Multiprocessor Architecture for Fuzzy-C-Mean Clustering Algorithm for Edge Computing

Author

Tauseef Mehmood

00000204018

Supervisor

Dr. Sajid Gul Khawaja

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

AUGUST, 2020

Multiprocessor Architecture for Fuzzy-C-Mean Clustering Algorithm for Edge Computing

Author

Tauseef Mehmood

00000204018

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Engineering

Thesis Supervisor

Dr. Sajid Gul Khawaja

Thesis Supervisor's Signature: _____

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

AUGUST 2020

Declaration

I certify that this research work titled "*Multiprocessor Architecture for Fuzzy-C-Mean Clustering Algorithm for Edge Computing"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Tauseef Mehmood

00000204018

Language Correctness Certificate

This thesis has been read by an English expert and is free of most typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

Tauseef Mehmood

00000204018

Signature of Supervisor

Dr. Sajid Gul Khawaja

Copyright Statement

Acknowledgements

I would like to thank Almighty Allah for helping me in completion of my Master's thesis. It was quite a strenuous task that could not have been completed without the help of Allah.

I am also thankful to my family for supporting me in every phase of the thesis every step of the way. Their support was what made me accomplish this herculean task. Without them, it would not have been possible.

Lastly, I would like to offer my sincere thanks to my incredible supervisors and the entire committee: Sir Usman Akram, Sir Sajid Gul Khawaja and Sir Arslan Shaukat. Without their support – in every sense of the word – and constant guidance, this thesis would never have been conceived or completed (and definitely not in time). I cannot thank them enough for their role in the completion of this thesis and report.

Dedicated to my family.

Abstract

Cloud-based systems have become very popular these days, which consists of a server and node devices. These node devices are called edge devices or edge of the system. With ever increasing size of data and complex algorithms, the load on servers have increased many folds. Hence there is a requirement of shifting this load to the edge devices. Shifting load from server to edge nodes is called edge computing. Now Fuzzy C-Means (FCM) is an unsupervised machine learning algorithm that is used for data clustering. It is also called soft k-means or soft clustering as each data point can belong to more than one cluster. In cloud-based applications it can affect the performance of server due to its complexity. In edge computing systems workload is put close to the edge where data is being created, this helps improve response time and save bandwidth. FCM can be incorporated in various smart nodes-based devices by assigning data to relative clusters. Image segmentation is one of such applications that can be implemented using FCM. In proposed design FCM is implemented using multicore architecture, which consists of P processing units called Tiles. Each Tile process a chunk of data in parallel and final results are calculated by sharing of data between the Tiles. This improves the overall performance of the system by speed up as compared to the traditional sequential architecture. The proposed architecture can be used to design a smart edge computing-based system with high performance and better throughput.

Key Words: *Fuzzy C-Mean, Clustering, Soft Clustering, Edge Computing, Smart Edge Node, Image Segmentation, Parallel Architecture, Homogeneous Architecture, FPGA, FCM, Verilog*

Table of Contents

List of Figures

List of Tables

Chapter 1 : INTRODUCTION

The modern technological revolution has profited human life in all fields of life, and the invention of computers is the panicle of the scientific age. In the start computers used to be bulky, hard to move, and weak in terms of computational power, but as time went by, they started to reduce in size and became more and more portable with greater processing speeds. This gave birth to the era of telecommunication via portable devices in the form of networks. Previously, computer networks faced many challenges, such as data accuracy, transmission speed, and security. However, one challenge that affected the network growth was the cost of data transmission. A lot of work has been done to reduce this cost and there are multiple solutions, however, one solution that can benefit the most in this regard is Edge Computing. In the past role of edge nodes in a cloud-based system was to collect and transmit the data to the main server, but after the invention of portable and smart processing devices processing load can be shifted from the server to edge devices. These devices can now process the input signal and generate preliminary data for further tasks.

Multiple applications can benefit from the above-mentioned model like autonomous vehicle systems, security solutions, and video conferencing. However, we have focused on Healthcare solutions. Medical imaging has become very useful in the early diagnosis of numerous diseases such as cancer, Alzheimer's disease, and hippocampal atrophy, etc. Image segmentation is used to slice the medical images into multiple portions to extract useful information and make the processing easier [1]. There are several image segmentation techniques but unsupervised machine learning algorithms have become popular for said process. Fuzzy C-Means Clustering (FCM) clustering algorithm and can be used to effectively segment monochrome medical images.

FCM is a soft clustering algorithm where a data member can belong to more than one cluster. Cluster boundary is determined based on membership values that are assigned to every data point for each cluster. We can easily control its fuzziness by tweaking one variable which is FCM fuzziness factor[2]. This makes it very suitable for image clustering applications where you can tune the cluster boundaries by regulating the fuzziness of the algorithm. Despite all its benefits, FCM is a computationally extensive algorithm and it consumes significant processing time to converge. Therefore, there is a need to implement FCM in such a manner that it can perform faster with minimum hardware resources, to make it suitable for node devices in edge computing.

Homogenous Multiprocessor System on Chip (MPSoC) provides us with the best hardware architecture to implement FCM. Identical Processing Entities (PE) can work in parallel to achieve faster convergence of the algorithm. Similarly, data sharing can be helpful with optimum hardware utilization.

We have implemented a parallel FCM algorithm for Image segmentation on an MPSoC architecture. We have achieved a significant boost in speed and reduction in hardware consumption compared to the original FCM. To get the best performance we have tried different variants of the algorithm. We have conducted the implementation process in different phases. In each phase, we altered one parameter of the algorithm and kept others constant. These parameters are the number of PE, number of clusters, and number of fractional bits, and resolution of the image. This research provides a comprehensive solution for implementation of FCM algorithm on a parallel MPSoC architecture.

1.1 Motivation

With the arrival of new powerful sensors and portable processing devices, the amount of data generated at edge nodes and cloud-based systems has hit the roof. This has contributed to the ever-present problem of data transmission costs in network applications. Thus, there is a requirement for a smart solution for this problem. Machine learning algorithms provide an opportunity to solve the said problem by processing the data at the edge node and only transmit useful data for further processing. Successful implementation of smart algorithms on edge devices can significantly cut back on the operational cost of cloud systems. This can also contribute to increasing the reliability and efficiency of the systems as well.

1.2 Problem Statement

Performance enhancement of Edge Nodes by implementation of Parallel Architecture for clustering algorithms on MPSoC.

1.3 Aims and Objectives

Major objectives of the research are as follow:

- To implement original FCM algorithm on a reconfigurable hardware.
- Implementation of proposed parallel FCM algorithm on reconfigurable hardware.
- Compute performance enhancement in terms of clock speed and area utilization.

- Establish the working similarity of both algorithms by comparison of results.
- To test different versions of proposed parallel FCM by changing key parameters.

1.4 Structure of Thesis

This work is structured as follows:

Chapter 2 covers the basics of Edge computing and image segmentation.
Chapter 3 describes the literature and the important work done by other researchers in the field implementation of machine learning algorithms for edge devices.
Chapter 4 consists gives and overview of original and proposed FCM algorithms, and working of both parts has been explained by examples. In addition to that pseudo code of original FCM is also provided for better understanding.
Chapter 5 gives us complete understanding of MPSoC implementation of proposed FCM on reconfigurable hardware with complete description of all components.
Chapter 6 provides the conclusion of the thesis and presents the way forward.

Chapter 2 : EDGE COMPUTING AND IMAGE SEGMENTATION

Edge computing is an evolving technology that facilitates the processing of a large amount of data gathered by nodes devices in a networking application[3]. In a server-based network, devices or sensors collects information from the connected environment and share the said information to a central processing unit using a unified protocol. Previously cloud computing was considered the ultimate approach for fulfilling the requirements of network applications. However, cloud computing had its own limitations e.g., bandwidth restrictions, high latency, server downtime/server shop closed, etc.

The challenges of cloud computing have paved the way for Edge computing that allows the applications/data processing to run at network nodes where the data is being produced or consumed instead of sending the data/information to a geographically distantly located central processing node. This is done so that data, especially real-time data does not suffer the latency issues caused by bandwidth limitations. Edge computing can also be beneficial in case of Internet of Things (IOT), as it addresses some of the basic issues faced by IOT.

2.1 Challenges in Edge Computing

The challenges of cloud computing were related to the main data server; however, edge computing challenges are mostly related to the management of the edge nodes. By allowing critical data and applications for processing of the data at the edge nodes introduce performance and scale challenges to be addressed.

2.1.1 Hardware Limitations

For an edge device to process, the data customization of the edge hardware may be required. However, an edge node may not be able to fit as much hardware as a full central data processing unit thus, keeping the hardware footprint of an edge node small/reduced may be a challenge[4].

The solution for the hardware footprint constraint can be solved using a software approach/solution.

2.1.2 Operational limitations

Requirement of highly trained technical staff for installation and maintenance of edge devices on remote locations is one of the biggest challenges for edge computing[4].

However, it can be solved by training of the personnel on regular basis and controlling the edge devices remotely.

2.1.3 Security

Placement of expensive hardware on remote locations is a security challenge that edge computing faces. Both hardware and data security is point of concern for the security of edge systems[5].

The solution for this challenge is the implementation of shielding measures for both hardware and data protection, for example automated threat alert systems and complete quick response mechanisms[5].

2.1.4 Diverseness

Possibility of edge computing usage in variety of fields also creates a challenge of heterogeneity, due to which communication between various systems becomes costly and cumbersome. This puts added pressure on scalability of the system.

We can cater this problem by establishing unified protocols across all fields of potential edge computing applications. It will ensure a smoother integration of systems under the umbrella of edge computing[5].

2.2 Benefits of Edge Computing

There are a number of benefits of edge computing that can actually make a difference as compared to the cloud computing. Some of these advantages are produced below.

2.2.1 Rapid response

One of the most beneficial feature of edge computing is real time response of the system, as most of the necessary handling of data is being done close to the end user. Therefore these devices can now perform sophisticated operations without the help of central node, hence creating a rapid and efficient response to the user query[3].

2.2.2 Cost effective

As processing work is now shared between the edge devices and the main server in edge computing, therefore a large amount of data is not required to transmit over the network. This helps in reducing the network traffic, bandwidth and storage costs.

2.2.3 Improved data security

Security of data is one of the biggest challenges in edge computing, but if handled correctly, it can become a big advantage. The key vulnerability regarding security in cloud-based applications is the transmission of data over the network. In edge computing most of the data handling is done at node, therefore reduced amount of data transmitted decreases the security risk as well. In addition to this encryption of small data is also relatively easy task[3].

2.2.4 Energy efficient

According to latest research the amount of energy used by data centers is on the rise, and it is expected to increase by three times in the next decade[6]. Hence there is a requirement to select energy efficient technologies. Edge computing can help in this regard as well. As most of the computational tasks are done at the node level therefore energy at data center is also reduced by many folds.

2.2.5 Low latency

Edge computing offers computational resources closer to the users, resulting in delay free services[7]. This helps users to perform their computationally intensive tasks on powerful edge devices[7].

2.3 Examples of Edge Computing

2.3.1 Autonomous Vehicles

According to a latest research a number of car manufacturers are working to introduce self-driving cars in near future, and it will be experimented first in the shared ride services[8]. Edge computing is going to play a major role in this regard. Route controlling can be done from a server but tasks like path determination, steering, obstacle avoidance and fuel management can only be done on the edge node. Therefore, autonomous vehicles are one of the key areas in the field of edge computing.

2.3.2 Health care devices

Area of health care devices can also benefit from edge computing. Connected medical instruments and machines can share work load of the main server to make an efficient Hospital management system. Some of health care facilities are placed at remote locations, therefore cannot afford a down time. Smart edge devices can fill this gap by efficiently managing the resources offline[9].

In addition to this a number of medical diagnosis is done on the basis of medical imaging. Smart devices can diagnose a disease without sending a huge image to the server.

2.3.3 Security Solutions

One of the most important application of the edge computing is security setups. Smart edge devices can be used for biometric identification, image processing for threat detection and feature extraction.

2.3.4 IoT Applications

Edge computing has its main roots in IoT applications as widely spaced sensors can share the load of main controller. Industrial IoT is a big candidate for edge computing boom and applications such as wind power mils, oil rigs, HVAC systems and production control systems can benefit from it [4].

2.3.5 Edge Computing and 5G

Mobile phone carriers across the world are installing 5G infrastructure, which is equipped with the power of high bandwidth and low latency. Instead of transporting all of the data to the cloud many operators are working on edge computing technology in 5G infrastructure. This can help in offering real time reliable services to mobile users [10].

2.4 Clustering in Machine Learning

Clustering is a field of unsupervised machine learning in which data is divided into smaller portions and members in each portion has similar properties. In supervised machine learning classification is done on the basis of a boundary, which is defined via training of algorithm by providing the know labels. However, in unsupervised machine learning the algorithm is allowed to discover patterns in data without any external help. It is also known as cluster analysis. There are a number of models that are used for clustering purposes. All of these models have several algorithms. Some of the popular models are “Connectivity-based clustering (hierarchical clustering)", "Centroid-based clustering", "Distribution-based clustering", "Density-based clustering" and "Grid-based clustering" [11]. FCM algorithms belongs to "Centroid-based clustering" algorithm model [11].

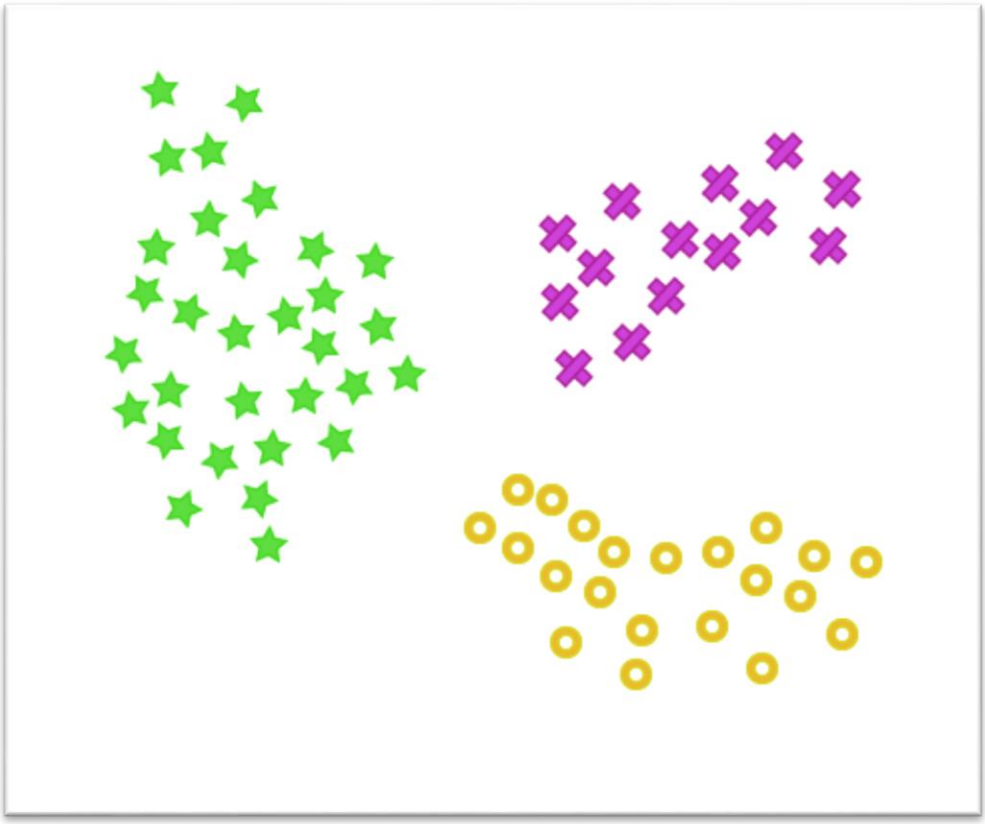


Figure 2-1: Cluster analysis

2.5 Image Segmentation and Clustering

Image segmentation is a technique that is used to partition an image into smaller sets [1]. Like pixels are grouped in each set. The basic purpose of this technique is to extract boundaries and objects from an image, and this is done by assigning a label to every pixel in the image. Pixels with identical labels lie in a segment[1]. It can be used on color as well as greyscale images. In this research we have used it on greyscale images only.

Segmentation is a very important task in image processing as it separates the contents of an image from its background and form one another [12]. This separation of image contents is a very important step in tasks like object recognition and object tracking and other high level tasks [12].

Image segmentation can be divided into two main classes. Traditional Image Processing Segmentation and Machine Learning bases Segmentation. In traditional segmentation the image contents are separated with the help of various techniques such as Thresholding, where pixels of an image are replace with white or black pixels depending upon their level [1]. Where as in machine learning segmentation clustering algorithms are used to separate contents of an image from one another, as clustering combines together similar elements in a group.

2.6 Multiprocessor Architecture (MPSoC)

Multiprocessor System on Chip (MPSoC) is part Very Large Scale Integration (VLSI) systems which has become very popular in the last couple of decades [13]. It is a System on Chip (SOC) which is comprised of multiple processing units. Where SOC is an integrated circuit that consists of almost all components of a computer such as memory, processor and input/output devices interface [14]. There are mainly two types of multiprocessor architectures, i.e. heterogeneous and homogeneous MPSoC. The homogeneous systems are comprised of multiple similar processing units and work together to achieve better performance, whereas heterogeneous MPSoC consists of multiple processing units that are different form one another. Both of these types of MPSoCs have their own benefits and either of these designs is opted on the basis of its advantages over other for a specific application.

2.7 Fuzzy C-Mean (FCM) Clustering

FCM is a machine learning algorithm and it has gained a lot of popularity in the past two decades because of its usefulness in multiple fields of engineering. Its main application fields include Neural Network, Clustering and Classification, Image Analysis and Structural Analysis of Algorithms [15]. Its other fields are Video Segmentation Sonar returns, Speech recognition and ECG arrhythmias. FCM has seen numerous advancements and upgrades in past to improve its performance. L. Jain et al. has compiled a comprehensive review on FCM development, its applications and work done [15]. Its main structure and equations with an example are produced in Chapter 4 in detail.

Chapter 3 : LITERATURE REVIEW

A lot of work has been done in the field of machine learning and MPSoC. In this section we will briefly discuss the previous work done and locate possible research gap that we are trying to fill.

3.1 Clustering Algorithms Implementation on Hardware

The idea of implementing clustering algorithms on reconfigurable hardware such as Field-Programmable Gate Array (FPGA) is not new. People have designed hardware architectures to get the best of cluster analysis algorithms. As clustering processes consumes a lot of processing time therefore speed is the prime motivator for hardware implementation of such algorithms. In the following section an overview of clustering algorithms on hardware is provided with focus on performance enhancement. In hardware implementation the goal of high speed has been achieved via many techniques such as, pipelined circuits [16]–[19] and multiprocessor architectures [20]. The purpose of all these implementations is to get the performance in terms of speed and area efficiency.

3.1.1 K-mean clustering algorithm on hardware.

K-mean is one of the most popular centroid oriented clustering algorithms. "In k-mean, each cluster is represented by the mean value of objects in the cluster" [21]. It can be used to find unseen patterns in business data sets such as market trends etc. There are a number of K-mean hardware implementations such as Canilho et al. implemented "an embedded hardware/software architecture targeting the K-means clustering algorithm" [22]. The lowest hardware variant of their design performs 10 times better than the software counterpart, and the hardware designed can be scaled up to achieve even more speed [19]. Similarly in another implementation of K-mean on FPGA Kutty et al. have achieved three times faster-operating frequency than previous works [19]. They have attained these results with the help of both pipelining and parallelism [19].

Biomedical applications have a great deal of potential for benefiting from the k-mean clustering, and a number of researchers have put their efforts to benefit from this new field of science. For example many hospitals in the world generate a large amount of medical information data in the form of images such as X-ray, MRI and ultrasound tests [23]. Machine learning algorithms are becoming very popular to extract useful information for early diagnosis and physician’s help [23]. Neshatpour et al. implemented a heterogeneous CPU+FPGA K-

mean architecture for such image processing applications in the field of biomedical informatics [23]. Their design shows 72 times speed up for large images because the computational intensive operations are implemented on FPGA [23].

Similarly "Microarray is a technique used by biologists to perform many genome experiments simultaneously, which produces very large datasets" [24]. Hanaa et al. implemented 5 different parallel K-mean cores on FPGA to achieve 10.3 times the speed-up as compare to the software implementation [24].

3.1.2 Mean Shift algorithm hardware implementation

Mean shift is another machine learning algorithm that is used for clustering. It converges to extract the maxima of the cluster. Like K-mean it is also computationally extensive and it can consume a large amount of time to converge; therefore, a number of people have attempted to decrease its convergence time with the help of modern hardware architectures.

D. Trieu et al. implemented mean shift algorithm on FPGA for color image segmentation to reduce its computational cost [25]. They have accelerated the architecture by using a cache for parallel processing [25]. Similarly S. Craciun et al. also worked on the mean shift for image segmentation, and tried to improve its execution time [12]. They have tested their deigned on multiple hardware and achieve a low power architecture with better performance. In another example amna et al. proposed a simpler version of mean shift version with parallel architecture and better performance [20]. They have achieved impressive speed up with fix point designs with and without fractional bits.

3.2 FCM Implementation on Hardware

FCM algorithm has captured the attention of researchers in the past. People have benefited from this algorithm in various applications and domains. It can provide an alternate solution to the k-mean clustering algorithm.

First of all W. Hwang et al. implemented the traditional version of FCM on FPGA and achieved the segmentation success rate of 94% with maximum 10 number of classes [26]. Their architecture is 10 times faster than its software counterpart [26]. They have tested their architecture compared to basic software version and a fast software implementation [27]. Their design is an evidence of the performance superiority of hardware over software.

Similarly in another research Yao-Jung Yeh et al. implemented a pipelined version of FCM with centroid and membership coefficient sharing, and achieved significant improvement in Area and computational performance [16]. FCM require large storage due to size of

membership matrix and it increases with the number of classes. They managed to exclude larger storage requirement by combining usual iterative process of matrix computation [16]. This shows that pipelined architectures serve the purpose of area management and performance enhancement. In another article Chien-Min Ou et al. used the same technique and reduced the requirement of large storage by combining the centroid calculation [28].

When it comes to the handling of large data, FCM performs poorly due to the requirement of large storage for partition matrix. The scale of matrix increases with the amount of data and number of clusters. John F. Kolen and Tim Hutcheson devised a new method to eliminate the requirement of large storage [27]. They have implemented a software version of FCM and combined the two updates of cluster centers into one. This also removes the overhead of accessing the large storage repeatedly, hence increasing the speed by many folds.

3.3 Parallel architecture MPSoC implementation for Clustering algorithms

There are numerous examples of MPSoC architectures that are designed to benefit from parallelism. Homogenous Tile architecture is a concept of dividing the work load among identical processing units to enhance the speed of the architecture. The data is shared between the Tiles through a shared bus. This concept has been implemented in the past to achieve improved speeds on K-Mean clustering algorithm [29], Mean-Shift clustering algorithm [20] and Partition around Medoids (PAM) algorithm [30]. In all of above work the researchers have improved the performance of respective algorithms by many folds and this method has considerable potential to be implemented on other algorithms.

3.4 Research Gap

Keeping in view the discussion in the above section the Homogenous Tile architecture has not been implemented on FCM algorithms, therefore there is an opportunity to exploit the benefits of this architecture even more with FCM. It is best suited for Tiled architecture design due to its similarity with K-Mean clustering. Furthermore, edge computing field require fast, efficient and intelligent devices, hence Parallel MPSoC architectures offer a unique solution to this problem by shifting the load to multiple processors that can work together to achieve improved speed. Machine learning applications such as Image Segmentation and Video Segmentation can take advantage of real time processing on edge devices. This can help reduce the cost and improve performance of the system.

Chapter 4 : Proposed Parallel Algorithm

4.1 FCM Algorithm

FCM algorithms was originally proposed by J.C. Dunn in 1973[31] and later it was perfected by J.C. Bezdek in 1981[32]. It can be divided into three main steps.

- Center's calculation
- Partition Matrix (PM) computation
- Object Function (OF) determination.

In the first step cluster centers are determined from the current PM and original dataset. After that the PM is updated with the mathematical computation of cluster centers and original data. In the third step object function is calculated with partition matrix, centers and dataset. These steps are best described in the form of following equations. The termination criteria are decided on the error value, which is calculated by subtracting current objective function from previous one. The mathematical model of FCM can be described with the help of following three equations.

$$c_j = \frac{\sum_{i=1}^{D}\left(u_{ij}^{m}.x_i\right)}{\sum_{i=1}^{D} u_{ij}^{m}} \quad (1)$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{N}\left(\frac{|x_i - c_j|}{|x_i - c_k|}\right)^{\frac{2}{m-1}}} \quad (2)$$

$$J_m = \sum_{i=1}^{D}\sum_{j=1}^{N} u_{ij}^{m} \left\|x_i - c_j\right\|^2 \quad (3)$$

Where m is the fuzziness coefficient and it controls the level of fuzziness in the algorithm, u_{ij} is the PM where i indicate the index of the data and j denotes the cluster number. x_i is the i^{th} element in the input dataset and c_j represents the j^{th} cluster center. Finally, J_m represents current object function.

4.1.1 FCM Pseudo Code

Working of FCM can also be illustrated in the form of following pseudo code.

FCM Algorithm

Input: N (No of Clusters), x (Dataset with D elements),
Outputs: C (Cluster Centers), u (Partition Matrix)
Initialization:

```
Fix m, 1 < m < ∞, (e.g., m = 2);
Fix ε, (e.g., ε = 0.0001);
Fix maxlerations, (e.g., maxlerations=200);
Fix Jm, (e.g., =0);
Randomly initialize uij
Begin
    for t = 1 to maxIterations

        for j=1 to N
            for i=1 to D
                Calculate Cj using Eq. (1);
            end
        end

        for i=1 to D
            for j=1 to N
                Calculate uij using Eq. (2);
            end
        end

        for j=1 to N
            for i=1 to D
                Calculate Jm using Eq. (3);
            end
        end

        if (abs (J tm - J t-1m) << ε) then
            break;
        else
            J t-1m = J tm;
        end if

    end
End
```

4.1.2 Example of original FCM algorithm

Working of FCM algorithm can be understood with the help of an example. Let us consider an example with dataset of 16 elements, which will be used to calculate membership matrix and centers for two clusters. Table 4-1 shows the input data elements.

Input Dataset
13, 5, 90, 84, 45, 209, 12, 34, 22, 222, 179, 250, 44, 3, 150, 120

Table 4-1: Input dataset for Original FCM example

x_i	u_{ij}		$(u_{ij}^{\wedge m} * x_i)$		$\sum (u_{ij}^{\wedge m} * x_i)$		$\sum u_{ij}^{\wedge m}$		C_j	
	u_{i1}	u_{i2}	$(u_{i1}^{\wedge m} * x_i)$	$(u_{i2}^{\wedge m} * x_i)$	$\sum(u_{i1}^{\wedge m} * x_i)$	$\sum(u_{i2}^{\wedge m} * x_i)$	$\sum u_1^{\wedge m}$	$\sum u_{i2}^{\wedge m}$	C_1	C_2
13	0.58813	0.11669	4.49667	0.177026						
5	0.89771	0.75128	4.02944	2.822113						
90	0.89153	0.23921	71.5344	5.150281						
84	0.81583	0.2548	55.9096	5.453792	526.59845	684.65903	6.5038	5.96983	80.9678	114.686
.	.	.	.	.						
.	.	.	.	.						
150	0.77524	0.67941	90.1506	69.23991						
120	0.03666	0.65078	0.16131	50.82267						

Table 4-2: Center's calculation

x_i	$\lvert x_i - C_j \rvert$	$\lvert x_i - C_k \rvert$	$\left(\frac{\lvert x_i - C_1 \rvert}{\lvert x_i - C_1 \rvert}\right)^{\frac{2}{m-1}}$	$\lvert x_i - C_j \rvert$	$\lvert x_i - C_k \rvert$	$\left(\frac{\lvert x_i - C_1 \rvert}{\lvert x_i - C_2 \rvert}\right)^{\frac{2}{m-1}}$	$\sum_{k=1}^{C}\left(\frac{\lvert x_i - C_1 \rvert}{\lvert x_i - C_k \rvert}\right)^{\frac{2}{m-1}}$	$unew_{ij}$
	j=1	k=1		j=1	k=2			J=1
13	67.96775	67.96775	1	67.9678	101.6864	0.446765837	1.446765837	0.69119
5	75.96775	75.96775	1	75.9678	109.6864	0.479681404	1.479681404	0.67582
90	9.03225	9.03225	1	9.03225	24.68641	0.133867769	1.133867769	0.88193
84	3.03225	3.03225	1	3.03225	30.68641	0.009764226	1.009764226	0.99033
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
150	69.03225	69.03225	1	69.0323	45.65416	2.286353319	3.286353319	0.2074
120	39.03225	39.03225	1	39.0323	75.65416	0.266183754	1.266183754	0.01819

Table 4-3: Membership Matrix (PM) calculation

Table 4-2 represents the first step of FCM algorithm where calculation of cluster centers is done with the help of equation (1).

$\|x_i - C_1\|^2$	$\|x_i - C_1\|^2 \times u_{i1}$	$\|x_i - C_2\|^2$	$\|x_i - C_1\|^2 \times u_{i2}$	$\sum\|x_i - C_j\|^2 \times u_{ij}$	J_m
4619.61584	1597.91513	10340.1262	140.805923	1738.721053	
5771.09994	4650.87137	12031.1088	6790.6308	11441.50217	
81.5814326	64.8431287	609.418916	34.874213	99.7173417	
9.19450399	6.11977717	941.655855	61.138046	67.25782317	90601.42843
.	.	.	.	.	
.	.	.	.	.	
4765.45071	2864.05802	1247.04952	575.63739	3439.69541	
1523.51607	2.04801903	28.234221	11.957822	14.00584103	

Table 4-4: Object Function calculation

4.2 Proposed Parallel FCM Algorithm

Our proposed parallel algorithm is inspired by the famous divide and conquer method, where we divide that dataset into smaller slices. Each data slice is passed to homogenous Processing Entities (PE's) of FCM algorithm. These PE's work in parallel locally to compute local results. These results are then shared via a common bus to each other to compute the global results.

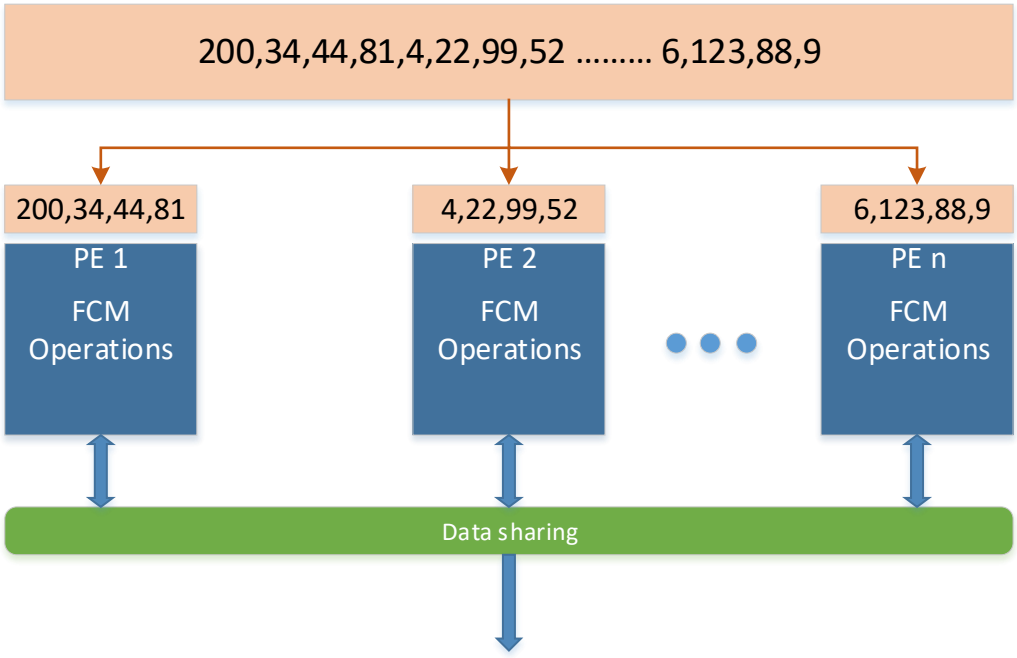


Figure 4-1: Working of a Parallel FCM Algorithm

FCM main equations are restructured to describe the proposed parallel architecture. First of all, equation (1) is divided into three equations splitting the Numerator (T_{pj}) and Denominator (B_{pj}) into separate equations (4b) and (4c), where p denotes the Processing Entity (PE) number and j represents the cluster index. Like original FCM equations i denotes the data point index. Final values of cluster centers are calculated with equation (4a).

$$c_j = \frac{\sum_{p=1}^{P} T_{pj}}{\sum_{p=1}^{P} B_{pj}} \qquad (4a)$$

$$T_{pj} = \sum_{i=1}^{F} \left(u_{pij}^{m} \,.\, x_i\right) \qquad (4b)$$

$$B_{pj} = \sum_{i=1}^{F} u_{pij}^{m} \qquad (4c)$$

Secondly the Partition Matrix is divided into sub matrices and final membership values are determined by appending all of the sub matrices as membership values can be calculated locally.

$$u_{ij} = \left(u_{1ij} | \, u_{2ij} \, | \ldots | u_{Pij} \right) \qquad (5a)$$

$$u_{pij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{|x_i - c_j|}{|x_i - c_k|} \right)^{\frac{2}{m-1}}} \qquad (5b)$$

Finally, Object Function is calculated by summing all the local values computed at PE level from equation (6a). Local OF are calculated with equation (6b)

$$J_m = \sum_{p=1}^{P} J_{pm} \qquad (6a)$$

$$J_{pm} = \sum_{i=1}^{F} \sum_{j=1}^{N} u_{pij}^{m} \,.\, \left\| x_i - c_j \right\|^2 \qquad (6b)$$

4.2.1 Working Example of Proposed Algorithm

Proposed FCM algorithm can best be understood with the help of an example. Let us consider the same dataset from original FCM example and solve one iteration for better understanding and validation of function similarity. Figure 4.2 (a) shows the same dataset that is used in section 4.2.2 and Figure 4.2 (b) shows equal division of data among the processing tiles.

Data
13, 5, 90, 84, 45, 209, 12, 34, 22, 222, 179, 250, 44, 3, 150, 120

(a)

PE 1 Data	PE 2 Data	PE 3 Data	PE 4 Data
13	5	90	84
45	209	12	34
22	222	179	250
44	3	150	120

(b)

Figure 4-2: (a) Input data set for example. (b) Equal division of data among PE

As we have discussed earlier that FCM algorithm has three main operations. Figure 4.3(a) shows the step-1 of first phase of FCM, which is center calculation.

(a) Step 1

PE 1 (p = 1)

x_i	u_{pij}		T_{pj}		B_{pj}	
	u_{1i1}	u_{1i2}	T_{11}	T_{12}	B_{11}	B_{12}
13	0.34589	0.01361				
45	0.00128	0.73552	41.0546	55.68035	1.39318	1.55498
22	0.4329	0.59328				
44	0.61309	0.21255				

PE 3(p = 3)

x_i	u_{pij}		T_{pj}		B_{pj}	
	U_{3i1}	U_{3i2}	T_{31}	T_{32}	B_{31}	B_{32}
90	0.79482	0.05722	176.67988	149.5041	1.61338	1.23279
12	0.14339	0.31549				
179	0.07415	0.39848	41.0546	55.68035	1.39318	1.55498
150	0.601	0.46159				

PE 2(p =2)

x_i	U_{pij}		T_{pj}		B_{pj}	
	U_{2i1}	U_{2i2}	T_{21}	T_{22}	B_{21}	B_{22}
5	0.80588	0.56442	114.55338	245.92218	2.04505	1.95688
209	0.47852	0.90208				
222	0.03757	0.24243	41.0546	55.68035	1.39318	1.55498
3	0.72305	0.24794				

PE 4(p = 4)

x_i	u_{pij}		T_{pj}		B_{pj}	
	U_{4i1}	U_{4i2}	T_{41}	T_{42}	B_{41}	B_{42}
84	0.66559	0.06492	194.31058	233.55238	1.45218	1.22516
34	0.26885	0.03196				
250	0.51639	0.70475	41.0546	55.68035	1.39318	1.55498
120	0.00134	0.42352				

(b) Step 2

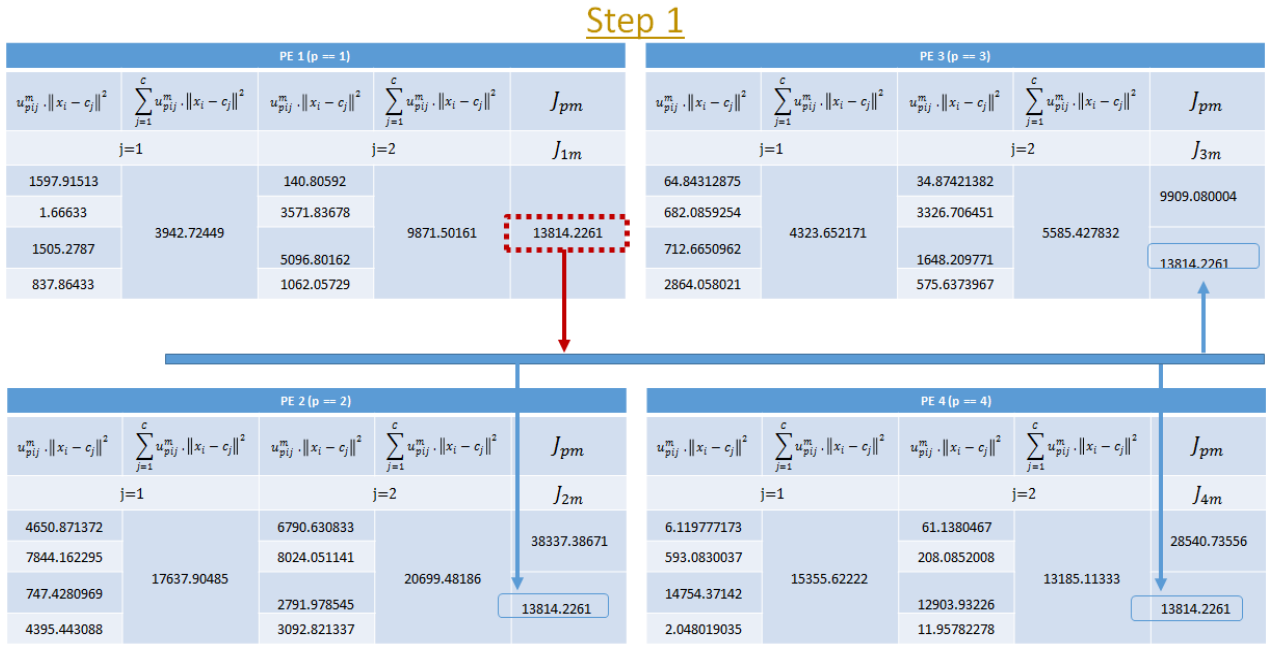
PE 1 (p == 1)

Source	T_{pj}		B_{pj}		$\sum_{p=1}^{P} T_{pj}$		$\sum_{p=1}^{P} B_{pj}$		c_j	
	T_{11}	T_{12}	B_{11}	B_{12}	$\sum_{p=1}^{P} T_{p1}$	$\sum_{p=1}^{P} T_{p2}$	$\sum_{p=1}^{P} B_{p1}$	$\sum_{p=1}^{P} B_{p2}$	c_1	c_2
PE 1	41.0546	55.68035	1.39318	1.55498	526.59844	684.65901	6.50379	5.96981	80.9679	114.6869
PE 2	114.55338	245.92218	2.04505	1.95688						
PE 3	176.67988	149.5041	1.61338	1.23279						
PE 4	194.31058	233.55238	1.45218	1.22516						

Figure 4-3: (a) Step 1 of Centers Calculation. (b) Step 2 of centers calculation

Tile 1 (p = 1), Cluster 1 (j = 1)								
xi	$\lvert x_i - c_j \rvert$ j=1	$\lvert x_i - c_k \rvert$ k=1	$\left(\frac{\lvert x_i - c_j \rvert}{\lvert x_i - c_k \rvert}\right)^{\frac{2}{m-1}}$	$\lvert x_i - c_j \rvert$ j=1	$\lvert x_i - c_k \rvert$ k=2	$\left(\frac{\lvert x_i - c_j \rvert}{\lvert x_i - c_k \rvert}\right)^{\frac{2}{m-1}}$	$\sum_{k=1}^{C}\left(\frac{\lvert x_i - c_j \rvert}{\lvert x_i - c_k \rvert}\right)^{\frac{2}{m-1}}$	u_{pij} u_{1i1}
13	67.96775	67.96775	1	67.96775	101.6864	0.446765837	1.446765837	0.691197
45	35.967.75	35.96775	1	35.96775	69.68641	0.266397619	1.266397619	0.789641
22	58.96775	58.96775	1	58.96775	92.68641	0.404759445	1.404759445	0.711866
44	36.96775	36.96775	1	36.96775	70.68641	0.273510615	1.273510615	0.785231

Figure 4-4: Partition Matrix determination for single Tile, single cluster



Step 2

PE 1 (p == 1)		
Source	J_{pm}	J_m
PE 1	13814.2261	90601.42837
PE 2	38337.38671	
PE 3	9909.080004	
PE 4	28540.73556	

Figure 4-5 Object Function calculation

4.2.2 Pseudo Code

Working of proposed FCM algorithm is described in the form of following pseudo code.

Proposed Parallel FCM Algorithm

Input: C (No of Clusters), x (Dataset with D elements), P (No of PE e.g P=8)
Outputs: C (Cluster Centers), u (Partition Matrix)
Initialization:

Fix m, $1 < m < \infty$, (e.g., $m = 2$);
Fix ε, (e.g., $\varepsilon = 0.0001$);

Fix maxIerations, (e.g., maxIerations=200);
Fix J_m, (e.g., =0);
M = Length(x), (e.g., M= 380928)
F = M/P; (e.g L=47616)
Randomly initialize u_{pij}

Begin

```
for t = 1 to maxIterations

    for p=1 to P
        for j=1 to N
            for i=1 to F
                Calculate Tpj using Eq. (4b);

                Calculate Bpj using Eq. (4c);
            end

        end
    end
    for j=1 to N
        for p=1 to P
            Calculate Cj using Eq. (4a);
        end
    end

    for p=1 to P
        for i=1 to F
            for j=1 to N
                Calculate upij using Eq. (5b);
            end
        end
    calculate uij using eq. (5a);

    end

    for p=1 to P
```

```
                for j=1 to N

                        for i=1 to F
                                Calculate Jpm using Eq. (6b);
                        end

                end
                Jm = Jm + Jpm;
        End

        if (abs (J tm - J t-1m) << ε) then
                break;
        else

                J t-1m = J tm;
        end if

    end
End
```

Chapter 5 Proposed MPSoC Architecture

The MPSoC architecture of proposed FCM is consists of the three main units. The communication, processing entities and main controller. These units work together to create a homogeneous parallel processing architecture dedicated to implement the FCM algorithms. Visual depiction of the whole architecture is shown in the following figure.

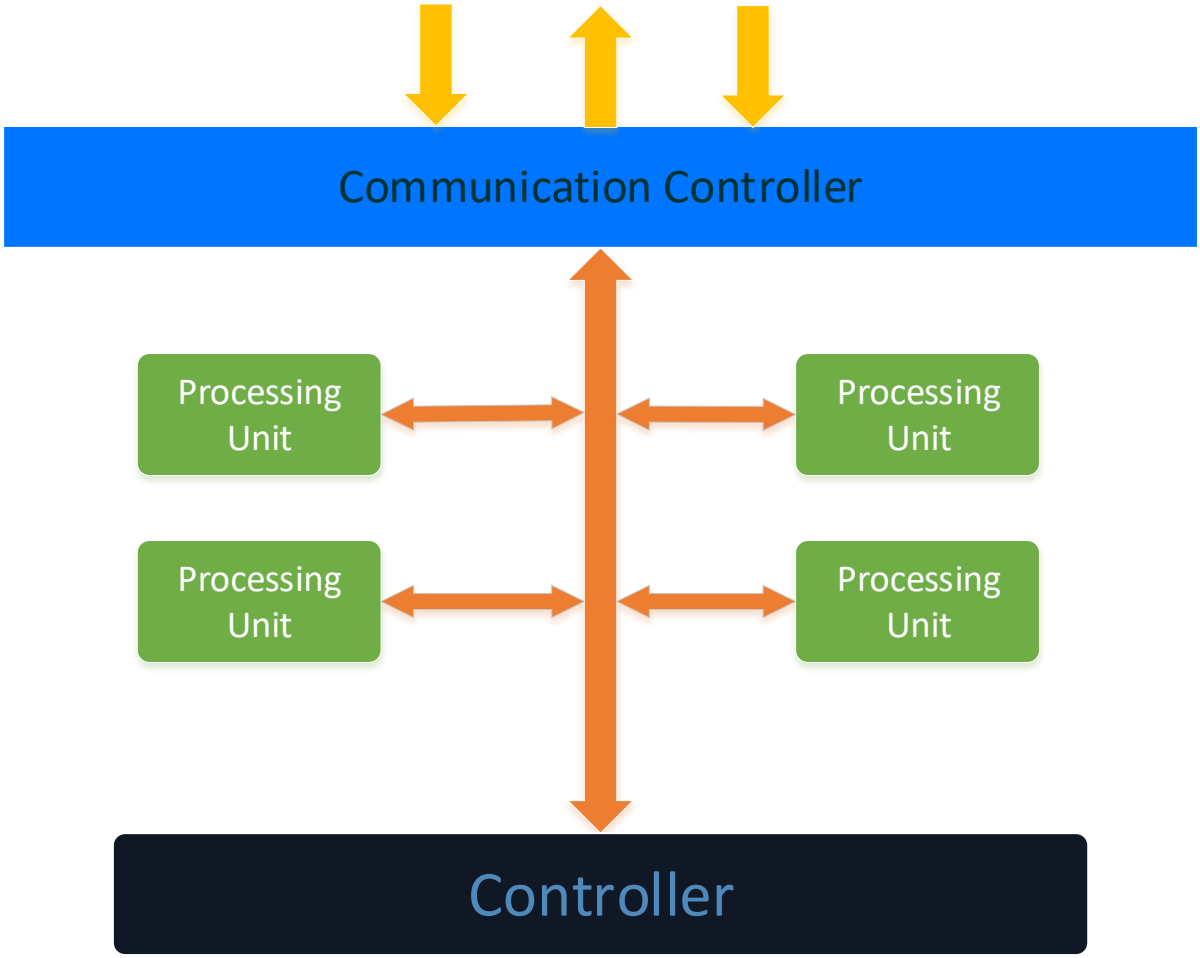


Figure 5-1: Visual depiction of proposed MPSoC architecture

5.1 Communication Controller

The communication controller is responsible for handling all of the information entering and leaving the architecture. It is the only interface to communicate with outside world. Its main objective is to pass on the input data to the system for image segmentation through clustering. The main inputs of the commination controller are produced below. Other inputs include number of clusters C, number of maximum iterations, minimum allowable error and a start

processing bit.
Similarly, the main output of the controller is the final state of the membership matrix as it defines the boundary of the clustering process. Its size is dependent on the number of clusters as it holds the membership value of every data element for every cluster. Other outputs include the stop flag to indicate completion of clustering process and number of iterations used to converge.

5.2 Main Controller

Main controller of our proposed architecture is responsible for implementation of the FCM algorithm in parallel. It controls the execution and flow of data from various entities of the architecture and ensures integrity of the data. Its main functions include generating control signals for the processing entities and computing the end criteria.

5.3 Design of a Single Tile (PE)

The parallelism offered in the proposed architecture is achieved due to processing entities of the architecture. They play a vital role in performance enhancement of the FCM algorithm, as they are mini FCM sequential machine running in parallel to achieve higher throughput.
Each Tile is composed the following sections.

- Tile controller
- Tile memory
- Partition matrix memory
- Center's calculation modules
- Partition matrix and object function modules

A tile processes its local part of data and create local results, which are shared with other tiles to compute the global results as explained in section 4.2.1. Communication between these tiles is controlled by the main controller of the architecture. Each tile has its own dedicated memory to store local data and other important variables required for the complete execution of the FCM algorithm.
Portion of partition matrix is stored and calculated locally and it is not shared among the tiles. This is because it is not used to compute any parameter globally. It is just a membership of the data elements which is updated locally. The local object function and local centers are computed with the help of local partition matrix independent of the local partition matrix in other tiles. Figure 5-1 shows the inner structure of the tile.

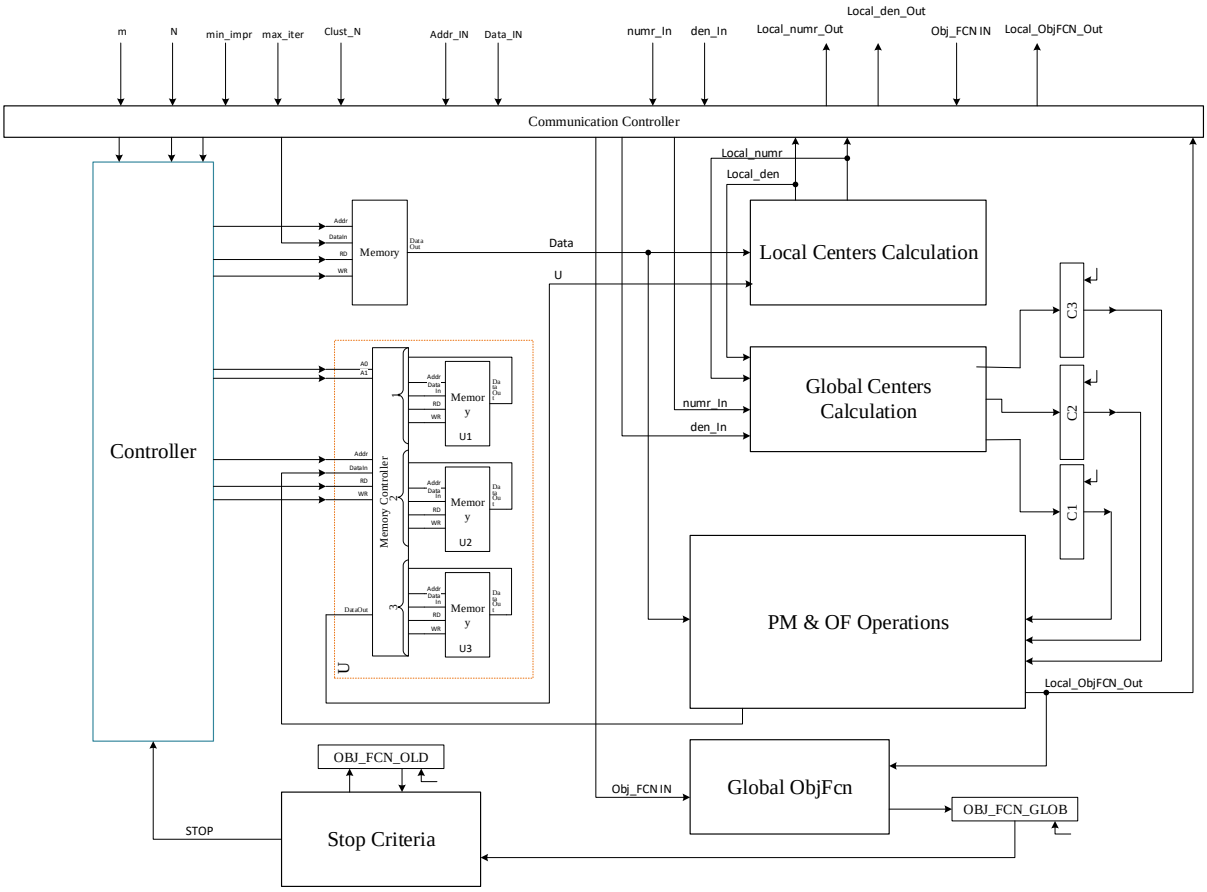


Figure 5-2: Inner architecture of a processing entity (PE)

5.4 Design of a single Tile

A processing entity in our architecture is the most important element of the structure. It is further comprised of multiple processing units and memories. These units work together to calculate the parameters at tile level. It consists of the following major components.

5.4.1 Tile Communication Controller

Each tile has its own communication controller. Like the main communication controller its job is to control and regulate the flow of information in and out of the tile. Some of the main inputs of this module are input data, data from other tiles and start flag. Each tile starts its processing after getting a start flag from the main controller.

The main outputs of the module are local numerator, denominator and local object function to be shared with other tiles. It places its data on a shared bus so that it can be read by other tiles for calculation of global results.

5.4.2 Tile Controller

Like main controller the tile controller handles the operations inside a tile. Its main responsibility is to regulate the operation of calculating local parameters and then receiving them from other tiles to calculate the global results. It generates the control signals for partition matrix calculation and memory management.

5.4.3 Tile Memory

The tile memory is used to store the input data that is provided to the tiles in the beginning of the algorithm. This data is further used in every iteration to calculate the local centers, partition matrix and object function.

5.4.3 Partition Matrix Memory

The partition matrix memories are used to store the membership matrix. The size of this memory depends upon maximum allowable size of input data chunk and number of clusters. This module is actually a combination of few normal memories with a dedicated memory controller. This controller controls the allocation of data to the correct memory in the module.

5.4.4 PM and Object Function Operations Module

These modules implement the basic FCM equations and they are used to compute the local numerator, denominator and object function. These units perform all mathematical computations. They consist of adders and subtractors.

5.4.5 Global and Local Centers Calculation Module

After receiving local numerators, denominators and object functions they are added up in these modules to get the global values of above parameters. Numerator and denominators are solved to get the global centers, where as we get global object function after addition.

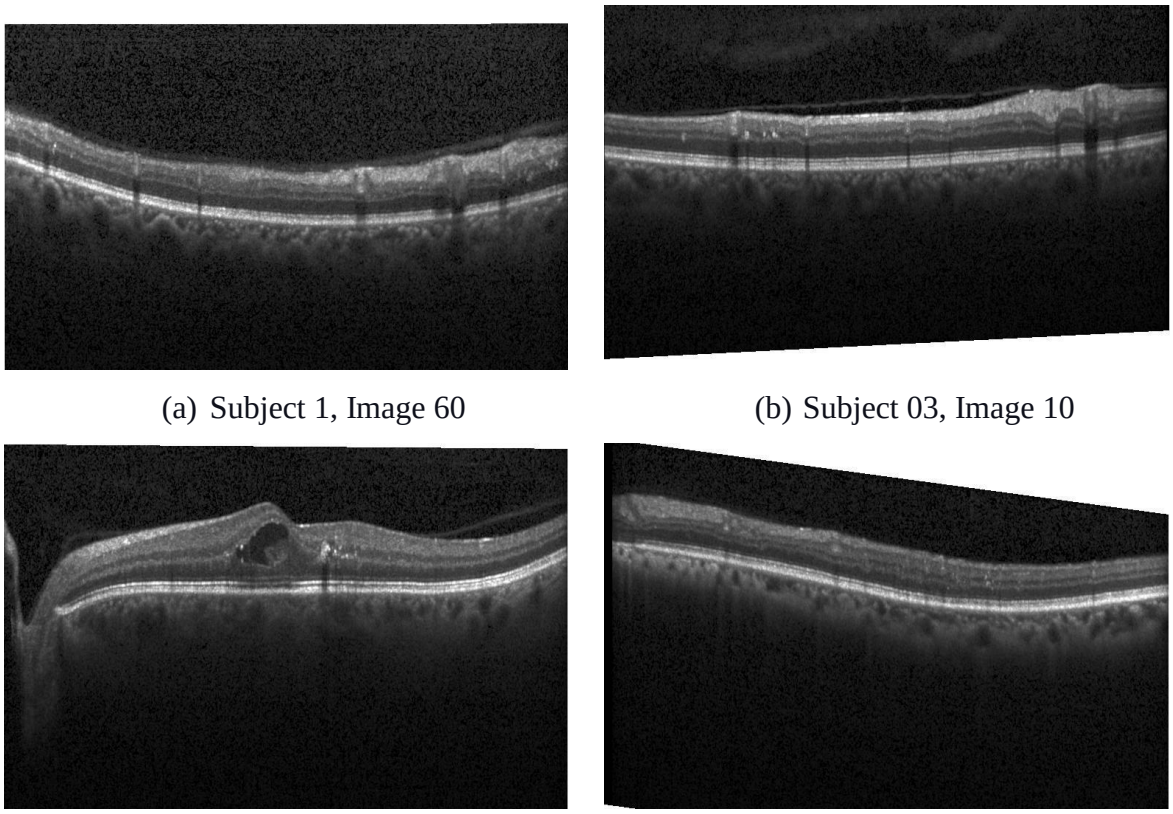
5.4.6 Stop Criteria Module

This module is used to determine the stop criteria of the whole system. If the error is equal to or less than the minimum allowable error then this module declares the end of clustering iterations.

Chapter 6 : EXPERIMENTAL RESULTS

6.1 Dataset

To fully understand the performance of our proposed architecture, we have selected image segmentation as a benchmark. Image segmentation is used to split a digital image into multiple portions to better analyze it. The dataset consists of grayscale Optical Coherence Tomography (OCT) images, these images have been taken for Diabetic Macular Edema (DME) [33]. 10 subjects were considered for the collection of data. Each subject is sampled 61 times, that makes total of 610 images in the dataset. The resolution of each image is 496 x 768 pixels.



(a) Subject 1, Image 60 (b) Subject 03, Image 10

(c) Subject 7, Image 33 (d) Subject 9, Image 47

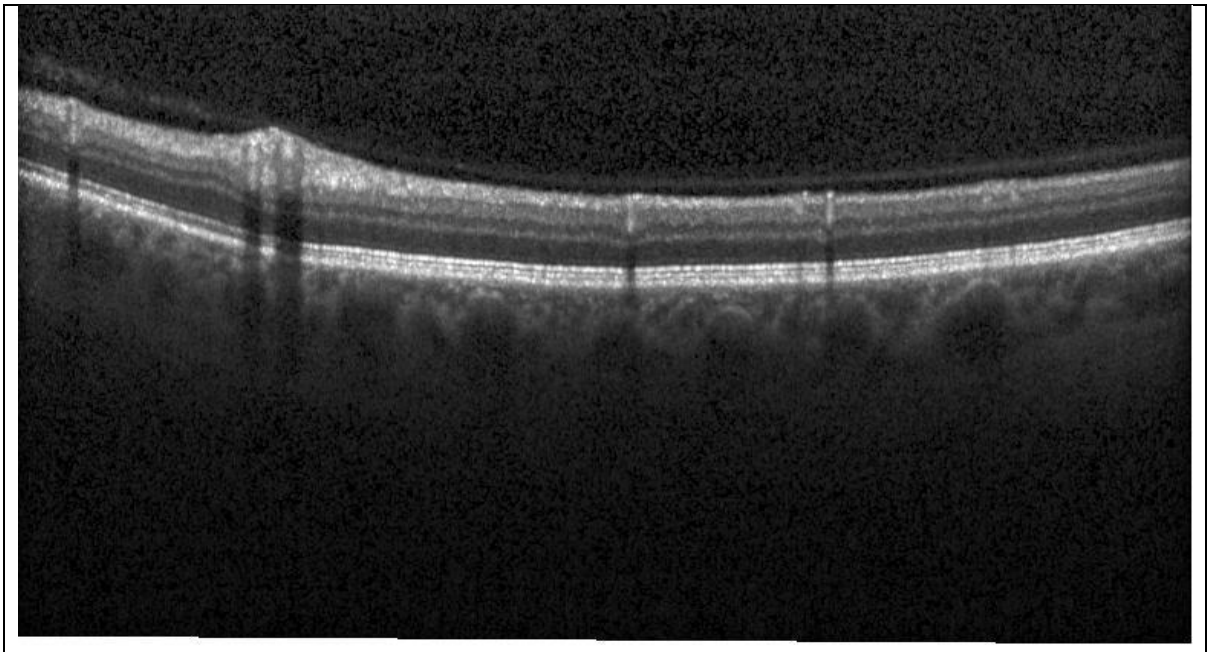
Figure 6-1 Original Images from selected dataset

6.2 Experimental Setup

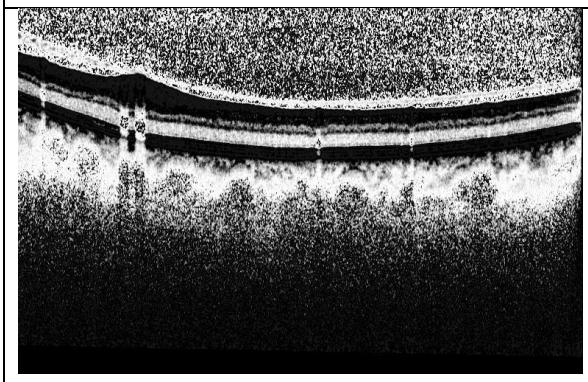
6.1.1 Matlab Simulation

Matlab is selected as the simulation software for performance verification and design validation. The hardware used for matlab simulation is a personal computer with Intel® core

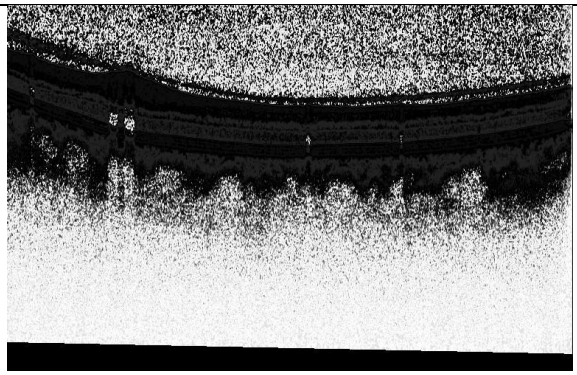
i3 processor and 6 Gigabytes RAM. The matlab simulation is carried out in following phases. First of all, original FCM algorithm is implemented on matlab without using any built-in function so that it can be converted to a parallel algorithm. After that a parallel version of FCM is implemented for the verification of performance enhancement.



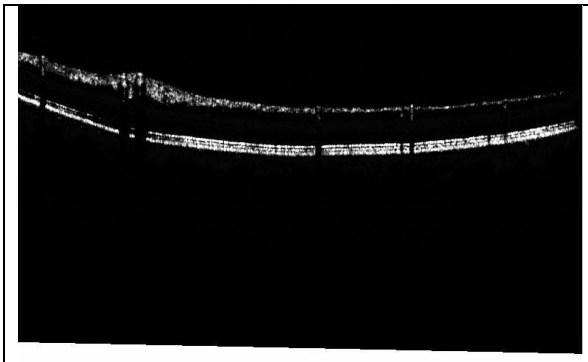
(a) Subject 7, Image 10 (Original Image)



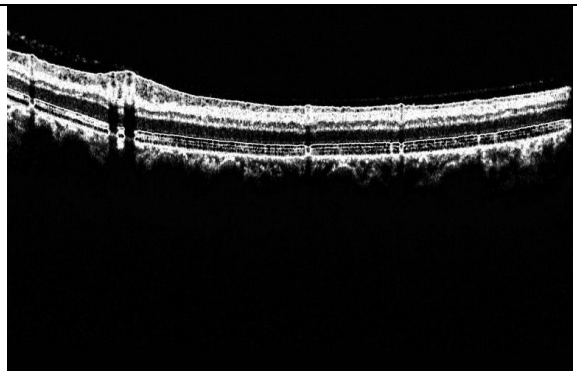
(b) Cluster 1



(c) Cluster 2



(d) Cluster 3



(e) Cluster 4

Figure 6-2: Original and segmented images after clustering

In order to fully test and validate the performance enhancement Matlab simulation is conducted in four phases. There are four main parameters that can affect the performance of the proposed algorithm. These parameters are produced below.

(i) Number of PE units P

(ii) Number of Clusters C

(iii) Image Scale

(iv) Number of Fractional Bits q

Phases	Parameter	Value
Phase I	Number of Cluster C	2
	Number of PE P	4,6,8,12,16
	Fractional bits	Floating Point
	Image Scale	Original
Phase II	Number of Cluster C	2,4,8,12
	Number of PE P	8
	Fractional bits	Floating Point
	Image Scale	Original
Phase III	Number of Cluster C	3
	Number of PE P	8
	Fractional bits	Floating Point
	Scaled Down	{12.5%, 25%,37.5%, …87.5%}
Phase IV	Number of Cluster C	2
	Number of PE P	8
	Fractional bits	Fixed Point {8,10,12}
	Image Scale	Original

Table 6-1: Parameter values for different phases of Matlab simulation

Table 6-1 shows different arrangement of parameter values for different phases of the matlab simulation

6.1.2 Hardware Implementation

Vivado HLS 2018.2 is used for the hardware implementation of the proposed parallel MPSoC architecture. The High-Level Synthesis (HLS) tool is developed by Xilinx to benefit the low time to market of C, C++ and System C languages for direct mapping of hardware for FPGA. Xilinx zynq xc7z020clg484-1 device is selected to implement both conventional and proposed versions of FCM with 2 clusters. For proposed model number of PE is set to 2.

6.2 Performance Measures

6.2.1 Clock cycles

We selected the total number of clock cycles spent on the final convergence of the FCM algorithm for a particular input dataset is the most important parameter for performance enhancement. For simplicity we supposed that every basic mathematical operation consumes one cycle, therefore if there is a mathematical operation that has 2 additions and 1 division then it simply requires 3 clock cycles to execute. For example, following expression consumes 4 cycles.

$$z = a * b + d^e$$

6.2.1 Structural Similarity Index Measure (SSIM)

Structural Similarity Index Measure (SSIM) that is considered as a parameter to determine the simila

rity between two images. We have used it as a benchmark for similarity of output for original and proposed FCM algorithms. SSIM works on human perception of images therefore it considers the brightness and contrast for comparison. We used built-in matlab function to calculate SSIM.

6.2.2 Area

In FPGA implementation one of the key parameters is that researchers are concerned about is Area consumption. The lesser the area occupied by a design, the better it is and vice versa.

6.3 Matlab Simulation Results

As discussed in section 6.1.1, matlab simulation is carried out in three phases, results of each phase are produced below.

6.3.1 Phase-I Results

In phase-I of our work, the original FCM algorithm was run on all 610 images of the data set for image segmentation. The number of cycles consumed with the number of iterations to converge was recorded. After that same images were tested on the proposed parallel FCM algorithm. In this phase the number of clusters is kept fixed whereas number of processing tiles P is varied according to the table 6-1. There was a significant increase in the clock cycles after the implementation of the parallel FCM algorithm. Figure 6.2 shows the comparison of speed improvement in the form of total cycles consumed by the original and different variants of the proposed algorithm. It is observed that by increasing the number of Tiles we can increase the speed of the system, as more and more work is done in parallel mode.

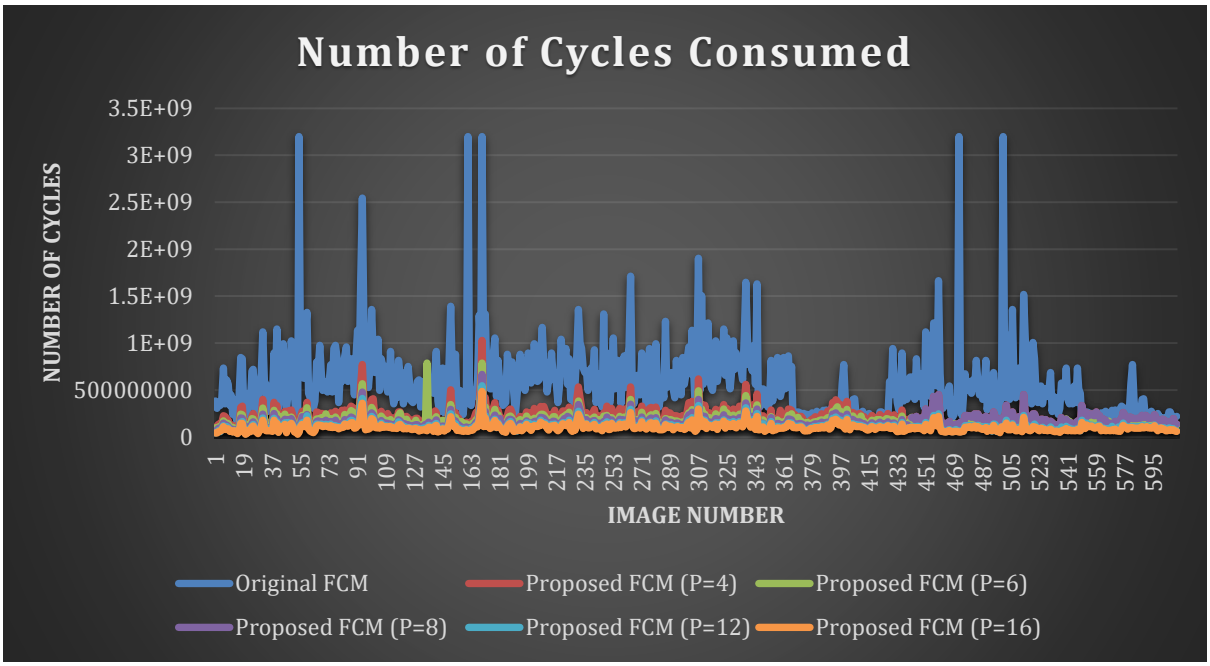


Figure 6-3: Number of cycles consumed by original and proposed algorithms

The average speed up against every simulation of the proposed architecture is displayed in following table.

S. No.	Number of Tiles (P)	Average Speed Up (%)
1	4	61.91632872
2	6	69.80974936
3	8	72.05762335
4	12	79.9781687
5	16	82.33257561

Table 6-2: Average speed up with different Tiles

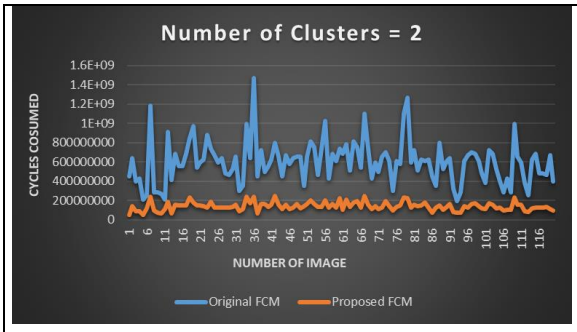
6.3.2 Phase-II Results

In phase-II of the Matlab simulation every 5th image of all subjects is selected for implementation of Parallel and original FCM algorithms because of the computational intensity and huge time required for simulation to run. In this phase the number of tiles P is kept constant and number of cluster C is changed in every implementation. The number of cycles consumed by each implementation were recorded to compare/evaluate the performance of the parallel architecture. When number of clusters is set to 12, the convergence of the algorithm to specified minimum error became difficult. Table 6-3 shows the average speed up of different implementations of the FCM algorithm.

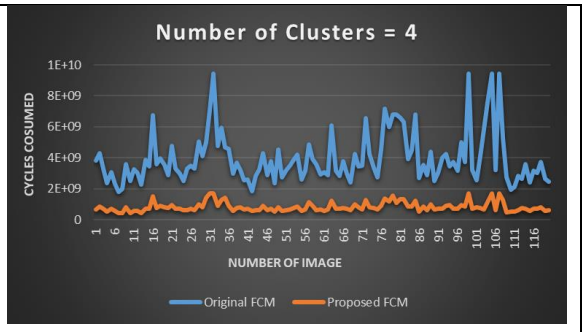
S. No.	Number of Clusters (C)	Average Speed Up (%)
1	2	76.568
2	4	79.1391675
3	8	83.3616812
4	12	85.03519644

Table 6-3: Average speed up with different number of Clusters

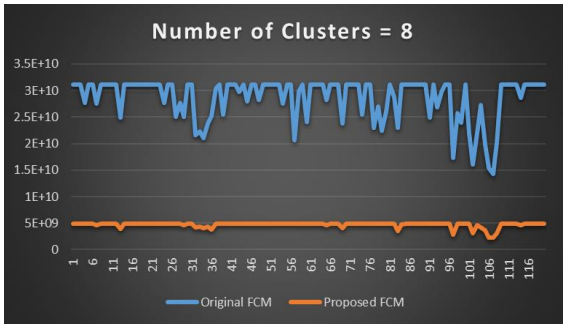
Figure 6-4 shows the speed up of our model with different number of clusters. In figure 6.4(c) the cycles consumed by both algorithms generated a straight line, which shows that algorithms are unable to converge before maximum number of iterations allowed.



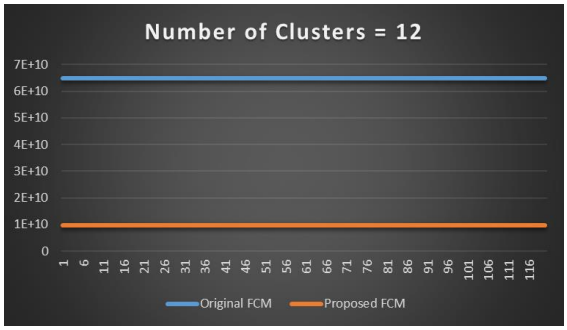
(a) C = 2, P = 8



(b) C = 4, P = 8



(c) C = 8, P = 8

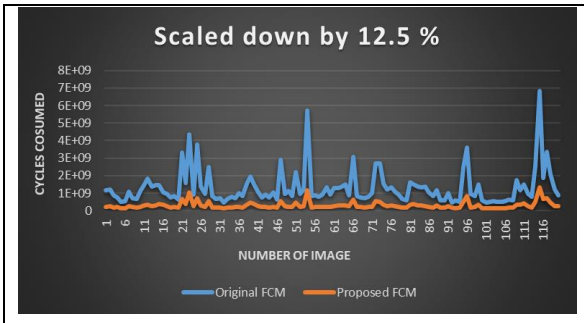


(d) C = 12, P = 8

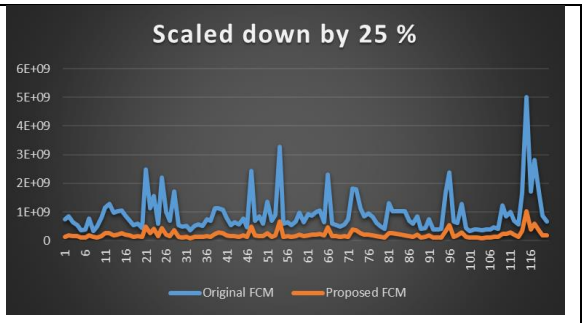
Figure 6-4 : Number of cycles consumed by different tests

6.3.3 Phase-III Results

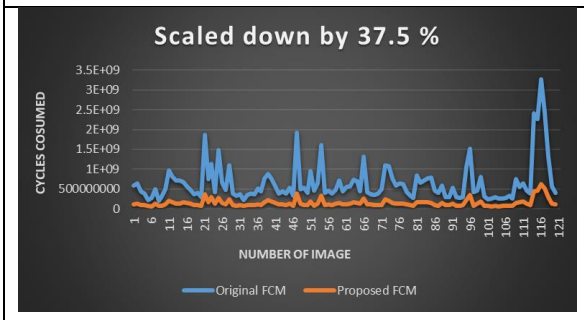
The third phase of our simulation is focused on performance of proposed model under different scaling levels of the input dataset. We have scaled down the images under test by 8 different scale factors ranging from 1/8 to 8/8. The proposed algorithm showed a consistent performance under all above test scenarios. Figure 6-5 shows the performance comparison of phase-III simulation.



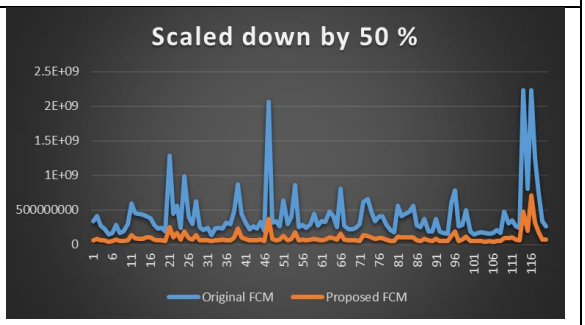
(a) C = 3, P = 8, 12.5%



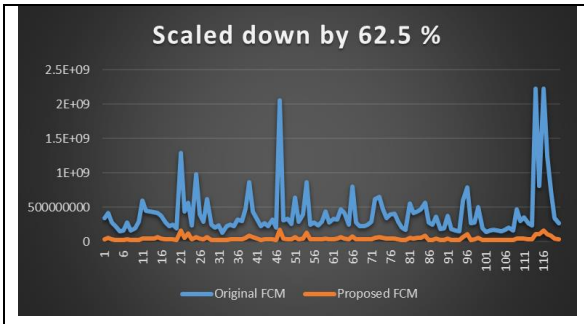
(b) C = 3, P = 8, 25%



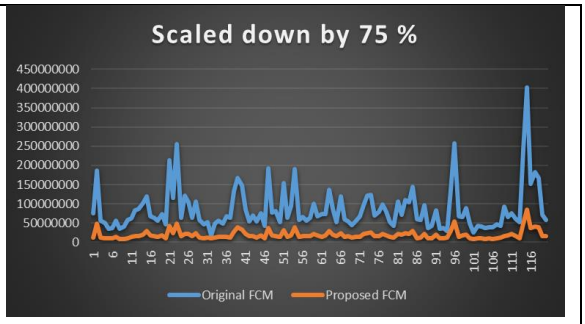
(c) C = 3, P = 8, 37.5%



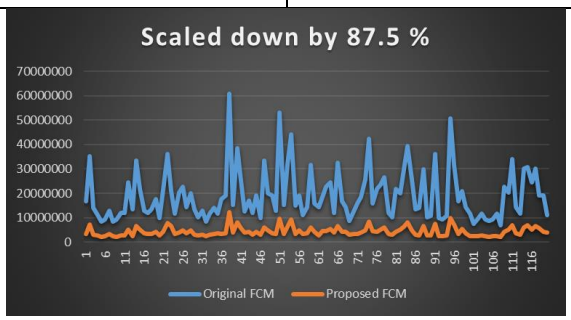
(d) C = 3, P = 8, 50%



(e) C = 3, P = 8, 62.5%



(f) C = 3, P = 8, 75%



(g) C = 3, P = 8, 87.5%

Figure 6-5: Performance comparison of phase-III simulation

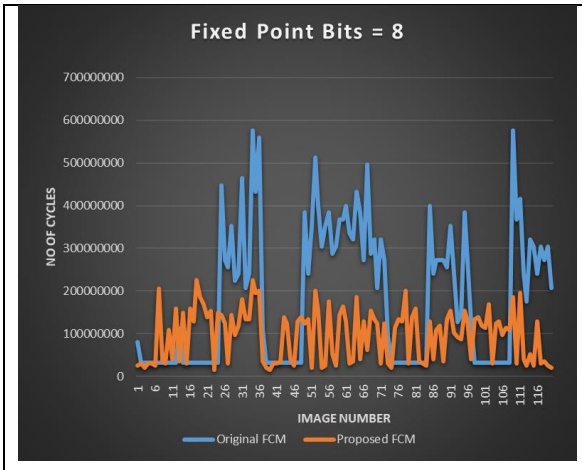
The average speedup of proposed algorithm in phase-III is produced in below table. The data shows that with reducing the number of input elements the average speed-up remains constant.

S. No.	Image Scaled Down	Average Speed Up (%)
1	12.5 (%)	77.17209
2	25 (%)	77.11426729
3	37.5 (%)	77.5812
4	50 (%)	76.58077808
5	62.5 (%)	77.26645484
6	75 (%)	77.15753219
7	87.5 (%)	77.20587121

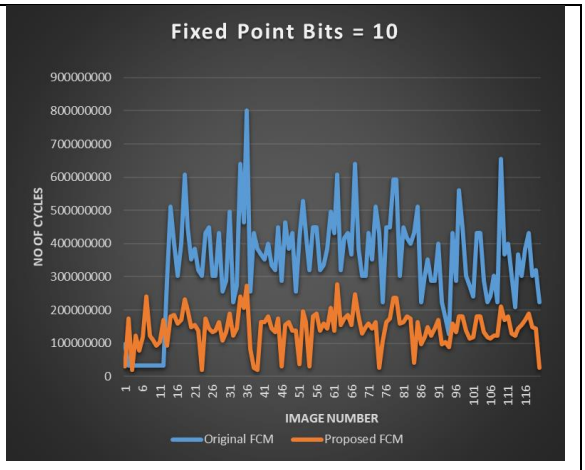
Table 6-4: Average speed-up in phase-III

6.3.3 Phase-IV Results

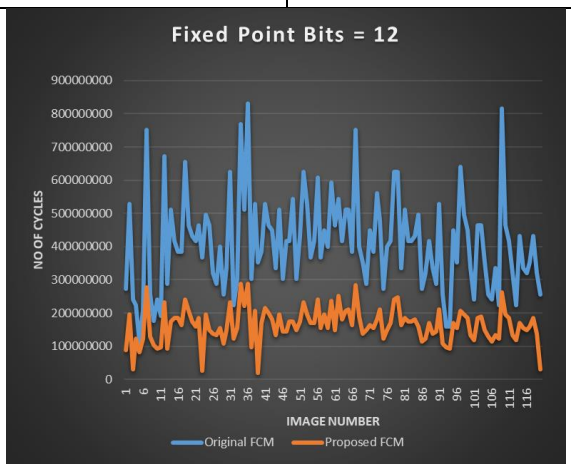
Fourth phase of the matlab simulation is focused on the effects of fixed-point implementation on performance of the proposed architecture.



(a) C = 2, P = 8, 8 bits



(b) C = 2, P = 8, 10 bits



(c) C = 2, P = 8, 12 bits

Figure 6-6: Performance comparison of phase-IV simulation

Due to computational complexity of the FCM algorithm performance can vary with floating point implementation. The algorithm converges after different iterations with same input parameters depending on the number of bits used for its implementation.

A comparison of average speed-up of all phases is shown in figure 6.7

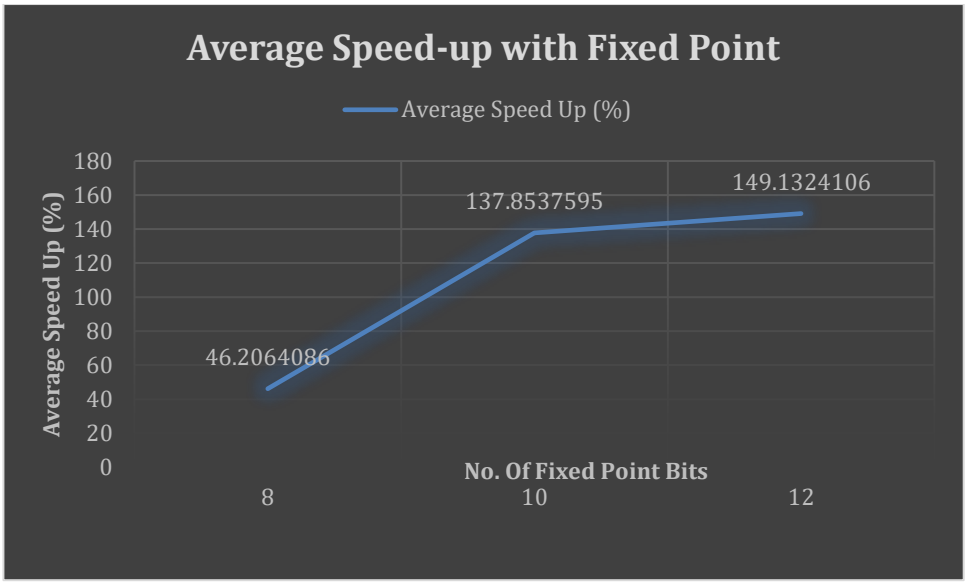
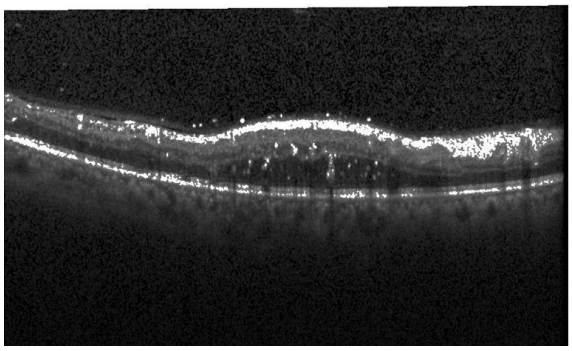


Figure 6-7: Comparison of average speed-ups

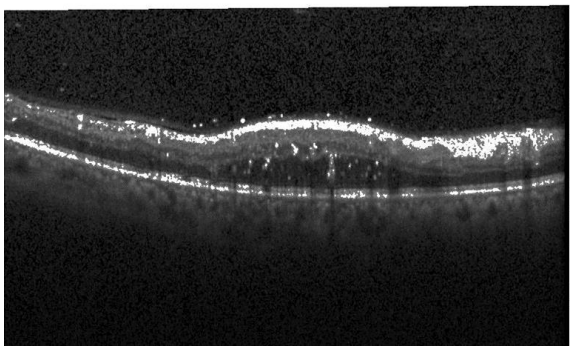
It is evident from above figure that the speed-up is highly effected by the number of bits used for the implementation of proposed architecture.

6.3.4 SSIM calculation

The structural similarities of resulted images were calculated in every phase of the matlab simulation for every run. Due to non-degradational nature of the proposed algorithm structural similarity between outputs of original and proposed FCM came to be 100% in first three phases. The value of every comparison is 1 as SSIM values may vary between 1 and 0.



(a) Subject 1, Image 4, Cluster 1 (Existing FCM)



(b) Subject 1, Image 4, Cluster 1 (Proposed FCM)

Figure 6-8: SSIM comparison of phase I, II and III

Highest match between two images is represented by 1 and lowest match is denoted by a 0. Figure 6-8 shows comparison of cluster images generated by original and proposed algorithms.

However, when we implement the fixed-point model to preserve area and gain speed a compromise has to be made on SSIM. In phase-IV of the simulation the SSIM varies due to different convergence points of the original and proposed algorithm.

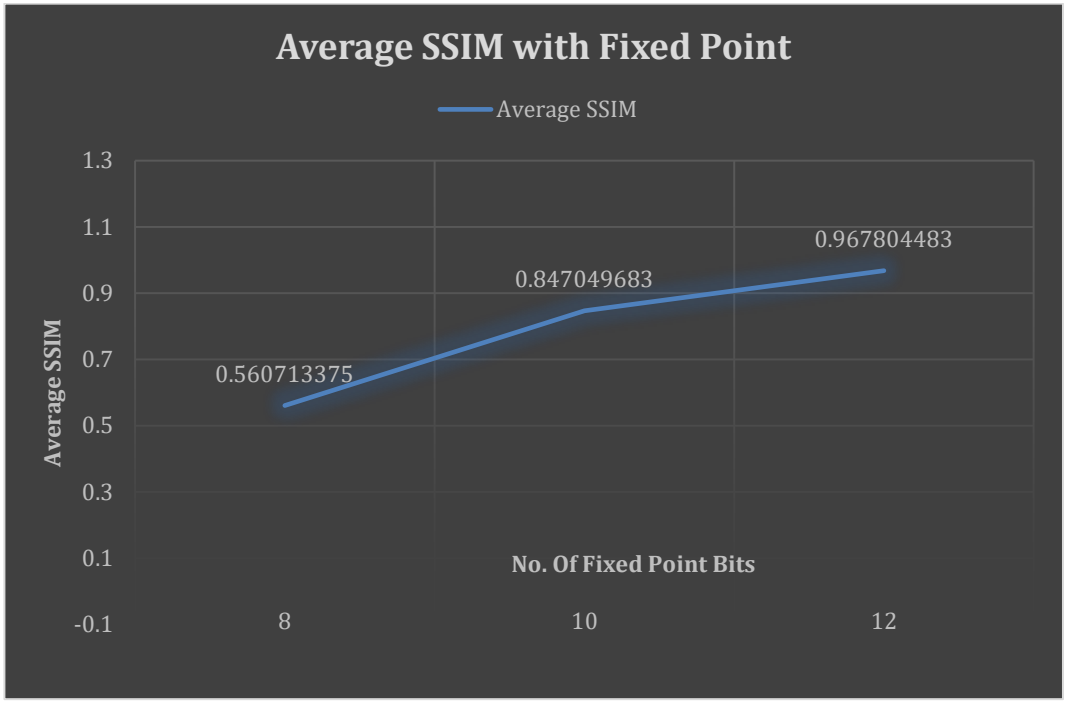


Figure 6-9: Average SSIM Value of Phase-IV with varying bits

Figure 6-9 shows that the SSIM values are very low when we use 8 bits fixed-point implementation but gets better with 10 and 12 bits.

6.4 Hardware Implementation Results

The FCM algorithm is a resource hungry algorithm, therefore to have a better understanding of the hardware implementation phase we developed conventional FCM for Vivado HLS and noted its performance parameters. These parameters include Clock, Area and Latency of the architecture. After that the proposed version of the FCM algorithm is implemented and its results are noted for comparison.

6.4.1 Clock

First of all, clock of both traditional and parallel architectures turned out to be exactly same and its value is 9.514 nanoseconds. This shows the operational similarity of both algorithms.

6.4.2 Hardware Resources

Following table shows the hardware resource utilization by both implementation of FCM algorithm.

Parameter	Original FCM		Proposed FCM		Available
	Used	Utilization (%)	Used	Utilization (%)	
BRAM_18K	69	24	67	23	280
DSP48E	197	89	200	90	220
Flip Flop	23084	21	23836	22	106400
LUT	21577	40	22790	42	53200

Table 6-5: Hardware utilization comparison of both algorithms

The data produced in above table shows that the proposed FCM algorithm consumed almost same hardware resources as the original algorithm. The conforms that the proposed model will not put any burden on the utilization of the resources of devices.

6.4.3 Latency

It is the most important parameter as it determines the overall speed-up of the proposed model. The latency of original FCM is measured to be 1484413 clock cycles, whereas the latency of the proposed model is 268630 clock cycles. This shows 81.9% increase the overall speedup of the system.

Chapter 7 : CONCLUSION & FUTURE WORK

7.1 Conclusion

FCM is a very useful and powerful algorithm and it has numerous applications in the field of unsupervised machine learning. One of the key applications is image segmentation. Now with ever-increasing technological advancements, demand for edge computing is also on the rise. Keeping in view above FCM can be used for the benefit of mankind by developing smart and intelligent devices that are capable of performing the necessary tasks onsite instead of sending the data to the cloud. Examples of such devices could be a smart drone monitoring the health of the crops or a self-driving car making critical decision on spot.

Our proposed design can help achieve above goals. It can tackle the challenges confronted by a complex computational algorithm. We have successfully demonstrated that FCM algorithm can be implemented in a parallel way to reduce the execution time and increase the efficiency. Our work can be summarized in the following lines.

- Implementation of parallel FCM algorithm.
- Verification of performance enhancement.
- Study of algorithm behavior by varying different parameters of the algorithm.
- Data integrity verification by computing the structural similarity of the results.
- MPSoC architecture implementation for hardware analysis.

7.2 Contribution

The following contributions to the field of embedded systems have been made as a result of our work.

- An improved hardware implementation of the FCM clustering algorithm has been made that can be used to gain enhance throughput at fields.
- A benchmark has been set for future researchers to test the performance of the FCM algorithm.
- The edge computing field can benefit from our work and can improve the performance of the nodes.
- The behavior of the FCM algorithm under parallel architecture is tested that can be beneficial for multiple fields of science.

7.3 Future Work

There is always room for improvement in every system. Therefore, we would like to point few areas of improvement that can be exploited in the future.

- Implementation of Network on Chip for improvement of parallel FCM architecture.
- Performance enhancement by architecture optimization of Parallel FCM.
- Implementation of our proposed architecture for various test cases and datasets to check its performance.

REFERENCES

[1] S. S. Harshvardhan, S. Kiran, and S. Kumari, "Image segmentation - wikipedia." https://en.wikipedia.org/wiki/Image_segmentation (accessed Nov. 16, 2020).

[2] M. Matteucci, "Clustering - Fuzzy C-means," 2019. https://matteucci.faculty.polimi.it/Clustering/tutorial_html/cmeans.html#dunn (accessed Nov. 16, 2020).

[3] I. Sittón-Candanedo and J. M. Corchado, "An Edge Computing Tutorial," *Orient. J. Comput. Sci. Technol.*, vol. 12, no. 2, pp. 34–38, 2019, doi: 10.13005/ojcst12.02.02.

[4] R. Parikh and V. Chemitiganti, "Edge Computing: Challenges and Opportunities with Kubernetes," *Platform9*, 2019. https://platform9.com/blog/edge-computing-challenges-and-opportunities/ (accessed Nov. 17, 2020).

[5] "Omnichannel Retail Challenges And How To Address Them." Accessed: Nov. 21, 2020. [Online]. Available: https://www.rfcode.com/blog/edge-computing-inherent-challenges-and-how-to-address-them.

[6] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," *Proc. - 2016 IEEE Int. Conf. Smart Cloud, SmartCloud 2016*, pp. 20–26, 2016, doi: 10.1109/SmartCloud.2016.18.

[7] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Futur. Gener. Comput. Syst.*, vol. 97, pp. 219–235, 2019, doi: 10.1016/j.future.2019.02.050.

[8] S. Narayanan, E. Chaniotakis, and C. Antoniou, "Shared autonomous vehicle services: A comprehensive review," *Transp. Res. Part C Emerg. Technol.*, vol. 111, pp. 255–293, Feb. 2020, doi: 10.1016/j.trc.2019.12.008.

[9] G. E. Measurement, "Healthcare for the industry." https://www.rfcode.com/blog/healthcare-for-the-edge (accessed Nov. 22, 2020).

[10] K. Shaw, "What is edge computing and why it matters | Network World," *Network World*, 2019. https://www.networkworld.com/article/3224893/what-is-edge-computing-and-how-it-s-changing-the-network.html (accessed Nov. 22, 2020).

[11] Wikipedia, "Cluster analysis - Wikipedia," 2017. https://en.wikipedia.org/wiki/Cluster_analysis (accessed Dec. 06, 2020).

[12] S. Craciun,R. Kirchgessner, A. D. George, H. Lam, and J. C. Principe, "A real-time, power-efficient architecture for mean-shift image segmentation," *J. Real-Time Image Process.*, vol. 14, no. 2, pp. 379–394, 2018, doi: 10.1007/s11554-014-0459-1.

[13] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1701–1713, 2008, doi: 10.1109/TCAD.2008.923415.

[14] "System on a chip - Wikipedia." https://en.wikipedia.org/wiki/Multiprocessor_system_on_a_chip (accessed Nov. 23, 2020).

[15] L. C. Jain, H. S. Behera, J. K. Mandal, and D. P. Mohapatra, "Computational Intelligence in Data Mining – Volume 2: Proceedings of the International Conference on CIDM, 20-21 December 2014," *Smart Innov. Syst. Technol.*, vol. 32, 2015, doi: 10.1007/978-81-322-2208-8.

[16] Y. J. Yeh, H. Y. Li, C. Y. Yang, and W. J. Hwang, "Fast fuzzy c-means clustering based on low-cost high-performance VLSI architecture in reconfigurable hardware," *Proc. - 2010 13th IEEE Int. Conf. Comput. Sci. Eng. CSE 2010*, pp. 112–118, 2010, doi: 10.1109/CSE.2010.22.

[17] K. Othman and A. Ahmad, "Quantification and segmentation of breast cancer diagnosis: Efficient hardware accelerator approach," *WSEAS Trans. Comput.*, vol. 12, no. 5, pp. 185–197, 2013.

[18] H. Y. Li, W. J. Hwang, and C. Y. Chang, "Efficient fuzzy c-means architecture for image

segmentation," *Sensors*, vol. 11, no. 7, pp. 6697–6718, 2011, doi: 10.3390/s110706697.
[19] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable FPGA architecture for k-mean clustering," *Proc. - IEEE Int. Symp. Circuits Syst.*, pp. 1801–1804, 2013, doi: 10.1109/ISCAS.2013.6572215.
[20] A. Tehreem, S. G. Khawaja, A. M. Khan, M. U. Akram, and S. A. Khan, "Multiprocessor architecture for real-time applications using mean shift clustering," *J. Real-Time Image Process.*, vol. 16, no. 6, pp. 2233–2246, 2019, doi: 10.1007/s11554-017-0733-0.
[21] J. Yadav and M. Sharma, "A Review of K-mean Algorithm," *Int. J. Eng. Trends Technol.*, 2013.
[22] J. Canilho, M. Véstias, and H. Neto, "Multi-core for K-means clustering on FPGA," *FPL 2016 - 26th Int. Conf. Field-Programmable Log. Appl.*, 2016, doi: 10.1109/FPL.2016.7577313.
[23] K. Neshatpour *et al.*, "Big biomedical image processing hardware acceleration: A case study for K-means and image filtering," *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 2016-July, pp. 1134–1137, 2016, doi: 10.1109/ISCAS.2016.7527445.
[24] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data," *Proc. 2011 NASA/ESA Conf. Adapt. Hardw. Syst. AHS 2011*, pp. 248–255, 2011, doi: 10.1109/AHS.2011.5963944.
[25] D. B. K. Trieu and T. Maruyama, "Real-time color image segmentation based on mean shift algorithm using an FPGA," *J. Real-Time Image Process.*, vol. 10, no. 2, pp. 345–356, 2015, doi: 10.1007/s11554-012-0319-9.
[26] W. Hwang, Z. Fan, and T. Shen, "Unsupervised Image Segmentation Circuit Based on Fuzzy C-Means Clustering," 2012, no. c, pp. 23–30.
[27] J. F. Kolen and T. Hutcheson, "Reducing the Time Complexity of the Fuzzy," *Brain*, vol. 10, no. 2, pp. 263–267, 2002.
[28] C. M. Ou, W. J. Hwang, and S. M. Yang, "Efficient hardware architecture for kernel fuzzy c-means algorithm," *Appl. Mech. Mater.*, vol. 284–287, pp. 3079–3086, 2013, doi: 10.4028/www.scientific.net/AMM.284-287.3079.
[29] S. G. Khawaja, M. Usman Akram, S. A. Khan, A. Shaukat, and S. Rehman, "Network-on-Chip based MPSoC architecture for k-mean clustering algorithm," *Microprocess. Microsyst.*, vol. 46, pp. 1–10, 2016, doi: 10.1016/j.micpro.2016.08.006.
[30] H. Mushtaq *et al.*, "A parallel architecture for the partitioning around medoids (PAM) algorithm for scalable multi-core processor implementation with applications in healthcare," *Sensors (Switzerland)*, vol. 18, pp. 1–17, 2018, doi: 10.3390/s18124129.
[31] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *J. Cybern.*, vol. 3, no. 3, pp. 32–57, 1973, doi: 10.1080/01969727308546046.
[32] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, vol. 66. 2012.
[33] "Segmentation of OCT images (DME) | Kaggle." https://www.kaggle.com/paultimothymooney/chiu-2015 (accessed Mar. 07, 2021).