To the Graduate Council:

I am submitting herewith a thesis written by Venkatesh Bhaskaran entitled "Parameterized Implementation of K-means Clustering on Re-configurable Systems". I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Gregory Peterson, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Donald W Bouldin, Professor

Dr. Hairong Qi, Professor

Dr. Chandra Tan, Professor

Accepted for the Council:

Vice Provost and Dean of Graduate Studies

PARAMETERIZED IMPLEMENTATION OF K-MEANS CLUSTERING ON RECONFIGURABLE SYSTEMS

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Venkatesh Bhaskaran

December 2003

DEDICATED

TO

AMMA & APPA

ACKNOWLEDGEMENTS

"Interdependence is certainly more valuable than independence"

- Anonymous

This thesis is the result of two years of work whereby I was accompanied and inspired by many people. I am glad to have this opportunity to express my gratitude to all of them. First, I would like to thank my direct advisor Dr. Gregory Peterson for giving me the opportunity to participate in his research group in the year 2001 and advising me all through, from then on. He has played several roles as a friend, advisor, listener, sympathizer etc. I owe him a great deal for all the efforts he has taken in helping me take the right path.

I would also like to thank my co-advisor Dr. Donald W Bouldin for letting me use his research laboratory and the latest and greatest design tools. His unmatchable research experience and flawless attitude, as a professor needs a special mention. I have been fortunate to take some of his classes and research leads.

Special thanks to Dr. Chandra Tan; without him, this thesis and the greatest learning experience that I have had, would have stood, a dream. There has not been a day that passed that he has not inspired me. His technical expertise, low-profile down to earth kind of attitude has truly amazed me. I am so very grateful to him for all the help and inspiration. We have had several debates, conversations and discussions ranging from design fundamentals to problem solving techniques to stock market and to life in Indonesia, India and so on. Thank you Dr. Chandra.

Special thanks, to my co-advisor Dr. Hairong Qi for helping me understand some of the concepts of hyper spectral imaging, dataset pre-analysis and other related fundamentals. I truly cherish those discussions as a rewarding experience.

I couldn’t forget to thank my family, my mom and my dad, primarily. I should have been truly blessed to have such amazing parents and take this opportunity to express my thanks for the unconditional love and support they have given me. They have stood by me during happy and hard times and I thank them once again for that. Love to my little nieces and nephew, Arti, Amita, Sahana and Sanjay. They are the cutest things in the world and it’s hard to forget all the funny days of horseback riding with Arti and Amita or playing pickaboo with sahana kutty and sanjay. My brother Mahesh and sister Preetha are my greatest assets that have supported me constantly and have been instrumental in helping me stay driven and focused. I take this opportunity to thank them for their thoughtfulness towards me. I would also like to thank my sister-in-law Juana for her encouragement, support and willingness to help. Special thanks, to my brother-in-law Dr. Natesan Venkateswaran for reviewing my thesis material and giving me valuable suggestions and feedback. I owe him a lot for helping me out of several tough situations and decisions that I have had to make as a student.

Some of the first digital lessons I leant from him like the K-N and F-M partitioning algorithms, was even before I started graduate school. I would say, my interest in digital design started right at that point. Thank you ‘Jijaji’ for all your help. All in all, without the support of my family, this work would have not been possible.

Last but not the very least, I feel honored to acknowledge my friends and co-researchers, first, Fuat Karakaya, who let me use some of his scripts that helped me avoid some laborious repetitive tasks conducting experiments. Several nights in the lab we have spent hours analyzing and solving mysteries of the digital world, not to forget some meaningful theological discussions. Thanks to other research members, Phani Teja for providing me with the PCA code, Srikala for proofreading this document, Ashwin, Sampath, Mahesh for some ‘kadi’ jokes in the lab and to Mardav for accompanying me for coffee breaks. Special thanks to the RecSports people Harold, Jim, Jeff, Brande, Jackie, Katy, Brett, Natalie, Steve, Judd, Alicia, Sarah, Megan, Ben and all the others who have made a difference in my life.

ABTRACT

Processing power of pattern classification algorithms on conventional platforms has not been able to keep up with exponentially growing datasets. However, algorithms such as k-means clustering include significant potential parallelism that could be exploited to enhance processing speed on conventional platforms. A better and effective solution to speed-up the algorithm performance is the use of a hardware assist since parallel kernels can be partitioned and concurrently run on hardware as opposed to the sequential software flow. A parameterized hardware implementation of k-means clustering is presented as a proof of concept on the Pilchard Reconfigurable computing system. The hardware implementation is shown to have speedups of about 500 over conventional implementations on a general-purpose processor. A scalability analysis is done to provide a future direction to take the current implementation of 3 classes and scale it to over N classes.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The goal of this chapter is to introduce the reader to various components in the current work. Data analysis is considered an important problem in the research circles and how that could be best achieved is an important question that appeals to many young researchers. Years of work in collecting samples and massive datasets do not have any value proposition if researchers do not understand what they mean. Therefore the necessity for faster data analyses has been the focus of many researchers.

1.1 DATA ANALYSIS

Several real world classification problems are characterized by a large number of inputs and moderately large number of classes that can be assigned to any input. Two popular simplifications have been considered for such problems: (i) feature extraction, where the input space is projected into a smaller feature space (ii) modular learning, where a number of classifiers, each focusing on a specific aspect of the problem, are learned instead of a single classifier [1]. Several methods for feature extraction and modular learning have been proposed in the computational intelligence community.

Analysis of land cover types from airborne/space borne sensors is an important classification problem in remote sensing [7]. Due to advances in sensor technology, it is now possible to acquire spectral data simultaneously in more than 100 bands, each of which measures the integrated response of a target over a narrow window of the electromagnetic spectrum. The bands are ordered by their wavelengths and spectrally adjacent bands are generally statistically correlated with target dependent groups of bands [7]. Using such high dimensional data for classification of land cover potentially improves distinction between classes but dramatically increases problems with parameter estimation and storage and management of the extremely large datasets [7].

1.1.1 HYPER-SPECTRAL DATA ANALYSIS

Hyper-spectral data is pixel information collected over 100-1000 spectral bands simultaneously. Hyper-spectral methods for deriving information about the Earth's resources using airborne or space-based sensors yield information about the electromagnetic fields that are reflected or emitted from the Earth's surface, and in particular, from the spatial, spectral, and temporal variations of those electromagnetic fields [10]. Chemistry-based responses which are the primary basis for discrimination of the land cover types in the visible and near infrared portions of the spectrum are determined from the data acquired simultaneously in multiple windows of the electromagnetic spectrum. In contrast to airborne and space-based multispectral sensors, which acquire data in a few (<10) broad channels, hyper-spectral sensors can now acquire data in hundreds of windows, each less than 10 nanometers in width. Because many land cover types have only subtle differences in their characteristic responses, this potentially provides greatly improved characterization of the unique spectral characteristics of each, and therefore increases the classification accuracy required for the detailed mapping of species from remotely sensed data [10].

1.2 CLUSTERING TECHNIQUES

Data analyses, for instance, interpretation of Landsat images that involve huge datasets imply no meaning and is impractical as well for direct manipulation [6]. Some methods of data compression must initially be applied to reduce the size of the dataset without losing the essential component of the data. Most of such methods sacrifice some detail though. Clustering technique is one such protocol that has been used for data analysis both as a compression algorithm and for quick-view analysis. Generically speaking, clustering involves dividing a set of data points into non-overlapping groups, or clusters, of points, where points in a cluster are more similar to one another than to points in other clusters. The main goal of clustering is to reduce the size and complexity of the dataset. Clustered sets of points require much less storage space and can be manipulated more quickly than the original data [3]. The value of a particular clustering method will depend on how closely the reference points represent the data as well as how fast the program runs. There have been several algorithms proposed in the past for clustering data for the purpose of compression and dimensionality reduction. Two such methods are the supervised (knn) and unsupervised (k-means) methods. In the Knn method, the feature space, described in chapter3 is partially divided into testing set and training set. The classifier is trained on the training set and the testing set is used to test the performance of the classifier derived [22]. This thesis focuses on the unsupervised learning scheme.

1.2.1 K-MEANS CLUSTERING

A non-hierarchical approach to forming good clusters is to specify a desired number of clusters, say, k, then assign each case (object) to one of k clusters so as to minimize a measure of dispersion within the clusters. A very common measure is the sum of distances or sum of squared Euclidean distances from the mean of each cluster. The problem can be set up as an integer-programming problem but because solving integer programs with a large number of variables is time consuming, clusters are often computed using a fast, heuristic method that generally produces good (but not necessarily optimal) solutions. The k-means algorithm is one such method.

The k-means algorithm [25] starts with an initial partition of the cases into k clusters. Subsequent steps modify the partition to reduce the sum of the distances for each case from the mean of the cluster to which the case belongs. The modification consists of allocating each case to the nearest of the k means of the previous partition. This leads to a new partition for which the sum of distances is strictly smaller than before. The improvement step is repeated until the improvement is very small. The method is very fast. There is a possibility that the improvement step leads to fewer than k partitions. In this situation one of the partitions (generally the one with the largest sum of distances from the mean) is divided into two or more parts to reach the required number of k partitions. The algorithm can be rerun with different randomly generated starting partitions to reduce the chances of the heuristic producing a poor solution. Generally the number of "true" clusters in the data is not known. Therefore, it is a good idea to run the algorithm with different values for k that are near the number of clusters one expects from the data to see how the sum of distances reduces with increasing values of k.

This algorithm has its origin in the data-mining field [1]. It is utilized for classification purposes and to discover anomalies and patterns in both small and large data sets. There exist many different variants of k-means clustering – most of which are variants adapted for special purpose environments. With the growth of data collected on operational and transactional data, the field of data mining has become increasingly important. The growth of data has been accelerated with the commercialization of the Internet and the increased use of personal computer. In this environment collection of individual metrics is relatively cheap and unobtrusive to the user. Companies who have been collecting vast amount of data on consumer habits are now confronted with the dilemma of what to do with all the data. This is where k-means clustering becomes useful. It provides a remedy tailored to this problem and reveals patterns that otherwise are obfuscated. In short, it can be said that k-means is a common solution to the segmentation of multi-dimensional data [8][9]. However, these large amounts of data sets require large computational capacity. The nature of this problem is ideally implemented on a high performance computing architectural node.

1.3 HARDWARE IMPLEMENTATION

K-means algorithm developed in high performance computing (HPC) architectures [19] drastically increases the achievable parallelism with small transformation in the conventional algorithm. K-means is an iterative algorithm that assigns to each pixel a label indicating which of K clusters the pixel belongs to. The conventional software implementation of k-means algorithm uses floating-point arithmetic and Euclidean distances. Floating-point arithmetic and the multiplication-heavy Euclidean distance calculation are efficient on a general-purpose processor, but they have large area and speed penalties when implemented on an FPGA [3]. In order to get the best performance of k-means on an FPGA, the algorithm needs to be transformed to eliminate these operations. An alternative distance measure, Manhattan distances, that does not require multiplier was used to develop the macro on the hardware. Measurement using full precision and truncated bit widths were performed, examined and presented. A direct translation of the standard software implementation of k-means would result in a very inefficient use of FPGA hardware resources. Alternatively, changes to the conventional algorithm have been done to better realize the performance on the hardware.

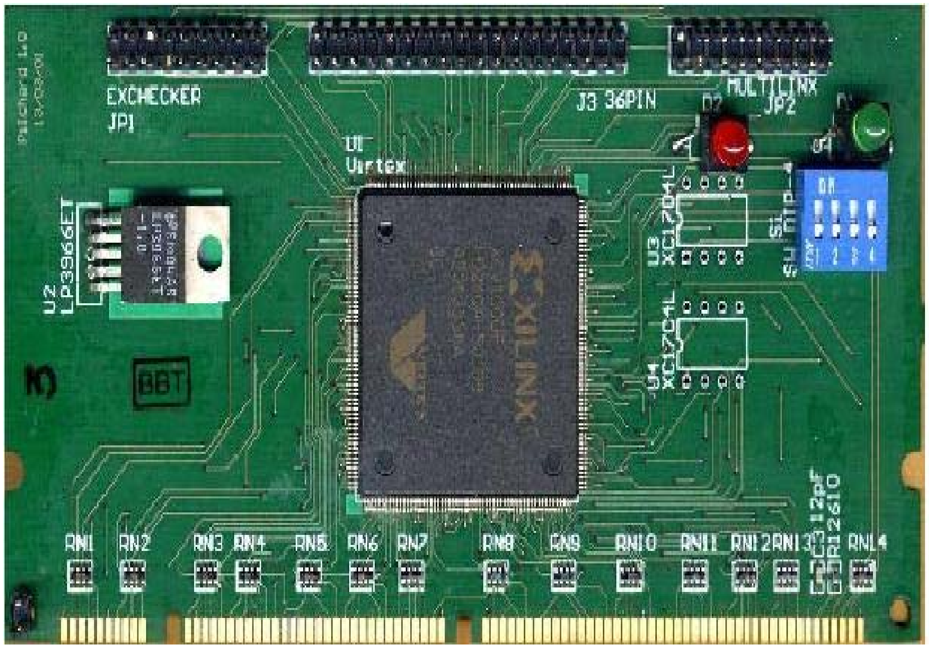


Figure 1.1: The Pilchard Board

1.3.1 PILCHARD SYSTEM

A high performance architecture called Pilchard [18] shown in Figure 1.1 developed at Chinese University of Hong Kong has been used to perform measurements of this clustering technique. One of the main advantages of trying to map k-means clustering is the inherent massive parallelism that can be exploited within the algorithmic level. Other hardware level optimizations have been discussed later in this manuscript.

The efficient interface and low cost model of the Pilchard architecture makes it suitable for various applications including cryptosystems, image processing and speech processing, clustering techniques, in addition to rapid prototyping. The unique features about pilchard are that it uses DIMM slot as an interface as opposed to PCI thereby leveraging bandwidth. A picture of the Pilchard card is shown in Figure 1.1 [18].

The other distinct advantages of Pilchard when compared with other FPGA platforms available at University Of Tennessee are the use of one big virtex chip, easy interface, minimal overhead either in hardware resources and timing. Developers can concentrate more on algorithmic partitioning, if needed, instead of worrying much about partitioning program or data within multi-chip modules on the prototype board. Additionally much time can be spent on algorithm development rather than focusing on complex interfacing. Chapter 4 explains some of the other features of the Pilchard system. For complete information, refer the Pilchard user guide [18].

1.4 SCOPE OF THE THESIS

K-means algorithm, presented as a proof of concept over a hardware unit and more specifically on the Pilchard system, is applied to both synthetic and hyper-spectral datasets. Primarily, the five points summarized below are the focus of the current work.

1. Implementing k-means software version.
2. Analysis of scalability of k-means on FPGA units.
3. Developing a methodology flow design to realize the algorithm on the hardware.

4. Discussing design issues of k-means clustering on hardware, with reference to Pilchard system and implementing it.
5. Cluster classification verification of the algorithm using synthetic and hyperspectral datasets.

Chapter 1 introduces the reader to all the main idea and briefly discusses about each of the topics. Chapter 2 details some of the background work done by various researchers in implementing clustering algorithms on hardware units as well as hardware-software co-design models. Chapter 3 provides information about the algorithm itself and some of its variants. Certain acceleration approaches are detailed followed by the description of hyperspectral datasets. Some pre-processing techniques used on the data have been explained. Chapter 4 describes a bit about the hardware platform that has been used in this work to realize the algorithm. The benefits and bottlenecks of using the system have been pointed out. Furthermore, hardware blocks available within the platform have been described to familiarize a reader with limited or no knowledge about the hardware system. Chapter 5 describes the methodology and the approach taken in the design of the algorithm. Chapter 6 details the implementation of the design including the scripts used. Also, the results obtained are presented followed by interpretation of the numbers. Chapter 7 discusses conclusion of the thesis work followed by future work that could be interpolated from the current one.

2. RELATED WORK

A number of researchers have worked on clustering algorithms, especially k-means. Enormous research focus has been on data mining and multi-spectral pattern classification applications using k-means algorithm. Some researchers have worked on classification of hyper-spectral images but the focus has been on the quality of clusters and handling of huge data sets. The nature of the algorithm as is discussed in chapter 3 reveals that the changes in final cluster centers/classification are robust to changes in the ground conditions. This reinforces the fact that speed is still a main concern and fast number crunching of massive datasets becomes a matter a significant importance.

2.1 K-MEANS CLUSTERING ON FPGA-BASED PLATFORMS

Few research groups have experimented to map k-means algorithm to re-configurable fabric in order to achieve some acceleration. Such experiments have ‘feel-good’ results but there has not been a high performance-computing platform coupled with reconfigurable elements to really be able to exploit the tremendous potential parallelism inherent within the algorithm. The architectural goal of high performance reconfigurable computing nodes is to achieve hardware-like performance and software-like flexibility. Image processing algorithms are natural candidates for high performance computing due to their inherent parallelism and intense computational demand [1].

Mainly, two thesis topics based on this have been produced. The first one [7] is a work titled ‘K-means clustering for Color Image processing on Reconfigurable Hardware Board’ conducted experiments on Annapolis Microsystems Wild Force FPGA-based custom computing machine. Two facts were established as an outcome of their experiments. One, custom-computing machines are suitable for intermediate-level image processing algorithms. Two, a custom computing approach permits image-processing applications to run at high speed.

The second thesis [1] also from the same research group at Northeastern University, Boston did similar work as the first one but on hyper-spectral images. A couple of transformation techniques have been employed to take advantage of FPGA elements and presented pros and cons of such implementation. Manhattan distance measure was employed to classify points under a certain category and this helped eliminate multipliers in hardware that turns out to be area and speed expensive. The other major technique involved in the work was improving computational time by the use of input data truncation. However, the number of bits to be truncated was not rationalized but has been observed at the same token, that a significant improvement was obtained by truncating input data by 2 bits. A speed up of two orders of magnitude faster than the same algorithm run in software was shown [1]. The current work presented in this document tries to rationalize the exact number of bits that could be truncated with an acceptable percentage error in accuracy of the final cluster centers. Nevertheless, the classification of points into N number of classes does not change. Again the hardware implementation was targeted to Annapolis Microsystems Wildstar PCI board with three Xilinx Virtex 1000 FPGAs and 40 MB of ZBT SRAM. The design classifies 614 X 512 pixel images with 10 channels of 12 bits data per pixel into 8 clusters. The design returns the cluster number for each pixel, as well as the accumulated values for each channel of each cluster and the number of pixels in each cluster. One of the processing elements PE1 is used by the design and has interface to the host PCI bus. The registers mapped onto the host PCI bus hold the control signals, cluster centers, and cluster accumulators. The two 64 bit memory ports hold the pixel data, so that 128 bits of the image can be accessed each clock cycle. Each pixel is 120 bits (10 channel/12 bits), and the image is mapped into the memories so that one whole pixel is accessed each clock cycle with 8 unused bits. The design has a 10-stage pipeline and one pixel is classified every clock cycle [1].

Thomas Fry and Scott Hauck of University of Washington, Seattle have done related work on a RC-based system which compresses the data stream before down linking. By developing image compression routines on a reconfigurable platform, they have established it is possible to obtain the computational performance required to compress a satellite’s data in real time and at the same time retain the ability to modify the system post-launch [27]. The algorithm used is the Set Partitioning in Hierarchical Trees (SPIHT) image compression algorithm, which is similar to k-means also regarded as a compression algorithm.

More importantly the hyper spectral data has been represented as a fixed-point number. Similar representation has been considered in this thesis.

Researchers at Los Alamos National Laboratory, USA and IRISA – CNRS, France have implemented a hardware/software co-processing model on a hybrid processor for k-means clustering [6]. The experiments were done on two models of the Altera Excalibur board, the first using the soft IP core 32-bit NIOS 1.1 RISC processor, and the second with the hard IP core ARM processor. A comparison of the performance of the sequential k-means algorithm with three different accelerated versions was reported as an outcome of the experiments. Granularity and synchronization issues were considered when mapping an algorithm to a hybrid processor. The results indicated that a speed-up of 1.8 X was achieved by migrating computation to the Excalibur ARM hardware/software as compared to software only on a Gigahertz Pentium III. Speedup on the Excalibur NIOS was limited by the communication cost of transferring data from external memory through the processor to the customized circuits. The dual port memories of the Excalibur ARM, accessible to both the processor and configurable logic, overcame the limitation and has had the biggest performance impact of all the techniques studied [6][2].

Dominique Lavenier, researcher at IRISA, CNRS France has conducted some independent research of k-means clustering on various prototype hardware and implemented k-means using systolic or linear arrays to exploit algorithmic level parallelism [6].

2.2 K-MEANS CLUSTERING ON OTHER PLATFORMS

Research groups at John Hopkins University, Laurel and University Of Maryland, College Park have looked into efficient k-means clustering heuristics like Lloyd's Algorithm but haven't experimented any of them on scalable architectures or hardware assisted platforms [28].

University Of Minnesota has researched advantages of implementing K-means algorithm on the DANCE (Definitive Axiomatic Notation for Concurrent Execution) Multitude parallel execution architecture and compared it to equivalent MPI based routine and showed improvements. The implementation was based on DANCE program that was invented specifically for the general-purpose parallel processing Multitude architecture. A DANCE program specifies the task without having to explicitly schedule the parallelism [29].

Researchers Inderjit S. Dhillon and Dharmendra S. Modha at the IBM TJ Watson Research Center developed k-means clustering algorithm on a distributed memory multiprocessor environment with message passing models [4].

Researchers S Ray and RH Turi [8] at the Monash University in Australia looked at determination of the number of clusters in K-means clustering and application in color image segmentation in a totally software kind of environment. Other researchers as R.A Schowengedt [9], David Langrebe [21] have conducted experiments related to K-means clustering but haven’t used any kind of hardware assist to upgrade the analysis and performance.

2.3 CHAPTER SUMMARY

This chapter discussed some of the contributions of other research groups relating to k-means clustering and some of the benefits of implementing k-means clustering on a variety of platforms. The rest of this thesis document discusses the approach taken in the current work followed by the implementation of the algorithm on the Pilchard system. The following chapter will introduce the reader the details of the k-means clustering algorithm and the advantages of using the algorithm over a hardware platform.

3. K-MEANS CLUSTERING

In the previous chapter, some related work done in the past with reference to K-means clustering was pointed out. In addition to describing the K-means clustering algorithm in some detail, this chapter presents the massive potential parallelism that can be tapped and few ways of accelerating the algorithm by the way of minor transformations and bit width reduction done on the input vector. Also, two popularly known pre-processing techniques that have been employed in the implementation are summarized.

3.1 ALGORITHM OVERVIEW

This algorithm is widely used in the data-mining field. It is utilized for classification purposes and to discover differences and patterns in both small and large data sets. There exist many different variants of K-means clustering – most of which are variants adapted for special purpose environments [2]. With the growth of data collected on operational and transactional data, the field of data mining has become increasingly important. The growth of data has been accelerated with the commercialization of the Internet and the increased use of personal computer. In this environment collection of individual metrics is relatively cheap and unobtrusive to the user. Companies who have been collecting vast amount of data on consumer habits are now confronted with the dilemma of what to do with all the data. This is where K-means clustering becomes useful. It provides a remedy tailored to this problem and reveals patterns that otherwise are obfuscated and not otherwise discernible [1]. There has been an increasing interest in clustering genomic data and analyzing DNA sequences. K-means is a common solution to the segmentation of multi-dimensional data. However, these large amounts of data sets require large computational capacity. The nature of this problem is ideally implemented on a high performance computing architectural node [19].

Given a set of N pixels, each composed of S spectral channels, and represented as a point in S-dimensional Euclidean space (that is, $x_n \varepsilon R^D$, with n = 1…N); we partition the pixels into K clusters with the property that pixels in the same cluster are spectrally similar [3]. Each cluster is associated with a “prototype” or “center” value which is representative of (and close to) the pixels in that class. One measure of the quality of partition is the within-class variance [3]; this is the sum of squared (Euclidean) distances from each pixel to that pixel’s cluster center. The k-means clustering algorithms (there are several variants) provide an iterative scheme that operates over a fixed number (K) of clusters, while attempting to simultaneously optimize center locations and pixels assignments.

To begin with, the algorithm passes over all the data points, and reassigns each to the cluster whose center it is closest to. After the pass through the data, the cluster centers are recomputed. Each iteration reduces the total within-class variance for the clustering, so it is guaranteed that after enough iteration, the algorithm will converge, and further passes will not reassign points. It bears remarking that this is only a local minimum. There may be an assignment of pixels to classes that produces a smaller within-class variance, but to search all possible assignments (there are $K^N/K!$ of them) would be an impossibly large task for all but the smallest values of N. This problem is known to be NP-complete [3].

3.2 APPROACHES FOR ACCELERATION

One of the most important goals of this work is to be able to utilize hardware units to increase processing speed. The standard software implementation of k-means uses floating-point arithmetic and Euclidean distances. Floating-point arithmetic and the multiplication heavy Euclidean distance calculation are fine on a general-purpose processor, but they have large area and speed penalties when implemented on a FPGA [1] [3] [7]. In order to get the best performance of k-means on FPGA, the algorithm needs to be transformed to eliminate these operations. Manhattan distance as opposed to Euclidean distance is considered in the present implementation. Manhattan distance measures uses adder and absolute value computational blocks that are far less expensive than the multiplication units on the hardware. K-means clustering technique is shown in Figure 3.1.

Fixed-point division operators have been implemented within the hardware fabric and thus have considerable effect in increasing the computational speed. The effects and results are discussed in chapter 6.

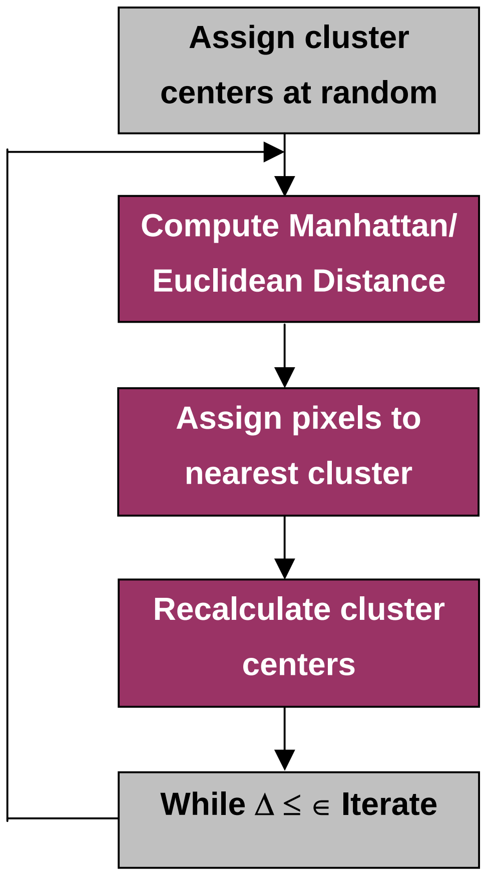

Assign cluster centers at random
Compute Manhattan/ Euclidean Distance
Assign pixels to nearest cluster
Recalculate cluster centers
While $\Delta \leq \in$ Iterate

Figure 3.1: K-means Algorithm

Two important approaches for acceleration have been reported positive for improvements on the hardware units. One, being the use of Manhattan distance measures. Input data bit width truncation being the other. The input data size reduction could be easily extended to data width truncation of intermediate representation upon careful analysis.

Essentially, in K-means clustering, the points are assigned to cluster centers to which they are closest; for the minimum-variance criterion, “closest” is defined in terms of the Euclidean distance. The definition in terms of Manhattan distance is reported to be less accurate but easier to implement and significantly faster on the hardware [1] [3] [7]. To perform k-means iteration, one must compute the distance from every point to every center. If there are N points, K centers, and D spectral channels, then there will be O (N K D) operations [1]. For the Euclidean distance, each operation requires computing the square of a number.
The Euclidean distance has several advantages. For one, the distance is rotationally invariant [1] [7]. Furthermore, minimizing the Euclidean distance minimizes the within-class variance. On the flip side Euclidean distances are more expensive than alternative approaches we are considering for acceleration.

The Manhattan distance, corresponding to $p = 1$, is the sum of absolute values of the coordinate differences [1]. In hardware, calculating the Euclidean distance would be significantly slower than calculating the Manhattan distance. This is due to the fact that a multiplication is required for every channel and every cluster per pixel, so the amount of parallelism that can be exploited in the hardware implementation would decrease drastically. The Manhattan distance is approximately twice as fast in software as Euclidean, but significantly faster in hardware [1] [7].

The research group at Northeastern University, Boston and Space and Remote Sensing Sciences Group at Los Alamos National Labs have conducted experiments and reported results stating that Manhattan measures are acceptable for k-means clustering [1]. Data independent and data dependent experiments were conducted to determine if the use of cheaper metrics were acceptable. Figure 3.2 shows the data independent experiments estimation on how often points would be mis-assigned because of the use of a cheaper metric.

Note that although the relative variance is decreasing for large values of D, the rate of misclassification is monotonically increasing. For the Manhattan metric, the misclassification rate saturates at about fifteen percent, but for the Max metric, the error rate begins to approach fifty percent. That is, for very large dimension D, the Max metric is not much better than just assigning points to clusters at random. As expected, the linear combination performs better than the Manhattan distance. The improvement is substantial for smaller dimensions (D <10), but the difference becomes small for large dimensions.

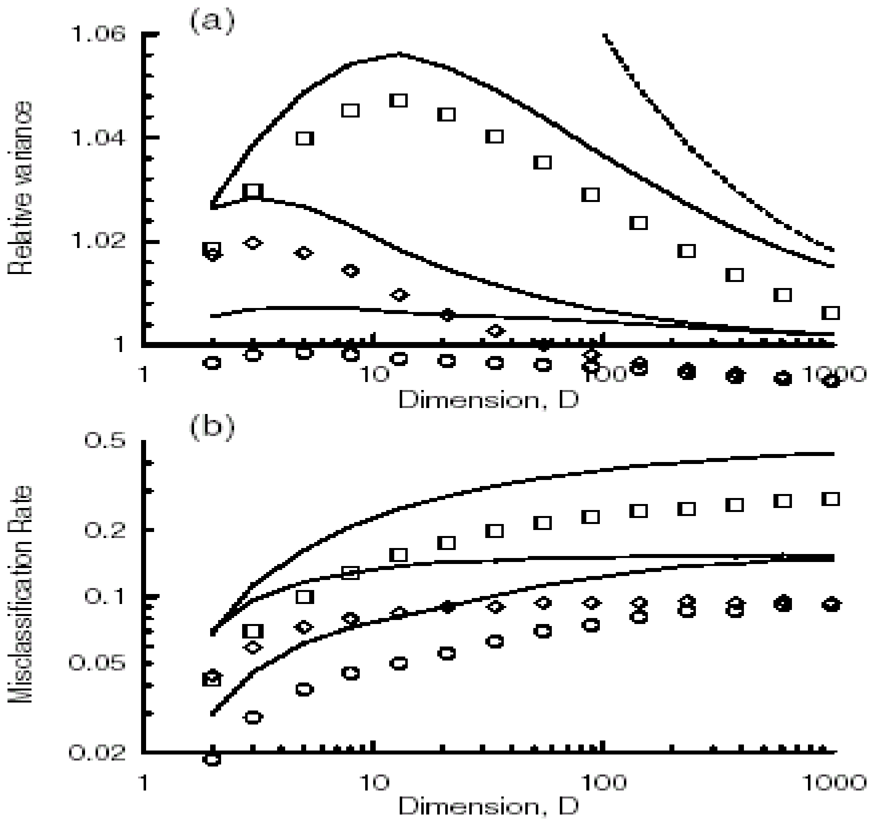


Figure 3.2: Misclassification Rate for Different Distance Metric [1]

The research group also performed data dependent experiment on AVIRIS data sets. Each AVIRIS image was classified with each of the distance measures, so there were twenty trials (5 images * 4 distance measures). Each trial used K = 16 clusters and stopped after convergence or fifty iterations. The within-class variance for the AVIRIS data cubes was measured with most of the images exhibiting less than a 6% increase for using Manhattans over Euclidean measures [1].

One of the major improvements in the use of technique is the capability that has been developed to establish the amount of truncation and still retain acceptable within-class variance. Previous researchers have reported that small amounts of truncation, figure 3.3 can vastly improve the performance of huge datasets comprising of hundreds and thousands of spectral information [1][17]. The current work attempts to quantify the term ‘small amount’ so as to answer the question, how small is small. To approach this problem, a fixed-point version of the algorithm is developed both full precision and truncation applied. The truncation of the bits is increased and run until the algorithm breaks. In other words till the algorithm puts out an acceptable within-class variance. With the aid of A|RT collector class and statistics class summarized briefly in chapter 5, overflows were detected and the iteration stopped when detected. The bit width at that point is taken as the final bit length to represent a data point with maximum truncation.

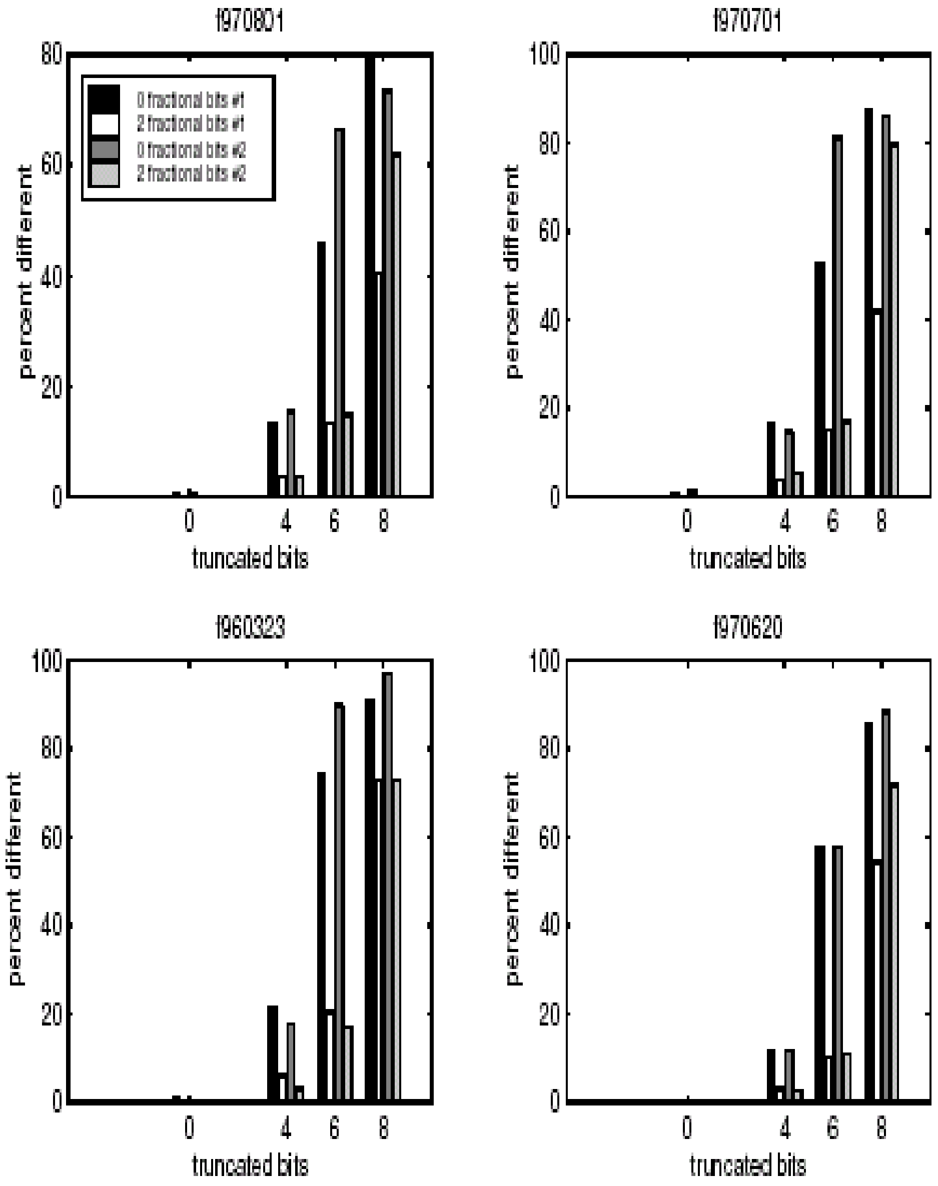


Figure 3.3: Percentage Difference vs. Amount of Truncation for Different Samples [1]

The other single most important approach is the implementation of division operators within the hardware fabric. Researchers who have done similar work have implemented division operators, a part of the k-means clustering on the host computer. This thesis work tries to analyze how much improvement in performance gain was obtained in eliminating some of the communication delay between the host computer and the pilchard subsystem. Additionally, data-path area and latency introduced in the design cycle has also been reported.

3.3 DATA SETS

The hardware implementation deals with the classification of synthetically generated datasets of different sizes. It also deals with classification of different materials based on the hyper-spectral reflectance data of the materials. The raw reflectance data is obtained from the JPL as observed by the earth looking satellites. There are three classes of materials namely manmade materials, minerals and soils. The high-order statistical and wavelet based features are extracted [section 3.3.2] from each sample to perform a feature space. Unsupervised learning algorithms are applied on the feature space to classify the given sample.

3.3.1 RAW DATA

Two kinds of data sets, synthetic and real data from JPL spectral library have been used to verify the design on the hardware. Initial cluster centers generated at random with mean of 0 and standard deviation of 1. The size of the matrix is 3 x M, where the number three represents the number of classes in the cluster and M is the number of principal features of the cluster. Normally, an image analyst determines the number of clusters.

The raw data consist of wavelength versus reflectance data of different materials obtained from Jet Propulsion Laboratories (JPL) spectral library [20]. This JPL Spectral Library includes laboratory reflectance spectra of 160 minerals in digital form. Data for 135 of the minerals are presented at three different grain sizes: 125-500µm, 45-125µm, and <45µm. This study was undertaken to illustrate the effect of particle size on the shape of the mineral spectra. Ancillary information is provided with each mineral spectrum, including the mineral name, mineralogy, supplier, sampling locality, and our designated sample number. In the original publication the spectra were separated into classes according to the dominant anion or anionic group present, which is the classification scheme traditionally used in mineralogy. The three main classes include manmade materials, minerals and soils [20].

The data for three classes exist in three different directories with each directory containing the text files for each material. Each sample text file consists of a header describing the chemical information and the range of wavelengths used etc. For feature extraction purposes these headers are removed with only wavelength and the reflectance data remaining in the text files.

3.3.2 DATA PRE-PROCESSING

A number of preprocessing steps are performed before data is used as an input vector to the hardware blocks. Two important steps are performed to transform the ram vector to a more usable format.

1. Feature Extraction
2. Principal Component Analysis (Critical features selection)

3.3.2.1 FEATURE EXTRACTION

The feature extraction is the main part of the project since the accuracy of the classification depends on deriving the non-correlated independent features, which distinctly describe the material. In this project we derived the high order statistics of each data sample describing the shape of the wavelength versus reflectance data and wavelet based features.

3.3.2.1.1 HIGHER ORDER STATISTICS

The hyper spectral data for each material exists in wavelength versus reflectance data format and thus studying about the statistics of this two dimensional data provides valid features since each different material will have particularly different reflectance and thus deriving the statistics will provide efficient index for classification. The high order statistics include the following four:

$$\text{mean} : \mu_{shape} = \frac{1}{S}\sum_{j=0}^{N-1} w_j r_j$$

$$\text{standard deviation} : \sigma_{shape} = \sqrt{\frac{1}{S}\sum_{j=0}^{N-1}\left(w_j - \mu_{shape}\right)^2 r_j}$$

$$\text{skewness} : \gamma_{shape} = \frac{1}{S}\sum_{j=0}^{N-1}\left(\frac{w_j - \mu_{shape}}{\sigma_{shape}}\right)^3 r_j$$

$$\text{kurtosis} : \beta_{shape} = \frac{1}{S}\sum_{j=0}^{N-1}\left(\frac{w_j - \mu_{shape}}{\sigma_{shape}}\right)^4 r_j$$

The mean gives the average reflectance of the data, standard deviation gives the variation of the data in the distribution, skewness gives the relative symmetry of the data and kurtosis gives the relative flatness of the graph of the given data.

3.3.2.1.2 WAVELET BASED FEATURES

Wavelet transform gives the time-frequency information of the given signal. This two-domain representation makes it a useful transform in signal analysis. This technique uses decomposition of the original signal into discrete wavelet coefficients. The wavelet coefficients effectively represent the signal completely and the knowledge of these coefficients we can reconstruct the whole signal. In this projects the statistics and the energy of these coefficients is derived to act as features for classification purposes. The original data is decomposed using the Daubechis wavelet is used for decomposing the reflectance data by four levels. The mean, variance and energy of the coefficients form each level are derived and used as features.

Thus the feature extraction gives 16 independent features for each sample text file or each material. The feature space is constructed with three different classes. Before the pattern classification is done the feature space is normalized so that the adjacent features are in comparable scale. Each column of the feature space is independently normalized using the formula

$$f_i = \frac{f_i - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation.

3.3.2.2 PRINCIPLE COMPONENT ANALYSIS

Principle component analysis is done to eliminate the correlated feature vector, thus making the feature space into useful minimum thus helping in dimensionality reduction. The algorithm of the PCA is as follows:

1. The covariance matrix of the feature space is calculated.
2. The Eigen values and the corresponding Eigen vectors are calculated.

3. The Eigen value matrix is arranged in ascending order of magnitude correspondingly arranging the Eigen vector matrix.
4. The sub matrix of Eigen vectors is chosen to be multiplied with the dataset depending on K where is k is obtained from the following formula, where n is the loss of energy.

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{M} \lambda_i} >= 1 - \eta$$

5. The obtained set of the Eigen vectors is multiplied with the training set to reduce the dimensionality of the data set.

3.4 CHAPTER SUMMARY

This chapter discussed the details of the k-means clustering algorithm and small transformations that could be applied to the algorithm to achieve better performance on the hardware. Also, a brief description of the hyper-spectral data pre-processing steps has been described. The next chapter discusses the platform for implementing the algorithm. Additionally, reasons for choosing the platform and design issues relating to the system are also discussed.

4. HARDWARE PLATFORM

In the previous chapter, a few variants of k-means clustering were introduced and an overview of the most common variant of the algorithm was discussed. Also, different algorithmic level transformations for acceleration and the organization of data sets that the algorithm operates on were detailed. This chapter introduces the hardware assist that has implemented for k-means clustering.

4.1 WHY PILCHARD?

Mainly, there are three reasons for choosing the Pilchard system. For one, the architecture of the pilchard system is such that it mounts on the DIMM slot as opposed to the PCI. The DIMM slot has a greater bandwidth and speed and hence is preferred over a PCI bus. Secondly, the host interface design takes relatively less time to comprehend and use. Hence, the learning curve is fairly short when compared to complicated interfaces of other prototype boards such Wildforce and Wildcard that are available at the Microelectronics Laboratory in the University of Tennessee. In this way, users can concentrate more on their designs rather than on understanding complicated host interfaces. Third, and by far the last main reason for choosing pilchard is that it’s a fairly new hardware assist for educational and research use and is readily available for use at the University of Tennessee. Unlike the Wildforce and the Wildcard hardware platforms that have multiple FPGA on the mezzanine card, the pilchard system has one big virtex FPGA on the board. This eliminates the use of any kind of partitioning methods for fairly big designs. Also, de-skew techniques for off-chip signals and vice versa can be avoided.

4.2 PILCHARD SYSTEM

Pilchard board is a high performance-computing card that has been used to target the algorithm down to Xilinx Virtex 1000e chip part [18]. This chapter briefly describes the board and the architecture of the FPGA part embedded in the mezzanine card. Figure 4.1 shows a picture of the Pilchard card.

This card developed by researchers led by Dr. Philip Leong [18] at the Chinese University of Hong Kong is not an off the shelf commercial board but a platform for the research environment and educational purposes. As we go along the way

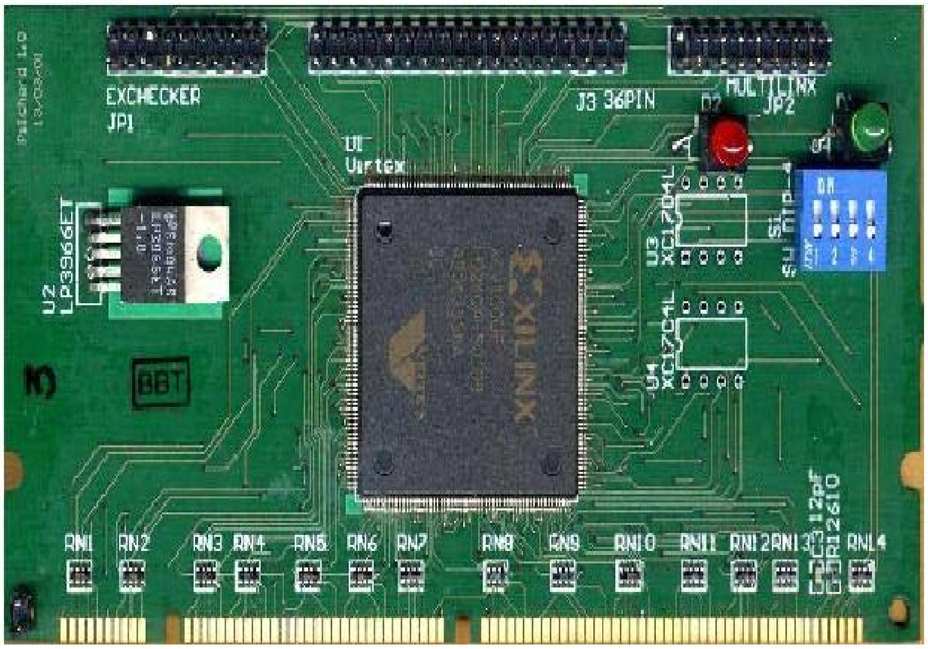


Figure 4.1: The Pilchard Card [18]

in successfully mapping the design to the FPGA on the Pilchard board, the advantages and the downsides of the system are discussed. Figure 4.2 describes some of the salient features of the Pilchard system.

It has been reported that the system has an efficient interface, is a low cost model and is suitable for various applications that include cryptosystem, image processing and speech processing, clustering and rapid prototyping. Other significant features include high data transfer rate capability due the DIMM RAM interface, communication speed of up to 133MHz directly with the CPU, shorter learning curve, 27 bits external I/O outlets including clock signal outs for hardware debugging and monitoring etc. Xchecker cable is required to download the bit stream to the FPGA.

The users are provided with a set of VHDL source and other useful files.

1. **pilchard.vhd:** The top level VHDL code which wraps the core and interfaces with the DIMM slot directly.
2. **pcore.vhd:** This is a standard template and all user designs go inside this. Figure 4.3 shows the entity port list of the VHDL core.

Figure 4.4 shows the internals of the Virtex 1000e chip part and an overview of the Virtex architecture is described in section 4.3.

Host Interface	DIMM interface
	64-bit Data I/O
	12-bit Address Bus
External (Debug) Interface	27 bits I/O
Configuration Interface	X-Checker, Multi-Link and JTAG
Maximum System Clock	Rate 133MHz
Maximum External Clock	Rate 240MHz
FPGA Device	XCV1000E-HQ240-6
Dimension	133mm X 65mm X 1mm
OS Supported	GNU/Linux
Configuration Time	16s (using Linux download program)

Figure 4.2: Features of Pilchard Platform [30]

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity pcore is
port (
      clk: in std_logic;
      clkdiv: in std_logic;
      rst: in std_logic;
      write: in std_logic;
      addr: in std_logic_vector(13 downto 0);
      din: in std_logic_vector(63 downto 0);
      dout: out std_logic_vector(63 downto 0)
      );
end pcore;
```

Figure 4.3: Entity Declaration of pcore.vhd

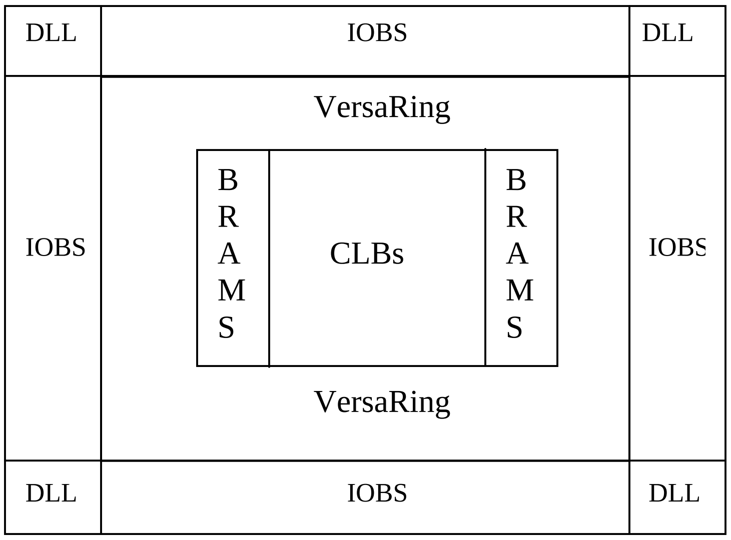


Figure 4.4: Virtex Architecture Overview [11]

3. **pilchard.ucf:** This a constraint file to the place and route program during the process of mapping the design to logic cells inside the FPGA. Pin assignments and the clock period of the design are specified here.
4. **iob_fdc.edif:** A synthesized netlist for the I/O blocks in pilchard.vhd

4.3 VIRTEX 1000E ARCHITECTURE

Virtex architecture comprises an array of configurable logic blocks (CLBs) surrounded by programmable input/output, all interconnected by fast, versatile routing resources.

- CLBs provide the functional elements for constructing logic
- IOBs provide the interface between the package pins and the CLBs
- The large amounts of routing resources allow the largest and the most complex designs to be mapped to these elements. Virtex FPGAs are SRAM-based devices that provide better performance than previous generations of FPGA.

Designs can achieve synchronous system clock rates up to 200 MHz including the I/O [18]. Furthermore; I/O's are fully compliant with PCI specifications.

CLBs interconnect through a general routing matrix (GRM). The GRM comprises an array of routing switches located at the intersections of horizontal and vertical routing channels. The Virtex architecture also includes the following circuits that connect to the GRM.

- Dedicated block memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control

3-state buffers (BUFTs) associated with each CLB that drive dedicated segmentable horizontal routing resources.

4.3.1 CONFIGURABLE LOGIC BLOCK

Configurable logic blocks have four logic cells (LC) as basic components. An LC includes a 4-input function generator; carry logic, and a storage element. The CLB also contains function generators to provide functions of five or six inputs. Other elements in a Virtex Slice include

- Look-Up Tables
- Storage Elements
- Additional Logic
- Arithmetic Logic
- BUFTs
- Block SelectRAM

The Block SelectRAM memories are organized in columns. All Virtex devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Virtex device 64 CLBs high contains 16 memory blocks per

Table 4.1: Virtex-E Block SelectRAM Amounts [11]

Virtex-E Device	# Of Blocks	Block SelectRAM bits
XCV50E	16	65,536
XCV100E	20	81,920
XCV200E	28	144,688
XCV300E	32	131,072
XCV400E	40	163,840
XCV600E	72	294,912
XCV1000E	96	393,216
XCV1600E	144	589,824
XCV2000E	160	655,360

column, and a total of 32 blocks. Table 4.1 shows the amount of block SelectRAM memory that is available in each Virtex device.

4.3.2 DELAY-LOCKED LOOP (DLL)

A DLL purpose in the Virtex device is to eliminate skew between the clock input pad and the internal clock-input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock and automatically adjusts a clock delay element. In addition to this, the DLL provides advanced control of multiple clock domains. It provides four quadrature phases of the source clock, in addition to provisions of doubling the clock or dividing the clock by multiple factors of 1.5, 2, 2.5, 3, 4, 5, 8 or 16.

The DLL also operates as a clock mirror. By driving the output from a DLL off chip and then back on again, the DLL can be used to de-skew a board level clock among multiple Virtex devices. In order to guarantee that the system clock is operating correctly prior to the FPGA starting up the configuration, the DLL can delay the completion of the configuration process until after it has achieved lock.

4.4 DESIGN ISSUES OF K-MEANS ON PILCHARD

There have been quite a few successful implementations like encryption engines that have been reported with the Pilchard [18]. However, there are still few critical design issues with the system, particularly with pattern classification algorithms like K-means. These issues and limitations offset some of the advantages pertaining to the Pilchard system.

4.4.1 SPEEDS AND FEEDS

Later in this chapter, a methodology flow of the design and ad-hoc approaches to decide number of bits to represent data points are discussed. The smaller the number of bits required to represent a dataset, better would be the performance in terms of speed and data-path area. However, accuracy needs to be traded off. Arguably, since the k-means clustering technique is used for quick view analysis and data compression, the speed needs for massive number crunching has been treated more important in the research community. Figure 4.5 refers to this problem.

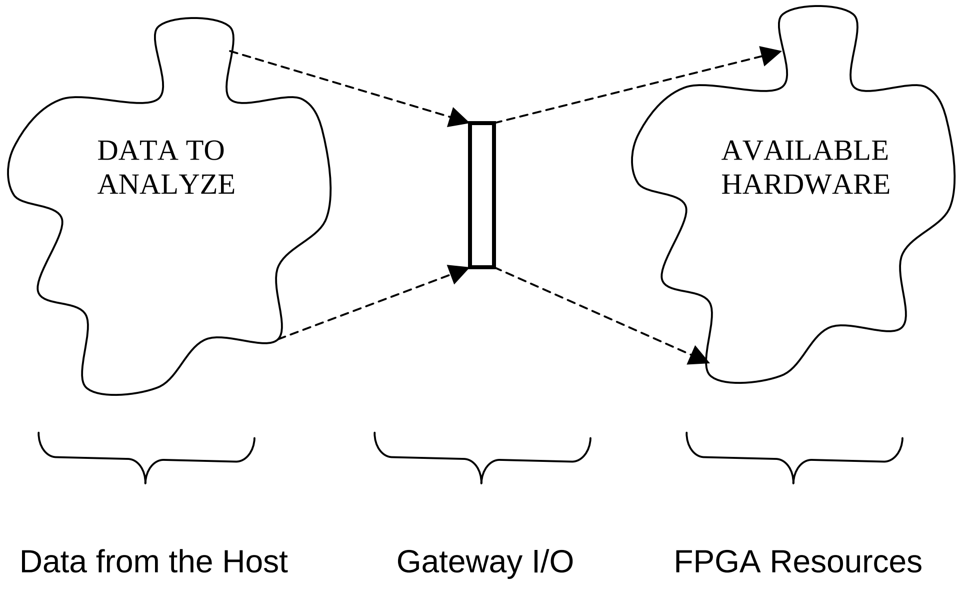


Figure 4.5: Problem of Speeds and Feeds

Smaller I/O indirectly limits the speedup but smaller bit representations can solve this problem to a certain extent. This is because the I/O channel now transports more of such data points that are packed together in one write cycle. However, if the pixel representation cannot be further reduced to smaller bit width; an increase in copies of hardware computational blocks does not enhance performance. Hardware units could be optimized for best achievable results but is almost always limited by the I/O of the prototype. The Virtex 1000e FPGA has about a million gates that the user designs could use. The K-means core takes about 10% of the Virtex chip and therefore, there is a natural thinking that the core engine could be replicated at least nine times to take advantage of the plenty of hardware resources. This is given the fact that processing of one observation point is independent of the processing of another and hence parallelism can be exploited at the algorithmic level. This proposition is, however, invalid, since the gateway I/O is band limited to a 64-bits transfer for every write64() transaction. The data can be packed if fewer bits represent the data point. Such data packing can be a solution to the bandwidth saturation problem.

4.4.2 LIMITATIONS AND BOTTLENECKS

Bottlenecks that limit performance numbers are

1. Limited block RAMs available in the chip part. Unlike other usual prototyping boards, the pilchard system is devoid of on board RAMs that would enable the user to store the entire image on the board before starting hardware operations. This means that the entire image or any data set needs to be stored within the FPGA. More importantly, the onboard RAMs normally are DMA mapped to the memory of the host and hence cuts down overheads cost in data transfer cycles.
2. Limited support for handshaking protocols complicate development of streaming applications on the hardware.
3. The core template for the I/O registers has 14bits of address lines but only 8bits could be used for hardware addressing.

I/O is a 64bit bi-directional data bus, so data transfer from the host to the re-configurable unit is bandwidth limited. Since the entire image is stored within the FPGA before the start of hardware operations, a number of cycles of latency are introduced from the data transfer from the host to the re-configurable blocks. The latency introduced increases as the data size increases and this eventually offsets the performance benefits obtained by exploiting inherent parallelism. One way that has been used to solve the problem is to transfer as many data points as possible per write cycle.

4.5 CHAPTER SUMMARY

This chapter briefly described the reasons for choosing the Pilchard development board for the k-means clustering implementation and the architecture of the Pilchard system as well. In addition to this, hardware subsystems within the Pilchard, such as the architecture of the Virtex 1000e FPGA have been described. The design issues and bottlenecks of the system have also been identified and presented. The next chapter discusses the approach and methodology adopted for the implementation.

5. METHODOLOGY

In the previous chapter, a few hardware platforms were mentioned and Pilchard was chosen as the current hardware assist for the clustering implementation. In addition to this, the architecture of the target Virtex 1000 e chip was summarized and the limitations of the system with respect to the k-means algorithm were pointed out. The chapter discusses the methodology and approach taken in this design with respect to the Pilchard system as the design platform.

5.1 INTRODUCTION

The Methodology and approach in problem solving is a key to an effective and productive design. This is particularly true for a design that needs to meet the 'time-to-market' constraints. This chapter discusses the top down approach with a focus on finding a near optimal implementation on the hardware assist. Here, the optimality is primarily based on the speed of the computation and the correctness of the classifier.

Bit truncation techniques during the input and intermediate stages have been shown to cause dramatic increases in performance and speed [1]. A protocol has been established to quantify the maximum number of the bits that could be truncated at different levels. Hardware implementations with fewer bits tend to decrease the data path area and the critical path of the design, thereby enhancing the performance and speed of the design. In addition to the quantifying bit truncation, tested approaches that have worked well with other platforms have been implemented on the Pilchard system.

Figure 5.1 shows the methodology flow of the design. The golden code is eventually translated to hardware units at the output of the flow.

5.2 MATLAB GOLDEN CODE

A given Matlab code shown in Figure 5.2 is treated as a golden code and all the results of the final implementation is based on the results of this Matlab code. The data set is a floating-point number and all the iterative operations are floating-point operations. An attempt has been made to port the algorithm to floating point C and this code was later modified into a fixed-point version. The floating point C was developed and the results are compared with the Matlab version. Since, at this point the operations in both Matlab and C are floating-point, the results had to match with 100% accuracy and was verified to be the same. A fixed-point version of C was then developed using the fixed-point package available in the A|RT library. While in the development stage, an iterative search was performed to identify minimum bit-width required to avoid any overflow of bits. The following figure shows the flowchart of an iterative search.

The bit length of the fractional part of the number was decided by comparing the results of the floating-point C versus the fixed-point C for an acceptable percentage error.

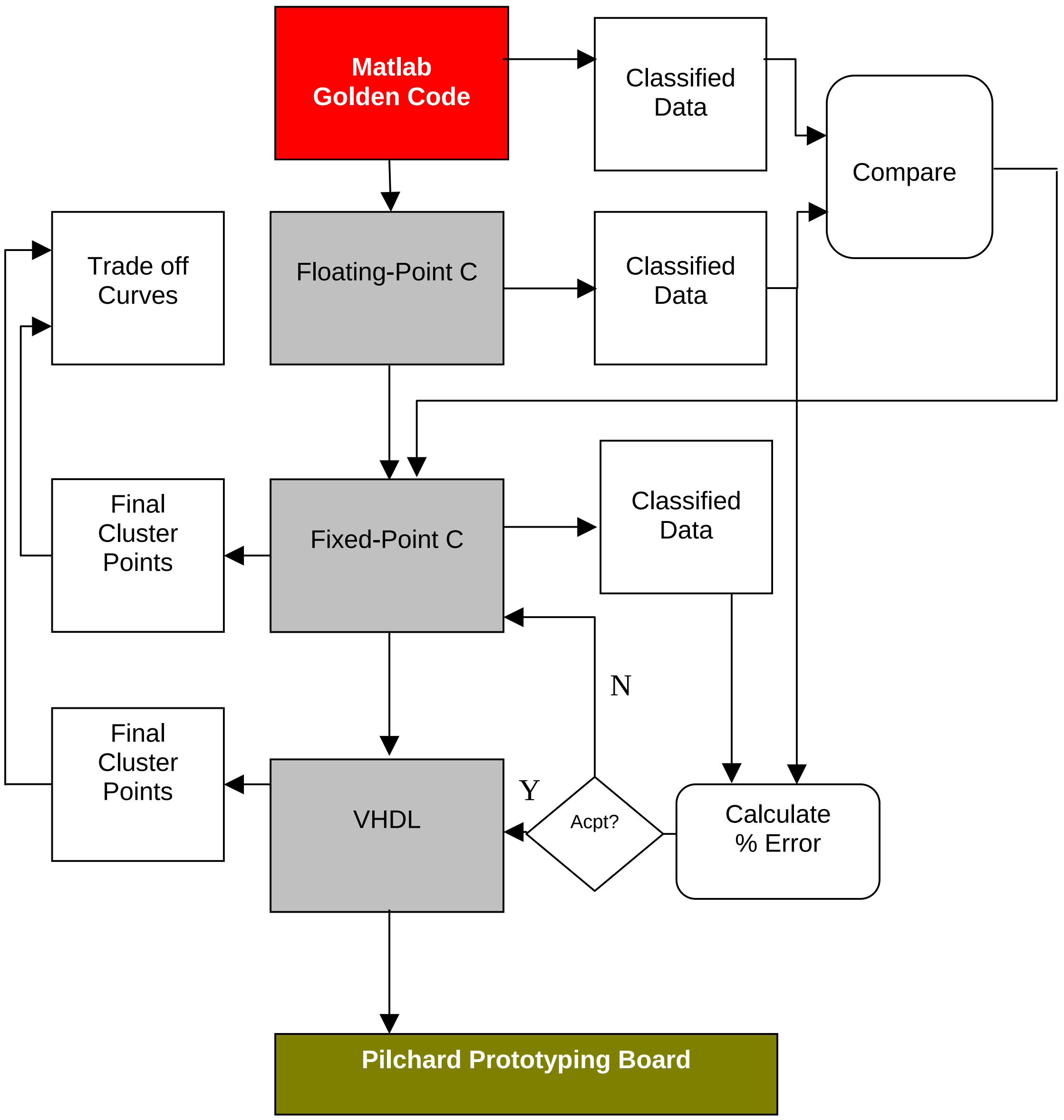


Figure 5.1: Methodology Design Flow

```
[m, n] = size(X);
% generate k random number of n-dimension as initial mean value
R1 = randn(k, n);

% start classification, initial class assignment
Y(:,1:n) = X;
diff = Inf;
while diff > 0
% choose the shortest distance to cluster
 for i=1:m
 D = sqrt(sum((repmat(X(i,:),k,1)-R1).^2, 2));
 [S, I] = sort(D);
 Y(i,n+1) = I(1);
 end
% recalculate the mean
 R2 = zeros(k, n);
 for j=1:k
 p = 0;
 for i=1:m
 if Y(i,n+1) == j
 R2(j,:) = R2(j,:) + Y(i,1:n);
 p = p + 1;
 end
 end
 if p > 0
 R2(j,:) = R2(j,:) / p;
 end
 R(j) = norm(R1(j,:)-R2(j,:));
 end
 % calculate the largest difference
 diff = max(R);
 R1 = R2;
 fprintf('Mean vector is \n');
 disp(R2);
 end
 mu = R2;
```

Figure 5.2: Golden Matlab Code

5.3 FIXED-POINT C

One of the main things in the present work is an attempt to be able to say that for an acceptable error percentage, n times the speed of computation of standard software implementation, where n >1, is the speedup of the algorithm implemented in hardware over a conventional microprocessor. K-means clustering inherently is robust and thus is insensitive to small changes in the ground conditions. This is to say that a predetermined small level of error is acceptable as long as a considerable amount of speedup results. The trade off factors and the numbers related to this are discussed in the chapter 6.

5.3.1 BIT-WIDTH TRUNCATION

A procedure was developed to determine exactly the number of bits that could be used to minimally represent a data point. The minimum bit representation helps achieve the fastest k-means algorithm on the hardware but not necessarily the best quality results. However, if the classification of data points does not change then the experiment is treated as successful regardless of the accuracy of the convergence. A trial and error attempt is made to repeat the process until an acceptable optimum point on the speedup vs. percentage error was determined.

Two experiments were conducted to individually identify the minimum number of integer and fractional bits to represent the input data point. Two separately run algorithms to determine minimum widths are described in five steps as shown below. A simple example is demonstrated to illustrate the idea of bit truncation method. As the number of bits required to represent the pixel is reduced the data-path area on the FPGA drastically roll off. A more compact data-path normally tends to reduce the critical path and increase the computation speed.

To determine number of integer bits

Step1 - Start with the fixed point of type <16, 0> with 16 integer bits and 0 fractional bits.

Step2 - Run the data points through the fixed-point k-means code.

Step3 - Feed the results to the statistics tool to check for overflows and validate cluster classifications with the results from the golden code.

Step4 - If overflows were not detected and the data correctly classified; truncate the data to 15 integer bits; < 15, 1>

Step5 - If detected, step back to the previous iteration to decide the minimum bit representation.

Figure 5.3 is the block diagram of the procedure developed to determine the minimum integer bits and Figure 5.4 is a similar diagram to determine minimum fractional bits.

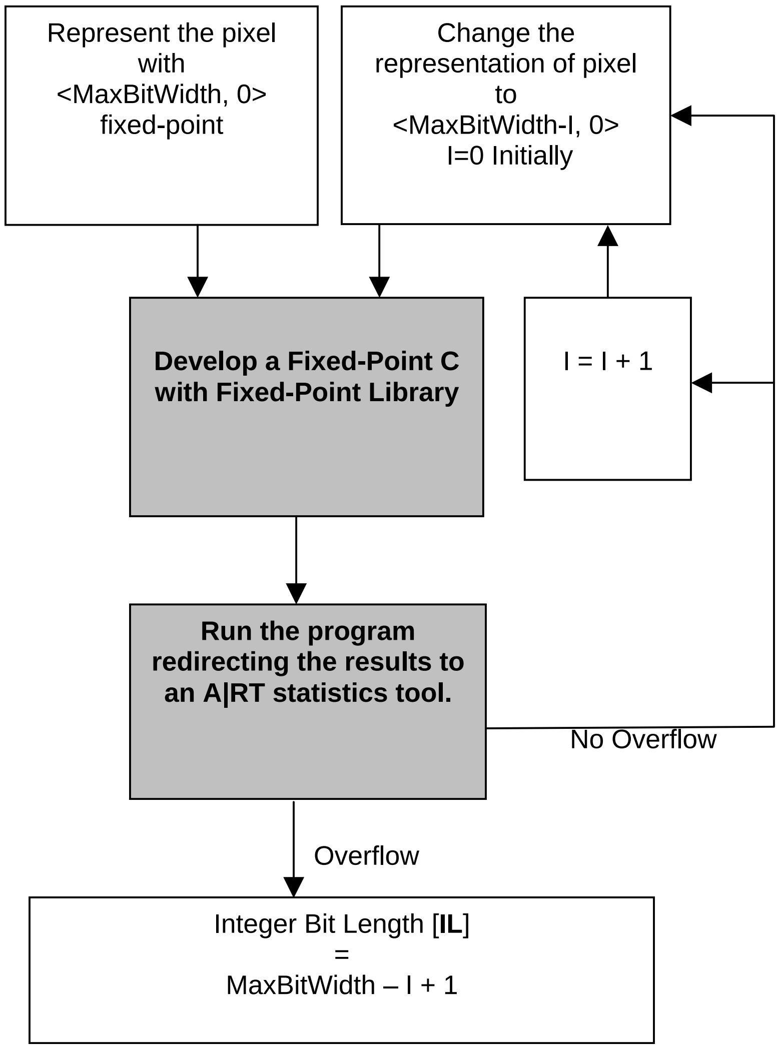

Represent the pixel with <MaxBitWidth, 0> fixed-point
Change the representation of pixel to <MaxBitWidth-I, 0> I=0 Initially
Develop a Fixed-Point C with Fixed-Point Library
I = I + 1
Run the program redirecting the results to an A|RT statistics tool.
No Overflow
Overflow
Integer Bit Length [IL] = MaxBitWidth – I + 1

Figure 5.3: Procedure for Determining Minimum Integer Bits

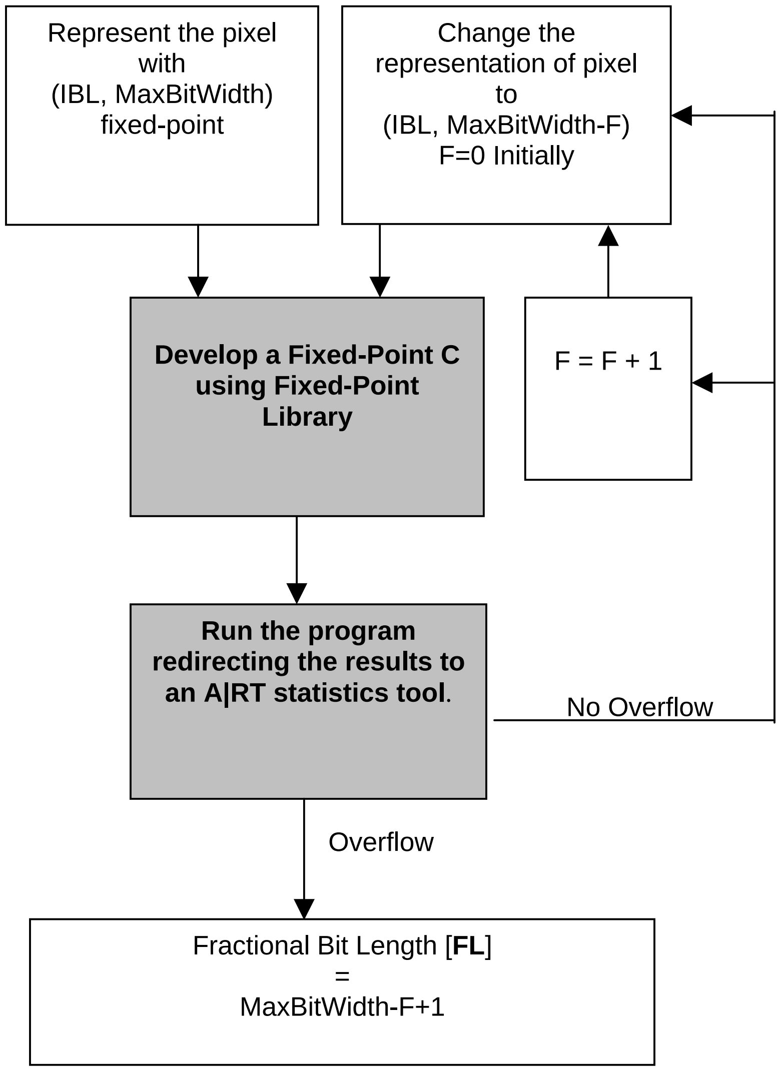

Represent the pixel with (IBL, MaxBitWidth) fixed-point
Change the representation of pixel to (IBL, MaxBitWidth-F) F=0 Initially
Develop a Fixed-Point C using Fixed-Point Library
F = F + 1
Run the program redirecting the results to an A|RT statistics tool.
No Overflow
Overflow
Fractional Bit Length [FL] = MaxBitWidth-F+1

Figure 5.4: Procedure for Determining Minimum Fractional Bits

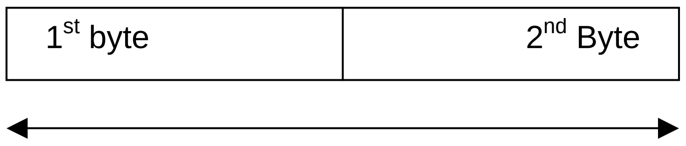


Figure 5.5 Full Precision (I + F) 16-bit Fixed Point

Consider a single dimensional 16-bit pixel as shown in Figure 5.5. Also, assume 16-bit representation as full precision and the normalized range of pixel values -1 to 1.

Determine number of fractional bits; A similar approach as the previous one is used to determine the fractional part.

Step1 - Start with the fixed point of type <2, 14> with 2 integer bits and 14 fractional bits.

Step2 - Run the data points through the fixed-point k-means code.

Step3 - Feed the results to the statistics tool to check for overflows and verify cluster classifications with the results from the golden code.

Step4 - If overflows were not detected and the data correctly classified; truncate the data to 13 fractional bits and so on; < 2, 13>.

Step5 - If detected; go back to the previous iteration to decide the minimum bit representation. If the classification breaks and or the overflows detected at lets say F=10, then the minimum fractional bits needed would be F=11;

The end result obtained is shown as in Figure 5.6.

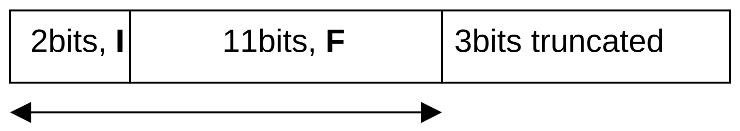


Figure 5.6: Modified 13-bit Representation of Pixel

Therefore the total number of bits needed to fully represent the pixel value would be 13bits. Right now this complete process is manually performed, but can be easily automated using scripting utilities. The overflow statistics is collected over different iterations with the help of fxpStatistics collector class described briefly below. The class is available in the A|RT fixed-point library.

5.4 A|RT BUILDER

A|RT Builder [31] is an electronic design tool that translates a C-based functional specification of an algorithm into an RTL (Register Transfer Level) HDL description. The input to A|RT builder is a description of an algorithm, expressed in a subset of C, optionally enhanced with fixed-point classes as provided by A|RT library. The fixed-point data types and operators in this library give the designer full control over the dimensions of the data path operators.

5.4.1 'fxpStatistics' COLLECTOR CLASS

A|RT Library [31] also provides an auxiliary class, `fxpStatistics`, of which instances can be hooked up to instances of the Number class (and its derivatives of course). They will then be notified when an A|RT Library variable is constructed, destructed, read from or written to. The `fxpStatistics` protocol class is typically used to collect statistics on the dynamics of the fixed-point variables, as well as to build special profiling and analysis functions.

5.4.2 OVERFLOW LOGGING

The statistics class has a built-in utility that automatically performs overflow logging on selected variables. Overflow occurs when an attempt is made to store a value that exceeds the range-capacity of a given variable. The Number class automatically keeps track of such an event, and sets a status flag in the `NbrRef` object that is passed to the statistics class. So, an `overflowDetect` class is created, which implements the `write()` protocol class such that the overflow flag is checked. For every overflow event a message is produced and the overflow counter is incremented.

5.4.3 ‘`fxpTrace`’ STATISTICS CLASS

This `fxpTrace` class can be used to gather statistics on A|RT Library variables, even if one does not have C++ knowledge. This class has been built in inside A|RT Library, and only requires the use of `fxptrace.h` as an include file. The `fxptrace.h` file can be found in the A|RT Library installation. Note that the statistics information cannot be gathered from global variables. The statistics output resulting from running the application code is produced in a file called `trace.out`.

5.5 CHAPTER SUMMARY

This chapter describes the approach adopted for implementation of the algorithm on the Pilchard. Procedures to determine minimum bit widths and different steps involved in taking the given golden code down to the bit level have been discussed. The next chapter describes the details of the actual parameterized implementation. Constraints and pragmatic considerations of the implementation have also been described.

6. IMPLEMENTATION AND RESULTS

This chapter deals with the hardware implementation of the classifier, k-means clustering and compares it with the standard software version implemented on a P3/Linux platform. It begins with details about the architectural block diagram description followed by the description of the memory and the core models with functional simulations. Subsequently, timing simulations are shown to meet time closure constraints that were specified earlier on. A variety of scripting utilities were employed to solve this iteratively tedious problem.

6.1 INTRODUCTION

In Figure 6.1 the Pilchard development board is shown to illustrate the architecture of the design. Different colors are shown to represent multiple clock circuits that have been incorporated in this design. The portion of the architecture shaded with plum color runs at the system speed of 133 MHz and the blue portion of the circuitry runs at half the system clock speed. The inputs are buffered into the dual port memory, which feeds into the core circuit at a speed that the design can handle. The current experiments show that the clock speed could be increased to about 66MHz, which is half the system clock speed.

The pilchard has clock DLL’s set up in a way that different factors of the system clock can be used to drive the design. Factors of 2,3,4,5 and so on could be used, as design gets bigger and complicated.

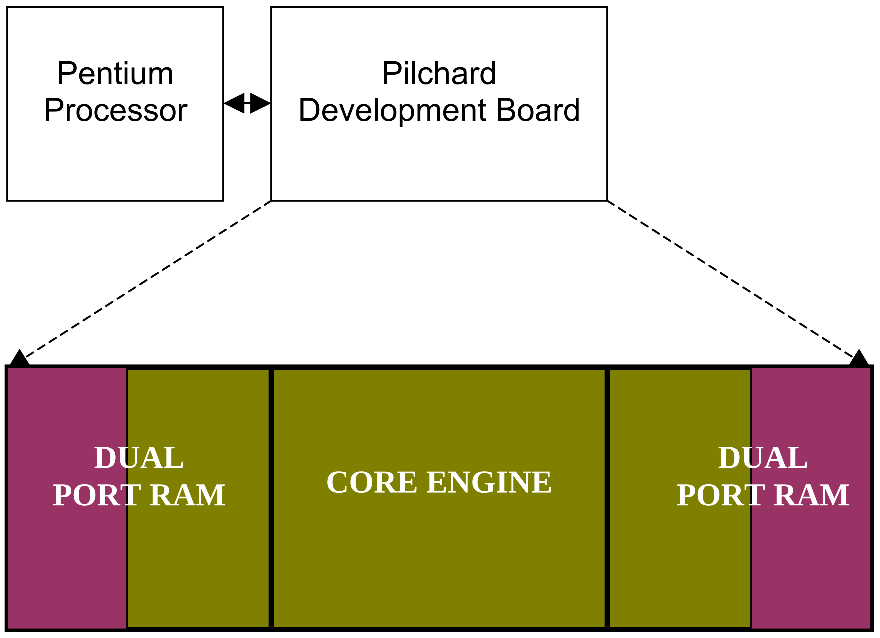


Figure 6.1: Block Diagram of the Implementation

The next section introduces the design flow procedure and the other portions chapter deals with the design details followed by the presentation of results and discussions.

6.2 DESIGN IMPLEMENTATION FLOW

A conventional design flow is used to implement the design on hardware. Figure 6.2 is the flow procedure block diagram that illustrates the steps involved in translation a hardware description language to bit pattern that can be loaded into the hardware via the Xchecker cable. Two important steps namely synthesis and place and route are involved.

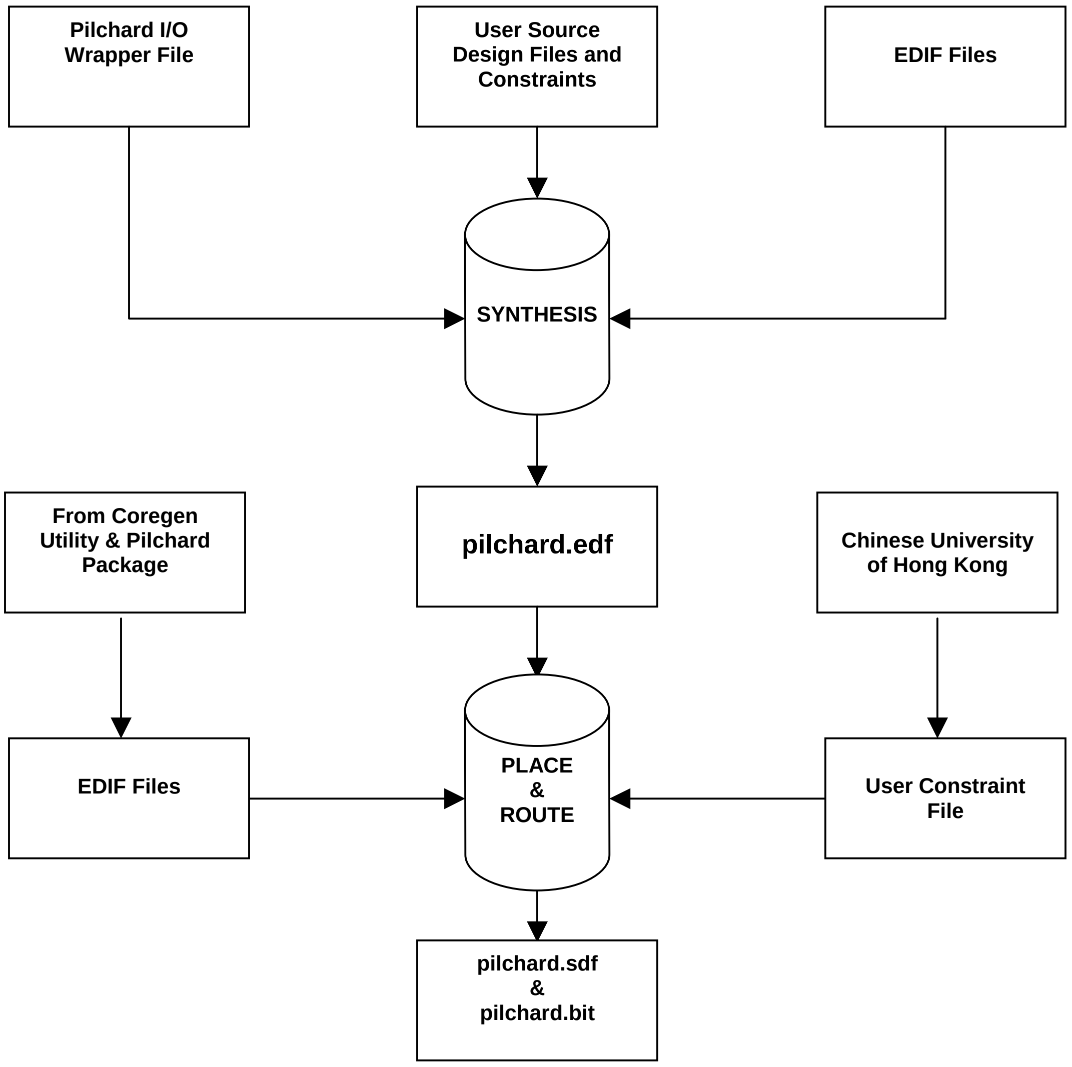

Pilchard I/O Wrapper File
User Source Design Files and Constraints
EDIF Files
SYNTHESIS
From Coregen Utility & Pilchard Package
pilchard.edf
Chinese University of Hong Kong
EDIF Files
PLACE & ROUTE
User Constraint File
pilchard.sdf & pilchard.bit

Figure 6.2: Design Implementation Flow

The inputs to the synthesis tool are the design files developed by the user along with the pilchard wrapper file and any other available synthesized netlist developed by the user earlier on. Two synthesis tools, in this case, FPGA Compiler and Synplify are used to synthesize the design to pilchard.edf. One of them cut the design area by almost 30%. The synthesized netlist is then fed into the Xilinx Place and Route tool with certain placement constraint included in the User Constraint File. During the MAP process, the User Constraint File provided by the developers of Pilchard and any user-defined constraints translates to a Physical Constraint File. The PAR, TRACE, BITGEN and XPOWER programs use this constraint file to create a characterized bitstream. The design is also back annotated for post layout gate level verification.

6.3 DESIGN VERIFICATION FLOW

Functional and post layout gate level simulations models are developed to verify the K-means implementation. Modelsim simulation software is used to develop the models. The beginning stages of the design are tested for functionality by feeding the test vectors to the compiled code using Modelsim. The blocks are tested individually and then incrementally to verify the functionality. The design files are then synthesized and routed on a chip for the purpose of back annotation. The annotated sdf file is then used instead of the original design files to verify gate level simulation with the same test vectors. Figure 6.3 is a block diagram to illustrate the idea of design verification.

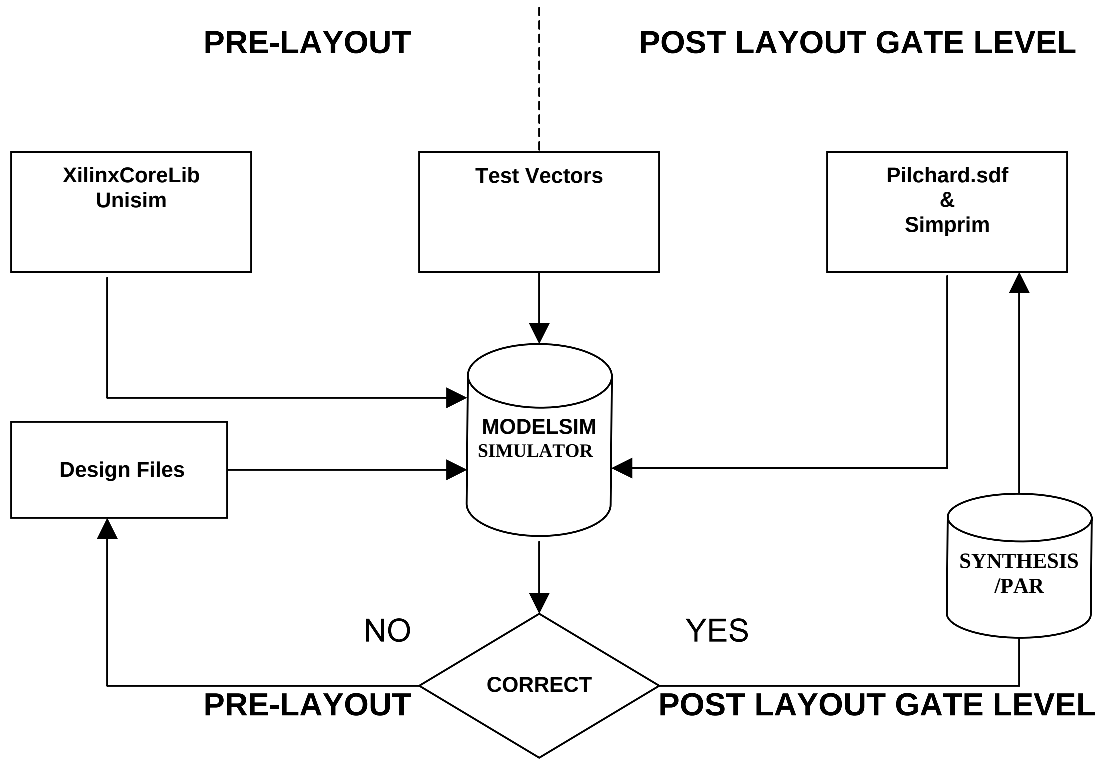


Figure 6.3: Design Verification

6.4 HOST - HARDWARE DESIGN FLOW

The bitstream generated is downloaded to the Pilchard system and the host software used the read and write routines to transfer data back and forth. These are the only two simple routines available for the pilchard interface. Absence of handshake signals or other sophisticated routines like DMA transfer or interrupt controller make it harder to implement designs with constant data streaming. Figure 6.4 illustrates the idea of interfacing software and hardware.

6.5 HARDWARE MODEL

A hardware model is developed to reproduce the results of K-means clustering algorthim on the FPGA system. Figure 6.5 is a 41-stage pipelined implementation of the algorithm and the sub-units of the architecture are detailed in this section.

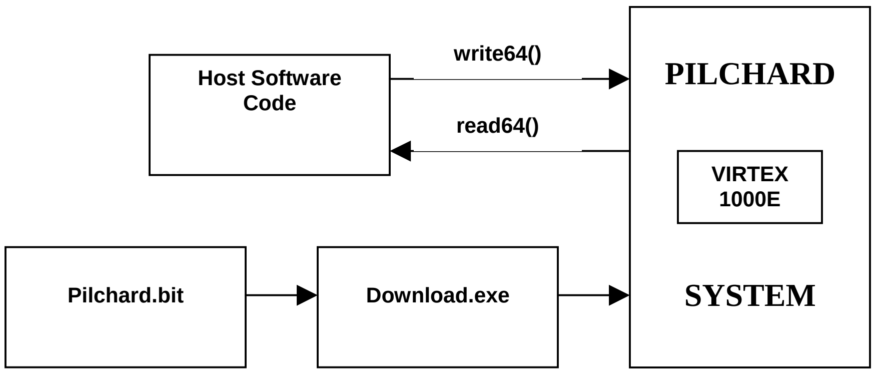


Figure 6.4: Interfacing Hardware and Software

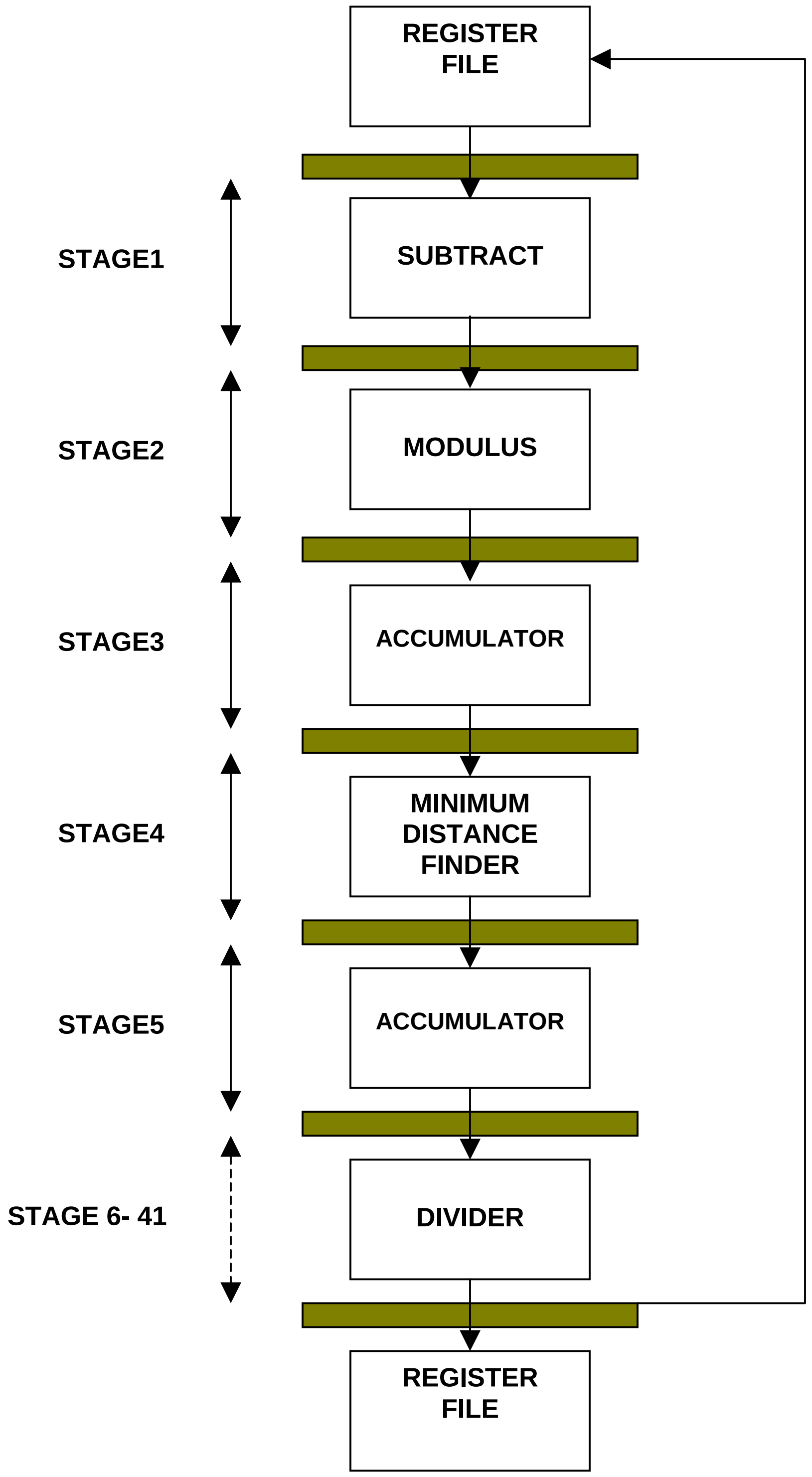


Figure 6.5: 41-Stage Pipelined K-means Clustering Algorithm

6.5.1 MEMORY MODEL

As described in chapter 4, one of the major drawbacks of the pilchard system is the unavailability of on-board RAMs. This necessitates the use the block RAMs available within the FPGA unit. Three sets of dual port rams have been used. The first one [R1] is used to store the entire image, the second [R2] to hold the initial and intermediate cluster centers and the third one [R3] for holding the final cluster centers.

The control signals to R1, R2 and R3 are from a global state machine that set/reset registers globally. At the end of the convergence, a soft reset is triggered to reset all global registers to the 'Idle' state.

The 64 bit data is transferred to R1 and R2 on the positive edge of the clock. The bus line is organized in a way that 16 bits of a pixel, 16 bits of initial cluster centers C1, C2 and C3 are packed together and transferred to the block RAMs at the same time. As is stated earlier in chapter 5, only 13 bits were required to represent the data set obtained from Jet Propulsion Laboratory. The remaining 12 bits were used for any extra addressing needs. The control signals from the FSM help unpack the bits for further number crunching.

The design is incorporated in such a way that the 64-bit bus line holds only a single element of the pixel, otherwise called 'feature vector' along with the class center values of similar features. Thus, if 10 feature vectors represent a pixel, also called an observation, it would take 10 clocks plus transfer overheads to load a single observation into the RAMs. The main advantage of packing and loading a pixel this way is that the number of features used to represent the data does not change or add additional circuitry. The downside though could be the use of extra RAM blocks required to store massive datasets. This one, along with unavailability of on-board rams limits the designer to handle data sizes with a certain limit. However, within the constraint, it certainly works to the advantage of the designer. The block level architecture of R1 and R2 is shown in Figure 6.6.

Consider a pixel with three features. Assume the features X, Y and Z represent a spatial domain. Figure 6.7 is a pixel located in a 3-dimensional space.

Let the pixels be represented as P_x^i, P_y^i, P_z^i and the class centers be C_x^j, C_y^j, C_z^j; where $i \varepsilon N$ and $0<j<3$. The first pixel is loaded into the RAM in 3 clock cycles and the next one takes three more clock cycles to load up. The process is continued until the block rams are filled up or till all the pixels get loaded if the size of the dataset is smaller than size of the block RAMs.

The current design is scalable for number of features from 1 to 16 and the memory model remains the same. This eliminates any additional circuitry, hardware costs and other engineering costs.

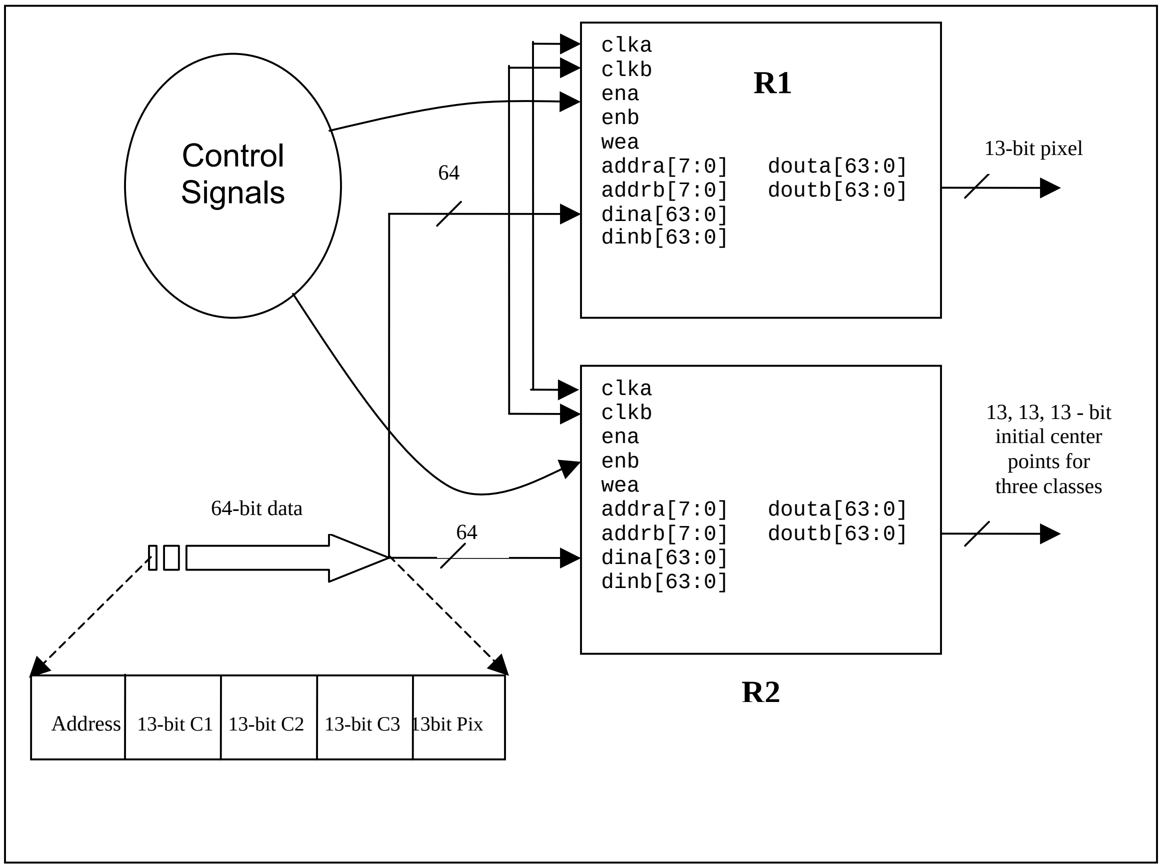


Figure 6.6: Dual Port Memory Receive Data Layout

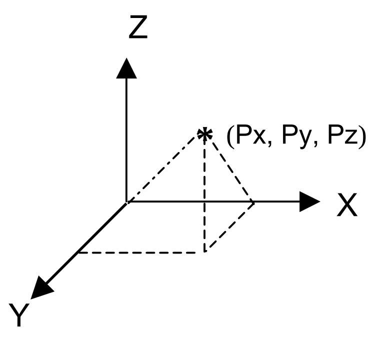


Figure 6.7: Spatial Representation of a Pixel with Three Features

Figure 6.8 illustrates the architecture of R3 dual port RAM. It mainly serves two purposes.

1. Transferring cluster centers and state outputs back to the host
2. Hardware debugging.

This is quite similar to looking at the simulation waveform for functionality verification, but is even better, since it outputs the resultant numbers in a real world environment that can then be verified.

The use of 27 pin outputs to spectrum analyzer is another available technique to do hardware debugging. The use of R3 just avoids usage of additional/external hardware resources.

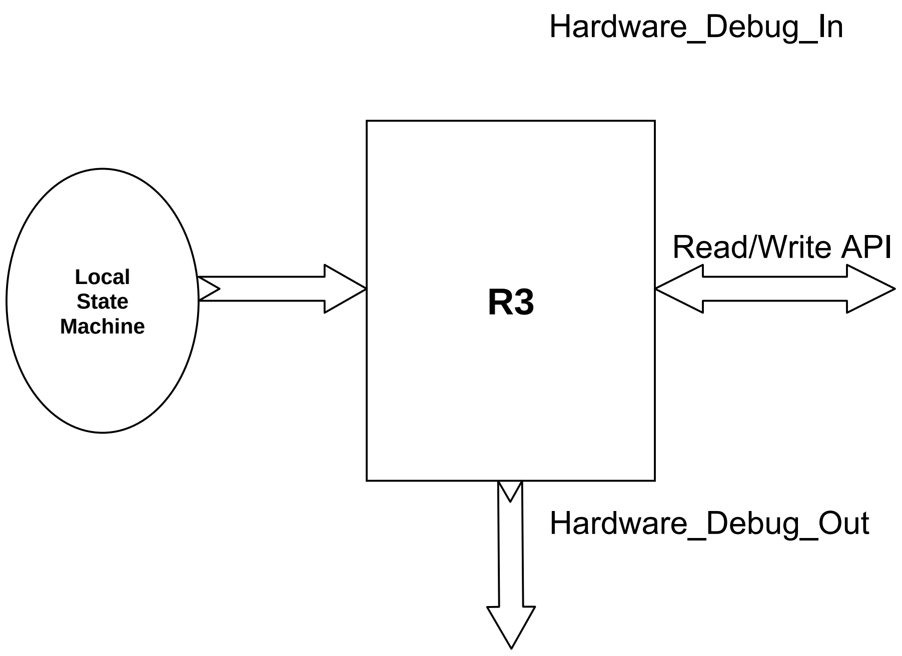


Figure 6.8: Dual Port Memory Transfer Data Layout

The state machine described in section 6.5.3 controls the start and stop of the processing cycle. Also, the state machine controls the way the RAM reads out and feeds into the core. Some of the controls signals emanating from the state machine are discussed in this section to show how exactly the memory model works in this design. The illustration of the idea is aided with pictures, pre-layout and post layout simulation models.

The RAM keeps accumulating the data until it fills up completely and when it reaches the last addressable line, the FSM sets a start register to signal high enabling the start of the core engine. Another register keeps track of the address count as the address start to move up.

The start signal is held high until the core completes all the iteration and spits out results back to the block rams.

The dual port RAMs are basically used to isolate the system speed with the design speed. The memory models have been tested for functionality and back annotated for post layout analysis.

6.5.2 CORE ENGINE

There are three pieces that basically integrate into a core engine. They are

1. Distance Determination
2. Pixel Classification/Re-Classification
3. Division and Cluster Centers Determination

These three pieces are at the heart of the implementation and are major components in the solving of this tediously iterative problem. The implementation as mentioned earlier is based on a heuristic approach and is not necessarily an optimal solution.

Each of the blocks is described in the following sections with corresponding RTL models.

6.5.2.1 DISTANCE DETERMINATION

This unit basically computes the distance between the pixel and the class's cluster center. Manhattan distance metric is used to find the closest distance.

The schematic for the distance determination data path is shown in Figure 6.9. The sub_abs RTL block takes each feature of the pixel and evaluates the Manhattan distance for each corresponding feature of the class centers. The Manhattan distances of all features then add up to determine the distance of the current pixel. The features feed into the processing engines serially that has been maximally pipelined. Amdahl's law limits the benefit of more pipeline stages.

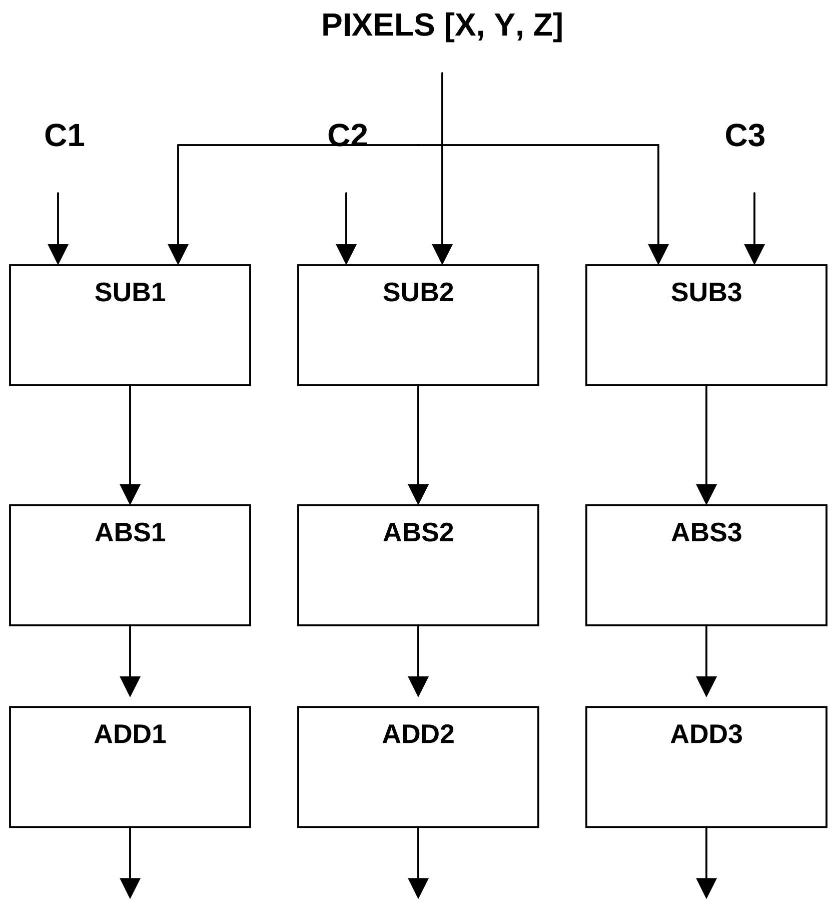
The state machine control signals keep track of the number of features to add up and reset the accumulator to zero for the next set of pixel features to be processed. The equation represents the logic implemented

$$\text{Manhattan Distance: } [P^1, C^1] = \left|P_x^1 - C_x^1\right| + \left|P_y^1 - C_y^1\right| + \left|P_z^1 - C_z^1\right|$$

$$\text{Manhattan Distance: } [P^1, C^2] = \left|P_x^1 - C_x^2\right| + \left|P_y^1 - C_y^2\right| + \left|P_z^1 - C_z^2\right|$$

$$\text{Manhattan Distance: } [P^1, C^3] = \left|P_x^1 - C_x^3\right| + \left|P_y^1 - C_y^3\right| + \left|P_z^1 - C_z^3\right|$$

The accumulator is reset to zero after 3 clock cycles. The control circuit controls the number of feature vectors that goes into the accumulator. This is important because the pixels are sent in serially to the processing engine and a counter keeps track of start and end of a pixel and sets/resets the accumulator accordingly.



$$\left|P_x^1 - C_x^2\right| + \left|P_y^1 - C_y^2\right| + \left|P_z^1 - C_z^2\right|$$

Figure 6.9: Manhattan Distance Computation

6.5.2.2 PIXEL CLASSIFICATION/RE-CLASSIFICATION

This piece of the design is a bit tricky and a picture is drawn to comprehend the design. Figure 6.10 shows the pseudo-RTL model for the pixel classification unit. To begin with, the Manhattan distance computed for each pixel to all the three class centers is compared against each other to determine the least distance. The classify logic determines which class the pixel belongs to. Since the pixel is already available for the add/shift logic to use, a look-ahead addition is performed for all the classes and presets back to the previous values if they don't belong to the class. The classify logic makes the decision whether to step back or not.

The number of pipeline stages change as the feature size is incremented. This is because the shift register keeps track of the feature value for a pixel set and resets the pre-calculated values to the previous state after all the features /pixel have been processed. This requires a shift pipeline to hold all and only the number of feature values per pixel. The VHDL for this is also parameterized. In chapter 7, a way to build a dynamically parameterized pipeline is detailed.

The look-ahead addition is similar to the carry look-ahead adder, the difference being that while the latter operates on bits and carries overflow bits to the adjacent bits, the look-ahead adder just operates on real numbers.

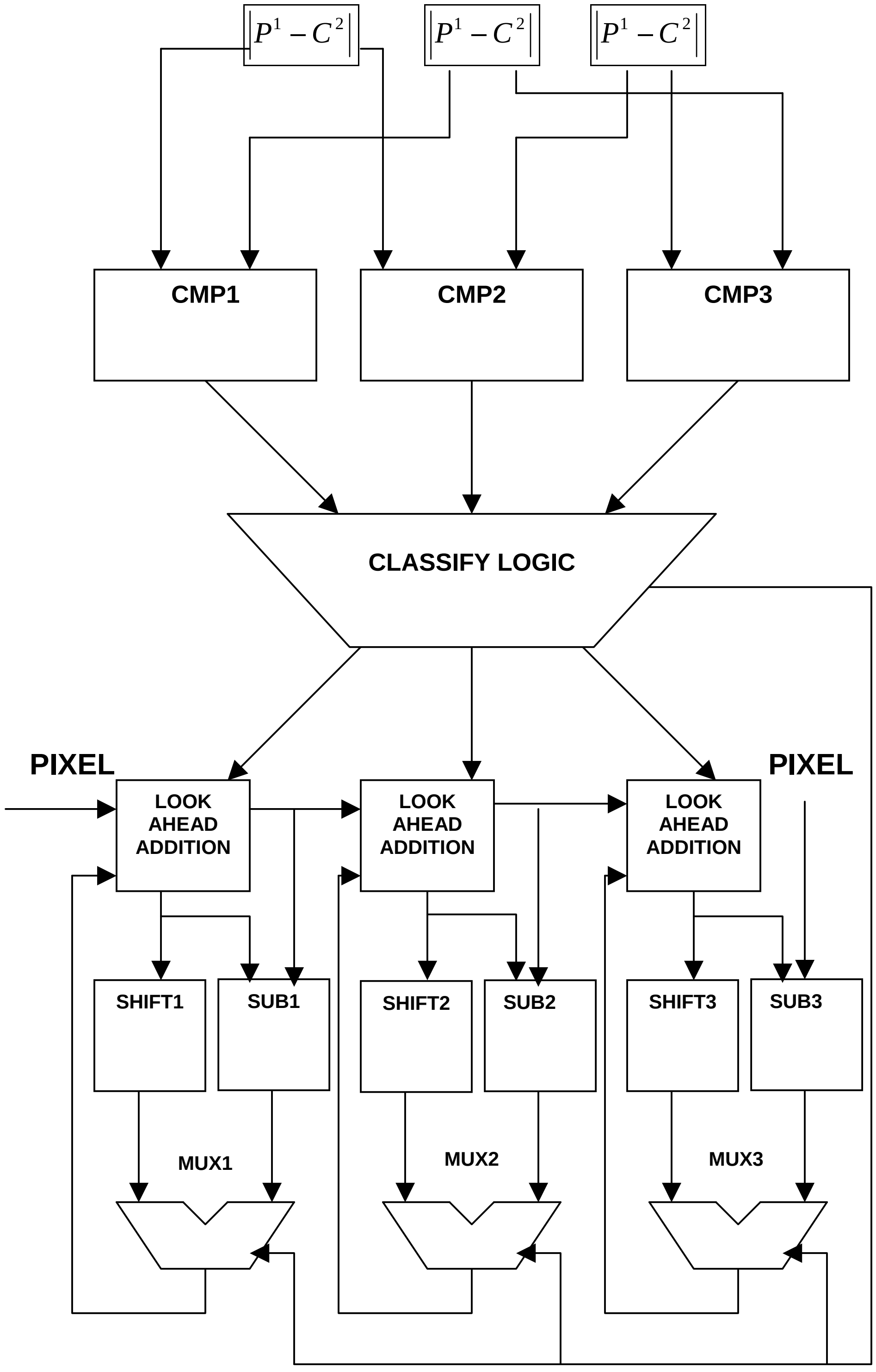


Figure 6.10: Pixel Classification Pseudo-RTL Model

6.5.2.3 FINAL CLUSTER CENTERS DETERMINATION

Three division operators have been implemented within the FPGA. The division operators are blocks generated from the Xilinx's core generator utility. A 32 bit pipelined divider model takes 36 clock cycles to fetch the division results. With the dividers inside the FPGA, it remains idle for a certain number of clock cycles before it starts to determine the updated centers for the cluster classes. However, it is found that the latency cycles are far fewer than having the dividers do their function on the host processor.

At the end of the division, a div_complete is sent out to the state machine to transition it to the next state. The values are then updated to the R2 dual port RAM for the next iteration to begin. The process continues until the centers converge.

6.5.3 FINITE STATE MACHINE

FSM is basically a machine with a fixed number of internal options or possibilities. These could be as few as 2 or any number of separate possibilities, each determined by some combination of input parameters.

The finite state machine is implemented as a Moore machine as shown in Figure 6.11. It sends out global control signals controlling the data path of the design. In other words, the state machine controls both the core engine and the memory modules of the design.

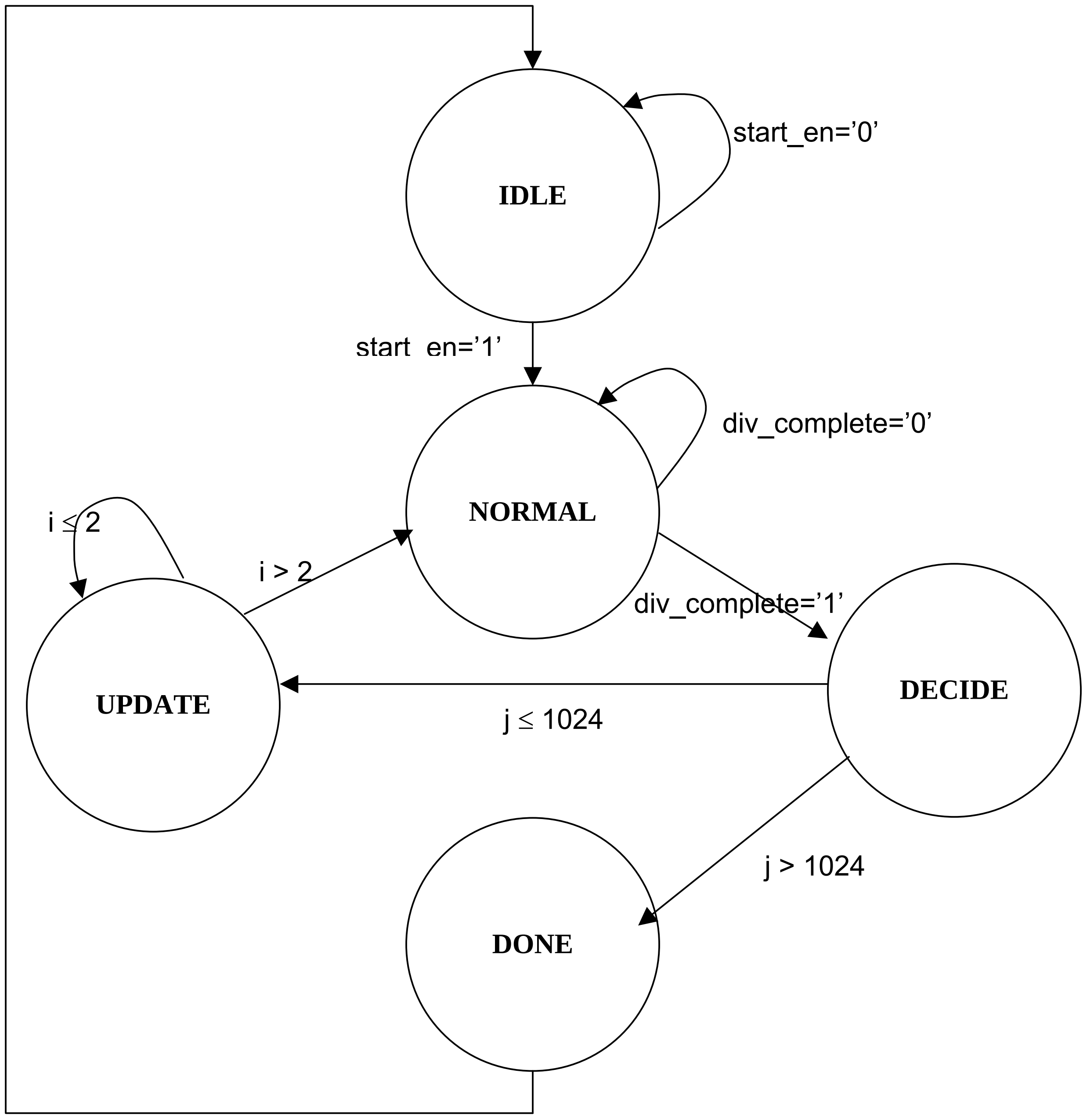


Figure 6.11: One Hot Encoded State Machine

The salient features of the state machine are listed below

1. One-hot encoded.
2. State variables are registered.
3. Control is global and resets all the registers globally after reaching the DONE state.

The VHDL entity of the state machine is shown in Figure 6.12

The control starts with an IDLE state and stays in the same state till the ‘start enable’ signal goes high. The start signal is basically a signal off another local state machine that controls writing, reading and addressing of the dual port RAMs. In other words, when the RAMs are ready to feed data into the core, it sends out a start enable signal. When the IDLE state detects this signal it jumps to the NORMAL state enabling the core to do its intended function.

The state is not transitioned until the entire set of the data is processed and pixels re-classified. At this point, the state machine detects the ‘division complete’ signal and moves to the DECIDE state. The purpose of the DECIDE state is to check for a condition such as the number of iterations before the core halts processing. For example, if the number were chosen as 50, a counter state variable checks for 50 counts before changing the state to the DONE state. Until this condition reaches, the states loop around first to UPDATE state where it updates the reclassified values, then to the NORMAL state and back to DECIDE state.

```
entity controller is
        port (
        clk:                    in std_logic;
        reset:                  in std_logic;
        start_en:               in std_logic;
        div_complete:           in std_logic;
        pix_en:                 out std_logic;
        soft_reset:             out std_logic;
        finish:                 out std_logic;
        state_out:              out std_logic_vector(4 downto0);
        start_update:           out std_logic);
end Controller
```

Figure 6.12: FSM Entity Declaration

6.6 HOST INTERFACE

As is stated in chapter 4, the pilchard has a very simple and efficient host interface. The read and write cycles are 64-bit bus transfers each. The downside is the unavailability of handshake protocols.

First, the code begins with a memory map of the hardware in the host machine. Second, the data to be processed is read into a matrix of p X q, where p is the number of the rows indicating an observation and q is the number of columns of features per observation. Third, the data is sent out to the pilchard using the write64() API in several cycles until the entire dataset is down linked to the hardware. The size of the dataset cannot exceed the size of all the blocks RAMs available in the FPGA. It is worthwhile to reiterate at this point one of the disadvantages of the Pilchard system, unavailability of on-board RAMs. The on-board RAMs as in Wild force system allows DMA transfers of the data to it or allows it to map directly to the external hardware with minimal bus cycles. Logically, this effect tends to get prominent when handling larger data sets.

Fourth, when the core finishes processing the entire data set, a soft reset is set high, all the registers are reset to zero and read64() API reads out the final cluster centers stored in the dual port RAM.

6.7 DESIGN SCRIPTS

Scripting is programming technique mainly used to automate repetitive tasks. Scripting has been widely used in hardware design circles where the flow of taking a specification down to bit patterns hasn't changed dramatically over more than a decade. Some of the scripts used in this design are discussed in detail in this section.

6.7.1 SIMULATION AND DESIGN VERIFICATION

Modelsim simulator is used to develop simulation models before and after layout. This simulator has been widely used in the digital and mixed design community to verify their designs for functionality and timing. The VHDL is first parsed, checked for syntax and semantics and then compiled using a Modelsim program 'vcom'. The compiled code is simulated for over a chosen time interval using 'vsim'.

The University of Tennessee has the latest version of the Xilinx Alliance tools but the Xilinx4.1i series pack has been used instead of the newer Xilinx5.1i. The reason for this is that the current Pilchard development board has built in packages that uses the former version and it has been found empirically that the primitives in the Xilinx5.1i series has compatibility issues.

Three important steps encoded into 3 scripts are required to verify the design for functionality and timing. Figure 6.13 shows the initial step to compile the Coregen memory blocks into a library, Figure 6.14 shows the next step of compiling the core and running it through the test bench and finally Figure 6.15 shows the script for timing simulation.

```
vmap XilinxCoreLib XilinxCoreLib
vcom -work XilinxCoreLib /sw/Xilinx4.1i/vhdl/src/XilinxCoreLib/ul_utils.vhd
vcom -work XilinxCoreLib
/sw/Xilinx4.1i/vhdl/src/XilinxCoreLib/mem_init_file_pack_v3_1.vhd
vcom -work XilinxCoreLib
/sw/Xilinx4.1i/vhdl/src/XilinxCoreLib/blkmemdp_pkg_v3_1.vhd
vcom -work XilinxCoreLib
/sw/Xilinx4.1i/vhdl/src/XilinxCoreLib/blkmemdp_v3_1_comp.vhd
vcom -work XilinxCoreLib /sw/Xilinx4.1i/vhdl/src/XilinxCoreLib/blkmemdp_v3_1.vhd
 vcom -work work ram64b_s64_s64.vhd
```

Figure 6.13: Script to Compile Memory Primitives into XilinxCoreLib

```
vmap dware /home/chandra/ModelSim/synopsys/dware

vmap dw01  /home/chandra/ModelSim/synopsys/dw01

vmap dw02  /home/chandra/ModelSim/synopsys/dw02

vcom -work work ./DW01_sub_inst.vhd

vcom -work work ./DW01_absval_inst.vhd

vcom -work work ./DW01_cmp2_inst.vhd

vcom -work work ./DW01_add_inst.vhd

vcom -work work ./cmp2.vhd

vcom -work work ./sub_abs.vhd

vcom -work work ./subtract_ver1.vhd

vcom -work work ./shiftN.vhdl

vcom -work work ./compare_ver2.vhd

vcom -work work ./controller.vhd

vcom -work work ./pcore.vhd
```

Figure 6.14: Script to Compile the Core

```
vmap simprim simprim

vcom -work simprim -explicit

/sw/Xilinx4.li/vhdl/src/simprims/simprim_Vpackage.vhd

vcom -work simprim -explicit

/sw/Xilinx4.li/vhdl/src/simprims/simprim_VITAL.vhd

vcom -work simprim -explicit

/sw/Xilinx4.li/vhdl/src/simprims/simprim_Vcomponents.vhd

vcom -work work time_sim.vhd

vcom -work work pcore_scmp_tb.vhd

vsim -sdftyp U4_PCORE=time_sim.sdf pcore_scmp_tb
```

Figure 6.15: Script to Verify Back-Annotated Results

The script shown in Figure 6.15 sets up and maps library components to the current working directory. Also, the last line in the script does a timing simulation of the design.

6.7.2 SYNTHESIS AND RTL GENERATION

Two tools have been employed to verify the correctness and quality of the design. Synplicity was primarily used to view gate-level RTL schematics and optimize unused or unrelated logic. Synopsys's FPGA Compiler is the other tool used to verify and synthesis the design into a gate-level netlist. The netlist is in the EDIF [Electronic Data Interchange Format] that the Xilinx's Place and Route tool uses to lay the design out.

6.7.2.1 SYNTHESIS FLOW

The synthesis script in Figure 6.16 is probably the most important script in the design and the brief explanation of the script is done in this section. Lines 1 through 7 define variables. The device present in the Pilchard system is targeted with a speed grade of 6. A directory export_dir is created to hold design information that needs to be exported for use with the other following programs.
Lines 8 through 9 remove old versions of the project and then create a new one.
Lines 10 through 12 opens the existing project and sets up project variables. Lines 13 through 26 identify the design source files and analyze each of them in hierarchical order. Lines 27 through 29 set up don't touch attribute to already optimized blocks. This saves some considerable CPU time and keeps the best-

optimized design untouched. Line 30 is a very important step in the synthesis process. The options specified with the create_chip command line decide performance results to a very great extent. For example, '-eliminate' option synthesizes the design by the way of flattening it. This leads to a very optimal design in terms of timing but the downside being large design cycle time. Essentially this creates a chip targeted for $TARGET with the default part and speed grade. The chip is named $chip and $top indicates the top-level design.

The remaining lines in the script basically optimizes, shows error and warning messages, writes out PPR netlist and constraints to the export directory and reports timing results. Figure 6.16 shows the details of the synthesis process.

6.7.2.2 OPTIMIZATION METHODS

Few optimization methods at different levels are employed. At the architectural level, the design is maximally optimized by the use of

1. 41-stage pipelined model
2. One hot encoded state machine
3. Carry look-ahead logic for adds and subtracts.
4. Re-use of DesignWare and Coregen models for efficient design etc.
5. Clock Gating

At the synthesis level, few optimization methods such as register retiming, synthesis of flattened design were used. In addition to this speed level of 6 is set and the design is over constrained to run at 150 MHz for better synthesis results.

```
set proj pilchard_proj
set top pilchard
set target VIRTEXE
set device V1000EHQ240
set speed -6
set chip pilchard
set export_dir .
exec rm -rf $proj
create_project -dir . $proj
open_project $proj
proj_export_timing_constraint = "yes"
default_clock_frequency = 150
add_file -library WORK -format VHDL vhdl/DW01_add_inst.vhd
add_file -library WORK -format VHDL vhdl/DW01_absval_inst.vhd
add_file -library WORK -format VHDL vhdl/DW01_sub_inst.vhd
add_file -library WORK -format VHDL vhdl/DW01_cmp2_inst.vhd
add_file -library WORK -format VHDL vhdl/sub_abs.vhd
add_file -library WORK -format VHDL vhdl/subtract_ver1.vhd
add_file -library WORK -format VHDL vhdl/shiftN.vhdl
add_file -library WORK -format VHDL vhdl/cmp2.vhd
add_file -library WORK -format VHDL vhdl/compare_ver2.vhd
add_file -library WORK -format EDIF vhdl/ram64b_s64_s64.edn
add_file -library WORK -format EDIF vhdl/ram64b_s256_s256.edn
add_file -library WORK -format EDIF vhdl/div32.edn
add_file -library WORK -format VHDL vhdl/controller.vhd
add_file -library WORK -format VHDL vhdl/pcore.vhd
add_file -library WORK -format VHDL vhdl/pilchard.vhd
analyze_file -progress
```

```
create_chip -eliminate -target $target -device $device -speed $speed -module -name
$chip $top
current_chip $chip
set_chip_retiming -enable
set opt_chip [format "%s-Optimized" $chip]
optimize_chip -name $opt_chip
list_message
report_timing
$export_dir
export_chip -dir $export_dir -no_timing_constraint
close_project
quit
```

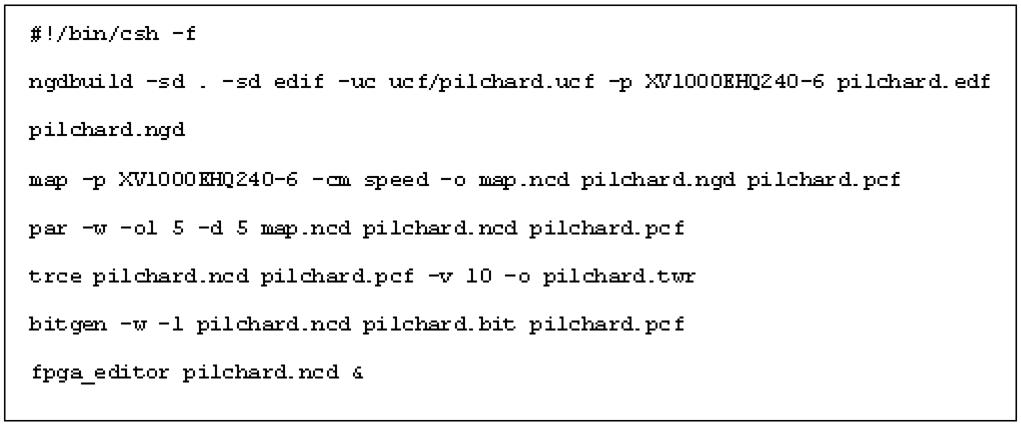
Figure 6.16: Synthesis Script with Optimization Parameters

6.7.3 PLACE AND ROUTE

A place and route program for Xilinx4.2i has been used to place and route the entire design. During the initial stages of development, a set of easy constraints was imposed on the par software just to be able to do a quick verification. The final stages of the design were set to tighter constraints and higher effort level to route the design. Additional optimization techniques such as pipelining, register retiming was done for better performance results. Figure 6.17 shows the script involved in the bit pattern generation.

A brief explanation of the script in Figure 6.17 is given below. More information about each utility can be found in the Xilinx User Guide [12].

Line1 performs an ngdbuild on the input netlist. NGDBuild performs the following steps to convert a netlist to an NGD file.



```
#!/bin/csh -f
ngdbuild -sd . -sd edif -uc ucf/pilchard.ucf -p XV1000EHQ240-6 pilchard.edf
pilchard.ngd
map -p XV1000EHQ240-6 -cm speed -o map.ncd pilchard.ngd pilchard.pcf
par -w -ol 5 -d 5 map.ncd pilchard.ncd pilchard.pcf
trce pilchard.ncd pilchard.pcf -v 10 -o pilchard.twr
bitgen -w -l pilchard.ncd pilchard.bit pilchard.pcf
fpga_editor pilchard.ncd &
```

Figure 6.17: Place and Route Script with Highest Effort Levels

1. Reads the source EDIF netlist.
2. Reduces all components in the design to NGD primitives
3. Checks the design by running a Logical Design Rule Check (DRC) on the converted design.
4. Writes an NGD file output.

The output NGD file can be mapped to the desired device family.

Line 2 runs a map program. MAP performs the following steps when mapping a design.

1. Selects the target Xilinx device, package, and speed. If the part is not specified, MAP issues an error message and stops. If the speed is not specified, MAP supplies a default speed though.
2. Reads the information in the input design file.
3. Performs a Logical DRC (Design Rule Check) on the input design. If any DRC errors are detected, the MAP run is aborted. If any warnings are detected, it continues to run though.
4. Removes unused logic. All unused components and nets are removed, unless the following Xilinx S (Save) constraint has been placed on the net or the –u option is used on the command line.
5. Maps pads and their associated logic into IOBs.
6. Maps the logic into Xilinx components (IOBs, CLBs, etc).
7. Update the information received from the input NGD file and write this updated information into an NGM file.
8. Creates a physical constraints (PCF) file.

9. Run a physical DRC on the mapped design. If successful writes a NCD file and a MAP report (MRP) file.

Line 3 invokes the placement and router tool. Figure 6.18 is a picture of the routed design generated by the PAR tool. It routes the design depending on cost-based or timed driven. Efforts level of 5 is set of the placer and the route that basically directs the router to search for a bigger search space for finding the optimum placement points. However, a guided PAR is a better technique for a better solution.

Line 4 of the script, TRACE (Timing Reporter and Circuit Evaluator) provides static timing analysis of a design based on input timing constraints. The TWR file is the timing report file with a .twr extension.

Line 5, BitGen produces a bitstream for Xilinx device configuration. After the design has been completely routed, it is necessary to configure the device so that it can execute the desired function. This is done using files generated by BitGen, the Xilinx bitstream generation program. It takes a fully routed NCD file as its input and produces a configuration bit pattern – a binary file with a .bit extension.

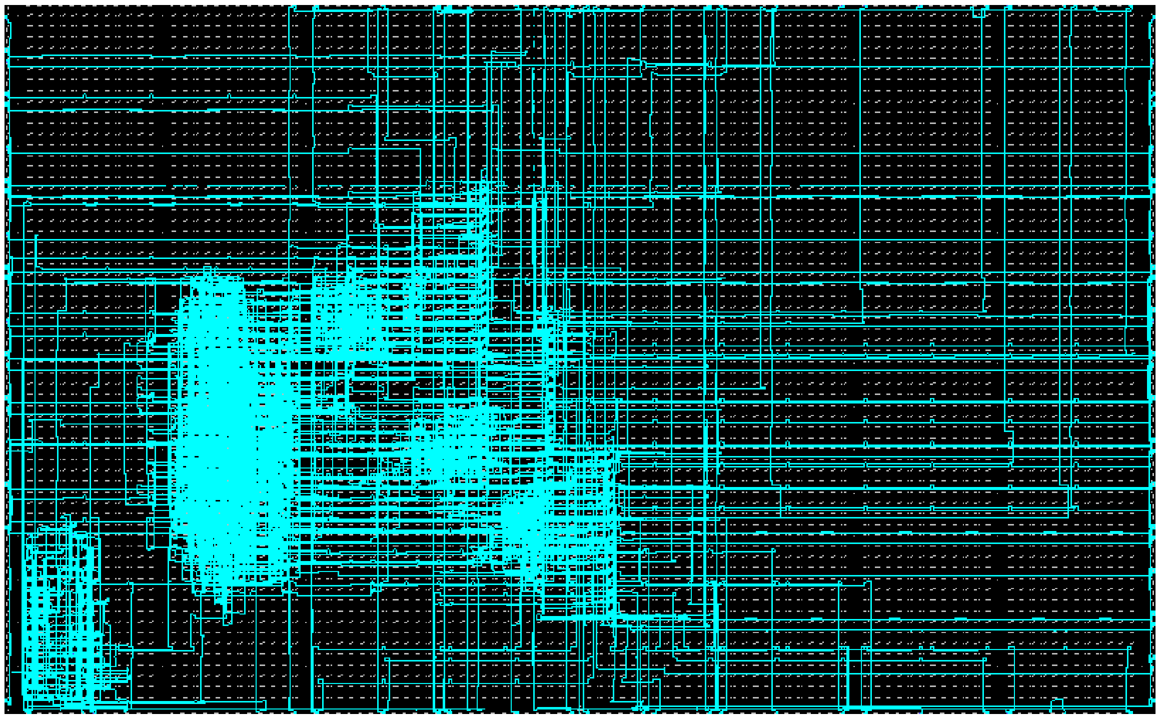


Figure 6.18: Layout of the K-means Core including Division Operators

6.8 RESULTS

This section discusses the results obtained and the conclusions that are drawn from the obtained results under certain specific constraints. A concluding speed up of hardware over software has been demonstrated for this application. A 117-point pre-processed hyper-spectral dataset from the Jet Propulsion Laboratory Library has also been analyzed to reinforce the judgment.

6.8.1 FLOATING POINT KMEANS

Different floating-point variations of k-means are tested in software. The code was then ported to fixed point C to analyze any performance benefits as is stated earlier in chapter 5. Initially, floating point with two different measures, Euclidean and Manhattan are analyzed followed by fixed-point K-means analysis using the design flow methodology described in chapter 5.

6.8.1.1 MATLAB AND C

Basically floating point Matlab is used to test the accuracy of the classifier and floating point C is used to compare run times of hardware vs. software. The golden Matlab code is ported to floating point C using Euclidean measure and none of the points misclassified for both synthetic datasets and hyper-spectral dataset from JPL library. The floating point is then taken as the reference to check the validity of the floating point C code with Manhattan distance measure.

For all the synthetic as well as real dataset tested, the classifier did not misclassify but there was definite accuracy loss in the displaced cluster centers.

While there has been no misclassification for the limited datasets tested, it does not, however, infer that the use of Manhattan distance does not misclassify. The Manhattan C is then transformed to fixed point C for further optimization.

6.8.1.2 CLASS CORRESPONDENCE

Different runs on K-means on the same dataset for example produces different order of classes depending on the initialization of the random vector. For instance, consider a 117-pt data set supposed to be classified in 3 classes, minerals, salts and manmade materials. For one run, the classifier does a classification of 9 points in class 1, 77 points in class 2, 31 points in class 3. On a different run, it might produce 9 point in class 2, 77 points in class 3, and 31 points in class 1. Hence, we can map class 1 of first run to class 2 of second run, class 2 of first run to class 3 of second run and so on. Therefore, the classes' minerals, salts and manmade materials can correspond to any of class1, class2 or class3. A class correspondence algorithm is developed to identify such a mapping procedure.

Table 6.1 shows the matlab runs for the pre-processed real data set before and after normalization. All the 16 features extracted have been used at this time. The accuracy listed in Table 6.1 is the accuracy of the classifier developed in software with respect to the ground truth. The error percentage ripples down as the design tries to optimize the speed from the floating-point software run to hardware implementation.

Table 6.1: Clusters Classification of 117pt Hyper Spectral data

Matlab Runs	Points in Class 1	Points in Class 2	Points in Class 3	Accuracy
Before Normalization	9	77	31	43.589%
After Normalization	18	68	31	35.897%

6.8.2 HARDWARE RUNS

A number of bit stream configurations were downloaded to the pilchard and tested for the validity of K-means. Methodical hardware debugging with the aid of R3 RAM is done to get the clustering algorithm working on the pilchard for a tiny dataset. Then, datasets with different numbers of observations were used to test the functionality and see if it consistently matched the results of the software. The hardware run times with and without the I/Os are observed. Table 6.2 shows the run time measurements of K-means clustering on different platforms. In addition to this, power measurements for 50% activity rate have been recorded.

6.8.2.1 POLLING FOR TIMING MEASUREMENTS

The host interface is very simple and easy to use but lacks sophisticated APIs that are available in hardware platforms like the Wildforce. Also, absence of handshake protocols necessitates the development of some polling techniques to constantly check the hardware status through one of the available registers.

Figure 6.19 is a polling algorithm for measuring run time on hardware.

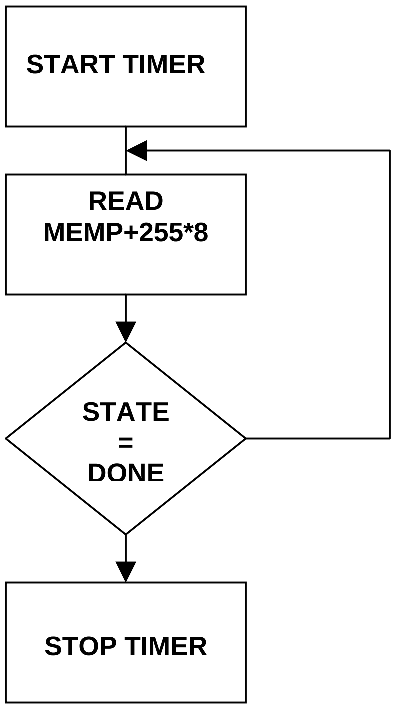


Figure 6.19: Flowchart for Recording Run Time Measurements

Table 6.2 Run Times of K-means Clustering A. Matlab

Number of Points	Matlab on Pentium III	Matlab on Sun Blade
6	4.6714 ms	8.220 ms
10	7.6816 ms	19.46 ms
25	18.330 ms	44.39 ms
50	38.000 ms	90.02 ms
117	349.70 ms	423.4 ms
234	692.00 ms	834.6 ms
468	1016.4 ms	1659.5 ms
936	1597.3 ms	3336.7 ms

B. Floating Point/Fixed Point C Run Times of K-means Clustering

Number of Points	Floating-C Manhattan	Fixed-C Manhattan
6	201 us	3229 us
10	383 us	4894 us
25	826 us	6999 us
50	1626 us	11537 us
117	3926 us	49507 us
234	6932 us	93669 us
468	17503 us	182214 us
936	35510 us	358833 us

C. Pilchard Run Times of K-means Clustering

Number of Points	Hardware without I/O	Hardware with I/O
6	2.040 us	34.04 us
10	5.040 us	45.04 us
25	8.630 us	78.63 us
50	14.62 us	134.62 us
117	15.39 us	269.39 us
234	30.00 us	518.00 us
468	61.48 us	1017.48 us
936	61.55 us	1953.55 us

For measuring execution time of the core, the timer starts after the write64() API and then reads the 255th address location constantly and keeps looping until the state of the hardware goes to DONE. Once the DONE state is reached the software quits out of the loop and records the end time. The difference of the start and end time provides the runtime of the core.

Table 6.3 characterizes the K-means Algorithm on the Pilchard Prototype board. The gate count and the power estimates recorded seem to be consistent for all the test vectors.

6.9 DISCUSSION

The implementation looked at three basic elements, viz core, memory and I/O. The Pilchard system has inadequate memory capabilities that require transferring data into the RAMs within the FPGA. A considerable amount of write cycles have to be spent in transferring data to the hardware units with the FPGA framework.

Table 6.3 Area and Power Estimates of K-means clustering On Pilchard

Number of Points	Area [Gate Count]	Power Estimate [mW]
6	430,471	1395.60
10	430,294	1398.12
25	430,459	1438.00
50	430,465	1435.00
117	430,417	1439.00
234	430,435	1435.00
468	430,435	1440.00
936	430,435	1440.00

However, in the software implementation, the data is already mapped to the main memory or even cache memory. Therefore, it is actually fair to compare the two software versions for core-to-core run times. In the case of other similar hardware assist platforms available at University Of Tennessee, the availability of on board RAMs facilitates DMA mapping of the local memories to the main memory of the host computer. The DMA mapping significantly reduce the I/O overheads. It has been thus identified that I/O still remains to be a bottleneck of the Pilchard system. However, for the purpose of understanding of the system, run time measurements including the reads and writes is recorded and analyzed.

Figure 6.20 shows the speed up numbers for different experimental observations excluding the I/O overheads. Figure 6.21 also shows the speed up figures but including the I/O. As can be seen, without the I/O, the speed up over a conventional software version explodes to a maximum of 25,000.

However, the speedup factor with respect to floating point and fixed point C versions are drastically different. The hardware beats the fixed point C by a factor of 3216 without the I/O included and 183 with the I/O overheads included. But the floating point C seems to be faster than fixed point C version of the k-means clustering algorithm. This is a quite intuitive because the Pentiums on the Linux boxes have dedicated floating-point silicon to run floating-point operations and hence faster. The hardware beats the floating point C by a factor of 255 without the I/O included and 14 with the I/O overheads included.

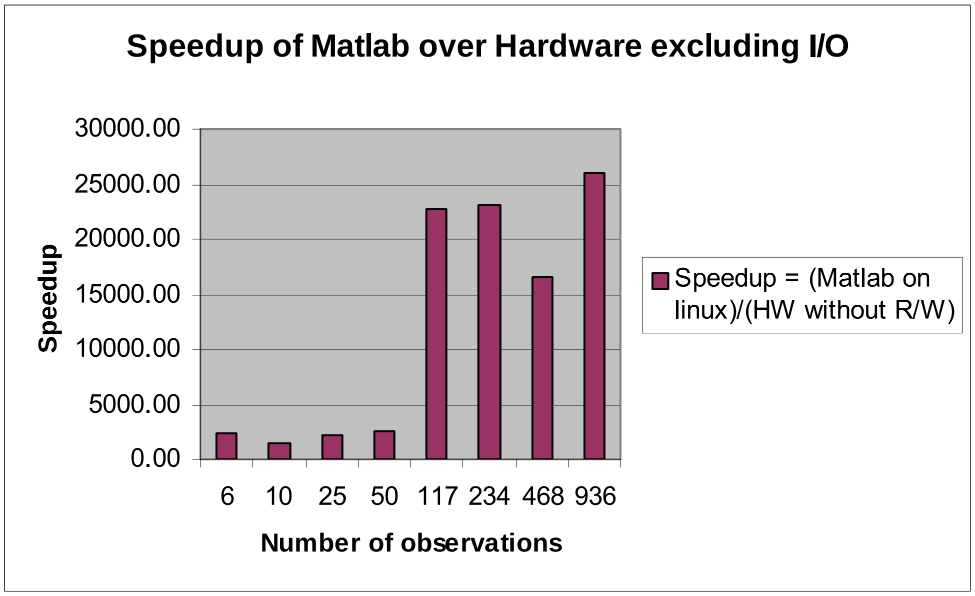


Figure 6.20: Speedup (1)

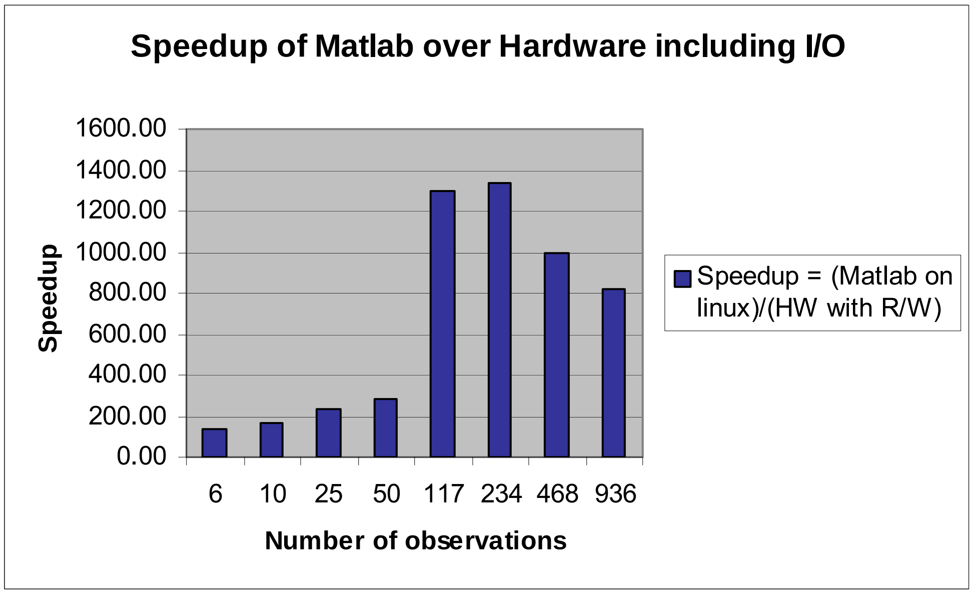


Figure 6.21: Speedup (2)

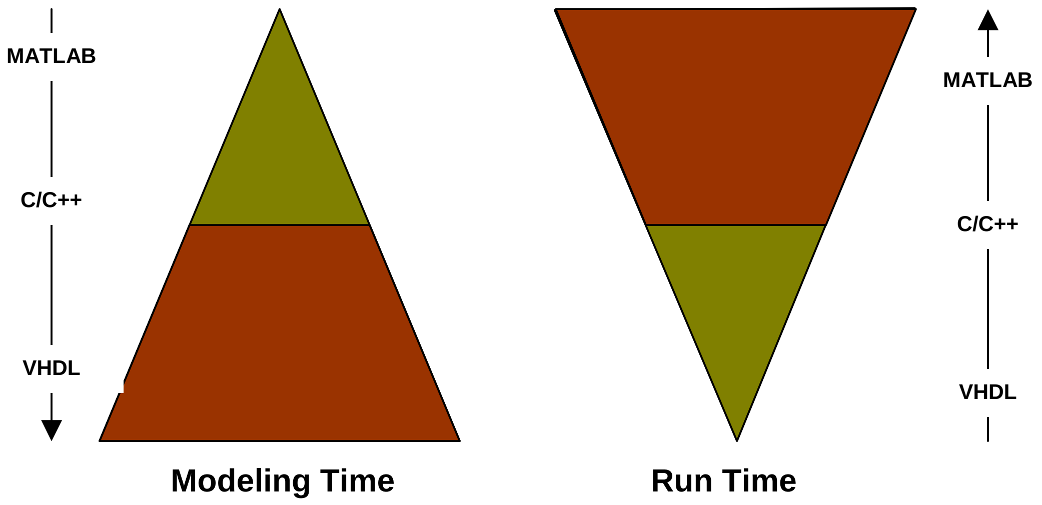


Figure 6.22: Trade-Offs in Modeling time vs. Runtime

The speedup of hardware implementation over the software version scales down drastically to a maximum of 1300 including the times to reads and writes. However, the speed is still fairly impressive but could be improved overall with efficient I/O and on board memories. Figure 6.22 shows the trade-off in development time vs. run time for different coding models.

Another interesting observation comes from the Table 6.4. A gate count with a consistent average of 430,000 out of a million gate chips is about 43%. It has been computed empirically that out of the 43%, the three dividers take up 34% of the area. Thus, the core logic occupies about 9% of the chip area. Conceivably, the core can be replicated about 6 times before the chip runs out of floor area. That means 6 times more speedup than the current reported speedup numbers.

However, the problems of speeds and feeds discussed in chapter 4 occur. This again could be improved if we had better memory and I/O capabilities.

One of the most important features of the implementation is the compilation of a parameterized VHDL. The implementation has been parameterized for two items.

1. Number of features
2. Number of Observations

However, there are limitations in the present work that can be eliminated quite easily as suggested in the chapter 7.

The generic items have been set to limit the number of features to 16 and the number of observations to 2^{15}. The number of features is parameterized and can be set to any desirable number less than or equal to 16. Similarly the number of observations is parameterized as well. For the case of 16 features and 2^{15} observations, only 18% of the blockRAMs have been utilized. Hence, we expect the design to support up to 2^{17} observations.

6.10 PILCHARD CHALLENGES

The formulation of the problem started with the availability of the Pilchard system and a curiosity to implement conventional clustering algorithms for the purpose of rapid prototyping. First the algorithm and details were studied, followed by literature survey of related research. Tons of researchers had been identified to have done or doing similar research. An attempt to isolate points that haven't been addressed was done. A methodology was laid out before starting to experiment with the hardware assist. The Pilchard system posed numerous design challenges and the process of identifying loopholes and fixing them serves a rewarding experience. Simple designs were composed and made to work on the pilchard system. Designs like an adder or an accumulator sandwiched between two dual-port RAMs were first implemented successfully. This example served as a tutorial to other research students in the group using the same system.

Along the way, several better ways to implement logic was also identified and described below. These could be applied particularly to the Pilchard system as some of the hardware designs and optimization methods do work fairly well with some platforms but do not work at all with others. The reason for this is because different platforms might use different technologies, different synthesis programs within the FPGA framework. For example, two synthesis tools treat the same piece of code in a different way and synthesize it to the gate level in an entirely different way.

6.11 CHAPTER SUMMARY

This chapter described the implementation details, followed by the analysis of the results obtained. A discussion involving analysis and interpolation of the results for better hardware platforms has been included. The next chapter presents conclusions and offers future directions for the present work.

7. CONCLUSION AND FUTURE WORK

The previous chapter described the details of the design and the results were presented. The objectives of the design were realized and the results shown are quite promising within the constraints specified. This chapter outlines the conclusion of the thesis and provides future directions of the research work.

7.1 OBJECTIVES ACHIEVED

Three important conclusions can be drawn from the results of the thesis work.

1. Acceleration approaches of k-means clustering on hardware seem to be very promising. An average speed-up of 10K for a core-core comparison (without I/O) and an average of 500 over Matlab with the I/O overheads included have been achieved with the current implementation.
2. System is I/O and memory limited. With some improvements to the architecture of the hardware assist, significant speed-up results can be achieved. This also necessitates more banks of memory or even on-board RAMs.
3. The implementation is highly scalable with minor changes to the control logic of the design. Also, multiple copies of the hardware resources need to be used until the chip floor runs out of logic gates/blocks.
4. Parallel kernels have been exploited to run concurrently on the hardware for significant speedup factors and therefore image processing algorithms such as k-means clustering are a good fit for hardware platforms and hardware accelerators.

7.2 FUTURE DIRECTIONS

There are enormous potential developments that can be done to take the current implementation to the higher levels. Some of these are pointed out as below.

1. As the chips sizes get denser and bigger in terms of logic gates, more stuff can be thrown into the FPGAs to build a comprehensive image-processing model. For example, raw data → filter → compressor (Principle Component Analysis) → classifier (K-means clustering) → processed data can all be burned in a single FPGA and deployed near the sensor. This way down linking and other transfer overheads could almost be eliminated. This saves a lot of latency cycles and therefore a faster processing method.

2. The division operations within the FPGA that consume large amounts of clock cycles per iteration can be replaced by shifts and subtracts and one final divider. The final divider could be implemented within the hardware or can be made available on the host.

3. The chip floor takes up about 34% of the floor area with 3 dividers that could be replaced by a single divider by appropriate scheduling and pipelining.

4. The parameterized implementation can also be made scalable across classes. Section 7.2.1 describes the steps required to implement a fully scalable design.

5. Massive parallelism at the course grain level can be exploited by HPRC development of the algorithm where N general-purpose processors are each attached to the respective RC nodes. The latency issues could be a bottleneck and appropriate research could identify the strength or weakness of such implementations.

7.2.1 SCALABILITY ANALYSIS

The following are the steps required to tweak the current design to make it fully scalable across classes. First, the three dividers need to be replaced with a single divider to create room for copies of hardware blocks for additional classes. The number of classes up to which scalability can be done depends on the size of the chip part and would need to be found out empirically. Figure 7.1 shows the block diagram of the scalable implementation. Proper scheduling can be done to pipeline the single division unit by appropriate latency delays per class.

The three dividers in the current implementation utilizes about 34% of the floor area, which is roughly about 10% for each divider. Three copies can be replaced by just one copy of the divider, which saves about 20% of the floor area. This gives some flexibility to include hardware copies for more classes in the scalable k-means architecture.

Second, the control unit needs to be tweaked to add logic to pick only the required outputs depending on the number of classes specified for the data set.

Third, the memory model needs to be changed to pack and unpack bits accordingly. Include all the 96 block RAMS available and let the additional control logic in the FSM pick the number of RAMS required for the design. This logic is similar to picking up only the required outputs for the number of classes. The R3 model needs no changes and the current model could still be used to store intermediate values for the purpose of hardware debugging.

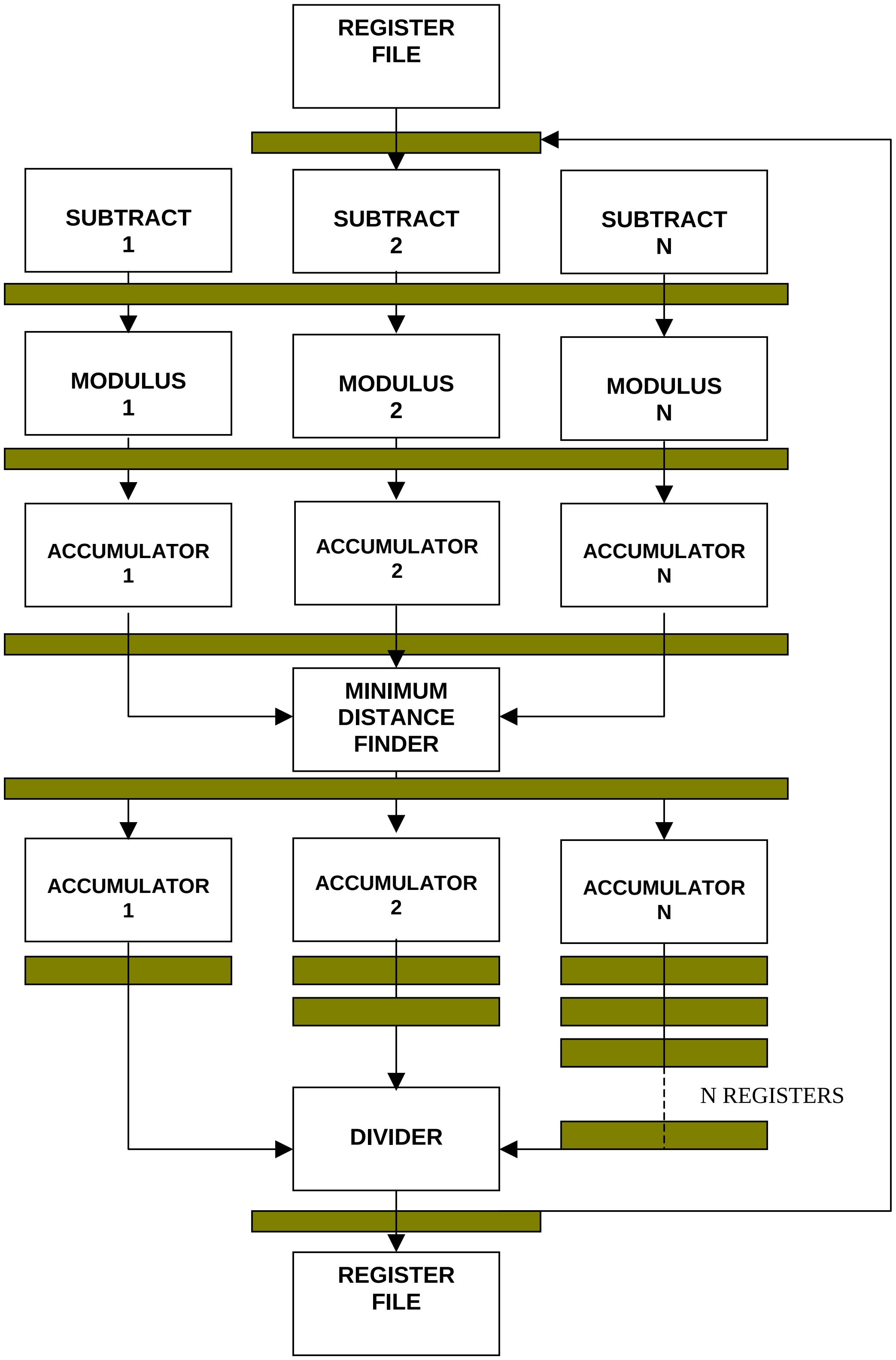


Figure 7.1: Scalable K-means Clustering Algorithm

Additional logic circuitry should be added in the NORMAL state of the state machine to decide on the fly the number of hardware copies to use. Generate statements can be used to parameterize the total number of copies replicated within the FPGA and the number of classes, the number of features and the number of observations could be written to the register file and the control logic within the finite state machine would read the registers and draw outputs from the required hardware copies on the fly. This design will also require one time configuration of the hardware design provided the bit width representation for the given datasets remains the same.

7.3 CHAPTER SUMMARY

This chapter concludes the objective of the thesis and points out contributions and limitations of the current work. Future directions of the work have been stated, in addition to laying out steps that needs to be done to scale this architecture for N classes, N>3.

BIBLIOGRAPHY

[1] M. Estlick, M. Leeser, J. Szymanski, and J. Theiler, *Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware*. ACM FPGA 2001, 2001.

[2] J. Frigo, M. Gokhale, and D. Lavenier, *Evaluation of the Streams-C C-to-FPGA Compiler: An Applications Perspective*. ACM FPGA 2001, 2001.

[3] M. B. Gokhale and J. M. Stone. *Co-synthesis to a hybrid RISC/FPGA architecture.* Journal of VLSI Signal Processing Systems, 24, March 2000.

[4] Inderjit S. Dhillon, Dharmendra S. Modha, A *Data Clustering Algorithm on Distributed Memory Multiprocessors*, IBM Research Report RJ 10134(95009), Nov.11, 98, Proc Large-scale Parallel KDD Systems Workshop, ACM SIGKDD, Aug. 15-18, 1999

[5] Jeffrey Wolff, *Dance Implementation of K-means Clustering*, May 2000

[6] D. Lavenier. *FPGA Implementation of the K-Means Clustering Algorithm for Hyper- spectral Images*. Los Alamos National Laboratory LAUR 00-3079, 2000.

[7] M. Leeser, M. Estlick, N. Kitaryeva, J. Theiler, and J.Szymanski. *Applying Reconfigurable Hardware to Segmentation for Multispectral Imagery*. In HPEC 2000, Boston, MA, Sept. 2000.

[8] S Ray and R H Turi, *Determination of number of clusters in K-means clustering and application in color image segmentation*, (invited paper) in N R Pal, A K De and J Das (eds), Proceedings of the 4th International Conference on

Advances in Pattern Recognition and Digital Techniques (ICAPRDT'99), Calcutta, India, 27-29 December, 1999, Narosa Publishing House, New Delhi, India, ISBN: 81-7319-347-9, pp 137-143.

[9] R. A. Schowengerdt. *Techniques for Image Processing and Classification in Remote Sensin*g. Academic Press, Orlando, 1983.

[10] J. Theiler, J. Frigo, M. Gokhale, and J. J. Szymanski. *Co-design of software and hardware to implement remote sensing algorithms*. Proc. SPIE, 4480, 2001.

[11] Xilinx Corporation, Virtex 1000e Datasheet, *http://www.xilinx.com/bvdocs/publications/ds022.pdf*, 2000.

[12] Xilinx Corporation, User Manual, /sw/Xilinx4.2i/userguide/docs, N.A

[13] Synopsys Inc, Designware IP Blocks, *http://www.synopsys.com/cgi-bin/designware/ipdir.cgi*, N.A.

[14] Synopsys Inc, Designware User Guide, *http://vlsi1.engr.utk.edu/ece/bouldin_courses/Designware_Docs/dwdg.pdf*, N.A

[15] Dr. Donald W Bouldin, *Lecture Notes of ECE 551*, 2002.

[16] Dr. Donald W Bouldin, *Lecture Notes of ECE 552*, 2002

[17] J. Theiler, M. Leeser, M. Estlick, and J.J. Szymanski. *Design issues for hardware implementation of an algorithm for segmenting hyper spectral imagery.* In M.R. Descour and S.S. Shen, editors, Imaging Spectrometry VI, volume 4132, 2000.

[18] P.H.W. Leong, M.P. Leong, O.Y.H. Cheung, T. Tung, C.M. Kwok, M.Y. Wong and K.H. Lee, *“Pilchard - A Reconfigurable Computing Platform with*

Memory Slot Interface", Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), California USA, 2001

[19] Peterson, G. D. and Smith, M. C., *Programming High Performance Reconfigurable Computers,* SGRR 2001, Rome, Italy.

[20] JPL spectral library document

[21] David Langrebe, *Some fundamentals and methods for hyperspectral image analysis*, SPIE conference proceedings, January 1999.

[22] R.B. Smith, *Introduction to hyperspectral imaging*.

[23] John A. Richards, Xiuping Jia, *Interpretation of hyperspectral image data*.

[24] N.R. Pal and S.K. Pal, A review on image segmentation techniques, Pattern Recognition, vol.26, pp. 1277-1294, 1993

[25] A.K. Jain and R.C. Dubes, Algorithms for Clustering Data, New Jersey: Prentice Hall, 1988.

[26] C. Funk, J. Theiler, D. A. Roberts, and C. C. Borel. *Clustering to improve matched-filter detection of weak gas plumes in hyper-spectral imagery*. IEEE Trans. Geosci. Remote Sensing, 39:1410–1419, 2001.

[27] Thomas Fry Thesis, Schott Hauck @ University of Washington

[28] Lloyd Algorithm, John Hopkins

[29] DANCE Implementation by Jeffrey Wolf

[30] Pilchard User Manual

[31] Frontier Design, A|RT Builder User Guide

VITA

Venkatesh Bhaskaran was born on January 21st, 1979 in the city of Hyderabad, India. He lived there for 21 years where he went to high school at St. George's Grammar School. Venkatesh then moved to Madras to get his Bachelor of Engineering degree at the University of Madras. He developed a great interest for digital design engineering while an undergraduate and came over to the United States of America to further his skills at the University of Tennessee. He graduated with a Masters of Science degree majoring in Electrical and Computer Engineering in the year 2003. He is now a huge fan of VOLS and hopes to fund the VOLS team a bit as an alumnus. GO VOLS GO.