Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Acceleration of Computer Vision Algorithms for Star Trackers on SoC FPGA Platforms

Διπλωματικη Εργασια

Πανουσόπουλος Βασίλειος

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων
Αθήνα, Μάιος 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Department of Computer Science
Microprocessors and Digital Systems Lab

Acceleration of Computer Vision Algorithms for Star Trackers on SoC FPGA Platforms

Diploma Thesis

Panousopoulos Vasileios

Supervisor : Dimitrios Soudris
Professor N.T.U.A.

Microprocessors and Digital Systems Lab
Athens, May 2022

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

Acceleration of Computer Vision Algorithms for Star Trackers on SoC FPGA Platforms

Διπλωματική Εργασία

Πανουσόπουλος Βασίλειος

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13η Μαΐου 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....
Δημήτριος Σούντρης	Κωνσταντίνος Σιώζιος	Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.	Αναπ. Καθηγητής Α.Π.Θ.	Καθηγητής Ε.Μ.Π

Αθήνα, Μάιος 2022

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

Ευχαριστίες

Θα ήθελα να ευχαριστήσω από καρδιάς τους κ.κ. Γιώργο Λεντάρη και Βασίλη Λέων για την άψογη συνεργασία που είχαμε καθόλη τη διάρκεια εκπόνησης της διπλωματικής εργασίας. Για τις καθοριστικές συμβουλές σε καίρια σημεία και τον προσωπικό χρόνο που διέθεσαν ώστε να ολοκληρωθεί το παρόν έργο.

Ευχαριστώ θερμά τον κ. Μάνο Κουμαντάκη από την εταιρία Infinite Orbits, ο οποίος παρείχε το θέμα και τους αλγόριθμους της εργασίας και αποτέλεσε μία πλούσια πηγή καινοτόμων ιδεών και μεγάλου ενδιαφέροντος.

Ένα τεράστιο ευχαριστώ στον κ. Δημήτριο Σούντρη, από τον οποίο εμπνεύστηκα βαθύτατα κατά τη διάρκεια των σπουδών μου και ο οποίος καθόρισε περισσότερο από κάθε άλλον την ακαδημαϊκή μου πορεία.

Ακόμη, ένα μεγάλο ευχαριστώ στην Καταλίνα, για την υπομονή και τη στήριξη που μου παρείχε όλον αυτόν τον καιρό.

Τέλος, δεν μπορώ παρά να αφιερώσω αυτό το έργο στους γονείς μου Σταύρο και Κατερίνα, και στην αδερφή μου Σταυρίνα, χωρίς την υποστήριξη και τις θυσίες των οποίων τίποτα δεν θα ήταν δυνατό.

Περίληψη

Οι διαστημικές εφαρμογές απαιτούν επακριβή και γρήγορη εύρεση του προσανατολισμού των δορυφόρων, κάτι το οποίο μπορεί να επιτευχθεί μόνο με την χρήση ανιχνευτών αστεριών (star trackers). Αυτό το όργανο αποτελείται από έναν οπτικό αισθητήρα, ο οποίος συλαμβάνει εικόνες του ουρανού, και κατάλληλο ψηφιακό υλικό που ανιχνεύει τα αστέρια της εικόνας και τα αντιστοιχεί σε γνωστούς χάρτες, με σκοπό την εκτίμηση της θέσης του δορυφόρου στο διάστημα. Λόγω του μεγάλου αριθμού δεδομένων που παρέχει η κάμερα, η διαδικασία ανίχνευσης αστεριών έχει μεγάλο υπολογιστικό κόστος και συνεπώς η εκτέλεση της σε έναν γενικού σκοπού ενσωματωμένο επεξεργαστή θεωρείται μη αποδοτική. Επιπλέον η σύγχρονη τάση αξιοποίησης εμπορικού ψηφιακού υλικού (Commercial Off-The-Shelf HW) για εφαρμογές διαστήματος, μας οδηγεί στο να εξετάσουμε τέτοιου είδους στοιχεία για την υλοποίηση της αρχιτεκτονικής επεξεργασίας δεδομένων, όπως για παράδειγμα μία υψηλής ανάλυσης κάμερα σε συνδυασμό με ένα υψηλής απόδοσης COTS FPGA.

Σε αυτή την εργασία, παρουσιάζεται η επιτάχυνση της διαδικασίας κεντραρίσματος (centroiding) σε μία COTS FPGA πλατφόρμα, κατά την οποία εκτιμάται η ακριβής θέση των αστεριών μίας εικόνας. Στο πλαίσιο αυτό, υιοθετούνται δύο αλγόριθμοι που χαρακτηρίζονται από διαφορετικά επίπεδα ακρίβειας και πολυπλοκότητας, οι οποίοι βελτιστοποιούνται και υλοποιούνται στο υλικό χρησιμοποιώντας και τις δύο μεθόδους προγραμματισμού FPGA, δηλαδή με χρήση VHDL και HLS (C++). Έτσι είναι δυνατή η διεξαγωγή μίας αναλυτικής μελέτης των πλεονεκτημάτων που παρέχει η κάθε μέθοδος. Για την εύρεση της βέλτιστης λύσης στο πρόβλημα της ανίχνευσης αστεριών πραγματοποιείται μία εκτενής εξερεύνηση του χώρου σχεδίασης, κατά την οποία τα σχεδιασμένα μοντέλα ελέγχονται με χρήση τεχνητών εικόνων και αξιολογούνται με κριτήρια την ακρίβεια και τον χρόνο εκτέλεσης.

Η απλότητα του αλγόριθμου Κέντρου Βαρύτητας (Center of Gravity) επιτρέπει την επιτάχυνση κατά 2 τάξεις μεγέθους, αλλά η ακρίβεια ανίχνευσης μπορεί να θεωρηθεί μη ικανοποιητική. Συνεπώς, προτείνουμε μία καινοτόμα πλήρως διοχετευμένη FPGA υλοποίηση, η οποία παρουσιάζει άριστη ακρίβεια και υψηλή ταχύτητα. Αυτή βασίζεται στον αλγόριθμο Γρήγορου Γκαουσσιανού Ταιριάσματος (Fast Gaussian Fitting) ο οποίος επιταχύνεται κατά 25 φορές. Στην εργασία εξηγείται μέσω μίας ενδελεχούς ανάλυσης, πως η υψηλή πολυπλοκότητα του αλγορίθμου, που συνδέεται άρρηκτα με την υψηλή χρήση πόρων στο FPGA, περιορίζει την συνολική επιτάχυνση. Η προτεινόμενη αρχιτεκτονική είναι κατάλληλη για χρήση σε εφαρμογές πραγματικού χρόνου, καθώς διευκολύνει την επεξεργασία δεδομένων και επιτρέπει την παράλληλη εκτέλεση των διάφορων διεργασιών που συνυπάρχουν σε έναν ανιχνευτή αστεριών.

Λέξεις Κλειδιά

Ανιχνευτής Αστεριών, Κεντράρισμα Αστεριών, FPGA, VHDL, HLS, Βελτιστοποίηση στο Υλικό, Επιτάχυνση, Κέντρο Βαρύτητας, Γρήγορο Γκαουσσιανό Ταίριασμα

Abstract

Space applications demand precise and fast measurement of the satellite's orientation, which can only be achieved with the use of star trackers. This instrument consists of a camera sensor which captures sky images and appropriate digital hardware that detects the stars within the image and matches them to known maps, targeting to determine the satellite's attitude in the inertial space. The large amount of sensor data makes the task of detecting stars computationally intensive and therefore its execution on a general purpose embedded processor is inefficient. Additionally, the latest trends of utilizing Commercial Off-The-Shelf (COTS) HW in space leads us to examine such components in the processing architectures, i.e., a high-resolution camera and a high-performance COTS FPGA.

In this thesis, we present the acceleration of the centroiding process on COTS FPGA platforms, during which the precise position of each star in the image is estimated. Two centroiding algorithms which demonstrate different levels of accuracy and complexity have been adopted. We optimize and implement these algorithms on hardware using both VHDL and HLS (C++) as FPGA programming methods and perform extensive design space exploration to find the most efficient solution. An in-depth study of the advantages that each method provides is also presented. The hardware models are validated using simulated star images and evaluated in terms of computation time and centroiding accuracy.

The simplicity of the Center of Gravity algorithm enables a high degree of acceleration by 2 orders of magnitude but its accuracy might be considered insufficient. Therefore, we propose a fully pipelined novel FPGA implementation that features excellent accuracy and high efficiency. This design adopts the Fast Gaussian Fitting algorithm which is accelerated by 25x. A thorough analysis that describes the limitation of achieved acceleration due to high complexity and high FPGA resource utilization is provided. Ultimately, this architecture is suitable for real-time applications as it removes the bottleneck of data processing and can enable parallel execution of the processes involved in a star tracker.

Keywords

Star Tracker, Star Centroiding, FPGA, VHDL, HLS, Hardware Optimization, Acceleration, Center of Gravity, Fast Gaussian Fitting

Contents

List of Figures

List of Tables

Εκτεταμένη Περίληψη

0.1 Εισαγωγή

Από την πρώτη εκτόξευση δορυφόρου, του Sputnik 1 το 1957, πολλά έχουν αλλάξει στο πεδίο των διαστημικών τεχνολογιών. Λόγω της εξέλιξης των υλικών, της ηλεκτρονικής και της επιστήμης των υπολογιστών τις τελευταίες δεκαετίες έχει υπάρξει μεγάλη πρόοδος στην εξερεύνηση του διαστήματος και έχει αναπτυχθεί μία σημαντική διαστημική υποδομή, της οποίας βασικό στοιχείο αποτελούν οι δορυφόροι. Η αυξημένη απόδοση και η υψηλή αξιοπιστία των σύγχρονων υπολογιστικών συστημάτων επιτρέπει την δημιουργία πολυπλοκότερων αλγορίθμων και την διαχείριση περισσότερων δεδομένων, ενώ ο κίνδυνος βλάβης εξαιτίας της ουράνιας ακτινοβολίας έχει ελαχιστοποιηθεί. Έτσι οι δυνατότητες αναφορικά με την εκμετάλλευση του διαστήματος ολοένα και επεκτείνονται με αποτέλεσμα την συνεχή εμφάνιση καινοτόμων τεχνολογιών. Εκτός από τις κλασικές καθημερινές υπηρεσίες που βασίζονται στους δορυφόρους, όπως οι μετεωρολογικές προβλέψεις και η δορυφορική τηλεόραση, η σύγχρονη τάση σχετίζεται με την παροχή υπηρεσιών σε δορυφόρους που βρίσκονται ήδη σε τροχιά (in-orbit).

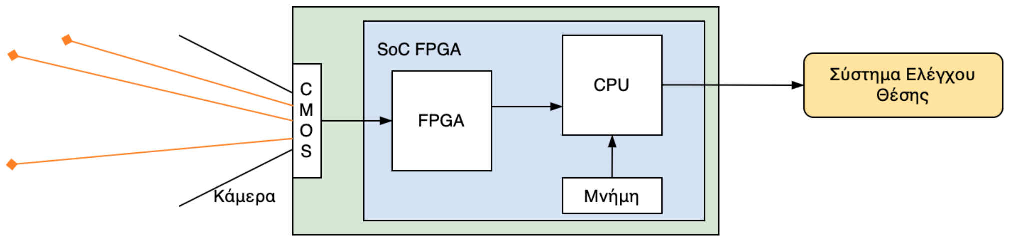
0.1.1 Κίνητρο

Η παρούσα διπλωματική εργασία αναπτύχθηκε σε συνεργασία με την εταιρία Infinite Orbits, η οποία χρησιμοποιεί καινοτόμες τεχνολογίες με σκοπό την παροχή τέτοιων υπηρεσιών. Η σημαντικότερη εξ αυτών αφορά την επέκταση ζωής των δορυφόρων, κατά την οποία απαιτείται η συνάντηση (rendezvous) δύο τέτοιων οχημάτων και η πρόσδεση (docking) του ενός στο άλλο, με μεγάλη ακρίβεια και ασφάλεια, κάτι το οποίο έχει αποδειχθεί ιδιαίτερα δύσκολο λόγω της μη συνεργατικής συμπεριφοράς του στοχευμένου δορυφόρου σε συνδυασμό με τους κινδύνους και τα χαρακτηριστικά της γεωστατικής τροχιάς (GEO). Η βασική ιδέα της συγκεκριμένης υπηρεσίας είναι η χρήση ενός μικρότερου δορυφόρου ως εναλλακτική πηγή καυσίμων για έναν παλαιότερο δορυφόρο που φθάνει στο τέλος της ζωής του, η οποία εν γένει καθορίζεται από τα διαθέσιμα καύσιμα. Λαμβάνοντας υπόψιν την υψηλή πολυπλοκότητα και το υψηλό κόστος που χαρακτηρίζουν ένα δορυφορικό σύστημα, το οποίο παρόλο που μπορεί να είναι πλήρως λειτουργικό πρέπει να αντικατασταθεί λόγω έλλειψης ενέργειας, είναι δεδομένο πως η συγκεκριμένη υπηρεσία καθιστά δυνατή τη διάσωση πόρων και τη μείωση του κόστους.

Βασική προϋπόθεση για την πραγματοποίηση τέτοιων πολύπλοκων εφαρμογών είναι η ύπαρξη ενός στιβαρού και αυτόνομου συστήματος πλοήγησης πραγματικού χρόνου. Η συγκεκριμένη εργασία ασχολείται με το πρόβλημα της πλοήγησης μακράς εμβέλειας (far-range navigation), κατά την οποία εκτιμάται η σχετική τροχιά του δορυφόρου ως προς κάποιο άλλο αντικείμενο, π.χ. έναν άλλο δορυφόρο, ώστε με τις κατάλληλες κινήσεις μετά από πεπερασμένο χρονικό διάστημα να βρεθούν στην ίδια τροχιά και να είναι δυνατή η πραγματοποίηση του ζητούμενου ραντεβού. Ένα θεμελιώδες στοιχείο της αυτόνομης πλοήγησης είναι η αδιάλειπτη και επακριβής γνώση της θέσης του δορυφόρου, το οποίο μπορεί να επιτευχθεί μόνο με τη χρήση των ανιχνευτών αστεριών.

0.1.2 Ανιχνευτές Αστεριών

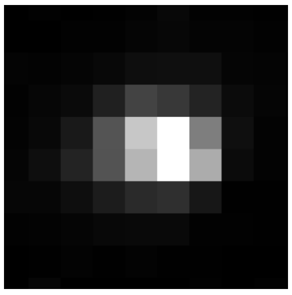
Οι ανιχνευτές αστεριών ανήκουν στο Σύστημα Καθορισμού & Ελέγχου Θέσης (Attitude Determination & Control System) ενός δορυφόρου και η λειτουργία τους βασίζεται στη μέθοδο προσανατολισμού που χρησιμοποιούσαν οι ταξιδευτές για χιλιάδες χρόνια, δηλαδή στην παρατήρηση αστεριών. Πιο συγκεκριμένα συλαμβάνοντας εικόνες του ουρανού και χρησιμοποιώντας κατάλληλους αλγορίθμους, καθορίζονται οι θέσεις των αστεριών της εικόνας και κατόπιν αντιστοίχισης τους σε γνωστούς χάρτες αστέρων είναι δυνατή η εύρεση της ακριβούς θέσης του δορυφόρου στο διάστημα. Μία τυπική διάταξη ενός ανιχνευτή αστεριών δίνεται στο Σχήμα 0.1.



Σχήμα 0.1: Απλοποιήμενη αναπαράσταση μίας τυπικής διάταξης ανιχνευτή αστεριών.

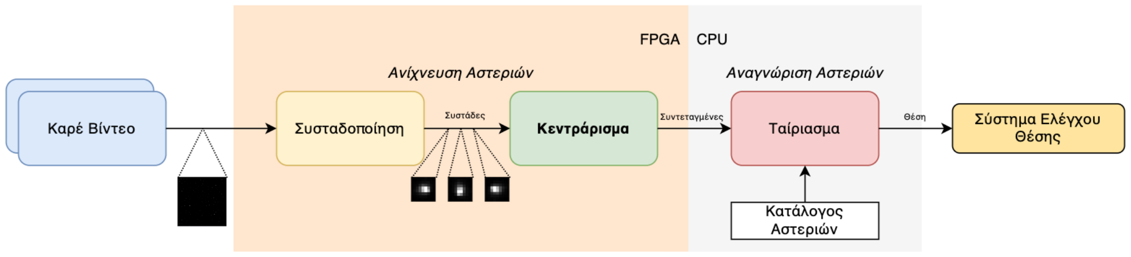
Οι ανιχνευτές αστεριών είναι όργανα ανθεκτικά στην κοσμική ακτινοβολία που αποτελούνται από μία ηλεκτρονική κάμερα και κατάλληλα ηλεκτρονικά. Στη σύγχρονη μορφή τους χρησιμοποιούν μία υψηλής ανάλυσης CMOS κάμερα η οποία δημιουργεί μία ροή δεδομένων που οδηγείται στην μονάδα επεξεργασίας, όπου αρχικά εντοπίζονται οι αστέρες μέσω επεξεργασίας εικόνας και ύστερα χρησιμοποιούνται αλγόριθμοι αναγνώρισης προτύπων για να αντιστοιχηθούν στους γνωστούς ουράνιους χάρτες και να βρεθεί η ακριβής θέση τους. Λόγω του τεράστιου αριθμού pixels που παράγει ο οπτικός αισθητήρας, η επεξεργασία τους αποτελεί μία υπολογιστικά απαιτητική εργασία. Λαμβάνοντας υπόψιν την ανάγκη για λειτουργία πραγματικού χρόνου, η εκτέλεση αυτής της διαδικασίας σε έναν παραδοσιακό γενικού σκοπού επεξεργαστή είναι μη αποδοτική και συνεπώς είναι αναγκαία η χρήση ενός System-on-Chip (SoC) που συμπεριλαμβάνει ένα COTS FPGA. Σε ένα τέτοιο σύστημα, το FPGA συνήθως συνδέεται απευθείας στον αισθητήρα εισόδου μέσω ενός πρωτοκόλλου υψηλής ταχύτητας για την επεξεργασία της εικόνας και στη συνέχεια ο υψηλής απόδοσης επεξεργαστής χρησιμοποιείται για την εύρεση του προσανατολισμού του δορυφόρου.

Λόγω των μη ιδανικών φωτογραφικών φακών που χρησιμοποιούνται στον πραγματικό κόσμο, η ροή δεδομένων που παράγεται από τον αισθητήρα περιέχει αστέρια που φαίνονται θολά στο ανθρώπινο μάτι, καθώς η φωτεινή πληροφορία διαχέεται σε έναν αριθμό από pixels δημιουργώντας φωτεινές συστάδες (clusters). Ένα τέτοιο θολό αστέρι φαίνεται στο Σχήμα 0.2.



Σχήμα 0.2: Μία φωτεινή περιοχή γύρω από ένα πλήθος pixels αντιστοιχεί σε ένα πραγματικό αστέρι μίας νυχτερινής φωτογραφίας.

Με δεδομένη μία ψηφιακή εικόνα που περιέχει τέτοιου είδους αστέρια και καλύπτει ένα μικρό μέρος του νυχτερινού ουρανού, το ζητούμενο είναι η εύρεση της ακριβούς θέσης του δορυφόρου. Η λύση σε αυτό το πρόβλημα μπορεί να δοθεί από την αλυσίδα διαδικασιών που φαίνεται στο Σχήμα 0.3.



Σχήμα 0.3: Η ακολουθία των διαδικασιών που περιέχονται σε έναν ανιχνευτή αστεριών.

Η πρώτη από τις τρεις διαδικασίες ονομάζεται συσταδοποίηση ή ομαδοποίηση (clustering) και είναι υπεύθυνη για την αναγνώριση των φωτεινών περιοχών σε μία εικόνα και για την εξαγωγή των αντίστοιχων pixels. Αυτό επιτυγχάνεται με την προσπέλαση ολόκληρης της εικόνας και την σύγκριση της φωτεινότητας (intensity ή brightness) καθενός pixel με κάποιο κατώφλι. Έτσι κάθε συστάδα δημιουργείται από ένα σύνολο γειτονικών pixels με ικανοποιητική φωτεινότητα.

Στη συνέχεια τα clusters που προέκυψαν οδηγούνται στην διαδικασία κεντραρίσματος (centroiding) με σκοπό την εύρεση του κέντρου κάθε περιοχής-αστεριού. Κάθε τέτοιο δισδιάστατο κέντρο αντιστοιχεί στην θέση του πραγματικού αστεριού στο επίπεδο της ψηφιακής εικόνας και η χρήση κατάλληλων αλγορίθμων επιτρέπει τον υπολογισμό του με υψηλή ακρίβεια σε επίπεδο subpixel.

Τελικά το κέντρο κάθε αστεριού χρησιμοποιείται από την διαδικασία ταιριάσματος (matching), κατά την οποία τα εντοπισμένα αστέρια αντιστοιχίζονται σε γνωστά αστέρια με χρήση καταλόγων με σκοπό την κατανόηση της εκάστοτε θέσης του δορυφόρου στο διάστημα.

Είναι γεγονός πως οι δύο πρώτες διαδικασίες επεξεργάζονται μεγάλο αριθμό από pixels, ενώ η τελευταία εφαρμόζει πολύπλοκους αλγορίθμους σε δεδομένα χαμηλών διαστάσεων. Λαμβάνοντας υπόψιν αυτά τα χαρακτηριστικά κρίνεται αναγκαίος ο διαχωρισμός των εργασιών, ώστε οι δύο υπολογιστικά απαιτητικές διαδικασίες να υλοποιηθούν στο FPGA με σκοπό την επιτάχυνση τους, ενώ η πιο σύνθετη φάση να αναπτυχθεί στον επεξεργαστή και να βελτιστοποιηθεί σε επίπεδο λογισμικού.

Σε κάθε περίπτωση για την επίτευξη λειτουργίας πραγματικού χρόνου είναι απαραίτητη η παράλληλη εκτέλεση των διεργασιών που περιγράφηκαν, ώστε κάθε μία να εκτελείται στον δικό της ρυθμό και η ταχύτητα του συνολικού σύστημα να περιορίζεται μονάχα από την διαθεσιμότητα των δεδομένων.

0.1.3 Στόχος Εργασίας

Ο κύριος στόχος του παρόντος ερευνητικού έργου είναι η βελτιστοποίηση της διαδικασίας κεντραρίσματος σε μία SoC FPGA πλατφόρμα για μία δεδομένη διάταξη ενός ανιχνευτή αστεριών. Αυτή η διάταξη παρέχεται από την Infinite Orbits και αποτελείται από τα παρακάτω στοιχεία.

1. Εξατομικευμένος συνδυασμός κάμερας και φακού με:
   - Οπτικό πεδίο (Field of View): 15.5° x 15.5°
   - Ανάλυση εικόνας: 2048 x 2048 pixels
   - Βάθος pixel: 12 bits
2. Αναπτυξιακή πλακέτα ZedBoard που ενσωματώνει το Xilinx ZYNQ-7020 SoC

Οι βασικές προδιαγραφές επίδοσης του ανιχνευτή αστεριών δίνονται κάτωθι.

- Απόδοση πραγματικού χρόνου περίπου 1-2 καρέ ανά δευτερόλεπτο
- Μέσο σφάλμα μικρότερο από 3 δευτερόλεπτα της μοίρας ως προς τον οπτικό άξονα της κάμερας (boresight)

Οι αντίστοιχες προδιαγραφές για το centroiding που λήφθηκαν υπόψιν κατά τη σχεδίαση των μοντέλων είναι οι παρακάτω.

- Επιτάχυνση των αλγορίθμων κεντραρίσματος για βέλτιστη επεξεργασία δεδομένων
- Τυπικό μέσο subpixel σφάλμα μικρότερο από 0.1 pixels

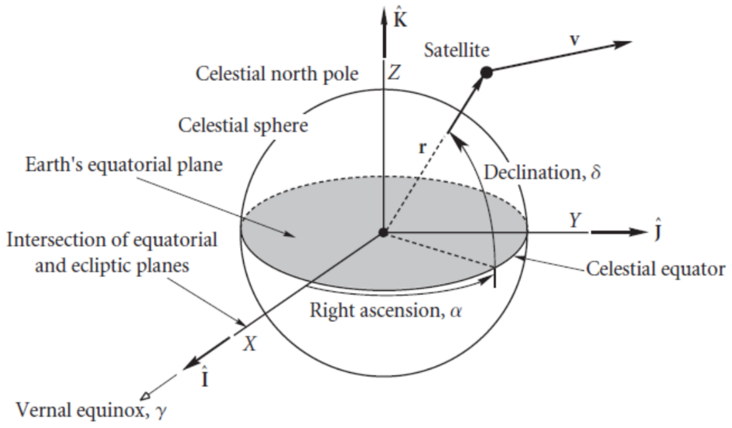
Για τον σκοπό αυτό, προτείνονται δύο διαφορετικές βελτιστοποιημένες αρχιτεκτονικές που αναπτύσσονται στο FPGA και υιοθετούν την τεχνική της διοχέτευσης με σκοπό τον γρήγορο υπολογισμό του κέντρου των αστεριών με υψηλή ακρίβεια. Επιπρόσθετα, πραγματοποιείται μία αναλυτική σύγκριση των υλοποιήσεων ως προς την ταχύτητα, την ακρίβεια και την χρησιμοποίηση πόρων στο FPGA. Στα πλαίσια της εργασίας χρησιμοποιήθηκαν και οι δύο διαθέσιμοι τρόποι προγραμματισμού FPGA, είτε με Γλώσσα Περιγραφής Υλικού (Hardware Description Language) είτε με Γλώσσα Υψηλού Επιπέδου (High Level Language). Πιο συγκεκριμένα αξιολογήθηκαν και συγκρίθηκαν τα αποτελέσματα προγραμματισμού με VHDL και C++.

0.2 Θεωρητικό Υπόβαθρο

0.2.1 Μετασχηματισμός Συντεταγμένων

Ένα σημαντικό ερώτημα που μπορεί να προκύψει σχετικά με τον τρόπο λειτουργίας ενός ανιχνευτή αστεριών, αφορά το πως υπολογίζεται η πραγματική θέση ενός αστεριού στο διάστημα με δεδομένη τη θέση του στο επίπεδο της ψηφιακής εικόνας. Για την απάντηση αυτού του ερωτήματος απαιτείται η εισαγωγή τριών διαφορετικών συστημάτων συντεταγμένων, καθένα εκ των οποίων παρέχει συγκεκριμένα πλεονεκτήματα σε διαφορετικές προσεγγίσεις του υπό εξέταση προβλήματος προσανατολισμού του δορυφόρου.

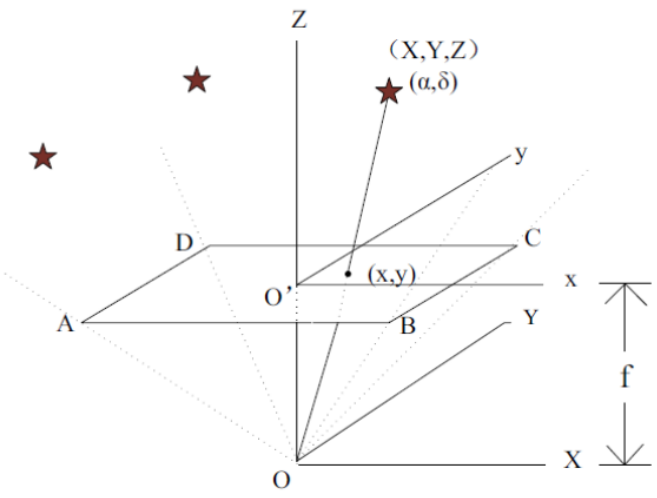
Για την περιγραφή της θέσης των αστεριών ως προς τη Γη χρησιμοποιείται ένα γεωκεντρικό καρτεσιανό αδρανειακό σύστημα συντεταγμένων που ονομάζεται Earth Centered Inertial J2000 και έχει την αρχή των αξόνων του στο κέντρο της Γης όπως φαίνεται στο Σχήμα 0.4. Λόγω των τεράστιων αποστάσεων των ουράνιων σωμάτων από τον πλανήτη μας, για έναν παρατηρητή που βρίσκεται στην επιφάνεια ή κοντά στην επιφάνεια της Γης (π.χ. σε έναν δορυφόρο) όλα τα αστέρια μοιάζουν να έχουν την ίδια μεγάλη απόσταση, σαν να βρίσκονται αγκιστρωμένα σε ένα σφαιρικό πλέγμα αυθαίρετης ή άπειρης ακτίνας. Για την ορθή περιγραφή της θέσης αυτών των σωμάτων χρησιμοποιείται ένα σφαιρικό σύστημα, με συντεταγμένες την δεξιά ανύψωση α και την απόκλιση δ.



Σχήμα 0.4: Γεωκεντρικό Σύστημα Συντεταγμένων

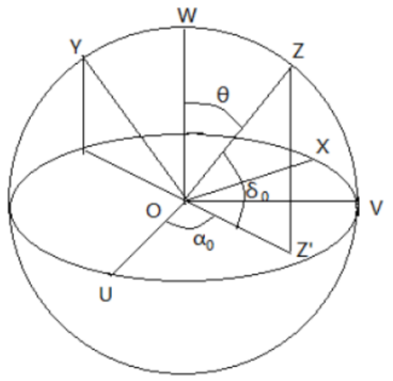
Η θεμελιώδης λειτουργία μίας φωτογραφικής κάμερας είναι η προβολή ενός σημείου από τον τρισδιάστατο χώρο στο δισδιάστατο επίπεδο της εικόνας. Η γεωμετρική αντιστοιχία που ορίζει αυτή την προβολή μπορεί να περιγραφεί ευκολότερα με την εισαγωγή ενός καρτεσιανού συστήματος συντεταγμένων με αρχή το κέντρο της κάμερας, το οποίο ονομάζεται σύστημα *σώματος αισθητήρα* (sensor body). Στο Σχήμα 0.5 δίνεται το συγκεκρίμενο σύστημα σε συνδυασμό με το τρίτο σύστημα συντεταγμένων, το οποίο ταυτίζεται με την επιφάνεια του αισθήτηρα και είναι το δισδιάστατο επίπεδο της εικόνας.

Το αποτέλεσμα ενός αλγόριθμου κενταρίσματος εκφράζει τη θέση ενός αστεριού στο δισδιάστατο επίπεδο της εικόνας, κάτι το οποίο δεν έχει φυσικό νόημα καθώς το συγκεκριμένο σύστημα συντεταγμένων χρησιμοποιείται για ευκολία και απλοποίηση των υπολογισμών. Επίσης η διαδικασία ταιριάσματος για να αντιστοιχήσει τα εντοπισμένα αστέρια στους γνωστούς καταλόγους χρειάζεται να γνωρίζει τη θέση τους στον τρισδιάστατο χώρο και όχι απλά εντός της εικόνας. Συνεπώς είναι αναγκαία η εύρεση και κατανόηση της σχέσης μεταξύ της πραγματικής θέσης ενός αστεριού στον χώρο και της θέσης του centroid στην εικόνα.



Σχήμα 0.5: Αναπαράσταση του Συστήματος Σώματος Αισθητήρα και του Επιπέδου Εικόνας.

Η σχέση αυτή πηγάζει από την ίδια τη διαδικασία σχηματισμού μίας ψηφιακής εικόνας, η οποία μπορεί να χωριστεί σε δύο διαδοχικούς μετασχηματισμούς συντεταγμένων. Ο πρώτος είναι ένας μετασχηματισμός περιστροφής από το γεωκεντρικό σύστημα στο σύστημα σώματος αισθητήρα και μπορεί να αναλυθεί με τη βοήθεια του Σχήματος 0.6.



Σχήμα 0.6: Συνδυασμένη αναπαράσταση Γεωκεντρικού Συστήματος και Συστήματος Σώματος Αισθητήρα.

Ένα σημείο προς καταγραφή που βρίσκεται στον χώρο, περιγράφεται με σφαιρικές συντεταγμένες ως $\mathbf{A_i} = [\alpha\ \delta]^T$. Αν χρησιμοποιηθεί η καρτεσιανή μορφής τους, το σημείο αυτό εκφράζεται ως:

$$\mathbf{A_i} = \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} \cos\alpha\cos\delta \\ \sin\alpha\cos\delta \\ \sin\delta \end{bmatrix} \tag{0.1}$$

Το ίδιο σημείο με αναφορά στο σύστημα σώματος αισθητήρα εκφράζεται ως:

$$\mathbf{A_i} = [X\ Y\ Z]^T \tag{0.2}$$

Ο μετασχηματισμός περιστροφής δίνεται παρακάτω.

$$[X\ Y\ Z]^T = \mathbf{M}[U\ V\ W]^T \tag{0.3}$$

όπου **M** ο πίνακας περιστροφής. Χρησιμοποιώντας την αρχή της 3-1-3 περιστροφής συντεταγμένων του Euler ο πίνακας αυτός ορίζεται ως:

$$\mathbf{M} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{0.4}$$

όπου ψ, θ και ϕ είναι οι γωνίες περιστροφής.

Ο δεύτερος σε σειρά μετασχηματισμός αναπαριστά την προβολή ενός σημείου από τον τρισδιάστατο χώρο, εκφρασμένος πλέον στο σύστημα σώματος αισθητήρα, στο δισδιάστατο επίπεδο εικόνας. Η προβολή αυτή ονομάζεται ¨προοπτική προβολή' και ορίζεται παρακάτω.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{0.5}$$

όπου $[x\ y]^T$ είναι οι συντεταγμένες του σημείου στο επίπεδο της εικόνας σε συνεχή μορφή και f η εστιακή απόσταση. Στη συνέχεια με μία απλή διακριτοποίηση προκύπτουν οι συντεταγμένες σε pixels.

Η παραπάνω διαδικασία εκφράζει μαθηματικά τον τρόπο σχηματισμού μίας ψηφιακής εικόνας. Η αντίστροφη διαδικασία μετασχηματισμού από τις δισδιάστατες στις τρισδιάστατες συντεταγμένες ονομάζεται *Προσαρμογή Θέσης* και είναι απαραίτητη για την εύρεση της πραγματικής θέσης ενός αστεριού στο χώρο με δεδομένη τη θέση του στην εικόνα.

0.2.2 Η Ανάγκη για Subpixel Ακρίβεια

Ένα εξίσου σημαντικό θεωρητικό ζήτημα αφορά τον τρόπο με τον οποίο προκύπτει η προδιαγραφή σφάλματος, που σύμφωνα με την Ενότητα 0.1.3 ορίζεται στα 0.1 pixels. Είναι προφανές πως η ακρίβεια σε επίπεδο pixel, η οποία είναι ζωτικής σημασίας σε συστήματα επεξεργασίας εικόνας, δεν έχει από μόνη της φυσική υπόσταση καθώς αναφέρεται σε ένα βοηθητικό σύστημα συντεταγμένων. Συνεπώς κρίνεται απαραίτητη η κατανόηση της αντιστοιχίας μεταξύ της ακρίβειας ως προς τον οπτικό άξονα (boresight), δηλαδή της πραγματικής ακρίβειας στον χώρο, και της ακρίβειας σε επίπεδο pixel.

Το οπτικό πεδίο (FOV) μίας κάμερας ορίζεται ως η στερεά γωνία μέσω της οποίας ο αισθητήρας είναι ευαίσθητος σε ηλεκτρομαγνητική ακτινοβολία. Δηλαδή εκφράζει το μέρος του πραγματικού κόσμου, σε μοίρες, που είναι ορατό από την κάμερα και συνεπώς προβάλλεται μέσω της διαδικασίας σχηματισμού εικόνων σε έναν πεπερασμένο αριθμό από pixels. Για κάθε διάταξη κάμερας-φακού το FOV και η ανάλυση της εικόνας, δηλαδή ο αριθμός των pixels σε κάθε διάσταση, είναι γνωστά. Συνεπώς το μέρος του πραγματικού τρισδιάστατου κόσμου που καλύπτεται από ένα pixel δίνεται από την παρακάτω σχέση.

$$\frac{\text{Μοίρες}}{\text{Pixel}} = \frac{\text{FOV}}{\text{Πλήθος Pixel}} \tag{0.6}$$

Με δεδομένα τα χαρακτηριστικά της αισθητήριας διάταξης που παρέχει η Infinite Orbits και λαμβάνοντας υπόψιν πως ένα δευτερόλεπτο αντιστοιχεί στο $\frac{1}{3600}$ της μοίρας, προκύπτει ο αριθμός των δευτερολέπτων της μοίρας που καλύπτονται από ένα pixel.

$$\frac{\text{Δευτερόλεπτα}}{\text{Pixel}} = \frac{\text{Μοίρες}}{\text{Pixel}} \cdot 3600 = \frac{\text{FOV}}{\text{Ανάλυση}} \cdot 3600 = \frac{15.5}{2048} \cdot 3600 = 27.246 \tag{0.7}$$

Αποδεικνύεται λοιπόν, πως η ζητούμενη ακρίβεια των 3 δευτερολέπτων της μοίρας ως προς τον οπτικό άξονα μεταφράζεται σε ψηφιακή ακρίβεια περίπου 0.1 pixels.

0.3 Αλγόριθμοι Κεντραρίσματος

0.3.1 Αλγόριθμος Κέντρου Βαρύτητας

Η χρήση του αλγόριθμου Κέντρου Βαρύτητας (Center of Gravity) αποτελεί την πιο παραδοσιακή προσέγγιση στο πρόβλημα centroiding καθώς είναι ο πιο απλός και γρήγορος διαθέσιμος αλγόριθμος. Το κέντρο κάθε αστεριού (x_c, y_c) ορίζεται από έναν αριθμητικό μέσο σε κάθε διάσταση σύμφωνα με τον παρακάτω τύπο.

$$x_c = \frac{\sum I_i x_i}{\sum I_i} \tag{0.8}$$

$$y_c = \frac{\sum I_i y_i}{\sum I_i} \tag{0.9}$$

όπου τα αθροίσματα υπολογίζονται ως προς όλα τα pixels i ενός cluster, με συντεταγμένες (x_i, y_i), και I_i είναι η φωτεινότητα κάθε pixel στην κλίμακα του γκρι.

Λόγω της απλότητας του συγκεκριμένου αλγορίθμου, η απόδοση του γενικά περιορίζεται από την υψηλή ευαισθησία που εμφανίζει στον θόρυβο υποβάθρου και στο σφάλμα τύπου S-καμπύλης που είναι έμφυτο στην διαδικασία δειγματοληψίας.

Για να είναι δυνατή η πλήρης εκμετάλλευση των πλεονεκτημάτων που προσφέρει η υλοποίηση ενός αλγορίθμου σε ένα FPGA είναι απαραίτητη η κατάλληλη τροποποίηση του ώστε να ταιριάζει καλύτερα στα χαρακτηριστικά αυτής της πλατφόρμας. Μία από τις βασικότερες αρχές ψηφιακής σχεδίασης είναι η ελαχιστοποίηση του μεγέθους των σημάτων, δηλαδή του αριθμού των ψηφίων (bits) που απαιτούνται για την αναπαράσταση τους. Με αυτόν τον τρόπο μειώνονται οι πόροι και τα λογικά επίπεδα που απαιτούνται για κάθε υπολογισμό με αποτέλεσμα την σχεδίαση βέλτιστων κυκλωμάτων.

Σύμφωνα με τις οδηγίες που παρέχει η Infinite Orbits, θεωρείται πως κάθε φωτεινή περιοχή έχει τετραγωνικό σχήμα. Λαμβάνοντας υπόψιν αυτό το χαρακτηριστικό, οι εξισώσεις του αλγόριθμου CG μπορούν να μετατραπούν στην παρακάτω μορφή.

$$x_c = X + \frac{\sum I_i x_i}{\sum I_i} \tag{0.10}$$

$$y_c = Y + \frac{\sum I_i y_i}{\sum I_i} \tag{0.11}$$

όπου τα X και Y αντιστοιχούν στην πρώτη στήλη και γραμμή του τετραγωνικού cluster αντίστοιχα και κάθε ζεύγος (x_i, y_i) εκφράζει πλέον την απόσταση κάθε pixel i από την πρώτη στήλη και γραμμή.

Δηλαδή, σε αντίθεση με τις Εξισώσεις 0.8, 0.9 όπου κάθε τέτοιο ζεύγος αποτελεί τις απόλυτες συντεταγμένες ενός pixel με αναφορά στο επίπεδο της εικόνας, πλέον εκφράζει τις σχετικές συντεταγμένες ως προς το cluster. Επειδή προφανώς οι σχετικές συντεταγμένες απαιτούν σημαντικά μικρότερο αριθμό ψηφίων για να αναπαρασταθούν σε σχέση με τις απόλυτες συντεταγμένες, οι πράξεις που ορίζονται στον αριθμητή και στον παρονομαστή των Εξισώσεων 0.10, 0.11 απαιτούν λιγότερους πόρους και εκτελούνται ταχύτερα με αποτέλεσμα την επίτευξη βέλτιστων σχεδιασμών.

0.3.2 Αλγόριθμος Γρήγορου Γκαουσσιανού Ταιριάσματος

Ο δεύτερος αλγόριθμος που υιοθετήθηκε στα πλαίσια του παρόντος ερευνητικού έργου ανήκει στην οικογένεια των μεθόδων ταιριάσματος (fitting methods) και αναμένεται να δώσει τα καλύτερα αποτελέσματα ως προς την ακρίβεια συγκριτικά με όλους τους διαθέσιμους αλγορίθμους κεντραρίσματος. Η βασική ιδέα αυτών των μεθόδων είναι πως κάθε φωτεινή περιοχή, δηλαδή κάθε cluster, μοντελοποιείται από μία Συνάρτηση Εξάπλωσης Σημείου (Point Spread Function), η οποία μπορεί να προσεγγιστεί από μία Γκαουσσιανή κατανομή και στόχος είναι η εύρεση των παραμέτρων αυτής της κατανομής.

Πιο συγκεκριμένα, θεωρείται πως η κάθε αστέρι εκφράζεται από την παρακάτω Γκαουσσιανή συνάρτηση.

$$S(x_i, y_i|\mathbf{v}) = A \exp\left(-\frac{(x_i - x_c)^2}{2\sigma_x^2} - \frac{(y_i - y_c)^2}{2\sigma_y^2}\right) \tag{0.12}$$

όπου $(x_i,\, y_i)$ είναι οι γνωστές συντεταγμένες κάθε pixel και $\mathbf{v} = (A, x_c, y_c, \sigma_x, \sigma_y)$ είναι οι άγνωστες παράμετροι που πρέπει να υπολογιστούν. A είναι το πλάτος της συνάντησης που αναπαριστά την γενική φωτεινότητα του αστεριού, δηλαδή όσο αυξάνεται το A τόσο πιο φωτεινό είναι το αντίστοιχο cluster. Επίσης το ζεύγος $(x_c,\, y_c)$ εκφράζει το κέντρο του αστεριού ενώ τα σ_x, σ_y αποτελούν την τυπική απόκλιση της συνάρτησης ως προς κάθε διάσταση. Αξίζει να σημειωθεί πως στην εξεταζόμενη εφαρμογή απαιτείται μονάχα η εύρεση των παραμέτρων x_c, y_c.

Το ληφθέν σήμα που απεικονίζεται σε κάθε pixel της εικόνας ορίζεται ως:

$$I_i = S_i + N_i \tag{0.13}$$

όπου I_i είναι η παρατηρούμενη φωτεινότητα, S_i είναι η πραγματική φωτεινότητα και N_i είναι ο προστιθέμενος θόρυβος που ακολουθεί επίσης μία Γκαουσσιανή κατανομή.

Ο κλασικός αλγόριθμος Γκαουσσιανού Ταιριάσματος (Gaussian Fitting) υπολογίζει τις ζητούμενες παραμέτρους με βάση την παρακάτω αντικειμενική συνάρτηση.

$$Z = \arg\min_{\mathbf{v}} \sum_{i \in U} [z_i]^2 = \arg\min_{\mathbf{v}} \sum_{i \in U} [N_i]^2 \tag{0.14}$$

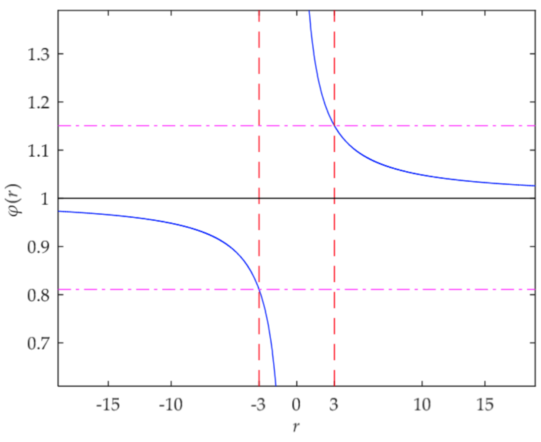
όπου $z_i = I_i - S_i = N_i$ αναπαριστά την απόσταση, μεταξύ της παρατηρούμενης τιμής I_i και της τιμής που ορίζει η υποβόσκουσα Γκαουσσιανή κατανομή, που πρέπει να ελαχιστοποιηθεί. Αν και αυτή η προσέγγιση δίνει τα βέλτιστα αποτελέσματα από άποψη ακρίβειας, η Εξίσωση 0.14 περιγράφει ένα πρόβλημα μη γραμμικών ελάχιστων τετραγώνων με συνέπεια η επίλυση του να χαρακτηρίζεται από υψηλές υπολογιστικές απαιτήσεις και έτσι η μέθοδος αυτή να καθίσταται μη αποδοτική για εφαρμογές πραγματικού χρόνου.

Ο αλγόριθμος Γρήγορου Γκαουσσιανού Ταιριάσματος (Fast Gaussian Fitting) μετασχηματίζει το αρχικό πρόβλημα σε ένα πρόβλημα γραμμικών ελάχιστων τετραγώνων και μπορεί να προσεγγίσει σε κλειστή μορφή τις λύσεις της αρχικής μεθόδου χωρίς απώλειες στην ακρίβεια ενώ ταυτόχρονα πετυχαίνει υψηλή επιτάχυνση και ανθεκτικότητα στο θόρυβο. Αυτή η τροποποίηση βασίζεται στην εφαρμογή του λογάριθμου στην Εξίσωση 0.14, η οποία μετά από ένα σύνολο βημάτων παίρνει την παρακάτω τελική της μορφή.

$$H = \arg\min_{\mathbf{v}} \sum_{i \in U} [\phi(r_i) N_i]^2 \tag{0.15}$$

Συγκρίνοντας τις Εξισώσεις 0.14, 0.15 γίνεται αντιληπτό πως ο FGF αποτελεί μία σταθμισμένη εκδοχή του GF αλγορίθμου, με τα βάρη να προκύπτουν από τον όρο $\phi(r_i)$, ο οποίος εξ ορισμού είναι συνάρτηση του σηματοθορυβικού λόγου (SNR). Η γραφική παράσταση αυτής της συνάρτησης δίνεται στο Σχήμα 0.7.

Συμπεραίνεται λοιπόν πως εάν ένα cluster περιέχει pixels με σηματοθορυβικό λόγο μεγαλύτερο από μία συγκεκριμένη τιμή ώστε η τιμή $\phi(r_i)$ αυτών των pixels να είναι αρκετά κοντά στο 1, τότε η Εξίσωση 0.15 προσεγγιστικά ταυτίζεται με την Εξίσωση 0.14 και έτσι ο αλγόριθμος FGF δίνει τα



Σχήμα 0.7: Η καμπύλη του $\phi(r_i)$. Η τιμή του $\phi(r_i)$ συγκλίνει στο 1 με την αύξηση του SNR. Όταν το r είναι ίσο με 3, η τιμή του $\phi(r_i)$ ισούται με 1.1507. Όταν το r είναι ίσο με -3, η τιμή του $\phi(r_i)$ ισούται με 0.8109.

ίδια αποτελέσματα με τον GF. Δηλαδή πετυχαίνει την ίδια ακρίβεια, επιλύοντας ωστόσο ένα πολύ πιο απλό πρόβλημα. Αναπτύσσοντας την Εξίσωση 0.15, προκύπτει η παρακάτω μορφή.

$$H = \underset{m,n,p,q,k}{\arg\min} \sum_{i \in U} (I_i x_i^2 m + I_i y_i^2 n + I_i x_i p + I_i y_i q + I_i k + I_i \ln I_i)^2 \tag{0.16}$$

Η Εξίσωση 0.16 περιγράφει το προς επίλυση πρόβλημα γραμμικών ελάχιστων τετραγώνων με 5 άγνωστες παραμέτρους m, n, p, q και k. Αυτές ορίζονται με βάση τις πραγματικά ζητούμενες Γκαουσσιανές παραμέτρους σύμφωνα με τα παρακάτω.

$$\begin{cases} m = \frac{1}{2\sigma_x^2} \\ n = \frac{1}{2\sigma_y^2} \\ p = -\frac{x_c}{\sigma_x^2} \\ q = -\frac{y_c}{\sigma_y^2} \\ k = \frac{x_c^2}{2\sigma_x^2} + \frac{y_c}{2\sigma_y^2} - I_i \ln A \end{cases} \tag{0.17}$$

Για την επίλυση του γραμμικού προβλήματος και την εύρεση του ολικού ελαχίστου εφαρμόζονται οι μερικές παράγωγοι, οι οποίες τίθενται ίσες με το 0. Έτσι παράγονται 5 γραμμικές εξισώσεις με 5 αγνώστους, οι οποίες ονομάζονται *κανονικές εξισώσεις* και δίνονται κάτωθι.

$$\left[\sum am_i^2\right] m + \left[\sum am_i an_i\right] n + \left[\sum am_i ap_i\right] p + \left[\sum am_i aq_i\right] q + \left[\sum am_i ak_i\right] k = -\sum am_i a_i \tag{0.18α'}$$

$$\left[\sum am_i an_i\right] m + \left[\sum an_i^2\right] n + \left[\sum an_i ap_i\right] p + \left[\sum an_i aq_i\right] q + \left[\sum an_i ak_i\right] k = -\sum an_i a_i \tag{0.18β'}$$

$$\left[\sum am_i ap_i\right] m + \left[\sum an_i ap_i\right] n + \left[\sum ap_i^2\right] p + \left[\sum ap_i aq_i\right] q + \left[\sum ap_i ak_i\right] k = -\sum ap_i a_i \tag{0.18γ'}$$

$$\left[\sum am_i aq_i\right] m + \left[\sum an_i aq_i\right] n + \left[\sum ap_i aq_i\right] p + \left[\sum aq_i^2\right] q + \left[\sum aq_i ak_i\right] k = -\sum aq_i a_i \tag{0.18δ'}$$

$$\left[\sum am_i ak_i\right] m + \left[\sum an_i ak_i\right] n + \left[\sum ap_i ak_i\right] p + \left[\sum aq_i ak_i\right] q + \left[\sum ak_i^2\right] k = -\sum ak_i a_i \tag{0.18ε'}$$

Οι συντελεστές am_i, an_i, ap_i, aq_i, ak_i και a_i ορίστηκαν για απλότητα με χρήση των γνωστών συντεταγμένων και της γνωστής φωτεινότητας κάθε pixel όπως φαίνεται παρακάτω.

$$\begin{cases} am_i = I_i x_i^2 \\ an_i = I_i y_i^2 \\ ap_i = I_i x_i \\ aq_i = I_i y_i \\ ak_i = I_i \\ a_i = I_i \ln I_i \end{cases} \tag{0.19}$$

Συνοψίζοντας την παραπάνω ανάλυση, έχει καταστεί φανερή η απλοποίηση του αρχικού μη γραμμικού προβλήματος, καθώς ο FGF αλγόριθμος απαιτεί την απλή επίλυση του γραμμικού συστήματος 5x5 που ορίζεται από τις Εξισώσεις 0.18 και το οποίο σε αλγεβρική μορφή πινάκων γράφεται ως:

$$A\mathbf{x} = \mathbf{b} \tag{0.20}$$

Σύμφωνα με την θεωρία Γραμμικής Άλγεβρας, ο πιο αποδοτικός αλγόριθμος επίλυσης ενός γραμμικού συστήματος είναι η Παραγοντοποίηση Cholesky. Στο σημείο αυτό πρέπει να τονιστεί πως οι μαθηματικές προϋποθέσεις για την χρησιμοποίηση της μεθόδου Cholesky ικανοποιούνται στην υπό εξέταση εφαρμογή. Η βασική ιδέα αυτής της μεθόδου είναι η αποδόμηση του αρχικού προβλήματος της Εξίσωσης 0.20 σε δύο απλούστερα προβλήματα που δίνονται κάτωθι.

$$L^\top \mathbf{x} = \mathbf{y} \tag{0.21}$$

$$L\mathbf{y} = \mathbf{b} \tag{0.22}$$

Αυτό επιτυγχάνεται με την παραγοντοποίηση του πίνακα συντελεστών A όπως παρακάτω.

$$A = LL^\top \tag{0.23}$$

όπου L ένας κάτω τριγωνικός πίνακας που υπολογίζεται με την εφαρμογή του αλγόριθμου Cholesky. Κατόπιν, με δεδομένο το διάνυσμα των σταθερών όρων $\mathbf{b}$, είναι δυνατή η επίλυση της Εξίσωσης 0.22 ως προς το διάνυσμα $\mathbf{y}$ και στη συνέχεια η εύρεση του άγνωστου διανύσματος $\mathbf{x}$ μέσω της Εξίσωσης 0.21. Τελικά, χρησιμοποιώντας τον ορισμό των $\mathbf{x}$ παραμέτρων από την Εξίσωση 0.17 υπολογίζεται το κέντρο του cluster όπως φαίνεται παρακάτω.

$$\begin{cases} x_c = -\frac{p}{2m} \\ y_c = -\frac{q}{2n} \\ \sigma_x = \frac{1}{\sqrt{2m}} \\ \sigma_y = \frac{1}{\sqrt{2n}} \\ A = \exp\left(\frac{p^2}{4m} + \frac{q^2}{4n} - k\right) \end{cases} \tag{0.24}$$

Αντίστοιχα με την τροποποίηση του πρώτου αλγορίθμου για βέλτιστη υλοποίηση στο FPGA, προτείνεται εξίσου η χρήση των σχετικών συντεταγμένων αντί των απόλυτων συντεταγμένων κάθε pixel στις παραπάνω εξισώσεις, ώστε να ελαχιστοποιηθεί η χρήση λογικών πόρων. Με αυτόν τον τρόπο αρχικά υπολογίζεται το σχετικό κέντρο κάθε cluster με αναφορά στην πρώτη γραμμή και στήλη του και στην συνέχεια με απλή πρόσθεση της θέσης του cluster μέσα στην εικόνα προκύπτουν τα ζητούμενα centroids. Αξίζει να σημειωθεί πως η διαδικασία αυτή, εν απουσία αναλυτικής μαθηματικής απόδειξης, είναι έγκυρη καθώς η υποβόσκουσα Γκαουσσιανή συνάρτησης και συνεπώς τόσο οι τιμές των pixels όσο και το κέντρο του cluster, εξαρτώνται μόνο από τις 5 προαναφερθείσες παραμέτρους και είναι ανεξάρτητες από το σύστημα συντεταγμένων.

0.4 Προτεινόμενες FPGA Αρχιτεκτονικές

0.4.1 Εισαγωγή

Βασικός στόχος της παρούσας διπλωματικής εργασίας είναι η επιτάχυνση των προαναφερθέντων αλγορίθμων κεντραρίσματος σε μία FPGA πλατφόρμα. Λαμβάνοντας υπόψιν τα ιδιαίτερα χαρακτηριστικά κάθε αλγορίθμου επιλέχθηκε διαφορετική στρατηγική σχεδιασμού για κάθε έναν, με σκοπό την ανάπτυξη βέλτιστων κυκλωμάτων και την εξαγωγή χρήσιμων συμπερασμάτων.

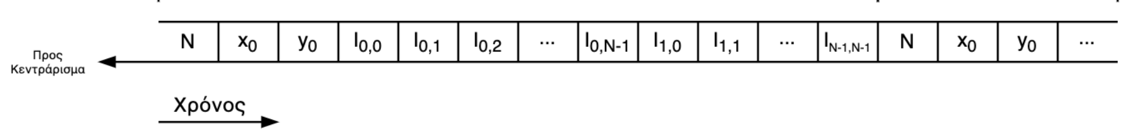
Αφενός λόγω της απλότητας του CG αλγόριθμου είναι εύκολη η μοντελοποίηση του στο Επίπεδο Μεταφοράς Καταχωρητών (Register Transfer Level) με χρήση της VHDL καθώς αυτή συνεπάγεται μικρό μέγεθος κώδικα και επιτρέπει την επίτευξη βέλτιστης υλοποίησης σε σύντομο χρονικό διάστημα. Επιπλέον καθίσταται δυνατή η σχεδίαση κυκλώματος και σε HLS χρησιμοποιώντας την C++ με στόχο την εξερεύνηση του χώρου σχεδίασης και την άμεση σύγκριση των δύο προγραμματιστικών μεθόδων.

Αφετέρου η υψηλή πολυπλοκότητα του FGF αλγόριθμου καθιστά αποδοτική την υλοποίηση του μόνο σε υψηλό επίπεδο, καθώς για την μοντελοποίηση του στο RTL θα απαιτούταν ένα ιδιαίτερα μεγάλο χρονικό διάστημα. Παρόλα αυτά η HLS προσέγγιση του συγκεκριμένου αλγορίθμου επιτρέπει την εξαντλητική εξερεύνηση του χώρου σχεδίασης ώστε να επιτευχθεί μία βέλτιστη λύση.

Για να είναι δυνατή η σχεδίαση ενός ψηφιακού κυκλώματος, σε κάθε περίπτωση είναι αναγκαία η καθιέρωση μερικών κανόνων σχετικά με την μορφή που έχουν τα δεδομένα εισόδου. Όπως έχει αναφερθεί, τα δεδομένα του centroiding συστήματος παράγονται από την διαδικασία της συσταδοποίησης, η οποία επίσης υλοποιείται στο FPGA, και θεωρείται πως μεταφέρονται μέσω ενός κατάλληλου πρωτοκόλλου υψηλής ταχύτητας.

Σύμφωνα με τις οδηγίες που παρέχει η Infinite Orbits, λόγω μίας διαδικασίας προεπεξεργασίας που λαμβάνει μέρος κατά τη διάρκεια του clustering και ονομάζεται binning, τα clusters που πρόκειται να λαμβάνει το υπό σχεδίαση σύστημα θεωρείται ότι βρίσκονται εντός εικόνων μεγέθους 1024x1024 pixels, δηλαδή η προεπεξεργασία αυτή δημιουργεί εικόνες μισής ανάλυσης σε σχέση με τις αρχικές εικόνες που παράγονται από τον αισθητήρα. Επιπρόσθετα αναμένεται πως τα ληφθέντα τετραγωνικά clusters θα έχουν ελάχιστο μέγεθος 3x3 και μέγιστο μέγεθος 5x5 pixels.

Λαμβάνοντας υπόψιν τις παραπάνω προδιαγραφές καθώς και τα ιδιαίτερα χαρακτηριστικά της εφαρμογής έχει συμπεραθεί πως ο πιο αποδοτικός τρόπος μεταφοράς της πληροφορίας, από την διαδικασία-παραγωγό (clustering) στην διαδικασία-καταναλωτή (centroiding), είναι με συνεχή μετάδοση των pixels το ένα μετά το άλλο (streaming). Μία απλουστευμένη αναπαράσταση του αντίστοιχου καναλιού μεταφοράς των δεδομένων δίνεται στο Σχήμα 0.8.



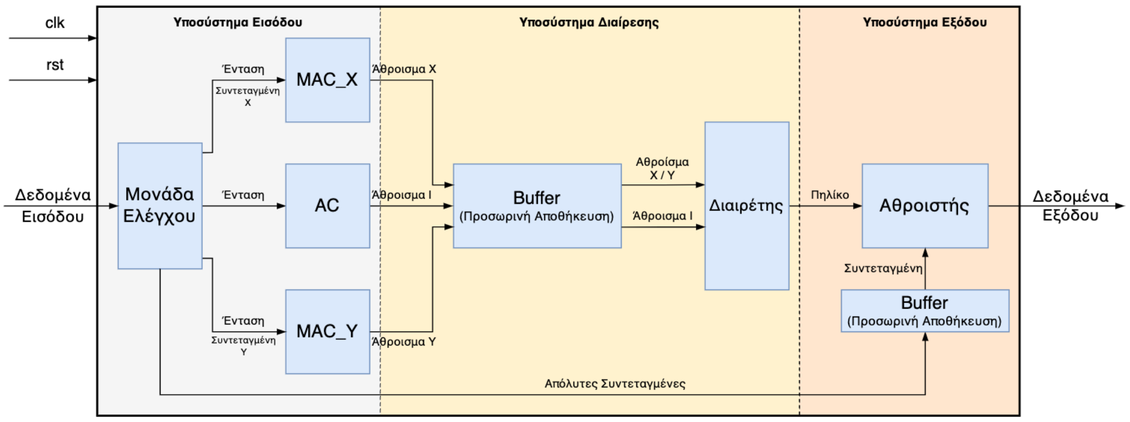
Σχήμα 0.8: Αναπαράσταση της μεταφοράς δεδομένων από το clustering σύστημα στο centroiding σύστημα.

Όπως παρατηρείται, πριν τη μετάδοση της φωτεινότητας κάθε pixel, που αποτελεί την χρήσιμη πληροφορία για την εύρεση του κέντρου κάθε αστεριού, μεταδίδεται ένα σύνολο από βοηθητικά δεδομένα. Αρχικά, λαμβάνεται η τιμή N που ενημερώνει για το μέγεθος του cluster που ακολουθεί. Αν για παράδειγμα ισχύει $N = 5$, τότε το επόμενο cluster αναμένεται να έχει μέγεθος 5x5. Στη συνέχεια παρέχονται οι τιμές x_0 και y_0 που αντιστοιχούν στις αρχικές συντεταγμένες του cluster στο επίπεδο της εικόνας, δηλαδή στην πρώτη στήλη και γραμμή του αντίστοιχα. Τελικά μεταδίδονται οι τιμές όλων των pixel ακολουθώντας την συμβατική μέθοδο προσπέλασης μιας ψηφιακής εικόνας (raster scan). Με δεδομένο πως η ένταση ενός pixel αναπαρίσταται με 12 bits, τα περισσότερα μεταξύ των μεταδιδομένων πακέτων, προκύπτει πως το κανάλι εισόδου πρέπει αντίστοιχα να έχει πλάτος 12 bits.

0.4.2 Αλγόριθμος Κέντρου Βαρύτητας

VHDL Μοντέλο

Ένα απλοποιημένο μπλοκ διάγραμμα της προτεινόμενης αρχιτεκτονικής παρουσιάζεται στο Σχήμα 0.9.



Σχήμα 0.9: Απλοποιημένο μπλοκ διάγραμμα της προτεινόμενης CG (VHDL) αρχιτεκτονικής.

Το σύστημα χωρίζεται σε 3 υποσυστήματα που συγχρονίζονται από ένα κοινό ρολόι (*clk*). Κάθε υποσύστημα περιέχει έναν αριθμό από μονάδες που πραγματοποιούν τις θεμελιώδεις πράξεις στο χαμηλότερο επίπεδο. Για την περιγραφή της ιεραρχίας του συνολικού συστήματος χρησιμοποιήθηκε δομική περιγραφή στη VHDL και για την μοντελοποίηση της ακολουθιακής λογικής στο χαμηλότερο επίπεδο υιοθετήθηκε περιγραφή συμπεριφοράς. Κατά την ανάπτυξη του συστήματος, δοκιμάστηκε ένα μεγάλο πλήθος διαφορετικών υλοποιήσεων για κάθε μονάδα. Το κριτήριο με βάση το οποίο επιλέχθηκαν οι βέλτιστες περιγραφές ήταν τα αποτελέσματα, ως προς τους πόρους και το κρίσιμο μονοπάτι, που παράγονταν από την διαδικασία της Λογικής Σύνθεσης (Logic Synthesis) σε κάθε περίπτωση.

Στο σύστημα του Σχήματος 0.9 τα δεδομένα αρχικά λαμβάνονται από το υποσύστημα εισόδου. Οι δύο μονάδες MAC (Multiplier Accumulator) και η μία μονάδα AC (Accumulator) είναι υπεύθυνες για τον υπολογισμό των δύο αριθμητών και του κοινού παρονομαστή στα δύο κλάσματα που ορίζονται από τον αλγόριθμο CG. Ως εισόδους λαμβάνουν σε κάθε κύκλο την φωτεινότητα και τις σχετικές συντεταγμένες ενός pixel. Σύμφωνα με τη βασική θεωρία δυαδικής αριθμητικής το αποτέλεσμα της πρόσθεσης M αριθμών των N ψηφίων, απαιτεί $N + \log_2 M$ ψηφία για την αναπαράσταση του. Με δεδομένο πως λόγω του μέγιστου μεγέθους των clusters οι σχετικές συντεταγμένες χρειάζονται 3 ψηφία για την ορθή αναπαράσταση τους και ο μέγιστος αριθμός pixels είναι 25, προκύπτει πως ο αριθμός N στην περίπτωση των αριθμητών ισούται με $N = 12 + 3 = 15$ ψηφία λόγω του γινόμενου και στην περίπτωση του παρονομαστή ισούται με $N = 12$ ψηφία. Επειδή ισχύει $32 > 25$, όπου 32 είναι η μικρότερη δύναμη του 2 η οποία είναι μεγαλύτερη από τον αριθμό των αθροισμάτων, προκύπτει πως για την σωστή αναπαράσταση των αριθμητών απαιτούνται $15 + \log_2 32 = 20$ ψηφία ενώ για τον παρονομαστή απαιτούνται $12 + \log_2 32 = 17$ ψηφία.

Αξίζει να σημειωθεί πως για την βέλτιστη υλοποίηση των συγκεκριμένων υπολογιστικών δομών είναι απαραίτητη η χρήση DSP μονάδων. Αντιθέτως, η Μονάδα Ελέγχου (Control Unit) που είναι υπεύθυνη για τον έλεγχο των υπόλοιπων μονάδων και της μετάδοσης της πληροφορίς, και συνεπώς είναι μία εξολοκλήρου λογική μονάδα, μοντελοποιείται ως μία Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine) και υλοποιείται με χρήση μόνο λογικών κυττάρων.

Ένα ιδιαίτερο χαρακτηριστικό της συγκεκριμένης αρχιτεκτονικής είναι η χρήση Αριθμητικής Σταθερής Υποδιαστολής (Fixed Point Arithmetic) αντι για την σημαντικά πολυπλοκότερη Αριθμητική Κινητής Υποδιαστολής (Floating Point Arithmetic), της οποίας η χρήση σε συστήματα που έχουν ως κύριο μέλημα την μεγιστοποίηση της ταχύτητας είναι συνήθως μη αποδεκτή. Αντιθέτως η Αριθμητική Σταθερής Υποδιαστολής είναι πολύ αποδοτική καθώς επιτρέπει την ολοκλήρωση πράξεων πραγματικών αριθμών κατά τάξεις μεγέθους ταχύτερα. Το χαρακτηριστικό μειονέκτημα αυτής της αριθμητικής είναι πως συνήθως οδηγεί σε απώλεια ακρίβειας και συνεπώς πρέπει να χρησιμοποιείται με προσοχή. Στη συγκεκριμένη περίπτωση, πρέπει να χρησιμοποιηθεί το κατάλληλο πλήθος ψηφίων για την αναπαράσταση του δεκαδικού μέρους ενός πραγματικού αριθμού, με στόχο την επίτευξη ικανοποιητικής ακρίβειας χωρίς όμως περιττή αύξηση της λογικής πολυπλοκότητας.
Ένας πραγματικός αριθμός με βάση την συγκεκριμένη αναπαράσταση ορίζεται όπως φαίνεται κάτωθι.

$$x = b_{I-F-1}b_{I-F-2}\dots b_1b_0.b_{-1}b_{-2}\dots b_{-F-1}b_{-F}$$

όπου η ακρίβεια καθορίζεται από το λιγότερο σημαντικό ψηφίο b_{-F}, όπου F το πλήθος των ψηφίων δεξιότερα της υποδιαστολής. Το ελάχιστο πλήθος τέτοιων ψηφίων που ικανοποιεί την προδιαγραφή σφάλματος μικρότερου από 0.1 pixels είναι $F = 4$ καθώς:

$$2^{-F} = 2^{-4} = \frac{1}{16} = 0.0625 < 0.1$$

Επειδή η πράξη που αφορά πραγματικούς αριθμούς στην συγκεκριμένη εφαρμογή είναι η διαίρεση, η υλοποίηση του πηλίκου ως αριθμό σταθερής υποδιαστολής οδηγεί σε σημαντική μείωση των ζητούμενων πόρων και των λογικών επιπέδων που απαιτούνται για αυτή την πράξη, με αποτέλεσμα την μείωση της καθυστέρησης (latency) και του κρίσιμου μονοπατιού.

Για την πράξη της διαίρεσης στην εξεταζόμενη αρχιτεκτονική επιλέχθηκε η χρήση μίας έτοιμης IP μονάδας από τον κατάλογο IP της Xilinx, ο οποίος διαθέτει ένα πλήθος υλοποιήσεων που βασίζονται σε διαφορετικούς αλγορίθμους διαίρεσης. Για τις ανάγκες του συστήματος επιλέχθηκε ο αλγόριθμος Radix-2, ο οποίος χρησιμοποιεί μόνο λογικά κύτταρα, δηλαδή Look-Up Tables (LUT) και καταχωρητές (Flip Flop).

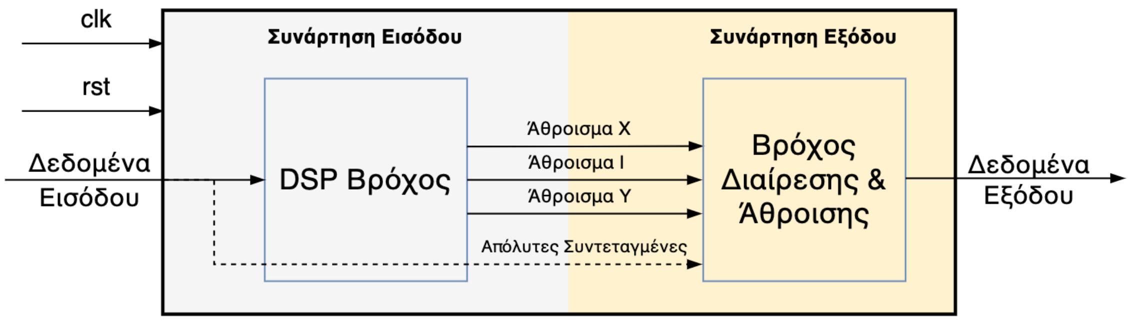
Η πιο σημαντική σχεδιαστική επιλογή αφορά τον ρυθμό διεκπαιρεωτικότητας (throughput) της συγκεκριμένης IP μονάδας, ο οποίος πρέπει να επιλεχθεί λαμβάνοντας υπόψιν τα ιδιαίτερα χαρακτηριστικά του συνολικού συστήματος. Επειδή η διαίρεση αποτελεί την πιο πολύπλοκη εκ των βασικών αριθμητικών πράξεων, αναμένεται να αποτελεί το κρίσιμο παράγοντα απόδοσης της αρχιτεκτονικής. Για αυτόν τον λόγο αρχικά επιλέχθηκε η χρήση ενός μοναδικού διαιρέτη που θα πραγματοποιεί και τις δύο διαιρέσεις που απαιτεί ο CG αλγόριθμος, μία για κάθε διάσταση. Επιπλέον, σύμφωνα με τις οδηγίσες της Infinite Orbits, η παραγωγή των clusters από το σύστημα συσταδοποίησηας αναμένεται να είναι μία σχετικά αργή διαδικασία με ελάχιστη καθυστέρηση μεταξύ διαδοχικών clusters γύρω στους 70 κύκλους. Αυτό το χαρακτηριστικό επιτρέπει την επιλογή του μικρότερου εκ των διαθέσιμων ρυθμών διεκπαιρεωτικότητας, ο οποίος επιτρέπει την εκκίνηση 1 διαίρεσης ανά 8 κύκλους ρολογιού, με σκοπό την ελαχιστοποίηση των απαιτούμενων πόρων.

Το γεγονός πως δύο διαιρέσεις θα εξυπηρετούνται από έναν κοινό διαιρέτη με σχετικά μικρή ταχύτητα δημιουργεί μία ανάγκη για σειριακή αποθήκευση και προσπέλαση των δεδομένων, κάτι που υποδηλώνει τη χρήση μίας First-In-First-Out (FIFO) μνήμης. Επειδή μάλιστα απαιτείται η ταυτόχρονη αποθήκευση δύο δεδομένων και η προσπέλαση ενός κάθε στιγμή, η χρησιμοποιούμενη μνήμη πρέπει να έχει την ικανότητα μείωσης του ρυθμού μεταφοράς δεδομένων. Αποδείχτηκε πως η ιδανική λύση σε αυτό το πρόβλημα είναι η χρήση μιας ασύμμετρης IP FIFO μνήμης ως ένα μεταβατικό αποθηκευτικό μέσο μεταξύ του υποσυστήματος εισόδου, που παράγει τους αριθμητές και τον παρονομαστή, και του διαιρέτη, που τους καταναλώνει. Σημειώνεται πως μία ίδιου τύπου μνήμη τοποθετείται μεταξύ των υποσυστημάτων εισόδου και εξόδου, καθώς το τελευταίο εν σειρά υποσύστημα είναι υπεύθυνο για την πρόσθεση των απόλυτων συντεταγμένων στα σχετικά κέντρα που έχουν υπολογιστεί, για την παραγωγή των τελικών centroids.

Ολοκληρώνοντας, έχει αξία να σημειωθεί πως η διεκπαιρεωτικότητα του συνολικού συστήματος καθορίζεται από το υποσύστημα εισόδου ενώ το κρίσιμο μονοπάτι, δηλαδή η μέγιστη συχνότητα λειτουργίας, από τον IP διαιρέτη.

C++ Μοντέλο

Ένα απλοποιημένο διάγραμμα του ισοδύναμου C++ συστήματος παρουσιάζεται στο Σχήμα 0.10.



Σχήμα 0.10: Απλοποιημένο μπλοκ διάγραμμα της προτεινόμενης CG (C++) αρχικτεκτονικής.

Στο HLS τα δομικά στοιχεία ενός συστήματος είναι οι C++ συναρτήσεις, οι οποίες αντιστοιχίζονται σε υποσυστήματα (blocks) στην RTL ιεραρχία ενώ τα ορίσματα κάθε συνάρτησης συντίθενται σε RTL θύρες. Συνεπώς το προτεινόμενο C++ σύστημα είναι παρόμοιο ως προς τη δομή του με την RTL περίπτωση. Ένα από τα σημαντικότερα πλεονεκτήματα της HLS προσέγγισης είναι το γεγονός πως ο μεταγλωτιστής εξάγει αυτόματα την λογική ελέγχου από τον πηγαίο κώδικα και δημιουργεί ένα FSM για να ταξινομήσει τις διάφορες πράξεις. Αυτή η διαδικασία είναι ισοδύναμη με την ύπαρξη της RTL Μονάδας Ελέγχου και συνεπώς δεν είναι πλέον αναγκαία η ρητή υλοποίηση της. Κατά την ανάπτυξη του συστήματος δοκιμάστηκαν διάφορες τροποποιήσεις στον πηγαίο κώδικα με σκοπό την παραγωγή του βέλτιστου κυκλώματος. Το κριτήριο αξιολόγησης αυτών των τροποποιήσεων ήταν τα αποτελέσματα που παρήγαγε η C Σύνθεση (C Synthesis) σε κάθε περίπτωση.

Η συνάρτηση εισόδου πραγματοποιεί την ίδια λειτουργία με το αντίστοιχο RTL υποσύστημα που περιγράφηκε παραπάνω, δηλαδή λαμβάνει την ροή δεδομένων εισόδου, προωθεί τις απόλυτες συντεταγμένες στη συνάρτηση εξόδου και υπολογίζει τους δύο αριθμητές και τον παρονομαστή κάθε cluster. Οι υπολογισμοί αυτοί πραγματοποιούνται εντός μίας τέλεια φωλιασμένης ιεραρχίας δύο επαναληπτικών βρόχων (loops), οι οποίο περιέχουν συνολικά τρεις εντολές. Με δεδομένο το τετραγωνικό σχήμα των clusters το όριο κάθε βρόχου ισούται με το πρώτο δεδομένο που λαμβάνεται για κάθε συστάδα, δηλαδή με την τιμή N. Παρατηρήθηκε πως ο μεταγλωττιστής υλοποιεί αυτόματα τις δύο MAC πράξεις με DSP κύτταρα. Αντιθέτως αυτό δεν κατέστη δυνατό για την AC πράξη, ούτε με την εφαρμογή των σχετικών οδηγιών (directives) που διαθέτει το HLS εργαλείο. Το γεγονός αυτό δείχνει πως μερικές φορές το εργαλείο περιορίζει την πρόσβαση του σχεδιαστή στα χαμηλότερα επίπεδα σχεδίασης.

Αναμένομενα η συνάρτηση εξόδου είναι υπεύθυνη για τον υπολογισμό των κέντρων των αστεριών, μέσω αρχικά μίας διαίρεσης για την εύρεση του σχετικού κέντρου και κατόπιν προσθέτοντας τις απόλυτες συντεταγμένες. Αντίστοιχα με την περίπτωση της RTL αρχιτεκτονικής, μία θεμελιώδης σχεδιαστική επιλογή αφορά τη χρήση ενός κοινού διαιρέτη που θα υπολογίζει και τα δύο σχετικά centroids (ως προς τις δύο διαστάσεις), με σκοπό την ελαχιστοποίηση του χώρου που καταλαμβάνεται στο FPGA. Μετά από αναλυτικό έλεγχο, αποδείχτηκε πως η χρήση μίας μονάδας διαίρεσης είναι δυνατή μόνο εάν οι δύο σχετικές C++ εντολές τοποθετηθούν εντός ενός κοινού βρόχου, ενώ η χρήση των σχετικών διαθέσιμων directives δεν οδηγούσε στο επιθυμητό αποτέλεσμα. Εξίσου σημαντική είναι η παρατήρηση πως ανεξάρτητα από το throughput του διαιρέτη που μπορεί να ορίσει ο σχεδιαστής μέσω του όρου Initiation Interval (II) του βρόχου, το HLS εργαλείο πάντοτε παράγει ένα συγκεκριμένο διαιρέτη με τη μέγιστη δυνατή διεκπαιρεωτικότητα, ο οποίος δηλαδή εκκινεί μία νέα διαίρεση ανά κύκλο ρολογιού. Αυτό καταδεικνύει πως ο μεταγλωττιστής δεν επιτρέπει καμία τροποποίηση του διαιρέτη από τον χρήστη, σε αντίθεση με την περίπτωση του VHDL μοντέλου όπου ήταν δυνατή η εκτενής παραμετροποίησή του. Αξίζει να σημειωθεί πως η συγκεκριμένη μονάδα υλοποιείται χρησιμοποιώντας μονάχα LUTs και FFs. Επίσης, τονίζεται πως οι βρόχοι και των δύο συναρτήσεων έχουν καταστεί πλήρως διοχετευμένοι μέσω της εφαρμογής της οδηγίας *Pipeline*.

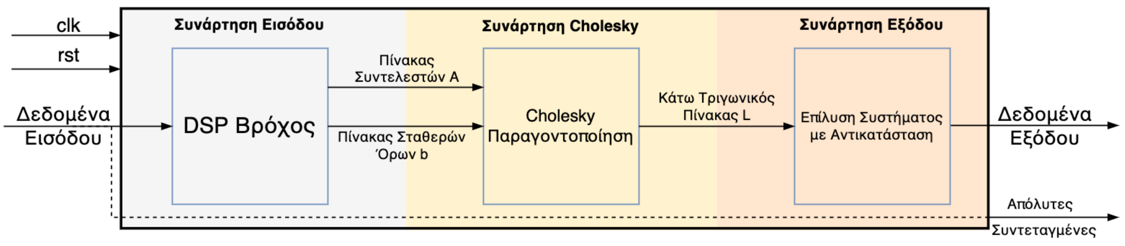
Σε μία C++ αρχιτεκτονική η διεπαφή μεταξύ των διάφορων συναντήσεων είναι κρίσιμη για την συνολική απόδοση του συστήματος και συνεπώς πρέπει να βελτιστοποιείται. Αυτό επετεύχθη με τη χρήση της ισχυρής *Dataflow* οδηγίας, η οποία δημιουργεί μία πολύπλοκη δομή επικοινωνίας που επιτρέπει στις σχετικές συναρτήσεις να εκτελούνται παράλληλα. Στο προτεινόμενο σύστημα, αυτή η δομή αποτελείται από 5 FIFO κανάλια που μεταφέρουν τα αποτελέσματα της Συνάρτησης Εισόδου. Είναι προφανές πως η εφαρμογή του συγκεκριμένου directive υποκαθιστά την ρητή μοντελοποίηση των ενδιάμεσων καναλιών, που είναι απαραίτητη κατά την σχεδίαση στο επίπεδο των καταχωρητών. Ωστόσο οδηγεί σε σημαντική αύξηση των απαιτούμενων πόρων του FPGA, οι οποίοι χρησιμοποιούνται για την υλοποίηση κυρίως μνημών και πολυπλεκτών. Σημειώνεται πως στη συγκεκριμένη περίπτωση τα κανάλια επικοινωνίας υλοποιούνται με Σειριακούς Καταχωρητές LUTs (Shift Register LUTs) και ορίζονται με το μικρότερο δυνατό μέγεθος των 2 θέσεων μνήμης, επειδή τα δεδομένα καταναλώνονται από την συνάρτηση εξόδου έναν κύκλο μετά την παραγωγή τους και συνεπώς τα κανάλια δεν γεμίζουν ποτέ.

Ολοκληρώνοντας, αξίζει να τονιστεί πως παρόμοια με την RTL αρχιτεκτονική, η διεκπαιρεωτικότητα του συνολικού συστήματος καθορίζεται από το υποσύστημα εισόδου ενώ το κρίσιμο μονοπάτι, δηλαδή η μέγιστη συχνότητα λειτουργίας, από τη μονάδα διαίρεσης.

0.4.3 Αλγόριθμος Γρήγορου Γκαουσσιανού Ταιριάσματος

Η καινοτόμα FPGA υλοποίηση που προτείνεται σε αυτή την διπλωματική εργασία, βασίζεται στον αλγόριθμο Γρήγορου Γκαουσσιανού Ταιριάσματος με στόχο την επίτευξη μεγάλης ακρίβειας και ταχύτητας. Όπως εξηγήθηκε, λόγω της μεγάλης πολυπλοκότητας του συγκεκριμένου αλγορίθμου θεωρήθηκε προτιμότερη η μοντελοποίηση του συστήματος σε C++, καθώς το HLS μειώνει τον απαιτούμενο χρόνο σχεδίασης και παρέχει τη δυνατότητα για γρήγορη επαλήθευση και εκτεταμένη εξερεύνηση του χώρου σχεδίασης, διευκολύνοντας έτσι την εύρεση μίας βέλτιστης λύσης. Ένα απλοποιημένο μπλοκ διάγραμμα της προτεινόμενης FGF αρχιτεκτονικής δίνεται στο Σχήμα 0.11.

Όπως μπορεί κανείς να παρατηρήσε, μία βασική διαφορά της συγκεκριμένης αρχιτεκτονικής σε σχέση με τις προηγούμενες είναι πως υπολογίζει μόνο το σχετικό και όχι το απόλυτο κέντρο του cluster, με αναφορά δηλαδή στην πρώτη γραμμή και στήλη. Η εξήγηση αυτής της σχεδιαστικής επιλογής θα δοθεί στη συνέχεια. Πρέπει να σημειωθεί, πως κατά την ολοκλήρωση των διάφορων συστημάτων (clustering, centroiding, matching) για την δημιουργία ενός ολοκληρωμένου ανιχνευτή αστεριών, σε αυτή την περίπτωση είναι απαραίτητη μία μία μικρή τροποποίηση στον κώδικα της διαδικασίας ταιριάσματος, που υλοποιείται στην πλευρά του επεξεργαστή, ώστε να πραγματοποιείται η πρόσθεση των



Σχήμα 0.11: Απλοποιημένο μπλοκ διάγραμμα της προτεινόμενης FGF αρχιτεκτονικής.

απόλυτων συντεταγμένων με τα σχετικά κέντρα για την εύρεση των τελικών centroids.

Μία θεμελιώδης διαφορά μεταξύ των υλοποιήσεων στο υλικό των δύο αλγορίθμων κεντραρίσματος αφορά την χρήση Αριθμητικής Κινητής Υποδιαστολής για την περίπτωση του FGF. Χονδρικά ο FGF αλγόριθμος εκφράζεται από τις Εξισώσεις 0.18, 0.23, 0.20 οι οποίες περιέχουν ένα τεράστιο αριθμό πράξεων με πραγματικούς αριθμούς. Για παράδειγμα μονάχα το δεξί μέρος της Εξίσωσης 0.18 ορίζει έναν λογάριθμο, έναν πολλαπλασιασμό και μία αφαίρεση για κάθε pixel. Για να είναι αποδοτική η χρήση Αριθμητικής Σταθερής Υποδιαστολής είναι αναγκαίος ο καθορισμός του μεγέθους, δηλαδή του αριθμού των ψηφίων, καθενός από τα ενδιάμεσα σήματα, κάτι το οποίο είναι πρακτικά αδύνατο. Αντιθέτως, η χρήση της Floating Point αναπαράστασης είναι ιδιαίτερα εύκολη και παραγωγική κατά τον προγραμματισμό με C++, καθώς οι πράξεις και οι απαιτούμενες μετατροπές αντιστοιχίζονται αυτόματα σε κατάλληλα Floating Point IP κύτταρα που διαθέτει η Xilinx.

Επιπλέον η συγκεκριμένη αριθμητική διαθέτει το πλεονέκτημα γρήγορης και επακριβούς επεξεργασίας τόσο μικρών όσο και μεγάλων αριθμών, κάτι το οποίο είναι ιδιαίτερα σημαντικό για την παρούσα εφαρμογή, όπου απαιτείται ο υπολογισμός ακέραιων αριθμών πολλών δυαδικών ψηφίων όπως θα φανεί παρακάτω. Ωστόσο, το μειονέκτημα της χρήσης αριθμών κινητής υποδιαστολής είναι η σχετικά μεγάλη πολυπλοκότητα τους, η οποία αναμένεται να οδηγήσει σε μεγάλες υπολογιστικές απαιτήσει που μεταφράζονται σε μεγαλύτερα και πιο αργά ψηφιακά κυκλώματα.

Ένα βασικό χαρακτηριστικό του συγκεκριμένου αριθμητικού συστήματος είναι πως μπορεί να αναπαριστά ένα μεγάλο εύρος αριθμών με αντίτιμο μία μείωση της ακρίβειας. Ειδικότερα, οι διαθέσιμοι FP αριθμοί δεν είναι ομοιόμορφα κατανεμημένοι καθώς η διαφορά μεταξύ διαδοχικών αριθμών που αναπαρίστανται ακριβώς εξαρτάται από την εκάστοτε κλίμακα στην οποία βρίσκονται. Έτσι σε μικρότερες κλίμακες αναπαρίστανται ακριβώς περισσότεροι αριθμοί συγκριτικά με τις μεγαλύτερες κλίμακες. Επειδή λοιπόν οι μικροί αριθμοί διατηρούν μεγάλη ακρίβεια στο δεκαδικό τους μέρος, επιλέχθηκε ο υπολογισμός μόνο των σχετικών centroids ενός cluster, τα οποία είναι μικροί αριθμοί ενός ακέραιου ψηφίου. Αντιθέτως η πρόσθεση των μεγάλων απόλυτων συντεταγμένων θα οδηγούσε σε απώλεια ακρίβειας στο κρίσιμο κλασματικό μέρος. Η πρόσθεση αυτή ωστόσο, μπορεί εύκολα να πραγματοποιηθεί στην πλευρά του επεξεργαστή όπου συνήθως χρησιμοποιείται αριθμητική διπλής ακρίβειας και είναι δυνατή η διατήρηση της επιθυμητής ακρίβειας.

Η πρώτη συνάρτηση του Σχήματος 0.11 είναι υπεύθυνη για τον υπολογισμό του πίνακα συντελεστών A και του διανύσματος σταθερών όρων $\mathbf{b}$, στοιχεία τα οποία ορίζουν τις κανονικές εξισώσεις (Εξίσωση 0.18). Αυτό επιτυγχάνεται με την χρήση ενός, αντίστοιχου με την CG υλοποίηση, διπλού επαναληπτικού βρόχου που περιέχει διαφορετικές πλέον εντολές.

Το πρώτο μέρος των εντολών αφορούν τον υπολογισμό του συμμετρικού πίνακα A, του οποίου το κάτω τριγωνικό μέρος δίνεται παρακάτω.

$$A = \begin{bmatrix} \sum am_i^2 & & & & \\ \sum am_i an_i & \sum an_i^2 & & & \\ \sum am_i ap_i & \sum ap_i aq_i & \sum ap_i^2 & & \\ \sum am_i aq_i & \sum an_i aq_i & \sum ap_i aq_i & \sum aq_i^2 & \\ \sum am_i ak_i & \sum an_i ak_i & \sum ap_i ak_i & \sum aq_i ak_i & \sum ak_i^2 \end{bmatrix} \tag{0.25}$$

Όπως φαίνεται, για κάθε συντελεστή απαιτείται ένας αριθμός πράξεων που περιλαμβάνουν τα δεδομένα εισόδου. Το πλήθος των bits που απαιτούνται για την αναπαράσταση κάθε συντελεστή προκύπτει με εφαρμογή απλής θεωρίας δυαδικών αριθμών. Ο πίνακας 0.1 περιέχει τα συγκεντρωτικά στοιχεία για τους 13 μοναδικούς συντελεστές του πίνακα A.

Πίνακας 0.1: Υπολογισμός του πλήθους ψηφίων για κάθε ακέραιο συντελεστή a_{ij}

Όρος	**Μορφή**	**Πλήθος Ψηφίων**	**Συντελεστής**	**Πλήθος Ψηφίων**
am_i^2	$I_i^2 x_i^4$	36	a_{11}	41
an_i^2	$I_i^2 y_i^4$	36	a_{22}	41
$am_i an_i$	$I_i^2 x_i^2 y_i^2$	36	a_{12}	41
$am_i ap_i$	$I_i^2 x_i^3$	33	a_{31}	38
$am_i aq_i$	$I_i^2 x_i^2 y_i$	33	a_{41}	38
$an_i ap_i$	$I_i^2 y_i^2 x_i$	33	a_{32}	38
$an_i aq_i$	$I_i^2 y_i^3$	33	a_{42}	38
ap_i^2	$I_i^2 x_i^2$	30	a_{33}	35
aq_i^2	$I_i^2 y_i^2$	30	a_{44}	35
$ap_i aq_i$	$I_i^2 x_i y_i$	30	a_{43}	35
$ap_i ak_i$	$I_i^2 x_i$	27	a_{53}	32
$aq_i ak_i$	$I_i^2 y_i$	27	a_{54}	32
ak_i^2	I_i^2	24	a_{55}	29

Η πιο σημαντική πρόκληση που αντιμετωπίστηκε κατά την σχεδίαση της συνάρτησης εισόδου αφορούσε τον τρόπο υπολογισμού κάθε συντελεστή στο FPGA. Αυτό μπορεί να γίνει καλύτερα κατανοητό με ένα παράδειγμα. Έστω ο συντελεστής $a_{53} = \sum I_i^2 x_i$, για τον οποίο απαιτείται μία MAC πράξη για κάθε pixel. Το γινόμενο $I_i^2 x_i$ μπορεί να υπολογιστεί με δύο διαφορετικούς τρόπους, είτε ως $I_i^2 \cdot x_i$ ή ως $(I_i x_i) \cdot I_i$. Αν και σε έναν συμβατικό επεξεργαστή η επιλογή μεταξύ των δύο μορφών είναι ασήμαντη, αυτό δεν ισχύει στην ψηφιακή σχεδίαση. Πιο συγκεκριμένα, παρατηρήθηκε πως ο HLS μεταγλωττιστής υλοποιεί το πρώτο γινόμενο, 24x3 ψηφίων, με LUTs, και το δεύτερο γινόμενο, 15x12 ψήφιων, με κύτταρα DSP.

Αν και σε κάποιες περιπτώσεις ο τρόπος υλοποίησης τέτοιων γινομένων δεν έχει μεγάλη σημασία, αποδείχτηκε πως στην υπό εξέταση εφαρμογή η υλοποίηση με χρήση LUTs οδηγούσε σε αύξηση του κρίσιμου μονοπατιού και μείωση της απόδοσης. Για τον λόγο αυτό μετά από εξαντλητική εξερεύνηση του χώρου σχεδίασης, κατά την οποία ελέγχθηκαν όλοι οι δυνατοί (31) τρόποι υπολογισμού των 13 συντελεστών, επιλέχθηκε η υλοποίηση τους με μονάδες DSP.

Μία συνοπτική περιγραφή του ακριβούς τρόπου υλοποίησης των συντελεστών δίνεται στον Πίνακα

0.2. Για ευκολία ορίζεται η *τάξη* των συντελεστών ίση με την δύναμη των όρων των συντεταγμένων ενώ και οι δύο συντεταγμένες x_i, y_i θεωρούνται ισοδύναμες και σημειώνονται με $\mathbf{x}_i$.

Πίνακας 0.2: Ορισμός κάθε τύπου συντελεστών, με βάση την μορφή του πολλαπλασιασταίου και του πολλαπλασιαστή.

Τάξη	**Τύπος**	**Πολλαπλασιασμός**	**Πλήθος Ψηφίων**	**Βοηθητικός Όρος**
4	$I_i^2\mathbf{x}_i^4$	$(I_i\mathbf{x}_i^2)(I_i\mathbf{x}_i^2)$	18x18	$I_i\mathbf{x}_i^2$, $I_i\mathbf{x}_i$
3	$I_i^2\mathbf{x}_i^3$	$(I_i\mathbf{x}_i^2)(I_i\mathbf{x}_i)$	18x15	$I_i\mathbf{x}_i^2$
2	$I_i^2\mathbf{x}_i^2$	$(I_i\mathbf{x}_i)(I_i\mathbf{x}_i)$	15x15	$I_i\mathbf{x}_i$
1	$I_i^2\mathbf{x}_i$	$(I_i\mathbf{x}_i)(I_i)$	15x12	$I_i\mathbf{x}_i$

Το δεύτερο μέρος των εντολών του βρόχου στην συνάρτηση εισόδου αφορά τον υπολογισμό των σταθερών όρων b_i, οι οποίοι δίνονται σε μορφή πίνακα παρακάτω.

$$\mathbf{b} = \begin{bmatrix} -\sum am_i a_i \\ -\sum an_i a_i \\ -\sum ap_i a_i \\ -\sum aq_i a_i \\ -\sum ak_i ai_i \end{bmatrix} = \begin{bmatrix} -\sum (I_i^2 x_i^2) \ln I_i \\ -\sum (I_i^2 y_i^2) \ln I_i \\ -\sum (I_i^2 x_i) \ln I_i \\ -\sum (I_i^2 y_i) \ln I_i \\ -\sum (I_i^2) \ln I_i \end{bmatrix} \tag{0.26}$$

Είναι φανερό πως ο λογάριθμος αποτελεί την πρώτη πράξη πραγματικών αριθμών του FGF συστήματος και πραγματοποιείται με ένα ειδικό κύτταρο απλής ακρίβειας κινητής υποδιαστολής που υλοποιείται με 13 DSP. Τόσο αυτό το κύτταρο όσο και η μονάδα μετατροπής σε αριθμούς κινητής υποδιαστολής των ακέραιων όρων που βρίσκονται εντός παρένθεσης, η οποία χρησιμοποιεί μόνο LUTs και FFs, παράγονται αυτόματα από το εργαλείο. Αντίστοιχοι υπολογιστικοί πυρήνες που υλοποιούνται με DSP εκτελούν τις πράξεις του πολλαπλασιασμού και της αφαίρεσης. Σημειώνεται πως όλοι οι πυρήνες είναι πλήρως διοχετευμένοι και συνεπώς όλες οι πράξεις εξυπηρετούνται από ένα μοναδικό αντίγραφο.

Αυτό που έχει ιδιαίτερη σημασία είναι το γεγονός πως τόσο η καθυστέρηση (latency) όσο και η διεκπαιρεωτικότητα (throughput) της συνάρτησης εισόδου καθορίζονται από τις πράξεις κινητής υποδιαστολής. Αφενός, το latency κάθε επανάληψης του βρόχου ορίζεται από την ακολουθία των πράξεων που απαιτείται για κάθε pixel στους όρους b_i. Αυτή η ακολουθία αποτελείται από μία μετατροπή ακέραιου σε πραγματικό αριθμό, έναν λογάριθμο, έναν πολλαπλασιασμό και μία αφαίρεση. Αφετέρου, το Initiation Interval του βρόχου περιορίζεται από τη μονάδα αφαίρεσης κινητής υποδιαστολής. Συμπερεσματικά, η υιοθέτηση αριθμητικής κινητής υποδιαστολής περιορίζει την απόδοση της συνάρτησης εισόδου και ακολούθως του συνολικού συστήματος, ενώ ταυτόχρονα καθορίζει και την συνολική κατανάλωση πόρων στο FPGA.

Η δεύτερη συνάρτηση του μπλοκ διαγράμματος του Σχήματος 0.11 είναι υπεύθυνη για την παραγοντοποίηση του πίνακα συντελεστών A σύμφωνα με την Εξίσωση 0.23. Αυτό επιτυγχάνεται με χρήση του Αλγόριθμου Cholesky που περιγράφεται από τους παρακάτω τύπους.

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \tag{0.27}$$

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) \tag{0.28}$$

όπου l_{ii}, l_{ij} είναι τα στοιχεία εντός και εκτός της κύριας διαγωνίου αντίστοιχα.

Αν και το HLS εργαλείο παρέχει διάφορες υλοποιήσεις του συγκεκριμένου αλγορίθμου μέσω της βιβλιοθήκης γραμμικής άλγεβρας, αποδείχτηκε πως όλες υλοποιούνταν ως επαναληπτικοί αλγόριθμοι που χρησιμοποιούσαν κάποια ιεραρχία επαναληπτικών βρόχων. Έτσι οι διαθέσιμες αρχιτεκτονικές δεν περιείχαν το χαρακτηριστικό της διοχέτευσης και παρουσίαζαν μεγάλη τιμή II, με συνέπεια η χρήση τους να κρίνεται μη αποδοτική. Για αυτό το λόγο, επιλέχθηκε η αποσύνθεση αυτών των βρόχων με χρήση της οδηγίας *Pipeline* και η δημιουργία μίας ξεδιπλωμένης διοχετευμένης συνάρτησης. Αυτή η συνάρτηση εκτελεί τις πράξεις που ορίζονται από τις Εξισώσεις 0.27, 0.28 χρησιμοποιώντας την τεχνική της διοχέτευσης με σκοπό τον υπολογισμό του κάτω τριγωνικού πίνακα L, που δίνεται παρακάτω.

$$\begin{array}{lllll}
\mathbf{l_{11}} = \sqrt{a_{11}} & & & & \\
\mathbf{l_{21}} = \frac{a_{21}}{\mathbf{l_{11}}} & \mathbf{l_{22}} = \sqrt{a_{22} - \mathbf{l_{21}}^2} & & & \\
l_{31} = \frac{a_{31}}{l_{11}} & \mathbf{l_{32}} = \frac{a_{32} - l_{21}l_{31}}{\mathbf{l_{22}}} & \mathbf{l_{33}} = \sqrt{a_{33} - l_{31}^2 - \mathbf{l_{32}}^2} & & \\
l_{41} = \frac{a_{41}}{l_{11}} & l_{42} = \frac{a_{42} - l_{21}l_{41}}{l_{22}} & \mathbf{l_{43}} = \frac{a_{43} - l_{31}l_{41} - l_{32}l_{42}}{\mathbf{l_{33}}} & \mathbf{l_{44}} = \sqrt{a_{44} - l_{41}^2 - l_{42}^2 - \mathbf{l_{43}}^2} & \\
l_{51} = \frac{a_{51}}{l_{11}} & l_{52} = \frac{a_{52} - l_{21}l_{51}}{l_{22}} & l_{53} = \frac{a_{53} - l_{31}l_{51} - l_{32}l_{52}}{l_{33}} & \mathbf{l_{54}} = \frac{a_{54} - l_{41}l_{51} - l_{42}l_{52} - l_{43}l_{53}}{\mathbf{l_{44}}} & \mathbf{l_{55}} = \sqrt{a_{55} - l_{51}^2 - l_{52}^2 - l_{53}^2 - \mathbf{l_{54}}^2}
\end{array}$$

Αυτό που αξίζει να τονιστεί είναι πως παρά την χρήση διοχέτευσης, η ταχύτητα της συνάρτησης περιορίζεται από τις έμφυτες εξαρτήσεις που εμφανίζει ο αλγόριθμος Cholesky. Αυτές οι εξαρτήσεις αφορούν τους όρους που έχουν σημειωθεί με έντονη γραφή. Είναι φανερό πως για τον υπολογισμό κάθε διαγώνιου στοιχείου απαιτείται να έχει υπολογισθεί προηγουμένως το ακριβώς πιο αριστερό στοιχείο, το οποίο με τη σειρά του εξαρτάται από το διαγώνιο στοιχείο της δικής του στήλης. Συνεπώς δημιουργείται μία αλυσίδα εξαρτήσεων που ξεκινάει από το πρώτο στοιχείο l_{11}, καταλήγει στο τελευταίο στοιχείο l_{55} και καθορίζει το συνολικό latency της συνάρτησης.

Αυτή η αλύσιδα αποτελείται από 3 πράξεις πραγματικών αριθμών που επαναλαμβάνονται για κάθε στήλη, μία αφαίρεση για τον υπολογισμό του υπόριζου των διαγώνιων στοιχείων, μία τετραγωνική ρίζα για τον εύρεση του διαγώνιου στοιχείου και μία διαίρεση για τον υπολογισμού του στοιχείου της επόμενης σειράς. Για την μείωση της καθυστέρησης, αντί για την χρονοβόρα ακολουθία μίας τετραγωνικής ρίζας και μίας διαίρεσης που απαιτείται για την εύρεση των στοιχείων κάτω από τη διαγώνιο, επιλέχθηκε ο υπολογισμός τους μέσω πολλαπλασιασμού με την αντίστροφη τετραγωνική ρίζα, η οποία υπολογίζεται ευκολότερα με χρήση μίας ειδικής μονάδας κινητής υποδιαστολής που διάθετει το HLS εργαλείο. Αυτή η σχεδιαστική επιλογή οδηγεί σε μείωση της καθυστέρησης κατά 140 κύκλους, ωστόσο απαιτεί τη χρήση επιπλέον 9 DSP. Τέλος υπογραμμίζεται πως επειδή όλες οι υπολογιστικές μονάδες κινητής υποδιαστολής είναι πλήρως διοχετευμένες, απαιτείται η χρήση μονάχα μίας για κάθε πράξη.

Όπως είναι αναμενόμενο, η συνάρτηση εξόδου είναι υπεύθυνη για την επίλυση των δύο απλών γραμμικών προβλημάτων που ορίζονται από τις Εξισώσεις 0.21, 0.22, με απλή αντικατάσταση. Κάτωθι παρουσιάζονται και οι δύο εξισώσεις σε μορφή πινάκων.

$$\begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & l_{51} \\ & l_{22} & l_{32} & l_{42} & l_{52} \\ & & l_{33} & l_{43} & l_{53} \\ & & & l_{44} & l_{54} \\ & & & & l_{55} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \tag{0.29}$$

$$\begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ l_{41} & l_{42} & l_{43} & l_{44} & \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} \tag{0.30}$$

Κατά τα γνωστά, επιλύοντας πρώτα την δεύτερη εξίσωση προκύπτουν οι παρακάτω δύο ακολουθίες πράξεων με στόχο τον υπολογισμό των άγνωστων παραμέτρων x_i.

$$y_1 = \frac{b_1}{L_{11}} \tag{0.31α'}$$

$$y_2 = \frac{b_2 - L_{21}\mathbf{y_1}}{L_{22}} \tag{0.31β'}$$

$$y_3 = \frac{b_3 - L_{31}y_1 - L_{32}\mathbf{y_2}}{L_{33}} \tag{0.31γ'}$$

$$y_4 = \frac{b_4 - L_{41}y_1 - L_{42}y_2 - L_{43}\mathbf{y_3}}{L_{44}} \tag{0.31δ'}$$

$$y_5 = \frac{b_5 - L_{51}y_1 - L_{52}y_2 - L_{53}y_3 - L_{54}\mathbf{y_4}}{L_{55}} \tag{0.31ε'}$$

$$x_5 = \frac{\mathbf{y_5}}{L_{55}} \tag{0.32α'}$$

$$x_4 = \frac{y_4 - L_{45}\mathbf{x_5}}{L_{44}} \tag{0.32β'}$$

$$x_3 = \frac{y_3 - L_{35}x_5 - L_{34}\mathbf{x_4}}{L_{33}} \tag{0.32γ'}$$

$$x_2 = \frac{y_2 - L_{25}x_5 - L_{24}x_4 - L_{23}\mathbf{x_3}}{L_{22}} \tag{0.32δ'}$$

$$x_1 = \frac{y_1 - L_{15}x_5 - L_{14}x_4 - L_{13}x_3 - L_{12}\mathbf{x_2}}{L_{11}} \tag{0.32ε'}$$

Με απλή παρατήρηση των παραπάνω εξισώσεων είναι φανερό πως κάθε όρος αφού υπολογιστεί, χρησιμοποιείται στον αριθμητή του επόμενου όρου. Αυτή η κατάσταση καταδεικνύει μία έμφυτη αλυσίδα εξαρτήσεων που περιορίζει την ταχύτητα της συνολικής συνάρτησης. Για κάθε όρο y_i ή x_i η συγκεκριμένη εξάρτηση ορίζεται από έναν πολλαπλασιασμό, μία αφαίρεση και μία διαίρεση. Η αλυσίδα εξαρτήσεων που καθορίζει την συνολική καθυστέρηση της συνάρτησης αποτελείται από 11 διαιρέσεις, 8 πολλαπλασιασμούς και 8 αφαιρέσεις. Σημειώνεται πως επειδή όλοι οι υπολογιστικοί πυρήνες είναι πλήρως διοχετευμένοι, ένα αντίγραφο καθενός εξυπηρετεί όλες τις πράξεις. Τελικά, τα σχετικά centroids βρίσκονται με την εκτέλεση των παρακάτω πράξεων.

$$y_c = -\frac{x_4}{2x_2} \tag{0.33α'}$$

$$x_c = -\frac{x_3}{2x_1} \tag{0.33β'}$$

Συνοψίζοντας, παρατηρήθηκε πως η συνάρτηση εισόδου καθορίζει την διεκπαιρεωτικότητα του συνολικού συστήματος και συνεπώς ο ρυθμός (II) των ακόλουθων συναρτήσεων επιλέχθηκε ίσος με αυτής. Η συνολική καθυστέρηση του συστήματος καθορίζεται από το άθροισμα των επιμέρους καθυστερήσεων. Την μεγαλύτερη εξ αυτών την εμφανίζει η συνάρτηση εξόδου λόγω της μεγάλης αλυσίδας εξαρτήσεων που εμφανίζεται κατά την επίλυση των γραμμικών συστημάτων με απλή αντικατάσταση. Σε κάθε περίπτωση, η επίδοση του FGF συστήματος περιορίζεται από την χρήση της Αριθμητικής Κινητής Υποδιαστολής.

0.5 Αξιολόγηση Αρχιτεκτονικών

Για την αξιολόγηση των αρχιτεκτονικών κεντραρίσματος που περιγράφηκαν είναι απαραίτητη η διεξαγωγή ενός συνόλου πειραμάτων. Ειδικότερα, χρησιμοποιήθηκαν τεχνητά δεδομένα αστεριών (clusters) με στόχο την σύγκριση των FPGA υλοποιήσεων με τους αντίστοιχους αλγορίθμους στο λογισμικό, ως προς την ακρίβεια και τον χρόνο εκτέλεσης. Επιπρόσθετα ελέγχθηκε η κατανάλωση πόρων στο FPGA από κάθε αρχιτεκτονική.

Όλα τα πειράματα σε επίπεδο λογισμικού πραγματοποιήθηκαν με χρήση της MATLAB 2016A σε διπύρηνο Intel Core i5 2.6 GHz επεξεργαστή. Για την ανάπτυξη των VHDL και C++ κυκλωμάτων στο υλικό χρησιμοποιήθηκαν τα εργαλεία Xilinx Vivado Design Suite 2019.1 και Xilinx Vivado HLS 2019.1 αντίστοιχα. Όλα τα μοντέλα υλοποιήθηκαν στο Xilinx ZYNQ-7020 SoC που ενσωματώνεται στην Αναπτυξιακή Πλακέτα ZedBoard.

Το τεχνητό σύνολο δεδομένων δημιουργήθηκε με χρήση μίας Γκαουσσιανής PSF που αναπτύχθηκε σε συνεργασία με την Infinite Orbits. Οι παράμετροι της συνάρτησης επιλέχθηκαν με τέτοιον τρόπο ώστε να δημιουργεί clusters, στα οποία όλοι οι αλγόριθμοι παρουσιάζουν καλά αποτελέσματα, καθώς στόχος είναι η σύγκριση μεταξύ των υλοποιήσεων σε software και hardware και όχι η αξιολόγηση των ίδιων των αλγορίθμων. Τα τεχνητά δεδομένα αποτελούνται από τρία διαφορετικά σύνολα 10000 δειγμάτων, βάση των οποίων θα παρουσιαστούν τα πειραματικά αποτελέσματα. Το πρώτο σύνολο περιέχει clusters μεγέθους 3x3, το δεύτερο clusters μεγέθους 5x5 και το τρίτο περιέχει clusters και των δύο τύπων.

0.5.1 Αξιολόγηση Ακρίβειας

Ένα σύνηθες μειονέκτημα κατά την υλοποίηση αλγορίθμων στο υλικό είναι η απώλεια ακρίβειας. Για την εκτέλεση των πειραμάτων ακρίβειας χρησιμοποιήθηκε η μετρική της Ευκλείδειας Απόστασης μεταξύ των κέντρων που υπολογίζονται στο λογισμικό (x_s, y_s) και των αντίστοιχων εκτιμήσεων στο υλικό (x_h, y_h). Ο τύπος της μετρικής σφάλματος δίνεται κάτωθι.

$$e = \sqrt{(x_h - x_s)^2 + (y_h - y_s)^2} \tag{0.34}$$

Το μέσο σφάλμα κεντραρίσματος μεταξύ λογισμικού και υλικού για κάθε σύνολο δεδομένων δίνεται στον Πίνακα 0.3.

Πίνακας 0.3: Μέσο σφάλμα κεντραρίσματος μεταξύ υλικού και λογισμικού σε κάθε σύνολο δεδομένων.

	Δεδομένα		
Αλγόριθμος	3x3	5x5	Μεικτά
CG	0.047814	0.048162	0.047990
FGF	0.000004	0.000020	0.000005

Είναι προφανές πως η ακρίβεια σε subpixel επίπεδο συνάδει με τις σχεδιαστικές επιλογές που έχουν γίνει. Αφενός, το σφάλμα στην περίπτωση του FGF αλγόριθμου είναι αμελητέο λόγω της υιοθέτισης της αριθμητικής κινητής υποδιαστολής απλής ακρίβειας. Αφετέρου, το σφάλμα της CG υλοποίησης στο FPGA είναι αυξημένο λόγω της απλούστερης αριθμητικής σταθερής υποδιαστολής. Παρόλα αυτά, η χρήση 4 δυαδικών ψηφίων για την αναπαράσταση του μέρους δεξιότερα της υποδιαστολής εξασφαλίζει πως σε κάθε περίπτωση η απόκλιση μεταξύ των υλοποιήσεων σε υλικό και λογισμικό δεν θα ξεπερνά τα 0.1 pixels. Εάν χρησιμοποιούνταν περισσότερα δυαδικά ψηφία, το σφάλμα στον Πίνακα 0.3 θα μειωνόταν.

Παρακάτω δίνονται τα αποτελέσματα όπως προκύπτουν για κάθε αλγόριθμο στο software και στο hardware για 4 τυχαία δείγματα από το σύνολο δεδομένων.

Πίνακας 0.4: Εντοπισμένα κέντρα από τις υλοποιήσεις του αλγόριθμου CG στο λογισμικό και στο υλικό.

Πραγματική Τιμή		**CG SW**		**CG HW**	
X	Y	X	Y	X	Y
210.593130	899.672780	210.846683	899.876934	210.8125	899.875
945.137136	729.932595	945.352640	729.947333	945.3125	729.9375
135.510762	829.973985	135.816945	829.990032	135.8125	829.937
753.193434	791.639461	753.391853	791.720189	753.375	791.6875

Πίνακας 0.5: Εντοπισμένα κέντρα από τις υλοποιήσεις του αλγόριθμου FGF στο λογισμικό και στο υλικό.

Πραγματική Τιμή		**FGF SW**		**FGF HW**	
X	Y	X	Y	X	Y
210.593130	899.672780	210.592542	899.673227	210.592539	899.673221
945.137136	729.932595	945.137295	729.932511	945.137295	729.932514
135.510762	829.973985	135.511087	829.973674	135.511093	829.973673
753.193434	791.639461	753.193277	791.639544	753.193274	791.639544

Είναι προφανές πως ο αλγόριθμος FGF είναι κυρίαρχος όσον αφορά την ακρίβεια. Επιπλέον φαίνεται ότι ο αλγόριθμος CG αδυνατεί να ικανοποιήσει τις προδιαγραφές ακόμα και στο software καθώς παρουσιάζει σφάλμα μεγαλύτερο από 0.1 pixels στα τεχνητά δεδομένα. Αυτό σημαίνει πως ακόμα και η επιλογή περισσότερων δυαδικών ψηφίων για την αναπαράσταση του κλασματικού μέρους των centroids στο υλικό, θα οδηγούσε σε μη ικανοποιητική ακρίβεια. Αντιθέτως, προτείνεται η επιλογή ενός καταλληλότερου αριθμού από κλασματικά ψηφία μόνο όταν είναι διαθέσιμα πραγματικά δεδομένα αστεριών, ώστε να είναι δυνατή η αξιολόγηση της πραγματικής επίδοσης του CG αλγόριθμου.

0.5.2 Αξιολόγηση Απόδοσης

Το πιο σημαντικό πείραμα του παρόντος ερευνητικού έργου, αφορά την μέτρηση του χρόνου εκτέλεσης κάθε υλοποίησης των αλγορίθμων, με σκοπό την αξιολόγηση της επιλογής ανάπτυξης τους σε FPGA. Στους Πίνακες 0.6, 0.7 παρουσιάζεται μία σύγκριση των διαφορετικών υλοποιήσεων στο υλικό, όσον αφορά τις μετρικές του latency, του throughput και του critical path.

Πίνακας 0.6: Σύγκριση μεταξύ των FPGA υλοποιήσεων (1).

Μοντέλο	**Latency**	**Initiation Interval**	**Critical Path (ns)**
CG VHDL	$N^2 + 42$	$N^2 + 3$	3.141
CG HLS	$N^2 + 37$	$N^2 + 6$	3.942
FGF	$8N^2 + 643$	$8N^2 + 37$	5.921

Για να είναι δυνατή η ευκολότερη κατανόηση των αποτελεσμάτων το latency και το Initiation Interval έχουν εκφραστεί ως συναρτήσεις του αριθμού των pixels N. Όπως είναι αναμενόμενο, αναφορικά με αυτές τις δύο μετρικές παρατηρείται παρόμοια συμπεριφορά από τις δύο υλοποιήσεις του CG

Πίνακας 0.7: Σύγκριση μεταξύ των FPGA υλοποιήσεων (2).

Μοντέλο	**Μέγιστη Συχνότητα** (MHz)	**Max Throughput (million clusters/s)**	**Επιτάχυνση** (×)
CG VHDL	320	18	200
CG HLS	250	12	160
FGF	170	1	25

αλγόριθμου. Ωστόσο η C++ αρχιτεκτονική παρουσιάζει μειωμένη καθυστέρηση λόγω της χρήσης του ισχυρού *Dataflow* directive, το οποίο δημιουργεί αυτόματα την διεπαφή μεταξύ των συναντήσεων, σε συνδυασμό με την χρήση ενός γρήγορου διαιρέτη που εκκινεί μία διαίρεση ανά κύκλο ρολογιού. Αντιθέτως, στην περίπτωση της RTL περιγραφής ήταν απαραίτητη η χρήση ενός IP διαιρέτη με χαμηλότερο ρυθμό καθώς και η υλοποίηση ενδιάμεσων buffers για καθυστέρηση της μεταφοράς δεδομένων. Αντίστοιχα, το VHDL μοντέλο παρουσιάζει αυξημένη διεκπαιρεωτικότητα καθώς η χαμηλού επιπέδου περιγραφή στο RTL επιτρέπει πιο λεπτομερή μοντελοποίηση της επιθυμητής συμπεριφοράς.

Η πιο σημαντική διαφορά μεταξύ των δύο μοντέλων αφορά την μέγιστη συχνότητα λειτουργίας. Το γεγονός πως το αυξημένο κρίσιμο μονοπάτι του C++ μοντέλου προκύπτει στο εσωτερικό του διαιρέτη, ο οποίος υλοποιείται αυτόματα από τον HLS μεταγλωττιστή και δεν είναι δυνατή η τροποποίηση του, αποδεικνύει πως η απόδοση περιορίζεται από το ίδιο το εργαλείο.

Επιπλέον παρατηρείται πως η FGF αρχιτεκτονική είναι σημαντικά πιο αργή από τα CG συστήματα, κάτι το οποίο απορρέει από την σημαντικά υψηλότερη πολυπλοκότητα του συγκεκριμένου αλγορίθμου. Όπως έχει αναλυθεί εκτενώς, τόσο το throughput όσο και το latency περιορίζονται από τις υπολογιστικές μονάδες κινητής υποδιαστολής. Ακόμα, σύμφωνα με τις αναφορές του HLS εργαλείου, το κρίσιμο μονοπάτι δημιουργείται λόγω αυτών των μονάδων. Μάλιστα, το γεγονός πως αποτελείται κατά 82% από καθυστέρηση διασυνδέσεων (net delay) αποδεικνύει πως η μέγιστη συνχότητα λειτουργίας περιορίζεται από την υψηλή κατανάλωση πόρων που οφείλεται στην υψηλή πολυπολοκότητα. Παρόλα αυτά, μετά από μία εξαντλητική εξερεύνηση του σχεδιαστικού χώρου αποδείχτηκε πως η συγκεκριμένη αρχιτεκτονική είναι η βέλτιστη δυνατή κατά την HLS περιγραφή.
Στον Πίνακα 0.8 παρουσιάζεται ο συνολικός χρόνος εκτέλεσης κάθε μοντέλου.

Πίνακας 0.8: Συνολικός χρόνος εκτέλεσης κάθε υλοποίησης σε δευτερόλεπτα.

Δεδομένα	**CG SW**	**FGF SW**	**CG VHDL**	**CG HLS**	**FGF HW**
3ξ3	0.126010	0.217139	0.000377	0.000591	0.006457
5ξ5	0.125483	0.269169	0.000880	0.001221	0.014036
Μιξεδ	0.151671	0.234976	0.000628	0.000906	0.010237

Λαμβάνοντας υπόψιν τα αριθμητικά δεδομένα από τους Πίνακες 0.7, 0.8 είναι προφανές πως είναι δυνατή η επιτάχυνση του αλγορίθμου CG κατά 2 τάξεις μεγέθους, ενώ ο FGF αλγόριθμος επιταχύνεται κατά 25 φορές. Προφανώς η επιτάχυνση του FGF αλγόριθμου περιορίζεται από την πολυπλοκότητα του, η οποία αντικατοπτρίζεται σε αυξημένη ζήτηση πόρων και οδηγεί σε μειωμένη διεκπαιρεωτικότητα και αυξημένο κρίσιμο μονοπάτι. Με δεδομένο μάλιστα πως το 80% αυτού αποτελείται από καθυστέρηση διασυνδέσεων είναι φανερό πως προκύπτει ένα πρόβλημα συμφόρησης στο FPGA, η οποία επηρεάζει άμεσα την δυνατή επιτάχυνση. Σε κάθε περίπτωση, μπορεί να ισχυριστεί κανείς πως η προτεινόμενη καινοτόμα αρχιτεκτονική παρουσιάζει ένα σημαντικό πλεονέκτημα ταχύτητας, διατηρώντας μάλιστα υψηλή ακρίβεια, με αποτέλεσμα να είναι κατάλληλη για κρίσιμες εφαρμογές πραγματικού χρόνου.

0.5.3 Αξιολόγηση Κατανάλωσης Πόρων

Στον Πίνακα 0.9 παρουσιάζεται η κατανάλωση πόρων στο FPGA κάθε υλοποίησης. Σε παρένθεση δίνονται οι συνολικοί πόροι που είναι διαθέσιμοι στο ZYNQ-7020 SoC.

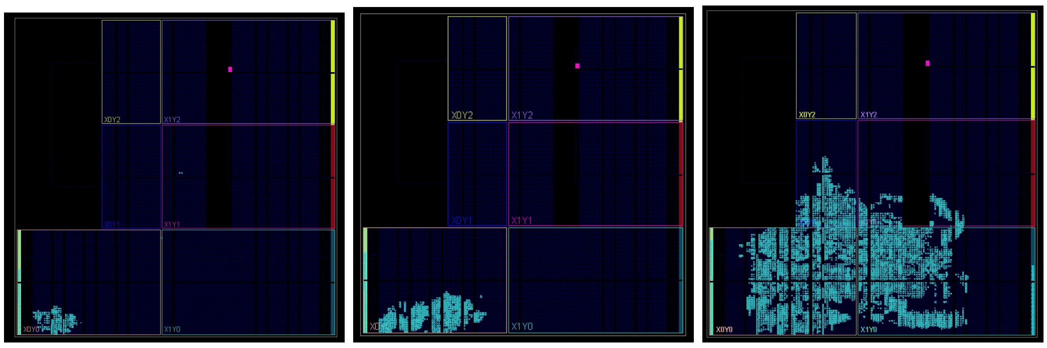
Πίνακας 0.9: Κατανάλωση πόρων στο FPGA.

Μοντέλο	**LUT** (53200)	**FF** (106400)	**DSP** (220)	**BRAM** (140)
CG VHDL	255	466	3	2
CG HLS	1362	1466	2	0
FGF	8661	13761	55	0

Μία ενδιαφέρουσα παρατήρηση είναι πως το C++ μοντέλο του CG αλγόριθμου απαιτεί περισσότερους λογικούς πόρους συγκριτικά με το VHDL μοντέλο. Αυτό μπορεί να εξηγηθεί από το γεγονός πως το τελευταίο χρησιμοποιεί 2 BRAMs για την υλοποίηση των FIFO καναλιών, ενώ στην περίπτωση της HLS προσέγγισης αυτά υλοποιούνται αυτόματα με Shift Register LUTs. Λαμβάνοντας υπόψιν τόσο τους αυξημένους πόρους όσο και το αυξημένο κρίσιμο μονοπάτι του C++ μοντέλου, συνάγεται το συμπέρασμα πως η HLS προσέγγιση δεν οδηγεί σε βέλτιστο κύκλωμα.

Αναμενόμενα, η FGF υλοποίηση απαιτεί σημαντικά περισσότερους πόρους στο FPGA κάτι που απορρέει από την σαφώς μεγαλύτερη πολυπλοκότητα της. Αξίζει να σημειωθεί πως ο μεγάλος αριθμός από DSP οφείλεται στον μεγάλο αριθμό παράλληλων πράξεων που αφορούν τα δεδομένα εισόδου και στην χρήση αριθμητικής κινητής υποδιαστολής.

Τέλος, είναι σημαντικό να ελεγχθεί η πραγματική τοποθέτηση κάθε κυκλώματος στο FPGA όπως δίνεται από το Vivado. Ο χώρος που καταλαμβάνει κάθε υλοποίηση φαίνεται στο Σχήμα 0.12.



(α΄) CG VHDL (β΄) CG HLS (γ΄) FGF

Σχήμα 0.12: Σύνοψη της τοποθέτησης κάθε κυκλώματος στο FPGA.

Ο αυξημένος χώρος που απαιτείται για το C++ κύκλωμα είναι μία ακόμα ένδειξη των διαφορών μεταξύ των δύο προγραμματιστικών μεθόδων, καθώς η VHDL μπορεί να απαιτεί τον μεγαλύτερο σχεδιαστικό χρόνο αλλά δίνει τη βέλτιστη υλοποίηση στο FPGA. Επίσης, η τοποθέτηση του FGF μοντέλου στο FPGA εξηγεί τον λόγο για τον οποίο η καθυστέρηση διασυνδέσεων αποτελεί τον κύριο παράγοντα του κρίσιμου μονοπατιού και τελικά περιορίζει την απόδοση του συστήματος. Συνολικά, πιστεύεται πως η προσέγγιση του προβλήματος με HLS δεν μπορεί να δώσει καλύτερα αποτελέσματα, κάτι το οποίο πιθανώς να είναι δυνατόν εάν ο αλγόριθμος FGF περιγραφεί σε επίπεδο RTL με VHDL.

0.6 Συμπεράσματα

0.6.1 Σύνοψη

Στην παρούσα διπλωματική εργασία παρουσιάζεται η επιτάχυνση δύο αλγορίθμων κεντραρίσματος σε μία πλατφόρμα FPGA, με σκοπό την χρήση τους σε σύγχρονες διαστημικές εφαρμογές. Κάθε υλοποίηση παρουσιάζει διαφορετικά πλεονεκτήματα και τελικά προκύπτει ένας συμβιβασμός (trade-off) μεταξύ ακρίβειας και ταχύτητας, ο οποίος αντικατοπτρίζεται πλήρως στις απαιτήσεις πόρων του FPGA. Ο αλγόριθμος Κέντρου Βαρύτητας, λόγω της απλότητας του, επιταχύνθηκε κατά 2 τάξεις μεγέθους αλλά η ακρίβεια του απλά περιορίστηκε εντός 0.1 pixels σε σχέση με το software μοντέλο. Αντιθέτως, ο πιο πολύπλοκος αλγόριθμος Γρήγορου Γκαουσσιανού Ταιριάσματος επιταχύνθηκε κατά 25 φορές, διατηρώντας παράλληλα άριστη ακρίβεια. Αυτή η υλοποίηση στο FPGA αποτελεί μία πρωτότυπη αρχιτεκτονική που μπορεί να χρησιμοποιηθεί στα πλαίσια της διαδικασίας εντοπισμού αστεριών, η οποία είναι ζωτικής σημασίας για την επακριβή και γρήγορο προσανατολισμό των δορυφόρων στο διάστημα. Λαμβάνοντας υπόψιν το γεγονός πως η ακρίβεια ενός ανιχνευτή αστεριών καθορίζεται κυρίως από την διαδικασία κεντραρίσματος, το προτεινόμενο μοντέλο εξασφαλίζει πως υπό συγκεκριμένες προϋποθέσεις (π.χ. χρήση κάμερας υψηλής ανάλυσης και ικανοποιητικό clustering) είναι δυνατή η ανίχνευση του κέντρου των αστεριών με αμελητέο σφάλμα. Παράλληλα είναι κατάλληλο για λειτουργία υψηλού ρυθμού καθώς μπορεί να επεξεργαστεί 10000 σε περίπου 10 ms.

Επιπρόσθετα, σε αυτό το ερευνητικό έργο μελετώνται σε βάθος οι μέθοδοι προγραμματισμού FPGA με χρήση VHDL και C++. Εκμεταλλευτήκαμε τα πλεονέκτημα της ευκολίας και της υψηλής παραγωγικότητας που χαρακτηρίζουν το HLS για την πραγματοποίηση εξαντλητικής εξερεύνησης του χώρου σχεδίασης και εκτεταμένου ελέγχου. Δεν θα ήταν αλλιώς δυνατή η ανάπτυξη και επαλήθευση τριών μοντέλων σε τόσο σύντομο χρονικό διάστημα. Ειδικά η VHDL περιγραφή του FGF αλγόριθμου θα ήταν ιδιαίτερα απαιτητική. Ωστόσο, η ποιότητα των αποτελεσμάτων που παρέχει η RTL μοντελοποίηση είναι αδιαμφισβήτητη. Ακόμα και στην περίπτωση του πολύ απλού CG αλγόριθμου, το HLS δεν ήταν σε θέση να πετύχει τη βέλτιστη απόδοση που παρατηρείται στο VHDL μοντέλο, όσον αφορά την μέγιστη συχνότητα λειτουργίας και την κατανάλωση πόρων. Τέλος, πιστεύεται πως ακόμα περισσότερη απόδοση μπορεί να εξαχθεί από τον FGF αλγόριθμο εάν υλοποιηθεί με χρήση της VHDL.

0.6.2 Μελλοντική Εργασία

Για να είναι δυνατή η πλήρης κατανόηση των πλεονεκτημάτων της προτεινόμενης πρωτότυπης αρχιτεκτονικής, είναι απαραίτητη η ενσωμάτωση της σε ένα πλήρες σύστημα ανίχνευσης αστεριών. Ιδανικά, στοχεύουμε στην χρησιμοποίηση του συγκεκριμένου μοντέλου σε μελλοντικές διαστημικές αποστολές της Infinite Orbits.

Επίσης θεωρείται ιδιαίτερα χρήσιμος ο έλεγχος της απόδοσης όλων των υλοποιήσεων σε πραγματικές εικόνες αστεριών. Το καινοτόμο μοντέλο αναμένεται να επιδείξει αντίστοιχα άριστη ακρίβεια. Ωστόσο είναι ενδιαφέρον ο έλεγχος της συμπεριφοράς των CG μοντέλων, καθώς δεν κατάφεραν να παρουσιάσουν αξιοσημείωτα αποτελέσματα στα τεχνητά δεδομένα. Σε περίπτωση που ο αλγόριθμος CG επιδείξει ικανοποιητική συμπεριφορά, προτείνεται η διεξαγωγή ενός πιο ενδελεχή ελέγχου, σχετικά με το αναγκαίο πλήθος δυαδικών ψηφίων για την αναπαράσταση του κλασματικού μέρους των centroids, με στόχο την επίτευξη καλύτερης ακρίβειας στο υλικό. Τέλος, η VHDL περιγραφή του FGF αλγόριθμου πιθανώς να μπορέσει να δώσει μία καλύτερη εικόνα αναφορικά με την μέγιστη απόδοση που μπορεί να εξαχθεί. Παρόλα αυτά, όπως τονίστηκε, η απόδοση της προτεινόμενης αρχιτεκτονικής είναι ικανοποιητική και συνεπώς μπορεί εύκολα να χρησιμοποιηθεί σε σύγχρονους ανιχνευτές αστεριών, που σημαίνει πως μία περαιτέρω επιτάχυνση δεν θα οδηγούσε σε σημαντικά κέρδη.

# Chapter 1

Introduction

Since the launch of the first satellite, Sputnik 1 in 1957, a lot has changed in the field of space technologies. In the recent decades space exploration has seen significant progress and an extensive space infrastructure has been developed. Hundreds of daily services rely on this infrastructure, such as weather forecasting, satellite television and navigation systems [8]. As computer, electronics and material sciences advance, a growing number of institutes and companies are attracted to the field of space exploitation.

1.1 Satellites

1.1.1 Applications

The most fundamental element of the established space infrastructure are artificial satellites. These vehicles orbit Earth at different heights, speeds and along different paths and can be used in a variety of applications. One of the most common use is for communications purposes. Typically, communications satellites cover large areas of the Earth's surface and are able to connect remote regions without visual contact, in cases where a terrestrial connection would be unaffordable or impossible. A significant number of sciences like astronomy and earth science are highly benefited by satellites, as they provide land survey data or can be used as space telescopes. Although an increasing number of space vehicles is constantly launched to observe objects of the outer space, the most common type of orbit is the geocentric, which is divided in the orbits shown in Figure 1.1.

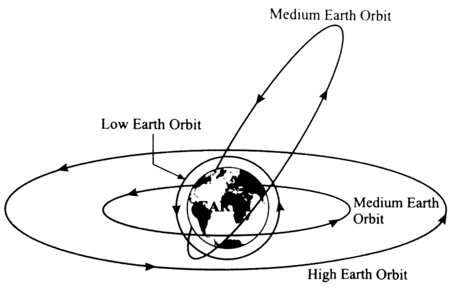


Figure 1.1: Satellite orbits in altitude classification.

Geostationary Orbit (GEO) circles Earth along equator with an altitude of 35,736 km and has a high commercial and strategic value. Satellites in this orbit travel with exactly the same rate as

Earth and as a result from a terrestrial observer they look like fixed points in the sky, which makes them useful for reliable broadcasting applications. The Low Earth Orbit (LEO) is the closest orbit to Earth's surface, ranges in altitude from 160 - 2,000 km and nearly 7,500 satellites are located there. Due to this proximity, this orbit can be used for high resolution satellite imaging and is expected to play an important role to the terrestrial 5G networks [9]. Unlike the GEO case, LEO satellites are not obliged to always follow a specific path around Earth, their plane can be tilted. However, their relatively low altitude leads to a decreased Field Of View (FOV) and therefore in order to increase coverage, groups of satellites are launched together and form a net around Earth, called constellation. Medium Earth Orbit (MEO) lies between LEO and GEO, providing navigation services such as the Global Positioning System (GPS).

1.1.2 Subsystems

Independently of space mission and selected orbit, each satellite includes some basic subsystems which cooperate in order to achieve functionality and reliability. A simplified schematic of these subsystems is given in Figure 1.2.

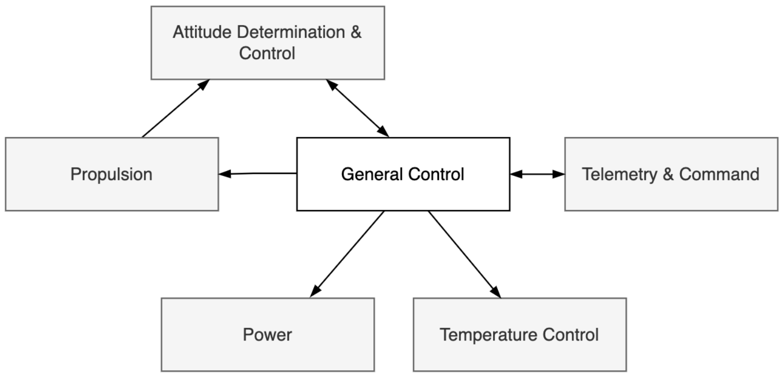


Figure 1.2: Abstract diagram of the most common satellite subsystems.

The desired function of a satellite is usually achieved using information collectors (e.g., cameras, particle detectors) to gather data, and communications equipment (e.g., antennas, repeaters) to transmit them to terrestrial stations. Gravitational forces due to sun's and moon's gravity tend to disrupt satellites' trajectories making altitude determination critical in any space mission, not only for navigation and orbit preservation purposes but also because the service's quality depends on it. For example, highly pointing antennas and star photo shooting can't tolerate variations regarding the satellite's position or velocity. Propulsion System includes thrusters that fire chemical gases and together with the Attitude Determination and Control System (ADCS) creates a closed loop that determines the necessary manoeuvres in any given moment. The temperature difference between the side of the satellite which is seen by the sun and the other one that isn't can be significant and such gradients must be monitored and controlled by the Temperature Control System as they could critically damage the functionality of the vehicle. Power System consists of a primary power generator (solar cells) that convert solar to electrical energy, a secondary power generation (rechargeable accumulator) that provides the required energy when there is no visual contact with the sun, and a supplement line in order to distribute it among the different parts of the satellite. Furthermore the Telemetry, Tracking & Command System monitors the other subsystems and transmits relevant data to the General Control System, while at the same time it determines the orbit's parameters and provides terrestrial stations with satellite tracing data.

1.1.3 Trends

1.1.3.1 Motivation

In the last decade, there has been a huge advancement of electronics and computer science. One of the many fields that has been strongly affected is space technology. The decreasing physical size of hardware imposes stricter constraints on available performance (e.g., smaller solar panels limit the ability of power collection) and subsequently better solutions should be found. A typical example are the CubeSats [10], whose low cost feature led to their extensive utilization in LEO applications. The increasing performance and reliability of processing systems, allow more data and more complex software to be exploited, while the risk of damage due to space radiation is minimized. As a result the limits of space exploitation are pushed even further and a bunch of highly innovative technologies are constantly made available. Apart from the terrestrial cases, that have been briefly mentioned, the modern trend is to provide services to the already in-orbit space assets. Considering the fact that almost 1,000 satellites are launched every year [11], it is easily understood why immediate and future markets with a value over \$10B are available [12].

This thesis was developed in collaboration with Infinite Orbits, a company that utilizes the most innovative technologies in Space Robotics in order to successfully provide an arsenal of in-orbit services. These trending services can be divided in docking and non-docking and are described next.

1.1.3.2 Services

Two of the most important non-docking services are space-based space surveillance [13] and in-orbit inspection [14]. The concept of both is to use an inspector-satellite in order to observe and estimate with high precision the orbits of space objects, such as other satellites, asteroids and debris.

The game-changing service however is life extension which requires rendezvous and docking with extreme accuracy and safety [15]. Docking is the mechanical process of joining one space module (i.e., satellite) to another one and is accomplished with robotic assistance. The process that precedes docking, is called rendezvouz and its goal is to firstly arrive at the same orbit and then approach the target satellite. Life extension service has proven to be a very challenging case of rendezvous and docking due to the non-cooperative nature of the aimed satellite, the risks involved and the characteristics of the target orbit (GEO).

Every satellite has a finite lifetime that depends on the amount of fuel it can carry. When they run out of fuel, they retire even though they may be fully functional in the scope they serve, and must be replaced by a new vehicle. The estimated lifetime of a GEO satellite is 15 years. Considering the complexity of these satellites and the huge construction (\$200M) and operation costs (\$20M/year) it is understood that this situation could be characterized as a massive waste of technological capabilities and money. The basic concept of the life extension service is to use a small satellite that can be docked onto a satellite, that has retired or reaches the end of its lifetime, in order to provide an alternative propulsion system [16].

Even though satellite collision accidents are extremely rare, more than 27,000 pieces of orbital debris, also called “space junk”, threaten human spaceflight and robotic missions. This kind of debris varies in size and consists of non-functional spacecrafts, abandoned launch vehicle stages, mission-related debris and fragmentation debris. The process of regenerating an already retired satellite can be very useful in means of debris active removal. Similar docking services include debris avoidance [17] and satellite repair or maintenance.

1.1.3.3 Key Technology: Autonomous Navigation

In order for such complex applications to be realized, a robust, autonomous and real time operating satellite navigation system is required. The high cost of the involved satellites means that mistakes cannot be tolerated, making the usage of space communications channel, which is characterized by high latency and a potential loss of connection due to fading, unacceptable. Therefore, ”smart” software that could process a wide variety of high rate information is required. Generally, the autonomous navigation process is divided into two categories, near range and far range navigation. The former, which is out of the scope of this research, is concerned with pose (position and altitude) determination of target object for inspection and docking, in distances lower than 250 meters. In this thesis, high accuracy and real time performance solutions to the problem of far range navigation are proposed.

1.2 Star Trackers

Far range navigation refers to the relative orbit estimation with respect to other space objects (i.e., satellites) in order to perform autonomous rendezvous. Today this process can start with an initial separation distance of up to 2,000 km. A fundamental element of autonomous navigation is the satellite attitude information, which must be constantly available and, as mentioned, is extracted through the Attitude Determination & Control System. In the past, ADCS has used various types of attitude sensors like sun sensors, horizon sensors, RF sensors or magnetometers (coils). However, it has been proven that the demand for high accuracy in modern applications can only be achieved using star trackers [18].

1.2.1 Operation & Layout

Star Trackers could be considered as a modern take to ancient technology, as they are based on star observation in order to determine satellite’s position in the celestial sphere, much like travelers and sailors did for thousands of years. More specifically these instruments, exploiting the fact that astronomy has provided the positions of many stars with a high degree of accuracy, capture night sky images and by analyzing the placement of the surrounding stars using software algorithms, can determine the exact location and attitude of the satellite. This process is valid and accurate due to the fact that for every pair of bright stars in the sky there is a unique angular separation between them [19]. A typical star tracker layout is described next.

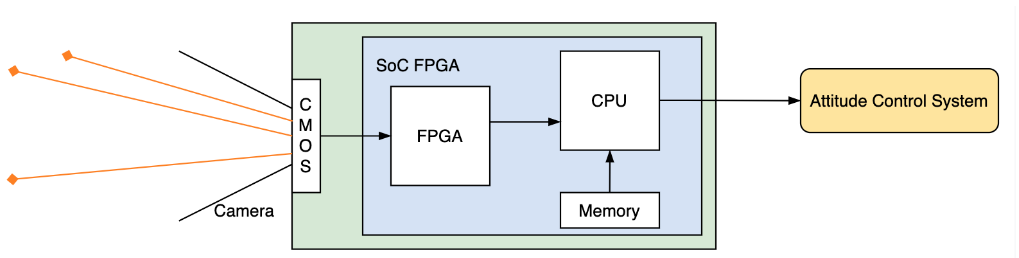


Figure 1.3: A simplified illustration of a typical star tracker layout.

Star Trackers are radiation-hardened instruments that consist of an electronic camera and the associated processing electronics, providing a vision-based navigation. Earlier designs used Charge Coupled Devices that could capture a limited number of stars, while extensive external processing was required to transform data from the CCD referenced to inertial referenced coordinates, which usually led to poor performance. State-of-the-art star trackers [20] integrate Complementary Metal Oxide Semiconductor (CMOS) technology in camera sensor in order to provide higher resolution at lower costs [21].

Then the input video stream is driven to the processing unit, where image processing is performed in order to detect the stars and pattern recognition algorithms are used to map them to known constellations and positions. Finally the output attitudes, which are referenced directly to the inertial space, are used by the rest of ADCS for the appropriate actions. During this operation the amount of pixels provided by the input sensor and need to be processed is huge, resulting in a high computational task. Considering the need for real time operation, the execution of the whole process in a conventional processor is inefficient, making the use of System-on-Chip (SoC) which integrates also a Field Programmable Gate Array (FPGA) necessary. In a typical configuration, the FPGA would directly connect to the image sensor through a high speed stream protocol in order to perform image processing and then the high performance processor would be used to perform attitude determination.

This kind of designs are fully autonomous, allow high speed batch image processing and provide very accurate results, while the total consumed power is kept low due to the characteristics of modern SoCs.

It should be highlighted that in space domain, FPGAs are used to improve the performance of space applications, as they outperform the conventional general-purpose processors (e.g., the radiation-hardened CPUs that have been used in space missions). In particular, the space community employs both space-grade [22, 23, 24] and Commercial Off-The-Shelf (COTS) [25, 26, 27] FPGAs (e.g., Zynq). Besides accelerating computationally intensive DSP functions, FPGAs are also used as framing processors in co-processing architectures [28, 29, 30], where they perform tasks such as I/O handling, data transcoding, and data compression.

Operation Modes

In modern systems there are two different modes of operation, as Liebe [31] suggests. Lost In Space (LIS) is a fully autonomous mode in which the whole input frames are analyzed. Tracking Mode succeeds LIS phase and uses the information previously obtained in order to increase the processing rate. This is achieved by adopting a windowing technique which allows limited search in smaller areas around the previously found stars. Although tracking mode is extensively used today, the critical phase which defines a star tracker's maximum operation rate is LIS and therefore only this case is discussed in this thesis.

1.2.2 Processing Flow

Given a two-dimensional image which contains only a small portion of the night sky, the problem is to determine the satellite's exact attitude in the inertial space. The solution to this problem can be found using the chain of blocks appearing in Figure 1.4.

Initially night sky images are captured, forming a video input stream. Due to the non-ideality of the camera's lens, each star in an image is blurred, creating a bright region which covers more than one pixel. The size of this region depends on various sensor's characteristics such as aperture size and focus setting. A zoomed-in image of such a blurred star is shown in Figure 1.5.

Even if one can intuitively think that this effect leads to a drop in accuracy, this is not the case. If an ideal sensor could be used, the captured star would be limited inside a single pixel. However, since the minimum reference unit in image processing is one pixel, the accuracy would be

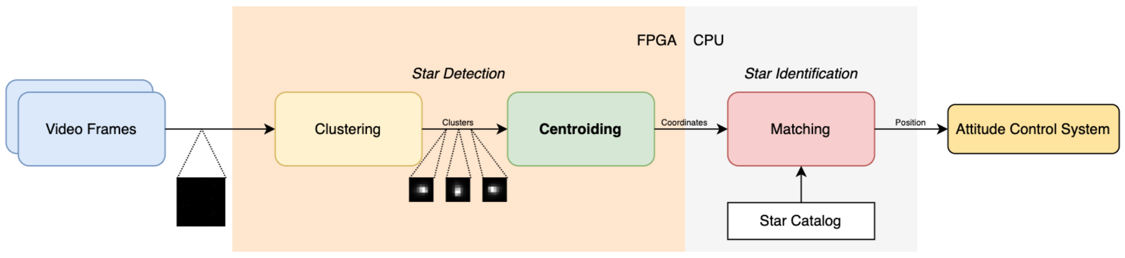


Figure 1.4: Star tracker's processing pipeline.

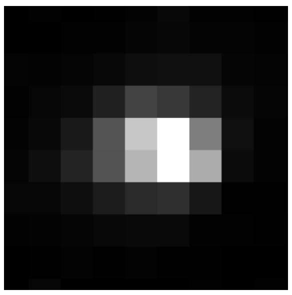


Figure 1.5: A bright region over a number of pixels represents a real star in a zoomed-in night sky photo. Image taken from von Wielligh [1].

also limited to one pixel. In reality the observed blurring spreads the underlying information over several pixels making the estimation of a star's center with subpixel accuracy possible. Therefore in real applications, depending on the anticipated star's intensity, lens is purposely defocused in order to typically create regions of 3x3 or 5x5 pixels, which are also called clusters (of stars).

1.2.2.1 Clustering

The first of the three processes shown in Figure 1.5, called clustering, is responsible for distinguishing the bright regions from the rest of the image and extracting the respective pixels. For this purpose, in LIS mode the whole image is searched and each pixel's intensity is compared to a threshold, which determines if the pixel could belong in a cluster. When a set of neighboring pixels is found, a desired region is detected. During this process, constraints over the allowed size and shape of the expected clusters are established. For example the maximum and minimum number of pixels are limited, while too many neighboring bright pixels may indicate invalid light sources like planets or nearby debris and too small regions could represent noise effects. It should be also mentioned that the threshold value can be pre-defined based on the application details or can be dynamically adjusted and dependent on the accuracy results at the end of the described processing pipeline.

1.2.2.2 Centroiding

The extracted clusters are then passed to the centroiding process, that aims to determine their centers, which for the rest of this thesis will be called *centroids*. These two-dimensional centroids (x- and y-axis) correspond to the real locations of stars on the image plane and are used later in order to calculate the angular distances. During this phase, a suitable algorithm is applied to the valid clusters in order to achieve centroid extraction with subpixel accuracy. In this work different optimized approaches to this process are proposed.

1.2.2.3 Matching

The final task in the described flow is matching, during which the detected stars are matched to known catalogue stars in order to understand satellite's orientation. This process uses the most complex algorithms that perform pattern recognition usually on the angular distances between the celestial bodies. Today large catalogues that cover the entire sky, optimized for low power and low computational demands are available, containing information for thousands of stars and star separations. In literature there have been many suitable matching algorithms proposed [32], which demonstrate different optimizations or extract different features. One should also notice that these algorithms could reject certain stars as false stars, if no matches or matches with low confidence are found, leading to error reduction. The results of this process could then be used by navigation filters to estimate relative orbits and command the Control System to make the appropriate moves.

1.2.2.4 Process Integration

Clustering and centroiding tasks constitute the *Star Detection* phase. On the one hand, these tasks process a large number of pixels, as clustering accesses entire frames and centroiding processes multiple clusters, which makes both of them computationally intensive. On the other hand matching process, which constitutes the *Star Identification* phase, operates on low dimensional vectors and performs more complex calculations.

In state-of-the-art designs, this scenario is suitable for process partition and co-design, where the computationally intensive tasks are developed on the FPGA system side, in order to be accelerated, and the more complex tasks on the processor side of the SoC, in order for software based optimizations to be performed. Hardware optimizations of various centroiding algorithms are the main concern of this research.

This kind of design decision removes the bottleneck of data processing and makes the time needed for image capturing the critical factor, as far as update rates are concerned. More specifically, there is a period of time which is called integration (or exposure) time and is the time needed for the sensor to be exposed to light, in order to provide stars with a desired level of brightness. Longer exposure times lead to brighter stars but limit the star tracker's update rate. Note that the integration time is also dependant on other factors such as sensor's sensitivity, lens' aperture and satellite's speed. For example, in fast changing dynamic conditions (i.e., high slew rate) blur effects tend to be more challenging and care should be taken regarding the exposure time and processing algorithms [33].

In order for the described process chain to achieve real time performance, it is necessary that the related tasks are interleaved. Ideally, while centroiding calculates a cluster's centroids, clustering is processing the next bright region of the image and the sensor has already started to integrate the next frame. In this way, each process is free to execute on it's own pace, limited only by the availability of data in the intermediate channels (i.e., buffers).

Finally it should be highlighted that the assumptions that are made during the star detection phase define the eventual performance of the overall design. For instance, approximations during clustering regarding the allowed cluster's shape and size could affect accuracy as well as processing speed. Similarly, the centroiding algorithm could determine the level of robustness to noise, which is a very important feature considering the noisy nature of space applications. Generally any error in centroid detection is propagated down the algorithm pipeline and reduce the overall accuracy of the star tracker.

1.2.3 Modern Designs

Second generation star trackers are capable of accuracies in the arcsecond range while update rates are typically 0.5 to 10 Hz. Especially in high performance ADCS which are used in agile micro- or nanosatellites and include a Control Moment Gyroscope (CMG) [34], the demands for high update rates are even stricter.

As satellites have a limited lifetime dependent on energy resources, power consumption is also a very important feature. Especially in smaller Cubesat compatible star trackers consumption is in the order of 1W, while in larger communications satellites cases of 10W are very usual. The trend towards smaller satellites demands instruments of decreased physical volume and mass.

When a commercial star tracker is to be chosen more characteristics should be taken into consideration. Some of these characteristics are Field of View, maximum slew rate and star catalogue size. A short list of state-of-the-art star trackers is displayed in Table 1.1.

Table 1.1: Modern Star Tracker Specifications

Model Name	CT-2020	ASTRO APS	ST400	ST-16RT2
Manufacturer	Ball Aerospace	Jena-Optronik	AAC Clyde Space	Sinclair Interplanetary
Accuracy (arcsec)	1.5	< 1	10	5
Update Rate (Hz)	10	16	5	2
Power (W)	< 8	< 12	0.7	< 0.5
Max Rate (deg/s)	8	< 5	> 1	< 3
Mass (kg)	3	2	0.28	0.158
Volume (cm)	14.8 x 58.4	15.4 x 15.4 x 23.7	5.4 x 5.4 x 9.1	6.2 x 5.6 x 3.8

CT-2020 is the latest available star tracker of the HAST (High Accuracy Star Trackers) series [35] designed by Ball Aerospace. It provides LIS accuracy in the order of 1.5 arcseconds, which improves either if two units are equipped onboard or when operating in tracking mode. This high-performance product provides a fast LIS update rate of 10 Hz, while full performance with moon in FOV and with 15 degrees sun angle is also achievable.

ASTRO APS is one of the four hosted TDPs (Technology Demonstration Payloads) of ESA (European Space Agency) currently carried on Alphasat, a mobile communications service spacecraft in GEO of Inmarsat, in order to validate in-orbit performance, robustness and lifetime and enable its use on other space vehicles. It uses the most advanced radiation hard CMOS Active Pixel Sensor detector technology for long-term missions in GEO, with a lifetime longer than 18 years. Optimized software algorithms are also used to identify and compensate for anomalies in captured images. Update rates up to 32 Hz are available on demand, while sun up to 26 degrees and earth up to 20 degrees in FOV are allowed.

Compared to the already described products, the ST400 is characterized by lower mass, lower power, smaller physical size and is designed for usage on pico- and nanosatellite platforms. Optional baffles in order to perform while sun is up to 40 degrees in FOV are available.

ST-16RT2 is the most power efficient case in the given table, a feature which comes at a price of lower update rate. However, unlike the ST400 this update rate refers to LIS mode which is the default processing mode for each frame, meaning that zero initial acquisition time is required. Optional usage of baffles can provide sun avoidance up to 34 degrees in FOV.

It is important to mention that full performance is usually available only up to a specific satellite slew rate. Above that threshold, lower accuracies are expected. The described products are just a small portion of the modern second generation star trackers. A significant number of companies and universities have developed custom implementations in the past years. Apparently larger instruments designed for large satellites like CT-2020 and ASTRO APS outperform smaller ones in most areas. However more compact star trackers cost less, consume less power and can be used in the increasingly popular nanosatellite applications.

1.3 Project Objective

It can be easily realized by now that second generation star trackers are very complex systems which require excellent integration and cooperation of the underlying systems and algorithms, in order to achieve real time operation and high accuracy. Star Detection phase can be the limiting factor regarding execution speed due to the computational intensive task of processing large amounts of data, which indicates that much of the system optimization can be done in this step.

The main objective of the project is to optimize the centroiding process on a SoC FPGA platform for a given system configuration. This configuration is provided by Infinite Orbits and combines a custom CMOS image sensor, a custom lens and the ZedBoard development board, which hosts the Xilinx ZYNQ-7020 SoC. This custom design allows complete control over the entire software-hardware pipeline and will be used in future space missions, led by Infinite Orbits. The basic requirements that the total system needs to meet are the following:

- Real time performance of 1-2 frames per second
- Mean accuracy error of less than 3 arcseconds across boresight

In this thesis we propose two different optimized architectures implemented on the FPGA side of the SoC, that perform fast and accurate centroid calculation in a pipelined fashion. In this way, the system doesn't need to wait the calculation of a cluster's centroids to be completed before it starts operating on the next cluster. The calculated centroids are then used by navigation filters in order to perform efficient computation of relative orbits with accuracy error less than 1%. The integration of the proposed architectures into the custom pipeline, allows the system to achieve the desired performance.

As part of the research presented in this thesis, two different centroiding algorithms have been used in order to create a baseline and a novel design. Comparisons between the algorithms related to speed, accuracy and FPGA resources are present in this thesis. FPGA programming methods with a Hardware Description Language or a High Level Language have been both used during system development. More specifically, VHDL and C++ approaches will be introduced and evaluated.

1.4 Thesis Outline

In Chapter 2 basic background knowledge necessary for the rest of this thesis and related work will be discussed. Chapter 3 analyzes the considered centroiding algorithms and their performance, while the whole development process is explained in Chapter 4. The evaluation of the proposed designs is given in Chapter 5 and the final conclusions are found in Chapter 6.

Chapter 2

Background

2.1 Space Navigation

This section covers basic theory in space navigation.

2.1.1 Coordinate Systems

Throughout this project three different coordinate systems are used, while each of them provide specific advantages when different approaches to the attitude determination problem are required. The details and the relationships between these systems are explained next.

2.1.1.1 J2000 Earth Centered Inertial

Since the studying objects in this thesis are satellites and stars which revolve around Earth, or alternatively around the center of the Earth, the choice of an Earth-centered coordinate system can heavily simplify the equations of motion that describe the orbital motion of these objects. The ease in calculations is even greater when the chosen frame is inertial, which means that it does not accelerate (rotate) and it is fixed in space relative to the stars. Otherwise if an non-inertial frame was chosen, the calculation of a relative velocity component would be necessary, making the studied system more complex.

The most suitable frame for this application is the Earth Centered Inertial J2000, which is a Cartesian coordinate system with its origin fixed at the center of the Earth. The fundamental plane (x-y plane) is the equatorial plane and the principal axis (x-axis) points to the Vernal Equinox direction. The term J2000 is used to define that the x-axis is aligned with the direction of the point where the Earth's equator intersected the Earth's orbit plane at 12:00 on 1 January 2000. The z-axis runs along the Earth's rotational axis as it was at that time, pointing North (celestial North Pole). Y-axis is rotated by 90° east on the equatorial plane with respect to x-axis and completes the right handed orthogonal system.

Due to the great distances to Earth that stars have, for an observer on Earth or close to it (i.e., on a satellite), they seem as they are equally far away and fixed on a sphere of arbitrary or infinite radius. Consequently in order for a such distanced object's position to be accurately expressed a spherical coordinate system should be used. The typically used coordinates are right ascension, denoted as α, and declination, denoted as δ, which are the angles measured from the Vernal Equinox and the equatorial plane respectively.

A very enlightening illustration of ECI frames is given in Figure 2.1.

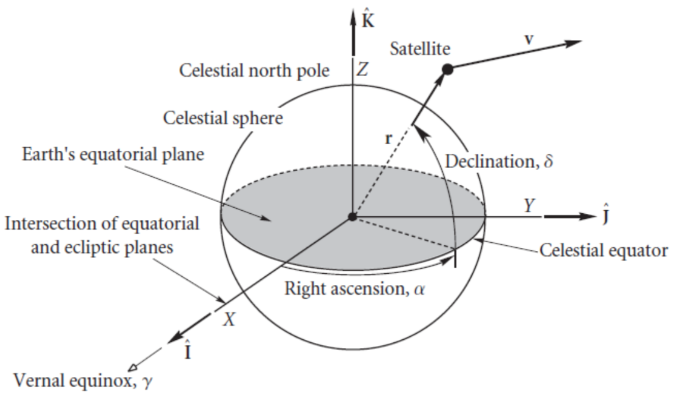


Figure 2.1: Illustration of the Earth Centered Inertial frames. Image is taken from website with unknown author [2].

2.1.1.2 Image Plane & Sensor Body Coordinate Systems

The fundamental function of a camera sensor is to project information from the three-dimensional world onto a two-dimensional image, which can be viewed in a monitor or on a printed paper. An image is a two-dimensional pattern of brightness and is formed based on a geometric correspondence between points in the scene (3D world) and points in the image [36]. In order to establish this correspondence and explain how the projection is performed, two different coordinate systems are introduced and illustrated in Figure 2.2.

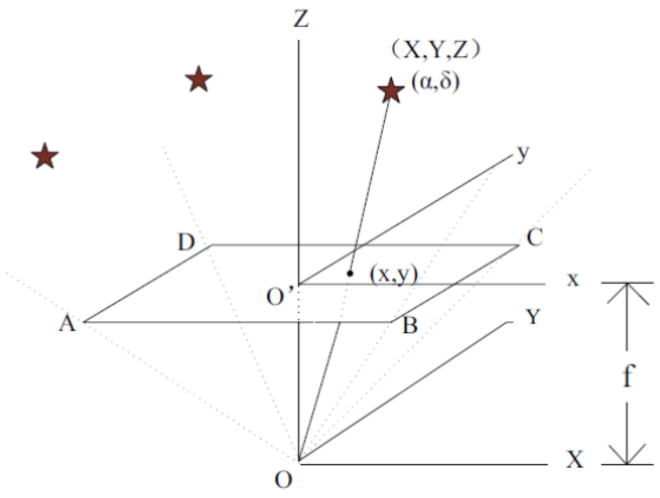


Figure 2.2: Illustration of the relationship between the image plane and the sensor body coordinate systems. Image taken from Qian et al. [3].

An ideal pinhole camera is an abstract camera model, which describes a box with a small hole in it [37] and it is often used in order to mathematically model the operation of the typical more complex cameras. In a pinhole, each point in the image corresponds to a particular direction defined by a ray from that point through the pinhole, exploiting the fact that light travels along straight lines. Cameras use a lens in order to bend light waves into a narrow beam that produces an image on the film. Thus the pinhole camera produces the same type of upside-down, reversed image as a modern camera.

Figure 2.2 displays such a pinhole camera model. The optical axis is defined to be the perpendicular from the camera center to the image plane and is also called lens boresight. A convenient Cartesian coordinate system, is introduced with the origin at the camera center, denoted as O, and the z-axis (principal axis) aligned with the optical axis, pointing towards the image. Fixing the system origin at the sensor center simplifies the problem as the external parameters (position and orientation of the camera) are known and don't need calculation [38]. Apparently this system, which is called *sensor body* coordinate system, is related to the camera viewpoint in 3D space and can provide information about the objects being imaged.

As illustrated in Figure 2.2 the image plane is a two-dimensional coordinate system which is aligned with the image sensor surface, represented by the points A, C and D. For digital images the coordinates are expressed in pixels and in digital signal processing problems the system origin is typically defined as the first pixel in the top-left corner of the sensor. For the rest of this project, the adopted image coordinate system will follow this typical definition and the raster scan method will be adopted for image transmission and processing purposes, as Figure 2.3 suggests. Note that indexing starts from 0 in order to conform with the manner that an array is usually indexed in computing systems.

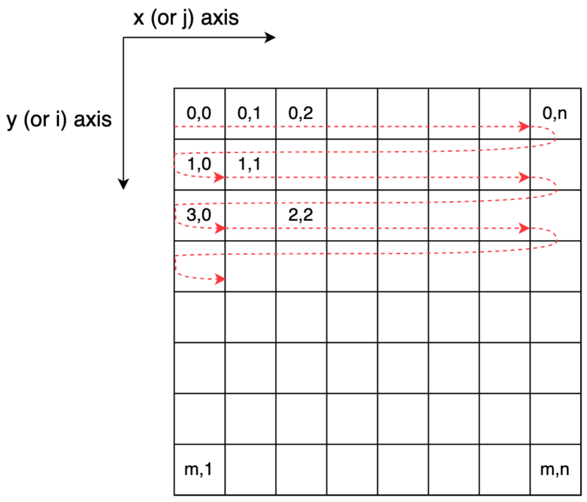


Figure 2.3: Image Plane Coordinate System. Origin is the first pixel in the top-left corner the axes are named x or j for columns and y or i for rows. Red curved line shows the raster scan processing method.

2.1.1.3 Coordinate Transformation

The result of a centroiding algorithm is expressed in the 2D image plane coordinates, ideally with subpixel accuracy. However, this result has no physical substance as the image plane itself is used only for convenience. In order for the matching algorithm to match the calculated centroid to a real star using the sky catalogue, information about the detected star's real position in the inertial space should be available. Although this research is only concerned with data processing and centroid calculation, it is important to establish the correspondence between a star's real location and its centroid position in the captured image.

This correspondence is naturally derived from the image formation process itself. This process can be divided in two different coordinate transformations. The first one is a rotational transformation from the 3D ECI coordinate system to the 3D Sensor Body coordinate system and is shortly described below, based on a paper from Qian et al. [3] and Figure 2.2.

A point in 3D the real world (i.e., a star) is expressed as $\mathbf{A_i} = [\alpha\ \delta]^T$ in the ECI system, if spherical notation is used. If Cartesian notation is used instead, the new expression is the following.

$$\mathbf{A_i} = \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} \cos\alpha\cos\delta \\ \sin\alpha\cos\delta \\ \sin\delta \end{bmatrix} \tag{2.1}$$

The same point, with respect to the sensor body system in Figure 2.2 is notated as:

$$\mathbf{A_i} = [X\ Y\ Z]^T \tag{2.2}$$

The usage of a rotational transformation is valid, assuming that the sensor is fixed at the center of the Earth. This is a safe assumption, considering the very small distance between the satellite and the center of the earth in comparison to the arbitrarily large distance between the satellite and the stars. This transformation is defined below.

$$[X\ Y\ Z]^T = \mathbf{M}[U\ V\ W]^T \tag{2.3}$$

where $\mathbf{M}$ is a rotation matrix. Using the principle of 3-1-3 Euler coordinate rotation [39], it can be written as follows.

$$\mathbf{M} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

where ψ, θ and ϕ are the 3D rotation angles, which are displayed in Figure 2.4.

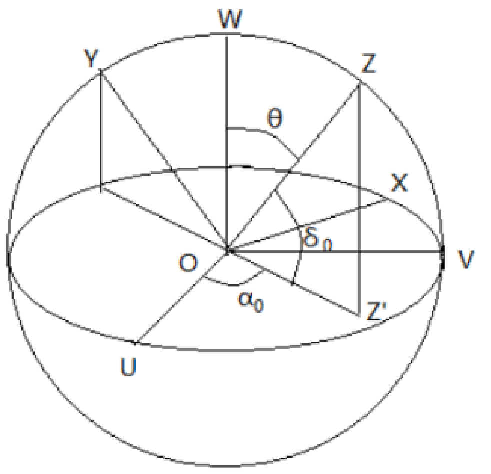


Figure 2.4: Illustration of ECI and Sensor Body coordinate systems together. Star coordinates from example are also displayed. Image taken from Qian et al. [3].

Sensor's boresight orientation is aligned with Z-axis and is expressed as $[\alpha_0\ \delta_0]^T$, while angle ψ defines the rotation around the boresight. The relations of the other rotation angles with the boresight are the following.

$$\theta = 90^\circ - \delta_0 \tag{2.5}$$

$$\phi = 90^\circ + \alpha_0 \tag{2.6}$$

By combining equations 2.3, 2.4 ,2.5, and 2.6, the star's sensor body coordinates are expressed below.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha\cos\delta \\ \sin\alpha\cos\delta \\ \sin\delta \end{bmatrix} \tag{2.7}$$

The second transformation represents the projection of a point from the real world to the two-dimensional image plane. A slightly modified image plane with its origin at the center of the sensor's surface O' is depicted in Figure 2.2. This transformation is named "perspective projection" and is defined by Horn [36] as follows.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{2.8}$$

where $[xy]^T$ are the image plane coordinates of the point's projection in the continuous domain and f the focal length. In order to acquire the coordinates expressed in pixels $[x'\ y']^T$, scaling, using the image resolution N_x, N_y, and translation to the top-left corner, using the distances between the old and the new plane origin d_x, d_y, are performed using the following formula.

$$x' = d_x - \frac{x}{N_x} \tag{2.9}$$

$$y' = d_y - \frac{y}{N_y} \tag{2.10}$$

The explained process describes mathematically how a digital image is formed. The reverse process of converting the 2D image plane coordinates to the 3D ECI coordinates is called *Position Calibration* and is performed before matching in order to acquire a detected star's position in the inertial space. These 3D vectors are the actual input of the matching algorithm and are used to find the angular distance between stars.

More about image formation, optical systems, computer vision and photogrammetry can be found in literature. Horn [36], Forsyth et al.[37], Castleman [40], Hecht [41] and Saleh et al. [42] provide fundamental knowledge on these fields.

2.1.2 Accuracy Measurement

Similar to the coordinate systems case, two types of error and accuracy are typically used in space navigation applications. The basic theory of accuracy-error measurement is covered next.

2.1.2.1 Boresight Accuracy (Sensor body coordinate system based)

As already defined in Section 2.1.1.2, the optical axis or boresight is the perpendicular from the camera center to the image plane, meaning that boresight is actually the direction along which the camera-lens system is pointing. A star tracker's accuracy performance is typically expressed in terms of angular distance error with respect to the boresight. For commercial products two error measurements are usually provided, which both have several names.

Cross (also called across, x-y, pitch-yaw) boresight error is related to the location of image plane on the celestial sphere. About (also called z, roll) boresight error is related to the rotation angle of image plane on the celestial sphere or equivalently to the twist around boresight. Figure 2.5 illustrates the axes that are used in order to determine motion with respect to the boresight.

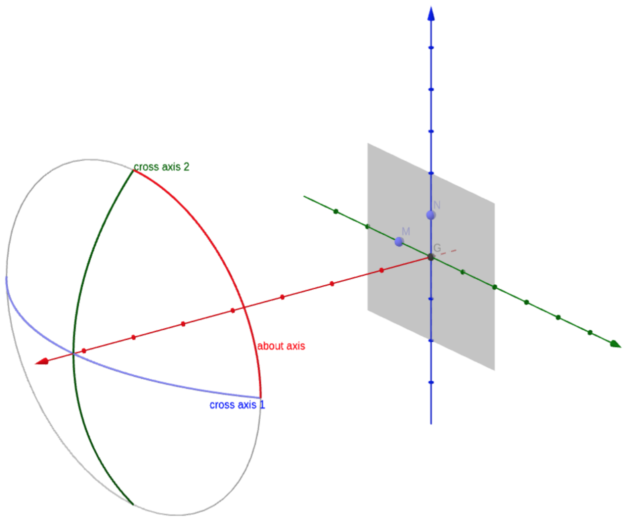


Figure 2.5: Illustration of the axes used to determine boresight accuracy performance with respect to the camera-lens system. In the diagram are displayed the following; gray plane: camera sensor, red axis: boresight, gray circle: FOV, green-blue axes: cross boresight motion, red circle: about boresight rotation. Image is taken from website with unknown author [4].

For a given camera-lens system, fixed on the axis origin and aligned with the gray plane, Field Of View is denoted as a gray circle and boresight as a red axis. Blue and green axes form a cross and the rotation of the satellite about them is related to the cross boresight accuracy. Similarly, the rotation about the red axis has to do with the about boresight performance. A rotation about the boresight along with rotations about the cross-axes constitutes the combined attitude accuracy of the star sensor. An analytical definition of both these accuracies is provided in [43]. However such an extended analysis is out of the scope of this thesis and therefore only a graphical description is given next.

The angular accuracy regarding the boresight directions can be better understood by observing Figures 2.6, 2.7. Remember that in digital systems accuracy is tightly related with resolution due to quantization. Quantization step size defines the minimum unit of reference that can be used, which in case of image processing is a pixel. Its attributes (i.e., number and size of pixels in an image) originate from the quantization parameters.

In Figure 2.6 the side view of a star tracker is displayed, where pixels are shown as circles on the right and focal length is marked as l. The cross boresight accuracy depends on the angle that the sensor would need to rotate for a star's information (i.e., intensity) to move from one pixel to its neighbor. For a given sensor size, higher resolution means smaller pixels across each direction. An increase in resolution or focal distance can accordingly increase the system's cross boresight accuracy.

In Figure 2.7 the front view of a star tracker is displayed, where pixels are shown as circles and sensor's size is marked as r. Similarly, the about boresight accuracy is equal to the angle that the

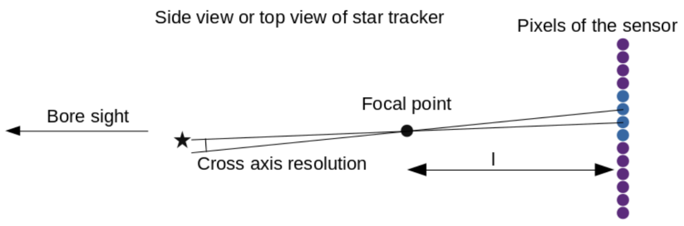


Figure 2.6: Cross boresight accuracy explained. Image is taken from website with unknown author [4].

sensor would need to rotate for a pixel's information to move to its neighbor. It depends not only on the sensor's size but also on the position of the captured star on the image plane. Increasing the sensor's size or the resolution leads to increased accuracy. The closer to the boresight the captured star is located, the worse the about boresight error gets.

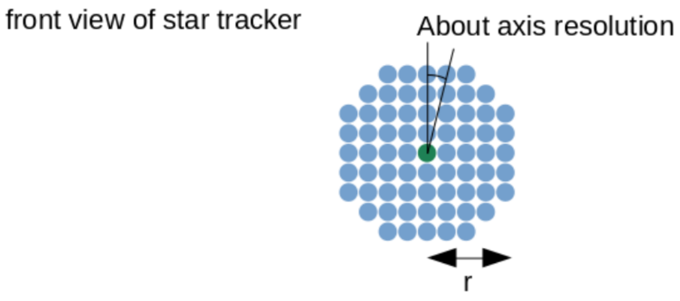


Figure 2.7: About boresight accuracy explained. Image is taken from website with unknown author [4].

After the provided descriptive explanation of the two most important error measurements, it should be noted that accuracy is usually quoted in 1σ and 3σ values. For narrow-angle star trackers, the about boresight error is proven to be about 6-16 times worse than cross boresight error [31, 44, 45]. Commercial products' datasheets usually focus on the cross boresight accuracy and therefore performance in Table 1.1 is expressed with respect to the same metric system.

Tan et al. [46] proposes a novel approach on compensating the cross boresight error induced due to the complex dynamic conditions that describe a star tracker's motion. In his work, he correlates adjacent star images together with their angular relations, which are determined by a gyroscope sensor. Increased compensation performance is achieved when 2 different cameras are used onboard.

2.1.2.2 Pixel Accuracy (Image plane coordinate system based)

It has been explained that the calculated centroids are expressed using two-dimensional coordinates with respect to the image plane. In order to estimate the accuracy of the related algorithm, it is necessary to use an error metric, that is derived from the same coordinate system and can simplify calculations. The appropriate metric is the two-dimensional euclidean distance between the calculated and the true (ground truth) centroid and is expressed in the pixel level. If the ground truth centroid is denoted as $\mathbf{c} = [x_c \; y_c]$ and the estimated centroid as $\hat{\mathbf{c}} = [\hat{x}_c \; \hat{y}_c]$ the pixel-level centroiding error is calculated as follows.

$$e = \sqrt{(\hat{x}_c - x_c)^2 + (\hat{y}_c - y_c)^2} \tag{2.11}$$

For the rest of this thesis, this definition of the centroiding error will be used.

2.1.3 Field of View

Until now, it should be clear that accuracy with respect to the boresight is tightly connected with the sensor body coordinate system, while accuracy with respect to pixels is correspondingly related with the image plane coordinate system. As a result, the pixel-level error has no physical substance by itself but its use is essential in digital signal processing problems. Similar to the correlation between the two coordinate systems that has been previously described, a correspondence between pixel-level and boresight accuracies should be found in order to be able to estimate the overall system's performance based only on software results.

It is known that the (angular) Field of View of an optical sensor is the solid angle through which the sensor is sensitive to electromagnetic radiation. It is a camera-sensor configuration specific feature and is usually found in the provided datasheets. It is related to the focal length f and the sensor size h, that is the photosensitive area, which can be derived by the product of total number of pixels along a single dimension and pixel's size. FOV if not given, can be calculated as shown below.

$$FOV = 2\arctan(\frac{h}{2f}) \tag{2.12}$$

Apart from the usefulness of FOV as a system specific feature, it can be used in order to convert from pixel accuracy to boresight accuracy and conversely. For a given camera-sensor configuration the FOV and the total number of pixels along both dimensions are known. As the FOV expresses the part of the real world, in degrees, that is visible through the camera and this part is projected onto a finite number of pixels, the part of the 3D scene that a pixel covers can be found using 2.13.

$$\frac{\text{degrees}}{\text{pixel}} = \frac{\text{FOV}}{\text{Number of Pixels}} \tag{2.13}$$

This mathematical relationship gives the number of degrees of the FOV, or equivalently the number of degrees of the real world view, that correspond to a pixel within the captured image. For example, let's consider the technical characteristics of the ASTRO APS product described in Section 1.2.3. FOV is 20°x20° and the detector resolution is 1024x1024 pixels. Consequently, each pixel covers $\frac{20}{1024} = 0.0195°$ of the FOV. An arcsecond is $\frac{1}{3600}$e of a degree, that is 2.7777×10^{-4}°. That means that each pixel corresponds to $\frac{0.0195}{0.0002777} = 70.3125''$ (arcseconds).

It has already been highlighted that state-of-the-art implementations such as the one currently analyzed have accuracies better than 10 arcseconds. ASTRO APS is characterized by error lower than 1 arcsecond (1σ) cross boresight. This level of accuracy corresponds to accuracy around $\frac{1}{70.3125} = 0.0142$ pixels, which is of subpixel level. This analysis clarifies why the so called subpixel performance of the Star Detection phase is crucial.

2.1.4 Performance Limiting Factors

As every real world application, space navigation also suffers from real world sources of error. In this section, the primary reasons that affect the accuracy of the overall system are shortly discussed.

2.1.4.1 Application Nonspecific

Due to the non-existence of optimal optical devices, vision-based applications suffer from lens distortion. This effect appears as variations in image magnification and more specifically, it causes different projection of an object onto different sections of the image. For instance, a captured object's size depends on whether it is projected near the center or the edges of the image. In reality, these lens aberrations appear in many forms as Weng et al. [47] and Simon [48] have described. However because of lens' symmetry, the most common pattern of this effect is radial symmetry, which leads to displacement of a point in the radial direction. Lens aberrations are more intense when larger apertures are used.

This kind of optical imperfection induces one of the most significant errors and therefore appropriate precautions should be taken. Brown et al. [49] exploited the property of radial symmetry and expressed lens distortion as a high order polynomial of radial distance. If such a model is available, camera calibration must be performed in order to correct the problems that arise by this embedded distortion. Even if sensor manufacturers sometimes provide the polynomial coefficients, an iterative algorithm should be used in software before matching, for better results.

As part of the Sunsat microsatellite's development, Jacobs [50] proposed a fast and simple calibration procedure, based on Brown's model, which requires only a few stellar images. In his thesis, he describes a method that corrects both lens distortion, by determining the polynomial parameters, and errors occurring due to non-optimal placement of the sensor relative to the optical axis and centre of the lens. Generally, several models of lens distortion exist and each of them demands a different calibration technique. Ricolfe et al. [51] evaluated several models and proposed an appropriate calibration method.

Apart from camera calibration, care can also be taken during clustering process. Considering that this effect appears worse near the image's edges, a crop factor is usually introduced which limits the search area to a smaller rectangular. In this way the processing of very distorted clusters is avoided and propagated error is reduced. Ultimately, calibration determines the accuracy of angular measurements and consequently the overall system's performance. More about proposed calibration methods can be found in [52, 53].

Another source of error, which is always present in digital sensors, is noise. Even if the used instruments are highly resilient to space radiation, some noise is always absorbed. A simple and adequate model that can be used is the Additive White Gaussian Noise (AWGN). The clustering and centroiding algorithms are usually responsible for eradicating errors caused by noise. However, if for some reason high levels of noise exist, the centroiding error increases and the final results are inaccurate.

2.1.4.2 Application Specific

The problems mentioned in the previous section are to a great extent solved, as they concern several scientific fields and much relative research has been done. On the contrary, a number of application specific factors can indeed limit the accuracy performance of the star tracker.

Although extracted clusters are constrained over size, sunlight reflections on the star tracker window or nearby debris and satellites can create optical illusions that may confuse the star identification algorithms. Similarly, planets and other celestial bodies can appear as stars in the image. Infinite Orbits has explained that only a small fraction of the total stars seen in an image are used in attitude determination, e.g., 6 or 7 stars out of 50. Taking into account this guideline, the false stars can be safely filtered and excluded using the star catalogues during matching.

Care should also be taken during the mechanical construction of the satellite with regard to the relative placement of the various systems. If for example the propulsion system is placed close enough to the star tracker, thrusting gases could cause contamination and blur of the FOV and lead to poor performance.

The last limiting factor that is mentioned is related to the three biggest celestial bodies that are seen from locations close to earth, that is the earth, the sun and the moon. The presence of any of them in the FOV during image capturing can heavily decrease star tracker's performance. That happens because of either the covered proportion of the FOV or the high brightness of the body which makes stars disappear. In simple star trackers this case can severely harm the overall functionality. However, as already described in Section 1.2.3, state-of-the-art commercial star trackers can perform under some of these circumstances. Typically full moon is accepted in FOV and optional or built-in baffles allow the presence of sun up to a specific angle. ASTRO APS could also operate properly with a part of earth in sight. An increasingly popular trend against this problem suggests the usage of two sensors with different orientations onboard in order to still capture clear images even when one of them is blocked.

2.2 Field Programmable Gate Arrays

The main challenge of this thesis is the implementation of different algorithms on hardware, in order to accelerate the computationally intensive task of centroiding. In this section, why an FPGA is the most appropriate platform for acceleration and basic concepts of hardware design are covered.

2.2.1 Overview

A Field Programmable Gate Array (FPGA) is an Integrated Circuit (IC) which can be programmed for different algorithms after fabrication. It consists of thousands or millions logic cells that can be reconfigured in order to implement any digital circuit and any software algorithm. The basic structure of an FPGA is displayed in Figure 2.8 and is shortly described next.

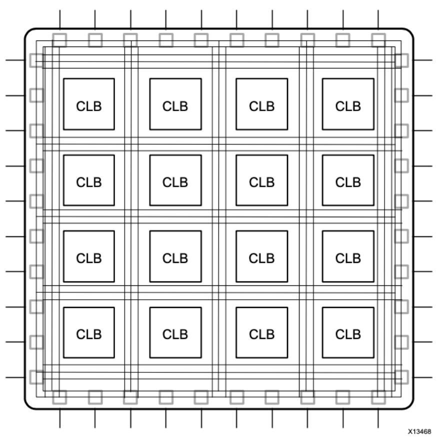


Figure 2.8: Basic FPGA structure. Image taken from Xilinx [5].

The basic logic block of an FPGA is called Configurable Logic Block (CLB) and consists of LUTs and FFs. A Look-Up Table (LUT) is the basic building block in current FPGAs and is used instead of logic gates. It is a truth table of N inputs which can implement a logic function of N variables which accesses 2^N memory locations [54]. The number of logic functions that an N-input truth table can implement is 2^{2^N}, while a typical case is $N = 6$ for Xilinx devices. When a large logic function must be implemented, it is divided in simpler pieces and each of them is assigned to a different LUT. LUTs can also be used as data storage elements. A Flip-Flop (FF) is the basic memory unit within the FPGA fabric and is always paired with a LUT to assist in logic pipelining and data storage.

The true performance advantages of an FPGA are derived from the incorporation of CLBs along with other computational and data storage blocks. The most complex computational block available is the DSP block, which is actually an Arithmetic Logic Unit (ALU) embedded in the programmable fabric. This dedicated core allows fast Multiplications-Accumulations (MAC) and its structure is given in Figure 2.9.

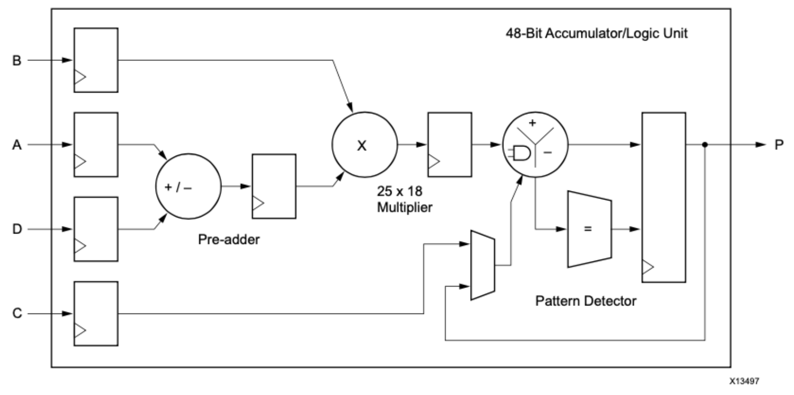


Figure 2.9: Structure of a DSP Block. Image taken from Xilinx [5].

The FPGA device contains also a number of embedded memory elements, among which are Block RAMs (BRAMs) and Shift Registers (SRLs). The BRAM is a dual-port RAM module instantiated into the FPGA fabric to provide on-chip storage for a relatively large set of data. When LUTs are used as memory elements, their content is specified during configuration. These are the fastest kind of memory available, because they can be instantiated in any part of the programmable fabric, that is as close as desired to the operation that uses memory's contents. The combination of these dedicated elements provides the FPGA with the flexibility to implement any software algorithm running on a processor.

2.2.2 Performance Advantages

Today FPGAs are widely used because they usually offer the same level of performance as Application-Specific Integrated Circuits (ASICs), while they provide significant cost advantages at the same time. Moreover, the fact that they can be dynamically reconfigured, in order to cover the needs of different application, makes them more flexible and the development time is considerably lower.

Compared to convenient processor architectures, the structures that comprise the FPGA fabric enable a high degree of parallelism in application execution. Two are the main factors that allow computationally intensive algorithms to run faster on an FPGA rather than on a CPU. The first feature of such algorithm is the requirement of a great number of operations. In processor architectures there is only a limited number of ALUs available, that must be shared for all operations. This case of limited resources leads to sequential execution of instructions. On the contrary, on FPGA independent elements, for example LUTs, are instantiated for each computation, thus instruction parallelism is maximized. In addition, computation algorithms usually need to process a large amount of data, which means that memory accessing and architecture are also important. It is known that processors suffer from time consuming accessing to external memories. In FP-

GAs however, fast memory architectures, spatially close to operations, are built to fit data layout, meaning that the memory arrangement is actually tailored to the application.

Apparently, the big performance advantages are derived from the fact that when FPGAs are targeted, the execution of a program is done in a custom program-specific circuit. Generally, FPGAs demonstrate at least 10x the performance of a processor for computationally intensive applications [5].

2.2.3 System-on-Chip FPGAs

A System-on-Chip is an IC that integrates a number of computing and other electronic components, among which are a processor, memories and analog to digital converters (ADC), in a single chip. Such tightly integrated architectures reduce power consumption, semiconductor die area and cost less, while performance is heavily improved. Today, SoCs are widely used in the embedded computing industry.

A SoC FPGA is an heterogenous platform which integrates together a CPU and programmable hardware, that is an FPGA, along with memory interfaces and other peripherals. This architecture allows lower consumption of power, improvements in chip sizes and generally increases performance, compared to older systems, where CPU and FPGA were connected through a central interfacing circuit board. Perhaps the most important feature of a SoC FPGA is that provides low communication overhead, therefore enabling efficient software-hardware co-design, which results in increased productivity. A schematic that illustrates a typical co-design flow in a SoC FPGA is given in Figure 2.10.

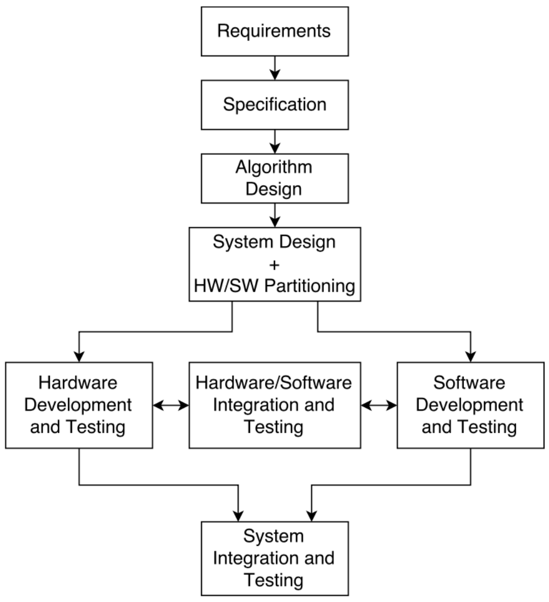

Requirements
Specification
Algorithm Design
System Design + HW/SW Partitioning
Hardware Development and Testing
Hardware/Software Integration and Testing
Software Development and Testing
System Integration and Testing

Figure 2.10: Software-Hardware Co-Design Flow. Image taken from lecture notes [6].

The most significant feature of this design flow is the offered convenience in developing software and hardware independently. Typically, parts of an algorithm that are computationally intensive are "redirected" to the FPGA side of the system for development. Respectively, parts that are changing dynamically or correspond to control logic are implemented on the processor side. Ultimately, both parts are integrated into one that meets the system's functionality. Today SoC FPGAs outperform standalone CPUs in terms of performance and energy efficiency.

For the purposes of this research, a device from the ZYNQ-7000 SoC family, provided by Xilinx, will be used. These products integrate a dual-core ARM Cortex-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPUs are the heart of the PS which also includes on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces. A block diagram of a ZYNQ-7000 SoC is given in Figure 2.11.

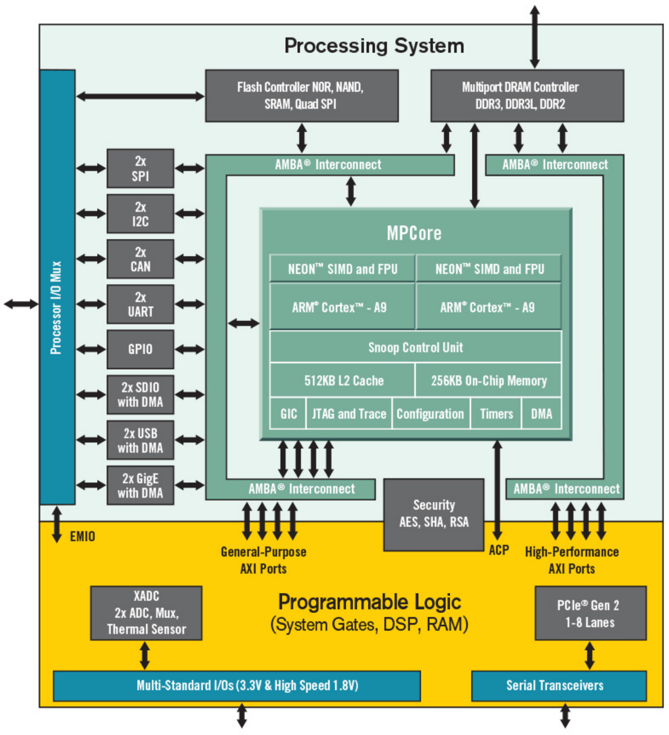


Figure 2.11: ZYNQ-7000 SoC Block Diagram. Image taken from Xilinx.

Xilinx adopts the AXI protocol, which is part of the Arm Advanced Microcontroller Bus Architecture (AMBA) family of microcontroller buses, in order to offer high performance data transfer between individual modules in the system.

As already stated, this thesis focuses on the PL side of the system in order to optimize centroiding algorithms in terms of execution speed. The parallel processing power of an FPGA can be exploited by adopting any of the two available FPGA programming models. As part of this work, both models have been used and evaluated. The special features and advantages of each model are the topic in the next two sections.

2.2.4 Register Transfer Level Programming Model: VHDL

The traditional programming model of an FPGA is centered on Register-Transfer Level (RTL) descriptions using a Hardware Description Language (HDL) like Verilog or VHDL. In the context of this project VHDL is used, which stands for Very (High Speed Integrated Circuit) Hardware Description Language. The objective in this programming model is to describe the functionality of the desired digital circuit, by using elements of the lowest level like wires, signals and by explicitly handling registers and clocks. In Figure 2.12 the development workflow, which is followed when an HDL is used, is illustrated.

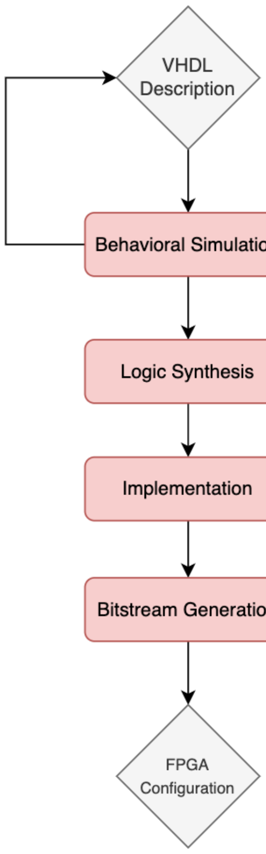


Figure 2.12: Register Transfer Level Development Workflow

Initially, the system should be modelled using the HDL. VHDL provides three different design styles. *Dataflow* style describes how data moves through the circuit by using concurrent statements. In *Structural* description complex logic functions are composed by hierarchically interconnecting simple building blocks. *Behavioral* description handles the circuit as a "black box" and just models the overall behaviour in a more abstract fashion.

Afterwards, *Behavioral Simulation* is executed in order to test the logical functionality of the system. In reality this is just a software simulation that tests if the given RTL description is correct in terms of behavior. The inputs of the Behavioral Simulation are the VHDL code, that describes the design, and a testbench, written also in VHDL, that acts as stimuli to the circuit and provides different input scenarios. This step is used for fast code debugging before the time consuming processes that follow are carried out.

The next step is *Logic Synthesis*, during which the FPGA synthesis tool (i.e., Vivado Synthesizer) that works as a compiler, generates a gate-level netlist given an RTL description. Synthesis breaks the desired function into logic elements (i.e., LUTs, DSP etc.) and defines the connections between them. These results are actually an estimate of how the function is going to be implemented on the FPGA. *Post-Synthesis Simulation* is available in order to test if the generated RTL retains the correct functionality.

The penultimate step defines how the synthesized design, that is the generated netlist, will be implemented on the targeted FPGA. *Implementation* consists of three processes, mapping, placement and routing. During mapping, optimizations on the given netlist are performed and the logic function is translated to available components. Placement determines where each piece of logic will be placed in the array of logic cells in order to minimize area and wire lengths. Finally, given the placement results, routing finds a valid pattern of wires to connect the used components, while aiming for area and wire delay minimization. Such wires are illustrated in Figure 2.8, where they surround CLBs in an island-like style. Following implementation, *Timing Simulation* can provide useful information about system's functionality with respect to the given timing constraints.

Finally, a bitstream is generated and loaded into the FPGA's ROM in order to configure it and implement the desired circuit.

2.2.5 High Level Programming Model: C++

As the complexity of a logic function or a digital circuit increases, the complexity of the respective VHDL code increases in an exponential manner. That converts system design using VHDL in a hard and time consuming task. For this reason, a High Level programming model that removes the differences in programming models between conventional CPUs and FPGAs, and therefore leads to decreased design complexity, has been developed.

High Level Synthesis (HLS) is an automated design process that transforms a high-level functional specification to an optimized RTL description (netlist) suitable for hardware implementation. The usage of a high-level language inserts an abstraction level between hardware and designer, which makes system development easier and faster. In the context of this project, C++ language will be used. The development workflow, which is defined from this high-level approach, is illustrated in Figure 2.13.

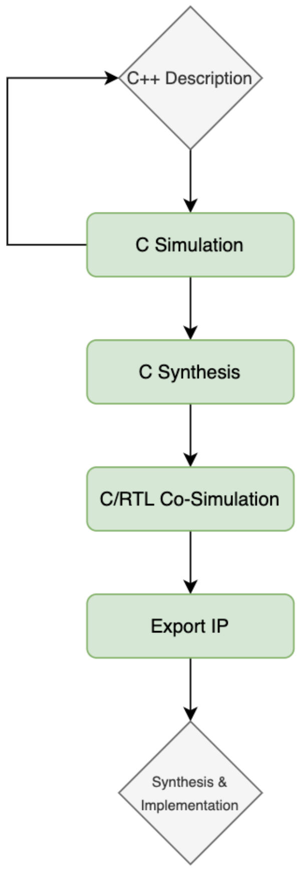


Figure 2.13: High Level Development Workflow

Initially, the system should be modelled using a high-level language, i.e., C++. One of the most significant advantages that the HL, over the traditional RTL approach, exhibits, is that it offers fast validation at the C-level. *C Simulation* step is used to test the functional correctness of the C++ algorithm before synthesizing it into a netlist (Pre-Synthesis Validation). This can be done by creating a C++ testbench, a process which is multiple times faster and easier than designing an RTL testbench.

When a C++ specification is proven to operate properly, *C Synthesis* is executed to create the RTL implementation. During this process, Vivado HLS Compiler extracts datapath and control from the source code. Datapath is matched to operations and control extraction creates a Finite State Machine (FSM) that sequences the operations in the RTL design. C Synthesis consists of two phases, Scheduling and Binding. During scheduling the compiler determines which operations occur at each clock cycle and binding determines which library core (logic cell) will be used for each operation. The C Synthesis process is controlled and guided by user-defined constraints and directives, regarding area and performance. This control over the Synthesis process can lead to specific high-performance hardware implementations and enables rapid design space exploration, which increases the likelihood of finding an optimal implementation.

The actual final step in this workflow is *RTL Verification* and is used to verify that the generated RTL implementation is functionally identical to the original C++ code. HLS tool has automated this process, as it uses the already designed C++ testbench for this purpose. Because both the RTL and C++ outputs are used during verification, it is also called *C/RTL Co-Simulation*. It should also be mentioned that the HLS tool can generate trace files that show the activity of the waveforms in the RTL design and which can be used to analyze and understand the RTL output, exactly as it is done using the Post-Synthesis Simulation in the RTL-based approach.

When the hardware implementation that achieves the desired performance is created and verified, the RTL can be exported and packaged as IP either for use in a larger system or in order to generate the bitstream and configure the FPGA. It should be highlighted that before exporting a design, logic synthesis and implementation can be executed in order to confirm the accuracy of the estimates made by the compiler during C Synthesis regarding performance and area.

Throughout Chapter 4, where the hardware implementations are discussed, the specific characteristics, advantages and disandantages of each FPGA programming model will be made clear.

2.2.6 Hardware Performance Metrics

The final part of theory on hardware design that should be covered is related to the metrics used for performance estimation. The degree of acceleration that is achieved when implementing an algorithm on an FPGA is related to three different factors.

The first one is the *Critical Path*, that is defined as the slowest path in a circuit. This path consists of logic delay, which is the effective delay due to operations, and net delay, which is determined by the routing process. In complex designs which require large percentages of the available resources, it is possible that the placement and routing problems cannot be solved optimal, which can result to congestion and increased net delays.

Consider that a critical path is found between a source point A and a destination point B, where A and B are points located inside the circuit. Critical path actually determines the minimum clock cycle length allowed, in order for a signal starting from A to cross the entire path and arrive at B on time, that is before the next clock cycle. Apparently critical path defines circuit's maximum operating frequency, which is given by the following formula.

$$f_{max} = \frac{1}{clock_{min}} = \frac{1}{CP} \tag{2.14}$$

The second factor is *Latency*, which is the number of clock cycles required for an operation, a function or a circuit to be completed. The third factor is called *Throughput*, which is the rate at which the system accepts new inputs. This feature typically determines the overall performance of the design.

The design call usually made in FPGA implementations is to create fully pipelined designs. This term refers to the process of segmenting data paths as much as possible by adding intermediate registers. The generated smaller paths contain the least possible logic which makes them faster. In this way, the maximum operating frequency, the throughput and the latency are all increased and overall performance is improved.

Another metric that should always be looked after is *Area*, which refers to the amount of available hardware resources required to implement the design. Among others these resources include LUTs, FFs and DSPs. Converting designs to fully-pipelined may lead to significant increase in resource utilization. The larger the resource utilization, the greater the possibility of congestion which can lead to sub-optimal implementation. Remember that a basic constraint when developing on a specific FPGA is to not exceed the finite resources it provides.

2.3 Related Work

In the preceding sections a number of sources have been cited, regarding the performance standards of modern star trackers and calibration methods. Because the main topic of this thesis is the star detection phase, and more specifically the centroiding process, it is essential to lay emphasis on previous work on related algorithms and FPGA implementations.

2.3.1 Processing Algorithms

Due to the increasing popularity of space navigation applications, a lot of researchers have focused on developing and evaluating star detection or identification algorithms.

2.3.1.1 Clustering

Researchers from University of Stellenbosch seems to have been intensively concerned with space navigation over the last decades. As part of developing the Stellenbosch UNiversity SATellite (SUNSAT), Steyn et al. [55] implemented a modified Region Growing Algorithm to accelerate clustering, which was initially proposed in [56]. The main idea was instead of checking all the pixels in the image plane, to exploit the fact that the minimum size of a valid cluster is 3x3 pixels and therefore access only every third pixel per row and column. The result was an increase in speed of almost one order of magnitude. When a pixel was found to be brighter than the given threshold, the area would star to grow by examining the four surrounding pixels. In this algorithm, it is assumed that clusters can be of arbitrary shape.

Zhu et al. [57] proposed a novel clustering process, in which valid star pixels are linked by pointers and stored in a cross-linked list. This list can then be used as input for the Region Growing Algorithm, which starts with the first element in the list. This approach completes clustering with a rate of 16 milliseconds per 1024x1024 frame.

2.3.1.2 Centroiding

According to Liebe [31], the traditional approach to the centroid calculation problem suggests the use of a Center of Gravity algorithm (CG). More on this approach will be discussed in Chapter 3. Many variations and optimized versions of the CG algorithm are available in literature.

SUNSAT engineers [55] used a modified algorithm, in which the multiplying factor in the numerator is not the the row/col number itself but a translated by 0.5 and scaled by the pixel size version of it. They showed that the original CG algorithm can provide estimates with error less than 0.2 pixels. Knutson [58] also adopted a modified version, in which the noise threshold was subtracted from a pixel's intensity before it was multiplied, in order to decrease computation time. Also, this modification makes the calculation less sensitive to background noise [59].

In [60, 61] it has been proven that a weighted center of gravity algorithm (WCG), in which sometimes for simplicity the weights are approximated by the corresponding pixels' intensities (squared CG), improves the accuracy of the centering approach. However, that does not apply when the initial estimate of the CG is significantly false, for example due to high background noise.

Akondi et al. [62] proposed an iterative calculation of the centroid, while the weights are updated in each step. Generally the accuracy of the Iterative WCG is higher but the computation is significantly slower.

Except for the very popular CG approach, there is also a family of fitting methods which is expected to yield the most accurate results [63]. These algorithms approximate a star as a 2D Gaussian distribution and aim to find the Gaussian function, that is its parameters, that fits better a given bright region.

Delabie et al. [63] provide an in-depth analysis and evaluation of the Gaussian fitting methods, which refer actually to a nonlinear least squares optimization problem. The most accurate approach is the two-dimensional, in which an asymmetric 2D Gaussian function is fitted to the data. They also present a less computationally intensive approach, in which the 2D problem is divided to two simpler 1D problems. This yields decent results when sensors with smaller FOVs are used. A reduced version of the 1D problem is also proposed, according to which the outermost pixels of a cluster are neglected as they are characterized by relatively low SNR (Signal to Noise Ratio). All of the three problems are solved with the Levenberg-Marquardt algorithm.

In this paper, a novel Gaussian Fitting method, called Gaussian Grid Algorithm, is also proposed. This approach transforms the initial complex problem to a linear least squares problem using the logarithm function and expresses each pixel in the square cluster with respect to the central one. In order to achieve the best possible accuracy, demonstrated by the 2D Gaussian Fitting Algorithm, the solutions of the Gaussian Grid are only used as initial parameters to the nonlinear problem. In this way, the highest accuracy of the 2D Gaussian Fitting Algorithm is obtained, while the computation time is by 33% lower for a 5x5 cluster.

Another attempt to address the shortcomings of the standard 2D Gaussian Fitting method, which are the increased computation time and sensitivity to the initial parameters, was made in [64]. The Gaussian Analysis algorithm (GA) finds a closed-form solution of the Gaussian parameters, based on the marginal distribution of the intensity accumulation on both dimensions, with significant time advantages. It is claimed to be the most accurate and efficient algorithm when the size of the star spot is not bigger than 5x5 pixels. However, this method was validated only with ideal, that is without noise, images.

Wan et al. [7] have proposed a novel Fast Gaussian Fitting Algorithm, which also transforms the initial problem to a linear least squares problem. This approach is extensively discussed in Chapter 3.

Lastly, in [65] a method with information theoretic weighting, in which signals are weighted with a confidence interval based on the information content, was presented and evaluated against the more traditional thresholding methods. However, this kind of approach is out of the scope of this research and will be no discussed further.

2.3.1.3 Matching

Star Identification is a much more complex process and is therefore implemented on the processor side of the system. Since the first generation of star trackers, there have been a lot of matching algorithms developed.

Erlank [66], as part of the CubeStar star tracker development project, classified matching algorithms as subgraph and pattern matching. He described and compared the subgraph-based Triangle and Match Group algorithms and the pattern-based Grid algorithm. He implemented however, the Geometric Voting Algorithm, which was earlier published by Kolomenkin [67].

Wang et al. [19] proposed a false star filtering algorithm which can be used to increase the robustness of a convenient matching algorithm when high levels of noise are observed. Their algorithm achieves rejection of more than 700 false stars in less than 10 frames.

A modern method using a 1D Convolutional Neural Network is proposed in [68]. It seems highly robust to noise and false stars while it achieves decent performance under dynamic conditions. In the coming years, a significant boost in the contribution of Deep Learning to the Star Identification phase is expected.

In-depth analysis and evaluation of star identification algorithms is provided in [69, 70].

2.3.2 FPGA Implementations

Von Wielligh [1] designed and validated an end to end star tracker which achieves 10 Hz update rate in Lost In Space mode. He implemented clustering and centroiding as one integrated process on the FPGA. Clustering was based on a modified version of the Growing Region Algorithm, refined for streaming pixel data. In order to optimize the clustering algorithm for hardware implementation, he adapted the idea of a rotating FIFO, described by Lindh [71]. For centroiding, the alternative Center of Gravity method, in which the noise threshold is subtracted from the intensity values, was used. The overall system demonstrates subpixel centroid detection at a rate of 98% but attitude estimation was not so consistent due to a high percentage of false matches, responsible for which was the matching algorithm.

Comparing Von Wielligh's work with earlier star trackers which share the same processing approach (i.e. Growing Region Algorithm and Center of Gravity), like CubeStar [66] or SUNSAT [55], is a good paradigm of how the modern SoCs significantly improve performance. In these older systems, before clustering was performed, the whole frames should be saved in memory and pixel streaming was not possible. These waiting periods reduced the overall performance. Another thing to mention was that CubeStar connected an FPGA board to a CPU board. However the FPGA, due to the significant lower performance standards of the time, was only used as a buffer between the sensor and the processor, in order to reduce the rate at which the data were driven to the processor.

Zhou et al. [72] proposed a two-step star detection FPGA implementation. During clustering a unique threshold for each frame was obtained. The first step defines the central pixel of a cluster, by calculating the first derivative along both dimensions. Afterwards, the subpixel centroid was determined using the standard CG calculation. It was proven that this approach demonstrates better results than a standard one-step method only when high levels of background noise are present.

Wang et al. [73] proposed an FPGA-based centroid extraction method with a dynamic rooted tree architecture. This method merges the equivalence table in the process of scanning, such that only one scan of the image is needed, and allows processing of irregularly shaped star spots. Lastly a novel method implemented on FPGA, which calculates centroids using the First Fourier Coefficient is proposed in [74].

Chapter 3

Considered Centroiding Algorithms

In this thesis two centroiding algorithms that demonstrate different accuracy, computation speed and complexity are used. These attributes result to significantly different implementations on hardware. In this chapter, a thorough description of each algorithm is provided.

3.1 Center of Gravity Algorithm

As already stated, the use of the Center of Gravity (CG) Algorithm constitutes the most traditional approach to the centroiding problem and it has been adapted by the most of the earlier projects in literature. It is also called Center of Mass (CM) or Mass Center Algorithm.

3.1.1 Definition

Consider a cluster, each pixel of which was given an index i based on the raster scan method. Then each pixel is denoted with its coordinates in the image plane (x_i, y_i), where x_i refers to the column number and y_i to the row number. Remember that this notation is reversed to the conventional digital image notation, in which the first dimension refers to the row number.

The simple formula which defines the centroid pair (x_c, y_c) for a cluster is given below.

$$x_c = \frac{\sum I_i x_i}{\sum I_i} \tag{3.1}$$

$$y_c = \frac{\sum I_i y_i}{\sum I_i} \tag{3.2}$$

where the sum is computed over all pixels i of the cluster and I_i refers to the grayscale intensity of a pixel, also called brightness or luminosity.

This is the less complex and faster centroiding algorithm available, but it demonstrates also a performance limitation, which is partially derived from it's high sensitivity to background noise [59]. That can be better understood, if one consider a not so well defined cluster, which contains some false pixels too. For example this could be the result of a non-optimal thresholding process, as these pixels would not be really derived from the spread light source but they would have a value sufficiently high to be included in the cluster. Due to the simple average mean calculation, these pixels negatively affect the centroid location and contribute to less accurate estimation. In order to mitigate this effect, a number of techniques can be used during the clustering process. However this topic is out of the scope of this research. Generally, the accuracy of this algorithm is also restricted by the S-curve error which is inherent in the sampling process due to undersampling [75]. Nevertheless, it is expected that the CG algorithm will provide centroid estimates with a minimum accuracy of 0.2 pixels [50].

3.1.2 Proposed Modification for Implementation on FPGA

When an algorithm is intended to run on hardware, it should be optimized in advance, in order to fully exploit the benefits that the FPGA platform provides. One of the most basic principles in hardware design is to minimize the bit width of each signal. That reduces the resources and the logic levels required to complete an operation and results in an optimized design.

In the context of this work, one of the most important assumptions is that each star is spread over square regions, which means that each provided cluster is of square shape. This assumption is based on the guidelines given by Infinite Orbits, as their camera-sensor system is configured in a manner to provide this type of data. More on this topic will be discussed in Chapter 4.
Given a cluster of N pixels, Equations 3.1, 3.2 can be written also as:

$$x_c = \frac{\sum\limits_{i=0}^{N} I_i x_i}{\sum\limits_{i=0}^{N} I_i} \tag{3.3}$$

$$y_c = \frac{\sum\limits_{i=0}^{N} I_i y_i}{\sum\limits_{i=0}^{N} I_i} \tag{3.4}$$

Based on the above assumption, each cluster contains a number of consecutive rows and columns. If the starting row and column are denoted as Y and X respectively, equations will be:

$$x_c = \frac{\sum\limits_{i=0}^{N} I_i (X + \tilde{x}_i)}{\sum\limits_{i=0}^{N} I_i} \tag{3.5}$$

$$y_c = \frac{\sum\limits_{i=0}^{N} I_i (Y + \tilde{y}_i)}{\sum\limits_{i=0}^{N} I_i} \tag{3.6}$$

where ($\tilde{x}_i$, $\tilde{y}_i$) is a pixel's distance to the first column and row in pixels. This distance expresses actually the relative coordinates in the cluster, compared to the absolute coordinates (x_i, y_i) in the image plane. For a 3x3 cluster $\tilde{x}_i$ gets 3 different values, that are 0, 1, 2. $\tilde{x}_i = 0$ is true for the 3 pixels of the first column, $\tilde{x}_i = 1$ for the 3 pixels of the second column and $\tilde{x}_i = 2$ for the 3 pixels of the third column. The same applies to the second dimension. Based on simple mathematical properties, Equation 3.5 will be:

$$x_c = \frac{\sum_{i=0}^{N} I_i X + I_i \tilde{x}_i}{\sum_{i=0}^{N} I_i} = \frac{\sum_{i=0}^{N} I_i X}{\sum_{i=0}^{N} I_i} + \frac{\sum_{i=0}^{N} I_i \tilde{x}_i}{\sum_{i=0}^{N} I_i} = \frac{X \sum_{i=0}^{N} I_i}{\sum_{i=0}^{N} I_i} + \frac{\sum_{i=0}^{N} I_i \tilde{x}_i}{\sum_{i=0}^{N} I_i} \implies$$
$$\implies x_c = X + \frac{\sum_{i=0}^{N} I_i \tilde{x}_i}{\sum_{i=0}^{N} I_i} \tag{3.7}$$

Similarly, Equation 3.6 transforms to:

$$y_c = Y + \frac{\sum_{i=0}^{N} I_i \tilde{y}_i}{\sum_{i=0}^{N} I_i} \tag{3.8}$$

A simple example is given, in order to demonstrate the benefits that this simple mathematical modification provides. Consider an extracted 3x3 cluster which starts at row 300 and column 580 of a frame, as displayed below.

300	4	5	6
301	7	8	1
302	2	3	4
	580	581	582

The value in each pixel represents its intensity. These values have been chosen just to ease the example's calculations, although the observed intensities in the real application will be much higher. If Equations 3.3, 3.4 were used, the calculations would be the following.

$$x_c = \frac{\sum_{i=0}^{9} I_i x_i}{\sum_{i=0}^{N} I_i} = \frac{580 \cdot 4 + 581 \cdot 5 + 582 \cdot 6 + 580 \cdot 7 + 581 \cdot 8 + 582 \cdot 1 + 580 \cdot 2 + 581 \cdot 3 + 582 \cdot 4}{4+5+6+7+8+1+2+3+4}$$
$$= \frac{\mathbf{580}(4+7+2) + \mathbf{581}(5+8+3) + \mathbf{582}(6+1+4)}{4+5+6+7+8+1+2+3+4} = \frac{23238}{40} = 580.95$$

$$y_c = \frac{\sum_{i=0}^{9} I_i y_i}{\sum_{i=0}^{N} I_i} = \frac{300\cdot4+301\cdot7+302\cdot2+300\cdot5+301\cdot8+302\cdot3+300\cdot6+301\cdot1+302\cdot4}{4+5+6+7+8+1+2+3+4}$$

$$= \frac{\mathbf{300}(4+5+6)+\mathbf{301}(7+8+1)+\mathbf{302}(2+3+4)}{4+5+6+7+8+1+2+3+4} = \frac{12024}{40} = 300.85$$

The same result is yielded if Equations 3.7 and 3.8 are used.

$$x_c = \frac{\sum_{i=0}^{9} I_i x_i}{\sum_{i=0}^{N} I_i} = 580 + \frac{0\cdot4+1\cdot5+2\cdot6+0\cdot7+1\cdot8+2\cdot1+0\cdot2+1\cdot3+2\cdot4}{4+5+6+7+8+1+2+3+4}$$

$$= 580 + \frac{\mathbf{0}(4+7+2)+\mathbf{1}(5+8+3)+\mathbf{2}(6+1+4)}{4+5+6+7+8+1+2+3+4} = 580 + \frac{38}{40} = 580.95$$

$$y_c = \frac{\sum_{i=0}^{9} I_i y_i}{\sum_{i=0}^{N} I_i} = 300 + \frac{0\cdot4+1\cdot7+2\cdot2+0\cdot5+1\cdot8+2\cdot3+0\cdot6+1\cdot1+2\cdot4}{4+5+6+7+8+1+2+3+4}$$

$$= \frac{\mathbf{0}(4+5+6)+\mathbf{1}(7+8+1)+\mathbf{2}(2+3+4)}{4+5+6+7+8+1+2+3+4} = 300 + \frac{34}{40} = 300.85$$

The benefits of the proposed modification can be better understood by comparing the multipliers, marked with bold. The large multipliers in the first case are represented in hardware with more bits than the multipliers in the second case. In this specific example numbers 300 and 580 require 9 and 10 bits respectively, while the multipliers in the second case (0, 1, 2) are represented using only 1 and 2 bits.

It is known that generally the result of a multiplication of a n_1-bit number with a n_2-bit number requires n_1+n_2 bits to be represented correctly. Thus in the first case, significantly more resources are demanded for the partial products, resulting in a non-optimal design. This overhead is removed using the modified Equations 3.7, 3.8.

For the rest of this work, in order to simplify the notations, the sums' limits will be dropped and instead of the $\tilde{x}_i$, $\tilde{y}_i$ symbols, x_i and y_i will be used. The final form of the equations are shown below.

$$x_c = X + \frac{\sum I_i x_i}{\sum I_i} \tag{3.9}$$

$$y_c = Y + \frac{\sum I_i y_i}{\sum I_i} \tag{3.10}$$

3.2 Fast Gaussian Fitting Algorithm

3.2.1 Overview

The second considered algorithm belongs to the fitting methods family. As already mentioned, the sensor system is configured in a way to purposely spread the captured star over a wide area of pixels. Basic theory on how this is achieved using defocused optics can be found in [56]. The region formation can be modeled by a Point Spread Function (PSF). Fitting methods approximate this function as a 2D gaussian distribution and aim to estimate the Gaussian parameters, that is the Gaussian function which better fits the given data. The adopted algorithm was proposed by Wan et al. [7].

3.2.2 Motivation

Fitting methods rely on the assumption that the source intensity of each pixel i in a cluster can be expressed by a 2D Gaussian function, which is defined below.

$$S(x_i, y_i|\mathbf{v}) = A \exp\left(-\frac{(x_i - x_c)^2}{2\sigma_x^2} - \frac{(y_i - y_c)^2}{2\sigma_y^2}\right) \tag{3.11}$$

where (x_i, y_i) are the pixel's known coordinates on the image plane and $\mathbf{v} = (A, x_c, y_c, \sigma_x, \sigma_y)$ are the unknown Gaussian parameters that should be determined. A is the amplitude of the function and represents the brightness level of the star, meaning that as A increases the star spot gets brighter altogether; x_c, y_c are the centroids of the star and σ_x, σ_y represent the standard deviation of the function along both dimensions. Note that for the examined application only the x_c, y_c parameters need to be found.
Suppose that a received image signal for a pixel i is modeled as:

$$I_i = S_i + N_i \tag{3.12}$$

where I_i is the observed intensity, S_i the source (real) intensity and N_i is the added noise, which also conforms to a Gaussian distribution. The standard Gaussian Fitting Algorithm (GF) estimates the function parameters using the corresponding objective function, which is displayed below.

$$Z = \arg\min_{\mathbf{v}} \sum_{i \in U} [z_i]^2 \tag{3.13}$$

where $z_i = I_i - S_i = N_i$ represents the distance between the observed value I_i and the underlying Gaussian function's value, that should be minimized. If there is no noise in the images, i.e., $N_i = 0$, then the given cluster can be perfectly approached by a Gaussian function. However, this is not true in real applications and therefore the best possible approach should be found. U is the set which contains all the pixels of the cluster.

This approach is expected to give the most accurate results among all the available centroiding algorithms [59]. However, since Equation 3.13 is a nonlinear least squares problem, it is also the most complex one which results in a high computational burden and therefore makes it unaffordable for real-time applications. This problem can be solved with iterative algorithms, such as the Levenberg-Marquardt or the Trust-Region-Reflective. Although these algorithms are generally sensible to the initial parameters' choice and converge to local and not global minima, Delabie et al. [63] showed that these problems are not important in the examined context.

3.2.3 Definition

Even though a number of similar algorithms that don't solve Equation 3.13 and speed up the GF approach have been developed, none of them is able to demonstrate the same accuracy. The Fast Gaussian Fitting Algorithm (FGF) is used in this research, as it seems to be the only one that can approximate the solution of the GF in a closed-form without iterations and without loss in accuracy, while achieving high noise robustness and high efficiency, which is critical for the star tracking process. The basic theory of the FGF algorithm, taken by the relative paper, is discussed next.

The main idea is to transform the objective function to a simpler form by performing the logarithm operation on both I_i and S_i in Equation 3.13. The transformed objective function is now:

$$G = \arg\min_{\mathbf{v}} \sum_{i \in U} [g_i]^2 \tag{3.14}$$

where $g_i = \ln I_i - \ln S_i$. Apparently if Equations 3.13, 3.14 share the same solution, then the FGF is an accelerated version of the GF. This would be the case if there was no noise in the images, which is however not true. Therefore it should be expected that the two problems are probably not equivalent. Using simple mathematical properties, the term g_i is transformed as shown below.

$$g_i = \ln I_i - \ln S_i = \frac{N_i}{I_i}\left(1 + \frac{S_i}{N_i}\right)\ln\left(1 + \frac{N_i}{S_i}\right) = \frac{N_i}{I_i}\phi\left(\frac{S_i}{N_i}\right) \tag{3.15}$$

where $\phi\left(\frac{S_i}{N_i}\right) = \left(1 + \frac{S_i}{N_i}\right)\ln\left(1 + \frac{N_i}{S_i}\right)$. If $r_i = \frac{S_i}{N_i}$, then $\phi\left(\frac{S_i}{N_i}\right)$ can be written as:

$$\phi(r_i) = (1 + r_i)\ln\left(1 + \frac{1}{r_i}\right) \tag{3.16}$$

Note that $|r_i|$ represents the SNR of a pixel. The curve of $\phi(r_i)$ is illustrated in Figure 3.1.

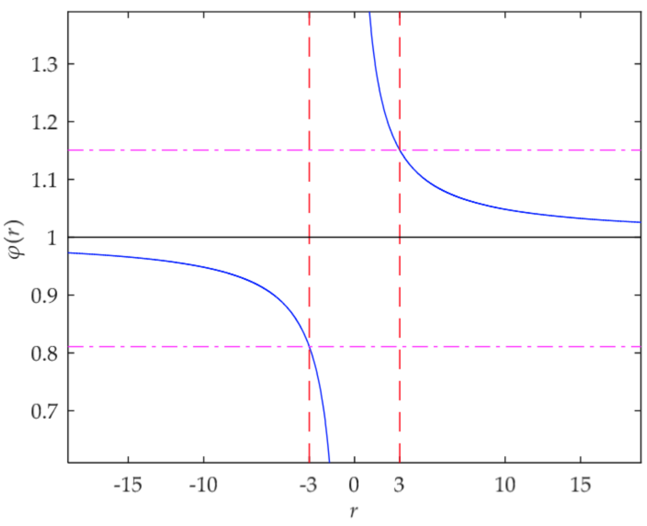


Figure 3.1: The curve of $\phi(r_i)$. The value of $\phi(r_i)$ converges to 1 with the increase of SNR. When r is equal to 3, the value of $\phi(r_i)$ is equal to 1.1507. When r is equal to -3, the value of $\phi(r_i)$ is equal to 0.8109. Image taken from [7].

Focus on the positive side of the $\phi(r)$ axis. As the SNR of a pixel increases, $\phi(r)$ converges to 1. By substituting Equation 3.12 into Equation 3.13 and if Equation 3.15 is multiplied by I_i, the objective functions are:

$$Z = \arg\min_{\mathbf{v}} \sum_{i \in U} [N_i]^2 \tag{3.17}$$

$$H = \arg\min_{\mathbf{v}} \sum_{i \in U} [\phi(r_i) N_i]^2 \tag{3.18}$$

This means that the FGF can be considered as a weighted GF with the weight of each pixel determined by $\phi(r_i)$. If pixels with SNR greater than a certain value are selected to ensure that $\phi(r)$ of these pixels is close enough to 1, Equation 3.18 is approximately equal to Equation 3.17 and therefore FGF provides the same results with GF.

Until now, the only actual modification to the FGF objective function was the multiplication with I_i. By writing function H with the initial notation, which was based on term g_i, we get a different form of the same objective function.

$$H = \arg\min_{\mathbf{v}} \sum_{i \in U} [I_i g_i]^2 = \arg\min_{\mathbf{v}} \sum_{i \in U} [I_i(\ln I_i - \ln S_i)]^2 \tag{3.19}$$

By substituting Equation 3.11 into Equation 3.19, the latter is converted to:

$$H = \arg\min_{\mathbf{v}} \sum_{i \in U} \left(\frac{I_i x_i^2}{2\sigma_x^2} + \frac{I_i y_i^2}{2\sigma_y^2} - \frac{I_i x_c x_i}{\sigma_x^2} - \frac{I_i y_c y_i}{\sigma_y^2} + \frac{I_i x_c^2}{2\sigma_x^2} + \frac{I_i y_c^2}{2\sigma_y^2} + I_i \ln I_i - I_i \ln A \right)^2 \tag{3.20}$$

Let us define the parameters m, n, p, q and k as it is shown below.

$$\begin{cases} m = \frac{1}{2\sigma_x^2} \\ n = \frac{1}{2\sigma_y^2} \\ p = -\frac{x_c}{\sigma_x^2} \\ q = -\frac{y_c}{\sigma_y^2} \\ k = \frac{x_c^2}{2\sigma_x^2} + \frac{y_c}{2\sigma_y^2} - I_i \ln A \end{cases} \tag{3.21}$$

If Equation 3.21 is substituted into Equation 3.20, then the objective function gets its final form which is the following.

$$H = \arg\min_{m,n,p,q,k} \sum_{i \in U} (I_i x_i^2 m + I_i y_i^2 n + I_i x_i p + I_i y_i q + I_i k + I_i \ln I_i)^2 \tag{3.22}$$

Obviously, Equation 3.22 describes a linear least squares problem, which can be efficiently solved in a closed-form, compared to the initial nonlinear least squares GF problem (Equation 3.13). Overall, it has been proven that, under specific conditions, i.e., using pixels with high enough SNR, the FGF algorithm has the same accuracy with the GF algorithm, and it is significantly faster as it is not an iterative method.

3.2.4 A Linear Least Squares Problem

3.2.4.1 Problem Statement

It is clear that the Gaussian fitting approach, for each cluster aims to find the function f, that is the function's parameters $\mathbf{v}$, that represents the observed data, that is the cluster's pixels. For a single pixel-observation i, this function is expressed as below.

$$y_i = f(\mathbf{x_i}, \boldsymbol{\beta}) \tag{3.23}$$

where scalar y_i is called dependent variable, $\mathbf{x_i} = [x_{i1}, x_{i2}, \dots, x_{ip}]^\top$ is a column vector of p variables, called independent variables or regressors, and $\boldsymbol{\beta}$ are the p unknown function parameters. In the examined application there are N pixels expected, with $N > p$. That makes the equations system overdetermined, which usually has no unique solution and thus an approximation method, such as the Least Squares, is required. Therefore, Equation 3.23 could be written as:

$$y_i = \tilde{y}_i + e_i = f(\mathbf{x_i}, \hat{\boldsymbol{\beta}}) + e_i \tag{3.24}$$

where e_i represents the approximation error, or the difference between the observed value y_i and the estimated value $\tilde{y}_i = f(\mathbf{x_i}, \hat{\boldsymbol{\beta}})$, and is called residual. Note that $\boldsymbol{\beta}$ refers to the real (unknown) parameters that generate the observed data, while $\hat{\boldsymbol{\beta}}$ refers to the approximated parameters that will be estimated.

The difference between FGF and GF is that the proposed method suggests that the relationship between the dependent variable y_i and the regressors $\mathbf{x}_i$ in Equation 3.23 is linear. Note that this doesn't mean that the regressors themselves should be linear, it just means that y_i should be written as a linear combination of parameters $\boldsymbol{\beta}$. In other words, FGF suggests that the following equation is a reasonable approximation of the statistical process that generates the data.

$$y_i = \mathbf{x_i}^\top \hat{\boldsymbol{\beta}} + e_i \tag{3.25}$$

The problem of estimating the unknown parameters $\boldsymbol{\beta}$ in Equation 3.25 is a linear regression problem. Estimating the parameters means finding the expressed by Equation 3.25 hyperplane that has the minimum distance to the given data. The Least Squares method does so by minimizing the sum of squared residuals as shown below.

$$S = \underset{\hat{\boldsymbol{\beta}}}{\arg\min} \sum_{i=0}^{N} e_i^2 = \underset{\hat{\boldsymbol{\beta}}}{\arg\min} \sum_{i=0}^{N} (y_i - \mathbf{x_i}^\top \hat{\boldsymbol{\beta}})^2 = \underset{\hat{\boldsymbol{\beta}}}{\arg\min} \sum_{i=0}^{N} (\mathbf{x_i}^\top \hat{\boldsymbol{\beta}} - y_i)^2 \tag{3.26}$$

The fact that the residuals are linear in the unknown vector $\boldsymbol{\beta}$ means that the problem is a Linear Least Squares (LLS) problem. Moreover as the residuals are unweighted, Equation 3.26 describes the Ordinary Least Squares (OLS) variant. The Gauss-Markov theorem states that under a given set of assumptions the OLS method is the optimal estimator of the unknown parameters. By comparing Equations 3.22, 3.26, it is easily concluded that they have the same form and therefore 3.22 is also a OLS problem.

It is known that the examined minimization problem has a unique closed-form solution, that is a unique parameter vector $\boldsymbol{\beta}$ under the following conditions:

1. The number of observations N equals or exceeds the number of unknown parameters p.

2. The regressors x_{ij} are linearly independent.

It is yet not clear however what is the number of parameters and which are the regressors in Equation 3.22. Let us define the following correspondence between Equations 3.22, 3.26.

$$\begin{aligned}\hat{\boldsymbol{\beta}} &=[m,\, n,\, p\,,q\,,\, k]\\ y_i &= -\, I_i \ln I_i\\ \mathbf{x_i} &=[I_i x_i^2,\, I_i y_i^2,\, I_i x_i,\, I_i y_i,\, I_i]\end{aligned} \tag{3.27}$$

It should be mentioned that the arbitrary choice of the dependent variable y_i is done without loss of generality for operational reasons. Moreover, although some of the regressors x_{ij} are nonlinear, the model remains linear as linearity is meant to be with respect to parameter vector $\boldsymbol{\beta}$. The correspondence described in Equation 3.27 means that the estimated hyperplane that "generates" the observed data i is $p = 5$-dimensional and is described by the following equation.

$$\begin{aligned} y_i &= x_{i1}\hat{\beta}_1 + x_{i2}\hat{\beta}_2 + x_{i3}\hat{\beta}_3 + x_{i4}\hat{\beta}_4 + x_{i5}\hat{\beta}_5 + e_i\\ \implies -I_i \ln I_i &= (I_i x_i^2)m + (I_i y_i^2)n + (I_i x_i)p + (I_i y_i)q + (I_i)k + e_i\end{aligned} \tag{3.28}$$

It is now clear that the number of unknown parameters is $p = 5$ and which are the regressors x_{ij}, and therefore the critical conditions can be evaluated. The satisfaction of the first one depends on the number of pixels in a cluster. It is required that a cluster should contain at least 5 pixels. As it will be discussed in Chapter 4, in this thesis it is assumed that the minimum number of pixels in a cluster is 9. By inspecting vector $\mathbf{x_i}$ in Equation 3.27, it is obvious that the second condition is also satisfied as the regressors are linearly independent.

3.2.4.2 Generating the Normal Equations

Until now it has been proven that FGF can provide a unique optimal solution by solving the optimization problem stated in Equation 3.22. In this section, the initial steps towards this solution are described.

The minimum of the objective function H is found by calculating the partial derivatives with respect to the unknown parameters and setting them to 0. Since there are 5 parameters, 5 equations are derived as shown below.

$$\begin{cases} \dfrac{\partial H}{\partial m} = \sum 2 I_i x_i^2 e_i = 0 & \text{(3.29a)}\\ \dfrac{\partial H}{\partial n} = \sum 2 I_i y_i^2 e_i = 0 & \text{(3.29b)}\\ \dfrac{\partial H}{\partial p} = \sum 2 I_i x_i e_i = 0 & \text{(3.29c)}\\ \dfrac{\partial H}{\partial q} = \sum 2 I_i y_i e_i = 0 & \text{(3.29d)}\\ \dfrac{\partial H}{\partial k} = \sum 2 I_i e_i = 0 & \text{(3.29e)} \end{cases}$$

The sum's limits have been dropped for simplicity reasons. Let us analyze only the first equation below, after dropping the constant 2 from all the equations.

$$
\begin{aligned}
&\sum I_i x_i^2 e_i = 0 \implies \sum I_i x_i^2 (I_i x_i^2 m + I_i y_i^2 n + I_i x_i p + I_i y_i q + I_i k + I_i \ln I_i) = 0 \\
\implies &\sum [I_i x_i^2](I_i x_i^2) m + [I_i x_i^2](I_i y_i^2) n + [I_i x_i^2](I_i x_i) p + [I_i x_i^2](I_i y_i) q \\
&+ [I_i x_i^2](I_i) k + [I_i x_i^2](I_i \ln I_i) = 0 \\
\implies &\sum [I_i x_i^2](I_i x_i^2) m + \sum [I_i x_i^2](I_i y_i^2) n + \sum [I_i x_i^2](I_i x_i) p + \sum [I_i x_i^2](I_i y_i) q \\
&+ \sum [I_i x_i^2](I_i) k + \sum [I_i x_i^2](I_i \ln I_i) = 0
\end{aligned}
\tag{3.30}
$$

By inspecting the above equation, it can be seen that the factor inside the brackets is present in all the partial products. The only difference between the 5 equations in 3.29 is this factor, and for each equation it is equal to the multiplication term of the respective unknown parameter in Equations 3.22, 3.28. These in-bracket terms for each equation are summarized below.

- Equation 3.29a: $[I_i x_i^2]$
- Equation 3.29b: $[I_i y_i^2]$
- Equation 3.29c: $[I_i x_i]$
- Equation 3.29d: $[I_i y_i]$
- Equation 3.29e: $[I_i]$

For the convenience of calculations, these factors will define the following parameters.

$$
\begin{cases}
am_i = I_i x_i^2 \\
an_i = I_i y_i^2 \\
ap_i = I_i x_i \\
aq_i = I_i y_i \\
ak_i = I_i \\
a_i = I_i \ln I_i
\end{cases}
\tag{3.31}
$$

By substituting these parameters into Equations 3.29, which are expanded like Equation 3.30, they are converted to:

$$\left[\sum am_i^2\right] m + \left[\sum am_i an_i\right] n + \left[\sum am_i ap_i\right] p + \left[\sum am_i aq_i\right] q + \left[\sum am_i ak_i\right] k + \sum am_i a_i = 0 \tag{3.32a}$$
$$\left[\sum am_i an_i\right] m + \left[\sum an_i^2\right] n + \left[\sum an_i ap_i\right] p + \left[\sum an_i aq_i\right] q + \left[\sum an_i ak_i\right] k + \sum an_i a_i = 0 \tag{3.32b}$$
$$\left[\sum am_i ap_i\right] m + \left[\sum an_i ap_i\right] n + \left[\sum ap_i^2\right] p + \left[\sum ap_i aq_i\right] q + \left[\sum ap_i ak_i\right] k + \sum ap_i a_i = 0 \tag{3.32c}$$
$$\left[\sum am_i aq_i\right] m + \left[\sum an_i aq_i\right] n + \left[\sum ap_i aq_i\right] p + \left[\sum aq_i^2\right] q + \left[\sum aq_i ak_i\right] k + \sum aq_i a_i = 0 \tag{3.32d}$$
$$\left[\sum am_i ak_i\right] m + \left[\sum an_i ak_i\right] n + \left[\sum ap_i ak_i\right] p + \left[\sum aq_i ak_i\right] q + \left[\sum ak_i^2\right] k + \sum ak_i a_i = 0 \tag{3.32e}$$

This is a linear system of 5 equations in 5 unknown parameters m, n, p, q, k. These equations are called *normal equations* which when solved, the parameter estimates are yielded. If the constant terms are transferred to the right side of the equations we get:

$$\left[\sum am_i^2\right] m + \left[\sum am_i an_i\right] n + \left[\sum am_i ap_i\right] p + \left[\sum am_i aq_i\right] q + \left[\sum am_i ak_i\right] k = -\sum am_i a_i \tag{3.33a}$$

$$\left[\sum am_i an_i\right] m + \left[\sum an_i^2\right] n + \left[\sum an_i ap_i\right] p + \left[\sum an_i aq_i\right] q + \left[\sum an_i ak_i\right] k = -\sum an_i a_i \tag{3.33b}$$

$$\left[\sum am_i ap_i\right] m + \left[\sum an_i ap_i\right] n + \left[\sum ap_i^2\right] p + \left[\sum ap_i aq_i\right] q + \left[\sum ap_i ak_i\right] k = -\sum ap_i a_i \tag{3.33c}$$

$$\left[\sum am_i aq_i\right] m + \left[\sum an_i aq_i\right] n + \left[\sum ap_i aq_i\right] p + \left[\sum aq_i^2\right] q + \left[\sum aq_i ak_i\right] k = -\sum aq_i a_i \tag{3.33d}$$

$$\left[\sum am_i ak_i\right] m + \left[\sum an_i ak_i\right] n + \left[\sum ap_i ak_i\right] p + \left[\sum aq_i ak_i\right] q + \left[\sum ak_i^2\right] k = -\sum ak_i a_i \tag{3.33e}$$

In a matrix notation, the above system is written as:

$$A\hat{\boldsymbol{\beta}} = \mathbf{b} \tag{3.34}$$

where A is the coefficient matrix, $\hat{\boldsymbol{\beta}}$ the unknown column vector and $\mathbf{b}$ the constant column vector.

3.2.4.3 Solving a Linear System

The previous analysis has proved that the solution to the fitting problem is equivalent to the solution of the linear system in Equation 3.34. Therefore, instead of choosing a complex iterative algorithm to yield the Gaussian parameters, the task is to chose a simple method to solve the linear system.

There are a few methods that can be adopted to solve a linear system, which differ in complexity and therefore not all of them can be used in real-time applications. Apart from the analytical methods, numerical algorithms also exist. Typically the latter category is useful when the examined systems are of a high order. However for 5x5 systems, the use of these algorithms is prohibited, as their iterative nature would significantly limit the design's efficiency. For a system of n equations in n unknowns, the most important analytical methods and their arithmetic complexities are summarized in Table 3.1.

Table 3.1: Available Methods for Solving Linear Systems

Name	**Complexity**
Cramer's Rule	$O(n! * n)$
Gauss Elimination	$O(n^3)$
LU Decomposition	$O(n^2)$
Cholesky Decomposition	$O(n^2)$

Apparently the Cramer's Rule is highly inefficient as it requires the calculation of $n+1$ determinants of order n, while Gauss Elimination requires much less time to perform the same calculations. However the algorithm that is practically preferred in computing systems is the LU Decomposition and it's variant, called Cholesky Decomposition.

Ideally, we would like to solve the linear system using the Cholesky Decomposition (also called Cholesky Factorization) method as it requires roughly the half operations compared to the LU. Cholesky method is widely used in Monte Carlo simulations and Kalman Filters, but it sets a constraint regarding coefficient matrix A; In order for this method to be used, A should be positive definite.

There is number of criteria that determine if an matrix is positive definite. However, as it can be concluded by inspecting Equation 3.33, matrix A is not a priori known, it is determined only after a cluster is provided. Thus, no one of the available criteria can be used to verify if the positive definiteness condition is satisfied. For this purpose, we should make reference to the OLS theory.

Based on the p regressors $\boldsymbol{x_i}$, the dependent variable y_i and the p unknown parameters $\hat{\boldsymbol{\beta}}_{\boldsymbol{i}}$, the theoretical form of the p normal equations for n pixels-observations is displayed below.

$$\sum_{i=1}^{n}\sum_{k=1}^{p} x_{ij}x_{ik}\hat{\beta}_k = \sum_{i=1}^{n} x_{ij}y_i,\ j = 1,\ldots,p. \tag{3.35}$$

In matrix notation, the normal are written as:

$$(\mathbf{X}^\top\mathbf{X})\hat{\boldsymbol{\beta}} = \mathbf{X}^\top\mathbf{Y} \tag{3.36}$$

and the optimal parameters are expressed as:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y} \tag{3.37}$$

Evidently, Equations 3.34, 3.36 have the same form based on the following correspondence.

$$\begin{aligned} A &= \mathbf{X}^\top\mathbf{X} \\ \mathbf{b} &= \mathbf{X}^\top\mathbf{Y} \end{aligned} \tag{3.38}$$

The matrix $A = \mathbf{X}^\top\mathbf{X}$ is known as the Gram matrix and is positive definite if and only if the matrix $\mathbf{X}$ is of full rank, that is its columns are linearly independent. Using Equation 3.27, we already proved that the regressors x_{ij} are linearly independent for each observation i. Therefore the columns of matrix $\mathbf{X}$ are linearly independent and the coefficient matrix A is positive definite. Ultimately this means that the most efficient method for solving the linear system in Equation 3.34, the Cholesky Decomposition, can be used.

As its name suggests, the Cholesky algorithm, as well as the LU, is based on decomposing the linear system into two simpler systems, that can be solved very easily, as shown below.

$$A = LL^\top \tag{3.39}$$

where L is a lower triangular matrix and $L^\top$ its transpose, an upper triangular matrix. How the matrix L is calculated, will be discussed in Chapter 4, where the implementation of the algorithm is described.

In order to conform with the standard notation used in linear systems for convenience, let us now change the notation in Equation 3.34 by substituting the unknown parameter vector $\hat{\boldsymbol{\beta}}$ for the unknown variable vector $\mathbf{x}$. Equation 3.34 is converted to:

$$A\mathbf{x} = \mathbf{b} \tag{3.40}$$

From now on there will be no other reference to the regressors, so the unknown vector $\mathbf{x}$ should not be confused with the prior notation. By substituting Equation 3.39 into 3.40, the problem is transformed to the following:

$$LL^\top\mathbf{x} = \mathbf{b} \tag{3.41}$$

By setting $L^\top\mathbf{x} = \mathbf{y}$, the system is decomposed into the following two simpler problems.

$$L^\top \mathbf{x} = \mathbf{y} \tag{3.42}$$

$$L\mathbf{y} = \mathbf{b} \tag{3.43}$$

As both L and $\mathbf{b}$ are known, the second system (3.43) is firstly solved with respect to the unknown variable vector $\mathbf{y}$ with forward substitution. Afterwards, the first system (3.42) is solved with respect to the unknown parameter vector $\mathbf{x}$ with reverse substitution.

This process yields the OLS estimate for the parameter vector $\mathbf{x} = \hat{\boldsymbol{\beta}} = [m, n, p, q, k]^\top$. The final step is to calculate the Gaussian parameter vector $\mathbf{v}$, given in Equation 3.11 and more specifically the cluster centroids x_c, y_c. These parameters are found using the definitions from Equation 3.21 as follows.

$$\begin{cases} x_c = -\frac{p}{2m} \\ y_c = -\frac{q}{2n} \\ \sigma_x = \frac{1}{\sqrt{2m}} \\ \sigma_y = \frac{1}{\sqrt{2n}} \\ A = \exp\left(\frac{p^2}{4m} + \frac{q^2}{4n} - k\right) \end{cases} \tag{3.44}$$

3.2.5 Centroiding Process

The main concern so far was the characteristics and functionality the FGF algorithm. Based on Equations 3.17, 3.18 it is considered that the FGF demonstrates approximately the same accuracy as the GF, if and only if the pixels i involved in the calculation, are characterized by $\phi(r_i)$ close enough to 1, that is they have SNR greater than a certain value.

In star images however, due to the uncertainty of the noise and star intensity, it is expected that not all pixels in an extracted cluster will satisfy this precondition, and it is also not known which of them do. For this reason Wan et al. [7], did not use the FGF as a complete centroiding algorithm, but they proposed a centroiding system based on the Fast Gaussian Fitting algorithm. The flow diagram of this system is illustrated below.

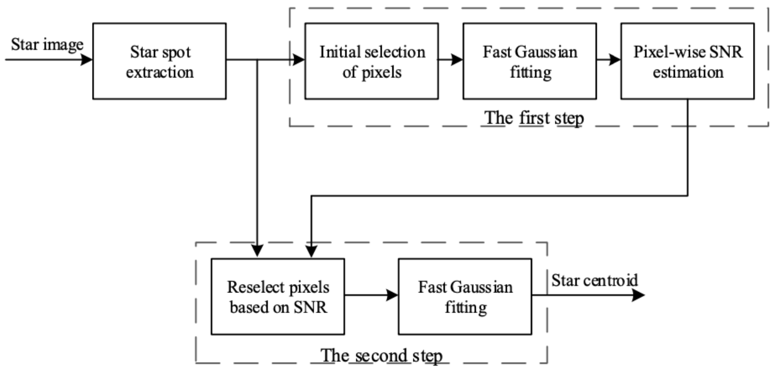


Figure 3.2: Flow diagram of the proposed centroiding process based on the FGF. Image taken from [7].

The first step of this method intends to roughly estimate the SNR of each pixel in the cluster by getting a solution of the underlying Gaussian parameters using the FGF. Based on this estimation, the pixels that satisfy the precondition of high enough SNR are given as input to the FGF in order to calculate the final centroids. This operation for a given cluster of $N > 5$ pixels is summarized below.

1. In the first step the SNR of each pixel is estimated.
   (a) Initially a limited number of pixels are selected, based on the fact that brighter pixels have higher SNR, in order for the FGF to be able to provide accurate results. The five brightest of the cluster are selected to form the set U'.
   (b) The FGF is performed over the initial set U' in order to roughly estimate the Gaussian parameters $(A', x'_c, y'_c, \sigma'_x, \sigma'_y)$. These parameters are used to establish the PSF using Equation 3.11 as following.

$$\hat{S}_i = A' \exp\left(-\frac{(x_i - x'_c)^2}{2\sigma'^2_x} - \frac{(y_i - y'_c)^2}{\sigma'^2_y}\right) \tag{3.45}$$

   (c) The established PSF is used in order to estimate the source intensity S_i for all pixels $i = 1 \ldots N$. Then the noise intensity and consequently the SNR of each pixel i can be estimated as shown below.

$$\hat{N}_i = I_i - \hat{S}_i \tag{3.46}$$

$$\hat{r}_i = S\hat{N}R_i = |\frac{\hat{S}_i}{\hat{N}_i}| \tag{3.47}$$

2. Using the SNR estimation for each pixel of the cluster, the second step calculates the final centroids.
   (a) For a given SNR threshold T, that defines a certain proximity of $\phi(r_i)$ to 1, the pixels of the cluster with estimated SNR greater than this threshold, are selected to form the final set U. This re-selection is expressed below.

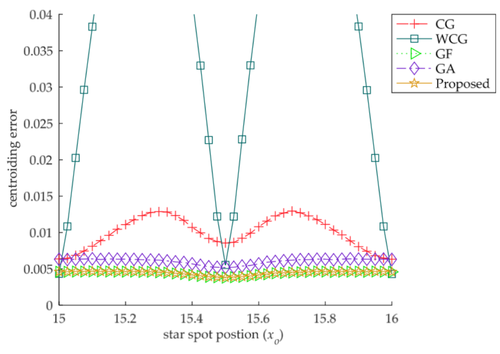
$$U = \{i,\, S\hat{N}R_i > T\} \tag{3.48}$$

   (b) The FGF is performed over the final set U to estimate the Gaussian parameters again. Comparing with the first execution of the FGF, more pixels are involved this time and the results will be more accurate.

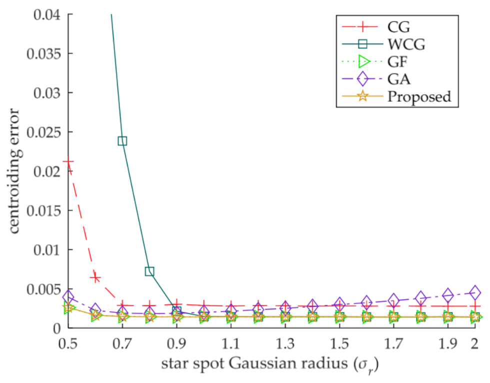
In [7] the proposed centroiding process has been extensively evaluated against the most important centroiding algorithms that have been mentioned in Section 2.3.1.2. The most interesting experiments are those in which, the accuracy is measured while changing the standard deviation (σ_x, σ_y), the noise level or the brightness level A of the Gaussian PSF. Details about the image generation and parameter selection can be found in the paper and are purposely omitted here. The generated error curves are summarized in Figure 3.3.

These experiments prove that the centroiding error of the proposed algorithm is almost the same as that of the GF, less than of the other algorithms, under all circumstances.

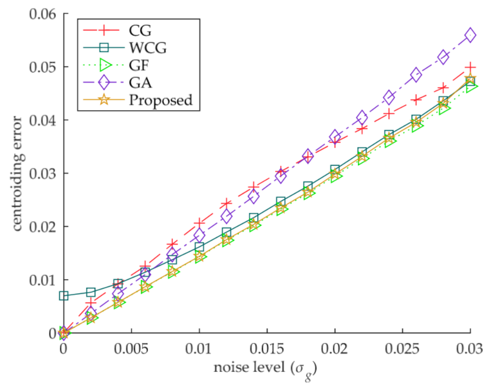
It is evident, that the FGF-based method is more robust to noise than the CG. When the centroid moves along the x-axis in the subpixel level, the centroiding error curve of the latter also suffers from the S-curve error as expected. Is is also apparent that the accuracy of the GF and FGF algorithms is kept unaffected when saturated pixels are present (Figure 3.3d). This is due to the fact, that as stated in [7], saturated pixels are excluded during pixel selection by definition. This kind of pixel filtering is essential for the fitting algorithms, because they are expected to be



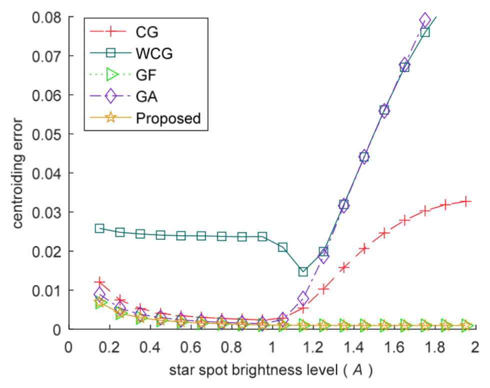
(a) Changing Centroid Position



(b) Changing Gaussian Radius



(c) Changing Noise Level



(d) Changing Brightness Level

Figure 3.3: Evaluation experiments of the proposed method's accuracy against popular centroiding algorithms. CG: Center of Gravity, WCG: Weighted (Squared) Center of Gravity, GF: Proposed Algorithm, GA: Gaussian Analytic, GF: Gaussian Fitting. Images taken from [7].

very sensitive to flat saturated sub-regions within a cluster, as such regions distort the cluster's Gaussian-like shape. This phenomenon has been also detected in [61].

Another experiment has been conducted, which allows a closer comparison of the considered in this thesis algorithms, CG and FGF, with respect to the position of the detected cluster on the image plane. The results of this experiment are displayed in Figure 3.4.

Apparently the FGF demonstrates an overall dominance over the CG algorithm. The positions that the CG performs better locate mainly near the edge of the FOV, due to the distortion of the optical system which is greater there, as already explained. This effect leads to clusters, whose intensity shape does not strictly conform to a Gaussian function, which consequently reduces the accuracy of the FGF [59].

Lastly the results of the efficiency experiment, which measured the total time consumption of each algorithm on 10,000 star images are given in Table 3.2.

Table 3.2: The total time consumption of each algorithm on 10,000 star images. Data taken from [7].

Method	**CG**	**WCG**	**GF**	**GA**	**FGF**
Time (s)	1.3504	1.3878	58.9863	1.4915	3.9675

It is confirmed that CG, due to its low complexity, is the faster algorithm available. The

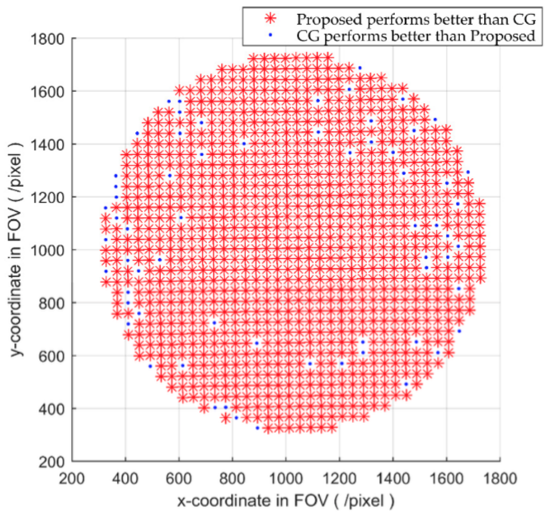


Figure 3.4: The colored part of the plane corresponds to the positions of all centroids on the image plane. Red asterisks represent the positions where FGF performs better than CG and blue dots the opposite. Image taken from [7].

most important finding is that FGF indeed improves the efficiency of the Gaussian Fitting algorithm without reducing the accuracy. This feature makes it suitable for the real-time application examined in this research.

3.2.6 Proposed Modifications for Implementation on FPGA

As already explained in the CG case, when designing a system on the FPGA, some modifications on the respective software algorithm are usually required in order to exploit better the benefits provided by the hardware platform.

3.2.6.1 Changing the Centroiding Process

In Section 3.2.5 the proposed in [7] centroiding process was analyzed. In order to achieve high accuracy under the uncertain and noisy space environment, the FGF algorithm is executed twice.

If the centroiding system illustrated in Figure 3.2 is to be implemented on hardware, it is expected that the FGF algorithm will be the design's critical component, with respect to both resource utilization and performance. There are two possible solutions in order to manage this component.

The first and most straightforward option, would be to implement the proposed system using two separate FGF blocks. It should be expected that in this case the resource utilization would be significant, possibly leading to timing problems due to area congestion and consequently to a drop in performance. One should remember, that apart from the centroiding algorithm, the clustering algorithm should also be implemented on the FPGA. Thus extra care should be taken when evaluating the trade-off between resources and performance.

The second option is to implement a single FGF block that will be used by both processing steps. However, creating feedback loops is generally not a very hardware-friendly design method. It can be also expected that the establishment of a feedback loop, would decrease the system's throughput, compared to the first design option, as the next cluster will need to wait longer before it can be driven to the FGF block. This extra time is determined by the time required, in order for the previous cluster to be processed by the Pixel-wise SNR Estimation, Pixel Reselection and FGF (for the second pass) blocks. Approximately, the use of a single FGF Block and creating a feedback loop would reduce the throughput by 50%.

Fortunately, already before this problem was taken into consideration, Infinite Orbits had determined that the accuracy provided by the FGF algorithm, if the whole extracted cluster is used in the first place, is sufficient regarding the mission's requirements. In other words they have determined that if the FGF algorithm is used as a complete centroiding method, instead of the proposed more complex process, the requirement of cross-boresight error is satisfied.

Therefore for the rest of this thesis, it is assumed that the centroids are extracted by a single execution of the FGF algorithm. It is important to point out that the drop in accuracy due to the modification of the proposed system can be even lower, if appropriate care is taken during clustering. More specifically, if the threshold value used in clustering is slightly increased, which means that the extraction criterion is stricter, then the provided clusters will be brighter and the respective pixels will have a higher SNR. This threshold value can be either predefined or dynamically adjusted. In the first case the increased threshold means that, if for a given set of data (pixel intensities), a 5x5 cluster was previously extracted, for the same data a 4x4 or 3x3 cluster will eventually be extracted. Otherwise, in order for the clustering threshold to be dynamically adjusted, a feedback channel from the Matching process to the Clustering process should be established. This channel can provide information to the clustering algorithm regarding the overall accuracy of the system. Therefore, if lower accuracy than expected is observed, the clustering can be instructed to increase the threshold value in order to provide "better" data to the centroiding algorithm, which is ultimately the critical factor as far as accuracy is concerned.

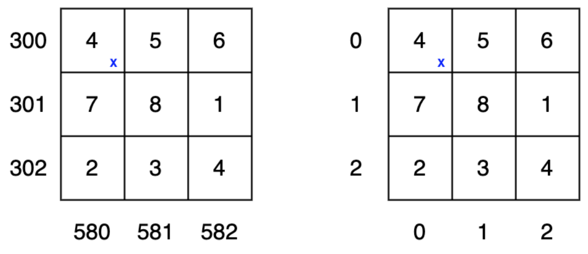
It should be also noted that in this work, it is assumed that the exclusion of the saturated pixels is accomplished during clustering and that clusters with no saturated pixels are expected.

Lastly we highlight, that the described design call removes completely the need to implement the Pixel-wise SNR Estimation component, which includes the operations stated in Equations 3.45, 3.46, 3.47. The fact that these operations would be performed for each pixel means that this block would add a significant number of cycles to the system's total latency, that are eventually avoided.

3.2.6.2 Changing the Reference System

Similarly to the CG algorithm modification, the second change of the FGF algorithm aims to minimize the bit widths within the design. However, unlike the CG case, there is not an analytical proof that can verify the validity of the intended modification, due to the large number of complex operations (e.g. exponential, powers etc.). The correctness of this change will be verified through an example.

Consider the following two clusters, which have been detected within a frame. Remember that the indexing in the adopted raster scan starts from 0.



Each cluster will be processed in the same way. Initially the intensity values and the respective coordinates will be used in order to calculate the coefficient matrix A and the constant vector $\mathbf{b}$ (Equation 3.33) and then the Cholesky Decomposition will be performed in order to calculate the centroids. Apparently, the matrices of each cluster will differ.

It is true that the estimated centroid subpixel position is correlated with the intensity values, as these values are generated by the underlying Gaussian function, whose one fundamental parameter is the real centroid itself. It's also true that the Gaussian distribution that generates the data and the respective centroids, is independent from the pixel coordinates. Considering also the uniqueness of the OLS solution which has been proved in Section 3.2.4, a given set of data (i.e., pixels and centroid) is always generated by the same Gaussian function and is independent from the position of the cluster within the image plane.

With regard to the examined example, this means that the centroid's position for both clusters will be the same. This position is marked as a blue x. Note that the second given cluster can be considered either as a cluster which starts at first row and column or the same cluster as the first but with coordinates that refer to a pixel's distance to the cluster's first row and column (relative coordinates). In other words it has been proven that the centroid estimation is independent from the reference system that is used. The estimated position is the same if either the absolute (within the image plane) or the relative (within the cluster) coordinate system is used.

That allows us to always use the relative coordinate system in order to minimize the bit widths of all the partial results within the design, similarly to the CG case. Finally, when the relative centroid is estimated, the absolute coordinates, that is the first row and column, of the cluster will be added in order to yield the real location of the star centroid in the image plane.

Chapter 4

Proposed Hardware Designs

The main objective of this thesis is the acceleration of the considered centroiding algorithms on an FPGA platform. In this chapter the implementation of each proposed hardware architecture is described. The encountered designing trade-offs and the differences in development process between the FPGA programming models are thoroughly explained. Technical details and results of the simulations, performed in order to test and verify the implemented systems, are also provided. The accuracy and performance evaluation of the proposed designs is covered in Chapter 5.

4.1 System Specifications

4.1.1 Configuration and Process Integration

In order to better understand the implementation details of the centroiding process, it is necessary to firstly establish the background regarding the system's configuration and the framework within which the proposed designs operate.

Considering the configuration created by Infinite Orbits, the algorithms have been implemented on the ZedBoard development board, which hosts the Xilinx ZYNQ-7020 SoC. The PL features of this platform will be discussed in Chapter 5. The provided system includes also a custom high speed sensor-lens system, with the following specifications.

- Field of View: 15.5° x 15.5°
- Image Resolution: 2048 x 2048 pixels
- Pixel depth: 12 bits

Therefore the gray-scale intensity of each pixel is represented with 12 bits and each image consists of 2048 rows and 2048 columns. Using a high speed streaming protocol the image data are read-out by the clustering process, which is also expected to be implemented on the FPGA side of the system. During clustering, a pre-processing technique called data binning takes place. This technique is typically used in order to reduce the effects of errors occurred during quantization. In our case, binning is performed by averaging the values of 2x2 regions of pixels and then rounding the calculated value towards the nearest equal or smaller integer. This method creates an image with half the initial resolution, that is 1024 x 1024 pixels. After binning, clusters are extracted from the frame and are supplied to the centroiding block.

As already stated in Chapter 3, it is assumed that each cluster is of square shape. More specifically, Infinite Orbits have recommended that each star in the captured image is spread over regions of 3x3 to 5x5 pixels. Consequently, the implemented designs are expecting as input clusters of these dimensions.

A basic prerequisite when designing a hardware system, is to know the format of the input data, which depends on the previous block in the pipeline. For this reason, it is important to establish the interface between the clustering (producer) and the proposed (consumer) blocks. The term "interface" refers to both the physical communication channel (e.g., possible intermediate components, number and width of IO ports) and the format of the transferred data. Since both systems are implemented on the FPGA, it is assumed that the interface consists of a single buffer, which holds the produced data until the consumer is able to receive and process them. Due to the nature of the examined application, it has been determined that the most efficient way to transfer data is by streaming pixels one-by-one. An abstract representation of the established streaming channel is shown in Figure 4.1.

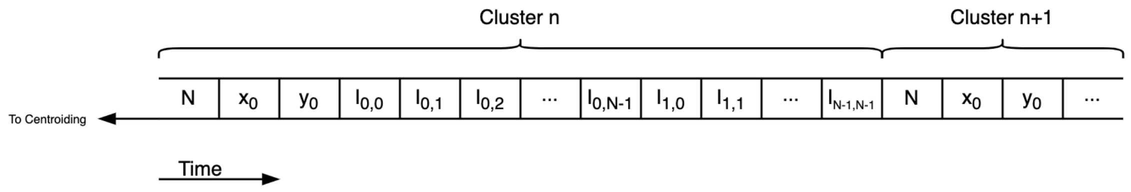


Figure 4.1: Illustration of the data transmission between clustering and centroiding blocks.

Before the reception of the pixel intensities, which are the genuine information for centroid detection, a set of auxiliary data in a package-like format is transferred for each cluster. The first package is a number that stands for the dimension of the following cluster. If a 5x5 cluster has been extracted and is to be received, then $N = 5$. Afterwards x_0 and y_0 are provided, which express cluster's first column and row in the image plane respectively. Finally the intensities of the cluster's pixels, following the raster scan pattern are streamed. Immediately after the reception of the last pixel's intensity, the streaming of the next cluster can start.

Since the expected clusters have a maximum size of 5x5 pixels, 3 bits are required to represent the N data. As stated, binning results in an image of 1024x1024 pixels, from which the clusters are extracted. Consequently, the number of bits that are required to represent x_0 and y_0 is 10. However, the image sensor generates high depth pixels whose intensities are expressed using 12 bits. Since all of these data are transferred through a single channel, the respective port of the implemented blocks has a width of 12 bits. This will apply in all of the proposed architectures.

As already mentioned, the matching process is expected to be developed on the ARM processor by Infinite Orbits. This process will be receiving the calculated centroids through the high performance AXI interface. The establishment of the described interfaces allows the interleaved integration of the processing blocks. Thus, while our proposed design calculates the centroid of a cluster, matching will be calculating the angular distances between stars of the previous frame, clustering will be extracting the following clusters and the image sensor will be integrating the next frames.

4.1.2 Performance Requirements

Before analyzing the proposed designs, it is essential to explain how the overall system's requirements introduced in Section 1.3 are related to the objectives that where placed during implementation. For convenience the basic requirements that the total system needs to meet are restated below.

- Mean accuracy error of less than 3 arcseconds across boresight
- Real time performance of 1-2 frames per second

Initially we should estimate the system's required accuracy in pixels. Considering that an arcsecond is $\frac{1}{3600}$ of a degree and by applying the sensor's specifications in Equation 2.13 we can calculate the correspondence between arcseconds and image pixels as follows.

$$\frac{\text{arcseconds}}{\text{pixel}} = \frac{\text{degrees}}{\text{pixel}} \cdot 3600 = \frac{\text{FOV}}{\text{Resolution}} \cdot 3600 = \frac{15.5^{\circ}}{2048} \cdot 3600 = 27.246 \tag{4.1}$$

Mathematically the required cross-boresight accuracy translates to a typical subpixel accuracy of roughly 0.1 pixels. However it should be mentioned that the specifications provided by Infinite Orbits allow small deviations, as the maximum accepted error is assumed to be 0.25 pixels.

On this point, it should be also pointed out that this thesis puts focus only on the hardware implementation of the centroiding process and doesn't develop an end-to-end star tracker. Consequently it is not possible to evaluate the proposed designs over the required overall update rate of 1-2 Hz. Instead during development, a more flexible objective of achieving the highest possible throughput was placed. For the same reason, the AXI interface between the PL and the PS hasn't been developed. Instead, in this thesis both FPGA programming models have been extensively studied.

4.2 Center of Gravity Algorithm

As already explained, the CG algorithm is the most traditional and simple approach to the centroiding problem. In this thesis it has been adopted as a baseline design. Both VHDL and C++ have been used in order to implement a CG-based model, which allowed a detailed comparison between the two programming models. The implemented designs are described next.

4.2.1 VHDL Model

4.2.1.1 Overview

At first VHDL was used in order to provide an RTL description of the centroiding block. A simplified block diagram of the proposed hardware architecture is displayed in Figure 4.2.

The system is divided in 3 sub-blocks, which are displayed with different colors and are synchronized under one global clock (*clk*). Each of these blocks contains also a number sub-modules, displayed with blue color, that perform the fundamental operations in the lowest level. During development with VHDL, the structural design style was adopted in order to easily describe the system's hierarchy. In order to model the sequential logic inside the lowest level's blocks, the behavioral style was used.

The operation of each displayed element will be explained in the following sections. For each developed block, a number of different architectures were tested. The criterion based on which

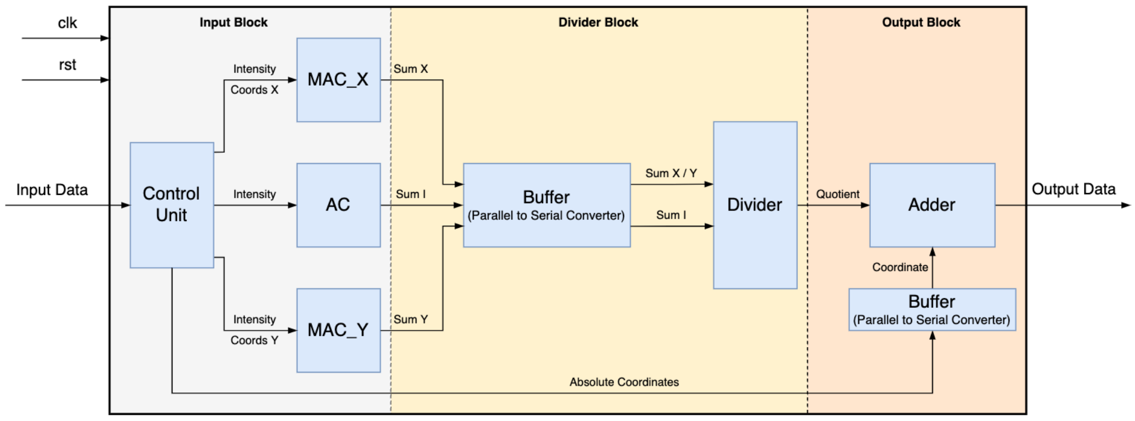


Figure 4.2: Simplified block diagram of the proposed hardware architecture.

a specific format was preferred over the others was the Logic Synthesis results. This is a typical method when designing on hardware, as Logic Synthesis creates the respective RTL model and estimates the final circuit with sufficient accuracy. It is important to mention, that due to the low complexity of the CG algorithm, in most cases the differences regarding the estimated resources were very small, e.g., 1-2 FPGA primitives (LUT or Registers). Although in some cases these differences could be eradicated during implementation, as a number of optimizations are performed, it is critical to adopt a systematic method of comparing different descriptions. Therefore even such small differences were taken into account and the optimal model was chosen based on Logic Synthesis minimum resources and shortest critical path.

As in every sequential digital circuit, apart from the clock, a reset signal is necessary in order to initialize and reset the system. In this design, this signal is denoted as *rst*, is connected to each building block and is active high, unless it's stated otherwise.

4.2.1.2 Input Block

The "front-end" of the CG circuit is the Input Block and is responsible for calculating the numerators and denominator of the CG formula, given in Equations 3.9, 3.10 and repeated below.

$$x_c = X + \frac{\sum I_i x_i}{\sum I_i}$$
$$y_c = Y + \frac{\sum I_i y_i}{\sum I_i}$$

Multiplier Accumulator

One of its basic elements is the MAC (Multiplier Accumulator) block, which calculates the sum in the numerator of the fraction and its RTL model is given in Figure 4.3.

This block is controlled by the Control Unit (CU) through the enable *en* and initialization *mac_init* signals. Through the *bright* port a pixel's 12-bit intensity value I_i is provided, while at the same time in the 3-bit *coord* port the relative coordinate x_i is available. The maximum number of pixels per cluster is $5 \times 5 = 25$ which means that the maximum number of products to be summed is 25. It is known from basic binary arithmetic theory that when M numbers of N bits are added, the result is represented with $N + \log_2 M$ bits. Since $32 > 25$, where 32 is the smaller power of 2 larger than the number of products, and $N = 12 + 3 = 15$ due to multiplication, the output port should be $15 + \log_2 32 = 20$ bits wide.

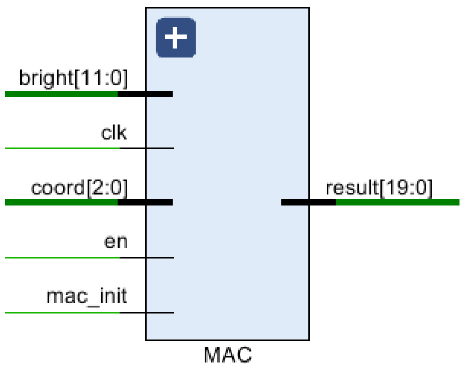


Figure 4.3: RTL model of the MAC block.

The MAC block can be implemented in several ways on hardware regarding operation and resources. Various behavioral descriptions were tried and it was determined that the lowest critical path was achieved when the block was mapped to a DSP core. This result was anticipated as this core can perform the MAC operation in the most efficient way. Because the synthesizer didn't generate this core automatically, the *use_dsp48* attribute was used instead. The exact operation of the MAC block, verified in behavioral simulation, is shown in Figure 4.4.

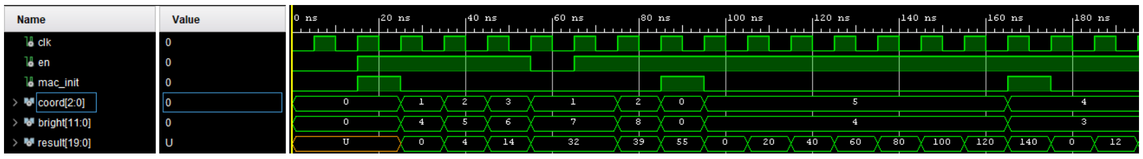


Figure 4.4: Simulation of the MAC block.

As it can be seen, the first pixel can be provided one cycle after the initialization signal. When simulations are visually observed, one must remember that if a signal is asserted at the exact moment when the clock is asserted, it cannot be read in this cycle. This is due to the setup time, which demands an input signal to be stable for a short period of time prior to the clock's positive edge. Therefore, the signal is actually read at the next positive edge. The MAC block generates a result with latency $= 0$, which means instantly after it reads the respective input.

Several other models were also tried. If the input data were provided in parallel with the initialization signal and it was required to instantly calculate the result, the respective critical path was significantly larger, as expected. Thus the optimal model is the one that was presented. Since CG requires the calculation of two numerators, one for each dimension, two MAC blocks were instantiated.

Accumulator

The denominator which is common for both dimensions and represents the sum of intensities is implemented with a similar block, called AC (Accumulator). This block is given in Figure 4.5.

The width of the output port is calculated as $12 + \log_2 32 = 17$ bits. The operation of this block is the same as for MAC and it is shown in Figure 4.6. The accumulation operation was also mapped to a DSP core. It was found that otherwise the synthesizer estimated a slightly larger critical path.

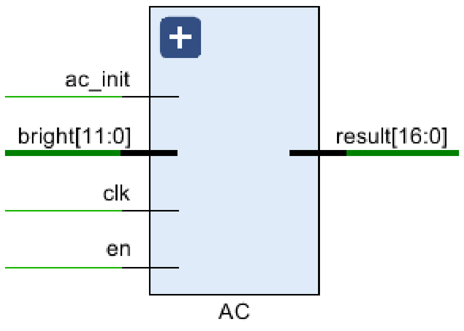
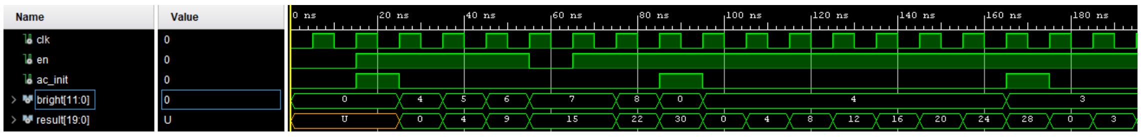


Figure 4.5: RTL model of the AC block.



```
Name            Value
clk             0
en              0
ac_init         0
bright[11:0]    0      0  4  5  6  7  8  0  4  3
result[19:0]    U      U  0  4  9  15  22  30  0  4  8  12  16  20  24  28  0  3
```

Figure 4.6: Simulation of the AC block.

Control Unit

The most complex module in the Input Block is the Control Unit (CU) which is responsible for controlling the other blocks and the data flow between them. The RTL model of this block is given in Figure 4.7.

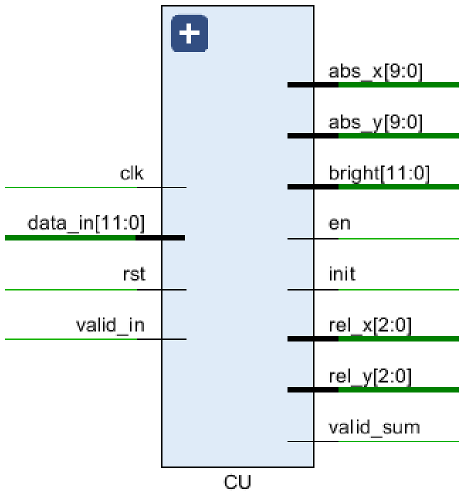


Figure 4.7: RTL model of the CU block.

Initially it checks if the input data should be received and processed, based on the *valid_in* signal which indicates their validity. This kind of signal is typical in digital systems and is generated by the clustering system. When valid data are available, the CU initializes the MACs and AC and supplies the input intensities through the *bright* port. At the same time, based on the N package which is always received at the start of a transmitted cluster, it generates internally the relative coordinates of each pixel and passes them through the *rel_x* and *rel_y* ports. This block is implemented as a Finite State Machine. Its structure is introduced in Figure 4.8 in order for the overall function to be better explained.

This FSM represents the process followed for each cluster. Each edge corresponds to the value

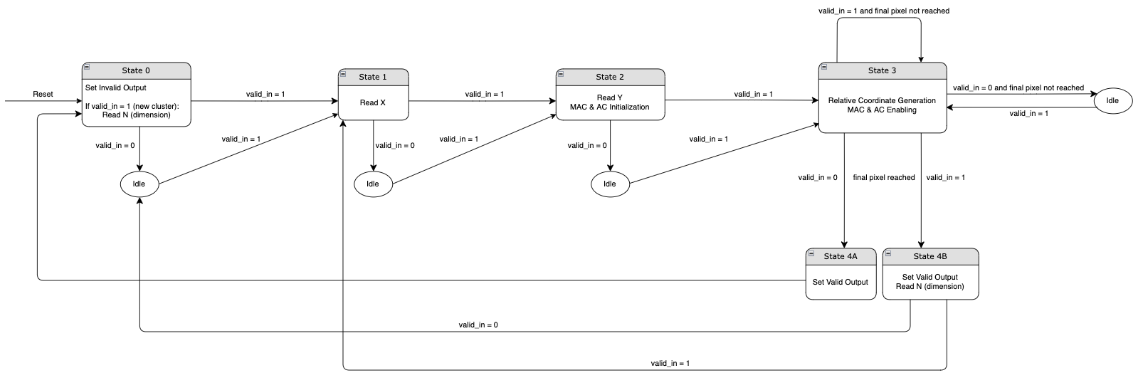


Figure 4.8: FSM that describes the CU function.

of the *valid_in* signal one cycle after the operation inside the source box. Suppose we start at State 0, after either the power up or the processing of a previous cluster. As long as the input is invalid, nothing happens. The first valid input corresponds to the N package which informs the CU regarding the cluster size. If the next data is invalid, the system goes to an idle state until the next valid input is received. Apparently the same procedure is followed regardless of the current state. Otherwise, the system moves to State 1, where the next package, which is the starting column X, is read. Similarly, at the next active cycle the starting row Y is read. After that, the intensities are streamed through the input port. For each intensity two respective coordinates are generated (for both dimensions) and these three data are passed to the computing blocks.

The most important part of the FSM is the transition from State 3 to State 4. Based on the N package, the CU keeps track of the received pixels and is able to recognize when the last pixel has been received. When this happens, there are two possible transitions at the next cycle. In each case the *valid_sum* signal is asserted in order to inform the next blocks that there are valid sums available, calculated by the MACs and AC. If the next input data is invalid, then the FSM returns to State 0 and waits for the next cluster to start. However it is possible, that data from the next cluster are transferred immediately after the one which was just processed. Thus, if the next data is valid, it means that the N package from the next cluster has been received. Then the FSM skips State 0, moves to State 1 and waits for the X package. This architecture enables the system to operate with the highest possible throughput, as there is no ”adjustment” period between consecutive clusters required.

Apart from the Divider Block, that receives the *valid_sum* signal in order to accept the calculated sums, this signal is also driven to the Output Block, which at the same time reads the cluster’s absolute coordinates X, Y through ports *abs_x* and *abs_y* respectively. More on this will be discussed later on.

Due to the nature of the CU module, it has been implemented only with logic cells, that is LUTs and registers. The simulated CU is displayed in Figure 4.9.

Notice how the output port *bright* follows the input port *data_in* with a delay of 1 cycle, due to the required setup time as explained. Similarly the MACs and AC read the intensity values one cycle after the *bright* signal is updated. This is why the *valid_sum* signal is asserted one cycle after the last intensity is passed through the bright port.

Overall Input Block

When the described modules interconnect, the Input Block is formed. Both the Input Block’s RTL model and the internal interconnections as generated in Vivado are shown in Figure 4.10.

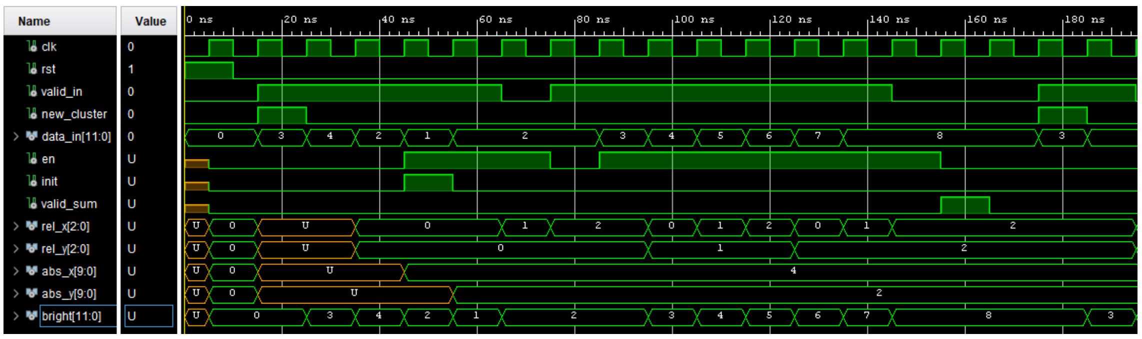
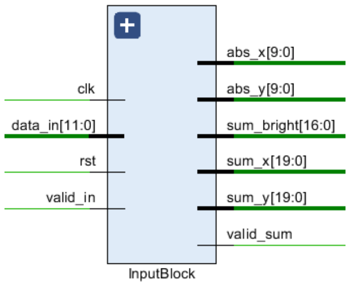
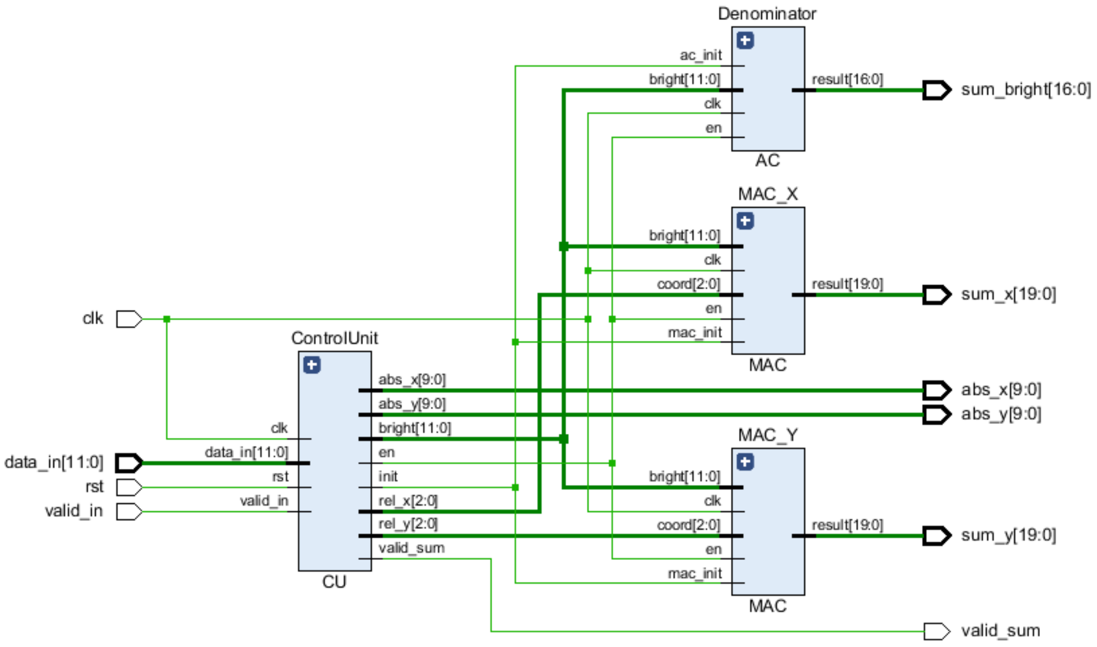


Figure 4.9: Simulation of the CU block.



(a) RTL model.



(b) RTL schematic.

Figure 4.10: Complete RTL representation of the Input Block.

Lastly, it is important to evaluate the examined block regarding the hardware metrics of latency and throughput. This can be easily done through simulation, which is displayed in Figure 4.11. Note that the signal *new_cluster* hasn't been implemented and is used in simulation for illustration purposes only.

A cluster of N pixels corresponds to $N + 3$ cycles of valid data, that is N cycles for the

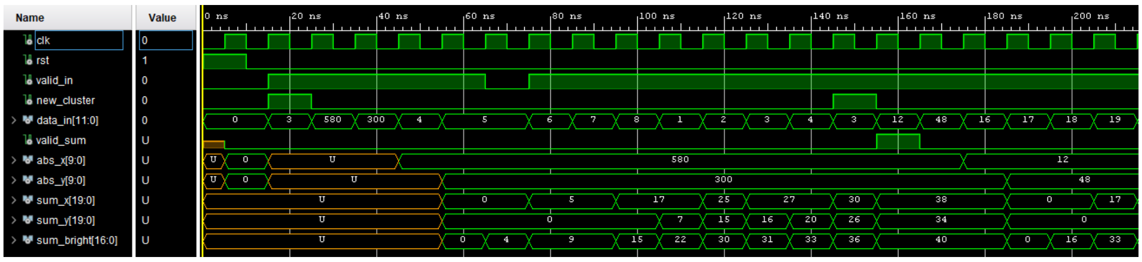


Figure 4.11: Simulation of the Input Block.

pixels themselves and 3 cycles for the initial auxiliary data. Since the CU is able to process the next cluster immediately after the previous cluster's last pixel the Input Block demonstrates Throughput $= N^2 + 3$ cycles. The latency analysis is a little more complex. As explained, the CU drives the input data to its output port one cycle after the *valid_in* is asserted, due to the setup time. Correspondingly the computing blocks read their input, and provide output, one cycle after they are enabled. Since there are N intensities, 3 initial packages and 1 cycle due to this delay, $Latency = N^2 + 4$ cycles.

Even though intuitively the latency should be measured with respect to the read cycle, usually it is measured from the moment in which the input is asserted as valid, which is one cycle earlier. For the rest of this thesis, this measurement method will be adopted. Lastly, it should be noted that for convenience, in simulations small numbers are used throughout this work. In the real application significant larger values are expected.

4.2.1.3 Fixed Point Arithmetic

At this point it is essential to shortly explain a concept that was adopted during system development. Fixed Point Binary Arithmetic is widely used in Digital Signal Processing applications to represent fractional numbers as a replacement to the significantly more complex Floating Point Arithmetic. In this representation a binary number consists of an integer and a fractional part, which are separated by the binary point.

Consider a $I + F = N$-bit number x, where I is the amount of integer bits and F the amount of fractional bits. This number is expressed as:

$$x = \sum_{n=0}^{N-1} 2^{n-F} x_n \tag{4.2}$$

where n is the bit position starting from the Least Significant Bit (LSB). Apparently, the bits on the right of the binary point carry negative weights, which means that they represent numbers smaller than 1.

Due to its close relationship with the integer representation, Fixed Point Arithmetic is highly efficient and allows high performance fractional number operations. In hardware designs where the performance is usually the most important objective, the use of Floating Point representation is unaffordable. It is computationally very demanding and therefore costly in terms of area and clock cycles. On the contrary, Fixed Point is by orders of magnitude faster but leads to loss of precision and should be used with caution.

Since there is a minimum required precision specification in our application (Section 4.1.2), the use of Fixed Point numbers to represent the subpixel fractional centroid is vital. The operation which involves fractional numbers in the CG algorithm is the division. If the quotient is implemented as a fixed point fractional number, the resources and logic levels required for the division are heavily reduced resulting in lower latency and shorter critical path.

A fractional binary number, defined as in Equation 4.2, is written as below.

$$x = b_{I-F-1}b_{I-F-2}\dots b_1 b_0 . b_{-1} b_{-2} \dots b_{-F-1} b_{-F}$$

where the subscript value is the weight of each bit. Apparently the "step size" which refers to the resolution of a given fixed point representation, that is the accuracy level, is determined by the LSB b_{-F} and therefore the number of fractional bits F.

The minimum number of fractional bits which satisfies the imposed requirement is $F = 4$ as:

$$2^{-F} = 2^{-4} = \frac{1}{16} = 0.0625 < 0.1$$

4.2.1.4 Divider Block

The second block in the proposed pipeline is the Divider Block, which is responsible for calculating the subpixel centroids within the cluster, that is the relative centroids.

Divider

Of course the fundamental module of this block is the Divider. It is known that the division is the most complex of the four basic arithmetic operations. In order to design high performance hardware, the Xilinx LogiCORE IP Divider Generator core is provided which offers three different division algorithms, allowing choice of resource, throughput and latency trade-offs. Due to the nature of our algorithm, a basic demand regarding the division operation is to produce a result with fractional remainder. The LUTMult solution doesn't offer this output type and was therefore rejected.

The most powerful algorithms are the Radix-2 non-restoring algorithm and the High Radix algorithm. The former uses only FPGA primitives (LUTs and FFs) and solves one bit of the quotient per cycle. The latter exploits DSP Slices and Block RAMs and is implemented as an iterative engine, solving multiple bits of the quotient per cycle. However, due to the use of prescaling prior to the iterative division, it causes an overhead of resource which makes this solution recommended for larger operands. By executing Logic Synthesis, it was indeed found that this algorithm uses more than twice the amount of resources that the Radix-2 requires.

The block diagram of the produced IP Divider core, based on the Radix-2 algorithm is given in Figure 4.12.

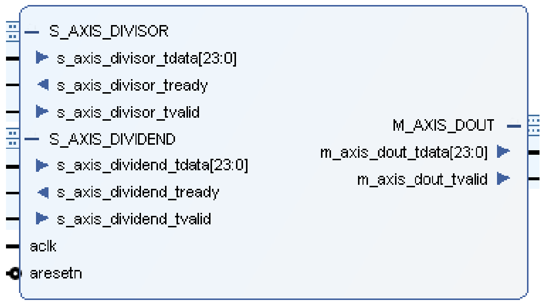


Figure 4.12: RTL block model of the Divider IP Core.

This cores adheres to the AXI4-Stream protocol which means that all the inputs and outputs are conveyed on AXI4-Stream channels. The input channels consist of the *tdata*, *tvalid* input signals and the *tready* output signal. The output channel doesn't contain a *tready* signal in this implementation. The validity of an input signal depends on the value of the *tvalid*, which together with the *tready* perform a handshake to transfer a message, through the *tdata*, which is the actual payload. The output *tready* indicates if the divider is ready to accept new data. Another feature of the AXI4-Stream specification is the implementation of an active low reset signal, which means that the input active high signal must be inversed. Moreover, both *tdata* ports are extended to fit a bit field which is a multiple of 8 bits. Since the dividend and divisor are 20 and 17 bit wide respectively, they are padded to 24 bits. By definition, the quotient has the same bit width as the dividend, that is 20 bits, while the fractional remainder is chosen to be 4 bits as explained.

Radix-2 algorithm offers a range of throughput options which result in different resource utilization. The most important design call during the whole development process had to do with the divider throughput choice. Due to division's high complexity, which makes it the CG system's critical operation, it is usually preferred to minimize the number of this operation in the algorithm. Since a single core can provide high throughput, it was decided that only one core for the divisions on both dimensions will be implemented, in order to efficiently exploit hardware resources.

In order to choose the throughput of the critical core, the rate at which the clusters are provided should be taken into account. It is expected that the minimum latency of the clustering block will be at least 70 cycles, which is a considerable value. This enables us to choose the slowest specification, which is one division per 8 clock cycles, in order to minimize the required resources.

In Radix-2 algorithm, latency is a function of the dividend and fractional remainder bit widths, denoted as A and F respectively, and of the selected throughput. In our case, $Latency = M + F + 3 = 27$ cycles. The complete operation of the divider core can be seen in Figure 4.13.

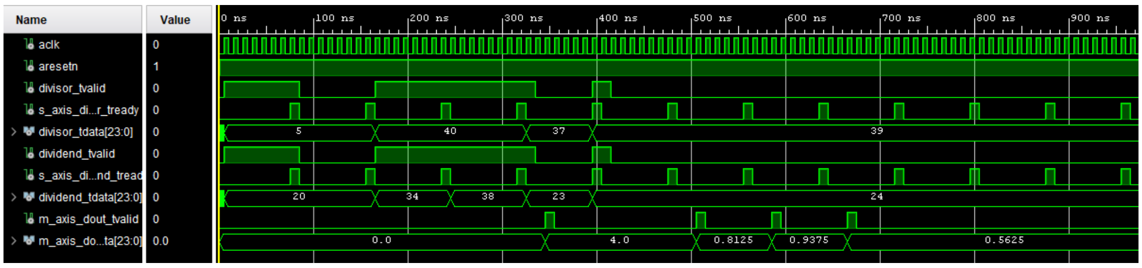


Figure 4.13: Simulation of the Divider IP Core.

As expected, the core asserts the *tready* signal every 8 cycles. If both *tready* and *tvalid* are asserted in the same cycle, a transfer occurs and the input data are read. The respective output is produced 27 cycles after the positive edge of the *tready* signal.

Buffer

Considering the fact that every cluster requires two divisions and only one core, with relatively low throughput, is implemented, a requirement for in-order buffering and retrieval arises. This behavior corresponds to a First-In-First-Out (FIFO) memory queue. As part of development, a number of custom descriptions were tested. However it was determined that the best Quality of Results (QoR) was achieved when the Xilinx LogiCORE IP FIFO Generator core was used. This fully verified core provides various optimized configurations with respect to performance and resource utilization.

For the purposes of our system, a Native Interface FIFO was used, which can be customized to utilize BRAM, Distributed RAM or Built-In FIFO resources. Since the FIFO is intended to be placed before the Divider core a specific constraint regarding its operation is imposed. As showed, the Divider core (consumer) asserts the *tready* when it's ready to process new data. In order for the core to read new data, the *tvalid* of the producer should be high at the very same cycle. The FIFO core offers two modes of operation. The standard mode is a typical memory-like mode, in which a read operation must be issued and the stored value is provided at the output port one cycle after the positive edge of the read signal. As this read operation relies on the Divider's *tready* signal, this mode of operation is not acceptable. On the contrary, the First-Word Fall-Through (FWFT) mode corresponds to a typical FIFO-like operation where the first data automatically appears on the output bus until the appropriate read signal is asserted.

Remember that the process of accepting two data in a single cycle, that is both numerators, and passing once at a time, refers also to a Parallel to Serial Conversion process. After an exhaustive exploration of the available FIFO types, it was determined that this bus width conversion is performed more efficiently when instead of two FIFOs, that store each numerator independently, a single non-symmetric FIFO is used. This kind of structure accepts two different data in a concatenated form of N bits, separates them internally, and provides at the output port one data of $\frac{N}{2}$ bits. In our case, one FIFO for storing both numerators concatenated and one FIFO that stores two duplicates of the denominator are implemented. Their RTL models are shown in Figure 4.14.

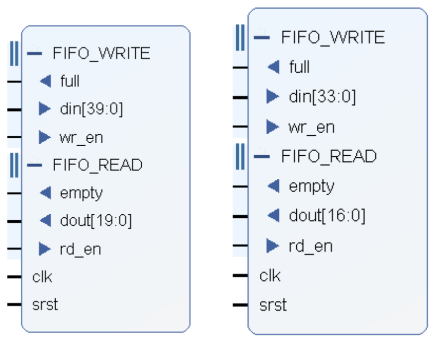


Figure 4.14: RTL block models of the FIFO IP Cores.

This function results in input ports of twice the output port's bit widths. The input data bus used for reading data is called *din*, while the output data bus driven when reading from the FIFO is called *dout*. Both FIFOs operate concurrently. The *write_en* signals are handled by the Input Block and indicate when valid sums are available for storage from MACs and AC. The *read_en* signals are connected to the Divider's *tready* signal as explained. This process, during which a following block drives a handshake signal which is used as input by a previous one is called back-pressure. The overall operation of this type of FIFOs is illustrated in Figure 4.15.

In this simulation, small values represented in the hexadecimal system are used for convenience. Notice how a complex input, which consists of two values, is fractionated and one value appears on the output in every moment. It is also important to notice that when the FIFO is empty, expressed with a high *empty* signal, and a write is issued, the newly received value is placed on the output port three cycles after the rising edge of the *write_en* signal.

When a memory module is implemented on hardware systems, it is vital to accurately determine

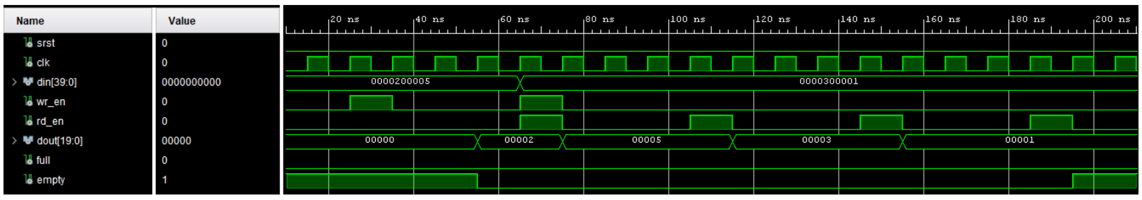


Figure 4.15: Simulation of the FIFO IP Core.

the required storage size. On the one hand, if smaller blocks than required are used, the system will probably not meet its objectives. On the other hand, if a more conservative approach is taken and a deeper FIFO is implemented, an overhead in resources occurs as more logic than necessary is assigned.

Next, an analysis regarding the required FIFO depth in our system is provided. Like the Divider throughput, the FIFO depth also depends on the characteristics of the clustering process. If it is expected that clusters are provided with a high rate, larger storing capabilities are demanded. However, since the clustering block is not developed in this thesis, the most conservative solution has been adopted.

It is known that the Divider core is able to accept new data every 8 cycles, which is expressed with a rising edge of the *tready* signal. The FIFO's input data are provided by the Input Block, $N^2 + 4$ cycles after the start of a cluster. The implementation of the FIFO cores was based on the worst case scenario, which refers to the existence of an arbitrary number of 3x3 clusters in the centroiding's input buffer. It is required that the centroiding block should be able to serve this amount of data without lagging. In this scenario, two input data are received from the Input Block every $N^2 + 3 = 12$ cycles and one output data is transmitted to the Divider core every 8 cycles. Due to the differences in input and output rates and data sizes, it is expected that the FIFO will be gradually filled. This situation can be better explained using the simulation shown in Figure 4.16.

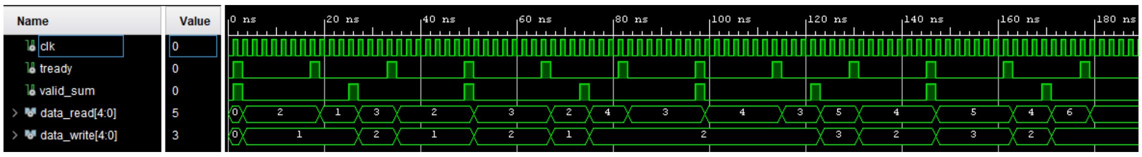


Figure 4.16: Simulation of the pre-division FIFO contents.

The relative distance between the first assertions of the *tvalid* and *valid_sum* signals is not important for the progress of the simulation. In the displayed figure, both signals are initially enabled at the same cycle, which doesn't allow the divider to read immediately the just received data. The first read operation takes place 8 cycles later. The signals *data_read* and *data_out* refer to the number of occupied memory positions in the output and input ports respectively. Remember that the implemented FIFOs are non-symmetric, which means that one input memory position corresponds to two output positions. Thus each time a high *valid_sum* is detected, one more input and two more output positions are occupied. This simulation provides a detailed insight of this process and by inspecting it, it is found that the number of occupied output positions (read positions) follows the following formula.

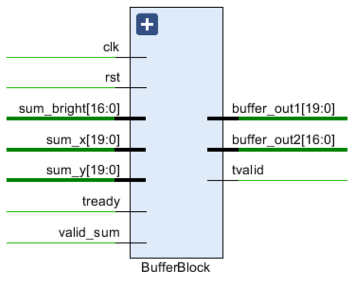
$$\text{Read Depth} = 2 + \text{ceil}(\frac{n}{2}) = 2 + \lceil\frac{n}{2}\rceil \tag{4.3}$$

where n is the number of write operations excluding the first one. In order to process a typical number of frames, that is 20, supposing that there are roughly 6 clusters in each, it is $n = 119$. Consequently we get:

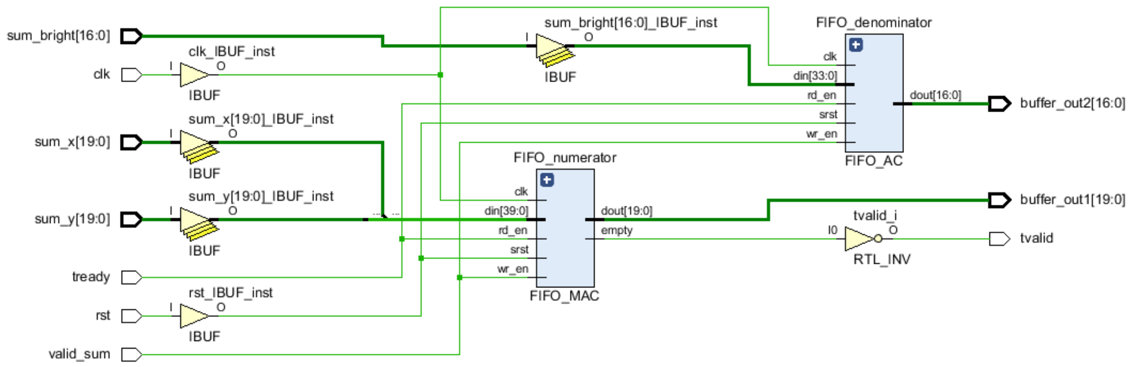
$$\text{Read Depth} = 2 + \lceil \frac{119}{2} \rceil = 62 \implies \text{Write Depth} = 31$$

Under the described assumptions, among the several depth solutions that the IP core provides, the second smallest which defines Read Depth = 64 and Write Depth = 32 was chosen. Since clustering is expected to produce data at a relatively slow rate and the existence of stacked data in the input buffer will not be the case, this approach is very conservative and will not be used in the actual implementation of the star tracker pipeline. However, due to the fact that this approach requires only the second smallest configuration, it can be safely assumed that the induced overhead in resources is not significant.

The combination of the two non-symmetric FIFOs creates the pre-division Buffer, whose RTL block model and the respective FIFO interconnections are given in Figure 4.17.



(a) RTL model.



(b) RTL schematic.

Figure 4.17: Complete RTL representation of the Buffer.

The combined operation of those 2 FIFOs as a Buffer block is tested in simulation, as Figure 4.18 shows.

Notice how a FIFO's *empty* signal is inverted in order to operate as an input *tvalid* signal for the Divider core. At this point a note regarding the *full* signal, which is asserted when a FIFO is full, should be given. This signal can be used by the CU as a control signal, in order to disable the computing modules and stop the input processing when the Buffer is full. Thus the Input Block stops the data generation and allows the consumer block, that is the Divider Block, to process the already stored data and gradually empty the occupied memory slots. Apparently, this feature also relies on the operation rates of the clustering block. Hence, in the proposed architecture the full

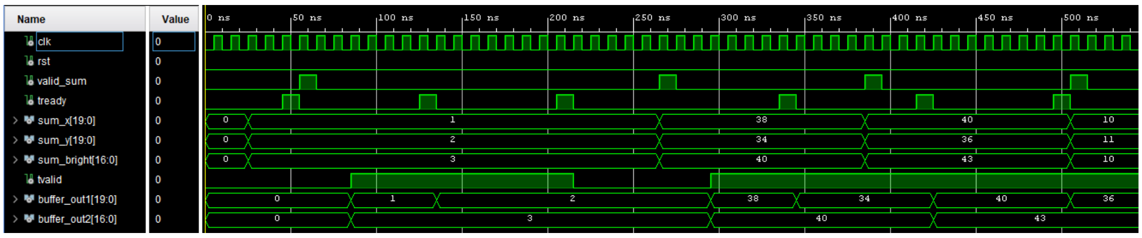
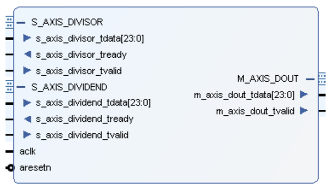


Figure 4.18: Simulation of the Buffer.

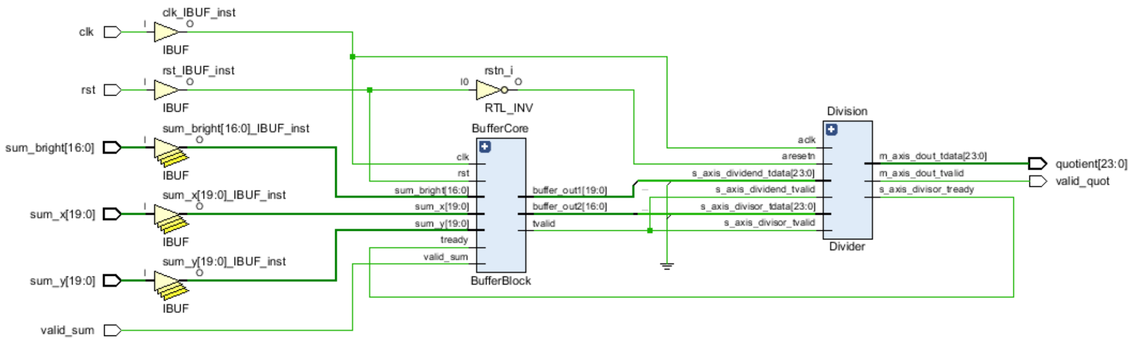
signals have not been used. However, one should remember that even when both star detection blocks are integrated together, it is not expected for the Buffer to be completely filled, due to the relatively slow rate of the clustering algorithm.

Overall Divider Block

The entire Divider Block RTL representation and the respective simulation, which can be easily observed based on the previous analysis, are provided in Figures 4.19, 4.20.



(a) RTL block model.



(b) RTL schematic.

Figure 4.19: Complete RTL representation of the Divider Block.

Marked with blue color are the internal signals that define the cooperation between the Buffer and the Divider Block. Due to the use of a buffering module, latency appears to be increased if data is presented when previous data are still stored in FIFOs. This will always be the case for the numerator on y-axis, which is used only after the other one. Hence only the Divider Block's minimum latency can be determined. If multiple clusters are congested in the input buffer, this minimum latency is noticed only during the processing of the first set of data. The minimum latency however depends also on the relative distance between the assertions of the *valid_sum* and *tready* signals. Since the empty Buffer provides valid data on the output bus 3 cycles after the respective valid input, the best case is observed when the *valid_sum* signal is enabled exactly 3 cycles before a valid *tready*. In this case the minimum latency is equal to $3 + 27 = 30$ cycles. The worst scenario regarding the relative distance of these two signals is if the *valid_sum* signal is

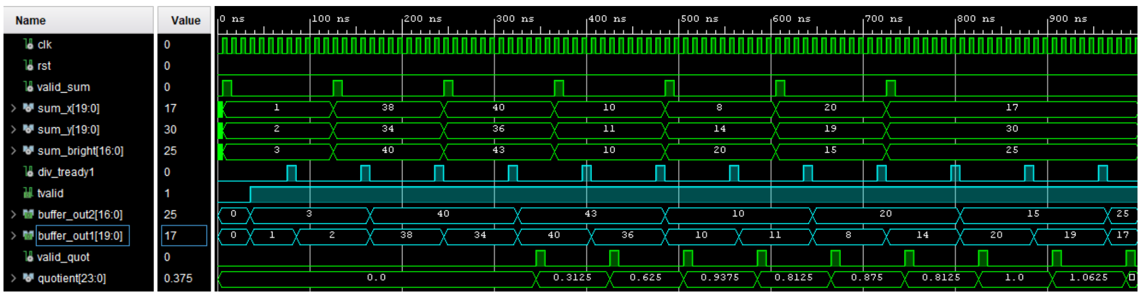


Figure 4.20: Simulation of the Divider.

asserted 2 cycles before *tready*. In this case the latency is defined as $3 + 7 + 27 = 37$ cycles.

In the integrated system that will be used in the actual application, it is expected that the latency will be always equal to $3 + 27 = 30$ cycles for the x-axis numerator and $30 + 8 = 38$ cycles for the y-axis numerator. The additional 3 cycles always appear, due to the slow producing rate that the clustering process is expected to exhibit, which means the next cluster will always be stored in an empty buffer. Hence, in order to be able to evaluate the different architectures, for the rest of this thesis, the latency of the Divider Block will be assumed equal to 38 cycles.

4.2.1.5 Output Block

The most simple block of the pipeline is the Output Block, which performs the addition of the cluster's relative subpixel centroids to the absolute coordinates in order to produce the eventual star centroid with reference to the image plane. The block diagram of the Output Block is shown in Figure 4.21.

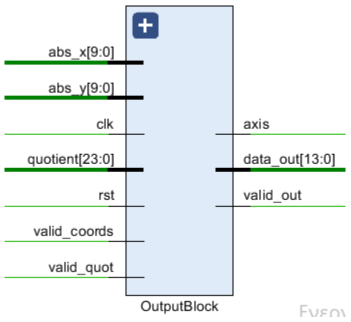


Figure 4.21: RTL model of the Output Block.

The absolute coordinates are provided by the CU, through ports *abs_x* and *abs_y*, at the same cycle when valid sums are available, and the relative centroid is driven by the Divider Block. The validity of the input depends on the values of the respective valid signals.

As explained, due to the attributes of the Divider IP core the quotient's bit width is defined as the sum of the dividend and fractional remainder bit widths, that is $20 + 4 = 24$. By definition the centroid will be detected within the range of the image plane, which is equal to the image's resolution. Since the provided image is of size 1024 x 1024, due to the binning procedure, the maximum bit width that is required to represent the entire image plane is equal to 10 bits. For this reason, internally the quotient signal is truncated and only the 14 LSBs are retained. Hence, the star centroid's value, which is transferred through the *data_out* port, will be represented using 10 integer and 4 fractional bits.

Beyond the validation signal, an auxiliary signal named axis can be used. As x- and y- centroids are not calculated concurrently and therefore a single output port will be implemented, it is essential to indicate the dimension to which a valid output value corresponds. Thus the low and high axis values correspond to the x- and y- axes respectively. It is important to highlight that in an integrated system, the axis signal can be omitted, as the matching algorithm on the CPU can be adjusted to expect a x-, y-centroid sequence for each cluster.

Similar to the Divider Block case, it is required to use a buffer structure in order to be able to store both absolute coordinates and retrieve one of them at a time. For this purpose the same type of FIFO was configured, with input and output bit widths of 20 and 10 bits respectively. In order to determine the demanded FIFO depth, an analysis based on the worst case scenario, that was described in Section 4.2.1.4, was conducted.

According to this scenario, the Input Block provides two data every 12 cycles and the Divider Block provides one quotient every 8 cycles, following an initial delay. This term refers to the Divider Block's worst minimum latency which was defined in Section 4.2.1.4 as 37 cycles. The simulation that displays this scenario is given in Figure 4.22.

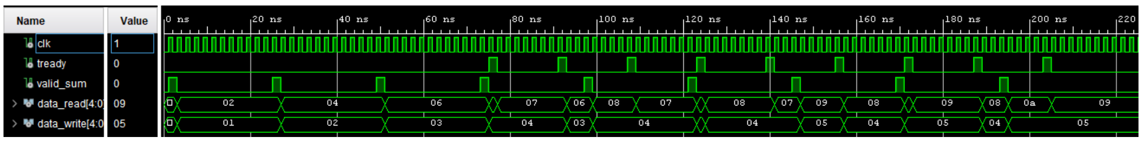


Figure 4.22: Simulation of the pre-addition FIFO contents.

Similar to the previous analysis, it can be easily found that the required output depth (read depth) at every moment follows a pattern that is described by the following formula.

$$\text{Read Depth} = 7 + \text{ceil}(\frac{n}{2}) = 7 + \lceil\frac{n}{2}\rceil \tag{4.4}$$

where n is the number of write operations excluding the initial three. In order to process a typical number of frames, that is 20, which roughly correspond to 120 clusters, it is $n = 117$. Consequently we get:

$$\text{Read Depth} = 7 + \lceil\frac{117}{2}\rceil = 66 \implies \text{Write Depth} = 33$$

Therefore the same FIFO with Read Depth = 64 and Write Depth = 32 can be implemented as part of the Output Block. It should be highlighted that the actual FIFO depth is generally influenced by the selected Read Mode, and more specifically the FWFT mode, which is adopted in our system, results in a FIFO with an actual depth which contains two slots more, compared to the GUI indication. Eventually the selected core satisfies the depth requirements. However, as was the case in the Divider Block, this approach is highly conservative and is expected to be modified during system integration. This modification refers to the use of the most shallow FIFO IP configuration available. The overall behavior of the Output Block can be seen in Figure 4.23.

Notice how the result is calculated at the same cycle with the quotient reading, which means that the Output Block contributes 1 cycle to the total latency. The add operation was implemented with LUTs and registers.

4.2.1.6 Proposed Centroiding Architecture

So far the basic building blocks of the proposed centroiding system have been extensively described. The architecture was abstractly illustrated in Figure 4.2. The generated RTL represen-

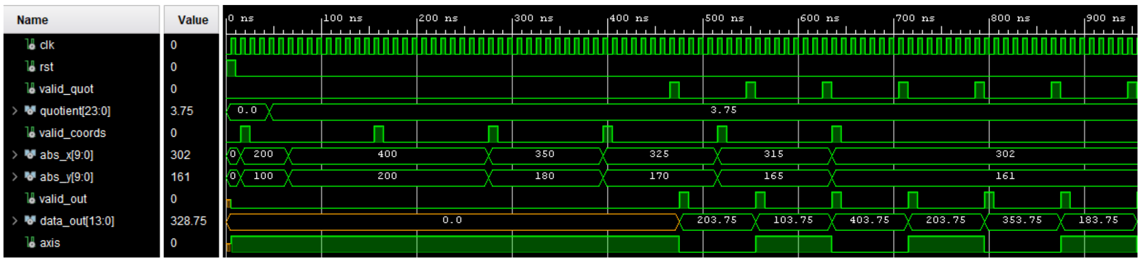


Figure 4.23: Simulation of the Output Block.

tation is displayed in Figure 4.24.

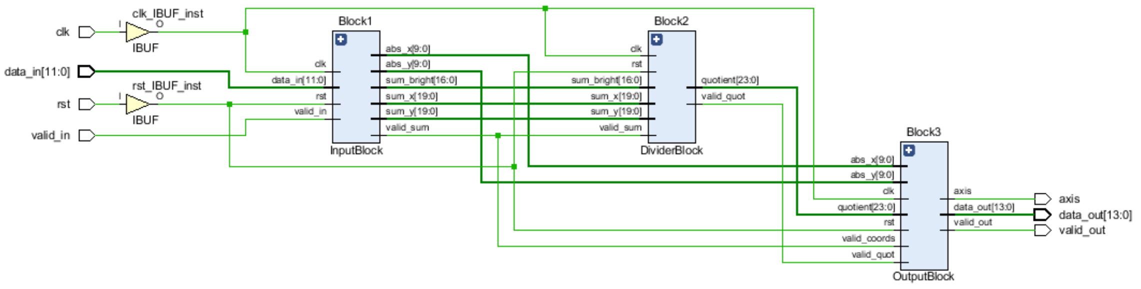


Figure 4.24: RTL model of the proposed architecture.

The entire design was tested and verified using Behavioral Simulation. The operation of the proposed architecture up until the generation of the first set of centroids is shown in Figure 4.25. Marked with blue color are the internal validation signals that provide useful insight regarding the cooperation of the previously analyzed sub-blocks.

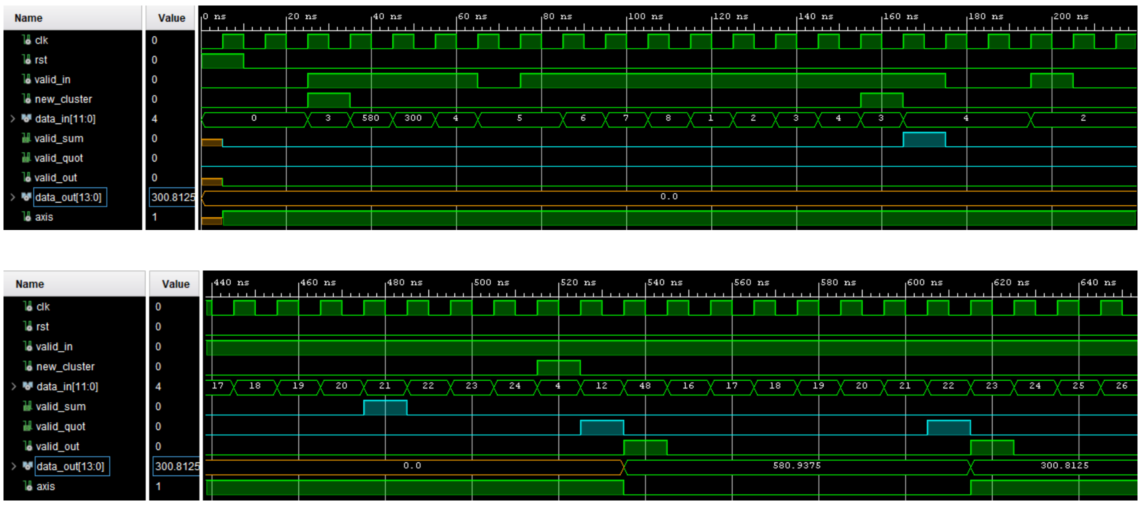


Figure 4.25: Simulation of the proposed architecture.

4.2.2 C++ Model

4.2.2.1 Overview

Typically a hardware system is implemented only once, using either VHDL or C++ modelling. However the relatively low complexity of the CG algorithm allowed us to design an additional C++ based architecture, without much effort, in order to perform rapid design space exploration and compare the Quality of Results (QoR) between the programming methods. A simplified block diagram that describes the C++ system is illustrated in Figure 4.26.

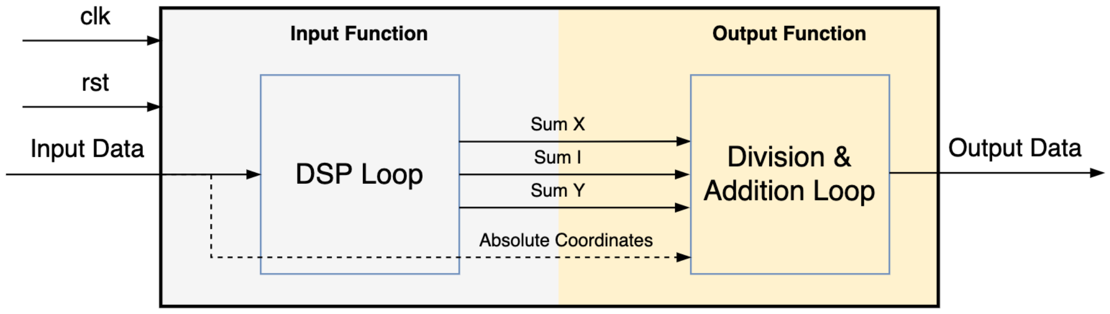


Figure 4.26: Abstract block diagram of the proposed C++ architecture.

The basic "building elements" in HLS are the C++ functions, which are synthesized into blocks in the RTL hierarchy, while the respective functions' arguments are synthesized into RTL ports. Hence, the designed system contains two basic functions that perform the operations required by the CG algorithm. It should be mentioned that the design's specifications regarding the variables' bit widths are exactly the same as for the VHDL model, as they depend solely on the algorithm.

One of the most important advantages of the HLS approach is that the compiler automatically extracts the control logic from the high level code and creates an FSM in order to sequence the existing operations. This process is the equivalent to the RTL Control Unit and consequently removes the demand of implemententing this module. The larger and more complex a system is, the greater the benefits of the automatic control logic extraction are. One can notice that in comparison to the diagram of Figure 4.2, the C++ based architecture doesn't contain any logic related modules.

When designing in High Level the clock and reset signals are also automatically inferred and there is no need for explicit definition. However the behavior of the reset signal can be configured. In this design the default functionality was selected, that resets only the control registers which are used in state machines and generate the IO protocol signals.

Usually a code that is designed for execution on a CPU, can be compiled without problem by the HLS compiler in order to create an RTL implementation. Such a CPU-based design approach however is not expected to demonstrate performance benefits. Instead, the code usually needs to be modified in order to conform with a typical hardware development methodology.

During development several modifications on the source code and concepts were tested. The typical method to evaluate them, is to execute C Synthesis and rely on the generated estimation of the results expected after RTL synthesis. When almost optimal designs were reached, Logic Synthesis and Implementation were also used to evaluate some low level details. The key features of the proposed C++ architecture are described next.

4.2.2.2 Interface Definition

Initially the appropriate interface should be determined, in order for the data transfer through the design's ports to be synchronized automatically and optimally with the internal logic. Typically, when designing in Vivado HLS a block-level I/O protocol is added, which controls the entire block. More specifically, four ports are implemented which control when the block can start processing data (*ap_start*), indicate when it is ready to accept new inputs (*ap_ready*) and if the design is idle (*ap_idle*) or has completed operation (*ap_done*). In order to perform RTL Verification, this protocol is required. However, when the complete star tracker will be integrated this protocol can be omitted.

Afterwards a port-level I/O protocol should be selected, in order to determine how the data are sequenced in and out of the implemented block. Similar to the VHDL model a streaming interface should be adopted for both input and output ports. The most suitable protocol is the AXI4-Stream, which implements a direct point-to-point communication channel between two modules in the FPGA fabric. Hence, the proposed centroiding block can receive and transmit data independently of the previous and following blocks, as long as the respective channels allow, depending on their state (full or empty). Since this interface is always extended to byte boundary, both input and output ports are 16-bit wide.

It should be noted that for convenience the previously present axis port, which indicated the dimension that corresponded to the provided centroid has not been included in this design. As mentioned, when the block is to be integrated into the complete system, the CPU side can be designed in a way to always expect sequences of x-, y-centroids. In total, both input and output AXI4-Stream channels consist of three ports, *tdata* which is the payload and *tvalid*, *tready* which perform the communication handshakes. Note that the selected protocol requires also a synchronous active low reset signal.

4.2.2.3 Input Function

The designed C++ Input Function corresponds to the Input Block that was defined in the RTL architecture. As shown in Figure 4.26 this function receives the input flow, forwards the absolute coordinates to the Output Function and calculates the numerators and the denominator of Equations 3.9, 3.10.

The entire computation is modelled by a single perfectly nested hierarchy of two loops. Considering that the expected clusters are of square shape, each loop corresponds to one dimension, while the generally variable loop limits are defined by the N package, that is the cluster's size, in each execution. For the rest of this thesis, one execution of the C++ block will be called transaction.

This loop contains three simple instructions, which correspond to the two MAC operations and the one accumulation (AC). The compiler automatically maps each MAC to a DSP core as expected. However, that is not the case for the AC operation which makes use of LUTs. It was also found that this mapping was not possible even with the use of the *Resource* optimization directive. This indicates that sometimes the HLS tool may limit the designer's accessibility to lower levels of the design.

It should be highlighted that the definition of one loop which contains all the operations is necessary in order for them to be performed in parallel. The use of three different loops would result in low performance sequential execution. Furthermore, it should be also pointed out that a nested hierarchy was preferred to a single loop with limit equal to N^2. In the first case, the multiplication factors are the loop index i or j, which correspond to the y, x dimensions respectively, and a sample from the input sequence, which corresponds to the streamed intensities. This sequence in C++ is modelled as an array or a pointer and therefore a complex expression is used to determine the "address" of each sample, as a combination of i and j. If a single loop was implemented, the array "address" would be the loop limit i, while complex expressions that would contain the division and remainder operations, with respect to the index i should be used to express the x, y dimensions-multipliers. In this case the compiler would implement these complex expressions as hardware operations, while in the first instance it infers the input "array" internally without any cost.

The loop's iteration latency is determined by the DSP cores, which demonstrate a latency of 3 cycles. In order to improve the overall throughput and latency, it is essential to pipeline this loop and allow loop iterations to be executed in an overlapping manner, as each iteration is enabled to begin before the previous one is completed. In this way an Initiation Interval (II) of 1 is achieved and the loop latency is defined as $N^2 + 1$ cycles. A view of the scheduled operations within the Input Function is provided in Figure 4.27.

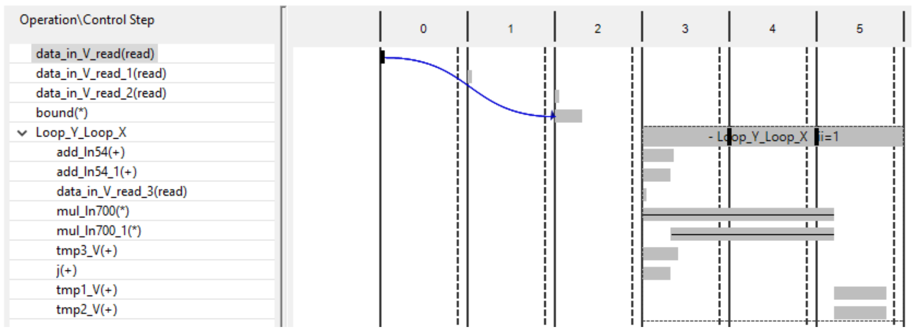


Figure 4.27: Scheduling of the operations in the Input Function

As it can be seen, the three cycles before the loop correspond to the three auxiliary input packages of each cluster. By definition, when a loop hierarchy is pipelined it is automatically flattened and therefore the calculation of the flattened loop's limit is required. This calculation is performed in parallel with the reading of the third package y_0. It should be noted that pipelining the loop increases the resource consumption. The overall latency of the Input Function is $N^2 + 5$, while apart from the loop latency and the 3 initial cycles, one additional cycle is required to enter the loop structure.

Apparently, the throughput of the Input Function defines the design's overall throughput. In the context of HLS designs, throughput is expressed through the *Initiation Interval* (II) metric, which is the number of cycles that are required before the function is executed again. The achieved II is $N^2 + 6$.

4.2.2.4 Output Function

The designed Output Function corresponds to both Divider and Output VHDL blocks and is responsible for calculating the star centroid, using the data provided by the Input Block.

Similar to the VHDL model, the fundamental requirement is the use of one Divider core in order to minimize area, while maintaining the same performance. After an exhaustive testing, it was found that the two C++ division operations are mapped to a single division core only if they are placed inside a loop, although dedicated optimization directives (*Resource*, *Allocation*) are supposedly available for this purpose. This indicates that the engineers may sometimes need to change their designing perspective, in order to control the HLS compiler's behavior.

Similar to the Input Function, the designed loop should be pipelined in order to achieve high performance and it is expected that the selected loop II will determine the throughput of the divider core. However it was discovered that regardless of the specified II, the same divider core was always implemented. It indeed demonstrated different throughput but the same amount of resources were always used for the divider core.

This behavior can be explained, as though the HLS tool always infers the same core and adjusts the external logic based on the II requirement. Hence in order to simplify the system, a divider with throughput of one input per cycle was implemented, without any increase in resources. In other words, the HLS tool doesn't allow any modification on the divider core, compared to the VHDL model in which extensive configuration was possible. This was also verified by the fact that nor the core's latency could be changed. Therefore a divider with Initiation Interval of 1 cycle and Latency of 28 cycles was used. Apparently, the latency is automatically defined based on the size of the operands. However it is greater than the latency of the IP core by 1 cycle. The divider is implemented only with LUTs and FFs.

The *loop rewind* option was also selected in order for the Output Function to achieve an overall II of 2 cycles. Otherwise the function's II would be dependent on the division latency, which is much greater than the Input Function, and the design's overall performance would be decreased. Instead, the establishment of an II of 2 cycles makes the Input Function, which exhibits an II of $N^2 + 6$, the determining factor regarding the throughput of the overall system, as expected.

The total latency of the Output Block is 30 or 31 cycles and additionally to the division delay, it contains one cycle for the addition operation and one cycle for entering the loop. The variable latency is due to the control FSM.

4.2.2.5 Overall Data Flow

The interface between sub-functions in a C++ design is critical for the overall system's performance and should be optimized. This is typically done using the powerful *Dataflow* directive, which establishes a complex communication structure and allows the respective functions to operate in parallel.

In the proposed architecture, this structure consists of five FIFO channels that store the results of each task-function and establish a completely data driven handshake interface. This data-level synchronization allows each task to execute at its own pace and therefore the overall throughput is limited by the slower one, which in our case is the Input Function. Apparently the *Dataflow* directive removes the demand of explicitly defining the intermediate FIFOs, as it was done during RTL development. This kind of function pipelining causes a great increase in hardware resources which are mainly used for implementing FIFO memories and multiplexers. The FIFOs in this case have been implemented using multiple Shift Register LUTs (SRL). The established interface can be better understood with the simulation results provided in Figure 4.28.

The *ap_start* signals activate a block in order to start processing data at the next cycle and the *ap_ready* signals indicate when a function is ready to start again, by asserting an *ap_start* signal at the next cycle. Lastly the *ap_done* signal shows that valid output data are available and that the function has completed its operation.

Notice how the Output Function is always activated one cycle after the the Input Block is completed. This behaviour ensures that the produced data will always be consumed one cycle after their production and therefore the intermediate FIFOs will never be full. Furthermore it allows resources minimization as the smallest FIFOs available, which contain only two memory

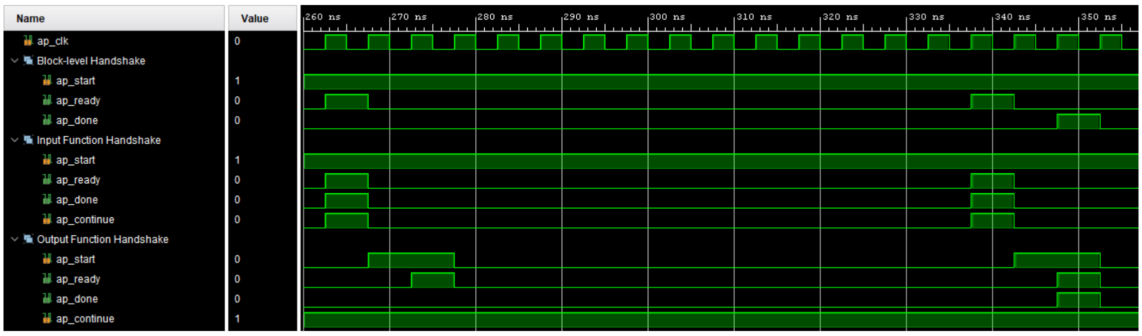


Figure 4.28: Block-level handshake between Input and Output Functions.

slots, can be used.

In Figure 4.29 one execution of the Input Function can be seen. One can note that the overall throughput, expressed with the block-level *ap_ready* signal, is indeed determined by the Input Block.

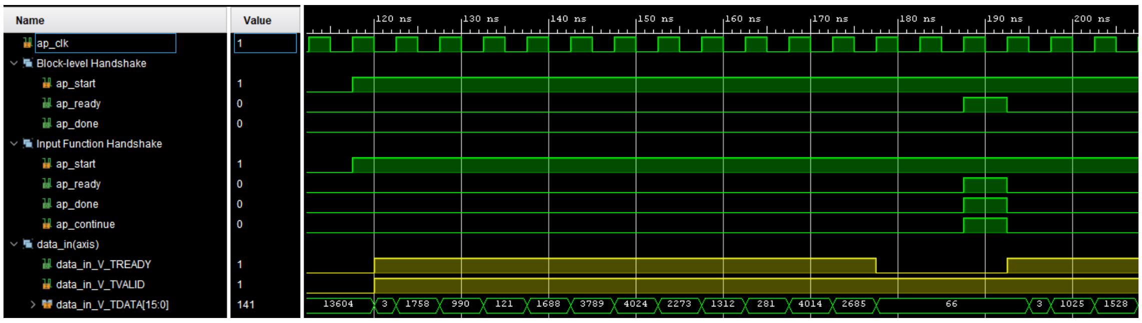


Figure 4.29: Simulation of the Input Function.

It has been proven that the C++ model demonstrates a throughput, reduced by 2 cycles compared to the VHDL model. This is due to the fact, that roughly the HLS compiler handles functions as black boxes and therefore a single function can only restart execution if it has been previously completed. In our example, the VHDL would restart immediately after the last intensity was read. However, this is not possible in the C++ case as the Input Function must complete operation, which only occurs 2 cycles after the last read operation, with the assertion of the *ap_done* signal, and then restart.

At last, one snapshot of the same simulation which illustrates how the centroid values are provided at the output bus by the Output Function is given in Figure 4.30.

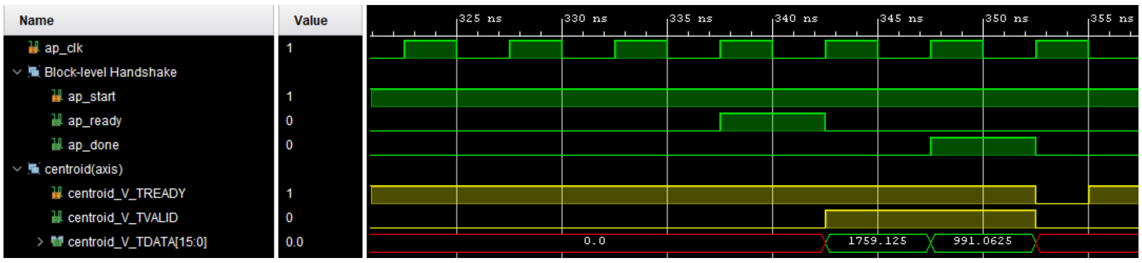


Figure 4.30: Simulation of the Output Function.

4.3 Fast Gaussian Fitting Algorithm

4.3.1 Overview

The novel FPGA implementation that this thesis proposes, is based on the Fast Gaussian Fitting Algorithm in order to achieve both high accuracy and high efficiency. Due to the high complexity of the considered algorithm, it was decided that C++ modelling was the most appropriate FPGA programming method, because it reduces development time and allows fast verification and design space exploration, which are fundamental requirements in order to reach an optimal solution. The simplified block diagram of the proposed architecture is given in Figure 4.31.

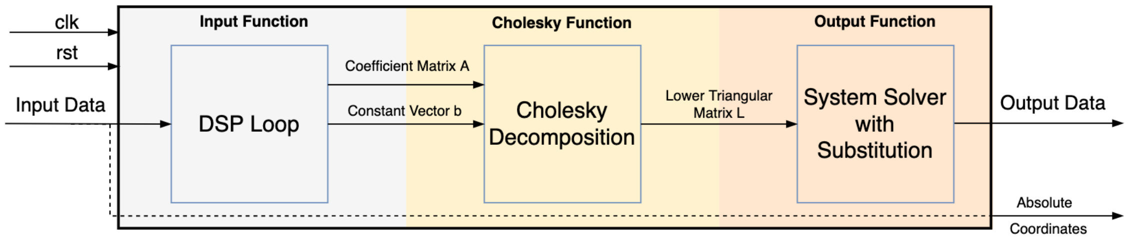


Figure 4.31: Abstract block diagram of the proposed FGF architecture.

The segmentation of the entire FGF algorithm in three different functions complies with the three different processing steps that were described in Section 3.2. The functionality of each block will be extensively explained. The basic details regarding reset behavior and I/O protocols are the same as for the CG design and will not be further discussed.

The most important block-level difference compared to the previous centroiding blocks is that the FGF system calculates the relative centroids, which refer to the distance from the cluster's first row and column, rather than the absolute centroids. This design call will be justified in the following section.

Due to this modification, a respective small adjustment at the CPU side of the SoC system is required, as the matching algorithm needs to perform the final addition in order to yield the absolute centroid positions. Moreover, it should be noted that in process integration, the clustering block could transmit each cluster's absolute coordinates directly to the processor because they aren't used within the centroiding block. However, in order to comply with the previous assumptions and be able to finally perform a valid evaluation, the input data channel was not changed.

4.3.2 Floating Point Arithmetic

It could be said, that a fundamental difference between the FGF and CG hardware implementations is that the former adopts the Floating Point instead of the Fixed Point Arithmetic. It has been explained that using Fixed Point representation of real numbers, exhibits great benefits regarding timing and resource utilization because these numbers are handled as integers.

Roughly, the FGF algorithm is expressed by 3 basic equations, which were introduced in Chapter 3 as Equations 3.33, 3.39, 3.41. These equations contain a huge number of operations on real numbers, for example a logarithm, a multiplication and a subtraction for each pixel as shown only in 3.33. In order to efficiently use fixed point numbers, the analytical determination of the size of each intermediate signal is required. The amount of these signals and the complexity of this process make this task practically impossible.

Instead, the use of floating point numbers is very easy and increases productivity, especially when programming with C++, because the respective operations are automatically mapped to Floating Point Xilinx IP cores, while conversions to and from floating point numbers are done implicitly. One could argue that in order to avoid using floating point numbers, it would be sufficient to just use a single fixed point format with a large fractional part, for example 30 bits, for all intermediate variables, to achieve high precision but lower complexity. However as it will be explained in the following section, in Input Function integers with large bit widths are calculated, which makes the above design approach inefficient, as the transformation from a large integer number to a real number during the first operation in Cholesky Function, requires an explicit bit width determination regarding the involved variables. On the contrary, Floating Point Arithmetic allows fast and sufficiently accurate processing of both small and large numbers and is the optimal solution for the examined problem. However due to this design call and the high complexity of the algorithm, it is expected that the generated circuit will demonstrate great computational demands.

At this point, it should be highlighted that the choice of not calculating the absolute centroids is based on this adoption of the single-precision floating point format. The most important attribute of this arithmetic system is that it provides great range with a cost of loss in precision. The result of this dynamic range is that the numbers that can be represented are not uniformly spaced and the difference between two consecutive representable numbers varies with the chosen scale. It is expected that the integer part of a calculated relative centroid will be a positive 1 digit decimal number as it is expressed with reference to the cluster's first row and column. If the absolute coordinate, which is the cluster's position in the image plane, was added to this value, the result would be typically three orders of magnitude larger. Since this result would be expected to be found in a different scale, its critical fractional part, which defines the achieved accuracy, would be also different, which means that a loss in accuracy would take place due to this operation. Instead it is preferred that this simple addition to be performed on the processor code, where the double-precision format can easily be used and the accuracy can be maintained.

Lastly, it is important to clarify that a use of the double-precision format in the proposed hardware implementation would result in an even larger and slower circuit without exhibiting equivalent gains in accuracy.

4.3.3 Input Function

As Figure 4.31 shows, the Input Function is responsible for calculating the coefficient matrix A and constant vector $\mathbf{b}$, that constitute the normal equations shown in Equation 3.33. This function uses the same nested loop hierarchy that was implemented in the CG centroiding block, in order to receive the pixel intensities in the same way. However, this loop contains a completely different set of operations.

4.3.3.1 Calculating Coefficient Matrix A

At first, the method for calculating the coefficient matrix A will be explained. For convenience, let us write the coefficients from Equations 3.33 in matrix notation. Since the matrix is symmetric, only the lower triangular part will be displayed.

$$A = \begin{bmatrix} \sum am_i^2 & & & & \\ \sum am_i an_i & \sum an_i^2 & & & \\ \sum am_i ap_i & \sum ap_i aq_i & \sum ap_i^2 & & \\ \sum am_i aq_i & \sum an_i aq_i & \sum ap_i aq_i & \sum aq_i^2 & \\ \sum am_i ak_i & \sum an_i ak_i & \sum ap_i ak_i & \sum aq_i ak_i & \sum ak_i^2 \end{bmatrix} \tag{4.5}$$

By using the definition of the parameters am_i, an_i, ap_i, aq_i and ak_i from Equation 3.31, and considering that the intensities I_i and the relative coordinates x_i, y_i are expressed with 12 and 3 bits respectively, the bit width of each term inside the sums can be calculated. By using the known formula $N + log_2 M$, where $M = 25$ represents the maximum number of pixels per cluster, the final bit width for each integer coefficient a_{ij} can be also found. This procedure is summarized in Table 4.1.

Table 4.1: Calculation of the bit width for each integer coefficient a_{ij}

Term	Form	Bit Width	Coefficient	Bit Width
am_i^2	$I_i^2 x_i^4$	36	a_{11}	41
an_i^2	$I_i^2 y_i^4$	36	a_{22}	41
$am_i an_i$	$I_i^2 x_i^2 y_i^2$	36	a_{12}	41
$am_i ap_i$	$I_i^2 x_i^3$	33	a_{31}	38
$am_i aq_i$	$I_i^2 x_i^2 y_i$	33	a_{41}	38
$an_i ap_i$	$I_i^2 y_i^2 x_i$	33	a_{32}	38
$an_i aq_i$	$I_i^2 y_i^3$	33	a_{42}	38
ap_i^2	$I_i^2 x_i^2$	30	a_{33}	35
aq_i^2	$I_i^2 y_i^2$	30	a_{44}	35
$ap_i aq_i$	$I_i^2 x_i y_i$	30	a_{43}	35
$ap_i ak_i$	$I_i^2 x_i$	27	a_{53}	32
$aq_i ak_i$	$I_i^2 y_i$	27	a_{54}	32
ak_i^2	I_i^2	24	a_{55}	29

One should notice that there are 13 unique coefficients displayed in Table 4.1, because the coefficients a_{51} and a_{41} if expanded, are found equal to a_{33} and a_{44} respectively.

The most important design challenge that was faced during development had to do with the way that each of the displayed products is implemented on the FPGA. Let us give a simple example. Consider the coefficient $a_{53} = \sum I_i^2 x_i$. For the calculation of this term, a MAC operation is performed for each pixel. However the product $I_i^2 x_i$ that is accumulated each time, can be computed in two different ways, either as $I_i^2 \cdot x_i$ or $(I_i x_i) \cdot I_i$. In a conventional processor, the sequence of calculating such integer products is insignificant, but in hardware designs that is not the case. Consider the following segment of C++ code that calculates both a_{53} and a_{54}, which are of the same form, as only the coordinate term changes.

```
tmp53 = Ii * Ii; a53 = tmp53 * xi;
tmp54 = Ii * yi; a54 = tmp54 * Ii;
```

It was found that the HLS compiler implements the final a_{53} and a_{54} products in a different way. The first product is generated by a multiplication of 24x3 bits and is implemented using LUTs, while the second product is generated by a multiplication of 15x12 bits and the compiler implements it by using a DSP core. This is due to the fact that the inference of DSP cores for a certain multiplication depends on the bit widths of its input operands. A general rule is that in order to map a multiplication to a DSP cell, it needs to involve operands wider or equal to 10 bits. Otherwise, multiplications lower than this limit are implemented using LUTs. Furthermore for the DSP48E cell, that is used by the ZYNQ-7020 SoC, the two involved operands can be at most 25 and 18 bits wide respectively. If any of the operands exceeds the defined limits, multiple DSPs will be combined for the product calculation.

In some applications there are no specific requirements regarding the way a multiplication is implemented, while in others this detail is critical as far as latency or throughput are concerned. In our case, it was proven that when one of the involved operands was small, the use of LUTs was creating a bottleneck regarding the maximum achievable system's frequency. More specifically, because the second operand was represented using a relatively large number of bits (i.e., variable tmp53 is 24 bits wide), a multiplication structure that consisted of a large number of LUTs was created. Ultimately it was found that within these structures long critical paths were emerged, which were limiting the overall performance. For this reason, the implementation of all 13 coefficient products using DSP cores is of critical importance.

In order for the DSP cores to be used, only integer multiplications must be performed and the respective operands are defined based on Table 4.2. As described in the previous section, this constraint, regarding the required range of numbers, affected the design call of using the floating point system.

Moreover, it was also found that a simple DSP-based implementation was not always enough. In some cases a loss of performance occurred, due to the precedent multiplications which formed the auxiliary terms. For example, if the coefficient am_i was defined as $I_i^2 \cdot x_i^4$, a long logic chain inside the LUT structures that calculated the auxiliary term as $x^4 = x^2x^2$ was observed. In order to achieve maximum performance, an exhaustive space exploration which tested all possible multiplication formats, 31 in number, was conducted. The results of this research are summarized in Table 4.2, where the multiplications that generate each type of coefficient are displayed.

For convenience, we define the *order* of a coefficient based on the number of coordinate terms that are included in the product. Since both x_i and y_i have the same size, they contribute to the result in the same way and thus they are considered as a single coordinate unit. For example the terms $I_i^2 x_i^4$ and $I_i^2 x_i^2 y_i^2$ are both of 4th order, while the term $I_i^2 x_i$ is of 1st order. In the summarized results both coordinate coordinates are denoted as $\mathbf{x_i}$.

Table 4.2: Definition of each coefficient type, based on the multiplicand and multiplier formats.

Order	Form	Multiplication	Bit Width	Auxiliary Term
4	$I_i^2\mathbf{x}_i^4$	$(I_i\mathbf{x}_i^2)(I_i\mathbf{x}_i^2)$	18x18	$I_i\mathbf{x}_i^2$, $I_i\mathbf{x}_i$
3	$I_i^2\mathbf{x}_i^3$	$(I_i\mathbf{x}_i^2)(I_i\mathbf{x}_i)$	18x15	$I_i\mathbf{x}_i^2$
2	$I_i^2\mathbf{x}_i^2$	$(I_i\mathbf{x}_i)(I_i\mathbf{x}_i)$	15x15	$I_i\mathbf{x}_i$
1	$I_i^2\mathbf{x}_i$	$(I_i\mathbf{x}_i)(I_i)$	15x12	$I_i\mathbf{x}_i$

Apparently, the illustrated optimal formats require the calculation of the following auxiliary terms, which are all implemented with LUTs.

$$I_i x_i$$
$$I_i y_i$$
$$I_i x_i^2 = (I_i x_i)(x_i)$$
$$I_i y_i^2 = (I_i y_i)(y_i)$$

4.3.3.2 Calculating Constant Vector b

The second set of operations inside the Input Function's loop is concerned with the calculation of the constant terms b_i of the normal equations system (Equation 3.33). Les us write these terms in matrix notation and use the the definition of the involved parameters, found in Equation 3.31, to write the expressions with respect to the input data.

$$\mathbf{b} = \begin{bmatrix} -\sum am_i a_i \\ -\sum an_i a_i \\ -\sum ap_i a_i \\ -\sum aq_i a_i \\ -\sum ak_i ai_i \end{bmatrix} = \begin{bmatrix} -\sum (I_i^2 x_i^2) \ln I_i \\ -\sum (I_i^2 y_i^2) \ln I_i \\ -\sum (I_i^2 x_i) \ln I_i \\ -\sum (I_i^2 y_i) \ln I_i \\ -\sum (I_i^2) \ln I_i \end{bmatrix} \tag{4.6}$$

The terms inside the parentheses are also required for the coefficient matrix A and can be considered to be known. The logarithm operation is the first encountered operation which involves real numbers. In order for this calculation to be performed, the HLS compiler implements a Conversion Block using logic cells (LUTs, FFs) to cast the input intensities from integers to floats. The number of the generated conversion blocks was found, during testing, to be dependent on the loop's interval, due to the different scheduling that the compiler accomplishes in each case. The following floating point logarithmic operation is performed using a dedicated floating point core which uses 13 DSP cells. Similarly, the terms inside the parantheses are converted by the Conversion Block and the floating point multiplication and subtraction operations are performed. The inferred floating point multiplication and subtraction cores were implemented using 3 and 2 DSPs respectively. These cores are fully pipelined and therefore a unique core for each operation was used.

As theoretically expected, it was proven that the loop's latency and interval were determined only by the floating point operations. Specifically the floating point logarithmic operation demonstrated a constant latency of 19 cycles. However it was found that the latency of the multiplication and subtraction operations depends on the target clock constraint. Typical values were 5 cycles for the multiplication and 9 cycles for the subtraction. Also, the conversion from integer to floating point representation demonstrated a latency of 8 cycles. Therefore the sequence of operations required for calculating a constant term b_i, which consists of one conversion, one logarithmic operation, one multiplication and one subtraction, determines the latency of one iteration.

Moreover, it was found that the floating point subtraction was the limiting factor regarding the minimum achieved Iteration Interval, which is ultimately defined as equal to the subtraction core's latency. It should be highlighted that the floating point operations contribute approximately to the half of the FFs and the two thirds of the LUTs that are used by the Input Function.

In total, it can be concluded that the use of floating point cores limits the Input Function's performance and consequently the system's overall performance, and determines the total resource consumption.

4.3.3.3 Loss in Accuracy: Converting an Integer to Floating Point

In Section 4.3.3.1, the method of implementing the coefficients a_{ij} has been extensively described. It was also explained that in order to efficiently perform the involved multiplications, DSP cores should be used, which requires the operands to be represented as integers. Ultimately this process results in the creation of integers of significant bit widths, e.g., am_i is expressed with 41 bits.

In the following section, it will be shown that all of the operations within the Cholesky Function involve real numbers, which means that the integer coefficient matrix A should be converted to its floating point equivalent. In order to exploit the conversion block that was by definition inferred inside the Input Function, it was decided that the conversion of the coefficients should be also performed inside the first function, and specifically inside the loop.

However, a critical point that needs clarification refers to the amount of accuracy that is lost during transforming a very wide integer (e.g. of 41 bits) to a narrower 32-bit single-precision floating point number. It is known that in different scales, the distance between consecutive representable integers also differs. For example, even though integers smaller than 2^{24} are exactly represented, larger integers are rounded. Typically integers between 2^n and 2^{n+1} round to a multiple of 2^{n-23}.

Consequently if a a_{11} coefficient is actually expressed with 41 bits, then the cast to the respective floating point number can cause in the worst case an accuracy loss of $2^{n-23} = 2^{40-23} = 131072$. Although this number is quite large, what is most important is the relative loss in accuracy, which is defined as the absolute value divided by the actual integer that can be represented in 41 bits, and shows the loss with respect to the initial order of magnitude. Thus, for the case of a 41-bit coefficient a_{11} we have:

$$\text{Relative Accuracy Loss} = \frac{\text{Absolute Accuracy Loss}}{\text{Order of Magnitude}} = \frac{2^{n-23}}{2^{n+1}} = \frac{2^{40-23}}{2^{41}} = 2^{-24} \approx 10^{-8}$$

Apparently, this accuracy loss is insignificant and therefore the conversion to floating point numbers can be safely done without any concerns regarding the centroiding accuracy.

4.3.4 Cholesky Function

Once the normal equations are defined, the Cholesky algorithm can be used in order to decompose the coefficient matrix A according to the formula below, which is given in Equation 3.39.

$$A = LL^\top$$

where L is the lower triangular matrix that will be used later. The Cholesky Algorithm is defined by the following formulas.

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \tag{4.7}$$

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) \tag{4.8}$$

where l_{ii}, l_{ij} are the elements on and outside the main diagonal respectively.

The HLS tool provides a number of commonly used C++ linear algebra functions, which include the Cholesky Function, with the HLS Linear Algebra Library. For every function, there are several implementations available that offer different levels of optimization. Extensive analysis and testing demonstrated that the three available Cholesky architectures were implemented as iterative algorithms with certain loop hierarchies, and each of them adopted a different processing and algorithmic approach. Consequently, these architectures displayed non-pipelined behavior because their Iteration Interval was defined as 1 cycle greater than their Latency. Considering the fact that this II was at best 5 times larger than the respective II of the Input Function, it is obvious that if any of these Cholesky architectures was used, it would result in a low performance non-pipelined centroiding block.

In order to achieve the desired performance levels a lower II was required, regardless of the final latency. This was achieved by pipelining a given Cholesky function with the *Pipeline* directive, which actually dissolved the loop hierarchy and created an unrolled data pipeline. By analyzing the generated circuit, it was found that in this way it indeed followed the underlying sequence of operations which is defined from Equations 4.7 and 4.8. This sequence of operations can be better understood, by inspecting the pseudocode below, which describes the Cholesky algorithm.

Algorithm 1 Cholesky Algorithm

```
1: for all columns $j$ do
2:     calculate the diagonal element $l_{jj} = l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$, where $i = j$
3:     for all rows $i$ do
4:         calculate the rest of the column elements $l_{ij} = \frac{1}{l_{jj}} \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right)$
5:     end for
6: end for
```

The produced lower triangular matrix L is defined as follows.

$$\begin{array}{lllll}
\mathbf{l_{11}} = \sqrt{a_{11}} & & & & \\
\mathbf{l_{21}} = \frac{a_{21}}{\mathbf{l_{11}}} & \mathbf{l_{22}} = \sqrt{a_{22} - \mathbf{l_{21}}^2} & & & \\
l_{31} = \frac{a_{31}}{l_{11}} & \mathbf{l_{32}} = \frac{a_{32} - l_{21} l_{31}}{\mathbf{l_{22}}} & \mathbf{l_{33}} = \sqrt{a_{33} - l_{31}^2 - \mathbf{l_{32}}^2} & & \\
l_{41} = \frac{a_{41}}{l_{11}} & l_{42} = \frac{a_{42} - l_{21} l_{41}}{l_{22}} & \mathbf{l_{43}} = \frac{a_{43} - l_{31} l_{41} - l_{32} l_{42}}{\mathbf{l_{33}}} & \mathbf{l_{44}} = \sqrt{a_{44} - l_{41}^2 - l_{42}^2 - \mathbf{l_{43}}^2} & \\
l_{51} = \frac{a_{51}}{l_{11}} & l_{52} = \frac{a_{52} - l_{21} l_{51}}{l_{22}} & l_{53} = \frac{a_{53} - l_{31} l_{51} - l_{32} l_{52}}{l_{33}} & \mathbf{l_{54}} = \frac{a_{54} - l_{41} l_{51} - l_{42} l_{52} - l_{43} l_{53}}{\mathbf{l_{44}}} & \mathbf{l_{55}} = \sqrt{a_{55} - l_{51}^2 - l_{52}^2 - l_{53}^2 - \mathbf{l_{54}}^2}
\end{array}$$

Pipelining the Cholesky algorithm expressed in the pseudocode 1, allows operations to be implemented in an overlapping manner. However this pipeline is limited by the inherent dependencies of the Cholesky method. These dependencies refer to the matrix elements which have been marked with bold above. As one can see, in order to start processing the next column, that is to start calculating the diagonal element of the next column, the element at the left of this diagonal element (same row, previous column) should have been previously calculated. For example the element l_{22} requires the previous calculation of the l_{21} and the element l_{33} requires the l_{32}. Each of those "previous" elements can be calculated only after the calculation of the respective column's diagonal value. For instance, the element l_{21} requires the l_{11} and the element l_{32} requires the l_{22} diagonal value. Consequently, the performance of the Cholesky Function is limited by the dependency chain which starts at the first element of the L matrix, that is l_{11}, follows the bold elements by moving down and right and ends at the last element l_{55}.

This processing chain consists of three operations that are repeated for each column. A floating point subtraction to calculate the term inside the square root of each diagonal element, except for the first element l_{11} for which the subtracted value is 0, a floating point square root operation to calculate the actual diagonal value and a floating point division to compute the element below the diagonal one. In order to improve performance, the HLS Cholesky Function instead of performing the typical division operation to calculate the values outside the diagonal, it computes the diagonal element's reciprocal and then performs a floating point multiplication.

The HLS tool contains a dedicated floating point reciprocal square root core that can be used in order to calculate the value $\frac{1}{l_{ii}} = \frac{1}{\sqrt{\cdots}}$, in parallel with the actual diagonal value $l_{ii} = \sqrt{\ldots}$ which is computed with a simple square root operation. It was found that the HLS compiler schedules the square root with a latency of 28 cycles, the reciprocal square root with a latency of 25 cycles and the division with a latency of 30 cycles. Therefore, this alternative approach of calculating the elements outside the diagonal, demonstrates a latency of $25 + 5 = 30$ cycles, where 5 is a typical value for the floating point multiplication latency. If the standard floating point division was used however, the total latency would be $28 + 30 = 58$ cycles. Since this calculation is performed 4 times within the dependency chain, the total gain in latency is approximately 140 cycles. However it comes with a cost in resource consumption, as the floating point reciprocal square root core is implemented with 9 DSPs, while the division core would require almost 800 FFs and 700 LUTs.

It should be mentioned that all of the referred cores are fully pipelined, which means that the Cholesky pipeline uses only one instance of each one. As it was pointed out during the Input Function analysis, the floating point addition and multiplication cores require 2 and 3 DSPs respectively. Thus the total DSP consumption within the Cholesky Function is 14 cells. In total, the proposed loop unrolling is essential in order to achieve the highest possible performance, but it results in a great increase in area which is mainly expressed by the usage of additional 1000 FFs.

4.3.5 Output Function

In Section 3.2.4.3 it has been explained that the Cholesky algorithm is used in order to decompose the given linear system of normal equation and create two more simple problems. These systems are defined in Equations 3.42, 3.43 and are shown below.

$$L^{\top}\mathbf{x} = \mathbf{y}$$
$$L\mathbf{y} = \mathbf{b}$$

Both equations are simple enough to be solved with the substitution method. Considering that the matrix L was calculated by the Cholesky Function and that the vector $\mathbf{b}$ was provided by the Input Function, the second equation can be solved at first. Let us rewrite the equation in matrix notation.

$$\begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ l_{41} & l_{42} & l_{43} & l_{44} & \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} \tag{4.9}$$

Apparently the five equations that determine the unknown variables y_i are the following.

$$y_1 = \frac{b_1}{L_{11}} \tag{4.10a}$$

$$y_2 = \frac{b_2 - L_{21}\mathbf{y_1}}{L_{22}} \tag{4.10b}$$

$$y_3 = \frac{b_3 - L_{31}y_1 - L_{32}\mathbf{y_2}}{L_{33}} \tag{4.10c}$$

$$y_4 = \frac{b_4 - L_{41}y_1 - L_{42}y_2 - L_{43}\mathbf{y_3}}{L_{44}} \tag{4.10d}$$

$$y_5 = \frac{b_5 - L_{51}y_1 - L_{52}y_2 - L_{53}y_3 - L_{54}\mathbf{y_4}}{L_{55}} \tag{4.10e}$$

By inspecting the terms marked with bold, it can be seen that each result unknown y_i is used on the numerator of the next unknown y_{i+1}. This inherent dependency limits the function's performance, as it defines the minimum possible latency. This limit is determined by the floating point operations that constitute the observed dependency chain and are repeated for each term y_i. This set of operations includes a multiplication, a substitution and a division. Since the inferred floating point cores are fully pipelined, the rest operations that are located in each numerator are performed in parallel, by using a single instance for each operation.

As already explained, the division core demonstrates a latency of 30 cycles and is implemented only with FPGA primitives (LUTs, FFs). Furthermore, the subtraction and multiplication operations have typical latencies of 9 and 5 cycles and are constructed with 2 and 3 DSPs respectively. In the same way, we solve the second equation, which is shown in matrix notation below.

$$\begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & l_{51} \\ & l_{22} & l_{32} & l_{42} & l_{52} \\ & & l_{33} & l_{43} & l_{53} \\ & & & l_{44} & l_{54} \\ & & & & l_{55} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \tag{4.11}$$

With simple backward substitution we get the following final equations, that solve the system and provide the desired parameter vector $\mathbf{x}$.

$$x_5 = \frac{\mathbf{y_5}}{L_{55}} \tag{4.12a}$$

$$x_4 = \frac{y_4 - L_{45}\mathbf{x_5}}{L_{44}} \tag{4.12b}$$

$$x_3 = \frac{y_3 - L_{35}x_5 - L_{34}\mathbf{x_4}}{L_{33}} \tag{4.12c}$$

$$x_2 = \frac{y_2 - L_{25}x_5 - L_{24}x_4 - L_{23}\mathbf{x_3}}{L_{22}} \tag{4.12d}$$

$$x_1 = \frac{y_1 - L_{15}x_5 - L_{14}x_4 - L_{13}x_3 - L_{12}\mathbf{x_2}}{L_{11}} \tag{4.12e}$$

As expected, the same type of dependencies are also present in Equations 4.12, while the term x_5 depends on the term y_5 of Equation 4.10e. The previously inferred floating point cores are also used for the calculations involved in this set of equations.

Ultimately, the relative centroids are estimated using the following two floating point operations.

$$y_c = -\frac{x_4}{2x_2} \tag{4.13a}$$

$$x_c = -\frac{x_3}{2x_1} \tag{4.13b}$$

In order for the division in Equation 4.13b to be performed immediately after the computation of the unknown x_1, the calculation should be performed as defined below.

$$x_c = (-0.5\, x_3) \cdot \frac{1}{x_1} \tag{4.14}$$

This is not required however for the y_c centroid, while both multiplication and division are scheduled in parallel with other operations which produce their results later. In conclusion, the latency of the Output Function is determined by the described dependency chain, which consists of 11 divisions, 8 multiplications and 8 subtractions. It should be also highlighted that the rest of the computations are performed in parallel with the critical calculations and a single core for each operation is inferred.

4.3.6 Proposed Centroiding Architecture

So far the basic building blocks of the proposed architecture have been extensively described. It has been explained that in C++ hardware modelling, the interaction between the sub-functions is critical for the overall performance. In our system this interaction is automatically optimized by using the *Dataflow* directive.

Since the Cholesky and Output functions are implemented as unrolled data pipelines, we are able to define an arbitrary small Initiation Interval, which however will result in greater resource consumption for higher throughput. By contrast, the Input Function which corresponds to a rolled loop pipeline is characterized by a maximum throughput due to the constraints that the computing cores impose. We saw that the floating point subtraction operation determines the iteration latency and thus both the latency and the II of the Input Function. Consequently, demanding lower intervals from the succeeding functions is pointless, as the overall throughput of the design will be eventually determined by the II of the Input Block. This behavior is illustrated in Figure 4.32, as the overall system's *ap_ready* signal is driven by the respective signal of the Input Function.

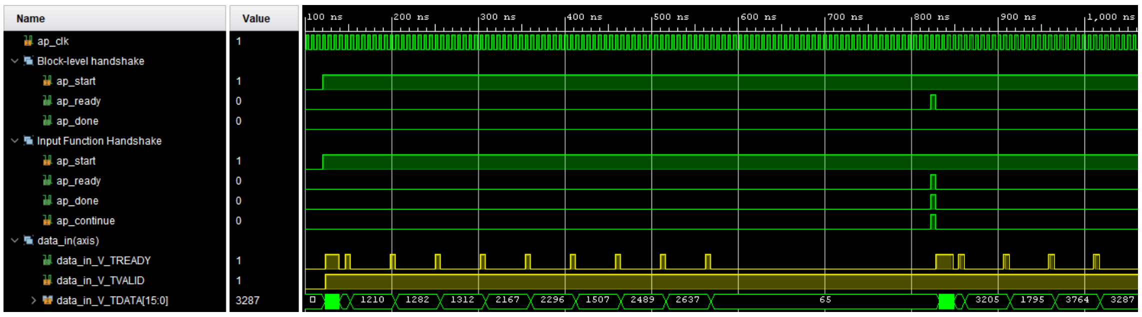


Figure 4.32: Simulation of the Input Function.

Notice how the centroiding block, reads consecutive input samples (i.e., intensities) with an II greater than 1, due to the limitation that the floating point core imposes, while the CG system was able to process new data in each cycle. Furthermore, the way that the proposed block outputs the detected centroids is displayed in Figure 4.33.

Ultimately in order to fully exploit the benefits of the *Dataflow* directive, the Initiation Interval of both the Cholesky and the Output functions should be defined as equal to the respective value of the Input Function. In this way, all the sub-blocks will execute at the same rate and they will be

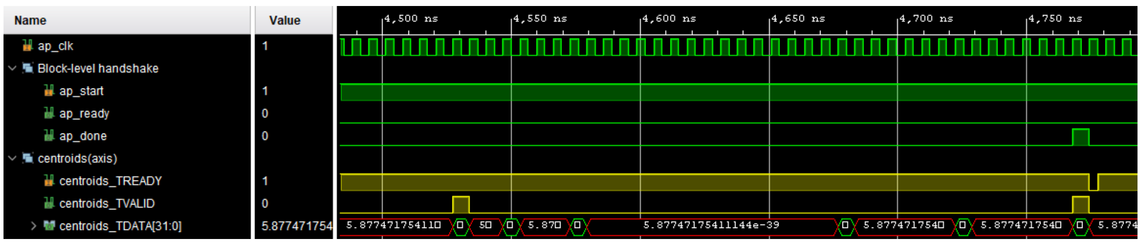


Figure 4.33: Simulation of the Output Function.

always able to consume the respective input data, one cycle after their production. The interaction between the first two functions is shown in Figure 4.34.

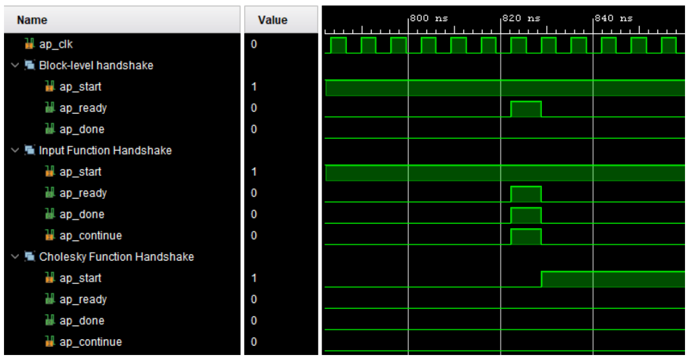


Figure 4.34: Block-level handshake between Input and Cholesky Functions.

As expected, the Cholesky Function will always restart one cycle after the production of its input data, that is the coefficient matrix A. Therefore, the FIFO channel of 2 slots that the HLS compiler used for the specific communication channel is sufficient. Figure 4.35 illustrates the communication channel between the Input and Output functions, which involves the constant vector $\mathbf{b}$.

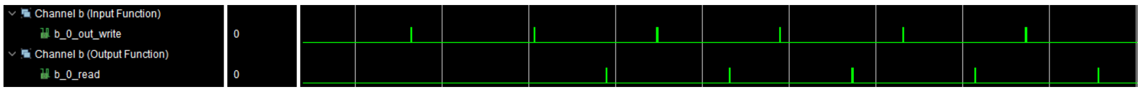


Figure 4.35: Block-level handshake between Input and Output Functions with regard to data $\mathbf{b}$.

This channel is generally more challenging, as the produced data cannot be consumed before the intermediate Cholesky function is complete, which could result in a bottleneck if enough data to fill the FIFO block were produced before the Output Block could start processing them. However as it seems, the Cholesky function demonstrates a delay which allows the activation of the Output function before the channel is blocked. Even though a FIFO of 2 slots is sufficient, the HLS compiler implements a channel with depth = 3, a feature which the tool doesn't allow the designer to modify.

At this point, one last comment regarding the implementation of the C++ variables that correspond to the transferred data between the sub-functions, that is matrices A, L and vector $\mathbf{b}$, should be made. The coefficient matrix A is required to be implemented as a set of registers, as this format allows convenient handling within the Input Loop. On the other hand, if the matrix was mapped to a Block RAM, a decrease in performance would be observed, because additional cycles would be required in order to initialize the values of the C++ array and to issue the final write operations, since a BRAM on the FPGA has a limited number on ports. Therefore the *Array Partition* directive was used in order for all of the 15 unique coefficients to be implemented as registers. Similarly, the constant terms have also been mapped to registers. Regarding the triangular matrix L, due to the way that the Cholesky and Output functions process the data, both channel types, that is a set of registers or a BRAM, are acceptable. Because the transferred data are not so many and therefore the benefits of using a Block RAM are reduced, it was decided to implement the elements of the matrix L as registers by also using the *Array Parition* directive.

Chapter 5

Design Evaluation

Once the theoretical and technical details of the proposed centroid detection designs have been established, it is vital to conduct a set of experiments in order to evaluate their performance. More specifically, simulated data were used and the hardware implementations were compared to the respective software algorithms regarding the accuracy and computational time. Furthermore, the FPGA resources demands of the various hardware models are discussed.

All of the software experiments are carried on the MATLAB 2016A platform, running on a dual-core Intel Core i5 2.6 GHz processor. The VHDL system was developed on Xilinix Vivado Design Suite 2019.1, while Xilinx Vivado HLS 2019.1 was used for the C++ designs. All of the models have been implemented on the Xilinx ZYNQ-7020 SoC which is hosted on the ZedBoard Development Board.

5.1 Dataset Generation

Since this thesis focuses only on the centroiding process and an end-to-end system is not available, simulated star clusters were created in order to conduct the appropriate tests. For this purpose a Point Spread Function (PSF), developed in MATLAB, was provided by Infinite Orbits, which generates clusters according to the following equation.

$$f = A \exp\left(-\frac{(x_i - x_c)^2}{2\sigma_x^2} - \frac{(y_i - y_c)^2}{\sigma_y^2}\right) \tag{5.1}$$

where x_i, y_i represent the i-th pixel in the image; x_c, y_c are the ground truth centroids of the cluster; σ_x, σ_y represent the standard deviation of the function and A is the amplitude, which represents the cluster's brightness level.

In this chapter we compare the differences between the hardware and software implementations of a specific algorithm and not the individual algorithms. More about the performance levels of each algorithm are discussed in Section 3.2.5. Therefore, instead of evaluating the results on a wide range of the Gaussian parameters, a specific set has been selected. Extensive exploration was conducted in collaboration with Infinite Orbits, in order to find a set of parameters that create clusters, on which both algorithms display sufficient performance. This parameter set is explained below.

5.1.1 Centroid Position

Throughout this thesis it has been assumed that an accurate clustering algorithm generates the clusters on which the centroiding algorithms operate. The centroid in such clusters is expected to be located close to the center, which ensures that all pixels demonstrate a sufficient value of intensity. Therefore for a cluster of size NxN, a centroid is placed within the following interval:

$$(x_c,\, y_c) \in (0.25N, 0.75N) \tag{5.2}$$

It was observed, as expected, that if a ground truth centroid was placed in an edge pixel, not enough bright pixels would exist inside the cluster and the Center of Gravity algorithm would fail.

5.1.2 Brightness Level

Considering the thresholding process that takes place during clustering, quite bright pixels are expected. However, during simulation it was found that the amplitude parameter doesn't heavily affect the results, thus lower levels of brightness were also allowed. This parameter is restricted within the following interval.

$$A \in (0.1, 1) \tag{5.3}$$

It should be noted, that no care has been taken regarding saturated pixels, and they may as well be detected in the generated clusters.

5.1.3 Gaussian Radius

The most important parameter of the generated PSF was the two-dimensional standard deviation of the Gaussian distribution, which models the spread of the star spot and in reality it depends on lens configuration. In order for both algorithms to achieve acceptable performance, a symmetric PSF was assumed and the Gaussian radius σ was defined as follows.

$$\sigma = \sigma_x = \sigma_y = 1.25 \tag{5.4}$$

Using the described PSF, three datasets of 10,000 samples were created, which contained 3x3, 5x5 and both 3x3, 5x5 star clusters respectively. Lastly, we highlight that noise hasn't been added to the simulated data, because it is expected that noise would only affect the general performance of the algorithm and not the differences between hardware and software results, which is the main focus of the presented evaluation.

5.2 Accuracy Experiment

A possible shortcoming when implementing an algorithm on hardware, is a loss in precision. Therefore, it is vital to evaluate the subpixel accuracy of the proposed designs by comparing the provided centroid positions to the respective results produced in software. For this reason, Infinite Orbits developed a MATLAB function for each algorithm, which was used to produce the baseline results.

The accuracy of each implementation was measured using as error metric the euclidean distance between the centroid estimated on hardware (x_h, y_h) and the centroid calulated on software (x_s, y_s), as shown below.

$$e = \sqrt{(x_h - x_s)^2 + (y_h - y_s)^2} \tag{5.5}$$

Remember that a raw evaluation of the hardware results against the ground truth centroids may lead to inaccurate conclusions, as in this case the algorithm's performance is not taken into account. If for example, due to some random condition, the software algorithm produces false results but only the respective hardware and ground truth centroids are compared, a false conclusion of bad hardware design could be extracted. Instead, it is important to evaluate how different the hardware design behaves in comparison to the software function, regardless of the general centroiding accuracy, which depends on the algorithm itself.

It is expected that both CG models will achieve the same accuracy, since they have been designed in the same way. In Table 5.1 we present the mean centroiding error between software and hardware for the CG and FGF algorithms.

Table 5.1: Mean centroiding error between hardware and software implementations on each dataset.

	Dataset		
Algorithm	3x3	5x5	Mixed
CG	0.047814	0.048162	0.047990
FGF	0.000004	0.000020	0.000005

Apparently the subpixel accuracy is consistent with the design calls that have been made. It is also clear that the single-precision floating point arithmetic that has been used in the FGF case provides a significant advantage, as the differences between software and hardware are negligible. It should be mentioned that the observed differences have to do with the use of double-precision numbers in the MATLAB function, while the hardware system adopts single-precision accuracy for lower logic complexity. On the contrary, the error in the CG algorithm is increased due to the use of the simple fixed point division. As it is explained in Section 4.2.1.3, the 4 fractional bits only ensure that the error will be lower than 0.1. If more fractional bits were used, the respective errors in Table 5.1 would be decreased.

In order to obtain a better insight into the algorithms, in Tables 5.2, 5.3 we provide the centroiding results as produced by the software and hardware implementations for four random clusters.

Table 5.2: Detected centroids by the CG software and hardware implementations.

Ground Truth		**CG SW**		**CG HW**	
X	Y	X	Y	X	Y
210.593130	899.672780	210.846683	899.876934	210.8125	899.875
945.137136	729.932595	945.352640	729.947333	945.3125	729.9375
135.510762	829.973985	135.816945	829.990032	135.8125	829.937
753.193434	791.639461	753.391853	791.720189	753.375	791.6875

It is obvious that the FGF algorithm dominates as far as accuracy is concerned. Furthermore it seems that the CG algorithm doesn't meet the typical error requirement in software, as it displays centroiding errors higher than 0.1 pixels on the simulated datasets. Consequently even if more fractional bits were used in the CG design, the accuracy would still be insufficient regarding the

Table 5.3: Detected centroids by the FGF software and hardware implementations.

Ground Truth		**FGF SW**		**FGF HW**	
X	Y	X	Y	X	Y
210.593130	899.672780	210.592542	899.673227	210.592539	899.673221
945.137136	729.932595	945.137295	729.932511	945.137295	729.932514
135.510762	829.973985	135.511087	829.973674	135.511093	829.973673
753.193434	791.639461	753.193277	791.639544	753.193274	791.639544

typical error. For this reason, we suggest that a better selection of the number of fractional bits should be done when a real star dataset is available and the actual performance of the CG algorithm can be evaluated.

5.3 Performance Experiment

In the most important experiment presented in this thesis, we measure the consumption time of each algorithm on all datasets, in order to establish the actual benefits of implementing a centroiding process on the FPGA. Initially, a comparison between the different hardware implementations should be provided, regarding the metrics of latency, throughput and achieved operation frequency. The results for each FPGA design are displayed in Tables 5.4, 5.5.

Table 5.4: Comparison between FPGA implementations (1).

Model	**Latency**	**Initiation Interval**	**Critical Path** (ns)
CG VHDL	$N^2 + 42$	$N^2 + 3$	3.141
CG HLS	$N^2 + 37$	$N^2 + 6$	3.942
FGF	$8N^2 + 643$	$8N^2 + 37$	5.921

Table 5.5: Comparison between FPGA implementations (2).

Model	**Max Frequency** (MHz)	**Max Throughput (million clusters/s)**	**Acceleration** ($\times$)
CG VHDL	320	18	200
CG HLS	250	12	160
FGF	170	1	25

The latency and throughput are expressed as a function of the number of pixels N in order to obtain a better view of the differences between the designs. It should be mentioned, that the provided latency for the VHDL model is an estimate and can't be accurately measured due to the usage of a buffer, as explained in Section 4.2.1.4. Furthermore, the two CG models demonstrate some small differences in latency and throughput. The C++ implementation is characterized by a lower latency due to the powerful *Dataflow* directive, which automatically constructs the communication channels between the different functions, in combination with the fast divider core which is able to receive one sample per cycle. By contrast the use of the IP Divider, which inputs data at a slower rate results in increased latency. However the VHDL block is able to receive new data sooner and offers better throughput. As explained in Section 4.2.2.5 this is due to the fact, that the C++ loop-based functions aren't able to re-execute before their previous run is complete. However, when we model a design in the RTL we are able to describe the desired behavior more accurately and create more "flexible" blocks.

Perhaps the most important difference between the two models is that the HLS implementation isn't able to achieve the operation frequency of the VHDL model. While the latter can execute at almost 320 MHz, the HLS block operates at 250 MHz, even though they model actually the same algorithm. The generated reports place the Critical Path within the Unsigned Division core, which is implemented with LUTs and is automatically inferred by the HLS compiler. We have highlighted that the tool doesn't allow the designer to intervene in this core. It is therefore concluded, that the performance is limited by the HLS tool.

Apparently, the FGF implementation is quite slower than the CG designs which is a result of the significantly higher complexity of this algorithm. According to the extensive analysis provided in Section 4.3, the throughput of the design is limited by the floating point subtraction core which calculates the constant vector **b** and is able to receive a new sample every 8 cycles. Moreover, the data depedencies within the Linear Problem Solver and especially within the substitution method, combined with the use of slow floating point cores, cause large delays. Ultimately the overall complexity of the algorithm which requires a great use of resources, results in a relatively high critical path. According to the generated reports, the critical path is created within the floating point subtraction core of the Cholesky Block. It should be highlighted also that the net delay constitutes the 82% of the total critical path which explains that the operation frequency is limited during implementation, due to the high resource utilization, as it will be discussed in the next section. Consequently, the placer and router aren't able to find an optimal solution to the respective placing and routing problems, in order to implement a faster circuit. During development an exhaustive space exploration was conducted, regarding the available Logic Synthesis and Implementation strategies provided by the Vivado tool, but no combination could provide better results. Therefore it is estimated, that the best possible C++ implementation of the FGF algorithm has been achieved.

After an in-depth description of the hardware-level differences between the various implementations, in Table 5.6 the total execution times of the algorithms on the provided datasets are displayed.

Table 5.6: Total consumption time of each implementation in seconds.

Dataset	CG SW	FGF SW	CG VHDL	CG HLS	FGF HW
3x3	0.126010	0.217139	0.000377	0.000591	0.006457
5x5	0.125483	0.269169	0.000880	0.001221	0.014036
Mixed	0.151671	0.234976	0.000628	0.000906	0.010237

As it is expected, the CG algorithm is faster on both software and hardware. On software it seems to be almost 2 times faster than the FGF algorithm which confirms the results from [7]. On hardware the VHDL implementation is slightly faster than the one produced with HLS. The timing results are summarized in Table 5.5, which also displays the overall acceleration that each proposed design achieves.

An acceleration of 2 orders of magnitude for the CG algorithm is observed, while the FGF algorithm is executed approximately 25 times faster on hardware. Apparently, the acceleration on this case is limited by the complexity of the algorithm which is reflected on the increased demands for logic resources. As explained, the FGF implementation is mainly characterized by low throughput and increased critical path. The floating point subtraction core within the input loop imposes a constraint on the initiation interval, while the critical path is created within the same type of core in the Cholesky function. Considering also the fact that this path is by 80% composed of net delay, it is clear that the algorithm's underlying high complexity, which is expressed with the use of floating point cores, results in congestion of resources in the FPGA and ultimately defines the amount of achieved acceleration. Nevertheless it can be claimed that the proposed novel design provides a significant speed advantage, while the high accuracy is maintained, making it suitable for real-time applications.

5.4 Resource Utilization Evaluation

In Table 5.7 we provide insight regarding the resource utilization on the FPGA for each implementation. Within the brackets, we give the total number of each type of resources that are available on the ZYNQ-7020 SoC.

Table 5.7: FPGA resource utilization.

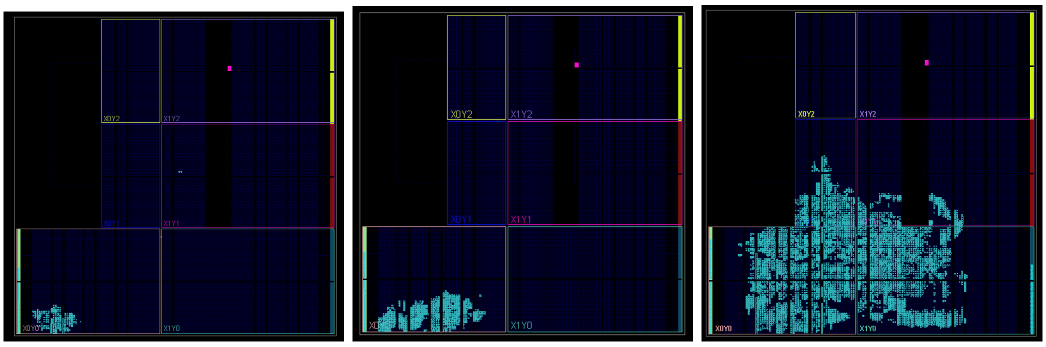
Model	**LUT** (53200)	**FF** (106400)	**DSP** (220)	**BRAM** (140)
CG VHDL	255	466	3	2
CG HLS	1362	1466	2	0
FGF	8661	13761	55	0

It can be seen, that the complexity of each algorithm reflects directly on its resources demands. It is interesting that the C++ design of the CG algorithm requires quite more FPGA primitives compared to the VHDL model. One explanation for this result is that the latter uses 2 Block RAMs to construct the FIFO memories, while in the HLS approach, according to the C Synthesis reports the compiler infers Shift Register LUTs. Considering the increased area and increased critical path that the C++ model demonstrates, it is believed that the HLS tool doesn't provide an optimal solution.

Furthermore, it is obvious that the FGF implementation requires significantly more logic resources on the FPGA. It should be mentioned that based on the HLS Synthesis reports and the designed C++ code, an amount of 50 DSPs was expected, while the calculation of both the coefficient matrix and the constant vector require in total 31 DSPs and the rest of the floating point operations use 19 cores. However, it seems that the Logic Synthesizer inferred 5 additional DSPs. In total the Input Function, which performs concurrently a large number of operations on the input samples, seems to use the most of the resources.

Lastly, it is important to have a look at the actual placement of each model on the FPGA, as it is provided by Vivado. In Figure 5.1 the consumed area in each case is displayed.

The expanded area which is required by the CG C++ circuit is another indication of the differences between the two programming models. The VHDL description gives the optimal implementation on the FPGA but comes with a cost in development time. Even though the FGF implementation doesn't use a very large percentage of the available resources, the placement on the FPGA seems to be quite wide which makes routing delay the determining factor of the circuit's critical path. The visual inspection of the FPGA area can corroborate the claims regarding the reasons behind limited acceleration in the FGF case. In total we believe that the HLS approach



(a) CG VHDL **(b)** CG HLS **(c)** FGF

Figure 5.1: Overview of the placement of each design on the FPGA.

isn't able to provide better results, which perhaps could be obtained if the algorithm was described in the Register Transfer Level with VHDL.

Chapter 6

Conclusion

6.1 Summary

In order to keep up with the requirements of modern space applications, in this thesis we attempt to accelerate two star centroiding algorithms on an FPGA platform. Each implementation demonstrates different advantages and ultimately a trade-off between accuracy and efficiency is emerged and reflected on utilization of FPGA resources. The Center of Gravity algorithm due to its simplicity was accelerated by 2 orders of magnitude but the accuracy was only constrained within 0.1 pixels compared to the software model. On the contrary, the more complex Fast Gaussian Fitting algorithm was accelerated by 25 times while maintaining excellent accuracy. This FPGA implementation constitutes a novel design which can be utilized as part of the star detection process, which is vital for precise and fast attitude determination in space. Considering the fact that the star tracker's accuracy is mainly determined by the centroiding process, the proposed model assures that under the appropriate system configuration (e.g., high resolution sensor, sufficient clustering) centroids with the lowest possible error will be obtained. Simultaneously it can enable very fast update rates, since it can typically process 10,000 stars in 10 ms. However, it is unlikely that such high rates can be required by a star tracker due to the high integration time that characterize the optic sensor.

Furthermore, in this work the different FPGA programming models with VHDL and HLS (C++) have been extensively studied. We exploited the advantages of convenience and high productivity that the HLS provides in order to conduct exhaustive space exploration and testing. It wouldn't be otherwise possible to develop and validate three different models in such a short period of time. Especially the VHDL modeling of the FGF algorithm would be considerably more challenging. However the quality of results that the RTL modelling demonstrates is undisputed. Even in the case of the very simple CG algorithm, the HLS didn't achieve the optimal performance of the VHDL design, as far as maximum frequency and resource utilization are concerned. We also assume that even more performance can be extracted by the FGF algorithm if it is implemented with VHDL.

6.2 Future Work

In order to fully understand the benefits that the proposed novel design can provide, it is necessary to include it in the complete star detection pipeline. We intent to combine the proposed centroiding implementation with the clustering and matching algorithms that have been developed by a fellow colleague and Infinite Orbits respectively. Hopefully this fully custom star tracking pipeline will be used in future space missions led by Infinite Orbits.

Regarding the work presented in this thesis, it is definitely useful to simulate the implemented designs on real star images. The novel design is expected to demonstrate excellent accuracy once again. However, it is interesting to observe the behavior of the CG models since they weren't able to produce remarkable results on simulated data. Should the CG software algorithm perform better when real data are used, we suggest a more detailed exploration, regarding the number of used fractional bits, to be conducted in order to enable the very fast hardware model to achieve better accuracy.

Lastly, a possible VHDL description of the FGF algorithm could provide a better insight regarding the maximum underlying performance that can be extracted. However, as explained, the efficiency of the proposed design is sufficient and therefore it can be used in all modern star trackers, which means that even a higher acceleration with VHDL wouldn't provide actual gains.

Bibliography

[1] C. L. von Wielligh, "Fast star tracker hardware implementation and algorithm optimisations on a system-on-a-chip device." 2019.

[2] "The different frames and the keplerian elements." [Online]. Available: https://adcsforbeginners.wordpress.com/tag/earth-centred-inertial-frame/

[3] Q. Hua-Ming, L. Hao, and W. Hai-yong, "Design and verification of star-map simulation software based on ccd star tracker." *2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pp. 383–387, 2015.

[4] "How is "cross boresight" accuracy of "about boresight" accuracy defined?" [Online]. Available: https://space.stackexchange.com/questions/54034/how-is-cross-boresight-accuracy-of-about-boresight-accuracy-defined

[5] Xilinx, *Introduction to FPGA Design with Vivado High-Level Synthesis.*

[6] I. Stratakos, "Lecture notes on design and implementation of image processing algorithms on soc platforms for embedded applications." May 2017.

[7] X. Wan, G. Wang, X. Wei, J. Li, and G. Zhang, "Star centroiding based on fast gaussian fitting for star sensors." *Sensors*, vol. 18, p. 2836, 08 2018.

[8] J. N. Pelton, S. Madry, and S. Camacho-Lara, *Handbook of Satellite Applications.* Springer Publishing Company, Incorporated, 2012.

[9] O. Kodheli, A. Guidotti, and A. Vanelli-Coralli, "Integration of satellites in 5G through LEO constellations." in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[10] F. Davoli, C. Kourogiorgas, M. Marchese, A. Panagopoulos, and F. Patrone, "Small satellites and cubesats: Survey of structures, architectures, and protocols." *International Journal of Satellite Communications and Networking*, vol. 37, no. 4, pp. 343–359, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1277

[11] "The space economy report." [Online]. Available: https://digital-platform.euroconsult-ec.com/wp-content/uploads/2020/07/Space-eco-report-brochure.pdf

[12] "Emerging space investment analysis, 3rd edition." [Online]. Available: https://www.nsr.com/?research=emerging-space-investment-analysis-3rd-edition%E2%80%AF

[13] E. M. Gaposchkin, C. von Braun, and J. Sharma, "Space-based space surveillance with the space-based visible." *Journal of Guidance, Control, and Dynamics*, vol. 23, no. 1, pp. 148–152, 2000. [Online]. Available: https://doi.org/10.2514/2.4502

[14] T. P. Setterfield, "On-orbit inspection of a rotating object using a moving observer." Ph.D. dissertation, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2017.

[15] W. Fehse, *Automated Rendezvous and Docking of Spacecraft.* Cambridge university press, 2003.

[16] C. Kaiser, F. Sjöberg, J. M. Delcura, and B. Eilertsen, "An orbital life extension vehicle for servicing commercial spacecrafts in GEO." *Acta Astronautica*, vol. 63, no. 1, pp. 400–410, 2008, touching Humanity - Space for Improving Quality of Life. Selected Proceedings of the 58th International Astronautical Federation Congress, Hyderabad, India, 24-28 September 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576507003633

[17] C. Bonnal, J.-M. Ruault, and M.-C. Desjean, "Active debris removal: Recent progress and current trends." *Acta Astronautica*, vol. 85, pp. 51–60, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576512004602

[18] A. R. Eisenman, C. C. Liebe, and J. L. Joergensen, "New generation of autonomous star trackers." in *Sensors, Systems, and Next-Generation Satellites*, H. Fujisada, Ed., vol. 3221, International Society for Optics and Photonics. SPIE, 1997, pp. 524 – 535. [Online]. Available: https://doi.org/10.1117/12.298121

[19] G. Wang, W. Lv, J. Li, and X. Wei, "False star filtering for star sensor based on angular distance tracking." *IEEE Access*, vol. 7, pp. 62 401–62 411, 2019.

[20] A. Read Eisenman and C. Liebe, "The advancing state-of-the-art in second generation star trackers." in *1998 IEEE Aerospace Conference Proceedings (Cat. No.98TH8339)*, vol. 1, March 1998, pp. 111–118 vol.1.

[21] E. Fossum, "CMOS image sensors: electronic camera-on-a-chip." *IEEE Transactions on Electron Devices*, vol. 44, no. 10, pp. 1689–1698, 1997.

[22] 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.

[23] 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.

[24] 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.

[25] C. Wilson and A. George, "CSP Hybrid Space Computing." *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 215–227, 2018.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] C. Liebe, "Accuracy performance of star trackers-a tutorial." *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 38, pp. 587 – 599, 05 2002.

[32] B. Spratling and D. Mortari, "A survey on star identification algorithms." *Algorithms*, vol. 2, 03 2009.

[33] L. Kazemi, J. Enright, and T. Dzamba, "Improving star tracker centroiding performance in dynamic imaging conditions." *2015 IEEE Aerospace Conference*, pp. 1–8, 2015.

[34] V. Lappas, "Practical results on the development of a control moment gyro based attitude control system for agile small satellites." 2002.

[35] D. Michaels and J. Speed, "Ball aerospace star tracker achieves high tracking accuracy for a moving star field." in *2005 IEEE Aerospace Conference*, March 2005, pp. 1–7.

[36] B. Horn, *Robot Vision*, 01 1986.

[37] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 01 2003.

[38] Y. Liu and C. Pomalaza-raez, *Self-Landmarking for Robotics Applications*, 08 2011.

[39] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors." *Matrix*, vol. 58, 01 2006.

[40] K. R. Castleman, *Digital image processing*, ser. Prentice-Hall signal processing series. Englewood Cliffs, N.J: Prentice-Hall, 1979 - 1979.

[41] E. Hecht, *Optics*, 4th ed. Addison-Wesley, 1998.

[42] B. Saleh and M. Teich, *Fundamentals of Photonics*. John Wiley Sons, Ltd, 1991. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471213748.gloss

[43] M. P. Ramachandran, "Analytical derivation of star sensor accuracies due to centroid error." *Sādhanā*, vol. 46, no. 4, p. 205, Oct 2021. [Online]. Available: https://doi.org/10.1007/s12046-021-01713-1

[44] E. D. Aretskin-Hariton and A. J. Swank, "Star tracker performance estimate with IMU." 2015.

[45] T. Delabie, "Star tracker algorithms and a low-cost attitude determination and control system for space missions." 2016. [Online]. Available: https://lirias.kuleuven.be/retrieve/353369

[46] W. Tan, S. Qin, R. M. Myers, T. J. Morris, G. Jiang, Y. Zhao, X. Wang, L. Ma, and D. Dai, "Centroid error compensation method for a star tracker under complex dynamic conditions." *Opt. Express*, vol. 25, no. 26, pp. 33 559–33 574, Dec 2017. [Online]. Available: http://opg.optica.org/oe/abstract.cfm?URI=oe-25-26-33559

[47] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, Oct 1992.

[48] S. R. E., *Electro-Optics Handbook.* RCA Corporations, 1974.

[49] D. Brown, "Decentering distortion of lenses." 1966.

[50] M. J. Jacobs, "A low cost, high precision star sensor." 1995.

[51] C. Ricolfe-Viala and A.-J. Sánchez-Salmerón, "Lens distortion models evaluation." *Applied optics*, vol. 49, pp. 5914–28, 10 2010.

[52] F. Zhou, Y. Cui, H. Gao, and Y. Wang, "Line-based camera calibration with lens distortion correction from a single image." *Optics and Lasers in Engineering*, vol. 51, no. 12, pp. 1332–1343, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0143816613001619

[53] J. de Villiers, F. Leuschner, and R. Geldenhuys, "Centi-pixel accurate real-time inverse distortion correction." *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 7266, 11 2008.

[54] M. D. C. M. Morris R. Mano, *Digital Design, 5th Edition.* Pearson, 2013.

[55] W. H. Steyn, M. J. Jacobs, and P. J. Oosthuizen, "A high performance star sensor system for full attitude determination on a microsatellite." 2004.

[56] B. C. Greyling, "A charge coupled device star sensor system for a low earth orbit microsatellite." 1995.

[57] X. Zhu, F. Wu, and Q. Xu, "A fast star image extraction algorithm for autonomous star sensors." in *Optoelectronic Imaging and Multimedia Technology II*, T. Shimura, G. Xu, L. Tao, and J. Zheng, Eds., vol. 8558, International Society for Optics and Photonics. SPIE, 2012, pp. 443 – 451. [Online]. Available: https://doi.org/10.1117/12.999641

[58] M. W. Knutson, "Fast star tracker centroid algorithm for high performance cubesat with air bearing validation." 2012.

[59] R. C. Stone, "A comparison of digital centering algorithms." *The Astronomical Journal*, vol. 97, p. 1227, 1989.

[60] D. A. Giancarlo Rufino, "Enhancement of the centroiding algorithm for star tracker measure refinement." *Science Direct*, p. 13, April 2001.

[61] M. Shortis, T. A. Clarke, and T. Short, “Comparison of some techniques for the subpixel location of discrete target images.” in *Other Conferences*, 1994.

[62] V. Akondi, M. B. Roopashree, and R. P. Budihala, “Improved iteratively weighted centroiding for accurate spot detection in laser guide star based Shack Hartmann sensor.” in *LASE*, 2010.

[63] T. Delabie, J. D. Schutter, and B. K. Vandenbussche, “An accurate and efficient gaussian fit centroiding algorithm for star trackers.” *The Journal of the Astronautical Sciences*, vol. 61, pp. 60–84, 2013.

[64] H. Wang, E. Xu, Z. Li, L. Jingjin, and T. Qin, “Gaussian analytic centroiding method of star image of star tracker.” *Advances in Space Research*, vol. 56, 09 2015.

[65] B. R. Flewelling and D. Mortari, “Information theoretic weighting for robust star centroiding.” *The Journal of the Astronautical Sciences*, vol. 58, no. 2, pp. 241–259, Apr 2011. [Online]. Available: https://doi.org/10.1007/BF03321167

[66] A. O. Erlank, “Development of cubestar : a cubesat-compatible star tracker.” 2013.

[67] M. Kolomenkin, S. Pollak, I. Shimshoni, and M. Lindenbaum, “Geometric voting algorithm for star trackers.” *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 44, pp. 441 – 456, 05 2008.

[68] B. Wang, H. Wang, and Z. Jin, “An efficient and robust star identification algorithm based on neural networks.” *Sensors (Basel, Switzerland)*, vol. 21, no. 22, p. 7686, Nov 2021, 34833762[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/34833762

[69] B. Spratling and D. Mortari, “A survey on star identification algorithms.” *Algorithms*, vol. 2, 03 2009.

[70] C. Padgett, K. Kreutz-Delgado, and S. Udomkesmalee, “Evaluation of star identification techniques.” *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 2, pp. 259–267, 1997. [Online]. Available: https://doi.org/10.2514/2.4061

[71] M. Lindh, “Development and implementation of star tracker electronics.” 2014.

[72] F. Zhou, J. Zhao, T. Ye, and L. Chen, “Fast star centroid extraction algorithm with sub-pixel accuracy based on fpga.” *Journal of Real-Time Image Processing*, vol. 12, no. 3, pp. 613–622, Oct 2016. [Online]. Available: https://doi.org/10.1007/s11554-014-0408-z

[73] X. Wang, X. Wei, Q. Fan, J. Li, and G. Wang, “Hardware implementation of fast and robust star centroid extraction with low resource cost.” *IEEE Sensors Journal*, vol. 15, no. 9, pp. 4857–4865, Sep. 2015.

[74] F. Kong, M. C. Polo, and A. Lambert, “On-sky results and performance of low latency centroiding algorithms for adaptive optics implemented in FPGA.” in *Unconventional and Indirect Imaging, Image Reconstruction, and Wavefront Sensing 2018*, J. J. Dolne and P. J. Bones, Eds., vol. 10772, International Society for Optics and Photonics. SPIE, 2018, pp. 223 – 232. [Online]. Available: https://doi.org/10.1117/12.2320084

[75] X. Wei, J. Xu, J. Li, J. Yan, and G. Zhang, “S-curve centroiding error correction for star sensor.” *Acta Astronautica*, vol. 99, p. 231–241, 06 2014.