ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

Hardware Acceleration of Image Registration Algorithm on FPGA-based Systems on Chip

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΔΗΜΗΤΡΙΟΥ ΓΟΥΡΟΥΝΑ

Επιβλέπων : Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

Hardware Acceleration of Image Registration Algorithm on FPGA-based Systems on Chip

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΔΗΜΗΤΡΙΟΥ ΓΟΥΡΟΥΝΑ

Επιβλέπων : Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10η Οκτωβρίου 2018.

.....	.....	.....
Δημήτριος Ι. Σούντρης Αναπληρωτής Καθηγητής Ε.Μ.Π.	Κιαμάλ Πεκμεστζή Καθηγητής Ε.Μ.Π.	Γιώργος Ματσόπουλος Αναπληρωτής Καθηγητής Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2018

.....

ΔΗΜΗΤΡΙΟΣ ΓΟΥΡΟΥΝΑΣ

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

ΠΕΡΙΛΗΨΗ

Η επεξεργασία εικόνας είναι ένας κλάδος της επιστήμης των υπολογιστών και της ψηφιακής επεξεργασίας σήματος. Σήμερα, λόγω της πληθώρας εφαρμογών και των συνεχώς αυξανόμενων απαιτήσεων στη βιομηχανία και όχι μόνο, το ερευνητικό έργο που σχετίζεται με τον κλάδο αυτό αυξάνει συνεχώς σε διαστάσεις. Με εφαρμογές στην ιατρική, στην αναγνώριση προτύπων και αντικειμένων, στην αυτοκινητοβιομηχανία, σε δορυφορικά συστήματα κλπ. ποικίλες και διαφορετικές στρατηγικές επεξεργασίας με ευρέως ανεπτυγμένο επιστημονικό υπόβαθρο έχουν προταθεί ανά τους καιρούς. Ένα από τα κύρια ζητήματα που αφορούν την επεξεργασία εικόνας είναι η απαίτηση για εκτέλεση σε πραγματικό χρόνο, η οποία τείνει στις περισσότερες περιπτώσεις να είναι ανέφικτη με τη συνηθισμένη υλοποίηση και εκτέλεση σε ένα υπολογιστικό σύστημα που αποτελείται από έναν επεξεργαστή. Ειδικά αν ο λόγος γίνεται για ενσωματωμένα συστήματα, υπάρχει επίσης η ανάγκη για όσο το δυνατόν λιγότερη κατανάλωση ενέργειας. Ως αποτέλεσμα, διαφορετικές τεχνολογίες και συνεργασία μεταξύ αυτών είναι ικανές να παρέχουν πιο ικανοποιητικά αποτελέσματα.

Σκοπός αυτής της διπλωματικής είναι η σχεδίαση και υλοποίηση μιας εφαρμογής επεξεργασίας εικόνας σε ένα ενσωματωμένο σύστημα που αποτελείται από επεξεργαστή και FPGA, με σκοπό να καλύπτονται οι προαναφερθείσες απαιτήσεις. Ως εφαρμογή επιλέχθηκε η καταχώριση (αποτύπωση) εικόνας για ένα σύνολο δεδομένων που αποτελείται από φωτογραφίες ματιών γύρω από την περιοχή της ίριδας. Σκοπός της εφαρμογής είναι η ταύτιση διαφορετικών εικόνων και απόφαση για το αν πρόκειται για το ίδιο ή διαφορετικό μάτι. Για την υλοποίηση του συστήματος χρησιμοποιήθηκε η πλατφόρμα Zybo της Digilent, που είναι βασισμένη στην οικογένεια συσκευών Zynq-7000 All Programmable SoC. Κατόπιν μελέτης των χρονοβόρων κομματιών της εφαρμογής, επιλέχθηκε ο κατάλληλος διαμερισμός υλικού και λογισμικού για τη βέλτιστη δυνατή λύση. Αυτή η συ-σχεδίαση πραγματοποιήθηκε από την πλευρά του Υλικού προγραμματίζοντας με τη γλώσσα VHDL και από την πλευρά του Λογισμικού με χρήση της γλώσσας C. Μελετήθηκαν επίσης αποτελεσματικές και εύχρηστες μέθοδοι επικοινωνίας μεταξύ Υλικού και Λογισμικού, οι οποίες μπορούν να εφαρμοστούν και σε πολλές άλλες εφαρμογές της επεξεργασίας εικόνας. Τέλος έγινε αξιολόγηση του τελικού συστήματος, στην οποία παρουσιάζονται η απόκλιση από τα αρχικά αποτελέσματα που παράγει το υλικό, η επιτάχυνση χρόνου που επιτεύχθηκε, η τελική κατανάλωση ισχύος και το ποσοστό επιτυχίας της αναγνώρισης των ματιών.

Λέξεις Κλειδιά

Επεξεργασία Εικόνας, Image Registration, FPGA, Ενσωματωμένα Συστήματα, CPU, Profiling, Επιτάχυνση, Υλοποίηση Hardware, Συ-σχεδίαση, Bandwidth Επικοινωνίας, Περιορισμοί Μνήμης, Αξιολόγηση Συστήματος

ABSTRACT

Image processing is a scientific field of computer science and digital signal processing. Today, the multitude of applications and the constantly increasing demands in the industry have imposed a great increase in the dimensions of the research associated with this sector. With applications in medicine, pattern recognition and recognition of objects, automotive industry, satellite systems, etc. varied and different processing strategies with a widely developed scientific background have been proposed over the times. One of the main issues regarding the image processing is a requirement for real-time execution, which in most cases tends to be unfeasible with the usual implementation on a computational system consisting of a standalone processor. Especially if the issue regards the embedded systems, there is also the need for the least possible energy consumption. As a result, different technologies and cooperation between them are likely to provide more satisfying results.

The purpose of this thesis is the design and implementation of an image processing application in an embedded system consisting of both a processor and an FPGA, in order to meet the abovementioned requirements. As an application the image registration problem was chosen, which focuses on a dataset that consists of eye photographs around the area of the iris. The goal of this application is the identification of different images and deciding whether or not it is the same or a different eye. For the implementation of the system the Zybo platform made by Digilent was used, which is based on the Zynq-7000 All Programmable SoC (System on Chip) device family, manufactured by Xilinx. After carefully studying the time-consuming sections of the application, an appropriate hardware and software partitioning for an optimum solution was selected. This co-design involved using the VHDL language for programming from the Hardware side and the C language from the software side. Efficient and convenient communication methods between hardware and software were examined, which can also be applied to many other applications of image processing. Lastly, the final system was assessed, with a presentation of the deviation from the initial results, the succeeded acceleration, the final power consumption and the success rate of the eye identification.

KeyWords

Image Processing, Image Registration, FPGA, Embedded Systems, CPU, Profiling, Acceleration, Hardware Implementation, Co-design, Communication Bandwidth Memory Constraints, System Evaluation

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα αρχικά να ευχαριστήσω τους επιβλέποντες καθηγητές μου, Δημήτριο Σούντρη ΕΜΠ και Γιώργο Ματσόπουλο ΕΜΠ, για την ευκαιρία που μου έδωσαν να ασχοληθώ με μια ενδιαφέρουσα εφαρμογή του απαιτητικού κλάδου της επεξεργασίας εικόνων. Μέσω της συνεργασίας των δύο εργαστηρίων Mlab και BIOMIG μου δόθηκε η δυνατότητα να εφαρμόσω τις γνώσεις μου πάνω στο software και το hardware πάνω σε έναν καινούριο για μένα κλάδο.

Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Καθηγητή Δημήτριο Σούντρη που μου επέτρεψε να εκπονήσω τη διπλωματική μου στο Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων (Microlab), καθώς και τους δύο διδακτορικούς Ιωάννη Στρατάκο και Βασίλειο Τσούτσουρα. Η υπομονή τους και η καθοδήγηση τους καθ' όλη τη διάρκεια της διπλωματικής ήταν άκρα βοηθητικές για μένα και με βοήθησαν να προχωρήσω και να ξεπεράσω κάθε δυσκολία. Επίσης θα ήθελα να ευχαριστήσω τον Διδάκτορα Θεόδωρο Οικονομόπουλο που μου παρείχε το απαραίτητο υλικό και χρήσιμες διευκρινίσεις ώστε να εμπεδώσω την εφαρμογή και να κατατοπιστώ πλήρως σχετικά με το αντικείμενο μελέτης μου. Θα ήθελα να ευχαριστήσω και όλα τα μέλη του Mlab για το φιλικό και ευχάριστο περιβάλλον εργασίας που έχουν καλλιεργήσει. Νιώθω ευγνώμων που είχα την ευκαιρία να συνεργαστώ με τους προαναφερθέντες επιστήμονες.

Τέλος θα ήθελα να ευχαριστήσω τους φίλους μου και την οικογένεια μου που με στήριξαν πνευματικά και υλικά στην μέχρι τώρα σταδιοδρομία μου.

ACKNOWLEDGEMENTS

First I would like to thank my supervisors, Professor Dimitrios Soudris, NTUA and George Matsopoulos NTUA, for the opportunity they gave me to deal with an interesting application of the demanding field of image processing. Through the cooperation of the two labs Mlab and BIOMIG I was allowed to apply my software and hardware knowledge and experience on a new scientific field.

I would especially like to thank Doctorates John Stratakos and Vasileios Tsoutsouras. Their patience and guidance throughout the course of my thesis was highly important for me and helped me to move on and get over any difficulties I met. I would also like to thank Doctor Theodore Economopoulos, who provided me with the necessary material and useful clarifications regarding the application, helping me fully apprehend the subject of my study. I express my gratitude to all members of the Mlab for the friendly and pleasant working environment they have fostered. I feel truly grateful for having the opportunity to cooperate with the aforementioned scientists.

Finally I would like to thank my friends and my family who have supported me spiritually and materially in my career so far.

CONTENTS

List of Figures and Tables

ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ

Εισαγωγή

Η επεξεργασία εικόνας είναι ένας κλάδος της επιστήμης των υπολογιστών και της ψηφιακής επεξεργασίας σήματος. Αναφέρεται στη διαδικασία εξαγωγής πληροφοριών (σχετικά με γεωμετρικά χαρακτηριστικά, μοτίβα, χρώματα κ.λπ.) από μια εικόνα ή ένα σύνολο εικόνων και χρήση των πληροφοριών αυτών για την αναγνώριση ή εφαρμογή ενός μετασχηματισμού σε αυτή την εικόνα. Σήμερα, λόγω της πληθώρας εφαρμογών και των συνεχώς αυξανόμενων απαιτήσεων στη βιομηχανία και όχι μόνο, το ερευνητικό έργο που σχετίζεται με τον κλάδο αυτό αυξάνει συνεχώς σε διαστάσεις. Με εφαρμογές στην ιατρική, στην αναγνώριση προτύπων και αντικειμένων, στην αυτοκινητοβιομηχανία, σε δορυφορικά συστήματα κλπ. ποικίλες και διαφορετικές στρατηγικές επεξεργασίας με ευρέως ανεπτυγμένο επιστημονικό υπόβαθρο έχουν προταθεί ανά τους καιρούς. Ένα από τα κύρια ζητήματα που αφορούν την επεξεργασία εικόνας είναι η απαίτηση για εκτέλεση σε πραγματικό χρόνο, η οποία τείνει στις περισσότερες περιπτώσεις να είναι ανέφικτη με τη συνηθισμένη υλοποίηση και εκτέλεση σε ένα υπολογιστικό σύστημα που αποτελείται από έναν επεξεργαστή. Ειδικά αν ο λόγος γίνεται για ενσωματωμένα συστήματα, υπάρχει επίσης η ανάγκη για όσο το δυνατόν λιγότερη κατανάλωση ενέργειας.

Ένα ενσωματωμένο σύστημα ορίζεται ως οποιαδήποτε συσκευή ενσωματώνει ένα προγραμματιζόμενο επεξεργαστή, αλλά δεν είναι από μόνη της ένας υπολογιστής γενικού σκοπού. Μερικές φορές το ενσωματωμένο σύστημα αποτελείται από πολλαπλούς επεξεργαστές, περιφερειακά, κρυφές μνήμες, διασυνδέσεις και μια συστοιχία επιτόπια προγραμματιζόμενων πυλών (FPGA) με σκοπό να υλοποιήσει μια καθορισμένη από το χρήστη και συγκεκριμένου υλικού σχεδίαση για την εφαρμογή. Τέτοια συστήματα αναφέρονται ως συστήματα σε ψηφίδα (SoC). Τα ενσωματωμένα συστήματα στοχεύουν ειδικές εφαρμογές και έχουν σχεδιαστεί έτσι ώστε να παρέχουν μια βέλτιστη υλοποίηση όσον αφορά την ταχύτητα επεξεργασίας, την κατανάλωση, το κόστος, την αξιοπιστία κ.λπ. Το φάσμα εφαρμογών τους είναι τόσο μεγάλο που σχεδόν το 98% των μικροεπεξεργαστών κατασκευάζονται προοριζόμενα για ενσωματωμένα συστήματα.

Όπως αναφέρθηκε, η υλοποίηση με αποκλειστική χρήση της CPU δεν είναι πάντα αποδοτική, καθώς η φύση των εφαρμογών επεξεργασίας εικόνας χαρακτηρίζεται από μεγάλη παραλληλία, την οποία δεν μπορεί να εκμεταλλευτεί επαρκώς ένας επεξεργαστής. Τα συστήματα σε ψηφίδα που ενσωματώνουν FPGA και CPU παρέχουν αποτελεσματική επικοινωνία μεταξύ των δύο, με αποτέλεσμα να επιτρέπεται η μείωση κατανάλωσης και η αύξηση ταχύτητας που προσφέρει το FPGA σε μια εφαρ-

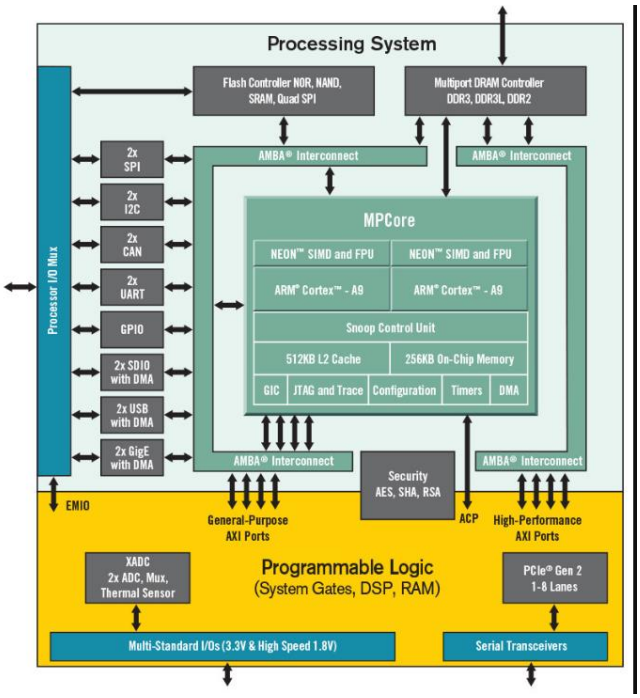
μογή πραγματικού χρόνου. Ωστόσο η ανάπτυξη της εφαρμογής με τη συ-σχεδίαση υλικού και λογισμικού εισάγει αυξημένη πολυπλοκότητα και μεγαλύτερο χρόνο ανάπτυξης. Τα τελευταία χρόνια όμως έχουν αναπτυχθεί ποικίλα εργαλεία που βοηθούν στην ανάπτυξη, παρέχοντας έτοιμες λειτουργίες και αποκρύπτοντας από το σχεδιαστή λεπτομέρειες του υλικού και της διεπαφής με το λογισμικό.

Η παρούσα διπλωματική έχει ως σκοπό τη σχεδίαση και υλοποίηση μιας εφαρμογής επεξεργασίας εικόνας που ονομάζεται καταχώριση (αποτύπωση) εικόνας (image registration), για ένα σύνολο δεδομένων που αποτελείται από φωτογραφίες ματιών γύρω από την περιοχή της ίριδας.

Περιγραφή Αναπτυξιακής Πλακέτας

Τα FPGAs είναι συσκευές ημιαγωγών που αποτελούνται από επαναπρογραμματιζόμενο υλικό. Βασίζονται γύρω από μια μήτρα διαμορφώσιμων μπλοκ λογικής (CLBs), τα οποία συνδέονται μέσω προγραμματιζόμενων διασυνδέσεων. Αυτές οι διασυνδέσεις προγραμματίζονται χρησιμοποιώντας μια γλώσσα περιγραφής υλικού (HDL), όπως η Verilog και η VHDL, με σκοπό την υλοποίηση μιας εφαρμογής με συγκεκριμένη επιθυμητή λειτουργικότητα. Δεδομένου ότι κάθε μπλοκ λογικής μπορεί να είναι υπολογιστικά ανεξάρτητο από τα υπόλοιπα, υψηλά επίπεδα παραλληλίας καθίστανται ικανά, με αποτέλεσμα τη σημαντική επιτάχυνση του χρόνου εκτέλεσης ενός προγράμματος.

Στο παρελθόν, τα FPGAs δεν ενσωματώνονταν στο ίδιο τσιπ με τον επεξεργαστή. Ως αποτέλεσμα, η off-chip επικοινωνία μεταξύ τους ήταν δύσκολη και αναποτελεσματική. Το 2010 η εταιρεία Xilinx εισήγαγε την οικογένεια SoC συσκευών Zynq-7000-All Programmable στην αγορά, η οποία ενσωματώνει προγραμματισμό λογισμικού ενός διπύρηνου ARM Cortex-A9 με τον προγραμματισμό υλικού των 28nm Artix-7 FPGA, προσφέροντας επιτάχυνση με χρήση υλικού και ενσωμάτωση CPU, DSP, μνήμης και πολλών περιφερειακών σε μία μόνο συσκευή (Εικόνα 1). Δεδομένου ότι ο ARM επεξεργαστής είναι ικανός να υποστηρίξει πλήρη λειτουργικά συστήματα, (το πιο συχνά χρησιμοποιούμενο από τα οποία είναι το Linux, δεδομένου ότι είναι ανοιχτού κώδικα και παρέχει μια εδραιωμένη υποστηρικτική κοινότητα) ο προγραμματιστής έχει τη δυνατότητα να αναπτύσσει την εφαρμογή χρησιμοποιώντας συνεργασία υλικού/λογισμικού, η οποία είναι ιδανική για σχεδίαση σε ενσωματωμένα συστήματα. Η Xilinx χρησιμοποιεί το πρωτόκολλο AMBA AXI για την επικοινωνία μεταξύ της προγραμματιζόμενης λογικής (FPGA) και του επεξεργαστή. Το AXI (Προηγμένη Επεκτάσιμη Διασύνδεση) είναι μέρος του ARM AMBA, μια οικογένεια διαύλων επικοινωνίας για μικροελεγκτές. Το AMBA είναι ένα συνηθισμένο πρότυπο για τη σύνδεση και τη διαχείριση των λειτουργικών μπλοκ σε ένα SoC. Διευκολύνει τη σωστή ανάπτυξη σχεδιάσεων που εμπεριέχουν πολλαπλούς επεξεργαστές με μεγάλο αριθμό ελεγκτών και περιφερειακά. Το AXI4 παρέχει βελτιώσεις, προς όφελος της παραγωγικότητας, της ευελιξίας και της διαθεσιμότητας.



Εικόνα 1: Η αρχιτεκτονική των συσκευών Zynq-7000 [4].

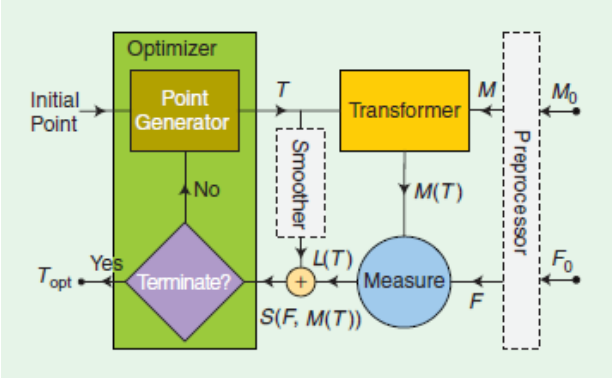
Υπάρχουν τρεις τύποι διεπαφών AXI4:

- AXI4 (ή AXI4-full) — για απαιτήσεις υψηλής απόδοσης και αμφίδρομης memory mapped επικοινωνίας. Επιτρέπει ριπές 256 δεδομένων / διεύθυνση.
- AXI4-Lite — για απλή, χαμηλής ταχύτητας memory mapped επικοινωνία μόνο. Σχεδόν μια ελαφριά παραλλαγή του AXI4-full αλλά με ριπές 1 δεδομένου/διεύθυνση.
- AXI4-Stream — για υψηλής ταχύτητας ροή δεδομένων. Παρέχει απεριόριστες ριπές δεδομένων με ένα μοναδικό κανάλι για μεταφορά συνεχούς ροής. Η επιτρεπτή φορά της ροής είναι μόνο από τον Master στον Slave.

Το SoC FPGA που χρησιμοποιήθηκε στην παρούσα διατριβή για το σχεδιασμό του registration σε ένα ενσωματωμένο σύστημα ήταν το Zybo Development Board, που κατασκευάζεται από την Digilent χρησιμοποιώντας το μικρότερο μέλος της οικογένειας Xilinx Zynq-7000, το Z-7010. Το Z-7010 βασίζεται στην αρχιτεκτονική Xilinx System-on-Chip (SoC), που ενσωματώνει έναν διπύρηνο επεξεργαστή ARM Cortex-A9 με ένα Xilinx 7-series FPGA στο ίδιο ολοκληρωμένο. Ακριβώς όπως και οι περισσότερες από τις άλλες συσκευές της οικογένειας Zynq-7000, το FPGA αυτό είναι κατασκευασμένο με το πρότυπο της Artix-7 προγραμματιζόμενης λογικής.

Σύντομο Θεωρητικό Υπόβαθρο

Το image registration είναι μια εφαρμογή επεξεργασίας εικόνας που συναντάται συχνά σε εφαρμογές όρασης υπολογιστών, ιατρικής απεικόνισης, συστημάτων ασφαλείας και αυτόματης αναγνώρισης στόχου, καθώς και για την ανάλυση εικόνων και δεδομένων που συλλέγονται από τους δορυφόρους. Για παράδειγμα, η ευθυγράμμιση εικονικών δεδομένων επιτρέπει σε ιατρικούς εμπειρογνώμονες να συγκρίνουν διαφορετικά στιγμιότυπα στο χρόνο μιας συγκεκριμένης ανατομικής περιοχής, με σκοπό την αξιολόγηση της προόδου ή της καταστολής ορισμένων παθήσεων ή του ποσοστού επιτυχίας μιας προτεινόμενης θεραπείας.



Εικόνα 2: Αρχιτεκτονική του Registration Solver [6].

Η Εικόνα 2 παρουσιάζει τα διάφορα συστατικά ενός γενικού registration solver, με τα πιο κύρια να είναι ένας μετασχηματιστής, ένα μέτρο σύγκρισης και μία μέθοδος βελτιστοποίησης. Ένα μέτρο της ομοιότητας ή της απόστασης υπολογίζεται μεταξύ των εικόνων σε κάθε επανάληψη και χρησιμοποιείται για να αποφανθεί για το αν είναι «επαρκώς» ευθυγραμμισμένες. Αυτή η διαδικασία ελέγχεται από τον optimizer, που ξεκινάει από μια αρχική εκτίμηση και καθορίζει μια σειρά από επόμενα βήματα για να επιτευχθεί μια βέλτιστη ευθυγράμμιση. Παρακάτω γίνεται μια σύντομη παρουσίαση του καθενός από τα τρία αυτά κύρια συστατικά.

Μέτρηση ομοιότητας: Για την μέτρηση ομοιότητας έχουν προταθεί διάφορα μοντέλα. Κάποια από αυτά είναι το άθροισμα τετραγώνων διαφοράς και το άθροισμα απόλυτων διαφορών. Ωστόσο αυτές οι δύο μέθοδοι είναι εξαιρετικά αδύναμες και δεν παράγουν καθόλου πρακτικά ή χρήσιμα αποτελέσματα. Δύο ικανοποιητικές και πιο συνηθισμένες μέθοδοι είναι η μέθοδος του συντελεστή συσχέτισης και η μέθοδος της από κοινού πληροφορίας του Matte. Και οι δύο αυτές μέθοδοι μετρούν ομοιότητα, πράγμα που σημαίνει ότι δύο πανομοιότυπες εικόνες θα έχουν ένα μέτρο ομοιότητας ίσο με ένα, που θα μειώνεται καθώς οι εικόνες θα απομακρύνονται. Η συσχέτιση, όπως αναφέρει και ο ίδιος ο όρος, εκφράζει το πόσο κοντά είναι δύο μεταβλητές στο να έχουν μια γραμμική σχέση μεταξύ τους. Ο συντελεστής συσχέτισης υπολογίζεται

διαιρώντας τη συνδιακύμανση των δύο μεταβλητών με το γινόμενο των τυπικών τους αποκλίσεων. Έστω $\rho_{x,y}$ η συσχέτιση μεταξύ των δύο μεταβλητών X και Y, σ_x, σ_y οι τυπικές τους αποκλίσεις και μ_x, μ_y οι αναμενόμενες τιμές τους. Με την παρακάτω εξίσωση υπολογίζεται ο συντελεστής συσχέτισης $\rho_{x,y}$ μεταξύ των δύο εικόνων:

$$r_{X,Y} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{n s'_x s'_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \quad (E\xi.1)$$

Matte's Mutual Information: Αυτή η μέθοδος μετρά την αμοιβαία εξάρτηση μεταξύ των δύο μεταβλητών, σύμφωνα με την θεωρία πιθανοτήτων. Η εξίσωση από την οποία εξάγεται η από κοινού πληροφορία είναι η ακόλουθη, όπου p(x,y) είναι η από κοινού πιθανότητα των μεταβλητών X και Y και p(χ), p(y) οι συναρτήσεις πυκνότητας πιθανότητας των X και Y αντιστοίχως.

$$I(X,Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right) \quad (E\xi.2)$$

Οι πυκνότητες πιθανότητας υπολογίζονται χρησιμοποιώντας το Kernel Density Estimation (KDE), επίσης γνωστή ως μέθοδος Parzen ιστογράμματος στην επεξεργασία σήματος.

Μετασχηματιστής: Ο μετασχηματιστής απεικονίζει σημεία της κινούμενης εικόνας σε νέες τοποθεσίες στην εικόνα που μετασχηματίζεται. Βάσει των απαιτήσεων του προβλήματος registration, ο μετασχηματιστής μπορεί είτε να είναι συγγραμμικός ή παραμορφώσιμος. Ο συγγραμμικός μετασχηματισμός ορίζεται από μια μήτρα 2 x 2. Παράδειγμα συγγραμμικού μετασχηματισμού, ο οποίος και χρησιμοποιήθηκε σε αυτή τη διπλωματική είναι ο affine μετασχηματισμός, του οποίου η εξίσωση φαίνεται παρακάτω, ο οποίος πραγματοποιείται γύρω από το κέντρο της εικόνας:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x - mw \\ y - mh \end{bmatrix} + \begin{bmatrix} b_1 + mw \\ b_2 + mh \end{bmatrix} \quad (E\xi.3)$$

, όπου α_{11},α_{22} οι συντελεστές κλιμάκωσης του μετασχηματισμού, α_{12},α_{21} οι συντελεστές περιστροφής, b_1,b_2 οι συντελεστές μετατόπισης, x',y' οι τελικές συντεταγμένες, x,y οι αρχικές συντεταγμένες και mw=mh=500 σταθερό για μέγεθος εικόνων 1000x1000 (middle Height, middle Width).

Optimizer: Ο optimizer είναι υπεύθυνος για την εφαρμογή μιας αποτελεσματικής και συχνά μη εξαντλητικής στρατηγικής για αναζήτηση της καλύτερης αντιστοιχίας μεταξύ των εικόνων στον επιτρεπτό χώρο των παραμέτρων μετασχηματισμού. Οι δύο μέθοδοι βελτιστοποίησης που μελετήθηκαν είναι οι Downhill Simplex και Powell Direction Method. Η μέθοδος Powell εφαρμόζει μια πιο εξαντλητική αναζήτηση και ως

αποτέλεσμα απαιτεί πολύ περισσότερο χρόνο επεξεργασίας. Παρατηρήθηκε όμως ότι για τις ανάγκες τις εφαρμογής ο Downhill Simplex ήταν επαρκής και παράγει ικανοποιητικά αποτελέσματα.

Ένα παράδειγμα που δίνει μια ιδέα για το χρόνο εκτέλεσης και τις βέλτιστες παραγόμενες παραμέτρους (T1-T6) για εκτέλεση σε έναν επεξεργαστή προσωπικού υπολογιστή των 2.4 GHz είναι το παρακάτω. Το παράδειγμα αναφέρεται σε επιτυχή registration μεταξύ δύο διαφορετικών εικόνων του ίδιου ματιού.

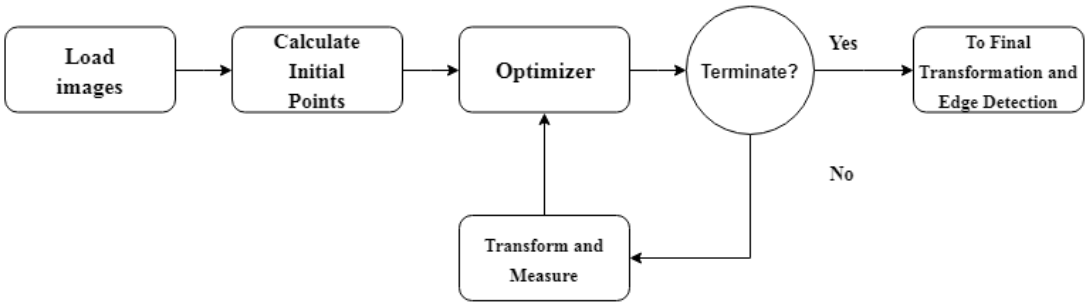
Πίνακας 1: Αποτελέσματα του Registration για αντιπροσωπευτικό δείγμα σε γρήγορο επεξεργαστή

T1	T2	T3	T4	T5	T6	Measure	Time(s)
1.0039	-0.0163	177.0986	0.0262	1.0002	-62.3076	0.78388	7.750371

Εν γένει παρατηρήθηκε ότι και οι δύο μέθοδοι σύγκρισης που αναφέρθηκαν παράγουν παρόμοια αποτελέσματα. Δεν παρατηρήθηκε περίπτωση στην οποία κάποιο registration πέτυχε με τη μια και απέτυχε με την άλλη. Για το λόγο αυτό επιλέχθηκε να υλοποιηθεί η μέθοδος συσχέτισης, καθώς η φύση των εξισώσεων της είναι πιο απλή και πιο εύκολο να υλοποιηθεί στο FPGA.

Ροή Επεξεργασίας

Μέχρι τώρα, έχει γίνει μια περιληπτική ανάλυση της θεωρίας πίσω από το image registration. Τώρα θα γίνει μια παρουσίαση της επεξεργαστικής ροής σε προγραμματιστικό επίπεδο για την υλοποίηση του συστήματος. Η Εικόνα 3 απεικονίζει αυτή τη ροή, που μοιάζει με μια απλοποιημένη εκδοχή της Εικόνας 2.



Εικόνα 3: Ροή επεξεργασίας του Registration Solver

Στο πρώτο στάδιο οι δύο εικόνες προς σύγκριση φορτώνονται από το dataset στο πρόγραμμα. Αυτό μπορεί να γίνει με διάφορους τρόπους, όπως π.χ. με χρήση της βιβλιοθήκης OpenCV ή με ανάγνωση από αρχείο κειμένου. Σε ένα πιο πρακτικό περιβάλλον όπου θα εφαρμοστεί το σύστημα για ιατρικές εφαρμογές, θα υπάρξει επίσης ένα στάδιο πριν για τη λήψη των εικόνων από κάποια κάμερα και φόρτωση τους στη μνήμη.

Στη συνέχεια, προκειμένου να εκκινήσει ο αλγόριθμος Downhill simplex για, N + 1 (δηλαδή 7 στην περίπτωση αυτή) αρχικά σημεία θα πρέπει να υπολογιστούν. Ένα σετ αρχικών ζευγών παραμέτρων μετασχηματισμού πού αντιπροσωπεύουν τα όρια του χώρου αναζήτησης αρχικοποιούνται και υπολογίζεται η ομοιότητα μεταξύ των εικόνων για κάθε ένα από αυτά τα ζεύγη.

Ο Optimizer για τη μέθοδο Downhill Simplex καλεί ορισμένες συναρτήσεις για να διεξάγει την εξερεύνηση του χώρου αναζήτησης. Μια από τις συναρτήσεις αυτές σχετίζεται με τη βελτιστοποίηση της τρέχουσας κατάστασης των παραμέτρων, ενώ μια διαφορετική χρησιμοποιείται για να μετρηθεί η ομοιότητα μεταξύ της σταθερής εικόνας και της κινούμενης εικόνας, αν στην κινούμενη εικόνα εφαρμοστεί ο τρέχω μετασχηματισμός. Το μέτρο ομοιότητας που υπολογίζεται στην εν λόγω συνάρτηση χρησιμοποιείται έπειτα για να αποφασίσει ο optimizer ποιο θα πρέπει να είναι το επόμενο βήμα βελτιστοποίησης.

Το 4ο στάδιο, που ονομάζεται “Transform and Measure” είναι αυτό που περιγράφηκε παραπάνω. Ο optimizer θα ελέγξει πρώτα αν ικανοποιείται κάποιο από τα δύο κριτήρια τερματισμού. Αν ναι τότε τερματίζει, αλλιώς αναζητά το επόμενο ζεύγος μετασχηματισμού που θα αυξήσει την ομοιότητα. Τα δύο αυτά κριτήρια είναι ο μέγιστος αριθμός επαναλήψεων και το κατώφλι ανοχής.

Το τελικό στάδιο είναι ένα off-line στάδιο της διαδικασίας και προσφέρει μια οπτική αναπαράσταση των αποτελεσμάτων του registration με χρήση εντοπισμού ακμών και συγχώνευσης εικόνων. Επειδή ο στόχος της διατριβής αυτής ήταν να επιταχύνει τον αλγόριθμο του registration, αυτό το στάδιο έμεινε έξω από την τελική υλοποίηση για λόγους απλότητας. Μελλοντικές επεκτάσεις με μια πιο πρακτική εφαρμογή θα μπορούσαν να ενσωματώνουν αυτό το τελικό στάδιο στο κομμάτι του λογισμικού, το οποίο δεν επιβαρύνει σημαντικά την επεξεργασία χρονικά.

Υπάρχουν ορισμένες παράμετροι οι οποίες εκτός από την αποτελεσματικότητα του αλγορίθμου επηρεάζουν και το χρόνο εκτέλεσης. Μία από αυτές προφανώς είναι το μέγεθος της εικόνας. Όσο μεγαλύτερη η ανάλυση των εικόνων τόσο μεγαλύτερος ο όγκος δεδομένων προς επεξεργασία. Επίσης τα δυο προαναφερθέντα κριτήρια επηρεάζουν και αυτά με τη σειρά τους. Πολύ μικρός αριθμός επαναλήψεων μπορεί να μην είναι επαρκής για τον optimizer να βρει μια βέλτιστη λύση, ενώ πολύ μεγάλος μπορεί να έχει ως αποτέλεσμα περιττές επαναλήψεις (επειδή ο optimizer έχει ήδη βρει το βέλτιστο δυνατό μετασχηματισμό και συνεχίζει να ψάχνει άσκοπα). Κατά παρόμοιο τρόπο επηρεάζει και το κατώφλι ανοχής. Ένα πολύ μικρό τέτοιο όριο προσδίδει στον optimizer ένα πολύ αυστηρό κριτήριο τερματισμού, με αποτέλεσμα πολλές

φορές η διαδικασία να γίνεται αέναη, λόγω αδυναμίας εύρεσης τόσο μικρής ανοχής. Από την άλλη αν αυτό το όριο είναι πολύ μεγάλο ο optimizer θα νομίζει ότι βρήκε βέλτιστη λύση πρόωρα και θα τερματίσει προτού όντως την πετύχει. Τέλος σημαντικό ρόλο παίζουν και οι μέγιστες επιτρεπτές τιμές στις παραμέτρους μετασχηματισμού που επιβάλει ο χρήστης. Αυτές εξαρτώνται συνήθως από τις προδιαγραφές του dataset, και επειδή χρησιμοποιούνται και στην αρχικοποίηση των παραμέτρων επηρεάζουν όχι μόνο τον χρόνο εκτέλεσης αλλά και την αποτελεσματικότητα του αλγορίθμου.

Συσχεδίαση Υλικού/Λογισμικού της Εφαρμογής στο Zybo

Ο όρος συσχεδίαση αναφέρεται στην παράλληλη τόσο του υλικού όσο και του λογισμικού. Το λογισμικό είναι αυτό που εκτελείται στην κεντρική μονάδα επεξεργασίας (CPU), ενώ το υλικό στο FPGA. Με κατάλληλη εκμετάλλευση των δυνατών χαρακτηριστικών του καθενός από τα δύο πεδία είναι δυνατόν να υλοποιηθεί μια πιο αποτελεσματική σχεδίαση. Ο διαχωρισμός σε λογισμικό και υλικό γίνεται ύστερα από εκτενή μελέτη διαφόρων χαρακτηριστικών του προγράμματος, με τα κύρια να είναι η πολυπλοκότητα των πράξεων, ο απαιτούμενος χρόνος εκτέλεσης και οι απαιτήσεις σε μνήμη. Στη συγκεκριμένη εφαρμογή πρωταγωνιστικό ρόλο για το διαχωρισμό έπαιξε ο χρόνος εκτέλεσης. Εκτός από τη μέτρηση του συνολικού χρόνου, μετρήθηκε και ο χρόνος που δαπανάται σε κάθε ένα από τα προαναφερθέντα στάδια επεξεργασίας. Ο παρακάτω πίνακας αντιπροσωπεύει το χρόνο (μαζί με τα ποσοστά) του "Transform and Measure" σταδίου στο Zybo. Μιας και το δεύτερο στάδιο που αφορά την αρχικοποίηση καλεί την ίδια συνάρτηση, στο χρόνο που παρουσιάζεται έχει συνυπολογιστεί ο χρόνος του $2^{ου}$ και του $4^{ου}$ σταδίου. Όπως είναι προφανές, η συνάρτηση "affine_correl_func" είναι το πιο χρονοβόρο στοιχείο του αλγορίθμου σε όλες τις περιπτώσεις.

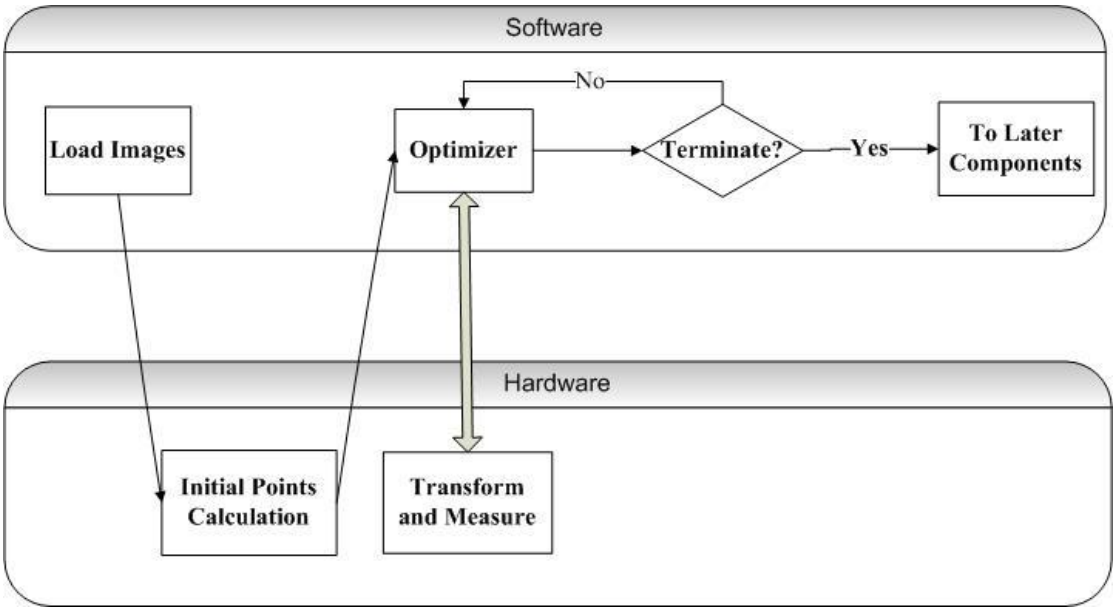
Πίνακας 2: Καταμέτρηση χρόνων στο Zybo για καλό και κακό ζεύγος ματιών

	Good Pair	Bad Pair
Total Time (s)	39,923572	103,084555
Transform and Measure time (s)	39,921616	103,080554
Percentage	99,99%	99,99%

Με μια απλή σύγκριση με το χρόνο εκτέλεσης που παρουσιάστηκε στον Πίνακα 1 γίνεται φανερό το πόσο αργός είναι ο ARM επεξεργαστής που προορίζεται για

ένα ενσωματωμένο σύστημα σε σχέση με έναν επεξεργαστή Desktop υπολογιστή. Γίνεται προφανές ότι η συνάρτηση "affine_correl_func" είναι αυτή που θα υλοποιηθεί στο FPGA. Ο όρος Good Pair αναφέρεται σε επιτυχές registration μεταξύ δύο εικόνων του ίδιου ματιού. Ο όρος Bad Pair αναφέρεται σε εικόνες του ίδιου ματιού οι οποίες δεν ήταν δυνατό να ευθυγραμμιστούν, είτε λόγω έντονων αρχικών παρεκκλίσεων είτε λόγω της παρουσίας ελαστικών παραμορφώσεων τις οποίες δεν μπορεί να αντιμετωπίσει το affine μοντέλου μετασχηματισμού. Μπορεί επίσης προφανώς να αναφέρεται σε δοκιμασμένο registration για εικόνες διαφορετικών ματιών, με τις οποίες είναι βέβαιο ότι δεν υπάρχει δυνατή ευθυγράμμιση. Είναι συνηθέστερο τα "κακά" ζευγάρια να χρειάζονται παραπάνω χρόνο εκτέλεσης, επειδή ο optimizer θα προσπαθήσει να αξιοποιήσει όλο το πλήθος των επιτρεπτών επαναλήψεων που του παρέχεται.

Από τα παραπάνω όπως ειπώθηκε γίνεται αντιληπτό το κομμάτι που θα υλοποιηθεί στο FPGA. Η φόρτωση της εικόνας και ο optimizer πρόκειται να υλοποιηθούν στο λογισμικό. Κάθε φορά που ο optimizer απαιτεί τον υπολογισμό του μέτρου ομοιότητας μεταξύ της σταθερής εικόνας και της εικόνας που μετασχηματίζεται, ο επεξεργαστής θα επικοινωνεί με το FPGA και θα περιμένει μέχρι να ολοκληρωθεί η επεξεργασία του. Η διαδικασία αυτή θα επαναλαμβάνεται μέχρι ο optimizer να πετύχει σύγκλιση και να τερματίσει. Το ακόλουθο σχήμα αναπαριστά αυτήν την αρχιτεκτονική, με έμφαση στο διαχωρισμό υλικού/λογισμικού και στην επικοινωνία.



Εικόνα 4: Διαχωρισμός υλικού/λογισμικού

Υλοποίηση του Συστήματος

Παρουσίαση των Συστατικών Στοιχείων στο Hardware

Η εφαρμογή αυτή συμπεριλαμβάνει πληθώρα αριθμητικών πράξεων όχι μόνο με ακεραίους, αλλά κυρίως με δεκαδικούς αριθμούς. Ως αποτέλεσμα προκύπτει το δίλημμα σχετικά με το τι ακρίβεια θα χρησιμοποιηθεί. Μικρή ακρίβεια δεκαδικών μπορεί να οδηγήσει σε σοβαρά σφάλματα ύστερα από πολλές επαναλήψεις με τη συσσώρευση πολλών αριθμητικών πράξεων, που θα έχουν ως αποτέλεσμα πιθανώς ανεπιτυχές registration. Η αρχική υλοποίηση αποκλειστικά σε λογισμικό χρησιμοποιούσε 64-bit αναπαράσταση των δεκαδικών. Ωστόσο, επειδή οι πόροι του FPGA είναι περιορισμένοι και τα αριθμητικά κυκλώματα διπλής ακρίβειας (64-bit) καταλαμβάνουν πολλούς πόρους, δοκιμάστηκε και η μονή ακρίβεια με 32-bit αναπαράσταση. Παρατηρήθηκε ότι τα αποτελέσματα είχαν πολύ μικρό σφάλμα σε σχέση με την αρχική υλοποίηση, σφάλμα το οποίο οπτικά δεν γίνεται αντιληπτό στο τέλος. Έτσι επιλέχθηκε η μονή ακρίβεια για υλοποίηση στο FPGA. Επειδή τα ψηφιακά κυκλώματα για την υλοποίηση αριθμητικών πράξεων με δεκαδικούς αριθμούς είναι πολύπλοκα, και η βελτιστοποίηση καθενός από αυτά ξεχωριστά αποτελεί ένα ερευνητικό έργο από μόνο του, έγινε χρήση έτοιμων κυκλωμάτων που παρέχει η Xilinx σε μορφή IP (intellectual property) και των οποίων η λειτουργικότητα είναι προσαρμόσιμη, ανάλογα με τις ανάγκες του σχεδιαστή. Πιο συγκεκριμένα, τα παρακάτω χαρακτηριστικά μπορούν να προσαρμοστούν:

- **Λειτουργία:** Το IP μπορεί να ρυθμιστεί για να υλοποιήσει διαφορετικές πράξεις (όπως πρόσθεση, πολλαπλασιασμό, συσσώρευση, αντιστροφή κλπ.).
- **Latency:** Latency είναι ο αριθμός των κύκλων που απαιτούνται μέχρι το αποτέλεσμα μιας πράξης να είναι έτοιμο, από τη στιγμή που θα δοθούν οι έγκυρες είσοδοι στο κύκλωμα. Η Xilinx έχει θέσει αυτό το χαρακτηριστικό σε μια συνιστώμενη τιμή, αλλά αν χρειαστεί, μπορεί να μειωθεί σε βάρος της συχνότητας λειτουργίας και με επιπλέον δέσμευση πόρων. Αυτή η μείωση ωστόσο δεν είναι σημαντική για την επιτάχυνση της εφαρμογής, καθώς η όλη διαδικασία είναι pipelined και έτσι ο χρόνος επεξεργασίας εξαρτάται από τον αριθμό των επαναλήψεων που χρειάζονται να γίνουν. Επομένως, η μέγιστη συνιστώμενη τιμή επιλέχθηκε για όλα αυτά τα κυκλώματα, ώστε να εξοικονομηθούν όσο περισσότεροι πολύτιμοι πόροι γίνεται.
- **Συχνότητα λειτουργίας:** Αυτή η παράμετρος υπαγορεύει τη μέγιστη συχνότητα που μπορεί να λειτουργήσει το κύκλωμα. Διατηρώντας το latency υψηλό και χρησιμοποιώντας περισσότερα DSPs μπορεί να οδηγήσει σε μεγαλύτερη επιτρεπτή συχνότητα λειτουργίας.
- **Χρήση πόρων:** Λόγω του μεγάλου ποσού πράξεων που πρέπει να εκτελεστούν, η παρακολούθηση των πόρων που χρησιμοποιούνται είναι ιδιαίτερα

σημαντική, ώστε να διασφαλιστεί ότι το τελικό στάδιο υλοποίησης είναι εφικτό, για υλοποίηση στο ZYBO. Εξοικονόμηση πόρων επιτυγχάνεται είτε διατηρώντας ψηλά το latency είτε χρησιμοποιώντας περισσότερα DSPs.

- **Ακρίβεια εισόδων και εξόδων:** Οι διαθέσιμες ακρίβειες ήταν η μισή (16-bit αναπαράσταση αριθμών κινητής υποδιαστολής), η απλή (32-bit) και η διπλή (64-bit).

Αυτά όσον αφορά τις ιδιότητες των κυκλωμάτων που θα χρησιμοποιηθούν για την υλοποίηση των πράξεων. Η σχεδίαση του συνολικού συστήματος που υλοποιεί το "Transform and Measure" απαιτεί την κατάλληλη διασύνδεση των ανωτέρω κυκλωμάτων, προσεκτικό συντονισμό των σημάτων και των δεδομένων, καθώς και κατάλληλους ελεγκτές λειτουργίας. Για τη διευκόλυνση της σχεδίασης η επεξεργασία χωρίστηκε σε 4 στάδια, το κάθε ένα από τα οποία συνδέεται με τα επόμενα του, και υλοποιήθηκαν και 4 μονάδες ελέγχου. Παρακάτω γίνεται περιληπτική παρουσίαση αυτών:

- **Μετασχηματισμός (Transformation):** Εφαρμόζει το μετασχηματισμό σε κάθε ένα από τα pixel της κινούμενης εικόνας. Είσοδοι είναι οι παράμετροι μετασχηματισμού και οι συντεταγμένες και έξοδοι οι μετασχηματισμένες συντεταγμένες (με το δεκαδικό μέρος αποκομμένο), και δύο τιμές που θα χρειαστούν για την παρεμβολή.
- **Υπολογισμός Βαρών Παρεμβολής (Interpolation weights):** Αξιοποιεί τις δυο τιμές που αναφέρθηκαν και υπολογίζει τα τέσσερα απαιτούμενα βάρη για εφαρμογή μιας παρεμβολής σε μια γειτονιά τεσσάρων pixel γύρω από τις τελικές συντεταγμένες.
- **Υπολογισμός τελικής τιμής μετασχηματισμένου pixel μετά την παρεμβολή (Interpolation calculation):** Δέχεται ως εισόδους τις τιμές των pixel για καθένα από τα 4 pixel και τα 4 βάρη του προηγούμενου σταδίου. Έξοδος είναι η τελική τιμή pixel που προωθείται στο τελικό στάδιο
- **Συσσωρευτές (Accumulations):** Υπολογίζει τα αθροίσματα της Εξίσωσης 1, λαμβάνοντας ως εισόδους την έξοδο του προηγούμενου σταδίου, καθώς και την τιμή του pixel της σταθερής εικόνας. Κατάλληλος ελεγκτής αναλαμβάνει την προώθηση των pixel από τη μνήμη όταν αυτά χρειάζονται.

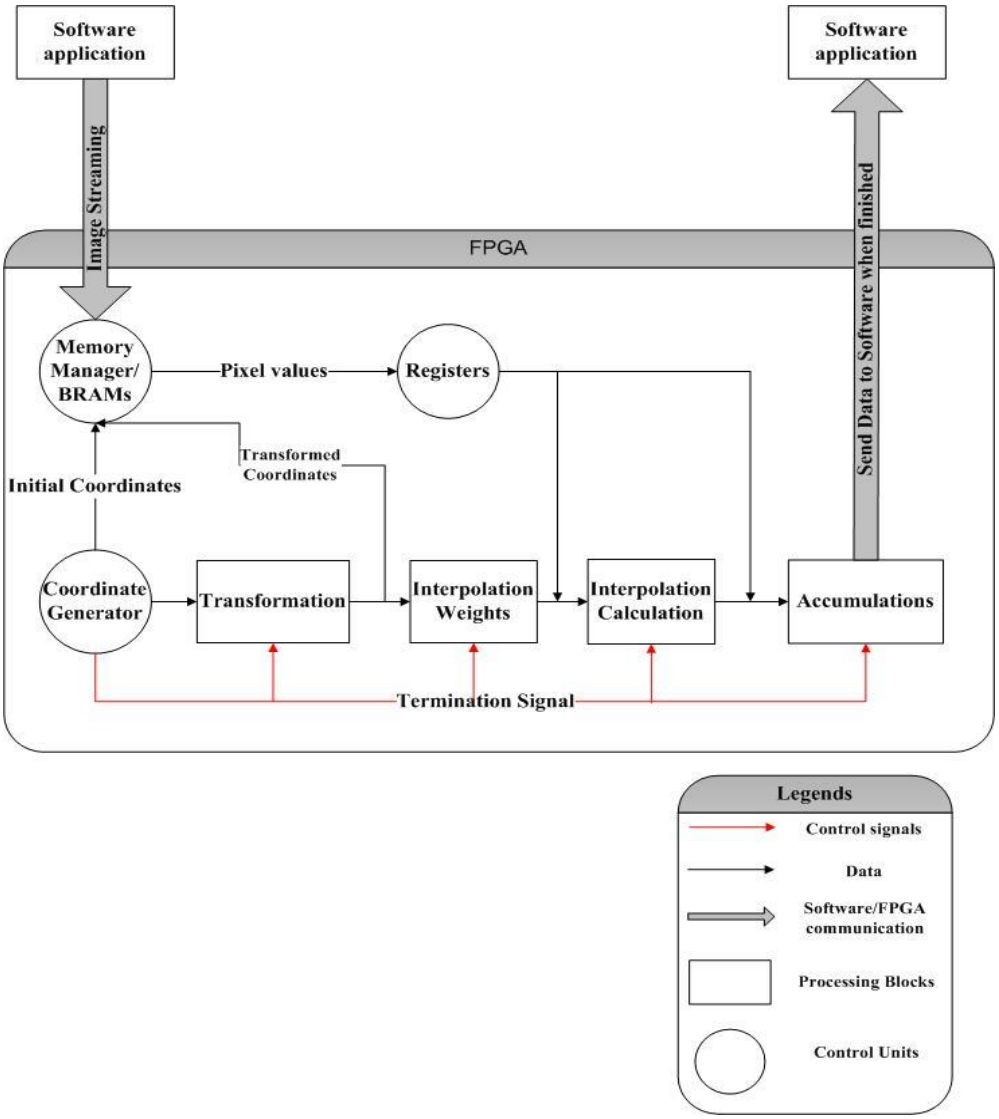
Οι ελεγκτές που σχεδιάστηκαν περιγράφονται παρακάτω:

- **Γεννήτρια Συντεταγμένων (Coordinate Generator):** Παράγει γραμμή ανά γραμμή ζεύγη συντεταγμένων και θέτει κατάλληλα σήματα εγκυρότητας. Επίσης ελέγχει το πότε τερματίζει η επεξεργασία και αναμένει και κατάλληλο σήμα επανεκκίνησης από το software.
- **Διαχειριστής Μνήμης (Memory manager):** Δέχεται ως είσοδο συντεταγμένες από τα στάδια επεξεργασίας και επιστρέφει τιμές των pixel που αντιστοιχούν στις συντεταγμένες αυτές. Επίσης είναι υπεύθυνος για τη φόρτωση των

εικόνων στις Block Rams του FPGA στην αρχή του προγράμματος. Το πώς λαμβάνονται οι εικόνες από το software θα αναφερθεί αργότερα.

- **Εγκυρότητα Συντεταγμένων:** Κάποιες συντεταγμένες μπορεί να αντιστοιχιστούν σε μη έγκυρες τοποθεσίες μετά το μετασχηματισμού, βγαίνοντας δηλαδή εκτός ορίων. Αυτός ο ελεγκτής φροντίζει να θέσει το σήμα εγκυρότητας των συντεταγμένων αυτών στο 0.
- **Καταχωρητές (Registers):** Φροντίζουν το συντονισμό, καθυστερώντας σήματα ή δεδομένα τόσους κύκλους ρολογιού όσους χρειάζονται.

Η Εικόνα 5 παρουσιάζει όλα τα αναφερθέντα συστατικά στοιχεία της σχεδίασης σε ένα πλήρες σύστημα.



Εικόνα 5: Υλοποίηση του Συστήματος στο Hardware

Υλοποίηση Επικοινωνίας Μεταξύ Επεξεργαστή και FPGA

Δεδομένου ότι το μέγεθος της μνήμης BRAM σε ένα FPGA δεν είναι αρκετά μεγάλο για να αποθηκευτεί μια ολόκληρη εικόνα υψηλής ευκρίνειας (256kB συνολική μνήμη στο Zybo), απαιτείται μια αποτελεσματική, αξιόπιστη και γρήγορης μετάδοσης μεταφορά δεδομένων. Το πρωτόκολλο AXI4-stream είναι η πιο αποτελεσματική διαθέσιμη επιλογή.

Η γενική στρατηγική στις περισσότερες εφαρμογές επεξεργασίας εικόνας είναι να ξεκινήσει το streaming των πρώτων γραμμών της εικόνας, έτσι ώστε να μπορεί να εκκινήσει την επεξεργασία του το FPGA. Τα δεδομένα που λαμβάνονται αποθηκεύονται προσωρινά σε BRAMs, και οι επόμενες γραμμές μπορούν να συνεχίσουν να αποστέλλονται όταν η επεξεργασία έχει φτάσει κάπου στη μέση και τα αρχικά δεδομένα που είναι ήδη αποθηκευμένα στη BRAM δεν είναι πια απαραίτητα. Χρησιμοποιώντας ένα ενιαίο κανάλι ροής, ένα νέο 32-bit δεδομένο μπορεί να μεταφερθεί κάθε διαδοχικό κύκλο ρολογιού (που ισοδυναμεί με τέσσερα pixel, γιατί ένα greyscale pixel απαιτεί 8 bits για την αναπαράσταση του). Αυτό το μοτίβο αποτελεσματικής επικοινωνίας στηρίζεται στο μηχανισμό άμεσης προσπέλασης μνήμης (DMA).

Το DMA είναι ένας πολύ σημαντικός μηχανισμός που πλέον χρησιμοποιείται στα περισσότερα ενσωματωμένα συστήματα, αλλά και σε όλους τους desktop υπολογιστές και όχι μόνο. Επιτρέπει την αποτελεσματική επικοινωνία μεταξύ του επεξεργαστή, της κεντρικής μνήμης και των περιφερειακών παντός είδους. Χωρίς το DMA, σε παλαιότερα υπολογιστικά συστήματα ο επεξεργαστής έπρεπε να εκτελεί συνεχώς εντολές ανάγνωσης και εγγραφής σε μνήμη για να επικοινωνεί με περιφερειακά, με αποτέλεσμα πολλές φορές να πρέπει να περιμένει άσκοπα μέχρι τα δεδομένα να είναι έτοιμα και να σπαταλάει πολύτιμο χρόνο που θα μπορούσε να αφιερώνει σε διαφορετικές εφαρμογές, με αποτέλεσμα την εξαιρετική μείωση του throughput και της ολικής ταχύτητας και κατανάλωσης. Το DMA συνδέει τις εξωτερικές συσκευές με τον κύριο δίαυλο μεταφοράς δεδομένων που συνδέει τον επεξεργαστή με τη μνήμη. Έτσι πλέον ο επεξεργαστής απαλλάσσεται από το καθήκον των πολλαπλών χειραψιών που είχε και πλέον μεσολαβεί μόνο μια φορά στην αρχή και στο τέλος της μεταφοράς δεδομένων, επιτυγχάνοντας έτσι να εκτελεί άλλα καθήκοντα ενώ περιμένει δεδομένα από τη συσκευή.

Επειδή η σχεδίαση ενός αποτελεσματικού μηχανισμού επικοινωνίας που χρησιμοποιεί το DMA είναι μια αρκετά πολύπλοκη διαδικασία που απαιτεί γνώση πολύπλοκων drivers και επιβαρύνει τον σχεδιαστή χρονικά, ένα έτοιμο εργαλείο που ονομάζεται Xillybus χρησιμοποιήθηκε για την υλοποίηση της επικοινωνίας. Εκτός του ότι ο πυρήνας του Xillybus ενσωματώνεται εύκολα στην εφαρμογή, είναι και προσαρμόσιμος στις ανάγκες και προαπαιτήσεις της εφαρμογής.

Τα Xillybus IP και Xillybus-Lite IP Cores

Το Xillybus είναι μια οικογένεια πυρήνων υλικού IP που αναπτύχθηκαν από την Xillybus Ltd. και είναι συμβατοί με πολλούς κατασκευαστές FPGA (Xilinx, Altera) και αναπτυξιακών πλακετών (Zybo, ZedBoard), καθώς και με διαφορετικά λειτουργικά συστήματα (Windows, Linux). Το Xillybus είναι ένα σύστημα που υλοποιεί το μηχανισμό DMA και παρέχει μια έτοιμη διεπαφή για επικοινωνία μεταξύ του επεξεργαστή και της προγραμματίσιμης λογικής στο FPGA.

Πιο συγκεκριμένα, ο επιταχυντής στο FPGA συνδέεται με το IP χρησιμοποιώντας συνηθισμένες FIFOs (First-In-First-Out δομές δεδομένων), το πλάτος και το βάθος των οποίων μπορούν να ρυθμιστούν από το σχεδιαστή. Διαφορετική FIFO χρησιμοποιείται για τα δεδομένα που μεταδίδονται από τον host προς τον accelerator και διαφορετική για τον accelerator προς τον host. Κατά την εκκίνηση του συστήματος, ο Xillybus driver εκχωρεί κατάλληλους DMA buffers στην κύρια μνήμη. Τα δεδομένα που μεταφέρονται από ή προς τον επιταχυντή αποθηκεύονται πρώτα σε αυτούς τους buffers. Από την πλευρά του λογισμικού, έχει αναπτυχθεί ένα τετριμμένο προγραμματιστικό μοντέλο που διαχειρίζεται το κάθε ρεύμα επικοινωνίας (ανάγνωση και εγγραφή) ως αρχεία συσκευών (device files). Αυτό σημαίνει ότι υποστηρίζονται οι κλασσικές εντολές “open”, “close”, “read”, “write”. Το όνομα της συσκευής επιλέγεται κατά το στήσιμο του συστήματος στο IP core factory και μπορεί να βρεθεί στο /dev/ directory σε ένα Linux λειτουργικό σύστημα. Έτσι μια τυπική εντολή ανοίγματος του αρχείου θα μπορούσε να είναι κάπως έτσι:

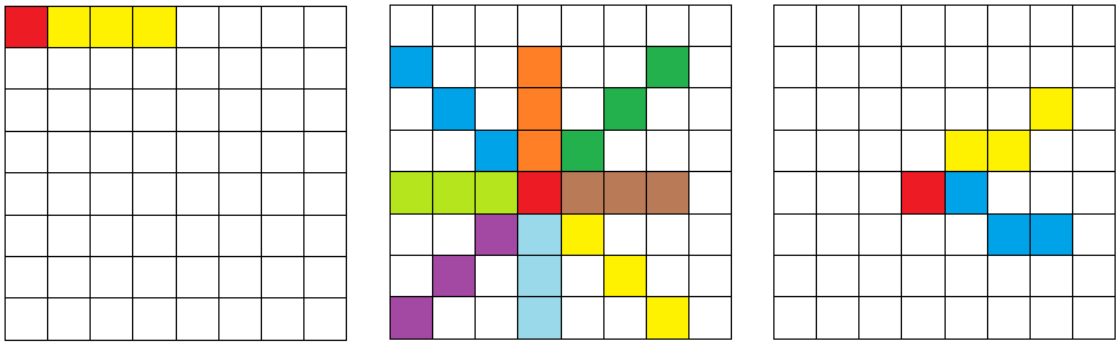
fd=open(“/dev/host_to_fpga_device”,O_WRONLY);

Το Xillybus-Lite είναι ένας πυρήνας που υλοποιεί το πρωτόκολλο επικοινωνίας AXI4-Lite, το οποίο είναι βολικό και αποτελεσματικό για επικοινωνία σημάτων (π.χ. σήματα εκκίνησης, τερματισμού, ενημέρωσης κλπ.), αφού προσφέρει εύκολη πρόσβαση σε μονάδες μνήμης εντός του FPGA. Το προγραμματιστικό μοντέλο είναι σχεδόν ίδιο με αυτό του Xillybus, με τη μόνη διαφορά να έγκειται στο ότι η CPU και το PL πρέπει να έχουν μια κοινή αντιστοίχιση μνήμης για αναφορά στις ίδιες διευθύνσεις. Επομένως, η διεύθυνση στο χώρο εικονικής μνήμης της διεργασίας software πρέπει να αντιστοιχιστεί στην πραγματική φυσική διεύθυνση της δομής μνήμης (η οποία είναι συνήθως μια 32x32bit RAM). Μετά τις κατάλληλες ρυθμίσεις και αρχικοποιήσεις, η ανάγνωση ή εγγραφή στη φυσική διεύθυνση της μνήμης RAM ανάγεται σε μια απλή λειτουργία ανάθεσης μεταξύ μεταβλητών σε προγραμματιστικό επίπεδο.

Το Πρόβλημα Μνήμης και η Επίλυση του

Η αρχιτεκτονική που περιγράφηκε δεν μπορεί να λειτουργήσει για το dataset αυτής της διατριβής, επειδή το μέγεθος των εικόνων είναι μεγαλύτερο από τη διαθέσιμη μνήμη στις Block Rams του FPGA. Κάθε εικόνα έχει μέγεθος 1Mbyte, ενώ η συνολική μνήμη που είναι διαθέσιμη στις Block Rams είναι 256Kbytes. Αυτό το πρόβλημα θα μπορούσε να αποφευχθεί εάν το registration χρησιμοποιούσε ένα κλασσικό μοτίβο πρόσβασης στη μνήμη όπως οι περισσότερές εφαρμογές στην επεξεργασία εικόνων. Αυτό που συνήθως γίνεται είναι να στέλνονται λίγες γραμμές κάθε φορά με γραμμικό τρόπο (δηλαδή από την αρχή και μια ή περισσότερες διαδοχικές γραμμές κάθε φορά) έτσι ώστε να έχει ο επιταχυντής δεδομένα να επεξεργαστεί, και μιας και θα είναι γνωστό το ποια δεδομένα θα χρειαστεί στη συνέχεια, αυτά να προετοιμάζονται για να αποσταλούν αμέσως μετά.

Αυτό δεν υφίσταται στην περίπτωση του registration solver, γιατί από τη φύση των μετασχηματισμών δεν είναι δυνατόν να είναι γνωστό το μοτίβο προσπέλασης της μνήμης εκ των προτέρων, καθώς διαφορετικά πρόσημα και τιμές των 6 παραμέτρων οδηγούν σε διαφορετικές διευθύνσεις κίνησης με διαφορετικές κλίσεις μέσα στην εικόνα. Ένα σχηματικό παράδειγμα για να γίνει αυτή η συμπεριφορά κατανοητή φαίνεται στην Εικόνα 6. Στον αριστερό πίνακα φαίνεται το αρχικό pixel της σταθερής εικόνας και με κίτρινο συμβολίζεται η κλασσική γραμμική της προσπέλαση. Στο μεσαίο πίνακα το κόκκινο pixel είναι το μετασχηματισμένο κόκκινο του αριστερού πίνακα, και με κάθε διαφορετικό χρώμα συμβολίζεται και μια διαφορετική πιθανή διεύθυνση προσπέλασης. Μάλιστα, λόγω αποκοπής του δεκαδικού μέρους μετά το μετασχηματισμό, η προσπέλαση μπορεί να είναι ακόμα πιο πολύπλοκη, όπως φαίνεται στον δεξιά πίνακα. Ως αποτέλεσμα είναι αδύνατον να βρεθεί ένας αποτελεσματικός τρόπος προσπέλασης της μνήμης, γιατί απλά αυτός δεν είναι γνωστός εκ των προτέρων, αλλά προκύπτει κατά τη διάρκεια της επεξεργασίας.



Εικόνα 6: Διαφορετικά μοτίβα προσπέλασης μετά την εφαρμογή του μετασχηματισμού

Για την αποφυγή του προβλήματος αυτού, δοκιμάστηκε να γίνει σμίκρυνση (downsizing) της εικόνας σε διαστάσεις 256x256. Αυτό που παρατηρήθηκε ήταν πο-

λύ ενδιαφέρον και χρήσιμο. Ο τελικός βέλτιστος μετασχηματισμός που παράγει ο optimizer για τις downsized εικόνες έχει πολύ μικρά σφάλματα όσον αφορά τις γωνίες και τις κλιμακώσεις, και οι παραγόμενες μετατοπίσεις είναι σχεδόν το ¼ των αρχικών (όπως προκύπτει από τη διαίρεση 1000/256). Αυτό σημαίνει ότι μπορούν να χρησιμοποιηθούν οι downsized εικόνες για να βρεθεί ο βέλτιστος μετασχηματισμός των αρχικών εικόνων. Λόγω λιγότερων δεδομένων προς επεξεργασία επιτυγχάνεται έτσι και μια επιτάχυνση, σε συνδυασμό με την επιτάχυνση που θα επιτευχθεί λόγω του FPGA. Απαιτείται βέβαια να ενσωματωθεί λογισμικό κομμάτι που θα υλοποιεί τη σμίκρυνση, του οποίου ο χρόνος εκτέλεσης είναι μικρός σε σχέση με το registration. Στα πλαίσια της διπλωματικής, χρησιμοποιήθηκε ένα έτοιμο εργαλείο για να ληφθούν οι downsized εικόνες του dataset, οπότε δεν ενσωματώθηκε σαν πρόγραμμα στο λογισμικό η διαδικασία αυτή.

Αξιολόγηση του Συστήματος

Στο κεφάλαιο αυτό θα γίνει παρουσίαση των υπολογιστικών πόρων που χρειάζεται η τελική υλοποίηση, καθώς και μια σύγκριση παραγόμενων αποτελεσμάτων και χρόνων εκτέλεσης μεταξύ της υλοποίησης με συσχεδίαση υλικού/λογισμικού και διαφόρων άλλων υλοποιήσεων. Ο πίνακας 3 παρουσιάζει το πλήθος των διαθέσιμων πόρων του FPGA που δεσμεύτηκαν από τη σχεδίαση.

Πίνακας 3: Χρησιμοποίηση Πόρων

Z-7010 resources	Available	Used	Utilization
LUT	17600	14164	80.48%
FF	35200	22621	64.26%
BRAM	60	35	58.33%
DSP	80	76	95%

Γίνεται φανερό γιατί η εφαρμογή απαιτεί πολλούς πόρους, ως αποτέλεσμα της εκτεταμένης χρήσης της αριθμητικής κινητής υποδιαστολής, και γιατί η διπλή ακρίβεια δε θα μπορούσε να υλοποιηθεί, δεδομένου ότι απαιτεί πολύ περισσότερους πόρους από την 32-bit αναπαράσταση. Λόγω οριακής χρήσης των Look Up Tables (LUTs), το συγκεκριμένο FPGA δεν μπορεί να υποστηρίξει παραλληλία. Αν υπήρχε επαρκές πλήθος περισσευόντων πόρων, θα μπορούσε το υλικό να διπλασιαστεί ή τριπλασιαστεί και ολόκληρη η επεξεργασία να χωριστεί σε ανεξάρτητα κομμάτια που θα λειτουργούν παράλληλα, πολλαπλασιάζοντας έτσι την επιτάχυνση κατά έναν παράγοντα 2 ή 3 αντίστοιχα.

Ο παρακάτω πίνακας επιδεικνύει πώς μεταβάλλεται το πλήθος λογικών μονάδων που χρησιμοποιείται από ένα κύκλωμα πολλαπλασιασμού με την εισαγωγή επιπλέον DSPs. Η πλήρης αξιοποίηση των DSP ήταν απαραίτητη ώστε να "χωρέσει" η σχεδίαση στο FPGA του Zybo.

Πίνακας 4: Μεταβολή Πόρων Πολλαπλασιαστή σε σχέση με πλήθος DSPs

Number of DSPs	LUTs	FFs
1	245	437
2	115	242

Ο τελικός χρόνος εκτέλεσης της συσχεδίασης στο Zybo συγκρίθηκε με τον αντίστοιχο χρόνο εκτέλεσης χρησιμοποιώντας εξολοκλήρου τον ARM του Zybo (με μέγιστη συχνότητα λειτουργίας 900MHz και DDR μνήμη 512 MB) και με τον αντίστοιχο χρησιμοποιώντας μια γρήγορη, αυτόνομη CPU ενός desktop υπολογιστή (με μέγιστη συχνότητα λειτουργίας 2.4GHz και RAM 6GB). Προτού παρουσιαστεί η σύγκριση των χρόνων ένα σημαντικό χαρακτηριστικό του κυκλώματος συσσώρευσης που χρησιμοποιείται για τον υπολογισμό των αθροισμάτων στην Εξίσωση 1 πρέπει να αναλυθεί. Αυτό το κύκλωμα (που είναι σχεδιασμένο από την Xilinx) πρώτα στρογγυλοποιεί τις εισόδους και μετά εκτελεί την συσσώρευση. Ως αποτέλεσμα, ένα μικρό σφάλμα συσσώρευσης προστίθεται σε κάθε επανάληψη στον τελικό υπολογισμό λόγω της στρογγυλοποίησης. Εν απουσία ελαστικών παραμορφώσεων, όπου ο optimizer είναι ευσταθής, αυτό το σφάλμα είναι μικρό και δεν επηρεάζει αρνητικά την οπτική απεικόνιση της καταχώρισης.

Υπάρχουν δυο ακόμη παράγοντες που εισάγουν σφάλμα στους υπολογισμούς. Ο ένας είναι η επιλογή μεταξύ μονής και διπλής ακριβείας για την αριθμητική κινητής υποδιαστολής. Ο δεύτερος είναι η μέθοδος ανάγνωσης της εικόνας στο λογισμικό. Η φόρτωση των εικόνων μπορεί να γίνει είτε χρησιμοποιώντας τη βιβλιοθήκη OpenCV είτε αποθηκεύοντας την εικόνα ως αρχείο κειμένου και κάνοντας ανάγνωση από αυτό. Δεδομένου ότι η εγκατάσταση της βιβλιοθήκης OpenCV στο Zybo δεν ήταν εφικτή, χρησιμοποιήθηκε η δεύτερη μέθοδος. Στην περίπτωση ευσταθών optimizers για καλά ευθυγραμμισμένες εικόνες χωρίς ελαστικές παραμορφώσεις, τα σφάλματα αυτά δεν είναι πολύ σημαντικά. Μόλις οι 256x256 εικόνες μετασχηματιστούν ξανά σε 1000x1000, το μέγιστο σφάλμα μετατόπισης που βρέθηκε (που είναι το πιο σημαντικό) μεταξύ των διαφόρων μεθόδων ήταν 9 pixels που είναι λιγότερο από το 1% του πλάτους της εικόνας. Πρακτικά αυτό σημαίνει ότι οπτικά το registration εξακολουθεί να πετυχαίνει.

Δεδομένου ότι ο optimizer εξερευνά το χώρο παραμέτρων για μια βέλτιστη λύση, μικρά σφάλματα στις μετρούμενες ομοιότητες μπορεί να οδηγήσουν σε διαφορετικά βήματα κατά τη διάρκεια της βελτιστοποίησης. Η τελική σύγκλιση μπορεί να

είναι ακόμα περίπου η ίδια, αλλά ανάλογα με τα σφάλματα, ένα διαφορετικό πλήθος επαναλήψεων ενδέχεται να απαιτούνται μέχρι να επιτευχθεί η εν λόγω σύγκλιση. Το αποτέλεσμα αυτού του φαινομένου είναι ότι η επιτάχυνση δεν είναι η ίδια σε όλες τις περιπτώσεις, αλλά ποικίλλει ανάλογα με τις εικόνες-εισόδους. Ο παρακάτω πίνακας απεικονίζει το χρόνο εκτέλεσης για 3 επιτυχημένα registrations και 1 ανεπιτυχές για την επεξεργασία των downsized εικόνων (256x256). Κάθε χρόνος που απεικονίζεται προέρχεται από το μέσο όρο των χρόνων 100 εκτελέσεων για μια πιο σταθερή και στατιστικώς ορθή μέτρηση.

Πίνακας 5: Σύγκριση χρόνων εκτέλεσης μεταξύ όλων των υλοποιήσεων

	CPU/FPGA co-design	Standalone CPU ARM in Zybo	Standalone CPU in desktop computer
First Successful registration time(seconds)	0,31169519	12,583982	1,10332952
Speedup		x40,37	x3,53
Second Successful registration time(seconds)	0,29721025	19,729760	0,87024528
Speedup		x66,38	x2,93
Third Successful registration time(seconds)	0,29172961	10,775993	0.6954013
Speedup		x36,94	x2,38
Unsuccessful registration time(seconds)	0,26560579	11,236088	0,98315164
Speedup		x42,30	x3,37

Το τελικό θέμα που πρέπει να ληφθεί υπόψη είναι η ποιότητα των παραμέτρων μετασχηματισμού που προκύπτουν από το registration που υλοποιείται με συσχεδίαση υλικού/λογισμικού. Οι συγκρίσεις θα γίνουν σε σχέση με τις αντίστοιχες παραμέτρους που δημιουργούνται σε ένα Desktop PC που χρησιμοποιεί έναν αυτόνομο επεξεργαστή. Στην πραγματικότητα, μια πιο εκτεταμένη σύγκριση έλαβε χώρα, και παρακάτω γίνεται μια απεικόνιση των αποτελεσμάτων που προκύπτουν από 4 διαφορετικές μεθόδους:

- Αυτόνομη Desktop CPU για αρχικά μεγέθη εικόνας (1000x1000) χρησιμοποιώντας OpenCV και διπλή ακρίβεια. **(Method 1)**
- Αυτόνομη Desktop CPU για downsized εικόνες χρησιμοποιώντας OpenCV και διπλή ακρίβεια. **(Method 2)**
- Αυτόνομη ARM CPU στο Zybo για downsized εικόνες χρησιμοποιώντας μονή ακρίβεια και ανάγνωση από αρχείο. **(Method 3)**

- FPGA/CPU συσχεδίαση στο Zybo για downsized εικόνες χρησιμοποιώντας μονή ακρίβεια και ανάγνωση από αρχείο. **(Method 4)**

Θα μελετηθούν δύο παραδείγματα για κάθε μέθοδο, το ένα με πολύ μικρό τελικό σφάλμα (ουσιαστικά ασήμαντο) και το άλλο με ένα σχετικά μεγαλύτερο σφάλμα. Και τα δύο παραδείγματα αναφέρονται σε επιτυχημένα registrations. Οι Πίνακες 6 και 7 αναφέρονται στα αποτελέσματα των πρώτων δύο μεθόδων.

Πίνακας 6: Αποτελέσματα Μεθόδου 1

Method 1		
Parameters	**First pair**	**Second pair**
T1	1.009586	0.992332
T2	-0.016319	0.018466
T3	178.587102	-194.799257
T4	0.027438	-0.015719
T5	1.003454	0.990693
T6	-62.193396	-23.551012
Measure of Match	0.783938	0.827221

Πίνακας 7: Αποτελέσματα Μεθόδου 2

Method 2		
Parameters	**First pair**	**Second pair**
T1	1.002974	1.045161
T2	-0.015803	0.026128
T3	45.320175	-48.672958
T4	0.024869	0.000492
T5	0.997309	1.011216
T6	-16.065662	-5.656442
Measure of Match	0,792186	0,797225

Προκειμένου να αποφανθούμε για το εάν η δεύτερη μέθοδος είναι επαρκής διεξάγεται μια σύγκριση των παραμέτρων με τη μέθοδο 1. Από τις παραμέτρους T1, T2, T4 και T5 που αναφέρονται στην κλιμάκωση και την περιστροφή παρατηρείται ότι αποκλίσεις μεταξύ των δύο μεθόδων είναι πολύ μικρές. Οι πιο σημαντικές παράμετροι είναι οι T3 και T6 που αναφέρονται στη μετατόπιση. Πολλαπλασιάζοντας την T3 στο πρώτο ζευγάρι με το 1000/256 = 3, 90625, προκύπτει T3new = 177, 032, το οποίο είναι περίπου 1,5 pixel λιγότερο από το T3 της μεθόδου 1. Αυτό το σφάλμα είναι ασήμαντο στην οπτική απεικόνιση, μετά τον εντοπισμό ακμών και του image fusion.

Οι Πίνακες 8 και 9 αναφέρονται στις επόμενες δύο μεθόδους, οι οποίες εφαρμόστηκαν στο Zybo.

Πίνακας 8: Αποτελέσματα Μεθόδου 3

Method 3		
Parameters	**First pair**	**Second pair**
T1	1.010045	1.040799
T2	-0.015795	0.025317
T3	45.801357	-48.795601
T4	0.027911	-0.001520
T5	1.003294	1.009725
T6	-15.859092	-5.936245
Measure of Match	0,792874	0,799729

Πίνακας 9: Αποτελέσματα Μεθόδου 4

Method 4		
Parameters	**First pair**	**Second pair**
T1	0.990064	1.002257
T2	-0.017116	0.023562
T3	44.415871	-48.941826
T4	0.021519	-0.011582
T5	0.989229	1.034919
T6	-15.909966	-5.749107
Measure of Match	0,780669	0,806234

Γίνεται αμέσως φανερό ότι τα σφάλματα δεν επηρεάζουν την ποιότητα του registration solver, αρκεί βέβαια να μην υπάρχουν ελαστικές παραμορφώσεις και ο optimizer να είναι ευσταθής. Σημειώνεται ότι όλα τα ζεύγη εικόνας στο σύνολο δεδομένων, τα οποία μπορούν να ευθυγραμμιστούν επιτυχώς με τη χρήση του συγκεκριμένου registration solver, είχαν το ίδιο ικανοποιητικά αποτελέσματα με χρήση όλων των μεθόδων που αναφέρθηκαν προηγουμένως. Ως αποτέλεσμα, επιτεύχθηκε επιτάχυνση όχι μόνο λόγω της συρρίκνωση της εικόνας, αλλά επίσης λόγω της χρήσης του FPGA, επιτυγχάνοντας ακόμη μεγαλύτερες ταχύτητες από ότι αρχικά. Παρατηρούμε ότι registrations που αρχικά χρειαζόντουσαν ίσως και 100 δευτερόλεπτα στο Zybo για να ολοκληρωθούν, τώρα χρειάζονται μόλις 200-300msec, εξακολουθώντας να παράγουν ικανοποιητικά αποτελέσματα.

CHAPTER 1: INTRODUCTION

1.1 EMBEDDED SYSTEMS AND IMAGE REGISTRATION

An embedded system is defined as any device that incorporates a programmable processor, but is not by itself a general-purpose computer. Programming of embedded systems can be done in machine code or by using any higher level programming language, given that the compiler of the corresponding language is available for the particular processor. Sometimes the embedded system consists of multiple processors, peripherals, caches, interfaces and a field-programmable gate array (FPGA), in order to implement a user defined and hardware specific design to the application. Such systems are referred to as Systems on Chip (SoC).

Embedded systems target specific applications and are designed to be optimal when it comes to processing speed, power consumption, cost, reliability etc. Range of applications is actually so immense that almost 98% of all microprocessors are manufactured as components of an embedded system. Concerning the main fields of application, cars and robotics stand out, because of the wide variety of sensors and cameras these systems use and the amount of data that needs to be processed. Noteworthy is also their widely use in the majority of domestic devices or appliances, such as refrigerators and electrical ovens, as well as in medical applications.

Image processing is a field of computer science and a subcategory of digital signal processing that has a huge variety of applications. It refers to the process of extracting information (regarding geometrical features, patterns, colors etc.) from an image or a set of images and using that information for recognition or to apply a transformation to that image. Image processing is the practical solution to the following problems:

- Classification
- Pattern recognition
- Feature extraction
- Projection

Image Registration is a fundamental task frequently encountered in image processing applications. It is used in computer vision, medical imaging, security systems, military automatic target recognition and analyzing images and data collected from satellites.

Figure 1.1: Registration Example

Figure 1.1 presents an example of how image registration works. The left picture is the so called floating image, meaning the image that will be transformed in order to be aligned with the middle picture, which is called the reference image. If the alignment works and the algorithm returns a good measure of match, then by observing the result of the transformation one can conclude whether or not these two photographs are of the same place.

In medical applications, images of similar or differing modalities often need to be aligned as a preprocessing step for many planning, navigation, data-fusion and visualization tasks. Commonly, the alignment of imaging data allows medical experts to compare different snapshots in time of a specific anatomical region, hence evaluate the progression or regression of certain pathomorphic conditions or the success rate of a followed treatment scheme.

It is quickly understood why embedded systems are widely used to run image processing applications. Application specific design will allow the effective use of the available resources in order to (sometimes) achieve real-time execution if required, and help meet the low power requirements.

1.2 SoC FPGA in Image Processing/Image Registration

Designing the application using FPGA has both advantages and disadvantages. The main advantages are listed below:

1. High parallelism: Since the FPGA consists of a large amount of independent (but connected in a programmable way) logic elements, it enables high levels of parallelism, since different segments of the program's code can

be mapped into different logic blocks and run separately from the rest of the code. Of course, data dependencies will still exist, obliging careful synchronization and design flow.

2. Effective Pipelining: The multi-stage pipelining of a standard processor is highly affected by data-dependencies, which means that a Read-after-Write dependency will always have to stall until the required data is valid and ready. This pipelining is also limited to only a few streams of operations. The FPGA however allows a multi-stage pipelining in which the data dependencies will only stall once in the first operation in a loop of operations (stall will be equal to the latency of the operation), whereas the processor's pipelining would need to stall in each of the iterations. What this means is that in the FPGA the IPC (instructions per cycle) will be equal to 1 (or n, where n the level of parallelism), whereas in a standard processor system that will never be the case. This means that operations that will be repeated multiple times in a loop will be executed a lot faster in an FPGA.

3. Processing Speed: Because of the first two reasons it is easily concluded that programs that require multiple iterations will be executed a lot faster in an FPGA. In most image processing applications, if the available resources in the FPGA allow it, the image can also be split in sub-blocks and the whole process will be partitioned into independent parallel processing blocks.

4. Lower power consumption: One of the major factors of power consumption in digital systems is the clock frequency. Since the FPGA can achieve higher speeds using parallelism and pipelining, the clock's period can be significantly increased (thus reducing the frequency proportionally), resulting in lower power consumption.

The main disadvantages are listed below:

1. High complexity and development time: Despite the obvious advantages that were previously listed, the development time in designing an application on an FPGA can be longer than expected. The main reasons are the high complexity of the lowest possible level of design, in combination with the time-consuming debugging, which requires re-generation of the bitstream that programs the FPGA every time a new change needs to be tested.

2. Resource and Memory Limitations: Although FPGAs have plenty of resources that can be sufficient for most common applications, it may not always be the case. Certain complex algorithms with a higher than average data processing may not be optimally implemented on an FPGA, because of limited resources. Also, unorthodox memory access patterns may render the problem memory-bound, thus making it difficult to implement the design effectively.

1.3 THE GOAL OF THIS THESIS

As mentioned above, it is obvious that image processing has a wide variety of applications and as such, more and more different algorithms have been introduced in order to address each one of the different problems. However, since the demand for better image quality, reliability and effectiveness has dramatically increased, the computational complexity of these algorithms is growing more and more. As a result, both the processing time and the power consumption have also been affected, rendering the standard processing system, using a standalone processor, not the optimal choice of hardware. Another important thing to keep in mind is that the processor of an embedded system is of a much lower cost and a lot slower in speed than a processor used in desktop computers, meaning that if a costly efficient, and yet fast and with low-power solution needs to be found a different architecture needs to be adopted. Such systems that aim for this kind of solution make use of a combination of the processor with a DSP or a GPU or a SoC FPGA. All these systems aim to exploit the parallel nature of the image processing algorithms, which the processor cannot make efficient use of, and also accomplish a reduction in the power needed.

The goal of this thesis is to make use of a SoC FPGA to accelerate and reduce the power required by an image registration algorithm. More specifically, the design will focus on acknowledging which parts of the algorithm are the most computationally expensive and implement a hardware design on the FPGA specifically for these parts. The application's dataset includes photographs of the iris of different eyes taken by high resolution cameras. Registration of these eyes can have security or medical purposes. The system will be receiving two images at a time, one reference and one floating, and will continuously apply geometric transformations to the floating image in an effort to align it to the reference image. The parameters of this transformation will be regulated by an optimizing algorithm, whose exact functionality will be presented later. Once and if a satisfactory transformation has been found (which achieves the best possible alignment) the optimizer will stop. Later a final transformation can be applied to produce the final image, as well as an edge detection algorithm to enable us to visually compare the reference image with the optimally transformed floating image and decide whether the alignment was successful or not.

One of the most important steps of the design is the application timing profiling to identify which part of the algorithm is required to be implemented on the FPGA as described previously. The hardware components of the design will be described using the hardware description language VHDL, the synthesis and implementation using the Vivado tool, while the software part of the application will be described using the programming language C. One final idea that is important to keep in mind, is that despite the fact that the first time development of a software-hardware co-design can be time consuming and complex for an engineer, the basic ideas and methodologies that he will use will act as great experience and guidelines for his future work and projects.

CHAPTER 2: RELATED WORK

There have been other research studies that deal with the acceleration of registration methods using an FPGA hardware implementation. Omkar Dandekar et al. [11] present a field-programmable gate array-based architecture for accelerated implementation of mutual information (MI)-based deformable registration. Their design addresses the need to register pre and intraprocedural images for improved intraprocedural target delineation in the image-guided-intervention (IGI) workflow. They managed to reduce the execution time of this complex and time-consuming algorithm from hours to a few minutes, thus making it usable and efficient in a clinical environment. Figure 2.1 illustrates a comparison of intraprocedural and preprocedural images for the same subject. The left image is an example of intraprocedural noncontrast Computed Tomography (CT), and the right image shows a preprocedural contrast-enhanced CT (preCT).

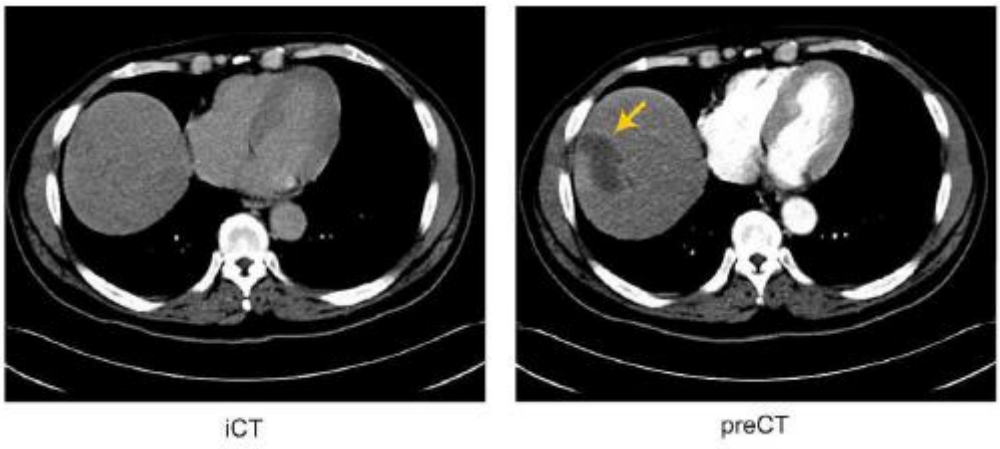


Figure 2.1: Example of an Image pair pre and intraprocedural CT.

In addition, Brandyn Allen White [12] presents a high performance FPGA-based direct affine image registration core. Direct methods are known for their high accuracy, however are very computationally expensive, due to the continuous intensity derivatives calculation, which often prevents their use in an embedded system. The system is run at 100Mhz and achieves a registration speed of 82 frames per second (0.01222 seconds per frame) for image sizes of 640x480, while a floating point Matlab Implementation on a 2.4Ghz Inter Core 2 Quad required 5 seconds per frame, thus succeeding a speedup of over 400 times. Figure 2.2 shows the results of the known transformation registrations. The proposed method achieves a better fit in these situations as compared to the feature-based method.

Figure 2.2: Result of the direct method registration (a) on a warped image (b).

Finally, Mainak Sen et al. [13] developed an innovative method for representing and exploring the hardware design space when mapping image registration algorithms onto configurable hardware. They designed a rigid image registration application under real-time performance constraints. Their workflow is quite similar to the one that will be presented in this thesis, with a major difference being the measure of match they chose, which is the Mutual Information. They achieved interesting results with a maximum PL operating Frequency of 74Mhz. Figure 2.3 depicts their architecture's processing flow, which is practically the same as the one that will be presented later in Figure 3.1.

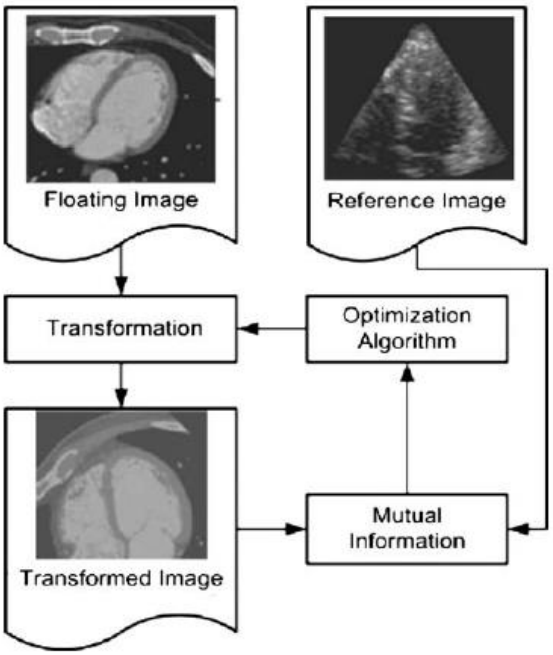


Figure 2.3: Mutual-Information-based image registration

CHAPTER 3: THEORETICAL BACKGROUND

3.1 THEORY ON IMAGE REGISTRATION

So far the idea of image registration has been presented, as well as its range of applications. A key element in fully understanding what it deals with is its practical distinction to a pattern recognition algorithm. The latter's goal is to identify certain characteristics within an image and recognize objects, patterns or regularities in data, which is why it is most commonly referred to as a branch of machine learning. In image registration the problem is not that of a classification as is the case with pattern recognition, but more like a comparison between two (or more images) to determine their differences. For example, in medical applications registration can be used to detect changes over time by comparing a new image to older images or to monitor tumors and other anomalies. In security applications, a new image of an iris of the eye can be compared to a dataset of valid images to determine whether the person can have access to a building or not. Since the range of applications to which image registration can be applied is vast, multiple methods have been studied and used, each designed to be optimal in a different case. These different methodologies concern the choice between different combinations of the major components that will be presented later and each combination is designed to meet specific requirements, such as accuracy, noise tolerance, processing speed and image quality.

Image registration algorithms can be divided in two categories: intensity based and feature based. As already described, one of the images will be the moving or the source, while the others will be the fixed or the targets. The reference frame in the fixed images is stationary, and the moving image is spatially transformed to match the target. Intensity-based methods compare intensity patterns in images via correlation metrics, while feature-based methods find correspondence between image features such as points, lines, and contours.

3.2 HIGH LEVEL PRESENTATION OF THE ALGORITHM

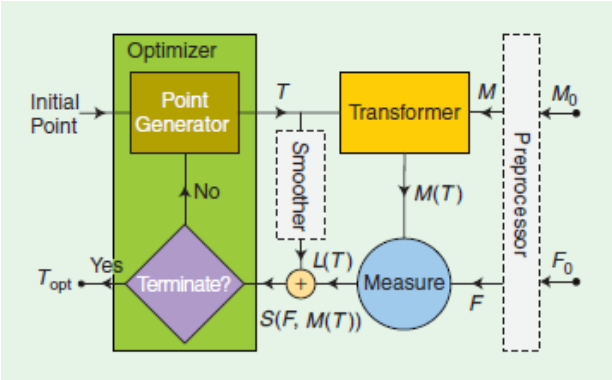


Figure 3.1: Registration Solver's Architecture [6].

Figure 3.1 shows various components of a general registration solver, with the main components being a transformer, a measure, and an optimizer. A measure of similarity or distance is computed between the images at each step and used to determine if they are "sufficiently" aligned. This process is controlled by the optimizer that starts from an initial guess and determines subsequent steps to reach an optimal alignment. Below there is a brief presentation of each of these three main components.

Measure: There are several methods of measuring the similarity of the two images. Based on what measure of match is used, the algorithm is distinguished as feature-based or intensity-based. The two methods that were examined in this thesis are the cross-correlation coefficient and the Matte's mutual information. Both these methods measure similarity, which means that two identical images will have a measure of match equal to one. That will decrease as the images move farther away. Correlation as a term refers to how close two variables are to having a linear relationship with each other. The correlation coefficient is calculated by dividing the covariance of the two variables by the product of their standard deviations. Let $\rho_{x,y}$ be the correlation between the two variables x and y, σ_x, σ_y their deviations and μ_x, μ_y their expected values. With the following equation the correlation coefficient $\rho_{x,y}$ can be obtained:

$$\rho_{X,Y} = corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

A more useful equation to calculate the correlation between the two images is the following:

$$r_{X,Y} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{n s'_x s'_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \quad \text{(Eq. 1)}$$

The x_i, y_i refer to the corresponding pixel values of the image X and Y respectively and n is the total number of the valid pixels in the images (which depends on the resolution of the image). Valid refers to whether or not the transformation generates a pixel, whose coordinates are within the accepted boundaries of the image coordinate system. In the algorithm the power of 2 of the said coefficient is used as a measure of match, to eliminate potential negative values.

Matte's Mutual Information: This method measures the mutual dependence between two variables, according to the probabilistic theory. The equation from which the mutual information is derived is the following, where p(x,y) is the joint probability of the variables X and Y, and p(x), p(y) the marginal probability distribution functions of X and Y respectively.

$$I(X,Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

The probability distributions are obtained using the Kernel Density Estimation (KDE), also known as Parzen histogram method in signal processing. It is easily understood that if the two variables are independent (or in the case of images they are very different) then p(x, y) = p(x) p(y) and log1=0, meaning that the mutual information will return a similarity of 0, as it should.

There are also other methods for calculating the similarity between the two images, like the sum of absolute differences, the sum of squared differences and the gradient correlation. The first two are rather weak methods and almost never produce a practical or realistic estimation of the similarity of the images. The third one was not evaluated in the context of this thesis and therefore shall not be examined.

Transformer: The transformer maps points in the moving image to new locations in the transformed image. Based on the requirements of the registration problem the transformer can either be collinear or deformable. The collinear transformation is defined by a 2x2 matrix for grayscale images. Examples of collinear transformations include rigid, affine and projection. The rigid transformation includes translation and rotation only, whereas the affine transformation also includes scaling and shear. The equation of the rigid transformation is described as below:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

, where x', y' are the coordinates of the transformed point, x and y are the initial coordinates of the moving image, and the rest are the parameters of the transformation. More specifically b_1 and b_2 are the parameters that represent the displacement of the transformation and θ represents the rotation. The method that was used in this thesis is the affine, which describes a transformation around the center of the image. The equation that described it is the following:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x - mw \\ y - mh \end{bmatrix} + \begin{bmatrix} b_1 + mw \\ b_2 + mh \end{bmatrix}$$

, where

- mw is the Width divided by two (midWidth) and mh the Height divided by two (midHeight).
- a_{11}, a_{22} describe the scaling and a_{12}, a_{21} the rotation.

In the case of our dataset the mw and mh parameters are constant and equal to 500. As can immediately be observed the complexity of the affine transformation is the same as that of the rigid transformation. Another important thing to note is the need to use an interpolation after the pixel's initial coordinates have been transformed. As is expected since the parameters are not integer numbers, the x' and y' will also not be integer numbers. Therefore, in order to determine which one of the neighboring pixels should be chosen as the final pixel, an interpolation of a neighborhood of 4 pixels is being used.

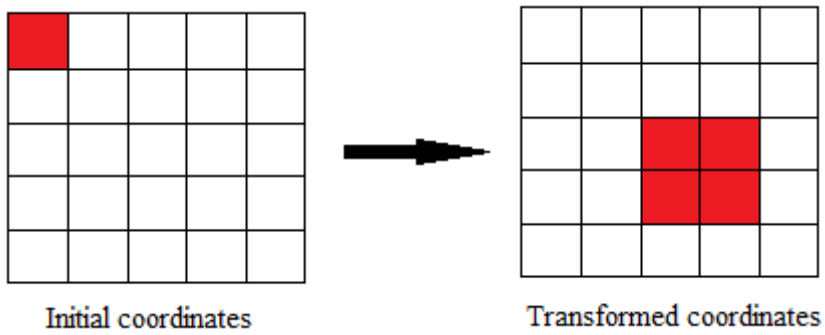


Figure 3.2: Interpolation example

In Figure 3.2 an example of this transformation is demonstrated. The pixel with coordinates (0, 0) is transformed to non-integer coordinates somewhere within the red boundaries in the right image. With the interpolation a different weight is calculated

for each of the red-filled pixels, which is then multiplied by the corresponding pixel's value (which is in the range 0-255 for grayscale images) and the four different products are then added to produce the final value of the wanted transformed pixel. Of course, what was just described will only occur if the transformed coordinates are within the accepted boundaries of the image. In that case, the required pixel will be loaded from the memory and propelled to the measure of match calculator.

Other transformation models are, as already mentioned, the deformable ones. These are used in registration applications, where the deformations present on the data sets are elastic, which is met when a change in the shape of a material is caused at low stress. This change is recoverable after the stress is removed. When such deformations take place, rigid models are not sufficient to describe the required transformation function and lead to misalignments around the elastically deformed regions of the image. This transformation is much more complex than the rigid model, provides more accurate results, but is also much more computationally expensive and requires a much larger execution time. Sometimes a pre-processing stage using a rigid-based registration scheme is used before the elastic transformation in order to partially align the images and greatly reduce the execution time.

Optimizer: The optimizer is responsible for applying an efficient and often non-exhaustive strategy to search the allowed transformation space for the best match between the images. According to the mathematical approach of the selected strategy, an optimizer can be categorized as gradient-based or gradient-free and global or local.

Gradient-based methods try to find the minimum value of a cost function through constant computation of its partial derivatives, in addition to the computation of the value of the cost function itself. Numerical estimations are used for the computation of the derivatives with use of finite differences. Gradient-based optimizers require fewer iterations to converge to the optimal parameters, but at the cost of computationally heavier iterations. The convergence rate depends on the size of the parameter space, the initial misalignment of the images and the termination criteria.

Local methods greatly depend on the selection of the initial point in the parameter space. They may converge to a misalignment if not properly initialized, as they try to find a local optimum within the area of the current point. Global methods on the other hand are given a specific acceptable range of parameters and find a global optimum. They can be more robust than the local methods, but converge a lot slower to the optimal point. These two methods are often combined to increase robustness and convergence rate.

In this thesis the two optimizing methods that were used are the Downhill Simplex, which is often used in multidimensional problems, and the Powell, both of which are classified as gradient-free and local. A brief presentation of each method is given below:

Downhill Simplex: This method requires only function evaluations, not derivatives. It is not very efficient in terms of the number of function evaluations that it requires. A simplex is the geometrical figure consisting, in N dimensions, of N + 1 points (or vertices) and all their interconnecting line segments, polygonal faces, etc. In two dimensions, a simplex is a triangle. In three dimensions it is a tetrahedron, not necessarily the regular tetrahedron. In general we are only interested in simplexes that are nondegenerate, i.e., that enclose a finite inner *N*-dimensional volume. If any point of a nondegenerate simplex is taken as the origin, then the *N* other points define vector directions that span the *N*-dimensional vector space. The downhill simplex starts with N+1 initial points (which in our case means 7 points, since the number of parameters is 6) , defining an initial simplex. If one of these points is P_0 then the other N points can be taken as:

$$\mathbf{P}i = \mathbf{P}_0 + \lambda \mathbf{e}i$$

where λ is a constant describing the problem's characteristic length scale and **e***i* N-vectors. After the initialization the downhill simplex takes a series of steps, trying to reshape the simplex in such a manner, so as to move its highest point to a lower point. These steps can either be reflections away from the high point, or reflections combined with expansions away from the high point, or contractions along one dimension from the high point or multiple contractions towards the low point. These actions are illustrated in Figure 3.3. An appropriate sequence of such steps will always converge to a minimum of the function.

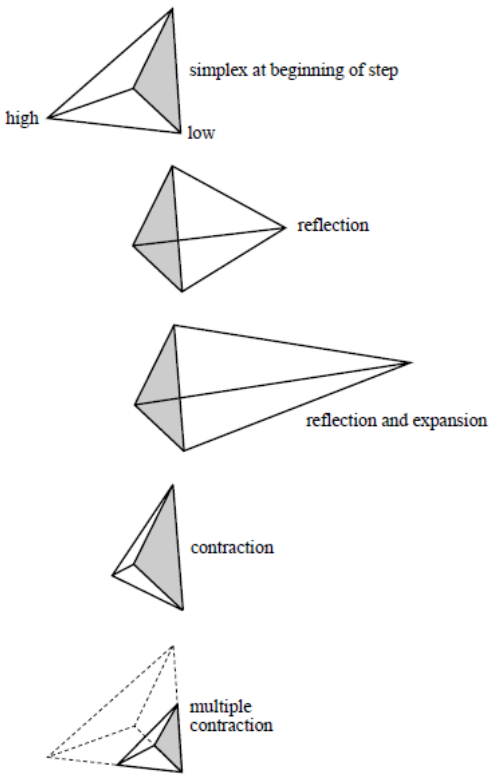


Figure 3.3: Possible Downhill Simplex Steps [8].

The termination criterion recognizes the convergence by measuring the distance the vector moved between two steps and comparing that to a constant tolerance. If the step distance is smaller in magnitude than that tolerance then the program terminates. Another criterion requires that the decrease in the function's value in that step is smaller than another tolerance "ftol".

Direction Set Powell's Method: Powell introduced a direction set method that produces N mutually conjugate directions. This method practically searches for a multidimensional function's minimum or maximum by searching along set directions. The problem lies in choosing these directions and quickly finding the best point along that direction. The method starts by initializing a set of directions u_i to the basis vectors e_i.

$$\mathbf{u}_i = \mathbf{e}_i,\ i = 1, \ldots, N$$

Now the following sequence of steps is repeated until the function stops decreasing:

- Save the starting position as **P**0.
- For $i = 1, \ldots, N$, move $\mathbf{P}i-1$ to the minimum along direction $\mathbf{u}i$ and call this point $\mathbf{P}i$.
- For $i = 1, \ldots, N - 1$, set $\mathbf{u}i \leftarrow \mathbf{u}i+1$.
- Set $\mathbf{u}_N \leftarrow \mathbf{P}_N - \mathbf{P}_0$.
- Move $\mathbf{P}_N$ to the minimum along direction $\mathbf{u}_N$ and call this point $\mathbf{P}_0$.

To find the minimum along a direction a procedure called linmin is used, whose definition is: Given as input the vectors **P** and **n**, and the function *f*, find the scalar λ that minimizes $f(\mathbf{P}+\lambda\mathbf{n})$. Replace **P** by $\mathbf{P} + \lambda\mathbf{n}$. Replace **n** by $\lambda\mathbf{n}$. This procedure is used by many multidimensional direction set minimization methods. In Powell's method this procedure does not use calculation of gradients. Figure 3.4 presents the search of a minimum along a direction.

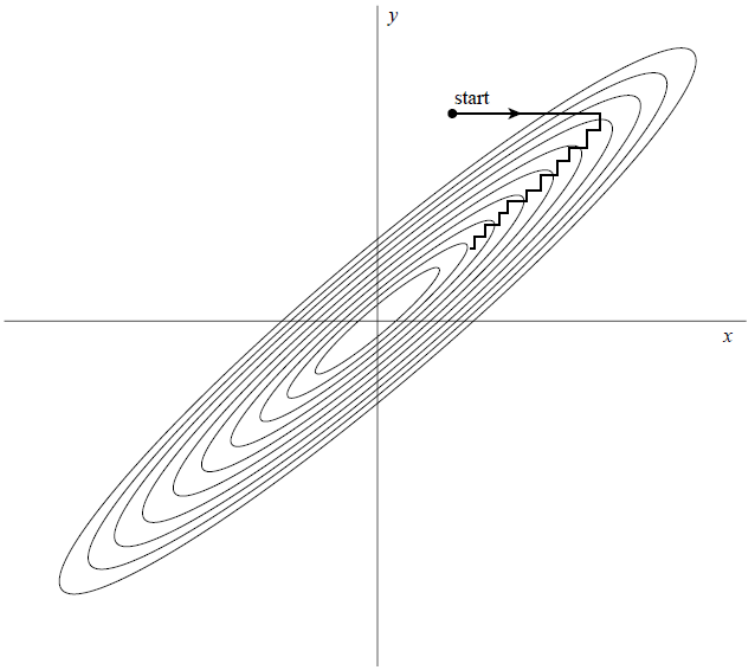


Figure 3.4: Powell's method minimum search across a direction [8].

So far the major components of the registration solver have been described. The 4 different pairs that were examined in this thesis are the following:

- Affine transformation/Correlation/Downhill Simplex
- Affine transformation/Mutual Information/Downhill Simplex
- Affine transformation/Correlation/Powell's
- Affine transformation/Mutual Information/Powell's

Another obvious observation that needs to be pointed out is that the registration will not always work. When the two images are of a different eye, then there is no possible transformation, capable of aligning the two images. The solver will just keep trying to find one until either of the termination criteria is satisfied, which either means satisfaction of tolerance or maximum number of iterations exceeded. There is also the possibility that images of the same eye cannot be aligned using the affine transformation model, either because of a very large initial misalignment or because of the presence of elastic deformations. Finally, there is the possibility of two images being successfully aligned, despite having a relatively small final measure of match.

There are also a few parameters that affect the performance of the solver. First of all the termination tolerance can have an effect on how correct and realistic the found solution is. Too small and the solver will terminate prematurely, most probably failing to align, even if the two images could be aligned. Secondly, the number of iterations can also have an effect, since again too few iterations may not be sufficient for the optimizer to approach the optimum solution, and too many may be unneces-

sary and pointlessly increase the computation time. The typically used values for these parameters that always perform sufficiently for this dataset are 10^{-5} for the tolerance and 500 for the maximum number of iterations.

Rules regarding the allowed transformation parameters also need to be considered. Limitations need to be set to these parameters on the maximum value they can reach. If the displacement is for example 600, the optimizer may find a higher correlation, although the measure of match can be unrealistic. In all of the images in the dataset, it was noticed that no more than 300 pixels of displacement are required. That means that if the optimizer goes for an even higher displacement than that, most of the transformed image would go off-bounds and the similarity measure will take into account only a small isolated part of the two images. These two parts may be quite similar and give a high measure of match, meaning that the optimizer will think it is moving in the right direction, while the registration will be failing even though with proper handling it could succeed. For the same reason maximum limitations need to be applied on the rotation and scaling. An exploration was conducted regarding the optimal use of these maximum values, the results of which are presented below. As the maximum rotation and scaling don't affect the results as much and their typical values of 30º and 10% accordingly are sufficient, Tables 3.5 and 3.6 concern the maximum displacement and how it affects the final measure of match (correlation coefficient or mutual information) between the fixed image and the optimally transformed one.

Table 3.5: Final Measure of Match for a good pair of eyes

Maximum Displacement	Simplex Correlation	Simplex Mutual	Powell Correlation	Powell Mutual
50	0,157844	0,261021	0,227008	0,285423
100	0,231213	0,374724	0,487449	0,385665
200	0,783975	0,788725	0,784077	0,788861
300	0,784035	0,789102	0,784041	0,789113

Table 3.6: Final Measure of match for a pair of eyes that can be aligned, but with a low measure of match

Maximum Displacement	Simplex Correlation	Simplex Mutual	Powell Correlation	Powell Mutual
50	0,205226	0,23432	0,271067	0,255322
100	0,425701	0,264828	0,463098	0,266285
200	0,447005	0,468757	0,463086	0,468897
300	0,463059	0,468894	0,463074	0,468902

What is also important to note here is a comparison between the results generated by the mutual information measure of match and that of the correlation, as well as the corresponding execution time measurements. Table 3.7 illustrates an example of the mentioned values, using the downhill simplex optimizer.

Table 3.7: Comparison between correlation and mutual information, where T stands for the six transformation parameters (T3 and T6 are the two displacements, T2 and T4 the rotations, and T1 and T5 the scaling).

	First-pair correlation	First-pair mutual	Second-pair correlation	Second-pair mutual
T1	1.003900	1.010139	1.008103	1.010794
T2	-0.016374	0.017403	-0.073785	-0.073385
T3	177.098624	178.385146	-19.983910	-18.699687
T4	0.026220	0.027185	0.082441	0.081337
T5	1.000249	1.001831	1.006750	1.008789
T6	-62.307664	-62.081498	-97.090582	-96.832628
Measure of Match	0.783880	0.783464	0.463121	0.461139
Total Time(s)	7.750371	12.823528	22.896096	18.051921

What was concluded from the measurements is that both methods achieve similar results when the maximum parameters are chosen wisely. The execution times for each can also vary, and since the mutual information needs to call the correlation function once at the end of the registration to finalize and normalize the results, the method that was chosen to be implemented was that of the correlation. The first pair of eyes in Table 2.7 refers to a well aligned set of images, where the registration succeeds and the optimizer terminates fast, needing a small number of iterations, as it converges fast to the optimum solution. As a result, since the optimizer is stable for aligned images, the results will be very close to each other, whatever measure of match is chosen and whatever precision is used (double or single). The second pair on the other hand, although the images are of the same eye and can be aligned, has a low measure of match, meaning that it is not enough to determine whether the registration was successful or not, rendering the edge detection and fusion steps necessary at the end of the processing flow. The optimizer in this specific example required a larger amount of iterations to converge to the optimum solution, a feature which is reflected in the execution time required. Visual representation of the result ensures that the registration was successful, despite the low measure of match.

If the images contain elastic deformations, the solver may be unable to find a transformation capable of aligning the two images. Some misaligned images can lead to the optimizer producing very different transformations if the precision is not strict enough, because of the instability of the registration in such cases.

Table 3.8 indicates the behavior of the solver for the above image sets, if the chosen optimizer is the Powell's Method. As is observed, the execution time is greatly increased and the produced parameters are almost the same, rendering the use of the Powell's Method unnecessary in these examples. In the given dataset for this thesis, no cases in which Simplex failed and Powell succeeded were found.

Table 3.8: Comparison between correlation and mutual information using Powell's Method as the optimizer

	First-pair correlation	First-pair mutual	Second-pair correlation	Second-pair mutual
T1	1.003894	1.010139	1.008103	1.010722
T2	-0.016374	-0.017403	-0.073769	-0.073385
T3	177.089815	178.377445	-19.983500	-18.704472
T4	0.026516	0.027185	0.082441	0.081360
T5	1.000425	1.001831	1.006750	1.008632
T6	-62.307664	-62.081499	-97.090619	-96.832628
Measure of Match	0.783901	0.789116	0.463121	0.468892
Total Time(s)	34.360532	22.546205	31.134632	36.850267

CHAPTER 4: APPLICATION/SYSTEM DESCRIPTION

4.1 FPGAs

Field Programmable Gate arrays (FPGAs) are semiconductor devices consisting of reprogrammable hardware. They are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. These interconnects are programmed using a Hardware Description Language (HDL), like Verilog and VHDL in order to match an application's desired functionality. Since each logic block can be computationally independent from the other ones, high levels of parallelism are enabled, capable of greatly speeding up the computation time of a program. This is one of the main advantages of the FPGA compared to conventional chips like a CPU, which are static and designed to be as efficient and fast as possible, but are limited to executing one command at a time. If the design has errors or needs to be updated, or even if one wants to execute a different application with the FPGA, all that needs to be done is reprogram it, which comes in contrast with the Application Specific Integrated Circuit (ASIC) idea. ASICs need to be very carefully designed and simulated before manufactured, since an unnoticed bug would mean that it has to be re-manufactured, greatly increasing the cost. A figure is shown below, presenting the abstract architecture of an FPGA as described above.

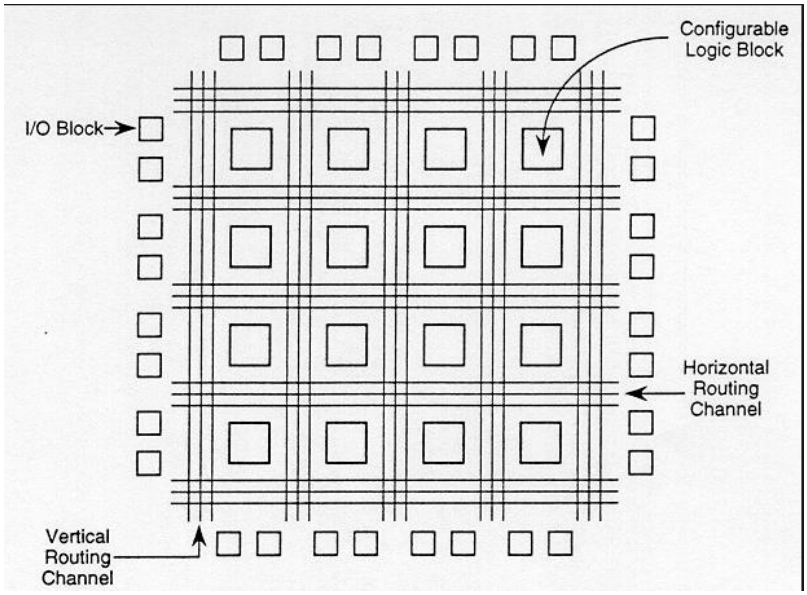


Figure 4.1: FPGA's abstract architecture

The configurable logic blocks consist of a large number of logic units, such as flip flops, multiplexers and Look-Up Tables (LUTs). Each FPGA has a set amount of instances of each of these units, which varies from model to model. An *n*-bit LUT can encode any *n*-input Boolean function by storing the truth table of the function inside it. This is an efficient way of encoding Boolean logic functions. Aside from these units, the FPGA also contains a number of memory blocks called Block Rams (BRAMs), which are small in size. Finally, recent FPGAs also contain a number of Digital Signal Processors (DSPs) which are extremely efficient in many digital signal processing applications, where accumulations and multiplications are repeated, and help speed up many processes due to the higher frequency in which they can operate. Usual cases where DSPs prove very useful are in the implementation of an FIR-filter, floating point mathematical operations and many more.

4.2 SoC FPGAs

In the past, the FPGAs would not be integrated on the same chip as the Processor. As a result, the off-chip communication between them was hard and inefficient, due to the limited I/O bandwidth. In 2010 Xilinx brought the Zynq-7000 all programmable SoC family devices to the market, which integrates the software programmability of a dual-core ARM Cortex-A9 processor with the hardware programmability of a 28nm Artix-7 or Kintex-7 based programmable logic , enabling key analytics and hardware acceleration while integrating CPU, DSP, memory and multiple peripherals on a single device (Figure 4.2). Since the ARM processor is capable of supporting full operating systems, (the most frequently used of which is Linux, since it is open source and has an established community support) the programmer is allowed to develop the application using hardware/software cooperation, which is ideal for embedded system designs, since the on-chip connectivity allows for high bandwidth, and low power consumption and latency. Xilinx uses the AMBA AXI protocol for communication between the Programmable Logic and the Processing System.

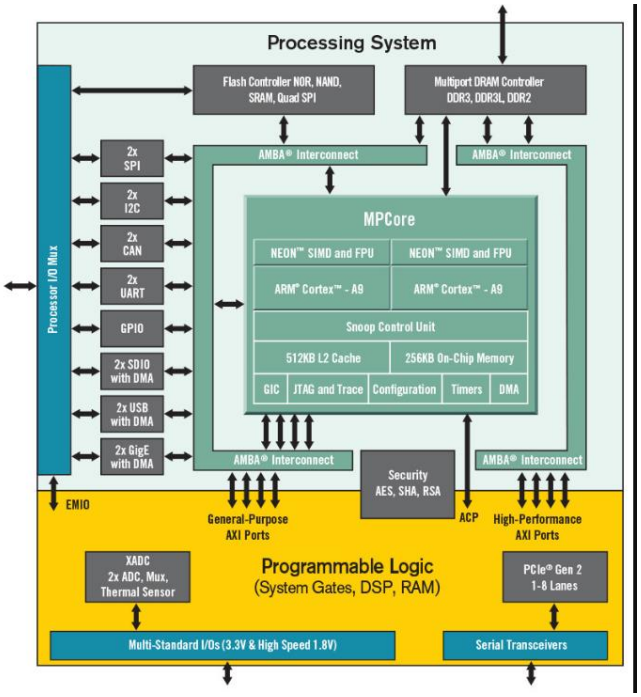


Figure 4.2: Zynq-7000 devices' architecture [4].

The AXI protocol: AXI (Advanced eXtensible Interface) is part of the ARM AMBA, a family of micro controller buses. AMBA is an open standard for the connection and management of functional blocks in a System-on-Chip. It facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals. AXI4 provides improvements and enhancements, benefiting Productivity, Flexibility, and Availability.

More specifically, the AXI protocol:

- is suitable for high-bandwidth and low-latency designs
- provides high-frequency operation without using complex bridges
- meets the interface requirements of a wide range of components
- is suitable for memory controllers with high initial access latency
- provides flexibility in the implementation of interconnect architectures
- is backward-compatible with existing AHB and APB interfaces.

The key features of the AXI protocol are:

- separate address/control and data phases
- support for unaligned data transfers, using byte strobes
- uses burst-based transactions with only the start address issued
- separate read and write data channels, that can provide low-cost Direct Memory Access (DMA)
- support for issuing multiple outstanding addresses

- support for out-of-order transaction completion
- Permits easy addition of register stages to provide timing closure.

There are three types of AXI4 interfaces:

- AXI4 (or AXI4-full) —for high-performance, bidirectional, memory-mapped requirements. Allows bursts of 256 data/ address.
- AXI4-Lite—for simple, low-throughput memory-mapped communication (for example, to and from control and status registers). Practically a light variant of AXI4-full but with bursts of 1 data/address.
- AXI4-Stream—for high-speed streaming data. Provides unlimited, unidirectional (from master to slave only) data bursts.

The protocol allows several AXI masters and slaves to be connected using a structure called an Interconnect block. AXI4-full and AXI4-lite interfaces consist of the following 5 channels, which can also be observed in Figure 4.3.

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

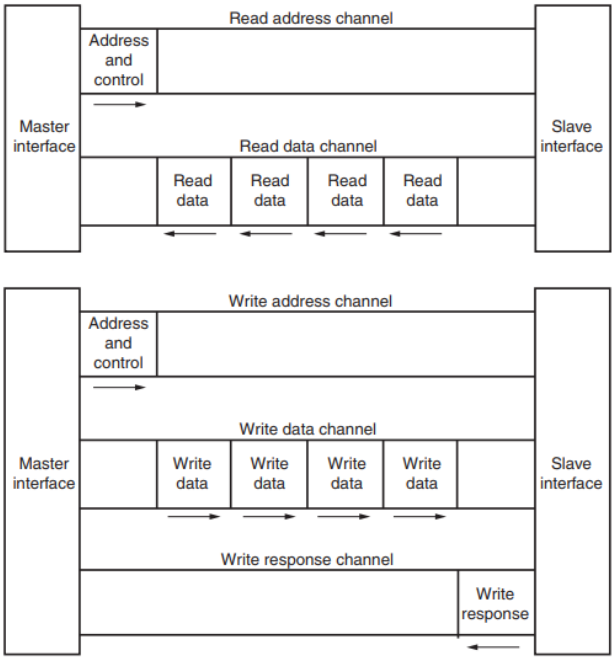


Figure 4.3: AXI4-channels architecture [5].

The top figure illustrates how a Read transaction takes place using the Read address and Read data channels. The bottom figure illustrates how a Write transaction uses the Write address, Write data and Write response channels. The Write Response practically informs the master that the write operation is successfully completed or in action and greatly helps in the communication and synchronization of certain operations.

Zybo Development Board

The SoC FPGA that was used in this thesis for the design of the registration solver in an embedded system was the Zybo Development Board, manufactured by Digilent using the smallest member of the Xilinx Zynq-7000 family, the Z-7010. The Z-7010 is based on the Xilinx System-on-Chip (SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series field programmable gate array (FPGA) logic. Just like most of the other Zynq-7000 family devices, the FPGA is fabricated in the logic of the Artix-7 Programmable logic. Below is a list of the device's available resources. This information is highly important to note out, for reasons that will be explained later.

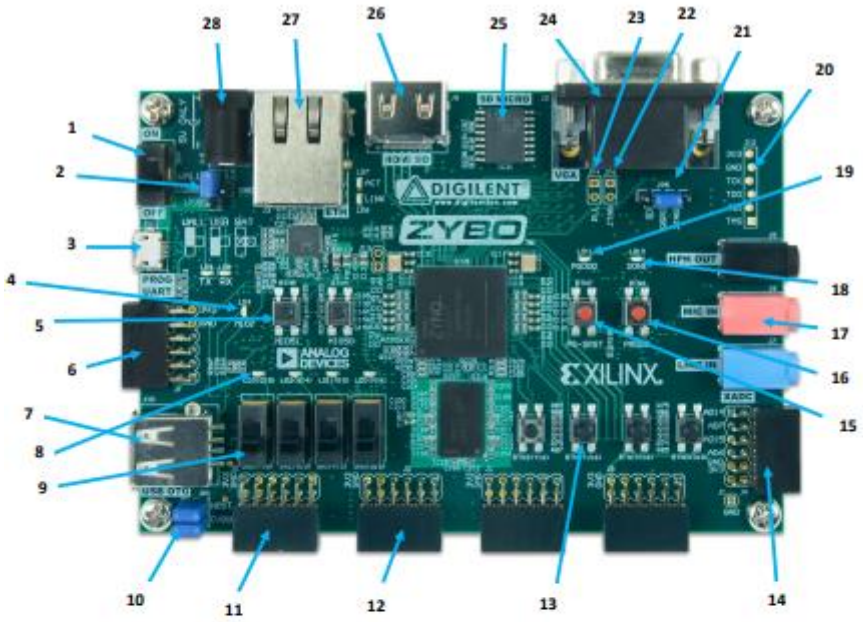
- Look-up Tables(LUTs): 17600
- Flip Flops: 35200
- Block Ram (# 36kb Blocks): 2.1Mb (60)
- DSP slices: 80

The board's features are listed below:

- 650Mhz dual-core Cortex-A9 processor
- DDR3 memory controller with 8 DMA channels
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
- 512MB x32 DDR3 w/ 1050Mbps bandwidth
- Dual-role (Source/Sink) HDMI port
- 16-bits per pixel VGA source port
- Trimode (1Gbit/100Mbit/10Mbit) Ethernet PHY
- MicroSD slot (supports Linux file system)
- OTG USB 2.0 PHY (supports host and device)
- External EEPROM (programmed with 48-bit globally unique EUI-48/64™ compatible identifier)
- Audio codec with headphone out, microphone and line in jacks
- 128Mb Serial Flash w/ QSPI interface
- On-board JTAG programming and UART to USB converter
- GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs

- Six Pmod ports (1 processor-dedicated, 1 dual analog/digital, 3 high-speed differential, 1 logic-dedicated)

In Figure 4.4 the board is presented with all the peripherals attached to it, as well as its functional features.



Callout	Component Description	Callout	Component Description
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

Figure 4.4: Zybo Peripherals and components

4.3 PROCESSING FLOW

Up until now, the theory behind image registration has been analyzed. Now the processing flow in a programming level will be presented. Figure 4.5 illustrates this flow, which looks like a simplified version of the Figure 3.1.

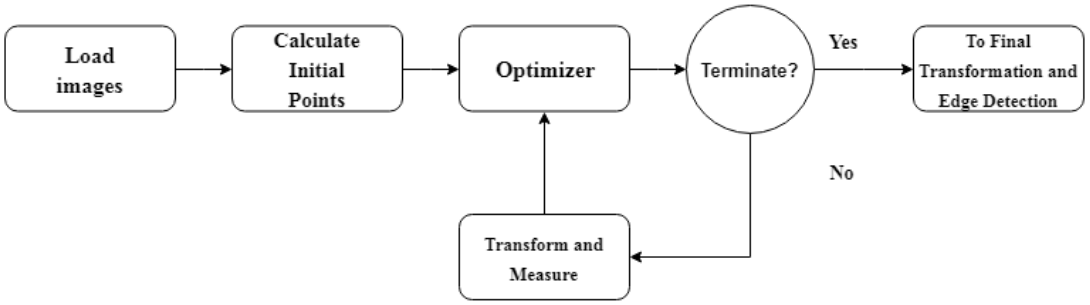


Figure 4.5: Processing Flow of the Registration Solver

In the first part the images from the dataset need to be loaded in the program. This can be done in various ways. Using the OpenCV library is one way, while reading an image from a text file is another. In a more practical environment where the solver will be applied for medical requirements, there will also be a stage before for the images to be captured by a camera.

Next, in order for the downhill simplex algorithm to begin, N+1 initial points need to be calculated. That's what is being done in the second stage. A set of initial parameters that cover the range of the exploration space are created, and for each of them a transformation is applied and the match is measured between the transformed images and the fixed image. Since a simplex has been created, the optimizer is ready to begin and a series of the previously presented steps regarding reflection, expansion etc. can be initiated.

The optimizer for the Downhill Simplex Method calls some functions to search the exploration space. One of the functions is related with optimizing the current state of the parameters, while the other is used to measure the matching between the fixed image and the transformed moving image, if the given transformation was to be applied, without actually generating the transformed image as a separate file and binding unnecessary memory. The calculated measure of match calculated by the aforementioned function is then used to decide what the next step of the method should be.

The 4^{th} stage, called Transform and Measure is the one that was described above. The optimizer will first check if either of the two termination criteria is satis-

fied, before seeking for the next transformation matching measurement. These criteria are the maximum number of iterations and the tolerance threshold.

The final stage is an off-line stage of the procedure and refers to the visual representation of the registration's results. Because the goal of this thesis was to speed up the registration algorithm, this part of the flow was left out of the final implementation for simplicity reasons only. If the optimum calculated parameters of the transformation are the same before and after the co-design, it is indicated that the implementation was successful. Future extensions with a more practical application could include this final low time-consuming stage in the software part of the system.

Figures 4.6 and 4.7 illustrate a successful and an unsuccessful registration example accordingly. The affine transformation with the downhill Simplex method was used to match two images of the same eye. The figures illustrate a fusion between the fixed image and the detected edges of the optimally transformed moving image.

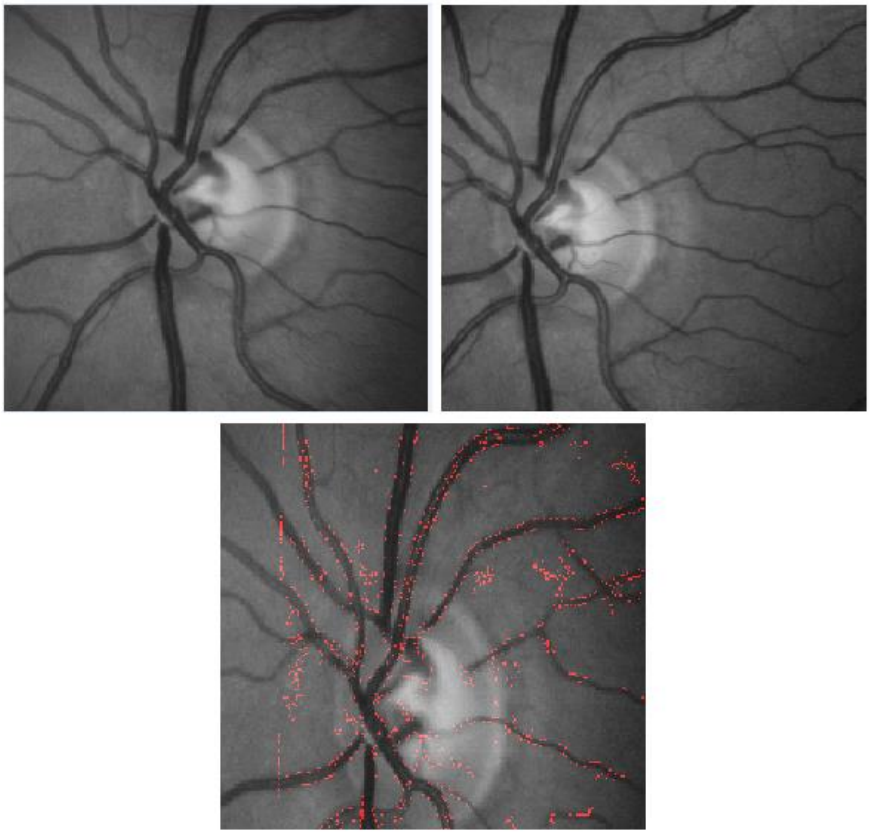


Figure 4.6: Affine Simplex Successful Registration Example

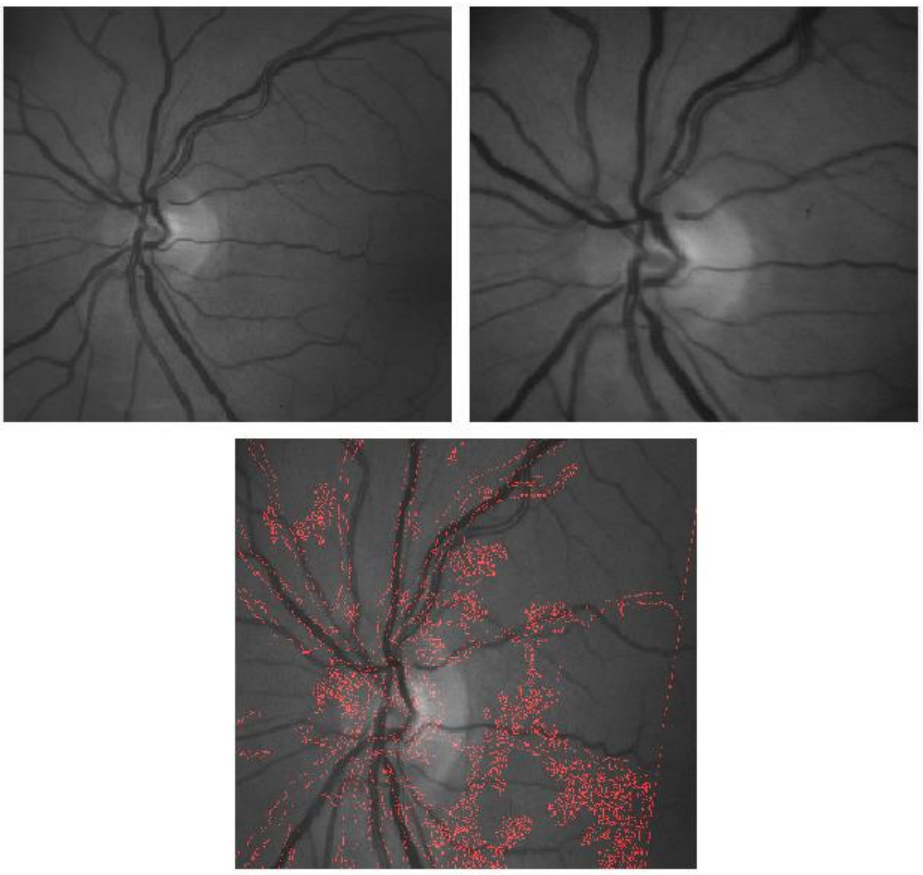


Figure 4.7: Affine Simplex Unsuccessful Registration Example

4.4 APPLICATION PROFILING IN ZYBO

The first matter that is taken in consideration when wanting to optimize an application's design is measuring the execution time in the platform of interest. The total execution time for the registration problem depends on a few parameters:

- **Size of images:** The bigger the image, the more the pixels that need to be transformed and measured, and as an obvious result the execution time is linearly dependent on the image size.
- **Maximum number of iterations:** If the tolerance threshold is not surpassed, the optimizer will keep searching for an optimum solution until the threshold is reached. This may never happen if the two images are of a different eye or the initial misalignment is too extended, meaning that the optimizer could potentially run forever. That's why a typical number of 500 maximum iterations are used. Less than that may negatively affect the optimization, while more than that is most often redundant and needlessly increases execution time.

- **Tolerance threshold:** Much like the maximum number of iterations, a large value of threshold would result in a premature termination of the program, with no satisfying matching achieved, whereas a rather small one would lead the optimizer to perpetually try and achieve something that is impossible.
- **Maximum Displacement, Rotation and Scaling:** Since the initialization parameters depend on these values, and certain transformations can be rejected based on whether the boundaries are respected, these parameters also have a direct effect on the execution time. After extensive exploration of the maximum values space and consideration of the nature of the given dataset, the typical numbers that were used were 200 pixels, 0.5 rad (~30º), 1.1(10% scaling) respectively.

Aside from measuring the total execution time, a complete profiling of the application was conducted to measure the time required by each of the different components. Table 4.7 represents the time (along with the percentages) of the transformation and measuring component on the Zybo. The initial point calculation is practically a call of the "Transform and Measure" component N+1 (which means 7) times, so the two corresponding times are merged. Because the final transformation and edge detection was not part of the on-line procedure in this thesis, their execution times are ignored (but nonetheless they are not very time consuming components). As is evident, the transformation and measure component is the most time consuming in all cases.

Table 4.8: Time profiling on Zybo for a good and a bad pair

	Good Pair	Bad Pair
Total Time (s)	39,923572	103,084555
Transform and Measure time (s)	39,921616	103,080554
Percentage	99,99%	99,99%

Bad pairs sometimes require a lot more time until the registration is completed, because the optimizer will make use of the maximum number of allowed iterations before it terminates. The same behavior is noticed when using either of the two optimizers or either of the two measuring methods. A quick comparison with the execution time required in a desktop PC with a higher frequency processor, as shown in Table 3.7 is that the registration requires 4-5 times more execution time on the Zybo.

The timing measurements indicate that the "Transform and Measure" component is the one that will be implemented on the hardware. The image loading and the optimizer are going to be implemented on the software. Whenever the optimizer requires the calculation of the measure of match for the transformed image, the proces-

sor will communicate with the FPGA and wait for its processing to complete. This procedure will be repeated until the optimizer terminates. Figure 4.8 represents this architecture, emphasizing on the referred hardware/software partitioning and communication. Finally, it is important to note that once the hardware implementation of the "Transform and Measure" component is completed, both the optimizers can make use of it (i.e. the Downhill Simplex and the Powell's Method).

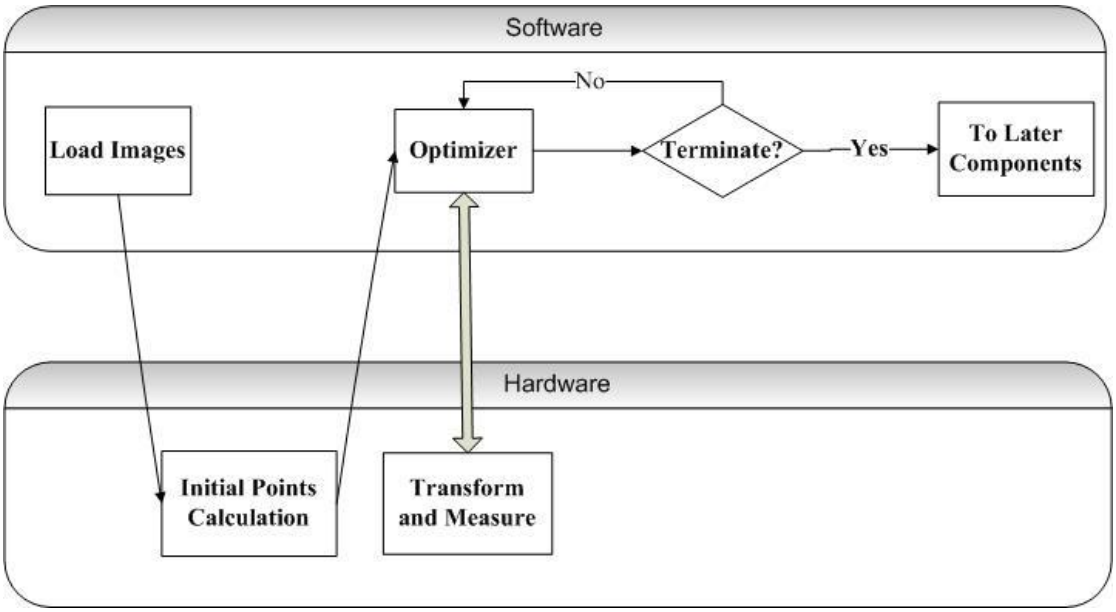


Figure 4.9: Hardware/Software Partitioning

CHAPTER 5: SYSTEM IMPLEMENTATION

5.1 PRESENTATION OF HARDWARE COMPONENTS

The presented registration algorithms make use of floating point arithmetic. Depending on the required precision, different representation sizes can be used, with the most common being the 32-bit and 64-bit (type float and double in programming language C respectively). Although the two sizes produce a different result, the percentage deviance between the two is rather small (less than 0.5 %), meaning that both can be used without any practical error that can disrupt the registration. This small difference in the results is due to possible minor accumulated errors in each iteration (total of one million iterations for the given image size). In order to save resources in the FPGA the 32- bit size was chosen for representing the floating point numbers. Since arithmetic operation circuits for floating point numbers are very complicated, Xilinx provides ready components for most floating point operations (addition, subtraction, multiplication, divider, comparators and more) in the form of Intellectual Property (IP). This IP is completely synchronous and pipelined, meaning that a new input can be given each subsequent clock cycle, and after a given initial latency, the corresponding results will be getting ready in each new clock cycle. The following IP‘s characteristics can be configured by the designer:

- **Operation:** As already mentioned, the IP can be configured to implement many different operations.
- **Latency:** Latency is the number of cycles required for the result of an operation to be ready after the corresponding valid operands are given as inputs to the circuit. Xilinx has set this characteristic to a recommended value, but if required, the latency of these circuits can be reduced at the cost of extra resources and reduced operating frequency. Reducing the latency however is not important in speeding up the registration’s calculations, as the whole process is pipelined and so the processing time will depend on the number of operations needing to be done. For example, if one million operations need to be done repeatedly, then one million clock cycles are required, whereas the initial latency of each IP is going to be less than 20 clock cycles, which is insignificant in comparison. Therefore, the maximum recommended value was chosen for all these components, so as to save as many valuable resources as possible.
- **Operating frequency:** This parameter dictates the maximum frequency in which the circuit can operate. Keeping the latency high is the only way provided by Xilinx to configure the highest possible frequency, which is obviously greatly needed.

- **Resource utilization:** Due to the great amount of operations that need to be executed, keeping track of the resources used is highly important, to ensure that the final-stage implementation is feasible when using the ZYBO board. Reducing the latency leads to more resources used, explaining the reason behind the maximum latencies selection. An interesting feature of this IP is the option to save resources through the usage of DSPs. Making use of the total of 80 DSPs available in the Zybo can greatly reduce the resources, and allows for higher operating frequencies.
- **Precision of inputs and output:** The available precisions were the half precision (16-bit representation of floating-point numbers), single precision (32-bit), double precision (64-bit) and custom precision.

Having analytically presented the hardware used for executing the operations, it is time to describe the overall system of the transformation and measure design in hardware. Four calculation components were designed, one for each different functionality. Each component is connected to the ones that follow. Aside from these, there were also a number of control components, used to determine the flow of data within the programmable logic and synchronize the whole procedure.

- **Transformation:** Applies the affine transformation to each of the pixel's coordinates. Its inputs are the transformation parameters and the pixel coordinates. The outputs are the transformed coordinates (with the decimal part cut off), as well as two values that are to be used in the next stage, which is the interpolation.
- **Calculation of Interpolation weights:** Making use of the two previously referred values, each of the four different interpolation weights required (Figure 2.2) is calculated in this component. All four weights need to be available simultaneously, so proper synchronization with the addition of registers was required. The method of adding registers to delay signals was used in most of the components, as it is a very efficient and easy way to synchronize the design.
- **Interpolated pixel's final value:** The inputs of this component are the previously calculated weights, as well as the four corresponding to each weight pixel values. The pixel values are loaded from the memory, therefore a special control unit takes care of synchronously loading these values from the block rams (BRAMs) where the images are loaded. The output is the pixel value which originates from applying the interpolation on the four neighboring pixels, as described in Figure 2.2.
- **Accumulations:** This is the final calculating component implemented on the hardware. Its inputs are the interpolated pixel's value and the floating image's value, which is synchronously supplied by another controller. The component calculates the sums in Equation 1. Knowing the latency of all the components

combined, it is possible to halt the accumulation after the final pixel's processing, so as to be able to send the valid data to the software, where the correlation is to be quickly calculated.

As mentioned, control units were used for synchronization and some additional functionalities. These control units are described below:

- **Coordinate generator:** This control unit contains a process which creates a new pair of coordinates in each clock cycle and sends it as an input to the transformation block. This generation is linear, meaning that the image will be accessed line by line. It is also responsible for terminating the calculating process when the final coordinate is generated, by setting the validity of the signals to 0, unless a reset signal is activated, in which case the whole procedure will be restarted.
- **Memory manager:** Some of the calculation components need the pixel values as inputs, as already described. This control unit is responsible for this coordination. It takes as inputs the transformed coordinates, loads the required values from the BRAMs and synchronously supplies them to the components that need them. The manager is also responsible for saving the image's pixel values sent from the software in the Block Rams. This only happens once at the start of the registration, and the BRAMs don't need to change, as long as the registration takes place. The way the image is sent to the FPGA from the Host will be described in the next section.
- **Coordinate validity:** Considering that the transformation may produce invalid coordinates that are out of bounds, a controller that makes the necessary comparisons is required, in order to reject them and not include them in the calculations. This controller takes as inputs the transformed coordinates and gives them as outputs with the appropriate validity signals.
- **Registers:** A simple register component with a generic cycle delay and signal width was designed and broadly used in all the components, as well as in between them to effectively synchronize the design.

Figure 5.1 shows how the abovementioned components are interconnected and make up the implementation of the hardware system. The images are streamed once at the beginning of the registration and saved in the Block Rams, using the AXI4 streaming protocol, as will be explained later. Since the division required in Equation 1 is only executed once at the end of the measure, it is not wise to include it in the hardware, as the divider for floating point number takes up a large amount of valuable resources. Generally, the mentality is to not use the FPGA for calculations that do not occur often. Having explained why, the accumulator sends the sum amounts of Equation 1 to the software, where the final division is executed.

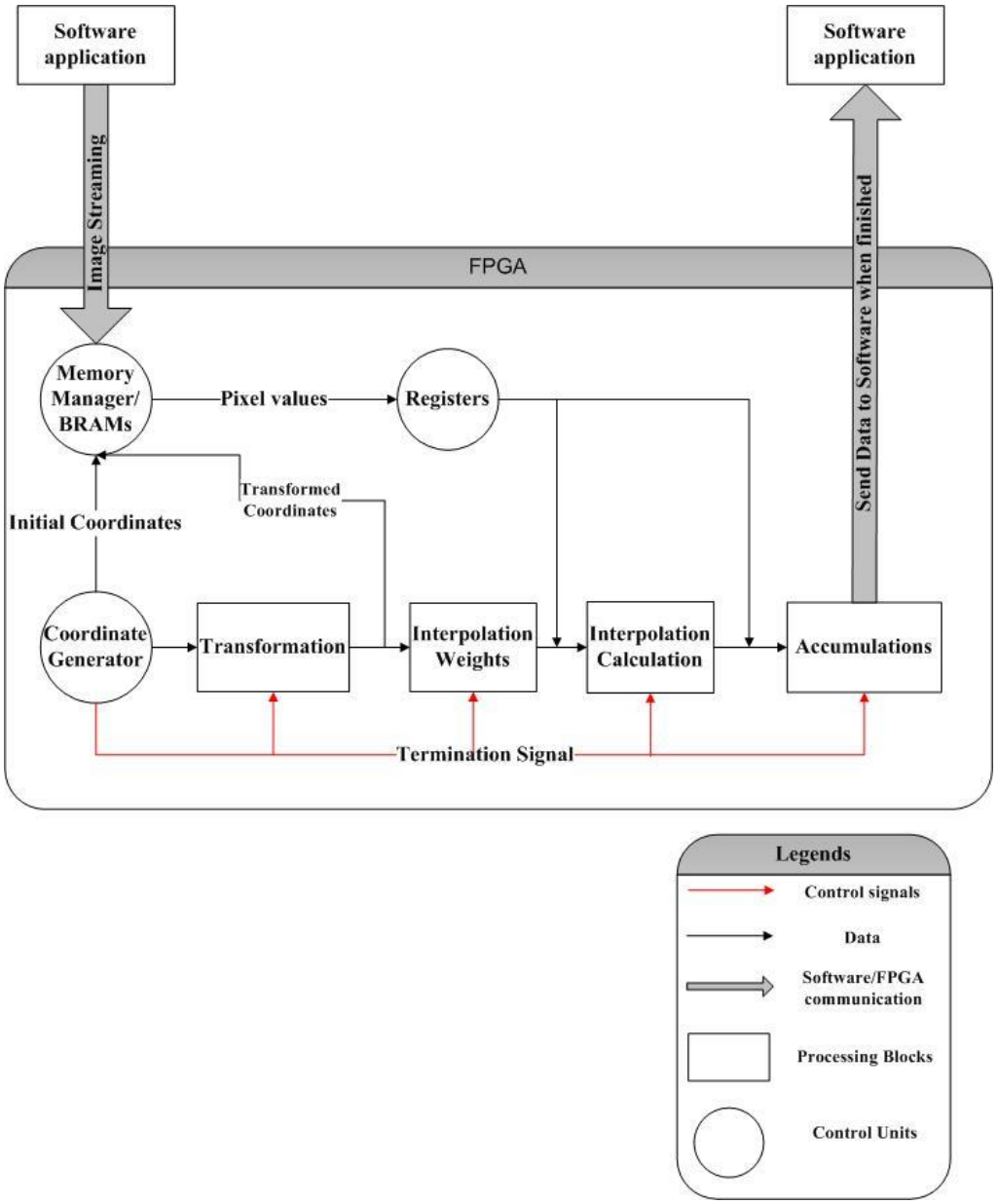


Figure 5.1: Implementation of the Hardware System

5.2 HARDWARE/SOFTWARE COMMUNICATION

Since the BRAM memory size in an FPGA is not large enough to save an entire high definition image (only 256kB total memory in the case of the Zybo), a sufficient data transfer with a high throughput is required in most image processing applications. The AXI4-stream protocol is therefore the most effective choice available, since the CPU handshakes will take place only once at the start and end of the transfer.

The general idea in image processing is to start streaming the first two or three pixel lines, so that the accelerator can begin processing the image's data. The data streamed can be saved in BRAMs, and after the data is used, new lines can start getting streamed, writing over the BRAM, so as not to waste cycles waiting for the data to be transferred. Using a single streaming channel, a new 32-bit data can be transferred each consecutive clock cycle, which is equivalent to four pixels, as a greyscale pixel requires 8 bits for its representation. This efficient communication pattern relies on the Direct Memory Access (DMA) mechanism.

DMA is a very important mechanism used in most embedded systems for the effective communication between the processor, the memory and the peripherals. With the use of a proper controller, DMA links the external devices with the main bus that connects the processor with the memory. Older architectures without a DMA controller required the processor to continuously perform memory access operations, in order to manage the necessary data transfers. This would also mean that if the needed data was not yet ready, the processor would remain occupied for the entire duration of the read or write operation and wait for a corresponding interrupt to let it know the data is finally ready. As a result, the throughput was greatly reduced and valuable computation time of the CPU was being wasted. With the use of a DMA, this problem is solved, since the CPU no longer needs to intervene all the time. It simply initiates the transfer, and then it deals with other operations while the transfer is in progress, until it finally receives an interrupt from the DMA controller when the operation is completed. While it is waiting for data it can do something else, as the DMA controller will be the one to perform the required data transfers from or to the peripherals. Not only is the throughput increased, but a better exploitation of the available architecture and resources is achieved, leading to higher overall performance and lower power consumption.

The DMA controller is used to control an AXI4-Stream protocol in SoC FPGAs image processing applications. However, implementing the entire protocol from scratch on the hardware level and installing the necessary drivers (which includes checking for correct version and kernel module insertions, a rather time consuming activity) on the operating system level is a highly complex procedure. As a result simpler protocols are often used. In the case of this thesis, in order to simplify the communication, while also maintaining high throughput and effectiveness, the Xillybus IP cores were used. These IP cores implement a communication based on the AXI4-Stream protocol, but provide a much more simplified interface from the FPGA side.

5.3 XILLYBUS IP CORE

Xillybus is an FPGA IP core for easy DMA communication, available in the Windows and Linux operating systems and developed by Xillybus Ltd. The Xillybus IP core as presented in Figure 5.2 implements an effective DMA-based streaming communication between the processor and the application logic that is ready to use using well-known interfaces. More specifically, the FPGA application logic connects to the IP using standard FIFOs (First-In-First-Out data structure), the data width and depth of which can be configured based on the designer's desired functionality. The "empty" and "full" signals of the FIFO are checked by the Xillybus (depending on the data direction) and when the FIFOs are ready for it, the data transfer is initiated. A different FIFO is used for data streamed from the host to the accelerator and from the accelerator to the host. For the software development on the side of the host, the user application simply needs to perform I/O operations on pipe-like device files.

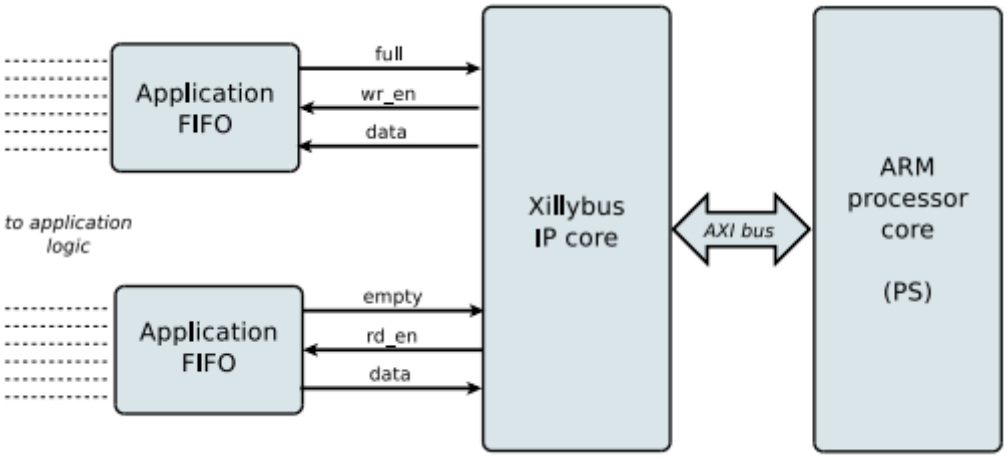


Figure 5.2: Communication interface of PS and PL using the Xillybus IP core [1].

There are plenty of advantages in using the Xillybus IP cores. First of all, it offers high flexibility in creating a different communication interface between different logic components (simply through the addition of extra FIFOs). Each of these components can be considered as a slave, while the Xillybus is the master of the communication protocol. Aside from that, the core is compatible with various operating systems (e.g. Windows, Linux), as well as different development boards (Zybo, ZedBoard, MicroBlaze etc.). Also, there is a simple and trivial programming model for different programming languages for the software part of the design, which will be illustrated later. Finally, there is support for both synchronous and asynchronous streams.

During system startup, the Xillybus driver allocates DMA buffers in the board's main memory. Data transferred from or to the accelerator is first saved to

these buffers. When a stream is marked asynchronous, it's allowed to communicate data between the FPGA and the host's kernel level software without the user space software's involvement, as long as the respective device file is open. Asynchronous streams have better performance, in particular when the data flow is continuous. Synchronous streams are easier to handle, and are the preferred choice when tight synchronization is needed between the host program's actions and what happens in the FPGA. Figure 5.3 illustrates the asynchronous stream's functionality.

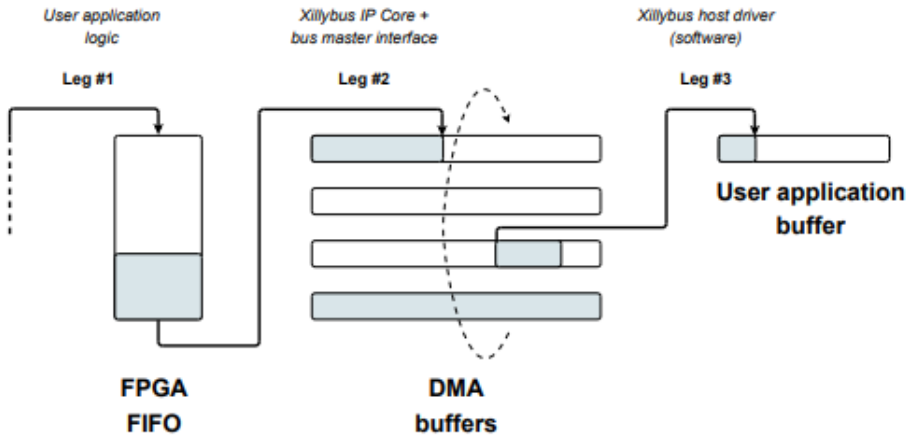


Figure 5.3: Memory Management of the Asynchronous stream [2].

On user application level, the two streaming interfaces (Host-To-FPGA, FPGA-To-Host) are treated as device files, and as such all the known system calls apply in the exact same way for Xillybus as well. This means that the driver supports the "open", "close", "read" and "write" commands. The device name is chosen when setting up the system in the IP core factory and can be found under the /dev/ directory in a Linux operating system. So typically an open command could be like this:

fd=open("/dev/host_to_fpga_device",O_WRONLY);

The O_WRONLY indicates that the file is only available for writing to, and cannot be read from. The file descriptor returned by the open command can then be used as an argument to the "read", "write" or "close" commands to refer to the device.

5.4 XILLYBUS-LITE IP CORE

The Xillybus IP core is responsible for implementing the AXI4-Stream communication protocol as explained before. However, with the hardware/software co-design of an application, a simpler protocol is required for the control communication. This is the kind of communication used when the processor wants to let the accelerator know when to start processing data, or when to halt the process and wait for some data to be sent, or to know at which processing stage the accelerator is if needed. In other words, control communication is very important in synchronizing the design and establishing a complete connection between the processor and the FPGA. Since the amount of control signals needing to be transferred are usually few in numbers, using a streaming protocol is a complex and undesired solution. That's where the AXI4-Lite protocol that was presented in chapter 3 comes to use. The Xillybus-Lite IP core offers the user application easy and shared access to registers or memory structures in the FPGA. Both the accelerator and the host can have access to these structures, allowing for quick and easy communication. Since this kit consists of an IP and a Linux driver, the designer is freed from having to deal with the complex AXI-bus interface and Linux kernel programming. Figure 5.4 demonstrates the IP's functionality.

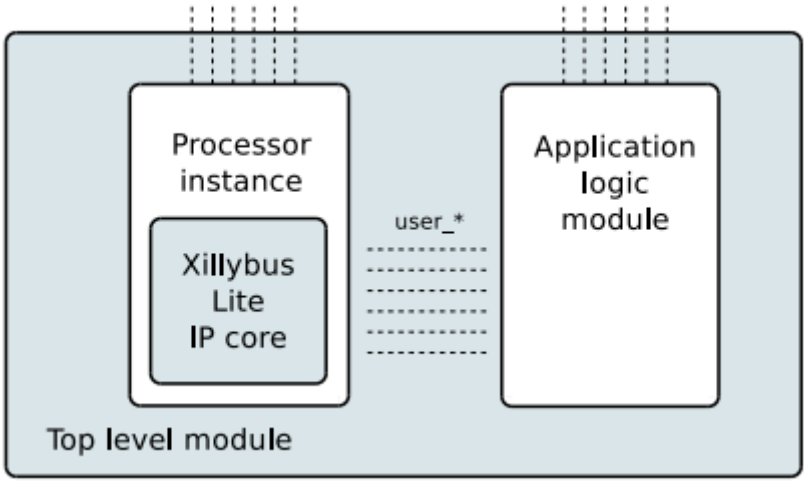


Figure 5.4: Illustration of the Xillybus Lite IP-core [3].

As stated, both the user space process and the accelerator have access to the same memory structure, which means that they need to have a common memory mapping to refer to the same addresses. Therefore, the address in the process's virtual memory space needs to be mapped to the actual physical address of the memory structure (which is usually a 32x32 bit RAM). Figure 5.5 depicts the standard code segment used for opening the Xillybus-Lite device file, mapping the address from the

virtual memory space to the physical one and accessing the memory structure with read/write operations:

```
int fd;
void *map_addr;
int size = ...;

fd = open("/dev/uio0", O_RDWR);

map_addr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED,
                fd, 0);

volatile unsigned int *pointer = map_addr;

*pointer = the_value_to_write;
the_value_read_from_register = *pointer;
```

Figure 5.5: Code Segment for Lite memory access [3].

The mmap() function is called to obtain the physical address for accessing the device. As can be noticed, reading or writing to the physical address of the RAM is deduced to a simple data operation in software level. The communication overhead for the signal to be transmitted is quite small, making it a highly efficient and productive method of communication.

5.5 HARDWARE/SOFTWARE INTEGRATION IN A COMPLETE SYSTEM

Having described all the hardware components used for implementing the affine transformation/measuring of the correlation measure of match on the accelerator and managing the communication between hardware and software, it is time to illustrate how the registration solver was implemented in a complete system.

The recurrent function that calculates the measure of match takes up over 99% of the processing's execution time as presented in Chapter 3. As a result, this is the function which was solely implemented on the hardware. The remaining components of the processing flow described in Chapter 3 (aside from the calculation of the initial components, which is practically a call of the previous function) were implemented in the software. The transformation parameters needed by the transformer are sent to the accelerator via the Xillybus-Lite system, together with the communication signals. The outputs of the function required for the correlation computation are sent to the user space via the Xillybus-Lite system as well. The Xillybus-IP is used to stream the images to the FPGA using the AXI4-Stream protocol in the initial stage of the image loading.

It is also important here to discuss the software environment that was used. Together with the Xillybus IPs presented above, Xillybus Ltd. provides the user with an already set up Linux distribution called Xillinux, which is based on the Ubuntu TS 12.04 for ARM which can be immediately deployed on Zybo. Xillinux is an operating system supporting most of the features as a regular desktop computer with a Linux operating system. The most important is the Graphical User Interface (GUI), which is greatly important in any image processing application for a visual understanding of the results and functionalities. Aside from that, Xillinux also supports a native compiler of user applications, relieving the designer from the need to use a cross-compiler in a desktop computer, and as such greatly reducing the debugging and development time. If necessary, multithreading is also supported to improve the execution time in parallel processing.

5.6 THE MEMORY PROBLEM & ITS SOLUTION

The implementation of the so far described architecture cannot work on the dataset of this thesis, because the size of the images is larger than the available memory in the FPGA's Block Rams. In fact, each image has a size of 1Mbyte, while the total memory available in Block Rams is 256Kbytes. This problem could be avoided if the registration had a typical memory access pattern like most image processing applications. This usually encountered pattern is based on streaming only a few pixel lines at a time and limiting the processing on these specific segments of the image. When the processing is even half way through these segments, the next segment transfer can be initiated, so as to be ready in the BRAMs when required. A prerequisite for this streaming pattern to be accurate and useful is that the access pattern is known beforehand.

However, that is not the case with the registration solver. Although the fixed image can be accessed with this pattern, the moving image cannot. The reason lies in the nature of the transformation itself. There is no standard pattern that can describe the memory accesses that will occur, due to the transformation parameters changing in every iteration of the optimizer. Assuming that the pixel with coordinates (0,0) is transformed to a pixel with coordinates (x_1,y_1), then the access pattern can either be linear or diagonal, with all 8 possible directions being possible (as shown in Figure 5.6), based on the sign and value of the six transformation parameters. The red painted pixel in Figure 5.6 symbolizes the initial pixel in the left image and the initial transformed pixel in the middle and right images. The location of this pixel can be completely random and this is just an example given for the comprehension of the pattern. The yellow color in the left image indicates the linear access pattern of the first image and each different color in the middle image indicates some of the possible direction patterns. Aside from the nine different patterns depicted in the second image, there are

even more, as the removal of the decimal part in the transformed coordinates can result in even more complex patterns, as depicted in the third image.

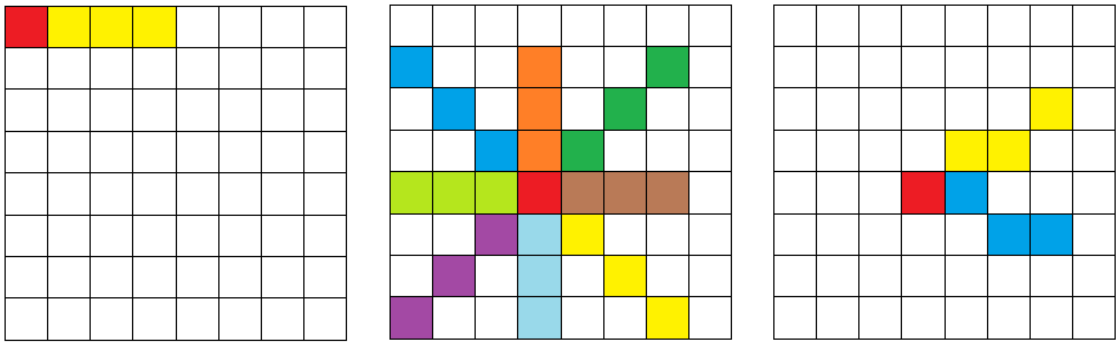


Figure 5.6: Different image access patterns with application of the transformation

This unorthodox access pattern is highly complex and cannot be optimally implemented using the DMA AXI4-Stream protocol that was described, as there is no way to predict each iteration's pattern before the transformation takes place. As a result, the transformation of the first pixel would first need to be completed in order to know where the streaming would have to begin from, and keeping in mind that some transformations will not be valid because of the out-of-bounds generated coordinates the throughput would be significantly reduced, even if an accurate control pattern could be designed.

In an attempt to bypass this problem, a downsizing of the images was tried out on software level. Image downsizing is a procedure in which the pixel information is changed and "shrinked", by discarding information, so as to reduce its size. The length and width of the image need to remain the same in order to fit the dataset prerequisites. After careful and detailed research of the generated transformation parameters, the best downsize produces images of 256x256 dimensions. The results of the registration for the downsized images were compared to those of the initial images and what was discovered was very interesting and useful. When it comes to the scaling and rotation parameters, the deviation between the two methods was very small (less than 1%) and the displacement parameter of the downsizing method (which is usually the most important parameter) was approximately 4 times smaller than the corresponding parameter of the initial method. Since $1000 \div 256 \cong 4$ this observation is very important, as the results of the downsized method can be used to get a highly accurate estimation of the results of the initial method. In fact, the scaling and rotation parameters will remain the same, while the displacement ones will be multiplied by 4.

Naturally, since the required calculations are now a lot less, there is an execution speedup caused by the downsizing. The execution time in a software implementation is 5-10 times faster, depending on the image pairing. A careful choice of the maximum allowed values for the transformation parameters is required to ensure that this method will be precise and accurate. Obviously, the initial downsizing of the images

constitutes an extra component at the start of the processing flow, which adds a small burden on the execution time. However this time interval required is much smaller than the time saved in the registration and therefore is not a problem. For the purpose of this thesis, an existing software tool was used to downsize the images, and the downsizing algorithm was not included in the design.

Since the image size is now 256x256=65536 bytes, both images can fit in the FPGA's Block Rams. The images will be streamed once at the beginning of the registration and whenever a pixel is required a simple memory addressing can be used to gain access to it.

CHAPTER 6: SYSTEM EVALUATION

6.1 RESOURCE UTILIZATION

This chapter provides an analytical view of the total FPGA resources used by the designed system, as well as problems that were encountered and how they were solved. Table 6.1 presents the detailed resource utilization of the finalized architecture.

Table 6.1: Resource Utilization of the completed architecture

Z-7010 resources	Available	Used	Utilization
LUT	17600	14164	80.48%
FF	35200	22621	64.26%
BRAM	60	35	58.33%
DSP	80	76	95%

It becomes obvious now why the application requires a lot of resources, as a result of the extended use of floating point arithmetic, and why the double precision could not be implemented, since it requires a lot more resources than the 32-bit single precision. Floating point circuits are bigger and more complex than the integer ones. As a result, the specific FPGA cannot support a processing parallelization, as that would multiply the amount of resources required, which is not feasible given the table above. In a platform with sufficiently more FPGA resources, the whole design could be doubled or tripled, thus dividing the processing in two or three independent partitions that will operate parallel to each other. As a result the speedup would be multiplied by a factor of 2 or 3 accordingly.

The routing algorithm of the Vivado tool that is part of the implementation of the design on the FPGA sometimes crashes when the resource utilization is stretched, for example more than 85%. This problem appeared a couple of times during the development. In order to avoid it, some of the floating point arithmetic circuits described in chapter 4 were modified to use more DSPs and fewer LUTs. This adjustment does not affect the final performance, since it only refers to which of the available hardware components are to be used to implement arithmetic circuits. Table 6.2 provides an example of how the use of DSP affects the utilization of LUTs and FFs. The chosen circuit was the multiplier of single precision floating point arithmetic and the detailed resource usage is presented for 1 and for 2 DSPs used separately. As a large number of multipliers and adders were part of the design, using more DSPs for each

of them significantly reduced the amount of logic blocks, freeing up valuable space for the design to be successfully implemented

Table 6.2: Resources based on DSP Usage of multiplier IP

Number of DSPs	LUTs	FFs
1	245	437
2	115	242

What is also important to note out is the amount of resources bound by the Xillybus IP cores, which were used for the hardware/software communication, as shown in Table 6.3. Keeping this information in mind during the development was important in order to make sure that some resources would be saved in the end for the implementation of the Xillybus cores.

Table 6.3: Xillybus IP Cores resource utilization

Z-7010 resources	Available	Used	Utilization
LUT	17600	2768	15.72%
FF	35200	2656	7.54%
BRAM	60	1	1.66%
DSP	80	0	0%

6.2 EXECUTION TIME COMPARISON

The final execution time of the hardware/software co-design on the Zybo was compared to the execution time on the software using the standalone ARM of the Zybo (with a maximum CPU frequency of 900MHz and DDR memory of 512 MB) and to the corresponding one using a fast standalone CPU of a desktop computer (with a maximum CPU frequency of 2.4GHz and a RAM memory of 6GB). Before presenting the times' comparison an important feature of the floating point circuit for the accumulation needs to be analyzed. This component (made by Xilinx) first rounds the floating point inputs and accumulates after. As a result, a small accumulation error is added to the final calculation because of the rounding. In the absence of elastic de-formations, where the optimizer is stable, this error is small and insignificant

There are two more factors that introduce error to the calculations. One is the choice between single and double precision for the floating point arithmetic. The second is the method of reading the image in the software. The loading of the images can be done either by using the OpenCV library or by saving the image as a text file and reading from it. Since the installation of the OpenCV library in the Zybo was not feasible the second method was used. In case of stable optimizers for well aligned images without elastic deformations, these errors are not very important. After converging the 256x256 found parameters to the correct ones for the image size of 1000x1000 (by properly adjusting the displacement), the maximum displacement error that was found (which is the most important) between the different methods was 9 pixels, which is less than 1% of the image's width. Practically this means that visually the registration will still succeed.

Although the quality of the results regarding the errors will be examined in section 6.3, it is important to say how this error affects the execution time speedup as well. Since the optimizer searches the exploration space for an optimum solution, small errors in the measured matches can lead to different steps taken during the processing. The final convergence might still be about the same, but depending on the errors, a different amount of iterations may be required for the optimizer to converge to the said solution. The outcome of this phenomenon is that the speedup is not standard, but varies depending on the inputs. Table 6.4 illustrates the execution time for 3 successful registrations and 1 unsuccessful for processing of the downsized images (256x256). Each time presented is derived from the average of 100 executions for a more stable and statistically correct measurement.

Table 6.4: Execution time comparison between all implementations

	CPU/FPGA co-design	Standalone CPU ARM in Zybo	Standalone CPU in desktop computer
First Successful registration time(seconds)	0,31169519	12,583982	1,10332952
Speedup		x40,37	x3,53
Second Successful registration time(seconds)	0,29721025	19,729760	0,87024528
Speedup		x66,38	x2,93
Third Successful registration time(seconds)	0,29172961	10,775993	0.6954013
Speedup		x36,94	x2,38
Unsuccessful registration time(seconds)	0,26560579	11,236088	0,98315164
Speedup		x42,30	x3,37

6.3 QUALITY OF RESULTS COMPARISON

The final matter that needs to be considered is the quality of the parameters generated by the registration solver implemented using the co-design, which is to be compared to the respective parameters generated in a Desktop PC using a standalone processor. In fact, a more extensive comparison will take place in this sub-chapter, as there will be a display of results generated in 4 different methods:

- Standalone Desktop CPU for initial image sizes (1000x1000) using OpenCV and double precision. **(Method 1)**
- Standalone Desktop CPU for downsized images using OpenCV and single precision. **(Method 2)**
- Standalone ARM CPU on Zybo for downsized images using single precision and reading from text file. **(Method 3)**
- FPGA/CPU co-design on Zybo for downsized images using single precision and reading from text file. **(Method 4)**

Two examples for each method will be illustrated, one with a very small (practically insignificant) error and one with a relatively larger one. Both are examples of registrations that were successful. Tables 6.5 and 6.6 refer to the results of Methods 1 and Method 2.

Table 6.5: Method 1 results

Method 1		
Parameters	**First pair**	**Second pair**
T1	1.009586	0.992332
T2	-0.016319	0.018466
T3	178.587102	-194.799257
T4	0.027438	-0.015719
T5	1.003454	0.990693
T6	-62.193396	-23.551012
Measure of Match	0.783938	0.827221

Table 6.6: Method 2 results

Method 2		
Parameters	**First pair**	**Second pair**
T1	1.002974	1.045161
T2	-0.015803	0.026128
T3	45.320175	-48.672958
T4	0.024869	0.000492
T5	0.997309	1.011216
T6	-16.065662	-5.656442
Measure of Match	0,792186	0,797225

In order to decide whether the second Method is sufficient a comparison of the parameters with the Method 1 is conducted. Since the parameters T1, T2, T4 and T5 which refer to scaling and rotation have small deviations between the two methods, the most important ones to look are T3 and T6 which refer to the displacement. Multiplying T3 in the first pair by the fraction $\frac{1000}{256} = 3{,}90625$, one can get $T3_{new}$=177,032, which is about 1.5 pixels less than the T3 of Method 1. This error is insignificant in the visual representation after the edge detection and fusion are applied. Thus, we conclude that using the registration solver on the downsized images one can get a very satisfying estimation of the required transformation, which can then be effectively applied on the initial images, without any significant errors.

Tables 6.7 and 6.8 refer to the relevant results regarding the next two methods, both of which were implemented on Zybo. A small error between them and Method 2 will result in a successful registration, where the errors are of no practical significance.

Table 6.7: Method 3 results

Method 3		
Parameters	**First pair**	**Second pair**
T1	1.010045	1.040799
T2	-0.015795	0.025317
T3	45.801357	-48.795601
T4	0.027911	-0.001520
T5	1.003294	1.009725
T6	-15.859092	-5.936245
Measure of Match	0,792874	0,799729

Table 6.8: Method 4 results

Method 4		
Parameters	**First pair**	**Second pair**
T1	0.990064	1.002257
T2	-0.017116	0.023562
T3	44.415871	-48.941826
T4	0.021519	-0.011582
T5	0.989229	1.034919
T6	-15.909966	-5.749107
Measure of Match	0,780669	0,806234

It immediately becomes obvious that the errors do not affect the performance and the credibility of the registration solver, as long as there are no elastic deformations and the optimizer is stabilized. It was noted that all the image pairs in the dataset, which can be successfully registrated, had satisfying results compared with all previously mentioned methods. As a result, the registration was sped up not only because of the image downsizing, but also because of the FPGA implementation, achieving even greater times than before.

CHAPTER 7: CONCLUSIONS

7.1 THESIS SUMMARY AND CONCLUSIONS

Image registration is a computationally demanding field of image processing with an impact on medical, military and navigating applications. Rendering the registration solvers effective and practical in embedded system designs can be a difficult and complex matter, as the traditional software designing methodologies are not sufficient in terms of execution time and sometimes power consumption.

The topic of this thesis was the implementation of a software/hardware co-design, capable of achieving the desired precision, whilst meeting the timing criteria. More precisely a hardware platform with a SoC FPGA (Zybo) was chosen, in order to implement the time consuming part of the registration algorithm. Alternative platforms suitable for the design could be GPUs or ASICs. After briefly presenting the scientific background of the registration problem, the chosen platform's capabilities and characteristics were analyzed, with an extensive reference to the basic aspects of the FPGA and the major communication protocols. In order to handle the designing complexity, the standard processing flow of a registration solver was modified in such a way so as to render the co-design feasible. After a proper and careful timing profiling, the computationally heavier function was removed from the software domain and implemented on the FPGA, thereby achieving the optimum partition between the two design levels. The communication between the FPGA and the ARM processor was achieved using an effective and simple system called Xillybus, which incorporates the AXI4-Streaming protocol using proper and hidden from the user DMA drivers. Having already designed the accelerator and established an optimal communication with the processor, the solver's final integration was illustrated. The methodologies and ideas explained in this thesis can be used in many other image processing applications, where effective image streaming protocols are required, in combination with a careful synchronization and communication.

Finally, the system's evaluation was carried out. Since, as already explained, there is a small precision error in the calculations on the FPGA, because of rounding of numbers caused by specific hardware components, the output of the design had a small deviation from the output of the initial implementation. For images that can be processed using the rigid model, this error is too small and doesn't affect the performance of the registration solver. However, in some cases, where elastic deformations may be present, the optimizer can be quite unstable, producing different results based on the architecture and the 32-bit or 64-bit choice of floating point numbers' representation. The speedup achieved can vary from 40 to 80 times when compared to the implementation on the standalone ARM processor, a number which can vary between image sets, because of different errors in each that may lead to a requirement for more

iterations of the optimizer. The corresponding speedup when compared to a laptop or desktop computer with a high frequency processor was 2-4 times.

7.2 FUTURE WORK/EXTENSIONS

Future work that could improve certain aspects of the design, as well as expand its characteristics could be the following. Firstly, the downsizing algorithm can be incorporated in the program, so that the images can get downsized immediately before the registration, whatever the image size and without the use of external tools. This procedure will obviously need to be executed by the processor, as it is not very time consuming and the remaining resources in the FPGA are limited. There can also be a software procedure, which will apply the optimum transformation of the optimizer to the moving image and run an edge detection and image fusion for a visual representation of the registration's results, in order to determine whether it was successful or not. Finally, different floating point representations can be examined to see if they match the application's criteria, such as the fixed point representation, which will offer a trade between resources utilized and available precision. If the experiments show that this design is feasible, the great amount of resources saved can allow for extra parallelization. Splitting the design in two identical components that will do the same processing on each half of the image can increase the speed by a factor of 2.

Aside from these, different platforms can also be examined to speed the registration up, such as implementation on a CPU-GPU platform or a SoC FPGA with more resources than the Zybo, which would allow for the implementation of the solver, without the need to downsize the images and potentially lose important precision in stricter applications. Acceleration of elastic deformation models can also be a very challenging task. As the complexity is significantly higher than the rigid model's, more resources are bound to be needed, as well as a possibly different design flow. A final comparison between all the said designs can lead to a decision as to which architecture is the optimum and most practical one to use.

REFERENCES

[1] Xillybus,Xillybus Ltd. Getting started with the FPGA demo bundle for Xilinx available on: http://xillybus.com/downloads/doc/xillybus_getting_started_xilinx.pdf

[2] Xillybus,Xillybus Ltd. Xillybus host application programming guide for Linux available on: http://xillybus.com/downloads/doc/xillybus_host_programming_guide_linux.pdf

[3] Xillybus,Xillybus Ltd. The guide to Xillybus Lite available on: http://xillybus.com/downloads/doc/xillybus_lite.pdf

[4] Zynq, Xilinx. SoCs with Hardware and Software Programmability available on: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[5] Xilinx. AXI Reference Guide available on: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf

[6] Shams R., Sadeghi P., Kennedy A. R., Hartley I. R. (March 2010). A Survey of Medical Image Registration on Multicore and the GPU. *IEEE Signal Processing Magazine* 27 (2): 50-60.

[7] Matsopoulos G.K., Mouravliansky N.A, Delibasis K.K., Nikita K.S. (March 1999). Automatic Retinal Image Registration Scheme Using Global Optimization Technique. *IEEE Transactions on Information Technology in Biomedicine 3* (1): 47-60.

[8] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. (1988). *Numerical recipes in C—the art of scientific computing*. Cambridge: Cambridge University Press.

[9] Goshtasby A.A. (2005). *2-D and 3-D Image Registration: for Medical, Remote Sensing, and Industrial Applications*. Dayton, Ohio: John Wiley & Sons.

[10] Stratakos I. (October 2016) *Design and Implementation of Image Processing Algorithms on SoC Platforms for Embedded Applications* available on: https://pergamos.lib.uoa.gr/uoa/dl/frontend/file/lib/default/data/1325663/theFile

[11] Dandekar O., Shekhar R. (June 2007). FPGA-Accelerated Deformable Image Registration for Improved Target-Delineation during CT-Guided Interventions. *IEEE Transactions on Biomedical Circuits And Systems* 1 (2): 116-127.

[12] White B. A., (Spring 2009). *Using FPGAs to Perform Embedded Image Registration*, available on: https://pdfs.semanticscholar.org/579f/f4cd8383fa39e73314c7fb8c0a5c68ff9e81.pdf

[13] Sen M., Hemaraj Y., Plishker W., Shekhar R., Bhattacharyya S. S. (March 2008). Model-based mapping of reconfigurable image registration on FPGA platforms. *J Real-Time Image Proc* (2008) 3:149–162.

[14] Kalla M.P, Economopoulos T.L., Matsopoulos G.K. (May 2017). 3D dental image registration using exhaustive deformable models: a comparative study. *British Institute of Radiology* 46 (7)

[15] Laliberte F., Gagnon L.,Sheng Y. (June 2003). Registration and fusion of retinal images: an evaluation study. *IEEE Transactions on Medical Imaging*; 22 (5): 404-418.

[16] Di Stefano L., Mattocia S.,Tombari F. (October 2004) . An algorithm for efficient and exhaustive template matching. *Image Analysis and Recognition* 3211: 408-415.