IMPLEMENTATION OF AN AUTONOMOUS STAR RECOGNITION ALGORITHM USING HARDWARE-SOFTWARE CO-PROCESSING APPROACH

DANG LE DANG KHOA

G1500337L

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

2018

IMPLEMENTATION OF AN AUTONOMOUS STAR RECOGNITION ALGORITHM USING HARDWARE-SOFTWARE CO-PROCESSING APPROACH

by

DANG LE DANG KHOA

Supervisor: Asst. Prof. Chen Shoushun

Co-Supervisor: Assoc. Prof. Goh Wang Ling

A thesis submitted to the
Nanyang Technological University
in partial fulfilment of the
requirements for the degree of
Master of Engineering

School of Electrical and Electronics Engineering
Nanyang Technological University

2018

Abstract

Implementation of An Autonomous Star Recognition Algorithm using Hardware-Software Co-Processing Approach

By Dang Le Dang Khoa

There are various types of Attitude Determination sensors such as sun sensors, magnetometer, RF beacon but only Star Trackers can achieve the accuracy to arc seconds. A Star Tracker is an embedded system mounted on a spacecraft comprised of an Image sensor and a Computer. It helps determine the attitude of the satellite based on an Autonomous Star Recognition Algorithm. The Star Sensor would take an image of stars at its current position, and then the star pattern recognition algorithms would extract features to construct a pattern from the images. This pattern then is compared with a prebuilt Star Pattern Database (SPD) to return the Star Identity of a star in the Image. This star identity is an important part to determine the attitude of the satellite.

To implement the Algorithm on a specific hardware, the computing system must be appropriately chosen. Programmable System-on-chip (PSoC) is a technology that replaces the traditional ASIC (Application Specific Integrated Circuit) by an FPGA (known as Programmable Logics) combined with an Embedded Processor (known as the Processing system), integrated memories, a variety of peripherals to form an embedded computing system. The Programmable Logic is ideal for implementing high-speed logic, arithmetic and accelerating subsystems while the Processing System supports software routines and Operating systems. Based on this property, an algorithm can be partitioned into submodules to be co-processed by the hardware-software combination.

The goal of this research is to partition and profile a Star Recognition Algorithm then implement these modules on both the Processing System (Software) and the Programmable Logics (Hardware) to analyze the Algorithm implementation regarding Performance, Area of Implementation, Power Consumption.

Keywords: Star tracker, Star pattern recognition, Star recognition, Algorithms, Attitude determination, Embedded System, Programmable System on Chip (PSoC).

Acknowledgements

There are many people I want to thank for helping me in completing this work. Foremost, I would express my sincerest gratitude to my supervisor, Asst. Prof. Chen Shoushun for supporting and guiding me through my work and study with his knowledge and patience. I am very grateful for his insights and instructions in the hardware design for FPGA.

Many thanks to all my labmates of Satellite Research Center II, especially MEHTA Deval Samirbhai and Chin Shi Tong for the first steps of my research and the help whenever I am in troubles and doubt.

I would also like to thank Mrs. Janet Tan and Mrs. Pamela Ng, the technicians of Satellite Research Center II for helping me with all the lab facilities instructions also, setting up the equipment and providing resources whenever I need.

Finally, I sincerely thankful to my parents, my brother and all of my friends for their love and support throughout my study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview and Motivations

Artificial satellites orbiting the earth are a critical component of modern society. They are used in areas such as communication, positioning, imaging and weather forecasting. Besides the ordinary purposes of satellites like Earth observation, communication, weather forecasting[5], we also use satellites for research purposes. An example of this is the International Space Station (ISS) and the Hubble telescope.

Satellites operate in different orbits. Some popular orbit classes are Low Earth Orbit (LEO), Medium Earth Orbit (MEO) and High Earth Orbit (HEO)[6]. Low Earth Orbit is the area below 2000km, High Earth Orbit is the orbit above 35,500 km, and Medium Earth Orbit is the area between Low Earth Orbit and High Earth Orbit. Satellites are launched to their orbit by a self-propelled rocket as a piggyback payload.

Nanosatellite is a small artificial satellite with a wet mass(the total mass of the hardware plus fuel and oxidizer) between 1 10 kg. Nanosatellites are cheap but capable of performing commercial missions and has led researchers and scientists to explore the new way to use the Nanosatellites in LEO[7]. Their designs are only comprised of low-cost components. Due to their cost-effective, fast development cycle and size convenience, Nanosatellites are becoming popular in educational and scientific development. Nanosatellites can be quickly built in a small university project.

A satellite needs an attitude determination system for navigation and acknowledging orientations. Its attitude needs to be calculated and predicted continuously during its operation on the orbit. By observing the celestial objects and properties around the satellite, the satellite can autonomously estimate the orientations. Some of the favorite reference objects are the Sun, the Earth magnetic field, the celestial sphere, and the stars. For Nanosatellites, the attitude determination system must be concise but operate effectively.

1.1.1 Introduction to spacecraft attitude determination

Based on the celestial reference objects, many attitude determination sensors have been developed. Some of the sensors are the sun sensor, magnetometer, gyroscope, Earth horizon sensor[8]. The sun sensor is the most common sensor to be mounted on Nanosatellites due to its low cost, low power consumption, and lightweight. The sun sensor is also integrated with an Inertial Measurement Unit (IMU) which is capable of measuring the magnetic field, rotational speed and acceleration or a GPS to provide satellites position and velocity vector. The only disadvantage of the sun sensor is that its accuracy can only reach 1 arcminute resolution while the attitude determination system need a higher efficiency up to arc seconds resolution[9].

1.1.2 Star Tracker for attitude determination approach

Another way of attitude determination approach is using star trackers. Star Tracker is an optical-electronics subsystem comprises of a CCD or CMOS Image sensor attached to an optical lens and a computing system, usually a microprocessor integrated into an ASIC board. The image sensor will firstly take an image of multiple star clusters at a specific fixed orientation. After that, the image is processed through the computing system to extract a particular feature of the model depending on a predefined algorithm. Then, the specific feature is compared with a prebuilt pattern or feature database stored in the memory. The output of the algorithm is often a star ID, which has a specific attitude, this attitude is again used to determine the attitude(orientation) of the spacecraft.

Figure 1.1: Star tracker hardwares

Figure 1.2: Commercial star trackers

The star tracker is a preferred attitude determination approach because of its high accuracy compared to other sensors[10]. However, the star tracker approach is high cost and more power consumption than the traditional approach by sun sensor and IMU. To achieve the best accuracy, the star tracker depends on the many factors such as the quality of the image sensor, the quality of the optical lens, and most importantly the processing time of the computing subsystem, and the runtime of the star tracking Algorithm.

Table 1.1 depicts the accuracy comparison between star reference object and other celestial objects for attitude determination[11].

Table 1.1: Attitude determination accuracy comparison

Reference Object	Potential Accuracy
Stars	1 arcsecond
Sun	1 arcminute
Earth(Horizon)	6 arcminutes
RF beacon	1 arcminute
Magnetometer	30 arcminutes

1.2 Objectives

The essential goals of this research project are as followed.

- To analyze several star tracking algorithms especially in Lost in space (LIS) mode, the state that the spacecraft first time entering the orbit without any knowledge about its prior orientations compared to Tracking mode which is the state that the star tracker has been operating for a while, and its current orientations can be computed based on the prior information.
- To implement and evaluate the performance of a star tracking algorithm on specific hardware using hardware and software co-processing approach.
- To benchmark and optimize the algorithm in terms of the power consumption and the area of transistor implementation.

1.3 Major contributions

The following are the significant contributions of this research project.

- Investigates and Analyses the star tracking algorithms on the advantages and weaknesses of traditional methods.
- Reviews and examines in details the proposed algorithm by precedent research.
- Implements the proposed method in term of hardware and software co-processing approach on an embedded system for star tracker.

1.4 Organization of the thesis

The rest of the thesis is organized as followed:

- Chapter 2: Introduces the basis of star tracking operation, reviews some of the state of the art star tracking algorithms and the chosen algorithm to be implemented by the software-hardware co-processing approach.
- Chapter 3: Introduces and analyzes a proposed algorithm to be implemented on the hardware.

- Chapter 4: Discusses the result of the implementation regarding runtime performance and hardware resources.

Chapter 2

Literature preview

2.1 Star Tracker Operations

The star tracking algorithm operates mainly in 2 different modes: the Lost in Space (LIS) and Tracking modes:

- LIS mode: the initial attitude is unknown, the star tracker must determine it which is the most critical part of the star tracking operation. LIS mode is operated at the time when the power system on or resets after an attitude determination attempt from tracking mode failed[12].

- Tracking mode: after the initial attitude is recognized. The star tracker is switched to the Tracking mode. With the initial is acknowledged, the computing can easily predict and track the attitude based on the previous information.

On this research topic, we will only focus on the LIS mode operation. The LIS mode operation of a star tracker is generalize in Figure 2.1 included in 3 main algorithms:

- Centroiding Algorithm: Perform an image preprocessing, the input is a star image captured from the image sensor processed to an output which are the coordinates of all star clusters presented in the image.

- Star Recognition Algorithm: The main part of LIS mode. The coordinates of the stars will be used to calculate a feature vector depending on a specific Star Algorithm. A Star Pattern Database (SPD) is also prebuilt from the SAO catalog (Smithsonian Astrophysical Observatory Star Catalog) based on

the specific pattern of the Star Algorithm and stored into the memory of the processing system for comparison and pattern recognition[13]. Then, the feature vector will be looked up and matched to an entry in the SPD. The entry will return a star ID which is an identification of a star presenting in the star image.

- Attitude Determination stage: The final stage of star tracking algorithm operated in LIS mode. The Star ID will be looked up again in the SAO catalog to extract the information about its attitude. Then a group of star attitude is still used to calculate the attitude of the satellite. Some of the well-known Attitude Determination methods are the QUEST and TRIAD.

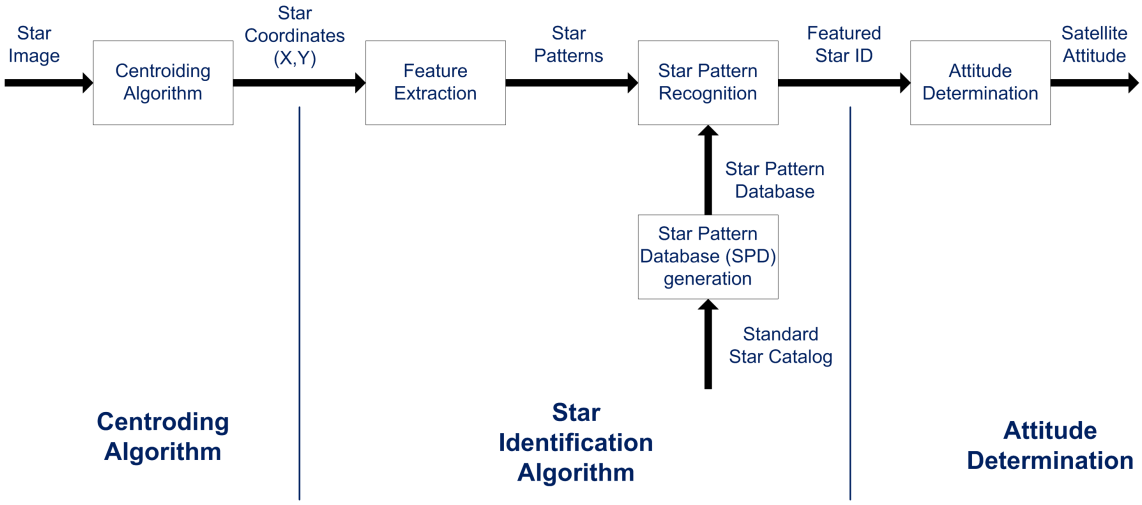


Figure 2.1: Stages of star tracking algorithm

This research topic will focus on the Centroiding and Star Identification Algorithm to be implemented on a computing system.

2.2 Centroiding Algorithm

The first step is to acquire the image which is accomplished by the image sensor. The image acquired has to be processed for estimating the coordinate of the stars in the image plane. This is completed by applying the centroiding process on the image. The input of this step is a star image with multiple star clusters, the output

is the coordinates of every stars presenting. Due to distributed energy surface, a star projection is spread over a 3x3 to 11x11 square area[14], the star projection is a group of connected pixels. Therefore, the best way to identify the star clusters is to apply the connected component algorithm.

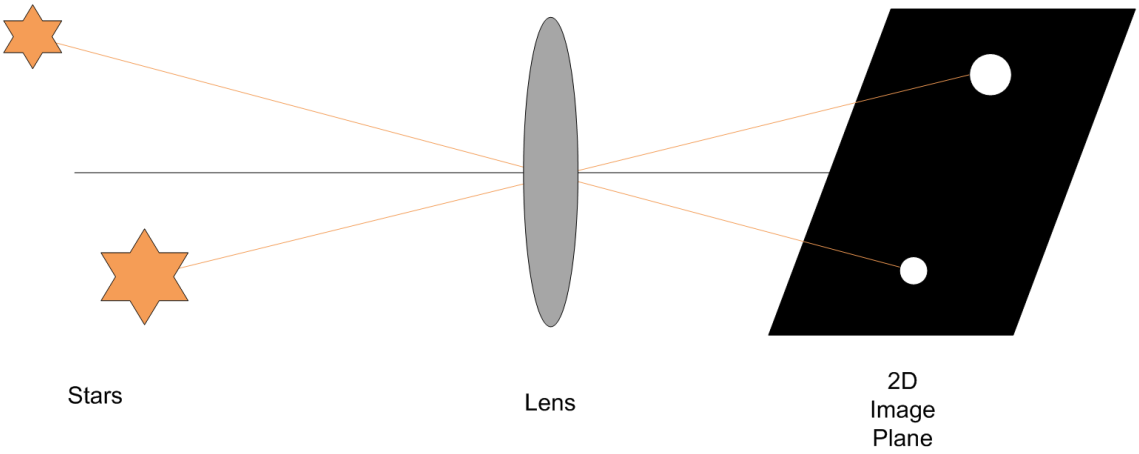


Figure 2.2: Stars projection onto a 2D image plane

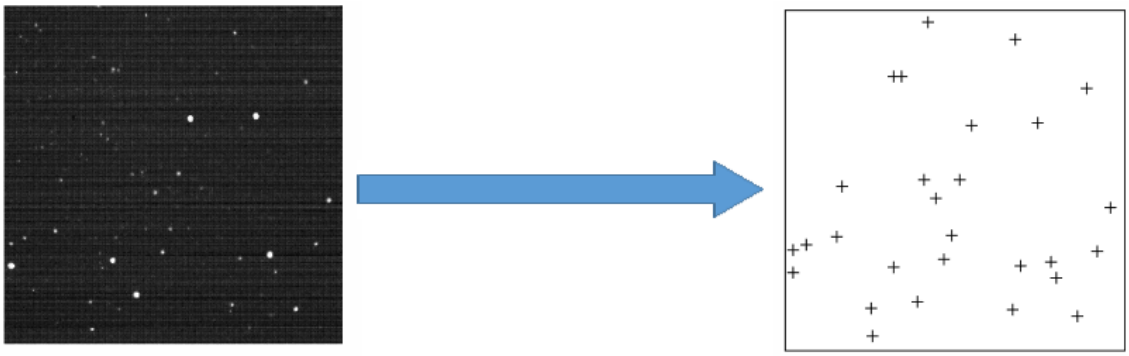


Figure 2.3: Centroid of stars in an image

Table 2.1.

A star pattern is spread over an area so we can calculate its centroid by applying the weighted pixel intensity technique. Assume that (i, j) is the horizontal and vertical centroid coordinate of a star, I_{ij} is each pixel intensity of the star pattern, i and j are the horizontal and vertical coordinates of each pixel of the star pattern.

Table 2.1: A list of star centroids in an image

Star	**Coordinates(X,Y)**
Star 1	(X_1, Y_1)
Star 2	(X_1, Y_1)
...	...
Star N	(X_N, Y_N)

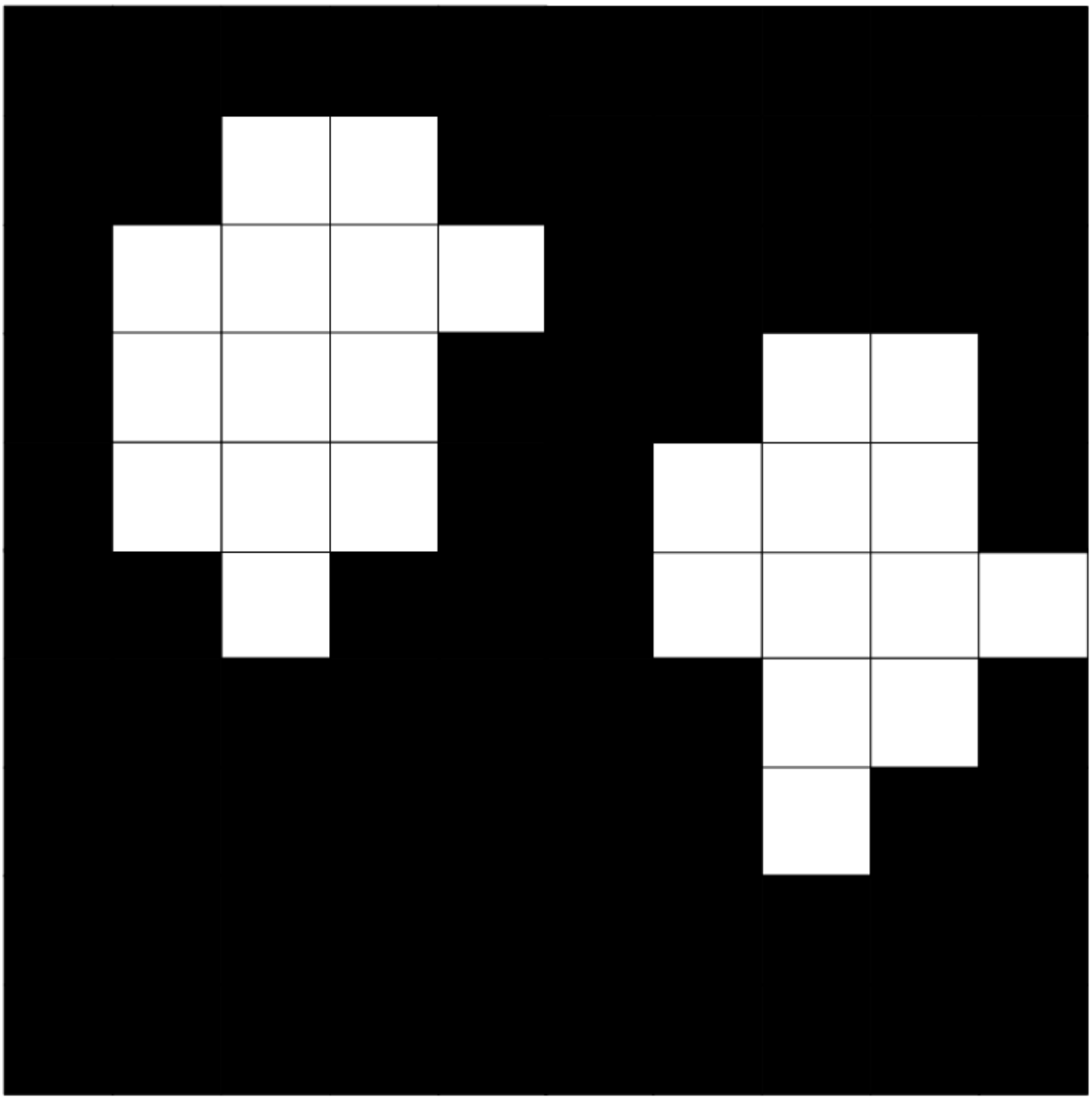


Figure 2.4: Star clusters represented in a 2D image plane

Equation 2.1

$$x = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(i * I_{ij})}{\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}}, y = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(j * I_{ij})}{\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}} \tag{2.1}$$

2.3 Liebe Star Recognition algorithm

The star pattern recognition algorithm based on the angular features formed by a star triplet was reported in[11, 15–18]. The idea of Liebe Algorithm is to utilize 3 nearest-to-the centre stars. The closest-to-the-centre star is chosen as the reference star which will then return the star ID for Attitude Determination stage. Then the next 2 stars are chosen from the neighboring stars of the reference star which have the minimum and the second minimum distance to it.

The star triplet will form 2 angular distance θ_1, θ_2 and 1 angular angle θ which are defined by Equation 2.2

$$\begin{aligned} \theta_1 &= \cos(V_A, V_B) \\ \theta_2 &= \cos(V_A, V_C) \\ \theta &= \cos(V_{AB}, V_{AC}) \end{aligned} \tag{2.2}$$

In which A is the reference star. B and C are the 2 stars which are closest to A.

The feature vector of the Liebe star recognition algorithm is defined as Equation 2.3.

The star triplet will form 2 angular distance θ_1, θ_2 and 1 angular angle θ which are defined by Equation 2.3

$$f = \{\theta, \theta_1, \theta_2\} \tag{2.3}$$

Table 2.2.
The most advantage of Liebe star recognition algorithm is that it is straightforward in implementation. It requires very few computations in the process.

The weakness of this algorithm is the weak robustness due to the limitation of star utilizing. It only employs the three nearest stars to the centre and discards other

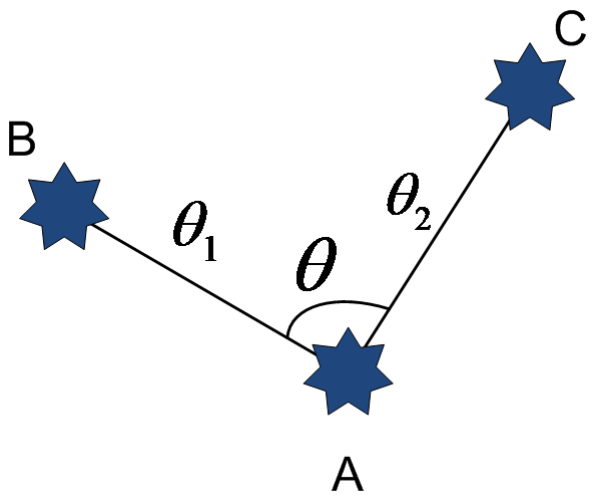


Figure 2.5: Liebe star pattern

Table 2.2: An example of Liebe star pattern database

θ_1	θ_2	θ
54	80	32
54	80	33
54	81	32
54	81	33
54	82	32
54	82	33
55	80	32
55	80	33
55	81	32
55	81	33
55	82	32
55	82	33
56	80	32
56	80	33
56	81	32
56	81	33
56	82	32
56	82	33

stars which will then lead to a mismatch if the 2 predefined nearest neighboring star does not appear in the star image. Besides, if the reference star appears in the final entry of the Star Pattern Database.

2.4 Pyramid star recognition algorithm

To overcome the mismatches caused by lack of information employed by Liebe algorithm, the pyramid star recognition algorithm is proposed in[19–23]. Its searching pattern is a combination of angular distances of 4 stars in the star image. In Figure 2.6, the 4 stars: A,B,C,D is chosen to build the pattern. The reference star which is star A in the example is the closest star to the centre of the image. Next, the 3 neighboring stars around star A are selected to form a feature vector from the 4-star angular distances.

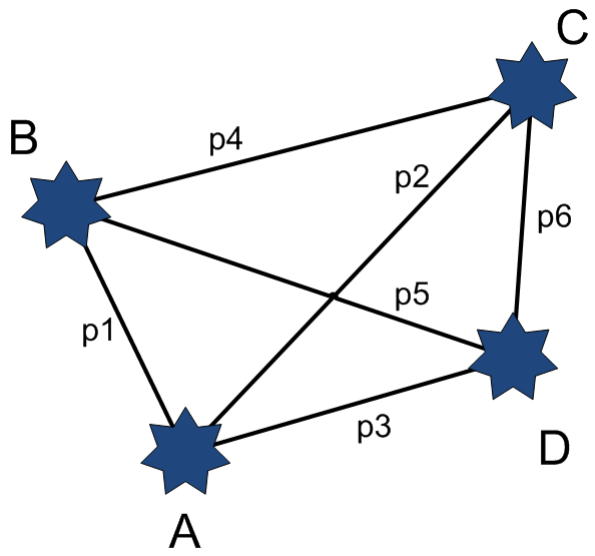


Figure 2.6: Pyramid star pattern

$$f = \{p_1, p_2, p_3, p_4, p_5, p_6\} \tag{2.4}$$

Table 2.3.

One of the advantages of the pyramid algorithm is that it employs 4 stars to build the feature vector compared to only 3 stars of the Liebe algorithm. Compared to

Table 2.3: A Pyramid star pattern example

Angular distances	Star 1	Star 2
p_1	A	B
p_2	A	C
p_3	A	D
p_4	B	C
p_5	B	D
p_6	C	D

Liebes, this method is more robust since it makes the star recognition process more accurate as six features are utilized for identification purpose compared to only three features used by the Liebe algorithm[24].

However, in a tradeoff, more memory space is needed since it requires to store at least 6 pairs of each starID entry this will lead to increase the memory required to store the star pattern database and the processing time will be increased due to the increase in pattern storage and comparisons. The time complexity is also a significant consideration. To solve this problem, the k-vector search has been introduced in[21, 23]. The time complexity has been reduced significantly which is equivalent to the Liebe searching method.

Moreover, the pyramid star recognition algorithm has not yet solved the mismatch problem caused by missing stars. In the case of one or more neighboring stars of the reference, the star is missing the algorithm will fail and do not produce a correct result.

2.5 Geometric Voting algorithm

To overcome the weakness of Liebe and Pyramid star recognition algorithm, the Geometric Voting is proposed based on voting of radial distances[25]. The Geometric Voting method considers all the nearby stars of the reference star appeared in the star image. Thanks to all stars in the image are employed, the Pyramid algorithm

becomes more robust than the first two algorithms.

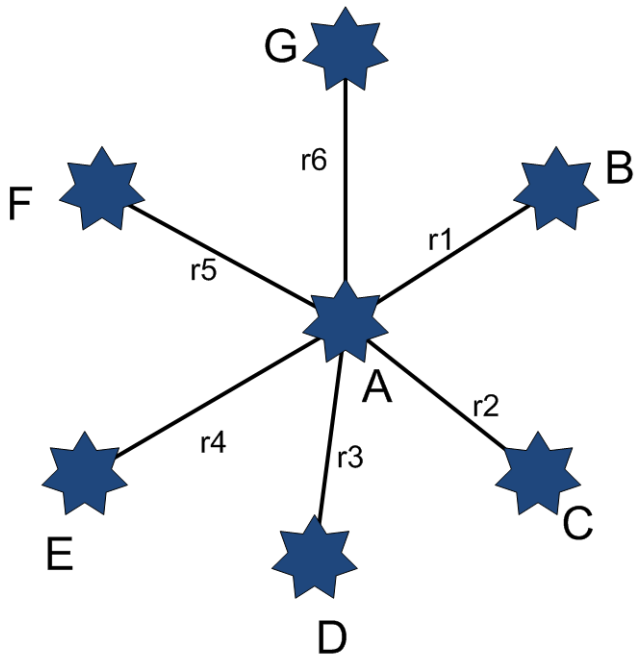


Figure 2.7: Geometric Voting star pattern

The feature vector is now not a fixed number of elements as in equation 2.5.

$$f = \{r_1, r_2, ..., r_n\} \tag{2.5}$$

After capturing the image, $r_1, r_2, , r_n$ are calculated based on the distance from the closest star to the centre which is the reference star to all other stars within the FOV of the star sensor. Then all the distances from the feature vector are compared to the predefined neighboring distances of every star ID in the SPD. With each distance is matched, a voting value of the star ID is increased. After the comparing distance stage processed, the star ID which has the highest vote is recognized as the correct star ID candidate.

For the advantage of this recognition method, firstly it employs all the data presenting in the image. As a result, it reduces the mismatching issued caused by lack of data proposed by Liebe and Pyramid method. Thus, it is the most robust star recognition method compared to the first two introduced methods.

The critical drawback of this method is the enormous space complexity needed to store the Star Pattern Database since each star ID entry must store the information of all of its neighboring stars. It leads to a requirement that a large memory must be used to store the Star Pattern Database to trade off the accuracy and robustness. Thus, the processing time for returning a star ID is increasing due to the increase in data processing and comparisons through the whole database.

2.6 Star Identification Algorithm with Optimized database

In a quick review and analysis of the 3 prestigious and famous star recognition, we can come up with a conclusion that there is a trade-off between accuracy and robustness with the space complexity and processing time required. For the star tracker system, the choice of algorithm to be implemented on will depend on the scale of the satellite project. For military satellite, we favor the accuracy and robustness over the complexity of time and space. Thus, the computer system on those satellites must be fast enough to handle the complexity.

For the implementation on the nanosatellites, the computer system is a small embedded processor with limited memory and processing ability which can not handle a large number of computations. Therefore, we need an optimized algorithm that is accurate, robust and can be implemented on a concise computer system.

The chosen star tracking algorithm is conducted by the research An Autonomous Star Recognition Algorithm with Optimized Star Catalogue for Fast Search Performance M.D. Pham, K.S. Low and S.H. Chen[12, 13]. The research introduces a novel star pattern identification using an optimized database and proposes a search tree data structure for quick and advanced parallel search.

Due to the advance of the previous research have been proven, this star tracking algorithm will be chosen to be implemented on the hardware.

This star identification algorithm employs all the star appeared in the FOV like the

geometric voting algorithm. The distances from the reference star to its neighboring stars are considered, and thus, another field is added to yield the star pattern, the number of stars appeared within the FOV of the star sensor.

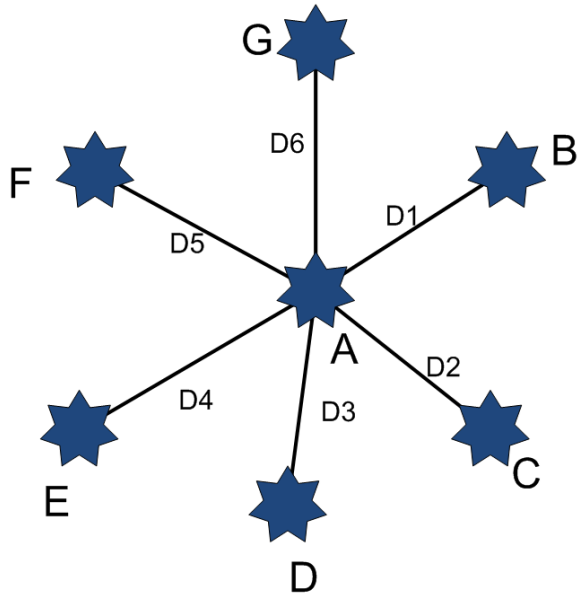


Figure 2.8: Star Identification Algorithm with Optimized database star pattern

The feature vector is described in Figure 2.8.

- N: Number of stars appeared within the FOV of the star sensor.
- D_1, D_2, D_3, D_n: Distance from the reference star to all of its neighbors.

To search and match the star pattern vector with the database, a special database structure is pre-built. The tree is built with multiple levels, the first level is N-Number of stars in the FOV, next layers will be the distances from the reference stars to all of its neighbors which are sorted ascendingly from nearest neighboring star to the farthest.

In the searching and matching stage, each parameter from the star pattern is dispatched and matched to the equivalent level. If the matching succeeds, the next layer is considered. If all layers are matched the correct star ID will be returned; otherwise, we return a statement that the star pattern is a false pattern, no stars

were matched. In this case, depend on the attitude determination system on the satellite, the star identification algorithm would be rerun or skipped to proceed the next star image.

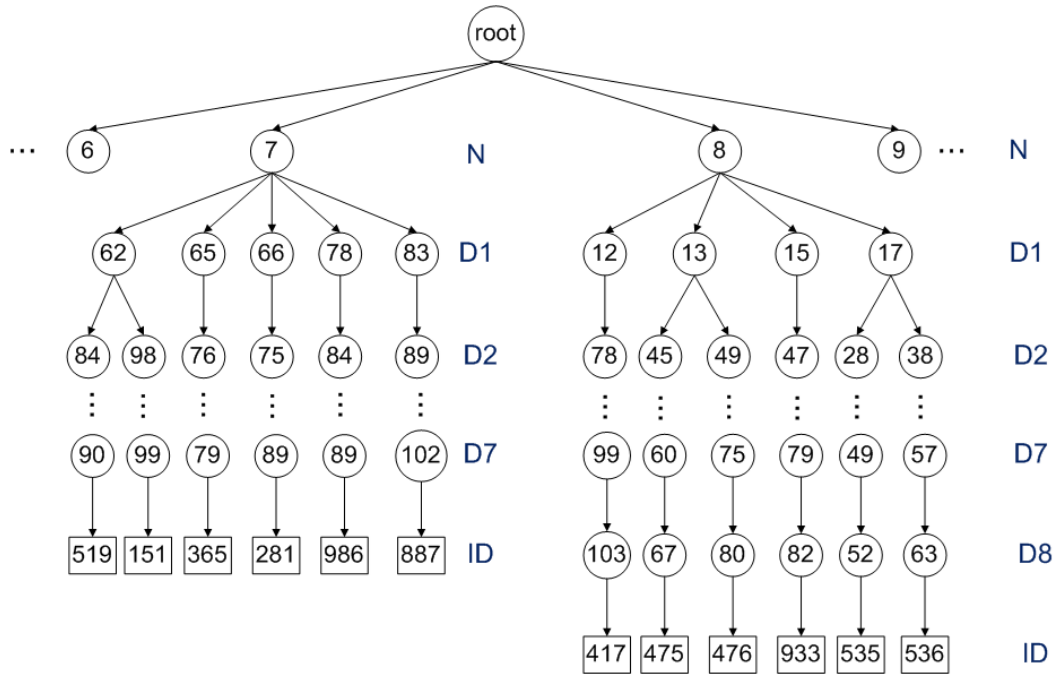


Figure 2.9: A star pattern database tree-styled structure

Chapter 3

Hardware and Software Co-processing Implementation of the Algorithm

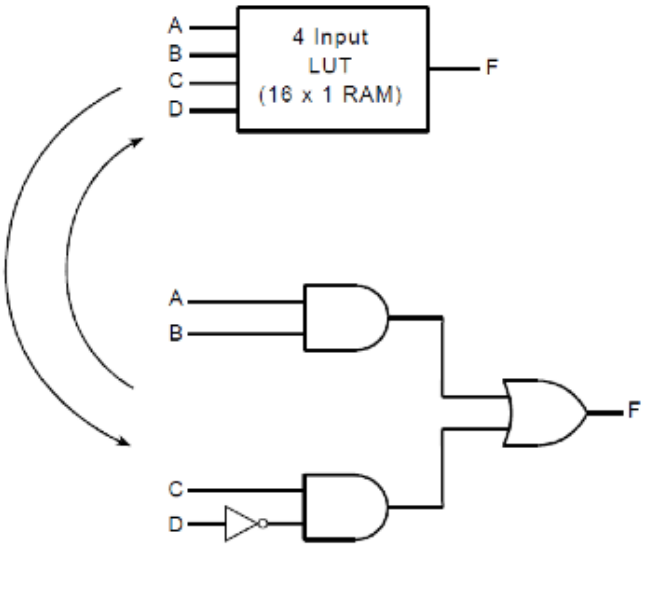
3.1 Overview

3.1.1 FPGA - Programmable Logic - IP Block

FPGA (Field Programmable Gate Array) provides a general purpose digital logic with great flexibility and utility. The fundamental logic cells of an FPGA work using look up tables (LUTs). The performance of an FPGA design depends on the area using and routing of LUTs[26].

LUTs: FPGAs were first designed to imitate Gate Arrays integrated circuits with a sea of gates that were connected by added metal layers (wire routing) to define the functionality of the device. However, instead of implementing the logic using transistor gates, the logic was in many cases implemented by memory fashioned as look up tables.

Routing: The routing in an FPGA is hierarchical. At the lowest level, we have a routing switch that is controlled by a bit of SRAM, or a FLASH bit, or a fuse. There are many routing switches in the local routing to connect various logic elements or blocks. The FPGA is made of a sea of these logic blocks. Groups of the blocks are



RAM Contents				
Address				Data
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Figure 3.1: A 4-input logic implemented by a Look up table (LUT)[2]

connected by long lines or very long lines. Common signals like clocks are routed on special routing networks called global networks.

FPGA design flow: The design begins with design entry, continues with simulation to verify the logic is implemented as expected, and then the tool will perform synthesis (also called mapping) to map the logic to the device architecture. Next the tool will create the interconnection between cells by place and routing (or fitting) the design. Another simulation after the fitting is done is a good practice. If this looks good, the next step to use the tool to create a programming file, which is then downloaded into the FPGA device for testing[2].

Programmable Logic: In Xilinx architecture, the FPGA part of the design is called Programmable Logic (PL). Some of the specialized components in the architecture include:

- **Configurable Logic Blocks (CLBs)** contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches.
- **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. IOBs support bidirectional data flow plus

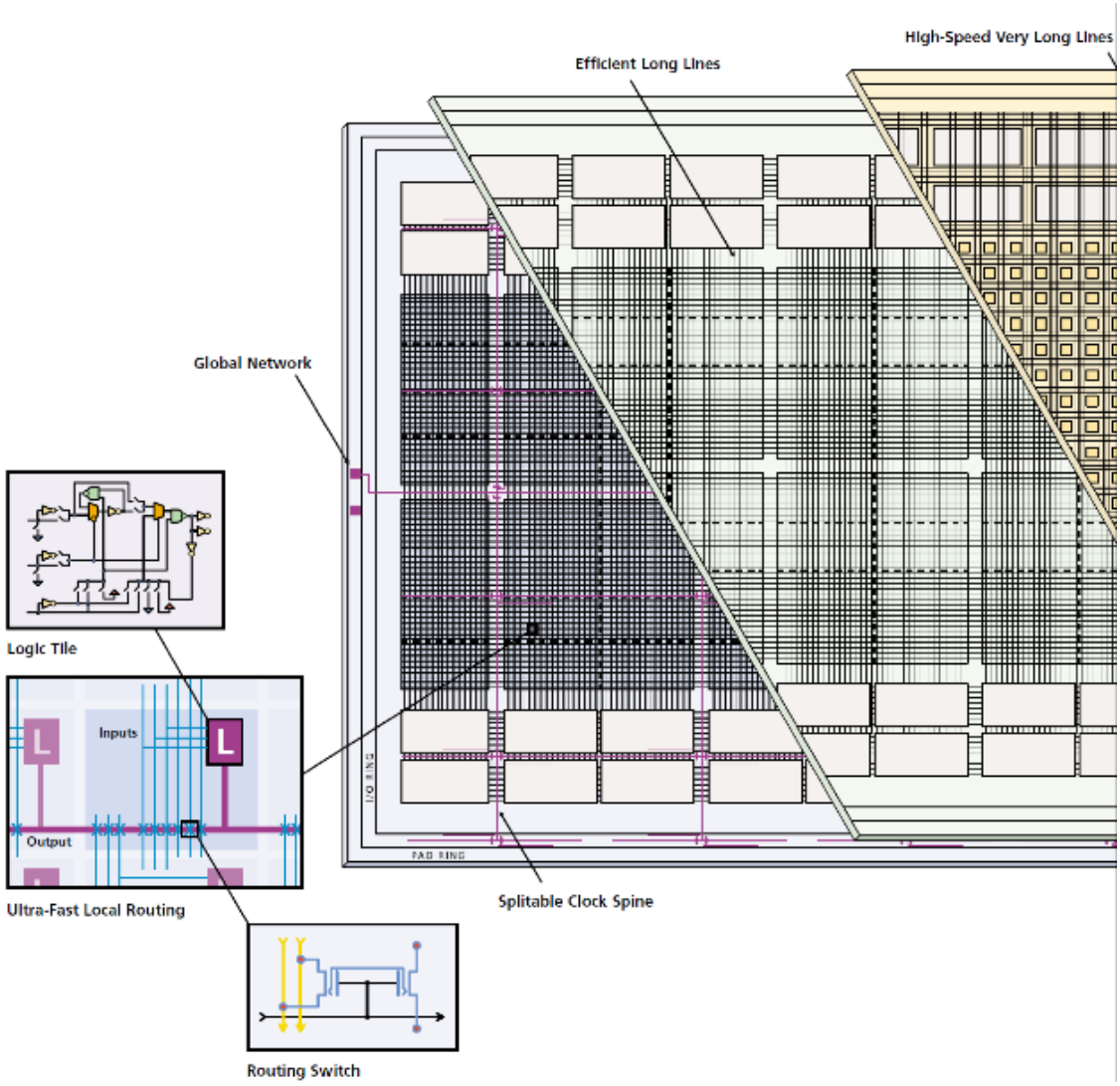


Figure 3.2: FPGA Routing Architecture[2]

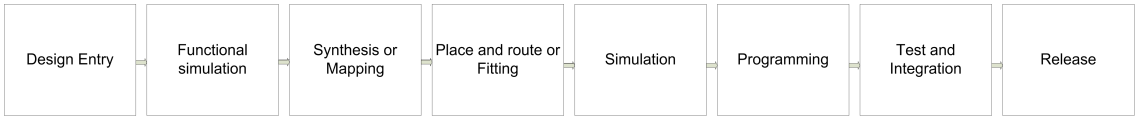


Figure 3.3: FPGA Development Process Design Flow[2]

3-state operation. They support a variety of signal standards, including several high-performance differential standards. Double Data-Rate (DDR) registers are included.

- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.
- **Digital Clock Manager (DCM) Blocks** provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

These elements are organized as shown in Figure 3.4. A dual ring of staggered IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device. The design has two DCMs in the middle of the two columns of block RAM and multipliers.

The Xilinx Programmable Logic features a rich network of traces that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing[3].

Intellectual property core, IP core, or IP block is a reusable unit of logic, cell, or integrated circuit (commonly called a "chip") layout design that is the intellectual property of one party. IP cores may be licensed to another party or can be owned and used by a single party alone. The term is derived from the licensing of the patent and/or source code copyright that exist in the design. IP cores can be used as building blocks within application-specific integrated circuit (ASIC) designs or field-programmable gate array (FPGA) logic designs[27].

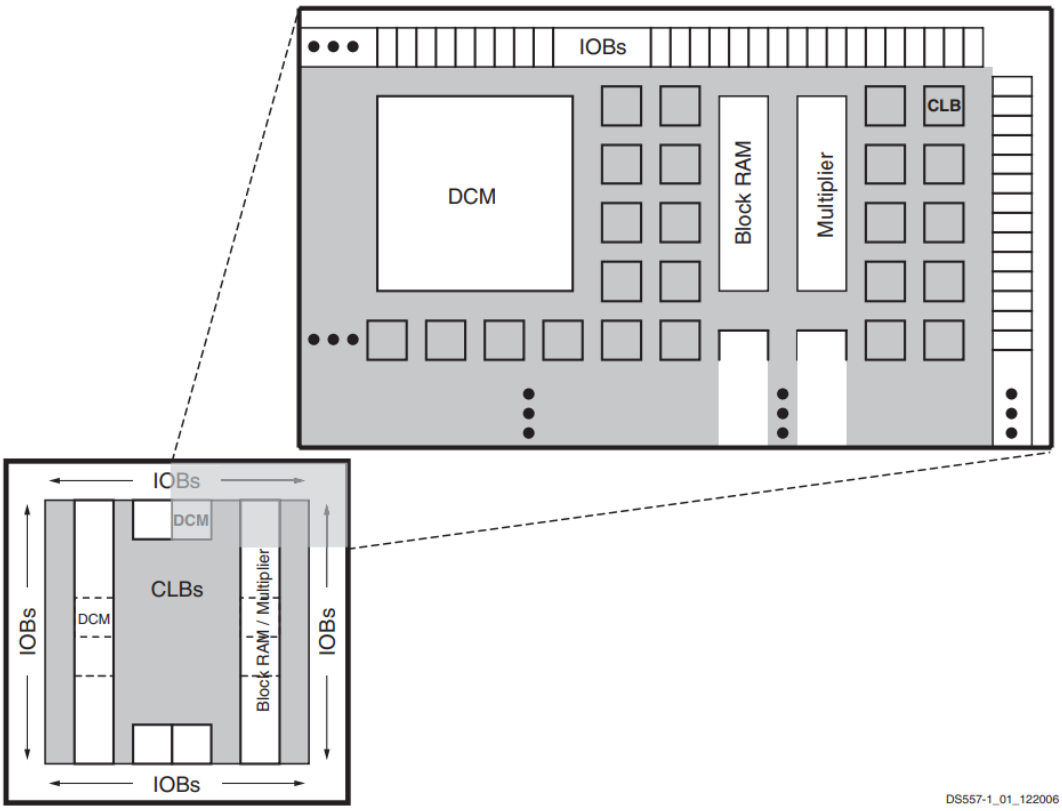


Figure 3.4: Xilinx Spartan-3AN Family Architecture[3]

3.1.2 System-on-a-chip

System-on-a-chip (SoC) is anatomy in embedded system which considers a processing board integrated with processor and hardware peripherals and memory for a specific function like digital, analog signal processing[28].

Traditional SoC board is a fixed Application Specific Integrated Circuit (ASIC) with is prebuilt for only one particular function or task. The processing time of ASIC board is speedy and high-efficiency due to the unique hardware prebuilt for this task. In the trade-off, the development time of an ASIC board takes much time due to multiple stages of hardware synthesis, testing and verification and the cost of development is enormous. ASIC board is only suitable for mass-production strategy with over 1 million products per a development cycle.

Figure 3.5: An ASIC board

Another approach is using an embedded processor for the implementation. All stages of the development are equivalent to a software development process. This approach

brings some advantages like low-cost development, flexibility, easy to fix and testing, short-time-development, and short-time-to-market. On the downside, the approach of using multipurpose processors causes high power consumption, low efficiency in the product operation.

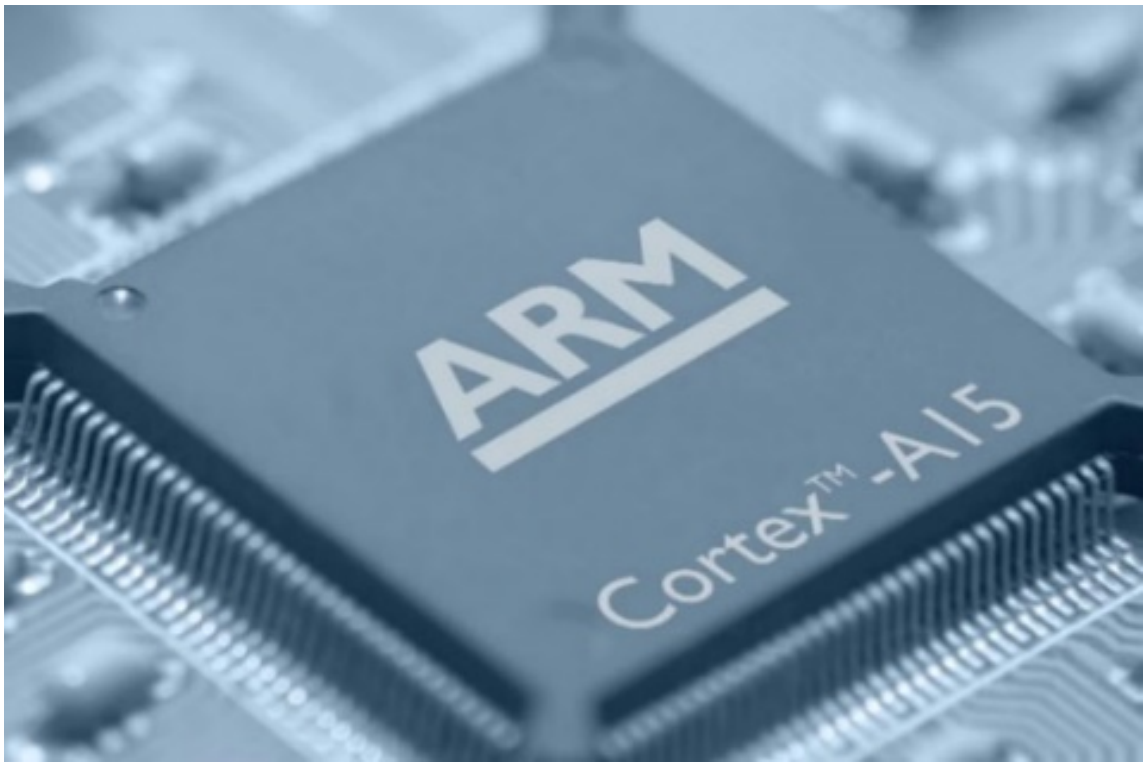


Figure 3.6: An ARM embedded processor

Programmable-System-on-a-chip (PSoC) is a hybrid way approach of SoC embedded system. It replaced the traditional ASIC by a Programmable Logic (FPGA) interfacing and embedded multipurpose processor with an integrated Processing system (usually a hard processor) through a high-speed bus. For zynq PSoC, the Advanced eXtensible Interface (AXI) is this high speed communicating bus. The PSoC cannot achieve the high-efficiency as the traditional ASIC approach but thanks to the modern technology and the short development time, the PSoC approach is the most suitable candidate for research and development projects which is below 1000 products for a cycle.

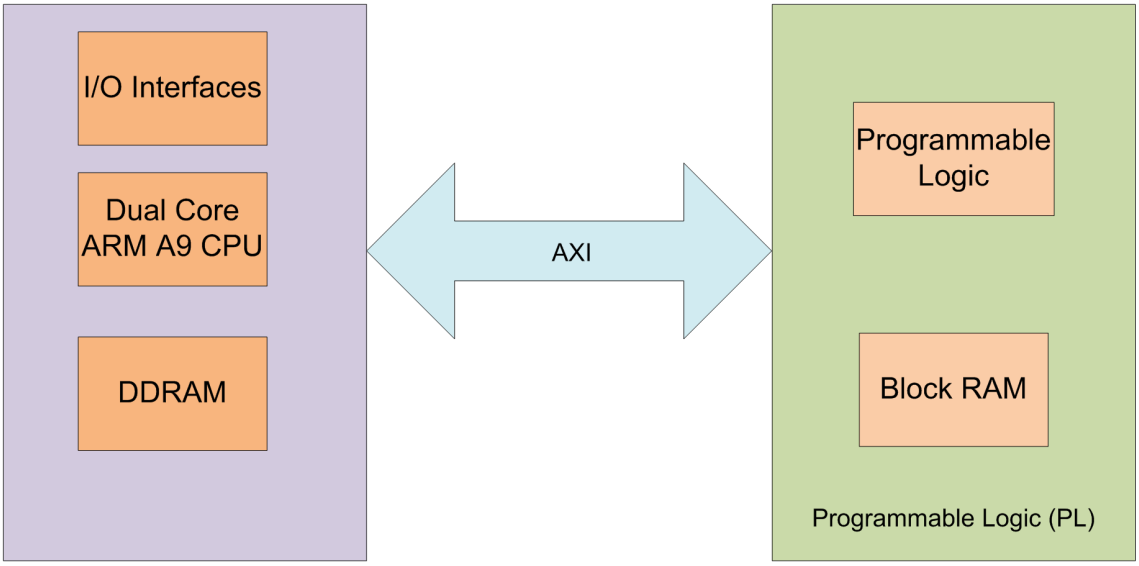


Figure 3.7: PS and PL connected through AXI

The PSoC board which is used to implement Star Tracking algorithm in this project is the Zynq zedboad illustrated in Figure 3.8. The defining feature of Zynq is that it combines a dual-core ARM-A9 processor with traditional FPGA logic fabric[29]. Although dedicated processors have been coupled with FPGAs before, it has never been quite the same proposition. In Zynq, the ARM Cortex-A9 is an application grade processor, capable of running full operating systems such as Linux, while the programmable logic is based on Xilinx 7-series FPGA architecture.

3.2 Partition and Profiling

3.2.1 Partition

Partitioning is the first stage of PSoC development. The algorithm or the program will be partitioned into suitable-size submodules. These submodules are later decided to implement on which side of the PSoC: Programmable Logics or Processing System. How the program partitioned is decided by the designer.

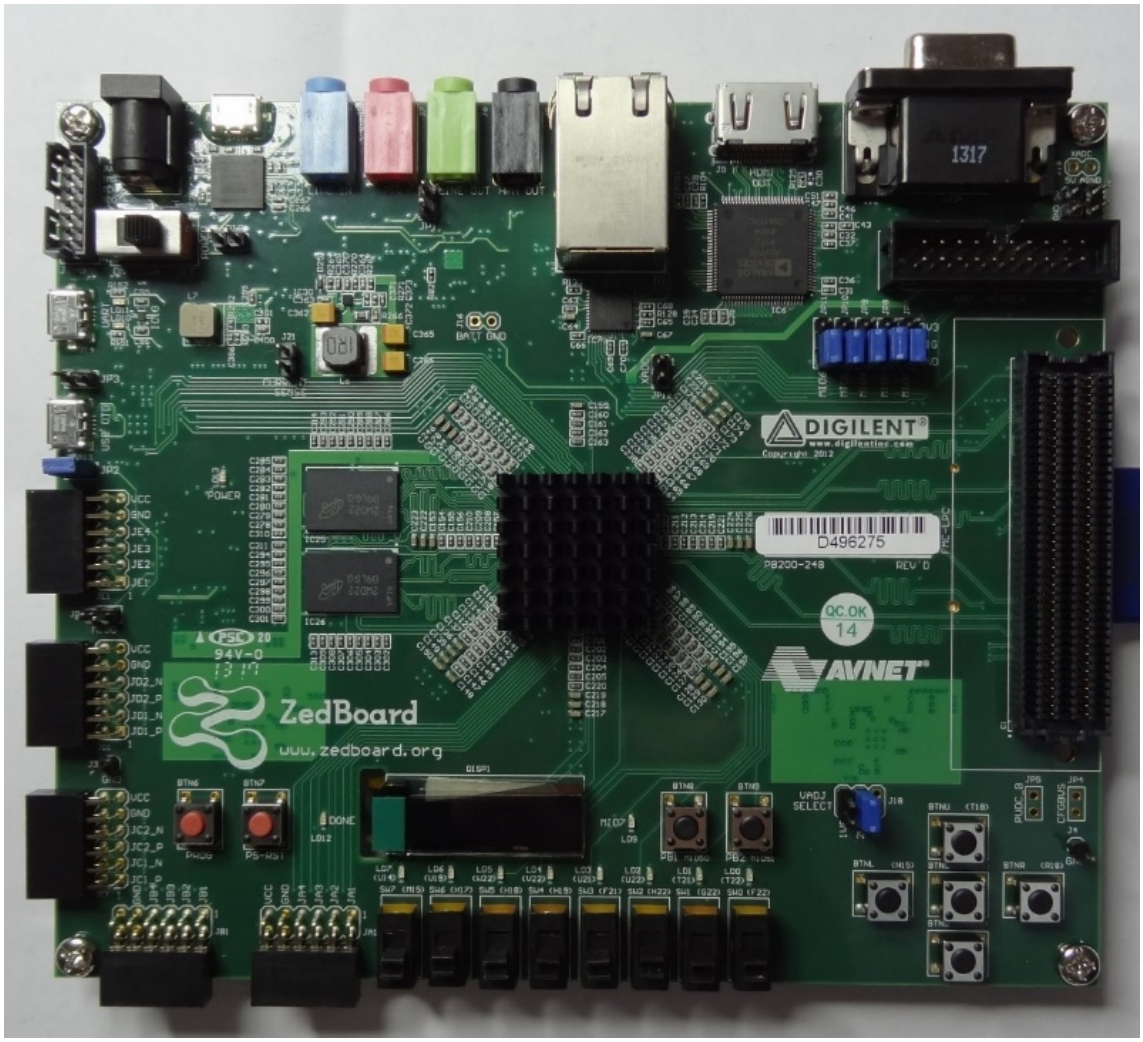


Figure 3.8: A Zedboard Programmable System on Chip

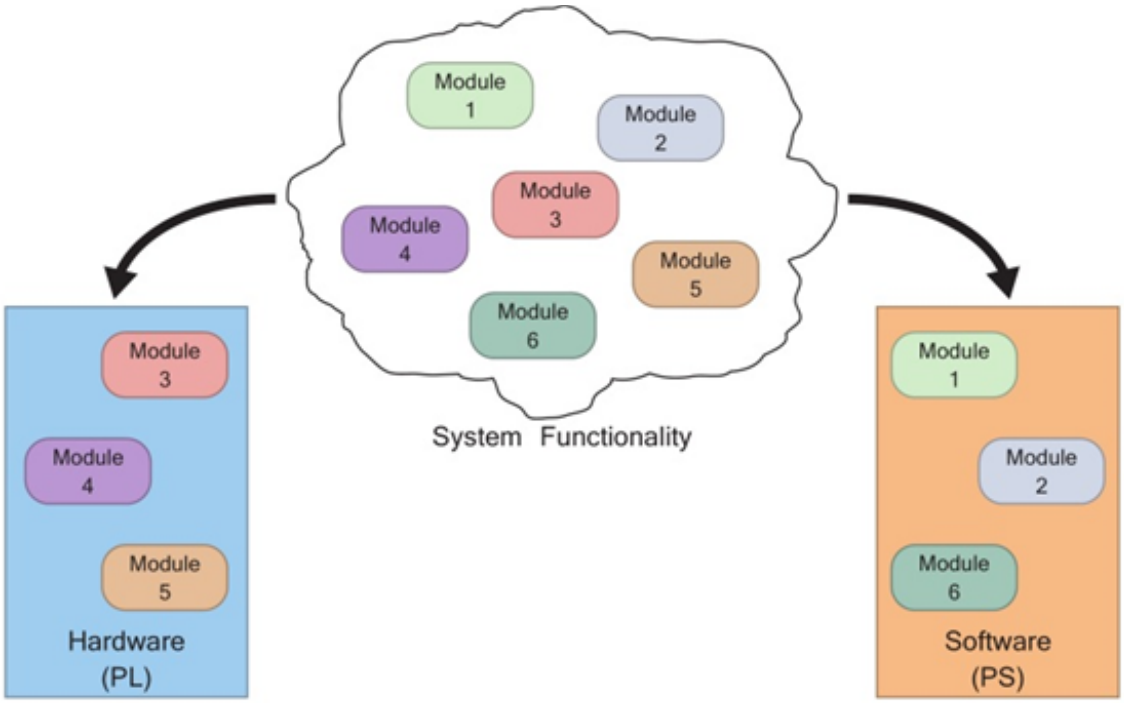


Figure 3.9: Module partitioning[4]

3.2.2 Profiling

Profiling is a practical analysis that measures and analyze the space and time complexity of an algorithm or a submodule of an algorithm. By examine the profiling result, we can specify the most critical time-consuming part or module of the system. Then based on the profiling and asymptotic analysis stated above, we can decide correctly which parts or submodules can be implemented in hardware instead of software.

All the submodules are implemented on Processing System side by software approach then determined which submodules take the most time to run. These modules then are optimized and designed as Intellectual Property core by Hardware description language (HDL) on Programmable Logics side.

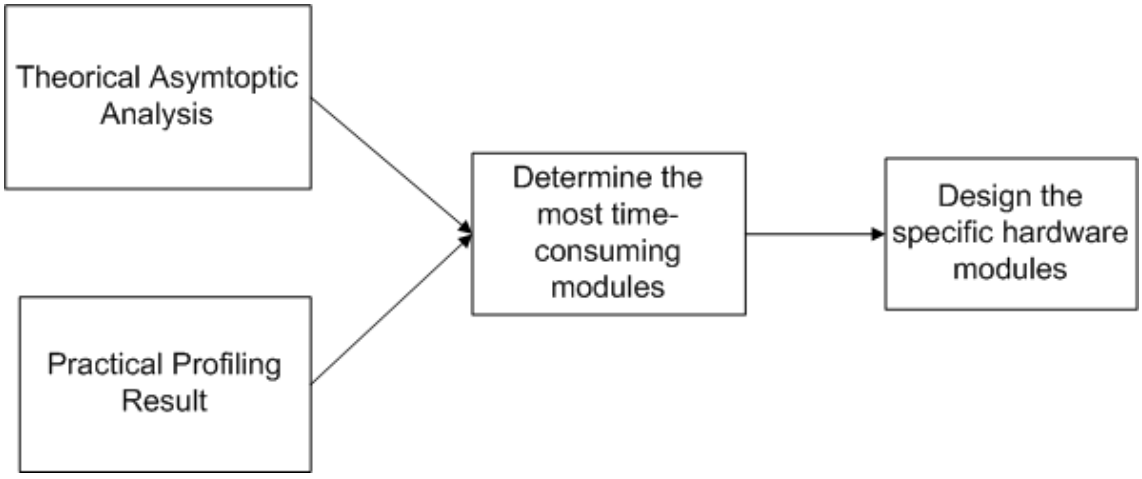


Figure 3.10: Profiling steps

3.2.3 Partitioning and profiling results

The star tracking algorithm is partitioned into 4 submodules shown in Figure 3.11. The 4 submodules include:

- **Centroiding**: Takes the star image and reproduce the star coordinates in the image.
- **Choose the reference star**: From the list of star coordinates, proceeds to choose the reference star.
- **Find the star pattern**: From the reference star, generates the feature vector of the image.
- **Pattern searching**: Search and match the feature vector to the prebuilt tree database to get the starID of the reference star.

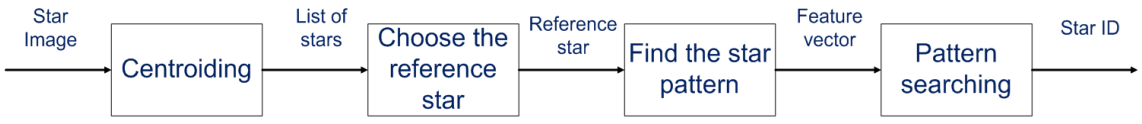


Figure 3.11: Star tracking algorithm submodules partitioned.

After partitioning, all submodules are implemented on the Processing system side to examine the runtime of each submodules. The result of profiling stage in Figure 3.12.

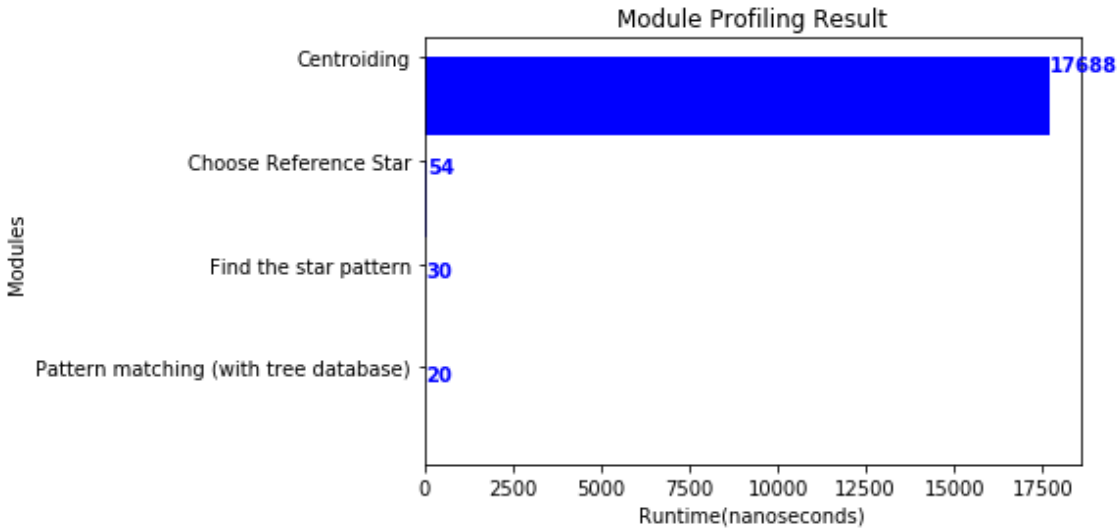


Figure 3.12: Profiling result.

Table 3.1.

Table 3.1: Module Profiling Result

Modules	**Runtime(nanoseconds)**
Centroiding	17688
Choose Reference Star	54
Find the star pattern	30
Pattern matching (with tree database)	20

Apparently, the most time-consuming submodule is the centroiding module. It can be explained because the centroiding stage is an image processing stage. Implementation of an image processing algorithm on general purpose processors is more costly and less effective than any other parallel processing approach like using a graphics processing unit (GPU), in this thesis, we design an IP core in the Programmable Logic side instead.

3.3 Module implementation

3.3.1 Centroiding

3.3.1.1 Thresholding

In the first stage, the sky is captured by a Charge-coupled device(CCD) through lens then discretized into digital star images. In which, Each pixel is an 8-bit integer value which represents the intensity or brightness of a pixel(0 represent darkest pixel and 255 represents the brightest pixel). The star image is usually contaminated with noise as a result of dark current noise and non-star stellar object. Dark current noise arises from thermal energy within the silicon lattice. It can be represented as a background image generated by a certain period of exposure with the shutter closed[8]. To eliminate the noise and separate star clusters from the background, we apply thresholding technique. The threshold level can be chosen by averaging all pixel intensities. The pixel which has a higher value than threshold level belongs to a star cluster while the pixel which has a lower value than the threshold belongs to the background. Figure 3.13 shows how to separate star clusters from the background, the threshold, in this case, is 70.

0	0	1	12	40	20	10	1	1	0	0	0	0	0
0	1	45	56	80	57	41	9	1	0	0	0	0	0
0	5	46	88	101	89	46	9	0	0	0	0	0	0
0	7	49	102	240	91	76	33	1	1	0	0	0	0
0	6	45	92	98	94	44	9	42	50	33	1	1	0
0	1	32	40	75	42	35	45	51	80	53	1	54	0
0	0	5	9	35	17	22	33	94	106	99	42	1	0
0	0	0	0	0	1	31	75	107	238	112	95	33	0
0	0	0	0	0	1	14	24	56	102	89	41	17	1
0	64	0	0	0	0	1	1	23	72	43	26	12	0
0	0	0	0	0	0	0	0	7	13	0	0	0	0

Figure 3.13: Thresholding separates star cluster pixels and background pixels

3.3.1.2 The connected-components problem in centroiding

The raw image is contaminated with noises and other non-stellar objects. These noises can be canceled by setting a threshold level. If a pixel Intensity is less than the threshold level, we will set its value to 0(the background level) to differentiate from star cluster pattern. After all false star noises are canceled, the centroiding algorithm proceeds.

The technique to determine star centroids in a star image employs the connected component finding. The traditional approach of connected component finding is considering each pixel in the star image as a vertex in a graph. If a group of pixels belongs to a star cluster pattern, those vertices are connected. Therefore we can determine each independent star clusters by using graph traversing technique like Depth-First-Search (DFS) or Bread-First-Search (BFS). However, this submodule of the star tracking algorithm is implemented as an IP core by Hardware Description Language (HDL) which lack abilities to implement a recursive function or advanced data structure like Queue, Stack.

The emergence of FPGAs with enough capacity to perform complex image processing tasks also led to high-performance architectures for connected component labeling[30]. Most of these architectures utilize the one-pass scan algorithm, because of the limited memory resources available on an FPGA. These types of connected component labeling architectures are able to process several image pixels in parallel, thereby enabling a high throughput at low processing latency to be achieved[31].

Figure 3.14 represents the star centroid finding problem as finding a connected-components problem. The same star cluster pixels are neighboring to each other. To determine the centroid of each star cluster, we must label the pixels into their corresponding star clusters(groups) then calculate each star cluster centroid of each cluster.

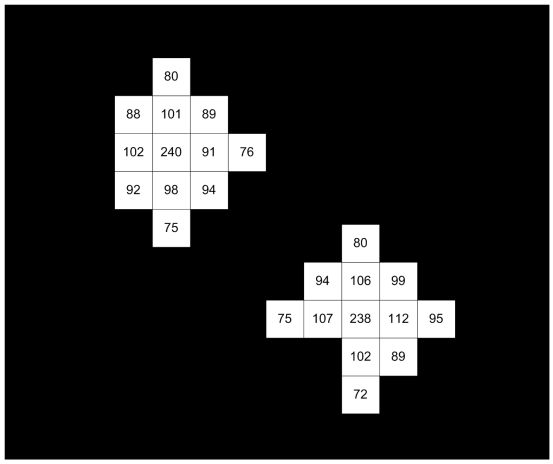


Figure 3.14: Star clusters represented as a connected-components problem

3.3.1.3 The one-pass scan algorithm

The one-pass scan algorithm is a connected component labeling approach which performs a scanning throughout the star image. A pixel in a star cluster will be put into an equivalent independent subset if two subsets having pixels connected to each other will be merged into one big set by the applying the following rules.

We maintain a disjoint set data structure when scanning performed if a pixel P is non-background, we consider its left and above pixels as in Figure 3.15. There are 3 cases:

1. **Case 1 - Both are background pixels**: Label the pixel (P) with a new number. Create a new subset.

2. **Case 2 - One(either pixel) is labeled, the other is not or both are labeled with the same number**: Label the pixel (P) with this number.

3. **Case 3 - Both are labeled with different numbers**: Label the pixel (P) with any number from the left or above pixels, merge the set of the above pixel into the set of the left pixel.

An example of the one-pass scan algorithm applied to label 2-star clusters: In Figure 3.16, the first non-background pixel P has been scanned and comes into case 1(both left and above pixels are background pixels), so we create a new disjoint set(set 1) and add this pixel to set 1.

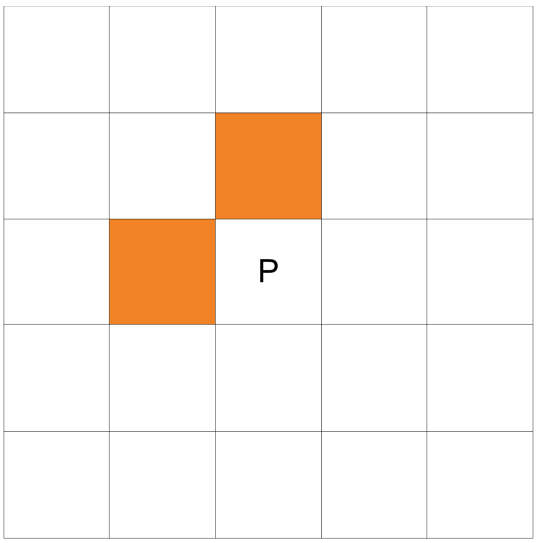


Figure 3.15: Left and above pixels of the current considering pixel.

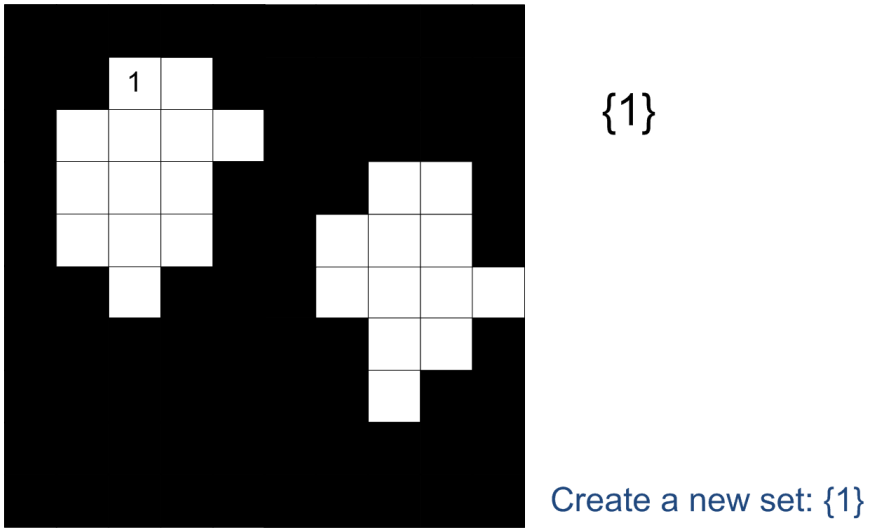


Figure 3.16: one-pass scan algorithm applied to label 2-star clusters - 1

Next, we scan the pixel in Figure 3.17, the above pixel is a background pixel, and its left pixel is in set 1. Therefore, we apply case 2 of the algorithm, set this pixel to the same set as its left pixel which is set 1.

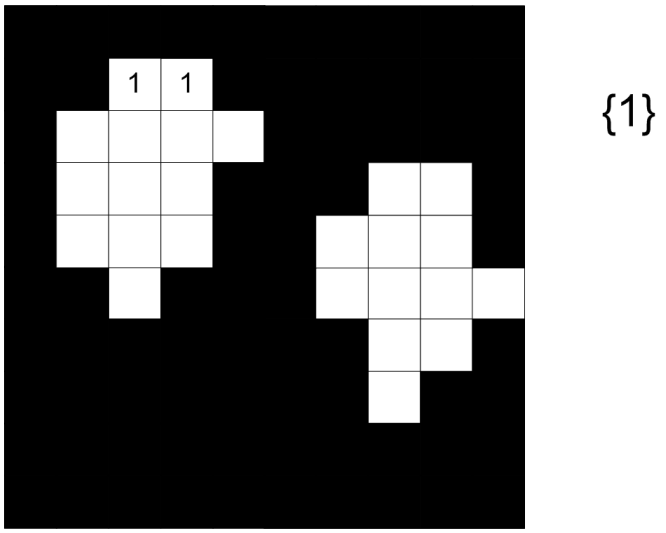


Figure 3.17: one-pass scan algorithm applied to label 2-star clusters - 2

After skipping background pixels, we consider the pixel shown in Figure 3.18, both above and left pixels of this pixel are background pixels(case 1). We create a new disjoint set(set 2) and add this pixel to set 2.

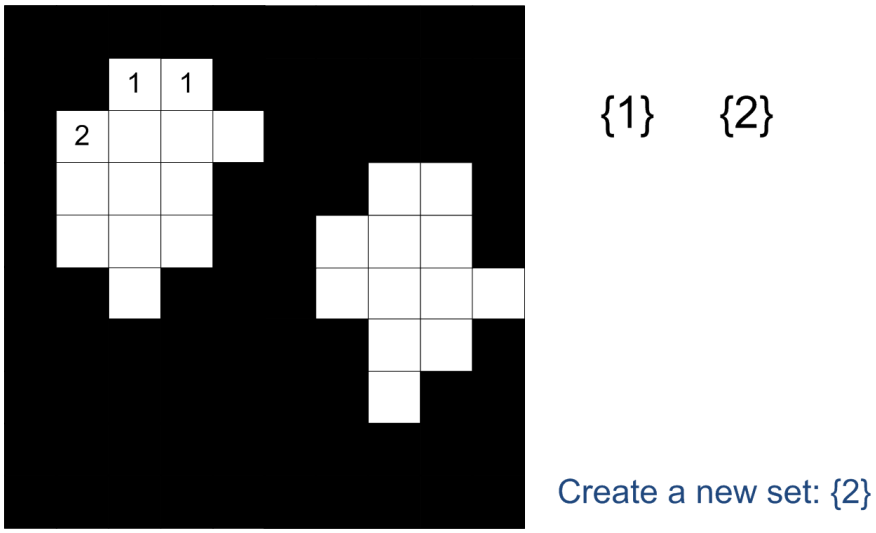


Figure 3.18: one-pass scan algorithm applied to label 2-star clusters - 3

The next pixel in Figure 3.19 belongs to case 3, so we merge 2 disjoint sets 1 and 2 and label this pixel as belonging to set 2.

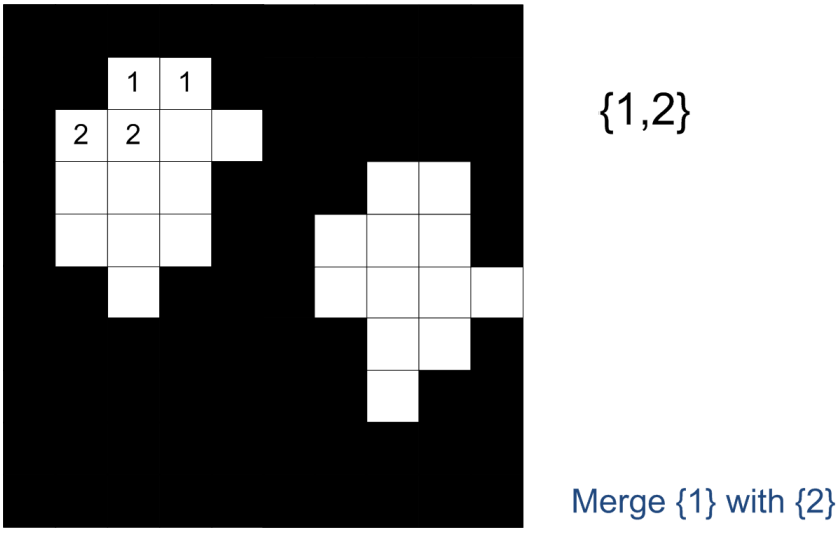


Figure 3.19: one-pass scan algorithm applied to label 2-star clusters - 4

At the final stage in Figure 3.20, we successfully find the 2 connected-component which are the pixels belongs to set 1,2, and the pixels belongs to set 3,4.

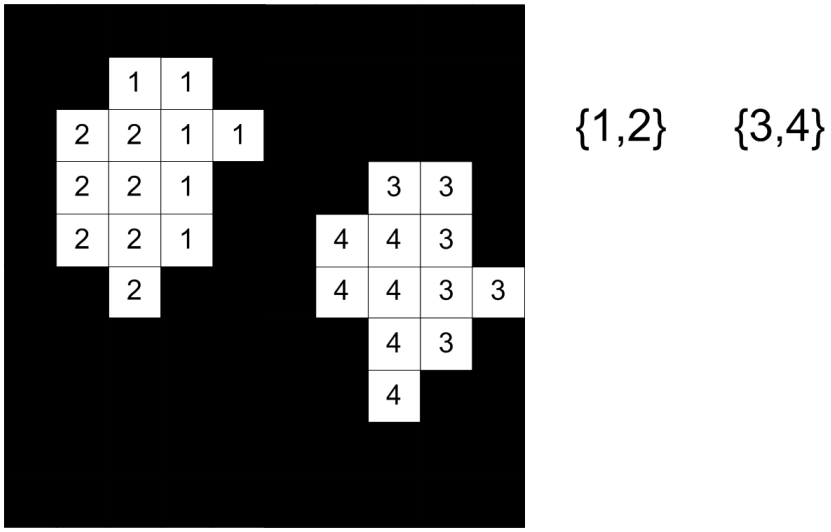


Figure 3.20: one-pass scan algorithm applied to label 2-star clusters - 5

After detecting the star clusters, we calculate the centroids of each star cluster by equation 3.1

$$x = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(i * I_{ij})}{\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}}, y = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(j * I_{ij})}{\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}} \tag{3.1}$$

The reason to choose the one-pass scan algorithm is that an image can not be loaded into the hardware processing unit as a matrix or tensor but as a pixel stream. We only know the previous pixels of the current receiving pixel.

3.3.1.4 IP core design for the one-pass scan algorithm

An image as streamed pixels

From the one-pass scan algorithm, we need to design an IP core for faster hardware processing. The input of the module is a stream of pixels, and the output is the centroid coordinates of the stars in this image as shown in Figure 3.21.

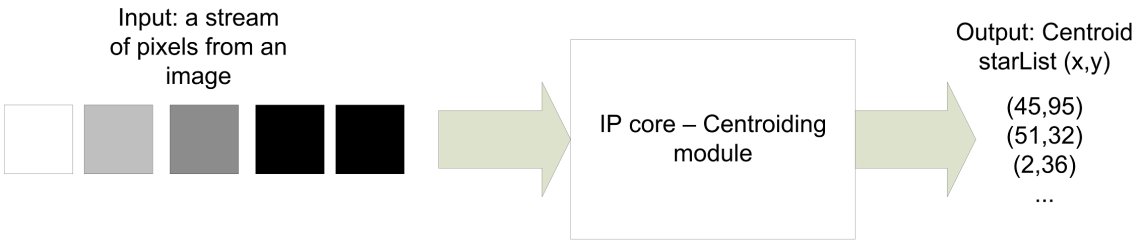


Figure 3.21: IP core design purpose

In figure 3.22, the pixel coordinates(i,j) in 2D images would be represented by the stream order(k). The reason we need to have a converting framework is that we need to look up the above and left pixels of a considering pixel in the one-pass scan algorithm.

The converting equations:

$$k = i * ImgHeight + j$$

$$i = k/ImgHeight$$

$$j = kmodImgHeight$$

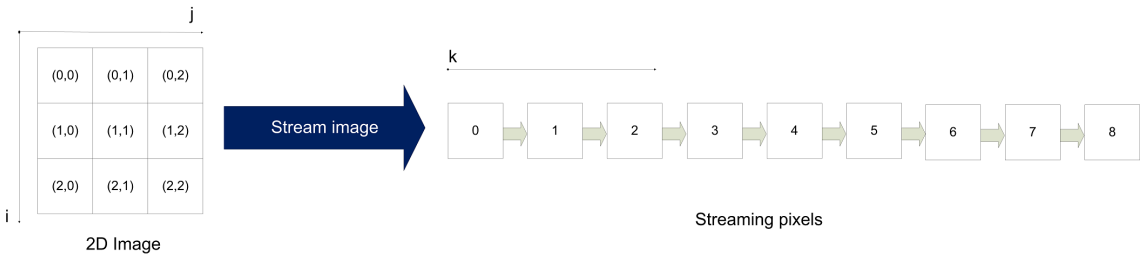


Figure 3.22: Stream an image into a pixel stream

IP core submodules

The centroiding again is partitioned into submodules for specific purposes as Figure 3.23.

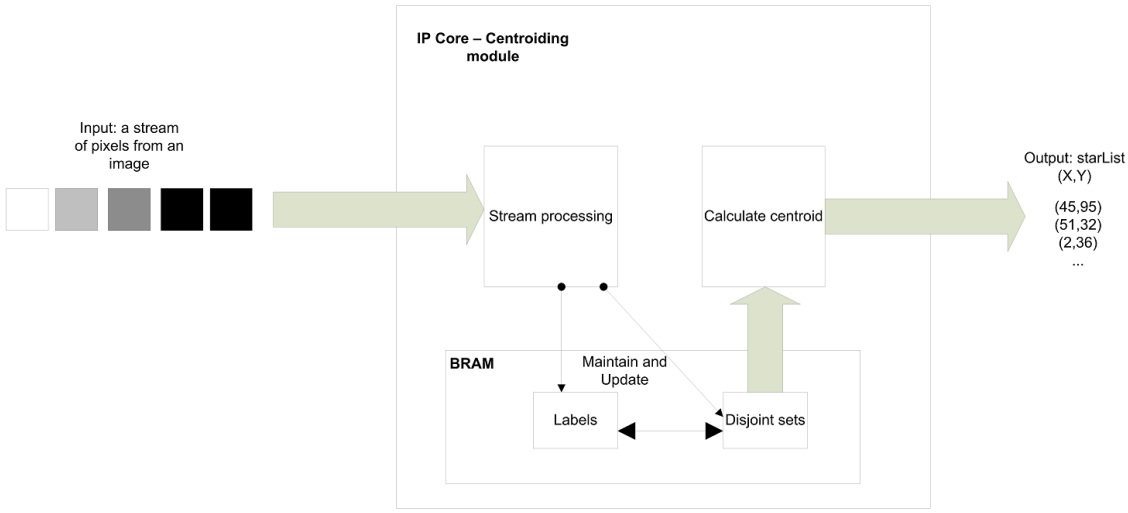


Figure 3.23: IP core submodules

- **stream processing**: The submodule task is to process the one-pass scan algorithm after receiving a pixel from the stream. The submodule also in responsibility of maintaining and updating the labels and disjoint sets data structures stored in the BRAM. The pseudo codes of the implementation are shown in Figure 3.24 and Figure 3.25
- **centroid calculating**: The submodule task is to calculate the centroid coordinates after the stream processing stage is completed based on Equation 2.1.

The inputs of the centroid calculating submodule is the disjoint sets data structures. The pseudo codes of the implementation are shown in Figure 3.26

- **BRAM**: served as the memory for the IP core. Stored the temporal data structures like: labels and disjoint sets and the final results(star centroid list).

```
def get_above_pixel(k):
    return k - imgHeight + (k mod imgHeight)


def get_left_pixel(k):
    return k + (k mod imgHeight) - 1
```

Figure 3.24: Get above and left pixel functions

```
def stream_processing(current_pixel, disjoint_sets, labels):
    # get all pixel postions
    above_pixel = get_above_pixel(current_pixel)
    left_pixel = get_left_pixel(current_pixel)

    # Apply the one-pass scan algorithm
    ## Case 1: Both are background pixels
    if (above_pixel == BACKGROUND) and (left_pixel == BACKGROUND):
        cur_label = labels.create_new_label()
        current_pixel.set_label(cur_label)
        disjoint_sets.create(cur_label)

    ## Case 2:  One(either pixel) is labeled, the other is not or
    ##          both are labeled with the same number
    else if isLabeled(above_pixel) or isLabeled(left_pixel) or
            get_label(above_pixel) == get_label(left_pixel):
        cur_label = labels.get_label(above_pixel)
        current_pixel.set_label(cur_label)

    ## Case 3:  Both are labeled with different numbers
    else if get_label(above_pixel) != get_label(left_pixel):
        above_pixel_label = labels.get_label(above_pixel)
        left_pixel_label = labels.get_label(left_pixel)
        cur_label = above_pixel_label
        current_pixel.set_label(cur_label)
        disjoint_sets.join(above_pixel_label, left_pixel_label)
```

Figure 3.25: Stream processing submodule

```
def calculate_centroid():
    centroidList = list()
    for each set in disjoint_sets:
        total_intensity = 0
        (x,y) = (0,0)
        for each pixel in set:
            x += pixel.x() * pixel.Intensity()
            y += pixel.y() * pixel.Intensity()
            total_intensity += pixel.Intensity()
        X = x / total_intensity
        Y = y / total_intensity
        centroidList.add(X,Y)

    return centroidList
```

Figure 3.26: centroid calculating submodule

```
def centroiding():
    # Initate
    disjoint_sets.Initiate()
    labels.Initiate()

    # submodule: Process stream
    while(receiving_input_pixel() and not timeout()):
        current_pixel = receiving_input_pixel()
        stream_processing(current_pixel, disjoint_sets, labels)

    # submodule: calculate centroids
    centroidList = calculate_centroid(disjoint_sets)

    return centroidList
```

Figure 3.27: The complete IP core module

3.3.2 Choose the reference star

Figure 3.28 shows a basic star image which then will be processed through the Choose the reference star stage. The first Image in an example is a typical star image. The outer circle represents the FOV of the star Image, and the small blue dots represent the star clusters.

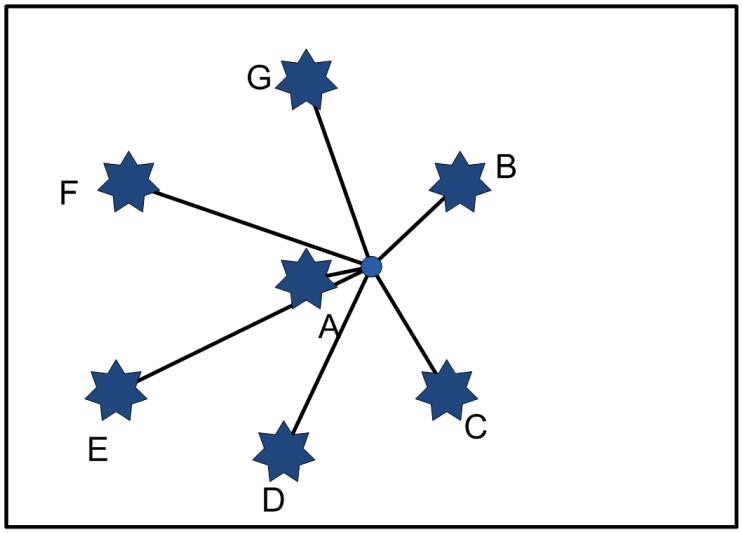


Figure 3.28: Distances from the centre of the image to all stars

From the centre of the star Image, we calculate the distances from each star cluster to the centre by Euclidean distance as equation 3.2. The star which is nearest to the centre (have the minimum distance value) will be chosen as the reference star.

$$d_{centre\_to\_star} = \sqrt{(x_{star} - x_{centre})^2 + (y_{star} - y_{centre})^2} \tag{3.2}$$

3.3.3 Find the star pattern

After we have the reference star, the next stage is to generate the star pattern. The number of stars in the image is counted, and their distances to the reference star are calculated as equation 3.3 to form a feature vector $f = \{N, D_1, D_2, D_3, D_n\}$

$$d_{centroid\_to\_star} = \sqrt{(x_{star} - x_{centroid})^2 + (y_{star} - y_{centroid})^2} \tag{3.3}$$

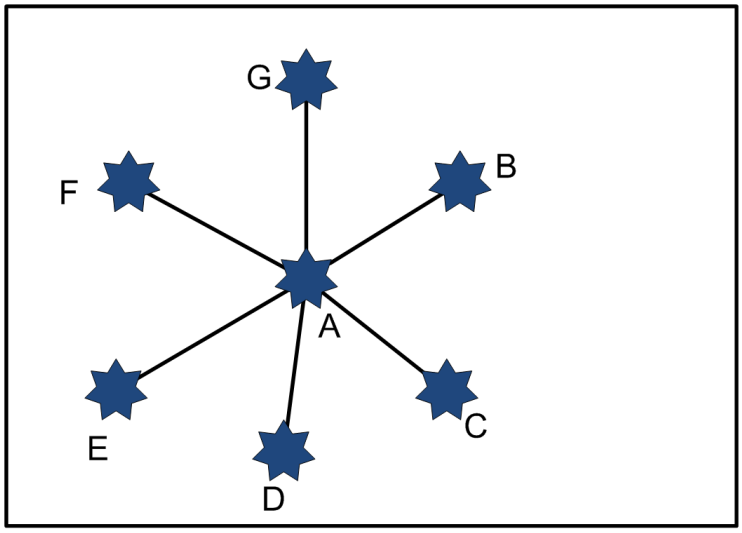


Figure 3.29: Distances from the reference star to its neighbors

3.3.4 Pattern searching

In the last stage, the feature vector is matched to the prebuilt tree pattern database. The feature vector from the Generate feature vector stage will be compared to the prebuilt tree structured star pattern database(SPD) to find the correct star cluster ID. Figure 3.30 shows how a star vector is compared and match with the SPD.

In case of mismatching, a tolerance will be added to research for another star ID. This search can be performed parallel to increase the searching speed due to the specific tree structure of the prebuilt SPD as in Figure 3.31.

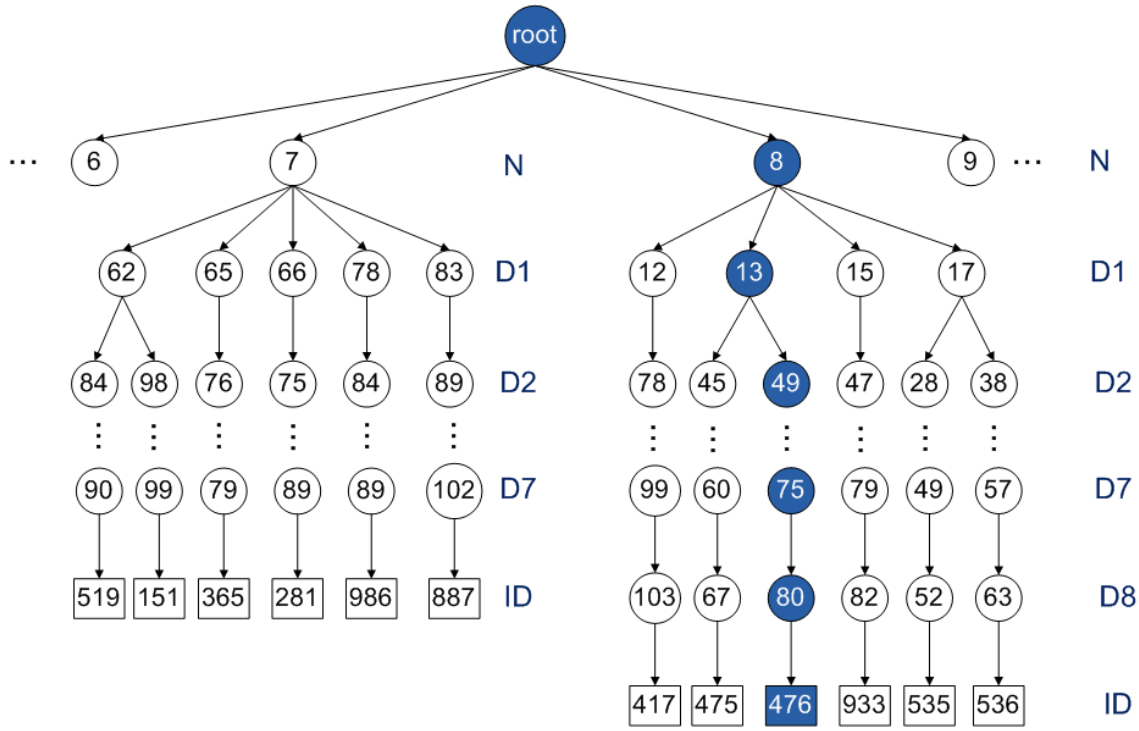


Figure 3.30: Star pattern matching

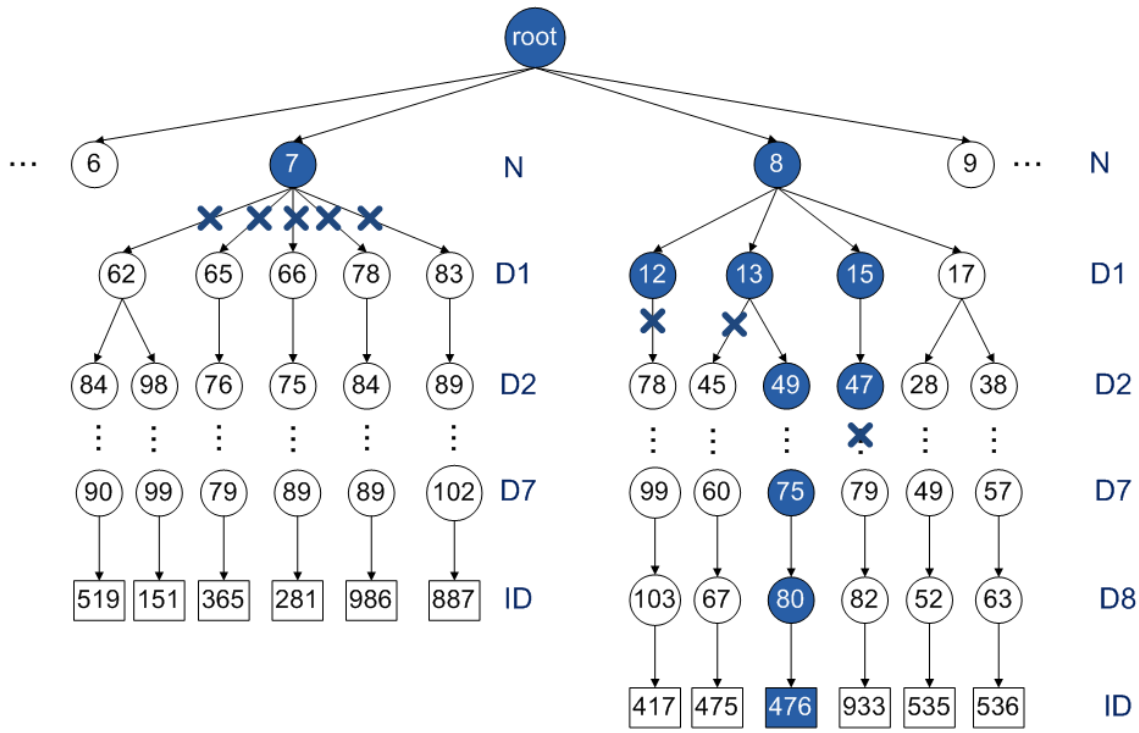


Figure 3.31: Star pattern matching with tolerance

Chapter 4

Experimental Results

4.1 Hardware design

4.1.1 Components and Design Flow

The hardware components shown in Figure 4.1 includes:

- Processing system.
- Centroiding IP core.
- Direct Memory Access controlling system.
- BRAM
- Hardware timer
- Interrupt controller

Data path

The pixel data are streamed directly into memory by DMA module. The IP core is designed for centroiding processing the algorithm as soon as the first pixel data arrived. After the centroiding algorithm finishing, the IP core store the results of centroiding algorithm into BRAM and informs the PS by an interrupt signal. PS then takes the results from BRAM and process other stages. The data path is implemented through AXI.

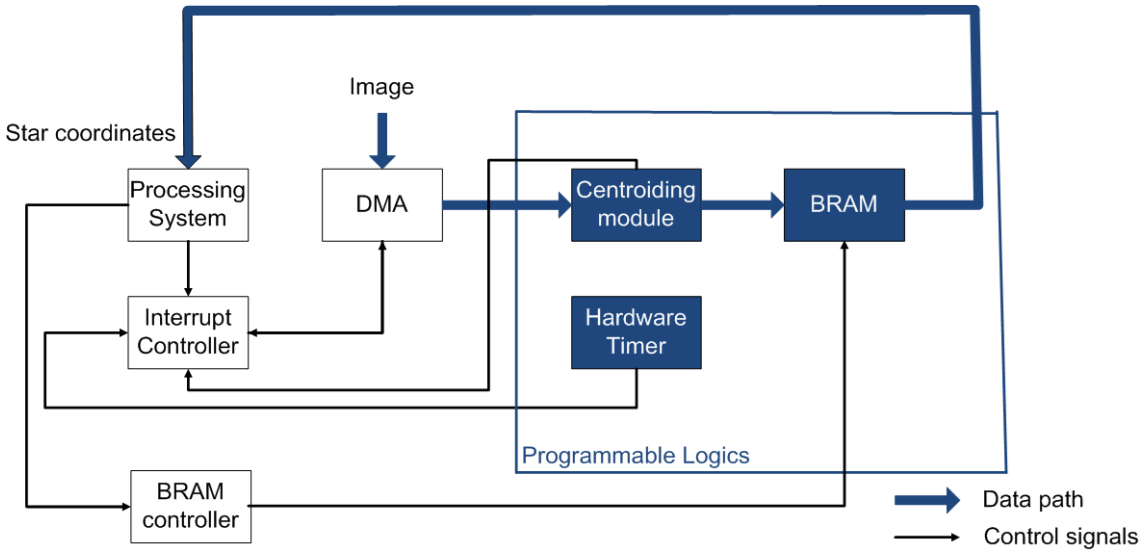


Figure 4.1: Hardware design diagram

Control signals

The PS and PL side communicate through an interrupt controller. The interrupt controller informs the PS if the centroiding algorithm or hardware timer has finished, informs the DMA controller if the memory is full, the DMA controller then flush the old data to free the memory. BRAM controller is a dual channel BRAM which is readable from PS side and writable from PL side.

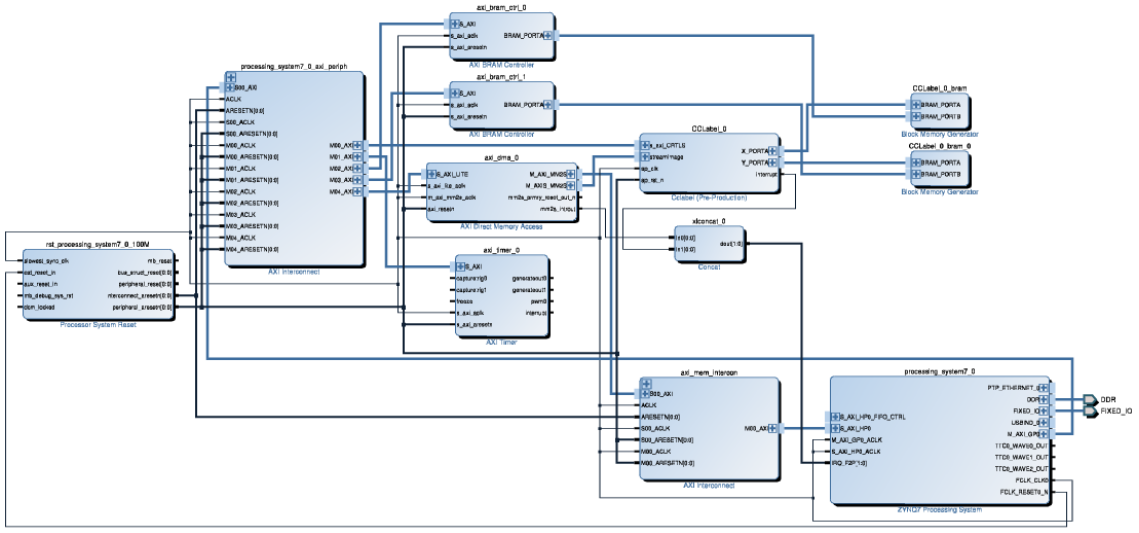


Figure 4.2: Hardware design block

The graph 4.3 shows the percentage of hardware resources used over available resources from the zedboard. We still have more spare resources of the FPGA for other tasks.

Table 4.1: Hardware resource consumption

Resource	**Used**	**Available**	**Percentage (%)**
BRAM_18K	17	280	6.07
DSP48E	20	220	9.09
FlipFlops	9278	106400	8.72
Lookup Tables	13387	53200	25.16

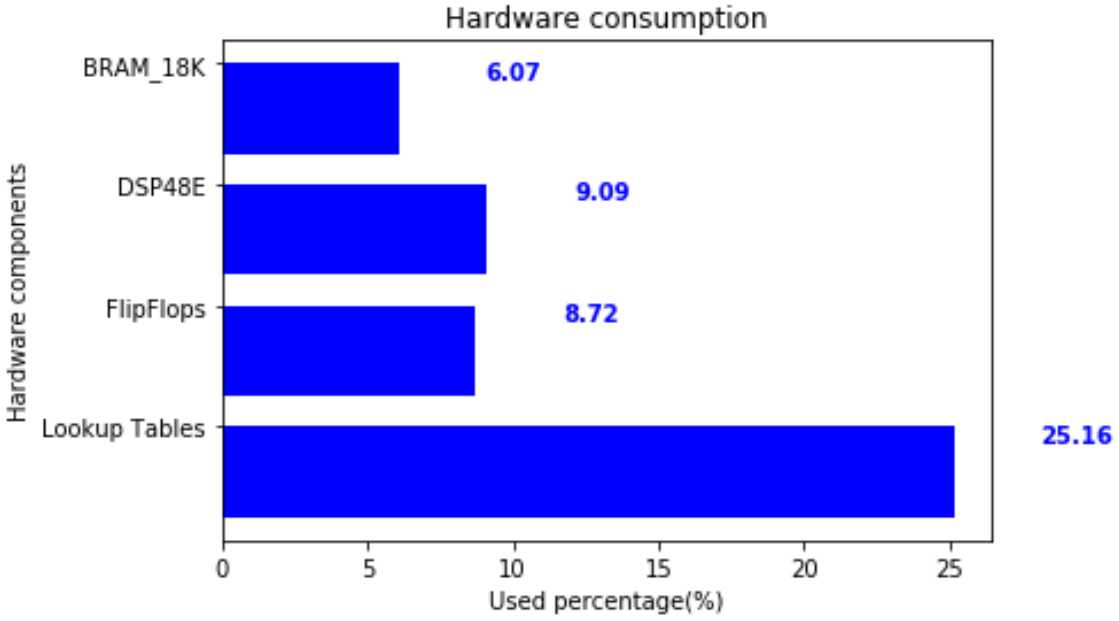


Figure 4.3: Hardware resource consumption

4.1.2 Power Analysis

The power consumption of the design is measured based on the condition described in Table 4.2.

Table 4.2: Environment condition

Condition	value
Ambient temperature	30.0^oC
Airflow	250 LFM
heat sink	medium
board temperature	25.0^oC

As in Figure 4.4 and Figure 4.5, the Processing System(the ARM A9 processor) consumes 77% of the power supplied while other hardware components like the IP core, BRAM consume less power. This is due to the fact that generic processors are designed for multiple purpose tasks; hence they are not optimal for power consumption as the hardware solution for a specific task.

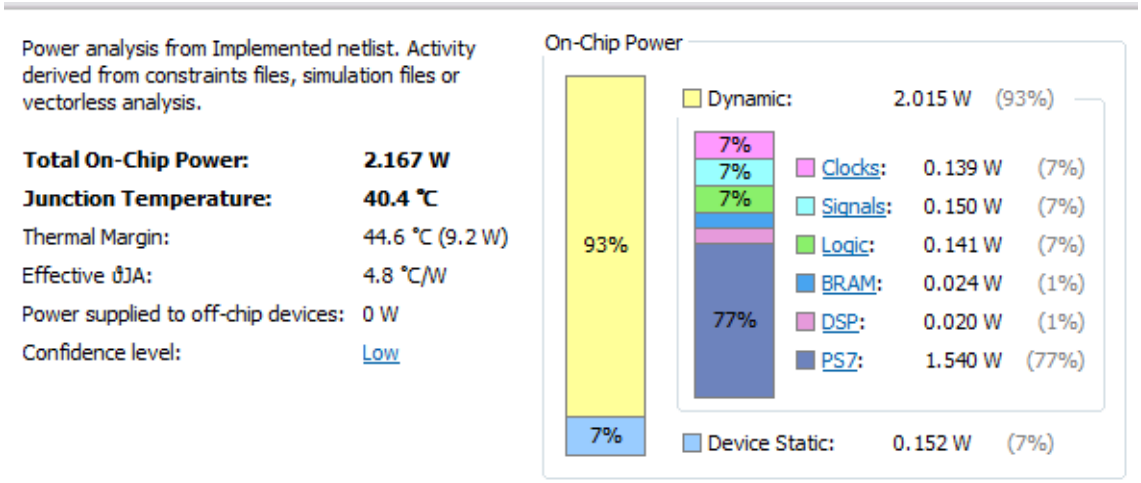


Figure 4.4: Power consumption summary

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	DSP (W)	PS7 (W)	Processor (W)
2.015 W (93% of total)	design_1_wrapper										
2.015 W (93% of total)	design_1_i (design_1)	0.139	0.15	0.145	0.004	0.001	0.141	0.024	0.02	1.54	0.277
1.544 W (71% of total)	processing_system7_0 (design_1_...	<0.0001	0.004	0.004	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	1.54	0.277
1.544 W (71% of total)	inst (processing_system7_v5_5_p...	<0.0001	0.004	0.004	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	1.54	0.277
0.353 W (16% of total)	CCLabel_0 (design_1_CCLabel_0_0)	0.09	0.117	0.115	0.001	<0.0001	0.115	0.01	0.02	<0.0001	<0.0001
0.352 W (16% of total)	inst (CCLabel)	0.09	0.117	0.115	0.001	<0.0001	0.115	0.01	0.02	<0.0001	<0.0001
<0.0001 W (<1% of total)	Leaf Cells (41)										
0.064 W (3% of total)	processing_system7_0_axi_peri...	0.028	0.019	0.018	0.002	<0.0001	0.016	<0.0001	<0.0001	<0.0001	<0.0001
0.018 W (1% of total)	axi_dma_0 (design_1_axi_dma_0_0)	0.009	0.002	0.001	<0.0001	<0.0001	0.002	0.006	<0.0001	<0.0001	<0.0001
0.007 W (1% of total)	axi_timer_0 (design_1_axi_timer_...	0.002	0.002	0.002	<0.0001	<0.0001	0.003	<0.0001	<0.0001	<0.0001	<0.0001
0.007 W (1% of total)	axi_mem_intercon (design_1_axi_...	0.004	0.002	0.001	<0.0001	<0.0001	0.001	<0.0001	<0.0001	<0.0001	<0.0001
0.007 W (1% of total)	axi_bram_ctrl_0 (design_1_axi_br...	0.003	0.002	0.002	<0.0001	<0.0001	0.002	<0.0001	<0.0001	<0.0001	<0.0001
0.007 W (1% of total)	axi_bram_ctrl_1 (design_1_axi_br...	0.003	0.002	0.002	<0.0001	<0.0001	0.002	<0.0001	<0.0001	<0.0001	<0.0001
0.004 W (<1% of total)	CCLabel_0_bram (design_1_CCLa...	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	0.004	<0.0001	<0.0001	<0.0001
0.004 W (<1% of total)	CCLabel_0_bram_0 (design_1_CC...	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	0.004	<0.0001	<0.0001	<0.0001
<0.0001 W (<1% of total)	rst_processing_system7_0_100M ...	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001

Figure 4.5: Power consumption by components

4.2 Runtime experiments

4.2.1 Star data analysis

The main factors affect the runtime of the designed system are:

- Image resolution
- Number of stars in one image
- Star area (in pixels) of one star

The images simulated are based on the SSAO star catalog and the standard ESO star map. Randomized distribution false stars are added to each image. For each resolution dataset, we have 12530 simulated images.

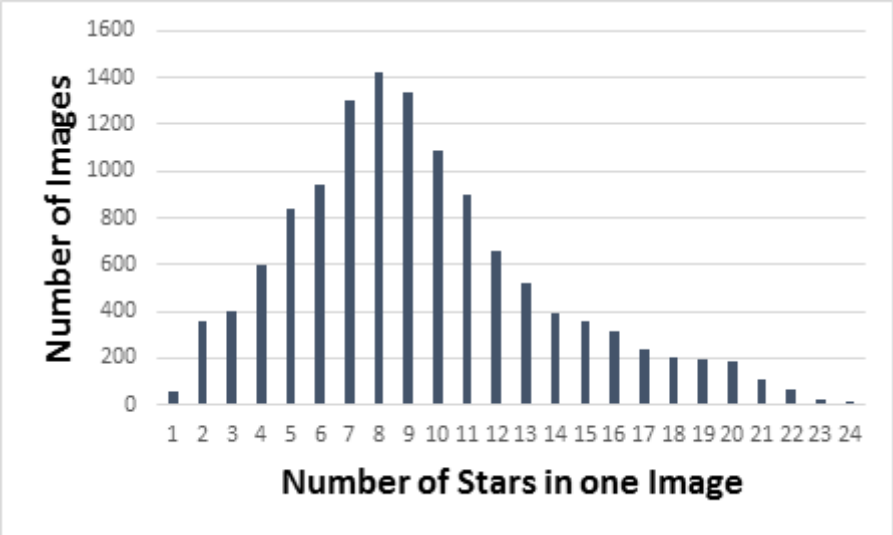


Figure 4.6: Number of stars in one image distribution

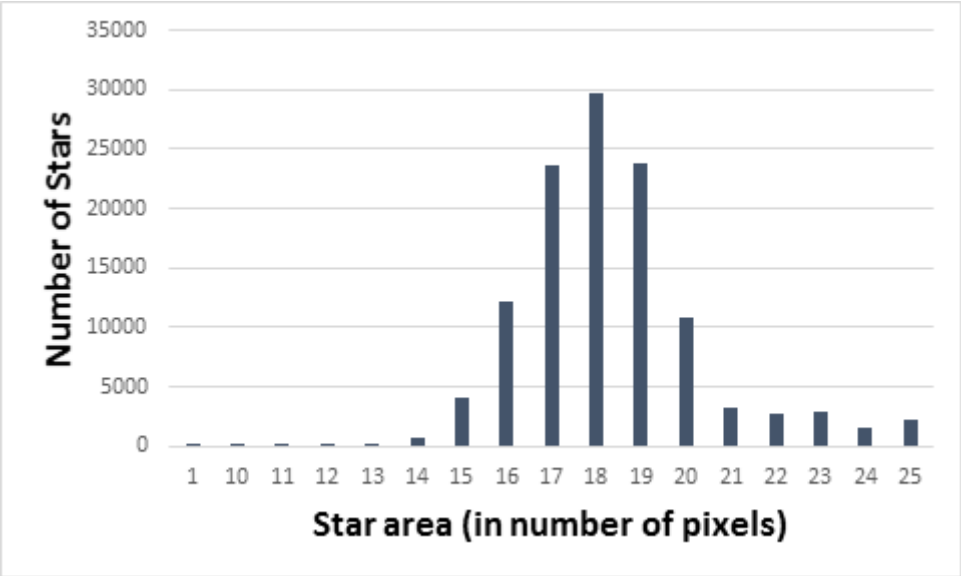


Figure 4.7: Star area distribution

The algorithm runtime performance is then compared between the implementation run on the embedded processor and the application executed on software-hardware co-processing strategy.

4.2.2 512x512 dataset

The star tracker is configured as Table 4.3

Table 4.3: Star tracker configuration

Parameters	**Value**
Field of View (FOV)	J2000
Inertial frame	8.0 degrees full cone
Visual magnitude threshold	5.0
Number of samples	12530

Software runtime

Table 4.4: Software runtime

	Average	**Worst case**
Centroiding	17688	18061
Choosing Reference Star	54	105
Finding Star Pattern	30	109
Pattern Matching	83	155

The average runtime is 17855 us (about 17.86 ms)

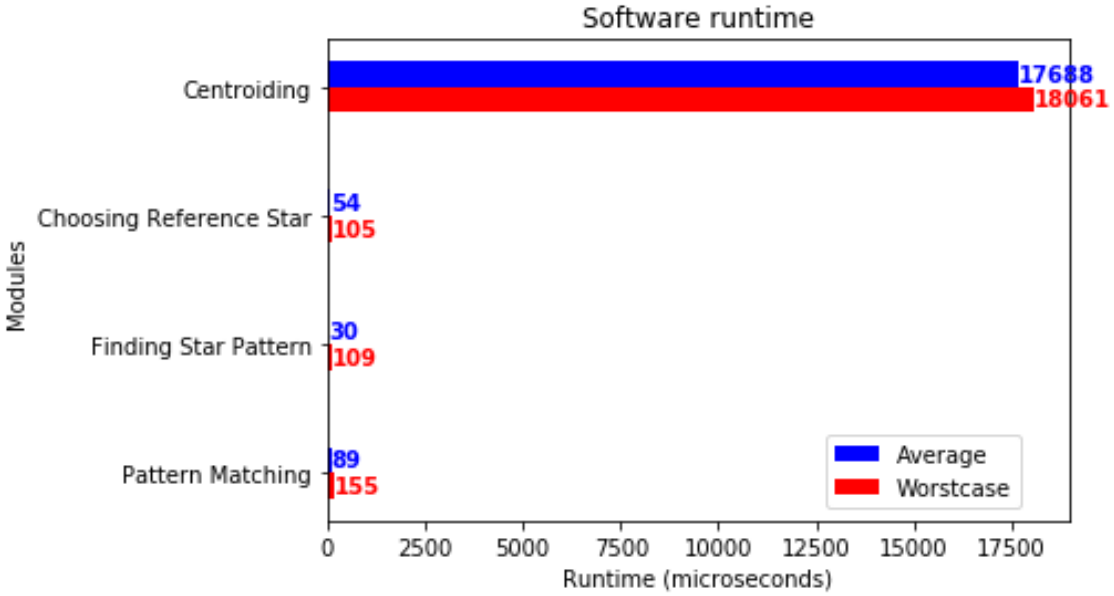


Figure 4.8: Software runtime

Hardware Software Co-processing runtime

Table 4.5: Hardware Software Co-processing runtime

	Average	**Worst case**
IP core Initiation	6	6
DMA Data Streaming	2460	2485
Centroiding	3719	3806
Choosing Reference Star	54	145
Finding Star Pattern	30	109
Pattern Matching	83	155

The average runtime is 6352 us(about 6.35 ms)

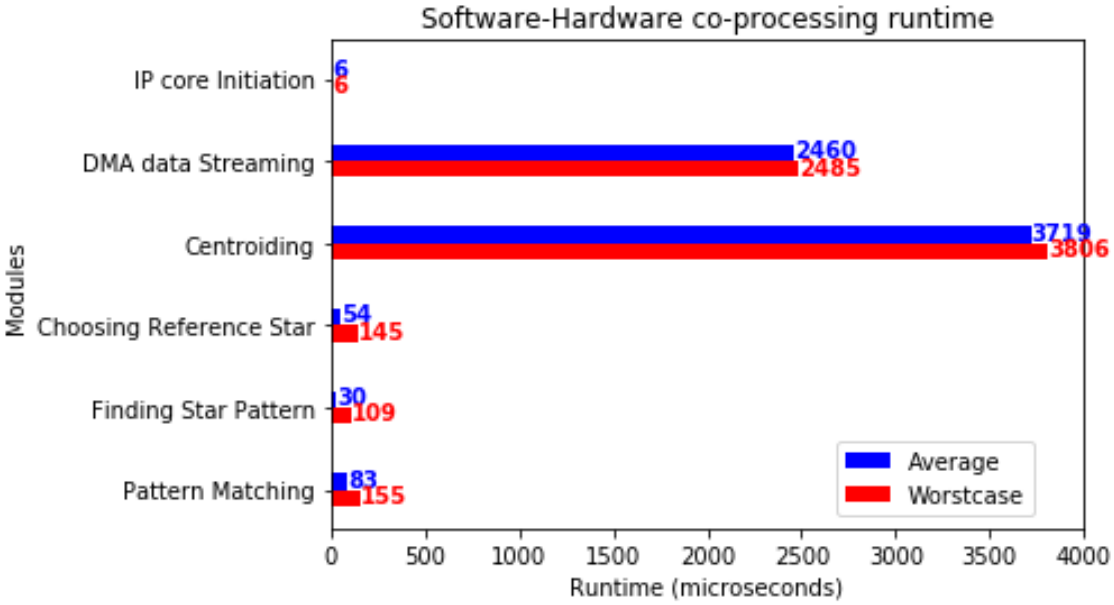


Figure 4.9: Software-Hardware co-processing runtime

4.2.3 1024x1024 dataset

The images are captured by the star tracker SST-20S described by Table 4.9

Table 4.6: Specifications of the SST-20S star tracker and the real image[1]

Parameter	Value
Field of View (FOV)	$15^o * 15^o$
Sensitivity (Mv)	6.3
Star catalog	SAO J2000[32]
Resolution (w * h)	1024 x 1024 pixels
Pixel size (ρ)	13 m
Focal length (f)	50 mm
Bits per pixel	8
Position accuracy	40 arcsec

Software runtime

The average runtime is 70780 us (about 70.78 ms)

Table 4.7: Software runtime

	Average	Worst case
Centroiding	70527	70561
Choosing Reference Star	63	108
Finding Star Pattern	54	115
Pattern Matching	136	155

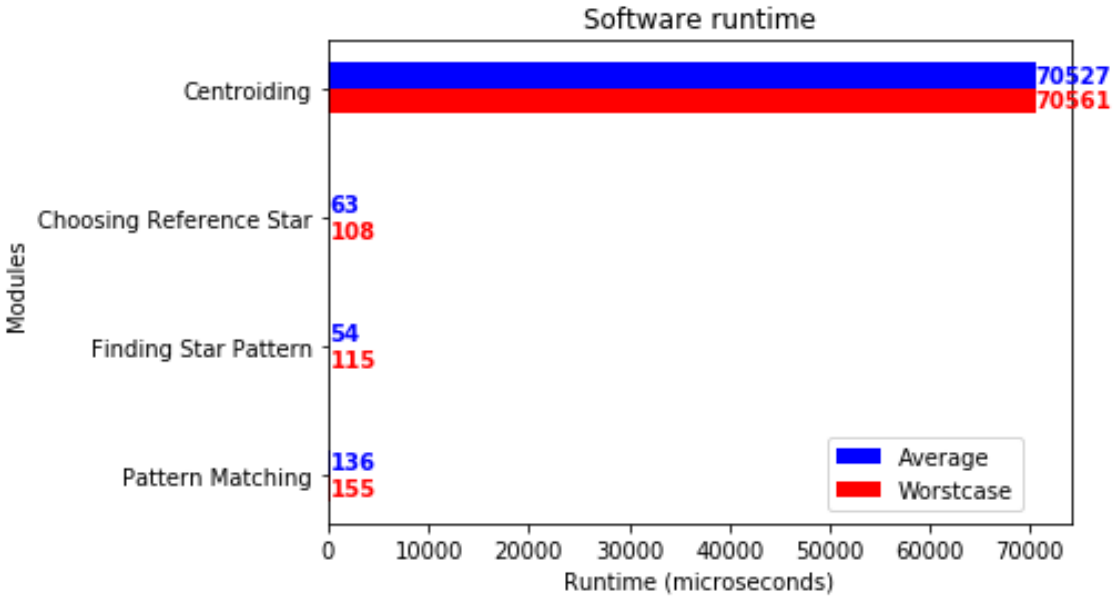


Figure 4.10: Software runtime

Hardware Software Co-processing runtime

Table 4.8: Hardware Software Co-processing runtime

	Average	**Worst case**
IP core Initiation	6	6
DMA Data Streaming	9834	9838
Centroiding	3543	3598
Choosing Reference Star	63	108
Finding Star Pattern	54	115
Pattern Matching	136	155

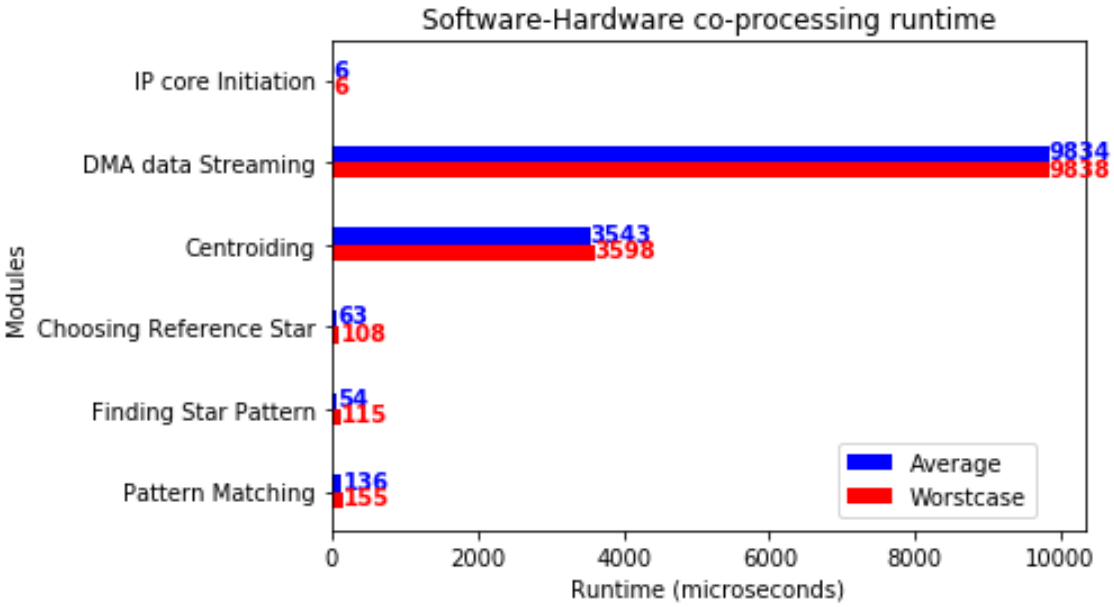


Figure 4.11: Hardware Software Co-processing runtime

The average runtime is 13636 us (about 13.64 ms)

4.2.4 Runtime Comparison

Table 4.9: Runtime comparison

	Average(ms)	Worst case(ms)
Liebe method	190.6	304.7
Geometric voting method SW	958.4	1044.7
Pyramid method SW	352.3	576.6
Proposed method SW	70.8	70.9
Proposed method SW-HW coprocessing	13.6	13.8

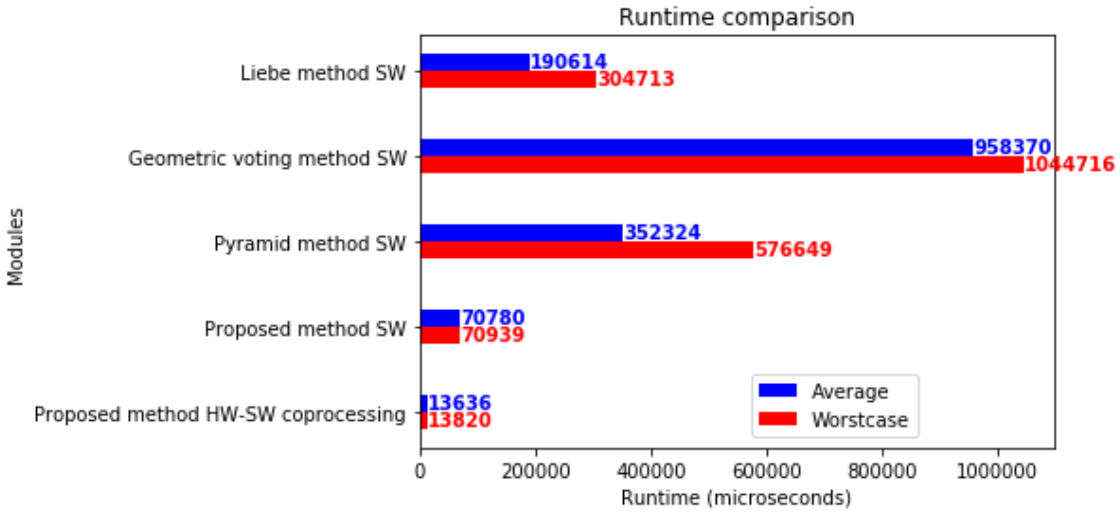


Figure 4.12: Runtime comparison

Chapter 5

Summary and Future Works

5.1 Summary

The purpose of this research topic is to implement a system evaluating a star tracking algorithm proposed by Mr. Minh Duc Pham[8, 12, 13], my senior and an ex post-graduated NTU student. In my thesis, I have summarized three traditional methods and one state-of-the-art star tracking method proposed in the literature review section. Then in the Hardware and Software Co-processing Implementation of the Algorithm section, I introduced the hardware system, deeply analyze the algorithm. I also proposed an adaptive algorithm based on the proposed algorithm that is suitable for the software-hardware co-processing implementation in the system. In the final section Experimental Results, I suggested my hardware architecture design for this project, carry out some experiments to evaluate the implementation and their results.

All of my work includes the simulation image generator, image datasets, hardware IP core design, system architecture, software algorithms and source code, results of the experiments, quick documentations are dedicated in this repository: https://github.com/dangkhoadl/Master-Thesis-Star-Tracking-System

5.2 Future Works

Currently, the system can not handle a high throughput image data streaming. In my experiments, I have to handle the images one-by-one to feed into the system.

Hence, I suggest implementing an input-processing module with can manage a large stream amount of image data(around 9.8 GBs) autonomously.

Moreover, the first dataset is generated by sky simulation with slightly normal distribution noise added, the second dataset is captured by a star tracker from the ground. In space, there are more noises affect the quality of the images from the electromagnetic field, Solar noise, Cosmic noise, etc. A noise cancellation module implemented in the PL would be needed.

Bibliography

[1] M D Samirbhai, S Chen, and K S Low. A hamming distance and spearman-correlation based star identification algorithm. *IEEE Transactions on Aerospace and Electronic Systems*, page 1, 2018.

[2] Scherr Timothy. Introduction to FPGA Design for Embedded Systems.

[3] Xilinx.com. *Spartan-3AN FPGA Family datasheet.* 2014.

[4] Louise Crockett, Ross Elliot, Martin Enderwitz, Bob Stewart, and David Northcote. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.* Strathclyde Academic Media, 2015.

[5] BBC Worldwide Ltd., Films for the Humanities Sciences Firm, and Films Media Group. In Orbit How Satellites Rule Our World, 2013.

[6] Wikipedia. Satellite.

[7] Walter Colitti, Kris Steenhaut, Nicolas Descouvemont, and Adam Dunkels. Satellite based wireless sensor networks: global scale sensing with nano- and pico-satellites., 2008.

[8] Minh Duc Pham. *Attitude determination system for nano-satelite.* PhD thesis, Nanyang Technological University, 2013.

[9] M D Shuster and S D Oh. Three-axis attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 4(1):70–77, 1981.

[10] James Richard Wertz. *Spacecraft attitude determination and control.* Astrophysics and space science library: v. 73. Dordrecht ; London : Kluwer Academic Publishers, c1978., 1978.

[11] Star trackers for attitude determination. *IEEE Aerospace and Electronic Systems Magazine, Aerospace and Electronic Systems Magazine, IEEE, IEEE Aerosp. Electron. Syst. Mag*, (6):10, 1995.

[12] M D Pham, K S Low, Shoushun Chen, and Y T Xing. A star pattern recognition algorithm for satellite attitude determination. In *ISIEA 2012 - 2012 IEEE Symposium on Industrial Electronics and Applications*, number ISIEA 2012 - 2012 IEEE Symposium on Industrial Electronics and Applications, pages 236–241, School of Electrical and Electronic Engineering, Nanyang Technological University, 2012.

[13] An Autonomous Star Recognition Algorithm with Optimized Database. *IEEE Transactions on Aerospace and Electronic Systems, Aerospace and Electronic Systems, IEEE Transactions on, IEEE Trans. Aerosp. Electron. Syst*, (3):1467, 2013.

[14] Steve B Howell. *Handbook of CCD astronomy.* Cambridge observing handbooks for research astronomers: 5. Cambridge ; New York : Cambridge University Press, 2006., 2006.

[15] Pattern recognition of star constellations for spacecraft applications. *IEEE Aerospace and Electronic Systems Magazine, Aerospace and Electronic Systems Magazine, IEEE, IEEE Aerosp. Electron. Syst. Mag*, (1):31, 1993.

[16] Accuracy performance of star trackers - A tutorial. *IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS*, 38(2):587–599.

[17] Micro APS based star tracker. *Proceedings, IEEE Aerospace Conference, Aerospace Conference Proceedings, 2002. IEEE, Aerospace conference*, page 5, 2002.

[18] Star tracker design considerations for the Europa Orbiter mission. *1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403), Aerospace Conference, 1999. Proceedings. 1999 IEEE*, page 67, 1999.

[19] Ying Dong, Fei Xing, and Zheng You. Brightness Independent 4-Star Matching Algorithm for Lost-in-Space 3-Axis Attitude Acquisition. *Tsinghua Science & Technology*, 11:543–548, 2006.

[20] D Mortari, John Junkins, and Malak Samaan. Lost-in-Space Pyramid Algorithm for Robust Star Pattern Recognition. 2001.

[21] D Mortari and Beny Neta. k-Vector Range Searching Technique. 105, 2000.

[22] D Mortari, Malak Samaan, Christian Bruccoleri, and John Junkins. The Pyramid Star Identification Technique. *NAVIGATION*, 51, 2004.

[23] Benjamin B. Spratling and D Mortari. A Survey on Star Identification Algorithms. *Algorithms*, 2, 2009.

[24] Crew, Vanderspek G. B., R., and Doty J. HETE Experience with the Pyramid Algorithm. *MIT Center for Space Research, Cambridge, MA, 02139 USA*, 2002.

[25] Geometric voting algorithm for star trackers. *IEEE Transactions on Aerospace and Electronic Systems, Aerospace and Electronic Systems, IEEE Transactions on, IEEE Trans. Aerosp. Electron. Syst*, (2), 2008.

[26] Andrew Moore. *FPGAs for Dummies*. 2014.

[27] Wikipedia. Semiconductor intellectual property core.

[28] Jens Rettkowski, Andrew Boutros, and Diana Göhringer. HW/SW Co-Design of the HOG algorithm on a Xilinx Zynq SoC. *Journal of Parallel & Distributed Computing*, 109:50–62, 2017.

[29] Tony Grant. *Xilinx Redefines Power, Performance, and Design Productivity with Three Innovative 28 nm FPGA Families: Virtex-7, Kintex-7, and Artix-7 Devices*. 2011.

[30] Donald Bailey, Christopher Johnston, and Ni Ma. Connected components analysis of streamed images, 2008.

[31] Michael J Klaiber, Yousef O Baroud, Sven Simon, and Donald G Bailey. A Resource-Efficient Hardware Architecture for Connected Component Analysis. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 26(7):1334–1349.

[32] J R Myers, C B Sande, A C Miller, W H Warren Jr, and D A Tracewell. SKY2000-master star catalog-star catalog database. *Bulletin of the American Astronomical Society*, 191(128.12), 1997.