ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΟΝ ΗΛΕΚΤΡΟΝΙΚΟ ΑΥΤΟΜΑΤΙΣΜΟ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Design and Implementation of Image Processing Algorithms on SoC Platforms for Embedded Applications

Ιωάννης Γ. Στρατάκος
Α.Μ. : 2013522

Επιβλέπων : Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής ΕΜΠ
Συνεπιβλέπων : Διονύσιος Ι. Ρεΐσης
Αναπληρωτής Καθηγητής ΕΚΠΑ

Αθήνα
Οκτώμβριος 2016

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΣΤΟΝ ΗΛΕΚΤΡΟΝΙΚΟ ΑΥΤΟΜΑΤΙΣΜΟ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Design and Implementation of Image Processing Algorithms on SoC Platforms for Embedded Applications

Ιωάννης Γ. Στρατάκος
Α.Μ. : 2013522

Επιβλέπων : Δημήτριος Ι. Σούντρης
Αναπληρωτής Καθηγητής ΕΜΠ
Συνεπιβλέπων : Διονύσιος Ι. Ρεΐσης
Αναπληρωτής Καθηγητής ΕΚΠΑ

Τριμελής Επιτροπή Εξέτασης

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
.....	.....	.....
Δημήτριος Σούντρης Αναπληρωτής Καθηγητής ΕΜΠ	Διονύσιος Ρεΐσης Αναπληρωτής Καθηγητής ΕΚΠΑ	Γεώργιος Λεντάρης Διδάκτωρ ΕΚΠΑ

Ημερομηνία Εξέτασης:
13 Οκτωβρίου 2016

Περίληψη

Σήμερα η συνεχής πρόοδος στην τεχνολογία έχει οδηγήσει στην ανάπτυξη πιο πολύπλοκων και υπολογιστικά απαιτητικών αλγορίθμων επεξεργασίας εικόνας. Πολλοί από αυτούς τους αλγόριθμους έχουν υιοθετηθεί σε ενσωματωμένα συστήματα τα οποία στοχεύουν σε μια ποικιλία εφαρμογών, όπως η αυτοκινητοβιομηχανία, 3D πλοήγηση, την επιτήρηση, κλπ. Ωστόσο, σε ενσωματωμένα συστήματα πραγματικού χρόνου, όπου η καθυστέρηρη μεταφοράς δεδομένων και τη κατανάλωση ισχύος διαδραματίζουν σημαντικό ρόλο, εφαρμογές λογισμικού προσανατολισμένες να εκτελούνται σε επεξεργαστές γενικής χρήσης δεν μπορούν να προσφέρουν ικανοποιητικές λύσεις.

Σκοπός της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός ενός συστήματος επεξεργασίας εικόνας για ενσωματωμένες εφαρμογές, η υλοποίησή του σε Σύστημα-σε-Ψηφίδα και η αξιολόγηση του. Ως εφαρμογή επιλέχθηκε ο εντοπισμός ενός αντικειμένου και η ενημέρωση του χρήστη για το πώς να μετακινήσει την κάμερα ώστε να ευθυγραμμιστεί με το αντικείμενο. Ένα βασικό στοιχείο αυτής της διπλωματικής εργασίας είναι η μελέτη των βημάτων επεξεργασίας της εικόνας και ο εντοπισμός των κρίσιμων βημάτων που επηρεάζουν σημαντικά το χρόνο επεξεργασίας. Με την εφαρμογή μεθοδολογιών συσχεδίασης Υλικού/Λογισμικού, επιμέρους τμήματα αναπτύχθηκαν ως κομμάτια του υλικού με τη χρήση VHDL και τα υπόλοιπα αναπτύχθηκαν ως συστατικά στοιχεία του λογισμικού χρησιμοποιώντας τη γλώσσα προγραμματισμού C. Η πλατφόρμα που χρησιμοποιήθηκε για την ανάπτυξη του συστήματος βασίζεται στην οικογένεια των συσκευών Zynq-7000 All Programmable SoC (AP SoC) της Xilinx. Η αξιολόγηση του συστήματος βασίστηκε στη μέτρηση του ποσοστού επιτυχίας της συνολικής λειτουργίας και τις μετρήσεις που σχετίζονται με την πλατφόρμα που χρησιμοποιείται, όπως η κατανάλωση ισχύος, η κατανάλωση πόρων, ο χρόνος εκτέλεσης κλπ. Επίσης πραγματοποιήθηκε σύγκριση με εναλακτικές πλατφόρμες οι οποίες εκτελούσαν την λειτουργία του συστήματος αποκλειστικά στο λογισμικό.

Λέξεις Κλειδιά— Συσχεδίαση Υλικού/Λογισμικού, Επεξεργασία Εικόνας, Σύστημα-σε-Ψηφίδα, Harris Corner Detector, Ενσωματωμένο Σύστημα, Zynq

Abstract

Nowadays the ever-increasing advancements in technology has led to the deployment of more complex and computationally intensive image processing algorithms. Many of these algorithms have been adopted in present-day embedded systems targeting a variety of applications such as automotive, 3D navigation, surveillance, etc. However in real-time embedded systems, where latency and power play an important role, software-oriented implementations running on general purpose CPUs may not offer satisfactory solutions.

The purpose of this thesis is the design of an image processing system for embedded applications, its deployment on a System-on-Chip (SoC) platform and the evaluation of the developed system. As a case study was selected to identify an object and to inform the user on how to move the camera in order to be able to stay aligned with the object. A key element of this thesis is the study of the image processing steps and the identification of critical steps that significantly affect the processing time. By applying hardware/software codesign methodologies individual parts were implemented as hardware components described using VHDL and the rest developed as software components using the C programming language. The platform used to deploy the system is based upon Xilinx's Zynq-7000 All Programmable SoC (AP SoC) family of devices. The system was evaluated based on measuring the success rate of the overall operation and on measurements related to the SoC platform used, such as power consumption, resource utilization, execution time etc. Also a comparison with software oriented approaches executed in CPUs was conducted.

Keywords— Hardware/Software codesign, Image Processing, System-on-Chip, Harris Corner Detector, Embedded System, Zynq

Ευχαριστίες

Πρώτον, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στους επιβλέποντές μου, Καθηγητή Δημήτριο Σούντρη ΕΜΠ και Καθηγητή Διονύσιο Ρεΐση ΕΚΠΑ που με εμπιστεύθηκαν με αυτήν την διπλωματική εργασία που αντιμετωπίζει ένα πολύ απαιτητικό θέμα σε ένα πολύ ανταγωνιστικό πεδίο, όπως η επεξεργασία εικόνας. Ειδικά τον Καθηγητή Δημήτριο Σούντρη που μου έδωσε την ευκαιρία να εκπονήσω την διπλωματική μου εργασία στο Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων (MicroLab) στο ΕΜΠ.

Επίσης, θα ήθελα να ευχαριστήσω τον μεταδιδακτορικό ερευνητή Γεώργιο Λεντάρη και τον υποψήφιο διδάκτωρ Κωνσταντίνο Μαραγκό για τη βοήθεια και τη συνεργασία τους καθ ΄όλη τη διάρκεια της διπλωματικής μου. Οι οξυδερκείς παρατηρήσεις τους κατά τη διάρκεια των συνομιλιών μας με βοήθησαν να βελτιώσω και να επεκτείνω τις γνώσεις μου γύρω από το θέμα της διπλωματικής αυτής εργασίας και είμαι ευγνώμων για το ότι θα έχω την ευκαιρία να συνεργαστώ μαζί τους στο μέλλον. Θα ήθελα επίσης να ευχαριστήσω όλα τα μέλη του Microlab για το ευχάριστο περιβάλλον εργασίας.

Τέλος θα ήθελα να ευχαριστήσω τους φίλους και την οικογένειά μου. Η συνεχή υποστήριξή τους καθ ΄όλη τη ζωή και τις σπουδές μου, μου έδωσε δύναμη να συνεχίσω να επιδιώκω τους στόχους μου.

Acknowledgments

First I would like to express my gratitude to my supervisors, Prof. Dimitrios Soudris NTUA and Prof. Dionysios Reisis NKUA for trusting me with this diploma thesis that tackles a very demanding topic in a very competitive field such as image processing. Especially Prof. Dimitrios Soudris who gave me the opportunity to do my diploma thesis at the Microprocessors and Digital Systems Lab (MicroLab) in NTUA.

Also i would like to thank Post-Doctoral researcher Georgios Lentaris and Phd candidate Konstantinos Maragos for their help and cooperation throughout the course of my diploma thesis. Their insightful comments during our conversations helped me improve and expand my knowledge around the topic of this diploma thesis and i am grateful that i will have the opportunity to work with them in the near future. I would also like to thank all the members of Microlab for the pleasant working environment.

Last but not least, I would like to thank my friends and family. Their continuous support throughout my life and studies gave me strength to continue pursue my goals.

Contents

List of Figures

List of Tables

Listings

Εκτεταμένη Περίληψη

Εισαγωγή

Σήμερα η συνεχής πρόοδος στην τεχνολογία έχει οδηγήσει στην ανάπτυξη πιο πολύπλοκων και υπολογιστικά απαιτητικών αλγορίθμων επεξεργασίας εικόνας. Πολλοί από αυτούς τους αλγόριθμους έχουν υιοθετηθεί σε ενσωματωμένα συστήματα τα οποία στοχεύουν σε μια ποικιλία εφαρμογών, όπως η αυτοκινητοβιομηχανία, 3D πλοήγηση, την επιτήρηση, κλπ. Ωστόσο, σε ενσωματωμένα συστήματα πραγματικού χρόνου, όπου η καθυστέρηρη μεταφοράς δεδομένων και τη κατανάλωση ισχύος διαδραματίζουν σημαντικό ρόλο, εφαρμογές λογισμικού προσανατολισμένες να εκτελούνται σε επεξεργαστές γενικής χρήσης δεν μπορούν να προσφέρουν ικανοποιητικές λύσεις. Αυτό προκύπτει από το γεγονός ότι οι επεξεργαστές έχουν περιορισμένες δυνατότητες παράλληλης επεξεργασίας για να υποστηρίξουν τις απαιτήσεις των εφαρμογών αυτών και καταναλώνουν σημαντικά ποσά ισχύος. Προκειμένου να ξεπεραστούν αυτά τα μειονεκτήματα και να αυξηθεί η απόδοση/watt, διάφορες προσεγγίσεις έχουν προταθεί όπου εξειδικευμένο υλικό χρησιμοποιείται παράλληλα με την CPU για την επιτάχυνση κρίσιμων τμημάτων ή ακόμη και ολόκληρων αλγορίθμων. Οι προσεγγίσεις αυτές βασίζονται σε διάφορους συνδυασμούς σε επίπεδο συστήματος, όπως CPU-DSP, CPU-GPU και CPU-FPGA. Ωστόσο, αυτό το είδος των συστημάτων πάσχουν από μειωμένη απόδοση που προκύπτει από την αύξηση της καθυστέρησης στην επικοινωνία μεταξύ των διαφόρων συσκευών. Μια άλλη εναλλακτική λύση είναι η κατασκευή ASIC. Ωστόσο, η προσέγγιση αυτή θεωρείται αναποτελεσματική από την άποψη του χρόνου διάθεσης στην αγορά και το κόστος ανάπτυξης και υλοποίησης.

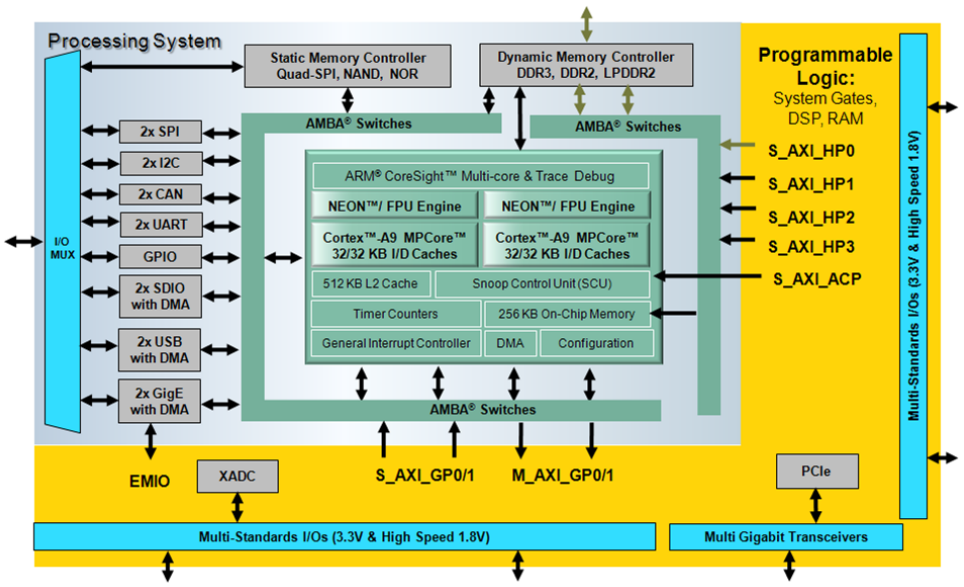
Τα τελευταία χρόνια, οι κατασκευαστές FPGA εισήγαγαν System-on-Chip (SoC) συσκευές, οι οποίες συνδυάζουν ενσωματωμένο επεξεργαστή(-ες) μαζί με προγραμματιζόμενη λογική FPGA στο ίδιο τσιπ (SoC FPGA). Η ενσωμάτωση αυτή επιτρέπει μεγαλύτερη απόδοση, εξοικονόμηση ενέργειας και τη βελτίωση γενικά της επικοινωνίας, σε αντίθεση με απλές υλοποιήσεις FPGA που εξωτερικά επικοινωνούν με CPUs Αυτά τα σημαντικά πλεονεκτήματα μπορούν να αξιοποιηθούν σε ενσωματωμένα συστήματα πραγματικού χρόνου που στοχεύουν απαιτητικές υπολογιστικές εφαρμογές, όπως η επεξεργασία εικόνας, και να παρέχουν ελκυστικές λύσεις με εντυπωσιακές τιμές αποδόσεων/watt. Ακόμα κι αν τα SoC FPGAs παρέχουν μια μεγάλη σειρά από οφέλη, η διαδικασία ανάπτυξης μιας εφαρμογής σε αυτές την πλατφόρμες αποτελεί μια πολύπλοκη και χρονοβόρα διαδικασία. Οι κατασκευαστές SoC FPGA έχουν επενδύσει πολλή προσπάθεια για την προώθηση ειδικών εργαλείων και framework με στόχο να διευκολυνθεί η διαδικασία ανάπτυξης SoC συστημάτων. Σήμερα, έχουν σημειώσει σημαντική πρόοδο όσον αφορά την μείωση της προσπάθειας προγραμματισμού υλικού με την εισαγωγή της Σύνθεσης Υψηλού Επιπέδου (High Level Synthesis - HLS) [3]. Παρόλ΄ αυτά η ενσωμάτωση υλικού/λογισμικού και η εφαρμογή της κατάλληλης επικοινωνίας μεταξύ τους εξακολουθεί να παραμένει ένα πολύ επίπονο και δύσκολο έργο για τον μηχανικό. Αυτό είναι αποτέλεσμα της μεγάλης προσπάθειας που απαιτείται για την κατανόηση των χαμηλού επιπέδου λεπτομεριών των πρωτοκόλλων επικοινωνίας, τη προσαρμογή των διεπαφών του επιταχυντή υλικού σε αυτά τα πρωτόκολλα και το cross-trigger debugging, προκειμένου να επικυρώθει η σωστή ενσωμάτωση υλικού/λογισμικού.

Περιγραφή συστήματος και διαδικασία υλοποίησης

Πλατφόρμα Υλοποίησης

Περίπου το 2010 η Xilinx παρουσίασε την οικογένεια συσκευών Zynq-7000 All Programmable SoC, τις πρώτες συσκευές SoC που συνδύαζαν τα χαρακτηριστικά ενός Dual-Core ARM Cortex A9 επεξεργαστή (υποσύστημα PS) με προγραμματιζόμενη λογική (υποσύστημα PL), ή με άλλα λόγια έναν διπύρηνο επεξεργαστή με ένα FPGA.

Στη συσκευή Zynq, ο ARM Cortex-A9 είναι ένας επεξεργαστής που μπορεί να εκτελέσει λειτουργικά συστήματα, όπως το Linux, ενώ η προγραμματιζόμενη λογική βασίζεται στην αρχιτεκτονική 7-series FPGA της Xilinx . Η ενσωμάτωση ενός επεξεργαστή ARM με FPGA, σε μία μόνο συσκευή, προσφέρει στους προγραμματιστές τη δυνατότητα εφαρμογής ενοποιημένης προσέγγισης στην σχεδίαση υλικού/λογισμικού για ενσωματωμένα συστήματα και προσφέρει επίπεδα απόδοσης που λύσεις με δύο συσκευές (π.χ. CPU εξωτερικά συνδεδεμένη με ένα FPGA) δεν μπορούν να προσφέρουν λόγω του περιορισμένου εύρους ζώνης επικοινωνίας, της καθυστέρησης και της κατανάλωση ενέργειας τους. Η Xilinx έχει υιοθετήσει το δίαυλο δεδομένων για μικροελεγκτές AMBA και το πρωτόκολλο AXI ως το κύριο μέσο επικοινωνίας μεταξύ του PS και της λογικής που υλοποιείται στο PL, όπως φαίνεται στην Εικόνα 1 όπου παρουσιάζεται η αρχιτεκτονική των συσκευών Zynq SoC.



Εικόνα 1: Αρχιτεκτονική συσκευών Zynq SoC.

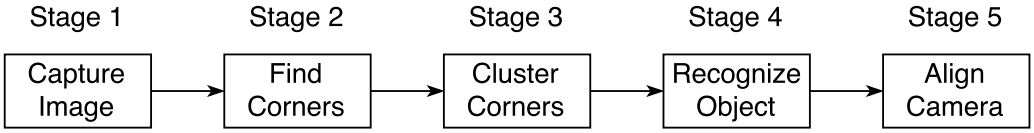
Το πρωτόκολλο AXI είναι ένα πρωτόκολλο επικοινωνίας που ανήκει στην οικογένεια πρωτοκόλλων επικοινωνίας για μικροελεγκτές ARM AMBA. Το AMBA είναι ένα ανοιχτών προδιαγραφών πρότυπο για on-chip διασύνδεση, που επιτρέπει τη σύνδεση και τη διαχείριση πολλών ελεγκτών και περιφερειακών σε ένα multi-master σύστημα. Απευθύνεται σε συστήματα υψηλής απόδοσης και συχνότητας. Η τέταρτη έκδοση του πρωτοκόλλου, AXI4, έχει σχεδιαστεί και βελτιστοποιηθεί για να χρησιμοποιείται ως μέσο επικοινωνίας μεταξύ πυρήνων υλικού IP σε συστήματα υλοποιημένα με FPGA. Υπάρχουν τρεις τύποι διεπαφών AXI4, καθεμία κατάλληλη για εφαρμογές με διαφορετικές απαιτήσεις [14]:

- **AXI4 (or AXI4-Full)**: Διασύνδεση υψηλής απόδοσης, κατάλληλη για memory mapped επικοινωνία που επιτρέπει ριπές δεδομένων έως και 256 δεδομένα ανά διεύθυνση.
- **AXI4-Lite**: Μια παραλλαγή της διεπαφής AXI4-Full, που χρησιμοποιείται για memory mapped επικοινωνία μόνο. Αυτή η παραλλαγή δεν υποστηρίζει ριπές δεδομένων και έτσι

μόνο ένα δεδομένο ανά διεύθυνση μεταφέρεται.

- **AXI4-Stream**: Η διεπαφή αυτή ορίζει ένα μόνο κανάλι για μεταφορά δεδομένων συνεχούς ροής (streaming), επιτρέποντας ριπές δεδομένων απεριορίστου μεγέθους. Η σύνδεση είναι από τον Master στον Slave μόνο, οπότε αν χρειάζεται αμφίδρομη μεταφορά μεταξύ δυο περιφεριακών, τα περιφερειακά πρέπει να υλοποιούν συγχρόνως διεπαφές τύπου Master και Slave.

Ροή επεξεργασίας εικόνων



Εικόνα 2: Ροή επεξεργασίας συστήματος.

Η υλοποίηση του συστήματος βασίζεται σε ροή επεξεργασίας έξι σταδίων όπως φαίνεται στην Εικόνα 2. Τα στάδια αυτά περιγράφονται στη συνέχεια:

- Το 1ο στάδιο αφορά τη λήψη της εικόνας με τη βοήθεια μιας κάμερας που συνδέεται με την πλατφόρμα. Η εικόνα αποθηκεύεται προσωρινά σε μια προκαθορισμένη περιοχή στην κύρια μνήμη του συστήματος ώστε να είναι έτοιμη προς επεξεργασία.

- Το 2ο στάδιο είναι η επεξεργασία της εικόνας για να βρεθούν οι γωνίες, χρησιμοποιώντας το γνωστό αλγόριθμο εύρεσης γωνιών Harris. Στο στάδιο αυτό, η αποθηκευμένη εικόνα γίνεται είσοσδος στον Harris, ο οποίος με τη σειρά του εξάγει τις γωνίες που βρέθηκαν επιστρέφοντας τις συντεταγμένες τους και τη δύναμη της απόκρισης της γωνίας, δηλαδή μία τιμή που υποδηλώνει πόσο καλή γωνία είναι. Τα αποτελέσματα από αυτό το στάδιο αποθηκεύονται στη δική τους περιοχή στην κύρια μνήμη του συστήματος.

- Το 3ο στάδιο είναι το στάδιο στο οποίο τα αποτελέσματα που προέκυψαν από το Στάδιο 2 επεξεργάζονται από έναν αλγόριθμο συσταδοποίησης (clustering alogrithm). Η έξοδος αυτού του σταδίου είναι ένας αριθμός από συστάδες (ομάδες). Στο τέλος αυτού του σταδίου κάθε γωνιά που βρέθηκε κατηγοριοποιείται σε μία από αυτές τις συστάδες και ένα αντιπροσωπευτικό σημείο (κεντροειδές) για κάθε ομάδα υπολογίζεται. Για αυτό το στάδιο δύο αλγόριθμοι συσταδοποίησης δοκιμάστηκαν, ο αλγόριθμος k-means και ο αλγόριθμος DBSCAN, και οι οποίοι θα παρουσιαστούν αργότερα.

- Το 4ο στάδιο υπολογίζει τη γεωμετρία του αντικειμένου, με βάση τα κεντροειδή που υπολογίστηκαν στο Στάδιο 3, και τη συγκρίνει με ένα προκαθορισμένο αφηρημένο μοντέλο του αντικειμένου που θέλουμε να παρακολουθείται.

- Τέλος κατά το 5ο στάδιο, εάν η ανίχνευση του αντικειμένου είναι επιτυχής, το σύστημα ενημερώνει πώς να κινήθει η κάμερα ώστε να ευθυγραμμιστεί με το αντικείμενο.

Συσχεδίαση Υλικού/Λογισμικού Συστήματος

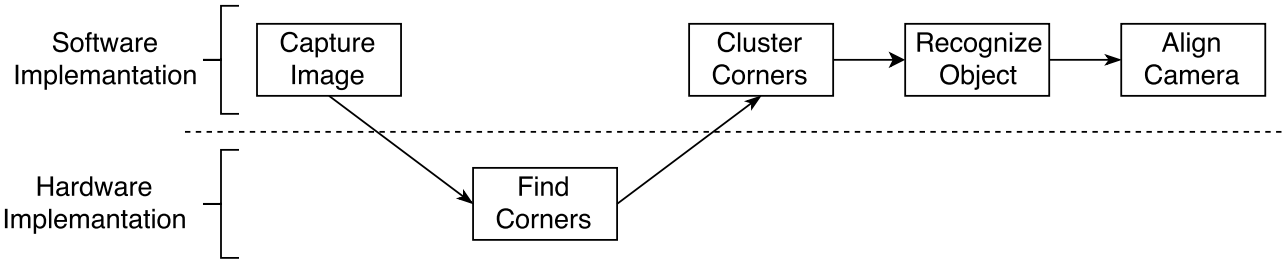
Η συσχεδίαση υλικού/λογισμικού αφορά στον ταυτόχρονο σχεδιασμό τόσο του λογισμικού όσο και του υλικού σε ένα σύστημα. Το λογισμικό είναι αυτό που εκτελείται σε επεξεργαστές, όπως η κεντρική μονάδα επεξεργασίας (CPU) και επεξεργαστές ψηφιακών σημάτων (DSP). Το υλικό υλοποιείται ως ASIC ή σε FPGA. Δεδομένου ότι τα συστήματα λογισμικού και υλικού έχουν

Πίνακας 1: Χρόνοι εκτέλεσης για τα στάδια επεξεργασίας του συστήματος.

Capture Image	Find Corners (Harris)	Cluster Corners		Recognize Object	Align Camera
		k-means	DBSCAN		
$\ll 1$ msec	~ 430 msec	$\ll 1$ msec	$\ll 1$ msec	$\ll 1$ msec	$\ll 1$ msec

διαφορετικά χαρακτηριστικά, ένας συνδυασμός τους έχει τη δυνατότητα να υοθετήσει τα καλύτερα χαρακτηριστικά και των δύο κόσμων. Ένα σημαντικό κομμάτι της συσχεδίασης είναι ο διαχωρισμός της εφαρμογής σε υλικό και λογισμικό. Σε απλές εφαρμογές ο διαχωρισμός με το «χέρι» μπορεί να είναι επαρκής. Σε πιο πολύπλοκες εφαρμογές, η εξερεύνηση του χώρου των δυνατών λύσεων είναι περιορισμένη και τα αποτελέσματα είναι απίθανο να είναι τα βέλτιστα. Για τον ανεπτυχθέν σύστημα, μετά από μέτρήσεις του χρόνου εκτέλεσης των σταδίων επεξεργασίας που αναφέρθηκαν νωρίτερα, ο διαχωρισμός με το «χέρι» υιοθετήθηκε λόγω του χαμηλού αριθμού των σταδίων επεξεργασίας που χρησιμοποιούνται στο σύστημα.

Στον Πίνακα 1 δίνεται ο χρόνος εκτέλεσης των διαφόρων σταδίων της επεξεργασίας. Με βάση τα αποτελέσματα αυτά στην Εικόνα 3 δίνεται ο τελικός διαχωρισμός των σταδίων επεξεργασίας σε υλικό και λογισμικό που χρησιμοποιούνται στο τελικό σύστημα.



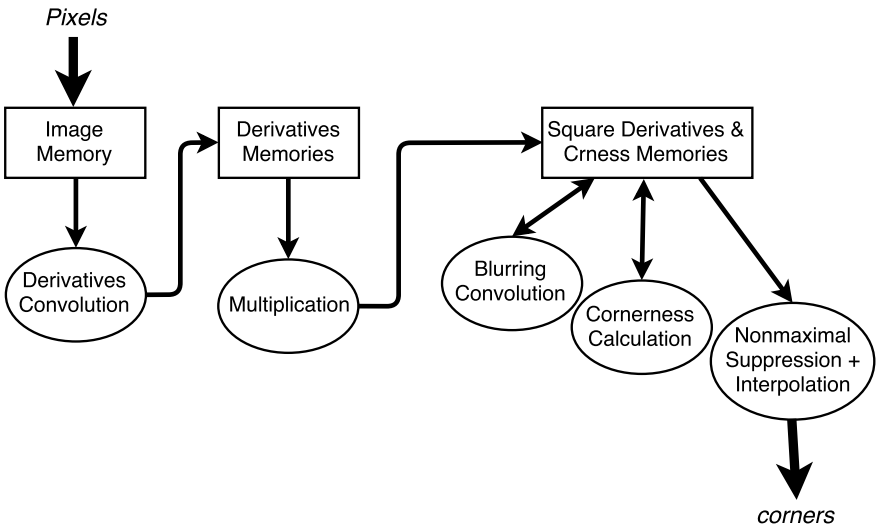
Εικόνα 3: Διαχωρισμός υλικού/λογισμικού του συστήματος που υλοποιήθηκε.

Αλγόριθμος ανίχνευσης γωνιών Harris

Η ανίχνευση γωνιών είναι μια πολύ κοινή λειτουργία σε πολλές εφαρμογές επεξεργασίας εικόνας και ο αλγόριθμος Harris [8] είναι αναμφισβήτητα ο πιο δημοφιλής αλγόριθμος για αυτό το σκοπό. Σε γενικές γραμμές, ο αλγόριθμος Harris δέχεται ως είσοδο μία grayscale εικόνα και ανιχνεύει τα πιο βασικά χαρακτηριστικά της, δηλαδή τις γωνίες (σημεία στην εικόνα που παρουσιάζουν σημαντική τοπική διακύμανση στην ένταση σε όλες τις κατευθύνσεις), οι οποίες μπορούν να ανιχνευτούν επανειλημμένα με αρκετή ακρίβεια κάτω από διαφορετικές συνθήκες (π.χ. φωτισμός, περιστροφή, κλπ). Η δημοφιλία του Harris και οι ποικίλες υπολογιστικές τεχνικές του (π.χ. συνέλιξη, σταθερής και κινητής υποδιαστολής αριθμητική) κάνουν τον Harris ένα ιδιαίτερα αντιπροσωπευτικό πυρήνα υλικού για τους σκοπούς της εργασίας.

Από αλγοριθμική άποψη, για κάθε εικονοστοιχείο, ο αλγόριθμος Harris υπολογίζει την δύναμη μιας γωνίας ("cornerness") σύμφωνα με τον τύπο $\mathrm{I}_x^2\mathrm{I}_y^2 - (\mathrm{I}_x\mathrm{I}_y)^2 - 0.04 \cdot (\mathrm{I}_x^2 + \mathrm{I}_y^2)^2$, όπου I_x^2, I_y^2 και $\mathrm{I}_x\mathrm{I}_y$ αντιπροσωπεύουν τα Gaussian-smoothed γινόμενα των παραγώγων της εικόνας, τα οποία υπολογίζονται μέσω ενός φίλτρου Sobel. Τιμές του cornerness που υπερβαίνουν ένα συγκεκριμένο όριο και οι αντίστοιχες τιμές των υπόλοιπων σε μια γειτονιά 3×3 του υπό εξέταση εικονοστοιχείου (αποτελούν τοπικό μέγιστο), ορίζουν γωνίες στην εικόνα. Αυτές οι γωνίες φιλτράρονται με την εφαρμογή μιας τετραγωνικής επιφάνειας στην προαναφερθείσα 3×3 περιοχή.

Στο FPGA, ο αλγόριθμος Harris επιταχύνεται σύμφωνα με την βαθιάς διοχέτευσης αρχιτεκτονική που φαίνεται στην Εικόνα 4, και η οποία περιγράφηκε σε VHDL. Τα εικονοστοιχεία της εικόνας φορτώνονται στην μνήμη "Image Memory", η οποία τροφοδοτεί τη μονάδα "Derivatives



Εικόνα 4: Μπλοκ διάγραμμα ανιχνευτή γωνιών Harris.

Convolution" με δύο διαδοχικές ριπές της εικόνας (1 εικονοστοιχείο ανά κύκλο) για να υπολογίσει τις παραγώγους της εικόνας dx και dy. Οι παράγωγοι της εικόνας τετραγωνίζονται και στη συνέχεια αποθηκεύονται προσωρινά σε διακριτές θέσεις μνήμης. Κάθε dx^2, dy^2 και $dxdy$ προωθείται μεμονωμένα στην μονάδα "Bluring Convolution" για να παράγου μοναδικά τα I_x^2, I_y^2 and $\mathrm{I}_x\mathrm{I}_y$ (οι προκύπτουσες τιμές αντικαθιστούν τις παλιές παραγώγους στη μνήμη). Στη συνέχεια τα τρία I προωθούνται παράλληλα στην μονάδα "Cornerness Calculator" (1 τριάδα $\langle \mathrm{I}_x^2, \mathrm{I}_y^2, \mathrm{I}_x\mathrm{I}_y \rangle$ ανά κύκλο) για τον υπολογισμό του ανωτέρου μαθηματικού τύπου για κάθε εικονοστοιχείο (ένα cornerness ανά κύκλο, αντικαθιστά την αντιστοιχή του τριάδα I στη μνήμη). Οι τιμές cornerness προωθούνται με ριπή (μία ανά κύκλο) στην τελική μονάδα για να υποστούν non-maximal suppression και interpolation.

Συσταδοποίηση δεδομένων

Η συσταδοποίηση δεδομένων είναι μια διαδικασία ανάθεσης ενός συνόλου δεδομένων σε υποσύνολα (κατηγορίες), που ονομάζονται συστάδες (clusters), έτσι ώστε τα δεδομένα στο ίδιο υποσύνολο να είναι παρόμοια και δεδομένα σε διαφορετικά υποσύνολα να διαφέρουν αρκετά [10]. Η συσταδοποίηση δεδομένων συχνά συγχέεται με την ταξινόμηση, στην οποία τα αντικείμενα εκχωρούνται σε προκαθορισμένες κατηγορίες. Στην συσταδοποίηση δεδομένων, οι κατηγορίες πρέπει να καθοριστούν.

Σε αυτή την εργασία δύο αλγόριθμοι συσταδοποίησης χρησιμοποιούνται και αξιολογούνται ως προς το πόσο καλά εκτελούν το ρόλο τους στη ροή επεξεργασίας της εικόνας που παρουσιάστηκε νωρίτερα. Αυτοί οι αλγόριθμοι είναι ο k-means και ο DBSCAN και μια σύντομη περιγραφή τους δίνεται παρακάτω.

Ο αλγόριθμος k-means

Ο αλγόριθμος για συσταδοποίηση k-means [15] είναι ο πιο ευρέως χρησιμοποιούμενος διαμεριστικός αλγόριθμος συσταδοποίησης. Ξεκινά με την επιλογή k αντιπροσωπευτικών σημείων ως αρχικά κεντροειδή. Κάθε δεδομένο στη συνέχεια αντιστοιχίζεται στο πλησιέστερο κεντροειδές με βάση μία μετρίκη για την απόσταση που έχει επιλεγεί. Μόλις δημιουργηθούν οι συστάδες, τα κεντροειδή για κάθε συστάδα ανανεώνονται. Ο αλγόριθμος στη συνέχεια επαναλαμβάνει αυτά τα δύο βήματα μέχρι τα κενροειδή να μην αλλάζουν ή οποιοδήποτε άλλο εναλλακτικό κριτήριο σύγκλισης να εκπληρωθεί. Ο αλγόριθμος k-means είναι ένας greedy αλγόριθμος που εγγυημένα συγκλίνει σε

τοπικό ελάχιστο.

Ο αλγόριθμος DBSCAN

Ο αλγόριθμος DBSCAN [6] υπολογίζει την πυκνότητα των δεδομένων μετρώντας τον αριθμό των σημείων σε μια γειτονιά σταθερής ακτίνας και θεωρεί δύο σημεία ότι συνδέονται αν βρίσκονται στο εσωτερικό της γειτονιάς του άλλου. Ένα σημείο ονομάζεται σημείο πυρήνας αν στη γειτονιά του ακτίνας eps περιέχονται τουλάχιστον $minPts$ σημεία, δηλαδή η πυκνότητα στη γειτονιά πρέπει να υπερβαίνει κάποιο όριο. Ένα σημείο q είναι άμεσα προσβάσιμο από ένα σημείο πυρήνα p, αν το q είναι εντός της ακτίνας eps του p. Δύο σημεία p και q λέμε ότι συνδέονται κατα πυκνότητα αν υπάρχει ένα τρίτο σημείο o από το οποίο τα p και q είναι άμεσα προσβάσιμα. Μια συστάδα είναι τότε ένα σύνολο σημείων συνδεδεμένα κατά πυκνότητα, η οποία πυκνότητα είναι μέγιστη σε σχέση με την πυκνότητα-προσβασιμότητας. Ο θόρυβος ορίζεται ως το σύνολο των σημείων από το αρχικό σύνολο δεδομένων που δεν ανήκουν σε καμία από τις συστάδες. Η συσταδοποίηση με βάση την πυκνότητα είναι να βρεθούν όλες οι συστάδες με βάση τις παραμέτρους eps και $minPts$ σε ένα συγκεκριμένο σύνολο δεδομένων.

Περιβάλλον λογισμικού

Το λειτουργικό σύστημα αποτελεί έναν σημαντικό παράγοντα που επηρεάζει την απόδοση του όλου συστήματος, καθώς και την προσπάθεια ανάπτυξης που απαιτείται για την εφαρμογή του. Παρά το γεγονός ότι σε ενσωματωμένα συστήματα μια ευρέως χρησιμοποιούμενη προσέγγιση είναι η ανάπτυξη λογισμικού σε bare-metal περιβάλλον, όπου ο προγραμματιστής μπορεί να τρέξει το λογισμικό απευθείας στο υλικό αντί να χρησιμοποιήσει κάποιο λειτουργικό σύστημα. Για εφαρμογές με χαμηλή πολυπλοκότητα αυτό οδηγεί σε πιο ελαφρύς λύσεις και σε εξοικονόμηση πόρων. Ωστόσο, η ανάπτυξη bare-metal εφαρμογών είναι μια επίπονη και χρονοβόρα διαδικασία λόγω της περιορισμένης διαθεσιμότητας σε βιβλιοθήκες λογισμικού και της ανάγκης προγραμματισμού σε χαμηλό επιπέδο των διαφόρων εξαρτημάτων του υλικού και των προγραμμάτων οδήγησης για συγκεκριμένες συσκευές. Ένα άλλο μειονέκτημα είναι ότι σε περιβάλλον bare-metal η εφαρμογή είναι single-threaded. Επιπλέον, δοκιμές και εντοπισμός σφαλμάτων μπορούν να πραγματοποιηθούν μέσω cross-compilation.

Η εναλλακτική λύση είναι να χρησιμοποιηθεί ένα λειτουργικό σύστημα ανοικτού κώδικα το οποίο προσφέρει μια πληθώρα από οφέλη. Τα πιο σημαντικά από αυτά τα οφέλη συνοψίζονται παρακάτω:

- Προσφέρει ένα υψηλό επίπεδο αφαίρεσης του συστήματος μέσα από μια ποικιλία βιβλιοθηκών για το υλικό. Ως εκ τούτου, ο προγραμματιστής μπορεί να επικεντρωθεί στην κύρια λειτουργικότητα της εφαρμογής χωρίς να ανησυχεί για λεπτομέρειες χαμηλού επιπέδου.
- Παρέχει έτοιμους προς χρήση οδηγούς συσκευών για ένα ευρύ φάσμα περιφερειακών.
- Μπορεί να γίνει χρήση multithreading για παράλληλη επεξεργασία.
- Προσφέρει ασφάλεια περιορίζοντας την εφαρμογή σε μία συγκεκριμένη περιοχή της μνήμης και προστατεύει από μη εξουσιοδοτημένες προσβάσεις στο υλικό.

Τα προαναφερθέντα οφέλη μπορούν να μειώσουν το χρόνο ανάπτυξης μιας εφαρμογής σημαντικά και προσφέρουν ένα πιο άνετο περιβάλλον για προγραμματισμό ενσωματωμένων συστημάτων. Με βάση αυτά τα πλεονεκτήματα έγινε χρήση λειτουργικού συστήματος για την γρήγορη ανάπτυξη του λογισμικού. Αρκετά λειτουργικά συστήματα ανοικτού κώδικα μπορούν να βρεθούν διαθέσιμα για SoC FPGAs βασισμένα σε επεξεργαστές ARM (π.χ. το Petalinux της Xilinx, το Linaro Ubuntu κ.λ.π.). Στην παρούσα εργασία χρησιμοποιήθηκε το Xillinux [1], μια διανομή Linux που

βασίζεται στο Ubuntu LTS 12.04 για ARM επεξεργαστές, λύση που ταιριάζει επίσης καλά με τον χρησιμοποιούμενο τρόπο επικοινωνίας PS-PL που περιγράφεται στη συνέχεια.

Υλοποίηση Επικοινωνίας PS-PL

Η επεξεργασία εικόνας απαιτεί υψηλά ποσοστά απόδοσης και χαμηλή καθυστέρηση ανταλαγής δεδομένων μεταξύ της CPU και του επιταχυντή υλικού. Λαμβάνοντας υπόψη αυτές τις απαιτήσεις, ένα πρωτόκολλο συνεχούς ροής (streaming protocol) είναι η πλέον ενδεδειγμένη λύση για τέτοιες εφαρμογές. Σε ένα πρωτόκολλο συνεχούς ροής οι μηχανισμοί χειραψίας (handshake mechanism) περιορίζονται στην αρχή της επικοινωνίας μεταξύ της CPU και του επιταχυντή υλικού και στη συνέχεια, έγκυρα δεδομένα μπορούν να μεταφερθούν για επεξεργασία σε κάθε διαδοχικό κύκλο ρολογιού. Η υλοποίηση μιας αποτελεσματικής επικοινωνίας συνεχούς ροής θα πρέπει να βασίζεται σε ένα μηχανισμό άμεσης προσπέλασης της μνήμης (DMA).

Η λογική DMA επιτρέπει στον επιταχυντή υλικού να έχει άμεση πρόσβαση στη μνήμη του συστήματος αποδεσμεύοντας την CPU από το να εμπλέκεται στην μεταφορά των δεδομένων. Ο μηχανισμός αυτός αυξάνει σημαντικά τη συνολική απόδοση του συστήματος, αφού η CPU μπορεί να συνεχίσει να επεξεργάζεται ταυτόχρονα άλλα δεδομένα. Στην παρούσα εργασία χρησιμοποιήθηκε αυτή η προσέγγιση για τη μεταφορά δεδομένων από το PS στο PL και αντίστροφα. Σε SoC FPGAs βασισμένα σε ARM ένα ευρέως χρησιμοποιούμενο πρωτόκολλο συνεχούς ροής είναι το AXI4-Stream, που βασίζεται στο πρωτόκολλο AXI4 [11]. Παρόλ΄ αυτά, η υλοποίηση του AXI4-Stream αποτελεί μεγάλη σχεδιαστική πρόκληση, λόγω του χρόνου που πρέπει να δαπανηθεί για την παραμετροποίηση των διεπαφών του επιταχυντή σύμφωνα με αυτό το πρωτόκολλο, την επικύρωση της ορθής λειτουργίας και την ενσωμάτωση στο σύστημα.

Όσο αφορά την μεταφορά σημάτων ελέγχου στον επιταχυντή υλικού (π.χ. start, enable, complete, κλπ.), καθώς και για την αρχικοποίηση του μηχανισμού DMA, χρησιμοποιούνται απλούστερα πρωτόκολλα επικοινωνίας. Τέτοια πρωτόκολλα βασίζονται σε ένα απλό μηχανισμό χειραψίας. Μια ευρέως χρησιμοποιούμενη προσέγγιση για SoC FPGAs που βασίζονται σε ARM επεξεργαστές είναι το πρωτόκολλο AXI4-Lite.

Προκειμένου να διευκολυνθεί η υλοποίηση της επικοινωνίας μεταξύ της CPU και του επιταχυντή υλικού έγινε χρήση μίας γενικής λύσης, η οποία μπορεί να ενσωματωθεί στο σύστημα σχετικά γρήγορα και επιπλέον είναι προσαρμόσιμη ώστε να καλύπτει συγκεκριμένες ανάγκες της εκάστοτε εφαρμογής.

Πυρήνας υλικού Xillybus

Το Xillybus είναι μια οικογένεια από πυρήνες υλικού IP, που αναπτύχθηκαν από την εταιρεία Xillybus Ltd. [2], και αποτελείται από δύο πυρήνες που στοχεύουν εφαρμογές με διαφορετικές ανάγκες. Ο πρώτος πυρήνας υλικού, που ονομάζεται Xillybus IP, είναι μια λύση που υιοθετεί το μηχανισμό DMA, και προσφέρει έτοιμες προς χρήση διεπαφές επικοινωνίας συνεχούς ροής μεταξύ της CPU και του επιταχυντή υλικού στο PL. Παρέχει μια σειρά από χαρακτηριστικά για αύξηση της παραγωγικότητας. Αυτά τα χαρακτηριστικά συνοψίζονται στη συνέχεια:

- Ευελιξία όσον αφορά τον αριθμό των διεπαφών προς το PL και την προσαρμογή των παραμέτρων διαμόρφωσης για κάθε ξεχωριστή διεπαφή.
- Συμβατό με διαφορετικούς κατασκευαστές FPGAs (π.χ. Xilinx, Altera) και με διαφορετικά λειτουργικά συστήματα (π.χ. Linux, Windows).
- Εύκολο προγραμματιστικό μοντέλο, που υποστηρίζει διαφορετικές γλώσσες προγραμματισμού (π.χ. C/C++, Java, Python κλπ.).

- Πολύ απλό μοντέλο διασύνδεσης από την πλευρά του PL, χωρίς να απαιτείται επιπλέον προσπάθεια ανάπτυξης για την ενσωμάτωση με το επιταχυντή υλικού.

Η επικοινωνία υλικού/λογισμικού με βάση το Xillybus μπορεί να πραγματοποιηθεί μέσω ενός τετριμμένου προγραμματιστικού μοντέλου. Στον Κώδικα 1 δίνετε ένα ενδεικτικό παράδειγμα επικοινωνίας μέσω Xillybus τόσο για την αποστολή όσο και για τη λήψη δεδομένων από/προς έναν επιταχυντή υλικού στο PL. Οι διεπαφές επικοινωνίας (π.χ. εγγραφής, ανάγνωσης) χειρίζονται ως αρχεία συσκευών (device files). Αξίζει να σημειωθεί ότι υπάρχει η δυνατότητα της ταυτόχρονης εκτέλεσης από την CPU άλλων διεργασιών μεταξύ των λειτουργιών εγγραφής και ανάγνωσης.

Ο δεύτερος πυρήνας υλικού, που ονομάζεται Xillybus-Lite IP, προσφέρει ένα επίπεδο αφαίρεσης στο πρωτόκολλο AXI4-Lite, προσφέροντας εύκολη πρόσβαση σε καταχωρητές ή μονάδες μνήμης που υλοποιούνται στο PL. Μπορεί να γίνει χρήση πολλαπλών πυρήνων και το προγραμματιστικό μοντέλο του είναι εξίσου απλό όσο και του Xillybus IP, με ορισμένες εξαιρέσεις. Για να μπορεί να χρησιμοποιηθεί, η φυσική διεύθυνσή του πρέπει να αντιστοιχηθεί στο χώρο διευθύνσεων της εφαρμογής και η πρόσβαση μέσω αυτού σε πόρους στο PL πραγματοποιείται με αναθέσεις μεταβλητών. Ένα παράδειγμα του προγραμματιστικού μοντέλου του Xillybus-Lite δίνεται στον Κώδικα 2.

```
int write_fd, read_fd;
/* host application data buffer */
unsigned char *buffer;

/* open Xillybus write interface  */
write_fd = open("/dev/
    xillybus_write_device",O_WRONLY);

/* write sizeof(buffer) number of bytes
    */
write(write_fd, buffer, sizeof(buffer));

/* close Xillybus write interface */
close(wrire_fd);
                .
                .
                .
/* open Xillybus read interface */
read_fd = open("/dev/
    xillybus_read_device",O_RDONLY);

/* read sizeof(buffer) number of bytes
    */
read(read_fd, buffer, sizeof(buffer));

/* close Xillybus read interface */
close(read_fd);
```

Κώδικας 1: Προγραμματιστικό μοντέλο για το Xillybus IP.

```
int xillite_fd;
void *map_addr;

/* open Xillybus-Lite interface */
xillite_fd = open("/dev/uio0",O_RDWR);

/* creates user address space for
    mapping of Xillybus-Lite interfacing
    */
map_addr = mmap(NULL, size, PROT_READ |
    PROT_WRITE,MAP_SHARED, xillite_fd,0);

volatile unsigned int *pointer =
    map_addr;

/* write to address */
*pointer = the_value_to_write;

/* read from address */
the_value_read_from_register = *pointer
    ;

/* deletes Xillybus-Lite mapping */
munmap(map_addr, size);

/* close Xillybus-Lite interface */
close(xillite_fd);
```

Κώδικας 2: Προγραμματιστικό μοντέλο για το Xillybus-Lite IP.

Τεχνικές Συγχρονισμού για Ενσωμάτωση Υλικού/Λογισμικού

Τα SoC FPGAs περιλαμβάνουν διάφορους τομείς που αναφέρονται στο PS, PL, και στη μνήμη και λειτουργούν με διαφορετική συχνότητα ρολογιού. Συνήθως, το ρολόι στο PL θεωρείται το ίδιο με αυτό των επιταχυντών υλικού και της υποδομής επικοινωνίας. Το πρόβλημα είναι ότι η υποδομή επικοινωνίας μπορεί να λειτουργήσει αξιόπιστα μέχρι μια συγκεκριμένη συχνότητα και ως εκ τούτου μπορεί να εμποδίσει την αύξηση της απόδοση ενός επιταχυντή, που ενδεχομένως μπορεί

να λειτουργήσει με μεγαλύτερη συχνότητα ρολογιού. Προκειμένου να ξεπεραστεί αυτό το εμπόδιο έχουν προταθεί διάφορες τεχνικές cross-clock domain [4].

Όταν ασχολούμαστε με δεδομένα 1-bit μια ευρέως γνωστή τεχνική που βασίζεται στην χρήση συνχρονιστών με διπλά flip-flops μπορεί να χρησιμοποιηθεί. Αυτή η τεχνική είναι πολύ απλή στο σχεδιασμό, αλλά απαιτεί την εφαρμογή πρόσθετων περιορισμών κατά τη φάση της υλοποίησης και παρουσιάζει αναξιόπιστη συμπεριφορά για υψηλές συχνότητες ρολογιού. Μια άλλη λύση είναι να εμπλουτιστούν οι συνχρονιστές με διπλά flip-flop με ένα μηχανισμό χειραψίας, προκειμένου να μειωθεί η πιθανότητα διάδοσης λανθασμένων δεδομένων, αυξάνοντας όμως την καθυστέρηση διάδοσης των δεδομένων. Παρόλ΄ αυτά, η προσέγγιση αυτή απαιτεί επιπλέον προσπάθεια ανάπτυξης και μπορεί να οδηγήσει σε ανακριβή αποτελέσματα που οφείλονται στη διακύμανση της καθυστέρησης διάδοσης που παρουσιάζει κάθε σήμα του 1-bit όταν αυτά αποτελούν μέρος σημάτων πολλαπλών bit (π.χ. δίαυλος δεδομένων).

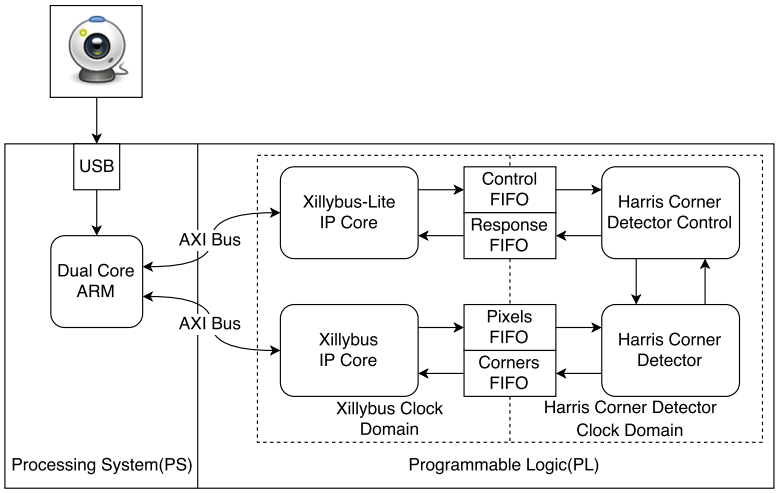
Μια εναλλακτική προσέγγιση για την επίτευξη του συγχρονισμού είναι η χρήση ασύγχρονων FIFO. Το κύριο πλεονέκτημα των ασύνγχρονων FIFO είναι ότι παρέχουν ασύγχρονη ανάγνωση και εγγραφή από περιοχές που λειτουργούν με διαφορετικές συχνότητες ρολογιού. Επιπλέον, η προσέγγιση αυτή παρέχει τα ακόλουθα πλεονεκτήματα:

- Όλοι οι κατασκευαστές FPGA παρέχουν έτοιμους προς χρήση προσαρμόσιμους πυρήνες IP για ασύγχρονες μνήμες FIFO, οι οποίοι έχουν βελτιστοποιηθεί για χρήση στην εκάστοτε συσκευή και η ενσωμάτωσή τους απαιτεί ελάχιστη προσπάθεια.

- Όταν ο επιταχυντής υλικού επεξεργάζεται ένα σύνολο δεδομένων, το επόμενο σύνολο δεδομένων μπορεί να αποθηκευεί προσωρινά και να είναι έτοιμο προς επεξεργασία αμέσως όταν χρειαστεί.

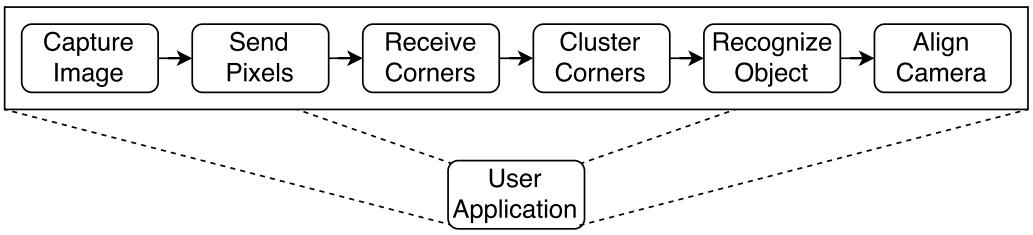
- Μη συμμετρικά πλάτη διαύλων δεδομένων μπορούν να χρησιμοποιηθούν στις διεπαφές εισόδου και εξόδου των μνημών FIFO. Για παράδειγμα πολλαπλά δεδομένα μπορούν να γραφούν παράλληλα και να διαβαστούν σειριακά από τον επιταχυντή υλικού.

Λαμβάνοντας υπόψη τα παραπάνω πλεονεκτήματα, χρησιμοποιήθηκαν ασύγχρονες μνήμες FIFO στο αναπτυχθέν σύστημα μεταξύ του πυρήνα Xillybus IP και του Harris, τόσο για την μεταφορά σημάτων ελέγχου όσο και για τη μεταφορά δεδομένων. Σε γενικές γραμμές αυτή η προσέγγιση διασφαλίζει το σωστό συγχρονισμό του συστήματος με αμελητέα προσπάθεια.

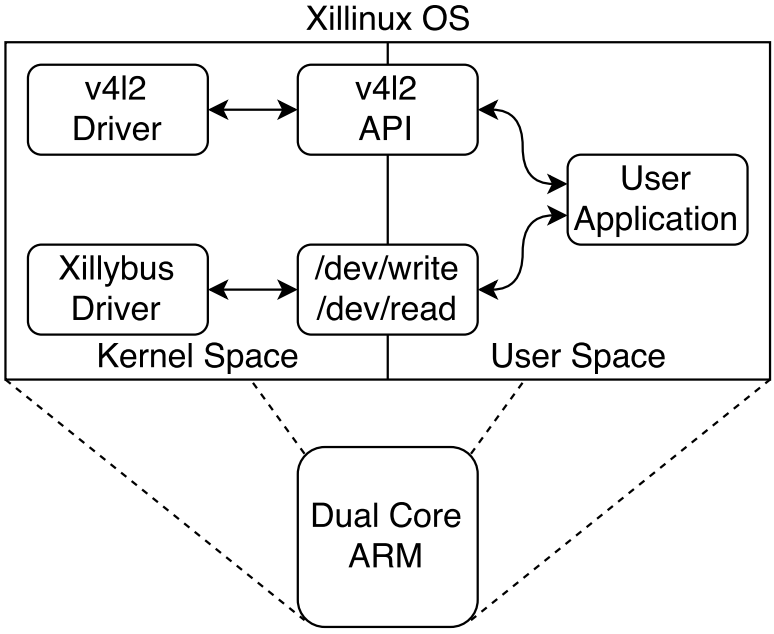


Εικόνα 5: Τελική αρχιτεκτονική υλικού.

Εφαρμόζοντας όλες τις τεχνικές που περιγράφηκαν σε αυτή την ενότητα, η τελική αρχιτεκτονική του συστήματος δίνεται στις Εικόνες 5 και 6. Στην Εικόνα 5 παρουσιάζεται η αρχιτεκτονική του υλικού, όπου φαίνεται πως διασυνδέονται τα στοιχεία υλικού στο PL (π.χ. Harris, Xillybus) μεταξύ



(a) Επισκόπηση εφαρμογής χρήστη.



(b) Επισκόπηση λειτουργικού σύστηματος και λογισμικού.

Εικόνα 6: Τελική αρχιτεκτονική λογισμικού.

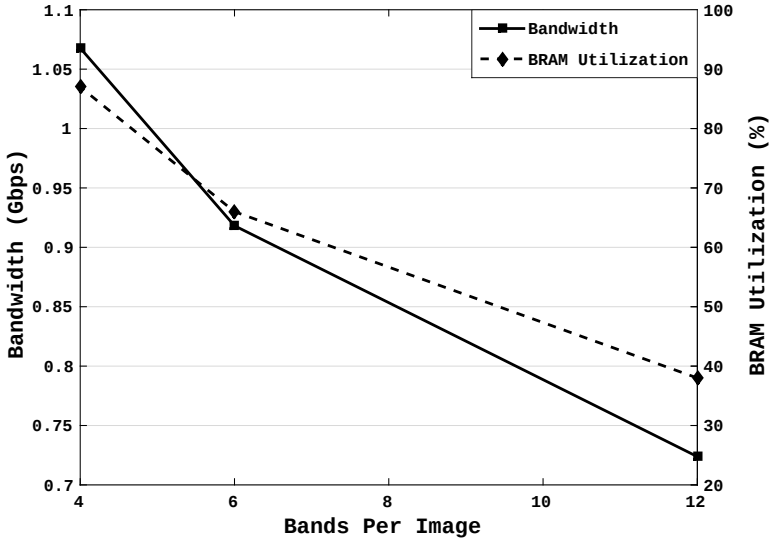
τους αλλά και με το PS, στην Εικόνα 6a δίνεται η ροή εκτέλεσης της εφαρμογής που εκτελεί ο χρήστης, ενώ στην Εικόνα 6b φαίνεται πως η εφαρμογή χρήστη επικοινωνεί με τα απαραίτητα περιφεριακά μέσω του λειτουργικού συστήματος.

Πειραματικά Αποτελέσματα και Ανάλυση Επιδόσεων

Η ενότητα αυτή επικεντρώνεται στην παρουσίαση των επιδόσεων του συστήματός που αναπτύχθηκε με βάση τις χρησιμοποιούμενες τεχνικές που παρουσιάστηκαν νωρίτερα. Η ανάλυση των επιδόσεων γίνεται σε συνάρτηση με τη συχνότητα λειτουργίας του Harris και συγκρίνετε η υλοποίηση στη συσκευή Zynq XC7Z020 SoC FPGA με εναλλακτικές πλατφόρμες επεξεργασίας από πλευράς χρόνου εκτέλεσης, κατανάλωση ισχύος και ενέργειας. Επιπλέον το τελικό σύστημα αξιολογείται ως προς την ποιότητα των αποτελεσμάτων του.

Όλα τα πειράματα διεξήθηκαν με τα ίδια δεδομένα εισόδου – εικόνα μεγέθους 512×384 – τα οποία ήταν προαποθηκευμένα στη μνήμη του συστήματος και ίδιες τιμές για τις παραμέτρους διαμόρφωσης του αλγόριθμου Harris. Επιπλέον, διασφαλίστηκε ότι δεν εκτελούνταν περιττές διεργασίες στο PS. Επειδή το μέγεθος της εικόνας είναι πολύ μεγάλο για να χωρέσει στους πόρους μνήμης του FPGA, η εικόνα χωρίστηκε σε πολλές μικρότερες ζώνες, προκειμένου να επεξεργαστούν διαδοχικά (με επαναχρησιμοποίηση των πόρων του FPGA). Σημειώνεται ότι ο αριθμός των ζωνών έχει σημαντική επίπτωση στο συνολικό χρόνο εκτέλεσης. Πρώτον, οι ζώνες επικαλύπτουν η μία την άλλη για να εξασφαλιστεί λειτουργικά ισοδύναμο αποτελέσμα με τον αρχικό αλγόριθμο, και ως εκ τούτου, η αύξηση του αριθμού των ζωνών εισάγει μεγαλύτερη γενικά καθυστέρηση

στο χρόνο εκτέλεσης. Δεύτερον, όσο μικρότερος είναι ο αριθμός των ζωνών, λιγότερες κλήσεις συστήματος πρέπει να πραγματοποιηθούν για να μεταφερθούν προς το PL τα δεδομένα. Στην



Εικόνα 7: Εύρος ζώνης συστήματος και κατανάλωση πόρων μνήμης στο FPGA συναρτήσει του αριθμού των ζωνών ανα εικόνα χρησιμοποιώντας 16-bit εύρος διαύλου εγγραφής από το Xillybus.

Εικόνα 7 απεικονίζεται η σχέση μεταξύ της χρήσης πόρων μνήμης στο FPGA και του εύρους ζώνης επικοινωνίας συναρτήσει του αριθμού των ζωνών ανα εικόνα. Επιπλέον, μια άλλη παράμετρος που επηρεάζει το εύρος ζώνης της επικοινωνίας είναι το εύρος της διεπαφής εγγραφής των δεδομένων από το Xillybus. Προκειμένου να επιτευχθεί το μέγιστο θεωρητικό εύρος ζώνης των ∼2,4 Gbps που προσφέρει το Xillybus IP, 32-bit εύρος διαύλου για τη μεταφορά των δεδομένων πρέπει να χρησιμοποιηθεί.

Σημειώνεται ότι για τα υπόλοιπα πειραμάτα, γίνεται χρήση 12 ζωνών ανα εικόνα, με εύρος διαύλου επικοινωνίας 16-bit. Με δεδομένη αυτή την υπόθεση, στον Πίνακα 2 δίνεται η συνολική χρήση πόρων και η συχνότητα λειτουργίας του συστήματος, όσο αφορά την πλευρά του PL και η οποία αναλύεται σε δύο μέρη, στον αλγόριθμο Harris (Harris Accelerator) και την επικοινωνία (Xillybus IP Cores). Αυτό που παρατηρείται είναι ότι το κόστος επικοινωνίας σε πόρους είναι

Πίνακας 2: Συχνότητα λειτουργίας και χρήση πόρων του συστήματος αναλύοντας τα στην επικοινωνία Xillybus και την υλοποίηση του Harris.

		Harris Accelerator		**Xillybus IP Cores**	
XC7Z020 PL Resources	Available	Used	Utilization	Used	Utilization
LUTs	53200	11922	22%	2768	5%
DFFs	106400	14038	13%	2656	2%
BRAMs	140	52	37%	1	1%
DSPs	220	54	25%	0	0%
Max. Operating Freq.		300 Mhz		100 Mhz	

σχετικά χαμηλό σε σχέση με το κόστος υλοποίησης του Harris.

Στην Εικόνα 8 απεικονίζεται ο μέσος χρόνος εκτέλεσης του αλγορίθμου Harris στο PL και ο μέσος χρόνος επικοινωνίας PS/PL για την επεξεργασία μιας εικόνας ως συνάρτηση της συχνότητας λειτουργίας του Harris. Ο χρόνος επικοινωνίας αναλύεται σε δύο μέρη, τον χρόνο επικοινωνίας για τον έλεγχο και τον χρόνο επικοινωνίας για τα δεδομένα. Η επικοινωνία ελέγχου αναφέρεται στη χρήση του πυρήνα Xillybus-Lite IP που χρησιμοποιείται για τη μετάδοση των σημάτων ελέγχου,

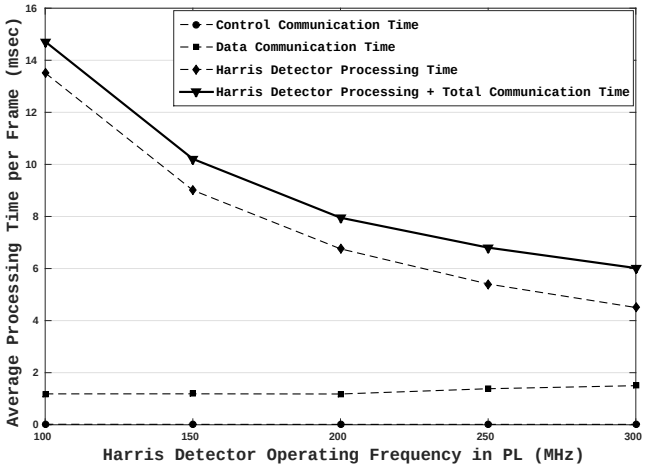
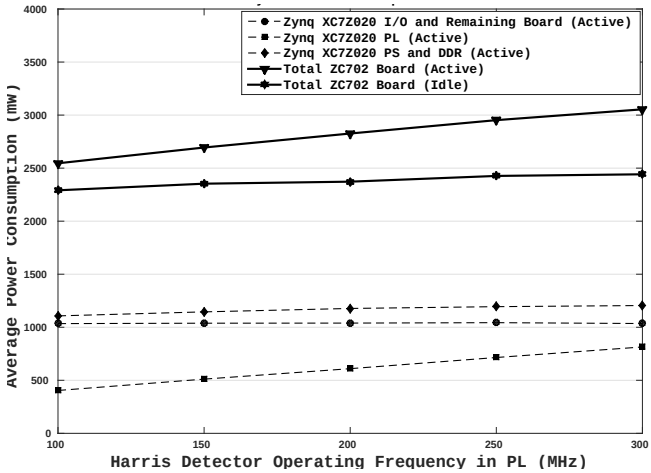


Figure 8: Χρόνος εκτέλεσης εφαρμογής συναρτήσει της συχνότητας λειτουργίας του Harris στο PL.

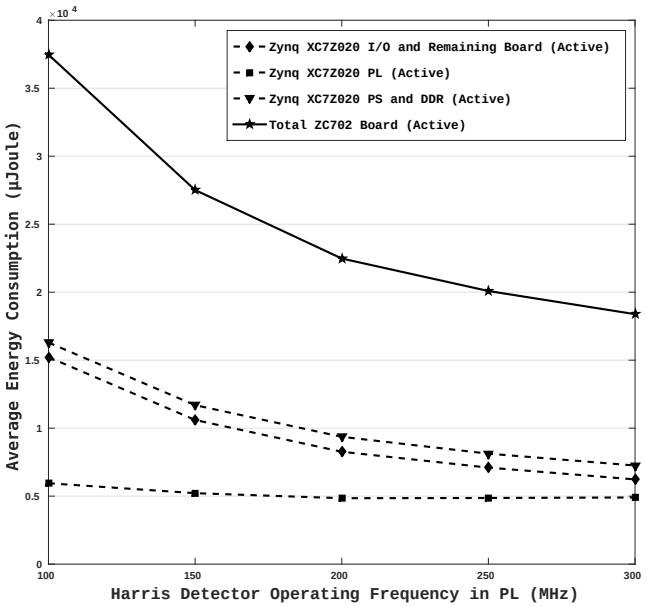
ενώ η επικοινωνία δεδομένων υποδηλώνει τον πυρήνα Xillybus IP για τη μεταφορά δεδομένων. Σημειώνεται ότι ο συνολικός χρόνος επικοινωνίας είναι σχεδόν σταθερός σε όλες τις συχνότητες λειτουργίας του Harris. Αυτό προκύπτει από το γεγονός ότι η επικοινωνία βασίζεται στους πυρήνες Xillybus IP των οποίων η λειτουργία είναι σταθερή στα 100 Mhz. Από την άλλη πλευρά, ο χρόνος εκτέλεσης του Harris μειώνεται καθώς αυξάνεται η συχνότητα του ρολογιού. Λαμβάνοντας υπόψη την Εικόνα 8 αυτό που παρατηρείται είναι ότι το ποσοστό του συνολικού χρόνου επεξεργασίας που δαπανάται στην επικοινωνία αυξάνεται καθώς αυξάνει η συχνότητα λειτουργίας του Harris. Για παράδειγμα, στα 100 Mhz το ποσοστό του χρόνο που δαπανάται στην επικοινωνία είναι 8% του συνολικού χρόνου επεξεργασίας, ενώ στα 300 MHz αυξάνει στο 25%.

Στην Εικόνα 9 παρουσιάζεται η κατανάλωση ισχύος του συστήματος ως συνάρτηση της συχνότητας λειτουργίας του Harris. Μετρήθηκε χωριστά η κατανάλωση ισχύος στο PL, στο PS (μαζί με



Εικόνα 9: Κατανάλωση ισχύος συναρτήσει της συχνότητας λειτουργίας του Harris στο PL.

την μνήμη DDR) και τα περιφερειακά της πλακέτας κατά την επεξεργασία ενός μεγάλου αριθμού εικόνων. Παρουσιάζεται επίσης η συνολική κατανάλωση ισχύος της πλακέτας (PS, PL και I/O) σε κατάσταση αναμονής και εκτέλεσης της εφαρμογής. Αυξάνοντας τη συχνότητα λειτουργίας του Harris παρατηρείται ότι η κατανάλωση ισχύος αυξάνει γραμμικά στο PL ενώ στο PS υπάρχει μόνο μια πολύ μικρή μεταβολή που οφείλεται στην αύξηση του αριθμού των κλήσεων συστήματος που πρέπει να εξυπηρετηθούν στο ίδιο χρονικό διάστημα. Η μικρή αύξηση της ισχύος που παρατηρείται όταν το σύστημα είναι σε κατάσταση αναμονής είναι αποτέλεσμα της στατικής κατανάλωσης ισχύος στο PL. Επίσης παρατηρείται ότι η κατανάλωση ισχύος από τα I/O της συσκευής και τα



Εικόνα 10: Κατανάλωση ενέργειας συναρτήσει της συχνότητας λειτουργίας του Harris στο PL.

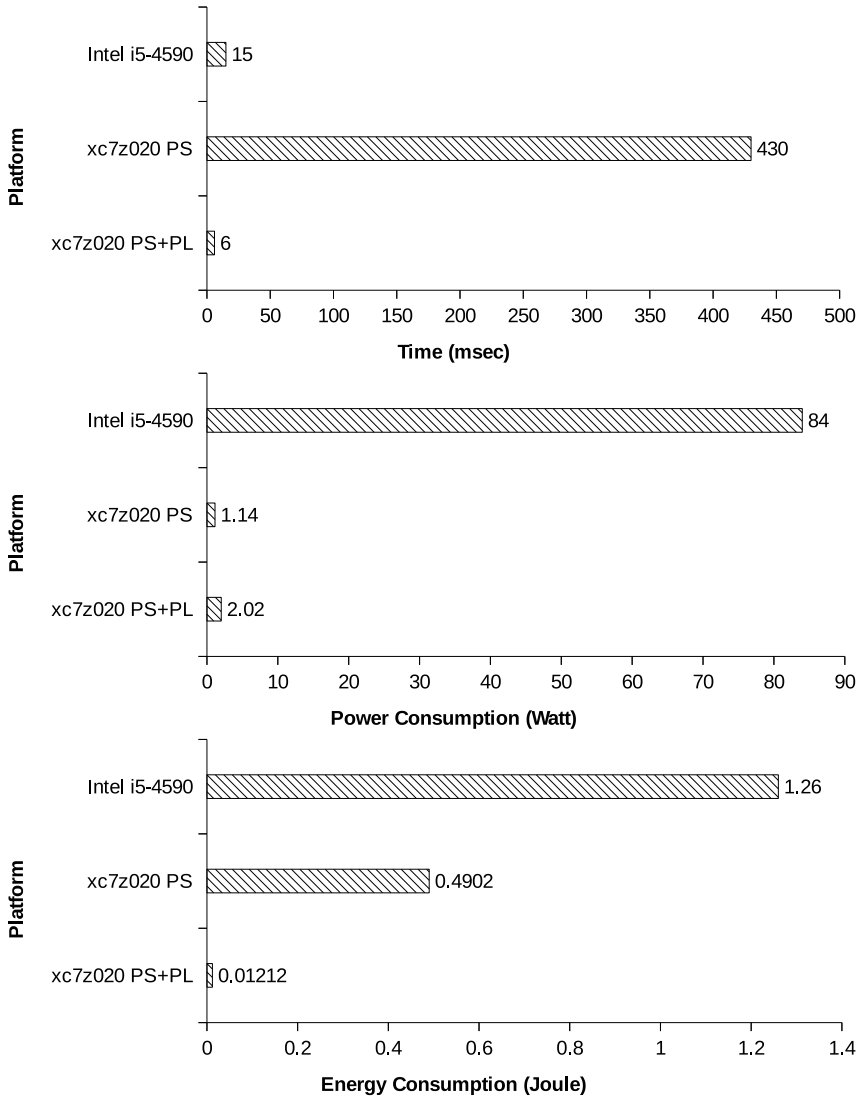
περιφερειακά της πλακέτας παραμένει σταθερή, καθώς δεν χρησιμοποιείται κάποιο από αυτά στο σύστημα που αναπτύχθηκε. Αξίζει να σημειωθεί ότι, ακόμα και αν το σύστημα καταναλώνει περισσότερη ισχύ σε υψηλότερες συχνότητες λειτουργίας του Harris, είναι ενεργειακά πιο αποδοτικό στις υψηλές συχνότητες για τον ίδιο φόρτο εργασίας, δηλαδή όταν εκτελείται η εφαρμογή. Οι μετρήσεις της ενέργειας που δίνονται στην Εικόνα 10 επιβεβαιώνουν αυτή την αρχή.

Για να αποδειχθεί η αποτελεσματικότητα του συστήματος που αναπτύχθηκε, όσο αφορά το χρόνο εκτέλεσης, την κατανάλωση ισχύος και ενέργειας, πραγματοποιήθηκε σύγκριση με εναλλακτικές πλατφόρμες επεξεργασίας. Η Εικόνα 11 παρουσιάζει τη σύγκριση των επιδόσεων της υλοποίησης σε SoC FPGA με υλοποιήσεις καθαρά σε λογισμικό που εκτελούνται σε επεξεργαστή ARM Cortex-A9 (μόνο PS) και επεξεργαστή Intel i5-4590. Με βάση αυτές τις μετρήσεις στον Πίνακα 3 παρουσιάζονται τα κέρδη σε απόδοση του συστήματος σε SoC FPGA έναντι των δύο εναλλακτικών πλατφορμών. Μπορεί να φανεί καθαρά ότι η SoC υλοποίηση είναι ανώτερη από άποψη απόδοσης/watt. Σημειώνεται ότι αν και η υλοποίηση σε ARM (XC7Z020 PS) καταναλώνει λιγότερη ισχύ η υλοποίηση του συστήματος σε SoC FPGA είναι πιο ενεργειακά αποδοτική.

Πίνακας 3: Κέρδος απόδοσης της υλοποίησης SoC FPGA με εναλλακτικές πλατφόρμες επεξεργασίας

	XC7Z020 PS	**Intel i5-4590**	
Execution Time	71.6×	2.5×	
Power Consumption	0.56×	41.6×	**XC7Z020 PS+PL**
Energy Consumption	40.4×	104×	

Για να ολοκληρωθεί η αξιολόγηση, το σύστημα δοκιμάστηκε σε ένα πραγματικό σενάριο όπου το σύστημα παρακολουθούσε ενα στατικό αντικειμένο ορθογωνίου σχήματος και προσπαθούσε να ευθυγραμμίσει την κάμερα με το αντικείμενο. Ο σκοπός αυτού του πειράματος ήταν να δείξει πόσο καλά το τελικό σύστημα εκτελεί τη λειτουργία του υλοποιώντας τη ροή επεξεργασίας που παρουσιάστηκε νωρίτερα. Για το σκοπό αυτό με την κάμερα ελήφθησαν 1000 διαδοχικές εικόνες προκειμένου να τροφοδοτήσουν το σύστημα επεξεργασίας. Το πρώτο μέτρο αξιολόγησης ήταν να μετρηθεί ο αριθμός των εικόνων στις οποίες το σύστημα ήταν σε θέση να αναγνωρίσει το αντικείμενο. Ταυτόχρονα, όταν το σύστημα ήταν σε θέση να αναγνωρίσει το αντικείμενο σε τρεις διαδοχικές εικόνες το αποτέλεσμα ερμηνευόταν ως επιτυχής ευθυγράμμιση με το αντικείμενο και



Εικόνα 11: Σύγκριση των επιδόσεων της υλοποίησης SoC FPGA με εναλλακτικές πλατφόρμες επεξεργασίας

καταγραφόταν επίσης. Με βάση αυτές τις μετρήσεις, η ποιότητα των αλγορίθμων αξιολογήθηκε με τον υπολογισμό του ποσοστού επιτυχών ευθυγραμμίσεων, δηλαδή πόσες ομάδες των τριών διαδοχικών εικόνων μετρήθηκαν από το μέγιστο δυνατό των 333.

Ο Πίνακας 4 παρουσιάζει τις μετρήσεις που ελήφθησαν όταν η κάμερα μετακινούνταν στον τρισδιάστατο καρτεσιανό χώρο και επιτρεπόταν ή όχι να περιστρέφεται γύρω από τον οπτικό της άξονα. Όπως μπορεί να φανεί από τον πίνακα, όταν η κάμερα επιτρεπόταν να περιστραφεί και ο αλγόριθμος k-means χρησιμοποιούταν για τη συσταδοποίηση των γωνιών που έβρισκε ο Harris, μόνο σε ένα πολύ μικρό αριθμό εικόνων, το σύστημα ήταν σε θέση να αναγνωρίσει και να ευθυγραμμιστεί με το αντικείμενο, έχοντας ποσοστό επιτυχίας μόνο 5%. Από την άλλη πλευρά, όταν χρησιμοποιήθηκε ο αλγόριθμος DBSCAN το ποσοστό επιτυχίας του συστήματος έφτασε το 75%. Αυτό το τεράστιο χάσμα μπορεί να δικαιολογηθεί από το γεγονός ότι ο k-means δεν μπορεί να διακρίνει μεταξύ θορύβου (outliers) και χρήσιμων δεδομένων, ένα χαρακτηριστικό που είναι ενσωματωμένο στην λογική του αλγορίθμου DBSCAN. Τέλος όταν δεν επιτρεπόταν περιστροφή της κάμερας περι τον οπτικό άξονα, ο k-means βελτίωσε σημαντικά την ακρίβεια του συστήματος, αλλά όχι αρκετά ώστε να θεωρηθεί ως μια καλή επιλογή για χρήση ακόμη και για αυτή την ειδική περίπτωση. Σε αντίθεση με την βελτίωση του k-means, ο αλγόριθμος DBSCAN αύξησε το ποσοστό επιτυχίας του $\sim$2%.

Πίνακας 4: Ποιότητα αλγορίθμων υλοποιημένου συστήματος.

	Processing pipeline with k-means		**Processing pipeline with DBSCAN**	
	Dataset 1	Dataset 2	Dataset 1	Dataset 2
Rotation	✓	✗	✓	✗
Num. of recognition	127	623	832	863
Successful alignment	17	175	250	256
Succes rate	5.1%	52.8%	75.1%	76.9%

Chapter 1

Introduction

1.1 Image Processing in Embedded Systems

Image processing is a rapidly growing area of computer science. Its growth has been fueled by technological advances in digital imaging, computer processors and mass storage devices. Fields which traditionally used analog imaging are now switching to digital systems, for their flexibility and affordability. Important examples are medicine, film and video production, photography, remote sensing and security monitoring. These and other sources produce huge volumes of digital image data every day, more than could ever be examined manually.

Image processing can be defined as the manipulation of an image for the purpose of either extracting information from the image or producing an alternative representation of the image. These operations can be grouped according to the type of data that they process as presented in Figure 1.1.

At the lowest level of the image processing pyramid are those operations which deal directly with the raw pixel values and can be though as prepossessing steps. Such operations includes distortion correction, contrast enhancement and filtering for noise reduction or edge detection etc. At the middle are those algorithms which utilize results obtained from the low level processing. Operations at this level can be grouped in two main classes named *Segmentation* and *Classification*. Segmentation operations such as thresholding, color detection, region growing and connected components labeling occur at the boundary between the low and intermediate levels. The purpose of segmentation is to detect objects or regions in an image, which have some common property. Classification operations use features of each region to identify objects or parts of objects, or to classify an object into one of several predefined categories. Classification transforms the data from regions to features, and then to labels. The data is no longer image

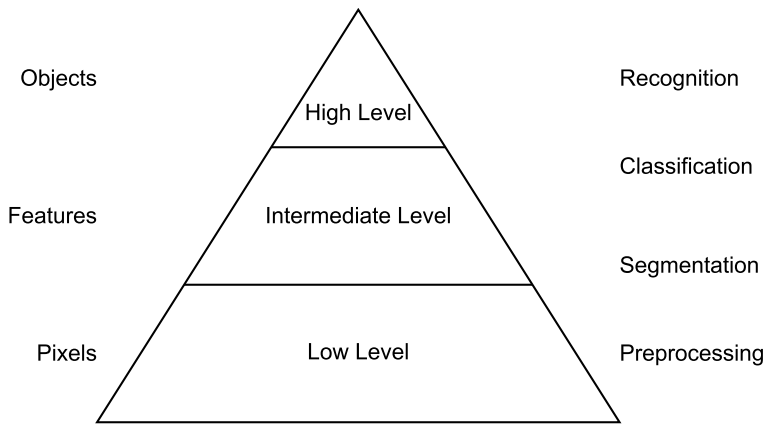
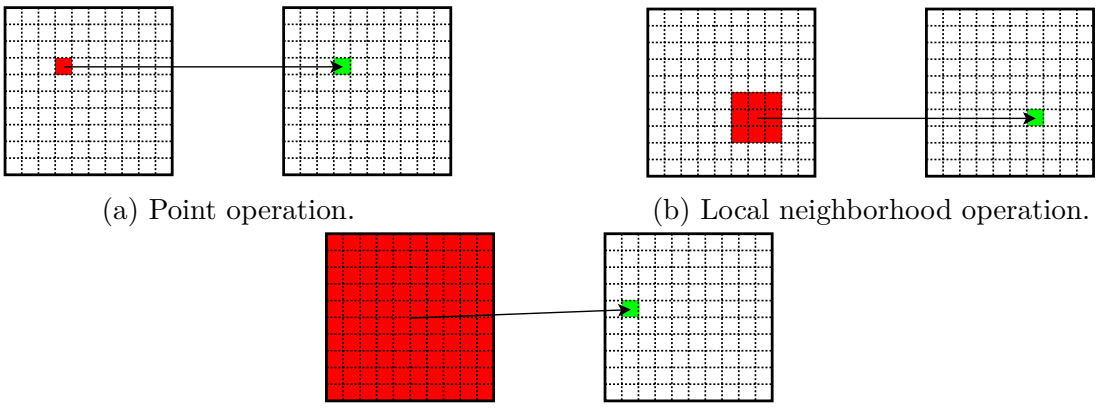


Figure 1.1: Image processing pyramid.



(a) Point operation.

(b) Local neighborhood operation.

(c) Global operation.

Figure 1.2: Various types of image operations.

based, but position information may be contained within the features, or be associated with the labels. At the highest level are those methods which attempt to extract semantic meaning from the information provided by the lower levels.

Figure 1.2 presents another way to categorize image processing operations based on how many input data must be processed to obtain a result or in other words the computational overhead associated with a given operation. These categories are

- Point operations : The output value at a specific coordinate depends only on the input value at that same coordinate. These operations have constant computation time as they just apply a simple transformation to the input data (Figure 1.2a).
- Local neighborhood operations : The output value at a specific coordinate depends not only on the input value of that specific coordinate, but also on the values on the neighborhood of that same coordinate. These operations have computation time proportional to the size of the region being processed (Figure 1.2b).
- Global operations : The output value at a specific coordinate depends on all the values in the input image. These operations have computation time proportional to the size of the image being processed (Figure 1.2c).

An embedded system is a computational unit embedded in electronic devices combining computer hardware and software. They are getting more and more common in cars, cameras, and even on washing machines and fridges. Some examples are auto-focus cameras, battery chargers, cell phones, temperature controllers etc. They are specifically designed and optimized for a particular function or a range of functions, to achieve shorter startup times, higher processing speed, greater reliability, low cost, low power consumption and/or some other property that a general-purpose computing system may not fulfill.

Embedded systems are particularly well suited to process data streams at high speeds with fairly small programs, that is why recent years have witnessed a dramatic increase in the use of embedded systems to run image processing applications. Other than on CPUs, data can be processed in a highly parallel fashion, with high speeds and low power requirements.

1.2 FPGAs in Image Processing

An FPGA based design is inherently parallel in nature. Different algorithm sequences will be mapped to different hardware modules in a FPGA, which operates concurrently. The main reasons for choosing FPGA as an embedded image processing platform are given below:

1. **Parallel operation** : FPGAs is a collection of logic elements which can be electrically re-wired. FPGAs implement an application by developing separate hardware for each functionality and hence such designs are inherently parallel. Each instruction that the programmer enters will be mapped into a separate hardware component. Thus, such a design is suitable in those image processing algorithms, which has significant amount of parallelism in them.

2. **Execution speed** : Due to parallel nature of FPGA's, the execution speed will be considerably increased. In practical applications, the image will be partitioned into different sub-blocks and then each block will be processed in parallel.

3. **Flexibility** : An FPGA based system provides full programming flexibility. Current FPGAs have sufficient logic resources to implement even complex applications in a single chip. Modern FPGA based systems will adaptively reconfigure according to the different operating environments. Hence FPGA based systems are inherently flexible.

4. **Low power design** : An FPGA based circuit implements several operations in one clock cycle simultaneously. This allow clock speed to be lowered significantly. In fact there will be a reduction in clock speed over a serial processor of the magnitude of 2 or more. Reduction in clock speed corresponds to reduction in dynamic power consumption of the system. Thus FPGA based design facilitates a low power design.

1.3 Thesis Goals and Organization

Nowadays the ever-increasing advancements in technology has led to the deployment of more complex and computationally intensive image processing algorithms. Many of these algorithms have been adopted in present-day embedded systems targeting a variety of applications such as automotive, 3D navigation, surveillance, etc. However in real-time embedded systems, where latency and power play an important role, software-oriented implementations running on general purpose CPUs may not offer satisfactory solutions. This stems from the fact that CPUs have limited parallel processing capabilities to support the performance requirements of these applications and consume considerable power. In order to overcome these downsides and increase the performance/watt, various approaches have been proposed where dedicated hardware is deployed alongside the CPU to accelerate the critical parts of the task or even the entire algorithm. These approaches base on various system-level combinations, such as CPU-DSP, CPU-GPU and CPU-FPGA. However, these kind of systems suffer from performance overhead arising from the increased communication delays between the different components. Another alternative solution is the fabrication of custom ASIC. However, this approach is considered inefficient in terms of time-to-market and development cost.

The primary goal of this thesis it to design and implement an image processing system for embedded applications and its deployment on a System-on-Chip (SoC) platform, specifically a SoC FPGA. As a case study the identification and tracking of a rectangular shaped object is selected. More precisely the system must be able to process images coming from a camera, recognize the predefined shape of the object and inform the user how to move the camera in order to keep tracking the object until it has focused in predefined matter. A key element of this thesis is the study of the image processing steps and the identification of critical steps that significantly affect the processing time. By applying hardware/software codesign methodologies individual parts are implemented as hardware components described using VHDL and the rest developed as software components using the C programming language. At the same time this thesis can be considered as a set of codesign guidelines, which aim to mitigate the development

process, increase productivity and decrease time-to-market when deploying image processing applications on SoC FPGA. More specifically:

1. It exhibits the benefits of using an open-source operating system on the embedded processor.
2. Describes the employment of ready-to-use communication approaches between the HW and SW components.
3. Presents synchronization techniques that guarantee the correct operation of the system.

Finally the system is evaluated based on measuring the success rate of the overall operation and on measurements related to the platform used, such as power consumption, resource utilization, execution time etc. The remainder of this thesis is organized as follows:

- Chapter 2 presents SoC FPGAs, as well as the SoC FPGA platform used in this thesis.
- Chapter 3 presents the image processing steps used in the system and how these steps are mapped to hardware and software. Also presents some theoretical and technical background on key features of the system.
- Chapter 4 discusses and compares different software environments for developing applications running on the CPU of the SoC platform.
- Chapter 5 discusses synchronization techniques that can be applied for the correct integration of the different components of the system and presents the solution adopted in this thesis.
- Chapter 6 presents the evaluation of the developed system .
- Chapter 7 concludes this thesis giving an overview of this thesis as well as future work that can be done.

Chapter 2

Hardware Platform

2.1 FPGAs

Invented back in the 1980's an FPGA device is comprised of reprogrammable hardware. Conventional microchips and Central Processing Units (CPUs) are static and their internal structure cannot be changed. They are specifically manufactured to process software as efficient and fast as possible, but always one single instruction at a time. An FPGA on the other hand, contains many Configurable Logic Blocks (CLBs) that can be connected to each other in different ways as defined by an HDL description of the desirable operation. This gives the freedom to process many input data in parallel with the possibility of speeding up computation time by orders of magnitude. If the architecture is faulty or outdated, it can simply be reprogrammed, while a static chip such as an Application Specific Integrated Circuit (ASIC) would have to be replaced completely.

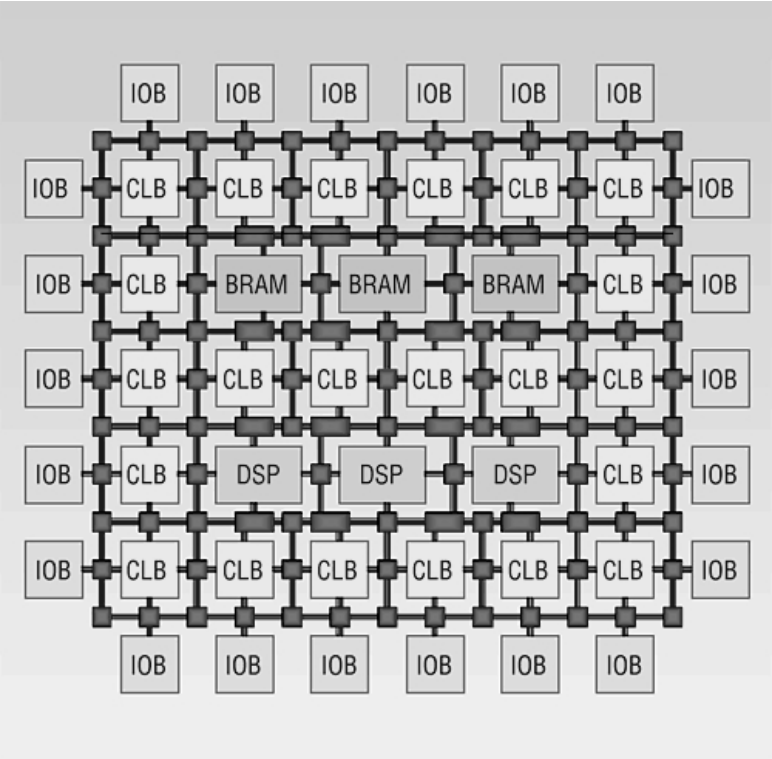


Figure 2.1: General FPGA architecture alternative.

The internals of an FPGA are made up of many instances of a couple of logic units such as multiplexers, Flip-Flops (FFs), Look-Up Tables (LUTs), and memory blocks called BRAM.

In later years Digital Signal Processors (DSPs) where added that can assist in floating point operations. The HDL languages (e.g. VHDL, Verilog) provide a way for the user to specify the behavior of the system. The code is then used to synthesize an actual digital circuit that is implemented using FPGA resources to determine which of its resources are connected to which. Figure 2.1 shows the general layout of an FPGA, where FFs, LUTs and multiplexers are grouped together to form the Configurable Logic Blocks(CLBs).

2.2 SoC FPGAs

In the past, the term System-on-Chip (SoC) has usually referred to an Application Specific Integrated Circuit (ASIC), which can include digital, analogue and radio frequency components, together with mixed signal blocks for implementing analogue-to-digital and digital-to-analogue converters (ADCs and DACs). Around 2010 Xilinx presented the Zynq-7000 All Programmable SoC family of devices, the first SoC device that combined the features of a Dual-Core ARM Cortex A9 Processing System (PS subsystem) with Programmable Logic (PL subsystem), or in other words a dual-core processor with an FPGA core. Although dedicated processors have been coupled with FPGAs before, it has never been quite the same proposition.

In Zynq, the ARM Cortex-A9 is an application grade processor, capable of running full operating systems such as Linux, while the programmable logic is based on Xilinx 7-series FPGA architecture. The integration of an ARM-based processor with an FPGA in a single device offers developers the capability of applying a hardware/software unified approach to embedded system designs and allows levels of performance that two-chip solutions (e.g. CPU externally connected with an FPGA) cannot match due to their limited I/O bandwidth, latency and power consumption. Xilinx has adopted the AMBA bus with AXI protocol/interface as the primary means of communication between the PS and the modules implemented in PL as shown in Figure 2.2 where the architecture of the Zynq SoC devices is presented.

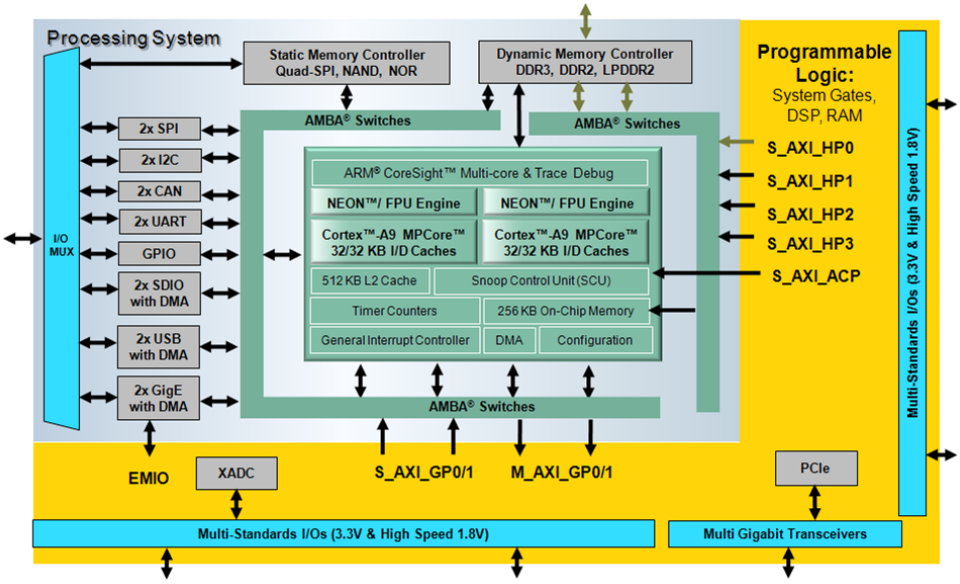


Figure 2.2: Zynq SoC architecture.

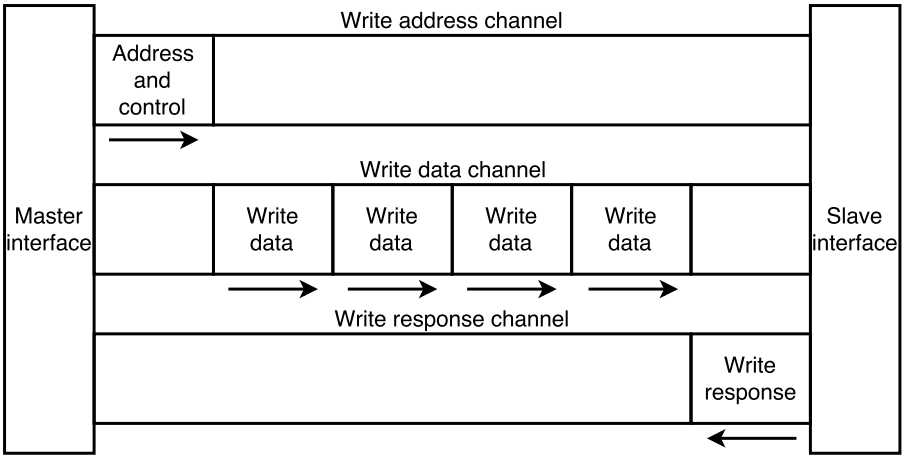
The AMBA AXI Protocol

AXI is a protocol belonging to the ARM AMBA family of microcontroller buses. The AMBA protocol is an open standard on-chip interconnect specification, allowing the connection and

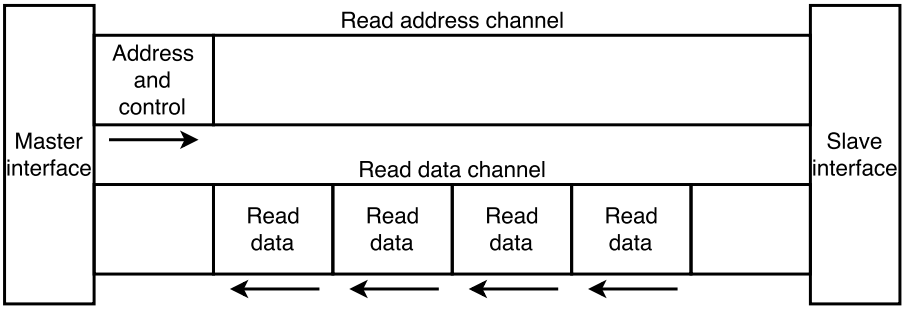
management of many controllers and peripherals in a multi-master design. Much like the requirements for the rest of the AMBA family it is targeted at high-performance, high-frequency system designs. The AXI protocol is optimised for FPGA implementation through coordinated development with Xilinx and is used as a means of communication between IP cores of an FPGA design. The AXI protocol in particular exhibits the following key features[axi spec.]:

- address/control phases are separate from data phases
- byte strobes enable unaligned data transfers
- burst-based transactions possible with only start address issued
- read and write data channels are separate allowing low-cost Direct Memory Access (DMA)
- multiple outstanding addresses can be issued
- transactions can be completed out-of-order

The AXI protocol features burst-based transactions, with each containing address and control information on the address channel. Several AXI masters can be connected to several AXI slaves through an AXI interconnect. The master issues all the commands and control signals and the slave must respond accordingly. The AXI protocol specification defines five different channels: 1. Write address 2. Write data 3. Write response 4. Read address and 5. Read data. Figures 2.3 demonstrates communication between an AXI master and slave.



(a) Write interface.



(b) Read interface.

Figure 2.3: AXI4 channels architecture.

There exists three types of AXI interface, each suited to a different nature of application[14]:

- **AXI4 (or AXI4-Full)**: The high-performance interface, suited for memory mapped communication allowing bursts of up to 256 data transfer cycles per address issued
- **AXI4-Lite**: A light-weight variant of the interface, used for memory mapped single transactions. This variant has the benefit of a smaller logic footprint with a simplified

interface. This variation does not support burst data and so only provides a single data transfer per transaction

- **AXI4-Stream**: No address phase means this is not memory mapped and allows for an unlimited data burst size. A single channel is defined for the transmission of streaming data, modeled after the Write data channel in Figure 2.3a but allowing bursts of an unlimited amount of data. Connection is from master to slave only, so if bidirectional transfers are required both peripherals must be of type master/slave.

ZedBoard developmnet board

In this thesis the Zedboard developemnt boad was used to develop the image processing system. The ZedBoard is a low cost, community-based board which features the XC7Z020 Zynq device. It is a joint venture between Xilinx, Avnet and Digilent. The XC7Z020 Zynq device is one of the smaller devices in the Zynq-7000 family and it is based on the Artix-7 logic fabric, with a capacity of 13,300 logic slices, 220 DSP48E1s and 140 BlockRAMs. The device also contains an XADC hard IP block, although it does not feature high-speed transceivers or PCIExpress blocks. ZedBoard features a variety of peripherals such as:

- General purpose I/O (9 leds, 8 switches, 7 push buttons)
- Audio codec (supports line in, line out, microphone and headphone)
- Video (VGA and/or HDMI)
- OLED display
- PMOD interfaces (5 total)
- Ethernet
- USB-OTG for connecting USB peripherals(ex. keyboard, mouse, camera)
- USB-JTAG for programming purposes and debugging
- USB-UART for serial communication with a host computer
- SD card slot
- FMC interface
- XADC header
- Xilinx JTAG header

In addition the Zynq device interfaces to a 256Mbit flash memory and 512MB DDR3 memory, both of which are found on the board. There are also two oscillator clock sources, one at 100MHz connected to the PL subsystem, and the other at 33.3333MHz connected to the PS subsystem. The block diagram of Zedboard is shown in Figure 2.4 and the board itself in Figure 2.5

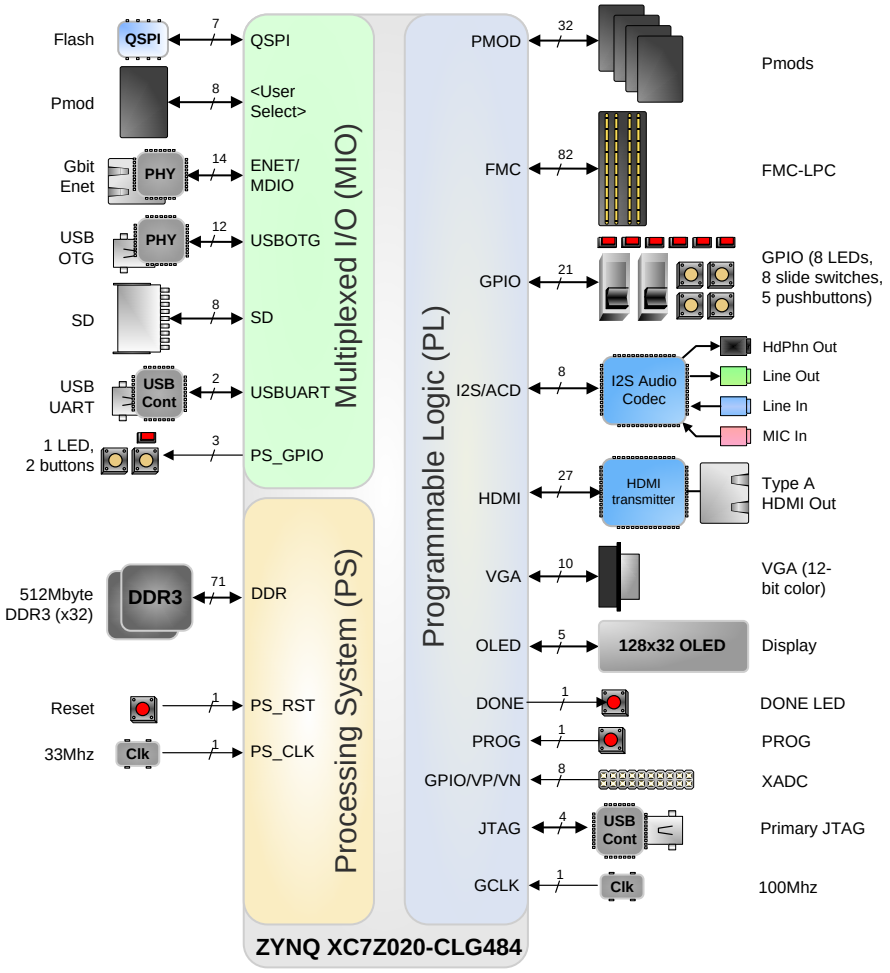


Figure 2.4: ZedBoard block diagram.

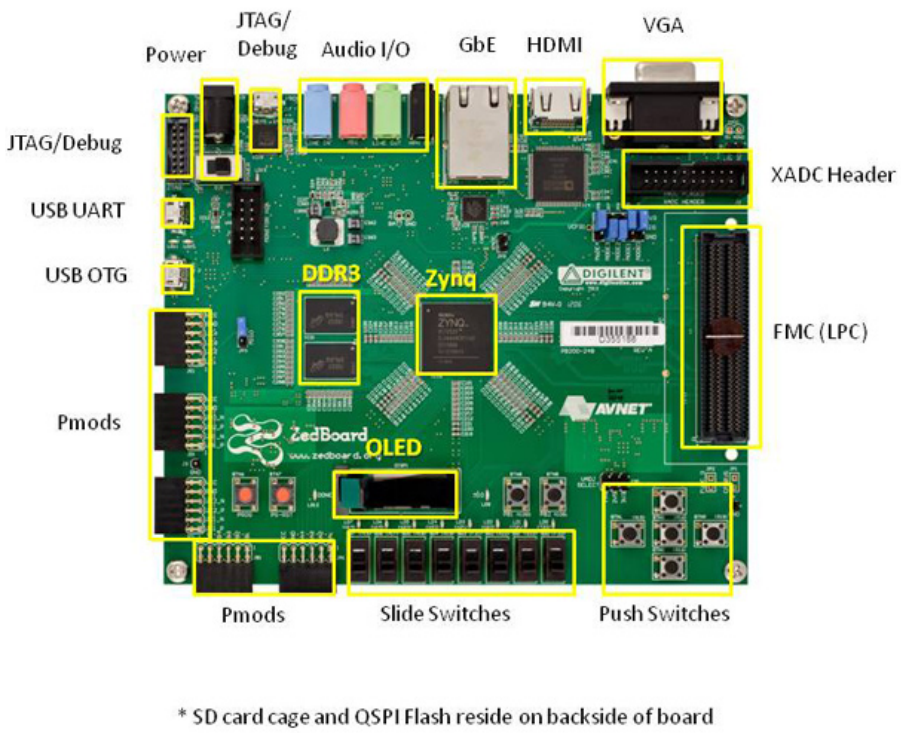


Figure 2.5: The ZedBoard development board.

Chapter 3

System Description

3.1 Processing Flow

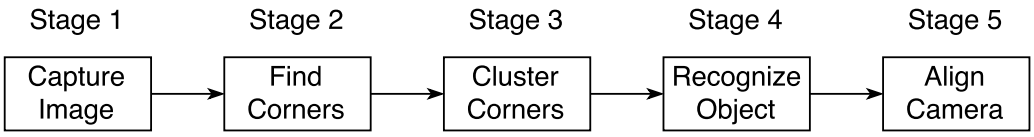


Figure 3.1: Processing flow of the developed system.

The implementation of the system is based on a six stage processing pipeline as shown in Figure 3.1. The stages are described below:

- The 1st stage refers to image capturing by means of a camera connected to the platform. The image is temporally stored in a frame buffer allocated in the main memory of the system ready to be processed.
- The 2nd stage is the processing of the image to find corners, using the well know Harris Corner Detector algorithm. Here, the stored image becomes input to Harris, which in turns outputs the corners found by means of pixel coordinates and the strength ("cornerners") of the corner response. The results from this stage is stored in their own buffer in memory.
- The 3rd stage is where the results obtained from the processing stage (Stage 2) are processed by a clustering algorithm. The output of this stage is a number of clusters (groups). At the end of this stage each corner found is categorized into one of this clusters and a representative point (centroid) is computing for each cluster. For this stage two clustering algorithms are tested, namely k-means and DBSCAN, which will be discussed later.
- The 4th stage computes object's geometry based on the centroids computed in Stage 3 and compares it with a predefined abstract model of the object being tracked.
- Finally during the 5th stage if the detection of the object is successful, the system commands how to move the camera in order to be aligned with the object.

3.2 Hardware/Software Codesign

Hardware/software codesign refers to the simultaneous design of both software and hardware in a system. Software is usually executed in processing elements, such as, Central Processing

Units (CPU) and Digital Signal Processors (DSP). Hardware logic is commonly implemented in Application-Specific Integration Circuit (ASIC) or FPGA. Since software and hardware systems have different characteristics, a combined system has the potential to take the best of both worlds. In [13] the author analyzes the major properties of the hardware/software codesign: coordination, concurrency, correctness, and complexity. Hardware/software codesign has largely evolved since the first time it was used in electric systems. In the first generation of codesign, partitioning was the main issue to be solved. Two approaches were proposed:

- Start with a hardware-only system and move functionality to software, with respect to system constraints, and end up with a codesigned system [7].
- Start from a pure software implementation of the system and then migrate software functions to hardware blocks in order to meet systems constraints [5].

In the second generation, multicore and multiprocessor have been utilized and thus, instead of a single thread, multithreads were used. Thread scheduling became one of the main challenges for the area. Moreover, hardware/software interface and communication are critical since they dramatically affect system performance and design space. According to Teich [13], the codesign is now in its third generation, where the time-to-market cycles are shortened by optimizing the hardware/software development flow (Figure 3.2).

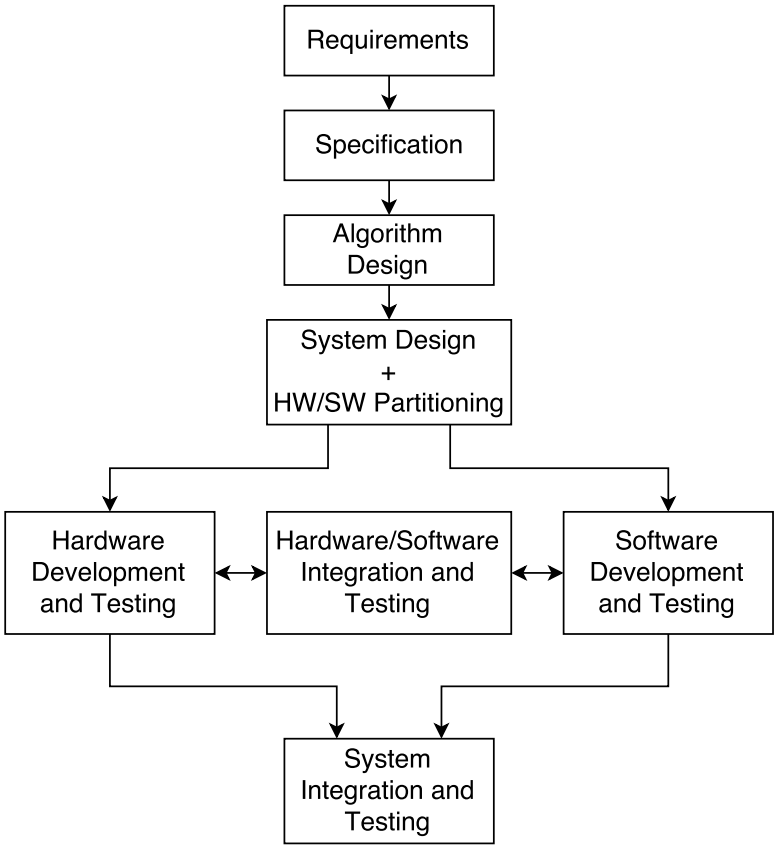


Figure 3.2: The modern approach on hardware/software codesign.

Moreover, languages for hardware/software codesign are necessary. The ideal language would be suitable for both hardware and software development. It seems though that neither Hardware Description Languages (HDLs) nor programming languages (e.g. C/C++, Python etc) can replace the other, even though a lot of time invested on designing new tools and frameworks to mitigate development effort and combine the two different approaches. A good example of these efforts are the High-Level Synthesis Framework (HLS) [3] based on C programming language and model based design framework such as MATLAB Simulink. Unfortunately,

there are significant differences between hardware and software that pose significant challenges to describing a hardware implementation using a software language. Although both of these frameworks can be used to get a first evaluation of the processing algorithm.

In hardware/software codesign the application is initially modeled in an abstract form, without committing tasks to either software or hardware. This enables the designer to concentrate on algorithmic development rather than implementation details. Higher degrees of abstraction enable models to be developed and simulated more quickly. Common to all modeling methods is the process of refinement from abstract behavioral model to implementation, during which the system must be partitioned and each part assigned to either hardware or software. In codesign, the initial modeling of inter-task communication is abstract. It is advantageous if abstraction can be retained during task implementation. This reduces the effort of swapping tasks between hardware and software at late stages of the design cycle.

As mentioned, a significant design task in codesign is the partitioning of the application into software and hardware. In simple applications manual partitioning may be sufficient. In more complex systems, the amount of design space exploration possible would be limited, and the results unlikely to be optimal. For the developed system, after profiling the processing steps mentioned earlier, the manual partitioning approach is adopted due to the low number of processing steps used in the system.

3.2.1 System Partitioning

In table 3.1 the results obtained from profiling the different processing steps after implementing the whole processing flow as software components are given.

Table 3.1: Profiling results for the processing steps of the developed system

Capture Image	Find Corners (Harris)	Cluster Corners		Recognize Object	Align Camera
		k-means	DBSCAN		
$\ll 1$ msec	~ 430 msec	$\ll 1$ msec	$\ll 1$ msec	$\ll 1$ msec	$\ll 1$ msec

- **Capture Image**: Capturing images is performed using a USB camera, which is connected to the system using the PS I/O interface. Using this approach for image capturing makes this task more easy to implement on software as there is a plethora of libraries that can be used to speed up the development. On the other hand if the capturing was implemented on hardware, extra IP cores should have been used as well as software drivers to control these IPs. This second approach adds extra delay and development effort to the process. Based on these observations and the results from Table 3.1 this functionality will be implemented in software.

- **Find Corners**: The process of finding corners has to process all the pixels in the input image in order to return the results needed. Many of the processing steps operates on a number of pixels to produce a single result. This constraint renders software solutions ineffective and time consuming. On the other hand developing a hardware accelerator not only provides solution to parallelize many of the internal computations but also will speed up the overall computation time. Moreover by applying a pipeline approach to the processing, system's throughput can also be increased. In addition taking into account Table 3.1 a hardware accelerated version of Harris Corner Detector algorithm will be used.

- **Cluster Corners**: Clustering in general can be quite expensive, especially when the dataset used is large, because a relation between all possible data combinations has to be

evaluated. The way clustering is used in the developed system and the number of data that are processed (a small subset of the number of pixels) makes it better candidate for implementation on software, which can also be supported from the results presented in Table 3.1.

- **Recognize Object**: In order to recognize the object a simplified geometric model of the object is used. The matching between the model used and the one computed needs only to perform some comparisons taking into account some variations from the predefined model in the form of an error associated with the model. The overall operation is computational inexpensive so it is implemented in software.

- **Align Camera**: In this step simple comparisons between some predefined properties and the ones computed during object recognition is done in order to decide how the camera must be moved and aligned with the object being tracked. This task is quite simple and computational inexpensive, so it is implemented on software.

Based on the analysis presented previously, Figure 3.3 presents the partition to software and hardware components used in the final system.

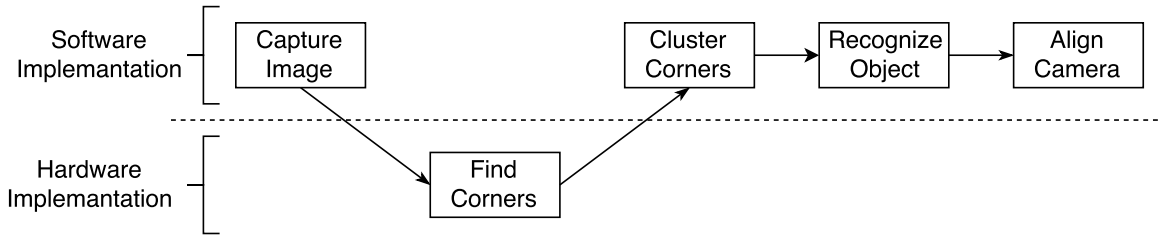


Figure 3.3: Hardware/software partitioning of the implemented system.

3.3 The Harris Corner Detector

3.3.1 Feature Detection

In computer vision and image processing the concept of feature detection refers to methods that aim at computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves or connected regions. In addition to such attribute information, the feature detection step by itself may also provide complementary attributes, such as the edge orientation and gradient magnitude in edge detection and the polarity and the strength of the blob in blob detection. Features can be categorized as follows

- **Edges** : Edges are points where there is a boundary between two image regions. In general, an edge can be of almost arbitrary shape, and may include junctions. In practice, edges are usually defined as sets of points in the image. Edges have a one-dimensional structure.

- **Corners** : The terms corners or interest points refer to point-like features in an image, which have a local two dimensional structure.

- **Blobs** : Blobs provide a complementary description of image structures in terms of regions, as opposed to corners that are more point-like.

- **Ridges** : From a practical viewpoint, a ridge can be thought of as a one-dimensional curve that represents an axis of symmetry, and in addition has an attribute of local ridge width associated with each ridge point.

The interest on this thesis is on corner detection. The corner detection, or interest point detection, is a useful method that used of an image to extract the feature or infer the context. It is predominantly applied in many aspects such as image mosaicing, tracking and recognizing, etc.

In image processing, a point can be considered as a corner if there is a intersection of two edges. A good indicator that determines the quality of a corner detection algorithm is to see if it can detect the same corner under multiple circumstances, in other words, different similar pictures that had done some other image processing such as rotation, darken, etc. The most frequently used algorithm for corner detection is proposed by Harris and Stephens [8], which is a further work on a method developed by Moravec [12] and was first published in 1988. In this thesis, the Harris Corner Detector is chosen and implemented as an accelerator.

3.3.2 Theoretical Background

The Harris Corner Detector is based on the thoughts of Moravec [12], which are explained briefly. Assuming a 2-dimensional image, whose intensity is denoted as I, Moravec starts with a window W centered at the pixel $p(x, y)$ and moves (shifts) this window in the neighborhood of p. If the movement is (u, v) the changes of the intensity are measured with the help of the auto-correlation function as

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{3.1}$$

where :

- $w(x, y)$ a window function equal to 1 inside the window W and 0 outside
- $I(x+u, y+v)$ the shifted intensity, where the shifts are $(u, v) = (1, 0), (1, 1), (0, 1), (-1, 1)$
- $I(x, y)$ the intensity of the image at position (x, y)

Moravec claims that small changes will appear in all directions for a constant intensity in the neighborhood of $p(x, y)$, which means there is a flat region (Figure 3.4a). Small changes in only one direction can be found for an edge (Figure 3.4b) whereas the direction of nearly no changes resembles the direction of the edge and finally big changes in all directions will be observed for a corner (Figure 3.4c).

Harris and Stephens [8] improved upon Moravec's corner detector by considering the differential of the corner score with respect to direction directly, instead of using shifts. To eliminate Moravec's algorithm shortcomings, which were noisy response due to the binary window function and anisotropic response due to the shifts used, first they applied a Gaussian window function

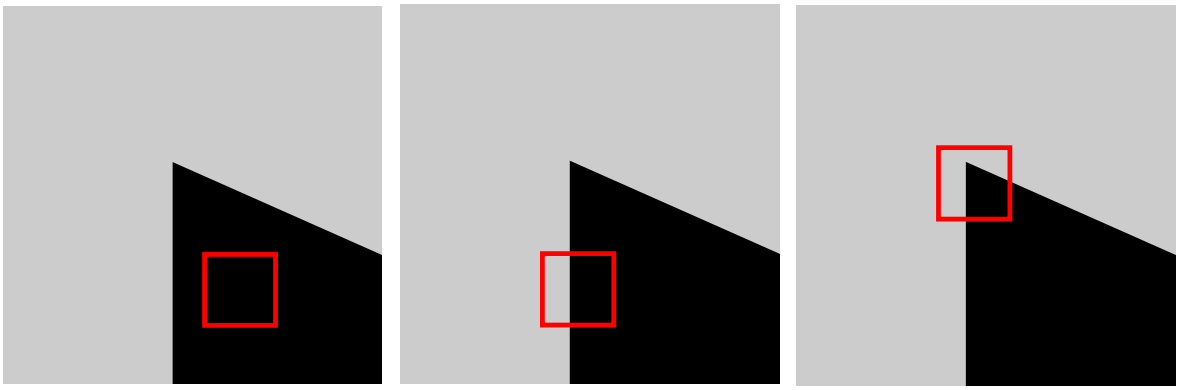
$$w(x, y) = e^{\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)} \tag{3.2}$$

and secondly they approximated $I(x + u, y + v)$, by a Taylor expansion to consider all small shifts, as

$$I(x + u, y + v) \approx I(x, y) + I_x(x, y)u + I_y(x, y)v \tag{3.3}$$

where I_x, I_y partial derivatives of I. Based on this approximation the expression 3.1 becomes

$$E(u, v) \approx \sum_{x,y} w(x, y)[I_x(x, y)u + I_y(x, y)v]^2 \tag{3.4}$$



(a) (b) (c)

Figure 3.4: An example of (a) a flat region, (b) an edge and (c) a corner.

or

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.5}$$

where A is a 2x2 matrix computed from the image derivatives

$$A = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{3.6}$$

The eigenvalues λ_1 and λ_2 of this matrix describe the changes inside the window similar to the moving window of Moravec. The way to compute the exact value of eigenvalues is computationally expensive and complex, in which the calculation of square root is needed, so Harris and Stephens developed another approach to "measure" the eigenvalues or in other words the corner response ("cornerness") by means of the function

$$R = \det A - \kappa(\operatorname{Tr} A)^2 \tag{3.7}$$

where $\det A = \lambda_1 \lambda_2$, $\operatorname{Tr} A = \lambda_1 + \lambda_2$ and $\kappa = 0.04 - 0.06$. Based on the value of R the following cases are considered:

- when $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- when $R < 0$, which happens when $\lambda_1 \gg \lambda_2$ or vice versa, the region is an edge.
- when R is large, which happens when λ_1, λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

Figure 3.5 shows a visual representation of the classification of image points into corners, edges and flat regions based on Harris Corner Detector.

3.3.3 Hardware Implementation

As presented earlier, from an algorithmic point of view, for each pixel, Harris calculates a "cornerness" strength according to the formula $\mathrm{I}_x^2\mathrm{I}_y^2 - (\mathrm{I}_x\mathrm{I}_y)^2 - 0.04 \cdot (\mathrm{I}_x^2 + \mathrm{I}_y^2)^2$, where I_x^2, I_y^2 and $\mathrm{I}_x\mathrm{I}_y$ denote the Gaussian-smoothed products of the image derivatives, which are themselves computed via a Sobel operator. Cornerness values that exceed a given threshold and the corresponding values of the remaining 3×3 neighborhood of the pixel under examination (i.e., they constitute a local maximum), designate corners on the image. These corners are refined with subpixel accuracy by fitting a quadratic surface on the aforementioned 3×3 region.

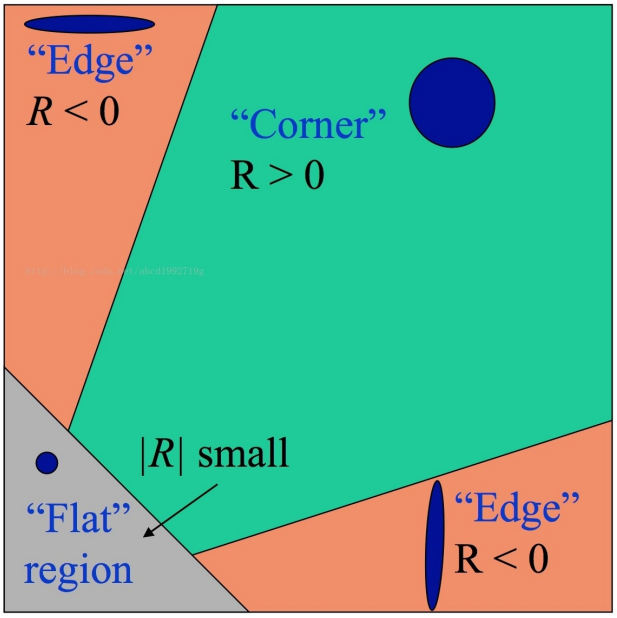


Figure 3.5: Classification of image points based on their corner response.

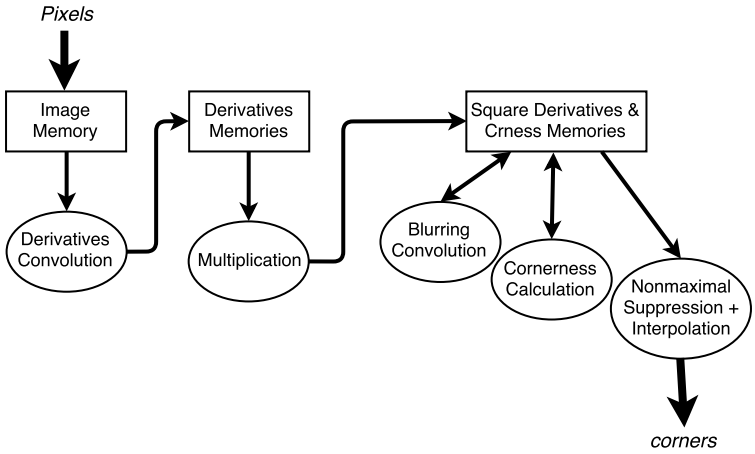


Figure 3.6: Harris Corner Detector block diagram.

On FPGA, Harris is accelerated according to the deeply pipelined architecture shown in Figure 3.6, which was coded in VHDL. The image pixels are loaded to the "Image Memory", which feeds the "Derivatives Convolution" module with two successive bursts of the image (1 pixel per cycle) to output the image derivatives dx and dy. The derivatives are squared and then temporarily stored in distinct memory blocks. Each block of dx^2, dy^2 and $dxdy$ is forwarded individually to the "Bluring Convolution" in a pixel-by-pixel fashion to produce individually the I_x^2, I_y^2 and $\mathrm{I}_x\mathrm{I}_y$ blocks (the resulting values overwrite the old derivatives in the memory). The three I blocks are then forwarded in parallel to the "Cornerness Calculator" (1 triplet $\langle \mathrm{I}_x^2, \mathrm{I}_y^2, \mathrm{I}_x\mathrm{I}_y \rangle$ per cycle) to evaluate the aforementioned formula for each pixel (one cornerness per cycle, overwrites its I triplet in the memory). The cornerness values are forwarded in a burst mode (one per cycle) to the final module to be non-maximally suppressed and interpolated.

Because the image size is too large to fit into the FPGA memory resources of XC7Z020 device, the image is divided into multiple smaller bands in order to be processed consecutively by reusing the same FPGA resources. The bands overlap each other in a specific way so the operation of the accelerator is functional equivalent to the original Harris Corner Detector.

3.4 Data Clustering

Data clustering is a process of assigning a set of records into subsets, called clusters, such that records in the same cluster are similar and records in different clusters are quite distinct [10]. Data clustering is often confused with classification, in which objects are assigned to predefined classes. In data clustering, the classes are also to be defined. A typical clustering process involves the following five steps:

1. **Pattern representation** : The number and type of the attributes which will be used are determined.

2. **Dissimilarity measure definition** : A distance measure appropriate to the data domain is defined. Various distance measures have been developed and used in data clustering. The most common one among them is the Euclidean distance.

3. **Clustering** : Here a clustering algorithm is used to group a set of records (data) into a number of meaningful clusters.

4. **Data abstraction** : One or more prototypes (representatives) of a cluster is extracted so that the clustering results are easy to comprehend. For example, a cluster can be represented by a centroid.

5. **Assessment of output** : In this final step the output of a clustering algorithm is assessed. There are three types of assessments [9]: external, internal, relative. In an external assessment, the recovered structure of the data is compared to the a priori structure. In an internal assessment, one tries to determine whether the structure is inherently appropriate to the data. In a relative assessment, a test is performed to compare two structures and measure their relative quality.

A cluster is defined as a group of data points satisfying various plausible criteria such as:

- Share the same or closely related properties.
- Show small mutual distances.
- Have "relations" with at least one other data point in the group.
- Can be clearly distinguishable from the rest of the data points in the dataset.

In this thesis two clustering algorithms are employed and evaluated as to how well they perform their role in the image processing flow presented earlier. These algorithms are the k-means and DBSCAN and a brief description is given next.

3.4.1 k-means Algorithm

The k-means algorithm for clustering [15] is the most widely used partitional clustering algorithm. It starts by choosing k representative points as the initial centroids. Each point is then assigned to the closest centroid based on a particular distance measure chosen. Once the clusters are formed, the centroids for each cluster are updated. The algorithm then iteratively repeats these two steps until the centroids do not change or any other alternative relaxed convergence criterion is met. k-means clustering is a greedy algorithm which is guaranteed to converge to a local minimum. Typically, the convergence condition is relaxed and a weaker condition may be used. Algorithm 1 provides an outline of the basic k-means algorithm.

Algorithm 1: k-means Algorithm

Input : $D = \{p_1, p_2, \ldots, p_n\}$ (set of points to be clustered)
k (number of clusters)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l_1, l_2, \ldots, l_n\}$ (set of cluster labels for points in D)

Randomly select k points as initial centroids;
repeat
  Form k clusters by assigning each point p_i to its closest centroid c_i;
  Recompute the centroid c_i of each cluster;
until *convergence criterion is met*;

A wide range of distance measures can be used within k-means algorithm while computing the closest centroid. The choice can significantly affect the centroid assignment and the quality of the final solution. The different kinds of measures which can be used here are Manhattan distance, Euclidean distance, Cosine similarity etc. In general, for the k-means clustering, Euclidean distance metric is the most popular choice. As one of the most often used clustering algorithms, the k-means algorithm has some important properties :

- It is efficient in clustering large data sets
- It often terminates at a local optimum
- The clusters have convex shapes
- It works on numerical data
- The performance is dependent on the initialization of the centers

3.4.2 DBSCAN Algorithm

DBSCAN [6] estimates the density by counting the number of points in a fixed-radius neighborhood and considers two points as connected if they lie within each other's neighborhood. A point is called core point if the neighborhood of radius *eps* contains at least $minPts$ points, i.e., the density in the neighborhood has to exceed some threshold. A point q is directly density-reachable from a core point p if q is within the *eps*-neighborhood of p, and density-reachability is given by the transitive closure of direct density-reachability. Two points p and q are called density-connected if there is a third point o from which both p and q are density-reachable. A cluster is then a set of density-connected points which is maximal with respect to density-reachability. Border points are defined as the points in the dataset that contains less than $minPts$ data points in their *eps*-neighborhood, but are reachable from some core point, while noise is defined as the set of points in the dataset not belonging to any of its clusters. The task of density-based clustering is to find all clusters with respect to parameters *eps* and $minPts$ in a given dataset. Algorithm 2 provides an outline of the DBSCAN algorithm.

Algorithm 2: DBSCAN Algorithm

Input : $D = \{p_1, p_2, \ldots, p_n\}$ (set of points to be clustered)
eps (maximum distance between points)
$minPts$ (minimum number of points required to form a dense region)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l_1, l_2, \ldots, l_n\}$ (set of cluster labels for points in D)

repeat
Arbitrary select a point p_i;
Retrieve all points density-reachable from p_i w.r.t. eps and $minPts$;
If p_i is a core point, a cluster is formed;
If p_i is a border point, no points are density-reachable from p_i;
Visit the next point on the dataset;
until *all the points have been processed*;

One limitation of DBSCAN is that it is sensitive to the choice of eps, in particular if clusters have different densities. If eps is too small, sparser clusters will be categorized as noise. If eps is too large, denser clusters may be merged together. In other words, if there are clusters with different local densities, then a single eps value may not suffice.

Chapter 4

Software Environment

4.1 Bare-metal Environment

A bare metal environment, is a development environment that aims to provide a very low-level of software modules that the system can use to access processor-specific functions. Regarding the Zynq platform specifically, Xilinx provides a standalone OS platform that provides functions such as configuring caches, setting up interrupts and exceptions and other hardware related functions. The standalone platform is used whenever an application requires to access processor features directly. For applications with low complexity this leads to more lightweight solutions and lot of resources can be saved. A standalone OS also enables close control over code execution but is fairly limited in terms of functionality, so it should only be used for applications where the software functions are straightforward and repetitive. The number of tasks being carried out by a standalone OS should be relatively small, as adding further tasks can increase the task management required by the standalone rapidly.

However the development of bare-metal applications is an arduous and time-consuming procedure due to limited availability of function libraries and low-level programming of the various hardware components and the drivers for specific devices. Another drawback is that the software in a bare metal environment is limited to be executed in a single thread. Moreover testing and debugging can be performed through cross-compilation.

4.2 Operating System

The alternative solution is to use an open-source OS which offers a plethora of benefits. The most important of these benefits are summarized below:

- Provides a high level of abstraction of the system through a variety of libraries for the underlying hardware. Hence, the developer can focus on the main functionality of the application without worrying about low-level details.
- Provides ready-to-use drivers for a wide range of peripherals.
- Enables multithreading for parallel processing.
- Offers code safety by confining the application in its private space in memory and adds protection against illegal accesses to the hardware.

The aforementioned benefits can reduce development time significantly and offer a more convenient environment for embedded programming. Several open-source OS can be found available for ARM-based SoC FPGAs (e.g., Xilinx's Petalinux, Linaro Ubuntu etc.). In this

thesis we use Xillinux [1], which also suits well with the PS-PL communication solution described later.

4.2.1 Xillinux OS

Xillinux is a Linux distribution based on Ubuntu LTS 12.04 for ARM which can be immediately deployed on ZedBoard. Xillinux is intended as a platform for rapid development of mixed hardware/software systems that embeds the infrastructure for communication between the PS and PL subsystems. Moreover Xillinux offers a Graphical User Interface (GUI) that is convenient when developing image processing algorithms because real time visualization of the results can help identify possible problems with the processing.

The biggest advantage of using Xillinux is the native compilation of user applications and kernel modules on the development board used without the need, as mentioned earlier, of a cross-compiler. Overall Xillinux roughly supports the same capabilities as a personal desktop computer running Linux.

Chapter 5

System Integration

5.1 Hardware/Software Communication

Image processing requires high throughput rates and low latency data transfers between the CPU and the hardware accelerator. Considering these requirements a streaming protocol is the most appropriate solution for such applications. In a streaming protocol handshake mechanisms are limited only at the start of the data movement between the CPU and the hardware accelerator and then, valid data can be transferred for processing at every consecutive cycle. The realization of an efficient streaming communication should rely on a Direct Memory Access (DMA) mechanism.

Direct Memory Access (DMA) allows external devices to gain access to the main bus linking the processor and the system memory; it moves data directly between the main memory and some other part of the system. The goal is an increase in data throughput and a decrease in the CPU load, which can reduce power requirements and enable more application work to be done. The need to handle more data at higher rates means DMA controller is now an integrated part of hardware and software design. Many embedded systems have internal and external interfaces, which produce and consume data. Usually, the main processor requires the data to be present in the main memory for performing any operation on it. A simple way of moving data between the peripheral device and main memory is to use the main processor to perform load or store operations for each byte or word of data to be moved. This method is time consuming and eats away precious CPU computation time. The alternative is to set up a DMA transfer that does the job of moving the data from source to destination. Once the processor has set up the transfer it can do something else while the transfer is in progress or it can wait to be notified when the transfer is completed via a status register or an interrupt. This method saves precious CPU cycles leading to a considerable boost in overall system performance and also increasing the data throughput.

In ARM-based SoC FPGAs a widely used streaming protocol is the AXI4-Stream which rely on the well established AXI4 protocol presented in Section **??**. Although, the implementation of AXI4-Stream is a major design challenge due to the considerable amount of time spend for the customization of the accelerator interface according to this protocol and the validation of its proper operation and integration in the system. Regrading the control signals transfers of the accelerator (e.g., start, enable, complete, setup options etc.) as well as the initialization of the DMA, simpler communication protocols are being employed. Such protocols are based on a straightforward hand-shaking mechanism. An extensively used approach for ARM based SoC FPGAs is the AXI4-Lite protocol. Figure 5.1 presents the general architecture of an AXI based hardware accelerator, where the AXI-Stream interfaces are used for sending and receiving data and the AXI-Slave interface is used for controlling the accelerator.

However in this thesis in order to ease the implementation of the communication between

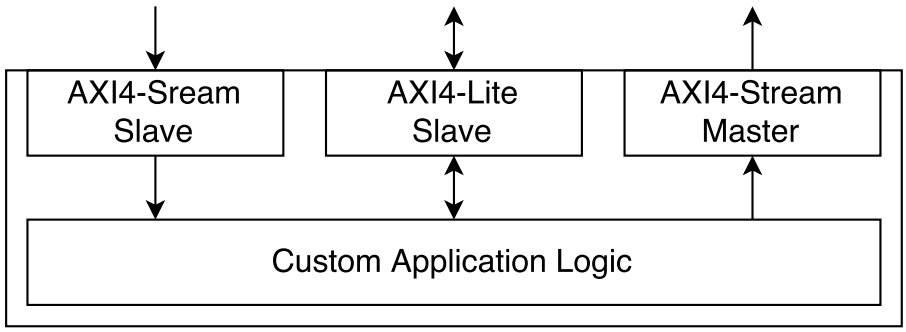


Figure 5.1: General architecture of an AXI based accelerator.

the CPU and the hardware accelerator an off-the-shelf, generic solution is utilized, which can be deployed relatively fast and additionally, can be adjusted to meet the specific application needs. This solution still communicates based on AXI protocol with the CPU, but on the PL side it presents a simpler more generic interface.

5.1.1 Data Streaming : Xillybus IP Core

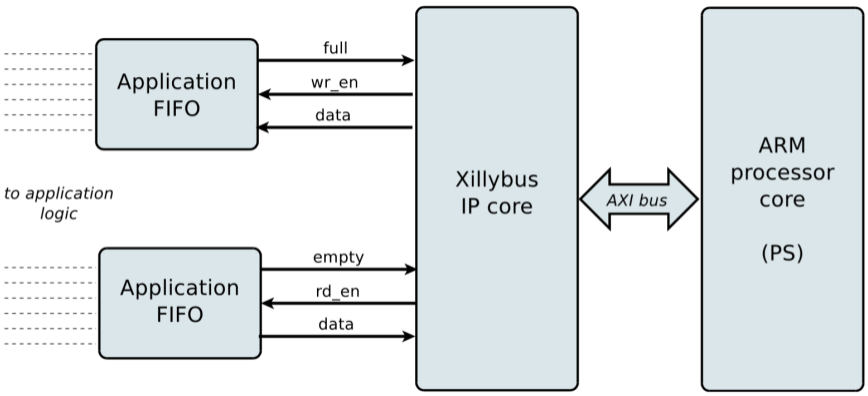


Figure 5.2: Xillybus IP core interface to PS and PL on Zynq.

Xillybus is a family of IP cores, developed by Xillybus Ltd. [2], consisting of two cores targeting different application needs. The first, named Xillybus IP core, presented in Figure 5.2, is a DMA-based solution, that offers ready-to-use implementation of streaming communication between the CPU and the hardware accelerator. It provides a number of features that makes it a versatile solution and increases productivity. These features are summarized below:

- Flexibility regarding the number of the employed PL interfaces and customization of the configuration settings for each distinct interface.
- Portable to different vendors FPGAs (e.g., Xilinx, Althera) and compatible with various OS (e.g., Linux, Windows).
- Easy programming model, supporting diverse programming languages (e.g., C/C++, Java, Python etc).
- Very simple interface model on the PL side, without requiring additional development effort for the integration with the hardware accelerator.

For each interface to PL the configuration settings are hardcoded to the IP and are retrieved during system startup from its device driver, which in turn allocates DMA buffers in host main memory. The hardware/software communication based on Xillybus can be realized through a trivial programming model. In Listing 5.1 an example of Xillybus communication for both reading and writing from/to an accelerator in PL is given. The write and read interfaces are manipulated as single device files. It is worth mentioning that there is the capability of the concurrent CPU processing between a write and read operation.

```
int write_fd, read_fd;
/* host application data buffer */
unsigned char *buffer;

/* open Xillybus write interface  */
write_fd = open("/dev/xillybus_write_device",O_WRONLY);

/* write sizeof(buffer) number of bytes */
write(write_fd, buffer, sizeof(buffer));

/* close Xillybus write interface */
close(wrire_fd);
                .
                .
                .
/* open Xillybus read interface */
read_fd = open("/dev/xillybus_read_device",O_RDONLY);

/* read sizeof(buffer) number of bytes */
read(read_fd, buffer, sizeof(buffer));

/* close Xillybus read interface */
close(read_fd);
```

Listing 5.1: Xillybus programming example.

5.1.2 Control Communication : Xillybus-Lite IP Core

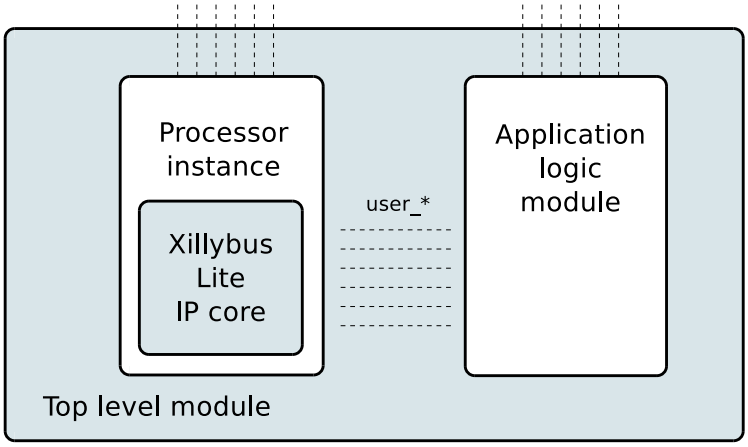


Figure 5.3: Xillybus-Lite IP core in systems hierarchy.

The second core, named Xillybus-Lite IP core, provides an abstraction to the underlying AXI4-Lite protocol, offering easy access to registers or memory modules implemented in PL. Xillybus-Lite presents an illusion of a bare-metal environment to the software, and a trivial

interface of address, data and read/write-enable signals to the logic design. This core also exposes an input signal which allows the application logic to send hardware interrupts to the processor. Figure 5.3 shows where in the systems hierarchy Xillybus-Lite is residing.

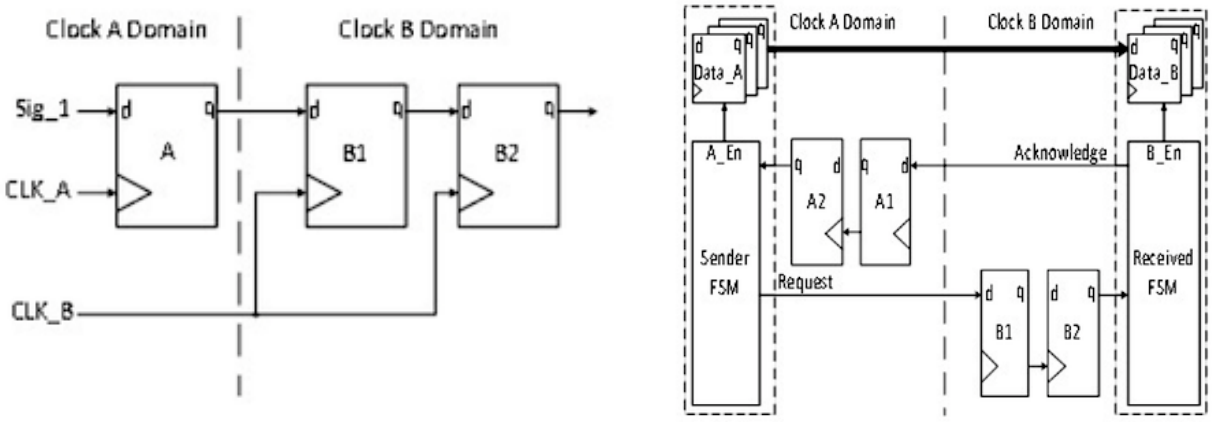
Multiple instances of the core can be employed and its programming model is as simple as Xillybus with some exceptions. Because Xillybus-Lite driver is based upon Linux's User I/O interface (UIO), which represents a peripheral as a device file, its physical address space must be mapped to the user address space and the access to PL resources is performed using pointer reading/writing. An example of Xillybus-Lite programming model is illustrated in Listing 5.2.

```
int xillybuslite_fd;
void *map_addr;

/* open Xillybus-Lite interface */
xillybuslite_fd = open("/dev/uio0",O_RDWR);

/* creates user address space for mapping of Xillybus-Lite interfacing */
map_addr = mmap(NULL, size ,PROT_READ | PROT_WRITE,MAP_SHARED, xillybuslite_fd ,0);

volatile unsigned int *pointer = map_addr;

/* write to address */
*pointer = the_value_to_write;

/* read from address */
the_value_read_from_register = *pointer;

/* deletes Xillybus-Lite mapping */
munmap(map_addr, size);

/* close Xillybus-Lite interface */
close(xillite_fd);
```

Listing 5.2: Xillybus-Lite programming example.

5.2 Synchronization Techniques for Hardware/Software Integration

SoC FPGA systems include various clock domains that refer to the processing system (PS), programmable logic (PL) and memory. Usually, the clock in PL is considered the same for the hardware accelerator(s) and the inter-communication interfaces. The problem is that the communication interfaces can operate reliably up to a specific frequency, and hence can hinder the increased performance of an accelerator which can potentially work in higher clock rates. In order to surpass this bottleneck various cross-clock domain techniques have been proposed [4]. When dealing with single bit signals a widely known technique based on the employment of double flip-flops synchronizers can be used (Figure 5.4a). This technique is very simple to design but requires the employment of additional constraints during the implementation phase of the system and presents unreliable behavior for higher clock rates. Another solution is to augment these double flip-flop synchronizers with a handshake mechanism in order to reduce the probability of faulty data propagation at the expense of additional transmission latency (Figure 5.4b). Although, this approach requires extra development effort and may lead to inaccurate results when transmitting multi-bit signals due to transmission delay variation presents each separate single-bit signal.



(a) Double flip-flop synchronizer. (b) Handshake based synchronizer

Figure 5.4: Examples of clock domain crossing syncronizers.

An alternative approach to achieve synchronization is the employment of asynchronous dual clock FIFO buffers. The main advantage of dual clock FIFOs is that provide asynchronous read and write operations from different clock domains. Furthermore, this approach provides the following benefits:

- All FPGA vendors provide ready-to-use customizable IP cores for asynchronous FIFOs which are optimized for the target device and their employment requires minimal development effort.
- When the accelerator processes a set of data the next set can be temporally buffered and be ready immediately when needed.
- Asymmetric data widths can be used in FIFOs input and output interfaces. For instance multiple data can be written in parallel from the host CPU and read serially from the hardware accelerator.

Taking into account the above advantages, the system utilizes asynchronous FIFO buffers between the Xillybus IP core and the accelerator for both the control and data transfers. This approach ensures the synchronization of the system with negligible development effort.

5.3 Finalized Architecture

Closing this chapter the finalized architecture of the developed system is presented. Based on the employment of all the proposed IP cores and techniques described in this section, the final architecture of the system is depicted in Figures 5.5 and 5.6. Figure 5.5 shows the hardware architecture of the system and how the hardware components are connected in the PL (Harris, Xillybus) between themselves and with the PS, Figure 5.6a gives the execution flow of user application, while Figure 5.6b shows how the user application communicates with the necessary peripherals using drivers and libraries of the operating system.

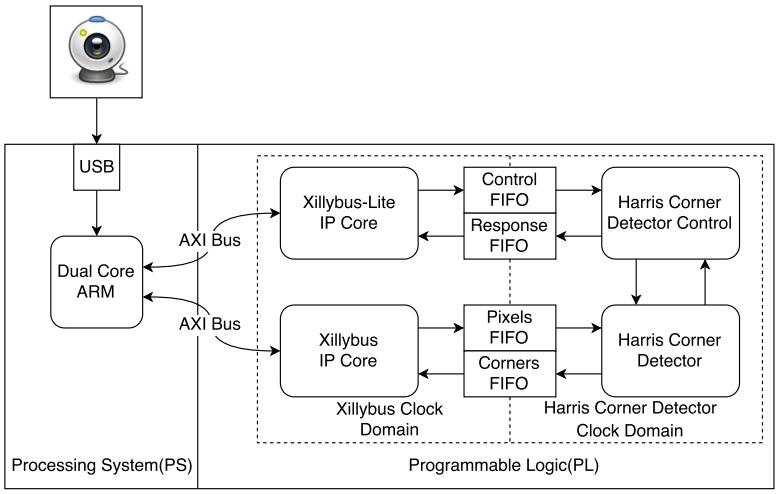
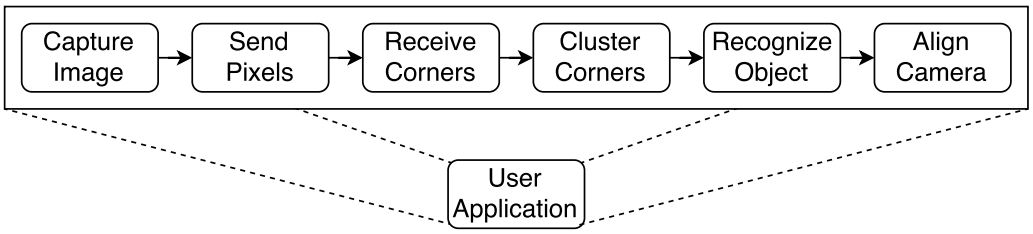
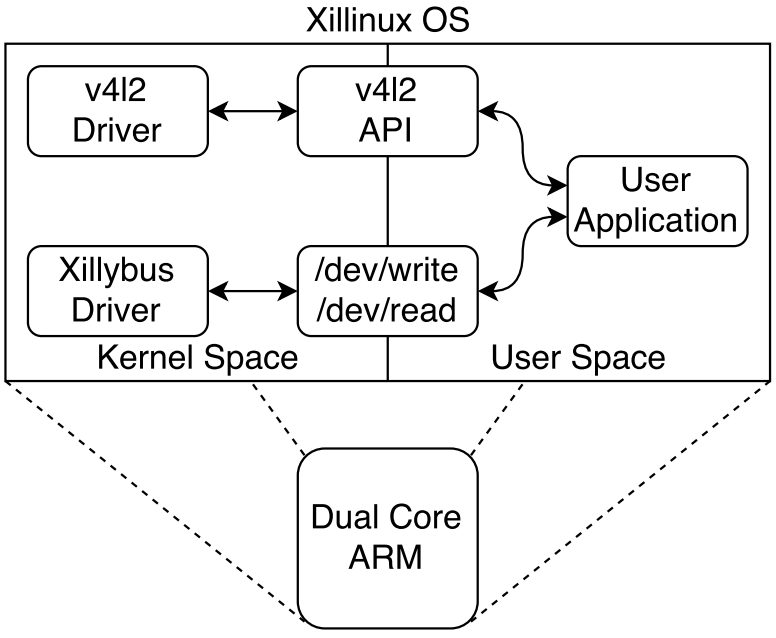


Figure 5.5: Finalized hardware architecture.



(a) User application overview.



(b) Operating system and software overview.

Figure 5.6: Finalized software components architecture.

Chapter 6

System Evaluation

6.1 Experimental Results

This chapter centers on the performance exhibition of the system based on the employed techniques presented earlier. The evaluation of the system is divided in two parts. The first part deals with the modules that resides on the PL side of Zynq, namely Harris Corner Detector and Xillybus IP Cores. The second part takes into account the full system implementation (hardware and software) and explores how well the system performs its operation.

Hardware modules evaluation

For the sake of fairness, all the experiments presented next were conducted with the same input data – images of size 512x384 – which are stored in system memory. The first step was to evaluate the effective communication bandwidth of Xillybus. Based on the fact that Xillybus performs better when a large number of data is send from PS to PL, the number of bands processed by Harris has a significant impact on communication bandwidth. This stems from the fact that the smaller the number of band is, the fewer system calls must be performed to transfer them to the PL side. Moreover the band number also affects the utilization of memory resources used by Harris in the PL. Figure 6.1 illustrates the trade-of

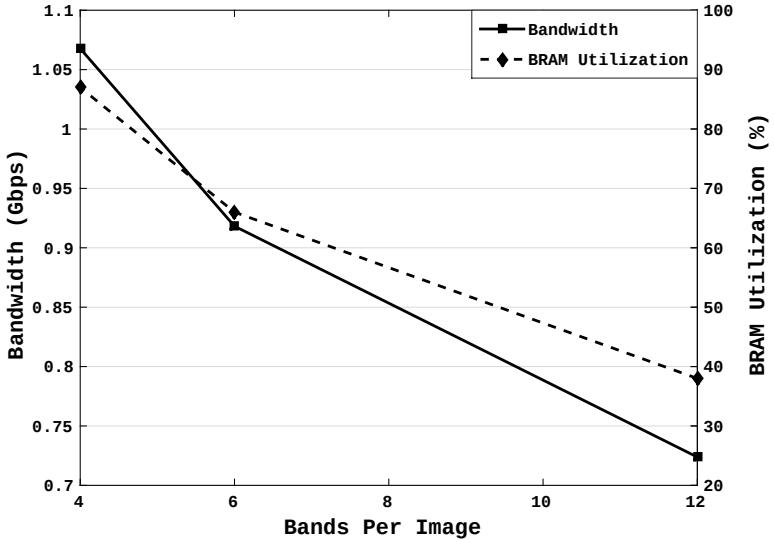


Figure 6.1: System bandwidth and FPGA memory resource utilization for various number of bands using a 16-bit Xillybus interface.

between the FPGA memory resource utilization and the communication bandwidth according

to the number of bands. It should be mentioned that the write interface of Xillybus is set to 16-bits, Xillybus operates at 100Mhz and Harris operates at 200Mhz. As can been seen reducing the number of bands used to partition the input image, from 12 to 6 and then to 4, communication bandwidth increases, but also the memory utilization increases and consumes 87% of the total BRAMs present in XC7Z020 device when 4 bands is used. An other aspect that impacts the communication bandwidth is the data width of the Xillybus interface. In order to have the best utilization of Xillybus 32-bit, interfacing for data transfers has to be used. Using 32-bit interfacing will double the communication bandwidth which will be closer to the theoretical bandwidth of ∼2.4 Gbps that Xillybus can offer.

For the rest of the experiments not only the input data and Xillybus write interface was the same, but also equivalent values for the configuration parameters of Harris algorithm where used. Moreover, ensuring that the minimal number of tasks are running in PS the measurements taken are free of random variations other than the ones introduced by running the software of the image processing system. Given this assumptions, Table 6.1 shows the total resource utilization and operating frequency of the system regarding the PL side and analyze this in two parts: the implementation of Harris and the Xillybus communication. Notice that the

Table 6.1: Operating frequency and resource utilization of the system analyzed in Xillybus communication and Harris implementation.

		Harris Accelerator		**Xillybus IP Cores**	
XC7Z020 PL Resources	Available	Used	Utilization	Used	Utilization
LUTs	53200	11922	22%	2768	5%
DFFs	106400	14038	13%	2656	2%
BRAMs	140	52	37%	1	1%
DSPs	220	54	25%	0	0%
Max. Operating Freq.		300 Mhz		100 Mhz	

communication cost is relatively low in contrast to Harris implementation. The maximum achievable frequency for Harris that ensured correct data was 300 Mhz, while for Xillybus a 150 Mhz frequency where tested but the system showed wrong data transfers.

Figure 6.2 illustrates the average execution time of Harris in PL and the average PS/PL communication time for the processing of an image as a function of the operating frequency of Harris. The communication time is analyzed into two parts, the control and the data.

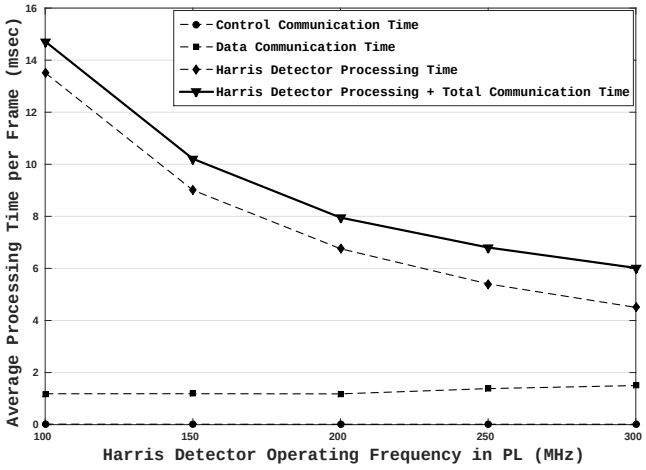


Figure 6.2: Execution time vs PL clock frequency.

Control communication refers to Xillybus-Lite IP that is used for the transmission of the control

signals while data communication denotes the Xillybus IP, which streams data to the PL. Note that the total communication time is almost stable over all the Harris operating frequencies. This results from the fact that the communication based on Xillybus IP cores operates at 100 Mhz. On the other hand, the execution time of Harris decreases as the clock rate increases. Considering Figure 6.2 one can observe that the percentage of the total processing time spend in the communication is increasing as the clock of Harris is boosted. For instance, at 100 Mhz clock rate the amount of time spend in communication is 8% of the total processing time, while at 300 Mhz clock rate increments to 25%.

In Figure 6.3 the power consumption of the system for the corresponding operating frequency of Harris is depicted. One of the ARM cores is responsible for monitoring the power controllers while the second one is dedicated to the application. Separate measurements were taken for PL

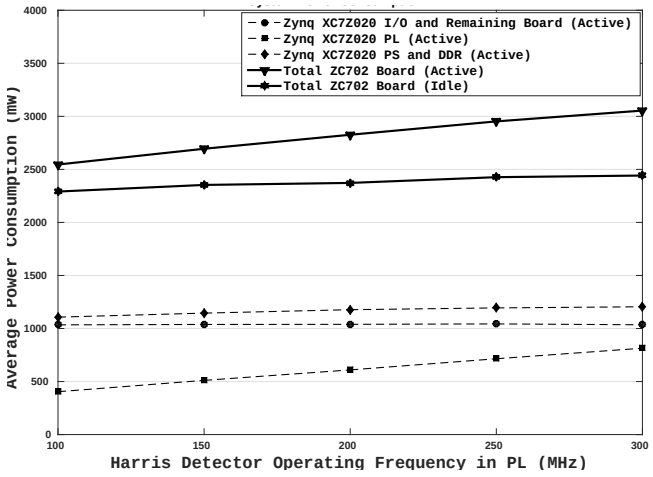


Figure 6.3: Power Consumption vs PL clock frequency.

and PS together with DDR and the peripherals on the board during the processing of a large set of input images. Also the total power consumption of the board (PS, PL and IOs) in idle and active state is shown. As the clock rate of Harris is boosted the power is linearly increases in PL while in PS there is only a very small alteration due to increased number of system calls that have to be served in the same amount of time. The small increase of power occurring

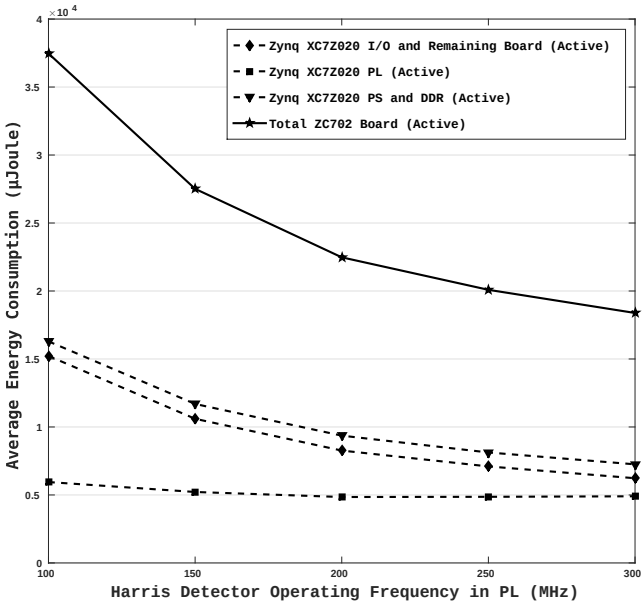


Figure 6.4: Energy Consumption vs PL clock frequency.

when the system is in idle state is a result of the static power consumption in PL. Notice that the power consumed by the device IOs and the peripherals of the board remains stable as they are not used in the system, as mentioned earlier. It is worth mentioning that even though the system consumes more power in higher operating frequencies of Harris, it is more energy efficient for the same workload when it is in active state. The energy measurements illustrated in Figure 6.4 confirms this principle.

In order to demonstrate the efficiency of the system in terms of execution time, power and energy consumption of Harris, a comparison with alternative processing platforms were performed. Figure 6.5 exhibits the performance comparison of the SoC implementation with software-oriented approaches running on an ARM Cortex-A9 (PS only) and an Intel i5-4590 CPU.

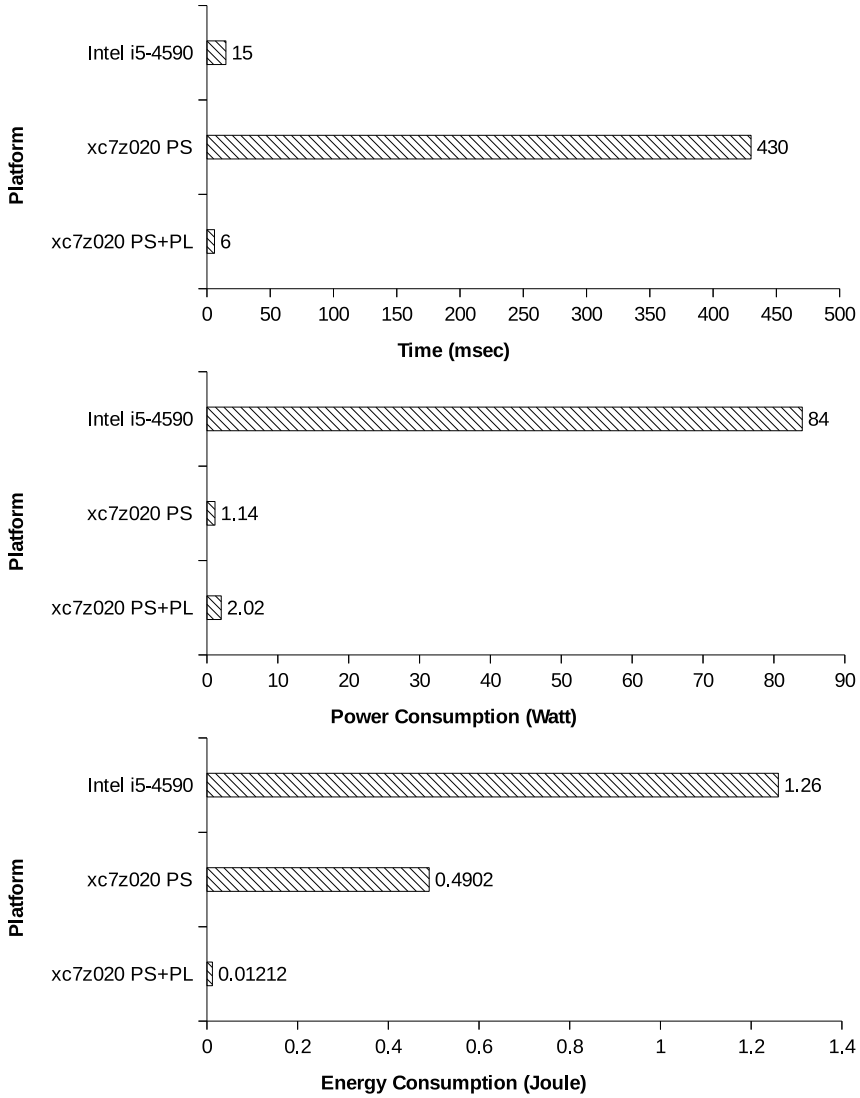


Figure 6.5: Performance comparison of SoC FPGA implementation with alternative processing platforms.

Based on these measurements in Table 6.2 the performance gain of the implemented system over the two alternative platforms are presented. It can be clearly seen that the SoC implementation is superior over the rest from the performance/watt aspect. Note that although the ARM approach (XC7Z020 PS) consumes less power the implemented system is more energy efficient.

Table 6.2: Performance gain of SoC FPGA implementation versus alternative processing platforms.

	XC7Z020 PS	**Intel i5-4590**	
Execution Time	71.6×	2.5×	
Power Consumption	0.56×	41.6×	**XC7Z020 PS+PL**
Energy Consumption	40.4×	104×	

Complete System Evaluation

To complete the evaluation, the system was tested in a real life scenario where a static target with rectangular shape was tracked. The purpose of this experiment was to see how well the overall processing flow performs its operation by processing 1000 frames captured by the camera. The first measure was to count the number of frames in which the system was able to recognize the object. At the same time when the system was able to recognize the object in three consecutive frames the result was interpreted as successful alignment with the object and it was also counted. Based on these measurements the quality of the algorithms was evaluated by computing the success rate, namely how many groups of three consecutive frames were counted out of the possible maximum of 333.

Table 6.3 presents the results obtained when the camera was allowed to move in the 3D cartesian space and could or couldn't rotate around its optical axis. As can be seen from the

Table 6.3: Algorithm quality of the implemented system

	Processing pipeline with k-means		**Processing pipeline with DBSCAN**	
	Dataset 1	Dataset 2	Dataset 1	Dataset 2
Rotation	✓	✗	✓	✗
Num. of recognition	127	623	832	863
Successful alignment	17	175	250	256
Succes rate	5.1%	52.8%	75.1%	76.9%

table, when rotation of the camera was allowed and k-means algorithm was used to cluster the corners found by Harris only in a very small number of frames the system was able to recognize and align with the object, achieving a success rate of only 5%. On the other hand when DBSCAN was used the success rate of the system peaked at 75%. This huge gap can be justified by the fact that k-means can't distinguish between noise (outliers) and useful data, a property that is embedded in DBSCAN algorithm.

Finally when rotation was prohibited, k-means improved significantly the accuracy of the system but not enough to be considered as a good option for use even for this special case. Contrary to the improvement of k-means, DBSCAN improved slightly.

Chapter 7

Conclusions

7.1 Thesis Summary

Image processing is a very demanding field, where both scientific community and industries try to provide new clever solutions to efficiently deploy image processing algorithms. Especially when targeting embedded systems the strict restrictions that these systems impose make the development of embedded image processing applications even more challenging.

This thesis dealt with image processing from a practical point of view. The main goal was to design and implement an image processing system and deploy it to a SoC FPGA platform. To demonstrate the process of such a task a case study was chosen. The system must be able to track an object, align itself with the object until it has focused on it. Based on this specification an image processing flow was designed. This flow consisted from algorithms with various degrees of complexity that led to applying a hardware/software codesign methodology to partition the system to hardware and software components. Having decided what components will be mapped to hardware and which to software, next a comparison of software environments led to selection of an open-source OS (Xillinux) as the platform to develop the software part of the system. This step concluded the design decisions that had to be made considering the system as distinct components without interaction. Next the intercommunication approach that glues together the hardware and software components where discussed and the finalized architecture of the system were presented. Up to this point the design flow was also meant to be viewed as guidelines for the development of image processing systems with SoC FPGAs that focus on increasing the productivity and decreasing time-to-market.

At the end an evaluation of the system was conducted. The experiments showed that the system can achieve a bandwidth of 0.7 Gbps (12 bands) up to 1.1 Gbps (4 bands) with constant write interface of 16-bits for Xillybus. Using 32-bits write interface the bandwidth can be doubled. Regarding its performance the system outperformed pure software implementations running on ARM and Intel processors achieving speedup of 71.6 and 2.5 respectively. Also it was proved by the experiments that it was more efficient from the performance/watt aspect, compared to the other two platforms.

For the recognition and tracking of the object the system worked poorly when k-means was used for clustering with a peak in accuracy of ∼53% when the camera was not allowed to rotate, while when DBSCAN was used its accuracy peaked at ∼75% in both cases. By applying

1. a more sophisticated clustering algorithm or using an adaptive DBSCAN w.r.t *eps* and $minPts$.

2. improve the model of the object used.

the accuracy of the system will be able to increase significantly, even beyond 90%.

7.2 Future Work

Future work that could be done on this project would be to use color images. Even though grayscale images has all the necessary attributes needed for a demonstration, in real life many image processing systems process color images. Another extension could be the addition of more image processing algorithms to the system architecture (e.g. feature description). While the addition of more image processing algorithms does not complicate the system architecture, it can help to demonstrate the full potential of SoC FPGAs with a more complex and demanding application and at the same time explore more thoroughly Xillybus capabilities and limitations.

Also a very useful addition for the future would be the explicit use of the NEON engine present in Zynq devices in order to complement the hardware accelerators during processing and offload the CPU even more. Finally to exploit software parallelism multi-threading can also be used. Multi-threading can also be utilized during communication with the hardware accelerator to decouple the send/receive operations.

Bibliography

[1] `http://xillybus.com/xillinux`.

[2] `http://xillybus.com/`.

[3] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[4] C. E. Cummings. Clock domain crossing (cdc) design & verification techniques using systemverilog. *SNUG-2008, Boston*, 2008.

[5] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Des. Test*, 10(4):64–75, Oct. 1993.

[6] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[7] R. K. Gupta and G. De Micheli. Readings in hardware/software co-design. chapter Hardware-software Cosynthesis for Digital Systems, pages 5–17. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[9] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[10] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[11] S. S. Math, R. Manjula, S. Manvi, and P. Kaunds. Data transactions on system-on-chip bus using axi4 protocol. In *Recent Advancements in Electrical, Electronics and Control Engineering (ICONRAEeCE), 2011 International Conference on*, pages 423–427. IEEE, 2011.

[12] H. P. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

[13] J. Teich. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1411–1430, May 2012.

[14] Xilinx. AXI Reference Guide. `http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf`.

[15] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.