Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

HW/SW Co-Design and Preprocessing for Accelerating Star Trackers on SoC FPGA

Διπλωματικη Εργασια

του

Παπαλουκά Εμμανουήλ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων
Αθήνα, Ιούλιος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

HW/SW Co-Design and Preprocessing for Accelerating Star Trackers on SoC FPGA

Διπλωματικη Εργασια

του

Παπαλουκά Εμμανουήλ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Ιουνίου 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Διονύσιος Ρεΐσης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Κ.Π.Α

Αθήνα, Ιούλιος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

(Υπογραφή)

....

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

Περίληψη

Οι διαστημικές εφαρμογές απαιτούν ταχεία και ακριβή μέτρηση του προσανατολισμού ενός δορυφόρου, κάτι που μπορεί να επιτευχθεί μόνο με τη χρήση ανιχνευτών αστέρων. Το όργανα αυτά αποτελούνται από ένα ψηφιακό αισθητήρα εικόνας που καταγράφει τον ουρανό, καθώς και από ψηφιακό υλικό το οποίο ανιχνεύει τα αστέρια στην εικόνα και τα αντιστοιχίζει σε γνωστούς αστερισμούς καταλόγου ώστε να προσδιορίσει τη θέση του δορυφόρου στο αδρανειακό σύστημα. Η ανίχνευση αστέρων είναι υφηλής πολυπλοκότητας και απαιτεί μεγάλους χρόνους εκτέλεσης, ιδίως εάν χρησιμοποιούνται συμβατικοί μικροεπεξεργαστές. Συνεπώς, η ανάγκη για ανιχνευτές αστέρων υψηλής απόδοσης οδήγησε στην αυξανόμενη χρήση εμπορικών συσκευών υλικού «από το ράφι» σε διαστημικές εφαρμογές.

Στην παρούσα διπλωματική εργασία αναπτύσσουμε σε ένα ενσωματωμένο σύστημα SoC FPGA τα στάδια προεπεξεργασίας δεδομένων της προγραμματιστικής ροής ενός ανιχνευτή αστέρων τα οποία αφορούν την ομαδοποίηση εικονοστοιχείων για μείωση του όγκου δεδομένων, και την ανίχνευση συστάδων που χρησιμοποιούνται για τον υπολογισμό κεντροειδών. Επιπλέον, υλοποιείται μια λογισμική προσέγγιση που εκτελείται στον επεξεργαστή της πλατφόρμας και συγκρίνεται με το ενσωματωμένο HW/SW σύστημα όσον αφορά την απόδοση. Η αρχιτεκτονική υποστηρίζει την επικοινωνία μεταξύ των PS και PL στοιχείων του SoC μέσω των AMBA AXI πρωτοκόλλων, καθώς επίσης και τη δυναμική προσαρμογή του κατωφλίου που χρησιμοποιείται κατά τη συσταδοποίηση ανάλογα με τα επίπεδο θορύβου της εικόνας.

Το σύστημά μας εξετάστηκε με ρεαλιστικές εικόνες που έχουν ληφθεί σε μια αποστολή της NASA και αξιολογείται ως προς την απόδοση, τη χρήση πόρων του FPGA και την κατανάλωση ισχύος. Είναι ικανό για ακριβή ανίχνευση εκατοντάδων συστάδων εντός του πλαισίου της εικόνας επιταχύνοντας την εκτέλεση κατά 60 φορές συγκριτικά με τον ARM επεξεργαστή του SoC FPGA, ενώ εκτιμάται οτι η επιτάχυνση μπορεί να αυξηθεί έως και 108 φορές. Τελικά, η ενσωματωμένη υλοποίηση που συνδυάζει υλικό και λογισμικό είναι ικανή για απόδοση σε πραγματικό χρόνο, καθώς επωφελείται από την αρχιτεκτονική παράλληλης επεξεργασίας του FPGA και τις ειδικά προσαρμοσμένες τεχνικές υλοποίησης.

Λέξεις Κλειδιά

Ανιχνευτές αστέρων, SoC, Zynq FPGA, Συσταδοποίηση, Ομαδοποίηση, Κατωφλίωση, Συνδυασμός HW/SW, Εφαρμογές Διαστήματος

Abstract

In space applications it is critical to measure the satellite's orientation fast and precisely, which can be only achieved using star trackers. This setup consists of a digital image sensor that captures images of the sky, as well as hardware that detects stars and maps them to known constellations in order to determine the inertial attitude of the satellite. Star detection is a process of high complexity due to large amounts of image data and thus, it takes significant time to execute, especially when operating on conventional microprocessors. Thus, the need for high performance star trackers leads to the use of Commercial Off-The-Shelf (COTS) FPGAs, which offer great parallelisation opportunities and provide remarkable speedups.

In this thesis, we focus on the implementation of an efficient algorithm for accelerating preprocessing operations of star trackers on COTS SoC FPGAs. More specifically, we develop a HW/SW embedded system for accelerating the preprocessing stages of a star tracker pipeline on Xilinx's Zynq. These stages refer to the image binning that decreases the data volume, and to the detection of clusters from which centroids will be subsequently extracted. The proposed architecture exploits parallelisation at multiple levels via parametric HDL circuit design. The HW/SW co-design integrates the PS and PL parts of Zynq, whose communication is established via AMBA AXI protocols. Our integrated system supports dynamic adjustment to the threshold used in clustering process depending on the noise floor level of the image frame.

The proposed design is tested with real images captured on a NASA mission and is evaluated in terms of performance, resource utilisation and power consumption. A software-oriented approach running on the PS of the SoC is also developed and compared to our HW/SW embedded system. Our proof-of-concept implementation accurately detects hundreds of clusters within the image frame while accelerating the execution, resulting to a speedup of 60x compared to the ARM processor with an estimated increase up to 108x. Hence, this HW/SW co-design achieves real-time performance as it benefits from the FPGA's parallel processing architecture and our custom implementation techniques.

Keywords

Star Tracker, SoC, Zynq FPGA, Clustering, Binning, Thresholding, HW/SW Co-Design, Space Applications

Ευχαριστίες

Θα ήθελα να ευχαριστήσω πρωτίστως τον επιβλέποντα καθηγητή μου κ. Δημήτρη Σούντρη που με εμπιστεύθηκε για την εκπόνηση της διπλωματικής μου εργασίας. Έπειτα, θα ήθελα να ευχαριστήσω ξεχωριστά το μεταδιδακτορικό ερευνητή Γιώργο Λεντάρη και τον υποψήφιο διδάκτορα Βασίλη Λέων για την αδιάλειπτη βοήθεια και την πολύτιμη καθοδήγηση που μου προσέφεραν. Ακόμη, θα ήθελα να ευχαριστήσω ιδιαίτερα το Γιάννη Στρατάκο για τις καίριες συμβουλές και παρεμβάσεις του πάνω σε πιο πρακτικά ζητήματα.

Επιπλέον, ευχαριστώ θερμά το Μάνο Κουμαντάκη και την εταιρεία του Infinite Orbits για την παροχή τεχνογνωσίας, απ΄ την οποία κατέστη δυνατή η ενασχόλησή μου με το συγκεκριμένο θέμα.

Θα ήθελα επιπροσθέτως να ευχαριστήσω την οικογένειά μου για την υπομονή, την υποστήριξη και την εμπιστοσύνη που μου έδειξε καθόλη τη διάρκεια της φοίτησής μου. Τέλος, θέλω να ευχαριστήσω ιδιαίτερα τους φίλους μου για τις στιγμές που μοιραστήκαμε και για τους οποίους νοιώθω ευγνωμοσύνη για τη δύναμη και το πάθος που μου δίνουν να ακολουθήσω τα όνειρά μου.

Contents

Περίληψη **1**

Abstract **2**

Ευχαριστίες **3**

Contents **5**

List of Figures **8**

List of Tables **12**

Εκτεταμένη Περίληψη **14**

1 Introduction **27**

1.1 Motivation 27

1.2 Thesis Scope 28

1.3 Project Objectives 30

1.4 Thesis Outline 30

2 Background **33**

2.1 Star Trackers 33

2.1.1 Star Tracker Characteristics 34

2.1.2 Sub-pixel Accuracy 35

2.1.3 Theoretical Background 36

2.1.4 Algorithms 38

2.1.5 Commercially Available Star Trackers 39

2.2 Related Work 40

2.3 Space Navigation 41

2.3.1 Celestial navigation for satellites 41

2.3.2 Celestial navigation for deep space probes 42

2.4 Rotational Kinematics 43

2.4.1 The Earth Centred Inertial (ECI) 43

2.4.2 Euler Angles . . . 43
2.4.3 Coordinate Transformation . . . 46
2.5 SoC FPGA Overview . . . 48
2.5.1 Xilinx Zynq SoC FPGA Architecture . . . 50
2.5.2 AXI4 Lite . . . 52
2.5.3 AXI4 Stream . . . 54
2.5.4 Design Flow . . . 54

3 Development on the Zynq SoC FPGA 58
3.1 HW/SW Co-design of Averaging 2D Binning . . . 58
3.1.1 High Level Architecture . . . 59
3.1.2 Low Level Implementation . . . 64
3.2 HW/SW Co-design of Clustering . . . 69
3.2.1 Cluster detection algorithm . . . 69
3.2.2 High Level Architecture . . . 72
3.2.3 Low Level Implementation . . . 79
3.2.4 Performance-Wise Optimization . . . 86
3.2.5 System Integration . . . 89

4 Experimental Evaluation 92
4.1 Experimental Setup . . . 92
4.2 Results and Analysis on Zynq FPGA . . . 93

5 Conclusion and Future Work 102

Bibliography 104

List of Figures

1 Διαφορετικά επίπεδα ομαδοποίησης εικονοστοιχείων. 16
2 Σχεδιαστική απεικόνιση του πυρήνα Ομαδοποίησης Εικονοστοιχείων. 17
3 Μηχανισμός αναζήτησης εικονοστοιχείων. 19
4 Διάγραμμα ροής για τη λήψη των δεδομένων εικόνας. 20
5 Σχεδιαστική αναπαράσταση του πυρήνα Συσταδοποίησης. 21
6 Αναπαράσταση καταμερισμού διεργασιών. 22
7 Εκτιμώμενη κατανάλωση ισχύος για το ολοκληρωμένο σύστημα. 24

1.1 Block diagram of a typical Star Tracker. 29
1.2 System-on-Chip Layout. 29
1.3 Process Flow of a Star Tracker. 29

2.1 Pointing and rolling accuracy schematic of the star tracker [1]. 34
2.2 Different Stars Utilising Hyperacuity technique [2]. 36
2.3 Image sensor's perspective Projection model [3]. 37
2.4 High level process flow of an ADCS. 38
2.5 Types of radial distortion [4]. 39
2.6 The Earth-Centred Inertial (ECI) frame [5]. 44
2.7 Roll-Pitch-Yaw notation for Euler angles [6]. 45
2.8 Euler Angle Sequence (3,1,3). 45
2.9 Coordinate model of a star tracker [7]. 46
2.10 Celestial coordinate system to star tracker's coordinate system [7]. 47
2.11 Pinhole Camera Model: ideal projection of a 3D object on a 2D image [8]. . 48
2.12 Field Programmable Gate Array (FPGA) schematic [9]. 49
2.13 Xilinx Zynq SoC FPGA Architecture [10]. 50
2.14 Channel architecture of reads [11]. 53
2.15 Channel architecture of writes [11]. 53
2.16 FPGA Design Flow. 55
2.17 VHDL code to synthesized design conversion. 56

3.1 Different levels of averaging binning. 59
3.2 Processing flow of the integrated system. 59
3.3 Block design of the HW/SW integrated system. 61

3.4 Zynq7 Processing System block design. 61
3.5 AXI DMA Block Diagram [12]. 63
3.6 Averaging 2D Binning block design. 63
3.7 Block design of the Binning kernel. 64
3.8 Adder Tree Structure for 2×2 binning. 65
3.9 Behavioural Simulation of the input FIFO buffer handling. 66
3.10 Behavioural Simulation of the output FIFO buffer handling. 67
3.11 Serial-to-Parallel block. 67
3.12 Behavioural simulation of Serial-to-Parallel operation. 68
3.13 Generic block design of Adder Tree. 68
3.14 Behavioural simulation of MAC operation. 69
3.15 Visualization of light spreading in a defocused star. 70
3.16 Configuration of Neighbourhoods of Search. 71
3.17 Processing flow of the HW/SW Co-design for Clustering. 72
3.18 Image partitioning in sliding windows. 73
3.19 Clustering block design. 73
3.20 Data transmission pipeline. 74
3.21 Block design of the Clustering kernel. 75
3.22 Illustration of every possible case for reading operation of starting pixels. . 76
3.23 Determination of Region-of-Interest. 77
3.24 Example of the addressing operation. 78
3.25 Pipeline of the processing steps handled by mainFSM. 79
3.26 Behavioural simulation of driving TREADY signal for a 256×256 image and winHeight of 16 rows. 81
3.27 Behavioural simulation of upperThresLoc. 81
3.28 VHDL code of the function that computes the addresses. 82
3.29 Behavioural simulation of fsm4ram. 82
3.30 Behavioural simulation of neighChecker. 83
3.31 Illustration of the correlation between the operations performed by fsm4RAM and neighChecker blocks. 84
3.32 VHDL code transformation regarding signal overflow. 86
3.33 Visualization of task partitioning. 88
3.34 Illustration of the three individual sub-processes performed by neighChecker component. 89
3.35 Pipeline of the processing stages of the HW/SW Co-design. 90
3.36 Block design of the HW/SW Co-design. 90

4.1 Zedboard Development Board [13]. 93
4.2 Board Connection Setup [13]. 94
4.3 Post execution result as shown in SDK terminal. 94
4.4 Estimated Power Consumption of Binning component. 95

4.5 Behavioural simulation of an output detected cluster. 96
4.6 Visualisation of a detected cluster on MATLAB. 96
4.7 Post execution results as shown in SDK terminal after clustering operation. 97
4.8 Estimated Power Consumption of Clustering component. 98
4.9 Behavioural simulation of the Binning-Clustering co-design. 98
4.10 Post execution results as shown in SDK terminal after Preprocessing operation. 98
4.11 Estimated Power Consumption of the Integrated System. 99
4.12 Analytic overview of the consumed power for each individual block. 99
4.13 Hierarchical presentation of the FPGA resource utilisation. 100

List of Tables

1 Συγκριτικός Πίνακας με τις συχνότητες ρολογιού που επετεύχθησαν κατόπιν βελτιστοποιήσεων 23
2 Σύγκριση μέσων χρόνων εκτέλεσης μεταξύ των δύο προσεγγίσεων για κάθε ένα από τα υλοποιημένα στοιχεία 23
3 Χρήση πόρων του FPGA για το ολοκληρωμένο σύστημα. 24

2.1 Table of commercially available star trackers 39

3.1 Table of the achieved clock period per component after optimisation 89

4.1 Comparison of mean execution time between SoC FPGA and ARM CPU for Averaging Binning operation 94
4.2 Resource Utilisation of FPGA for Binning implemented design. 95
4.3 Comparison of mean execution time between SoC FPGA and ARM CPU for Clustering operation 97
4.4 Resource Utilisation of FPGA for Clustering implemented design. 98
4.5 Comparison of mean execution time between SoC FPGA and ARM CPU for the Integrated HW/SW Co-design 99
4.6 Resource Utilisation of FPGA for the Integrated System. 99

Εκτεταμένη Περίληψη

Εισαγωγή

Η παρούσα διπλωματική εργασία διεξήχθη σε συνεργασία με την Infinite Orbits, μια εταιρεία που δραστηριοποιείται στη διαστημική βιομηχανία και παρέχει καινοτόμες υπηρεσίες για δορυφορικά συστήματα. Οι διαστημικές εφαρμογές είναι γενικά ιδιαίτερα απαιτητικές διεργασίες και είναι ζωτικής σημασίας το να ικανοποιούνται συγκεκριμένες προδιαγραφές. Πρέπει να χαρακτηρίζονται από υψηλά επίπεδα αυτονομίας ώστε να ελαχιστοποιούνται οι καθυστερήσεις που εισάγονται από τα κανάλια επικοινωνίας και ελέγχου που βρίσκονται σε περιβάλλον εδάφους. Επιπλέον, τα δορυφορικά συστήματα πρέπει να μπορούν να λειτουργούν αδιαλείπτως και χωρίς προβλήματα ή έστω να είναι σε θέση να τα αντιμετωπίζουν με ασφάλεια και συνεπώς πρέπει να διακατέχονται από ευρωστία. Τέλος, πρέπει να είναι εξαιρετικά γρήγορα ώστε να ανταποκρίνονται άμεσα και να αντιμετωπίζουν ανεπιθύμητες καταστάσεις που μπορούν να οδηγήσουν σε ατυχήματα.

Το όργανο εκείνο που παρέχει τη δυνατότητα ταχείας και υψηλής ακρίβειας μέτρησης για τον προσδιορισμό του προσανατολισμού ενός δορυφορικού συστήματος είναι ο ανιχνευτής αστέρων. Ο ανιχνευτής αστέρων αποτελείται από ένα οπτικό σύστημα και από το ψηφιακό υλικό. Το οπτικό σύστημα περιλαμβάνει ένα ψηφιακό αισθητήρα εικόνας που τραβάει φωτογραφίες του ουρανού οι οποίες χρησιμοποιούνται από το υλικό για τον προσδιορισμό της θέσης του δορυφόρου. Το υλικό περιλαμβάνει κάποιο σύστημα επεξεργασίας που είναι συνήθως κάποιος μικροεπεξεργαστής, ο οποίος πραγματοποιεί τους απαραίτητους υπολογισμούς για να εξάγει χρήσιμες πληροφορίες. Η διαδικασία αυτή αποτελείται από επιμέρους υπολογιστικά στάδια μερικά από τα οποία παρουσιάζονται στα πλαίσια αυτής της διπλωματικής εργασίας. Γενικά, η ανίχνευση αστέρων χαρακτηρίζεται από υψηλή υπολογιστική πολυπλοκότητα η οποία ενισχύεται από το γεγονός οτι απαιτείται επεξεργασία μεγάλου όγκου δεδομένων αποτελώντας έτσι περιοριστικό παράγοντα στην απόδοση του συστήματος, ιδίως εάν η εκτέλεση πραγματοποιείται σε κάποιον συμβατικό μικροεπεξεργαστή.

Τα τελευταία χρόνια, η ανάγκη για αποδόσεις πραγματικού χρόνου έχει στρέψει το ενδιαφέρον σε άλλες εναλλακτικές δημιουργώντας νέες τάσεις στην βιομηχανία του διαστήματος. Πλέον, έχουμε ραγδαία αύξηση χρήσης εμπορικών συσκευευών υλικού «από το ράφι» σε διαστημικές εφαρμογές με την πιο διαδεδομένη από αυτές να είναι τα FPGA, αφού η αγορά παρέχει ήδη ένα μεγάλο εύρος πλατφορμών κατάλληλες για τέτοιου είδους εφαρμογές.

Στα πλαίσια διερεύνησης του πεδίου εφαρμογών των COTS FPGA στο διάστημα με τις οποίες ασχολείται το Εργαστήριο Μικροϋπολογιστών & Ψηφιακών Συστημάτων, διεκπεραιώθηκε η διπλωματική αυτή η οποία πραγματεύεται την ανάπτυξη αλγορίθμων σε ένα ενσωματωμένο σύστημα SoC FPGA οι οποίοι υλοποιούν τα πρώτα στάδια της προγραμματιστικής ροής ενός ανιχνευτή αστέρων. Αυτά τα στάδια αφορούν την προεπεξεργασία δεδομένων και χωρίζονται στις εξής δύο υποδιεργασίες: την ομαδοποίηση των εικονοστοιχείων με σκοπό τη μείωση των διαστάσεων της υπό επεξεργασία εικόνας και την ανίχνευση συστάδων αστεριών εντός του πλαισίου αυτής. Οι εξαγόμενες αυτές συστάδες χρησιμοποιούνται στα επόμενα στάδια για τον υπολογισμό του ύψους του δορυφόρου. Σκοπός είναι να αξιοποιήσουμε τις δυνατότητες της πλατφόρμας του υλικού προτείνοντας μια αρχιτεκτονική που συνδυάζει τόσο το λογισμικό όσο και το υλικό του SoC FPGA.

Θεωρητικό υπόβραθρο

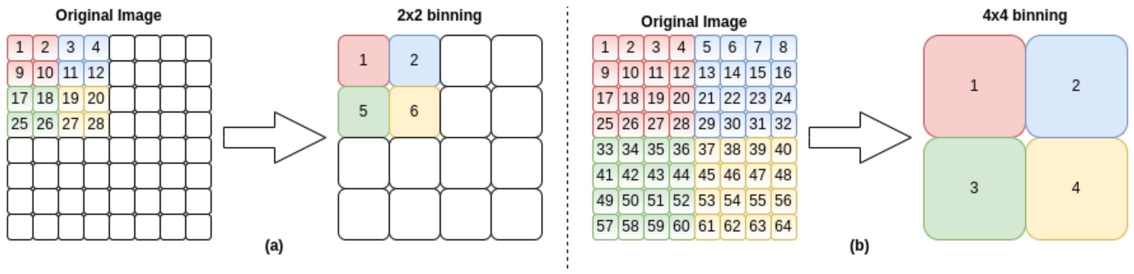
Οι ανιχτευτές αστέρων χρησιμοποιούνται για τον προσδιορισμό της θέσης ενός διαστημικού σκάφους, επειδή παρέχουν τη μεγαλύτερη ακρίβεια από οποιονδήποτε άλλον αισθητήρα. Προκειμένου να μεγιστοποιηθεί η ακρίβεια της εκτίμησης της θέσης ενός αστεριού, η λάμψη που εκπέμπει αυτό αποκεντράρεται σε μάσκες εικονοστοιχείων μεγέθους από 3×3 έως 15×15 ανάλογα με τα χαρακτηριστικά του αισθητήρα εικόνας. Η εξάπλωση του φωτός που αποτυπώνεται στα εικονοστοιχεία της εικόνας του αισθητήρα περιγράφεται από μαθηματικές συναρτήσεις που ονομάζονται PSF (Point Spread Functions) οι οποίες είναι ουσιαστικά Γκαουσιανές συναρτήσεις. Εν συνεχεία, χρησιμοποιούνται κατάλληλοι αλγόριθμοι αναγνώρισης προτύπων οι οποίοι με την κατάλληλη επεξεργασία των δεδομένων μπορούν να εκτιμήσουν τον προσανατολισμό του συστήματος με ακρίβεια εικονοστοιχείου κάτω από 1σ.

Στη διαστημική πλοήγηση απαιτείται ένα σταθερό σύστημα συντεταγμένων ως σημείο αναφοράς ώστε να υπάρχει συσχέτιση με το αντίστοιχο σύστημα του κινούμενου διαστημικού οχήματος. Το σύστημα που χρησιμοποιείται ονομάζεται Earth Central Inertial (ECI) και η αρχή των αξόνων του είναι ακριβώς στο κέντρο της Γης, χωρίς να ακολουθεί όμως την κίνησή της. Ο ανιχνευτής αστέρων συσχετίζει το ECI με το σύστημα συντεταγμένων του οχήματος που τοποθετείται στο κέντρο μάζας του προκειμένου να εξάγει πληροφορίες σχετικά με τη θέση αυτού. Προκειμένου να γίνει η συσχέτιση της θέσης του ανιχνεύοντος αστέρα στο πλαίσιο της εικόνας με την πραγματική του θέση στον ουρανό, απαιτούνται κάποιοι μετασχηματισμοί στα αντίστοιχα συστήματα συντεταγμένων. Αρχικά, οι συντεταγμένες του αστέρα στο ECI σύστημα πρέπει να μεταφερθούν στο αντίστοιχο σύστημα συντεταγμένων του οπτικού αισθητήρα. Έπειτα, πρέπει να γίνει η προβολή της θέσης του αστέρα από τον τρισδιάστατο χώρο στο πλαίσιο της δισδιάστατης εικόνας, το οποίο επιτυγχάνεται με τη χρήση του μοντέλου κάμερας pinhole, το οποίο παρέχει μια απλή περιγραφή της σχέσης μεταξύ των τρισδιάστατων συντεταγμένων ενός σημείου και της προβολής του σε μια ιδανικής κάμερας η οποία θεωρείται σημείο αναφοράς. Το τελευταίο βήμα αφορά τη μεταφορά από το σύστημα συντεταγμένων της εικόνας στο σύστημα συντεταγμένων του εικονοστοιχείου (Εικόνα 2.11).

Η παρούσα διπλωματική εργασία επικεντρώνεται στην υλοποίηση ενός αλγορίθμου για ανιχνευτές αστέρων σε μια συσκευή υλικού SoC FPGA αξιοποιώντας τις δυνατότητες που προσφέρει λόγω της ευελιξίας και της απόδοσής της. Οι συσκευές SoC FPGA συνιστούν μια καινοτόμα βελτιωμένη προσέγγιση στο χώρο των πλατφορμών υλικού καθώς περιλαμβάνουν στο ίδιο ολοκληρωμένο chip έναν επεξεργαστή καθώς επίσης και ένα FPGA. Κατά συνέπεια, έχουν μεγαλύτερο βαθμό ολοκλήρωσης, χαμηλότερη ισχύ και υψηλότερο εύρος ζώνης επικοινωνίας μεταξύ του επεξεργαστή και του FPGA. Ως εκ τούτου, τα SoC FPGA συνιστούν μια ιδιαίτερα ανταγωνιστική εναλλακτική προσέγγιση έναντι των συμβατικών μέσων υλικού της αγοράς. Για τις ανάγκες της διπλωματικής αυτής, χρησιμοποιήθηκε η πλακέτα Zedboard της AVNET η οποία περιλαμβάνει το σύστημα Xilix Zynq-7020 SoC.

Υλοποίηση του στοιχείου για την Ομαδοποίηση Εικονοστοιχείων

Το πρώτο στοιχείο του συνδυαστικού μας συστήματος υλικού/λογισμικού είναι υπεύθυνο για την ομαδοποίηση εικονοστοιχείων σε δισδιάστατη εικόνα. Η τεχνική αυτή είναι ευρέως διαδεδομένη σε εφαρμογές που βασίζονται σε συστήματα οπτικών αισθητήρων και αφορά το συνδυασμό των τιμών γειτονικών εικονοστοιχείων ώστε να σχηματιστεί ένα υπερ-εικονοστοιχείο και να αυξηθεί κατά κάποιον τρόπο το μέγεθος του αισθητήρα εικόνας. Ως εκ τούτου, πετυχαίνουμε την αποτελεσματική αύξηση της ευαισθησίας του εικονοστοιχείου που προκύπτει με κόστος τη μειωμένη χωρική ανάλυση. Επιπλέον, σε εφαρμογές όπου επεξεργάζονται εικόνες μεγάλου μεγέθους είναι πολύ ευεργετική τεχνική για την απόδοση του συστήματος καθώς οι διαστάσεις της εικόνας μειώνονται σημαντικά. Έτσι, όχι μόνο μειώνεται ο χρόνος επεξεργασίας, αλλά και οι απαιτούμενοι πόροι ελαττώνονται σημαντικά.

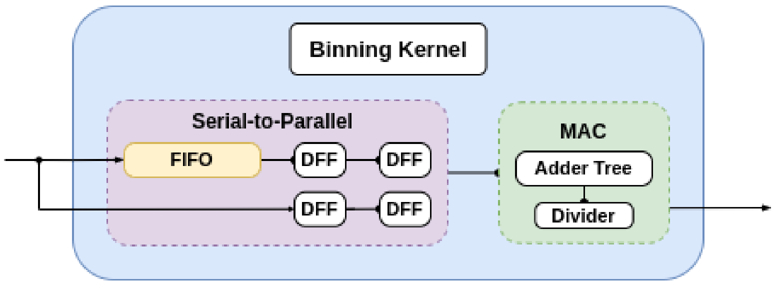


Σχήμα 1: Διαφορετικά επίπεδα ομαδοποίησης εικονοστοιχείων.

Το συγκεκριμένο μπλοκ που υλοποιήσαμε, λειτουργεί σε περιοχές εικόνας 2×2 ή 4×4 με βηματισμό 2 ή 4 αντίστοιχα και υπολογίζει τη μέση τιμή τους. Λαμβάνοντας υπόψη τις απαιτήσεις του πρότζεκτ, επιλέξαμε να προχωρήσουμε με ομαδοποίηση 2×2 κρίνοντας πως είναι καταλληλότερη επιλογή για τη διατήρηση της ισορροπίας μεταξύ της ανάλυσης της εικόνας και της ευαισθησία της.

Όπως έχει ήδη αναφερθεί, η προτεινόμενη αρχιτεκτονική του συστήματός μας συνδυάζει τόσο το υλικό (PL) όσο και το λογισμικό (PS) του Zynq SoC FPGA. Για λόγους απλότητας, στην παρούσα εφαρμογή οι εικόνες είναι αποθηκευμένες στο μνήμη της πλατφόρμας, αντί να λαμβάνονται από τον αισθητήρα εικόνας. Και στις δύο περιπτώσεις, ο επεξεργαστής είναι ο διαμεσολαβητής ώστε να προωθούνται τα δεδομένα από τη μνήμη ή την κάμερα στην προγραμματιστική λογική. Ως εκ τούτου, πρέπει να διαμορφωθεί καταλλήλως το σύστημα ώστε να επιτυγχάνεται η επικοινωνία μεταξύ των δύο διαφορετικών υπο-στοιχείων της πλατφόρμας. Επομένως, το στοιχείο μας έχει διαμορφωθεί ώστε να υποστηρίζει το πρωτόκολλο AXI4-Stream ώστε να μπορεί να λαμβάνει έγκυρα δεδομένα από το PS.

Η μονάδα ομαδοποίησης εικονοστοιχείων αποτελείται από δύο βασικά υπο-κυκλώματα. Τα εισερχόμενα εικονοστοιχεία της εικόνας λαμβάνονται σειριακά ακολουθώντας σάρωση τύπου raster, δηλαδή από δεξιά προς τα αριστερά και από πάνω προς τα κάτω στο πλαίσιο της εικόνας. Συνεπώς, απαιτείται ένα κύκλωμα που θα λαμβάνει τα εικονοστοιχεία σειριακά και με κάποιο τρόπο θα τα προωθεί στα επόμενα στάδια παράλληλα. Εν προκειμένω, τα εικονοστοιχεία λαμβάνονται ένα-ένα και ομαδοποιούνται σε τετράδες ώστε να υπολογιστεί η μέση τιμή τους και να προκύψει ένα υπερ-εικονοστοιχείο. Ένα ακόμα ζήτημα εδώ είναι πως πρέπει να σαρωθεί μια ολόκληρη γραμμή εικόνας ώστε να μπορέσει να ξεκινήσει η διαδικασία υπολογισμού των μέσων τιμών. Οι γραμμές επεξεργάζονται σε ζευγάρια και η πρώτη από κάθε ζεύγος αποθηκεύεται προσωρινά σε έναν FIFO buffer. Αυτό προσθέτει σημαντική καθυστέρηση στη λειτουργική απόδοση του συστήματος. Παρ᾽ όλ᾽ αυτά, ο επιταχυντής FPGA υποστηρίζει παράλληλη επεξεργασία, επομένως ο υπολογισμός των μέσων τιμών κάθε πλέγματος 2×2 και η αποθήκευση της επόμενης γραμμής εικόνας συμβαίνουν ταυτόχρονα.



Σχήμα 2: Σχεδιαστική απεικόνιση του πυρήνα Ομαδοποίησης Εικονοστοιχείων.

Το έτερο μπλοκ είναι η μονάδα πολλαπλασιαστή-συσσωρευτή (MAC) η οποία λαμβάνει παράλληλα τις τιμές των εικονοστοιχείων της 2×2 ή 4×4 γειτονιάς. Όπως βλέπουμε στην Εικόνα 2, το πρώτο υπο-κύκλωμα είναι μια δομή δέντρου από πλήρεις αθροιστές που πραγματοποιεί ταχύτατα προσθέσεις αξιοποιώντας τον παραλληλισμό. Στη συνέχεια, το συνολικό άθροισμα μεταβιβάζεται στον Διαιρέτη ο οποίος υπολογίζει τη μέση τιμή διαιρώντας απλώς με το 4 (ή με το 16). Γενικότερα στον προγραμματισμό υπολογιστών, οι αριθμητικές μετατοπίσεις συνιστούν έναν πολύ αποτελεσματικό τρόπο για τις πράξεις του πολλαπλασιασμού ή της διαίρεσης. Η μετατόπιση των bit προς τα αριστερά κατά n θέσεις σε έναν δυαδικό αριθμό έχει ως αποτέλεσμα τον πολλαπλασιασμό του επί 2n, ενώ αντίστοιχα η μετατόπιση των bit προς τα δεξιά κατά n θέσεις καταλήγει σε διαίρεση με το 2n. Συνεπώς, ο διαιρέτης απλώς εκτελεί μια αριθμητική μετατόπιση προς τα δεξιά κατά N-bit στο άθροισμα εξόδου του δέντρου αθροιστή, όπου N είναι η αντίστοιχη διάσταση του τετραγωνισμένου πλέγματος που χρησιμοποιείται στην ομαδοποίηση. Η έξοδος του κυκλώματος αναφέρεται στο νέο υπερ-εικονοστοιχείο της παραγόμενης εικόνας.

Συνδυαστική υλοποίηση HW/SW για τη Συσταδοποίηση

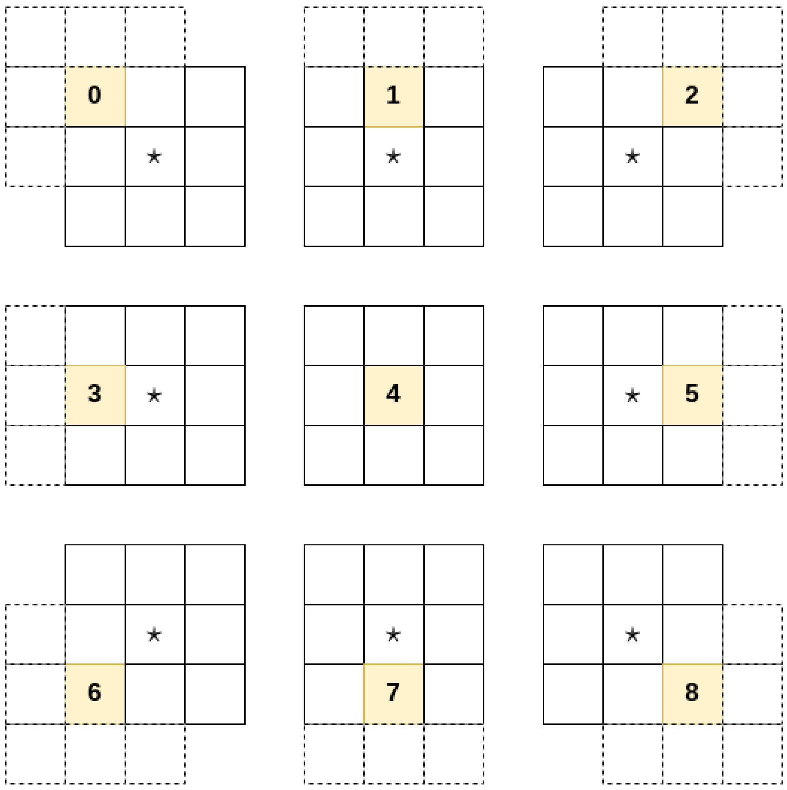
Το κύριο θέμα που πραγματεύεται η παρούσα διπλωματική είναι η υλοποίηση ενός αποδοτικού αλγορίθμου για συσταδοποίηση των εικονοστοιχείων που συγκροτούν αστέρι. Αυτό είναι το πρώτο στάδιο της προγραμματιστικής ροής ενός Ανιχνευτή Αστέρων. Ο παραδοσιακός τρόπος προσέγγισης για ανίχνευση των αστέρων στην εικόνα περιγράφεται ακολούθως. Αρχικά, κάθε εικονοστοιχείο συγκρίνεται με ένα σαφώς καθορισμένο κατώφλι και εάν η τιμή του είναι μεγαλύτερη ή ίση, τότε ορίζεται καταλλήλως μια περιοχή ενδιαφέροντος εντός της οποίας πραγματοποιείται η αναζήτηση του αστέρα. Κατα τη διάρκεια της αναζήτησης εντός της περιοχής ενδιαφέροντος χρησιμοποιείται ένα άλλο, συνήθως μικρότερο κατώφλι για τη σύγκριση των τιμών των εικονοστοιχείων. Τα γειτονικά εικονοστοιχεία που βρίσκονται πάνω από το λεγόμενο κατώφλι αύξησης, συγκεντρώνονται σε ομάδες και σχηματίζουν μια συστάδα σε ένα μέγιστο πλέγμα 5×5 εικονοστοιχείων. Ο σχεδιασμός πάνω στο υλικό είναι ιδιαίτερα απαιτητικός εξαιτίας των δομών δεδομένων και της έλλειψης χρήσιμων συναρτήσεων. Επειδή λοιπόν ο αλγόριθμος συσταδοποίησης είναι από τη φύση του επαναληπτικός και χρησιμοποιεί αναδρομικές υπορουτίνες, είναι πολύ σημαντική η βελτιστοποίησή του ώστε να έχει χαμηλούς χρόνους εκτέλεσης.

Μηχανισμός Αναζήτησης Εικονοστοιχείων

Μετά την κατωφλίωση με το πρωταρχικό κατώφλι, επιτελείται η αναζήτηση στην περιοχή ενδιαφέροντος για κάθε ένα εικονοστοιχείο που εντοπίστηκε. Προκειμένου να επιταχυνθεί η διαδικασία αυτή, προτείναμε έναν μηχανισμό αναζήτησης με σκοπό τη μείωση των απαιτούμενων κύκλων ρολογιού. Έτσι, θεωρούμε τα εικονοστοιχεία που προέκυψαν από το πρώτο κατώφλι ως σημεία εκκίνησης της αναζήτησης και στο εξής θα αναφέρονται ως 'εικονοστοιχεία εκκίνησης'. Αρχικά, για κάθε εικονοστοιχείο εκκίνησης ελέγχονται τα 8 γειτονικά του εικονοστοιχεία στο πλέγμα 3×3 συγκρίνοντας τις τιμές τους με το επονομαζόμενο 'αυξανόμενο κατώφλι' και στη συνέχεια πραγματοποιείται η ίδια διαδικασία για κάθε ένα από αυτά που έχουν τιμή μεγαλύτερη ή ίση αυτού. Η διαφορά όμως έγκειται στο οτι κάποια από τα γειτονικά εικονοστοιχεία της 3×3 περιοχής έχουν ήδη ελεγχθεί προηγουμένως. Η διαδικασία αναζήτησης των εικονοστοιχείων περιορίζεται στις εξής 9 περιπτώσεις που αποτυπώνονται στην παρακάτω Εικόνα 3. Κάθε εικονοστοιχείο που εξετάζεται φέρει έναν αριθμό που υποδηλώνει τη σχετική του θέση στη γειτονιά στην οποία ανιχνεύτηκε ώστε να αναγνωρίζεται σε ποιά από τις 9 περιπτώσεις βρίσκεται και να ελέγχονται τα αντίστοιχα εικονοστοιχεία.

Διαδικασία λήψης δεδομένων εικόνας

Στο σημείο αυτό θα αναλύσουμε λίγο περισσότερο σε βάθος τα επιμέρους στοιχεία που απαρτίζουν τη μονάδα Συσταδοποίησης. Πρωτίστως, να αναφέρουμε πως σε αυτήν την περίπτωση χρησιμοποιήθηκαν τα πρωτόκολλα επικοινωνίας AXI4 Stream καθώς επίσης και AXI4 Lite. Όσον αφορά το πρώτο πρωτόκολλο, εξυπηρετεί την επικοινωνία με το μπλοκ Ομαδοποίησης εικονοστοιχείων απ΄ το οποίο λαμβάνει τα εικονοστοιχεία της νέας επεξεργασμένης εικόνας, ενώ το δεύτερο εξυπηρετεί τη δυναμική προσαρμογή κατωφλίου λαμβάνοντας τις τιμές από έναν συγκεκριμένο καταχωρητή στη μνήμη του SoC FPGA. Προκειμένου να είναι δυνατή η αναζήτηση των αστεριών εντός της εικόνας, είναι απαραίτητη η προσωρινή της αποθήκευση

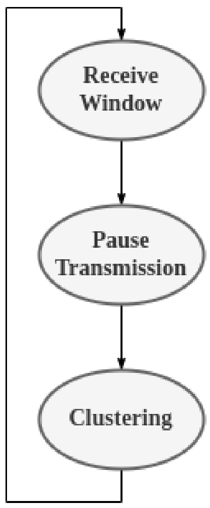


Σχήμα 3: Μηχανισμός αναζήτησης εικονοστοιχείων.

στο τμήμα του PL όπου εκτελείται η συσταδοποίηση. Ωστόσο, εξαιτίας των περιορισμένων πόρων του FPGA, καθίσταται αδύνατη αποθήκευση ολόκληρης της εικόνας ακόμα και μετά την ομαδοποίηση των εικονοστοιχείων που υποδιπλασιάζει τις διαστάσεις της. Συνεπώς, η εικόνα στέλνεται τμηματικά σε κομμάτια που ονομάζουμε 'κυλιόμενα παράθυρα' και έτσι πραγματοποιείται η αναζήτηση συστάδων. Συνεπώς η διαδικασία επεξεργασίας της εικόνας παρουσιάζεται στο παρακάτω σχήμα. Τα κυλιόμενα παράθυρα είναι επικαλυπτόμενα προκειμένου να είναι κατά το δυνατόν πιο ακριβής η αναζήτηση των συστάδων. Η επικάλυψη είναι προκαθορισμένη και παραμετροποιήσιμη και σχετίζεται άμεσα με το μέγεθος των συστάδων που είναι προς αναζήτηση. Πιο συγκεκριμένα, η περιοχή ενδιαφέροντος ορίζεται γύρω από το εικονοστοιχείο εκκίνησης οριοθετώντας γύρω του μια τετραγωνική περιοχή $7{\times}7$. Η τιμή αυτή προκύπτει ώστε να είναι δυνατή η ανίχνευση συστάδων σε πλέγμα μέχρι και $5{\times}5$. Έτσι λοιπόν, εάν το εικονοστοιχείο εκκίνησης βρίσκεται στις τελευταίες γραμμές του παραθύρου, δεν μπορεί να οριστεί η απαραίτητη περιοχή ενδιαφέροντος και χωρίς βλάβη της γενικότητας, θεωρούμε οτι η συστάδα εντοπίζεται καλύτερα στις παρακάτω γραμμές της εικόνας οπότε και εξετάζεται στο επόμενο παράθυρο.

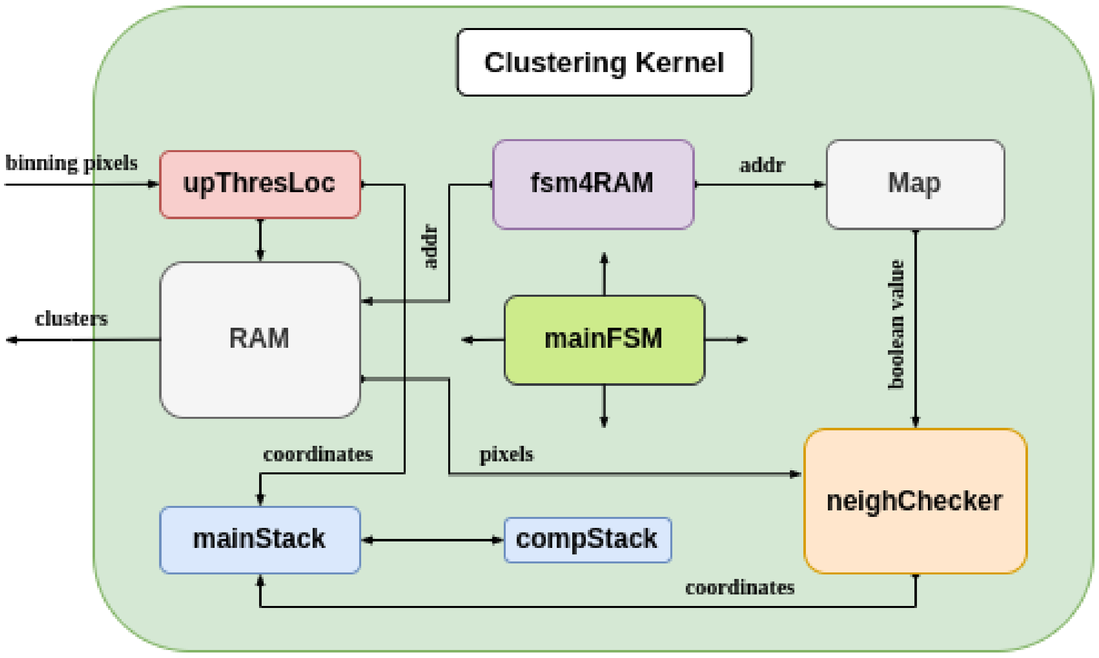
Περιγραφή του πυρήνα Συσταδοποίησης

Το περιγραφικό διάγραμμα του πυρήνα Συσταδοποίησης απεικονίζεται παρακάτω στην Εικόνα 5. Όλα τα επιμέρους στοιχεία του μπλοκ ελέγχονται από μια μηχανή πεπερασμένης κατάστασης (FSM). Αρχικά, τα εισερχόμενα ομαδοποιημένα εικονοστοιχεία εξετάζονται από το upThresLoc στο οποίο πραγματοποιείται η πρώτη κατωφλίωση, πρωτού καταλήξουν στη



Σχήμα 4: Διάγραμμα ροής για τη λήψη των δεδομένων εικόνας.

μνήμη RAM όπου και αποθηκεύονται προσωρινά. Οι συντεταγμένες των εικονοστοιχείων που προκύπτουν από την πρώτη κατωφλίωση αποθηκεύονται στην αντίστοιχη στοίβα. Οι mainStack και compStack είναι δομές που αποτελούνται από 3 στοίβες εκ των οποίων οι δύο είναι για τις συντεταγμένες i, j των εικονοστοιχείων και η τρίτη αναφέρεται στη σχετική θέση αυτών που ανιχνεύτηκαν στην 3×3 γειτονιά αναζήτησης. Τα δύο κύρια μπλοκ είναι αυτό της fsm4ram και του neighChecker. Όσον αφορά το πρώτο, η λειτουργία του χωρίζεται σε 3 κατευθύνσεις οι οποίες είναι οι εξής: η πρώτη αφορά τον υπολογισμό διευθύνσεων στις οποίες θα γράφονται τα αντίστοιχα εισερχόμενα εικονοστοιχεία στη μνήμη, η δεύτερη αφορά τον υπολογισμό των κατάλληλων διευθύνσεων των εικονοστοιχείων κατά τη διαδικασία συσταδοποίησης και η τρίτη σχετίζεται με τον υπολογισμό διευθύνσεων κατά την εξαγωγή της συστάδας που ανιχνεύτηκε. Σχετικά με το δεύτερο και σημαντικότερο υποστοιχείο, αυτό επιτελεί ουσιαστικά τον έλεγχο της συσταδοποίησης. Δέχεται τις τιμές των κατάλληλων εικονοστοιχείων στη γειτονιά αναζήτησης και τις συγκρίνει με το αυξανόμενο κατώφλι. Εάν προκύψει οτι κάποιο απ΄ τα εικονοστοιχεία ανήκει στη συστάδα, τότε εξάγονται οι συντεταγμένες αυτού ώστε να ελεγχθούν μετέπειτα τα δικά του γειτονικά εικονοστοιχεία. Παράλληλα, επιτελείται και η διαδικασία υπολογισμού των διαστάσεων της συστάδας. Αρχικά, ορίζονται 4 μεταβλητές με τις συντεταγμένες του εικονοστοιχείου εκκίνησης οι οποίες αντιστοιχούν στο πάνω, κάτω, δεξί και αριστερό άκρο της συστάδας καταγράφοντας ουσιαστικά τις κινήσεις που πραγματοποιούνται κατά την αναζήτηση εντός της περιοχής ενδιαφέροντος. Όταν ολοκληρωθεί η διαδικασία υπολογίζεται η διάσταση της τετραγωνικής συστάδας και οι συντεταγμένες του άνω αριστερού εικονοστοιείου αυτής. Στο σημείο αυτό, αξίζει να σημειωθεί πως είναι πιθανό να εντοπιστεί ένα μόνο εικονοστοιχείο εντός της περιοχής ενδιαφέροντος. Σε αυτήν την περίπτωση θεωρούμε πως δεν συνιστά έγκυρη συστάδα οπότε και απορρίπτεται. Επιπλέον, δεν αποκλείεται να προκύψει και συστάδα σε πλέγμα 7×7, όμως σε αυτήν την περίπτωση εξαιτίας των προδιαγραφών που δόθηκαν από την Infinite Orbits, επιλέγεται ένα τμήμα 5×5 εντός του πλέγματος αυτού το οποίο και εξάγεται. Τα υπόλοιπα στοιχεία του πυρήνα συσταδοποίησης είναι η μνήμη στην οποία αποθηκεύονται τα κυλιόμενα παράθυρα και ένα ακόμα που ονομάζεται Map και ουσιαστικά είναι κι αυτός μια μονάδα μνήμης. Ως ένας τρόπος για βελτιστοποίηση του αλγορίθμου αναζήτησης, εισήχθη το συγκεκριμένο στοιχείο στο οποίο καταγράφονται τα εικονοστοιχεία που έχουν ελεγχθεί προκειμένου να αποφεύγονται επαναληπτικοί έλεγχοι για αυτά στην περιοχή ενδιαφέροντος. Αυτό επίσης υποδηλώνει πως ένα εικονοστοιχείο θα ελεγχθεί στο πλαίσιο μίας και μόνο συστάδας.

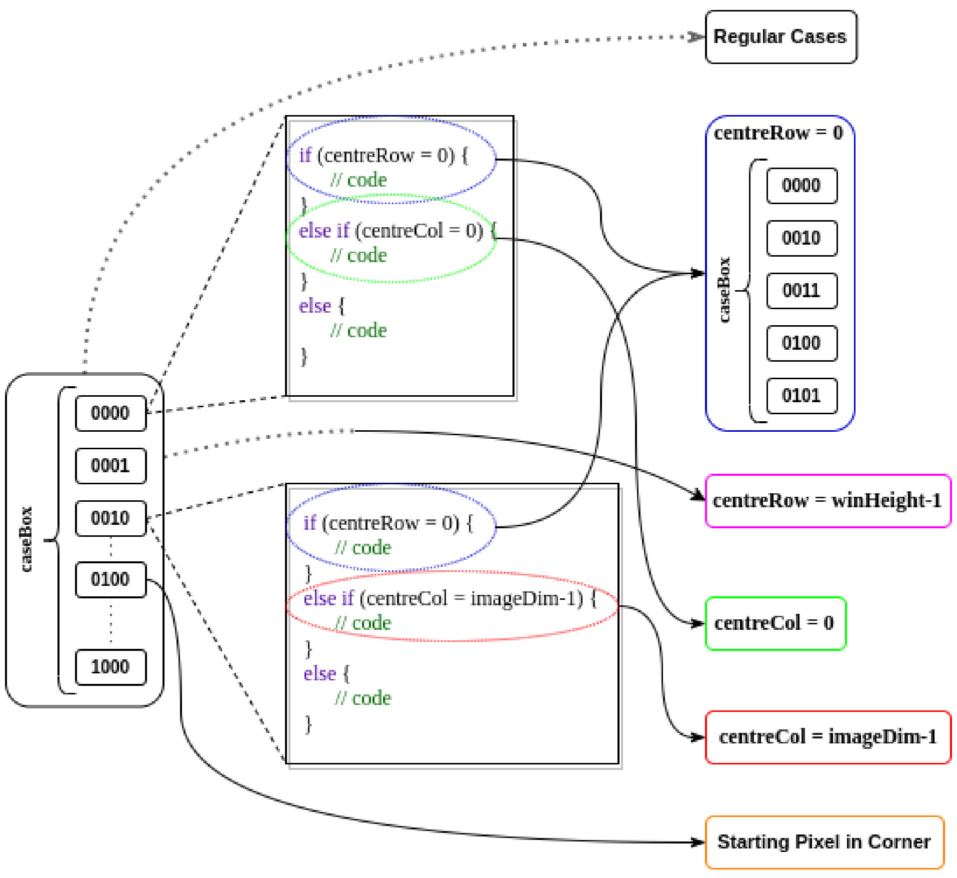


Σχήμα 5: Σχεδιαστική αναπαράσταση του πυρήνα Συσταδοποίησης.

<u>Βελτιστοποιήσεις ως προς την απόδοση</u>

Στο σημείο αυτό θα παρουσιάσουμε μερικές από τις βελτιστοποιήσεις που πραγματοποιήσαμε. Κατα τη διαδικασία ανάγνωσης, υπάρχουν ορισμένες ακραίες περιπτώσεις όσον αφορά τη θέση του κεντρικού εικονοστοιχείου που διαφοροποιούν τη διαμόρφωση της γειτονιάς αναζήτησης. Αυτές οι περιπτώσεις εμφανίζονται όταν το κεντρικό εικονοστοιχείο βρίσκεται στα άκρα του παραθύρου που σημαίνει ότι συνορεύει με λιγότερα από 8 εικονοστοιχεία, οπότε δεν μπορεί να σχηματιστεί γειτονιά 3×3. Στην περίπτωσή μας, ο αρχικός κώδικα γράφτηκε με τέτοιον τρόπο που περιείχε πολλά εμφωλευμένα if-else. Αυτός ο τρόπος συγγραφής κώδικα με βάση τον προγραμματισμό σε γλώσσες υψηλού επιπέδου είναι εξαιρετικά αναποτελεσματικός για σχεδιασμό υλικού καθώς επηρεάζει τον τρόπο με τον οποίο αποτυπώνεται η λογική πάνω σε αυτό. Υποθέτοντας πως κάθε κατάσταση if αντιπροσωπεύει μια διεργασία που εκτελείται από ένα FSM, ομαδοποιούμε τις παρόμοιες διεργασίες που εκτελούνται υπό την ίδια συνθήκη. Αυτή η τεχνική μπορεί να ερμηνευθεί ως ένας τρόπος καταμερισμού εργασιών και οδηγεί σε ενίσχυση του παραλληλισμού. Αυτή η βελτιστοποίηση γίνεται περισσότερο κατανοητή με την παρακάτω εικόνα 6.

Όπως περιγράφηκε προηγουμένως, υπάρχει ισχυρή συσχέτιση μεταξύ των εργασιών που εκτελούνται από το fsm4RAM και το neighChecker. Επομένως, η ίδια λογική που περιγράφηκε



Σχήμα 6: Αναπαράσταση καταμερισμού διεργασιών.

παραπάνω υιοθετήθηκε και από τη μονάδα neighChecker, ώστε να σχηματίζονται έξι επιμέρους μπλοκ για αύξηση του παραλληλισμού. Ωστόσο, η ιδέα μπορεί να επεκταθεί και στα χαμηλότερα επίπεδα της σχεδίασης. Για κάθε εισερχόμενη τιμή του εικονοστοιχείου, πραγματοποιούνται τρεις λειτουργίες ελέγχου, ώστε να μπορεί να θεωρηθεί το εικονοστοιχείο ως μέρος μιας συστάδας. Αυτά τα τρία βήματα ελέγχου περιγράφονται ως εξής:

- Σύγκριση με το αυξανόμενο κατώφλι και αποφυγή επανελέγχου
- Έλεγχος ώστε το προς έλεγχο εικονοστοιχείο να βρίσκεται εντός της περιοχής ενδιαφέροντος
- Εξαγωγή συντεταγμένων και ενημέρωση των διαστάσεων της συστάδας

Έτσι, αντί να εκτελούνται αυτά τα τρία αλληλοεξαρτώμενα στάδια ελέγχου, διαχωρίζονται σε τρεις μεμονωμένες και ανεξάρτητες διεργασίες. Καθένα από τα έξι υποσυστήματα που σχηματίζονται εκτελεί αυτά τα ξεχωριστά στάδια ελέγχου. Κάθε ένα εξ αυτών ενεργεί σαν να ικανοποιούνται οι υπόλοιπες δύο συνθήκες ελέγχου και προκύπτει ένα τελικό σήμα ώστε εάν ικανοποιούνται και οι τρεις τα αποτελέσματα που εξάγονται να θεωρούνται έγκυρα. Μετά τα βήματα βελτιστοποιήσεων που περιγράφησαν παραπάνω, καθώς και κάποια άλλα που εξετάζονται αναλυτικά στο κύριο μέρος της διπλωματικής εργασίας προέκυψαν τα παρακάτω αποτελέσματα που αναγράφονται στον Πίνακα 1.

Πίνακας 1: Συγκριτικός Πίνακας με τις συχνότητες ρολογιού που επετεύχθησαν κατόπιν βελτιστοποιήσεων

Στοιχείο	**Περίοδος Ρολογιού**	**Βελτιστοποιημένη Περίοδος Ρολογιού**	**Ποσοστό Βελτίωσης**
upperThresLoc	3.7 ns	2.4 ns	35.1%
fsm4RAM	5 ns	5 ns	30%
clusterCal	4.3 ns	3.2 ns	25.6%
neighChecker	9.4 ns	4.4 ns	53.2%
Clustering kernel	11.2 ns	4.3 ns	61.6%

Πειραματική Αξιολόγηση

Η πλακέτα που χρησιμοποιήθηκε για τη πειραματική αξιολόγηση του συστήματος που υλοποιήσαμε είναι το Zedboard το οποίο περιλαμβάνει το Zynq-7020 SoC FPGA. Η σχεδίαση του υλικού πραγματοποιήθηκε στο εργαλείο Vivado Design Suite v2019.1 της Xilinx ενώ επιπλέον χρησιμοποιήθηκε το συμπληρωματικό εργαλείο Xilinx Software Development Kit (XSDK) για τη δημιουργία ενσωματωμένων εφαρμογών που εκτελούνται στον διπύρηνο επεξεργαστή ARM Cortex του Zynq SoC, έτσι ώστε να εδραιωθεί την επικοινωνία PS-PL. Στο πλαίσιο της παρούσας διπλωματικής, τα πειράματα που πραγματοποιήθηκαν επικεντρώνονται στα στάδια προεπεξεργασίας των δεδομένων ενός ανιχνευτή αστέρων δίνοντας έμφαση στην επιτάχυνση του αλγορίθμου συσταδοποίησης. Όπως έχει ήδη αναφερθεί χρησιμοποιήθηκαν εικόνες σε μορφή κλίμακας του γκρι με ανάλυση 2048×2048, αν και το σύστημα εξετάστηκε και με εικόνες μικρότερου μεγέθους. Το σύνολο δεδομένων εισόδου δημιουργήθηκε από ένα αποθετήριο που περιέχει το πλήρες αρχείο καταγεγραμμένων φωτογραφιών από τη διαστημική αποστολή Cassini της NASA που ελήφθησαν από το Φεβρουάριο του 2004 έως το Σεπτέμβρίο του 2017 [14]. Στον παρακάτω συγκεντρωτικό Πίνακα 1 παρουσιάζονται τα αποτελέσματα όσον αφορά την απόδοση των επι μέρους συστημάτων για τις δύο διαφορετικές προσεγγίσεις οι οποίες αφορούν το συνδυασμό υλικού/λογισμικού στο Zynq SoC FPGA, καθώς επίσης και την υλοποίηση λογισμικού που εκτελείται στον ARM επεξεργαστή της πλατφόρμας.

Πίνακας 2: Σύγκριση μέσων χρόνων εκτέλεσης μεταξύ των δύο προσεγγίσεων για κάθε ένα από τα υλοποιημένα στοιχεία

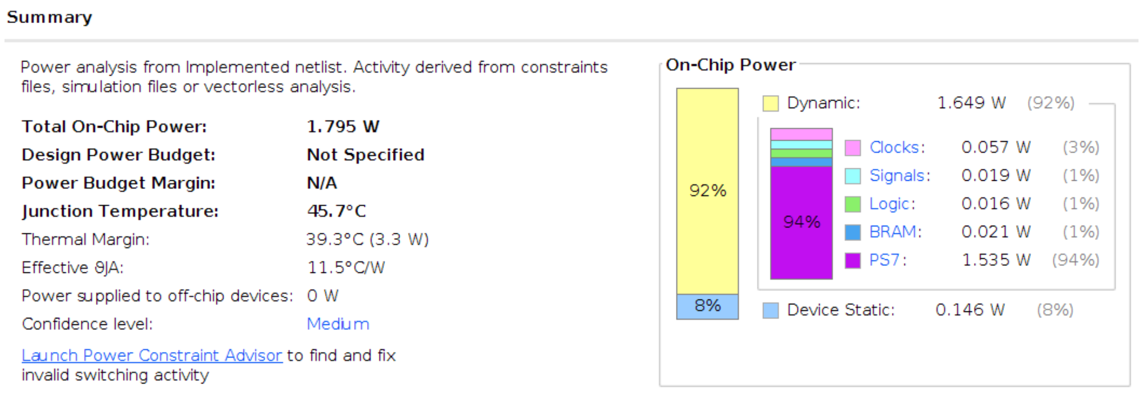
Σύστημα	**ARM**	**SoC**	**Επιτάχυνση**	**Εκτιμώμενος χρόνος εκτέλεσης**	**Εκτιμώμενη επιτάχυνση**
Σύστημα Ομαδοποίησης	1607.5 ms	33.6 ms	47.9x	16.8 ms	95.8x
Σύστημα Συσταδοποίησης	383.1 ms	8.9 ms	43.1x	5.11 ms	75.4x
Πλήρες Σύστημα	2050.3 ms	34.1 ms	60.2x	18.9 ms	108.2x

Ο παρακάτω συγκεντρωτικός πίνακας παρουσιάζει μια σύνοψη της χρήσης πόρων του FPGA μετά την αποτύπωση της λογικής στο υλικό.

Πίνακας 3: Χρήση πόρων του FPGA για το ολοκληρωμένο σύστημα.

Πόροι	**Χρήση**	**Διαθέσιμοι**	**Χρήση (%)**
LUT	8837	53200	16.61
LUTRAM	849	17400	4.88
FF	11733	106400	11.03
BRAM	20	140	14.29

Τέλος, η Εικόνα 7 παρουσιάζει την εκτιμώμενη κατανάλωση ισχύος του FPGA η οποία υπολογίζεται περίπου στα 1.795 Watt. Όπως βλέπουμε, η δυναμική ισχύς που καταναλώνεται αφορά το μεγαλύτερο μέρος της συνολικής ισχύος σε ποσοστό 92%, ενώ το αντίστοιχο ποσοστό της στατικής ισχύος εκτιμάται σε μόλις 8%. Επιπλέον, αξίζει να αναφερθεί πως όσον αφορά τη δυναμική ισχύ, για το μεγαλύτερο μέρος της υπεύθυνο είναι σύστημα του επεξεργαστή του Zynq SoC FPGA σε ποσοστό περίπου 94%.


Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.795 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**45.7°C**
Thermal Margin:	39.3°C (3.3 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.649 W	(92%)	
	Clocks:	0.057 W	(3%)
	Signals:	0.019 W	(1%)
	Logic:	0.016 W	(1%)
	BRAM:	0.021 W	(1%)
	PS7:	1.535 W	(94%)
Device Static:	0.146 W	(8%)	

Σχήμα 7: Εκτιμώμενη κατανάλωση ισχύος για το ολοκληρωμένο σύστημα.

Συμπερασματικά, η προτεινόμενη αρχιτεκτονική σχεδίασης χρησιμοποιεί λιγότερο από 20% των πόρων του FPGA για κάθε συστατικό στοιχείο. Ως εκ τούτου, δίνεται η δυνατότητα ανάπτυξης πρόσθετων στοιχείων HDL για την ολοκλήρωση των αλγοριθμικών βημάτων του ανιχνευτή αστέρων. Σε τελική ανάλυση, υλοποιήσαμε ένα στοιχείο στο Zynq-7020 SoC FPGA που συνδυάζει το υλικό με το λογισμικό και επιτελεί τα πρώτα στάδια προεπεξεργασίας δεδομένων με εκτιμώμενη κατανάλωση ισχύος στο chip κάτω από 2 Watt.

Συμπεράσματα και Μελλοντικές Προεκτάσεις

Η παρούσα διπλωματική εργασία πραγματεύεται την ανάπτυξη ενός επιταχυντικού αλγορίθμου για την προεπεξεργασία δεδομένων ενός ανιχνευτή αστέρων, ο οποίος υλοποιείται σε μια συνδυαστική αρχιτεκτονική υλικού/λογισμικού σε μια πλατφόρμα SoC FPGA. Πιο συγκεκριμένα, η προεπεξεργασία χωρίζεται σε δύο βήματα λειτουργίας τα οποία είναι η ομαδοποίηση εικονοστοιχείων και η συσταδοποίηση αντίστοιχα. Και οι δύο λειτουργίες σχεδιάστηκαν, υλοποιήθηκαν και δοκιμάστηκαν μεμονωμένα χρησιμοποιώντας πραγματικές εικόνες. Επιπροσθέτως, παρουσιάζεται μια λεπτομερής επισκόπηση των βελτιστοποιήσεων που πραγματοποιήθηκαν στον αλγόριθμο συσταδοποίησης αξιολογώντας τις επιτευχθείσες συχνότητες λειτουργίας. Επιπλέον, οι προτεινόμενες αρχιτεκτονικές αναπτύχθηκαν και σε επίπεδο λογισμικού ώστε να εκτελεστούν στον ενσωμεταμένο ARM επεξεργαστή του Zynq SoC FPGA. Ο συνδυαστικός σχεδιασμός Υλικού/Λογισμικού υπερτερεί πλήρως έναντι της εκτέλεσης στο επεξεργαστή όσον αφορά την απόδοση καθώς κατάφερε να μειώσει το συνολικό χρόνο εκτέλεσης κατά περίπου 60 φορές, ενώ εκτιμάται οτι η επιτάχυνση αυτή μπορεί να αυξηθεί έως και τις 108 εάν το σύστημα πιάσει το μέγιστο της εκτιμώμενης συχνότητας λειτουργίας. Όσον αφορά την κατανάλωση ενέργειας, η συνολική ισχύς εκτιμάται περίπου στα 1.8 W, πράγμα που σημαίνει πως η προτεινόμενη αρχιτεκτονική είναι κατάλληλη για ένα ευρύ φάσμα οικονομικά αποδοτικών διαστημικών εφαρμογών. Τέλος, το προτεινόμενο ενσωματωμένο σύστημα είναι σχετικά μη απαιτητικό όσον αφορά τη χρήση των πόρων του FPGA, αφήνοντας περιθώριο για πρόσθετες υλοποιήσεις υλικού που αφορούν την προγραμματιστική ροή ενός συστήματος ανιχνευτή αστέρων.

Η μελλοντική μας εργασία θα επικεντρωθεί στη βελτίωση της απόδοσης του συστήματος, συμπεριλαμβανομένης της περαιτέρω εξέτασης των παραμέτρων που περιορίζουν τη συχνότητα λειτουργίας, ώστε να επιτευχθεί η μέγιστη εκτιμώμενη συχνότητα και να αυξηθεί περαιτέρω εάν είναι δυνατόν. Επιπλέον, θα εκτελέσουμε περαιτέρω πειράματα ώστε να επεκτείνουμε την αξιολόγηση και την επαλήθευση των τεχνικών μας με τη χρήση πραγματικών δεδομένων προσαρμοσμένων στις απαιτήσεις της Infinite Orbits. Τέλος, ο κύριος στόχος μας είναι να συνδυάσουμε το προτεινόμενο σύστημα με τις υπόλοιπες προγραμματιστικές διαδικασίες ώστε να ολοκληρωθεί το σύστημα του ανιχνευτή αστέρων και να συμπεριληφθεί σε μια μελλοντική διαστημική αποστολή καθοδηγούμενη από την Infinite Orbits.

Chapter 1

Introduction

This chapter's goal is to describe the motivation behind this work and introduce readers to some of the problems appearing in space industry. In addition, it contains the main scope of the thesis, as well as a brief outline of it.

1.1 Motivation

Nowadays, the technological evolution has brought significant achievements in almost every scientific field. More specifically, when it comes to space industry, the progress that has been made has had great influence on the development of human society. However, other than opening up new horizons and opportunities, some major challenges have arisen.

This current thesis was conducted in cooperation with Infinite Orbits in the context of the research activity of the Microprocessors and Digital Systems Lab regarding the applications of COTS FPGAs in space. Infinite Orbits is a company which operates in space industry and provides innovative in-orbit services. Among the cases that Infinite Orbits, as well as other companies are facing appears to be life extension. Satellite's lifespan ranges between 12 and 18 years, but the prevailing design life remains at 15 years. As per NSR's recent study, commercial GEO communications satellites in-orbit today have an average age of 8.9 years[15]. The idea of in-space service prolonging the life of a satellite in orbit is yet to be accomplished. Instead, of replacing the entire satellite, we could get another one to rendezvous and dock onto the outdated system providing a refueling source. Life extension is reportedly a very challenging case of rendezvous and docking due to the difficulties associated with the non-cooperative nature of the target satellite, the risks involved and the characteristics of the orbit, and thus extreme accuracy and safety is required. The main idea proposed by Infinite Orbits to resolve these challenges, is developing an embedded satellite navigation system which uses neural network-based models for image processing aiming to achieve real-time performance using a custom FPGA accelerator.

There are specific requirements that system in orbit needs to meet and are summarised as follows. First of all, the communication channel of the satellite is characterised by high latency and unreliability. Thus, the software is expected to be smart enough and capable of high levels of autonomy. Another challenge is that satellites involve excessive costs and therefore mistakes must be avoided. The system needs to work constantly or at least be able to fail safely. Because of the mission's nature and lack of validation environment on ground, it is really exacting to achieve robustness. Finally, fast performance is needed so that the system is responsive to deal with unwanted situations and avoid accidents. Some of the most common instruments to measure satellite's orientation fast and precisely are star trackers.

Determining the position of a satellite plays a prominent role among the majority of space missions. This type of information is vital for space navigation, firing thrusters which control altitude and pointing antennas to required positions. There are various types of sensors used for attitude determination, some of the most common are horizon sensors (or conical Earth scanners), sun sensors, star sensors and magnetometers[16]. Each type can be used individually or in combination with the others and in the aggregate they constitute part of an integrated attitude determination and control system (ADCS).

Among the above instruments mentioned, star trackers are capable of providing high accuracy for attitude determination. In many cases where there are demanding scientific objectives, an attitude control system of high accuracy is required. It is therefore understood why star trackers are of high importance. Additionally, there are missions where a certain level of agility is required for specialized tasks. For example, satellites should be agile and maneuverable as it considerably increases the return of earth and science mission data making them more efficient.

1.2 Thesis Scope

In this thesis, we focus on the first stages of a star detection algorithm which in particular are associated with the preprocessing of the image captured by the sensor of the system. More specifically, we propose an efficient algorithm which applies in a star tracking system that detects clusters of stars, which are to be used during the next stages for centroid extraction. Furthermore, we aim to implement a HW/SW co-design which runs at a SoC FPGA accelerated system in order to achieve fast performance.

We suppose that we have a satellite in orbit that is being provided with frames that were captured by the system's sensor. Each frame is a two dimensional image that contains only a portion of the night sky. Star trackers, need to evaluate this data and determine the attitude recognising patterns of constellations and mapping them to known star positions in a catalogue. The challenge here is that large images need to be processed and the computation should be fast and accurate as long as star tracker's update rate depends on that. Thus, we can understand that fast execution while processing each frame is critical to achieve real-time performance.

Several commercial star tracker solutions exist but are not easily accessible. They are of high cost and it is often a preferable option to develop a custom one according to the mission's requirements. The block diagram of a typical star tracker is shown below in Figure 1.1.

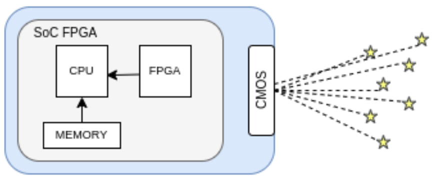


Figure 1.1: Block diagram of a typical Star Tracker.

Generally, star trackers play an important role to the entire satellite's Attitude Determination and Control System (ADCS) and they consist of a CMOS Image Sensor that is exposed to light and digital hardware that processes the data. Usually, a System-on-Chip (SoC) device which integrates a processor, a Field Programmable Gate Array (FPGA) including interconnections, memory and some peripherals all in one chip, is responsible for the processing of the image data. When considering CPU & FPGA co-processing, the most prominent devices are the COTS SoC FPGAs, which integrate both types of processors in a single chip [17]. However, space community employs both space-grade [18][19][20][21] and Commercial Off-The-Shelf (COTS) [18][22][23] FPGAs. In general, FPGAs are not only utilised to accelerate complex DSP functions of high computational intense, but they can be used as well as add-on processors in co-processing architectures [24][25][26] where they perform operations such as I/O handling, data transcoding and data compression.

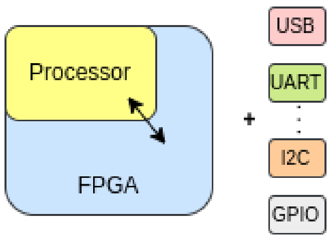


Figure 1.2: System-on-Chip Layout.

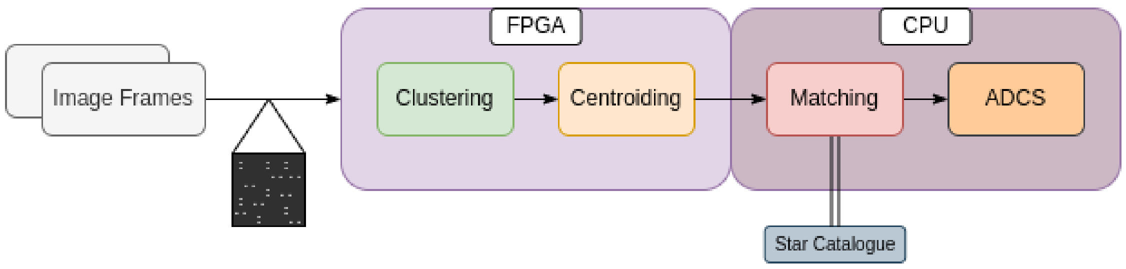


Figure 1.3: Process Flow of a Star Tracker.

The sequential process flow of a Star Tracker is shown in figure 1.3. Exposure time is the first stage of the process and is the time needed for the sensor to produce image data of the night sky. Afterwards, data are processed on the FPGA platform so as to extract centroids which consequently are forwarded to the processor. The final stage of the process is matching. During that stage, a pattern recognition algorithm is executed in order to identify the extracted centroids by mapping them to a known star catalogue. The main scope of this thesis is to develop an efficient algorithm that performs the preprocessing tasks and is presented in Chapter 3. That is a non trivial problem due to the limitations of the FPGA platform while certain requirements of the star tracker need to be met such as robustness, low cost, precision and low power consumption.

1.3 Project Objectives

This thesis was a collaboration of Microlab-NTUA and Infinite Orbits and due to their uttermost contribution it was completed successfully. In addition to their excessive support, Infinite Orbits provided a custom camera lens which is to be used for future work and extension of this thesis. More specifically, the lens is a backside illuminated (BSI) scientific CMOS image sensor, with 4MP resolution (2048×2048) and 12 bits per pixel whose size is 6.5 µm. For the development and evaluation of the system a COTS SoC FPGA was utilised and in particular, Zedboard 7020 chip that hosts Xilinx ZYNQ SoC with ARM processor. Thus, the project objective is to optimise an algorithm used for star trackers and can be applied to an entire ADCS, on Zynq SoC for that specific image sensor which is provided by Infinite Orbits. However, as it will be discussed in the following chapters, the proposed system has been designed to be configurable in order to adjust to different conditions according to the mission's requirements and therefore it supports many different camera types. The specifications of this project are the following:

- Field of View (FOV): $15.5^\circ \times 15.5^\circ$
- Image Resolution: 4 MP
- Pixel Depth: 12 bits
- Real-time performance of 1-2 frames per second
- Accelerated clustering algorithm
- Low Power Consumption

1.4 Thesis Outline

The remaining of the thesis is structured as follows.

Chapter 2 presents the full background, which is helpful to completely understand the concept of our ideas and implementations in the following chapters. It involves some general information about star trackers, their characteristics and what is the main purpose they are used for. In addition, we will review some of the related work that has been published or is currently available for space missions and finally, we will provide an overview of SoC FPGAs, as well as a general outlook of the interconnection protocols that are used.

Chapter 3 focuses on the ideas that our algorithm is based on. We analyze quite extensively the proposed architectures presenting them both at high and low level. The first subsection of this chapter emphasizes on the averaging binning, while the second one describes the main algorithm for cluster extraction. Finally, we review the optimization steps we made aiming to achieve high performance.

Chapter 4 presents the experimental evaluation of the proposed system, along with a results section which basically compares the hardware to the software outcomes. It also refers to the power consumption and resource utilisation of our integrated system.

Chapter 5 is the final chapter, summarizing the conclusions drawn from the total contributions of our work.

Chapter 2

Background

In this section we are going to present background material for the explanation of the methods and ideas that have been used in the main part of the thesis. Our main focus is on the background of star trackers regarding some of their characteristics and algorithms used for their development.

2.1 Star Trackers

A star tracker is an optical device that measures the positions of stars. As long as this happens to a high degree of accuracy, a star tracker on a satellite or spacecraft may be used to determine its orientation with respect to the stars. In order to do this, the star tracker needs to obtain an image of the stars and measure their apparent position using pattern recognition algorithms so as to identify them compared to their known absolute position on a star map.

Generally, star sensing and tracking devices can be divided into three major classes which are star scanners, gimbaled star trackers and fixed head star trackers. Star scanners use the spacecraft's orientation to provide the searching and sensing function while gimbaled star trackers search out and acquire stars using mechanical action. Finally, the last class of star trackers has electronic searching and tracking capabilities over a limited field of view[27].

Attitude determination based on charge couple devices (CCD) area array image sensors was pioneered in the early 1970s. There are the so-called "First-Generation CCD Star Trackers" and consist of a CCD sensor, associated optics and dedicated electronics[28]. Within the last decade, a new generation of star trackers has been developed which involves units identified as "Second-Generation Star Trackers". The main difference from the prior generation is that pattern recognition algorithms are performed autonomously so as to identify constellations using internal catalogues. In addition, they are more advanced as expected and they are capable of high performance in adverse conditions.

2.1.1 Star Tracker Characteristics

Commercial star trackers vary in terms of performance according to their characteristics. Depending on the mission's requirements, there are several parameters that can be adjusted in order meet these exact objectives.

Accuracy

As we have pointed earlier, accuracy correlates with the star tracking algorithms. The more demanding the application is, the more increasing the need is to develop algorithms of high accuracy, and so the performance becomes computationally intensive. Liebe in his article describes extensively how performance accuracy is measured[29]. Actually, there are two types of accuracy metrics when we refer to star trackers, pointing accuracy and rolling accuracy. Accuracy measurement is essentially the angle error curve of pointing and rolling axis with respect to the boresight[1]. Usually, they are also referred as cross boresight accuracy (pitch and yaw) and about boresight accuracy (roll) correspondingly and are quoted in 3σ values. Cross boresight errors are significantly less than about boresight errors and modern star trackers are capable of accuracies that range from 0.1 to 20 arcseconds (cross boresight)[30]. In order to achieve such low levels of accuracies, sub-pixel accuracy is critical while executing centroid detection.

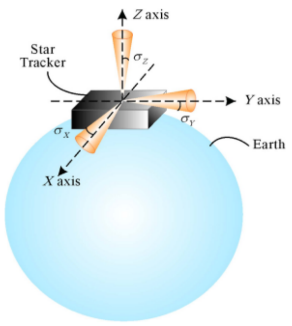


Figure 2.1: Pointing and rolling accuracy schematic of the star tracker [1].

Field of View

The size of the Field of View is probably the most important parameter. A narrow FOV implies high precision and is a requirement for astronomical star trackers. Low cost general purpose star trackers offer wide FOV whose size ranges from a few degrees to over 40° diagonally and are usually utilized as a part of the ADCS in micro satellites. High precision star trackers are capable of accuracies of 1 arcsecond. On the contrary, the typical precision of general purpose star trackers is about 15 to 20 arcseconds. However, decreasing the FOV has some disadvantages as well. First of all, it is more difficult to determine the initial attitude. When the FOV contains a small number of stars, the algorithm tend to reject the frame. Moreover, a much bigger star catalogue is required, which is significantly undesirable since it increases the amount of memory needed. Furthermore, initial attitude acquisition becomes a more complicated task and also the complexity of pattern recognition algorithms increases rapidly with the number of stars [2].

Update Rate

The update rate basically depends on the exposure time and the processing time of the image. There is a strong correlation between exposure time and the optical and hardware design. The longer the exposure time, the more photons are captured by the image sensor and this results to a better signal-to-noise ratio. Choices regarding the sensitivity of the sensor, the aperture of the lens, the sky coverage and some other factors affect the exposure time. Generally, it is required to have a specific number of stars on average in the FOV. Thus, to compensate the decreased FOV, we usually increase aperture of the lens so as the fainter stars to be captured. The aperture primarily determines image sensor's sensitivity. Increasing the exposure time is another technique that is used in order to increase system's sensitivity. However, this method has disadvantages as it leads to lower update rates and smeary images that affect the performance of the ADCS. Therefore, exposure time and accuracy are trade-offs for a stable star tracker. Lastly, second generation star trackers need to process large amounts of data which is a significant limiting factor to update rate. In this thesis we focus on the acceleration of a clustering algorithm used in a star tracker sub-system in order to improve the processing speed.

Physical Characteristics

Physical dimensions of a star tracker are important factors which are to be taken into consideration when discussing about commercial star trackers. The mass of a star tracker usually varies from 1 to more than 20 kg. At this point, it should be mentioned that aperture of the lens is proportional to the system's mass and size. High-performance star trackers are huge and are designed for more scientific space missions, hence the assumption that they are more accurate. However, the increasing market for microsatellites tends to push the interest towards smaller designs.

Other Characteristics

Other than the aforementioned, there are a few other characteristics that describe star trackers such as power consumption, sky coverage, SNR, star catalogue size and average number of stars tracked per frame. These all directly affect the performance of the system in a certain way. However, as we discussed earlier, there are no optimal choices as there are some trade-offs and so preference is given upon the parameters that meet the design's requirements.

2.1.2 Sub-pixel Accuracy

Generally, among the most limiting factors for the resolution of optical instruments, is the size of the sensor's pixel. In digital image processing, a sub-pixel resolution method is utilized to enhance the resolution of images and therefore to improve the algorithm's performance. A widely known technique to increase the accuracy, is the hyperacuity technique, also known as sub-pixel accuracy.

In a focused image, the star appears as a point source, so all the luminous power gathers in a single pixel. However, images might often be slightly defocused, even if they're taken by an expensive image sensor of high accuracy. In these cases, a star occupies several pixels in the image plane. This makes it easier to model the centre of the star mathematically.

It may seem intuitively inconsistent with the sampling theorem to increase the accuracy beyond one pixel, but it is the a priori knowledge of the pointspread function that is utilized in combination with the actual measure. The performance of the hyperacuity technique is defined by the algorithm used to determine the center of the star [2]. The achieved sub-pixel accuracy depends on the S/N ratio. Unlike the fancy mathematical models, the hyperacuity technique normally uses empirical calibrations which appear to be really consistent.

There is an enormous dynamic range of illumination between the brightest and the dimmest star. The resolution is limited when digitalising the CCD signal received from the sensor (typically 8 to 16 bits are used). This leads to a contrasting situation where the brightest stars will overflow and the dimmest stars will be hardly perceived due to noise. These different situations are shown in Figure 2.2

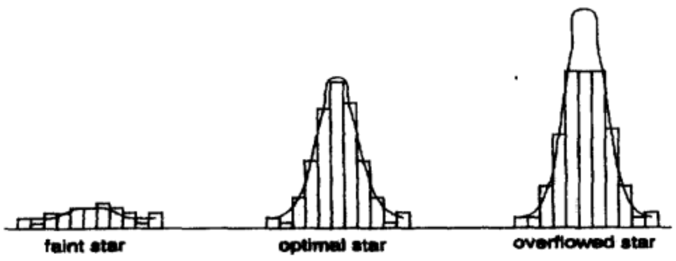


Figure 2.2: Different Stars Utilising Hyperacuity technique [2].

2.1.3 Theoretical Background

At this point it is important to introduce a theoretical background regarding the information that is obtained by the image. The image sensor captures only a small portion of the night sky that contains just a fraction of the stars which often appear dimmer or distorted. Stars within the FOV are projected onto an image plane, where their magnitudes are converted to a corresponding electron count. The magnitudes are expressed as two-dimensional Gaussian functions in order to account for the (u,v) image directions. The standard deviation of each Gaussian is determined by the sensor and optics of the system and is defined in units of pixel width.

The FOV is described here by only one angle, which is defined by the angle from the center of the lens to the outer edge of it. Camera's frame is defined with the z-axis pointing out the boresight, y-axis out of the top of the camera and x-axis completes the right-hand frame. Once the frame and the FOV are defined, we should determine camera's attutude, or orientation with respect to another reference frame. The most common one that is used is the J2000 Earth centered inertial (ECI) frame [31]. Hence, the entire catalogue in matching process is searched to find all stars within the camera's FOV by determining the angular distance between z-axis and each star's ECI unit vector.

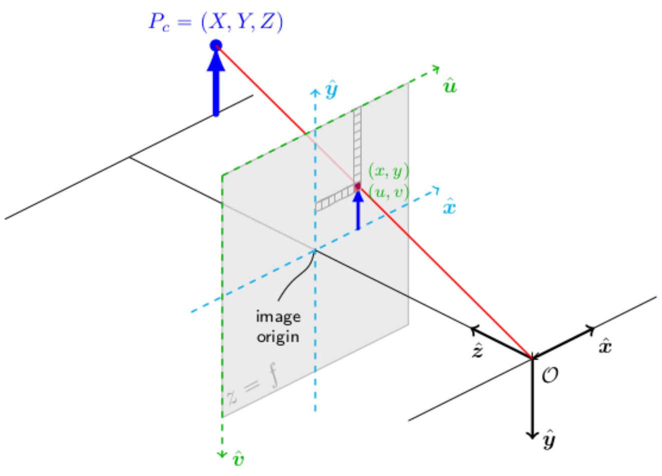


Figure 2.3: Image sensor's perspective Projection model [3].

We should mention that each star on the is projected onto a two-dimensional image frame and so a determination of the three-dimensional points that represent the unit vector is needed. To do so, we should take into consideration some camera parameters: the intrinsic parameters regarding the properties of the sensor; and the extrinsic parameters, which are related to the location and attitude of the camera with respect to the inertial frame. These parameters form the camera matrix. The image plane is digitized and forms the (u,v) pixel array, so the characteristics of the optical system define how the plane's coordinates transform to the (u,v) pixel coordinates.

So far, we made a short report of how the pixel location for a star in the FOV is determined. When it comes to the image processing, we need to perform certain calculations regarding the corresponding brightness of the image. In astronomy, the brightness, or magnitude, of a celestial body is defined in a logarithmic scale and is often measured in a specific wavelength, spectrum or photometric band. Typically, magnitudes are measured in the visual or near-infrared spectrum, wavelengths from hundreds of nanometers up to micrometers. Since we have a logarithmic scale, the closer to zero in absolute scale a star is, the brighter it is.

The magnitudes of stars in the FOV corresponds to a number of electrons. The magnitude is related to solar irradiance, which is defined as the power per unit area emitted by a star. Once the electron count is calculated through a series of steps, which are considered superfluous for the needs of this thesis, we can determine the area of pixels taken up by a star in the FOV. The electron count can be described by a two-dimensional Gaussian function, which provides a more intuitive way in order to determine the spread of a star. The standard deviation of the Gaussian is determined by sensor and optic parameters and is kept constant for each star regardless of brightness.

2.1.4 Algorithms

Although star trackers might differ in quite a few parameters, they mostly follow the same process flow. The entire process can be divided into four primary sub-processes that cooperate to provide the system with the information needed for attitude determination.

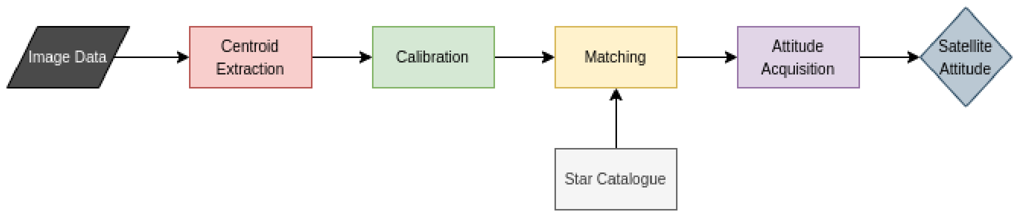


Figure 2.4: High level process flow of an ADCS.

Centroid Extraction

The first step of the algorithm is to extract centroid positions of possible stars on the image plane. This actually happens in three steps, thresholding, clustering and finally centroiding. This thesis puts a strong focus on the first two steps proposing two optimised algorithms for hardware implementation that will be discussed in detail on Chapter 3. Some of the most common algorithms that are used for centroid extraction are the center of gravity and the fast gaussian fitting.

Distortion Correction

Distortion correction is the process where lens distortion correction is done in order to convert the image plane coordinates to the corresponding three-dimensional sensor body coordinates in Cartesian form. Unfortunately, there is no perfect lens, so it is inevitable to have some distortion to the images due to variations in image magnification. Although distortion can be irregular or follow many patters, the most commonly encountered distortions are radially symmetric and can be classified as either barrel distortions or pincushion distortions[32]. Figure 2.5 illustrates these two types of radial distortions. In barrel distortion, image magnification decreases with distance from the optical axis, whereas in pincushion distortion it increases. Radial distortion can be corrected using Brown's distortion model, also known as the Brown-Conrady model. This model utilizes a series of polynomial coefficients that are often provided by the lens manufacturer, but in most cases, it is necessary to calibrate the camera and the lens.

Matching

During this step, pattern recognition algorithms are used to map constellations with known star positions in a catalogue. The algorithms are of the lost-in-space type or the recursive type, wich runs off of some prior position knowledge. One of the most widely known star catalogue is the Hipparcos catalogue of nearby stars. The Hipparcos catalogue was obtained from the European Space Agency's Hipparcos astrometric mission that operated from November 1989 to March 1993 viewing the celestial sphere. The mission returned very high quality star astrometric and photometric data, specifically high precision data on 118,218 stars. It is important to mention that this process is of

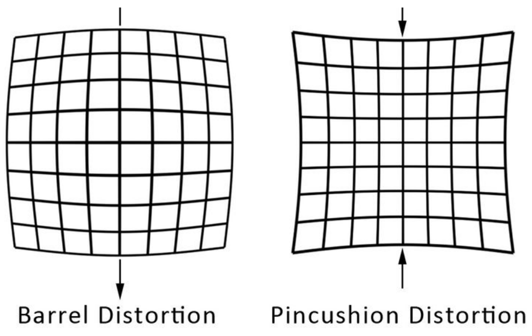


Figure 2.5: Types of radial distortion [4].

great complexity and software optimisations are required to improve its processing time. Lately, new techniques have been proposed to accelerate this process using neural network models.

Attitude Acquisition

The final step is the attitude acquisition. Once the centroids have been extracted an matched to known ECI coordinates, they can be used to determine the satellite's attitude. There are many different types of attitude determination algorithms for star trackers in use today. Some of them which are commonly used are the TRIAD algorithm as well as the Quaternion Estimator (QUEST) algorithm.

2.1.5 Commercially Available Star Trackers

Indicatively, a list of commercially available star trackers with high performance standards is presented as follows in table 2.1.

Table 2.1: Table of commercially available star trackers

Manufacturer	Redwire	Vectronic Aerospace GmbH	Terma	Ball
Model	SpectraTRAC	VST-68M	HE-5AS	HAST
FOV	N/A	$14° \times 14°$	N/A	$8° \times 8°$
Update Rate	4 Hz	5 Hz	4 Hz	configurable
Accuracy (x, y axis)	10 arcsec	5 arcsec	<1 arcsec	<0.5 arcsec
Accuracy (z axis)	27 arcsec	30 arcsec	<5 arcsec	N/A
Mass	0.475 kg	0.470 kg	1 kg	N/A
Volume	$120 \times 61 \times 61$ mm	$60 \times 60 \times 138$ mm	$120 \times 120 \times 33$ mm	N/A
Power	2.5 W	3 W	1.5 W (camera) 5.5 W (processor)	N/A
Temperature	-30°C - 55°C	-20°C - 65°C	-40°C - 70°C	N/A

2.2 Related Work

A broad interest in space exploration has led to a proliferation of algorithms for star detection that can be used for spacecraft's attitude determination. Carl Liebe gives an extensive overview in his articles about the operation and performance of autonomous star trackers. He presents in detail the new class of second-generation star trackers. These designs, are fully autonomous, have higher accuracy, smoother and more robust operation and are capable of solving the lost-in-space problem. Star trackers are able to operate in two basic modes: lost-in-space mode and tracking mode. In tracking mode, the star tracker is provided with initial attitude information as a way of increasing the update rate. In tracking mode the image is processed in portions during centroiding. Lost-in-space attitude acquisition is a more complex task resulting to higher processing times, so the advertised update rates of commercial star trackers often correspond to tracking mode operation [30][2].

R. van Bezzooijen from the Lockheed Palo Alto Research Laboratory presents a detailed development of an autonomous star tracker prototype including an in-depth evaluation after testing with real data. The AST is designed to perform in lost-in-space mode, to update its attitude autonomously and provide attitude information continuously as well. The paper also describes a number of functions that are enabled or enhanced by the AST including attitude safing, fast fault recovery, autonomous optical navigation and uncalibrated attitude acquisition. He concludes that after performing realistic simulations, the results showed that the AST with an 11.3° FOV, spatial accuracy of 10 arcseconds, brightness accuracy of 0.3 magnitude, a database of 4148 guide stars, a highly efficient non-iterative star pattern recognition algorithm, and an MC68030 class microprocessor running at 25 MHz is capable of determining its attitude in approximately 0.6 s having no initial attitude information. It is noteworthy that it can do so with a demonstrated success probability of 99.25%, while the probability of false identification is less than 0.1% [33].

Rufino et al. published a paper that demonstrates the enhancement of the centroiding algorithm for a star tracker. The hyperacuity technique to image processing is analyzed and an error-budget analysis is performed to find out type and magnitude of the effects on centroiding. The centroiding error results to be a function of two parameters: a systematic term and a random one, so numerical models of defocused point spread functions were introduced to evaluate their contribution. Since the first error contribution is systematic, a backpropagation neural network is adopted to improve the performance. He ends up that the application of the correction to the PSF numerical models introduces an improvement of the overall centroiding accuracy to 0.005 pixels [34].

Wei et al. presented a real-time star identification using synthetic radial pattern and implemented a hardware design. Their paper proposes a novel lost-in-space algorithm for star determination based on synthetic radial pattern, which is dedicated to the pipelined paraller architecture of FPGAs. The synthetic radial pattern consists of two single radial patterns connected by their two respective polestars. In the algorithm, the polestar-pair is firstly matched and then radial pattern filtering is performed so as optimum identification results can be obtained. This results to a significant reduced number of spurious matches. They also developed a mathematical model to demonstrate the efficiency compared with conventional radial algorithms. The algorithm was then described both in software and FPGA implementation details. Software simulations have indicated that the proposed algorithm was highly efficient, robust and less sensitive to false stars. In addition, the time cost of less than 1 ms, demonstrates that the implementation can meet the real-time requirements of high-dynamic star sensors [35].

2.3 Space Navigation

In recent years, there is a growing interest in deep space exploration. The current approach for planetary navigation is based on ground-based radiometric tracking. This technique is a bit of disadvantageous as the radio signals that are transmitted from a spacecraft are very weak and have to be picked from background noise and additionally it might take hours to reach the Earth. As there are several constraints for equipment on board spacecraft, most of the complex communications technologies are incorporated on ground. A new era of low-cost small satellites for space exploration will decrease the reliance on ground-based tracking and provide a substantial reduction in operational costs due to crowded communication networks. Autonomous navigation methods are widely investigated and one of the most promising methods, is called crosslink radiometric navigation which uses inter-satellite communication link. Fully autonomous navigation is only possible if it is performed on-board without any intervention from ground. In general, autonomy can provide minimal-cost if ground operations or hardware are reduced and lead to increased performance [36].

2.3.1 Celestial navigation for satellites

Celestial navigation method requires knowledge of celestial bodies' position (Sun, Moon, Earth and stars) in inertial frame at certain time and correlation between their observed position in the satellite body frame and the satellite's position. Satellite's position and velocity can be defined by the error between the measured and estimated celestial body data. For attitude determination, the measurements of stars are sufficient. For Low Earth Orbit (LEO) satellites, the horizon sensing accuracy is the most determining factor which affects celestial navigation accuracy. Regarding horizon sensing, there are two major approaches: directly sensing horizon method and indirectly horizon sensing method. In the first method, an horizon sensor is used to provide directly Earth-relative information and as well as other celestial bodies' direction. These measurements are often combined with a Kalman filter to determine the orientation and attitude of a spacecraft. In the second method, high accuracy star sensor and a mathematical model of starlight refraction in the atmosphere are used to in order to provide information about horizon's direction. The method of directly horizon sensing is robust, prompt and easy to use due to simplicity. However, the achieved navigation accuracy is insufficient, so the indirect method is highly preferable. An alternative technique utilizes a combination of these two complementary approaches providing a reliable celestial navigation system with a significant improvement in terms of performance. Furthermore, the accuracy can be enhanced with a supplementary measurement technique such as the Doppler velocity related to on-ground operations measured by a standard component. Navigation accuracy is affected by two major factors which are the state model's accuracy and the filtering method. Among the most common filtering methods that are widely used, are the Extended Kalman Filter, the Unscented Kalman Filter and the Particle Filter [36][37].

2.3.2 Celestial navigation for deep space probes

Navigation of satellites refers to the mission orbit's knowledge with respect to the central body (absolute) or with respect to another object (relative). It is also related to the knowledge of where the object resides in the past, present and future[36]. Every reported navigation method applicable for deep space is sorted to on-board and off-board navigation method as of where navigation knowledge is obtained. Some of the most common autonomous deep space navigation methods are presented as follows.

Optical navigation

Optical navigation refers to a variety of methods of determining the spacecraft states, relative position and velocity between a spacecraft and a target body with on-board optical sensors. Basically, optical sensors, whose characteristics determine resolutions, sensitivities, and uncertainties, estimate Line-of-Sight (LOS) to beacons or to known locations at the surface. In principle, optical navigation methods compute a body position in the camera reference frame and derive target location in space from the corresponding location in the image frame [38]. The main advantage of these methods is that can be used in various mission phases such as cruise, flyby, rendezvous, orbiting and landing. A few deep space small satellite missions have planned to use one of the proposed optical methods. In brief, optical navigation provides major advantages over other architectures, such as moderate to high accuracy navigation solutions, while being compatible with all mission phases [39].

Pulsar navigation

Pulsar navigation uses the periodic X-ray signals emitted from pulsars to determine spacecraft's states by estimating timing and direction of the pulses arriving. Stable neutron stars spinning nearly 1000 times a second, for example, can provide a solution for autonomous navigation: the arrival time of each pulsar updates a temporal database and subsequently the extracted data is used to determine or update attitude, position or velocity of the spacecraft [40]. Probably, the most important advantage of this method is that is capable of stabilizing on-board clocks via the periodic pulse signals. Furthermore, it is applicable to missions not only in close proximity to Earth but in deep space as well. Pulsars emit radio frequencies that range between 100 MHz and a few GHz. In order to detect those signals, radio frequency systems require huge antennas with diameter greater than 25 meters [41]. Overall, pulsar navigation offers a better and more accurate solution over optical navigation. However, a major disadvantage of the first one, is the sensor size and the required integration time which makes it incapable of operating in some mission phases, such as close-proximity[39][42][43].

2.4 Rotational Kinematics

Kinematics is the study of motion of a system of bodies without directly considering the forces or potential fields affecting the motion. Navigation requires a fixed reference coordinate system so as to relate the corresponding moving frame of the spacecraft. This fixed reference frame which in used for space navigation is known as Earth Centred Inertial.

2.4.1 The Earth Centred Inertial (ECI)

The Earth Centred Inertial frame, also known as Geocentric Equatorial Coordinate System, is fixed in inertial space and positioned right at the centre of the Earth, but it does not rotate following Earth's motion. The fundamental plane contains the equator and the positive X-axis points in the Vernal equinox direction. The Z-axis points in the direction of the geographical North Pole and the Y axis consequently completes the right hand set of coordinate axes [5]. It is assumed that celestial objects are inconsistent units in great distances, so any position on the sphere is defined by the following two coordinates: right ascension and declination. Right ascension α is the angle measured from the vernal equinox considering that counterclockwise rotation has a positive angle measure. Declination δ is the angle measured from the celestial equinox respectively. An illustration of the ECI coordinate system is shown in Figure 2.6.

A frame associated with the spacecraft's body is needed as well. This body frame is fixed with respect to its vessel and its position is at the centre of its mass. Both ECI and body frames are fundamentally important for attitude acquisition. The star tracker provides the means find the correlation between the two coordinate systems and extract information about the spacecraft's attitude.

2.4.2 Euler Angles

The orientation of a rigid body with respect to an inertial reference frame N can also be described by a sequence of three rotations, each about a single axis in the body frame B. These three rotations are the most frequently used method for representing an object's attitude and are known as the Euler angles.

Coordinate Rotations

The special orthogonal group of all 3×3 rotation matrices is denoted by SO(3). A coordinate rotation is a rotation about a single coordinate axis. Enumerating the x, y, z-axes with 1, 2 and 3, the coordinate rotations, $R_i : \Re \rightarrow SO(3)$, for $i \in$1,2,3, are

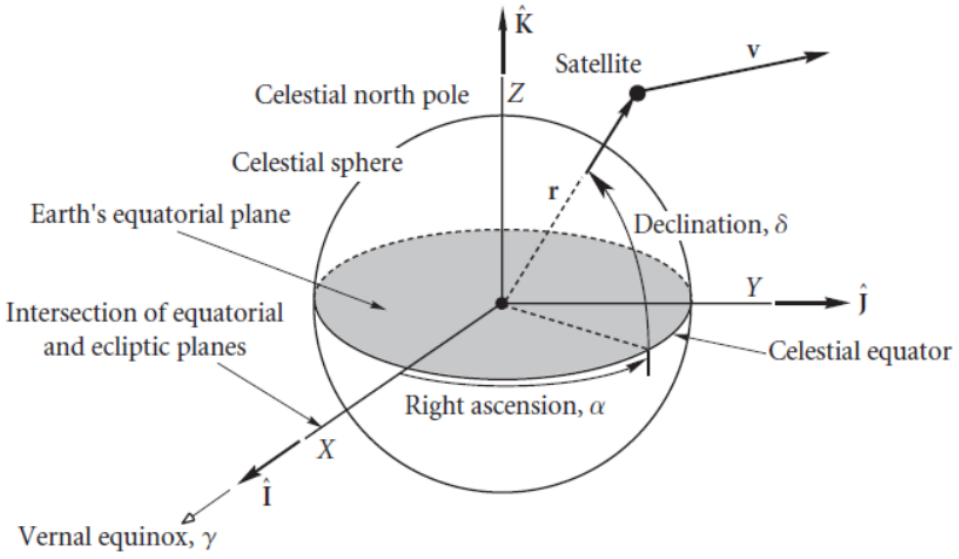


Figure 2.6: The Earth-Centred Inertial (ECI) frame [5].

$$R_1(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{2.1}$$

$$R_2(\alpha) = \begin{bmatrix} cos(\alpha) & 0 & -sin(\alpha) \\ 0 & 1 & 0 \\ sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{2.2}$$

$$R_3(\alpha) = \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Considering three coordinate rotations in sequence, in which the first one is an angle ψ about x-axis, the second is an angle θ about y-axis, and the third is an angle ϕ about z-axis. For the sake of brevity, we arrange these angles in a three-dimensional vector called the *Euler angle vector*, defined by

$$u := \begin{bmatrix} \phi, \theta, \psi \end{bmatrix}^T \tag{2.4}$$

The function that maps an Euler angle vector to its corresponding rotation matrix, $R_{xyz} : \Re^3 \rightarrow SO(3)$, for $i \in$1,2,3, is

$$R_{xyz}(\phi, \theta, \psi) := R_x(\phi) R_y(\theta) R_z(\psi) \tag{2.5}$$

As in the general case, if $z \in \Re^3$ is a vector in the world coordinates and $z' \in \Re^3$ is the same vector expressed in the body-fixed coordinates, then the following relations hold

[44]:

$$z' = R_{xyz}(u)z \tag{2.6}$$

$$z = R_{xyz}(u)^T z' \tag{2.7}$$

Euler Angle Sequence (3,1,3)

The most common sequence associated with the name Euler angles is (3, 1, 3), named for Leonhard Euler, an 18th century Swiss mathematician and physicist. For aircraft motion, we usually refer the motion to a horizontal rather than to a vertical axis. In a description of aircraft motion ψ would be the "roll" angle; ϕ the "yaw" angle; and θ the "pitch" angle [45].

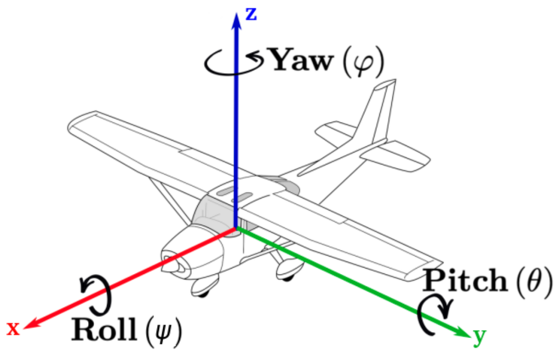


Figure 2.7: Roll-Pitch-Yaw notation for Euler angles [6].

In the (3,1,3) rotation sequence the first rotation is an angle of ψ about z-axis (yaw), the second rotation is an angle of θ about x-axis (roll) and the third rotation is an angle of ϕ again about z-axis. The rotation sequence is depicted in Figure 2.8.

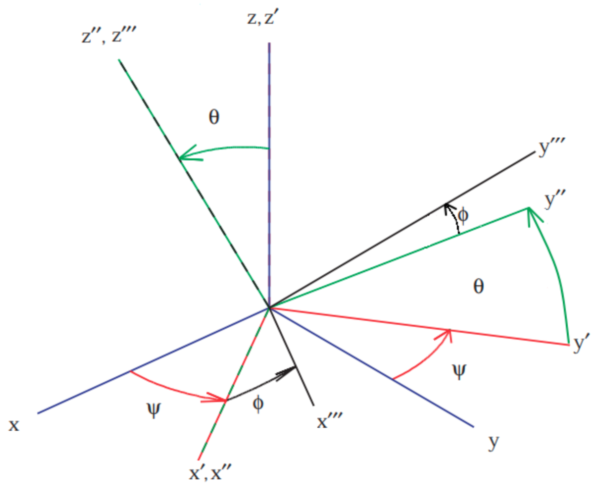


Figure 2.8: Euler Angle Sequence (3,1,3).

For compact notation, we write $c_\theta := cos(\theta), s_\phi := sin(\phi)$, etc. The function that maps a vector of Euler angles to its rotation matrix is:

$$R_{313}(\psi,\theta,\phi) = R_3(\psi)R_1(\theta)R_3(\phi) =$$

$$= \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\theta & s_\theta \\ 0 & -s_\theta & c_\theta \end{bmatrix} \cdot \begin{bmatrix} c_\phi & s_\phi & 0 \\ -s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

2.4.3 Coordinate Transformation

The post-centroiding process as shown in Figure 2.4, is matching. In order to proceed to this operation, coordination transformation is needed so as to yield centroid's exact position in the sky. This coordinate transformation technique is adapted to Wielligh's master thesis [46] and a paper by Qian et al. [7]. The first step required, is a rotation from the ECI coordinates to the star tracker sensor's body coordinate system. Consequently, we need to project the image sensor's coordinates onto the two-dimensional image plane. Figure 2.9 illustrates exactly how the preceding coordinate systems mentioned are related.

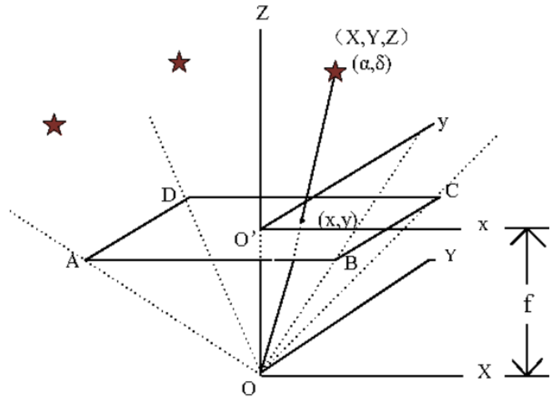


Figure 2.9: Coordinate model of a star tracker [7].

The ECI coordinates of a star are expressed as (α, δ). The star tracker sensor body frame is noted as (U, V, W) and the image plane coordinates are expressed as (x,y). Let us assume that the camera boresight is right at the Earth's centre. Any translation transformation between the two coordinate systems can be ignored regarding the long distance between the Earth and other stars considered fixed. A rotation-matrix, R is expressed as follows:

$$\left[X, Y, Z\right]^T = \mathbf{R} \cdot \left[U, V, W\right]^T \tag{2.9}$$

where $[U, V, W]^T$ is the ECI celestial coordinates in Cartesian notation. Supposing that we use the (3,1,3) Euler angles rotation sequence that was presented in the previous subsection. From the equation (2.8) we have that the rotational matrix $R = R_{313}(\psi, \theta, \psi)$, where ψ, θ, ϕ are the 3D rotation angles as shown in Figure 2.8 respectively. As we can

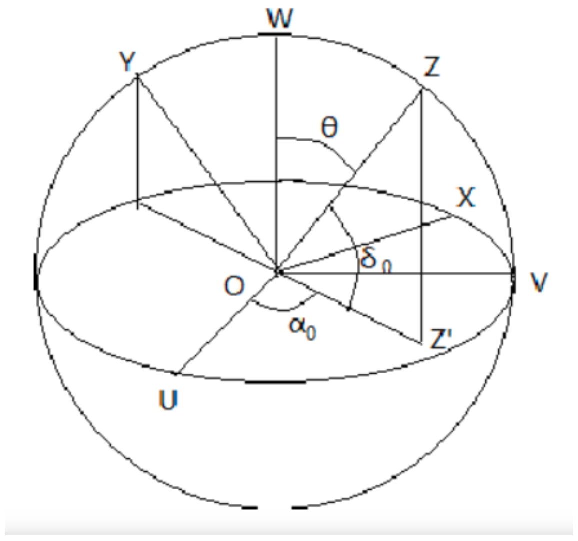


Figure 2.10: Celestial coordinate system to star tracker's coordinate system [7].

see in Figure 2.10, the rotation angles θ and ϕ can be expressed in relation to the boresight as:

$$\theta = 90^\circ - \delta_0 \tag{2.10}$$

$$\phi = 90^\circ + \alpha_0 \tag{2.11}$$

where (α_0, δ_0) refere to the boresight orientation. The ECI coordinates vector of a fixed star can be also expressed as:

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} c_\alpha c_\delta \\ s_\alpha c_\delta \\ s_\delta \end{bmatrix} \tag{2.12}$$

Combining the equations (2.8), (2.9) and (2.12) we come up with the following result for $[X, Y, Z]^T$:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & s_{\delta_0} & c_{\delta_0} \\ 0 & -c_{\delta_0} & s_{\delta_0} \end{bmatrix} \cdot \begin{bmatrix} -s_{\alpha_0} & c_{\alpha_0} & 0 \\ -c_{\alpha_0} & -s_{\alpha_0} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_\alpha c_\delta \\ s_\alpha c_\delta \\ s_\delta \end{bmatrix} \tag{2.13}$$

where (α, δ) is the coordinate of a star in the ECI frame, (α_0, δ_0) is the selected boresight orientation and ψ is the rotation angle about the boresight (z-axis of the sensor's body coordinate system).

The next step is to project that $[X, Y, Z]^T$ vector onto the image plane. Hence, we need to project the three-dimensional points representing the unit vector of each star onto a two-dimentional image plane. We use the pinhole camera model, which provides a simple technique to describe the relation between the 3D coordinates of a point and its projection an ideal pinhole camera. Through some similar triangles formed, the 2D image coordinates $[x, y]^T$ are expressed as:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{2.14}$$

where f is the focal length in millimetres.

Finally, to determine the pixel coordinates we need to scale the image plane coordinates, rotate the axes and shift them to the right corner as shown in Figure 2.11. The

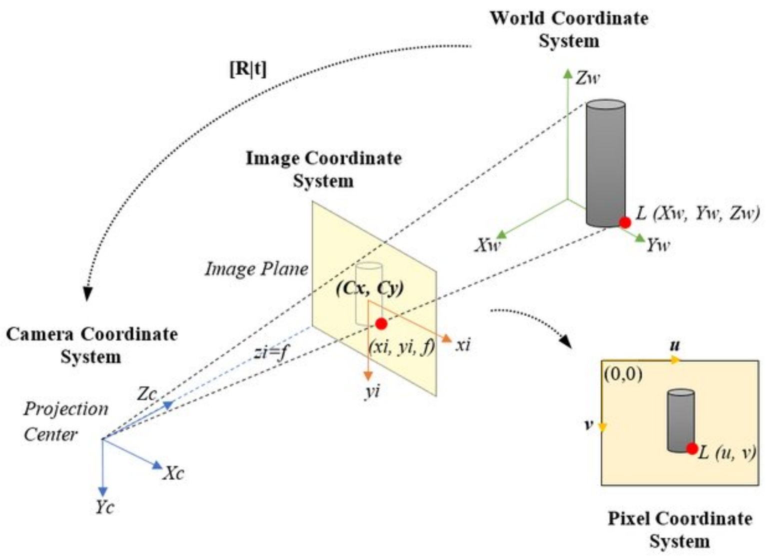


Figure 2.11: Pinhole Camera Model: ideal projection of a 3D object on a 2D image [8].

mathematical functions that describe this transformation are:

$$\begin{bmatrix} u \\ v \end{bmatrix} = -[\frac{1}{S_x} \frac{1}{S_y}] \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} o_x \\ o_y \end{bmatrix} \tag{2.15}$$

where S_x, S_y is the pixel resolution divided by the size of the sensor and o_x, o_y represent the coordinates of the sensor's centre in pixel units.

2.5 SoC FPGA Overview

This current thesis, puts a strong focus on the implementation of an algorithm for star trackers exploiting the FPGA features due to its flexibility and performance. Field-Programmable-Gate-Arrays (FPGAs) are integrated semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. Their major advantage is reconfigurability meaning that they can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Thus, FPGAs provide higher flexibility compared to ASIC solution. Although there are also one-time programmable (OTP)

FPGAs, the most dominant types are SRAM based which are capable of reprogramming while the design's evolving [47].

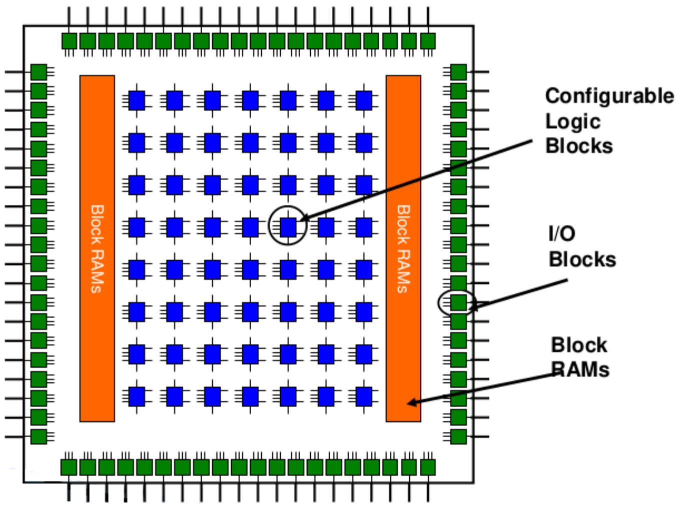


Figure 2.12: Field Programmable Gate Array (FPGA) schematic [9].

.

ASIC and FPGAs have different value propositions, and they must be carefully evaluated before choosing any one over the other. While ASICs outperform FPGAs, the second ones have evolved rapidly during the last decade and now they can easily reach significantly high frequencies. FPGAs consist of logic blocks, I/O blocks and a hierarchy of reprogrammable interconnects allowing blocks to be wired together and interact. They may also contain some complete memory blocks as well. These Configurable Logic Blocks can be configured to implement complex combinational functions or simply implement nearly any digital logic circuit.

FPGA is a constantly evolving technology, especially in terms of logic density and speed. Among the newest improvements in the FPGA world are System-on-Chip (SoC) FPGA devices. A SoC FPGA integrates a hard processor core and programmable logic on the same die [48]. Consequently, they provide higher integration, lower power, smaller board size, and higher bandwith communication between the processor and the FPGA. They also include a set of peripherals, on-chip memory, and FPGA-style logic array, and high speed transceivers. The three largest FPGA vendors, Xilinx, Altera and Microsemi, manufacture SoC devices having in common that they put a hard ARM processor.

Comparing the utilisation of a stand-alone processor and a stand-alone FPGA, the approach of a SoC FPGA is cheaper, consumes less power, and is easier to apply into a design. Combining the two components implies reduced design time, smaller physical size, hence lower cost. Another important benefit to using of SoC FPGAs is the faster communication between processor and FPGA. Thus, increased bandwidth and reduced latency are achieved. Therefore, SoC FPGAs have become a highly relevant competitive alternative to their traditional counterparts [49].

2.5.1 Xilinx Zynq SoC FPGA Architecture

In this thesis, the Hardware used consists of a Zedboard development board that hosts a Xilinx Zynq-7020 SoC device. At this point, a brief overview of the Xilinx Zynq-7000 SoC architecture is presented. The three main components that constitute the Zynq-7000 SoC FPGA, as depicted in Figure 2.13 are the following: the *Processing System* (PS), the *Programmable Logic* (PL) and the *AMBA AXI Communication Protocol*.

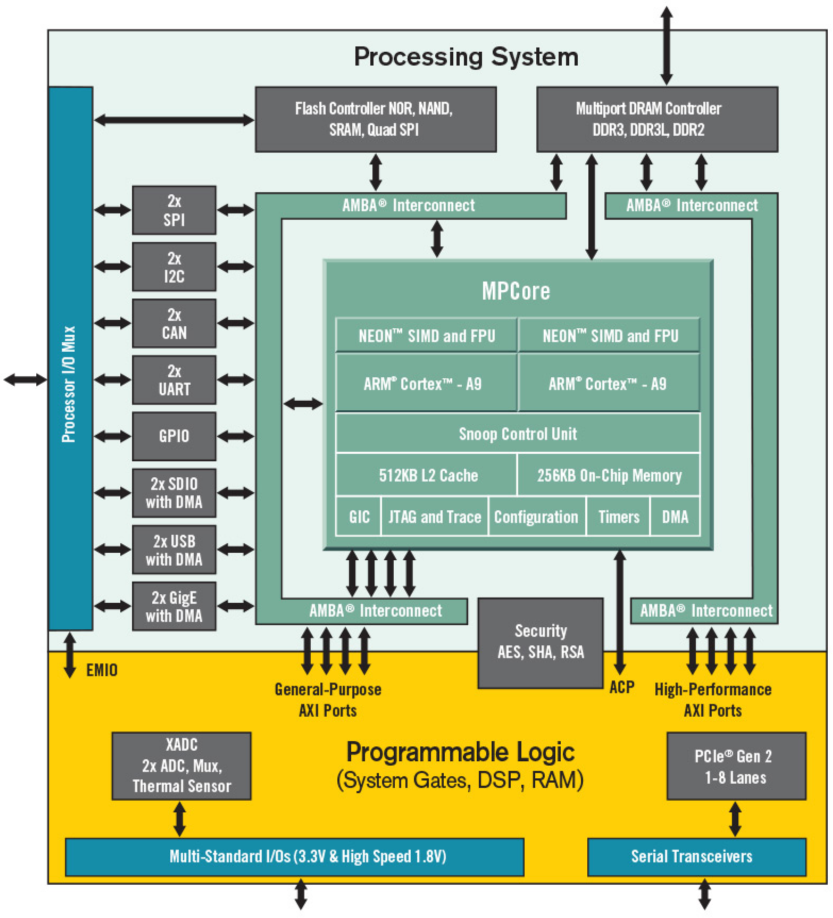


Figure 2.13: Xilinx Zynq SoC FPGA Architecture [10].

Zynq Processing System

The PS is equipped with the following:

- 32-bit Dual core *ARM Cortex-A9 processor* capable of up to 1GHz operation frequency
- 64 KB *L1 Cache Memory*
- 512 KB *L2 Cache Memory*
- 256 KB *On Chip Memory* (OCM)

- *NEON and FPU extensions* for Single Instruction, Multiple Data (SIMD) and floating point processing
- *Memory Management Unit* (MMU)
- *Snoop Control Unit* (SCU), which manages the data cache coherency between the two processors and L2 cache

Zynq Programmable Logic

Zynq-7000 SoC is equipped with the PS integrated with a 28 nm Artix-7 or Kintex®-7 based programmable logic which consists of:

- *Configurable Logic Blocks* which in turn includes:
  - LUTs
  - D Flip Flops
  - Multiplexers
  - Carry Chain Logic
- *DSPs*
- *Block RAMs*
- *Clock Tiles*
- *Input/Output Blocks*
- *Various Interfaces*
  - Gigabit Transceivers
  - XADC
  - PCI

AMBA AXI Communication Protocol

The AXI4 Standard is an interconnect protocol which sets up the communication between the Processing System and the FPGA. It belongs to ARM AMBA family of microcontroller buses and allows the connection and management of many controllers and peripherals in a multi-master design. It is worth mentioning that the protocol was optimised for FPGA implementation through coordinated development with Xilinx as its aim is high performance and high frequency system designs [50].

The AMBA AXI Communication Protocol follows a master-slave logic within its connection. It is distinguished in two phases which are the address/control phase and the data phase. Also, it has two separate channels for reading and writing and it allows single and burst-based transactions that may as well be out-of-order. That means that read data can be returned from slaves who response faster and are given priority above the slower ones. There three variations of AXI4 protocols which are presented in brief as follows:

- AXI4
  - Memory-mapped protocol
  - Data burst transfer of up to 256 data words
  - Bidirectional
- AXI4-Lite
  - Simplified memory-mapped protocol
  - Transfer of a single data word
  - Bidirectional
- AXI4-Stream
  - Non-memory-mapped
  - Burst transfers of unrestricted size
  - Unidirectional

In this thesis, we utilize the AXI4 Lite and the AXI4 Stream protocol in order to establish the communication between the PS and the PL so as to implement the HW/SW co-design for the accelerated clustering algorithm. Thus, a more extensive report about the aforementioned communication protocols is presented below.

2.5.2 AXI4 Lite

The AMBA AXI protocol is suitable for high-bandwidth and low-latency designs and provides high-frequency operation without using complex bridges. In addition it is suitable for memory controllers with high initial access latency and it offers makes the implementation of interconnect architectures flexible.

The key features of the AXI protocol are that it has separate address/control and data phases and that is capable of unaligned data transfers using byte strobes. Furthermore, burst-based transactions with only the start address issued are processed and other than that, it provides support for out-of-order transaction completion. Last but not least, it has separate read and write data channels, that lead to low-cost Direct Memory Access (DMA).

The AXI protocol is burst-based and defines the following independent transaction channels:

- read address
- read data
- write address
- write data

- write response

An address channel carries control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either *write data* channel to transfer data from the master to the slave, or *read data* channel to transfer data from the slave to the master. In a write transaction, the slave uses the *write response* channel to signal the completion of the transfer to the master. Figure 2.14 shows how a read transaction uses the read address and read data channels, while Figure 2.15 shows how a write transaction similarly happens. AXI4-Lite is a light-weight, single transaction

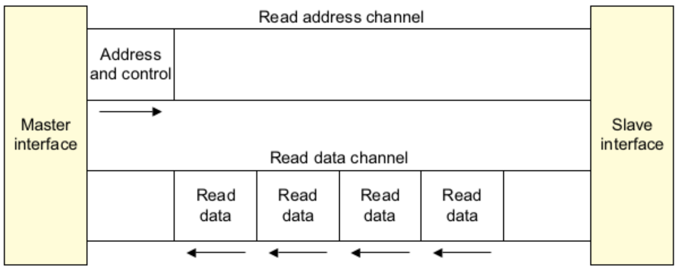


Figure 2.14: Channel architecture of reads [11].

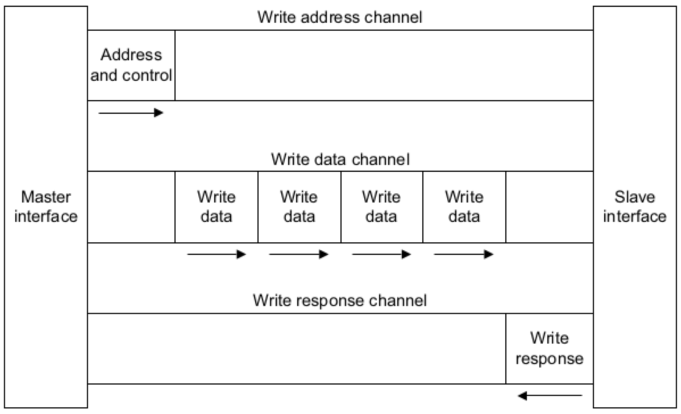


Figure 2.15: Channel architecture of writes [11].

memory mapped interface. It has a small logic footprint and is a simple interface to work with both in design and usage. The key functionality of AXI4-Lite operation is:

- all transactions are of burst length 1
- all data accesses use the full width of the data bus
  - AXI4-Lite supports a data bus width of 32-bit or 64-bit
- all accesses are non-modifiable, non-bufferable
- Exclusive accesses are not supported

2.5.3 AXI4 Stream

The AXI4-Stream protocol is similar to the AXI4-Lite that was described above. As well as the AXI4-Lite, it is used as a standard interface to connect components that wish to exchange data. The AXI4-Stream protocol is used for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required. Each AXI4-Stream acts as a single unidirectional channel for a handshake data flow.

The protocol supports multiple data streams using the same set of shared wires, allowing a generic interconnect to be constructed that can perform upsizing, downsizing and routing operations. Thus, this particular protocol can be better optimized for performance in data flow applications, but also tends to be more specialized around a given application space.

Unlike AXI4-Lite protocol, AXI4-Stream interface does not require an address to proceed to the transaction. On the other hand, a single transfer is defined by a single TVALID, TREADY handshake. The TVALID and TREADY handshake determines when information is passed across the interface. A two-way flow control mechanism enables both the master and slave to control the rate at which the data and control information is transmitted across the interface. Therefore, this is the main principle which defines the protocol, that in order for a transfer to occur, a handshake is required so both the TVALID and TREADY signals must be asserted [51].

2.5.4 Design Flow

Computer Aided Design (CAD) tools are used as an interface between the user and the hardware that turns the design ideas into FPGA programmable logic. Designs are written in HDL, which stands for Hardware Description Language, and they are converted by the CAD tool into a stream of bits used to program the FPGA. In this thesis we mainly used Xilinx Vivado design suite. The Design Flow followed to implement our design is shown bellow in Figure 2.16.

1. **VHDL Description**. We write the code in VHDL (Very High Speed Integrated Circuit-VHSIC Hardware Description Language) which describes the desired functionality of the hardware design. This includes all ports, logic, registers, arithmetic blocks and a few other hardware blocks.

2. **Functional Simulation**. During this stage, a behavioural simulation is executed in order to evaluate the logical functionality of the design. A waveform diagram is produced which simulates inputs, outputs, wired signals and basically any value imported in the design.

3. **Logic Synthesis**. Here, the HDL description is converted into a set of logic gates, Flip-Flops, adders etc. A netlist between the individual components is also produced.

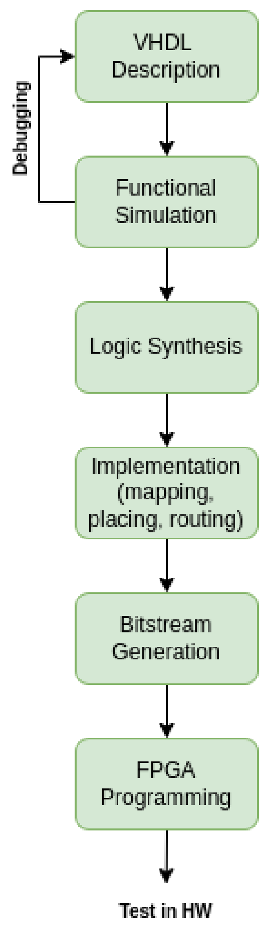


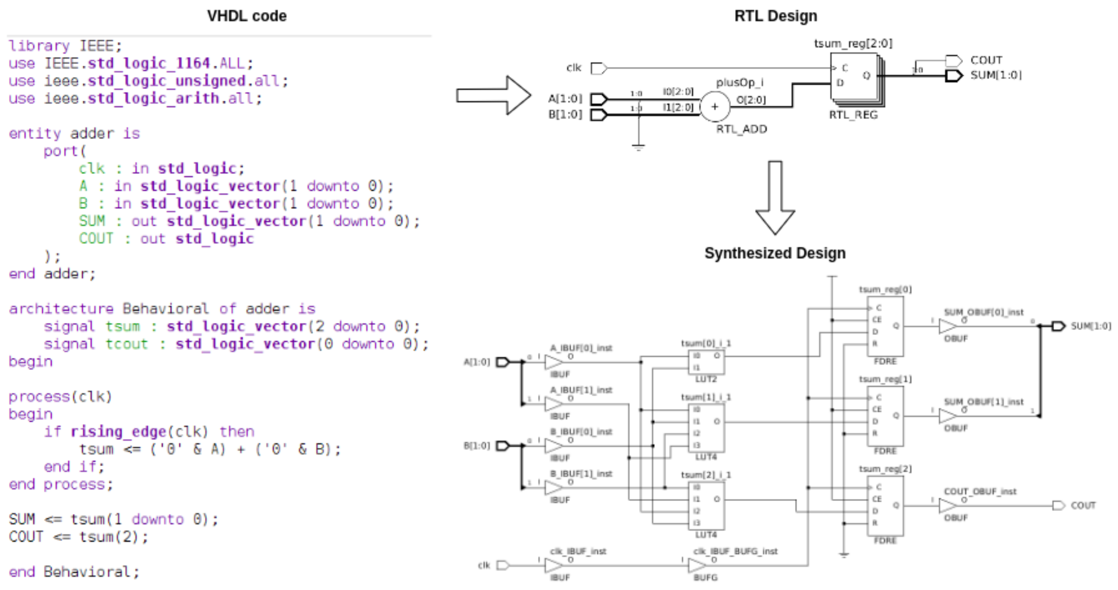
Figure 2.16: FPGA Design Flow.

4. **Implementation**. Here, the synthesized design is mapped to specific hardware of the platform. In addition, optimizations occur and all the connections are placed and routed in order to structure the designed circuit.

5. **Bitstream Generation**. A stream of bits called bitstream is produced after the extraction of the implemented design.

6. **FPGA Programming**. The generated bitstream is exported to the FPGA platform in order to configure and program it.

Vivado also has other useful features, that make designer's life easier. For example, an RTL design can be provided which exports the logic into a gate-level illustration. Figure 2.17 shows the different stages followed while transforming the HDL code to synthesized design. The HDL code describe a simple synchronous adder which is capable of adding two 2-bit numbers. The output products are the sum and the carry which identifies a possible overflow.


VHDL code

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity adder is
    port(
        clk : in std_logic;
        A : in std_logic_vector(1 downto 0);
        B : in std_logic_vector(1 downto 0);
        SUM : out std_logic_vector(1 downto 0);
        COUT : out std_logic
    );
end adder;

architecture Behavioral of adder is
    signal tsum : std_logic_vector(2 downto 0);
    signal tcout : std_logic_vector(0 downto 0);
begin

process(clk)
begin
    if rising_edge(clk) then
        tsum <= ('0' & A) + ('0' & B);
    end if;
end process;

SUM <= tsum(1 downto 0);
COUT <= tsum(2);

end Behavioral;
```

Figure 2.17: VHDL code to synthesized design conversion.

Chapter 3

Development on the Zynq SoC FPGA

Chapter 2 provided fundamental information and theoretical background necessary to fully understand the core of our methods regarding the proposed algorithm for star trackers. In the following sections, an extensive description of our proposed algorithm is presented. Also, the optimisation steps followed are discussed in this section considering the performance enhancement of the clustering algorithm.

3.1 HW/SW Co-design of Averaging 2D Binning

The first component of our HW/SW co-design performs averaging 2D binning. Pixel binning is a feature becoming increasingly common in camera-based applications.

Let us introduce the idea of what a pixel is. Pixels, also referred as photosites, are physical bits on a camera sensor that catch light in order to create pictures. A large pixel is able to collect more light than a smaller one. Thus, when pictures are captured in low light conditions, a sensor with large pixel size is required so as to achieve higher image quality. High resolution image sensors, usually have a smaller pixel size sensor, which might be limiting for the sensitivity of the camera.

In many applications, sensor resolution has exceeded the optical resolution which means that the additional hardware complexity to increase pixel density, might be incapable of increasing the image quality. However, if the size of each pixel in the high-density image sensor becomes smaller, low sensitivity and noise amplification may occur, especially in low light images. To solve this problem, a proposed technique that is usually adopted is pixel binning.

Pixel binning refers to the concept of combining the electrical charges of neighbouring pixels together to form a *superpixel* resulting to an effectively increased size of the image sensor. Therefore, we manage to increase the sensitivity of the resulting pixel at the cost of reduced spatial resolution. Additionally, in applications where large image data need to be processed, it is really beneficial for the algorithm's performance to decrease the image dimensions. Not only the processing time is reduced, but also the resources used are significantly decreased. More precisely, the required memory is remarkably less, as for example, the size of the image stored is under-quadrupled for a 2×2 binning. Usually, pixels are combined in 2×2 or 4×4 grids to compose a single pixel. The Figure 3.1 bellow depicts an example of two different levels of image binning. The original image has a size of 8×8 pixels which corresponds to the maximum resolution. By binning at 2×2, each 2×2 grid combines into a pixel and the resulting image has only 16 pixels leading to reduced resolution. Binning at 4×4 results to a 2×2 grid with just 4 pixels so arguably, there is a noticeable reduction of image resolution.

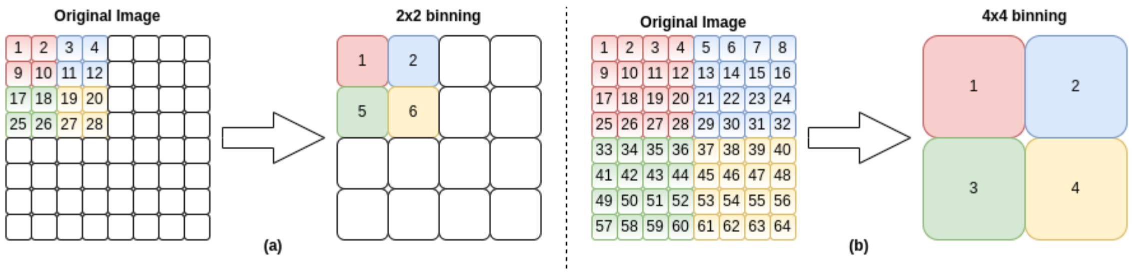


Figure 3.1: Different levels of averaging binning.

3.1.1 High Level Architecture

At this point we will present a top-down architecture of the integrated system design. Our approach is centered around the fast communication of the HW and SW components. More specifically, we established the communication between the processing system (PS) and the Programmable Logic (PL) of the SoC FPGA supposing that the image frames are received from the camera. In this thesis, instead of receiving the images directly from the camera, we assume due to simplicity that a set of different image frames captured by the CMOS sensor has been stored in the internal DDR memory of the SoC device. The implementation of our system is based on three stage processing pipeline as shown in Figure 3.2. The stages are described as follows:

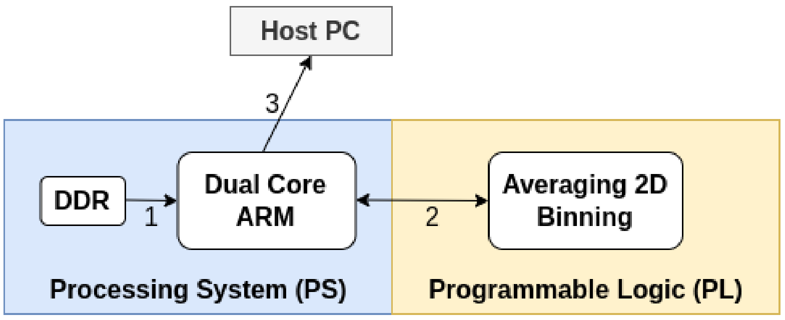


Figure 3.2: Processing flow of the integrated system.

1. The first stage refers to the stage of storing the image data to an allocated buffer in DDR memory. This process is implemented in PS using the well-defined libraries for the drivers of the memory unit in order to overcome the problem of limited memory resources of the PL.
2. Next, the processing stage follows. In this stage, the stored image frames are transferred from PS to PL for processing where the Averaging 2D Binning component is hosted. The binning component, and as well as the clustering component, which is described in Chapter 3.2, are implemented in PL so as to exploit the capabilities of the FPGA. In addition their parallel nature is well suited for implementation in this platform and therefore we are capable of reaching high performance.
3. In the last stage, the results obtained from the processing stage, are transferred from PL back to PS in order to be extracted to our Host PC for further use. Additionally, the system is evaluated as of the validity of its functionality as well as of the achieved performance. A SW-based algorithm is also implemented to be executed on the dual core ARM processor and the results received from our system design are compared to those produced by the SW operation.

PS-PL Communication

Image processing applications require high throughput rates and low latency data transfers between the processor and the HW accelerator. Taking into consideration these requirements, a streaming protocol is the most suitable approach for such applications. In a streaming protocol data transactions are determined through a handshake mechanism between the PS and the PL, which happens only at the beginning of the process and afterwards, valid data can be transferred at every consecutive clock cycle. In order to develop an efficient streaming communication, it should rely on a Direct Memory Access (DMA) mechanism. DMA enables accelerator to perform direct access to system memory and offloads CPU from being involved in data transfers. This mechanism considerably boosts the overall system performance as CPU can keep working concurrently on other tasks.

In this thesis, we use this approach to perform high throughput data transfers from PS to PL and vice versa. As we've discussed in the previous section, in ARM-based SoC FPGAs a widely used streaming protocol is the AXI4-Stream which relies on the well established AMBA AXI4 protocol. However, the implementation of AXI4-Stream is a non trivial task. The utilization of the protocol is arguably a major design challenge as of the substantial amount of time spent for customization of the FPGA accelerator interface and the validation of its operation in the integrated system.

Regarding the control signals transfers of the accelerator as well as the initialization of the DMA, simpler communication protocols have been developed such as the AXI4-Lite protocol which is also adopted in this project. In order to facilitate the implementation of the PS-PL communication a proposed generic approach is proposed, which can be deployed relatively fast and can be adjusted to meet specific application requirements.

Figure 3.3 presents an analytical block design of the integrated system. As we can see, the block design consists of the Zynq processing system, our binning component, the AXI DMA and some other peripherals. Each block is described bellow:

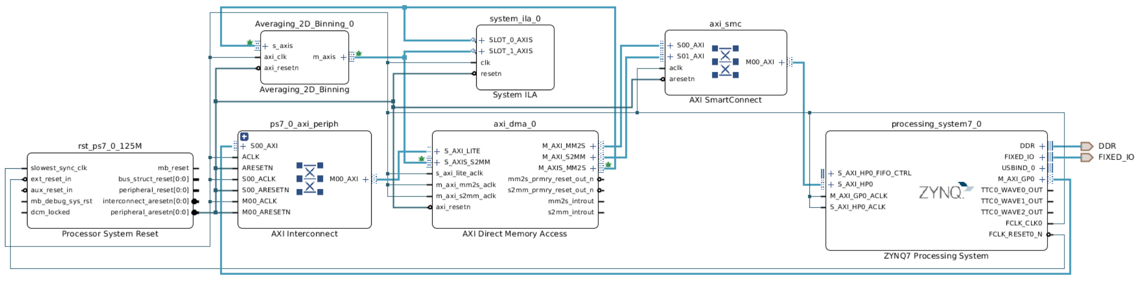


Figure 3.3: Block design of the HW/SW integrated system.

Zynq Processing System

The Processing System 7 core is the software interface around the Zynq-7000 platform processing system. The Zynq-7000 family consists of an SoC style integrated PS and a PL unit, providing an extensible and flexible SoC solution on a single die. The PS7 core acts as a logic connection between the PS and the PL while assisting you to integrate customized and embedded IP cores with the processing system using the Vivado IP integrator. More specifically it stitches the interface signals with the rest of the embedded system in the PL. The interfaces between the PS and PL mainly consists of three main groups: the extended multiplexed I/O (EMIO), programmable logic I/O, and the AXI I/O groups [52]. A block design which shows in detail all peripheral options and interfaces of the PS is illustrated in Figure 3.4.

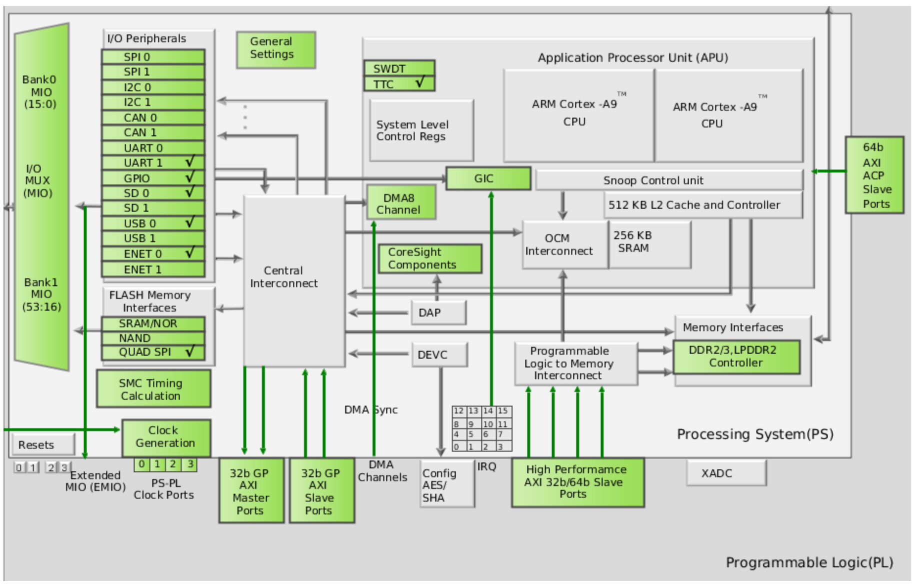


Figure 3.4: Zynq7 Processing System block design.

AXI Direct Memory Access

The Xilinx LogiCORE IP AXI Direct Memory Access (AXI DMA) core is a soft

Xilinx IP core for use with the Xilinx Vivado Design Suite. The AXI DMA provides high-bandwidth direct memory access between internal memory and AXI4-Stream target peripherals. Its optional scatter/gather (SG) capabilities also offload data movement tasks from the Central Processing Unit (CPU). It provides a number of features for increased productivity. These features are summarized below:

- It is AXI4 compliant providing enhanced flexibility to the user regarding the number of the employed PL interfaces
- It also offers optional Scatter/Gather (SG) Direct Memory Access (DMA) support
- It supports multichannel operation of up to 16 channels and it is capable of two-dimensional transfers in multichannel mode
- AXI4 data width support of 32, 64, 128, 256, 512 and 1,024 bits
- Primary AXI4-Stream data width support of 8, 16, 32, 64, 128, 256, 512 and 1,024 bits
- It inlcudes Optional Data Re-alignment Engine,which allows data realignment to the byte (8 bits) level on the primary memory map and stream datapaths

Initialization, status, and management registers are accessed through an AXI4-Lite slave interface. Figure 3.5 illustrates the functional composition of the core. Primary high-speed DMA data movement between system memory and stream target is through the AXI4 Read Master to AXI4 memory-mapped to stream (MM2S) Master, and AXI stream to memory-mapped (S2MM) Slave to AXI4 Write Master. AXI DMA also enables up to 16 multiple channels of data movement on both MM2S and S2MM paths in SG mode. The MM2S channel and S2MM channel operate independently. The AXI DMA provides 4 KB address boundary protection, automatic burst mapping, as well as providing the ability to queue multiple transfer requests using nearly the full bandwidth capabilities of the AXI4-Stream buses. Furthermore, the AXI DMA provides byte-level data realignment allowing memory reads and writes starting at any byte offset location [12].

Averaging 2D Binning

At this point, let us present a detailed description of the Averaging 2D Binning block. A high-level architecture of our developed component is presented within this subsection. The binning component works on 2×2 or 4×4 image regions with stride 2 or 4 respectively to calculate their mean value. Considering the requirements of our project, we chose to proceed with 2×2 bins as a more appropriate option so as to keep balance between the image resolution and its sensitivity. Figure 3.6 shows the extensive block diagram of our binning kernel.

As we have mentioned, our component has been configured in order to support the AXI4-Stream protocol so as to be able to receive valid data from the PS. More specifically, we customised the ports of the component with the specified signals which compose the

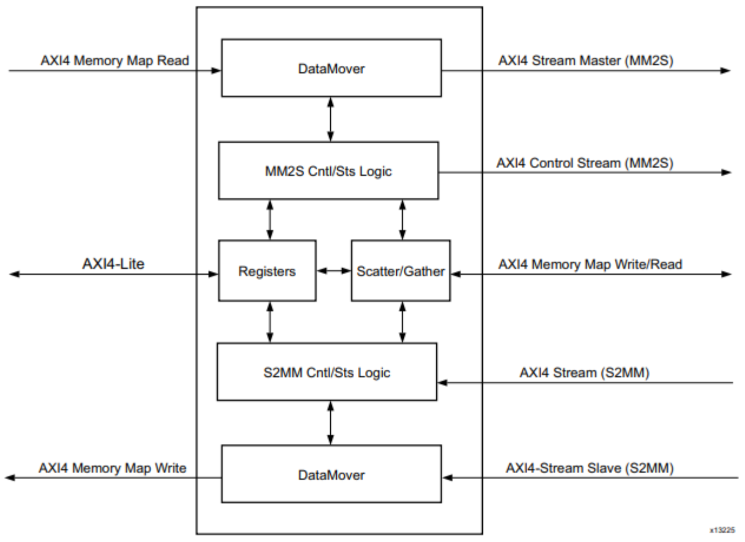


Figure 3.5: AXI DMA Block Diagram [12].

master-slave logic and we implemented the handshake mechanism. At this stage, our component receives the image data from TX_DMA (the transmitting part of DMA) and transfers the generated output to RX_DMA (the receiving part of DMA). Both TX_DMA and RX_DMA refer to the same AXI_DMA block, although it is a common designing choice to use two separate DMAs for the receiving and the transmitting part.

As we can see from Figure 3.6, the binning kernel is between two FIFO blocks. These blocks refer the Xilinx FIFO Generator IP which is actually a buffer that follows the First In-First Out logic. Both FIFOs have the minimum possible depth of 8 pixels. Supposing that each image frame has a size of 4MP, this equals to 2048×2048 pixels so after binning at 2×2, the output image contains 1024×1024 pixels. The main reason why we used these FIFOs, is to secure that no pixels will be lost during the transmissions from and to the DMA. For example, if the binning component for some reason is no longer ready to accept data, it sends a signal to the TX_DMA so as to pause the transmission. In that case, it is possible that the TX_DMA may continue sending pixels that are not supposed to be sent. So, these data are temporarily stored in the buffer until the binning component re-enables the transmission process.

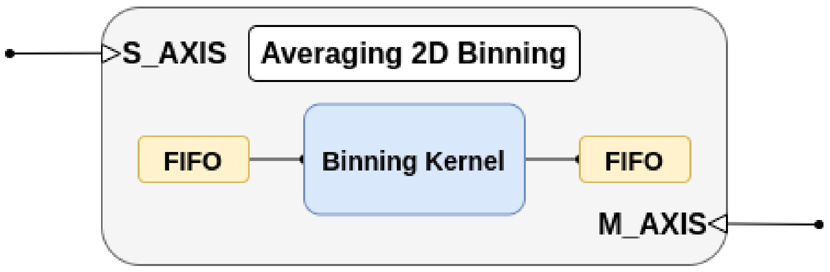


Figure 3.6: Averaging 2D Binning block design.

Binning Kernel

Now, let us present a more detailed description of the Binning Kernel. Figure 3.7 illustrates the block design of the binning kernel. As we can see, it consists of two main blocks which are the Serial-to-Parallel block and the Multiplier-Accumulator (MAC) unit. First of all, we need to clarify that the image data are received serially following raster scan ordering. This means that the binning kernel receives pixels row by row. Therefore, a serial-to-parallel block is needed in order to forward multiple data coming serially in consecutive clock cycles to the next blocks. Another issue is that an entire image row needs to be scanned so as to be able to calculate the mean values in 2×2 regions. The rows are processed in pairs of two and the first of each pair is stored temporarily in the FIFO buffer. This adds significant overhead to the operational throughput. However, the FPGA accelerator supports parallel processing, so mean values computation of each 2×2 grid and temporary storing of the following row happen concurrently.

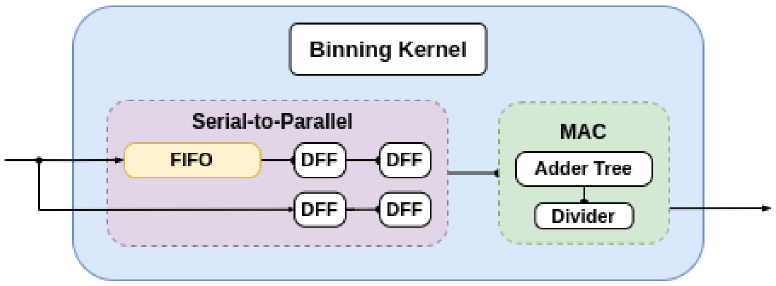


Figure 3.7: Block design of the Binning kernel.

The other main block is the Multiplier-Accumulator (MAC) unit. This is the unit where the 2×2 (or 4×4) bins are calculated. It consists of an *Adder Tree* and the *Divider* block. The Adder Tree, as the name suggests, is a tree structure of full adders for parallelism which performs fast additions. As we operate on 2×2 regions, the sum of 4 pixel values needs to be calculated. Thus, the structure of the Adder Tree block is shown in Fig. 3.8. There are two levels of additions which support parallelism. Firstly, the 4 pixels are grouped in pairs of two and the output sums are added together on the second level adder. Afterwards, the total sum is passed to the Divider which calculates the mean value simply by dividing by 4. The output refers to the new superpixel of the generated image.

3.1.2 Low Level Implementation

Previously, we presented a high-level description of the Averaging 2D Binning component. In this subsection, a more detailed overview of the implementation is discussed with a low-level perspective.

Handshake process

The handshake process is mainly defined by the following rules:

- The **TVALID** and **TREADY** handshake determines when information is passed accross the interface. A two-way flow control mechanism enables both the master and

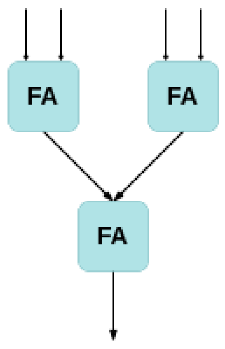


Figure 3.8: Adder Tree Structure for 2×2 binning.

slave to control the rate at which the data and control information is transmitted. For a transfer to occur both the **TVALID** and **TREADY** signals must be asserted.

- A master is not permitted to wait until **TREADY** is asserted before asserting **TVALID**. Once **TVALID** is asserted it must remain asserted until the handshake occurs. Once the master has asserted **TVALID**, the data or control information must remain unchanged until the slave drives the **TREADY** signal HIGH, indicating that it can accept the data and control information. In this case, transfer takes place once the slave asserts **TREADY** HIGH.

- A slave can assert **TREADY** before, during or after the cycle in which **TVALID** is asserted. The assertion of **TREADY** might be dependent upon the value of **TVALID**. A must either drive **TREADY** independently or pre-assert **TREADY** to minimize latency. For example, a slave drives **TREADY** HIGH before the data and control information is valid. This indicates that the destination can accept the data and control information in a single cycle of ACLK. In that case, transfer takes place once the master asserts **TVALID** HIGH.

Averaging 2D Binning

Now, let us explain how the handshake process is implemented on our VHDL code. We assume that each component which transmits data acts as a master and the one which accepts these data is the slave. In that case, TX_DMA is the master and Averaging 2D Binning is the slave. However, the last one is also the master in the handshake process between itself and the RX_DMA. So, the block that follows the A2DB (here is the AXI_DMA) indicates whether it is ready to accept data by asserting *TREADY_IN*. *TREADY_IN* as well as *TVALID_IN*, which indicates the transmission of valid data from the master (here the AXI_DMA), must both be asserted so that data can be passed within the input FIFO buffer to the Binning kernel. Hence, the signals wr_en and rd_en which enable writing and reading operations for the input FIFO are defined by these specific signals combined. The handling of these signals is particularly important because it is actually triggering mechanism for the binning process. This process is described in the Figure 3.9. We should mention that *TREADY_IN* and *TVALID_IN* signals are used indirectly by referring to the corresponding delayed registered signals to achieve synchronization.

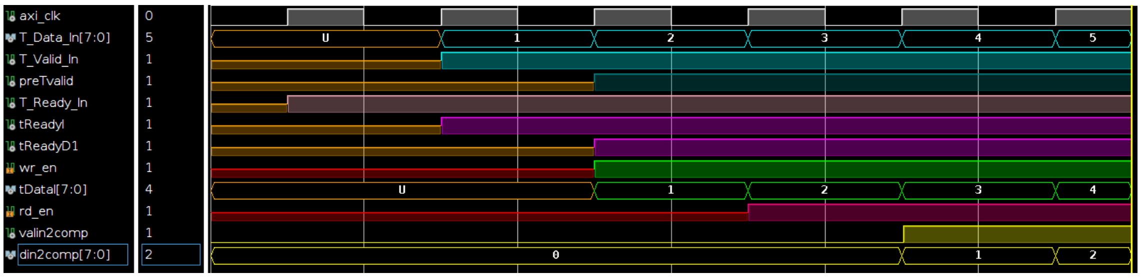


Figure 3.9: Behavioural Simulation of the input FIFO buffer handling.

Similarly, the same logic is used to handle the output FIFO buffer. On the contrary, writing operation is determined by the validity of the data produced by the Binning kernel while reading operation is obviously defined by the *TREADY_IN*. If AXI_DMA is ready to accept data, is drives *TREADY_IN* HIGH and therefore valid data are exported from the output FIFO directly to the output port of the component. The rd_en signal that enables reading from the buffer is driven by the output signal of an AND gate which has *TREADY_IN* and the inversion of FIFO's own empty signal as inputs. Empty signal becomes HIGH, if the buffer is empty. That means that reading is enabled if *TREADY_IN* is asserted, provided that there are valid data stored in the buffer.

It is noteworthy that there is a strong dependency between *TREADY_OUT* and *TREADY_IN* signals. If the following component that receives data from A2DB is ready to accept, then A2DB also indicates that it can accept valid data too, meaning that it will drive *TREADY_OUT* HIGH. Additionally, there is also an extra FIFO buffer of the same depth as the aforementioned output FIFO buffer but its width size is of 1 bit. This buffer is used to synchronize the *TLAST_OUT* signal with the rest of the output signals. Figure 3.10 shows the behavioural simulation of the how output FIFO buffer is handled. It's wr_en and valid out signal of the Binning kernel are identical. As for rd_en, it can be expressed as:

$$rd\_en = TREADY\_IN \ \mathbf{AND}\ [\mathbf{NOT}(empty)]$$

Binning Kernel

As we have already explained, the functionality of Binning kernel is based on a Serial-to-Parallel block and a MAC unit.

<u>Serial-to-Parallel</u>

We implemented our design so that it can support image binning at both at 2×2 and 4×4 regions using a generic variable named stride. For notational brevity, let us use the letter N when referring to the stride. In order to be able to use in parallel the serially incoming pixels, we need to store them temporarily until the full $N \times N$ region is completed. So in general, $N - 1$ entire image rows need to be scanned so that the computing of the

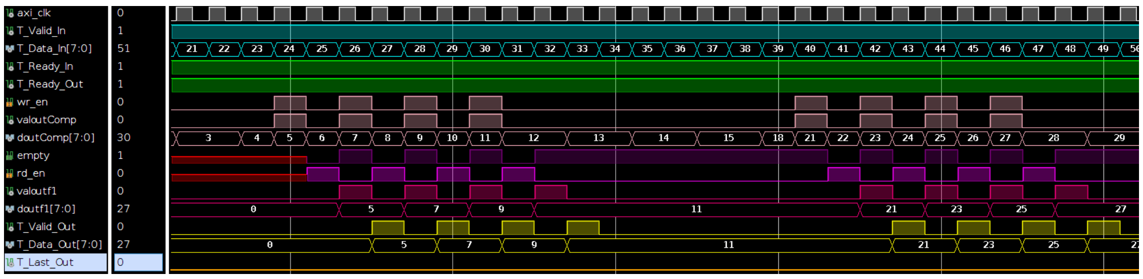


Figure 3.10: Behavioural Simulation of the output FIFO buffer handling.

Note: signals *valoutf1* and *doutf1* correspond to the valid and dout ports of the FIFO unit respectively.

mean values of each region can start. Therefore, exactly $N-1$ FIFOs of width equal to the original's image width are required. The Serial-to-Parallel also consists of a $N \times N$ grid of D Flip-Flops which are connected in series as shown is Fig. 3.11. The pixels that belong to the N-th row of the $N \times N$ region are directly driven to the last D Flip-Flops series. Afterwards, the pixels in the $N \times N$ grid are concatenated in a vector which is then forwarded to the MAC unit. The wire that refers to the incoming pixels, is connected to the input port of every FIFO. However, reading and writing operations are controlled by a Finite-State-Machine (FSM). The FSM functionality is described as follows: the number of the incoming pixels is counted and each of them is stored in the corresponding FIFO according to its position on the image plane. For example, the pixels of the first row are stored in the first FIFO, the pixels of the second in the second one and so on. The FSM also produces *enableMac* signal which indicates whether the MAC unit receives valid data. As we can see in Fig. 3.12, MAC unit receives valid data every $N-1$ clock cycles.

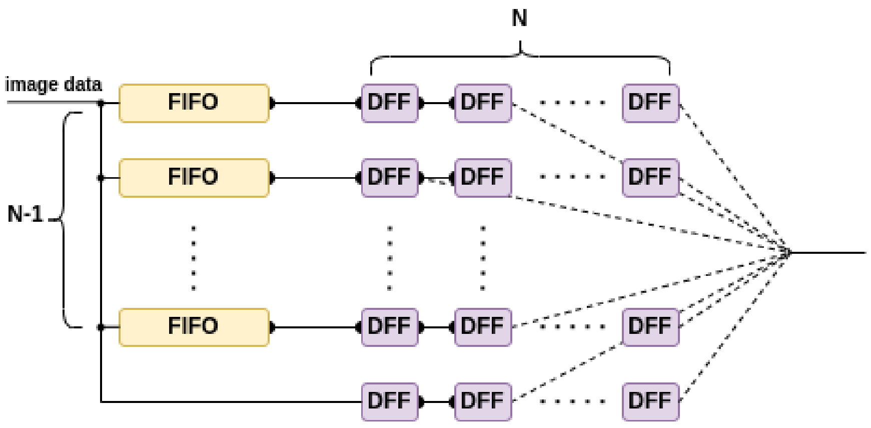


Figure 3.11: Serial-to-Parallel block.

<u>MAC Unit</u>

As we have already described, MAC unit consists of an Adder Tree and a Divider. The Adder Tree is a structure of Full Adders. The generic block design of MAC units is

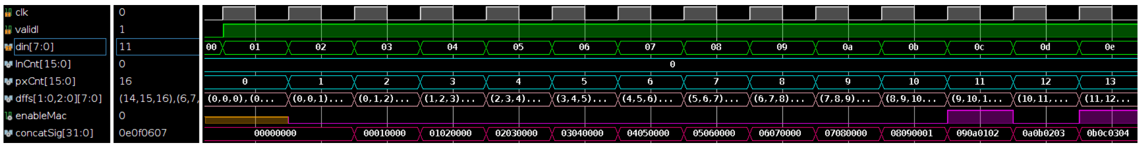


Figure 3.12: Behavioural simulation of Serial-to-Parallel operation.

depicted in Fig. 3.13 regarding stride N. The tree structure has exactly 2 and 4 levels of full adders for stride equal to 2 and 4 respectively. The exact number of levels of the tree structure is generalised for strides at powers of two as $log_2(N^2)$. At first level, there is a certain amount of full adders which is defined as $\frac{N^2}{2}$, where N refers to the stride. Each subsequent level as we descend towards the base of the tree structure, consists of a number of FAs that is equal to the one of its prior level divided by two. We should mention that the bullets in the block diagram refer to one clock cycle delay caused by the addition of D Flip-Flops. That is a very common technique used when designing pipelined components so as to decrease the critical path. The critical path is actually the longest path in the circuit and is a major limiting factor of the clock frequency. Supposing that each pixel's value is represented by a k-bit number, the resulting sum after each addition has a length of k+1 bits, so as to handle a possible overflow. Thus, the total sum is a number of $k + log_2(N^2)$ bits. As for the $TLAST\_OUT$ signal, MAC unit utilizes a counter for the generated pixels, and knowing the exact number of them in the new image, it is able to determine when the last pixel value is calculated. The total pixels number in the generated image is used indirectly by converting it to its corresponding width and height.

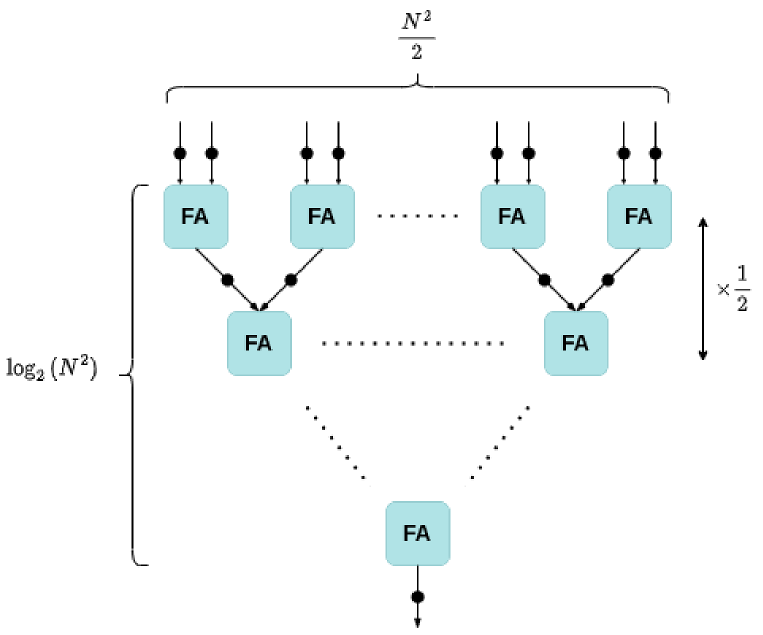


Figure 3.13: Generic block design of Adder Tree.

Divider

In computer programming, arithmetic shifts can be really useful as efficient ways to perform multiplication or division of binary numbers by powers of two. Shifting left by n-bits on a binary number has the effect of multiplying it by 2^n, shifting right by n-bits results to a division by 2^n. Therefore, the divider simply performs an arithmetic right shift by N-bits to the output sum of the Adder Tree, where N is the corresponding dimension of the squared grid used for binning. For example, if we operate binning at 2×2 regions with stride of 2, an arithmetic right shift by 2 bits occurs meaning that we divide by 4 and that is the resulting mean value of the generated pixel. Considering that binary numbers cannot be expressed with decimal places, the result after division is rounded to the nearest integer less than or equal to the actual result as if it was computed with a calculator. Figure 3.14 shows the behavioural simulation of the MAC unit block while binning at 2×2.

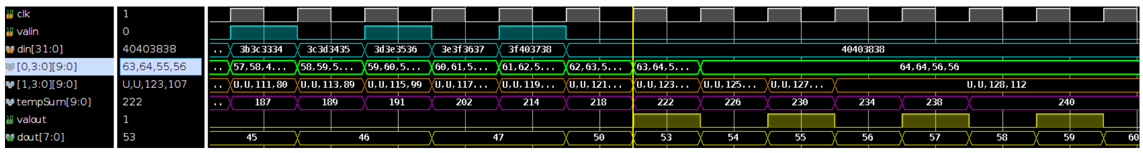


Figure 3.14: Behavioural simulation of MAC operation.

3.2 HW/SW Co-design of Clustering

In this section the main core of our project is presented, which refers to the proposed cluster detection algorithm for Star Trackers.

3.2.1 Cluster detection algorithm

In this current thesis, we developed an efficient algorithm that is able to detect clusters in an image of a portion of the night sky. Clustering is the first processing step in a star tracker operation. It is during this step where hardware optimisation is done to improve operation frequency aiming to real-time performance. The proposed algorithm is designed for hardware and is adapted to a MATLAB source code provided by Infinite Orbits.

A star on an image sensor can be considered a point light source for all practical purposes. Generally, an image formation through a lens can be described by a Point Spread Function (PSF). Depending on the aperture size and focus setting, the blurring of the point light source changes. Spreading introduces additional information about the star as it spreads the light source over several pixels. This allows centroiding algorithms to determine the centre of a star to sub-pixel accuracy. Depending on the intensity of the light source, defocusing is done so as to have stars spread over 3×3 to 5×5 pixel regions. For the purposes of a star tracker, light spreading can be accurately approximated as a 2D Gaussian distribution. Figure 3.15 illustrates a zoomed-in image of a single star that is slightly defocused.

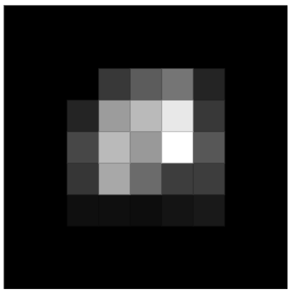


Figure 3.15: Visualization of light spreading in a defocused star.

A traditional approach for star detection algorithms is described as follows. Firstly, each pixel in the image is compared to a threshold that is usually pre-defined or it can be adjusted dynamically. If the pixel value is greater than the threshold, a region of interest is identified. Then the searching is focused on that specific Region of Interest (RoI), where all the neighbouring pixels are checked with an alternative slightly decreased threshold. Neighbouring pixels that are above the so called growing threshold, are clustered together and form a cluster in a maximum grid of 5×5 pixels. Once all of the neighbouring pixels are grouped, and the region is considered valid, a centroiding processing step is performed on the region pixels. In this work, the optimised algorithm follow the same logic as the traditional approach but is adapted for hardware-friendly calculations. When designing in hardware, data structures and lack of useful functions are among the biggest challenges. This cluster detection algorithm is reiterative by its nature and uses several recursive subroutines. Thus, it is vital to develop an algorithm which operates efficiently regarding the operation frequency. That is a non trivial challenge, but we tried to utilize the characteristics of the FPGA accelerator such as its ability for parallelism in order to improve performance.

Growing Region Algorithm

As we already mentioned, the whole image plane is checked and pixels with value above the threshold are stored temporarily in a stack data structure. Afterwards, each pixel is considered as centre pixel of a cluster, which is referred as "starting pixel", and forms a Region of Interest (RoI) where the search is focused. In every iteration while searching within the RoI, the 8 adjacent pixels around the centre one form a 3×3 grid called "Neighbourhood of Search" (NoS) and each pixel in it is compared to the growing threshold. The same procedure is repeated for every pixel inside the RoI whose value is greater than the growing threshold and hence, it is considered as part of the cluster. As we will describe extensively in the following paragraphs, the image is stored in a memory block at the PL. So every reference to a pixel is actually a reading operation resulting to a memory transfer so as to bring the required data. This is a considerably slow operation as we know from computer architecture, so what we actually do is bringing from memory exactly the pixels needed to form the NoS and don't belong in the 3×3 region in which the current central pixel was detected. An illustrative description of how the nine different Neighbourhoods of Search are configured is shown in Fig. 3.16. The noted star represents the centre pixel of the block which was identified during the first thresholding process. Each pixel in the 3×3 NoS, is noted with a unique number that refers to its position in it. For example, if the pixel on the right corner is part of the cluster, it is stored in the stack so as to check its neighbouring pixels. So, instead of searching the entire 3×3 region, only 5 pixels need to be checked. The search of the alternative cases of pixel position happens in a similar way. The identified pixels through the thresholding process, are grouped together and form a square cluster that can be either 3×3 or 5×5. Another challenge for our design was to avoid double and triple checks for the exact same pixel in the RoI. The algorithm while searching for possible pixels that form a star, performs repeated checks in 3×3 regions, which belong to a relatively limited RoI. Hence it is very likely that the same pixel might be checked more than once, which is not only restrictive in terms of efficiency, but it may also lead to endless loops. This is a non trivial problem as in hardware we have to deal with hard-to-handle data structures. What we did in order to resolve this, was introducing a boolean map where the pixel positions in the image plane are assigned. Every time a pixel within the RoI is checked, its position in the map, which refers to the corresponding position on the image, is marked.

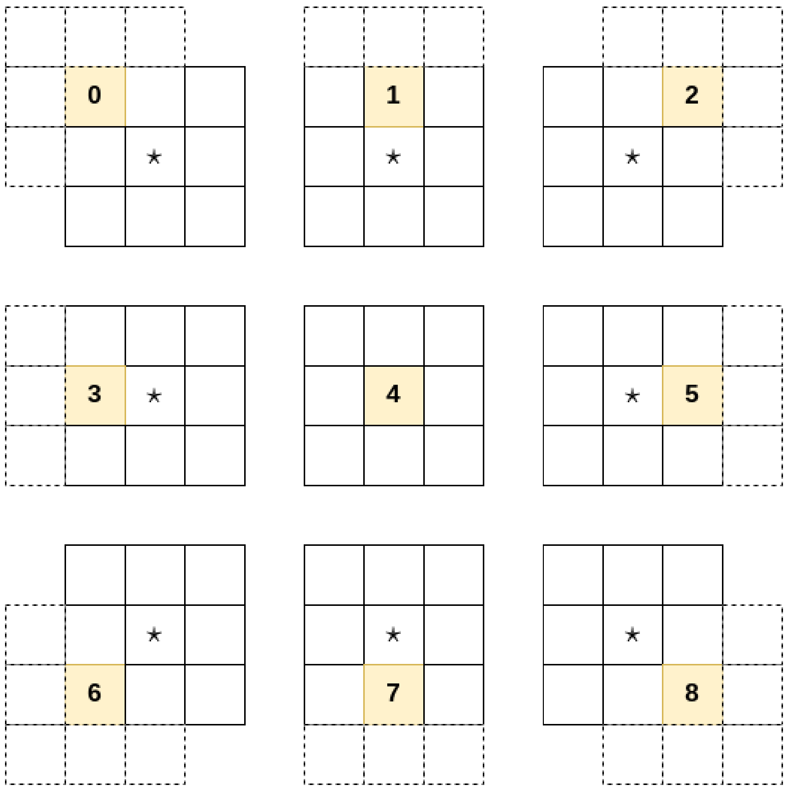


Figure 3.16: Configuration of Neighbourhoods of Search.

3.2.2 High Level Architecture

The proposed algorithm, is designed for hardware in order to take advantage of the features offered by the FPGA accelerator. Here, the Clustering component is hosted on the PL. Thus, it was implemented in a way to support AXI4 Streaming protocol in order to be able to communicate with other peripherals. More specifically, the Clustering component receives data directly from the DMA component as it is illustrated in the block design in Fig. 3.17.

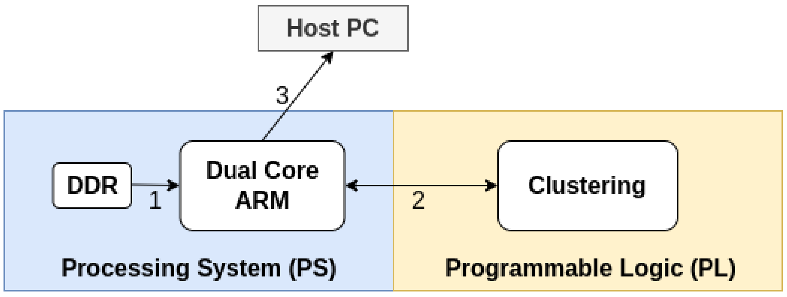


Figure 3.17: Processing flow of the HW/SW Co-design for Clustering.

As we can see, this block design is almost identical with the one depicted in Fig. 3.2, although here the Clustering block took the place of the A2DB. In this case, the only difference in the processing pipeline is the operation which is executed. The establishment of the communication between the PS and the PL was based on the same logic and ideas described analytically in subsection 3.1.1. However, it is worth mentioning how the blocks interact. Due to the recursive nature of the clustering algorithm, it is imperative to store the image in a memory block so as to be easily accessible. Therefore, we need to store the image in the PL where the algorithm is performed. The problem here is that the FPGA has limited resources, especially regarding memory, and so it cannot store the entire image even after reducing its size after binning operation. This was a major challenge for our design. We managed to resolve this issue by partitioning the image and transmitting it into windows. Each window has certain dimensions which are pre-defined and are expressed by two generic variables. The height is referred as *winHeight* and the width as *imageDim*. In order to be more accurate and to provide valid clusters, each window has an M-lines overlap defined by *maxInd* variable. Figure 3.18 shows exactly the configuration of the sliding windows.

Clustering component

As we have already mentioned, Clustering component supports AXI4 Stream interface in order to receive the pixel data after the binning process. The handshake mechanism is implemented similarly as in the A2DB. The main difference is that we need to control data transmissions as the image is sent partitioned. When a full window has been received and stored, we need to send an identification signal to the AXI_DMA to stop transmission of the original image. The stages of this pipelined process are described in the following diagram in Fig 3.20. We control the data transmission by driving the TREADY_OUT

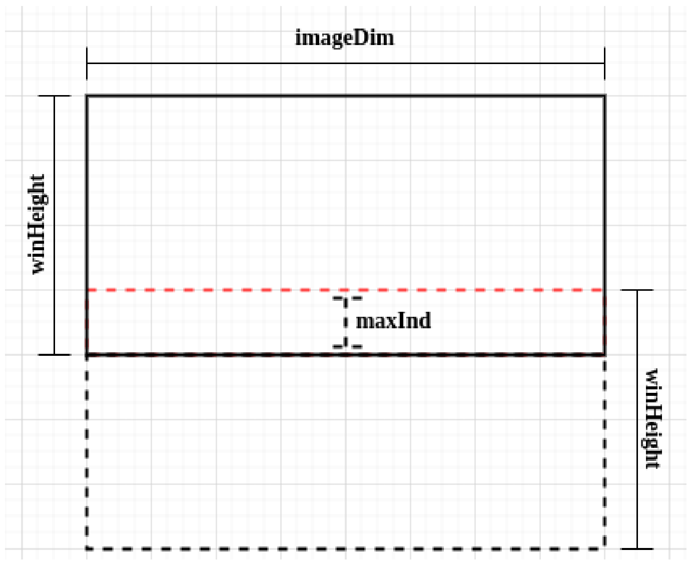


Figure 3.18: Image partitioning in sliding windows.

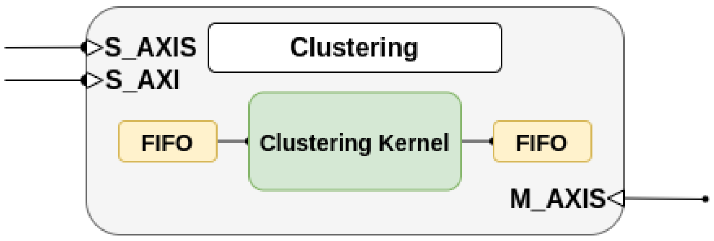


Figure 3.19: Clustering block design.

signal. Again, a pixel counter is used so as to determine whether an entire sliding window has been received.

Considering that a star tracker should be able to operate under different conditions that might affect its functionality and therefore its precision, we designed the integrated system in a way that it can adjust to those conditions in order to meet the requirements of the mission. For example, the image sensor used by the star tracker is a major factor that have a significant impact on its performance. Depending on its specific characteristics such as the lens apperture, sensitivity or calibration as well as the lighting conditions, the image that is captured is affected substantially. Thus, we implemented a design which is capable of tuning the threshold dynamically regarding the SNR ratio. For example, invalid light sources such as planets, the moon or reflections from the sun are perceived as noise sources. In these cases, it is really hard to distinguish a star from noise, so what we actually do is configuring the threshold according to the levels of noise in the image. Thus, the pre-defined value of the threshold implies a noise reduction technique and is defined experimentally. In order to configure that value dynamically during the execution of the algorithm, we set AXI4-Lite interface. As we can see in Fig. 3.19, there is an extra set of ports named AXI_LITE, so as to be able to receive data directly from the PS without requiring a handshake. More specifically, AXI4-Lite protocol supports a set of 32-bit registers which are addressed to well-defined positions in memory. Therefore, the desired values of the threshold and the growing threshold are written to these specific addresses and the Clustering component determines them, simply by reading the corresponding register. Hence, our HW/SW co-design can be adjusted regarding the project requirements and offers support for several camera types.

Finally, let us explain how TLAST_OUT signal is driven. Due to the nature's algorithm it is clearly uncertain when the last cluster will be detected. Although it is quite unlikely, there is a possibility that no clusters will be detected at all. Even in that case, our component must indicate that the process has been completed. This is done by asserting the TLAST_OUT signal. Again, there is a signal named *lastIter* which is set when the last sliding window is received. Also, the Clustering Kernel has a port that drives a signal called *endProc* which indicates that the search of the current window is finished. TLAST_OUT is controlled by the logical conjunction of the aforementioned signals and is followed by a valid data transmission of a zero pixel.

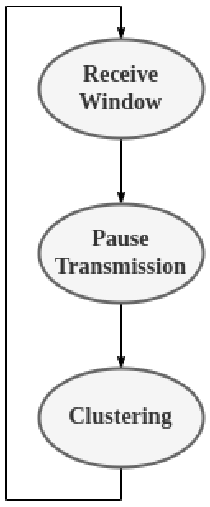


Figure 3.20: Data transmission pipeline.

Clustering Kernel

At this point, an in-detail description of the Clustering Kernel is presented. The exact block diagram is depicted in Figure 3.21. As we can see, this block consists of several units which interact and cooperate in order to perform the star detection algorithm. The functionality of the clustering component is described by the following:

upperThresLoc

This unit, as the name suggests, performs a streamlined thresholding operation while receiving the binned pixels. Every pixel that is received, is compared with the upper threshold and if its value is greater or equal, its corresponding coordinates are stored in the mainStack. Actually, mainStack consists of three individual stack structures, two for the x, y coordinates of the pixel and one which refers to its position in the 3×3 NoS.

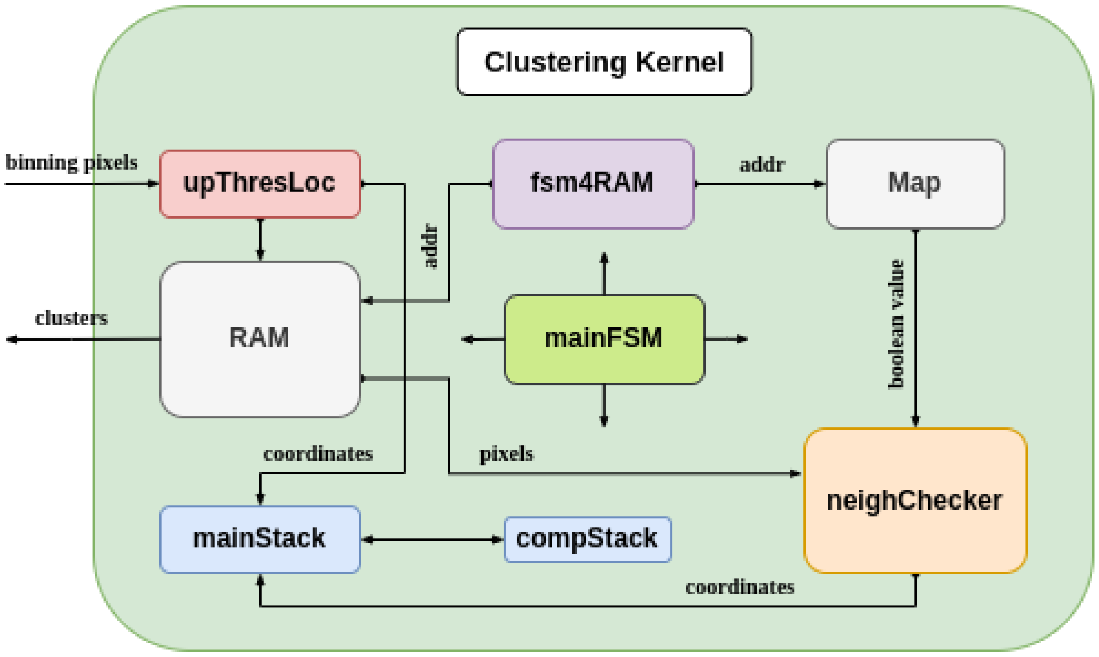


Figure 3.21: Block design of the Clustering kernel.

Every pixel identified during this process is supposed to be the initial point of the search and furthermore it is the centre pixel of the square RoI. Thus, these pixels are referred as *starting pixels*.

<u>fsm4RAM</u>

This is the second main block of the Clustering Kernel. What it actually does, is converting the x, y coordinates of the pixels to the corresponding address of the RAM memory block. This is a non trivial computation, because as it will be described in the following paragraphs, RAM is considered as a cyclic buffer and all the functional operations refer to a virtual address that needs to be transformed into the physical address. Moreover, in every iteration the virtual addresses are mapped in a different way to the corresponding physical addresses. Additionally, this block controls reading and writing operations of both RAM and Map units.

<u>neighChecker</u>

The third main block of the Clustering unit is called neighChecker. This block performs the growing region algorithm, meaning that it checks the adjacent pixels in a 3×3 region. This was a major challenge for the proposed algorithm, since the way that it is executed, affects the performance of the whole system significantly. As it has been already described previously, an efficient pixel search mechanism was introduced in this thesis. Therefore, this block needs to adapt to this mechanism in order to identify the exact position of the incoming pixels in the image plane. Thus, it has knowledge of the centre pixel coordinates in 3×3 grid, and it compares each neighbouring pixel to the growing threshold. The results of this comparison are extracted as pixel coordinates as long at the pixel is part of the cluster. These coordinates as well as the pixel's position in the NoS are stored in the corresponding stacks so that the referenced pixel to be consequently checked. The neighChecker unit receives the centre pixel's coordinates and a number that refers to its unique position in the block that was detected. Having this information, it can determine which of the 9 cases of NoS is currently being checked. For each distinct case, there are extreme cases which must be taken into account. These cases arise when the central pixel of the 3×3 region is located at the edges of the window, and therefore the NoS is configured differently. Figure 3.22 shows every possible case where the centre pixel of the block is a starting pixel.

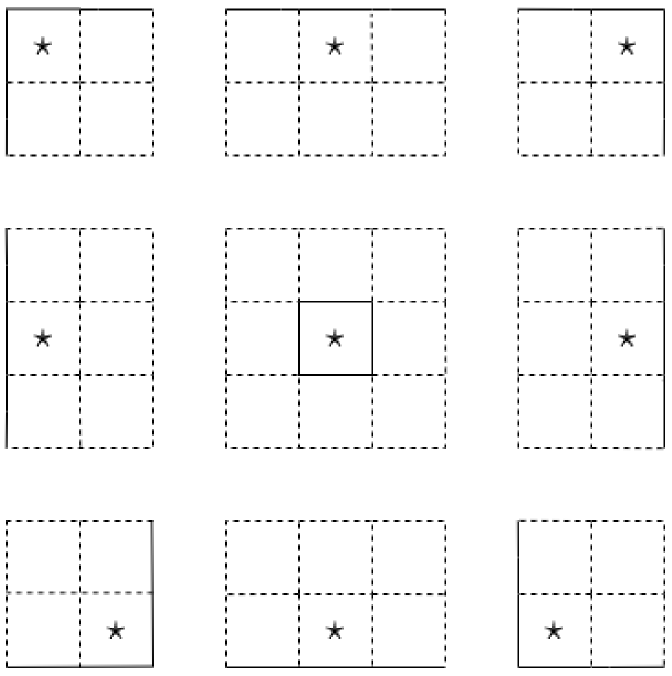


Figure 3.22: Illustration of every possible case for reading operation of starting pixels.

At this point, let us describe how cluster determination occurs. As it's already mentioned, pixels which are potentially part of a cluster are stored in mainStack. The algorithm begins the search from the initial point referred as starting pixel. Then, the RoI is determined according to a generic variable called *maxInd* that is pre-defined regarding the project requirements. In this thesis, we assumed that there are clusters of maximum 5×5 pixels. Thus, the value of *maxInd* is equal to 3, so as to limit the searching area and to be able to detect 5×5 clusters. An illustration of the RoI determination is shown in Fig. 3.23 bellow. Once the searching within the RoI begins, the value of the stack pointer of the mainStack is stored temporarily and specifies the initial state of the stack. During the search into the RoI, the value of the stack pointer fluctuates depending on the amount of pixels that belong to the cluster. So eventually, when the value of the stack pointer becomes equal to the initial value, the search has been completed. In order to identify the cluster dimensions, four variables are used which are initialized with the starting pixel's coordinates. Each variable refer to an up, down, left and right movement while searching into the RoI. Upon completion of the cluster detection, these variables are forwarded to an internal block called *clusterCal*. This block is responsible for computing the cluster dimensions. Furthermore, it determines the coordinates of the up-left pixel of the cluster. However, it might be possible that a single pixel has been detected within the RoI. In this case, we suppose that an individual pixel cannot form a cluster so it is considered invalid and is rejected. Additionally, it is possible that a cluster of 7×7 pixels is detected. In that case, pixels in a 5×5 grid out of the total 7×7 region, are grouped together to form a smaller cluster.

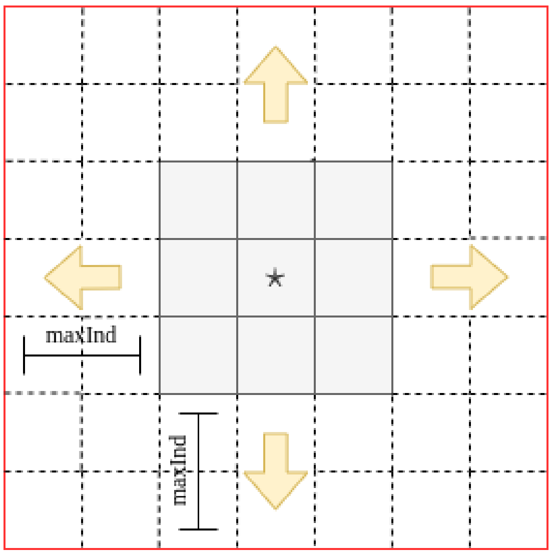


Figure 3.23: Determination of Region-of-Interest.

<u>RAM</u>

RAM is a memory block where each sliding window of the received image is stored. It supports reading and writing operations which are controlled by the enabling signals *re* and *we* respectively. The depth of the RAM block is defined regarding the window's dimensions, hence it is equal to $winHeight \times imageDim$ pixels (fig. 3.18). Since the sliding windows are overlapping, the total amount of the incoming binned pixels in each iteration (except for the first and the last one) is equal to $(winHeight - maxInd) \times imageDim$. Thus, we use a virtual pointer that points to the physical base address of the current sliding window. Every block in each iteration refers to the virtual coordinates of the pixel in the window. Figure 3.24 represents the addressing operation for the first three iterations with an overlap of 3 image rows. Due to simplicity, we suppose that the RAM block is a 2D array of the same size as the window. On the left side, the numbers refer to the physical addresses as if they were actual image rows. On the right side, the virtual pointer (vp) points to the corresponding physical base address of the window in RAM. The numbers on the right side, indicate the physical coordinate of the image that refers to its row.

<u>Map</u>

Map is actually a memory unit that operates as a boolean map. It is designed similarly

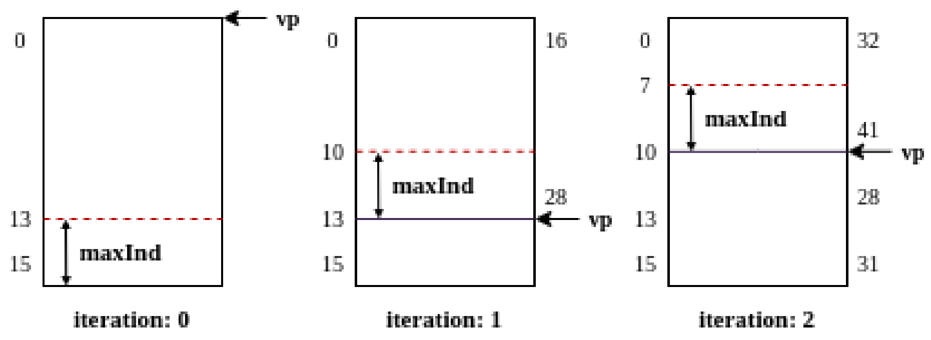


Figure 3.24: Example of the addressing operation.

to the RAM block and has the same size. The only difference between them is that it stores data of 1 bit. Reading and writing operations are driven by the same signals used in RAM. Each address of the block, represents a pixel on the window in the corresponding address in RAM memory. As we described earlier, when checking of the neighbourhood is completed, the central pixel's position on the Map is marked with an ace which indicates that it has been checked. Once, this happens, it cannot be checked again meaning that a pixel can belong to only one cluster.

mainStack

The mainStack is used for notation brevity, and it actually refers to a set of three stack structures. The first two are used to store row and column coordinates of the pixels on the sliding window. The third one is used to store the relative position of the detected pixel in the 3×3 NoS. Each position is represented by a number in range 0 to 8 and is used to improve reading operation from the memory. Hence, 4 bits are required for the data width in order to be able to store the specific positions. The stack has an enabling port and a port that defines whether push or pop happens. In addition, it has an output port for the stack pointer so as to provide the current state of the stack as well as *empty* and *full* output signals.

compStack

This unit consists of a set of three stacks as well and is a complementary structure to the mainStack. These stacks are used to store the same objects as in the mainStack. Starting pixels with row coordinate greater or equal to $winHeight - maxInd$ are stored in compStack in order to be checked within the next sliding window. Starting pixels that are bellow that borderline, cannot define a 7×7 RoI and we assume that the cluster is better detected among the following image rows. Before, storing these pixels to the compStack, we need to convert the coordinates into the virtual coordinates of the next sliding window so that they follow the same notation.

mainFSM

The last block of the Clustering Kernel is the mainFSM which is a finite state machine that controls the interaction between the individual components. It drives the signals of reading and writing operations for both the RAM and the Map and it actually handles the individual processing steps of the clustering algorithm. The pipeline of these consequent stages is shown in the following diagram.

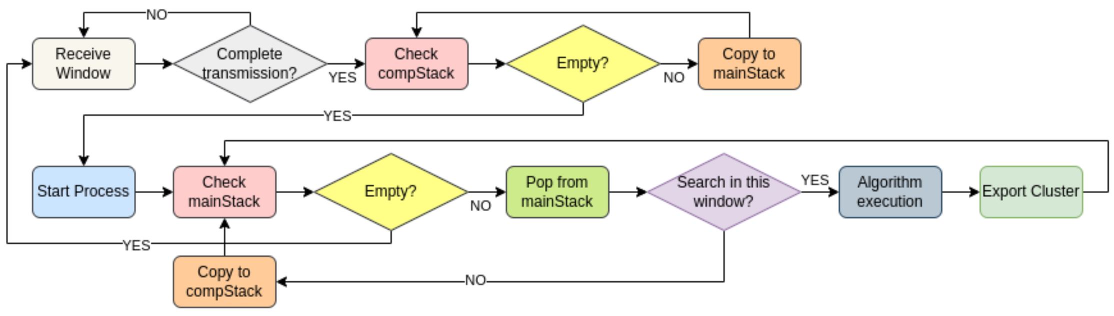


Figure 3.25: Pipeline of the processing steps handled by mainFSM.

Design parameters

At this point, the exact values of the main generic variables that have been used in this design are listed bellow:

- Data width: 12 bits
- Original Image Size: 2048×2048
- imageDim: 1024
- winHeight: 16
- Column Width: 10 bits (Columns in range 0 to 1023)
- Row Width: 4 bits (Rows in range 0 to 15)
- Cases Width: 4 bits (Cases in range 0 to 8)
- Depth of RAM and Map: 16384 (imageDim×winHeight)
- Address Width: 14 bits (Address in range 0 to 16383)
- Depth of mainStack: 1024
- Depth of compStack: 512

3.2.3 Low Level Implementation

In the previous subsection, a high-level description of Clustering component was presented. Here, a more detailed overview of the low-level implementation is discussed.

Clustering Component

As shown in Fig. 3.19, two FIFOs are used in order to support the handshake mechanism. Reading and writing operations for these IPs are handled exactly as it was described for the Averaging 2D Binning. However, it is worth presenting how the data transmission is controlled. Firstly, a set of constant variables is introduced so as to indicate the exact number pixels that are received per sliding window. The total number of iterations required so as the entire image to be processed is obtained as follows:

$$\text{itDiv} = (\text{imageDim-winHeight}) \div (\text{winHeight-maxInd})$$
$$\text{itMod} = (\text{imageDim-winHeight}) \bmod (\text{winHeight-maxInd})$$

```
if   (itMod = 0) {
totalIter = itDiv+1
} else {
totalIter = itDiv+2 }
```

In each iteration, a certain amount of pixels are supposed to be received. For every possible case, the exact number of the incoming pixels are calculated as follows:

$$\text{px1stWin} = \text{imageDim}\times\text{winHeight}$$
$$\text{pxPerWin} = \text{imageDim}\times(\text{winHeight-maxInd})$$
$$\text{pxLastWin} = \text{imageDim}\times\text{itMod}$$

Every time the transmission of an entire window is completed, an iteration counter increases its value. There is also a pixel counter for the incoming data so as to indicate whether the transmission should be paused comparing its value to the aforementioned variables. Thus, TREADY_OUT is driven by a signal named *pxtReady* based on the pixel counter as well as by an output signal of the Clustering kernel named *recNext* which indicates that the search operation is finished. Also, there is no need to assign TREADY_OUT when the entire image has been checked so the inversion of *lastIter* is used as well.

$$TREADY\_OUT = (\textit{pxtReady}\ \mathbf{OR}\ recNext)\ \mathbf{AND}\ [\mathbf{NOT}(lastIter)]$$

That process of how the handshake mechanism is implemented is illustrated bellow in Figure 3.26.

As for the AXI4 Lite communication, an instantiated IP block is provided by Xilinx Vivado suite which implements the logic architecture needed in order to support the AXI4-Lite protocol. What we did, was simply adding the desired user logic so as to read from the specified registers and export the data to the top level architecture of our Clustering component.

Clustering kernel

At this point, an analytical low-level description of each individual component of the Binning kernel is presented.

upperThresLoc

As we already pointed out, this block performs the initial thresholding and identifies the starting pixels while receiving the incoming binned pixels. Two variables are used which basically are mapped to the equivalent virtual coordinates of the sliding window.

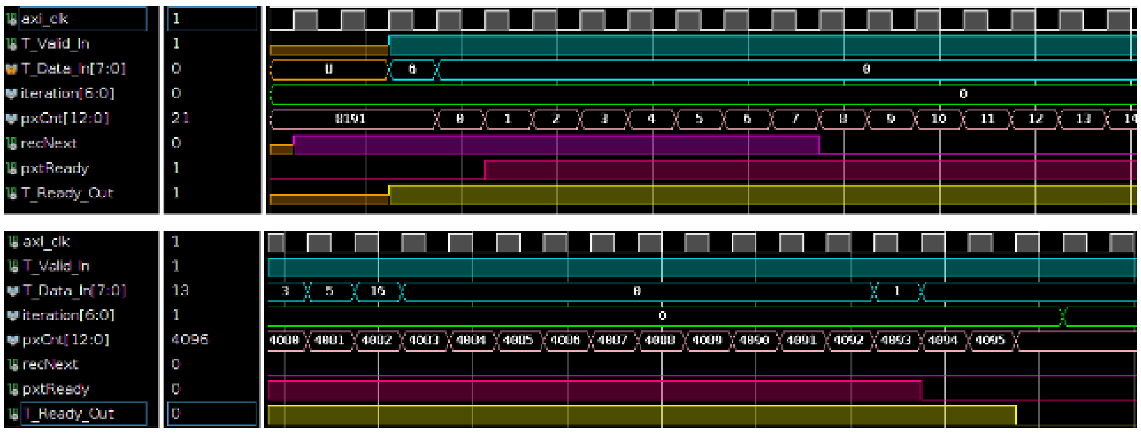


Figure 3.26: Behavioural simulation of driving TREADY signal for a 256×256 image and winHeight of 16 rows.

Since the first sliding window has been received, the row coordinate is re-initialized to the value of *maxInd* to match the overlap. Although it is not necessary for our project, a mechanism to stall the operation is developed so as to remain idle when a pause in transmission occurs, excluding the case of a complete window transmission, caused by the de-assertion of TREADY_IN signal.

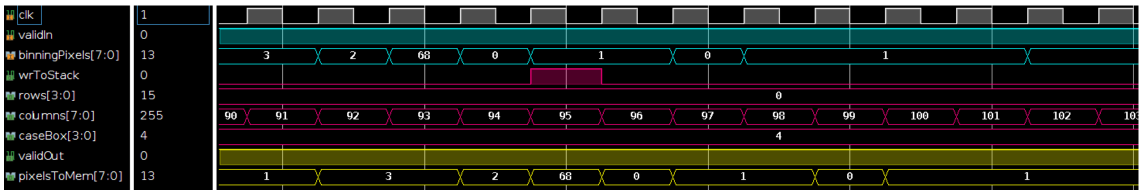


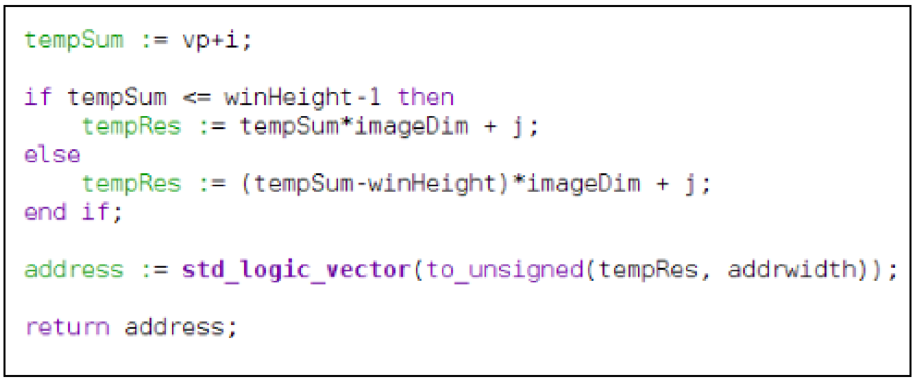
Figure 3.27: Behavioural simulation of upperThresLoc.

fsm4RAM

Regarding the functionality of this particular block, we could say that is partitioned into three basic operations which concern the generation of addresses for reading and writing from/to the memory units. The first one is about the addresses in which the incoming pixels are written in the RAM block. The second one is the most complex and refers to the generation of addresses where the desired pixels are stored in RAM. This process, needs to adapt to the growing region algorithm that was described in paragraph 3.2.1 and also needs to cover every possible extreme case. As we already mentioned before, these extreme cases arise when the central pixel of the 3×3 neighbourhood is located at the edges of the window plane. In the following chapter, an extensive description is presented of how these cases are formed so as to improve the performance of the system and achieve higher clock frequency. Briefly, these extreme cases are divided as follows: every possible relevant position where the pixel was detected and happens to be in the first row of the window; respectively for the last window row as well as for its first and last column. The last alternative case only concerns the starting pixels and is depicted in figure 3.22. The third operation that is performed in this block, is about generating the reading addresses so that the clustered pixels to be exported.

The fsm4RAM receives a pulsing signal that indicates whether to start the reading or writing process. It also receives the current coordinates of the central pixel and its relative position in the 3×3 NoS so as to be able to compute the desired addresses. As for the cluster-centric reading, it is provided with the calculated cluster dimension and with the up-left pixel's coordinates. Supposing that the cluster dimension is equal to K, the addresses of the corresponding K×K region are produced.

In order to calculate each address, it requires knowledge of the physical base address of the RAM block. Thus, it has an input port to receive the window row where the vp points. We developed a function which given the i and j coordinates as well as the vp, computes the physical address in the RAM. The following picture shows the exact VHDL code used to implement this function.



```
tempSum := vp+i;

if tempSum <= winHeight-1 then
    tempRes := tempSum*imageDim + j;
else
    tempRes := (tempSum-winHeight)*imageDim + j;
end if;

address := std_logic_vector(to_unsigned(tempRes, addrwidth));

return address;
```

Figure 3.28: VHDL code of the function that computes the addresses.

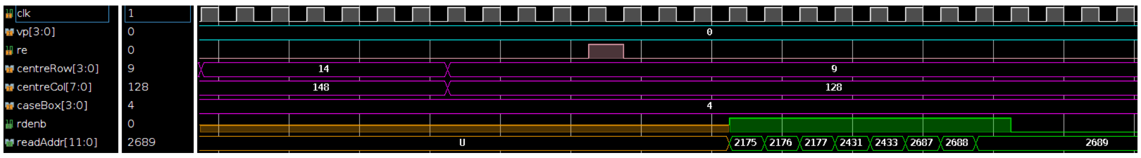


Figure 3.29: Behavioural simulation of fsm4ram.

neighChecker

This block is responsible for the growing thresholding operation. It receives the values of the adjacent pixels within the 3×3 region and it compares them to the growing threshold. If their value is greater than or equal to the threshold, it gives as outputs the corresponding coordinates and the relative position of the detected pixel in the grid, which is referred as *caseBox*. However, there are a few more conditions which need to be met so that a pixel to be considered as part of the cluster. We could split the pixel check into three individual processes. First of all, a pixel needs to be inside the RoI so as to be considered as part of the cluster. Secondly, it is obvious that value needs to be over the threshold so as to check its neighbourhood. Finally, as we are trying to improve the algorithm performance, an increased level of efficiency is required. Thus, the idea is to exclude unnecessary pixel searches which add considerable latency. This happens when the pixel to-be-checked is at the corners of the window. However, it might belong in the cluster so it should be marked in the Map as well.

Other than the aforementioned operations, the neighChecker block also keeps track of the positional movement while the searching within the RoI happens. As we already described, a set of variables is used so as to store the outer edges of the cluster that is currently detected. Whenever a new central pixel is checked, its coordinates are compared to these exact variables and they're updated if needed.

This component actually receives pixel values. Therefore, it is crucial that it has knowledge of the centre pixel's relative position on the image plane. In order to obtain this knowledge, it receives the i, j coordinates as well as the *caseBox* of the current central pixel in the 3×3 grid, so as to map the incoming pixel values to the actual pixel's position in the sliding window. So, the correlation must be determined between the addresses generated by the fsm4ram and the pixel values received by this block. This procedure is illustrated in the following Figure 3.31.

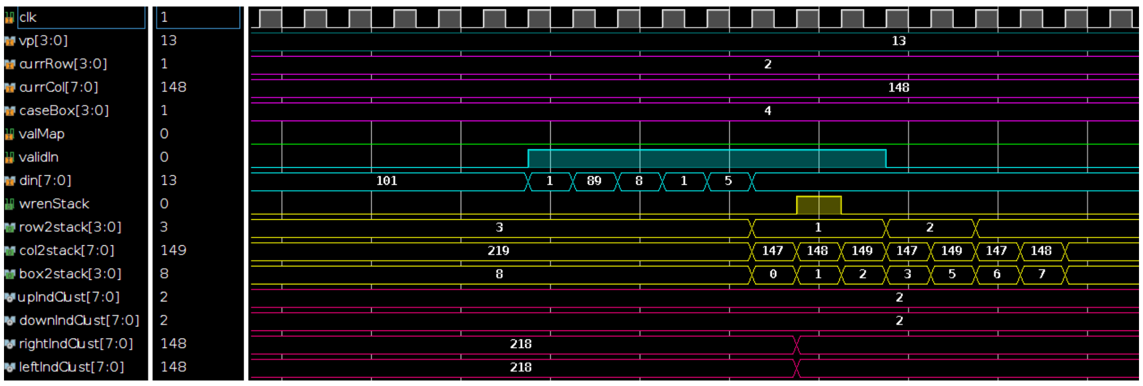


Figure 3.30: Behavioural simulation of neighChecker.

<u>RAM</u>

The RAM block consists of a basic logic as of its design implementation. However, it is worth mentioning how the addressing operation is controlled. During the first iteration, a full window is received so as to fill the memory. In this case, the virtual and the physical addresses are identical, as the first pixel of the image is stored to the first memory address, the second pixel to the second address and so on. The memory addresses of the RAM block are consecutive in the PL memory; however let us consider it as a 2D array where in each cell a pixel is stored. So, here the virtual pointer points to first row of the array meaning that the first pixel is located to address 0. Supposing that the overlap between the windows is of *maxInd=3*, the last three rows of the image are kept, and the next ($winHeight - maxInd$) rows are received to complete the sliding window. During second iteration, the first row of this current window is supposed to be the first one among the three rows kept from the previous window. Thus, the virtual pointer points to this exact

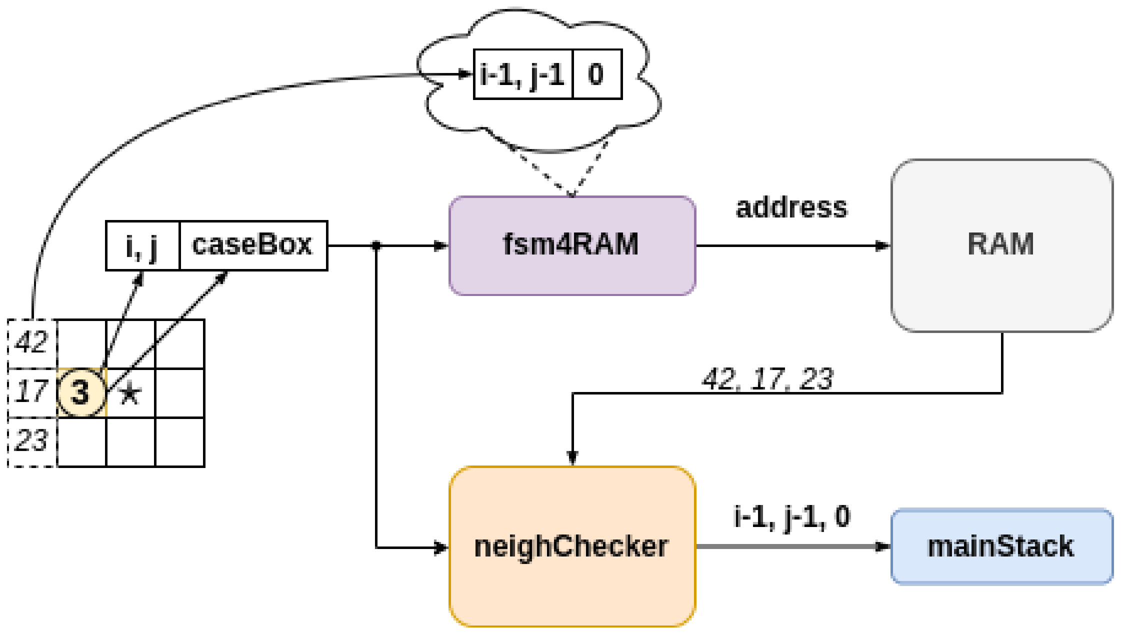


Figure 3.31: Illustration of the correlation between the operations performed by fsm4RAM and neighChecker blocks.

row which is stored in the $(winHeight - maxInd)$-th physical row of the RAM. All the above are summarised for each iteration as follows:

- In every iteration: VP = VP-maxInd. If maxind > VP, then VP = winHeight+VP-maxInd
- The physical row of the image is in the following range: iter×(winHeight-maxInd) ≤ i ≤ iter×(winHeight-maxInd)+winHeight-1

Map

The Map unit is driven by the exact same signals as for the RAM. Hence, reading and writing operation happen concurrently. We should mention that while the incoming binned pixels are being stored to the RAM memory, a writing operation occurs to the map as well, so as to clear the marked cells which refer to the previous sliding window.

mainStack & compStack

The mainStack and compStack follow the stack data structure logic. As we've already mentioned, they consist of three separate stack units of the same depth. However, they are able to store data of different widths according the generic variables that are defined by the project requirements. They support a single channel meaning that push and pop operations cannot happen at the same time. These operations are controlled by two input signals, one which defines what kind of operation occurs and one that generally enables it.

mainFSM

This last component named mainFSM, is responsible for handling the individual processes. Figure 3.25 presents the pipelined steps that are controlled by this FSM. First of all, we need to clarify that each iteration is divided into three main stages. These stages are defined by which component operates to mainStack. During the first stage, the binned pixels are being received and stored to the RAM memory. While this is happening, the *upperThresLoc* is performing the thresholding so it writes to the mainStack. When the storing process has been completed, the data from the compStack are copied to the mainStack until it becomes empty. Afterwards, the algorithm is executed meaning that neighChecker operates to mainStack by writing the coordinates of the detected pixel. So these components should be able to drive a single port. When designing in hardware, it is not trivial how to handle this. The way it is resolved, is by designing a multiplexer (MUX) which has as inputs the three individual signals and is controlled by a selecting signal which is driven by mainFSM. For each stack we have three different MUXs; one for the data to-be-stored and two for *enable* and *rd/wr* signals.

The mainFSM uses the components output signals to identify which processing step should be executed next. For example, every time a neighbourhood is checked, it reads the state of the mainStack's stack pointer which is compared to the initial state of the mainStack after popping the starting pixel. If initSP and currSP are equal, we proceed to the extraction stage, where as the name suggests, the cluster is being extracted to the output. In order to achieve synchronization between the individual operations, the mainFSM uses a counter named *cc*, so having knowledge of the latency introduced by each block it can determine which processing step will happen.

As it was described in the previous paragraphs, it is possible that a cluster of a single pixel is detected. This cluster is discarded as we suppose that it is invalid. What we do is driving the second and the third bit of the signal that holds the cluster dimension into an OR logic gate. If the dimension is equal to 1 (001 in binary), the output of the gate is 0. This signal is driven to an AND logic gate with the validOut signal which indicates that a valid output is exported from the Clustering kernel. This is a way to ground the validOut signal efficiently in case that an invalid 1×1 cluster is detected.

Finally, we will describe how it is identified whether the growing region algorithm of a pixel is executed in this current window or not. We introduced three separate signals for each condition that needs to be met so that the pixel will be checked. These three conditions are checked individually and are defined as follows:

- *pass1* is set if current pixel's row is less than (winHeight-maxInd)
- *pass2* is set if current pixel is not a starting pixel
- *pass3* is set if the last sliding window of the image is processed

These three signals are driven to an OR logic gate, which means that if none of the above conditions is met, the pixel should be checked within the next sliding window and therefore, it is stored to compStack.

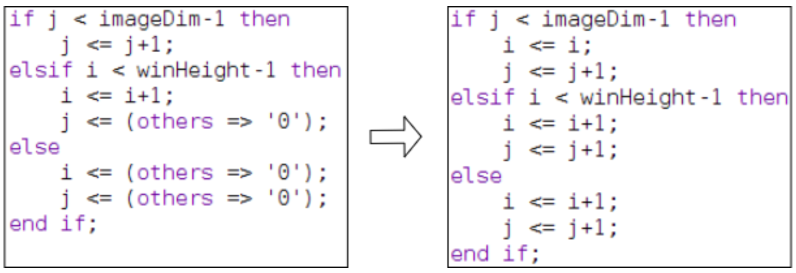
3.2.4 Performance-Wise Optimization

At this point let us present the main optimisation steps that we followed in order to improve the algorithms performance. In order to measure the improvement we used some timing constraints and we tried to achieve the highest possible post-implementation clock frequency. The description mostly refers to the steps that increased the clock frequency significantly.

First of all, a widely used technique is to store the incoming data to registers instead of using the inputs directly. Most commonly, the output data are stored in registers before being exported. In both methods, we manage to lower the critical path, which refers to the path with the maximum delay between input and output. In our design where, data are forwarded from one component to another, and in some cases to multiple blocks, we limited the critical path of the entire design along the paths of the internal blocks. As it will be discussed in the following paragraphs, the critical path of the Clustering kernel concerns the neighChecker block, which is the most complex one.

upperThresLoc

A technique that was adopted by almost every block which refers to pixel coordinates is that we handle signal overflow. After performing a lot of experimentation, we came to conclusion that it is an exceptionally slow operation to reset a signal, especially for signals of many bits. In terms of hardware, resetting a signal to a specific value such as 0 or 1 implies connecting it to ground or Vdd (supply voltage). So, instead of setting a variable to zero, we take advantage of the overflow that happens when a signal that holds the maximum possible value is increased. Additionally, an if-else condition in VHDL is converted to circuit logic for hardware that is probably implemented by a multiplexer. By that proposed transformation, it is obvious that we simplify the circuit's logic. As shown in fig. 3.32, for each of the three conditions, the logic for j signal is identical. As for the i signal, we can see that on the left code it follows a different logic for each condition while on the right code these cases are decreased.



```
if j < imageDim-1 then
     j <= j+1;
elsif i < winHeight-1 then
     i <= i+1;
     j <= (others => '0');
else
     i <= (others => '0');
     j <= (others => '0');
end if;
```

```
if j < imageDim-1 then
     i <= i;
     j <= j+1;
elsif i < winHeight-1 then
     i <= i+1;
     j <= j+1;
else
     i <= i+1;
     j <= j+1;
end if;
```

Figure 3.32: VHDL code transformation regarding signal overflow.

fsm4RAM

One major improvement to this block's performance was precomputing some of the variables used. More specifically, in this unit the central pixel's coordinates are received and the addresses of the adjacent pixels in the 3×3 region are computed. According to the *caseBox*, a different address needs to be calculated. So, instead of performing these calculations in a series of nested if-cases, the addresses of every pixel in the NoS are calculated all at once at the beginning of the process. Thus, the logic blocks used to implement these calculations in hardware are excluded from the path resulting to a significant reduction of the critical path latency.

Additionally, the overflow handling technique is also used in this block for the generation of the writing addresses. The coordinates are controlled in a similar way as described previously. However, they are used indirectly by the developed function so as to compute the corresponding address.

As it is already discussed there are some extreme cases regarding the position of the central pixel that differentiate the reading operation from the RAM memory. These cases occur when the central pixel is located at the edges of the sliding window meaning that it borders less than 8 pixels so a 3×3 neighbourhood cannot be formed. In our case, the code was written in a way that it contained a series of nested if-else statements. This high-level-language (HLL) programming-based way of code writing is highly inefficient for hardware designs. We suppose that each if-statement represents an individual task performed by an FSM. So what we actually do is group together the similar tasks that are performed under the same condition. This technique can be interpreted as a way of task partitioning and it results to a derived parallelism. This process is explained more clearly in the illustrated Figure 3.36 bellow. For example, those cases which involve the possibility of a centre pixel to be located in the first row of the window, are grouped into a separate task which operates constantly regardless of whether the condition from which they arose is satisfied. Therefore, the code is divided into six separate units that operate autonomously and individually and each of them generates its corresponding results. In the end, there is a multiplexer that simply identifies which condition was met in order choose the output data correctly.

neighChecker

This block is actually the block that performs the most demanding operations and therefore it consists of the most complex logic. So it is clear that it vital to improve its performance in order to increase the operation frequency of the whole system.

As described earlier, there is a strong correlation between the operations performed by the fsm4RAM block and the neighChecker. Therefore, the same logic described above for the fsm4RAM was also adopted by this unit so that six individual blocks to be formed to increase parallelism. Yet, the idea can be expanded to the lower levels of the design. For every incoming pixel value, three check operations happen so that it can be considered as part of a cluster. These three processing steps are described as follows:

- Comparison to the growing threshold and rechecking avoidance
- Check so that the pixel to-be checked is within the RoI
- Coordinates extraction and cluster dimension update

So instead of performing these three interdependent functional levels, we split them into

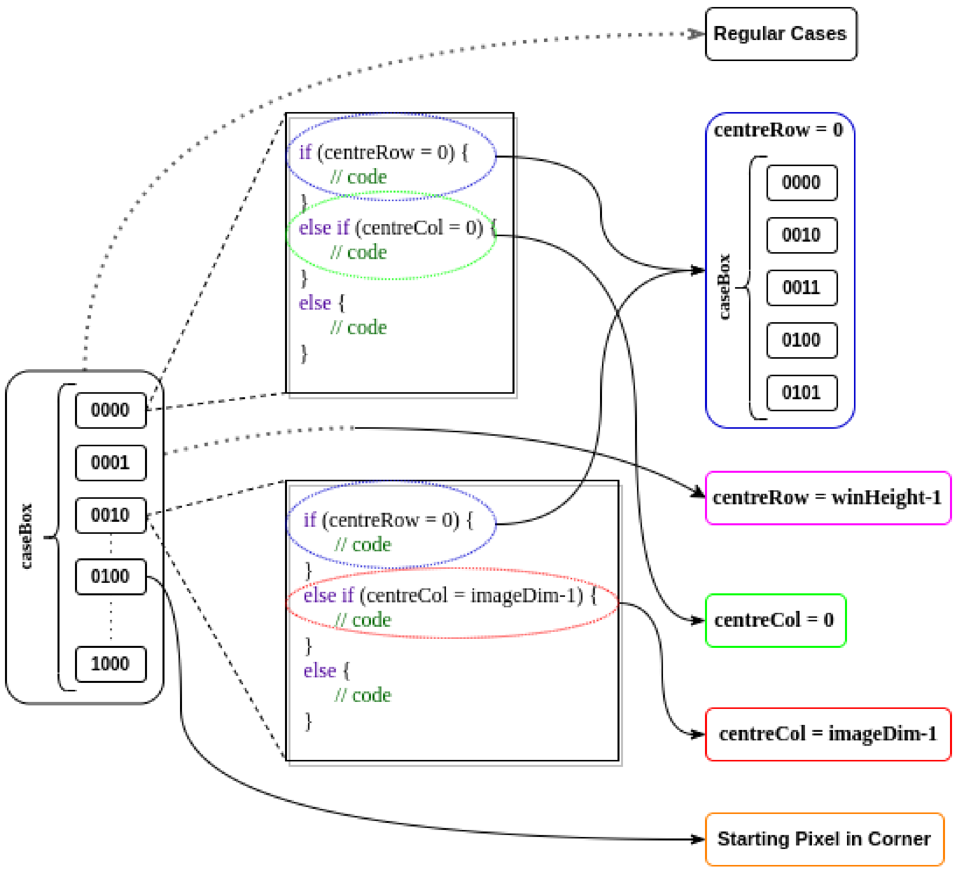


Figure 3.33: Visualization of task partitioning.

three individual and independent processes. Each of the six so-formed sub-components performs these separate processing stages. It needs to be mentioned that the third process, which is responsible for the coordinates computation and the update of the cluster outer edges, acts as if both of the two other conditions are satisfied. Each of these two sub-processes, drive a well-defined signal which holds the post-checking result. If either condition is not met, the output signal, which indicates the validity of the output data, is grounded. The challenging task was how to handle the extreme cases where a pixel is located to the corners of window. If the pixel belongs to the cluster region, there is no need to check the neighbouring pixels, as it is certain that they have been already checked. However, an update to the cluster dimensions should happen if needed. Again, there are two blocks; one for the regular and one for the extreme cases, where each of them operates as if their own condition is satisfied. So, it is required that they communicate with each other in order to keep the cluster dimensions updated.

Map

The last idea that is proposed in this thesis in order to improve the algorithm performance is the implementation of a Map. As it's been already described, this map keeps track of the checked pixels in the sliding window so as to avoid unnecessary actions. More specifically, there are two levels of rechecking avoidance control. The first one, is operated

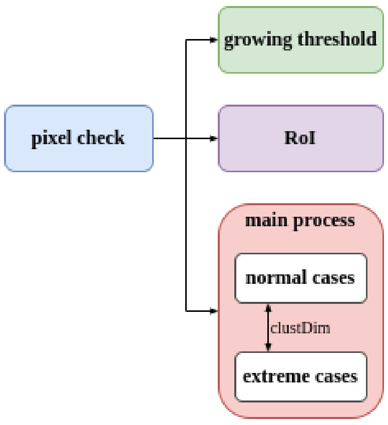


Figure 3.34: Illustration of the three individual sub-processes performed by neighChecker component.

by the mainFSM while a pair of pixel coordinates is popped out of the mainStack. If the corresponding pixel's value on the Map is equal to one, the pixel has been already checked so it is rejected and the next pair of coordinates is received. The second level of rechecking control is performed by the neighChecker block as described in the previous paragraph. The block basically receives the input pixel's value as well as its corresponding value on the Map and acts proportionally.

After proceeding to the aforementioned performance-wised optimisation steps, the results regarding the achieved clock period are presented to the following Table 3.1.

Table 3.1: Table of the achieved clock period per component after optimisation

Component	**Clock Period Before Optimisation**	**Clock Period After Optimisation**	**Percentage Improvement**
upperThresLoc	3.7 ns	2.4 ns	35.1%
fsm4ram	5 ns	3.5 ns	30%
clusterCal	4.3 ns	3.2 ns	25.6%
neighChecker	9.4 ns	4.4 ns	53.2%
Clustering kernel	11.2 ns	4.3 ns	61.6%

3.2.5 System Integration

In this subsection, we present the integrated system which contains both the Averaging 2D Binning and the Clustering component. The A2DB component as well as the Clustering component are hosted on the PL. More specifically, the Clustering component receives data after the binning process and it follows the A2DB component as it is illustrated in the block design in Fig. 3.35.

As we can see, it is almost the same block design as the one depicted in Fig. 3.17, with the difference that the Averaging 2D Binning block interpolates. In that case, there are

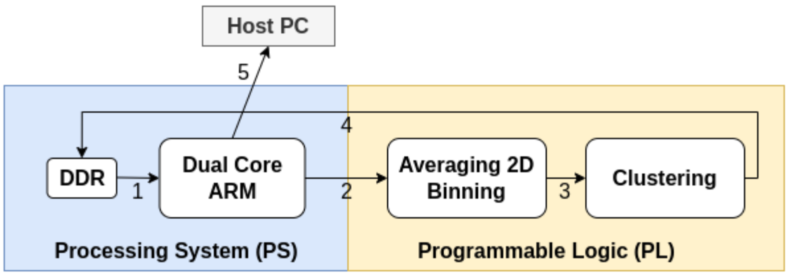


Figure 3.35: Pipeline of the processing stages of the HW/SW Co-design.

two extra stages in the processing pipeline which apparently refer to the binning process. The binned pixels are directly forwarded to the Clustering component and transmitted to the AXI_DMA so as to be stored to the DDR. When a full window has been received and stored, we need to send an identification signal to the A2DB unit so as to pause the transmission and it will subsequently inform the AXI_DMA to stop its individual transmission of the original image. The fifth step actually is refers to the reading operation from the DDR memory handled by the processor, where we are able to extract the results to our Host PC. One more significant difference between this design and the one pictured in fig. 3.17, is that the data received from A2DB aren't consecutive. However, this does not affect the operation of the system at all. The block design of the integrated system as it is configured in Vivado Design Suite is illustrated in the following figure 3.36.

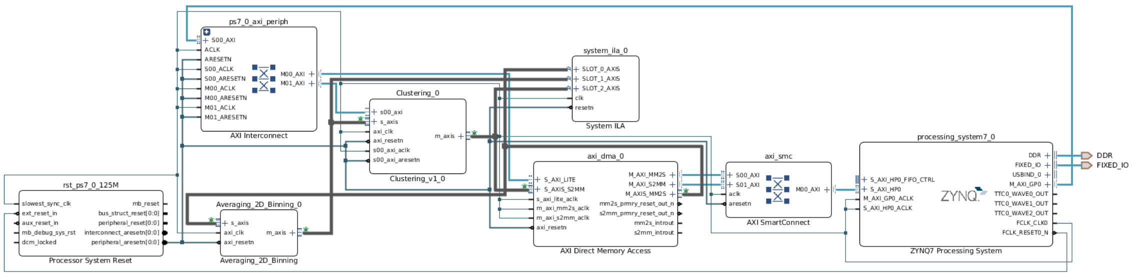


Figure 3.36: Block design of the HW/SW Co-design.

Chapter 4

Experimental Evaluation

In this chapter, a complete overview of the experimental setup is presented regarding the tools and the datasets being used for our experimental and evaluation procedures. Additionally, a full report of the interfaces results and an in-depth performance analysis regarding the resource utilization of the FPGA, the operation frequency and the power consumption is discussed. Finally, an estimation is of the entire star tracker system performance is made so as to demonstrate the capabilities of our proof-of-concept architecture.

4.1 Experimental Setup

Firstly, the experimental setup and the specifications of the board, on which the experiments were conducted, is presented. Our target evaluation System-on-Chip FPGA, as mentioned, is Xilinx Zynq-7020 SoC. An extensive description of the SoC architectures was already discussed in Chapter 2.5. The CAD tool utilised for HW design was Xilinx Vivado Design Suite v2019.1 for Ubuntu Linux Operating System. In addition, we used the supplementary tool Xilinx Software Development Kit (XSDK) to create embedded applications running on the dual ARM Cortex processor of the Zynq SoC so as to set up the PS-PL communication. Finally, the images used for the simulations were processed in MATLAB 2018a, in order to be converted into the desired format. More specifically, text files were created containing the pixel values of the 2048×2048 greyscale images. These files were simply imported to the DDR memory of the SoC FPGA implying a simplified test case where the images are received from the image sensor.

For the scope of this thesis, the conducted experiments focus on the preprocessing stages of a star tracker emphasising on the accelerating clustering algorithm. As already mentioned, greyscale images with resolution of 2048×2048 were used as inputs, although lower sized images were tested as well due to the configurability of our system. An unbiased input dataset was created from a repository of NASA that contains the full record of the Cassini spacecraft's raw images taken from Feb. 20, 2004 to Cassini's end of mission on Sept. 15, 2017 [14]. Figure 4.2 illsutrates exactly how the board is connected to the development host PC so as to establish the USB connections for the UART and JTAG programming.

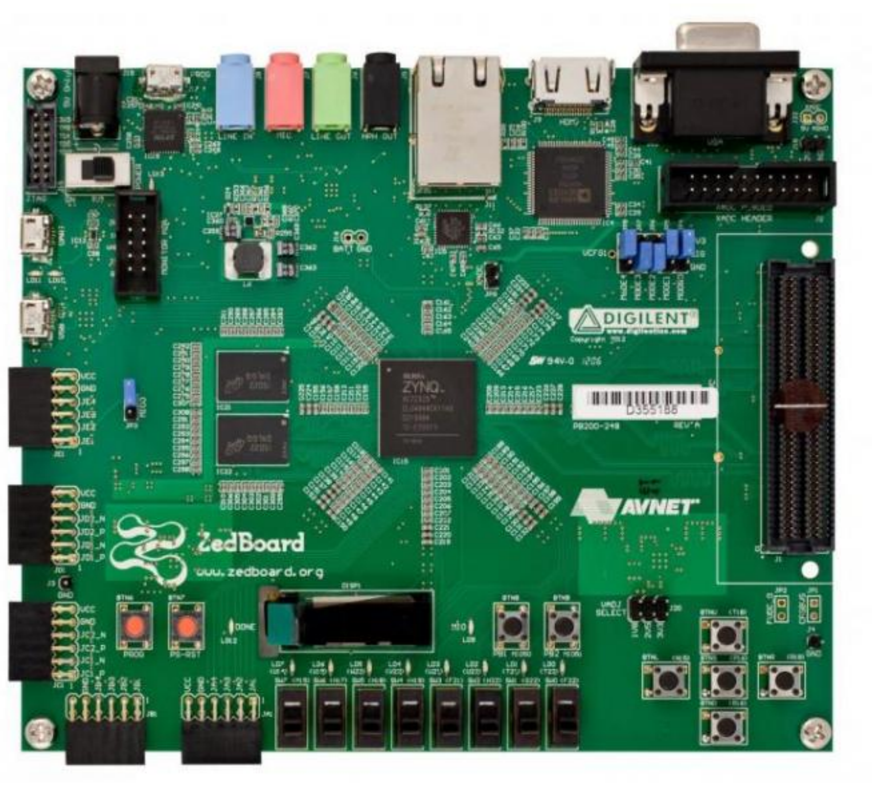


Figure 4.1: Zedboard Development Board [13].

4.2 Results and Analysis on Zynq FPGA

In the following section the results of the implemented components are presented separately as well as the complete integrated system architecture. Each of them is analyzed and evaluated regarding performance, power consumption and resource utilization.

Initially, we examined the correct functionality of our designs by performing behavioural simulations using specified testbenches. Consequently, experimental tests were performed repeatedly for several image sizes and different parameters. However, for the objectives of this thesis a single case scenario is discussed. Additionally, a software-oriented implementation running on the PS of the SoC, which is adapted to the the HW designs, is also developed and compared to our HW/SW embedded system regarding performance.

Averaging 2D Binning

We successfully performed data transmission without errors for 12-bit 2048×2048 frames at 125 MHz. We should mention that AXI4-Stream interface supports data transmissions at widths of powers of 2. Thus, each pixel is represented by a 16-bit value and internally in the PL block it is handled as a 12-bit width number respectively. Additionally, we managed to achieve the highest possible clock frequency at 250 MHz, although within the integrated block design these timing requirements weren't met due to remark-

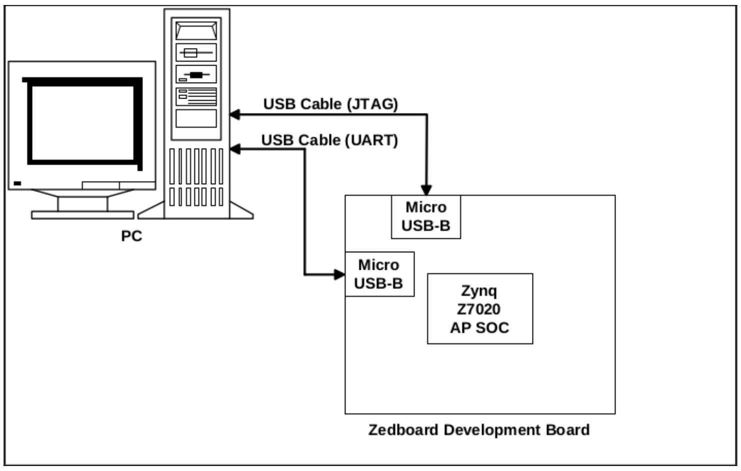
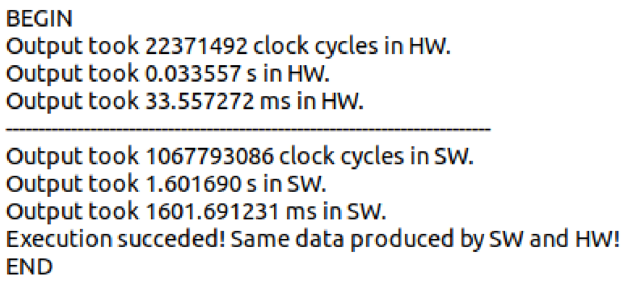


Figure 4.2: Board Connection Setup [13].

ably slower DMA engine of the Zynq SoC. In that case, we achieved a total processing time of approximately 33.56 ms on average containing the latency of the PS-PL loopback. The total execution time as derived from the behavioural simulation in Vivado was extremely accurate as it produced a mean error of 0.0027 ms. As we can see, there is a major difference between the measured processing timings as the proposed architecture is significantly faster than the ARM processor. The following Table 4.1 presents the gained speedup that we achieved with SoC FPGA acceleration compared to a single thread SW implementation on ARM Cortex processor.



```
BEGIN
Output took 22371492 clock cycles in HW.
Output took 0.033557 s in HW.
Output took 33.557272 ms in HW.
------------------------------------------------------------------
Output took 1067793086 clock cycles in SW.
Output took 1.601690 s in SW.
Output took 1601.691231 ms in SW.
Execution succeded! Same data produced by SW and HW!
END
```

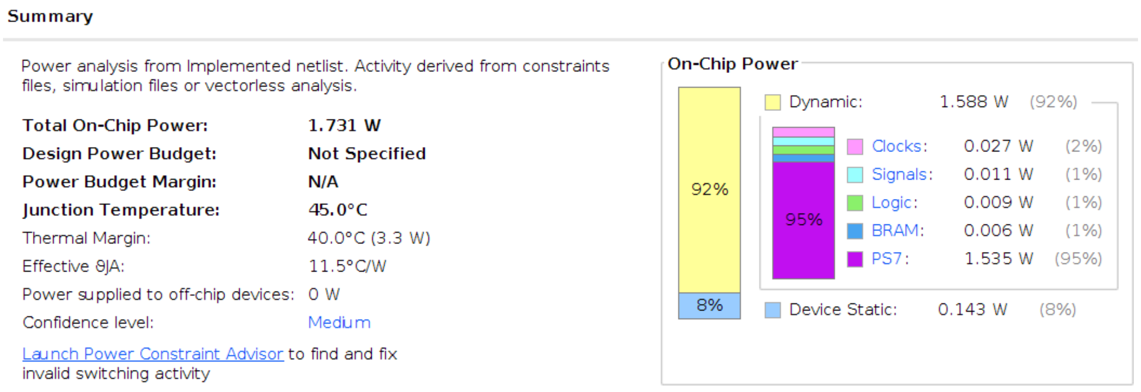
Figure 4.3: Post execution result as shown in SDK terminal.

Table 4.1: Comparison of mean execution time between SoC FPGA and ARM CPU for Averaging Binning operation

Platform	ARM CPU	SoC FPGA	Speedup	SoC FPGA (Max Freq)	Speedup (Max Freq)
Execution Time	1607.46 ms	33.58 ms	47.9x	16.78 ms	95.8x

In space applications, power consumption is among the most important factors when considering a star tracker. Thus, in the Fig 4.4 we present the power consumption of the Averaging 2D Binning block as it is estimated after implementation in Vivado. Finally, an analytical overview of the resource utilisation of the FPGA is presented on the following

Table 4.2.


Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.731 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**45.0°C**
Thermal Margin:	40.0°C (3.3 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.588 W	(92%)	
	Clocks:	0.027 W	(2%)
	Signals:	0.011 W	(1%)
	Logic:	0.009 W	(1%)
	BRAM:	0.006 W	(1%)
	PS7:	1.535 W	(95%)
Device Static:	0.143 W	(8%)	

Figure 4.4: Estimated Power Consumption of Binning component.

Table 4.2: Resource Utilisation of FPGA for Binning implemented design.

Resources	**Utilisation**	**Available**	**Utilisation (%)**
LUT	5739	53200	10.79
LUTRAM	803	17400	4.61
FF	8317	106400	7.82
BRAM	9	140	6.43

Clustering

This component is the main block of the preprocessing operation and and therefore plays a prominent role in the performance of the integrated system design. Nevertheless, let us present an extensive analysis of it before examining the whole system. Again, the DMA engine is a limiting factor to the achieved operation frequency. A cross-clock domain technique was proposed so as to resolve this issue, however due to the definition of the required DMA parameters we weren't able to drop the clock period bellow 7 ns ($\sim$ 142MHz). For correspondence purposes, we proceeded with a frequency of 125 MHz. However, after the proposed optimisations which have been already discussed in Chapter 3.2.4, the achieved clock frequency is at 232 MHz out of 250 MHz maximum frequency supported by Zync platform. Figures 4.5 and 4.6 depict an example of an output cluster that was detected after the algorithm's execution.

Again, several tests were conducted of different images and bellow an example is depicted of the output result in SDK terminal. The following Table shows the gained speedup using the SoC FPGA accelerating platform. We should mention that the PS of the platform can generate certain clock frequencies and therefore it cannot generate the required clock period of 4.3 ns. Hence, the estimations refer to the closest possible frequency which is 214 MHz.

As for the power consumption of this particular component, the estimated power as reported by Vivado is listed bellow. Finally, the Table 4.4 below shows the resource

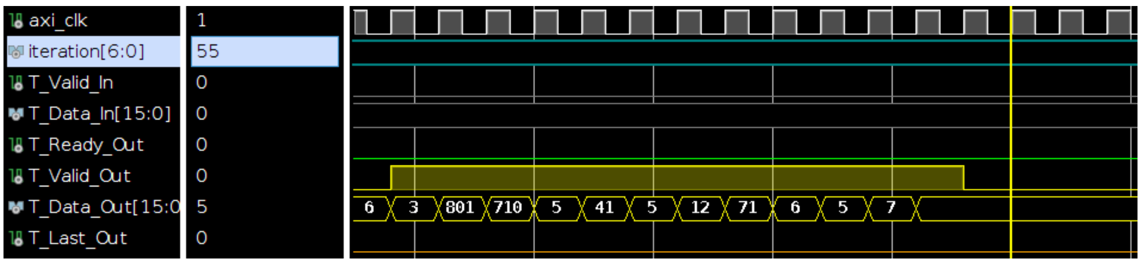
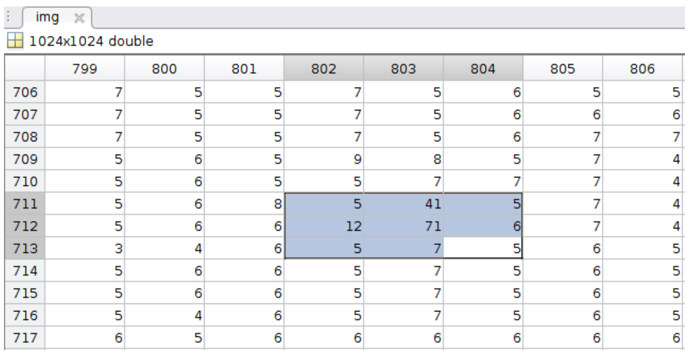


Figure 4.5: Behavioural simulation of an output detected cluster.



img

1024x1024 double

	799	800	801	802	803	804	805	806
706	7	5	5	7	5	6	5	5
707	7	5	5	7	5	6	6	6
708	7	5	5	7	5	6	7	7
709	5	6	5	9	8	5	7	4
710	5	6	5	5	7	7	7	4
711	5	6	8	5	41	5	7	4
712	5	6	6	12	71	6	7	4
713	3	4	6	5	7	5	6	5
714	5	6	6	5	7	5	6	5
715	5	6	6	5	7	5	6	5
716	5	4	6	5	7	5	6	5
717	6	5	6	6	6	6	6	6

Figure 4.6: Visualisation of a detected cluster on MATLAB.

utilisation of the FPGA for the implementation of the Clustering HW/SW co-design.

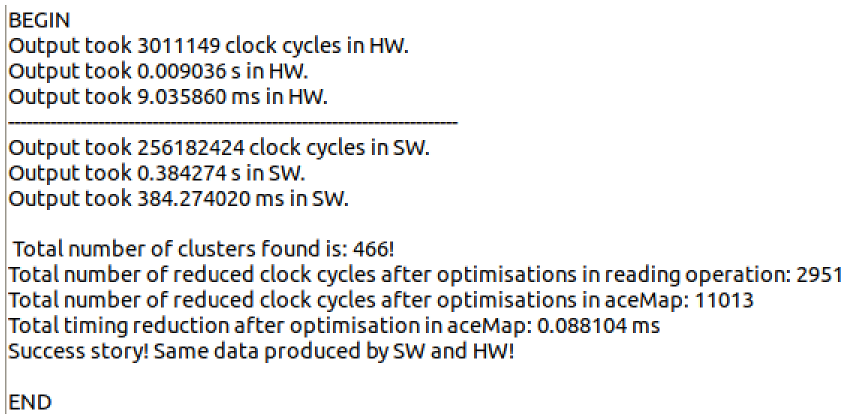
Integrated System

At this point, having examined how the two main components behave individually, let us present the full overview of the proposed system's architecture. The achieved clock frequency is at 125 MHz as well, although it is estimated that the whole system can operate at a frequency which is defined as the minimum frequency among the compared operation frequencies of the individual designed blocks. An example of a behavioural simulation is figured at the following picture 4.9.

In order to check the functional validity of the integrated system, several tests were performed on the SoC FPGA accelerating platform, where a software-based implemented algorithm was also executed on the ARM Cortex A9 processor. The produced results were compared autonomously leading to the following output extracted to the SDK terminal.

As we can see, the PS-PL cooperating mechanism totally outperforms the PS implemented operation. The Table 4.6 summarises in detail the results regarding the performance of the integrated system's proposed architecture.

At this point, it is worth commenting on the considerable timing overhead added by the binning operation and whether it effectively contributes in the entire preprocessing operation. As we know, large images are being processed for the cluster detection. The SoC FPGA has limited resources so the image processing occurs in sliding windows. Additionally, the size of each window is dependant to the original image's size so larger images might lead to smaller window transmissions adding considerable timing overhead as well as increased resource utilisation. Considering all the above, we can assume that averaging



```
BEGIN
Output took 3011149 clock cycles in HW.
Output took 0.009036 s in HW.
Output took 9.035860 ms in HW.
------------------------------------------------------------------
Output took 256182424 clock cycles in SW.
Output took 0.384274 s in SW.
Output took 384.274020 ms in SW.

 Total number of clusters found is: 466!
Total number of reduced clock cycles after optimisations in reading operation: 2951
Total number of reduced clock cycles after optimisations in aceMap: 11013
Total timing reduction after optimisation in aceMap: 0.088104 ms
Success story! Same data produced by SW and HW!

END
```

Figure 4.7: Post execution results as shown in SDK terminal after clustering operation.

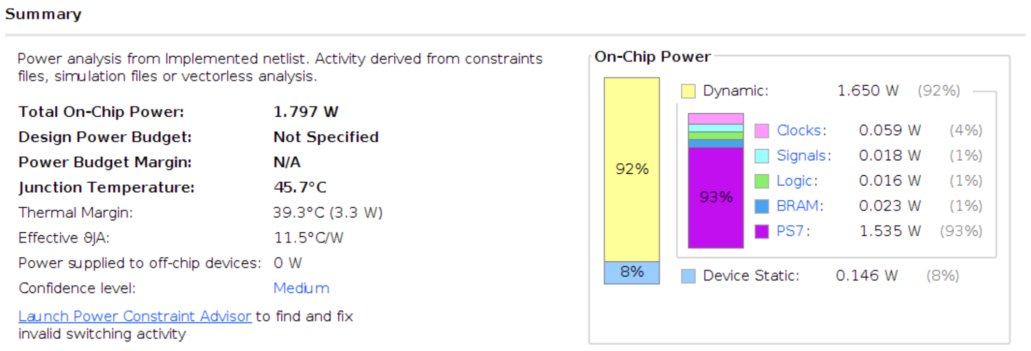
Table 4.3: Comparison of mean execution time between SoC FPGA and ARM CPU for Clustering operation

Platform	**ARM CPU**	**SoC FPGA**	**Speedup**	**SoC FPGA (Max Freq)**	**Speedup (Max Freq)**
Execution Time	383.14 ms	8.9 ms	43.1x	5.11 ms	75.4x

binning is not an unnecessary processing step for the entire clustering extraction operation. Either way, the processing time of approximately 34 ms per image frame, which can be decreased up to 19 ms, successfully leads to the detection of hundreds of clusters relatively fast. As of the power consumption, an extensive analysis is illustrated in Figures 4.11 and 4.12. As we can observe, the PS is responsible for the 92% of the total power consumption and among the other units of the integrated system, the clustering component is the most demanding due to the computationally intensive logic operations performed.

Finally, the following Table 4.6 presents a post-implementation summary of the resource utilisation of the FPGA platform. In addition, Figure 4.13 illustrates an extensive report of the FPGA resource usage at an hierarchical form.

In conclusion, our proposed design architecture utilizes <17% of the total chip's resources for every FPGA primitive to support the proprocessing operation of 4 MPs 12-bit image frames. Hence, there is space for additional HDL components of the algorithmic pipelined steps of the star tracker. Ultimately, we developed an accelerating algorithm for cluster extraction on the Zynq-7020 SoC FPGA with estimated on-chip power consumption bellow 2 Watt.



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.797 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**45.7°C**
Thermal Margin:	39.3°C (3.3 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.650 W	(92%)	92%
Clocks:	0.059 W	(4%)	
Signals:	0.018 W	(1%)	
Logic:	0.016 W	(1%)	
BRAM:	0.023 W	(1%)	
PS7:	1.535 W	(93%)	93%
Device Static:	0.146 W	(8%)	8%

Figure 4.8: Estimated Power Consumption of Clustering component.

Table 4.4: Resource Utilisation of FPGA for Clustering implemented design.

Resources	Utilisation	Available	Utilisation (%)
LUT	9254	53200	17.39
LUTRAM	987	17400	5.67
FF	12258	106400	11.52
BRAM	20	140	14.29

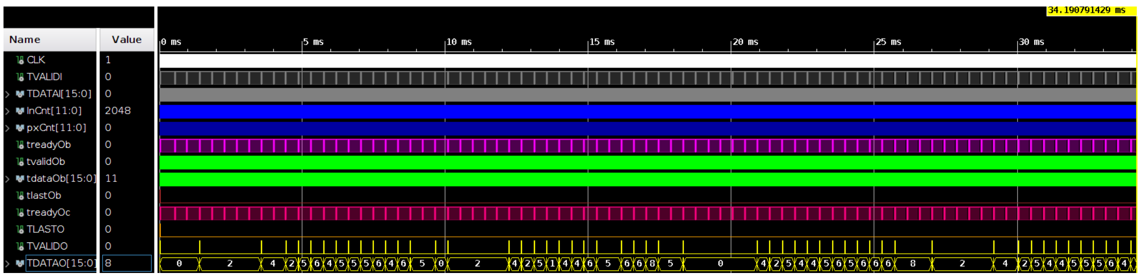
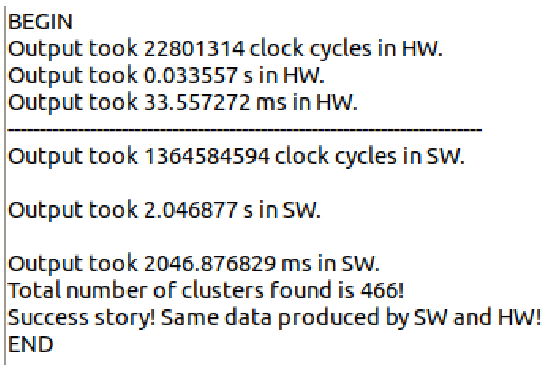


Figure 4.9: Behavioural simulation of the Binning-Clustering co-design.

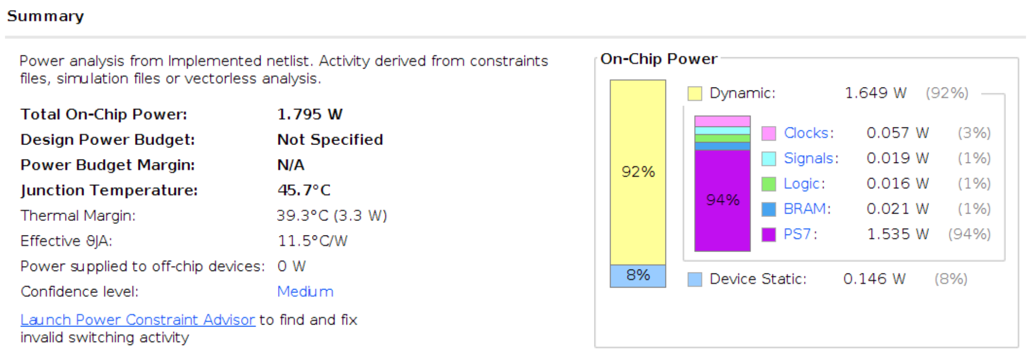


```
BEGIN
Output took 22801314 clock cycles in HW.
Output took 0.033557 s in HW.
Output took 33.557272 ms in HW.
----------------------------------------------------------------------------
Output took 1364584594 clock cycles in SW.

Output took 2.046877 s in SW.

Output took 2046.876829 ms in SW.
Total number of clusters found is 466!
Success story! Same data produced by SW and HW!
END
```

Figure 4.10: Post execution results as shown in SDK terminal after Preprocessing operation.

Table 4.5: Comparison of mean execution time between SoC FPGA and ARM CPU for the Integrated HW/SW Co-design

Platform	**ARM CPU**	**SoC FPGA**	**Speedup**	**SoC FPGA (Max Freq)**	**Speedup (Max Freq)**
Execution Time	2050.32 ms	34.06 ms	60.2x	18.94 ms	108.2x



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

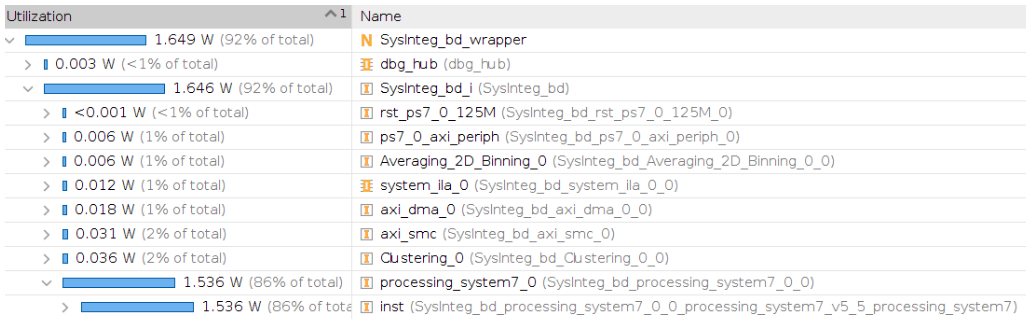
Total On-Chip Power:	**1.795 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**45.7°C**
Thermal Margin:	39.3°C (3.3 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.649 W	(92%)
Clocks:	0.057 W	(3%)
Signals:	0.019 W	(1%)
Logic:	0.016 W	(1%)
BRAM:	0.021 W	(1%)
PS7:	1.535 W	(94%)
Device Static:	0.146 W	(8%)

Figure 4.11: Estimated Power Consumption of the Integrated System.

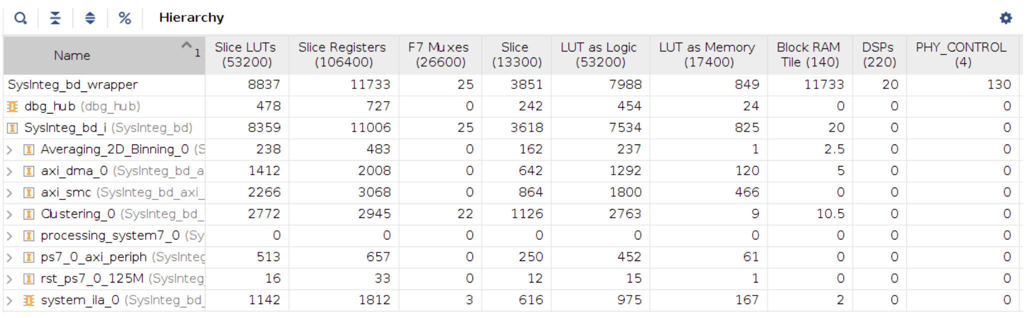


Utilization	Name
1.649 W (92% of total)	SysInteg_bd_wrapper
0.003 W (<1% of total)	dbg_hub (dbg_hub)
1.646 W (92% of total)	SysInteg_bd_i (SysInteg_bd)
<0.001 W (<1% of total)	rst_ps7_0_125M (SysInteg_bd_rst_ps7_0_125M_0)
0.006 W (1% of total)	ps7_0_axi_periph (SysInteg_bd_ps7_0_axi_periph_0)
0.006 W (1% of total)	Averaging_2D_Binning_0 (SysInteg_bd_Averaging_2D_Binning_0_0)
0.012 W (1% of total)	system_ila_0 (SysInteg_bd_system_ila_0_0)
0.018 W (1% of total)	axi_dma_0 (SysInteg_bd_axi_dma_0_0)
0.031 W (2% of total)	axi_smc (SysInteg_bd_axi_smc_0)
0.036 W (2% of total)	Clustering_0 (SysInteg_bd_Clustering_0_0)
1.536 W (86% of total)	processing_system7_0 (SysInteg_bd_processing_system7_0_0)
1.536 W (86% of tota	inst (SysInteg_bd_processing_system7_0_0_processing_system7_v5_5_processing_system7)

Figure 4.12: Analytic overview of the consumed power for each individual block.

Table 4.6: Resource Utilisation of FPGA for the Integrated System.

Resources	**Utilisation**	**Available**	**Utilisation (%)**
LUT	8837	53200	16.61
LUTRAM	849	17400	4.88
FF	11733	106400	11.03
BRAM	20	140	14.29



Hierarchy

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	DSPs (220)	PHY_CONTROL (4)
SysInteg_bd_wrapper	8837	11733	25	3851	7988	849	11733	20	130
dbg_hub (dbg_hub)	478	727	0	242	454	24	0	0	0
SysInteg_bd_i (SysInteg_bd)	8359	11006	25	3618	7534	825	20	0	0
Averaging_2D_Binning_0 (S	238	483	0	162	237	1	2.5	0	0
axi_dma_0 (SysInteg_bd_a	1412	2008	0	642	1292	120	5	0	0
axi_smc (SysInteg_bd_axi_	2266	3068	0	864	1800	466	0	0	0
Clustering_0 (SysInteg_bd_	2772	2945	22	1126	2763	9	10.5	0	0
processing_system7_0 (Sy	0	0	0	0	0	0	0	0	0
ps7_0_axi_periph (SysInteg	513	657	0	250	452	61	0	0	0
rst_ps7_0_125M (SysInteg	16	33	0	12	15	1	0	0	0
system_ila_0 (SysInteg_bd_	1142	1812	3	616	975	167	2	0	0

Figure 4.13: Hierarchical presentation of the FPGA resource utilisation.

Chapter 5

Conclusion and Future Work

Nowadays, space applications have become extremely demanding and complex scientific requirements need to be met. Taking these into consideration, we understand that it is of vital importance to have a reliable attitude determination system which combines high accuracy and real-time performance. For the time-being the identification speed is the bottleneck of satisfying real-time requirements on the premise of robustness. Recently, there is a growing interest in FPGA-based systems due to larger capabilities, lower costs of reprogrammable logic devices and parallel processing architecture. Star trackers are increasingly designed based on novel embedded HW platforms suchs as SoC FPGAs which consist of an FPGA and a CPU core.

The work presented in this thesis demonstrates an accelerating algorithm for preprocessing of a star tracker which is implemented in a HW/SW co-design on a SoC FPGA platform. More specifically, the preprocessing is devided into two operation steps which are averaging binning and clustering respectively. Both operations were designed, implemented and tested individually using various real images. Additionally, performance-wised optimisations for the cluster detection algorithm are also discussed and a detailed overview of their effects on the operation frequency is presented.

The main core of this current thesis focuses on the integration of the aforementioned sub-processes into a custom HW/SW co-design methodology so as optimise and map efficiently the combinational logic to a compact embedded platform. Furthermore, the algorithms were described in SW details and tested on the processing system of the SoC FPGA as well. A full comparison between the proposed architecture designs is also presented leading to draw important conclusions. The HW/SW co-design totally outperforms the ARM processor of the embedded device as it managed to decrease the processing time by approximately 60x, although it is estimated that the speedup can be increased up to 108x. As of the energy consumption, the total on-chip power is estimated to be about 1.8 W meaning that the proposed architecture is highly suitable for a wide range of cost-effective space applications. Last but not least, the proposed embedded system is relatively non-demanding regarding the utilisation of the FPGA's resources, leaving room for additional HDL implementations for the purpose of a complete processing pipeline of an

ADCS.

Our future work will focus on the improvement of the HW performance including further examination of the limiting parameters which affect the operation frequency, so that our proposed system reach the maximum estimated frequency and even exceed it if possible. Moreover, we will perform further experiments so as to extend the evaluation and verification of our techniques using real data adjusted to the requirements of Infinite Orbits. Finally, our major objective is to combine the proposed preprocessing design with the centroiding and matching algorithms to complete the star tracker pipeline and include that integrated system in a future space mission led by Infinite Orbits.

Bibliography

[1] T. Sun, F. Xing, X. Wang, Z. You, and D. Chu, "An accuracy measurement method for star trackers based on direct astronomic observation," in *Scientific Reports*, 2016.

[2] C. Liebe, "Star trackers for attitude determination," in *IEEE Aerospace and Electronic Systems Magazine*, 1995.

[3] Stack Exchange, "Camera perspective projection model."

[4] Photography Courses, "Look Out For Lens Distortion."

[5] ADCS For Beginners, "The Different Frames and the Keplerian Elements."

[6] P. Degond, A. Diez, and M. Na, "Bulk topological states in a new collective dynamics model," 2021, p. 28.

[7] Q. Hua-Ming, L. Hao, and W. Hai-Yong, "Design and verification of star-map simulation software based on ccd star tracker," in *8th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, 2015.

[8] L. Ortiz, L. Goncalves, and E. Cabrera, "A generic approach for error estimation of depth data from (stereo and rgb-d) 3d sensors," in *8th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, 2017.

[9] D. Soudris, "Reconfigurable architectures or FPGAs," in *NTUA-ECE, Class: Digital VLSI - Lec. 5.*

[10] Xilinx, "Zynq-7000 SoC)."

[11] ARM Developer, "AMBA AXI and ACE Protocol Specification."

[12] Xilinx, "Axi dma v7.1," in *Vivado Design Suite*, 2016.

[13] AVNET, "ZedBoard."

[14] NASA-Solar System Exploration, "Cassini Raw Images."

[15] Northern Sky Research, "Satellite EOL: Not One Size Fits All," 2018.

[16] Istituto Nazionale di Fisica Roma, "Attitude Determination and Attitude Control," 2015, p. 19.

[17] G. Lentaris, I. Stratakos, I. Stamoulias, D. Soudris, M. Lourakis, and X. Zabulis, "High-performance vision-based navigation on soc fpga for spacecraft proximity operations," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 4, 2020, pp. 1188–1202.

[18] K. Maragos, V. Leon, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, A. Pastor, D. M. Codinachs, and I. Conway, "Evaluation methodology and reconfiguration tests on the new european ng-medium fpga," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 127–134.

[19] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, R. Domingo, M. Verdugo, D. Gonzalez-Arjona, D. M. Codinachs, and I. Conway, "Systematic Evaluation of the European NG-LARGE FPGA & EDA Tools for On-Board Processing," in *2nd European Workshop on On-Board Data Processing (OBDP)*, 2021, pp. 1–8.

[20] C. Urbina-Ortega, G. Furano, G. Magistrati, K. Marinis, A. Menicucci, and D. Merodio-Codinachs, "Flash-based fpgas in space, design guidelines and trade-off for critical applications," in *14th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2013, pp. 1–8.

[21] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, D. M. Codinachs, and I. Conway, "Development and Testing on the European Space-Grade BRAVE FPGAs: Evaluation of NG-Large Using High-Performance DSP Benchmarks," *IEEE Access*, vol. 9, pp. 131 877–131 892, 2021.

[22] A. Pérez, A. Rodríguez, A. Otero, D. González-Arjona, A. Jiménez-Peralo, M. A. Verdugo, and E. De La Torre, "Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation," *IEEE Access*, vol. 8, pp. 59 891–59 905, 2020.

[23] V. Leon, G. Lentaris, D. Soudris, S. Vellas, and M. Bernou, "Towards Employing FPGA and ASIP Acceleration to Enable Onboard AI/ML in Space Applications," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–4.

[24] V. Leon, G. Lentaris, E. Petrongonas, D. Soudris, G. Furano, A. Tavoularis, and D. Moloney, "Improving Performance-Power-Programmability in Space Avionics with Edge Devices: VBN on Myriad2 SoC," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 3, pp. 1–23, 2021.

[25] F. C. Bruhn, N. Tsog, F. Kunkel, O. Flordal, and I. Troxel, "Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space," *CEAS Space Journal*, vol. 12, pp. 551–564, 2020.

[26] V. Leon, C. Bezaitis, G. Lentaris, D. Soudris, D. Reisis, E.-A. Papatheofanous, A. Kyriakos, A. Dunne, A. Samuelsson, and D. Steenari, "FPGA & VPU Co-Processing in Space Applications: Development and Testing with DSP/AI Benchmarks," in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–5.

[27] Space Alliance, "Star Tracker Assembly-Introduction," 2010.

[28] M. Salomon and W. Goss, "A microprocessor-controlled ccd star tracker," 1976.

[29] C. Liebe, "Accuracy performance of star trackers-a tutorial," in *IEEE Transactions on Aerospace and Electronic Systems*, 2002.

[30] A. Eisenman, C. Liebe, and J. Joergensen, "The new generation of autonomous star trackers," 1997.

[31] Wikipedia, "Equinox (celestial coordinates)."

[32] P.Walree, "Distortion," in *Photographic optics*, 2009.

[33] R. Bezooijen, "Autonomous star tracker development," in *IFAC Automatic Control in Aerospace*, 1992.

[34] G. Rufino and D. Accardo, "Enhancement of the centroiding algorithm for star tracker measure refinement," in *Science Direct*, 2002.

[35] Wei, Y. Zhao, G. Wang, and J. Li, "Real-time star identification using synthetic radial pattern and its hardware implementation," in *Acta Astronautica*, 2016.

[36] S. Schaire, S. Altunc, Y. Wong, O. Kegege, M. Shelton, and G. B. at al., "Investigation into new ground based communications service offerings in response to smallsat trends," in *32nd Annual AIAA/USU Conference on Small Satellites*, 2018.

[37] F. Jiancheng and N. Xiaolin, "Celestial navigation methods for space explorers."

[38] J. Rebordao, "Space optical navigation techniques: An overview," in *SPIE Digital Library*, 2013.

[39] E. Turan, S. Speretta, and E. Gill, "Autonomous navigation for deep space small satellites: Scientific and technological advances," in *Acta Astronautica*, 2022.

[40] S. Sheikh, D. Pines, J. Hanson, and P. Graven, "Spacecraft navigation and timing using x-ray pulsars," in *NAVIGATION: Journal of The Institute of Navigation*, vol. 58, no. 2, 2011.

[41] J. Dong, "Pulsar navigation in the solar system," 2018.

[42] K. Fujimoto, J. Leonard, R. McGranaghan, J. Parker, R. Anderson, and G. Born, "Simulating the liaison navigation concept in a geo + earth-moon halo constellation," 2012.

[43] K. Hill and G. Born, "Autonomous interplanetary orbit determination using satellite-to-satellite tracking," in *Journal of Guidance, Control and Dynamics*, vol. 30, no. 3, 2007.

[44] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," 2006, pp. 5–10.

[45] J. Peraire and S. Widnall, "Lecture L29 - 3D Rigid Body Dynamics."

[46] C. Wielligh, "Fast star tracker hardware implementation and algorithm optimisations on a system-on-a-chip device." no. 59-61, 2019.

[47] Xilinx, "Field Programmable Gate Array (FPGA)."

[48] Altera, "What is an SoC FPGA?" in *Architecture Brief*.

[49] N. Torsvik, "SoC FPGA Evaluation Guidelines."

[50] K. Maragos and D. Soudris, "Introduction to SoC FPGAs," in *NTUA-ECE, Class: Digital VLSI - Lec. 7*.

[51] ARM Developer, "About the AXI4-Stream protocol."

[52] Xilinx, "Processing system 7 v5.5," in *Vivado Design Suite*, 2017.