Implementation of Machine Learning on an Innovative Embedded Processor for IoT

by

Parampal Singh Gill

B.Tech, Punjab Technical University, 2014

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Science

SIMON FRASER UNIVERSITY
Spring 2017

Approval

Name:	**Parampal Singh Gill**
Degree:	**Master of Engineering**
Title:	***Implementation of Machine Learning on an Innovative Embedded Processor for IoT***
Examining Committee:	**Chair:** Dr. Andrew Rawicz Professor

Lakshman One
Senior Supervisor
Senior Lecturer _____

Fabio Campi
External Examiner _____

Date Defended/Approved: December 19th, 2016

ABSTRACT

The "Internet of Things" (IoT) is a very rapidly increasing market segment for electronics, and it holds the promise to be one of the most significant drivers for innovation in the semiconductor industry in the near future. "IoT" is providing new and different specifications to the design of embedded systems, and such specifications are likely to change the constraints that drive embedded systems design. In particular, "IoT" is introducing a wave of innovation on the design of embedded microprocessors that are the heart and soul of such systems.

This report took place in the context of larger investigation on innovative embedded processor architectures for "IoT". The work started from an existing processor design, developed in Simon Fraser University in form of a Hardware Description Language (HDL) open source library. Such processor design advantages on the RISC-V instruction set distributed since 2011 by the University of California at Berkley. This work focused on analyzing a reference algorithmic application of relevance for the "IoT" (Linear Discriminant Analysis, a well-known Machine Learning tool for data classification), that is currently being utilized in two different research projects in Simon Fraser University. This report contributed to the larger project by:

1) *Porting a C version of the LDA algorithm developed for ARM cores on the newly proposed processor architecture*
2) *Evaluating the performance of the LDA algorithm on the proposed architecture in terms of available data sample rates and required energy consumption*
3) *Profiling the LDA algorithm on the proposed processor in order to determine the critical operation kernels that mostly affect the performance*
4) *Defining the hardware configuration for the proposed processor that leads to the most efficient implementation of LDA*

The work demonstrated that

1) *Floating-point computation is necessary for enabling the expected precision in LDA classification.*
2) *The critical kernel of the LDA classification reside in the Multiply-Accumulation operator. The critical kernel of LDA Training resides in the calculation of matrix determinant, and the availability of a floating point division is critical for an efficient implementation*
3) *Given the constraints imposed by the IoT context, floating point emulation appears too costly in terms of memory occupation and low data rates, so an hardware implementation of Floating point operators (Add, Sub, Compare, Mul, Multiply-Accumulation and Division*
4) *With the chosen hardware configuration, the proposed processor occupies XX mm2 on CMOS 045 nm technology and dissipates on average mW*
5) *With the chosen software configuration, the LDA Training may process up to XX samples/s in the training phase and YY samples/s in the classification phase, while requiring a consumption of XX Joules per sample in Training and YY Joules per sample in classification on real life applications.*

Acknowledgements

I thank my supervisor, Fabio Campi, who gave me this amazing opportunity to work with him and enhance my knowledge. He supported me with great patience throughout the work. In addition, I thank my senior supervisor, Lakshman One, who came forward to represent me and was always ready to help me in any possible way. I also thank Professor Andrew Rawicz as a chair of my supervisory committee.

A special thanks and appreciation for my parents, who have supported me not only during the project work, but throughout my entire life, my grandparents, who have always inspired me and motivated me two work hard, and a big thanks to my brother for always guiding me. Thank you.

Table of Contents

List of Figures

List of Tables

List of Acronyms

SoC	System on Chip
RFID	Radio Frequency Identification
CB	Crescent Beach
ISA	Instruction Set Architecture
ALU	Arithmetic and Logic Unit
OS	Operating System
LDA	Linear Disc ruminant Analysis
FP	Floating Point
FPU	Floating Point Unit
IC	Integrated Circuit

1. INTRODUCTION

Today's everyday life is based on a variety of electronic devices, from desktop computers, laptops, notebooks, mobile devices such as smartphones and tablets, and many more. A processor is the brain of all these systems. The processor is just one of the many components that make up these electronic devices, but it is without a doubt the heart of each system that determines their performance and usefulness [1].

Intel [11], AMD [12] and ARM [13] are the three leading companies responsible for processors design and manufacturing. Intel Corporation is the world leading chipmaker, based on revenue. It develops integrated circuits and supplies processors to known computer system manufacturers such as Apple Inc., Lenovo, HP, Samsung, Sony, Dell, etc. AMD is the runner-up in making processors and the only true competitor Intel has in personal computing domain. AMD is now more focused on building budget-oriented systems, sand a cost effective alternative to Intel.

The growing popularity of portable electronics has given rise to new class of ARM processors. ARM is well known for design of mobile and power efficient processors. Apple, Samsung, Texas Instruments, etc. all use ARM processors into what is known as system-on-chip (SoC). SoC combine many important components on a computer such as CPU, RAM, ROM etc., on a single chip, which helps in lightweight and compact design of devices. ARM conquers 95% of the market approximately [1] [2] [3].

Internet of Things is the internetworking and connectivity of all the electronic devices that can exchange data and communicate among themselves [5]. It is a system of devices, objects or people that are capable of transferring data over a network automatically [16]. A 'Thing', in Internet of Things, can be an automobile sensor to warn the driver, an integrated band on wrist of a rehabilitation patient to assist him or any other natural and man-made object to which we can assign an IP address [16]. IoT is a major change in the consumer electronics market, and it holds the promise to be as big a change as introduction of ARM into the market of embedded processors [14].

This work is a part of a larger project of investigating suitable processor architectures for the Internet-of-Things [15]. In particular, this MENG project aims at investigating a specific application field that is relevant for IoT applications, the classification of sensor data based on machine learning algorithms, and evaluate perspective processor architectures. We are going to implement Linear Discriminant Analysis algorithm on an innovative embedded processor that can sense and classify collected data for IoT. An IoT processor requires a balance between size, cost, energy consumption, reliability and function. The next section describes IoT, its applications and hardware requirements in detail.

2. Internet of Things (IoT)

When IoT is searched on internet today, there is use of a word “smart” everywhere. For appliances like refrigerator, over or toaster, which are in the market from many years. One may wonder how these devices became smart today. Almost all the manufactured goods include an embedded processor along with user interface, through which these devices can be programmed and controlled. For example, an old toaster mechanically controlled the color of the toast, which is now controlled by the microcontroller inside a toaster. The new toaster completes task more reliably and consistently and can communicate through touch pads. When the devices become smart through integration of embedded processing, the next step is to control the device remotely. For example, controlling the lights at home through a mobile phone [7].

Internet of Things is the connected group of items such as embedded electronic devices, sensors, actuators, etc. which can communicate with each other. In 2013, the Global Standards Initiative on Internet of Things (IoT-GSI) defined the IoT as "the infrastructure of the information society" [4]. With IoT, different sensors connected to the network can sense and collect data, which can sent across the network and processed by a processor elsewhere. The IoT devices can also be controlled remotely and provides improved efficiency, accuracy and economic benefit. Combining IoT with sensors and actuators can help us develop technologies like smart grids, smart homes, intelligent transportation and smart cities. It is estimated that by year 2020, 50 billion devices will be connected to IoT [5].

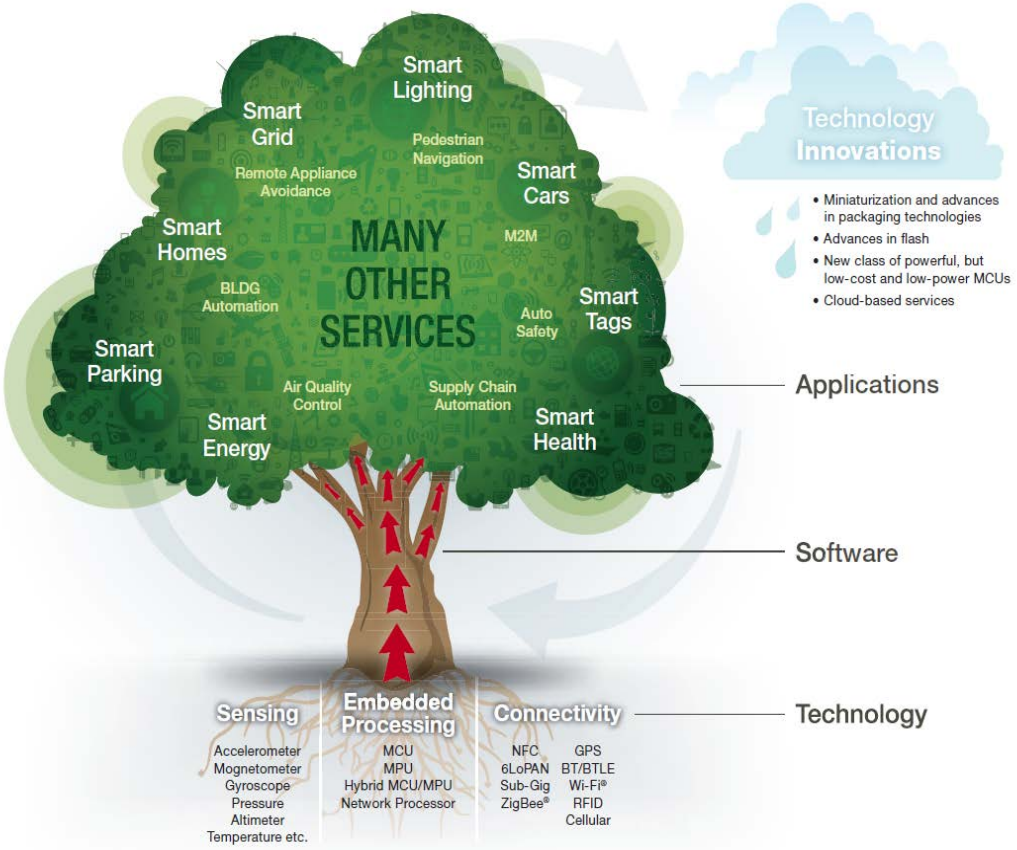

Figure 1: IoT Tree [7]

Figure 1 shows the IoT technology in a tree form. The roots of the tree represent the technologies that drive IoT (sensing, embedded processing and network connectivity). The trunk represents the software, which is next fundamental building block element for developing IoT. The leaves represent the different kinds of IoT systems that can be built.

2.1 Applications of IoT

Today, we have embedded devices and ability to network them, giving IoT application in almost every field such as environment monitoring, infrastructure management, energy management, transportation, health care, automation, and much more. Environmental monitoring application of IoT include air and water quality analysis, earthquake and tsunami warning systems, and safeguard our environment. Integration of IoT in switches, power outlets and other electronic devices can help us to optimize the energy usage. User can also remotely control their devices like powering on or off the heating systems, changing lighting conditions, etc. Smart grid, which is integration of electrical power grids with IoT, will lead to efficient distribution and use of energy [6]. IoT in medical industry can help in monitoring health and general well-being of citizens. Wearable health monitors can also be implemented with IoT to encourage people for healthy living, motivate rehabilitation in patients and much more [5]. Inter and Intra vehicular communication through IoT can help in self-driving cars. Smart traffic control, vehicle control, safety and road assistance is all possible with IoT.

2.2 Building Blocks in IoT Devices

Each IoT system has its own requirements in terms of power, processing time, speed, reliability and cost. IoT system may be as simple as a Bluetooth beacon to as complicated as smart grid application. The building blocks of IoT are described as follow.

1. Sensors: Different IoT systems require different sensors based on the application. The sensor may be a camera for video monitoring, water or gas flow meters, radar vision for safety, radio frequency identification (RFID) for sensing objects or simple thermometer for temperature measurement. Each sensor node in IoT will have a unique ID and can be controlled remotely [7].

2. Embedded Processor: Embedded processor is the brain of IoT. The high volumes of data which is collected by sensor nodes is processed by the processor into useful decisions that can command and control things to make our lives easier safer and reduce our impact on environment. IoT processors provide the local processing capability and for most IoT applications, real time processing is the key requirement. The IoT processors should be small so that the IoT device fits into the area easily. IoT processors classify the data collected by sensors and route it to the cloud. They must also be capable of receiving instructions from the cloud and process it. This project focuses on implementing Machine Learning algorithm, which can sense and classify data [7] [8]. A few requirements of an ideal embedded processor are as follow:

    - Energy Efficiency: It is the most important requirement for embedded processors. Mostly, the IoT system will be located away from an immediate power source and run on a battery. For example, any IoT system required on human body will be powered by battery. Therefore, an IoT processor must operate on low power specifications. To achieve this,

the IC designers must use low-leakage process technologies, low power flash memory technologies, various clocking schemes and the possible ways to make IoT processors energy efficient [7] [8].

- Software Development: While developing the IoT applications, availability of efficient tools, integrated development environment and software stacks is an important consideration. A software development environment that can link together application, command, control and routing processing, security of the IoT system, and easily accessible support is the key for development of embedded processing nodes and IoT applications. The availability of tools such as network analysers, debug capabilities, etc. can accelerate development significantly [7] [9].

- Different Tiers of Devices: IoT consists of a diversity of systems, with different services, different interfaces in heterogeneous environments and different applications that will lead to a different tier of devices. A "one size fits all" approach will not be cost and performance effective and will not satisfy the needs of market [7].

- Cost Effective: The cost of an IoT system is a very important factor. Any developed product is used on a large scale only if it reaches a certain price point. The total cost of the product is the sum of parts of the system and the services required for it. The overall cost must be affordable for the mass adoption of the product. For the companies developing IoT, software reusability and scalability will be a key success factor and advantage existing software investment.

- Quality and Reliability: The average life of electronic gadgets is 2 – 3 years and are then replaced. If the IoT processor were also of same life, it will be difficult to replace them too frequently, depending on their location and application. Therefore, the IoT processors must be of high quality and highly reliable to serve at least 15 – 20 years. The remote data processing on a supercomputer node is also an option to increase the life of local nodes but there is still a balance between remote and local data processing is needed [7].

- Security: Researches in IoT have concluded that they are prone to frightening vulnerabilities. Hackers can hack and monitor live feeds, change camera settings and control the monitor. It was also proven that Internet connected cars can be hacked and taken control of including unlocking doors and even shutting down car in motion. Wearable IoT are also a threat to a person's privacy. The motion sensors on wearable devices can be hacked and to steal information that a person is typing, or track the routine of a person and pose a threat to him and much more. The silver lining is that for the local embedded processing at physical layer, there are a variety of cryptographic engines and security accelerators which support data encryption and authentication [7] [10].

3. Communication Nodes: The communication nodes may be wired or wireless. The role of communication nodes is to transfer the information processed by local processor to different remote destinations. Data processing takes place at remote destinations and new commands

to complete the task are generated which are transferred back to the processor by communication nodes [7].

Different IoT application vary drastically. For example, in a simple IoT application of sensing if a fridge door is left open, the processor will process the data collected by the sensor based on energy use and make a decision after analysing data and command a mechanical arm to close the door or switch on an alarm to let the home owners know.

The IoT will be in every aspect of human life and there is no limit to the distances the data on communication nodes will travel. In 10 years from now, the communication technologies may be completely different from the ones that are considered today, or new and improved revisions of existing standards may come in place. On the other hand, new communication technologies may be available then, which will better suit the IoT applications. The IoT applications are so diverse, big and cost-conscious that there is no single commination technology that can fit into all.

4. Automate Tasks: While there are different IoT segments, one of the important task is to get all the segments talking to each other. This will be achieved by developing and using a lot of software that can enable IoT devices to talk to each other. For example, in a smart meter IoT application, the analog front end, which reads the meter, gives input to the processor and the processor controls the meter. The processor is communicating with the central embedded processing node in the house and with the hub of the locality as well. For a reliable service in this configuration, a lot of middleware software is needed.

5. Remote Embedded Processing Nodes: There is no particular industry-wide standard for IoT today. While some companies claim that the IoT devices will be “dumb nodes” and most of the processing and decision making will be done by the cloud, other say that the processing will be done locally and there will be a minimal access to the cloud [7].

6. Full Security across the Entire Signal Path: instead of addressing just the software security, this topic needs separate attention as a category. The real meaning of security is that the information that is passed on the internet is context and service dependent. For example, knowing the location of a person when he is lost is useful information. However, tracking his location all the time is compromising his privacy. So by secure information, we do not just mean the security from the hackers, it also means security of the person whom the information belongs to. Below are some points to clarify it further.

    - Information available at right time: The information should be available when needed. For example, if a thief enters a house, and the IoT security system alarms the police after few hours or the next day, that information is not useful. The IoT system should process, store and deliver the information when and where it is needed.

    - Information must be accurate: The IoT system should be able to deliver accurate, timely, authentic and complete information. Continuing above example, if the IoT

system is not sure that person entered the house is thief and alerts the police, it is waste of resources and time. Unless the information can be completely trusted, it cannot be used to take any action.

- Information should be confidential: It must be the right of the owner of the information to decide who can access the data. There must be mechanisms in place to ensure the confidentiality of the information.

2.3 Power Consumption of IoT Devices

In the beginning of the processor era, from 1970s, there was no concern regarding power consumption in devices. The only attention was the computational speed and clock speed. The industries considered power consumption only when hand-held devices came into existence. During 1990s, power consumption was one of the important requirements of hardware design for hand held devices [8].

Due to scaling of nanometer CMOS technologies, reliability has become an increasing challenge. In CMOS technology, nodes with channel length < 65 nm, most of the causes for IC malfunctioning are due to power consumption in chip area [8]. The effects of power consumption and low power design techniques are discussed in [8]. Our main purpose here is to minimize power consumption without compromising speed and with little impact as possible on the area of the processor.

2.4 Box Level View of IoT / IoT Protocol Stack

The IoT building blocks are represented in the box level view in Figure 2. IoT system can be described in three level hierarchy, the lowest level or the edge is IoT sensors. The IoT processor processes data at the lowest level and transfers it to hierarchical gateways, which form the second level of the system, through the communication topologies (PAN/BAN/LAN). The gateways are connected to the third level i.e. the cloud via WAN communication technology. The data will be routed to the server for data analysis as well as actions [7] [9]. The work in this project focuses on the innovative embedded processor at the lowest level of the hierarchy. We have to implement Linear Discriminant Analysis algorithm on the embedded processor for classification of data.

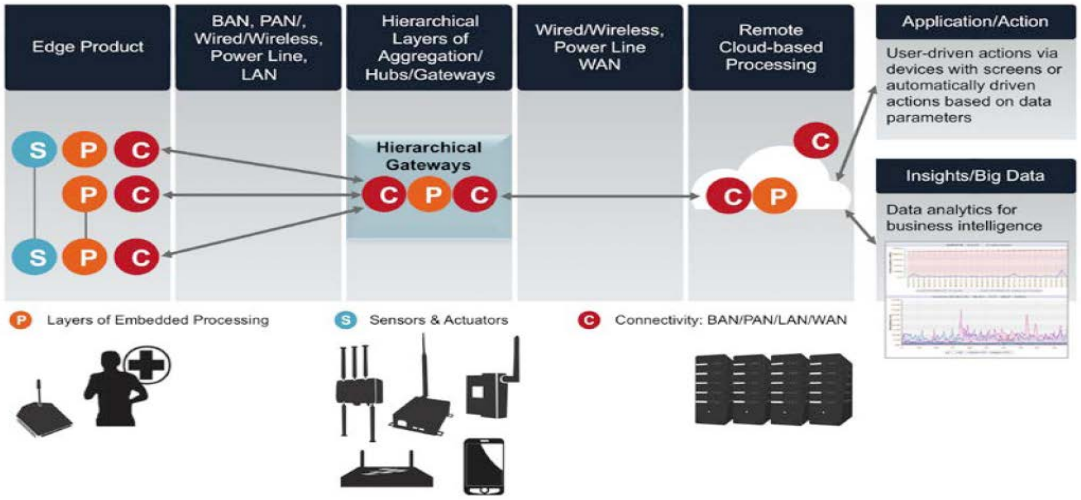

Figure 2: Box level view of building blocks of IoT [7]

3. The Crescent Beach (CB) Processor Architecture

Scope of this MENG work is evaluate the trade-offs related to the implementation of Machine Learning on an existing processor architecture targeted at IoT applications.

This involves both

1) the definition of the most suitable configuration of the processor architecture, that is defined as a configurable HDL suite
2) The choice of the most suitable software implementation of the chosen reference algorithm

This work is part of a larger project aimed at the design of an embedded processor for IoT applications. The processor architecture has the code-name "*Crescent Beach*" (CB). The reference instruction set architecture (ISA) is the well-known RISC-V. RISC-V was developed in Computer Science Division at University of California, Berkeley [18], with a purpose to support research in computer architecture and education, and has become a standard open architecture for industrial implementation under the governance of RISC-V Foundation [17]. It is an open source processor Instruction Set Architecture (ISA) for embedded systems and optionally supports revision 2008 of the IEEE-754 floating-point standard [8] [17]. One of the main goals of RISC-V ISA is it avoids 'over-architecting for an implementation technology like FPGA [17]. In this project, we are using an open source VHDL suite of RISC-V ISA developed as an educational and research tool in Simon Fraser University. The "Crescent Beach" is the release of the same VHDL suite including Floating Point support. Figure 3 below shows the architecture of the CB processor:

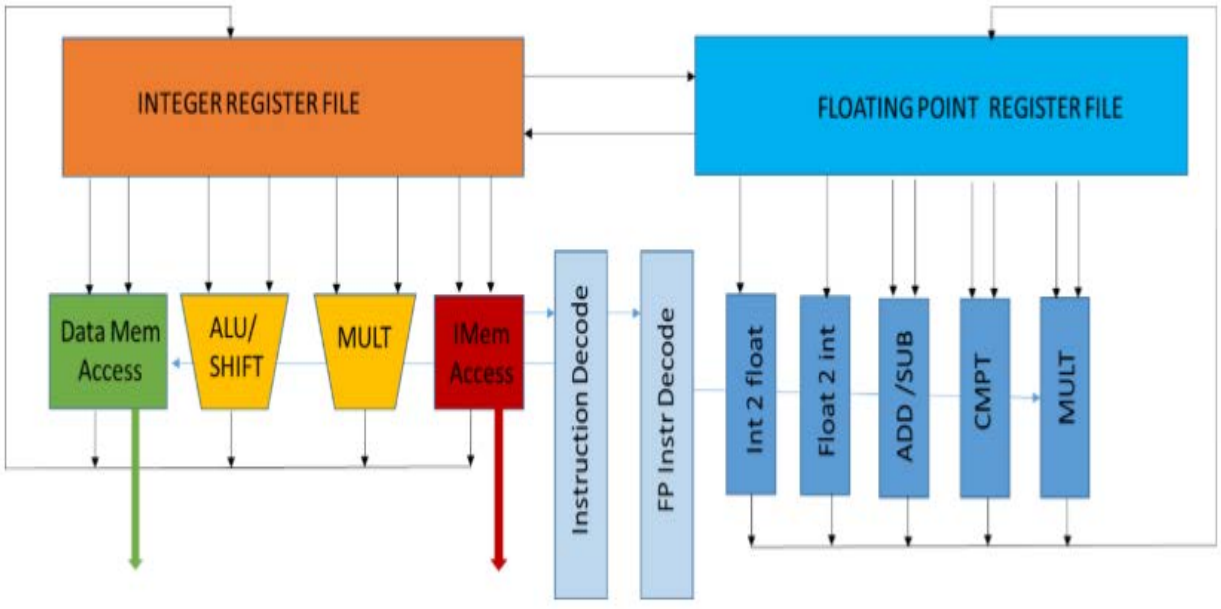

Figure 3: Schematic representation of the open source "Crescent Beach" architecture based on RISC-V instruction set [8]

CB processor incudes a standard RISC configuration with an instruction decode, Arithmetic and Logic Unit (ALU), Multiplier, Integer Register File and Program Counter control logic. On top this, CB features an embedded hardware floating point unit as a coprocessor which has separate floating point instruction decode, floating point register, adder/subtractor, multiplier, divider, square root, convertors (integer to float and float to integer).

4. The Linear Discriminant Analysis Algorithm

In this work, our focus is on the best HW/SW configuration of an IoT processor for data classification. There are various other many other algorithms for data classification, which we could have chosen as a reference. Decision tree, Boosting and Support Vector Machines are examples of some of the other machine learning algorithms that are successfully used in data classification [20].

4.1 Machine Learning and Data Mining

Machine Learning is a branch of Computer Science that lets computer learn automatically instead of being precisely programmed [26]. It evolved from the study of pattern recognition and computational learning theory in artificial intelligence [26]. It involves the development of the programs that can learn, grow and change with their experience on data [27].

Data Mining is another interdisciplinary field in Computer Science, which is a mixture of artificial intelligence, machine learning, statistics and database systems with a goal to extract information from a data set and transform it to a structure for further use [28]. In simple words, it is a process of finding patterns among dozens of fields in large relational databases [29].

Machine Learning is often confused with Data Mining. The process of Machine Learning is similar to Data Mining because both look through data to recognize patterns. But Data Mining focuses on exploratory data analysis and is known as unsupervised learning, whereas Machine Learning uses data to detect patterns and adjust program actions accordingly [26] [27].

There are various Machine Learning algorithms varying with type of application. Some of them are:

a) Artificial Neural Networks: It a machine-learning algorithm inspired from structure and function of biological neural networks. They are used to model the complex relationships between inputs and outputs [26].
b) Deep Learning: It consists of multiple hidden layers in an artificial neural network. This approach tries to replicate the human brain in which it processes light and sound [26]. It is widely used in computer vision and voice recognition.
c) Genetic Algorithms: It is a search technique, which is based on natural selection and uses methods and mutation and crossover to generate a new genotype (a set of parameters that define a proposed solution to the problem).
d) Linear Discriminant Analysis: It a data classification algorithm and a generalization of Fischer's linear discriminant technique to find the linear combination of features (Class variance and between class variance) to separate two or more classes of data. In LDA, a linear function of attributes is created and the class function giving the highest probability represents the predicted class [30].
e) Support Vector Machines: These are a set of supervised learning techniques that are used for regression and data classification [26]. Based on their learning from past samples, these predict the category or class of new sample. SVM tries to find a decision boundary at maximum distance from any sample in the training data [30].

4.2 LDA vs SVMs

Both LDA and SVM are implemented as machine learning algorithms for signal processing, as a part of work done in. In that work, two digital modules were designed for capacitive proximity sensing data applications. LDA and SVM were applied to process the results of a proximity sensor to calculate rough distance and profile information. The results collected by capacitive proximity sensor were extracted to make training and testing data. Thus, for both algorithms, same training and testing data was used [30]. The training data contained 7 classes and 50 data samples in each class. The training data was fed to both LDA and SVM solutions were in MATLAB.

1) LDA can only be used effectively when the distance between classes is significant enough. In [30], the sensor values were small and the difference between different samples even smaller, which led to invalid classification. On the other hand, SVM gave 100% accuracy. LDA has much lower precision than SVM.
2) However, LDA is a very simple and light algorithm, due to this reason it was possible to realize the classification in the FPGA chip. This makes the sensing system more portable and self-contained [30]. On the other hand, SVM is more complex and was realized using PC. This advantage makes LDA suitable for embedded systems.
3) We can easily map LDA on a very small embedded system, with low clock speed, medium size, small memory, no Operating System (OS) and no cache memory, as it would be an IoT processor. This may be very complicated or even impossible with better, more precise algorithms

4.3 LDA Theory and C Language Implementation

Linear Discriminant Analysis (LDA) is one of the oldest machine learning algorithms, first published in 1936 [30]. It was first proposed by Ronald Fisher in his research to solve taxonomic problems. LDA is a data classification algorithm used for dimensionality reduction while maintaining as much separation as possible between classes in reduced dimension [24]. LDA creates linear attributes for each class to be identified and the class function giving the highest probability represents the predicted class [24]. It creates the linear function equal to the number of classes. The term Classes is used for different categories of data. LDA works in such a way that

1) it tries to minimize the within class variance of each class
2) while maximize the separation between the classes

LDA is related to principal component analysis (PCA) and factor analysis; as they both look for linear combinations of variables which best represent the data [31]. LDA models the difference between he classes of data while PCA does not take into account any difference in classes [31].

Figure 4 and Figure 5 below represent a two-dimension example of LDA with two classes. LDA reduces the dimension from 2 to 1, while squeezing the data in such a direction that the separation between classes can be obtained.

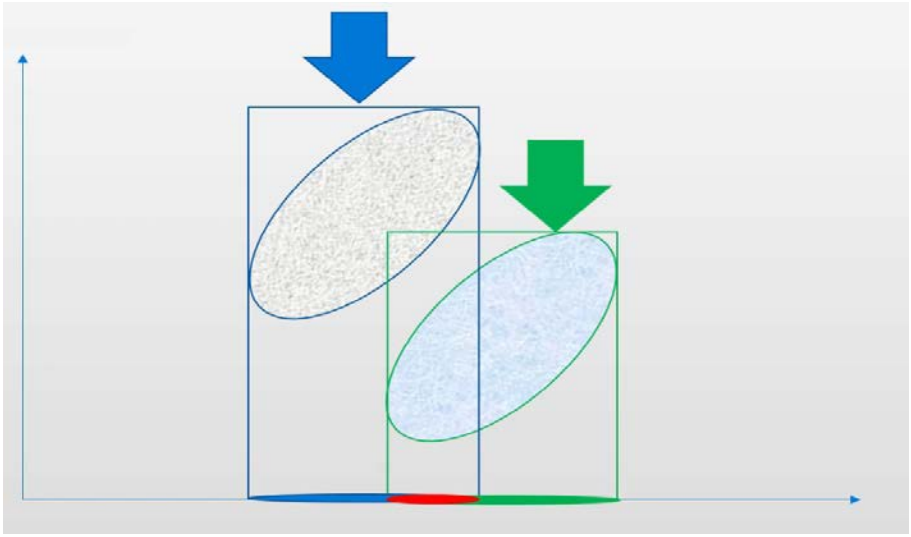

Figure 4: Squeezing data in non-optimal direction [24]

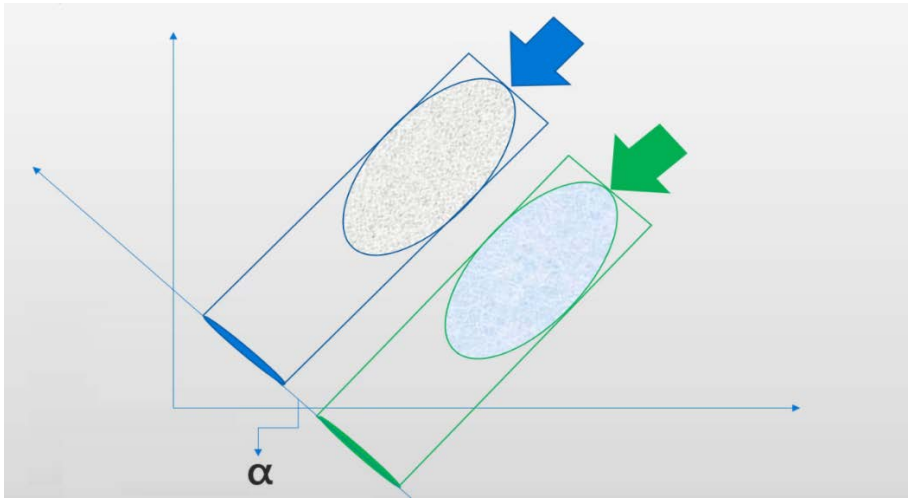

Figure 5: LDA squeezing data in optimal direction while obtaining a clear separation between classes [24]

The C Language implementation of LDA used in this work represent a porting of a MATLAB implementation and was developed in the context of [19].

The algorithm calculates the mean, variance and covariance of each class. Then the covariance of each class is used to calculate pooled covariance matrix. The pooled covariance matric is then inverted and LDA coefficients are calculated [32]. The inverting of pooled covariance matrix involves the calculation of determinant, which takes the most of the computation time of LDA. Figure 6 shows the flow diagram of LDA training algorithm.

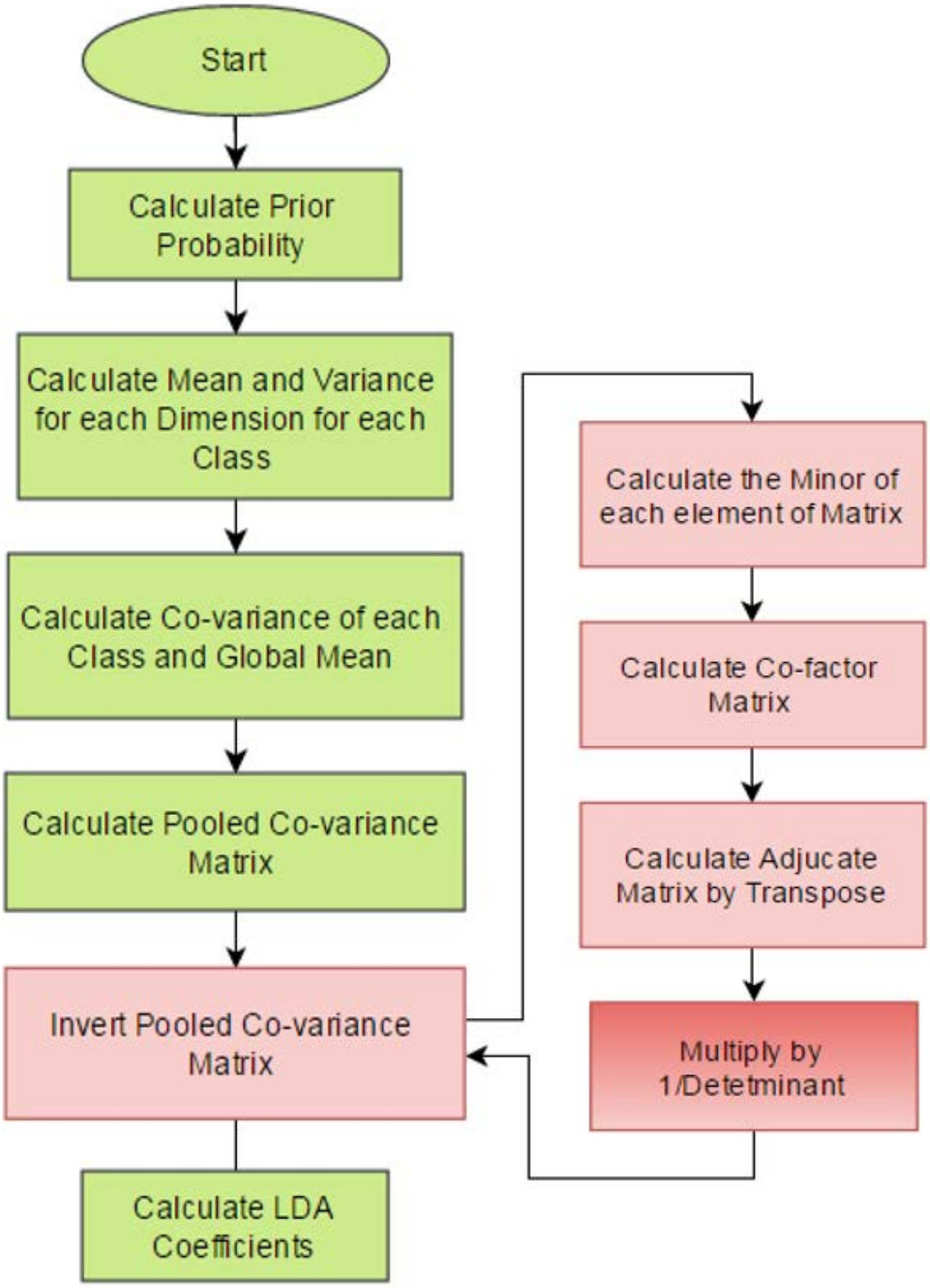

Figure 6: Flow chart of LDA Training algorithm

Our main goal is to classify the data collected by IoT sensors. After the calculation of LDA coefficients, the classification of the data is just matrix multiplication. Figure 7 shows the flow chart for classification of the data.

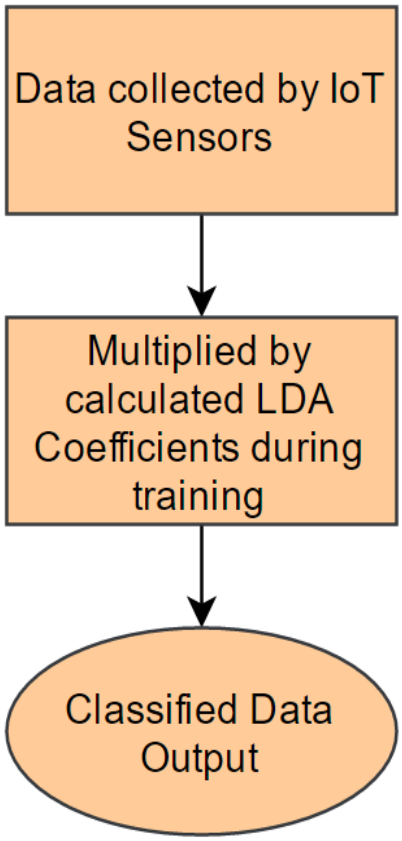

Figure 7: Flow chart of LDA data classification algorithm

5. Work performed in the MENG Project

The scope of this project is:

a) *Evaluate the suitability and performance in the IoT context of the "Crescent Beach (CB)" processor architecture*
b) *Defining the Linear Discriminant Analysis (LDA) machine-learning algorithm as reference application, develop a software environment to measure the performance of the CB processor against the constraints for IoT processor design outlined in section 2*
c) *Define the HW configuration of the CB processor that best implements the LDA algorithm given the constraints and measurements defined above*

The MENG project included the following phases:

1) Acquire the methodologies and tools for writing software on the CB processor, set up a compilation environment, develop all the necessary libraries to develop and debug code on the CB processor model without the support of an operating system. Although this initial setup phase did not produce measurable results, it nevertheless occupied a large portion of the project time.
2) Develop the LDA algorithm in C language on a UNIX workstation, and identify the critical kernels of the code.
3) *In order to maximize performance and minimize hardware cost, the algorithm is run on the processor without the support of an Operating System (OS). The practice of implementing code without an underlying OS is rather common in performance-driven embedded systems and is define "Bare-Metal programming":* A third step in this work was to port it to a version where OS support is not needed, propose different solutions to implement the critical kernel of the LDA code and evaluate the most suitable software configuration for the code
4) Acquire the HDL model of the CB processor, run the LDA code on different hardware configurations of the same and provide a set of measurement in order to evaluate the LDA performance against the specifications

5.1 Phase 1: The RISCV Processor suite and its Compilation Toolchain

The CB processor is a VHDL hardware implementation produced in 2016 in Simon Fraser University as an open-source, reusable VHDL model based on the processor Instruction Set Architecture (ISA) distributed as open source by the RISCV foundation at university of California, Berkley [18].

The software toolchain for compiling C code on the CB processor is distributed in open source form from [17]. The toolchain is based on the GNU-Gcc compiler [25].

The first task in the MENG work was to acquire the two resources (SW and HW environments) and set up a compilation environment including basic utilities, and tools for loading executable code from the toolchain environment in to the HDL simulation environment (based on the ModelSim software CAD tool

distributed by Mentor Graphics). Such libraries and tools are made available as part of the “Crescent Beach” HDL model package.

The C program codes are written in a Linux environment and the RISCV compiler generates files representing instruction memory (Imem.mem) and data memory (Dmem.mem) image files of the C program, which are then loaded for simulation in Modelsim.

Since there is no I/O in the processor model, except for a standard output debug facility, (the Modelsim HDL simulation generates in every simulation a text file representing the standard output of the processor), input data for the processor computation are compiled as part of the main program.

In the first tests, input data were described as local variables in the main program. Then the coding style evolved to defining input data as global variables in the data memory: this represents is a more realistic representation of an embedded system configuration. In fact, in an IoT system performing ML-based data classification, a sensor would collect relevant data for classification and load such data into an array in the data memory. Then an interrupt would signal to the processor the need to start the data classification process.

5.2 Phase 2: Developing LDA Algorithm

The heart of the LDA training algorithm is the calculation of determinant of a matrix. Therefore, the first significant step in this project was to evaluate the complexity of the matrix determinant calculation and propose a suitable algorithm for its completion. In fact, a large portion of this work was focused on efficient calculation of determinant of a matrix.

The most relevant parameters to be evaluated in the selection of the appropriate matrix determinant implementation are related to the trade-off between the system performance and the relative hardware cost:

1) Necessity of Floating point computation
2) Necessity of specific Floating point operators

In itself, the LDA algorithm needs large data ranges to ensure good precision. So the usage of

The comparison is based on the following three parameters.

a) Number of clock cycles: We will get number of clock cycles from the hardware counter that we are using in each program.
b) Size of Data memory
c) Size of Instruction memory

Floating Point Simulation Problems:

While performing initial test on the simulation platform, I could identify a few interesting problems related to the utilization of floating point numbers while interacting between the compiler toolchain and the HDL simulation environment. The identification and resolution of such issues provided a useful feedback for the development of the processor suite. This will positively affect the work of other individuals, students or professionals that will be re-using the same environment.

Issue 1: When defining a floating point constant, the compilation would complete correctly but the simulation would not terminate and keeps running for infinite time. There are three categories of data in a C Program:

(a) Uninitialized variables (e.g. Int a)

(b) Variables that we do initialize in the program code (e.g. Int a=3)

(c) Constant data, that are specified as read only values that are integral part of the program code (e.g. a=a+1.0)

Compilers tend to save Initialized variables values, both in the case of integer and floating point numbers, ".sdata" (Static Data) section of the data memory, and load such values on the processor as part of the Data Memory image where they can be read or even written during computation. Constant data, on the contrary, being part of the program, cannot be overwritten. As a consequence, compilers save such values on a specific, reserved memory sections flagged as read-only. For reason related to the compiler internal architecture, INTEGER constants are stored in a .rodata (Read-Only Data) section, correctly processed by the environment into the program Image files. But when constants are float, the compiler handles then differently, defining new memory section for such float constant data with names such as ". srodata. cst4". In the compilation toolchain, the. srodata. cts4 was not correctly converted into the memory model files (. mem) because the use of a floating point constant in the code was never performed before on the processor suite.

Issue 2: An important aspect to mention is that the CB processor supports only single precision Floating Point numbers (32-bit floating point numbers) and not double precision, at the scope of limiting the hardware cost of the FP operators. It should be underlined that double precision FP operators would have a very high hardware cost in terms of area and power consumption. Indeed, it is extremely rare to encounter embedded implementation of double-precision floating point operators, that are on the contrary quite common in Desktop processors.

On the other hand, in order to preserve precision, C compilers as a rule interpret float constants in C as double precision floating point numbers, and the RISCV compiler developed at UCB is no exception. This created tricky situation because legal code compiled with success may then create errors when run on the HDL model. Again, this issue was fixed specifying an explicit casting to single precision every floating point constant in the code: Example: x += (float)0.8;

5.2.1 Determinant of a 2nd Order Matrix

The first task in the project was to develop a basic program for determining the determinant of a matrix and we started with 2 by 2 matrix. To analyze the trade-off between performance and hardware complexity we will compare in Table 1Table 1: Comparison of different characteristics of Determinant Program with Int and Float variables different versions of the same calculation.

In case of determinant calculation of second order matrix, the number of cycles given by counter in case of floating point variables was just 1. Comparatively, it was 15 when variables were int. The size of Instruction memory (Imem.mem) was same for both while the size of Data memory (Dmem.mem) was

slightly more in case of floating point. This also proves the fact that floating point multiplication is less complex than int addition.

Characteristic	Determinant2x2 (Integer)	Determinant2x2 (FLOAT)
Number of Cycles (in decimal)	15	23
Data Memory Size (in decimal)	12	14
Instruction Memory Size (in decimal)	152	152

Table 1: Comparison of different characteristics of Determinant Program with Int and Float variables

5.2.2 Determinant of N^{th} Order Matrix

The next step was to calculate determinant of any matrix that can be of N^{th} order. We had two approaches to calculate the determinant of N^{th} order matrix, one through recursion and second through reduction of matrix into upper triangular form and multiplying its diagonal elements.

The main difference between the two approaches is that the reduction of matrix in upper triangular form involves a division while the recursion method involves only multiplication. The CB processor has only floating point division and we had to enable division in the architecture description of the processor for upper triangular approach to work. Of course, it will increase the size, cost and power consumption of the processor and we will discuss that in the next chapter and conclude the best hardware/software configuration of CB processor for data classification.

Another thing to mention is that the upper triangular approach can only work with float point variables because the usage of an integer divider would lead to gross approximations for the range of data in our set of application.

We calculated number of cycles for these three programs with N varying from 2 to 8 i.e. when N is a 2x2 matrix to when N is 8x8 matrix. Table 2: Comparison of different characteristics of determinant programs of varying order summarises the performance results. From the table, we can see the difference between the three programs when the matrix size varies from 2X2 to 8X8. Let us first compare the recursion programs. The determinant Recursion program using integer values is computationally faster than its floating point version, but in a very limited fashion. This demonstrates that the additional complexity of floating point hardware does not significantly affect performance. It will, significantly affect chip area and power consumption as described in section 5. However, although it would be possible to re-write the LDA algorithm in order to utilize only integer numbers (this was not in the scope of this report work) its precision in classification would be significantly low to the point of making it unsuitable for real-life application.

We will discuss the major differences between the Floating Point and Integer number in the hardware section; here the focus is on the computation time and memory. In any case, from the Table 2 it appears evident how the determinant calculated by reducing matrix to upper triangular form is computationally much more efficient, compared to recursion program. As the matrix size increases, the total number of cycles it takes for calculation were very surprising compared to the recursion program. Again, the drawback of such solution is that it involves the processing of Floating Point division, which is costly to implement, and increases the area of the processor and its power consumption. Let us discuss the trade of between the processor hardware that involves (1) Processor without Floating Point (Integer numbers only), (2) Floating Point but no divider and (3) Floating Point with Divider in the next chapter.

Order of Matrix (N)	Characteristic	determinant _Recursion _NxN_INT_GV	determinant _Recursion _NxN_FLOAT_GV	determinant_ UpperTriangular _NxN_FLOAT_GV
2	Number of Cycles	13	60	131
3	Number of Cycles	855	1193	326
4	Number of Cycles	12,795	13,364	733
5	Number of Cycles	270,659	282,213	1330
6	Number of Cycles	7,703,452	7,954,523	2194
7	Number of Cycles	(>1Gcycles)	(>1Gcycles)	3371
8	Number of Cycles	(>1Gcycles)	(>1Gcycles)	4933

Table 2: Comparison of different characteristics of determinant programs of varying order

Drawback of Upper Triangular Approach and Area for Improvement:

As mentioned earlier, the upper triangular approach involves a division to reduce the matrix for calculating determinant. The division is between the diagonal element as numerator and each of the matrix element below the diagonal element (depending on the order of the matrix) as denominator. The problem will arise when any element in the lower triangular form of the matrix is zero and the division will tend to be infinite.

When we will classify the sensor data, the program should work well since it is not reasonable for the output of the sensor to be zero. Therefore, we have no issue with upper triangular Program in our application.

To be more general, we can include a check to avoid this condition in many possible ways, for example

(a) the program will exit with a failure when it encounters a zero in the lower triangular matrix
(b) an additional loop can check that if the lower element is zero, there is no need to reduce it
(c) or if there are many zeros, the program calls the recursion function to calculate the determinant

This would take additional processing time for configuration of inputs that are not likely to happen; therefore, we are just ignoring the condition depending on our sensor that it will not give zero as an input. For any application, where the input might be zero, the upper triangular program should be changed.

5.3 Phase 3: Cost of LDA

The third step in this work was to acquire the LDA code developed in the context of [19], port it to a version where OS support is not needed, propose different solutions to implement the critical kernel of the LDA code and evaluate the most suitable software configuration for the code.

After completing the code, the LDA training cost was calculated. The LDA classification is a simple matrix multiplication, so its cost is essentially that of a multiply-addition FP operation and a few others simple address update operations (equivalent to 20 cycles) multiplied by number of samples to be classified, numbers of classes in LDA and number of dimensions in each sample.

On the contrary, the LDA training cost is more complicated and is a non-linear function of the parameters above. The critical kernel of the training is the determinant function, especially for sample dimensions >>1.

Figure 8 shows the contribution of Matrix determinant n the overall cost of LDA Training.

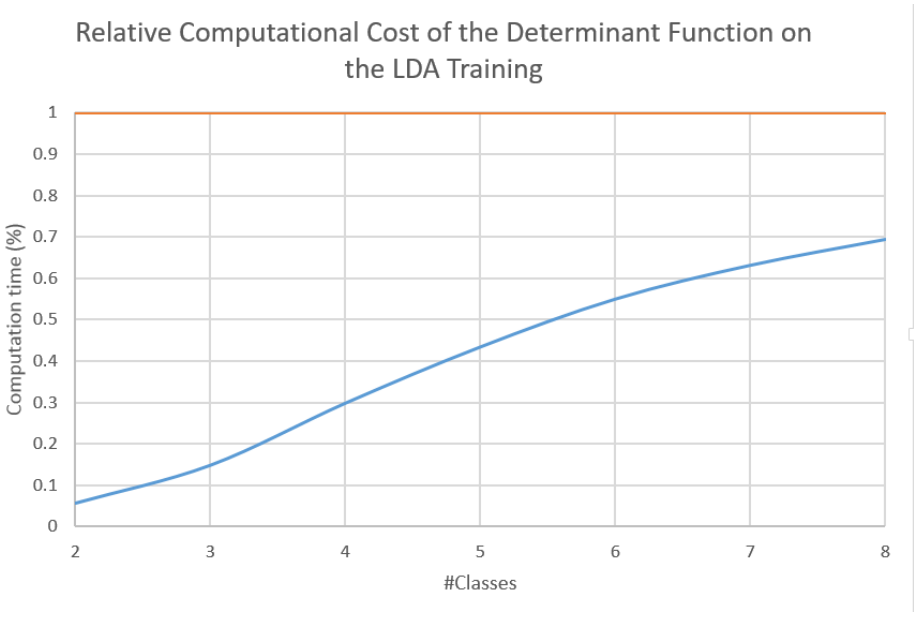

Figure 8: Contribution of the Matrix Determinant operation on the overall cost of LDA Training

Supposing the case of a FP processor running at 250MHz, each cycle is 4ns, so we can process one sample per CLASS per DIMENSION every 80 ns. In simple example, suppose we have 2 classes, and dimension of

each sample is also 2; we can process one sample every (80*2*2) 320 ns. For IoT applications, this level of performance is quite sufficient.

A classic application may have 2 to 10 classes, with dimension of each sample 5 to 10, so in a typical case we may process each sample in (10* 10) * 80ns ~= 8 us

The complexity of training of LDA scales with DIM*DIM, with a fixed DIM has a small dependency on number of samples, while it is almost independent from number of classes. Figure 9 and Figure 10 show the dependency of training cost on number of dimensions and numbers of samples in training data set respectively.

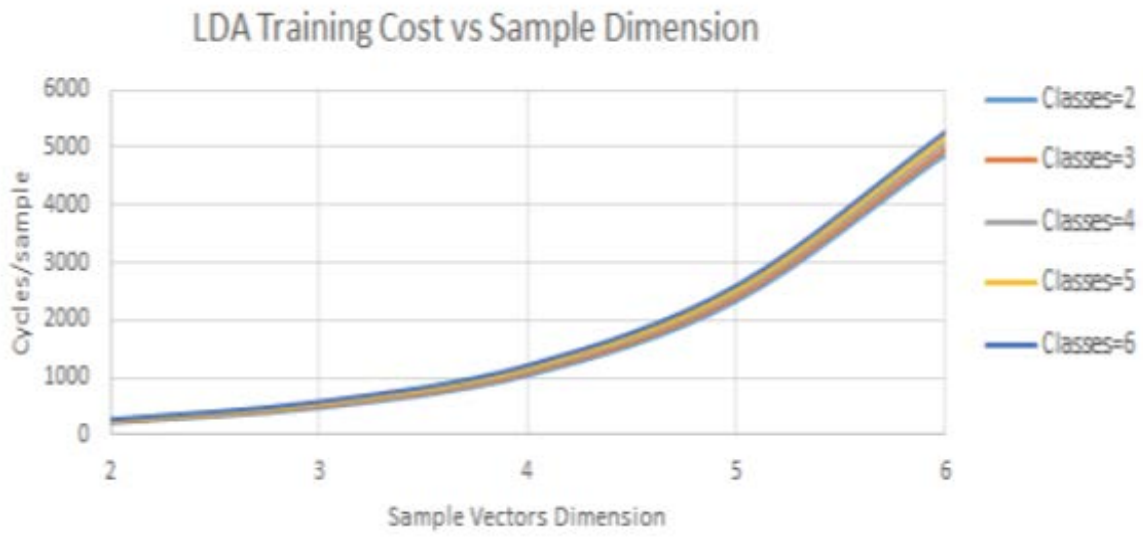

Figure 9: Effect of Number of Dimensions in a sample on LDA training cost

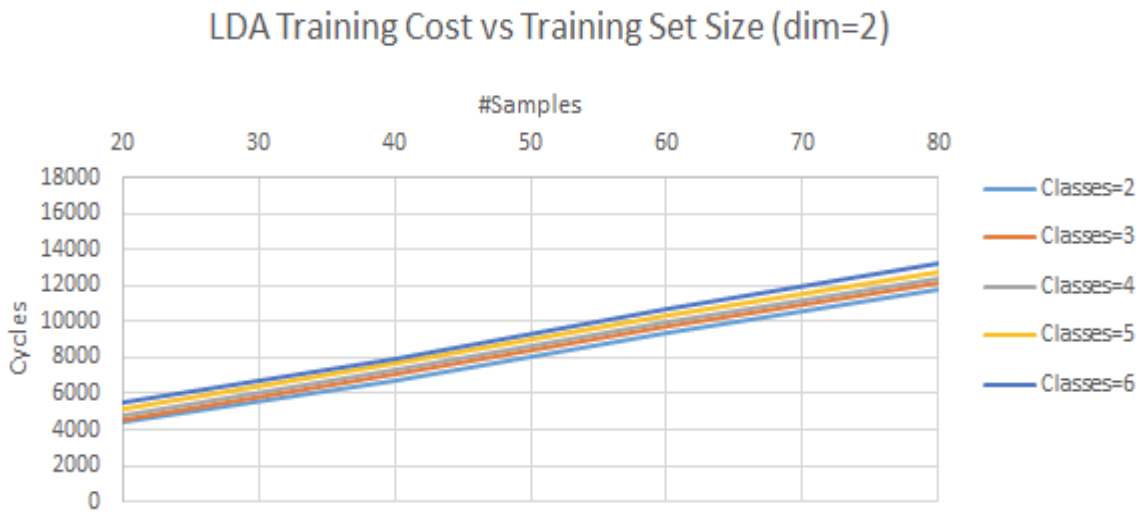

Figure 10: Effect of Number of Samples in Training Data Set on LDA Training Cost

Figure 11 and Figure 12 show the dependency of Classification cost on number of dimensions and numbers of samples in training data set respectively.

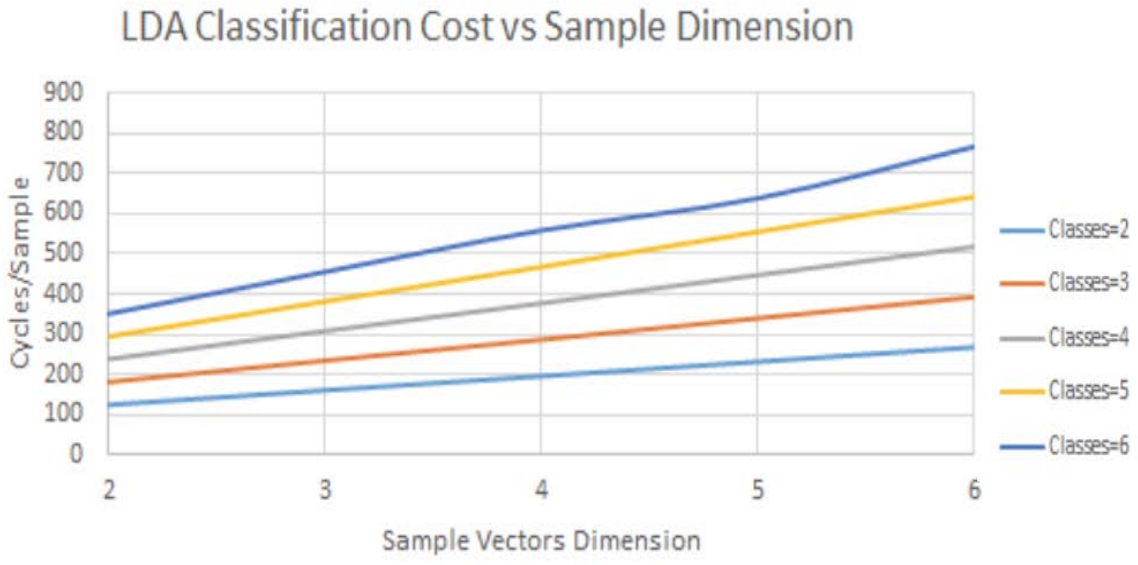

Figure 11: Change in LDA Classification Cost with Number of Samples

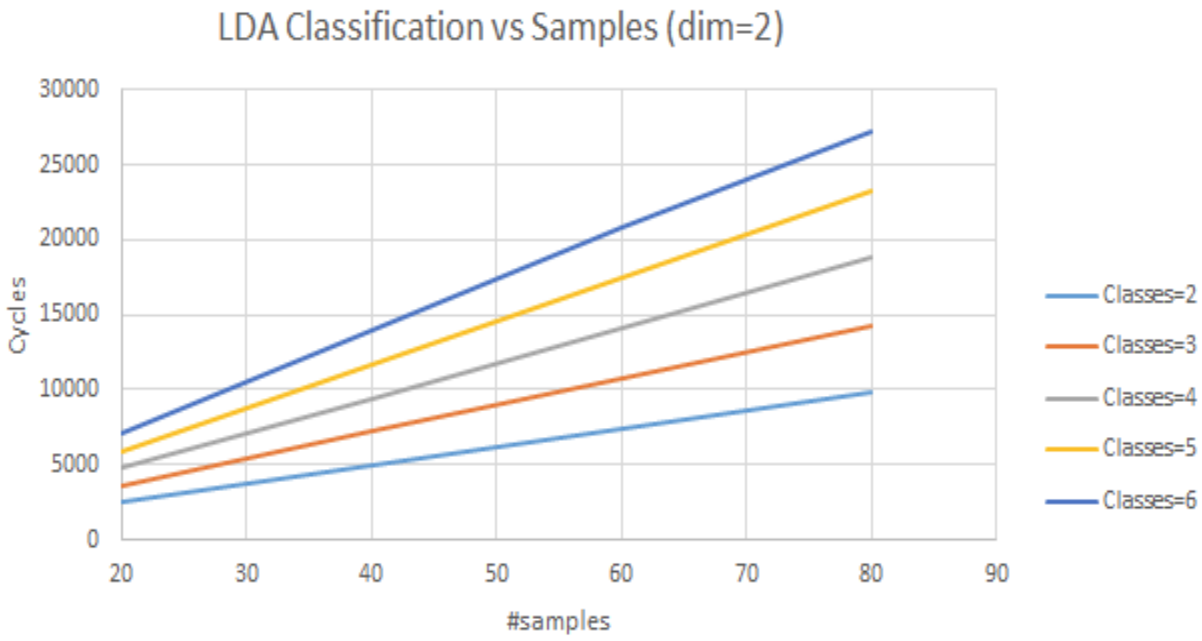

Figure 12: Change in LDA Classification Cost with number of samples

6. Hardware Configuration for CB Processor

The previous chapter provides information on the LDA algorithm, its critical kernels, and its performance on the CB processor depending on the different parameters that affect its computational cost. This chapter applies the notions introduced in the previous in order to define the best configuration of the CB processor that can match the requirements of the algorithm. As a reference test case to define the computational requirements of a "real-life" LDA application we selected two "IoT" application areas, where the implementation of LDA training and classification on an embedded system is being investigated. Both applications are active research fields in Simon Fraser University [19] [30] [35] [36]:

a) Pressure sensors in manufacturing robots for human/machine interaction: When a robot touches something, it should be able to distinguish between the object and human. The pressure sensors collect the data when robot will try to touch something and the processor must be capable of making a decision fast to distinguish between object and human [30] [36].
b) Medical Systems for Rehabilitation: The work done in [19] is the classic example how IoT applications play an important role in medical systems as well. In order to design wearable feedback system, obviously we have to consider the size, power consumption, processing time, etc. for the IoT processor.

In the introductory part of this MENG report, we determined the following parameters for evaluating the suitability of a processing system to the "IoT" domain, in strict order of priority:

(a) Real Time Response
(b) Reliability (ensured by means of power peaks mitigation)
(c) Small size and small energy consumption
(d) Low Economic Cost

In chapter 2 of this report, we introduced how one of the prerequisite for a diffusion of IoT devices is their low cost, which must be obtained without compromising real-time performance. For this reason, reference technology for exploring hardware issues, we chose to use a CMOS technology node that is not too recent as finFET, because it would impose severe manufacturing costs, but still performant enough to safely meet real time constraints: CMOS 45 nm.

All figures that will be described in the following are derived utilizing the FreePDK Technology Kit distributed as open source by NCSU [33] [34].

6.1 Floating Point Vs Integer

In order to work efficiently, LDA needs to work on a large range of data. This range to data cannot be achieved with integer numbers. **Even with normalization, the range of integer numbers tend to saturate the calculation yielding** bad classification results: The main differences between Floating Point and Integer numbers are discussed next.

1) Operations with Floating point numbers are less precise than integers, because FP computations include truncation while integer computations only have overflow

2) Floating point can represent much smaller numbers, the smallest number being 1.18 E-38 while with integer the smaller number is 1

3) Floating point have much higher range than integer numbers. Integer goes up to 2,147,483,647 whereas FP to 3.4E38

Therefore, an IoT processor should have floating point capabilities. Now another point that arises is how to provide the floating point capabilities to the processor. We can have floating point hardware or floating point calculation can be emulated on integer numbers

Every embedded processor is designed according to its requirements and applications. Whenever a floating-point operation has to be performed, the embedded processor has options [21].

1) A software Floating Point emulator
2) A hardware Floating Point Unit (FPU)

A FPU can be integrated or a co-processor, which is just the difference in design and does not matter at this point. Some embedded systems have the FPU as a coprocessor rather than integrated. In cases where floating point hardware is not available, floating point calculation are done in software, which we call as floating point emulation [21].

Below are some key differences between floating point (FP) hardware and floating point emulation.

1) Floating point emulation definitely avoids extra cost of the hardware, but
    a. It would create large instruction memory
    b. Long cycle counts, which would require more power and more processing time
2) If we have a floating point hardware, floating point computation is almost as fast as integer, or only a little bit slower (as demonstrated in Table 2), but we pay a high price in area and power

To clarify the difference between Floating Point emulation and Floating Point Hardware, the Table 3: "SoftFloat" FP Emulation Vs FP Hardware on CB Processor below shows the comparison in performance of CB processor, when it uses Floating Point hardware vs when it uses “SoftFloat” Floating Point emulation library, distributed as open source by the University of California, Berkley.

Moreover, the adoption of the SoftFloat library (including only the single-precision functionality) would require an overhead of 26Kbytes on the program memory, with relative impact on chip area and power consumption (The standard instruction memory of the CB processor is 32Kb, so it would have to be almost doubled)

From Table 3: "SoftFloat" FP Emulation Vs FP Hardware on CB Processor, it can be seen that the average computational cost when using “SoftFloat” Floating Point emulation is 10 to 20 times bigger. In particular, LDA classification is strictly based on matrix multiplication, which is in turn based on repeated Multiply-Accumulation operations. As a consequence, we can expect 21 cycles (11 + 10) from Floating Point hardware and 444 cycles (250 + 194) from Floating Point emulation, giving a degradation in performance by x20 (multiple of 20). Of course, we will save a lot of chip area and peak power consumption if we use Floating Pont Emulation, but that would lead to an unacceptable overhead in response time and energy consumption. The power consumption by the CB processor is discussed in detail in next section; to give a brief example, on the reference case of robot hand shape recognition described in [29], the presence of the FPU allows for completing a training cycle in 600 us, while classification processes a new sample every 1 us (Sample rate, 1MHz).

Parameter	Floating Point Hardware (Number of Cycles)	"SoftFloat" Floating Point Emulation (Number of Cycles)
Convert INT to Float	10	60
Convert Float to INT	7	100
Float Sum / Subtract	11	250
Float Multiply	10	194
Float Division	18	260

Table 3: "SoftFloat" FP Emulation Vs FP Hardware on CB Processor

Using an integer processor and SoftFloat emulation would allow for a processor area that is less than 1/3rd, that becomes ½ considering the program memory overhead and a peak power consumption, that including the additional memory, is roughly comparable. But, this option would demand a decrease of the training time and sampling rate. While a training time of 12 ms would still be acceptable, a decrease of the sample rate to 20 us (sample rate 5KHz) may degrade the recognition of fast object to the point where the system is not able to react appropriately in dangerous conditions.

It can be concluded from the above, that an IoT processor for LDA provides floating point capabilities and that floating point capabilities should be provided by Floating Point hardware.

6.2 Comparison of Different Architecture

By far and large, the LDA training algorithm spends its computational time in the calculation of a Matrix determinant. An efficient calculation of the Matrix Determinant will make an efficient calculation of the training phase. Referring back to section 4.2.3, we investigated two solutions for calculation of Matrix Determinant, one that uses recursion and second that uses upper triangular approach. The recursion approach involves only multiplication and can be used with both Integer numbers and Floating point numbers, while upper triangular approach involves division and can be used with only Floating Point Numbers. Thus, we have three possible architectures of CB processor on which we can implement LDA algorithm. These three options are

a) Processor with Integer (without Floating Point)
b) Processor with FP (but no Divider)
c) Processor with FP and Divider

Table 4: Area of different architectures of CB Processor shows the area covered by different versions of the processor that is utilized in this investigation in a CMOS 045nm technology running at 250 Mhz.

Area of the CB Processor with different Architecture Configurations	
Processor with Integer	0.025 mm^2
Processor with Floating Point (no Divider)	0.058 mm^2
Processor with Floating Point and Divider	0.082 mm^2

Table 4: Area of different architectures of CB Processor

Of course, as we keep adding more functionality to our processor system, its size will grow.

The simplest configuration of the CB processor is its integer configuration: such configuration features 32-registers Register File, Integer Multiplier and Arithmetic and Logic Unit (ALU). Figure 13 below specifies the actual size contribution of each component in mm square and the percentage contribution of each as well.

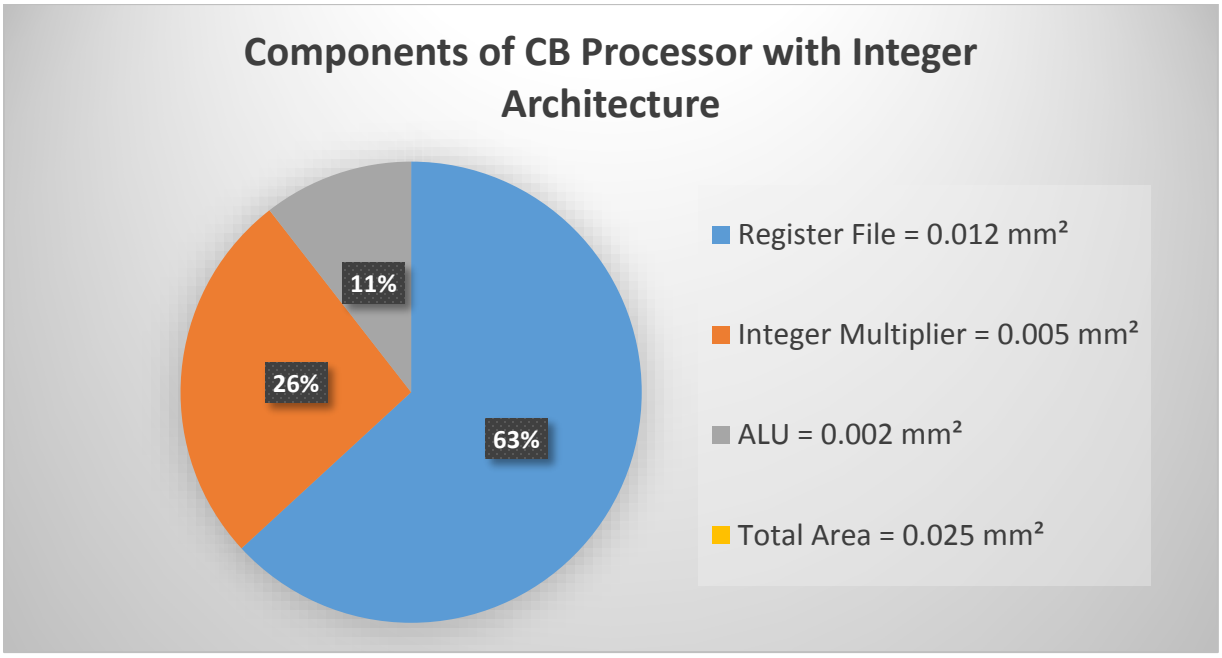

Figure 13: Components of CB Processor with Integer Architecture

The configuration of the processor supporting single precision floating point includes a FP multiplier and FP adder with the Integer architecture. Figure 14 shows both actual size and percentage contribution of the components in CB processor.

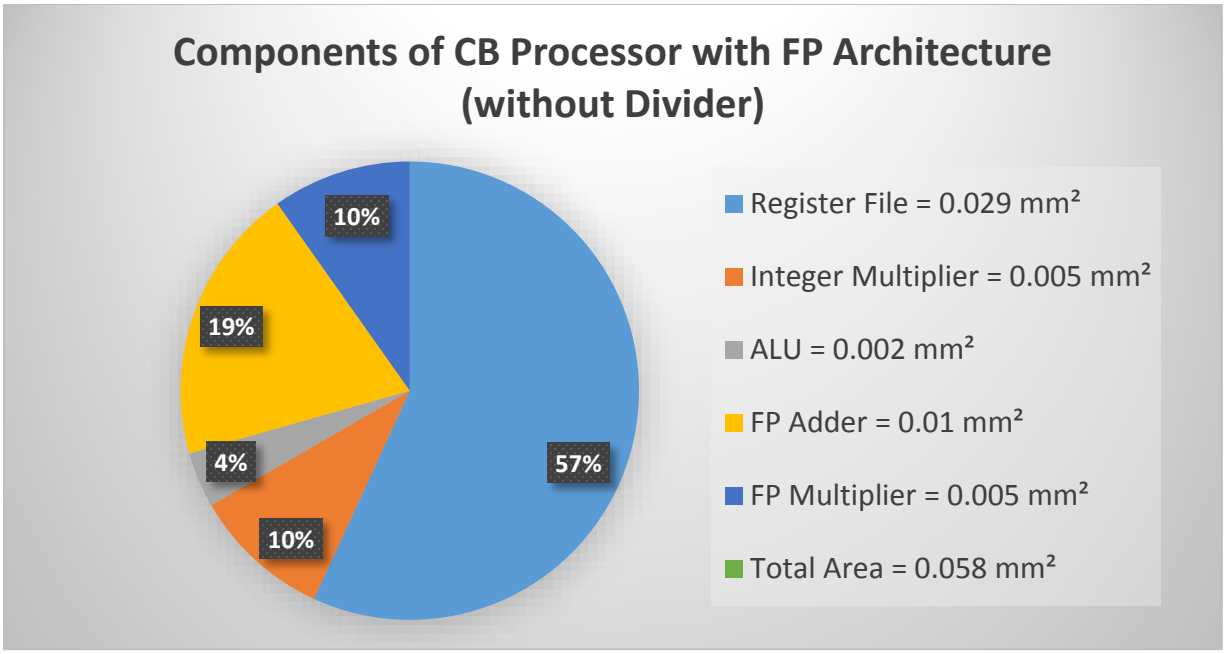

Figure 14: Components of CB Processor with FP Architecture

LDA classification consist in a matrix multiplication, so it only needs an efficient Multiply-Accumulation operator (that is implemented by joining the FP Multiplier and the adder). On the other hand, the LDA Training phase is based on Vector average and covariance calculation, and on the calculation of the inverse matrix, which heavily requires FP division. The hardware FP Divider makes the total area of the processor 0.082 mm square, 228% larger than the integer processor. Figure 15 shows the components of the CB processor with floating point and divider.

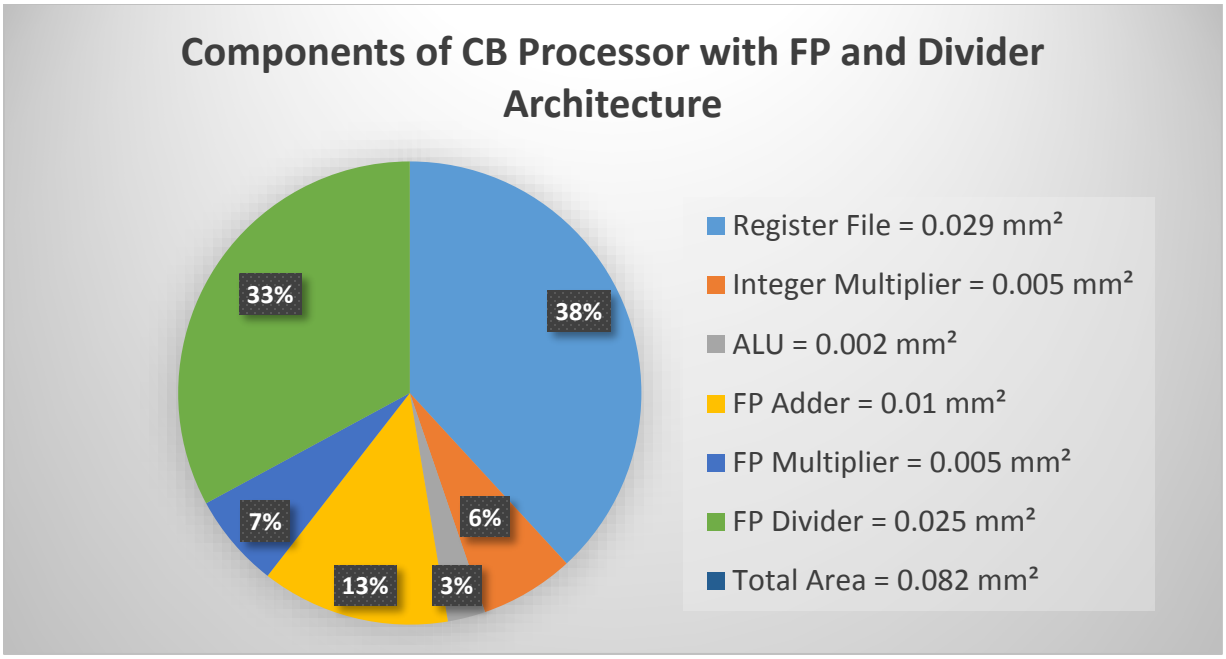

Figure 15: Components of CB Processor with FP and Divider Architecture

Of course, it may be acceptable to use emulation of the division operator for the simplest implementations of the LDA Training. However, increasing the number of classes and especially the dimension of the samples such approach would quickly become unfeasible, making the training operation unable to process inputs at a low hardware cost.

As hinted in the introductory chapters of this report, the peak power consumption of a CMOS VLSI circuit is strictly linked to its reliability [8], as most failure phenomena are directly or indirectly related to power dissipation. Power consumption of Integrated Circuits (ICs) is usually classified in two components:

a) Dynamic Power Consumption: due to charging and discharging capacitors or due to switching activities of a circuit [23] and depending on the clock
b) Static or Leakage Power Consumption: due to leakage currents [23] and with clock fixed

Table 5: Power Consumption of the main units composing the CB Processor describes the power consumption related to the main units composing the CB processor in CMOS 45nm technology at 250 MHz.

Figure 16 and Figure 17 show the percentage of Leakage power consumption and Total power consumption respectively, by each component of the CB processor.

Component	Dynamic Power Consumption (mW)	Leakage Power Consumption (mW)
CB Processor	2.2	1
Register File	0.82	0.35
Integer Multiplier	0.04	0.004
ALU	0.03	0.002
FP Adder	0.23	0.12
FP Multiplier	0.13	0.05
FP Divider	0.7	0.29

Table 5: Power Consumption of the main units composing the CB Processor

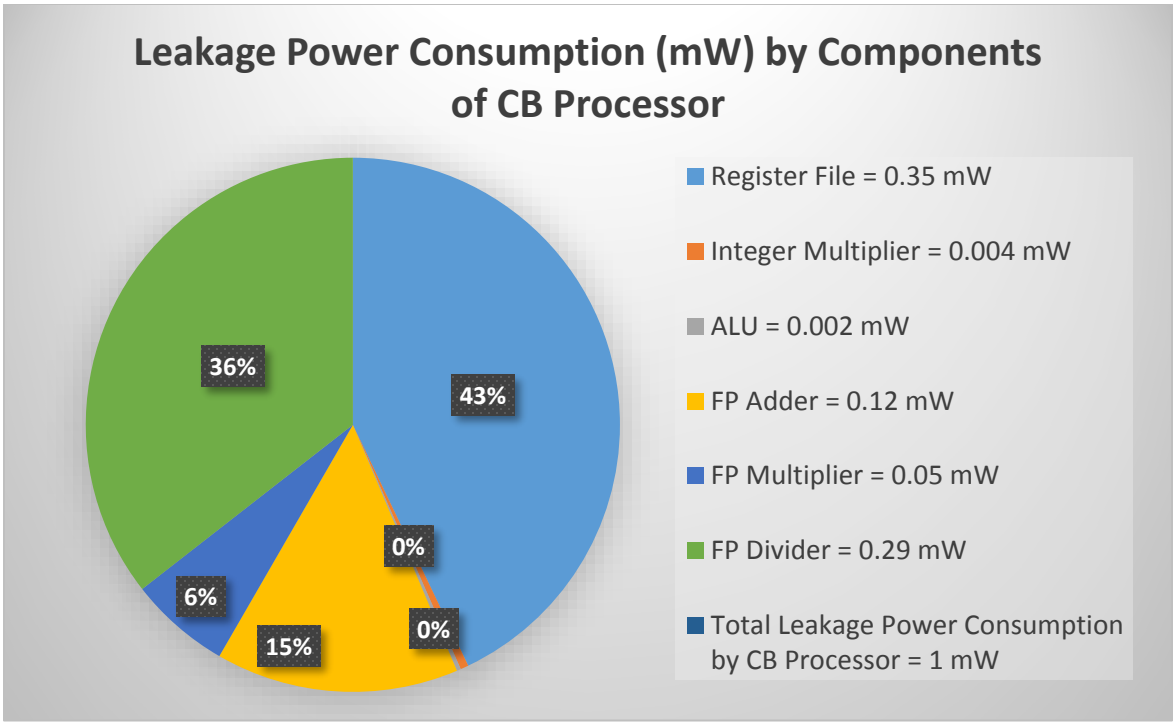

Figure 16: Leakage Power Consumption by individual components of CB Processor

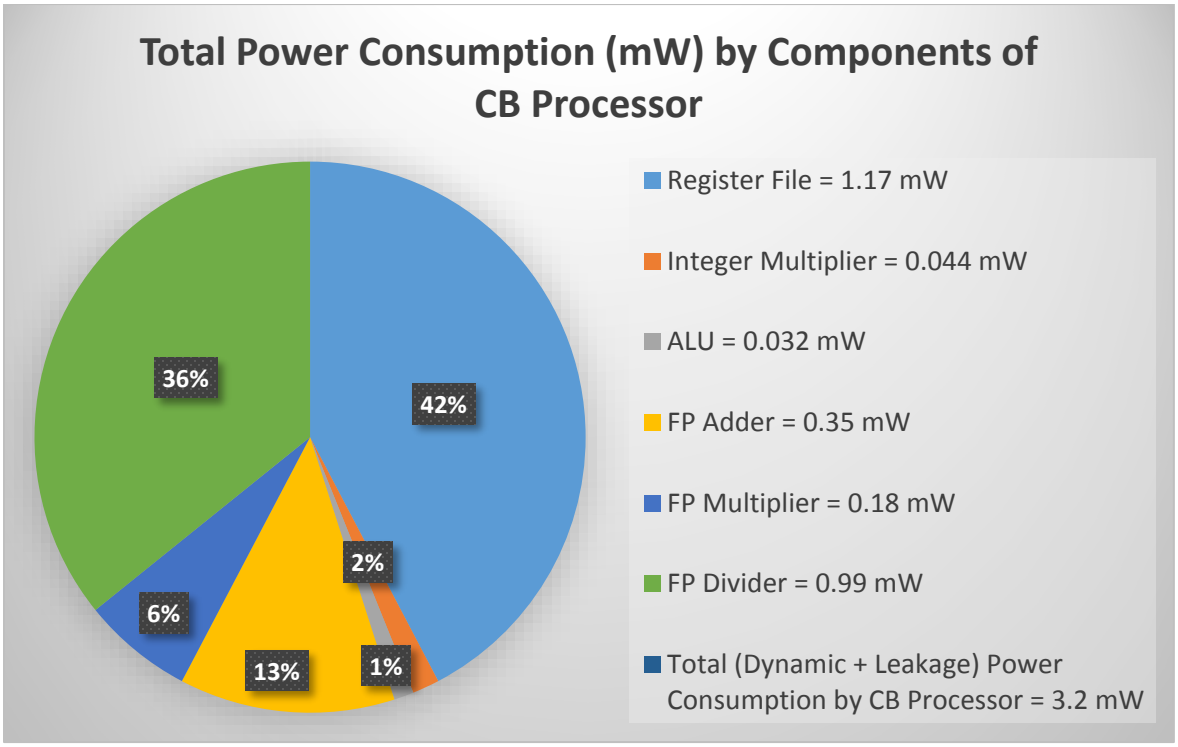

Figure 17: Total (Dynamic + Leakage) power Consumption by components of CB Processor

From above power consumption data, below is a Table 6 detailing the total (dynamic and leakage) power consumption by the three processor architectures of CB Processor; Integer, FP and FP with Divider.

Power Consumption of the CB Processor with different Architecture Configurations	
Processor with Integer	1.67 mW
Processor with Floating Point (no Divider)	2.2 mW
Processor with Floating Point and Divider	3.2 mW

Table 6: Total Power Consumption (Dynamic + Leakage) of CB Processor with different architecture configurations

7. Conclusion

This work focused on finding the best HW/SW configuration for the CB processor for IoT applications. The CB processor was developed as open source hardware at IP in Simon Fraser University, and is based on the RISC-V instruction set developed at University of California, Berkeley.

The aim of the processor is to classify the data collected by IoT sensors using the machine learning LDA algorithm. The main requirements that an IoT processor should fulfill are fast computation, small size and low power consumption and there is a trade-off between these requirements.

1) We can agree that the calculation of determinant with upper triangular matrix approach is much more efficient than the recursion method. It includes FP division, which will also increase the size and power consumption of the processor. The processor will be 228% larger (0.082 mm^2) compared to Integer processor and power consumption will be more by almost 92 % (3.2 mW). However, in order to get high computational speed we will have to opt for upper triangular approach.
2) Secondly, using a hardware divider would be the best approach to maintain the computational speed. Using “SoftFloat” FP emulation is almost 20 times slower than using FP hardware.
3) LDA is a best machine learning algorithm suitable for embedded system applications. All other algorithms are complex and so they would require a heavier hardware cost (especially in terms of embedded memory area and OS support) and computation time.

REFERENCES

1. Ryan H., “Intel, AMD & ARM Processors.”, https://kb.wisc.edu/showroom/page.php?id=4927 (Retrieved September 2016)
2. “Performance inside, 2007 Annual Report.”, http://www.intc.com/intelAR2007/index.html (Retrieved September 2016)
3. “Intel.”, Wikipedia, https://en.wikipedia.org/wiki/Intel, (Retrieved December 2016)
4. "Internet of Things Global Standards Initiative" ITU, http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx (Retrieved September 2016)
5. “Internet of Things.”, Wikipedia https://en.wikipedia.org/wiki/Internet_of_things (Retrieved September 2016)
6. Al-Ali, A. R., & Aburukba, R. (2015). Role of internet of things in the smart grid technology. *Journal of Computer and Communications*, *3*(05), 229.
7. Karimi, K., & Atkinson, G. (2013). What the Internet of Things (IoT) needs to become a reality. *White Paper*, *FreeScale and ARM*, 1-16.
8. Gurveer Kaur, “Design of Hardware Accelerators with Configurable Pipeline”, MENG Project presentation, Simon Fraser University, August, 2016
9. Kevin Char, “Internet of Things System Design with Integrated Wireless MCUs, http://www.silabs.com/Support%20Documents/TechnicalDocs/Internet-of-Things-System-Design-with-Integrated-Wireless-MCUs.pdf (Retrieved November 2016)
10. Ben Dickson, “Why IoT Security is So Critical”, https://techcrunch.com/2015/10/24/why-iot-security-is-so-critical (Retrieved November 2016)
11. “Intel”, http://www.intel.ca/content/www/ca/en/homepage.html (Retrieved December 2016)
12. “AMD”, http://www.amd.com/en-us (Retrieved December 2016)
13. “ARM”, https://www.arm.com (Retrieved December 2016)
14. Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, *1*(1), 22-32.
15. F.Campi, “Power Shaping Configurable Microprocessors for IoT devices”, Springer, 2016, ISBN 3319423045, 9783319423043)
16. Margaret Rouse, “Internet of Things (IoT)”, http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (Retrieved October 2016)
17. RISC-V: The Free and Open RISC Instruction Set Architecture. The RISC-V foundation, https://riscv.org (Retrieved December 2016)
18. “University of California, Berkeley”, http://www.berkeley.edu (Retrieved December 2016)
19. D. Yang, “Implementation of a Wearable Feedback System Monitoring the Activities of Upper-extremities”, MENG Project presentation, Simon Fraser University, April, 2015
20. Rob Schapire, “Machine Learning Algorithms for Classification”, http://www.cs.princeton.edu/~schapire/talks/picasso-minicourse.pdf (Retrieved December 2016)
21. “Floating-point unit”, Wikipedia, https://en.wikipedia.org/wiki/Floating-point_unit (Retrieved November 2016)
22. John R. Hauser, “Berkeley SoftFloat” http://www.jhauser.us/arithmetic/SoftFloat.html (Retrieved December 2016)

23. Anju S. Pillai, Isha T. B., “Factors Causing Power Consumption in an Embedded Processor – A Study”, http://www.ijaiem.org/volume2issue7/IJAIEM-2013-07-23-077.pdf (Retrieved December 2016)

24. Vikram Singh, “Pattern Recognition Chapter 2 – Fischer Linear Discriminant Analysis”, Youtube, https://www.youtube.com/watch?v=hEEj8MEvfG4 (Retrieved November 2016)

25. “GNU Operating System”, http://www.gnu.org/home.en.html (Retrieved December 2016)

26. “Machine Learning”, Wikipedia, https://en.wikipedia.org/wiki/Machine_learning (Retrieved December 2016)

27. Margaret Rouse, “Machine Learning”, http://whatis.techtarget.com/definition/machine-learning (Retrieved December 2016)

28. “Data Mining”, Wikipedia, https://en.wikipedia.org/wiki/Data_mining (Retrieved December 2016)

29. “Data Mining: What is Data Mining”, http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm (Retrieved December 2016)

30. Xia, F., Bahreyni, B., & Campi, F. (2016, November). Design of digital modules for capacitive proximity sensing system applications. In *Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on* (pp. 1-4). IEEE.

31. “Linear Discriminant Analysis”, Wikipedia, https://en.wikipedia.org/wiki/Linear_discriminant_analysis (Retrieved December 2016)

32. Kardi Teknomo, Linear Discriminant Analysis Numerical Example, people.revoledu.com/kardi/tutorial/LDA/Numerical%20Example.html (Retrieved December 2016)

33. “FreePDK”, http://www.eda.ncsu.edu/wiki/FreePDK

34. FreePDK: An Open-Source Variation-Aware Design Kit James E. Stine; Ivan Castellanos; Michael Wood; Jeff Henson; Fred Love; W. Rhett Davis; Paul D. Franzon; Michael Bucher; Sunil Basavarajaiah; Julie Oh; Ravi Jenkal 2007 IEEE International Conference on Microelectronic Systems Education (MSE'07) Year: 2007 Pages: 173 - 174, DOI: 10.1109/MSE.2007.44

35. Yang, D., Chhatre, N., Campi, F., & Menon, C. (2016, November). Feasibility of Support Vector Machine gesture classification on a wearable embedded device. In *Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on* (pp. 1-4). IEEE.

36. Fan Xia et al, "Multi-Functional Capacitive Proximity Sensing System for Industrial Safety Applications", 2016 IEEE SENSORS conference