AN FPGA-BASED HARDWARE ACCELERATOR FOR K-NEAREST NEIGHBOR CLASSIFICATION FOR MACHINE LEARNING

by

MOKHLES AAMEL MOHSIN

B.S., University of Technology, 2010

A Thesis submitted to the Graduate Faculty of the

University of Colorado Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

2017

This thesis for Master of Science Degree by

Mokhles Aamel Mohsin

has been approved for the

Department of Electrical and Computer Engineering

By

Darshika G. Perera, Chair

T.S. Kalkur

John Lindsey

Date 11/15/2017

Mohsin, Mokhles Aamel (M.S., Electrical Engineering)

AN FPGA-BASED HARDWARE ACCELERATOR FOR K-NEAREST NEIGHBOR CLASSIFICATION FOR MACHINE LEARNING.

Thesis directed by assistant professor Darshika G. Perera.

ABSTRACT

Machine learning has become the cornerstones of the information technology. Machine learning is a type of artificial intelligence (AI), which enables a program (or a computer) to learn, without being explicitly programmed and without human intervention. Machine learning algorithms can be categorized into supervised learning (classification) and unsupervised learning (clustering).

Among many classification algorithms, K-nearest neighbor (K-NN) classifier is one of the most commonly used machine learning algorithms. This algorithm is typically used for pattern recognition, data mining, image recognition, text categorization, and predictive analysis.

Many machine learning algorithms, including K-NN classification are compute and data intensive, requiring significant processing power. For instance, the K-NN consists of many complex and iterative computations, which include computing the distance measure between the training dataset and the testing dataset, element by element, and simultaneously sorting these measurements.

In this research work, our main objective is to investigate and provide an efficient hardware architecture to accelerate the K-NN algorithm, while addressing the associated requirements and constraints. Our hardware architecture is designed and developed on an FPGA-based (Field Programmable Gate Array–based) development platform. We perform experiments to evaluate our hardware architecture, in terms of speed-performance, space, and accuracy. Our hardware architecture is also evaluated with its software counter-part running on the same development platform. Experiments are performed using three different benchmark datasets with varying data sizes.

We introduce unique techniques, including pre-fetching and burst transfers, to reduce the external memory access latency issue that is common in embedded platforms. We also address various issues with the proposed hardware support for K-NN algorithm in the existing literature.

This investigation demonstrates that machine learning algorithms can indeed benefit from hardware support. Our proposed hardware architecture is generic and parameterized. Our hardware design is scalable to support varying data sizes. Our experimental results and analysis illustrate that our proposed hardware is significantly faster, for instance, our hardware architecture executed up to 127 times faster than its software counter-part.

Dedicated to my family

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Darshika G. Perera, for giving me an opportunity to conduct this research work.

I am also thankful to the thesis panel members, Dr. T.S. Kalkur and Dr. John Lindsey, for setting aside time from their academic and professional commitments to review and evaluate my research work.

A special thanks to my fellow researcher, S. Navid Shahrouzi, for providing significant help during my research.

I would also like to give special thanks to my family for supporting me academically.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

CHAPTER I

INTRODUCTION

Machine learning is a type of artificial intelligence (AI), which enables a program (or a computer) to learn, without being explicitly programmed and without human intervention [33]. The basic premise of machine learning is to explore and construct algorithms that can learn from the input data received, which includes performing statistical analysis on this data, and then providing accurate predictions on this data. Machine learning allows the programs (or algorithms) to learn, adjust, develop, and grow autonomously, based on the new data [34]; thus, the programs become more accurate in predicting outcomes.

Machine learning has become one of the cornerstones of the Information Technology within the last two decades [35]. Machine learning has been incorporated into wide range of applications that have become parts of our daily lives. Few examples include: effective web searching; online recommendation engines that suggest friends on social media networks (such as Facebook), or suggest movies/shows on Netflix; self-driving Google cars; cyber-fraud detections; practical speech recognitions; and a vastly improved understanding of the human genomes [34], [33]. Furthermore, applications such as data security, personal security, financial trading, healthcare, marketing personalization, natural language processing, all incorporate some form of machine learning techniques [39]. In addition, according to market data [36], the global market for machine learning was $2.5 billion in 2014, and is expected to reach $12.5 billion in 2019. The above facts illustrate

that machine learning market will continue to flourish over the long term as new and autonomous applications emerge.

Machine learning involves many important data mining tasks. Data mining is the science of extracting useful information from big datasets [42]. Data mining commonly involves many high-level tasks such as classification, regression, clustering, summarization, dependency modeling, and change and deviation detection [14]. Furthermore, depending on the algorithms, machine learning can be categorized into supervised learning and unsupervised learning.

In this research work, we focus mainly on classification task, which is a supervised learning technique. Classification techniques are employed by many applications in various domains including medical, security, and marketing. There exist many classification algorithms. After our investigations on different classifiers, we decide to focus on the K-nearest neighbor (K-NN) classification algorithm, which is one the most popular classification algorithm. The K-NN classification algorithm is widely used in many machine learning applications.

Most of today's classification algorithms (including algorithms used for machine learning) are becoming more complex (compute/data intensive) requiring significant processing power, thus affecting the speed-performance of these applications. Furthermore, existing classification algorithms for machine learning are processor-based software-only designs. These processor-based algorithms, as is, are incapable of analyzing and processing the enormous amount of data efficiently and effectively.

Consequently, some kind of hardware support is imperative to address the constraints and requirements of the algorithms used for machine learning applications. In

this regard, Field Programmable Gate Array-based (FPGA-based) hardware support are becoming increasingly popular among hardware designers, since FPGAs provide higher level of flexibility than Application-Specific-Integrated-Circuit-based (ASIC-based) hardware designs, and also provides higher performance than processor-based software only designs. Since FPGAs re-programmable post-fabrication, hardware designers can utilize FPGAs to create any hardware architectures for any applications, and also modify the architectures during and after the design process as needed. As a result, in this research work, we decide to utilize FPGAs for our hardware designs.

As detailed in [38], "FPGA are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects". The most dominant FPGAs are typically SRAM-based (Static-Random-Access-Memory-based), which can be reprogrammed; however, One-Time Programmable (OTP) FPGAs are also available [38]. The ability to reprogram the SRAM-based FPGAs gives its advantages over ASICs, which are customized for specific task pre-fabrication [38]. Another advantage of FPGAs is that customized architectures can be provided for a specific application, compared to the general-purpose circuits of a processor [41]. Furthermore, compared to processor-based designs, FPGAs avoid the overhead associated with fetch/decode/execute instructions and the overhead associated with the operating systems.

As illustrated in [40], the global market for FPGAs was $6.94 billion in 2017, and will be $10.199 billion in 2022. FPGAs are being utilized in various applications in various domains, for instance, aerospace and defense, ASIC prototyping, audio, automotive, broadcast and pro AV, consumer electronics, data center, high-performance computing and

data storage, industrial, medical, security, video and image processing, wired and wireless communications.

1.1 Our research objective

In this research work, our main goal is to create a scalable, generic, and parameterized hardware architecture for K-nearest neighbor classification algorithm for machine learning applications utilizing an FPGA. In order to evaluate the feasibility and efficiency of our hardware architecture, we also design and develop software architecture for this algorithm, which is executed on an embedded processor on the same development platform for fair comparison purposes. We also execute the software on a desktop processor for comparison purposes.

1.2 Thesis organization

Our thesis has five chapters, and these are organized as follows.

In the second chapter, we provide a general background on machine learning and classification, and detail the K-NN classifier. Next, we present our investigation on the existing research works on hardware architectures for K-NN classifier.

In the third chapter, we explain the design approach utilized to create our designs, the experimental platform used to perform the experiments, the proposed system-level design, and the benchmark datasets used to test our design.

In the fourth chapter, we explain the embedded software architecture, and how our software design works to classify the data and the platform we use to test our software. Also, we explain the embedded hardware architecture, how our hardware design works, the data path for every step of the hardware and the specification of the components we use

in our design. Although not implemented, we proposed a parallel architecture to double the processing speed of the distance measurement step.

In the fifth chapter, we analyze the results obtained from testing our design using three different benchmark datasets. We use different value of K and different scenario where the test set can be larger, smaller or equal to the data set.

CHAPTER II

BACKGROUND

In this chapter, we present the background of machine learning and classification. We focus on the work of K-NN classifier. Also, we investigate the existing research work for hardware support for K-NN classifier.

2.1 Machine learning

According to Langley and Simon Machine learning is defined as "the study of computational methods for improving performance by mechanizing the acquisition of knowledge from experience" [1]. It is one of artificial intelligence (AI) applications that create a system or a program with the ability to learn from set of data to give accurate prediction outcomes, without being explicitly programmed [27],[28]. Like example, to build algorithm that can use statistical analysis to predict an output value based on the input set of data [29].

Machine learning has wide range of applications. For example, banks use machine learning algorithms to build models to analyze data in credit card applications. It is also used in medical fields for medical diagnoses. In telecommunication, machine learning is used to analyze call patterns for network optimization and enhance quality of service. Also, machine learning is used for control, optimization, and troubleshooting purpose in the industrial [4].

Machine learning algorithm process data set in varies field. The data sets typically consist of number of vectors (objects). Each vector usually consists of number of elements (instances). In this thesis we use the terms elements and instances, interchangeably.

Continuous features like age, high, weight and speed or categorical features like gender, color and marital status or binary features usually used to represent every instance in any dataset used by machine learning algorithms [2].

Machine-learning algorithms can be divided into lazy learning or eager learning algorithms. In lazy learning algorithm for example, K nearest neighbor (K-NN), the training set is stored in a memory and process does not occur until new data that needs to be classified is received, therefore, the computation time required during the classification phase is higher than that of the training phase. In contrast, eager learning algorithms such as decision tree, classification model is built before receiving new data. Therefore the computation time required during the classification phase is less than that of the training phase [2],[6].

The learning is called supervised learning if known labels that represent the correct outputs are given for the instances. In contrast, if the instances are unlabeled then the learning is called unsupervised learning [2],[6],[8]. Supervised learning algorithms require a pre-classified data that contain complete values of predictor variables and the target variables, then "the algorithm can learn which values of the target variables are associated with which values of the predictor variables" [3]. Regression and classification algorithm are the most common supervised learning algorithms. Unsupervised learning algorithm search for structures and patterns among the data variables, hence no target variables is required. Clustering is the most common unsupervised learning algorithm [3]. In the next sub-section, classification as a supervised learning technique, and classification algorithms are presented.

2.2 Classification

Classification is a two-phase process. The first phase is known as the training phase, and the second phase is known as the classification phase. In the training phase, pre-classified dataset that has several dimensions or attributes, (which are either continuous or categorical) is fed to the algorithm. The training phase is also known as the learning phase because the algorithm builds classification model based on the pre-classified dataset. The learning here is supervised because the class of the data is already given to the algorithm. In the classification phase, new data with unlabeled classes are fed to the algorithm. The algorithm uses the model that is built in the first phase to find the classes of the new data [8],[9],[10]. In other words, classification is supervised learning technique that process and analyzes training dataset to create a model that is able to find the class label of the new unlabeled data [8],[9],[10].

Classification tasks and applications can be found in many filed. Foe examples, in banking classification is used to determine whether a person is eligible for a loan or not. In medicine, classification is used to diagnose whether a certain disease is existing or not. In security, classification is used to identify fraud [3]. Also, classification applications include document classification, risk analysis, spam filtering, and image classification [9].

The most popular classification algorithms include Naïve Bayes, Support Vector Machines (SVMs), and the nearest neighbor.

Naïve Bayes algorithm is a parametric method, where the algorithm assumes a model that is valid for all the inputs [22]. Naïve Bayes classifier is built based on Bayes theorem, which describes the probability of an event, based on prior knowledge of the conditions that might be related to the event. The classifier can be trained efficiently, by

using a small number of samples [18],[21]. In comparison with other classification algorithms, in term of classification accuracy, Naïve Bayes classifier suffers from low classification performance, which is consider as the main disadvantage of the Naïve Bayes classifier [18].

SVM requires only a dozen of examples for training, and it is not sensitive to the number of dataset dimensions in the dataset. SVM has high generalization performance, and the highest accuracy among all other well-known algorithms, which makes it very effective method for general pattern, recognition, regression, and classification [5], [20].

K-NN algorithm is a nonparametric methods, where the algorithm can learn different patterns from the training data freely, which makes the algorithm flexible, and lead to high performance [22],[23],[29]. Nearest neighbor (NN) classifier identifies the classes of unknown data, based on the nearest neighbor, with the known class [21]. The classifier needs to store the pre-classified samples to classify the new samples. The NN classifier is wildly used in pattern recognition, object recognition, and text categorization. The K Nearest neighbor (K-NN) use the K nearest neighbors instead of one nearest neighbor, to find the class of unknown samples or data [21],[22]. In the next sub-section, K-NN classifier will be presented.

2.3 K-NN Classifier

K-Nearest Neighbor (K-NN) algorithm is a supervised and instance-based learning algorithm introduced by Hodges and Fix in the early 1950s. However, this algorithm did not become popular until higher computing power became available at 1960s [1],[2],[3],[11]. K-NN algorithm is considered as one of the most straightforward, simple and popular algorithms, where all the pre-classified data are stored in the memory, and then

the algorithm classifies the new data based on the similarity or the distance measure with the pre-classified data [11],[13]. Also, K-NN algorithm is an effective and efficient algorithm for pattern recognition, data mining, image recognition, text categorization, predictive analysis, etc. Therefore, it has become one of the most important algorithms developed in the 20^{th} century [10],[11].

In general, the K-NN algorithm is based on the following steps [14]

• Calculate the distance or the similarity between the testing set and the training set element by element.

• Sort the distance and determine the K nearest classes to the new data with the unknown class.

• Apply majority voting to decide the class of the new data.

K-NN classification results depend on the K parameter, hence choosing the right K is a crucial factor. It is hard to determine the optimal value of K, but the value of K should not be very small because that cause overfitting problem. Furthermore, the value of K should not be too large because, if K is too large then underfitting problem could occur, and some patterns could be overlooked [3]. It is common to select K value between 3 and 10, to overcome the overfitting and the underfitting problems [3]. An alternative method to select K is by trying different values of K, and calculating the classification error, then selecting the value of K with the least classification error ratio [3],[16].

There are many ways to measure the distance between the attributes in the testing set and the training set. The most significant ones are the Euclidean and the Manhattan Distances shown in equation (1) and equation (2) respectively [2].

Euclidean: $D(x,y) = \left(\sum_{i=1}^{m} |xi - yi|^{2}\right)^{1/2}$ (1)

Manhattan: $D(x,y) = \sum |xi - yi|$ (2)

Also, Minkowsky, Chebychev, and Camberra can also be used to measure the distance as shown in equation (3), (4), and (5) respectively.

Minkowsky: $D(x,y) = \left(\sum_{i=1}^{m} |xi - yi|^{r}\right)^{1/r}$ (3)

Chebychev: $D(x,y) = max_{i=1}^{m} |xi - yi|$ (4)

Camberra: $D(x,y) = \sum_{i=1}^{m} \frac{|xi - yi|}{|xi - yi|}$ (5)

Euclidean Distance is the most common technique to measure distance between attributes, but it is possible to use the other techniques, such as Manhattan Distance with K-NN classifier algorithm [2],[3],[18],[19]. As demonstrated, in [3] "When measuring the distance, however, certain attributes that have large values, can overwhelm the influence of other attributes, which are measured on a smaller scale". Therefore, a normalization technique is used on the dataset before applying any distance measurements. Min–max normalization or Z-score standardization is typically used for continuous variables [3],[11].

Min–Max normalization can rescale the dataset attributes within a certain range by using the equation (6):

$$Xn = \frac{X - \min(X)}{range(X)} = \frac{X - \min(X)}{\max(X) - \min(X)} \quad (6)$$

Min(X) is the minimum attribute value and Max(X) is the maximum attribute value where X is the value of attribute that is required to be normalized. Xn represents the new normalized value within 0-1 range scale. Conversely, Z-score standardization as shown in equation (7), works by finding the difference between the value of attribute that is required

to be normalized (X), and the mean value of the field mean(X), divided on the standard deviation of the field values SD(X) [3].

$$Xn = \frac{X - \text{mean}(X)}{SD(X)} \quad (7)$$

The distance is sorted in ascending order, to determine the K nearest distances to the new data, with the unknown classes. The classes of the K nearest distance measurements are obtained from the dataset.

The algorithm applies a majority voting similar to the majority voting used in elections, in that case the algorithm chooses the class with the highest iteration among the k nearest classes as the new class of the new data. If two or more classes have the same number of iterations, then the algorithm check the distance measurements of these classes, and chooses the nearest class to the new data with the unknown class to be the class of the new data.

2.4 Existing research work for hardware support for K-NN classifier

We investigated the existing research work on hardware support for K-NN algorithm in the literature.

In [59], the authors presented a high performance and high-level synthesis (HLS) based solution for K-NN algorithm. They stated that using HLS improved the productivity of the hardware designer by working at higher-levels of abstraction, and it was more effective than traditional HDLs because the functionality of the design can be verified at C-Level. They used C code to write their hardware architecture and used Vivado HLS tool to optimize their design and ensure parallelism and pipelining of the design. They exported their design as an IP core and used Xilinx ZC706 FPGA board with Vivado 2015.2 design

suite running at 100 MHz to evaluate their design. They tested their design with the following parameters: 1024 training vector, 2 dimensions, 2 classes and K is equal to 7. They used Matlab2012b running on Intel 15-2400 CPU at 3.1GHZ and 12G RAM to run their design on GPP. They claimed that their hardware architecture design is 35.1x faster than GPP based implementation. Testing their design with only 2 dimensions does not show the real capability of the design. The distance measurement step processing time depends on the number of training vector and the number of dimension.

In [10], the authors present an efficient parallel implementation of K-NN algorithm using FPGA-based heterogeneous computing systems architecture, using an Open Computing Language (OpenCL). The goal was an optimal parallel implementation to accelerate the K-NN algorithm. In this paper, Altera's OpenCL compiler was used to compile the OpenCL code onto the FPGA. The algorithm was implemented on CPU, GPU and FPGA for comparison purpose. The CPU is Intel Core i7-3770K running at 3.5GHz. The GPU is an AMD Radeon HD7950 with maximum working frequency 900MHz. The FPGA board was a Terasic DE4 with a Stratix IV 4SGX530. They used 20480 records out of 50000 records of KDD-CUP 2004 quantum physics data set to test the design. The K-NN implementation on GPU accelerated the speed 410 the speed of the CPU implementation while FPGA achieved 148 times. Although the GPU implementation was faster than the FPGA implementation, but they claim that FPGA consumed less power and the energy efficiency ratio (EER) was 3 times better than the GPU implementation. The accuracy of classification was not mentioned in the results.

In [17],[30], the authors presented an FPGA implementation of a single core K-NN classifier and compared it with an equivalent implementation running on general purpose

processors (GPP) in terms of time and the effect of data dimensionality. The implemented K-NN on FPGA because the classification of highly dimensional Microarray consumes a lot of time when implemented on GPPs. High parallelism and pipelining single core K-NN classifier architecture consist of four processing blocks, which were memory block, distance computation, K-NN finder, and class label finder. The design was parametrized and can be re-used for different number of training vectors (N) and the training samples (M) but the number of processing elements (PEs) of the internal architecture depended on N and M values, so the number PEs changed every time with N/M. The design was tested on GPP using MATLAB (R2009b) Bioinformatics toolbox running on an Intel dual core E5300 2.60 GHz, 3 GB RAM workstation. Xilinx ML 403 platform board which had Xilinx XC4VFX12-FF668-10 was used to test the design on FPGA. In [17], the synthetic data that contains 1024 training vector, 8 training sample and 16 bits per sample which mimic Microarray dataset was used in the test. The value of K was fixed at 13 during the test. The FPGA implementation was found to be 60.4x faster than the GPP implementation. The FPGA became 92x faster than GPP when the training sample increased from 8 to 30. In [30], dataset had 1024 training vector and 16 training sample used to test the design. The FPGA implementation was found to be 76x faster than the GPP implementation. The accuracy of classification was not mentioned in the results and since the K value was fixed during the test, so it is not clear if the K value has an effect on the speedup or not.

In [31],[23], the authors presented a systematic design of two linear array IP cores for k-NN classifier. The first architecture used M+k+1 processors and the second architecture used 2N+1 processors, where N is the number of vector and M the number of features. The data-path of the two designs were pipelined to reduce the classification time.

The design was parametrized and can be re-use for different M, N, and K, but the number of processors changed since it depended on M and K in architecture1, and it depends on N in architecture2. They needed two architectures because when N << M the first architecture did not fit in the FPGA and when N >> M the second architecture does not fit in the FPGA device. In [31] they run architecture 1 at 100 MHz on Virtex 2 Pro XC2VP30-6 FPGA board but in [23], they ran architecture1 at 100 MHz on Virtex 5 XC5VLX110T-3 FPGA. They compared the results for their design with the results of the other authors of [32], where the authors in [32] used CUDA API-based times that used a Pentium 4 PC, (2GB of DDR memory, running at 3.4 GHz) equipped with an NVIDIA GeForce 8800GTX GPU running at 575 MHz .The results based on the same instances are used in [33] and for different values of K and N. the results showed that in [31] the FPGA was 1.5x to 2.96x faster than the GPU, but in [23] the FPGA was slower the GPU most of the cases and in the best case the FPGA speed was equal to the speed of the GPU. They estimated the speedup when the design did not fit in the FPGA. The accuracy of classification is not mention in the results.

In [24], the authors proposed a SOC VLSI based on the K-NN algorithm and operating on both steady state and dynamic slope response of the data from the gas sensor array. They decided to use K-NN algorithm because of the simplicity of the algorithm which makes the hardware implementation simpler. They achieved significant saving in silicon area when they used a single analog processor to process the data of entire vector. Also, the circuit size was independent of the length of the vector and the number of data vectors. They faced leakage problem when the number of data pattern was too large where

the K-NN classifier would not work properly because the leakage current could discharge the pre-charged dynamic nodes of the current mirrors.

In [25], the authors explored the design space of hardware architecture of K-NN classification using stochastic computing (SC). The SC could be an appropriate solution to design area efficient K-NN hardware architecture because of the extensive use of multiplier in K-NN computation. “The SC only needed one logic gate to perform multiplication.” They developed K-NN hardware architecture Based on multi-line-based stochastic adder and multiplier. The number of required multiplier and adders slightly changed from two-lined-based to four- lined-based hardware. Handwritten digit recognition MNIST dataset was used to test the design classification accuracy. For average classification accuracy, 256-bit Stochastic bipolar, two-line and four lines achieved 84.24%, 84.52% and 84.41% respectively. For the best-case classification accuracy, the two-lined-based design achieved 5% improvement than the traditional bipolar design with 90.10% accuracy for 8-bit precision and 92.65% for 9-bit precision.

In [26], they investigated the feasibility of implementing k-NN classifier on vote count circuit. Low cost classification hardware was the reason of using a vote count circuit. Vote count circuit may be implemented using several different memory architectures like DRAM or NAND Flash memory. They implemented an EVC (enhanced vote count) prototype using 32*32 matrix of discrete nMOSFETs to emulate an FPGA as a controller. Software simulation was used to evaluate the vote count algorithm performance, since FPGA-based prototype was ROM-like and could not change its database or configuration parameters easily. They used two different datasets to test the classification accuracy. The first dataset had 210 objects, 7 vectors and 3 classes. For different odd values of K from 1

to 9, the best classification accuracy was around 90%. The second dataset had 310 objects, 6 vectors and 2 classes. Also for different odd values of K from 1 to 9, the best classification accuracy was around 80%.

CHAPTER III

DESIGN APPROACH AND DEVELOPMENT PLATFORM

In this chapter, we present our system level design and the design approach we used to design, develop, and implement the K-NN classification algorithm.

3.1 Hierarchical Platform-based Design Approach

We use hierarchical platform-based design approach to design our software and hardware. The main advantage of this approach is simplifying the hardware and the software design by using modular-based designs that can be reused [48]. As shown in Figure 1, higher-level functions utilize lower-level sub-functions and sub-component reuse by various functions.

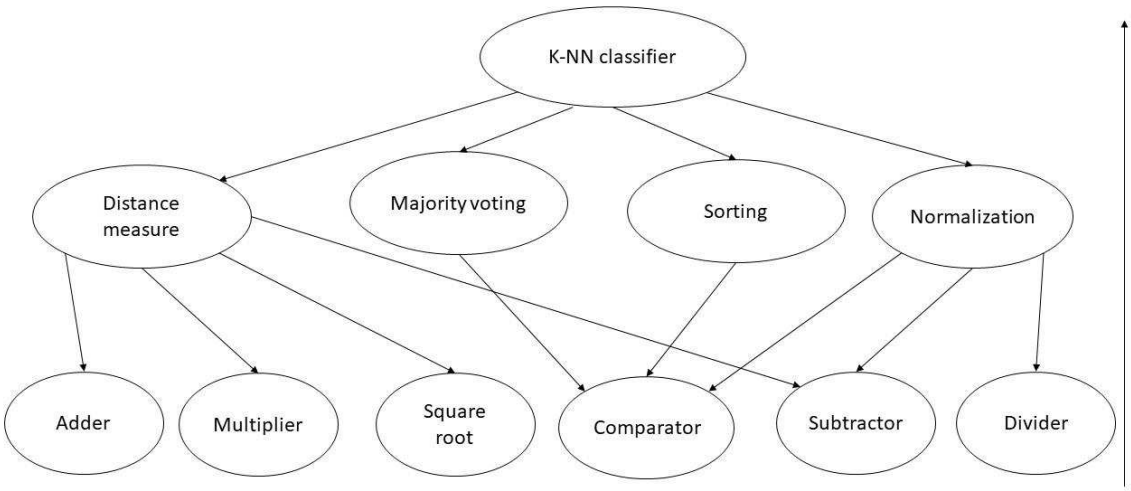

Figure 1: Hierarchical Platform-Based Design Approach

The lowest levels of the platform-based design hierarchy are the fundamental operators including addition, multiplication, square root, subtraction, division and comparator. The Distance Measure, the Normalization and the sorting operations are

implemented using these fundamental operators. The top-level K-NN classifier is developed by using the Normalization operations, the Distance Measure and the sorting.

We use single precision floating-point units [49] from Xilinx IP core library for the hardware design, where Multiplier, Divider and square root are selected from Xilinx IP core library while all the other functions and operators are designed using Verilog and developed on the FPGA. Also, for the software design, The MicroBlaze is also configured to use single precision floating-point unit for the software modules. The higher-level functions are developed in C and executed on the MicroBlaze.

3.2 System-Level Interface

The system-level interfacing for our hardware and software designs for the K-NN classifier is shown in figure 2.

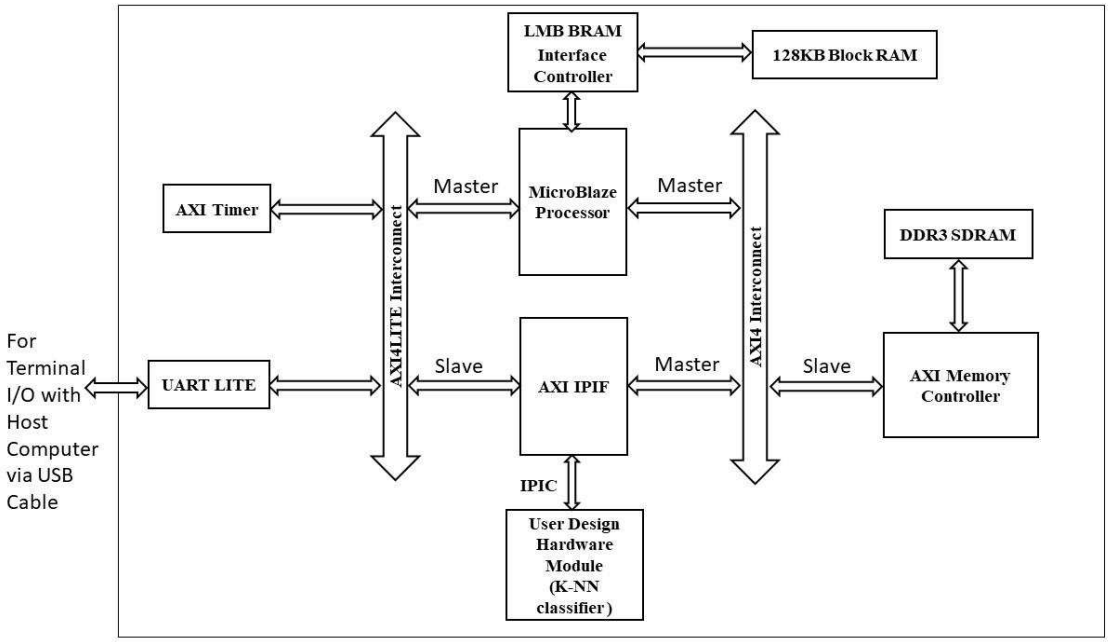

Figure 2: System level design

The software design is executed on the MicroBlaze Processor. In addition, the MicroBlaze Processor triggers the hardware design. XPS automated flow [50] is used to create the system-level design. The XPS generates wrapper to connect user defined IP cores with the necessary Xilinx IP cores.

The system-level design is developed using VHDL; our hardware design is developed using Verilog HDL. In Chapter IV we present a detailed description of our software and hardware architectures. The following sections give a brief description of the Xilinx IP cores used in the system.

3.2.1 MicroBlaze Soft Processor Core

The software design for the K-NN classifier is executed on the MicroBlaze. The processor is configured to run at 100 MHz clock. Single precision floating-point unit is used to perform accurate division, multiplication and square root operations for the K-NN classifier software design. The Instruction and Data caches for the processor are set at 128kB. In addition, the MicroBlaze runs as the primary host processor that controls the hardware designed for the K-NN classification algorithm.

3.2.2 DDR3 Controller

The DDR3 controller works at 200 MHz and has two AXI masters: MicroBlaze processor and the K-NN classifier hardware module. The controller directly connects to the DDR3-SDRAM pins via the FPGA I/O pins. The DDR3 controller. The DDR3 Controller module [51] provides an address-mapped AXI-based interface to access the SODIMM-based DDR3-SDRAM external memory.

3.2.3 DDR3-SDRAM External Memory

The ML605 board provides a physical 512MB Dual In-line Memory Module (DIMM)-based DDR3-SDRAM for data storage. The DDR3-SDRAM works at 200MHz giving an effective memory access rate of 400 MHz. For our design, the memory is parametrized and partitioned into three regions: the dataset region, the hardware region and the software region. The processor can map these regions to non-overlapping memory locations via register programming. The addresses of the three regions are changing depends on the dataset size. The dataset region is where the dataset will be saved. The hardware region is where the hardware steps results will be saved. The software region is where the software steps results will be saved. The organization of the DDR3-SDRAM for our design is shown if figure 3.

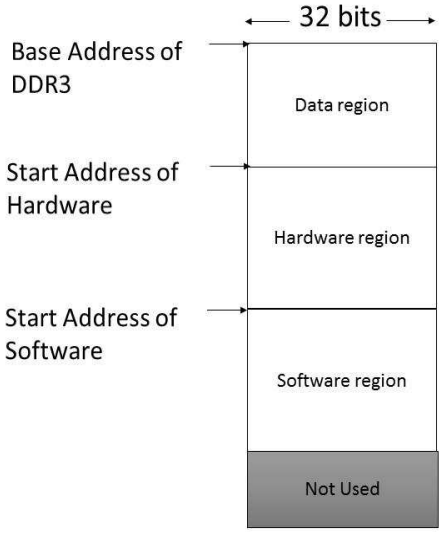

Figure 3: DDR3-SDRAM Organization

3.2.4 AXI Interconnect

AXI interface connects the AXI-based masters and slaves in the system. It consists of two instances [52] – the AXI4 and the AXI4-Lite.

The AXI4-Lite protocol is a subset of the AXI4 protocol with reduced signals and single beat transfers [52]. The AXI4 instance connects one slave, which is AXI memory controller that control on the DDR3 SDRAM to two masters, the MicroBlaze and AXI IPIF. The AXI4-Lite connects one master, which is the MicroBlaze processor to AXI timer. In addition, The AXI4 instance connects one slave, which is AXI IPIF to the UART LITE.

3.2.5 AXI Timer

We use AXI timer to measure the execution time of the hardware and the software. The AXI timer is connected as slave to the MicroBlaze via AXI4-lite.

3.2.6 UART Lite

The MicroBlaze processor can communicate with the host computer via a Universal Serial Bus (USB) connection Using RS232-based UART Lite [53]. The UART is configured for a baud rate of 115200 with 8-bit Data, no parity bit, 1 stop bit and none flow control. For our experiments, the benchmark dataset is sent from the host computer to the DDR3-SDRAM via MicroBlaze using the UART. The results for every step of the software and the hardware including the execution time are transmitted back to the host computer via this interface.

3.2.7 IP Version and Memory Map

The Xilinx IP core versions and the corresponding memory-mapped addresses is shown in Table 3-1. XPS is generated the address map automatically.

Table 3-1: Memory map and IP Version.

IP Name	IP Type	IP Version	Base Address	High Address
AXI4	AXI_INTERCONNECT	1.06.a	N/A	N/A
AXI4LITE	AXI_INTERCONNECT	1.06.a	N/A	N/A
MICROBLAZE_DLMB	LMB_V10	2.00.b	0X00000000	0X0001FFFF
MICROBLAZE_ILMB	LMB_V10	2.00.b	0X00000000	0X0001FFFF
MICROBLAZE	MICROBLAZE	8.50.c	0X41200000	0X4120FFFF
MICROBLAZE_BRAM_BLOCK	BRAM_BLOCK	1.00.a	N/A	N/A
MICROBLAZE_D_BRAM_CTRL	LMB_BRAM_IF_CNTLR	3.10.c	N/A	N/A
MICROBLAZE_I_BRAM_CTRL	LMB_BRAM_IF_CNTLR	3.10.c	N/A	N/A
DDR_SDRAM	AXI_V6_DDRX	1.06.a	0XA4000000	0XA7FFFFFF
AXI_TIMER	AXI_TIMER	1.03.a	0X41C00000	OX41C0FFFF

3.3 Experimental Platform

Our experiments are executed by using Xilinx ML605 development platform [43]. The platform uses a Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA [44], which consists of 768 XtremeDSP™ Slices, 15MB of Block RAMS, 240K logic gates, and 3.6Kb of distributed RAMs. Also, it consists of 512MB DDR3-SDRAM external memory and various off-chip non-volatile memories including: 32MB BPI Linear Flash, 128MB of Platform Flash XL and 2GB Compact Flash, to hold the configuration bitstreams, which is

a binary file that sets all the FPGA's programmable bit locations to configure the logic and routing resources appropriately. The DDR3-SDRAM external memory can be scaled up to 2GB to fit the design needs. The board contains Joint Test Access Group (JTAG) and Universal Asynchronous Receiver/Transmitter (UART) ports to connect to a host processor via Universal Serial Bus (USB). Xilinx Platform Studio (XPS) and the Xilinx Software Development Kit (SDK) were used to program the FPGA.

The development platform also consists of The MicroBlaze embedded processor, which reduced instruction set computer (RISC) optimized for implementation in Xilinx Field Programmable Gate Arrays (FPGAs). To communicate with other peripherals in the system the MicroBlaze embedded processor uses the Advanced Extensible Interface (AXI) bus [46]. The user-defined peripherals integrated through AXI interface can be connected to any processor compatible with AXI infrastructure that allows the peripherals to be re-used in similar systems [47]. Using Xilinx Platform Studio (XPS), we integrated our custom-based design for K-NN classifier, MicroBlaze processor, DDR3 controller via AXI interface and other Xilinx IP cores for UART. The address space for each peripheral in the system-level design were automatically is assigned by XPS and mapped it to the processor addressable range.

We use Verilog to design our hardware architecture, and VHDL to implement Xilinx IP cores. The system level design is built using XPS where 100 MHZ clock is used for processor frequency. Our software design is written in C and executed on the MicroBlaze processor (using the SDK) to make a fair comparison with the hardware design implemented on the same development platform. The software design on MicroBlaze is

compiled with level 2 optimization. Level 2 is selected over other optimization levels as it activates nearly all optimizations that do not involve a speed-space tradeoff.

We use UART to send and store the benchmark dataset to the DDR3-SDRAM via MicroBlaze processor, where MicroBlaze collects the data from the UART and writes it to the external memory. Experiments are performed on the benchmark dataset to evaluate the hardware design over the software designs for the K-NN classifier.

3.4 Benchmark Dataset

Initially, small synthetic dataset is used to evaluate the results of our software and hardware design. The synthetic dataset consists of 20 vectors, each vector has 13 attributes and the class of every vector is given. The dataset is split into two parts. The training set has the first 15 vectors with the class of the vectors is given and the test set has 5 vectors where the class of the vectors is eliminated.

Next, we tested our design with a real benchmark dataset with substantial number of vectors. We use three different type of benchmark with free distinct size.

The first benchmark is the iris database obtained from UCI Machine Learning Repository [54]. It has 150 instances, 4 attributes, and 3 classes.

The second benchmark is the heart database obtained from UCI Machine Learning Repository [55]. It has 270 instances, 13 attributes, and 2 classes.

The third benchmark is the user knowledge modeling database obtained from UCI Machine Learning Repository [56]. It has 403 instances, 5 attributes, and 4 classes.

Our experimental results and analysis are presented in chapter V.

CHAPTER IV

EFFICIENT EMBEDDED ARCHITECTURE FOR K-NN ALGORITHM

In this chapter, we present our hardware/ software architecture and the Xilinx IP cores that we used in our design.

4.1 Embedded Software Architecture

We started our design by writing the C code of the K-NN algorithm on Code blocks [57] to verify the K-NN algorithm functionality before we use it on MicroBlaze. We use Xilinx SDK to run the C code on MicroBlaze. The software flow design is shown in figure 4.

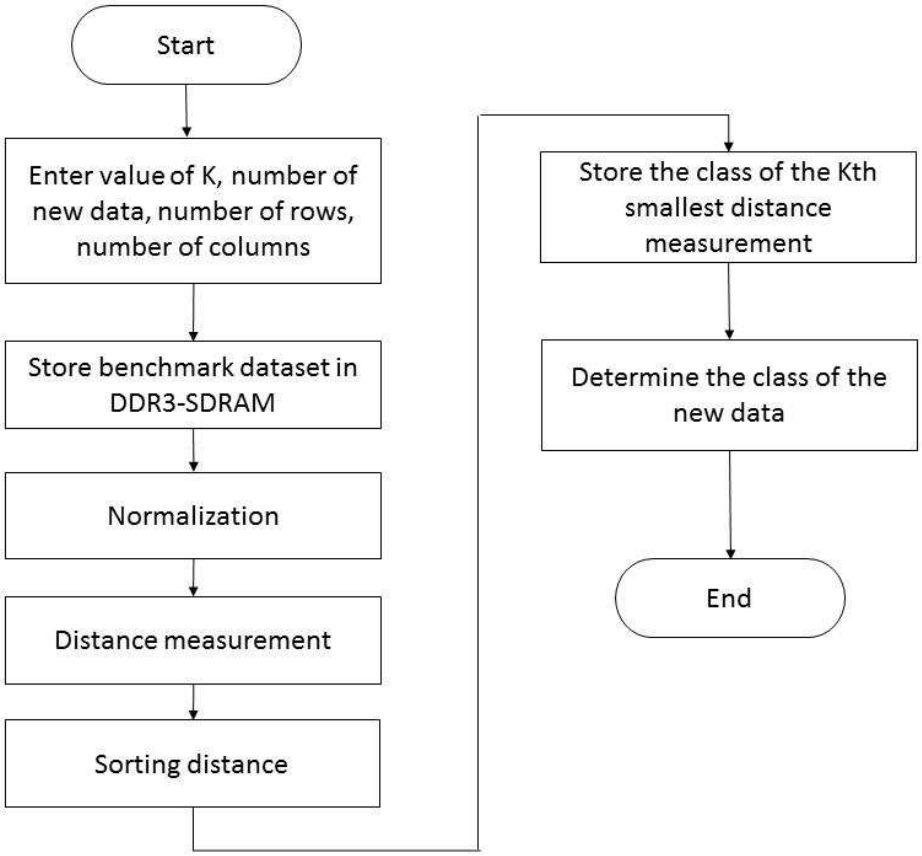

Figure 4: Software flow design

Step 1, the program starts with entering the value of K, number of new data, number of rows and number of columns of the dataset. Choosing the value of K is discussed in chapter 2. Number of new data represents the number of the data in the testing set. Number of rows and number of columns are given in any dataset description. We use synthetic dataset consist of 20 vectors, each vector has 13 attributes and the class of every vector is given. We split the dataset into two sets, training set and test set. The training set has the first 15 vectors with the class of the vectors is given and the test set has 5 vectors where the class of the vectors is eliminated.

Step 2, we use the host computer to send the synthetic dataset to the DDR3-SDRAM via MicroBlaze using the UART as we explain in chapter 3 section 3.3.6.

Step 3 is normalizing the data set. We use min-max normalization as shown in the equation 6 (chapter 2) to normalize the dataset.

$$XN = \frac{X - \min(X)}{range(X)} = \frac{X - \min(X)}{\max(X) - \min(X)} \quad (6)$$

We split this step into two parts for the sake of ease of debug and verifying. The first part is to find the minimum and maximum instance (min(x) and max(x)) in every column of the dataset. The second part is subtracting min(x) from every instance in the dataset then divide the result on the result of subtraction of min(x) from max(x). We use print function to verify the normalization results.

Step 4 is finding the distance between the test set vectors and the training set vectors. We use Euclidean measurement to find the distance. The equation of the Euclidean measurement is shown in equation 1(chapter 2).

$$\text{Euclidean: } \mathrm{D}(\mathrm{x},\mathrm{y}) = \left(\sum_{i=1}^{m} |xi - yi|^{2} \right)^{1/2} \quad (1)$$

We split this step into three parts for the sake of ease of debug and verifying. The first part is to subtract the attributes of each vector in the test set from the attributes of every vector in the training set. The second part is to multiply the result from the first part by two. The third part is finding the square root of the summation of the second part. We use print function to verify the results of this step.

Step 5, sorting the distance results from step 3. We sort the distance results in ascending order. Also, the row number of each distance measurements is store so the algorithm can obtain the class associated with that row number from the dataset in the next step.

Step 6, storing the class of the Kth smallest distance measurement. The algorithm obtains the class of Kth smallest distance measurement from the dataset and store it.

Step 7, determining the class of each vector in the test set by applying majority voting. We split this step to 3 parts. The first part is to find the iteration of the kth class for each vector. The second part is to sort the iteration of the kth class for each vector in ascending order. The third part is to determine the class of each vector in the test set.

Figure 5 and figure 6 are illustration for the work of K-NN algorithm. The scenario in figure 5 is that a dataset has two classes (class A and class B) and there is new data with a known a class (the question mark symbol in figure 1) and the user choose the number of K is equal to 3. The algorithm measures the distance between the new data with the unknown class and all the data in the training set with the known classes and sort the results

in ascending order. Since K is 3, the algorithm applies a majority voting on the nearest three class to the data with unknown class. In this case the nearest three class to the data with unknown class shown in figure 1 are two A and one B. The algorithm selects A as the new class of the unknown class.

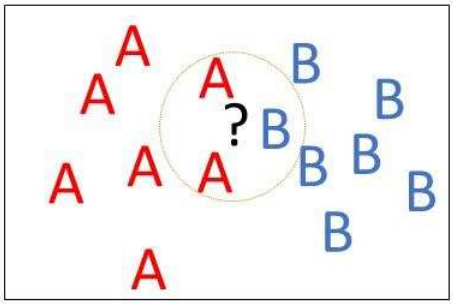

Figure 5: K=3, dataset has two classes.

The scenario in figure 6 is that a dataset has three classes (class A, class B and class C) and there is new data with a known a class (the question mark symbol in figure 1) and the user choose the number of K is equal to 3. Since K is 3, the algorithm applies a majority voting on the nearest three class to the data with unknown class. In this case the nearest three class to the data with unknown class are: one class A, one class B and one class C so no class has the majority. The algorithm chooses the class with smallest distance measurement, in other words, the algorithm choose the closest class to the data with the unknown class.

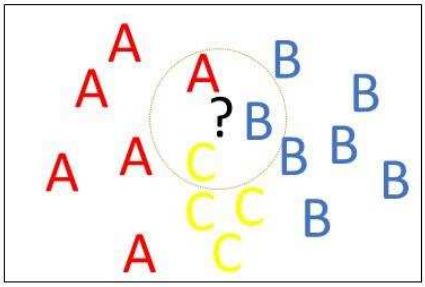

Figure 6: K=3, dataset has three classes.

4.2 Embedded Hardware Architecture

During the design and development, we partitioned our hardware architecture into three layers modules: User-Design module, Top-level module and K-NN classifier module. All the modules are designed in Verilog using Xilinx ISE 14.7 [58]. We used Xilinx IP cores for the division, multiplication, square root operators and for the Block Random Access Memories (BRAM). Also, for some cases we used Xilinx IP cores to generate a comparator. The high-level module is shown in figure 7.

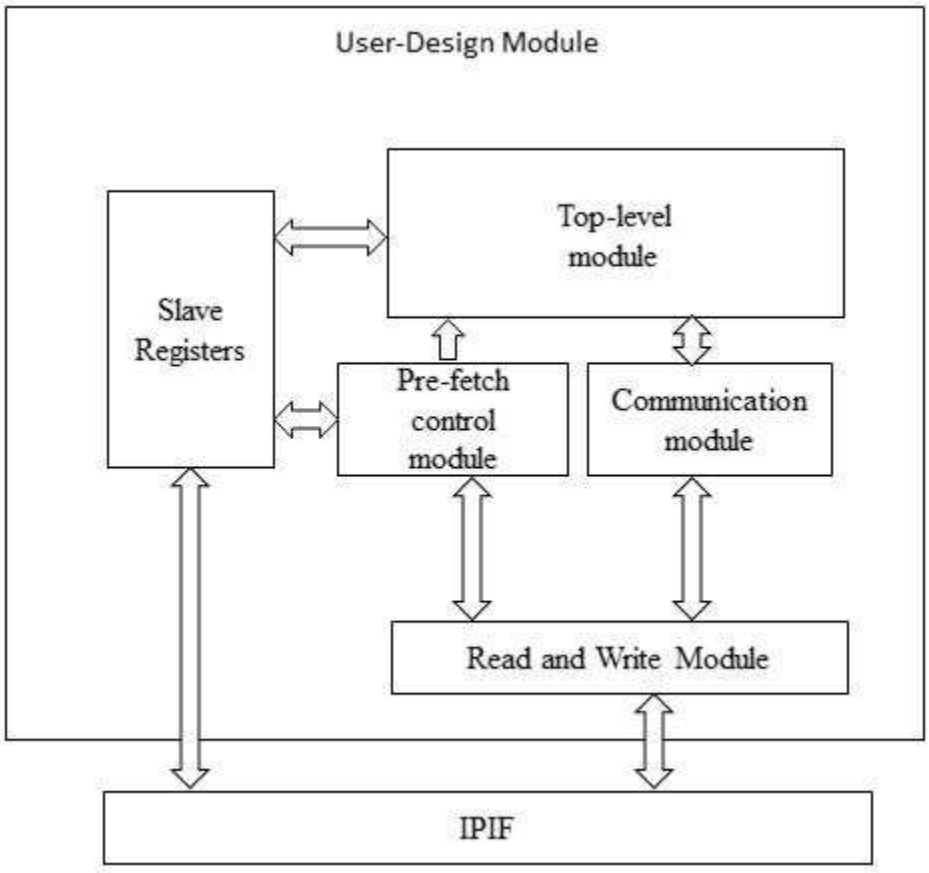

Figure 7: High-level architecture

The user-design module connects the read and write module to the MicroBlaze via IPIF and to the Top-level module via the Pre-fetch control module and the communication module. Also, the user-design module connects the slave registers directly to the

MicroBlaze via IPIF and connects the communication module, the pre-fetch control module and the slave register to the Top-level module. The Top-level module is shown in Figure 8.

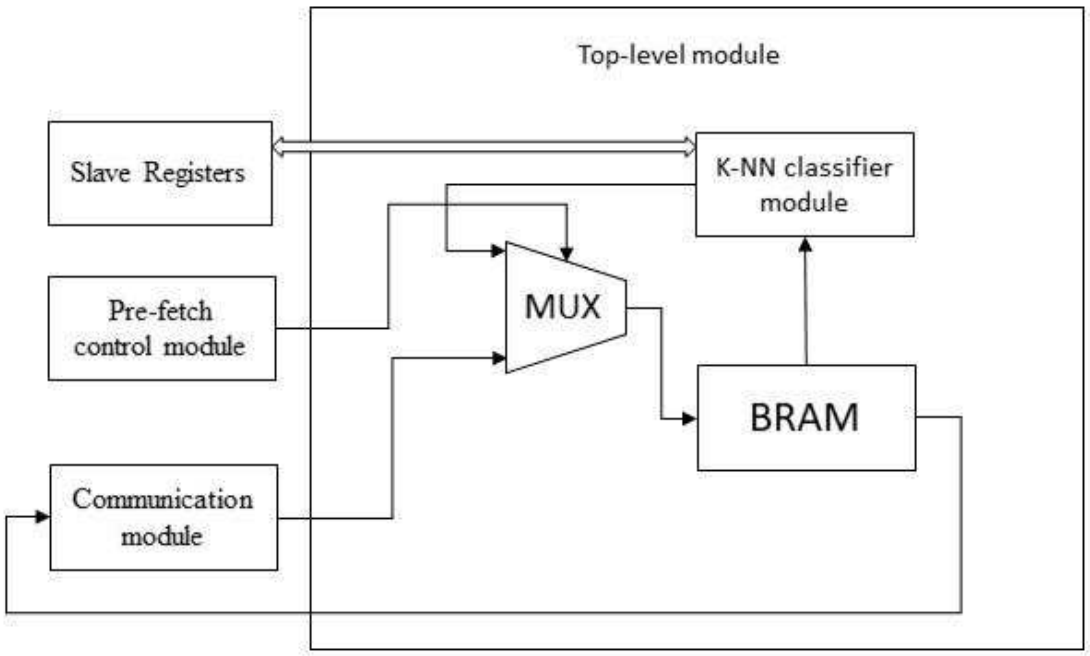

Figure 8: Top-level module

The user-design module receives signals from the MicroBlaze using IPIF via AXI4 bus. We used the MicroBlaze to trigger the hardware and start processing, read, write data and results from and to the BRAM in the top-level module. The slave registers receive signals from the MicroBlaze to trigger the K-NN classifier modules, also the slave registers receive signals from K-NN classifier and send it back to the MicroBlaze when the process of each K-NN steps is done. The communication module is used to control the data flow from the DDR3-SDRAM to the BRAM and from the BRAM to the DDR3-SDRAM. Accessing the DDR3-SDRAM takes significant amount of time so we use pre-fetch control module to overcome this issue. In reading mode, the required data is pre-fetched and store

in the BRAM, and K-NN classifiers can access the BRAM directly. In the writing mode, the data is forward from the K-NN classifier to the BRAM, and then from the BRAM to the DDR3-SDRAM via the communication module. The K-NN classifier hardware data path modules will be explained thoroughly in the next sub-sections.

4.2.1 Step 1 normalization

We used Min-Max normalization in equation 6 to normalize the dataset. The computation data paths for the Min-Max normalization are depicted in figure 9 and figure 10 respectively. The data path in figure 9 contains of three registers and two comparators. Clk, Reset and Load signals are the signals that the control path use to control on the data path.

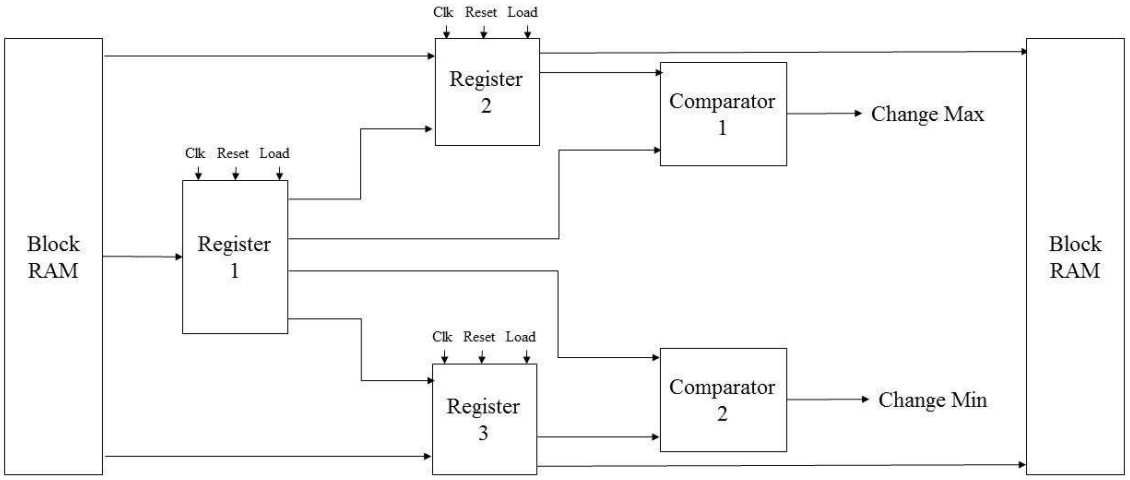

Figure 9: Min-Max data path

The goal of this data path is finding the minimum and the maximum value in each column of the dataset. The process starts by load the first value in the first column from the BRAM and store it in register2 and register3 and load the second value in first column and store it in register1. Comparator1 compare between the value in register2 and the value in register 1. If the value in register2 is smaller than the value in register1 then “change Max”

signal will be send from comparator1 to the control path. The value in register2 will be assign as Max, otherwise no change in Max will accrue. Comparator2 compare between the value in register3 and the value in register1. If the value in register3 is greater than the value in register1 then “change Min” signal will be send from comparator2 to the control path and the value in register1 will be assign as Min, otherwise no change in Min will accrue. The process continues for every instance in every column to find the Max and the Min value in every column in the dataset and store in the BRAM.

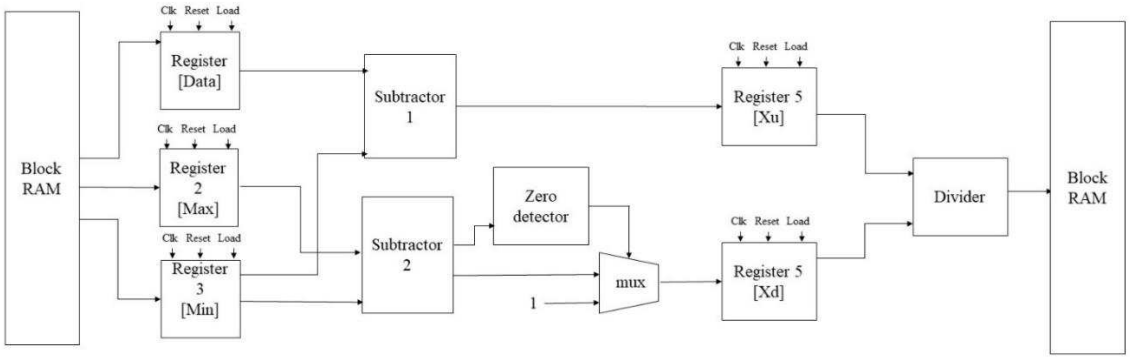

Figure 10: Normalization data path

The data path in figure 10 contains of five registers, two subtractor, zero detector, mux and divider. The goal of this data path is finding normalization results. The Max and the Min values of each column are read from the BRAM and stores in register2 and register3 respectively. The subtractor2 subtracts the min value from max value for each column and the zero detector checks if the subtraction value is equal to zero or not. If the subtraction result is equal to zero then the multiplexer will insert number one to registerXd, else the subtraction value will be inserted to registerXd. The need for the Zero detector is to prevent division by zero. The subtractor1 subtracts the Min value from every instance in each

column and store the result in registerXu. The divider divides “Xu” over “Xd” and stores the result in the BRAM.

4.2.2 Step 2 distance measurement

We use Euclidian distance in equation 1 to measure the distance between the test set and the training set. The computation data paths for the distance measurement are depicted in figure 11 and figure 12. We use two data paths to simplify the design and for the sake of ease of debug and verifying. The data path shown in figure 11 consists of four registers, a subtractor, and a multiplier.

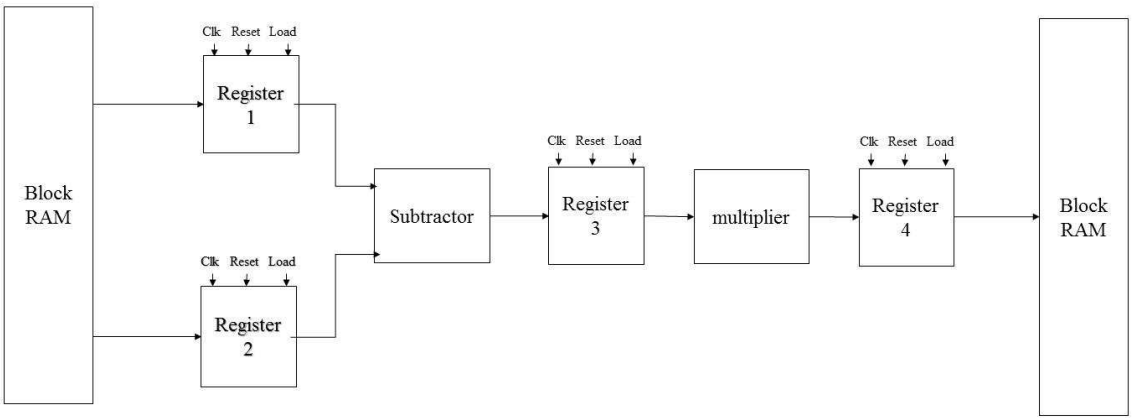

Figure 11: Distance measurement data path 1

The first instance in the training set and the test set are loaded from the BRAM and stored in register1 and register2 respectively. The subtractor subtracts the value in register2 from the value in register1. The subtraction result is stored in register3 then multiplied by 2 and stored in register4. The result in register 4 is store in the BRAM. The process is repeated for every instance in the training set to find the distance between every instance in the test set with all the data in the training set.

The data path shown in figure 12 consists of two registers, an accumulator, and a square root. The accumulator is designed as a sequence of an adder and an accumulator register with a feedback loop to the adder.

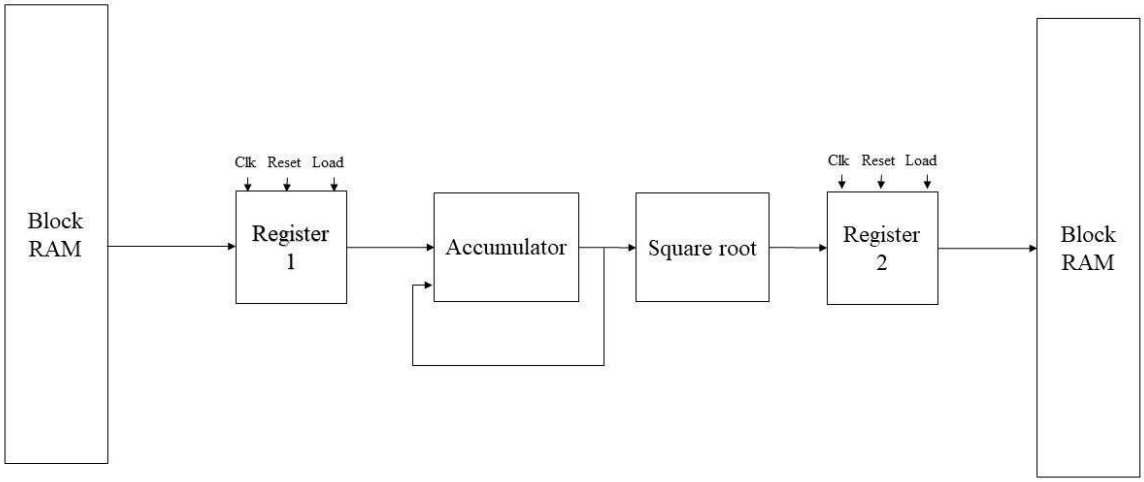

Figure 12: Distance measurement data path 2

The results of the first data are loaded to register1 then to the accumulator to find the summation results. Square root is performed on of the summation result to find the distance, which is store in the register 2. The process is repeated to find the distance result for every row in the test set and is stored in the BRAM.

4.2.3 Step 3 Sorting

The computation data path for the sorting is depicted in figure 13. The data path contains of four registers and one comparator. The goal of this data path is to sort the distance measurements in ascending order.

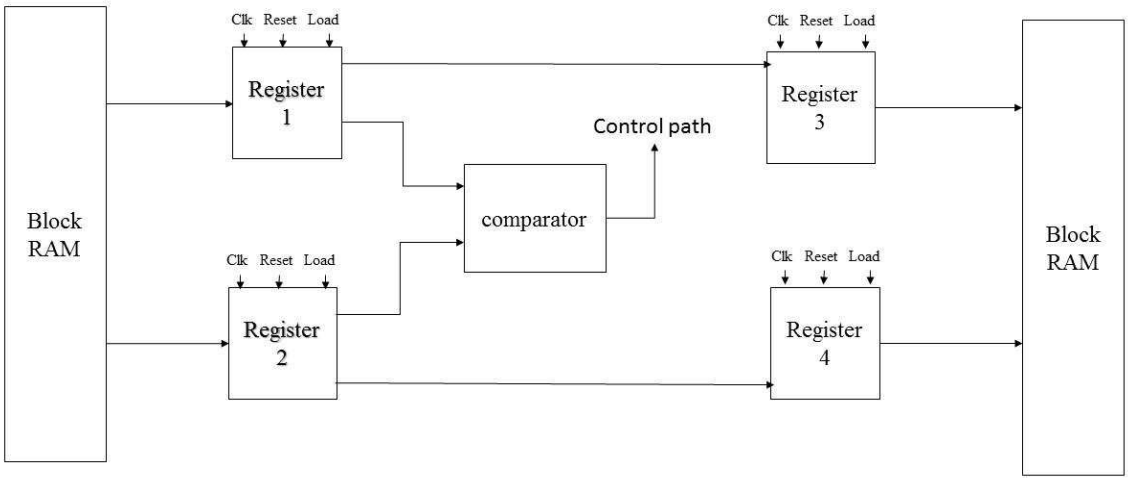

Figure 13: Sorting distance data path

The first and the second distance measurement are loaded form the BRAM and stored in register1 and register2 respectively. The comparator checks if the first distance 1 greater than the second distance, then sends a signal to the control path to store the data in register3 and register4. The data in register3 stores at the same memory address of the data in resister 2 in the BRAM and the data in register4 will store at the same memory address of the data in resister 1 in the BRAM. If the first distance not greater than the second distance, a signal is sent to the control path to get new data. The sorting is done when all the distance measurements are checked. The same data path in figure 8 is used to store the number of the rows associated with the distance measurements, so the algorithm can obtain the class that associated with the number of the rows from the dataset in the next step.

4.2.4 Step 4 new class determination

To determine the class of the new data, the algorithm follow the steps below:

- Finding the Kth nearest classes to the new data with the unknown class.
- Finding the class iteration of Kth nearest classes and sort the results in ascending order.
- Applying majority voting to decide the class of the new data.

The computation data paths for this step is depicted in figure 14, figure 15, and figure 16 respectively.

The data path shown in figure 14 has only two registers. The goal of this data path is to obtain the Kth nearest classes to the new data with the unknown class from the dataset.

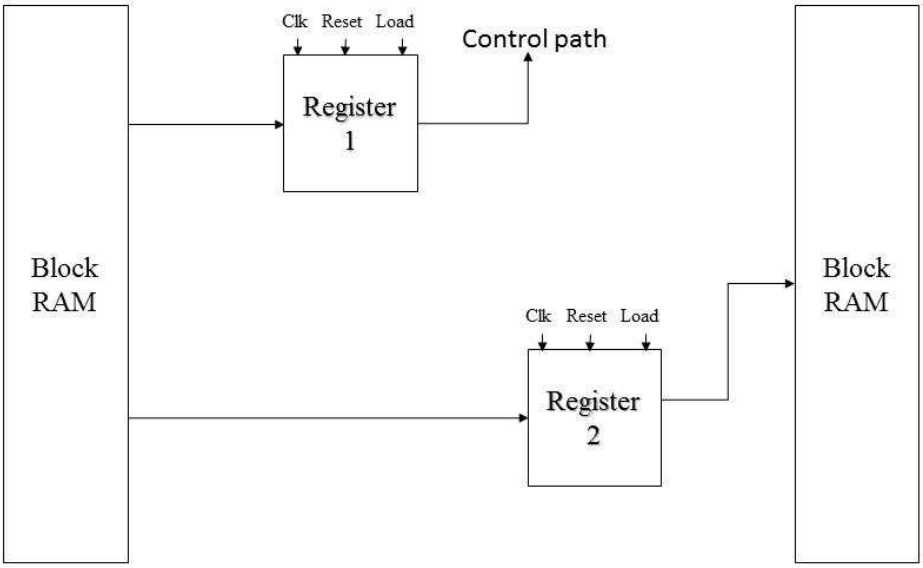

Figure 14: Kth class data path

The number of the rows is needed to obtain the class of training data from the dataset. The number of the rows of the kth smallest distance measurement is loaded from the BRAM and is stored in register1. A signal sends back to the control path from register1 so the

address of class of the kth smallest distance measurements can be determined and stored in register2 then stored in new address in the BRAM.

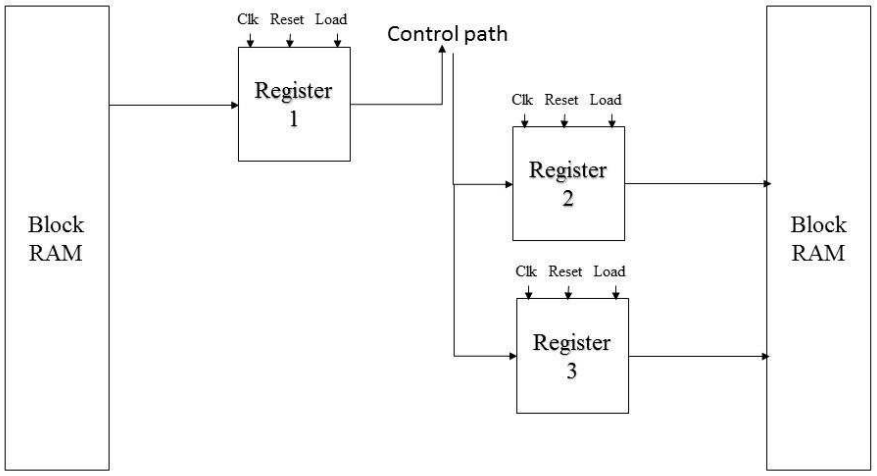

Figure 15: Class counter data path

The data path shown in figure 15 has three registers. The goal of this data path is to calculate the iterations of the Kth nearest classes. The Kth nearest classes are loaded from the BRAM and stored in register1. A signal sends to the control path from register1, so a counter in the control path counts the iteration of each class and send the results to register2 and register3. The number of each class is stored in register2 and the iteration of each class is stored in register3. The results will store back to the BRAM. We use the same data path that is shown in figure 8 to sort the iteration of the Kth nearest classes in ascending order.

The data path shown in figure 16 has five registers and two comparators. The goal of this data path is to determine the class of the new data with the unknown class by applying majority voting.

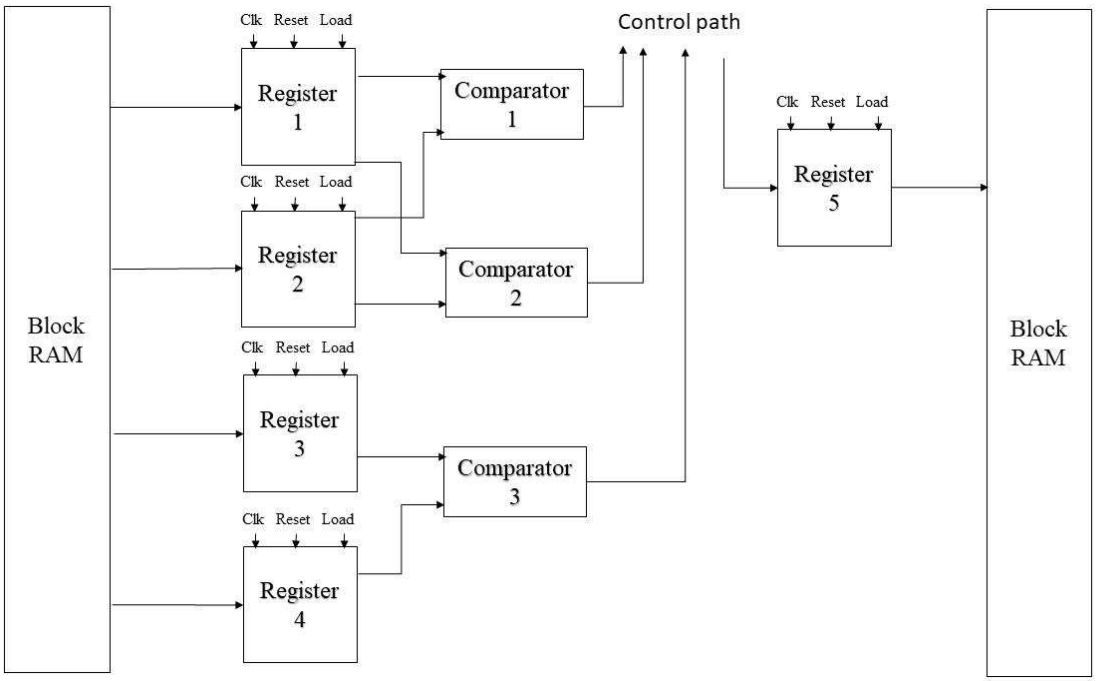

Figure 16: The data path to find the class of the new data

The iterations of each class are stored in ascending order, therefore finding the new class is an easy task. The classes with the highest two iterations are loaded from the BRAM and stores in register1 and register2. Comparator1 compares if the iteration value in register1 is greater than the iteration value in register2; if that is the case the new class is the class in register1. Otherwise the value in register1 is equal to the value in register2. The class with the third highest iteration will be load from the BRAM to register1. Compactor2 compare if the iteration value in register1 is equal to the iteration value in register2, and then send a signal to the controller to increase a counter value in the counter. The counter counts the number of the classes that has the same iteration. The process does not stop until finding all the k nearest classes with the same iteration. The class of the smallest distance measurement is loaded from BRAM and stored in register3. One of the classes that has the same iteration is stored in register3. Comparator3 compares if the class in register3 is equal

to the class register4. If the two classes are equal a signal is sent to the controller to store the class in register4 in register5 as the new class. Otherwise another class of the classes with the same iteration will be loaded form BRAM and store in register4. This process continues until comparing all the classes that has the same iteration with the class of the smallest distance measurement. In case none of the classes is equal to the class of the smallest distance measurement, the class of the second smallest distance measurement is loaded from the BRAM and store in register3 and the comparison will be repeated. The process is repeated until finding the new class of the new data. The new class is stored in register5 then to the BRAM.

4.3 Xilinx LogiCORE CoreGen IPs

In this section, we present the Xilinx IP cores that we use in our hardware architecture.

4.3.1 Multiplier

This module performs floating point multiplication [49]. The configuration of the multiplier we used is shown in Table 4-1.

Table 4-1: Multiplier IP configuration

Configuration Type	Values
precision	Floating point single precision
width	32 bits
Architecture optimization	High speed
Latency	0
DSP48E usage	Max usage

4.3.2 Subtractor

This module performs floating point subtraction [49]. The specification of the multiplier we used is shown in Table 4-2.

Table 4-2: Subtractor IP configuration

Configuration Type	Values
precision	Floating point single precision
width	32 bits
Architecture optimization	High speed
Latency	0
DSP48E usage	Full usage

4.3.3 Adder

This module performs floating point addition [49]. The specification of the multiplier we used is shown in Table 4-2.

Table 4-3: Adder IP configuration

Configuration Type	Values
precision	Floating point single precision
width	32 bits
Architecture optimization	High speed
Latency	0
DSP48E usage	Full usage

4.3.3 Divider

This module performs floating point division [49]. The specification of the multiplier we used is shown in Table 4-2.

Table 4-4: Divider IP configuration

Configuration Type	Values
precision	Floating point single precision
width	32 bits
Architecture optimization	High speed
Latency	0
DSP48E usage	No usage
Cycle per operation	1

4.3.3 Square root

This module performs floating point Square root [49]. The specification of the multiplier we used is shown in Table 4-2.

Table 4-5: Square root IP configuration

Configuration Type	Values
precision	Floating point single precision
width	32 bits
Architecture optimization	High speed
Latency	0
DSP48E usage	No usage
Cycle per operation	1

4.3.4 Comparator

This module performs floating point Square root [49]. The specification of the multiplier we used is shown in Table 4-2.

Table 4-6: Comparator IP configuration

Configuration Type	Values
precision	Floating point single precision
width	32 bits
Architecture optimization	High speed
Latency	0
DSP48E usage	No usage
Cycle per operation	1

4.3.5 BRAM

This module performs floating point Square root [49]. The specification of the multiplier we used is shown in Table 4-2.

Table 4-7: Subtractor IP configuration

Configuration Type	Values
Memory type	True dual port RAM
Interface type	Native
Clocking option	Common clock
BRAM primitives	Minimum area
Write width	32 bits
read width	32 bits
Write depth	252864

4.4 Hardware Parameter List

Our hardware architecture is generic and parametrized; hence the user can apply any classification dataset by changing some of the parameters listed in table 4-8. *NumRows* and *NumColumn* parameters control the size of the dataset to be processed by specifying: the total number of vectors, the number of attributes per vector. *Number_of_new_data* helps the user to split any dataset to training set and a test set. *NumClass* is to specify the number of classes that the dataset has. *K* is the number of neighbors that can vote to decide the new class of the new data with the unknown class.

Table 4-8: Hardware parameters

Parameter	Description
NumRows	Number of vectors in the dataset
NumColumn	Number of attributes per vector in the dataset
number_of_new_data	Number of unclassified rows in the dataset
NumClass	Class numbers in the dataset
K	K value

4.5 Proposed parallel architecture

The distance measurement step in in section 4.2.3 requires a significant amount of time for large dataset since it measure the data between every instance in the test set with every instance in the training set therefore we propose a parallel architecture to decrease the processing time by nearly 2. The proposed architecture is shown in figure 17.

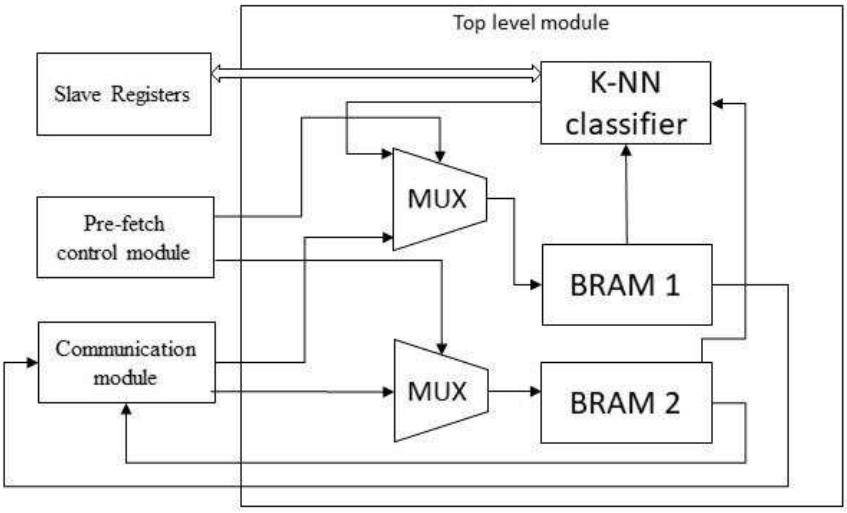

Figure 17: Proposed parallel architecture

By adding another BRAM and another MUX to the original architecture it is possible to split the test set into two parts and save the first part in BRAM1 and the second part in BRAM2. In this way K-NN classifier can read/write from four ports instead of two ports which leads to finish the process in half of the previous time.

CHAPTER V

EXPERIMENTAL RESULTS AND ANALYSIS

In this chapter, we present our experimental results and analysis performed to evaluate the efficiency of our hardware architecture. Our hardware design is executed on virtex-6 FPGA with 100MHz clock frequency. Our software design is executed on the MicroBlaze processor running at 100MHz on the same FPGA. We use three different benchmark datasets with distinct attribute and different size. The test set is taken as a percentage of the dataset to test the classification accuracy for different scenarios; where the test set can be larger, equal or smaller than the training set. The test set start as 10% of the dataset and increase by 10% up to 90% of the dataset.

5.1 Execution time for CPU, Embedded HW, Embedded SW and classification accuracy

To evaluate the speed performance of hardware over software, we measure the execution time (in processor clock cycles) for both hardware and software designs. The initial time taken to store the benchmark dataset in the DDR3-SDRAM is excluded from both the software and hardware time. We use equation 8 to calculate the speedup.

$$Speedup = \frac{Software\ excution\ time}{Hardware\ excution\ time} \tag{8}$$

Furthermore, the execution time of the K-NN classifier C code running on a PC with Intel(R) Core™ i7-4700MQ CPU @ 2.4 GHz, 8 GB RAM running on windows 10 pro is presented in milliseconds(ms).

We use three different benchmark datasets; includes Iris benchmark dataset [54], heart benchmark dataset [55], and user knowledge modeling benchmark dataset [56] to test

our design. We split each dataset into training set and test set. We eliminate the classes in the test set.

We run a C code to count classification error at the end of each test. The C code compares the classification result of our test with the original classes that is eliminated from the test set. An error occurs when the new class that assigns by the K-NN classifier is different from the original class of the test set. We calculate the classification accuracy manually using equation 9 below.

$$classification\ accuracy = \frac{number\ of\ test\ set\ instances - classification\ error}{number\ of\ test\ set\ instances} \times 100\% \quad (9)$$

The experimental results are presented in the next sub-sections.

5.1.1 Iris benchmark dataset

The results of the Iris benchmark dataset [same from ch3] are represent in the Tables 5-1 to 5-5.

Table 5-1: Execution time, speedup and accuracy for iris benchmark when K =1

K =1	Test set percent of the dataset	CPU execution time /(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	69505591	1043880	66.58	0	100%
	20%	15	111860710	1622892	68.93	1	96.67%
	30%	15	133272609	1922506	69.32	3	93.33%
	40%	15	135224117	1873170	72.19	3	95.00%
	50%	15	122993528	1656100	74.27	6	92.00%
	60%	15	97145005	1270848	76.44	4	95.56%
	70%	15	69558087	865296	80.39	7	93.33%
	80%	1	42983684	461956	93.05	3	97.50%
	90%	1	16543167	155062	106.69	17	87.41%

Table 5-2: Execution time, speedup and accuracy for iris benchmark when K =3

K =3	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	70124286	1054842	66.48	0	100.00%
	20%	15	125059601	1623508	77.03	2	93.33%
	30%	15	133302322	1923444	69.30	3	93.33%
	40%	15	135258958	1874388	72.16	1	98.33%
	50%	1	123065895	1657640	74.24	3	96.00%
	60%	1	97209643	1272696	76.38	4	95.56%
	70%	1	69648482	867438	80.29	5	95.24%
	80%	1	43088551	464434	92.78	6	95.00%
	90%	1	16586229	157862	105.07	17	87.41%

Table 5-3: Execution time, speedup and accuracy for iris benchmark when K =5

K =5	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	69510729	1039372	66.88	0	100.00%
	20%	4	115943918	1624124	71.39	2	93.33%
	30%	6	133351425	1924354	69.29	3	93.33%
	40%	15	135308466	1875634	72.14	3	95.00%
	50%	1	123117987	1659194	74.20	5	93.33%
	60%	1	97284421	1274516	76.33	5	94.44%
	70%	3	70176854	866556	80.98	7	93.33%
	80%	1	43411145	466954	92.96	9	92.50%
	90%	1	16767524	160718	104.33	11	91.85%

Table 5-4: Execution time, speedup and accuracy for iris benchmark when K =7

K =7	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	69543134	1044790	66.56	0	100%
	20%	4	111935479	1624740	68.89	2	93.33%
	30%	6	133378014	1925278	69.28	3	93.33%
	40%	6	135358342	1876908	72.12	3	95.00%
	50%	15	123180184	1660818	74.17	6	92.00%
	60%	1	97360508	1276350	76.28	7	92.22%
	70%	1	70265589	871722	80.61	8	92.38%
	80%	1	43513877	469586	92.66	8	93.33%
	90%	1	16880866	163602	103.18	37	72.59%

Table 5-5: Execution time, speedup and accuracy for iris benchmark when K =9

K =9	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	69555030	1045098	66.55	0	100%
	20%	15	111960111	1625328	68.88	3	90.00%
	30%	15	133413935	1926174	69.29	3	93.33%
	40%	6	135459395	1859618	72.84	2	96.67%
	50%	15	123252867	1662260	74.15	5	93.33%
	60%	15	97432909	1278184	76.23	7	92.22%
	70%	1	70345459	873878	80.49	7	93.33%
	80%	1	43608626	471910	92.41	8	93.33%
	90%	1	16990819	166248	102.20	46	65.93%

From Tables 5-1 to 5-5, it is observed that the best classification accuracy, which is 100% occurs when the percentage of the test set is 10% of the dataset, for all the K values. When the size of the test set increase and the size of the training set decrease, the accuracy typically tend to decrease.

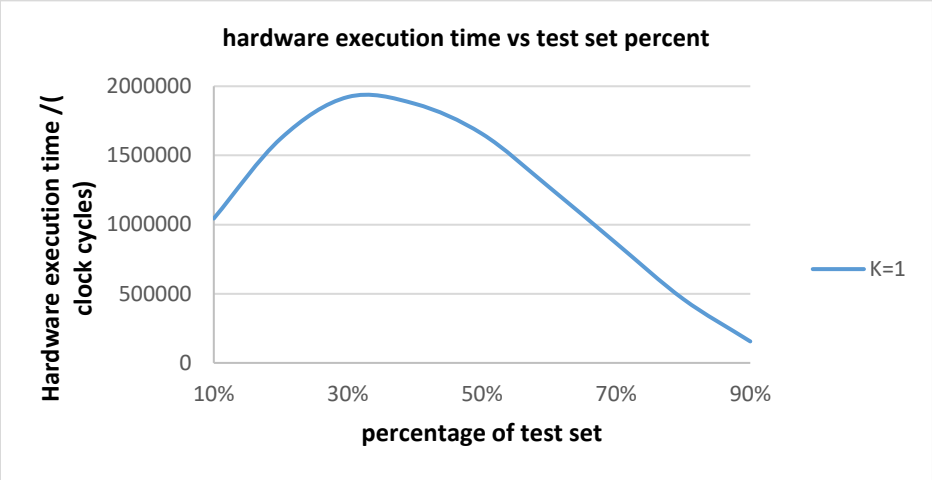

Figure 18: Hardware execution time vs percentage of test set when K=1

Figure 18 shows that the time taken for the hardware to finish the classification task increases when the test set percentage of the dataset increases from 10% to 30 %, the execution time is the highest at 30 %. The time starts decreasing when the test set percentage of the dataset increases from 40% to 90%, and the execution time is the lowest at 90%. The execution time for the hardware depends mainly on the distance measurements step and sorting step, because these two steps take the longest time to process. The process time for distance measurements step and sorting step depend on the number of the attributes in the test set and the training set. As observed from figure 1 when the test set is 30% and the training set is 70% of, the data set the processing time of the distance measurements step and sorting step is the highest.

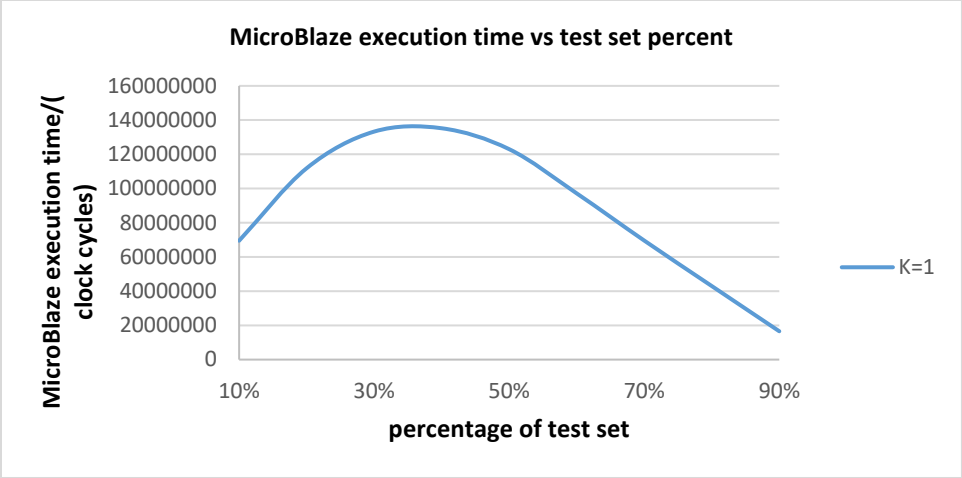

Figure 19: MicroBlaze execution time vs percentage of test set when K=1

Figure 19 shows that the execution time of the software follows the same behavior of the execution time of the hardware in figure 5-1.

The figure 20 shows the speedup of the hardware compared to the software running on MicroBlaze for different values of K. As illustrated the speedup changes slightly for different values of K, therefor overlap happens when we plot the speedup for K from 1 to 9. The highest speedup occurs when the test set is 90% of the dataset and K is equal to 1.

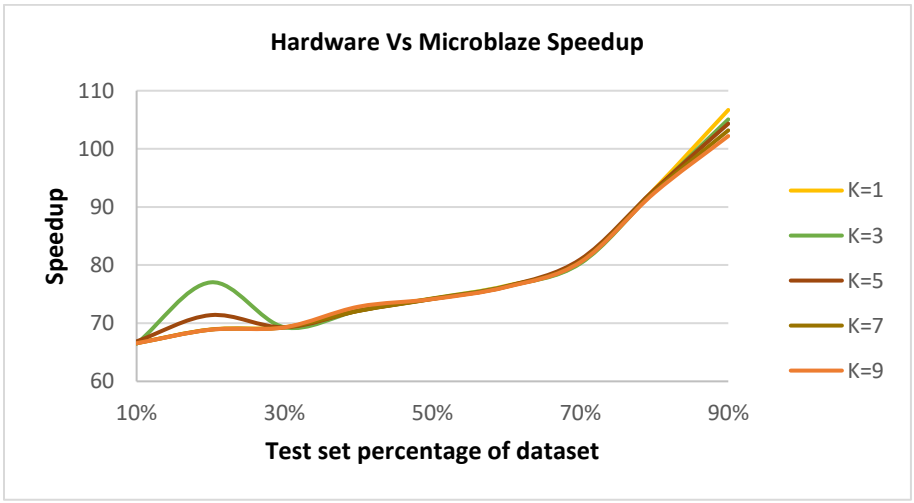

Figure 20: Hardware Vs MicroBlaze Speedup

The hardware is also compared to the CPU for different percentage of test set and for different values of K. in this case, the speedup varies from 0.6 to 1.4. Since the CPU is running at 2.4 GHz and our hardware design is single processing element running at 100 MHz, then the speedup result is considered as a good result.

5.1.2 Heart benchmark

The results of the heart benchmark are represented in the Table 5-1 to the Table 5-5.

Table 5-6: Execution time, speedup and accuracy for heart benchmark when K =1

K =1	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	473691091	6106224	77.58	7	74.07%
	20%	3	758511232	9753784	77.77	13	75.93%
	30%	46	891642220	11319572	78.77	23	71.60%
	40%	31	903579078	11097532	81.42	28	74.07%
	50%	31	821818180	9913944	82.89	34	74.81%
	60%	31	675247385	7712332	87.55	38	76.54%
	70%	15	479285548	5167132	92.76	43	77.25%
	80%	15	283827257	2759384	102.86	51	76.39%
	90%	15	114747003	905686	126.69	66	72.84%

Table 5-7: Execution time, speedup and accuracy for heart benchmark when K =3

K =3	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	473675220	6194620	76.47	5	81.48%
	20%	27	758511232	9754652	77.76	13	75.93%
	30%	31	891769244	11320832	78.77	20	75.31%
	40%	31	903660467	11099254	81.42	26	75.93%
	50%	31	821943180	9836944	83.56	32	76.29%

	60%	31	675368586	7708160	87.62	30	81.48%
	70%	15	479457281	5170198	92.73	34	82.01%
	80%	15	284016147	2762898	102.79	41	81.02%
	90%	15	114928865	909550	126.36	54	77.78%

Table 5-8: Execution time, speedup and accuracy for heart benchmark when K =5

K =5	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	473731853	6195026	76.47	6	77.77%
	20%	15	758551974	9755520	77.76	11	79.63%
	30%	31	891764246	11322162	78.76	15	81.48%
	40%	31	903741232	11100990	81.41	23	78.70%
	50%	31	821986331	9918242	82.88	29	78.52%
	60%	31	675491183	7710778	87.60	31	80.86%
	70%	16	479570925	5173236	92.70	34	82.01%
	80%	15	284151945	2766370	102.72	41	81.02%
	90%	8	115114994	913456	126.02	57	76.54%

Table 5-9: Execution time, speedup and accuracy for heart benchmark when K =7

K =7	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	Speedup	Classification error	accuracy
	10%	15	473715848	6195460	76.46	6	77.77%
	20%	31	758503111	9756402	77.74	14	74.07%
	30%	31	891891531	11323450	78.76	16	80.25%
	40%	203	903822001	11102726	81.41	22	79.63%
	50%	31	822141140	9841284	83.54	28	79.26%
	60%	31	675612636	7713382	87.59	29	82.09%
	70%	31	479711238	5176288	92.67	34	82.01%
	80%	15	284340589	2769814	102.66	37	82.87%
	90%	9	115292050	917320	125.68	61	74.89%

Table 5-10: Execution time, speedup and accuracy for heart benchmark when K =9

K =9	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	15	473772244	6195894	76.47	5	81.48%
	20%	31	758543939	9757242	77.74	11	79.63%
	30%	31	891885861	11324752	78.76	14	82.72%
	40%	31	904014065	11104462	81.41	20	81.48%
	50%	31	822189080	9843440	83.53	24	82.22%
	60%	31	675735655	7715972	87.58	26	83.95%
	70%	16	479882944	5179298	92.65	31	83.59%
	80%	16	284502702	2773286	102.59	38	82.41%
	90%	1	115477155	921198	125.36	70	71.19%

From Tables 5-6 to 5-10, we can see that the best classification accuracy, which is 83.59% occurs when the percentage of the test set is 70% of the dataset with K equal to 9.

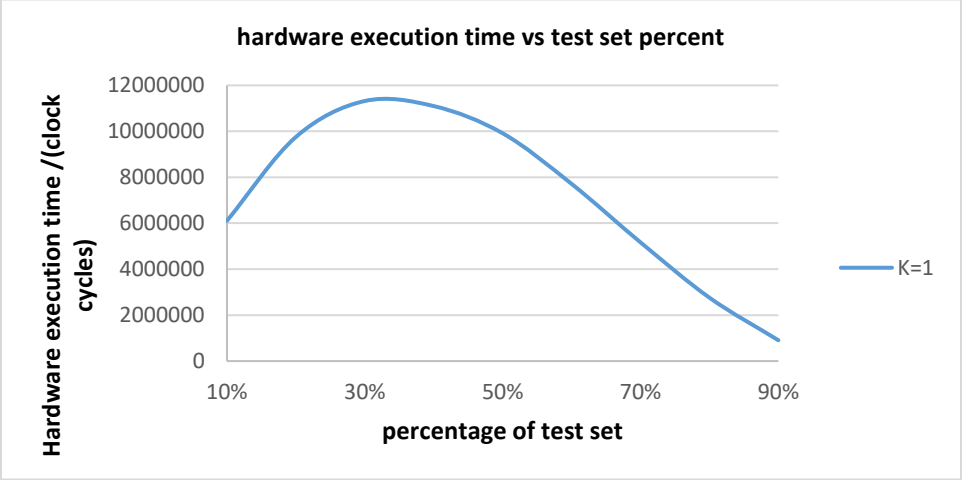

Figure 21: Hardware execution time vs percentage of test set when K=1

Figure 21 shows that the hardware execution time behavior for heart benchmark dataset is the same behavior shown in figure 5-1 for Iris benchmark dataset.

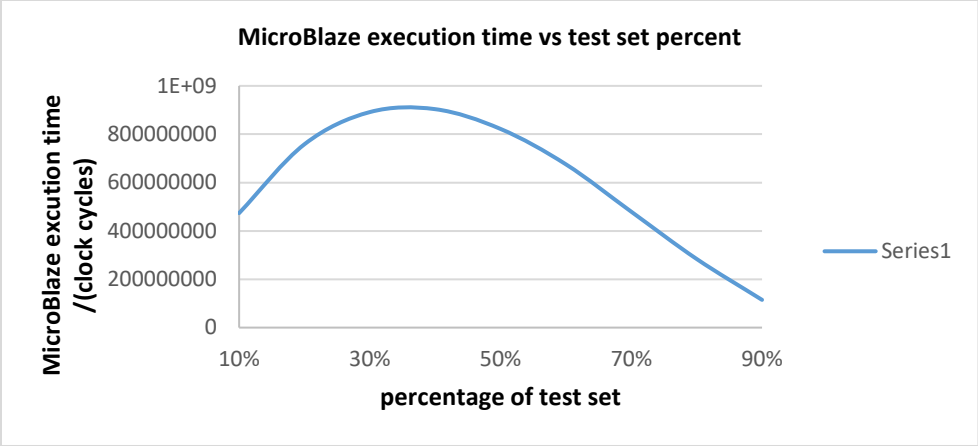

Figure 22: MicroBlaze speed vs test set percent K=1

Figure 22 shows that the MicroBlaze execution time follow the same behavior of the hardware execution time.

The figure 23 shows that the speedup of the hardware compared to the software running on Microblaze for different values of K. As illustrated the speedup changes slightly for different values of K, therefor overlap happens when we plot the speedup for K from 1 to 9. The highest speedup occurs when the test set is 90% of the dataset and K is equal to 1.

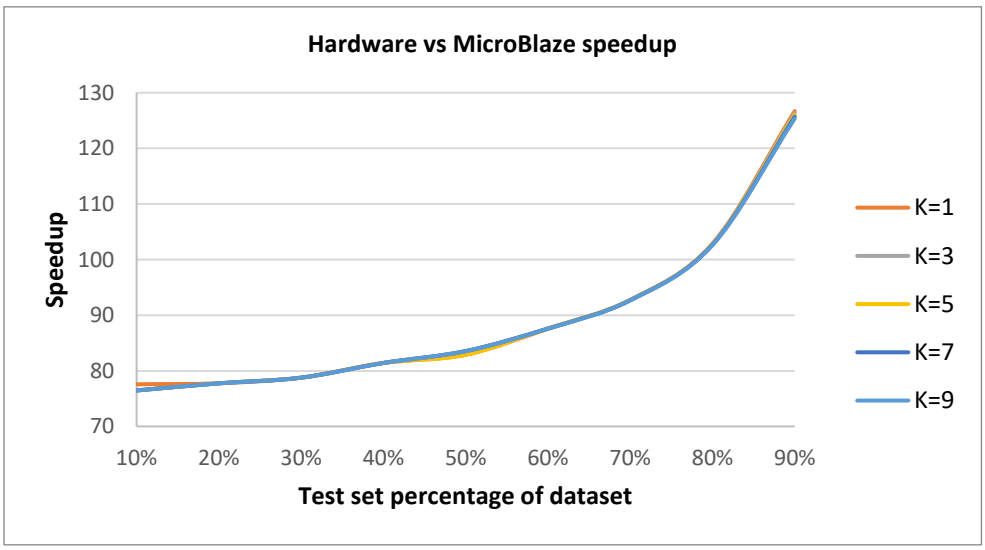

Figure 23: Hardware Vs MicroBlaze Speedup

The hardware is also compared to the CPU for different percentage of test set and for different values of K. in this case, the speedup varies from 0.2 to 1.6. Since the CPU is running at 2.4 GHz and our hardware design is single processing element running at 100 MHz, then the speedup result is considered as a good result.

5.1.3 User knowledge modeling benchmark

The results of the user knowledge modeling benchmark dataset are represented in the Table 5-11 to the Table 5-15.

Table 5-11: Execution time, speedup and accuracy for user knowledge modeling benchmark when K =1

K =1	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	46	1344016776	20053500	67.02	6	85.00%
	20%	62	2092109202	31614910	66.17	12	85.19%
	30%	63	2447638619	36389302	67.26	23	80.99%
	40%	62	2487210498	35825550	69.43	28	82.61%
	50%	62	2216859031	31467770	70.45	37	81.68%
	60%	78	1761630966	24425280	72.12	58	76.03%
	70%	46	1233334864	16312658	75.61	79	71.99%
	80%	31	670422416	8430938	79.52	109	66.15%
	90%	15	232986331	2504010	93.05	97	73.28%

Table 5-12: Execution time, speedup and accuracy for user knowledge modeling benchmark when K =3

K =3	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	46	1344053042	20057532	67.01	3	92.50%
	20%	78	2092178384	31617052	66.17	9	88.89%
	30%	78	2447751421	36392466	67.26	20	83.47%
	40%	78	2487510636	35829764	69.43	24	85.09%
	50%	62	2217045845	31594498	70.17	36	82.18%
	60%	47	1749028742	24431664	71.59	48	80.17%
	70%	31	1233598425	16320162	75.59	70	75.18%
	80%	31	666552562	8439688	78.98	103	68.01%
	90%	15	233328127	2514076	92.81	110	69.69%

Table 5-13: Execution time, speedup and accuracy for user knowledge modeling benchmark when K =5

K =5	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	31	1344200133	20058512	67.01	6	85.00%
	20%	78	2092265669	31619054	66.17	12	85.19%
	30%	62	2448003907	36395532	67.26	25	79.34%
	40%	63	2487663173	35833782	69.42	32	80.12%
	50%	78	2217227236	31356666	70.71	38	81.19%
	60%	62	1762072448	24437670	72.10	42	82.64%
	70%	31	1225539534	16327260	75.06	69	75.53%
	80%	31	671012668	8447738	79.43	91	71.74%
	90%	1	233640223	2523190	92.59	135	62.81%

Table 5-14: Execution time, speedup and accuracy for user knowledge modeling benchmark when K =7

K =7	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware	speedup	Classification error	accuracy
	10%	31	1344235080	20059520	67.01	6	85.00%
	20%	62	2092437208	31621056	66.17	11	86.42%
	30%	62	2447968031	36398486	67.25	29	76.03%
	40%	78	2487800103	35837758	69.42	36	77.64%
	50%	78	2201457200	31361734	70.19	37	81.68%
	60%	46	1762294092	24443732	72.09	39	83.88%
	70%	46	1234107114	16334260	75.55	68	75.89%
	80%	31	667012912	8455606	78.88	107	66.77%
	90%	1	232804195	2532192	91.94	135	62.81%

Table 5-15: Execution time, speedup and accuracy for user knowledge modeling benchmark when K =9

K =9	Test set percent of the dataset	CPU execution time/(ms)	MicroBlaze execution time /(clock cycles)	Hardware execution time /(clock cycles)	speedup	Classification error	accuracy
	10%	46	1344162014	20060486	67.01	9	77.50%
	20%	78	2092398862	31623030	66.17	13	83.95%
	30%	62	2448235374	36401440	67.26	27	77.69%
	40%	78	2487795921	35841790	69.41	40	75.16%
	50%	62	2217859044	31366676	70.71	38	81.19%
	60%	46	1762514229	24566078	71.75	41	83.06%
	70%	31	1234363937	16341232	75.54	74	73.76%
	80%	25	667306044	8463586	78.84	108	66.46%
	90%	15	234314628	2541012	92.21	150	58.68%

From Tables 5-11 to 5-15, it is observed that the best classification accuracy, which is 92.5%, occurs when the percentage of the test set is 10% of the dataset with K equal to 3.

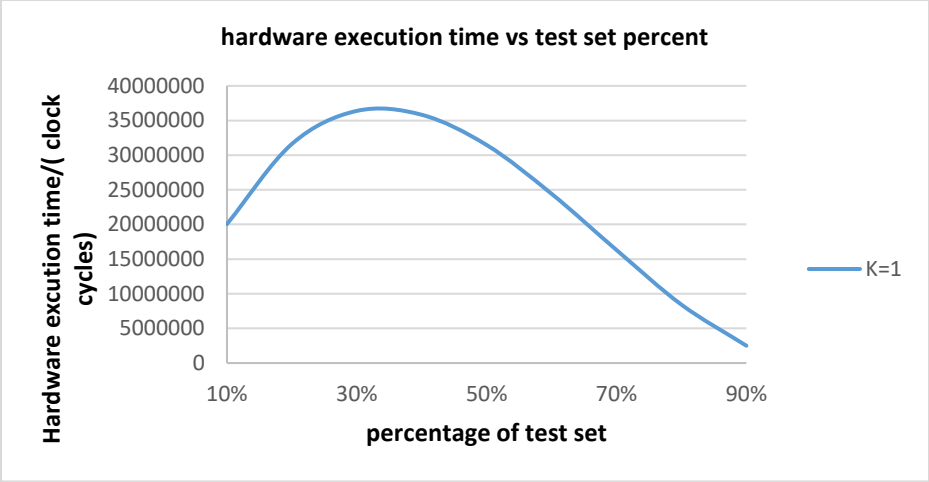

Figure 24: Hardware execution time vs percentage of test set when K=1

Figure 24 shows that the hardware execution time for the user knowledge modeling benchmark dataset follow the same behavior for the hardware execution time of the iris and heart benchmark datasets in figure 18 and figure 21. The hardware execution time keeps the same behavior regardless of the benchmark size and type. Figure 25 shows that the execution time of the software running on MicroBlaze follow the same behavior of the hardware execution time.

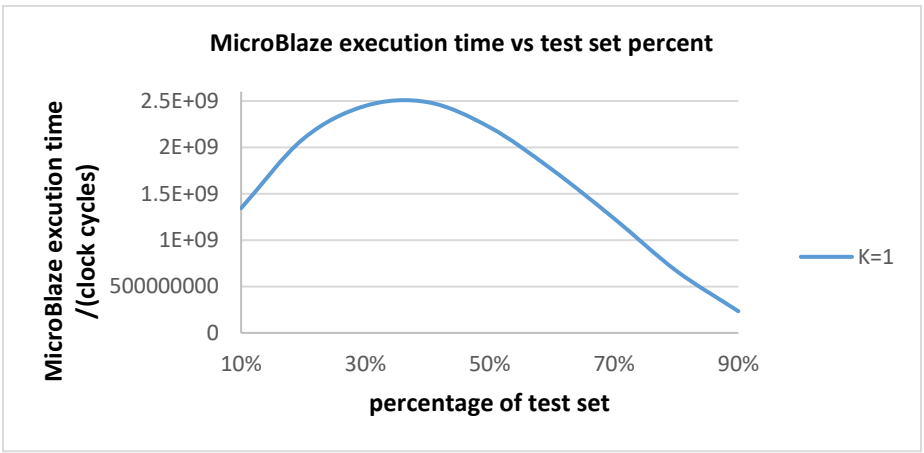

Figure 25: MicroBlaze speed vs test set percent K=1

The figure 26 shows that the speedup of the hardware compared to the software running on Microblaze for different values of K. As illustrated the speedup changes slightly for different values of K, therefor overlap happens when we plot the speedup for K from 1 to 9. The highest speedup occurs when the test set is 90% of the dataset and K is equal to 1.

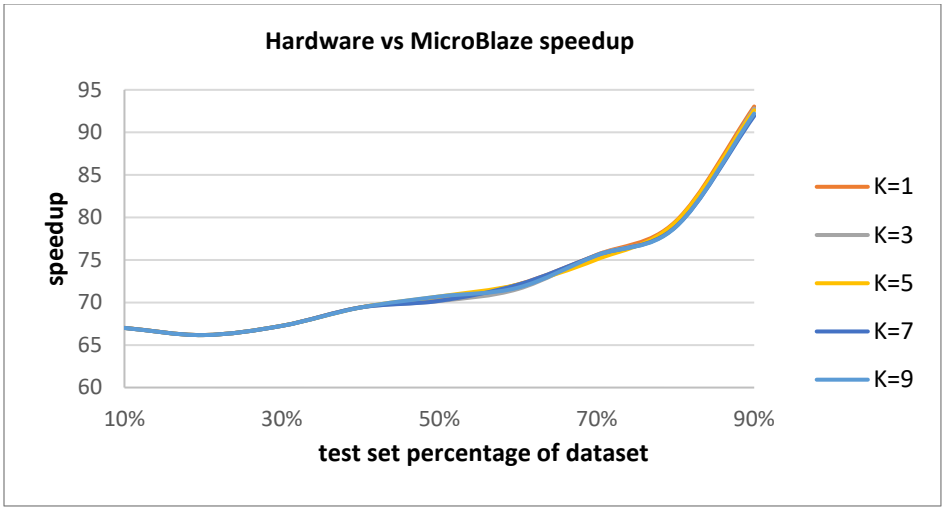

Figure 26: Hardware Vs MicroBlaze Speedup

The hardware is also compared to the CPU for different percentage of test set and for different values of K. in this case, the speedup varies from 0.1 to 0.5. Since the CPU is running at 2.4 GHz and our hardware design is single processing element running at 100 MHz, then the speedup result is considered as a good result.

Our tests show that our hardware design is significantly faster than our equivalent software design running on MicroBlaze regardless of the dataset type and size. Our designs are generic and parametrized, and the tests shows that the hardware and the software design processing times follow the same behavior regardless of the dataset type, size and number of K which means our hardware and software design are stable. Our design achieves 100% classification accuracy for the iris dataset, which means our design does not have any issues

achieving the highest classification accuracy. The effect of the dataset type on classification accuracy is discussed in the next section.

5.2 Dataset effect on classification accuracy

From the experiments we notice that the classification accuracy is better if each class is well separated from the other classes, as shown in three cases in figure 27.

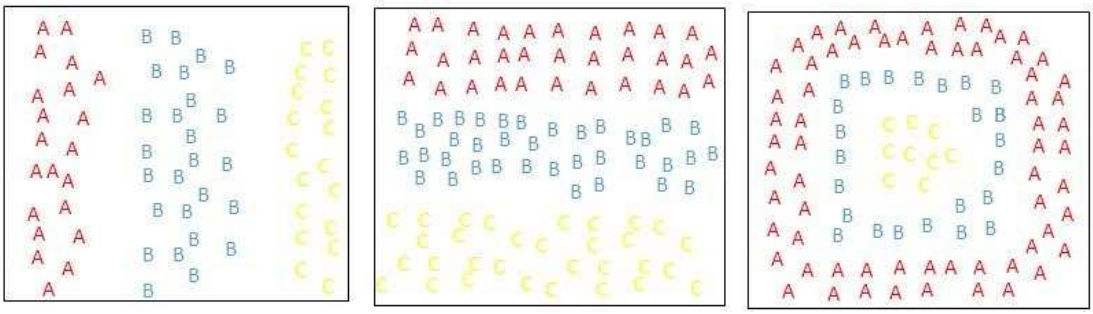

Figure 27: Dataset types

When the classes are separated as in figure 27, increasing the K value increases the accuracy until it reaches a saturation point, where the accuracy is not affected by K increasing. In a scenario, where classes are mixed or poorly separated the classification accuracy is low, and it is difficult to predict the outcome of the increasing of K value.

5.2 Hardware space statics

For our design XPS was is used to generate the resource utilization summary for the system level design running on Virtex-6 FPGA. The utilization summary is shown in table 5-16.

Table 5-16: Utilization summary

Number of occupied Slices	Number of DSP Slices	Number of BRAM
7,788	47	293

The number of occupied slices are 20% of the total number of slices of Virtex-6 FPGA. The number of DSP slices used are 6% of the available slices. MicroBlaze uses 68 of the total BRAM number uses by the system, which is 293.Since existing designs are executed on different platform on different FPGA; it is difficult to make a comparison in form of resource utilization. Furthermore, most of the existing hardware design does not propose a system-level design architecture or propose a system level-design architecture different from ours, which also make it difficult to compare.

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this chapter, we summarize our contributions and discuss our intention for future work.

6.1 Conclusion

In this research work, we investigated the K-NN as a supervised machine learning algorithm. Our investigation on the existing research works on hardware architectures for K-NN algorithm, illustrated several issues with the existing designs. In this thesis work, we proposed an efficient FPGA-based hardware architecture for K-NN algorithm. We also designed and developed a software architecture on an embedded processor to evaluate our hardware design. Our experimental results demonstrated that our hardware design was significantly faster (executed 127 times faster) compared to our equivalent software design executed on the MicroBlaze processor on the same FPGA. Our proposed hardware achieved this significant speedup due to many hardware optimization techniques that we implemented and incorporated in our design, including: burst transfers and pre-fetching techniques to reduce the memory access latency; fully pipelined designs; customized and optimized circuits; efficient system-level architectures. We used single precision floating-point to design our hardware to achieve high accuracy. From the results and analysis, our design achieved 100% classification accuracy.

Our main contributions, in this research work, are as follows:

1) We introduced three designs for K-NN classification algorithm: software design on a personal computer (PC); accelerated software design on MicroBlaze processor; and most importantly accelerated hardware design on the FPGA.

2) We introduced an efficient system-level design for the K-NN algorithm, in order to process the data effectively and efficiently, and considering the real-time constraints.

3) We implemented and incorporated many hardware optimization techniques, including: burst transfer and pre-fetching in our design; and unique pipelined hardware architecture design for the K-NN algorithm; customized and optimized internal circuits; in order to enhance the speed-performance, area efficiency and accuracy of our proposed architecture.

4) Our hardware architecture is generic and can be implemented on any development platform. Furthermore, our hardware design is parameterized and scalable and can be used to process varying datasets with varying sizes, and for different applications, without changing the internal architecture. We evaluated our hardware design using three different benchmark datasets from various fields.

6.2 Future work

We achieved our major goal of this research work, by designing an efficient hardware architecture that is significantly faster (127 time faster) than the equivalent software design. However, while conducting this research work, we observed several key points that we could address in the future to improve our design:

- Investigate and design a parallel hardware architecture to increase the processing speed.
- Implement the Z-score standardization in hardware, then provide a mechanism for dynamic reconfiguration of FPGA to give the user the ability to choose between normalizing the data either using Min-Max normalization or Z-score standardization.

- We used Euclidean distance metric to measure the distance between continues variable. However, this method does not work with categorical variables. Hence, we want to implement a special function, illustrated in [3], in hardware to deal with categorical variables, and provide a mechanism for dynamic reconfiguration of FPGA so our K-NN classifier design can work with continues and categorical variables.

BIBLIOGRAPHY

1. Jackson J., "Data Mining; A conceptual overview" Communications of the Association for Information Systems: V. 8, 19, 2002.

2. Kotsiantis S., "Supervised Machine Learning: A Review of Classification Techniques", in Proceeding of conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies PP: 3-24, July 2007.

3. Larose D., "Discovering Knowledge in Data: An Introduction to Data Mining", John Wiley & Sons, Inc. 2005.

4. Alpaydın E., "Introduction to Machine Learning", Massachusetts Institute of Technology, Second Edition, 2010.

5. Bharathi A. and Deepankumar E., "Survey on Classification Techniques in Data Mining", International Journal on Recent and Innovation Trends in Computing and Communication, V. 2, July 2014.

6. Cruz J. and Wishart D., "Applications of Machine Learning in Cancer Prediction and Prognosis", Cancer Informatics Journal, 2006.

7. Han J., Micheline K. and Jian P., "Data Mining: Concepts and Techniques", Third Edition. Morgan Kaufmann Publishers, 2012.

8. Amanpreet S., Narina T. and Aakanksha S., "A Review of Supervised Machine Learning Algorithms", in Proceeding of International Conference on Computing for Sustainable Global Development (INDIACom), March 2016.

9. Battula B. and Prasad R., "An Overview of Recent Machine Learning Strategies in Data Mining", in Proceeding of International Journal of Advanced Computer Science and Applications, Vol. 4, No.3, 2013.

10. Pu Y., Peng J., Huang L. and Chen J., "an efficient K-NN algorithm implemented on FPGA based heterogeneous computing system using OpenCL", in Proceeding of IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines May 2015.

11. Jabbar M., Deekshatulu B. and Chandra, P., "Classification of Heart Disease Using K- Nearest Neighbor and Genetic Algorithm", in Proceeding of International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA), 2013.

12. Li H., Yeh Y. and Hwang W., "Fast K-NN Classification Based on Softcore CPU And Reconfigurable Hardware", Intelligent Automation & Soft Computing Journal, Vol.17, No.4, pp. 431-446, 2011.

13. Murty M. and Devi V., "Pattern Recognition: An Algorithmic Approach", 2011.

14. Kantardzic M., "Data mining: Concepts, Models, Methods, and Algorithms", IEEE Press, John Wiley & Sons, Inc., Second edition, 2011.

15. Guo G., Wang H., Bell D., Bi Y. and Greer K., "K-NN Model-Based Approach in Classification." In: Meersman R., Tari Z. and Schmidt D.C. (eds) On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. OTM 2003. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg, 2003.

16. Bali R. and Sarkar D., "R Machine Learning by Example", 2016.

17. Hussain H., Benkrid K., Hong C. and Seker H., "an adaptive FPGA implementation of multi-core K-nearest neighbor ensemble classifier using dynamic partial reconfiguration", in Proceeding of 22nd International Conference on Field Programmable Logic and Applications (FPL), August 2012.

18. Khan A., Baharudin B., Lee L. and khan K., "A Review of Machine Learning Algorithms for Text-Documents Classification", journal of advances in information technology, Vol.1, No.1, February 2010.

19. Neelamegam S. and Ramaraj E., "Classification algorithm in Data mining: An Overview", International Journal of P2P Network Trends and Technology (IJPTT) – Volume 4 Issue 8, September 2013.

20. Wu W., Kumar V., Quinlan J., Ghosh J., Yang Q., Motoda H., McLachlan G., Ng A., Liu B., Yu P., Zhou Z., Steinbach M., Hand D. and Steinberg D., "Top 10 algorithms in data mining", Springer-Verlag London Limited, 2007.

21. Satyanarayana N., Ramalingaswamy C. and Ramadevi Y., "Survey of Classification Techniques in Data Mining", International Journal of Innovative Science, Engineering & Technology (IJISET), Vol. 1 Issue 9, November 2014.

22. Islam M., Wu Q., Ahmadi M. and Sid-Ahmed M., "Investigating the Performance of Naive- Bayes Classifiers and K- Nearest Neighbor Classifiers", in Proceeding of International Conference on Convergence Information Technology, November 2007.

23. Stamoulis I. and Manolakos E., "Parallel Architectures for the K-NN Classifier – Design of Soft IP Cores and FPGA Implementations", ACM Transactions on

Embedded Computing Systems journal, Vol. 13, No. 2, Article 22, September 2013.

24. Minghua S., Bermak A. and Belhouari S., "A Real-time Architecture of SOC Selective Gas Sensor Array Using K-NN Based on the Dynamic Slope and the Steady State Response", in Proceeding of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications, July 2004.

25. Cannisi D. and Yuan B., "Design Space Exploration for K-Nearest Neighbors Classification Using Stochastic Computing", in Proceeding of IEEE International Workshop on Signal Processing Systems (SiPS), October 2016.

26. Shu H., Yu R., Jiang W. and Yang W., "Efficient Implementation of k-Nearest Neighbor Classifier Using Vote Count Circuit", IEEE Transactions on Circuits and Systems II: Express Briefs, 2014.

27. http://www.expertsystem.com/machine-learning-definition/

28. http://whatis.techtarget.com/definition/machine-learning

29. http://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/

30. Hussain H., Benkrid K. and Seker H., "An Adaptive Implementation of a Dynamically Reconfigurable K-Nearest Neighbour Classifier On FPGA", in proceeding of NASA/ESA Conference on Adaptive Hardware and Systems (AHS), June 2012.

31. Manolakos E. and Stamoulis I., "IP-cores design for the K-NN classifier", in proceeding of International Symposium on Circuits and Systems (ISCAS) Conference, May 2010.

32. Garcia V., Debreuve E. and Barlaud M., "Fast k nearest neighbor search using GPU", in proceeding of Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Computer Society Conference, June 2008.

33. https://www.coursera.org/learn/machine-learning#ratings

34. https://www.simplilearn.com/what-is-machine-learning-and-why-it-matters-article

35. Smola A. and Vishwanathan S., "Introduction to Machine Learning", Cambridge University Press 2008.

36. http://www.ironpaper.com/webintel/articles/machine-learning-market-statistics/

37. Fu Y., "Data mining: Tasks, Techniques, and Application", university of Missouri -Rolla.

38. https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

39. https://www.forbes.com/sites/bernardmarr/2016/09/30/what-are-the-top-10-use-cases-for-machine-learning-and-ai/#57083d5994c9

40. http://www.businesswire.com/news/home/20170725005968/en/Field-programmable-Gate-Array-FPGA-Market---Forecasts

41. http://fpgacenter.com/fpga/fpga_or_cpu.php

42. Karahoca A., Karahoca D. and Şanver M., "Survey of Data Mining and Applications", Intech 2012.

43. Xilinx Inc., ML605 Hardware User Guide, UG534 ed., Xilinx Inc, 2012.

44. Xilinx Inc., "Virtex-6 Family Overview," 20 Aug 2015. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

45. Hauck S. and Dehlon A., "Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing", Morgan Kaufmann Publishers, 2008.

46. Xilinx Inc., "MicroBlaze Processor Reference Guide," 2 Oct 2013. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/mb_ref_guide.pdf

47. Sundaramoorthy N., "Simplifying Embedded Hardware and Software Development with Targeted Reference Designs," 2009.

48. Perera D. and Li K., "FPGA-Based Reconfigurable Hardware for Compute Intensive Data Mining," in International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2011.

49. Xilinx Inc., "LogiCORE IP Floating-Point Operator v5.0," 1 Mar 2011. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

50. Xilinx Inc., "Embedded System Tools Reference Guide (EDK 11.3.1)," 16 Sep 2009. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf

51. Xilinx Inc., "Virtex-6 FPGA Memory Interface Solutions," 1 March 2011. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp469-microblaze-for-cost-sensitive-apps.pdf

52. Xilinx Inc., "LogiCORE IP AXI Interconnect (v1.06.a)," 18 Dec 2012. [Online]. Available:http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_06_a/ds768_axi_interconnect.pdf

53. Xilinx Inc., "LogiCORE IP AXI UART Lite," 25 Jul 2012. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_uartlite/v1_02_a/axi_uartlite_ds741.pdf

54. https://archive.ics.uci.edu/ml/datasets/Iris

55. https://archive.ics.uci.edu/ml/datasets/Statlog+(Heart)

56. https://archive.ics.uci.edu/ml/datasets/User+Knowledge+Modeling

57. http://www.codeblocks.org

58. Xilinx Inc., ISE Design Suite 14: Release Notes, Installation, and Licensing, vol. UG631, Xilinx Inc., 2013.

59. Li Z., Jin J., Zhou X. and Feng Z., "K-Nearest Neighbor Algorithm Implementation on FPGA Using High Level Synthesis", in proceeding of 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), October 2016.