FPGA BASED KNN SEARCH ENGINE

by

Dakota James Koelling

APPROVED BY SUPERVISORY COMMITTEE:

Dinesh K. Bhatia, PhD, Chair

Poras T. Balsara, PhD

William Swartz Jr, PhD

To my Parents,

For their constant support and encouragement.

FPGA BASED KNN SEARCH ENGINE

by

DAKOTA JAMES KOELLING, BS

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTERS OF SCIENCE IN

COMPUTER ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

May 2017

ACKNOWLEDGMENTS

I would like to thank Dr. Dinesh Bhatia for the amazing opportunity to work under him in the IDEA lab. I have learned so much from this project. The guidance you have given me is greatly appreciated.

I would also like to thank Megan Kelley and Lauren Percy for helping me get my ideas down on paper and making them coherent.

I would also like to thank Dave Baker for helping me think through my designs and providing feedback.

Last but not least, I would like to thank my family for all the encouragement and support they have given me.

January 2017

PREFACE

This thesis was produced in accordance with guidelines which permit the inclusion as part of the thesis the text of an original paper or papers submitted for publication. The thesis must still conform to all other requirements explained in the "Guide for the Preparation of Master's Theses and Doctoral Dissertations at The University of Texas at Dallas." It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this thesis to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts which provide logical bridges between different manuscripts are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the student's contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the thesis attest to the accuracy of this statement.

FPGA BASED KNN SEARCH ENGINE

Publication No. _____

Dakota James Koelling, MS
The University of Texas at Dallas, 2017

Supervising Professor: Dinesh K. Bhatia, PhD

Today's processing applications, such as machine learning and text mining, are becoming more complex and process large amounts of data. Computational demands imposed by such applications are not easily met by traditional computing platforms. There is increasing shift towards GPGPUs (General Purpose Graphics Processing Units) and FPGAs (Field Programmable Gate Arrays) for executing high performance applications. Both of these devices can be used to process more efficiently large datasets by working in parallel. The complex applications are made of many smaller algorithms such as K Nearest Neighbor (KNN), that enable themselves to be performed in parallel and are easy to speed up. Finding an architecture that can utilize this more efficient technology to accelerate the algorithms would improve the run time of the original application. Intels next generation of Xeon server grade processors will include FPGAs and allow for faster large data parallelization. The ZedBoard, having the Xilinx Zynq XC7Z020 architecture that provides approximately 85,000 logic cells for the accelerator as well as the ARM Cortex-A9 for control, can be thought of as a small scale version of the Xeon server. Accelerators implemented on the ZedBoard utilize the AXI interfaces that allow for compatibility and interchangeability between many

other accelerators. This thesis discusses in detail how to design and implement the KNN algorithm on the Xilinx Zynq architecture in order to improve its run time. Also discussed are performance comparisons between the hardware-accelerated implementation and its software implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Many of today's applications require substantial amounts of computation power to handle massive amounts of data. Currently we use large data centers with many servers to process this data. These servers are made up of many generalized processors that can perform well enough but would benefit from improvements. The continuous growth and development of big data applications creates a demand for this data to be processed at ever increasing speeds. One way to increase the speed of the overall end application is to improve the speed of individual building blocks that are repeatedly used. GPGPUs (General Purpose Graphics Processing Units) are essentially a large number of small processor cores packed together that accelerate the execution by processing in parallel. While they are able to successfully accelerate execution, they are very power hungry which is undesirable in a large server farm. A common method of decreasing the run times of the smaller blocks is through the use of hardware accelerators. Instead of using software on a general architecture, the accelerators utilize specialized hardware architectures. However, one consequence of using hardware is that it needs to be rebuilt from scratch for every application. Field programmable gate arrays (FPGA) are a balanced middle ground that allow for the achievement of significant increases in speed over a general architecture without needing to rebuild new silicon. FPGAs allow for the parallelization of processing of large data sets without being constrained to a single use. Specific applications such as large graphs [5] and image processing [26] are already being accelerated using FPGA technology. The next generation of Intel's server grade Xeon processors will include an FPGA core that will be able to assist computation of big data in the cloud using a library of FPGA based hardware accelerators [18].

There are algorithms that take large sets of data and attempt to classify the points into different groups for better and more specific processing. These algorithms are looking for patterns that can be used by higher level applications to find a useful meaning. K Nearest Neighbor (KNN) is one such big data algorithm that aims to add new points to the previously classified dataset. KNN is used in many applications from banking to botany [29]. One of the most common ways this algorithm is used is in text mining [15]. KNN is also used in agriculture to predict weather patterns, evaluate forest inventories using satellite imagery, climate forecasting, and estimating soil water parameters [15]. Its applications can be extended to finance for stock market forecasting, currency exchange rate, bank bankruptcies, understanding and managing risk and loan, trading futures, credit rating, and money laundering analysis [15]. The following shows a hypothetical example of how KNN could be used in the real world.

1.1 KNN Banking Example

Consider working at a bank and approving a loan. The normal process to get approved for a loan consists of collecting personal and financial information such as a person's name, age, address, salary, and account balance and sending it off to be analyzed. This takes a considerable amount of time to complete. The bank keeps a database of this same data for all its customers. After some analysis, specific attributes like age and salary may be found to be reliable predictors of which customers will receive approval for a loan. Figure 1.1 shows example data for a bank with loan acceptance vs. age vs. salary. A new customer comes to the bank looking for a loan. To speed up the acceptance process, the bank would ask for the age and salary of the customer. The new customer is the green dot on the figure. The KNN search algorithm is run on the data to find the K most demographically similar customers to the new customer or K closest classified points to the reference point. The K customers

returned would then be checked for loan acceptance. The new customer would get the loan if most of the similar customers received the loan.

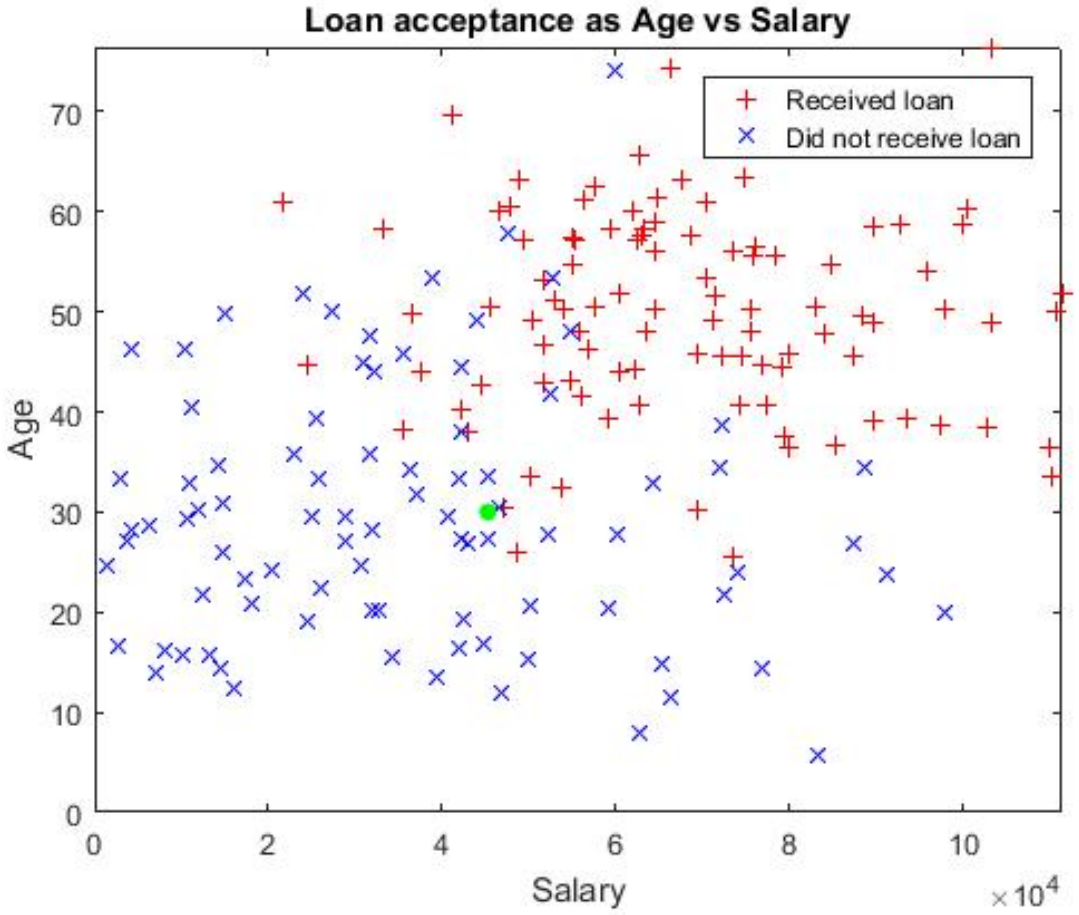

Figure 1.1. Banking Example Data

1.2 K Nearest Neighbor Algorithm

The KNN algorithm classifies a D dimensional unclassified reference point R where $R = [R_1, R_2, ..., R_D]$ by comparing it to a dataset of N existing classified data points S. Each point $S_i \in S$ is made up of multiple dimensions $S_i = [S_{i1}, S_{i2}, ..., S_{iD}]$, each corresponding to an important data parameter. In the above example, D is 2 for the age and salary dimensions and N is 100 for the 100 customers in the set. Age and salary for the new customer R is

placed in R_1 and R_2 respectively. Similarly, age and salary for customer S_i is placed in S_{i1} and S_{i2} respectively. The algorithm calculates the Euclidian distance, shown in equation 1.1, between the new unclassified point and every other pre-classified point in the dataset.

$$Dist(S_i, R) = \sqrt{\sum_{j=1}^{D}(S_{ij} - R_j)^2}, 1 < i < N \tag{1.1}$$

The N points in S are then sorted based on their distance from the reference point, and the K points with the smallest distance from reference point are returned. Of the returned points, the reference point is classified as the classification that appears the most.

Listing 1.1. KNN Search Pseudocode

```
Search(S,R,k)
// S: training data
// R: unknown sample
// k: size of return set
for i=1 to N do
        Compute distance Dist(S_i,R)
end for
Compute set I containing indices for
                the k smallest distances Dist(S_i,R).
return I
```

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 describes related literature for KNN and reconfigurable hardware accelerators.

Chapter 3 goes into great detail about the KNN accelerator hardware. First, an overall discussion of the architecture. Next, a detailed description of the process of iterative design used to get to the final design. Finally, a discussion of the final implementation as a whole.

Chapter 4 describes the test setup and procedure used to ensure a working and consistent design.

Chapter 5 discusses the resulting speedup of the accelerator compared to a desktop CPU.

Finally, chapter 6 concludes the thesis and discusses future work.

CHAPTER 2

RELATED LITURATURE

Many researchers are looking at ways to make Big Data Analytics more efficient using FPGAs. They are looking at scalable heterogeneous dataflow architectures to move the data more efficiently [12], as well as identifying more energy efficient solutions [20]. Advanced techniques such as Dynamic Partial Reconfiguration have been used in the implementation of accelerators when used with applications like Bioinformatics [14]. Dynamic Partial Reconfiguration is when the FPGA has the ability to change its configuration on the fly as it computes depending on its current requirements. KNN is such a wide spread algorithm that researchers have been analyzing the advantages and disadvantages of similar algorithms [9]. Bhatia specifically has looked at algorithms based on KNN such as Weighted KNN [4], Model based KNN [10], Condensed NN [7], Reduced NN [11], Generalized NN [33], k-d tree [28], ball tree [21], Principal Axis Tree [32], Nearest Feature Line [25], Tunable NN [34], and Orthogonal Search Tree [6]. Researchers are also looking at higher level languages such as OpenCL in order to implement their KNN algorithm [22]. Implementations for the KNN Classifier have been designed based on the Wavelet Transform and Partial Distance Search [35]. Accelerators have been designed to speed up the KNN based thinning algorithm for reducing a set of vectors in a multi-dimensional feature space to a smaller set of representative vectors [27].

2.1 Nearest Neighbor Techniques

Bhatia discusses the advantages and disadvantages of many different nearest neighbor algorithms [6]. The algorithms are split into two types; structure less and structure based. The

structure less type reduces the number of calculations that need to be performed by reducing the dataset in a way that results in lower memory requirements. Usually the dataset is thinned or reduced in an attempt to lessen redundant data points and thus redundant calculations. The structure based type uses structures like trees to organize the dataset resulting in faster access to the data and faster execution time. An advantage to K Nearest Neighbor (KNN), the original algorithm that is discussed in chapter 1, includes efficiency when dataset is large. KNN is simple, easy to learn, and targets large data samples. Disadvantages are that it is biased by value of K, contains computational complexity, and memory limitations. The following are examples of structure based and structure less algorithms.

2.1.1 Structure Less

Weighted K Nearest Neighbor (WKNN) also targets large data samples, but modifies the original KNN algorithm described in this paper by assigning weights to neighbors as per distance calculated when deciding the reference point's classification [4]. The advantages are that it has a reduced chance of a tie when sample size is small, it uses all dataset points and not just K, and makes the algorithm a global one. On the other hand, the disadvantages are that its computation complexity increases when calculating weights and the algorithm runs slow. Condensed Nearest Neighbor (CNN) eliminates samples from the dataset which show similarity in a constructive fashion by building up the set without adding extra information [7]. The advantages consist of reduced dataset size, improved query time and memory requirements, and reduced recognition rate. The disadvantages consist of being order dependent, unlikely to pick up points on boundaries, and computation complexity. CNN targets datasets where memory requirement is a main concern. Reduced Nearest Neighbor (RNN) targets large datasets and removes samples which do not affect the dataset results in a destructive fashion by starting from a full set and removing samples [11]. The advantages consist of reduced size of dataset and elimination of templates, improved query

time and memory requirements, and reduced recognition rate. Disadvantages to RNN are computation complexity, and time consumption. Model based K Nearest Neighbor (MKNN) is a modification where the model is constructed from the data and then classified with new data using the model [10]. The advantages include a higher classification accuracy, value of K is selected automatically, and higher efficiency due to reduced number of data points. A disadvantage is that it does not consider marginal data outside the region. MKNN targets applications in dynamic web mining for large repositories. Rank Nearest Neighbor (KRNN) assigns ranks to datasets for each category [24]. The advantages include better performance when there are too many variations between features, robustness when it is based on rank, and being less computationally complex when compared to KNN. However, the multivariate version has the disadvantage of depending on the distribution of the data. KRNN targets applications with class distributions of the Gaussian nature. Modified K Nearest Neighbor (MKNN) uses the weight and validity of a data point to classify nearest neighbor [13]. Advantages are the ability to partially overcome low accuracy of WKNN, with stability and robustness, while the disadvantages include being computationally complex. MKNN targets methods facing outlets. Pseudo/Generalized Nearest Neighbor (GNN) does not just focus on only the nearest neighbor but also utilizes information of n-1 neighbors [33]. The advantage of this is that it uses n-1 classes which consider the whole dataset, while the disadvantage is being computationally complex and inaccurate for small datasets. Due to this inaccuracy, GNN targets large datasets. Clustered K Nearest Neighbor targets text classification and utilizes the formation of clusters to select the nearest neighbor [36]. The advantages include ability to overcome defects of an unevenly distributed dataset and being robust in nature, while disadvantages include a bias by value of K for clustering, and the difficulty of selection of the threshold parameter before running the algorithm.

2.1.2 Structure Based

Structure based algorithms partition the dataset into various regions and provide a structure to quickly and easily reduce the search to a subset of regions. In the case of trees, the important information is usually stored in the leaves. Ball Tree K Nearest Neighbor (KNSI) uses a ball tree structure to improve upon the original KNN speed by partitioning the search space into ball shaped regions [21]. Advantages include the ability to be tuned well to the structure of the represented data, dealing well with high dimensional entities, and being easy to implement. Disadvantages include costly insertion algorithms caused by needing to insert the new sample into all intersecting balls and a degradation in the algorithm as the distance increases because smaller balls are preferred. KNSI targets geometric learning tasks like robots, vision, speech, and graphics. K-d Tree Nearest Neighbor (KdNN) divides the dataset into half plane with the advantages of producing a perfectly balanced tree, and being fast and simple [28] The disadvantages include requiring more computation and intensive search, and blindly slicing points in half which may result in a missing data structure. KdNN targets organization of multidimensional points. Nearest Feature Line Neighbor (NFL) calculates by taking advantage of multiple templates per class [25]. The advantages of this method include improved classification accuracy, high efficacy for small size sets, and utilizes information ignored in nearest neighbor i.e. templates per class. The disadvantages include its computational complexity, the fact that the computation fails when the prototype in NFL is far away from the query point, and describing feature points by straight line is a hard task. NFL targets face recognition problems. Local Nearest Neighbor focuses on the creation of a nearest neighbor prototype of the query point [30]. The advantage of this method is that it covers the limitations of NFL, while the disadvantage is the number of computations. Facial recognition is also the target of this algorithm. Tunable Nearest Neighbor (TNN) targets discrimination problems and modifies the original with the addition of a tunable metric [34]. The advantage of this is that it can be effective for small

datasets, with the disadvantage of large numbers of computations. Center based Nearest Neighbor (CNN) calculates a center line to aid its calculations [23]. This method targets pattern recognition and has the same advantages and disadvantages as TNN. Principal Axis Tree Nearest Neighbor has the advantage of good performance, fast search, and targeting pattern recognition while having the disadvantage of computational time [32]. Orthogonal Search Tree Nearest Neighbor also targets pattern recognition and uses orthogonal trees and has the advantage of less computational time and being effective for large data sets, while the disadvantage is having a larger query time [19].

2.2 KNN implementation using OpenCL

Pu et al. discusses the use of OpenCL to implement the KNN algorithm on an FPGA based heterogeneous computing system [22]. OpenCL is a framework for parallel programming on heterogeneous platforms. It is based on C and was modified to include mechanics required for parallel computing such as vectorized data and synchronization points among other things. A typical architecture to run OpenCL consists of the host and many different computation devices along with a memory structure split into global, local, and individual.

In the paper, the host is a desktop CPU with the computation devices being FPGA cores. The devices are grouped together to do similar tasks. The global memory is accessible by all devices and the host. The local memory is accessible by only devices in the group, and individual memory is only accessible by the specific device. OpenCL implementations are split into two parts: platform and device level. The device level handles the data computations with kernels and runs on the devices, while the platform level handles queuing the data into the devices and runs on the host. Using OpenCL, the KNN algorithm is implemented by configuring the devices to calculate the distances from the dataset with the reference set loaded into local memory. After all distances are calculated, the devices are reconfigured to do a bubble sort to find the K nearest neighbors.

Due to the nature of OpenCL, the program can be deployed to any multi-core device such as a CPU or GPU. To compare with the FPGA implementation, Pu does just that using the CPU as a reference. The FPGA was able to achieve a speedup of 148 over the CPU while the GPU was able to achieve a speedup of 410 over the CPU. The power is also measured on the devices and used to calculate the energy efficiency ratio (EER). The EER of the GPU and FPGA were 268 and 804, respectively. This shows that while the GPU is the fastest, the FPGA is the more efficient.

2.3 KNN FPGA implementation based on Wavelet Transform and Partial Distance Search

Yeh et al. discusses a FPGA implementation that leverages three methods in order to speed up the KNN algorithm in addition to the parallelization achievable on the FPGA [35]. These three methods include subspace search, bitplane reduction, and multiple-coefficient partial distance accumulation. The subspace search utilizes the Haar wavelet of the discrete wavelet transform to make the dataset discrete. The bitplane reduction removes the LSBs of the datasets to minimally sacrifice accuracy in exchange for a significantly smaller memory requirement. The last piece is the multiple-coefficient partial distance accumulation that calculates parts of the distance and then compares that to the current K nearest neighbors during execution. If the partial distance is larger than the largest distance in the minimum buffer, then that data point is removed and no further calculation is performed on it. If the partial distance is less than the largest distance, then the calculation of the distance continues until it is complete. If it is still within the K nearest, then the new point is added to the list. This method reduces the amount of non-optimal distances calculated without compromising on accuracy. The paper details a pipelined implementation utilizing these methods. The architecture is then packed into the NIOS softcore CPU and loaded on a FPGA for testing. Testing was done with the original KNN algorithm in software, the

subspace PDS in software, and finally the FPGA accelerator. The execution time in us of the KNN, PDS, and FPGA were 16079.2, 262.5, and 137.77, respectively. These results show a significant speedup over the original KNN algorithm.

2.4 KNN Thinning HW Accelerator

Schumacher et al. discusses a FPGA implementation for a hardware accelerator for KNN thinning [27]. The KNN thinning algorithm reduces the dataset size down to a more managable one and is used when the computation time and memory requirements need to be reduced. It starts by calculating the distance between all points in the set to create the distance matrix. Each row of the matrix is then sorted in ascending order. Once all rows have been sorted, the algorithm starts looking for unique minimums in each column starting with column 3 (1 and 2 are not unique). When a unique minimum is found, that row is removed from the distance matrix. The row removal process is repeated until the distance matrix is at the desired size.

With this algorithm in mind, the implementation was split into four parts: the vector table, the distance calculator, the distance sorter, and the row selector. The vector table is used to store the vectors and their indexes and has the ability to remove entries when a row is removed. The distance calculator calculates the square distance between each of the vectors, one entry at a time and progresses down the row first. Once a full row is complete, the calculator moves on to the next row and the distance sorter for the first row is started by the controller. The distance sorter executes a bubble sort of the entries in the distance row RAM. This continues until all distances have been calculated and rows sorted. After that, the row selector is activated by the controller and searches for and selects a unique minimum. If a unique minimum cannot be found then the first entry that has a minimum is selected. The selected entry is then sent to the vector table for removal. The accelerator is synthesized into an FPGA and served data from a host processor. The SPEA2 algorithm is

used for a testing scenario, and achieves a speedup of 6.6 times when run with the accelerator compared to a software implementation.

CHAPTER 3

ARCHITECTURAL DESIGN OF KNN ACCELERATOR

The algorithm is easily parallelized because the distance of each point from the reference point is independent of all other distances. A pipeline can be used to calculate these distances by splitting it into stages. The idea behind the core pipeline architecture is similar to one used for online learning [1]. While that design was specific to an application, the design in this paper focuses on a more generic use case. This study focuses on an algorithm that calculates the distance from a single point to every other point to find the K nearest neighbors.

3.1 Accelerator IP Block

All data is written sequentially into the start of the pipeline by the host, beginning with the reference point first. Due to the reference point being required for every point calculation, there is a circular buffer to store every dimension of the point. The buffer is the size of the reference point, D 32-bits locations, one for each dimension. As the reference point is received, its dimensions are saved into a circular buffer, R_j in location j of the buffer, with a counter tracking the location from 1 to D then resetting to 1. Immediately after the reference point is finished, the dataset begins to be received and the counter resets back to 1. As the dataset is shifted in, the counter is used to keep the corresponding reference feature, R_j, in sync with the dataset point feature, S_{ij}, in the first stage of the pipe. The difference, $S_{ij} - R_j$, between the two corresponding features, S_{ij} and R_j, are calculated in the second stage. This difference is squared to get $(S_{ij} - R_j)^2$ in the third stage. In the fourth stage, the square difference $(S_{ij} - R_j)^2$ is added to an accumulator until all D features in the data point are added in. Once all features are added, the square distance $Dist(S_i, R)^2$ is sent on

to the next stage and the accumulator is cleared. This pipeline takes $D + 2$ clock cycles to calculate the square distance. In this architecture, taking the square root to calculate the distance is skipped due to the complexity of the function and the fact that the calculation only requires the relative order, not the distance itself. The fifth stage inserts the square distance and its index into a sorted buffer of size K. The index is the order in which the points are received starting at 0. The sort is done with K comparators C comparing entry i in the buffer to the new square distance to be inserted. If C_i shows entry i to be greater than or equal to the new entry and C_{i-1} shows entry $i-1$ to be less than the new entry, then the new entry is shifted into location i and entries i to $D-1$ are shifted up one location in a shift register fashion. The simplicity of comparison and parallel shifting allow this sort to be done in a single clock cycle and is shown in figure 3.1. P number of pipes can be implemented in parallel to speed up execution. Each of the P pipes will result in a sorted list of size K that needs to be merged into a single K sized list. After all pipes have been cleared, a finite state machine (FSM) starts merging the lists into a single list by enabling each list's output and multiplexing that into the input of a phase two sorting block that works the same as above by inserting the new element into the sorted buffer. When the list's output is enabled, it sends out the entries one at a time starting at the smallest square distance in that list. This merging of lists into a single list uses $K * P + 1$ number of clock cycles to complete. After the final list has been created, the list's output is enabled and the index of the data points are shifted out to the host beginning with the nearest neighbor. A high level block diagram of the final accelerator architecture is shown in figure 3.2, while a more detailed diagram showing bit widths and register stages, is shown in figure 3.3. Once the basic architecture was designed, it was implemented in Verilog to prepare for the IP block generation. To see exact changes to the design as it matured through multiple iterations, everything was checked into a Github repository [17].

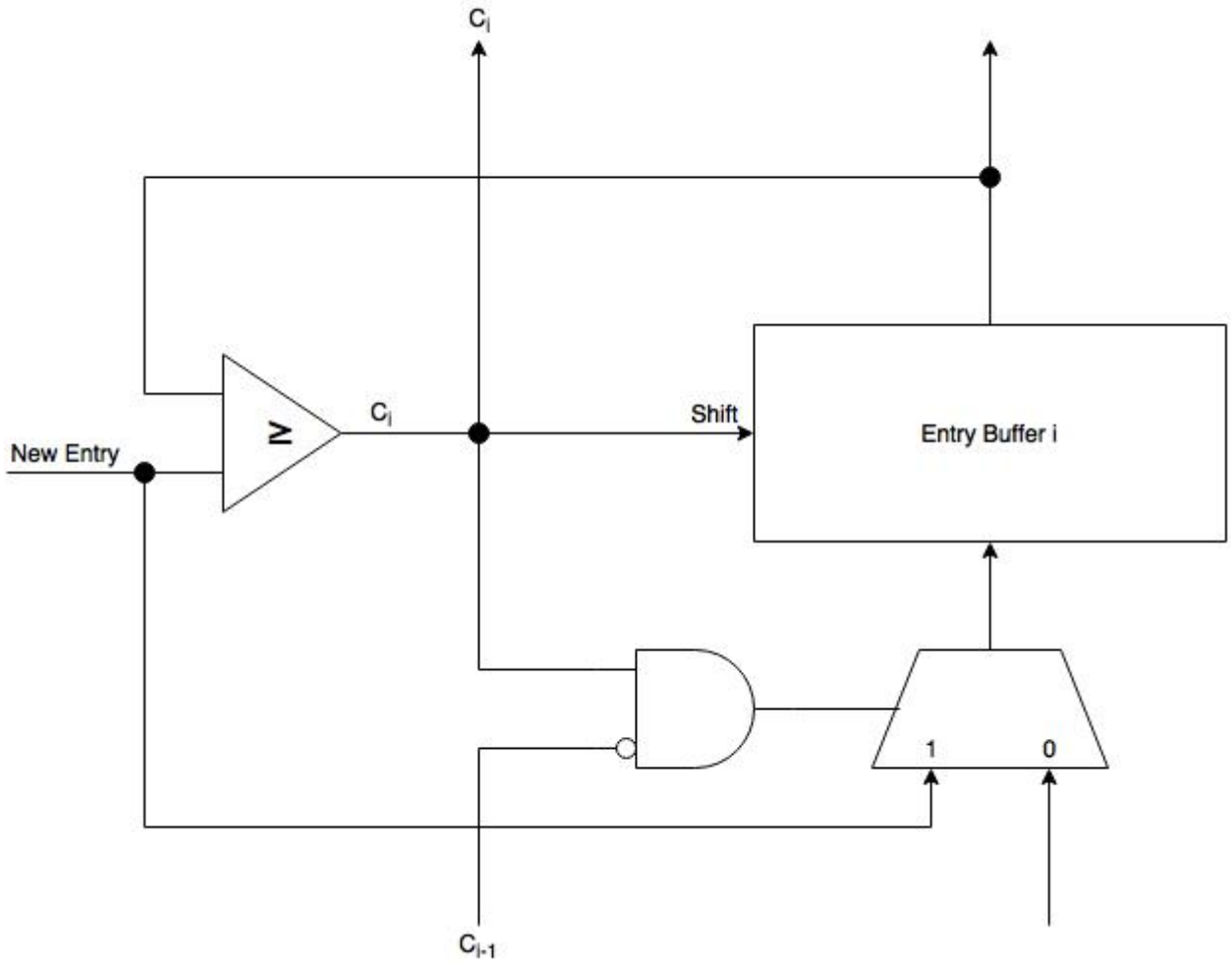

Figure 3.1. Sorting Architecture

3.2 AXI Interface

To connect the programmable logic and the processing system in the Zynq, Vivado generates AXI4, AXI4 Stream, and AXI4-Lite master/slave interfaces according to the specification [2]. While AXI Stream only has the write data channel, AXI and AXI-Lite consist of the following 5 channels: write address, write data, write response, read address, and read data. To write data, the master sends the write address on the write address channel followed by the data on the write data channel. Once the slave has processed the write, it will respond on the write response channel. To read data, the master sends the address it would like to

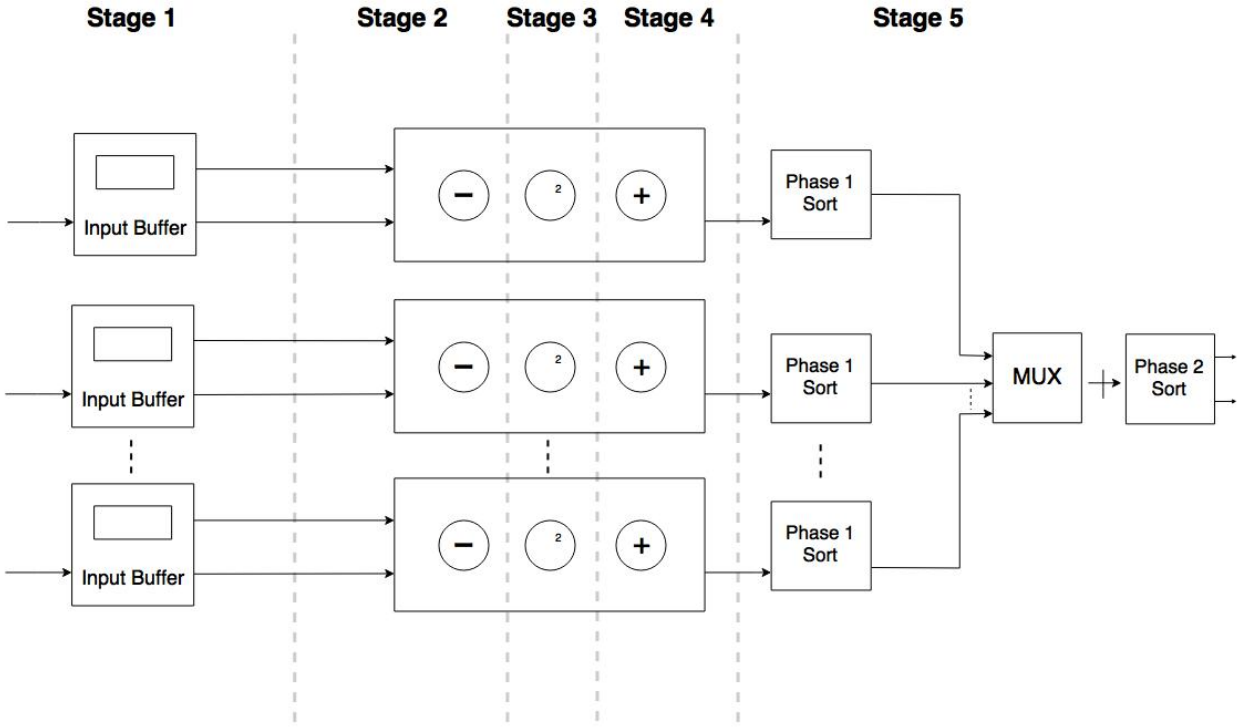

Figure 3.2. High level accelerator block diagram

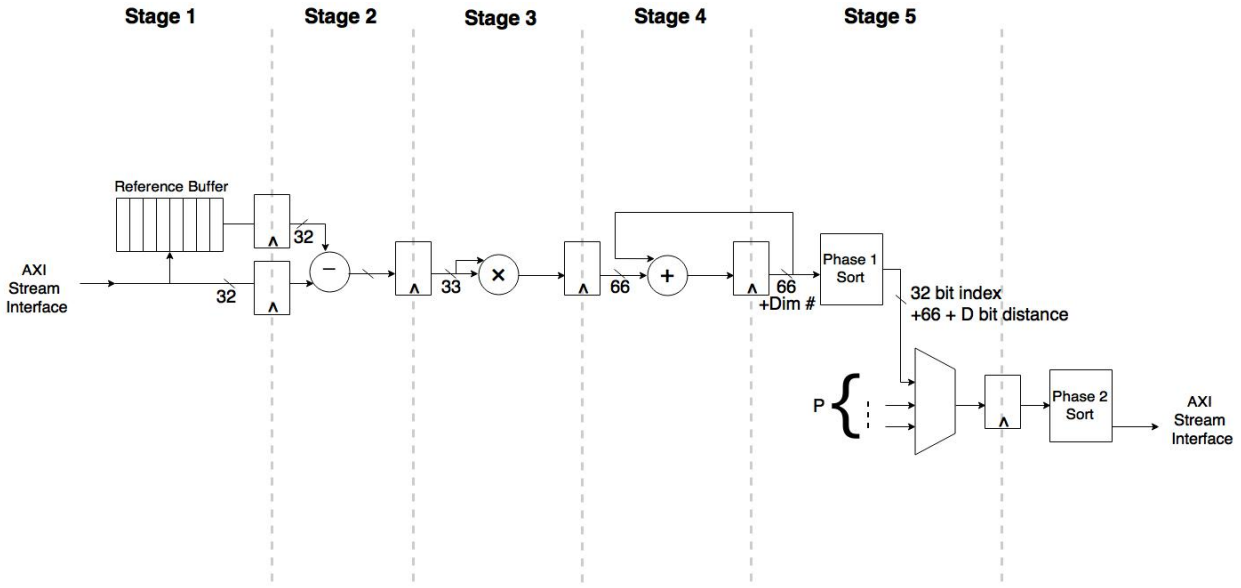

Figure 3.3. Detailed accelerator block diagram

read on the read address channel. The slave reads the address and sends it back on the read data channel. Each channel has 3 main signals, data_ready, data_valid, and data where data is whatever information that channel is sending (address, data, response). The master or controlling device raises the data_valid signal when it has placed data on the data bus. The slave or controlled device raises the data_ready signal when it is ready to receive data. When both signals are asserted, the slave accepts the data on the data bus and the master updates it. This interaction of signals is described as a handshake because both sides are required to send information. AXI and AXI-Lite have address channels which means they are memory mapped and require a location to read or write data. The difference is that AXI-Lite can only send one transaction or burst at a time, but AXI can send multiple. The AXI Stream has no address channel and only sends data as a stream that has no limit to its length.

3.3 Zynq Hardware System

The system was created with the help of this FPGAdeveloper tutorial [16] that provides a step by step guide to create a custom IP in Xilinx's Vivado tool for a Zynq platform. The IP block implemented in the programmable logic (FPGA) communicates with the processing system (ARM core) using AXI interfaces. An AXI-Lite slave interface provides the ARM core access to control registers in the accelerator. This accelerator has 2 32-bit registers for control purposes and 2 more 32-bit registers for timing during testing. The first 2 registers are the *control* register that has reset and done bits and the *Second Start Index* register that holds the index of *N/2* as shown in table 3.1 and 3.2. The last 2 registers provide access to a 64-bit counter running at 100MHz used to measure the execution time of the accelerator with accuracy down to 10 ns. In order to move data between DDR memory that is controlled by the processing system and the accelerator in the programmable logic, shown in figure 3.4, multiple AXI Stream interfaces were added to the design. Figure 3.5 shows the maximum DDR memory bandwidth of the Zynq architecture to be 2 64-bit transactions

Table 3.1. Control Register at offset 0x0

31 Reserved 2	1 (DONE)	0 (RESET)

Table 3.2. Second Start Index Register at offset 0x4

31 Second_Start_Index 0

with the 2 arrows connecting the Memory Interfaces to the Programmable Logic to Memory Interconnect then to the Programmable Logic. To maximize data transfers to the accelerator, 2 64-bit stream slave interfaces were used as well as 1 32-bit stream master interface for the transfers in the opposite direction. Each 64-bit transaction is split by the accelerator into 2 32-bit numbers and served by 2 pipes, resulting in a total $P = 4$. When the DMA transfers the K nearest neighbors back to DDR memory, only the indexes of the data points are sent and not the calculated distances because the relative order of the points is what is important, not the absolute distance.

The accelerator was designed and implemented as an IP block in Vivado that can be used by anyone using the IP block diagram. The designer adds the Zynq IP block to use the ARM core, the accelerator IP block to speedup the algorithm, and up to two Direct Memory Access (DMA) cores to saturate the memory bandwidth, and Vivado wires it all together as shown in figure 3.6. A DMA core is a system that is used to offload movement of data from the processor. The DMA is an IP block provided by Xilinx and accesses the memory using an AXI interface then streams it to the accelerator using an AXI Stream interface. The DMA runtime configuration is set by the ARM core that is running C++ code using the AXI-Lite protocol. The accelerator requires the configuration of K and D which can be done by customizing the IP block in Vivado. The DMA implementation configurations are not identical as DMA0 has both data transfer directions (read/write channel) enabled while

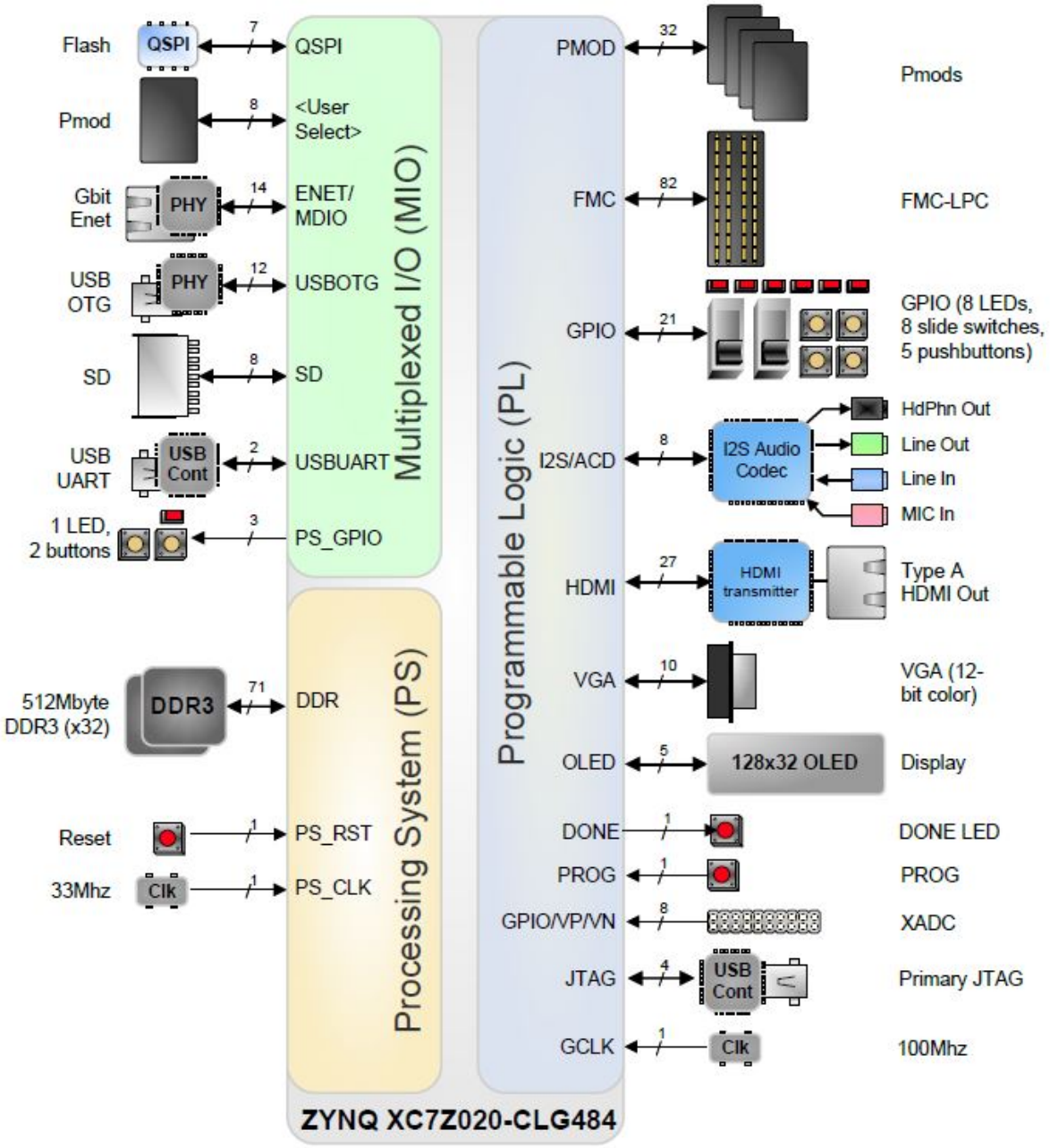

Figure 3.4. ZedBoard Block Diagram from ZedBoard User Guide [3]

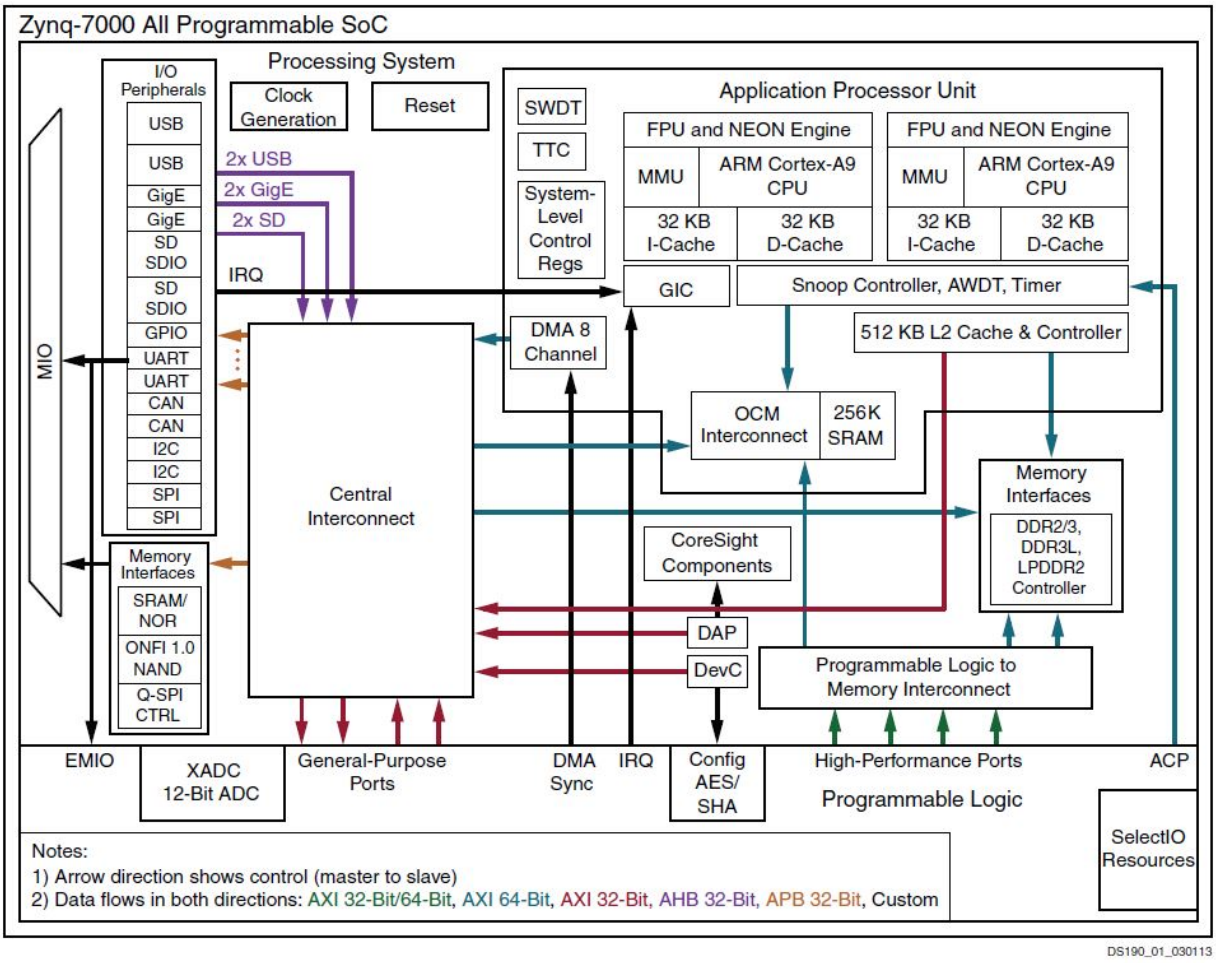

Figure 3.5. Zynq-7000 All Programmable SoC Overview from ds190 [31]

DMA1 only enables data transfer to the accelerator (read channel). Vivado customization settings are shown in table 3.3. All read channel data widths are set to 64-bits, while max burst size on all channels is set to 256 because that is the maximum. Vivado does full implementation of the design into the FPGA platform.

3.4 Zynq Software System

Once the hardware side is created, the SDK is used to load the data set into the DDR and control the accelerator and DMAs. The accelerator requires that the ARM core write the

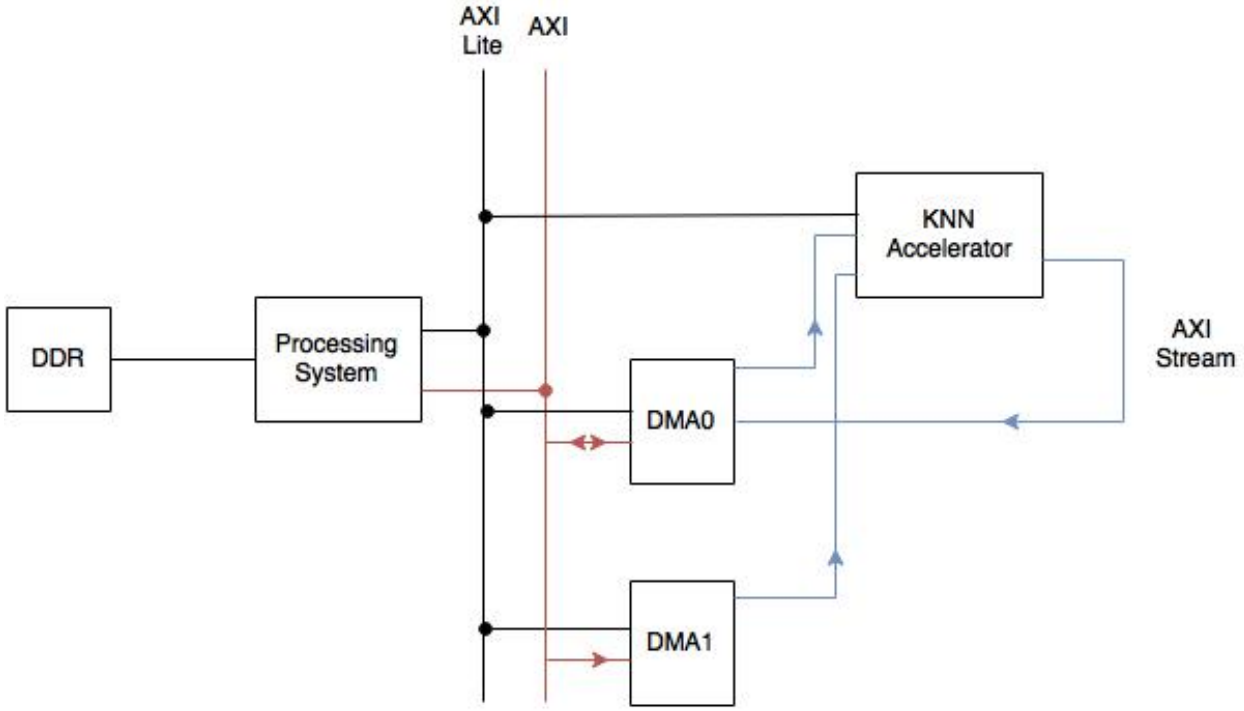

Figure 3.6. Implementation block diagram

Table 3.3. DMA hardware configurations

Setting	DMA0	DMA1
Enable Scatter Gather Engine	unchecked	unchecked
Enable Micro DMA	unchecked	unchecked
Width of Buffer Length Register	23 bits	23 bits
Address Width	32 bits	32 bits
Enable Read Channel	checked	checked
Memory Map Data Width	64	64
Stream Data Width	64	64
Max Burst Size	256	256
Allow Unaligned Transfers	unchecked	unchecked
Enable Write Channel	checked	unchecked
Memory Map Data Width	Auto	N/A
Stream Data Width	Auto	N/A
Max Burst Size	256	N/A
Allow Unaligned Transfers	unchecked	N/A

index of the data point that the second DMA is starting at into the *Second Start Index* register and then reset by writing a 1 and then a 0 to the reset bit in the *control* register. The ARM core, using the AXI-Lite interface for control, resets the DMA by setting the reset bit in the *control* register, then sets the runstop bit in the *control* register, the starting address of data to transfer in the *destination address* register, and the length in bytes of data to transmit in the *buffer length* register. The DMA then sends the data in a single uninterrupted burst to the accelerator for processing. The AXI DMA provided by Xilinx has a limitation of the transfer length in bytes of 23 bits. The DMAs are configured so that DMA0 will transfer the first half of the data and DMA1 will transfer the second half of the data to the accelerator. The ARM core polls both DMAs' *status* registers waiting for the idle bits to set, signaling the DMAs have finished sending the data. The ARM core sets the DONE bit in the accelerator's *control* register. DMA0 then waits until the end of the computation and transfers the index of the K nearest neighbors back into memory. The ARM core learns of the completion of this transfer by polling the idle bit in the *status* register of DMA0. Once the DMAs are finished, the ARM core reads the indexes out of memory to the screen.

The advantages to this architecture is the ability for it to achieve significant speedup over a CPU due to its use of a parallel and pipelined design. A key limitation of this design however is the complexity of the sorting block when dealing with large values of K. On the ZedBoard there was only enough resources for K to be roughly 25.

CHAPTER 4

TEST SETUP

The board used for testing is the ZedBoard made by Digilent. This board has a Xilinx Z020 Zynq 7000 chip on it that has a Dual ARM Cortex-A9 MPCore with an FPGA and 512 MB of DDR3 RAM. The dataset is stored in DDR, the C++ program shown in appendix B.1 that configures the accelerator is run on the ARM core, and the accelerator is placed in the FPGA. The CPU being used for comparison is an Intel i7 3770K running at 4GHz with 16GB of DDR3 RAM and an SSD. The setup is shown in figure 4.1.

The data is available in a data file. When testing on the CPU, the data is read from the data file into an array in memory before the timer is started. The execution time includes the algorithm being run on the array in memory and writing the output to a final array. After the timer is stopped, the array is read out to the screen. Time is measured from a computer time function to maintain accuracy to the millisecond. When testing the FPGA setup, all subsystems are reset and the data is loaded into memory before the timer is started. The execution time includes the DMAs transferring the data to the accelerator, the computation by the accelerator, and the DMAs transferring the results back to memory. After the timer is stopped, the array is again read out to the screen. Pre-processing Python scripts were used to format the dataset in various ways primarily for the FPGA but also for the CPU. In both setups, when using fixed-point notation, a Python script from appendix C.2 was used to convert the fixed-point decimal number to binary fixed-point; moving the binary point from 16-bits to the right and 16-bits to the left of the binary point format to 32-bits the left of the binary point format, then converting the binary back to decimal. This operation changes all points in an equal amount, making it irrelevant because this study maintains

Figure 4.1. Test Setup with CPU (back) and FPGA (front)

Table 4.1. Single channel dataset file format

-1	R_0	R_1	...	R_D
0	$S_{0,0}$	$S_{0,1}$	...	$S_{0,D}$
1	$S_{1,0}$	$S_{1,1}$	...	$S_{1,D}$
...	...	...	...	...
n	$S_{n,0}$	$S_{n,1}$	...	$S_{n,D}$

interest in relative position. This created an integer form of the file to be used by the CPU C++ program as well as the pre-processors for the FPGA. At this point, the file is assumed to be in the format of each point in the dataset being on a different line, and the line starts with the point ID/index followed by the features of the point all separated by a tab. The reference point is denoted by having the index -1. An example is shown in table 4.1.

The first FPGA script in appendix C.1 converts the integer data set format to an interleaved data format that is compatible with the multiple concurrent channels of the accelerator. Interleaving the data allows the DMA to receive 2 corresponding 32-bit features from 2 separate data points with each 64-bit sequential read, as each DMA is serving 2 independent pipes. The format is similar to the input format with two data points interleaved into a single line. The reference point is duplicated within its own line with index -1, and an all zero line is inserted at the end with index -2. The data is split in half and each half follows this format independently, one after the other. Splitting the data allows each DMA to operate in its own space with no waste of time or duplication of data. An example is shown in table 4.2.

The second Python script in appendix C.3 is used to format the above format, either with or without channel interleaving, into C++ source code. The script created a header file that defines the sizes of the array, and K as well as a source code file that defines the array of data itself. Without an effective way to load a raw data file directly into memory, it was decided to add the dataset as a literal in the source code to be generated by the script. The Python script in appendix C.4 is used to call all the subscripts at once.

Table 4.2. Multi-channel dataset file format

-1	R_0	R_0	R_1	R_1	...	R_D	R_D
0	$S_{0,0}$	$S_{1,0}$	$S_{0,1}$	$S_{1,1}$	...	$S_{0,D}$	$S_{1,D}$
...	...	...	...	...	...	...	...
(n/2)-2	$S_{(n/2)-2,0}$	$S_{(n/2)-1,0}$	$S_{(n/2)-2,1}$	$S_{(n/2)-1,1}$	...	$S_{(n/2)-2,D}$	$S_{(n/2)-1,D}$
-2	0	0	0	0	...	0	0
-1	R_0	R_0	R_1	R_1	...	R_D	R_D
n/2	$S_{n/2,0}$	$S_{(n/2)+1,0}$	$S_{n/2,1}$	$S_{(n/2)+1,1}$	...	$S_{n/2,D}$	$S_{(n/2)+1,D}$
...	...	...	...	...	...	...	...
n-1	$S_{n-1,0}$	$S_{n,0}$	$S_{n-1,1}$	$S_{n,1}$	...	$S_{n-1,D}$	$S_{n,D}$
-2	0	0	0	0	...	0	0

CHAPTER 5

RESULTS

The accelerator was tested using six sets of data, the first five composed of synthetic integers (DataSet3, DataSet5, DataSet6, DataSet7, DataSet8) created by the Python script in appendix C.5 and the last being fixed-point numbers truncated from a floating-point set by the KDD 2004 Conference (phy_train) [8]. The synthetic datasets was generated using Python's *randint()* function to create a random set of 32-bit integers. The KDD dataset was generated in high energy collider experiments, and the task was to predict accuracy, ROC area, cross entropy, and q-score. Q-score being a particle physics performance metric for the effectiveness of prediction models. Each set includes varying number of points, each with 78 features. The DataSet8, DataSet5, DataSet6, DataSet7, DataSet3, and phy_train contain 1000, 5000, 10000, 25000, 50000, and 50000 points respectively. The resulting speedup of the FPGA accelerator versus the CPU when run with the six datasets and two values of K is shown in table 5.1. The datasets were preprocessed by the Python scripts, then run on both the FPGA and CPU recording the execution time. Ten trials of each test were executed; the mean time calculated for each.

5.1 Resource Utilization Statistics

When the accelerator was implemented on the ZedBoard with $K = 10$, less than half of the resources were utilized as shown in table 5.2. When the value of K was increased to 25, the slice LUTs and registers utilization increased, while the block RAM and DSPs utilization remained the same as shown in table 5.3. This is due to the increase in logic required for the sorting block.

Table 5.1. Resulting speedup (Execution Time(ms) / Speedup over CPU)

Device	CPU		FPGA	
K (# of nearest neighbors)	10	25	10	25
DataSet8 (1K points)	15.3/-	11.8/-	0.2/76.5	0.201/58.706
DataSet5 (5K points)	31.7/-	31.6/-	0.98/32.347	0.981/32.212
DataSet6 (10K points)	57.7/-	57.6/-	1.955/29.514	1.956/29.447
DataSet7 (25K points)	134.6/-	135.3/-	4.88/27.582	4.881/27.719
DataSet3 (50K points)	266.4/-	266.9/-	9.755/27.309	9.756/27.357
phy_train (50K points)	202.6/-	208.8/-	9.755/20.768	9.756/21.402

Table 5.2. Resource utilization of implementation when $K = 10$

Resource	Used	Available	Util%
Slice LUTs	23394	53200	43.97
Slice Registers	27612	106400	25.95
# of 36 Kb Block RAM	11.5	140	8.21
DSPs	16	220	7.27
BUFG	1	32	3.13

The power estimate of total on-chip power was 1.778W and 1.904W for the $K = 10$ and $K = 25$ implementations respectively. An increase in K shows an increase in power due to requiring more resources. Both estimates are still within Vivado's reasonable limits.

The timing analysis of both implementations show they were able to meet timing requirements for the accelerator clocked at 100 MHz. That means setup and hold time of the registers were not violated and there was enough time for the values to fully propagate through the logic.

Table 5.3. Resource utilization of implementation when $K = 25$

Resource	Used	Available	Util%
Slice LUTs	45295	53200	85.14
Slice Registers	41397	106400	38.91
Block RAM	11.5	140	8.21
DSPs	16	220	7.27
BUFG	2	32	6.25

CHAPTER 6

FUTURE WORK

Due to the accelerator's requirement of an uninterrupted stream of data, datasets much larger than what was used for testing cannot be processed in a single process. Possible fixes to this issue include modification of the accelerator to accept interrupted streams, or use of a DMA that has a larger data transfer limit. As the paper describes, this accelerator implements a KNN search that writes the index of the K nearest neighbors into memory. This search can be used to accelerate any classifier that uses this type of search such as KNN classifier. The classifications of each data point will be stored in memory and will be accessed through the ARM core using the indexes returned by the search. The ARM core will find the most common classification in the K nearest neighbors, as in KNN, and assign that to the R. Similar NN algorithms can use this search with a simple change to the ARM core's final classification code described above.

With an improvement in performance by more than 20 times, the hardware accelerator proposed in this paper can be used to enhance similar algorithms. Similar models can be developed to create a library of accelerators to allow FPGAs to reconfigure as needed. This allows many different applications connected to the cloud to benefit from "on the fly" technology.

APPENDIX A

HARDWARE

A.1 KNN_accelerator_v3_0.v

```
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 09/19/2016 11:57:31 PM
// Design Name: KNN accelerator v3 IP wrapper
// Module Name: KNN_accelerator_v3_0
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

        module KNN_accelerator_v3_0 #
        (
                // Users to add parameters here
                parameter WIDTH = 32,
                parameter NUM_DIM = 32,
                parameter KNN_DEBUG = 0,
                parameter NUM_CH = 4,
                parameter K = 10,
                // User parameters ends
                // Do not modify the parameters beyond this line
```

```
        // Parameters of Axi Slave Bus Interface S00_AXI
        parameter integer C_S00_AXI_DATA_WIDTH  = 32,
        parameter integer C_S00_AXI_ADDR_WIDTH  = 5,

        // Parameters of Axi Slave Bus Interface S00_AXIS
        parameter integer C_S00_AXIS_TDATA_WIDTH        =
            64,

        // Parameters of Axi Slave Bus Interface S01_AXIS
        parameter integer C_S01_AXIS_TDATA_WIDTH        =
            64,

        // Parameters of Axi Master Bus Interface M00_AXIS
        parameter integer C_M00_AXIS_TDATA_WIDTH        =
            32
)
(
        // Users to add ports here

        // User ports ends
        // Do not modify the ports beyond this line


        // Ports of Axi Slave Bus Interface S00_AXI
        input wire  s00_axi_aclk,
        input wire  s00_axi_aresetn,
        input wire [C_S00_AXI_ADDR_WIDTH-1 : 0]
            s00_axi_awaddr,
        input wire [2 : 0] s00_axi_awprot,
        input wire  s00_axi_awvalid,
        output wire  s00_axi_awready,
        input wire [C_S00_AXI_DATA_WIDTH-1 : 0]
            s00_axi_wdata,
        input wire [(C_S00_AXI_DATA_WIDTH/8)-1 : 0]
            s00_axi_wstrb,
        input wire  s00_axi_wvalid,
        output wire  s00_axi_wready,
        output wire [1 : 0] s00_axi_bresp,
        output wire  s00_axi_bvalid,
        input wire  s00_axi_bready,
```

```
                input wire [C_S00_AXI_ADDR_WIDTH-1 : 0]
                  s00_axi_araddr,
                input wire [2 : 0] s00_axi_arprot,
                input wire  s00_axi_arvalid,
                output wire  s00_axi_arready,
                output wire [C_S00_AXI_DATA_WIDTH-1 : 0]
                  s00_axi_rdata,
                output wire [1 : 0] s00_axi_rresp,
                output wire  s00_axi_rvalid,
                input wire  s00_axi_rready,

                // Ports of Axi Slave Bus Interface S00_AXIS
                input wire  s00_axis_aclk,
                input wire  s00_axis_aresetn,
                output wire  s00_axis_tready,
                input wire [C_S00_AXIS_TDATA_WIDTH-1 : 0]
                  s00_axis_tdata,
                input wire  s00_axis_tvalid,

                // Ports of Axi Slave Bus Interface S01_AXIS
                input wire  s01_axis_aclk,
                input wire  s01_axis_aresetn,
                output wire  s01_axis_tready,
                input wire [C_S01_AXIS_TDATA_WIDTH-1 : 0]
                  s01_axis_tdata,
                input wire  s01_axis_tvalid,

                // Ports of Axi Master Bus Interface M00_AXIS
                input wire  m00_axis_aclk,
                input wire  m00_axis_aresetn,
                output wire  m00_axis_tvalid,
                output wire [C_M00_AXIS_TDATA_WIDTH-1 : 0]
                  m00_axis_tdata,
                output wire [(C_M00_AXIS_TDATA_WIDTH/8)-1 : 0]
                  m00_axis_tstrb,
                output wire  m00_axis_tlast,
                input wire  m00_axis_tready
        );
// Instantiation of Axi Bus Interface S00_AXI
        KNN_accelerator_v3_0_S00_AXI # (
                .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
                .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
```

```
    ) KNN_accelerator_v3_0_S00_AXI_inst (
            .control_reset(control_reset),
            .control_done(control_done),
            .second_start_index(second_start_index),
            .dataName(AXIS_dataOut),
            .dataValue(dataValueOut),
            .S_AXI_ACLK(s00_axi_aclk),
            .S_AXI_ARESETN(s00_axi_aresetn),
            .S_AXI_AWADDR(s00_axi_awaddr),
            .S_AXI_AWPROT(s00_axi_awprot),
            .S_AXI_AWVALID(s00_axi_awvalid),
            .S_AXI_AWREADY(s00_axi_awready),
            .S_AXI_WDATA(s00_axi_wdata),
            .S_AXI_WSTRB(s00_axi_wstrb),
            .S_AXI_WVALID(s00_axi_wvalid),
            .S_AXI_WREADY(s00_axi_wready),
            .S_AXI_BRESP(s00_axi_bresp),
            .S_AXI_BVALID(s00_axi_bvalid),
            .S_AXI_BREADY(s00_axi_bready),
            .S_AXI_ARADDR(s00_axi_araddr),
            .S_AXI_ARPROT(s00_axi_arprot),
            .S_AXI_ARVALID(s00_axi_arvalid),
            .S_AXI_ARREADY(s00_axi_arready),
            .S_AXI_RDATA(s00_axi_rdata),
            .S_AXI_RRESP(s00_axi_rresp),
            .S_AXI_RVALID(s00_axi_rvalid),
            .S_AXI_RREADY(s00_axi_rready)
    );

// Instantiation of Axi Bus Interface S00_AXIS
    KNN_accelerator_v3_0_S00_AXIS # (
            .C_S_AXIS_TDATA_WIDTH(C_S00_AXIS_TDATA_WIDTH)
    ) KNN_accelerator_v3_0_S00_AXIS_inst (
            .fifo_wren(AXIS0_in_wr_en),
            .AXIS_data(AXIS0_dataIn),
            .S_AXIS_ACLK(s00_axis_aclk),
            .S_AXIS_ARESETN(s00_axis_aresetn),
            .S_AXIS_TREADY(s00_axis_tready),
            .S_AXIS_TDATA(s00_axis_tdata),
            .S_AXIS_TVALID(s00_axis_tvalid)
    );
```

```
// Instantiation of Axi Bus Interface S01_AXIS
        KNN_accelerator_v3_0_S00_AXIS # (
                .C_S_AXIS_TDATA_WIDTH(C_S01_AXIS_TDATA_WIDTH)
        ) KNN_accelerator_v3_0_S01_AXIS_inst (
                .fifo_wren(AXIS1_in_wr_en),
                .AXIS_data(AXIS1_dataIn),
                .S_AXIS_ACLK(s01_axis_aclk),
                .S_AXIS_ARESETN(s01_axis_aresetn),
                .S_AXIS_TREADY(s01_axis_tready),
                .S_AXIS_TDATA(s01_axis_tdata),
                .S_AXIS_TVALID(s01_axis_tvalid)
        );

// Instantiation of Axi Bus Interface M00_AXIS
        KNN_accelerator_v3_0_M00_AXIS # (
                .K(K),
                .C_M_AXIS_TDATA_WIDTH(C_M00_AXIS_TDATA_WIDTH)
        ) KNN_accelerator_v3_0_M00_AXIS_inst (
                .AXIS_data(AXIS_dataOut),
                .wr_en(AXIS_out_wr_en),
                .M_AXIS_ACLK(m00_axis_aclk),
                .M_AXIS_ARESETN(m00_axis_aresetn && ~control_reset
                    ),
                .M_AXIS_TVALID(m00_axis_tvalid),
                .M_AXIS_TDATA(m00_axis_tdata),
                .M_AXIS_TSTRB(m00_axis_tstrb),
                .M_AXIS_TLAST(m00_axis_tlast),
                .M_AXIS_TREADY(m00_axis_tready)
        );

        // Add user logic here

        //slv_reg0 = done, reset
        //slv_reg1 = second_start_index
        //slv_reg2 = count low
        //slv_reg3 = count high
        //slv_reg4 = AXIS_dataOut
        //slv_reg5 = dataValueOut

        //stream = dataValueIn

        wire control_reset;
```

```
        wire control_done;
        wire [31:0] second_start_index;
        wire AXIS0_in_wr_en;
        wire [C_S00_AXIS_TDATA_WIDTH-1:0] AXIS0_dataIn;
        wire AXIS1_in_wr_en;
        wire [C_S00_AXIS_TDATA_WIDTH-1:0] AXIS1_dataIn;
        wire AXIS_out_wr_en;
        wire [C_M00_AXIS_TDATA_WIDTH-1:0] AXIS_dataOut;
        wire [WIDTH-1:0] dataValueOut;

        knnTop #(
                .DATA_WIDTH(WIDTH),
                .DIMENSIONS(NUM_DIM),
                .DEBUG(KNN_DEBUG),
                .NUM_CH(NUM_CH),
                .K(K)
        ) knnTop (
                .mclk(s00_axi_aclk),
                .reset(control_reset),
                .done(control_done),
                .AXIS0_in_wr_en(AXIS0_in_wr_en),
                .AXIS1_in_wr_en(AXIS1_in_wr_en),
                .second_start_index(second_start_index),
                .dataValueIn0(AXIS0_dataIn),
                .dataValueIn1(AXIS1_dataIn),
                .AXIS_out_wr_en(AXIS_out_wr_en),
                .dataNameOut(AXIS_dataOut),
                .dataValueOut(dataValueOut)
        );

        // User logic ends

        endmodule
```

A.2 KNN_accelerator_v3_0_S00_AXI.v

```
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 09/19/2016 11:57:31 PM
```

```
// Design Name: AXI interface
// Module Name: KNN_accelerator_v3_0_S00_AXI
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

        module KNN_accelerator_v3_0_S00_AXI #
        (
                // Users to add parameters here

                // User parameters ends
                // Do not modify the parameters beyond this line

                // Width of S_AXI data bus
                parameter integer C_S_AXI_DATA_WIDTH    = 32,
                // Width of S_AXI address bus
                parameter integer C_S_AXI_ADDR_WIDTH    = 5
        )
        (
                // Users to add ports here
                output                  control_reset,
                output                  control_done,
                output [31:0]   second_start_index,
                input [31:0]    dataName,
                input [31:0]    dataValue,
                // User ports ends
                // Do not modify the ports beyond this line

                // Global Clock Signal
                input wire  S_AXI_ACLK,
                // Global Reset Signal. This Signal is Active LOW
                input wire  S_AXI_ARESETN,
```

```
// Write address (issued by master, acceped by
   Slave)
input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_AWADDR
   ,
// Write channel Protection type. This signal
   indicates the
// privilege and security level of the transaction
   , and whether
// the transaction is a data access or an
   instruction access.
input wire [2 : 0] S_AXI_AWPROT,
// Write address valid. This signal indicates that
    the master signaling
// valid write address and control information.
input wire  S_AXI_AWVALID,
// Write address ready. This signal indicates that
    the slave is ready
// to accept an address and associated control
   signals.
output wire  S_AXI_AWREADY,
// Write data (issued by master, acceped by Slave)
input wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_WDATA,
// Write strobes. This signal indicates which byte
    lanes hold
// valid data. There is one write strobe bit for
   each eight
// bits of the write data bus.
input wire [(C_S_AXI_DATA_WIDTH/8)-1 : 0]
   S_AXI_WSTRB,
// Write valid. This signal indicates that valid
   write
// data and strobes are available.
input wire  S_AXI_WVALID,
// Write ready. This signal indicates that the
   slave
// can accept the write data.
output wire  S_AXI_WREADY,
// Write response. This signal indicates the
   status
// of the write transaction.
output wire [1 : 0] S_AXI_BRESP,
```

```
// Write response valid. This signal indicates
   that the channel
// is signaling a valid write response.
output wire  S_AXI_BVALID,
// Response ready. This signal indicates that the
   master
// can accept a write response.
input wire  S_AXI_BREADY,
// Read address (issued by master, acceped by
   Slave)
input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_ARADDR
   ,
// Protection type. This signal indicates the
   privilege
// and security level of the transaction, and
   whether the
// transaction is a data access or an instruction
   access.
input wire [2 : 0] S_AXI_ARPROT,
// Read address valid. This signal indicates that
   the channel
// is signaling valid read address and control
   information.
input wire  S_AXI_ARVALID,
// Read address ready. This signal indicates that
   the slave is
// ready to accept an address and associated
   control signals.
output wire  S_AXI_ARREADY,
// Read data (issued by slave)
output wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_RDATA
   ,
// Read response. This signal indicates the status
    of the
// read transfer.
output wire [1 : 0] S_AXI_RRESP,
// Read valid. This signal indicates that the
   channel is
// signaling the required read data.
output wire  S_AXI_RVALID,
// Read ready. This signal indicates that the
   master can
```

```
        // accept the read data and response information.
        input wire  S_AXI_RREADY
);

// AXI4LITE signals
reg [C_S_AXI_ADDR_WIDTH-1 : 0]  axi_awaddr;
reg     axi_awready;
reg     axi_wready;
reg [1 : 0]     axi_bresp;
reg     axi_bvalid;
reg [C_S_AXI_ADDR_WIDTH-1 : 0]  axi_araddr;
reg     axi_arready;
reg [C_S_AXI_DATA_WIDTH-1 : 0]  axi_rdata;
reg [1 : 0]     axi_rresp;
reg     axi_rvalid;

// Example-specific design signals
// local parameter for addressing 32 bit / 64 bit
   C_S_AXI_DATA_WIDTH
// ADDR_LSB is used for addressing 32/64 bit registers/
   memories
// ADDR_LSB = 2 for 32 bits (n downto 2)
// ADDR_LSB = 3 for 64 bits (n downto 3)
localparam integer ADDR_LSB = (C_S_AXI_DATA_WIDTH/32) + 1;
localparam integer OPT_MEM_ADDR_BITS = 2;
//------------------------------------------------
//-- Signals for user logic register space example
//------------------------------------------------
//-- Number of Slave Registers 6
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg0;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg1;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg2;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg3;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg4;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg5;
wire    slv_reg_rden;
wire    slv_reg_wren;
reg [C_S_AXI_DATA_WIDTH-1:0]    reg_data_out;
integer  byte_index;

// I/O Connections assignments
```

```
assign S_AXI_AWREADY    = axi_awready;
assign S_AXI_WREADY     = axi_wready;
assign S_AXI_BRESP      = axi_bresp;
assign S_AXI_BVALID     = axi_bvalid;
assign S_AXI_ARREADY    = axi_arready;
assign S_AXI_RDATA      = axi_rdata;
assign S_AXI_RRESP      = axi_rresp;
assign S_AXI_RVALID     = axi_rvalid;
// Implement axi_awready generation
// axi_awready is asserted for one S_AXI_ACLK clock cycle
   when both
// S_AXI_AWVALID and S_AXI_WVALID are asserted.
   axi_awready is
// de-asserted when reset is low.

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_awready <= 1'b0;
    end
  else
    begin
      if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID)
        begin
          // slave is ready to accept write address when
          // there is a valid write address and write data
          // on the write address and data bus. This
             design
          // expects no outstanding transactions.
          axi_awready <= 1'b1;
        end
      else
        begin
          axi_awready <= 1'b0;
        end
    end
end

// Implement axi_awaddr latching
// This process is used to latch the address when both
// S_AXI_AWVALID and S_AXI_WVALID are valid.
```

```
always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_awaddr <= 0;
    end
  else
    begin
      if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID)
        begin
          // Write Address latching
          axi_awaddr <= S_AXI_AWADDR;
        end
    end
end

// Implement axi_wready generation
// axi_wready is asserted for one S_AXI_ACLK clock cycle
   when both
// S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready
    is
// de-asserted when reset is low.

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_wready <= 1'b0;
    end
  else
    begin
      if (~axi_wready && S_AXI_WVALID && S_AXI_AWVALID)
        begin
          // slave is ready to accept write data when
          // there is a valid write address and write data
          // on the write address and data bus. This
             design
          // expects no outstanding transactions.
          axi_wready <= 1'b1;
        end
      else
```

```
        begin
          axi_wready <= 1'b0;
        end
    end
end

// Implement memory mapped register select and write logic
    generation
// The write data is accepted and written to memory mapped
    registers when
// axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID
   are asserted. Write strobes are used to
// select byte enables of slave registers while writing.
// These registers are cleared when reset (active low) is
   applied.
// Slave register write enable is asserted when valid
   address and data are available
// and the slave is ready to accept the write address and
   write data.
assign slv_reg_wren = axi_wready && S_AXI_WVALID &&
   axi_awready && S_AXI_AWVALID;

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      slv_reg0 <= 0;
      slv_reg1 <= 0;
      slv_reg2 <= 0;
      slv_reg3 <= 0;
      slv_reg4 <= 0;
      slv_reg5 <= 0;
    end
  else begin
    if (slv_reg_wren)
      begin
        case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:
           ADDR_LSB] )
          3'h0:
            for ( byte_index = 0; byte_index <= (
               C_S_AXI_DATA_WIDTH/8)-1; byte_index =
               byte_index+1 )
```

```
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as
            per write strobes
        // Slave register 0
        slv_reg0[(byte_index*8) +: 8] <=
          S_AXI_WDATA[(byte_index*8) +: 8];
      end
  3'h1:
    for ( byte_index = 0; byte_index <= (
      C_S_AXI_DATA_WIDTH/8)-1; byte_index =
      byte_index+1 )
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as
            per write strobes
        // Slave register 1
        slv_reg1[(byte_index*8) +: 8] <=
          S_AXI_WDATA[(byte_index*8) +: 8];
      end
  3'h2:
    for ( byte_index = 0; byte_index <= (
      C_S_AXI_DATA_WIDTH/8)-1; byte_index =
      byte_index+1 )
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as
            per write strobes
        // Slave register 2
        slv_reg2[(byte_index*8) +: 8] <=
          S_AXI_WDATA[(byte_index*8) +: 8];
      end
  3'h3:
    for ( byte_index = 0; byte_index <= (
      C_S_AXI_DATA_WIDTH/8)-1; byte_index =
      byte_index+1 )
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as
            per write strobes
        // Slave register 3
        slv_reg3[(byte_index*8) +: 8] <=
          S_AXI_WDATA[(byte_index*8) +: 8];
      end
  3'h4:
```

```
                for ( byte_index = 0; byte_index <= (
                    C_S_AXI_DATA_WIDTH/8)-1; byte_index =
                    byte_index+1 )
                  if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                    // Respective byte enables are asserted as
                        per write strobes
                    // Slave register 4
                    slv_reg4[(byte_index*8) +: 8] <=
                      S_AXI_WDATA[(byte_index*8) +: 8];
                  end
              3'h5:
                for ( byte_index = 0; byte_index <= (
                    C_S_AXI_DATA_WIDTH/8)-1; byte_index =
                    byte_index+1 )
                  if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                    // Respective byte enables are asserted as
                        per write strobes
                    // Slave register 5
                    slv_reg5[(byte_index*8) +: 8] <=
                      S_AXI_WDATA[(byte_index*8) +: 8];
                  end
              default : begin
                          slv_reg0 <= slv_reg0;
                          slv_reg1 <= slv_reg1;
                          slv_reg2 <= slv_reg2;
                          slv_reg3 <= slv_reg3;
                          slv_reg4 <= slv_reg4;
                          slv_reg5 <= slv_reg5;
                        end
            endcase
          end
      end
    end

    // Implement write response logic generation
    // The write response and response valid signals are
       asserted by the slave
    // when axi_wready, S_AXI_WVALID, axi_wready and
       S_AXI_WVALID are asserted.
    // This marks the acceptance of address and indicates the
       status of
    // write transaction.
```

```
always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_bvalid  <= 0;
      axi_bresp   <= 2'b0;
    end
  else
    begin
      if (axi_awready && S_AXI_AWVALID && ~axi_bvalid &&
         axi_wready && S_AXI_WVALID)
        begin
          // indicates a valid write response is available
          axi_bvalid <= 1'b1;
          axi_bresp  <= 2'b0; // 'OKAY' response
        end                   // work error responses in
           future
      else
        begin
          if (S_AXI_BREADY && axi_bvalid)
            //check if bready is asserted while bvalid is
               high)
            //(there is a possibility that bready is
               always asserted high)
            begin
              axi_bvalid <= 1'b0;
            end
        end
    end
end

// Implement axi_arready generation
// axi_arready is asserted for one S_AXI_ACLK clock cycle
   when
// S_AXI_ARVALID is asserted. axi_awready is
// de-asserted when reset (active low) is asserted.
// The read address is also latched when S_AXI_ARVALID is
// asserted. axi_araddr is reset to zero on reset
   assertion.

always @( posedge S_AXI_ACLK )
```

```
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_arready <= 1'b0;
      axi_araddr  <= 32'b0;
    end
  else
    begin
      if (~axi_arready && S_AXI_ARVALID)
        begin
          // indicates that the slave has acceped the
             valid read address
          axi_arready <= 1'b1;
          // Read address latching
          axi_araddr  <= S_AXI_ARADDR;
        end
      else
        begin
          axi_arready <= 1'b0;
        end
    end
end

// Implement axi_arvalid generation
// axi_rvalid is asserted for one S_AXI_ACLK clock cycle
   when both
// S_AXI_ARVALID and axi_arready are asserted. The slave
   registers
// data are available on the axi_rdata bus at this
   instance. The
// assertion of axi_rvalid marks the validity of read data
    on the
// bus and axi_rresp indicates the status of read
   transaction. axi_rvalid
// is deasserted on reset (active low). axi_rresp and
   axi_rdata are
// cleared to zero on reset (active low).
always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_rvalid <= 0;
```

```
        axi_rresp  <= 0;
      end
    else
      begin
        if (axi_arready && S_AXI_ARVALID && ~axi_rvalid)
          begin
            // Valid read data is available at the read data
                bus
            axi_rvalid <= 1'b1;
            axi_rresp  <= 2'b0; // 'OKAY' response
          end
        else if (axi_rvalid && S_AXI_RREADY)
          begin
            // Read data is accepted by the master
            axi_rvalid <= 1'b0;
          end
      end
end

// Implement memory mapped register select and read logic
   generation
// Slave register read enable is asserted when valid
   address is available
// and the slave is ready to accept the read address.
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~
   axi_rvalid;
always @(*)
begin
        // Address decoding for reading registers
        case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:
           ADDR_LSB] )
          3'h0   : reg_data_out <= slv_reg0;
          3'h1   : reg_data_out <= slv_reg1;
          3'h2   : reg_data_out <= counter[31:0];
          3'h3   : reg_data_out <= counter[63:32];
          3'h4   : reg_data_out <= dataName;
          3'h5   : reg_data_out <= dataValue;
          default : reg_data_out <= 0;
        endcase
end

// Output register or memory read data
```

```
always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_rdata  <= 0;
    end
  else
    begin
      // When there is a valid read address (S_AXI_ARVALID
         ) with
      // acceptance of read address by the slave (
         axi_arready),
      // output the read dada
      if (slv_reg_rden)
        begin
          axi_rdata <= reg_data_out;     // register read
             data
        end
    end
end

// Add user logic here

reg [63:0] counter;

assign control_reset = slv_reg0[0];
assign control_done = slv_reg0[1];
assign second_start_index = slv_reg1;

always @(posedge S_AXI_ACLK or negedge S_AXI_ARESETN)
begin : proc_counter
        if(~S_AXI_ARESETN)
        begin
                counter <= 0;
        end
        else
        begin
                counter <= counter + 1;
        end
end

// User logic ends
```

```
endmodule
```

A.3 KNN_accelerator_v3_0_M00_AXIS.v

```
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 09/19/2016 11:57:31 PM
// Design Name: AXI interface
// Module Name: KNN_accelerator_v3_0_M00_AXIS
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////

        module KNN_accelerator_v3_0_M00_AXIS #
        (
                // Users to add parameters here
                parameter K = 10,
                // User parameters ends
                // Do not modify the parameters beyond this line

                // Width of S_AXIS address bus. The slave accepts
                   the read and write addresses of width
                   C_M_AXIS_TDATA_WIDTH.
                parameter integer C_M_AXIS_TDATA_WIDTH  = 32
        )
        (
                // Users to add ports here
                input [C_M_AXIS_TDATA_WIDTH-1:0] AXIS_data,
                input wr_en,
```

```
        // User ports ends
        // Do not modify the ports beyond this line

        // Global ports
        input wire  M_AXIS_ACLK,
        //
        input wire  M_AXIS_ARESETN,
        // Master Stream Ports. TVALID indicates that the
           master is driving a valid transfer, A transfer
           takes place when both TVALID and TREADY are
           asserted.
        output wire  M_AXIS_TVALID,
        // TDATA is the primary payload that is used to
           provide the data that is passing across the
           interface from the master.
        output wire [C_M_AXIS_TDATA_WIDTH-1 : 0]
           M_AXIS_TDATA,
        // TSTRB is the byte qualifier that indicates
           whether the content of the associated byte of
           TDATA is processed as a data byte or a position
            byte.
        output wire [(C_M_AXIS_TDATA_WIDTH/8)-1 : 0]
           M_AXIS_TSTRB,
        // TLAST indicates the boundary of a packet.
        output wire  M_AXIS_TLAST,
        // TREADY indicates that the slave can accept a
           transfer in the current cycle.
        input wire  M_AXIS_TREADY
);

// function called clogb2 that returns an integer which
   has the
// value of the ceiling of the log base 2.
function integer clogb2 (input integer bit_depth);
  begin
        for(clogb2=0; bit_depth>0; clogb2=clogb2+1)
          bit_depth = bit_depth >> 1;
  end
endfunction

// bit_num gives the minimum number of bits needed to
   address 'depth' size of FIFO.
```

```
localparam bit_num  = clogb2(K);

// Define the states of state machine
// The control state machine oversees the writing of input
    streaming data to the FIFO,
// and outputs the streaming data from the FIFO
parameter [1:0] IDLE = 2'b00,        // This is the
    initial/idle state
                                SEND_STREAM   = 2'b10; //
                                    In this state the
                                    stream data is output
                                    through M_AXIS_TDATA
// State variable
reg [1:0] mst_exec_state;
// Example design FIFO read pointer
reg [bit_num-1:0] read_pointer;
// Example design FIFO write pointer
reg [bit_num-1:0] write_pointer;

// AXI Stream internal signals
//streaming data valid
wire    axis_tvalid;
//streaming data valid delayed by one clock cycle
reg     axis_tvalid_delay;
//Last of the streaming data
wire    axis_tlast;
//Last of the streaming data delayed by one clock cycle
reg     axis_tlast_delay;
//FIFO implementation signals
reg [C_M_AXIS_TDATA_WIDTH-1:0]  FIFO [K-1:0];
reg [C_M_AXIS_TDATA_WIDTH-1:0]  stream_data_out;
wire    tx_en;
//The master has issued all the streaming data stored in
   FIFO
reg     tx_done;

integer i;


// I/O Connections assignments

assign M_AXIS_TVALID    = axis_tvalid_delay;
```

```
assign M_AXIS_TDATA     = stream_data_out;
assign M_AXIS_TLAST     = axis_tlast_delay;
assign M_AXIS_TSTRB     = {(C_M_AXIS_TDATA_WIDTH/8){1'b1
   }};


// Control state machine implementation
always @(posedge M_AXIS_ACLK)
begin
        if (!M_AXIS_ARESETN)
                // Synchronous reset (active low)
                begin
                        mst_exec_state <= IDLE;
                end
        else
        case (mst_exec_state)
        IDLE:
                // The slave starts accepting tdata when
                // there tvalid is asserted to mark the
                // presence of valid streaming data
                if (write_pointer == K && read_pointer ==
                   0 )
                begin
                        mst_exec_state  <= SEND_STREAM;
                end
                else
                begin
                        mst_exec_state  <= IDLE;
                end
        SEND_STREAM:
                // The example design streaming master
                   functionality starts
                // when the master drives output tdata
                   from the FIFO and the slave
                // has finished storing the S_AXIS_TDATA
                if (tx_done)
                begin
                        mst_exec_state <= IDLE;
                end
                else
                begin
                        mst_exec_state <= SEND_STREAM;
```

```
                end
        endcase
end


//tvalid generation
//axis_tvalid is asserted when the control state machine's
    state is SEND_STREAM and
//number of output streaming data is less than the K.
assign axis_tvalid = ((mst_exec_state == SEND_STREAM) && (
   read_pointer < K));

// AXI tlast generation
// axis_tlast is asserted number of output streaming data
   is K-1
// (0 to K-1)
assign axis_tlast = (read_pointer == K-1);

// Delay the axis_tvalid and axis_tlast signal by one
   clock cycle
// to match the latency of M_AXIS_TDATA
always @(posedge M_AXIS_ACLK)
begin
        if (!M_AXIS_ARESETN)
        begin
                axis_tvalid_delay <= 1'b0;
                axis_tlast_delay <= 1'b0;
        end
        else
        begin
                axis_tvalid_delay <= axis_tvalid;
                axis_tlast_delay <= axis_tlast;
        end
end


//read_pointer pointer

always@(posedge M_AXIS_ACLK)
begin
        if(!M_AXIS_ARESETN)
        begin
```

```
                read_pointer <= 0;
                tx_done <= 1'b0;
        end
        else
        begin
                if (read_pointer <= K-1)
                begin
                        if (tx_en)
                        // read pointer is incremented
                           after every read from the FIFO
                        // when FIFO read signal is
                           enabled.
                        begin
                                read_pointer <=
                                   read_pointer + 1;
                                tx_done <= 1'b0;
                        end
                end
                else if (read_pointer == K)
                begin
                        // tx_done is asserted when K
                           numbers of streaming data
                        // has been out.
                        tx_done <= 1'b1;
                end
        end
end

//write_pointer pointer

always@(posedge M_AXIS_ACLK)
begin
        if(!M_AXIS_ARESETN)
        begin
                write_pointer <= 0;
        end
        else
        begin
                if (wr_en && write_pointer <= K-1)
                begin
                        write_pointer <= write_pointer +
                           1;
```

```
                end
        end
end


//FIFO read enable generation

assign tx_en = M_AXIS_TREADY && axis_tvalid;

// Streaming output data is read from FIFO
always @( posedge M_AXIS_ACLK )
begin
        if(!M_AXIS_ARESETN)
        begin
                stream_data_out <= 0;
        end
        else if (tx_en)// && M_AXIS_TSTRB[byte_index]
        begin
                stream_data_out <= FIFO[read_pointer];
        end
end

// Add user logic here
always @( posedge M_AXIS_ACLK )
begin
        if(!M_AXIS_ARESETN)
        begin
                for(i = 0; i < K; i = i + 1)
                begin
                        FIFO[i] <= 0;
                end
        end
        else if (wr_en)// && M_AXIS_TSTRB[byte_index]
        begin
                FIFO[write_pointer] <= AXIS_data;
        end
end
// User logic ends

endmodule
```

A.4 KNN_accelerator_v3_0_S00_AXIS.v

```
`timescale 1 ns / 1 ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 09/12/2016 01:49:31 PM
// Design Name: AXIS Interface
// Module Name: KNN_accelerator_v3_0_S00_AXIS
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

        module KNN_accelerator_v3_0_S00_AXIS #
        (
                // AXI4Stream sink: Data Width
                parameter integer C_S_AXIS_TDATA_WIDTH  = 32
        )
        (
                // Users to add ports here
                output fifo_wren,
                output [C_S_AXIS_TDATA_WIDTH-1:0] AXIS_data,
                // User ports ends
                // Do not modify the ports beyond this line

                // AXI4Stream sink: Clock
                input wire  S_AXIS_ACLK,
                // AXI4Stream sink: Reset
                input wire  S_AXIS_ARESETN,
                // Ready to accept data in
                output wire  S_AXIS_TREADY,
                // Data in
```

```
            input wire [C_S_AXIS_TDATA_WIDTH-1 : 0]
                S_AXIS_TDATA,
            // Data is in valid
            input wire  S_AXIS_TVALID
    );

    assign S_AXIS_TREADY = 1;
    assign AXIS_data = S_AXIS_TDATA;
    assign fifo_wren = S_AXIS_TVALID;

    endmodule
```

A.5 knnTop.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 04/20/2016 09:50:48 AM
// Design Name: KNN Top
// Module Name: knnTop
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// (* mark_debug = "true" *)
module knnTop #(
    parameter DATA_WIDTH = 32,
    parameter DIMENSIONS = 32,
    parameter DEBUG = 0,
    parameter NUM_CH = 1,
    parameter K = 1
```

```
) (
        input
            mclk,
        input
            reset,
        input
            done,
        input
            AXIS0_in_wr_en,
        input
            AXIS1_in_wr_en,
        input [31:0]
            second_start_index,
        input [63:0]
            dataValueIn0,
        input [63:0]
            dataValueIn1,
        output
            AXIS_out_wr_en,
        output [31:0]
            dataNameOut,
        output [DATA_WIDTH-1:0]                    dataValueOut
);

        localparam VAL_WIDTH = 2*(DATA_WIDTH+1)+DIMENSIONS;

        wire [31:0] dataNameP1 [NUM_CH-1:0];
        wire [VAL_WIDTH-1:0] dataValueP1 [NUM_CH-1:0];
        reg [31:0] dataNameP1Reg;
        reg [VAL_WIDTH-1:0] dataValueP1Reg;
        reg [NUM_CH-1:0] channelOutEnable;
        wire transferDone;
        wire AXIS_in_wr_en;
        reg validP2;
        reg validP2Reg;
        reg [1:0] channelSelect;
        reg finalOutEnable;
        reg finalOutEnableReg;
        reg [9:0] wr_en_dly;
        integer transferCounter;

        generate
```

```
        genvar channel;
        for (channel = 0; channel < NUM_CH; channel =
            channel + 1)
        begin: fifo_distance_gen
                wire wr_en;
                wire [127:0] dataConcat;
                wire [31:0] dataValueIn;

                assign wr_en = channel < 2 ?
                    AXIS0_in_wr_en : AXIS1_in_wr_en;
                assign dataConcat = {dataValueIn1,
                    dataValueIn0};
                assign dataValueIn = dataConcat[((channel)
                    *DATA_WIDTH) +: DATA_WIDTH];

                knnPipe #(
                        .DATA_WIDTH(DATA_WIDTH),
                        .DIMENSIONS(DIMENSIONS),
                        .INSTANCE(channel),
                        .DEBUG(DEBUG),
                        .K(K)
                ) pipe (
                        .mclk(mclk),
                        .reset(reset),
                        .done(done),
                        .AXIS_in_wr_en(wr_en),
                        .start_index(second_start_index),
                        .dataValueIn(dataValueIn),
                        .outEn(channelOutEnable[channel]),
                        .dataNameOut(dataNameP1[channel]),
                        .dataValueOut(dataValueP1[channel
                            ])
                );
        end
endgenerate

kSortingP2 #(
        .DATA_WIDTH(DATA_WIDTH),
        .DIMENSIONS(DIMENSIONS),
        .VAL_WIDTH(VAL_WIDTH),
        .PASS_THOO_DEBUG(DEBUG),
        .K(K)
```

```
) P2 (
        .clk(mclk),
        .reset(reset),
        .valid(validP2Reg),
        .done(done),
        .outEn(finalOutEnableReg),
        .dataNameIn(dataNameP1Reg),
        .dataValueIn(dataValueP1Reg),
        .transferDone(transferDone),
        .dataNameOut(dataNameOut),
        .dataValueOut(dataValueOut)
);

always @(posedge mclk)
begin
        // Write enable delay to clear pipe
        if (reset)
        begin
                wr_en_dly <= 10'd0;
                dataNameP1Reg <= 0;
                dataValueP1Reg <= 0;
                validP2Reg <= 0;
                finalOutEnableReg <= 0;
        end
        else
        begin
                wr_en_dly <= {wr_en_dly[8:0],
                    AXIS_in_wr_en};
                dataNameP1Reg <= dataNameP1[channelSelect
                    ];
                dataValueP1Reg <= dataValueP1[
                    channelSelect];
                validP2Reg <= validP2;
                finalOutEnableReg <= finalOutEnable;
        end
end

// Need to add state machine to update channelSelect and
    channelOutEnable after 10ish low AXIS_in_wr_en and done
// finalOutEnable high after transfer complete
always @(posedge mclk)
begin
```

```
// Switching stuff
if (reset)
begin
        transferCounter <= 0;
        channelSelect <= 8'd0;
        channelOutEnable <= {NUM_CH{1'b0}};
        finalOutEnable <= 1'b0;
        validP2 <= 1'b0;
end
else if (done && (wr_en_dly == 0))
begin
        if (finalOutEnable)
        begin
                validP2 <= 1'b0;
        end
        else if (~validP2)
        begin
                validP2 <= 1'b1;
                channelOutEnable <= 1;
        end
        else
        begin
                if (channelOutEnable == 0)
                begin
                        finalOutEnable <= 1'b1;
                        validP2 <= 1'b0;
                end
                else if (transferCounter < K-1)
                begin
                        transferCounter <=
                           transferCounter + 1;
                end
                else
                begin
                        transferCounter <= 0;
                        channelSelect <=
                           channelSelect + 1;
                        channelOutEnable =
                           channelOutEnable << 1;
                        if (channelOutEnable == 0)
                        begin
```

```
                                                finalOutEnable <=
                                                    1'b1;
                                                validP2 <= 1'b0;
                                            end
                                        end
                                    end
                                end
                            end

            assign AXIS_in_wr_en = AXIS0_in_wr_en | AXIS1_in_wr_en;
            assign AXIS_out_wr_en = finalOutEnableReg & ~transferDone;

endmodule
```

A.6 knnTop_regwrap.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 07/29/2016 09:50:48 AM
// Design Name: KNN Top Register Wrapper
// Module Name: knnTop_regwrap
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module knnTop_regwrap #(
        parameter dataWidth = 32,
        parameter numberOfDimensions = 32,
        parameter numberOfChannels = 1,
        parameter k = 1
```

```
) (
        input                                       clk,
        input                                       reset,
        input                                       wr_en,
        input                                       done,
        input [(numberOfChannels*dataWidth)-1:0]
           dataValueIn,
        output                      AXIS_out_wr_en,
        output [31:0]                       dataNameOut,
        output [dataWidth-1:0]      dataValueOut
);

        reg                                 reset_reg;
        reg                                 AXIS_in_wr_en_reg;
        reg                                 done_reg;
        reg [(numberOfChannels*dataWidth)-1:0] dataValueIn_reg;

        always @(posedge clk)
        begin
                reset_reg <= reset;
                AXIS_in_wr_en_reg <= wr_en;
                done_reg <= done;
                dataValueIn_reg <= dataValueIn;
        end

        knnTop #(
                .DATA_WIDTH(dataWidth),
                .DIMENSIONS(numberOfDimensions),
                .NUM_CH(numberOfChannels),
                .K(k)
        ) top (
                .mclk                   (clk),
                .reset                  (reset_reg),
                .AXIS_in_wr_en  (AXIS_in_wr_en_reg),
                .done                   (done_reg),
                .dataValueIn    (dataValueIn_reg),
                .AXIS_out_wr_en (AXIS_out_wr_en),
                .dataNameOut    (dataNameOut),
                .dataValueOut   (dataValueOut)
        );

endmodule // knnTop
```

A.7 knnTop_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 07/29/2016 09:50:48 AM
// Design Name: KNN Top Testbench
// Module Name: knnTop_tb
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module knnTop_tb();

        parameter width = 32;
        parameter dim = 5;
        parameter ch = 2;
        parameter k = 3;

        reg clk;
        reg reset;
        reg wr_en;
        reg done;
        reg [(ch*32)-1:0] dataValueIn;
        wire [31:0] dataNameOut;
        wire [31:0] dataValueOut;

        knnTop_regwrap #(
                .dataWidth(width),
                .numberOfDimensions(dim),
```

```
        .numberOfChannels(ch),
        .k(k)
) uut (
        .clk                    (clk),
        .reset                  (reset),
        .wr_en                  (wr_en),
        .done                   (done),
        .dataValueIn    (dataValueIn),
        .dataNameOut    (dataNameOut),
        .dataValueOut   (dataValueOut)
);

initial
begin
        clk = 0;
        forever begin
                #10 clk = ~clk;
        end
end

initial
begin
        reset = 1;
        done = 0;
        dataValueIn = 0;
        wr_en = 0;
        #90;
        reset = 0;
        #20;
        wr_en = 1;
        dataValueIn = {32'd0, 32'd1}; // ref start
        #20;
        dataValueIn = {32'd0, 32'd2};
        #20;
        dataValueIn = {32'd0, 32'd2};
        #20;
        dataValueIn = {32'd0, 32'd2};
        #20;
        dataValueIn = {32'd0, 32'd3}; // ref end
        #20;
        dataValueIn = {32'd1, 32'd5}; // start 2,1
        #20;
```

```
                dataValueIn = {32'd1, 32'd10};
                #20;
                dataValueIn = {32'd1, 32'd7};
                #20;
                dataValueIn = {32'd1, 32'd9};
                #20;
                dataValueIn = {32'd1, 32'd6}; // end 2,1
                #20;
                dataValueIn = {32'd2, 32'd2}; // start 4,3
                #20;
                dataValueIn = {32'd2, 32'd2};
                #20;
                dataValueIn = {32'd2, 32'd2};
                #20;
                dataValueIn = {32'd2, 32'd2};
                #20;
                dataValueIn = {32'd2, 32'd2}; // end 4,3
                #20;
                dataValueIn = {32'd0, 32'd5}; // start X,5
                #20;
                dataValueIn = {32'd0, 32'd5};
                #20;
                dataValueIn = {32'd0, 32'd5};
                #20;
                dataValueIn = {32'd0, 32'd5};
                #20;
                dataValueIn = {32'd0, 32'd5}; // end X,5
                #20;
                wr_en = 0;
                #200;
                done = 1;
                #200;
        end

endmodule
```

A.8 knnPipe.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
```

```
//
// Create Date: 10/06/2016 06:05:39 PM
// Design Name: KNN Pipe
// Module Name: knnPipe
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////

module knnPipe #(
        parameter DATA_WIDTH = 32,
        parameter DIMENSIONS = 32,
        parameter INSTANCE = 0,
        parameter DEBUG = 0,
        parameter K = 1
) (
        input
            mclk,
        input
            reset,
        input
            done,
        input
            AXIS_in_wr_en,
        input [31:0]
            start_index,
        input [DATA_WIDTH-1:0]                  dataValueIn,
        input
            outEn,
        output [31:0]
            dataNameOut,
        output [VAL_WIDTH-1:0]                  dataValueOut
);
```

```
localparam VAL_WIDTH = 2*(DATA_WIDTH+1)+DIMENSIONS;

wire [DATA_WIDTH-1:0]    currentRefPoint;
wire [DATA_WIDTH-1:0]    currentDataPoint;
wire                                    dataValid;
wire                                    distanceValid;
wire [VAL_WIDTH-1:0]     distance;

fifo #(
        .DIMENSIONS(DIMENSIONS),
        .DATA_WIDTH(DATA_WIDTH)
) fifo (
        .clk(mclk),
        .rst(reset),
        .wr_en(AXIS_in_wr_en),
        .start(~reset),
        .dataIn(dataValueIn),
        .currentRefPoint(currentRefPoint),
        .currentDataPoint(currentDataPoint),
        .dataOut_Valid(dataValid)
);

distanceCalculationAccumulator #(
        .DATA_WIDTH(DATA_WIDTH),
        .DIMENSIONS(DIMENSIONS),
        .VAL_WIDTH(VAL_WIDTH)
) dist(
        .clk(mclk),
        .reset(reset),
        .wr_en(AXIS_in_wr_en),
        .dataIn_Valid(dataValid),
        .done(done),
        .data1(currentRefPoint),
        .data2(currentDataPoint),
        .distance(distance),
        .distanceValid(distanceValid)
);

kSortingP1 #(
        .DATA_WIDTH(DATA_WIDTH),
        .DIMENSIONS(DIMENSIONS),
        .VAL_WIDTH(VAL_WIDTH),
```

```
                .INSTANCE(INSTANCE),
                .PASS_THOO_DEBUG(DEBUG),
                .K(K)
        ) P1 (
                .clk(mclk),
                .reset(reset),
                .valid(distanceValid),
                .done(done),
                .outEn(outEn),
                .start_index(start_index),
                .dataValueIn(distance),
                .dataNameOut(dataNameOut),
                .dataValueOut(dataValueOut)
        );

endmodule
```

A.9 fifo.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 04/20/2016 09:50:48 AM
// Design Name: FIFO
// Module Name: fifo
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module fifo #(
        parameter DATA_WIDTH = 32,
```

```
        parameter DIMENSIONS = 32
) (
        input                                   clk,
        input                                   rst,
        input                                   wr_en,
        input                                   start,
        input [DATA_WIDTH-1:0]  dataIn,
        output reg [DATA_WIDTH-1:0] currentRefPoint,
        output reg [DATA_WIDTH-1:0] currentDataPoint,
        output reg
           dataOut_Valid
);

        reg [DATA_WIDTH-1:0] mem [DIMENSIONS-1:0];
        reg firstTime;
        integer counter;
        integer i;

        always @(posedge clk)
        begin : proc_mem
                if(rst) begin
                        for (i = 0; i < DIMENSIONS; i = i + 1)
                        begin
                                mem[i] <= 0;
                        end
                end
                else
                begin
                        if(wr_en && start && firstTime)
                        begin
                                mem[counter] <= dataIn;
                        end
                end
        end

        always @(posedge clk)
        begin : proc_counter
                if(rst)
                begin
                        counter <= 0;
                        firstTime <= 1;
                end
```

```
                else
                begin
                        if(wr_en && start)
                        begin
                                if(counter < DIMENSIONS-1)
                                begin
                                        counter <= counter + 1;
                                end
                                else
                                begin
                                        counter <= 0;
                                        firstTime <= 0;
                                end
                        end
                end
        end

        always @(posedge clk)
        begin : proc_output_reg
                if(rst)
                begin
                        dataOut_Valid <= 0;
                        currentRefPoint <= 0;
                        currentDataPoint <= 0;
                end
                else
                begin
                        dataOut_Valid <= (wr_en && start && ~
                           firstTime);
                        currentRefPoint <= mem[counter];
                        currentDataPoint <= dataIn;
                end
        end

endmodule
```

A.10 distanceCalcAcc.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
```

```
//
// Create Date: 04/20/2016 09:50:48 AM
// Design Name: Distance Calculator
// Module Name: distanceCalculationAccumulator
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module distanceCalculationAccumulator #(
        parameter DATA_WIDTH = 32,
        parameter DIMENSIONS = 32,
        parameter VAL_WIDTH = 32
) (
        input clk,
        input reset,
        input wr_en,
        input dataIn_Valid,
        input done,
        input signed [DATA_WIDTH-1:0] data1,
        input signed [DATA_WIDTH-1:0] data2,
        output reg [VAL_WIDTH-1:0] distance,
        output reg distanceValid
);

localparam dif_width = DATA_WIDTH+1;
localparam sqr_width = dif_width*2;
localparam acc_width = sqr_width+DIMENSIONS;

        reg [dif_width-1:0] difference;
        reg [sqr_width-1:0] squared;
        reg [acc_width-1:0] accumulator;
        reg diff_stage_valid;
        reg sqr_stage_valid;
```

```
reg acc_stage_valid;
integer i;

always @(posedge clk)
begin : proc_datapath
        if (reset)
        begin
                difference <= 0;
                squared <= 0;
                accumulator <= 0;
                distance <= 0;
        end
        else if (wr_en)
        begin
                if(data1 > data2)
                begin
                        difference <= data1 - data2;
                end
                else
                begin
                        difference <= data2 - data1;
                end
                squared <= difference * difference;
                if (sqr_stage_valid | acc_stage_valid)
                begin
                        if (i >= DIMENSIONS-1)
                        begin
                                accumulator <= squared;
                                distance <= accumulator;
                        end
                        else
                        begin
                                accumulator <= accumulator
                                    + squared;
                        end
                end
        end
end

always @(posedge clk)
begin : proc_controlpath
        if (reset)
```

```
                begin
                        diff_stage_valid <= 1'b0;
                        sqr_stage_valid <= 1'b0;
                        acc_stage_valid <= 1'b0;
                        i <= -1;
                        distanceValid <= 0;
                end
                else if (wr_en)
                begin
                        diff_stage_valid <= dataIn_Valid;
                        sqr_stage_valid <= diff_stage_valid;
                        acc_stage_valid <= sqr_stage_valid;
                        if (sqr_stage_valid | acc_stage_valid)
                        begin
                                if (i >= DIMENSIONS-1)
                                begin
                                        i <= 0;
                                        distanceValid <= 1;
                                end
                                else
                                begin
                                        i <= i + 1;
                                        distanceValid <= 0;
                                end
                        end
                        else
                        begin
                                distanceValid <= 0;
                        end
                end
        end

endmodule
```

A.11 kSortingP1.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 09/14/2016 12:50:48 PM
```

```
// Design Name: Sorting Phase 1
// Module Name: kSortingP1
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////

// Phase 1 Sorting

module kSortingP1 #(
        parameter DATA_WIDTH = 32,
        parameter DIMENSIONS = 32,
        parameter VAL_WIDTH = 32,
        parameter INSTANCE = 0,
        parameter PASS_THOO_DEBUG = 0,
        parameter K = 1
) (
        input clk,
        input reset,
        input valid,
        input done,
        input outEn,
        input [31:0] start_index,
        input [VAL_WIDTH-1:0] dataValueIn,
        output [31:0] dataNameOut,
        output [VAL_WIDTH-1:0] dataValueOut
);

        reg [DATA_WIDTH-1:0] nameMem [K-1:0];
        reg [VAL_WIDTH-1:0] valueMem [K-1:0];
        wire [K-1:0] comparator;
        reg [31:0] outputPointer;
        reg [31:0] entryId;
```

```
generate
        genvar i;
        for (i = K-1; i >= 0; i = i - 1)
        begin:memory
                if (i > 0)
                begin
                        always @(posedge clk)
                        begin
                                if (reset)
                                begin
                                        nameMem[i] <= 32'
                                            hFFFFFFFF; //
                                             all 1s
                                        valueMem[i] <= {
                                            VAL_WIDTH{1'b1
                                            }}; //all 1s
                                end
                                else if (valid &
                                    comparator[i] &
                                    comparator[i-1]) //
                                    shift
                                begin
                                        nameMem[i] <=
                                            nameMem[i-1];
                                        valueMem[i] <=
                                            valueMem[i-1];
                                end
                                else if (valid &
                                    comparator[i] & ~
                                    comparator[i-1]) // put
                                     new
                                begin
                                        nameMem[i] <=
                                            entryId;
                                        valueMem[i] <=
                                            dataValueIn;
                                end
                        end
                end
                else if (i <= 0)
                begin
                        always @(posedge clk)
```

```
                        begin
                                if (reset)
                                begin
                                        nameMem[i] <= 32'
                                           hFFFFFFFF; //
                                            all 1s
                                        valueMem[i] <= {
                                           VAL_WIDTH{1'b1
                                           }}; //all 1s
                                end
                                else if (valid &
                                   comparator[i])
                                begin
                                        nameMem[i] <=
                                           entryId;
                                        valueMem[i] <=
                                           dataValueIn;
                                end
                        end
                end

        end
endgenerate

// Comparators
generate
        genvar j;
        for (j = 0; j < K; j = j + 1)
        begin:comparing
                assign comparator[j] = valueMem[j] >=
                   dataValueIn ? 1 : 0;
        end
endgenerate


always @(posedge clk)
begin
        //Outputing stuff
        if (reset)
        begin
                outputPointer <= 0;
        end
```

```
                else if (done && outEn && outputPointer < K-1)
                begin
                        outputPointer <= outputPointer + 1;
                end
        end

        always @(posedge clk)
        begin
                // internal ID tag generation
                if (reset)
                begin
                        case(INSTANCE)
                                0: entryId <= 0;
                                1: entryId <= 1;
                                2: entryId <= start_index;
                                3: entryId <= start_index + 1;
                                default: entryId <= 0;
                        endcase
                end
                else if (valid)
                begin
                        entryId <= entryId + 2;
                end
        end

        // Debug junk
        generate
                if(PASS_THOO_DEBUG)
                begin
                        assign dataNameOut = entryId;
                        assign dataValueOut = dataValueIn;
                end
                else
                begin
                        assign dataNameOut = nameMem[outputPointer
                            ];
                        assign dataValueOut = valueMem[
                            outputPointer];
                end
        endgenerate

endmodule
```

A.12 kSortingP2.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dakota Koelling
// Engineer: Dakota Koelling
//
// Create Date: 09/14/2016 12:50:48 PM
// Design Name: Sorting Phase 2
// Module Name: kSortingP2
// Project Name: KNN Hardware Accelerator
// Target Devices: Zedboard, Zybo
// Tool Versions: Vivado 2016.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Phase 2 Sorting

module kSortingP2 #(
        parameter DATA_WIDTH = 32,
        parameter DIMENSIONS = 32,
        parameter VAL_WIDTH = 32,
        parameter PASS_THOO_DEBUG = 0,
        parameter K = 1
) (
        input clk,
        input reset,
        input valid,
        input done,
        input outEn,
        input [31:0] dataNameIn,
        input [VAL_WIDTH-1:0] dataValueIn,
        output reg transferDone,
        output [31:0] dataNameOut,
        output [DATA_WIDTH-1:0] dataValueOut
```

```
);

        reg [DATA_WIDTH-1:0] nameMem [K-1:0];
        reg [VAL_WIDTH-1:0] valueMem [K-1:0];
        wire [K-1:0] comparator;
        reg [31:0] outputPointer;

        generate
                genvar i;
                for (i = K-1; i >= 0; i = i - 1)
                begin:memory
                        if (i > 0)
                        begin
                                always @(posedge clk)
                                begin
                                        if (reset)
                                        begin
                                                nameMem[i] <= 32'
                                                    hFFFFFFFF; //
                                                     all 1s
                                                valueMem[i] <= {
                                                    VAL_WIDTH{1'b1
                                                    }}; //all 1s
                                        end
                                        else if (valid &
                                            comparator[i] &
                                            comparator[i-1]) //
                                            shift
                                        begin
                                                nameMem[i] <=
                                                    nameMem[i-1];
                                                valueMem[i] <=
                                                    valueMem[i-1];
                                        end
                                        else if (valid &
                                            comparator[i] & ~
                                            comparator[i-1]) // put
                                             new
                                        begin
                                                nameMem[i] <=
                                                    dataNameIn;
```

```
                                        valueMem[i] <=
                                            dataValueIn;
                                end
                        end
                end
                else if (i <= 0)
                begin
                        always @(posedge clk)
                        begin
                                if (reset)
                                begin
                                        nameMem[i] <= 32'
                                            hFFFFFFFF; //
                                             all 1s
                                        valueMem[i] <= {
                                            VAL_WIDTH{1'b1
                                            }}; //all 1s
                                end
                                else if (valid &
                                    comparator[i])
                                begin
                                        nameMem[i] <=
                                            dataNameIn;
                                        valueMem[i] <=
                                            dataValueIn;
                                end
                        end
                end

        end
endgenerate

// Comparators
generate
        genvar j;
        for (j = 0; j < K; j = j + 1)
        begin:comparing
                assign comparator[j] = valueMem[j] >=
                    dataValueIn ? 1 : 0;
        end
endgenerate
```

```
        always @(posedge clk)
        begin
                //Outputing stuff
                if (reset)
                begin
                        outputPointer <= 0;
                        transferDone <= 0;
                end
                else if (done && outEn)
                begin
                        if (outputPointer < K-1)
                        begin
                                outputPointer <= outputPointer +
                                   1;
                        end
                        else
                        begin
                                transferDone <= 1;
                        end
                end
        end

        // Debug junk
        generate
                if(PASS_THOO_DEBUG)
                begin
                        assign dataNameOut = dataNameIn;
                        assign dataValueOut = dataValueIn;
                end
                else
                begin
                        assign dataNameOut = nameMem[outputPointer
                           ];
                        assign dataValueOut = valueMem[
                           outputPointer];
                end
        endgenerate

endmodule
```

APPENDIX B

SOFTWARE

B.1 main.cpp

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include "xil_io.h"
#include "xparameters.h"
#include "xaxidma_hw.h"
#include "KNN_accelerator.h"
#include "platform.h"
#include "DataSet.h"
using namespace std;

int main() {

        init_platform();
        print("Now calculating...\n\r");

        u32 RX_BUFFER_BASE = 0x00500000;
        u32 *myOutputArray = (u32 *) RX_BUFFER_BASE;
        UINTPTR BuffInAddr = (UINTPTR)myIntDataSet;
        UINTPTR BuffOutAddr = (UINTPTR)myOutputArray;
        u32 Length1 = NUM_FEATURES * (((NUM_POINTS-1)/2)+1) * 4;
        u32 Length2 = NUM_FEATURES * ((NUM_POINTS-1)/2+1) * 4;

        printf("Input Array Address: %p\n\r", myIntDataSet);
        printf("Output Array Address: %i\n\r", (int)RX_BUFFER_BASE
            );

        KNN_ACCELERATOR_mWriteReg(
            XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
            KNN_ACCELERATOR_S00_AXI_SLV_REG0_OFFSET, 1); // go into
              reset
```

```
KNN_ACCELERATOR_mWriteReg(
    XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
    KNN_ACCELERATOR_S00_AXI_SLV_REG1_OFFSET, (NUM_POINTS-4)
    ); // second starting point

// Setup DMA
//Reset
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_RX_OFFSET + XAXIDMA_CR_OFFSET,
    XAXIDMA_CR_RESET_MASK);
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET,
    XAXIDMA_CR_RESET_MASK);
XAxiDma_WriteReg(XPAR_AXI_DMA_1_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET,
    XAXIDMA_CR_RESET_MASK);
while((XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_RX_OFFSET + XAXIDMA_CR_OFFSET) &
    XAXIDMA_CR_RESET_MASK) == 1) {}
while((XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET) &
    XAXIDMA_CR_RESET_MASK) == 1) {}
while((XAxiDma_ReadReg(XPAR_AXI_DMA_1_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET) &
    XAXIDMA_CR_RESET_MASK) == 1) {}

//MM2S 1 Setup
u32 reg = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET);
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET, reg |
    XAXIDMA_CR_RUNSTOP_MASK); // start
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_DESTADDR_OFFSET, BuffInAddr
    ); // addr in 1

//MM2S 2 Setup
reg = XAxiDma_ReadReg(XPAR_AXI_DMA_1_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET);
XAxiDma_WriteReg(XPAR_AXI_DMA_1_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_CR_OFFSET, reg |
    XAXIDMA_CR_RUNSTOP_MASK); // start
```

```
XAxiDma_WriteReg(XPAR_AXI_DMA_1_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_DESTADDR_OFFSET, BuffInAddr
     + Length1); // addr in 2

//S2MM Setup
reg = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_RX_OFFSET + XAXIDMA_CR_OFFSET);
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_RX_OFFSET + XAXIDMA_CR_OFFSET, reg |
    XAXIDMA_CR_RUNSTOP_MASK); // start
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_RX_OFFSET + XAXIDMA_DESTADDR_OFFSET,
    BuffOutAddr); // addr out

// GET START TIME
u32 count1_low = KNN_ACCELERATOR_mReadReg(
    XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
    KNN_ACCELERATOR_S00_AXI_SLV_REG2_OFFSET);
u32 count1_high = KNN_ACCELERATOR_mReadReg(
    XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
    KNN_ACCELERATOR_S00_AXI_SLV_REG3_OFFSET);

KNN_ACCELERATOR_mWriteReg(
    XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
    KNN_ACCELERATOR_S00_AXI_SLV_REG0_OFFSET, 0); // pull
    out of reset

// Activate DMA
//MM2S 1 Activate
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_BUFFLEN_OFFSET, Length1);
    //length

//MM2S 2 Activate
XAxiDma_WriteReg(XPAR_AXI_DMA_1_BASEADDR,
    XAXIDMA_TX_OFFSET + XAXIDMA_BUFFLEN_OFFSET, Length2);
    //length

//S2MM Activate
XAxiDma_WriteReg(XPAR_AXI_DMA_0_BASEADDR,
    XAXIDMA_RX_OFFSET + XAXIDMA_BUFFLEN_OFFSET, K*4); //
    length
```

```
// Setup Complete

//Wait for MM2S to complete
while(XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,
   XAXIDMA_TX_OFFSET + XAXIDMA_SR_OFFSET) != 4098) {}
while(XAxiDma_ReadReg(XPAR_AXI_DMA_1_BASEADDR,
   XAXIDMA_TX_OFFSET + XAXIDMA_SR_OFFSET) != 4098) {}

KNN_ACCELERATOR_mWriteReg(
   XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
   KNN_ACCELERATOR_S00_AXI_SLV_REG0_OFFSET, 2); // tell
   KNN that we are done
while(XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,
   XAXIDMA_RX_OFFSET + XAXIDMA_SR_OFFSET) != 4098) {}

// GET FINISH TIME
u32 count2_low = KNN_ACCELERATOR_mReadReg(
   XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
   KNN_ACCELERATOR_S00_AXI_SLV_REG2_OFFSET);
u32 count2_high = KNN_ACCELERATOR_mReadReg(
   XPAR_KNN_ACCELERATOR_0_S00_AXI_BASEADDR,
   KNN_ACCELERATOR_S00_AXI_SLV_REG3_OFFSET);

u64 count1 = (u64)count1_high*pow((u64)2,32) + (u64)
   count1_low;
u64 count2 = (u64)count2_high*pow((u64)2,32) + (u64)
   count2_low;
u64 diff = count2 - count1;
printf("[FPGA] Time: %4.6f sec\n\r", diff*pow(10,-8));

printf("[FPGA] K returns: \n\r");
u32 results[K] = {48394, 40097, 25701, 3018, 16511, 24116,
    27482, 35514, 21016, 13906, 29928, 40295, 33986,
   47997, 45967, 44834, 37149, 16450, 26481, 43695, 12176,
    34739, 7256, 35663, 10251};
u32 passing = 1;
for (int i = 0; i < K; i++) {
        u32 num = Xil_In32(((UINTPTR)myOutputArray) + (4*i
           ));
        printf("[FPGA] ID: %i \n\r", (int)num);
        if (num != results[i])
```

```
                {
                        passing = 0;
                }
        }
        if(passing)
        {
                printf("[FPGA] PASS \n\r");
        }
        else
        {
                printf("[FPGA] FAIL \n\r");
        }

        return 0;
}
```

APPENDIX C

PREPROCESSING

C.1 channel_data_format.py

```
import re

def data_format(name):
        "convert data from .dat into a cpp file for compile into
           zynq"

        numberOfChannels = 2 # change for different channels

        dataInName = name + '.dat'
        dataOutName = name + '-channels.dat'

        #find number of points
        data_file = open(dataInName, 'r')
        num_points = 0
        for line in data_file.readlines():
                num_points = num_points + 1
        data_file.close()
        inject_point = ((num_points-1)/2)+1
        #print('inject: '+str(inject_point))
        #print('total: '+str(num_points))

        data_file = open(dataInName, 'r')
        data_file_out = open(dataOutName, 'w')

        numbers = []
        full_ref = []
        pointCounter = 0
        num_points = 0
        isRefPoint = True
        for line in data_file.readlines():
                num_points = num_points + 1
                #print(str(num_points))
```

```
numbers.append(re.split('\t', line))
if isRefPoint:
        #print('IF')
        ref = numbers[0]
        isFirst = True
        for num in ref:
                if num[-1:]== '\n': # Check last
                    char
                        num = num[:-1] # Remove
                            last char
                if isFirst: # id number
                        data_file_out.write(num)
                        isFirst = False
                else:
                        data_file_out.write('\t' +
                            str(num))
                        full_ref.append(num)
                        for x in range(1,
                            numberOfChannels):
                                data_file_out.
                                    write('\t' +
                                    str(num))
        data_file_out.write('\n')
        isRefPoint = False
        del numbers[:]
else:
        #print('ELSE')
        pointCounter += 1
        if pointCounter >= numberOfChannels:
                data_file_out.write(numbers[0][0])
                for col in range(1, len(numbers
                    [0])):
                        for row in range(
                            numberOfChannels):
                                if numbers[row][
                                    col][-1:]== '\n
                                    ': # Check last
                                     char
                                        numbers[
                                            row][
                                            col] =
                                            numbers
```

```
                                                    [row][
                                                    col
                                                    ][:-1]
                                                    #
                                                    Remove
                                                    last
                                                    char
                                            data_file_out.
                                                write('\t' +
                                                str(numbers[row
                                                ][col]))
                                    data_file_out.write('\n')
                                    del numbers[:]
                                    pointCounter = 0
                            if num_points == inject_point:
                                    #print('inject!')
                                    #print(full_ref)
                                    first = True
                                    for x in full_ref:
                                            if first:
                                                    first = False
                                                    data_file_out.
                                                        write('-2\t0\t0
                                                        ')
                                            else:
                                                    data_file_out.
                                                        write('\t0\t0')
                                    data_file_out.write('\n')
                                    first = True
                                    for x in full_ref:
                                            if first:
                                                    first = False
                                                    data_file_out.
                                                        write('-1\t' +
                                                        x + '\t' + x)
                                            else:
                                                    data_file_out.
                                                        write('\t' + x
                                                        + '\t' + x)
                                    data_file_out.write('\n')
        first = True
        for x in full_ref:
```

```
                if first:
                        first = False
                        data_file_out.write('-2\t0\t0')
                else:
                        data_file_out.write('\t0\t0')
        data_file_out.write('\n')

        data_file.close()
        data_file_out.close()
```

C.2 convert_fixed_point.py

```
import re, math

def convert_fp(name):
        "convert data from floating point .dat into int .dat file"

        dataInName = name + '-fp.dat'
        dataOutName = name + '.dat'

        data_file = open(dataInName, 'r')
        data_file_out = open(dataOutName, 'w')

        max_val = 0
        min_val = math.inf
        index = -1
        for line in data_file.readlines():
                line_split = re.split('\t', line)
                isId = True
                isClass = True
                for num in line_split:
                        if num[-1:]== '\n': # Check last char
                                num = num[:-1] # Remove last char
                        if isId: # id number
                                data_file_out.write(str(index))
                                index += 1
                                isId = False
                        elif isClass: # ignore classification
                                isClass = False
                        else: # process number
                                # print(num)
                                myfloat = abs(float(num))
```

```
                                binary = []
                                position = pow(2, 31)
                                intNum = 0
                                for x in range(15,-17,-1): # into
                                    binary
                                        if myfloat > pow(2,x):
                                                myfloat -= pow(2,x
                                                    )
                                                binary.append(1)
                                        else:
                                                binary.append(0)
                                for x in binary: # from binary
                                        if x:
                                                intNum += int(
                                                    position)
                                        position /= 2
                                if myfloat > max_val:
                                        max_val = myfloat
                                if myfloat < min_val:
                                        min_val = myfloat
                                data_file_out.write('\t' + str(
                                    intNum))
                        data_file_out.write('\n')

        data_file.close()
        data_file_out.close()
        print(str(min_val))
        print(str(max_val))
```

C.3 data_code_gen.py

```
import re

def code_gen(K, name):
        "convert data from .dat into a cpp file for compile into
            zynq"

        dataInName = name + '-channels.dat'

        data_file = open(dataInName, 'r')
        h_file = open('DataSet.h', 'w')
        cpp_file = open('DataSet.cpp', 'w')
```

```
        h_file.write('''/*
 * DataSet.h
 *
 *  Created on: Jul 20, 2016
 *      Author: dkoellin
 */

#ifndef SRC_DATASET_H_
#define SRC_DATASET_H_

''')

        cpp_file.write('''/*
 * DataSet.cpp
 *
 *  Created on: Jul 20, 2016
 *      Author: dkoellin
 */

#include "DataSet.h"

const int myIntDataSet[NUM_POINTS][NUM_FEATURES] = {
''')

        num_points = 0
        num_features = 0
        first_start = True
        for line in data_file.readlines():
                num_points = num_points + 1
                nums_inline = re.split('\t', line)
                num_features = 0
                if first_start:
                        cpp_file.write('\t{')
                        first_start = False
                else:
                        cpp_file.write(',\n\t{')
                first_inline = True
                second_inline = True
                for num in nums_inline:
                        if first_inline:
                                #cpp_file.write(num)
```

```
                        first_inline = False
                    elif second_inline:
                        cpp_file.write(num)
                        num_features = num_features + 1
                        second_inline = False
                    else:
                        if num[-1:]== '\n': # Check last
                            char
                                num = num[:-1] # Remove
                                    last char
                        cpp_file.write(', ' + num)
                        num_features = num_features + 1
                cpp_file.write('}')

        h_file.write('#define NUM_FEATURES ' + str(num_features) +
            '\n')
        h_file.write('#define NUM_POINTS ' + str(num_points) + '\n
            ')
        h_file.write('#define K ' + str(K) + '\n')

        h_file.write( '''
extern const int myIntDataSet[NUM_POINTS][NUM_FEATURES];

#endif /* SRC_DATASET_H_ */
''')

        cpp_file.write( '''\n};

''')

        data_file.close()
        h_file.close()
        cpp_file.close()
```

C.4 process_dataset.py

```
from convert_fixed_point import convert_fp
from data_code_gen import code_gen
from channel_data_format import data_format

K = 10
FP = False
```

```
name = 'phy_train'

if FP:
    convert_fp(name)
data_format(name)
code_gen(K, name)
```

C.5 random_data_gen.py

```
# create random data for knn

from random import randint

setNumber = 4
numberPoints = 40
numberDim = 10

data_file = open('training-data' + str(setNumber) + '.dat', 'w')

for i in range(-1, numberPoints):
        for j in range(numberDim):
                num = randint(0,2147483640) # num fit in 32bits
                if i == -1 and j == 0:
                        data_file.write(str(i)+'\t'+str(num))
                else:
                        if j == 0:
                                data_file.write('\n'+str(i)+'\t'+
                                    str(num))
                        else:
                                data_file.write('\t'+str(num))

data_file.close()
```

REFERENCES

[1] An, F., H. J. Mattausch, and T. Koide (2011). An fpga-implemented associative-memory based online learning method.

[2] ARM (2011, October). *AMBA AXI and ACE Protocol Specification*. ARM.

[3] Avnet (2014, January). *ZedBoard (Zynq Evaluation and Development) Hardware User's Guide*. Avnet. Rev. 2.2.

[4] Bailey, T. and A. K. Jain (1978). A note on distance weighted k-nearest neighbor rules. *IEEE Trans. Systems, Man Cybernatics 8*, 311–313.

[5] Betkaoui, B., D. B. Thomas, W. Luk, and N. Przulj (2011). A framework for fpga acceleration of large graph problems: Graphlet counting case study.

[6] Bhatia, N. and Vandana (2010). Survey of nearest neighbor techniques. *CoRR abs/1007.0085*.

[7] Chidananda, K. and G. Krishna (1979). The condensed nearest neighbor rule using the concept of mutual nearest neighbor. *IEEE Trans. Information Theory IT-25*, 488–490.

[8] Cornell (2004). Kdd cup 2004. `http://osmot.cs.cornell.edu/kddcup/index.html`.

[9] Cover, T. M. and P. E. Hart (1967, January). Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory IT-13*, 21–27.

[10] G. Guo, H. Wang, D. B. Knn model based approach in classification.

[11] Gates, G. W. Reduced nearest neighbor rule.

[12] Ghasemi, E. (2015). A scalable heterogeneous dataflow architecture for big data analytics using fpgas.

[13] H. Parvin, H. A. and B. Minaei (2008). Modified k nearest neighbor. *Proceedings of the world congress on Engg. and computer science*.

[14] Hussain, H. M., K. Benkrid, and H. Seker (2015). Dynamic partial reconfiguration implementation of the svm/knn multi-classifier on fpga for bioinformatics application.

[15] Imandoust, S. B. and M. Bolandraftar (2013). Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background.

[16] Johnson, J. (2014, August). Creating a custom ip block in vivado. http://www.fpgadeveloper.com/. http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html.

[17] Koelling, D. (2016). Thesis. https://github.com/Dakota01011/Thesis.

[18] Leopold, G. (2016, October). Intels fpgas target datacenters, networking. http://www.enterprisetech.com/. http://www.enterprisetech.com/2016/10/05/intel-latest-fpgas-target-datacenters-networking/.

[19] Mcname, J. Fast nearest neighbor algorithm based on principal axis search tree.

[20] Neshatpour, K., M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun (2015). Energy-efficient acceleration of big data analytics applications using fpgas.

[21] Omohundro, S. N. (1989). Five ball tree construction algorithms. *Technical Report*.

[22] Pu, Y., J. Peng, L. Huang, and C. John (2015). An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl.

[23] Q. B. Gao, Z. Z. W. (2007). Center based nearest neighbor class. *Pattern Recognition*, 346–349.

[24] S. C. Bagui, S. Bagui, K. P. (2003). Breast cancer detection using nearest neighbor classification rules. *Pattern Recognition 36*, 25–34.

[25] S. Z Li, K. L. C. (2000). Performance evaluation of the nfl method in image classification and retrieval. *IEEE Trans On Pattern Analysis and Machine Intelligence 22*.

[26] Saldana-Gonzalez, G. and M. Arias-Estrada (2009). Fpga based acceleration for image processing applications.

[27] Schumacher, T., R. Meiche, P. Kaufmann, E. Lubbers, C. Plessl, and M. Platzner. A hardware accelerator for k-th nearest neighbor thinning.

[28] Sproull, R. F. Refinements to nearest neighbor searching.

[29] Sutton, O. (2012, feb). Introduction to k nearest neighbour classification and condensed nearest neighbour data reduction.

[30] W. Zheng, L. Zhao, C. Z. (2004). Locally nearest neighbor classifier for pattern classification. *Pattern Recognition*, 1307–1309.

[31] Xilinx (2016, January). *Zynq-7000 All Programmable SoC Overview*. Xilinx. Rev. 1.9.

[32] Y. C. Liaw, M. L. L. Fast exact k nearest neighbor search using orthogonal search tree.

[33] Y. Zeng, Y. Yang, L. Z. (2009). Pseudo nearest neighbor rule for pattern recognition. *Expert Systems with Applications 36*, 3587–3595.

[34] Y. Zhou, C. Z. (2004). Tunable nearest neighbor classifier. *DAGM LNCS 3175*, 204–211.

[35] Yeh, Y.-J., H.-Y. Li, W.-J. Hwang, and C.-Y. Fang (2007). Fpga implementation of knn classifier based on wavelet transform and partial distance search.

[36] Yong, Z. (2009). An improved knn text classification algorithm based on clustering. *Journal of Computers 4*, 3.

VITA

Dakota J. Koelling began his study at The University of Texas at Dallas in Fall 2012 and graduated in Spring of 2016 with a BSCE. He started work on his Masters Thesis in Spring of 2016. By the completion of his MSCE degree requirements in Fall 2016 and multiple internships at Freescale and Cirrus Logic, he had learned much about the field of FPGAs.