9.5

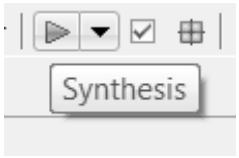
9.2 Σύνθεση

Προς το παρών, θα επικεντρωθούμε στα βήματα που θα ακολουθήσουμε ώστε να δημιουργήσουμε ένα Pcore με AXI bus από το Vivado HLS, γι'αυτό θα πρέπει να επικεντρωθούμε ιδιαιτέρως στη δημιουργία του AXI bus που θα ενώνει τον επεξεργαστή με το Pcore για να επικοινωνούν. Για αυτό το λόγο προσθέτουμε στο project τα παρακάτω directives, κάνοντας τα insert στο πεδίο directives που βρίσκεται πάνω δεξιά:

```
############################################################
## This file is generated automatically by Vivado HLS.
## Please DO NOT edit it.
## Copyright (C) 2012 Xilinx Inc. All rights reserved.
############################################################
set_directive_interface -mode ap_none "pid" y
set_directive_interface -mode ap_none "pid" kp
set_directive_interface -mode ap_none "pid" ki
set_directive_interface -mode ap_none "pid" kd
set_directive_interface -mode ap_none "pid" yp
set_directive_interface -mode ap_vld "pid" u
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid"
return
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid" y
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid"
kp
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid"
ki
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid"
kd
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid"
yp
set_directive_resource -core AXI4LiteS -metadata {-bus_bundle slv0} "pid" u
```

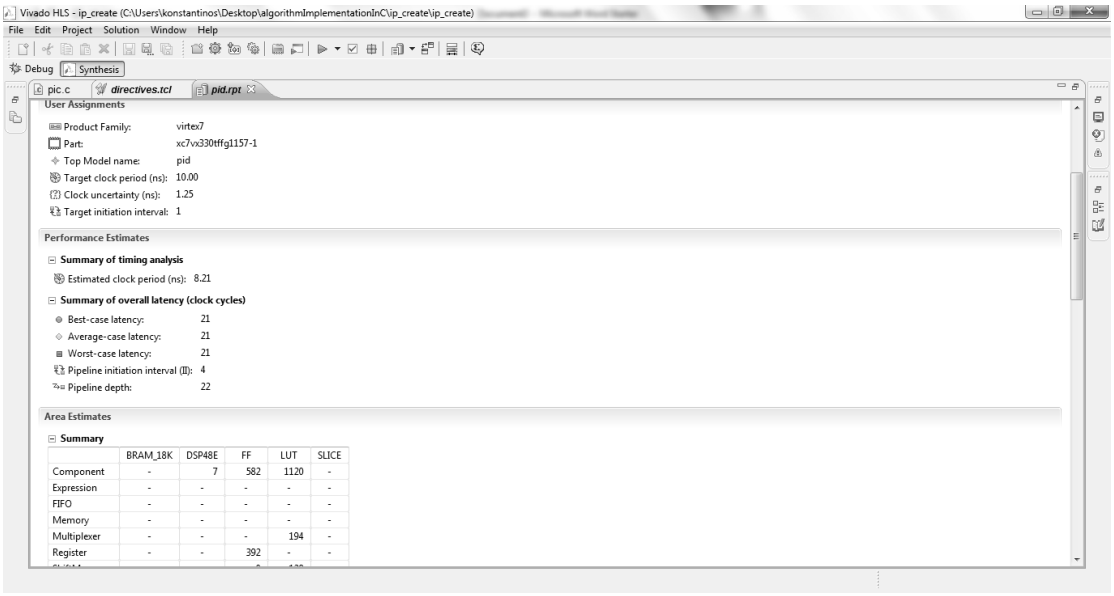
```
set_directive_pipeline -II 1 "pid"
set_directive_reset "pid" laste
set_directive_reset "pid" sum
set_directive_interface -mode ap_ctrl_hs "pid"
```

Στη συνέχεια πατάμε την επιλογή synthesis από τη γραμμή εργαλείων,



Σχήμα 9.6

η σύνθεση ξεκινάει και μόλις ολοκληρωθεί επιτυχώς εμφανίζονται τα αποτελέσματά της :



Σχήμα 9.7

πλέον ο χρήστης μπορεί να δει τα αποτελέσματα της σύνθεσης και να αποφασίσει ποιούς μετασχηματισμούς θα εφαρμώσει ώστε να βελτιώσει την απόδοση της υλοποίησης.

9.3 Export RTL

Έχοντας κάνει σύνθεση, πατάμε το πλήκτρο Export RTL και εμφανίζεται το παράθυρο:

Σχήμα 9.8

Επιλέγουμε Pcore for EDK, και για γλώσσα ανάλογα με το τι θέλουμε verilog ή vhdl και πατώντας ok δημιουργείται το Pcore στον impl φάκελο λύσης (solution).

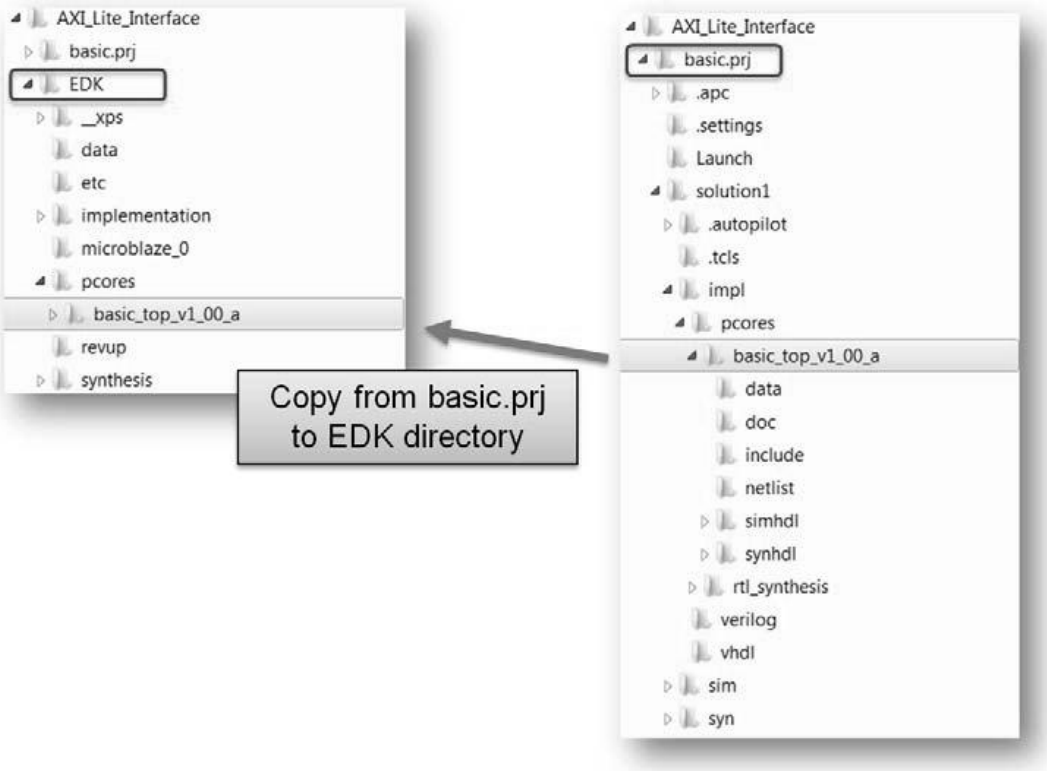
Όταν ολοκληρωθεί η διαδικασία του export RTL, αμέσως θα δημιουργηθεί ένα Pcore, το οποίο στη συνέχεια και θα συνδέσουμε στον Microblaze μέσω ενός άλλου εργαλείου της Xilinx, του Platform Studio (XPS).

Συνοψίζοντας με την δημιουργία του Pcore, έχουμε πλέον δημιουργήσει ένα περιφερειακό σε γλώσσα περιγραφής υλικού, το οποίο μπορούμε να συνδέσουμε κατευθείαν με το υπόλοιπο ενσωματωμένο σύστημα.

9.4 Ενσωμάτωση του Pcore

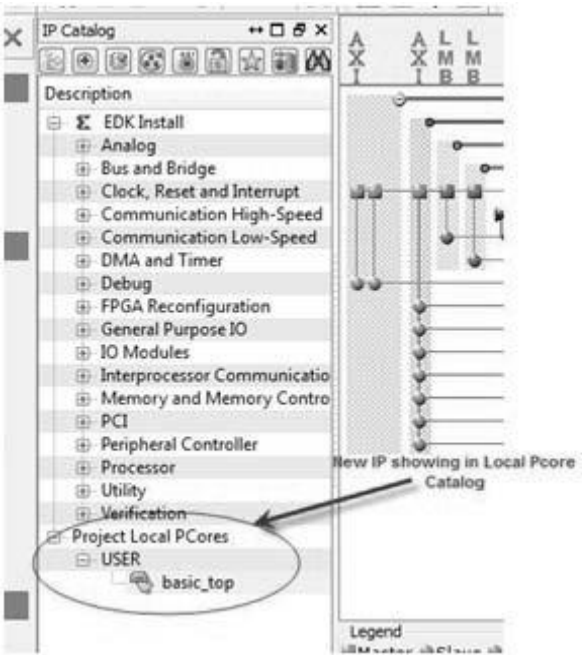
Για να ενσωματώσουμε το Pcore που δημιουργήσαμε στον Microblaze επεξεργαστή χρησιμοποιώντας το εργαλείο XPS ακολουθούμε την διαδικασία:

- Αντιγράφουμε τον φάκελο με το Pcore που φτιάξαμε από τον φάκελο του Vivado HLS στον φάκελο του XPS, όπως φαίνεται στο σχήμα.
-



Σχήμα 9.9

- Στο XPS, προσθέτουμε το Pcore από τον IP catalog επιλέγοντας το. Το νέο Pcore εμφανίζεται κάτω από το Project Local Pcores

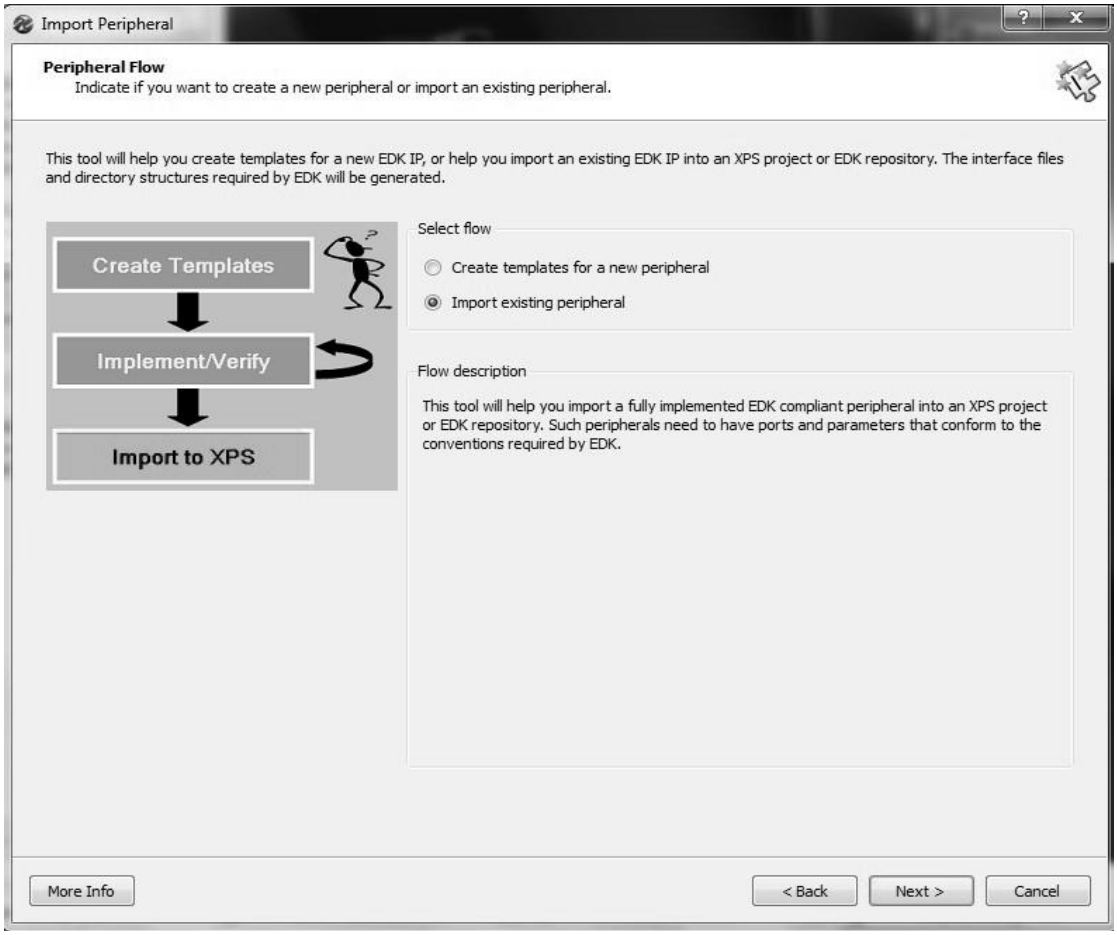


Σχήμα 9.10

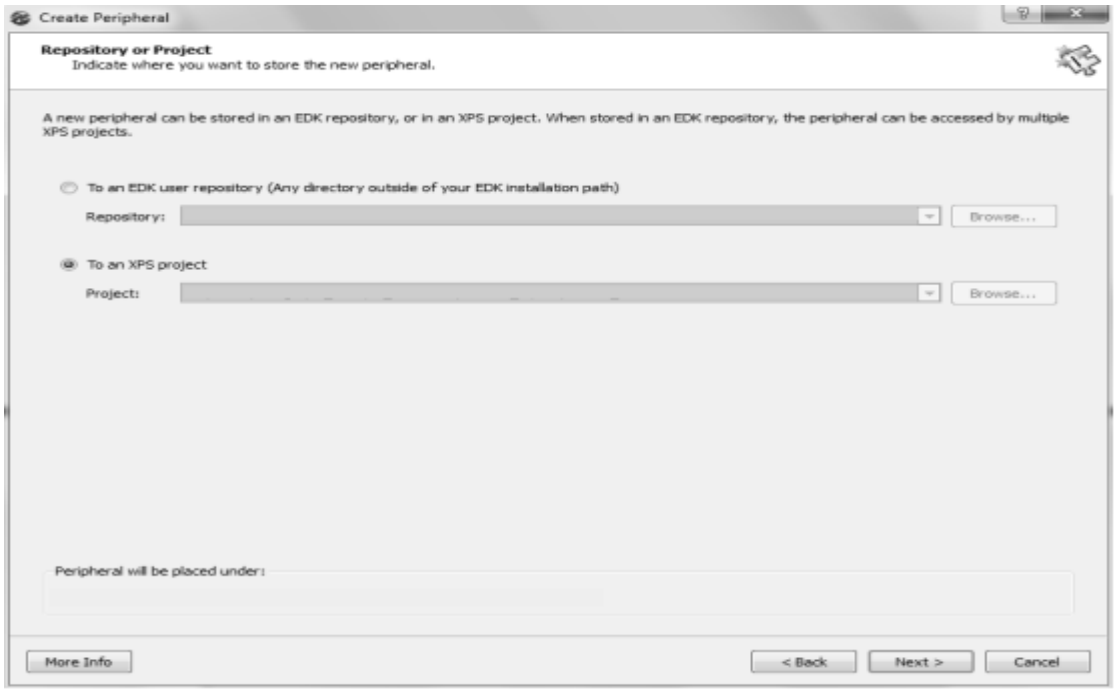
9.4.1 Εφαρμογή του Βοηθού για προσθήκη του περιφερειακού pid στον Microblaze

Στην συγκεκριμένη ενότητα θα περιγράψουμε αναλυτικά την ένωση του Pcore που δημιουργήθηκε από το εργαλείο Vivado HLS με την πλατφόρμα στην οποία έχουμε συνθέσει τον Microblaze.

Ο βοηθός ξεκινά επιλέγοντας Hardware → Create or Import Peripheral Wizard. Επιλέγοντας το Import Existing Peripheral, ο χρήστης καλείται να αποφασίσει εάν θα αποθηκεύσει τα αρχεία, και την ειδική δομή φακέλου που ακολουθούν, μέσα στους φακέλους του τρέχοντος project ή σε κάποια τοποθεσία ειδική για όλα τα περιφερειακά του χρήστη.

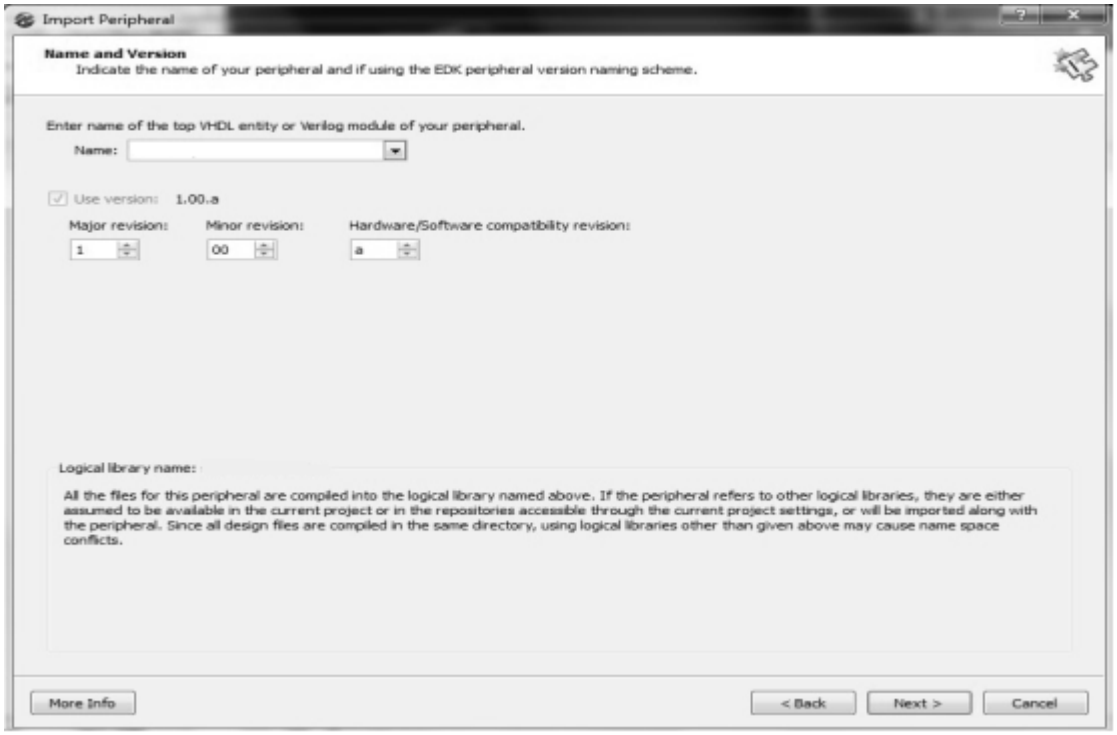


Σχήμα 9.11



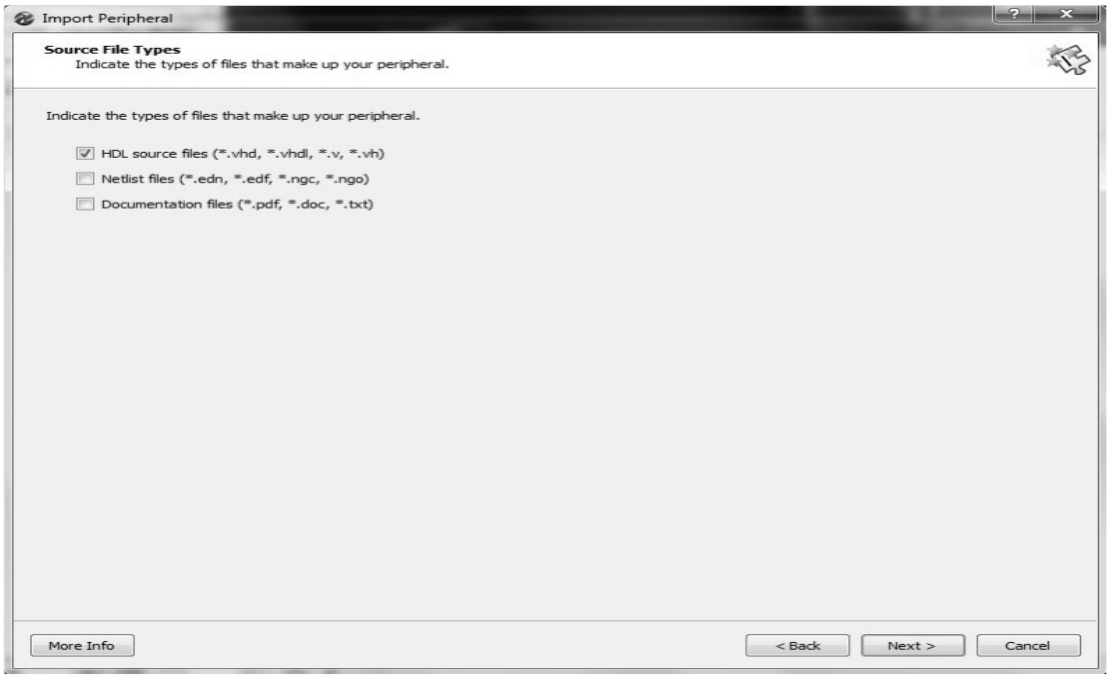
Σχήμα 9.12

Μετά την απόφαση για τον τόπο στον οποίο θα αποθηκευτούν τα αρχεία του νέου περιφερειακού, ο χρήστης πρέπει να δώσει την ονομασία του περιφερειακού που είναι ταυτόχρονα και αυτή του top-level HDL αρχείου στην ιεραρχία. Επίσης μπορούν να δοθούν αναγνωριστικά, όπως αριθμός κύριας και δευτερεύουσας αναθεώρησης και αναγνωριστικό συμβατότητας υλικού / λογισμικού.



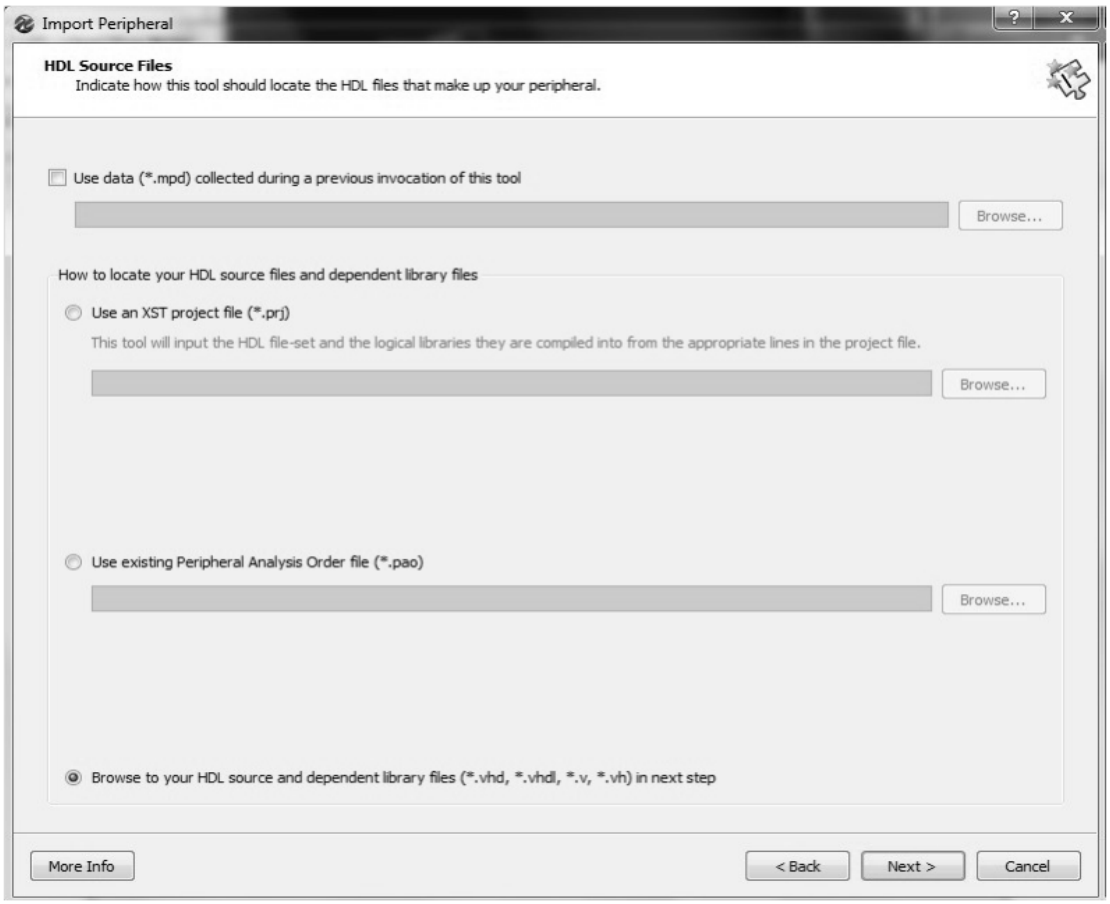
Σχήμα 9.13

Επόμενο βήμα του χρήστη είναι η επιλογή του κώδικα σε Vhdl ή Verilog του Pcore που έχει δημιουργήσει και περιγράφουν τον τρόπο δημιουργίας του περιφερειακού.



Σχήμα 9.14

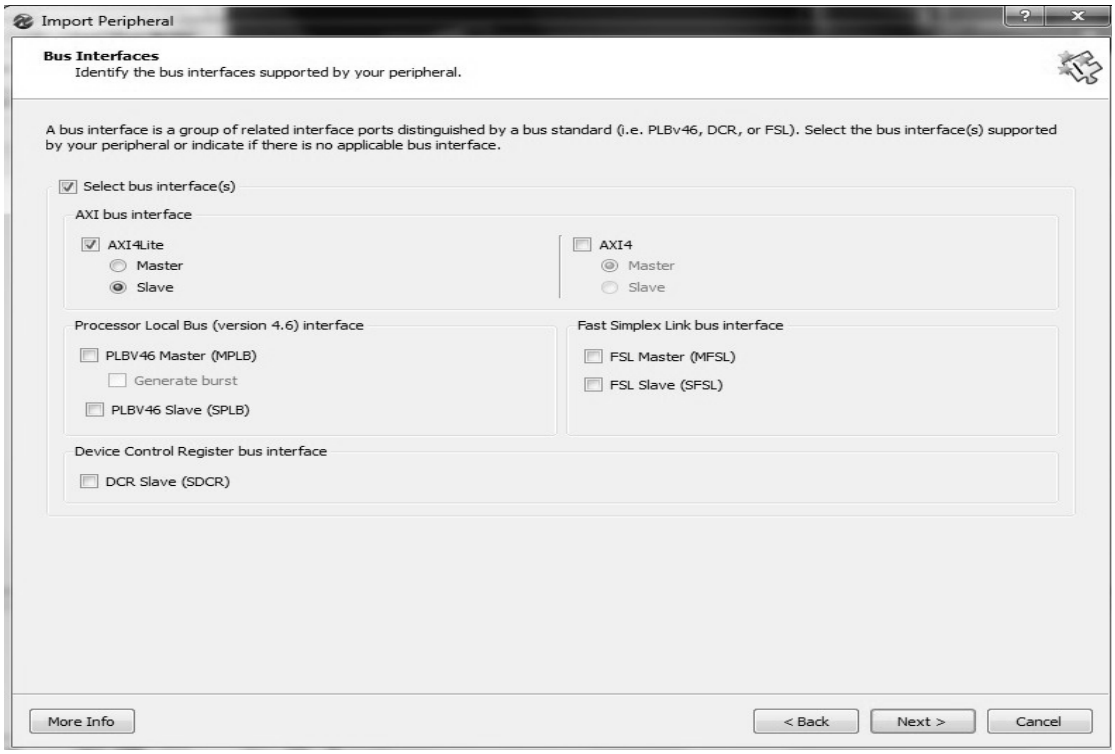
Μετά επιλέγουμε να ανεβάσουμε τα συγκεκριμένα αρχεία HDL στον βοηθό.



Σχήμα 9.15

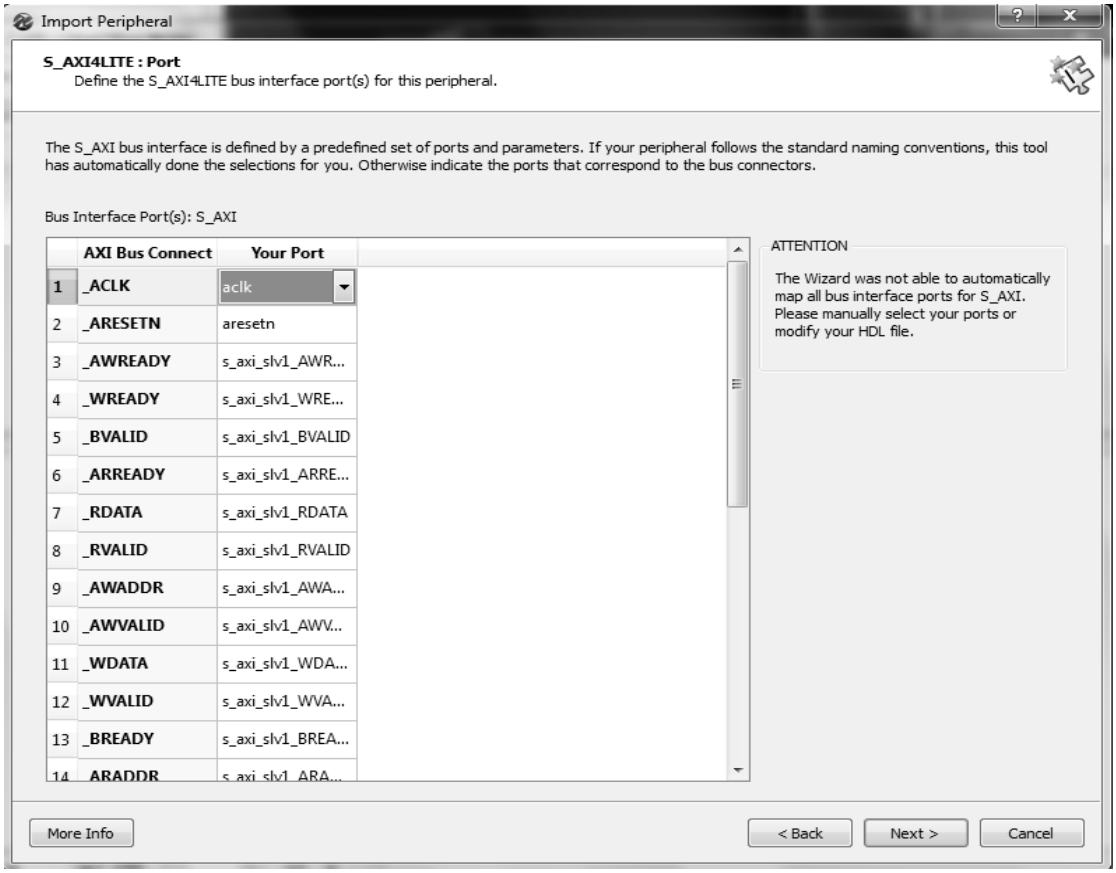
Και μολις τα βρουμε τα προσθέτουμε πατώντας Add Files.

Επόμενο βήμα του βοηθού είναι η επιλογή του διαδρόμου δεδομένων στον οποίο θα συνδεθεί το περιφερειακό, προκειμένου να υλοποιηθεί η κατάλληλη διεπαφή. Εμείς στην σχεδίαση που υλοποιήσαμε ως Bus Interface χρησιμοποιήσαμε το AXI4Lite και μάλιστα με συμπεριφορά slave.



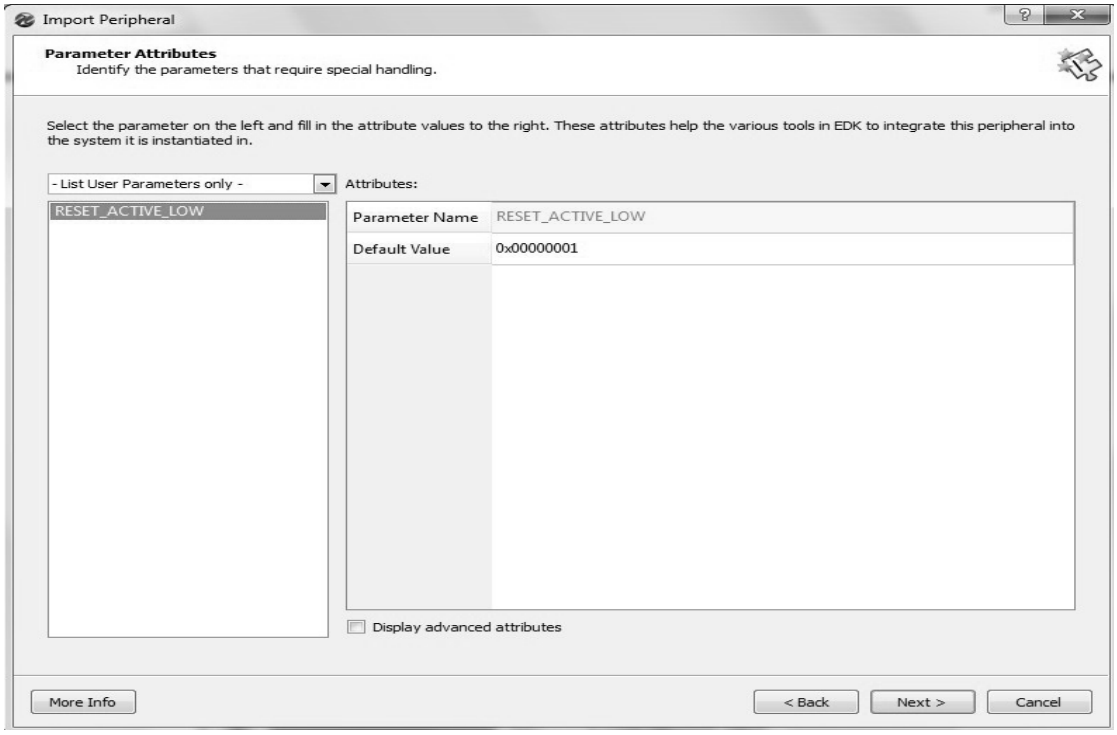
Σχήμα 9.16

Ο βοηθός χρησιμοποιεί τον κώδικα HDL για να ορίσει τα bus interface ports για το AXI4Lite, ωστόσο κάποιοι ορισμοί πρέπει να γίνουν manually ή προσθέτοντας κάποιες παραπάνω γραμμές στον κώδικα.



Σχήμα 9.17

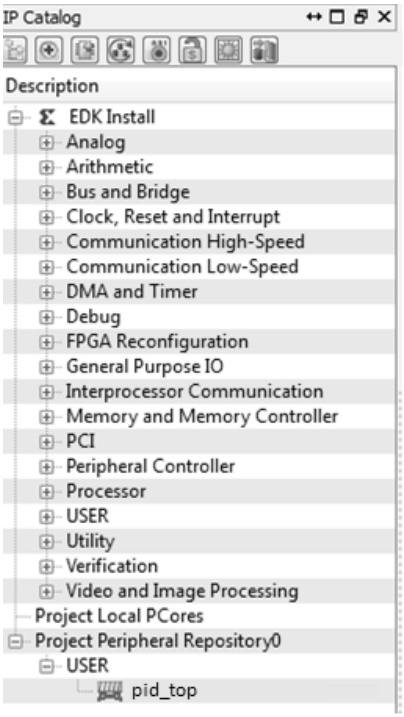
Στην συνέχεια αρχικοποιούμε τις απαραίτητες παραμέτρους



Σχήμα 9.18

Ο βοηθός δεχόμενος σαν είσοδο όλες τις επιλογές που κάνει ο χρήστης, εισάγει την αντίστοιχη πληροφορία στα αρχεία που δημιουργεί. Μια περίληψη του περιφερειακού και των λειτουργιών του γίνεται στο τελευταίο παράθυρο διαλόγου του βοηθού, καθώς και μία αναφορά στα αρχεία που δημιουργούνται.

Μόλις ολοκληρωθεί η δημιουργία του περιφερειακού τότε αυτό προστίθεται στον IP κατάλογο όπως φαίνεται και στο σχήμα. Όπως αναφέρθηκε και πιο πάνω το περιφερειακό θα πρέπει να ενσωματωθεί στο υπόλοιπο σύστημα. Για να γίνει η προσθήκη από τον χρήστη αρκεί αυτός να επιλέξει με αριστερό click το περιφερειακό fib_top και να πατήσει add στο αναδυόμενο παράθυρο.



Σχήμα 9.19

Τέλος, επιλέγουμε Generate BitStream και Export Design->Export & Launch SDK έτσι ώστε να δημιουργήσουμε και το software application που θα τρέξει στο ενσωματωμένο σύστημα που δημιουργήσαμε.

Βιβλιογραφία

[1] http://en.wikipedia.org/wiki/Embedded_system

[2] J. Catsoulis "Designing Embedded Hardware",O'Reilly Media, Second Edition edition, May 2005

[3] D. Soudris "Embedded Systems Design, lecture 1", ΕΜΠ

[4] C. Evans "Notes on the OpenSURF Library", January 2009

[5] G.Bradski και A.Kaehler "Learning OpenCV: Computer Vision with the OpenCV Library", O'Reilly Media, 1st edition, October 2008

[6] Xilinx Inc. "Microblaze processor reference guide",2008

[7] Xilinx Inc. "Xcell Journal" Issue 81, fourth quarter 2012

[8] Xilinx Inc. "Vivado Design Suite User Guide" UG902 (v2012.2) July 25, 2012

[9] Xilinx Inc. "Vivado Design Suite Tutorial" UG871 (v2012.2) August 20, 2012

[10] M.McFarland, A.Parker and R.Camposano " Tutorial on high-level synthesis", IEEE Computer Society Press Los Alamitos, CA, USA ©1988

[11] Petru Eles and Zebo Peng " High Level Synthesis", lecture 3, Linkoping University http://www.ida.liu.se/~petel/SysSyn/lect3.frm.pdf

[12] Ζ. Πούλος "Υλοποίηση με γλώσσα περιγραφής υλικού VHDL του πρωτοκόλλου συμπίεσης εικόνας JPEG-2000 σε πλατφόρμα Xilinx Virtex-5" , ΕΜΠ, Απρίλιος 2011

[13] http://en.wikipedia.org/wiki/Computer_vision

[14] http://en.wikipedia.org/wiki/High-level_synthesis

[15] https://el.wikipedia.org/wiki/FPGA

[16] http://en.wikipedia.org/wiki/OpenCV

[17] http://en.wikipedia.org/wiki/Lookup_table

[18] http://zone.ni.com/reference/en-XX/help/371599G-01/lvfpgaconcepts/fpga_basic_chip_terms/

[19] http://www.ustudy.in/node/7616

[20] "All about FPGAs" http://www.eetimes.com/document.asp?doc_id=1274496

[21] http://en.wikibooks.org/wiki/Programmable_Logic/FPGAs

[22]http://www.geniuswiki.com/page#%2FEmbeddedSystem%2BTechnology%2FEmbedded%2BSystems%2BTesting%2B%7C%2BA%2Bshort%2Bdefinition

[23] "Advantages of the Xilinx Virtex-5 FPGA"
http://www.ni.com/white-paper/7440/en/

[24] "Virtex 6 Logic Slice Description"
http://www.eecg.utoronto.ca/vpr/utfal_ex4.html

[25] "Lab 7 lecture- Introduction to Accumulators and FPGAs"
http://www.ece.unm.edu/vhdl/Labs2004/fall2004/lab07/lab07_lecture.htm