Novel Hardware Architecture for Deep Learning and Computer Vision

M.Tech. Thesis

By

VISHAL BHARTIY

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2019

(This page is intentionally left blank)

Novel Hardware Architecture for Deep Learning and Computer Vision

A THESIS

Submitted in partial fulfillment of the

requirements for the award of the degree

of

Master of Technology

Communications and Signal Processing

by

VISHAL BHARTIY

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2019

(This page is intentionally left blank)

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **NOVEL HARDWARE ARCHITECTURE FOR DEEP LEARNING AND COMPUTER VISION** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July, 2017 to June, 2019 under the supervision of Dr. Santosh Kumar Vivhvakarma, Associate Professor, Indian Institute of Technology Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Signature of the student with date
VISHAL BHARTIY

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

Signature of the Supervisor of
M.Tech. thesis # 1 (with date)
Dr. Santosh Kumar Vishvakarma

Signature of the Supervisor of
M.Tech. thesis # 2 (with date)

VISHAL BHARTIY has successfully given his M.Tech. Oral Examination held on **1st July, 2019**.

Signature(s) of Supervisor(s) of M.Tech. thesis
Date:

Convener, DPGC
Date:

Signature of PSPC Member # 1
Date:

Signature of PSPC Member # 2
Date:

(This page is intentionally left blank)

ACKNOWLEDGEMENTS

I would like to sincerely thank Dr. Santosh Kumar Vishvakarma, my thesis supervisor and advisor for the last two years of my M.Tech. He has been very supportive since day one and I am grateful to him for devoting his time in guiding and motivating me to make the right decision when overwhelmed with options, or in moments of distress. I am thankful to him for providing me with the opportunities that shaped my M.Tech to be as it is today. I would also like to thank Ph.D. scholars Gopal Raut for his technical guidance in the first year. The discussions with my batchmates, especially, Pallab, Anil, Abhishek, Himanshu, Richa and Sarika were enriching.

I sincerely acknowledge the support of IIT Indore and MHRD for supporting my M.Tech. by providing lab equipment and facilities, and TA scholarship, respectively.

Last but not the least, my work would not have been possible without the encouragement of my parents, whose tremendous support helped me stay positive and overcome the worst of hurdles. To them, I will forever be grateful.

(This page is intentionally left blank)

DEDICATION

Dedicated to my parents

(This page is intentionally left blank)

Abstract

An on-chip Neural Network Accelerator is becoming more popular in recent years as they have proved to be the one of best algorithms for various important detection and classification problem in image, speech and many more never ending applications in intelligent system design. While their on-chip presence is desired, their heavy computational demand stands still as barrier in making the next big steps in System on Chip design. Although hardware accelerators for artificial neural networks are often not very complex designs yet a tradition design approach have not been very fruitful as they often end up taking large silicon area and power. Researchers have indicated that fixed point processing element can result in significant reduction in resources utilization with a rather negligible impact on accuracy. An artificial neural network (ANN) is very popular for a many problems that are very difficult for the other computational model like image processing, pattern recognition, prediction and classification. The use of hardware architectures can have the more parallel structure of ANNs for desire optimize performance or reduce the cost of the implementation, particularly for applications demanding high parallel computation. However, many unique disadvantages is with the hardware platforms such as limitations with high data precision which has relation to hardware cost of the necessary computation, and the reconfigurability in the hardware implementation compared to software. Many error resilient applications can be approximated using multi-layer perceptron (MLP) with insignificant degradation in output quality on hardware platforms. Field programmable Gate Arrays (FPGAs) and Application Specification Integrated Circuit (ASICs), have edge over Graphics Processing Units (GPUs) on cost. This work presents a high in literature, the challenge is to investigate an ANN architecture, especially in pattern recognition with less hardware and high performance

(This page is intentionally left blank)

LIST OF PUBLICATIONS

Published:

1) Goapl Raut, Vishal Bhartiy, Gunjan Rajput, Sajid Khan, Ankur Beohar and Santosh Kumar Vishvakarma. "Efficient Low-Precision CORDIC algorithm for Hardware Implementation of Artifcial Neural Network." *23rd International Symposium On VLSI Design And Test - VDAT 2019.*

Under Review:

1) Goapl Raut, Vishal Bhartiy, Ambika Prasad Shah and Santosh Kumar Vishvakarma. "FPGA-based Fully Configurable Layer Multiplexed Artificial Neural Network Accelerators." *Neural Processing Letters* Springer, 2018.

To be Submitted:

1) Goapl Raut, Vishal Bhartiy, Gunjan Rajput, Santosh Kumar Vishvakarma and Michael Huebner. "Low Complexity Approximate Multiply-Accumulate Unit for FPGA-Based Artificial Neural Network Accelerator".

(This page is intentionally left blank)

Contents

1 Introduction **3**

1.1 Intelligence 3

1.1.1 Psychometric Theory of Human Intelligence 4

1.1.2 Cognitive Theory of Human Intelligence 4

1.1.3 Cognitive contextual theory of human Intelligence 4

1.1.4 Biological Theories of Human Intelligence 5

1.2 Artificial Intelligence 5

1.2.1 Methods and Goals in AI 7

1.2.2 Turing Test 8

1.3 Early Milestones: The AI Winter 8

1.4 Machine Learning 9

1.4.1 Supervised learning 10

1.4.2 Unsupervised learning 10

1.4.3 Artificial Neural networks 10

1.4.4 Support Vector Machines 11

1.4.5 Bayesian Networks 11

1.5 ML Accelerators 12

1.5.1 Heterogeneous Computing Platforms 12

1.5.2 ASICs and FPGA 12

2 FPGA-based Fully Configurable Layer Multiplexed Artificial Neural Network Accelerator 13

2.1 Introduction 14

2.2 System Architecture 16

2.2.1 System Overview 16

2.2.2 Efficient Micro-architecture for Memory Addressing 18

2.3 Hardware Implementation of Configurable ANN 20

2.3.1 Proposed ANN Implementation 21

2.3.2 Dynamic Multiply and Accumulate (MAC) Engine 22

2.3.3 Design of LUT based Sigmoid Activation Function(AF) 25

2.3.4 Benchmark Methodology for Hardware implementation 27

2.3.5 Hardware Evaluation and Verification of the Proposed Accelerator 28

2.4 Results and Discussion 28

2.5 Conclusion 30

3 Efficient Low-Precision CORDIC algorithm for Hardware Implementation of Artificial Neural Network 31

3.1 Introduction 32

3.2 Cordic Algorithm Explanation 33

3.3 Exploration of Computational Unit for ANN using CORDIC algorithm . 35

3.4 Activation Function Design Technique 37

3.4.1 LUTs based implementation by storing function values 37

3.4.2 LUTs based implementation by storing parameters 37

3.4.3 Approximation in Calculation 38

3.4.4 Implementation using CORDIC algorithm 38

4 Low Complexity Approximate Multiply-Accumulate Unit for FPGA-Based Artificial Neural Network Accelerator 41

4.1 Introduction 42

4.2 Releted Work 42

4.3 An Architectural Overview 44

4.4 Hardware Implementation 46

4.4.1 Design Architecture 47

4.4.2 Computational Unit 48

4.4.2.1 Multiply-Accumulate Unit 48

4.4.2.2 Squashing Function 50

4.4.3 Hardware Evaluation and Verification of the ANN Accelerator . 51

4.5 Results and Discussion 51

4.5.1 Low Complexity Multiplier Architecture and LUT based PWL Activation Function 51

4.5.2 Hardware Implementation using Approximate MAC 52

4.6 Conclusion 53

(This page is intentionally left blank)

List of Figures

2.1 The architecture of the ANN co-processor. The co-processor uses the ANN core(programmable logic) on-chip BRAM memory banks are used to cache weights/biases. The computational engine carries out the computation of input data with trained constants using MAC. 17

2.2 BRAM address for weight and bias memory to be access by the individual neuron 19

2.3 Configurable multiplexed 4 layer ANN core that can be used for pattern recognition. It consist of array of neuron layer1/3 and intermediate to serialize, layer multiplexed single neurons layer2. A real-world ANN can have several layer depends on pattern extraction 22

2.4 Neuron connection in architecture, x_i, w_i, b, and $f(.)$ are the input feature, weight, bias, non-linear function 23

2.5 The proposed configurable fixed-point MACs with date representation schema. 24

2.6 The operation of proposed configurable fixed-point MAC in FPGAs. Fig.(a) shows the operation in multiplexed single neuron in layer2 and Fig.(b) shows the operation of MAC in array of neuron in Layer1. . . . 25

3.1 Artificial single neuron model architecture 32

3.2 The rotation and pseudo-rotation of a vector of length about an angle with the origin. The pseudo-rotation is the key idea about the CORDIC circuit for computation and realization of mathematical functions 34

3.3 CORDIC algorithm based proposed architecture for versatile computation 35

4.1 Fully connected efficient artificial neural network (ANN) architecture . 44

4.2 Embedded blocks design with ANN core on Vivado. 45

4.3 MATLAB neural network toolbox to generate graphical user interface networks [**b18**] 46

4.4 The proposed configurable fixed-point MAC with data representation schema. 48

4.5 The proposed fixed-point 4×2 approximate multiplier 49

4.6 The design fixed-point 8×4 using proposed approximate 4×2 multiplier 49

4.7 Placement of the reconfigurable ann_core area 52

4.8 Dynamic Power Comparision for FPGA based 4(8-4-4-1) layer ANN Accelerator 53

List of Tables

2.1 LUT based Activation Function Implementation 27

2.2 Resource Utilization for XC7Z010clg400 28

2.3 Dynamic Power Comparision for FPGA based 4 layer ANN Accelerator 29

3.1 40

4.1 LUT based Activation Function Implementation 50

4.2 Performance comparison of Proposed approximate comparator 52

(This page is intentionally left blank)

Chapter 1

Introduction

The artificial intelligence is the intelligence possessed by machines to demonstrate the cognitive abilities which are comparable to the intelligence displayed by humans as well as animals. The field was founded on the fact that human intelligence can be precisely defined and hence, machines can simulate these definitions. Before understanding the concept of artificial intelligence, its necessary to understand the concept of intelligence in general

1.1 Intelligence

Human intelligence has been the main focus of researchers because of the higher cognitive abilities and high level of self-awareness. This includes the capability to learn, form concepts, understand, reasoning, pattern recognition, planning, problem-solving, language for communication. The study of intelligence is not free from controversy and researchers to have disputes over the certainty of possession of intelligence in animals and plants. Many researchers have produced enough evidence that animals also possess some kind of intelligence. One evidence is the presence of g factor in non-humans. G factor is a psychometric parameter which can be estimated in both humans and non-humans. Since g factor contributes directly to the standard IQ primitives and hence there is enough evidence in support of different kinds of intelligence. Human Intelligence is the set of mental qualities which are the ability to learn from experi-

ence, to adapt a new situation, understanding and handling of new concepts and ability to use them to manipulate the environment. Historically, Psychologists have different opinions about the definition of intelligence, which mainly fell into two categories one being one's ability to think abstract and others being able to learn and answer correctly to questions. In modern psychology, it is well agreed that the ability to adapt to the environment is the fundamental key to understand intelligence. Theories of intelligence are mainly classified based on cognition and the structured process by which the mind functions, based on this there are four theories of intelligence

1.1.1 Psychometric Theory of Human Intelligence

It is a structured theory to understand the intelligence and uses the composite data from mental tests such as analogies, classification, series completion etc. Because psychometric theory relies on the data from mental testes and thus is quantifiable. The weakness in one area may be the cause of strong abilities in another area e.g., one's vulnerability in mathematics is the cause of their strong reasoning and vice versa

1.1.2 Cognitive Theory of Human Intelligence

When evaluating test scores of psychometric test, one can reach a misleading conclusion. For example, if a person doesn't have a strong vocabulary, then he/she may not perform well in verbal reasoning test, and hence the evaluation score will indicate that the person has weak reasoning abilities and therefore the wrong conclusion of his/her intelligence. Thus cognitive theory states that intelligence is a mental operation performed on representation or images of scenarios and abstract operations performed on those images to conclude.

1.1.3 Cognitive contextual theory of human Intelligence

This theory was first proposed by Howard Gardner and explained the existence of multiple – intelligence. According to this theory there exists eight separate intelligence also knows as 'spheres of intelligence' namely linguistics, musical, logical-mathematical,

spatial, bodily-kinesthetic, interpersonal, intrapersonal, naturalistic. Any individual possesses some fraction of each sphere and cognitive profile for each is unique. Thus it is impossible to quantify one's intelligence with data like IQ scores.

1.1.4 Biological Theories of Human Intelligence

Biological theory also known as reductionism theory states that [1] understanding of intelligence is only possible by understanding of its biological basis. Biological theory mainly falls in three catagories namely Hemispheric , Brain wave studies, Blood Flow studies

1.2 Artificial Intelligence

The artificial intelligence aims to design the "intelligent agents". The intelligent agents are defined as any system that possess abilities such as perception of their environment and plan and executed action such that it optimizes its chance of succeeding.

The research in the artificial intelligence have the areas that study the following traits:

1) **Learning:** One of the simplest form of learning is based on trails and error. Machine or program try random approaches to solve a problem and when it finds correct solution it memories it. One example of this kind of learning is a computer program which solves mate-in-one chess problem. Program tries random moves until it succeeds and then it stores that solution so that it can use it next time. This kind of learning is useful when ai encounters problem that it has learned in past but fails to generalize [1]. Implementation generalization in ai it's a hard problem. In generalization, ai most be able to apply past experience to solve problems of new kind citeMichie1994. For example, by learning past form of a verb "learn" ai should be able to find past form of "jump" by concluding that it need to add "ed" at the end.

2) **Reasoning:**The reasoning is to draw meaningful conclusions from a set of meaningful expressions through an argument or situation. Reasoning is clarified as de-

ductive or inductive. The basic difference between these two kinds is that in case of deductive reasoning if the premises is true that means the conclusions based upon the premises will also be true while in inductive reasoning, the conclusion developed over a true premise need not be true. Therefore, deductive reasoning is used in fields such as mathematics and logic where bigger conclusions are drawn based on some axioms and rules. On the other hand, inductive reasoning is used in science and engineering because models defined in present may be overruled by new concepts in future.

3) **Problem Solving:**Problem solving in contrast with ai may be is termed as systematic set of actions from a range of meaningful action to achieve a defined goal. Special purpose and general purpose are the two kinds of problem solving techniques in ai. Special purpose techniques are used to solve a particular problem where in general purpose technique, ai try to perform step by step actions from a well defined set in order to reduced the difference between it's current state and desired solution

4) **Perception:**In perception, environment it's scanned by the means of various sensory organs which can be real or artificial and the meaningful conclusion is made from finding the correlation amount various objects present. Also the perception is influenced by the angle of observation, intensity of illumination within the scene and the contrast with surroundings. Examples of these includes deep neural neural networks which have shown significant impact in making visual sensing meaningful to help autonomous vehicle and robots to roam around.

5) **Language:**Language is a system of spoken, manual or written symbols by which human beings expresses themselves. According to Henry Sweet, an English phonetician and language scholar, "Language is the expression of ideas by means of speech sounds combined into words. Words are combined into sentence, this combination answering to the ideas into thoughts" Several computer program are able to responds in human language but that doesn't mean the program actually understands language. It's just that the program has reached a level of sophistication that it's response in human Language is indistinguishable from human.

6) **Ability to Move Objects:** AI is widely used in manipulation of objects such as

robotic arms [2],industrial controllers etc.Proper planning is used to break down motion into several small primitives [3].Smaller the primitives, more complicated motion planning as indicated by Maravec's Pardox [4]

1.2.1 Methods and Goals in AI

1) **The neats and the scruffiest:**Logic in ai was introduced by John McCarthy in late 50s and early 60s [5]. Neat and scruffy are the two different types of ai researches. Neats consider that ai solutions should be clear, elegant and have correctness while scruffies believe that computational complexity makes intelligence too complicated to have any logic. Gerald Sussman said that" using precise language to describe inherently imprecise concepts doesn't make them any more precise".In 1975 , Minsky [6] described that scruffy ai researchers are using constructs which are" common sense" . These constructs are not logical but fits well in contrast with everything we think and see. He named these structures as "frames" which would be adopted many years later in object oriented programming as the idea of inheritance

2) **Symbolic VS Connectionist Approach:**These are two approaches that are Top-to down and Bottom-up approaches. In symbolism, the top to down approach, the researcher tries to achieve the mimic intelligence by exploring the cognition, and they do not explore the underlying neural structure. On the other hand, Connectionist tries bottom-up approach, and they design artificial neural Networks to explore the intelligence

3) **Strong AI and Weak or Applied AI:** Philosophy of strong AI was first Introduced in 1980 by John Searle. Strong AI research aims to build the machines that have thinking ability which can not be distinguished from that of human. Strong AI is an ambitious and challenging domain. Weak AI is the feasible applied AI in commercial uses of intelligent system design.

1.2.2 Turing Test

Turing test was developed by Alan Turing, the father of Computer science in 1950 to determine whether a machine is capable of thinking like a human or not. Turing test involves three participants who is an interrogator, a computer, and a human foil. The interrogating person asks questions to the machine through keyboard or mouse etc. and machine must try to impose as if it is a human. The interrogator is allowed to ask any kind of question, and human foil must help the interrogator to make the correct decision. If at the end of the test machine succeeds to impose as a human, then if passes the Turing test. Untill now no AI machine or program have been able to pass the turing test without any dilution.

1.3 Early Milestones: The AI Winter

In 1940 from the invention of the programmable digital computer, scientists inspired to design an electronic brain. Philosophers tried to explain human's process of thinking as the manipulation of symbols. During 1956, at Dartmouth College, an AI workshop was founded. Many of the aspiring scientists were part of that, and millions of dollars were funded. But their anticipation for the time required to produce some benchmark progress was greatly underestimated. In 1970s , researchers ran into some very fundamental limits and they were able to overcome it by the next decades. These problems were:

1) **Limited computer power:** It was argued that computers are still too weak in memory and processing speed to possess any kind of intelligence. One of the examples of barrier in Computing power was Ross Quillian's work on natural language processing. He was successful in his research, but memory could only demonstrate twenty words. Modern computer vision application requires 10000 to 1000000 MIPS, which was nowhere near possible in that time. Even the fastest supercomputer Cray-1 in 1976 was able to perform only a maximum of 130 MIPS.

2) **Combinatorial complexity:**Progressing on cook-levin theorem, Richard Karp, in 1972, demonstrated that there are many problems that can only be solved in exponential time complexity and the optimum solution for this problem will require an unfeasible amount of time.

3) **Lower Databases:**Many AI applications will require a huge amount of data of the same kind and the ai application must be about to identify at least some hidden constructs of it. Researchers soon learned that this would require a huge amount of data which was not possible back in the 70s.

4) **Moravec's paradox:**AI researchers encountered a very fundamental problem in the area of vision and robotics that it is comparatively easy for computers to solve the complex mathematical and geometrical problem than to solve a basic problem of vision such as recognition of face or crossing obstacles without bumping into them.

5) **Frame and Qualification Problems:**John McCarthy and other AI researchers who used logic concluded that it is not possible to present ordinary deductions which involve planning or reasoning without changing the underlying logic itself. Hence they developed new logics such as non-monotonic logic, which are capable of capturing and representing defeasible inferences which allow reasoners to draw a tentative conclusion.

1.4 Machine Learning

Machine learning (ML) is the process by which a software or machine learns to process a task without instructions sets. ML is used on a set of data known as "training data" and draws a mathematical models for it. Based on this model it detects the new input and produces the appropriate output Arthur Samuel in 1959. the ML is best suited for the task which are so complex and diverse that instructions code can not be written for image classification if we want to classify the image of say few animals, how can we write image code to classify a cat image from a dog image that can have infinitely large number of postures if you have a cat picture in which it is only partially visible say it

is partially under the blanket? So the software must learn the underlying relationship among the different picture and draw a mathematical model which can classify a new cat pic from the pics of other animals or objects.The idea to learn from training data set fall broadly under two categories one is called supervised learning and other is unsupervised learning. Apart from that many researchers uses some hybrid of the two methods. Below we shall have a closer look at the two methods:

1.4.1 Supervised learning

In SL algorithm works by building mathematical models of the data that contains some label for both input and desired output. Thus optimization is applied for given input output sets. This kind of learning is used for a wide range of classifiers, ranking systems, image detections etc.

1.4.2 Unsupervised learning

In unsupervised learning, algorithm will optimizes the mathematical models by using the unlabelled data i.e., only input is available. Data clusters analysis is an important application of unsupervised learning.

ML is done on some model which can be a software or a hardware and software (HW/SW) co-design. These models are trained through training data for an specified task. Below is some detailed description of these algorithms

1.4.3 Artificial Neural networks

Artificial Neural Networks (ANN) are biological brain inspired neural networks comes under the bottom-up or connectionism approach of AI. An artificial Neural Networks is a connected network of Artificial Neurons. Each connection through neuron called a synaptic connection transmits signal from one artificial neuron to others. Each neural synaptic connection has some weight that gets optimized during the training of data. So when training if finished over a large data set, we have a highly optimized model

of ANN which have great accuracy. Convolution Neural Networks- Convolution neural networks (CNN) are similar to ANN but are capable of doing advanced classification. Unlike ANN, CNN are not brain inspired algorithm. CNN are inspired by visual cortex of mammals and how the brain detects images through vision. CNN models are the hottest topics these days since it's win of ImageNet challenge in 2012 [7].

1.4.4 Support Vector Machines

Support Vector Machines (SVM) are the type of ML models which represent training data into an N dimensional hyperspace and tries to find out the optimum plan that divide the space space into two. Thus new data set falls under either of the two sides other the decision space.

1.4.5 Bayesian Networks

A Bayesian network is a network that is used to draw a probabilistic model of the possible outcomes through a directed acyclic graph. Bayesian Networks which models sequences of one type are called dynamic Bayesian networks and generalization of Bayesian Networks are influence diagrams.
These were the few of many Machine learning algorithms. The discussion of ML algorithms are not limited to these algorithms. Many more robust algorithms are there these days which are widely used by computer scientists which are out of the scope of this thesis.

Our discussion will move towards the implementation of these algorithms on the platforms which doesn't have massive GPU power. Development of hardware architecture to execute these algorithms over an ASIC and FPGA is the primary research goal so that they can be used in a huge variety of application in the domain of intelligent system design. In the next section we will discuss about the so called ML accelerators which is term used to describe the use of optimum hardware to accelerate these computationally heavy algorithms.

1.5 ML Accelerators

ML accelerators are hardware which process or co process the Machine learning algorithms especially Neural Networks. Earlier attempts of ML accelerators were making the Use of Digital signal processing to do pixel to pixel multiplication and accumulation. Acceleration on computers are achieved by graphic processing units or GPUs but since the rise of CNN more dedicated chips are being developed. Following is a brief discussion of different approaches of ML accelerators.

1.5.1 Heterogeneous Computing Platforms

Heterogeneous Computing refers to a platform on which several dedicated hardware are integrated together for example a CPU coupled with GPU and DSPs.A FPGA embedded with processor over a single chip.Since the approach of ANN or CNN is similar to that of image processing, the basic computational unit such as GPUs which are designed to perform image and video processing are very efficient to implementation of these algorithms. But this comes with a lot of trade-off such as cost and area and space. We simply can not use GPU as coprocessor for a cellphone SoC thus industry is working on working ASIC to integrate them on their platform.

1.5.2 ASICs and FPGA

ASIC and FPGA are the obvious choice in the research field of ML accelerators. FPGA are widely used for various servers and other usefully applications. In addition of this modern FPGA have processor embedded within those chips and thus allowing a SW-HW based co-design. ASICs which have been developed by Intel such as Nervana and Movidius. NVIDIA has a division working in developing Hardware for Smart Vehicles. The following chapters contains discussion of the Novel Hardware developed for Artificial Neural Networks for FPGA or ASICs.

Chapter 2

FPGA-based Fully Configurable Layer Multiplexed Artificial Neural Network Accelerator

Artificial neural network (ANN) is one of the most prominent machine learning techniques and have a wide variety of applications. Despite many decades of research on high-performance ANN accelerator, still their heavy computational demand has required specialized architecture for computational acceleration. In this Chapter we discuss fully multiplexed ANN acceleration aimed to reduce resource requirement and high computational demand. The efficient look-up-table based hardware implementation of nonlinear activation function with approximation scheme is also proposed which have lesser area and delay compared to state-of-the-art. The on-chip memory with optimized technique is used to catch input features and weight/bias which reduces the external memory bandwidth requirement. The memory data-flow is dynamically adjusted to attain high computational throughput for an individual neuron in the architecture. The FPGA-based ANN core is proposed featuring configurable and layer multiplexed to reduce inter layer complexity on hardware Zybo board Xilinx Zynq xc7z010clg400 SoC, and running at the $100MHz$ clock frequency. Proposed ANN architecture is 4 times more power efficient and nearly 0.5 time resources utilized compared to the conventional fully connected ANN technique.

2.1 Introduction

An artificial neural network has become a very popular algorithm in pattern recognition since the recent evolution of computer hardware, which fulfilled the high computational power requirement of learning algorithms[8]. One reason is that ANN has the edge over other prediction techniques is its capability to learn hidden relationship in heteroscedastic data i.e. highly volatile data with non-constant variance[9]. ANN is a robust algorithm when it comes to learn from examples, high-level parallelism, and robustness to noise. This makes ANN a popular choice for image processing and feature learning, financial forecasting, speech and biomedical signal processing, biometrics, structural and constructional engineering, industry risk monitoring[10]. Due to its diverse application, ANN it's applied on various non-linear detection problems some of which as lane detection, pattern recognition, fault detection and monitoring in the industry[11][12] and so on. The platform like, who generally do not have access to high-performance computers are often run by dedicated hardware.

The FPGA based approach has a flexible trade-off among power, reconfigurability, and processing speed and therefore they are highly flexible and have shorter development time than ASIC[13]. Moreover, the developers have high control over new generation FPGA's SoC which has dedicated hard or soft microprocessor hosts. The implementing of digital circuits using conventional digital design procedure, tend to take more chip area on FPGA than the semi-custom or full custom ASIC design approach[14]. Smart vehicle often requires high computational power while in real-time image processing on the image provided by car cameras. This type of applications often involves feature learning and opens up a need for on-chip accelerators[15] which can provide fast multiplication and accumulation operation on steaming pixels as well as a learning feature. The features provided by FPGA have the edge over other platforms such as CPU, GPU on cost in addition efficiency, flexibility to reuse and reconfigure to optimize performance for any specific type of application[16]. Overall, High level of parallel data processing and easy data access to reduce the overall processing time and its low-cost, FPGA becomes a natural choice over other platforms. High level of parallel data processing and easy data access to reduce the overall processing time and its low-cost,

FPGA becomes a natural choice over other platforms for ANN implementaation.

The FPGA provides a flexible way to implement artificial neural networks However, its require more chip area due to their low packing density. Furthermore, FPGA based implementation of ANN requires an appropriate number system to represent the weights and biases[17]. While designing ANN accelerator on FPGA attention must be paid to precision, clock frequency, parallelism and limitation to the configuration. The 8-bit, 16-bit, 32-bit or 64-bit floating and fixed point format is used as data width. Whereas, floating point format will lose more Chip area but provides improved precision while Fixed point is used when resources are limited and some degree of error can be tolerated[18]. In order have a good design of ANN, the approach should be divided into two parts: Basic ANN core logic and control unit. The core logic is for storing and processing the computation. The weight/bias can be stored on Block RAM, distributed FPGA memory or even external RAM if present in sufficient amount[19]. The FPGA features, full control on reiterative learning is achieved and modification in the weights and biases as well as architecture. Back-propagation is also easily achieved on FPGA[20].

Approximate computing research area has evolved a lot due to the constant increase in the areas of interest demanding systems be designed with limited silicon area and overall less energy consumption but with tolerable errors[21]. The proposed ANN architecture utilities the reconfigurability power of FPGA and is made fully reconfigurable i.e. number of layers in ANN, type of layer and number of neurons in each layer can be adequately controlled by as per application desire. To maximize the utilization density of 8-bit operands, the LUT units could be added, this in turns optimize the Xilinx on-chip utilization, this remarkably increases the performance of Xilinx FPGA[22]. Each neuron of ANN having multiplication and accumulation circuit which usually takes most of the chip area. As a number of neuron increases, the chip area increase and this problem is more dominant in case of high-resolution image feature processing. The proposed ANN core unit includes activation functionality and its LUT based realization.

Recent research on FPGA based ANN, many architeture have been proposed [23] [24][25]. As the parallel structure of FPGAs matches the topologies of ANNs, their are quite limitations for the implementation of ANNs. While the ANN architecture is researched well enough, the MAC unit it's still untouched. This is because the de-

velopment of ANN has been researched in the software domain but in FPGA based implementation problem of high energy, expenses arise. The key to resolve this problem is to have an efficient data representation along with the configurable feature of MAC. MAC generally takes three inputs which are input features, weights & biases from its designated data paths. Its contribution to FPGA Based ANN architecture is summarized in the following three points:

- We confirm that ANN architecture based on approximation techniques are better when efficient energy consumption is taken into account.
- Optimize configurable MAC unit is designed and presented for layers with different feature.
- Energy Efficient LUT Based peace-wise linear approximate Activation function.

2.2 System Architecture

The many accelerator design have been proposed and implemented based on GPU, ASIC and FPGA. However General purposes processors are not cost-effective for deep neural network due to large parallel computation. Thus, among these FPGA based approach is the best option and FPGA accelerator have the many advantage like good performance, fast implementation, energy efficient and its reconfigurability[26]. The detail system overview and on-chip memory architecture of the ANN coprocessor is described below.

2.2.1 System Overview

System architecture comprises of two main parts i.e FPGA for real-time implementation and host computer for synthesis & training. Conceptual block level embedded architecture of ANN accelerator is shown in Fig. 2.1. An on-chip BRAM is used for weight and bias storage which gives the maximum throughput. Whereas through BRAM interface, host will read and write the weight/bias memories to the address. The trained weight/bias constants and image features that have to be classified are feed to

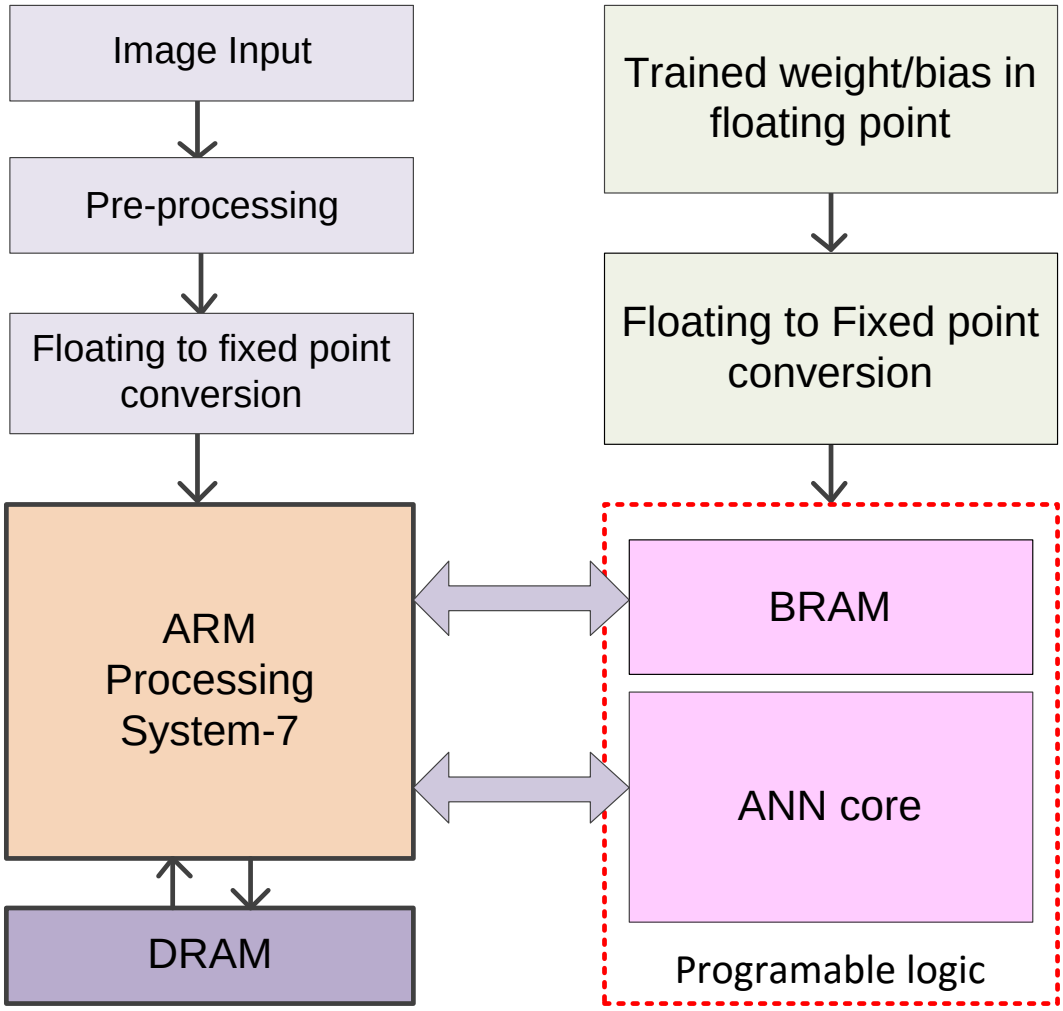

Figure 2.1: The architecture of the ANN co-processor. The co-processor uses the ANN core(programmable logic) on-chip BRAM memory banks are used to cache weights/biases. The computational engine carries out the computation of input data with trained constants using MAC.

the co-processor. However, for more computational efficiency, the input image data is converted to the 8-bit fixed point format. The proposed ANN architecture is fully parameterizable; Define parameters in Hardware Description Language (HDL) code can modify according to the user. The coprocessor is implemented using fix-point sign-magnitude representation. The embedded application used DRAM to process the algorithm. The block automation is run by FSM code in 'C' programing with Xilinx SDK tool and hardware implementation is done using Vivado. We have done preprocessing on image to reduce the pixel size and used fix-point representation for efficient computation.

The configurable ANN core is realized in programmable logic(PL) part of Zynq device. The input and output side of ANN core are connected to the ARM host processing system(PS)[27] of Zynq device through AXI interface protocols. The on-chip register is used to convert serial to parallel and vice versa data interface in the intermediate stage. The computational operation are performed in MAC block and the weight/bias for real-time computation will access from the BRAM by the MAC unit. The control logic is about memory access architecture, finite state machine based (FSM) controlling logic and an internal signal of the programmable logic data path. The programmable ANN core is access by the on-chip processor through AXI interface and process to be performed. The accelerator currently supports a maximum image size of 14×14, 8-bit serial transmission to the logic core. To process and compress the image to the specified image size, encoder is used in processing system(PS) side. The multiplexing to reduce interlayer connection is the unique feature of proposed architecture and additionally approximation technique. The neural network algorithm was modeled using Xilinx System Generator for learning and its realization in Xilinx FPGA. The configurable ANN accelerator can be optimized with a different number of layer.

In supervised learning, for each training input pattern, the network is presented with the desired output. The proposed ANN core is verified for the popular Modified National Institute of Standards and Technology (MNIST) database. The key finding of our exploration is that, the deep neural networks can be trained using low-precision fixed-point arithmetic, provided that the stochastic rounding scheme is applied while operating on fixed-point numbers. For the digital implementation of neural network, the precision of the various blocks is an important part. The output resolution will depend on word length precision. However, effective hardware implementation can be obtained by optimizing the resolution for arithmetic representations.

2.2.2 Efficient Micro-architecture for Memory Addressing

The reconfigurable architecture is dependant on the design vaiable parameter which are upgradeable and extend the useful architeture. The proposed ANN core, being highly reconfigurable and generates weight/bias memory architecture, the memory-based FSM implemented. The address space for weight and bias memory is unique BRAM inter-

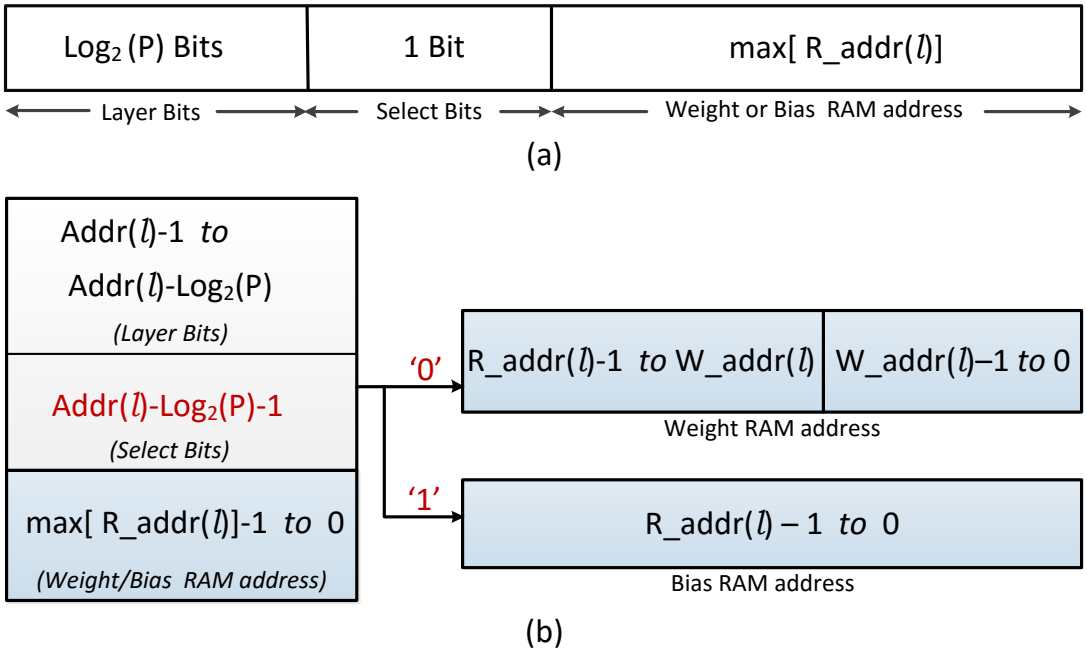

Figure 2.2: BRAM address for weight and bias memory to be access by the individual neuron

connect address. This address also contains sub-address of several memory locations which does not correspond to any physical implementation. Therefore, to avoid read or write operation on these unimplemented addresses the following schemes are considered:

- The total number of layers in ANN = P
- The number of neurons or biases in l^{th} layer = $N(l)$
- Number of inputs to the l^{th} layer = $K(l)$

The format for weight/bias memory address selection for the individual neuron is described in Fig. 2.2. First $log_2(P)$ bits represents the layer of which weight and bias is to be address. Next bit is called *select bit*, '0' corresponds to the following bits represent weight address whereas '1' correspond to the following bits are representing bias address. After select bit, rest of a bits correspond to weight and bias address RAM. Bits required for representing layer, select bit and RAM address are represented in Fig. 2.2(a). RAM Address length for l^{th} layer is given by

$$R\_addr(l) = log_2 N(l) + log_2 K(l)$$

Address length for biases of l^{th} layer is $B\_addr(l)$= $log_2N(l)$ similarly the address length for weight $W(n,i)$ of l^{th} layer corresponding to i^{th} input of n^{th} neuron of the layer is $W\_addr(l) = log_2K(l)$ The detailed description of bit count by taking in account number of layers, weights and neurons is represent in Fig. 2.2(b). Total address length to access the weight/bias by the co-processor can be expressed as:

$$Addr(l) = log_2P\ + 1 + \ max[R\_addr(l)]$$

The address representation for reconfigurable architwecture is dependent on parameter specification used in ANN core. The algorithm 1 expressing the footstep for the selection of weight and bias memory address.

Algorithm 1 On-chip memory architecture for Weight and Bias

Input: In: Gave the BRAM memory address
Output: Out: Reading data from the Input address location

1: Choose log_2P bits to indicate a layer in little endian format
2: For a bias choose select bit as one

   1) find out the address length required to represent the biases of l_{th} layer by using:

      $B\_addr(l) = log_2N(l)$

   2) Use bits $B\_addr(l) - 1$ down to 0 to address a bias

3: For a weight $W(n,i)$ make select bit as

   1) Find weight address length and RAM address length by using following equations:

      $W\_addr(l) = log_2K(l)$
      $R\_addr(l) = log_2N(l) + log_2K(l)$

   2) Address using n^{th} neuron using bits $R\_addr(l) - 1$ down to $W\_addr(l)$

   3) Address the i^{th} input of nth neuron using rest of the bits from $W\_addr(l) - 1$ down to 0

2.3 Hardware Implementation of Configurable ANN

A multilayer perceptron consists of neurons in fully connected layers, which are input and output layers and the layers in between may contains several number of hidden

layers. Deeper the network, better accuracy with compromising the power efficiency. Therefore in our proposed design, the maximum number of layers and number of neuron are user-defined parameters. The proposed ANN architecture is shown in Fig. 2.3 described, the layer1 is conventional which having parallel neuron for the computation whereas in layer2 is with a single neuron. The layer2 is having a unique architecture with a single neuron and work as a serialize intermediate between the two conventional layers. The configurable ANN core layer can be increase and re-synthesize for the deep feature application. For digital implementation of neural network the precision of the various blocks is an important part. However effective hardware implementation can be obtained by optimizing the resolution for arithmetic representations.

2.3.1 Proposed ANN Implementation

ANNs are computationally expensive for data-intensive applications, which may contain a large amount of neurons and several number of parameters. Consequently, a significant amount of research effort has been spent to implement hardware based neural networks in order to achieve high energy-efficiency. To overcome the above issues proposed a configurable ANN accelerator which have two types of layer. The layer1 is an array of neuron and layer2 is an implemented using single neuron as shown in Fig. 2.3. The second multiplexed layer with single neuron can be reused for multiple neurons of the previous layer and it advantage is better resource utilization. However, its latency depends on the number of input from the privious layer. The fully connectred and parallel neuron architecture in layer1 is the most common technique at it is used in proposed architecture which receives input serially using AXI interface. The method is adapted from the serialized and Deserialized application to minimize internal traffic[23]. In addition to running back-propagation, this phase selects a network topology that balances between accuracy and efficiency.

The designed hardware neural network can satisfy the given output constraints with notable energy gain. Our first experiment is conducted to compare original energy consumptions on memory and computation, and to evaluate the energy benefits of layer multiplexed ANN against the fully connected implementation. Here we want to emphasise that it is necessary to consider both memory and computation to get better

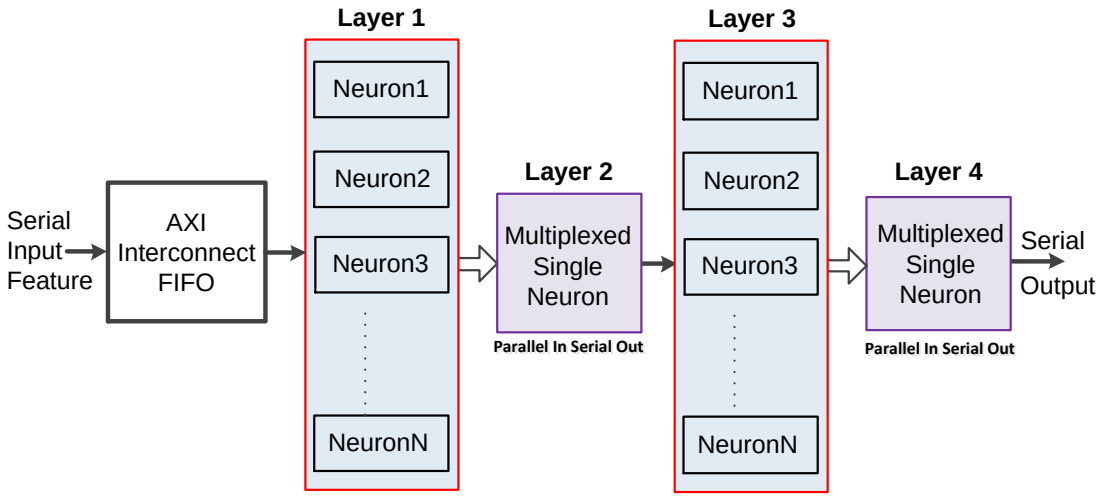

Figure 2.3: Configurable multiplexed 4 layer ANN core that can be used for pattern recognition. It consist of array of neuron layer1/3 and intermediate to serialize, layer multiplexed single neurons layer2. A real-world ANN can have several layer depends on pattern extraction

energy efficiency on FPGA based neural network designs. The fixed-point format in place of conventional floating-point representation comes with two advantages, one is that fixed point computational units are usually fast and consume less hardware and power resources as compared to floating point format and second advantages is memory footprint will be reduced thus allowing larger models in given memory capacity. This dramatically increases data level parallelism and also smaller logic of fixed point circuits will allow more instantiation for the given area.

The weight and bias memories are stored in BRAM through BRAM port and it communicates with ANN core through on-chip processor. The architecture of single neuron used in the proposed ANN core is shown in Fig. 4.2. ANN architecture is comprised of intensive MACs units were multiplication and accumulation operation is done and followed non-linear transformations. This non-linear transformation present along with MAC unit at each neuron. An optimized LUT for our approx piece-wise linear sigmoid activation function is implemented.

2.3.2 Dynamic Multiply and Accumulate (MAC) Engine

The fixed point MAC is designed, as floating point MAC have a high complexity and power consumption. The limitation with fixed-point based system cannot express the

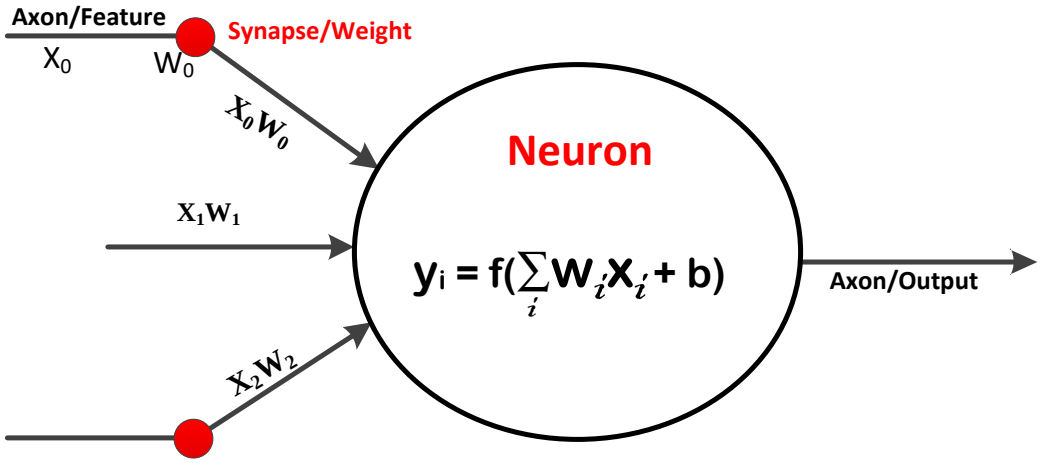

Figure 2.4: Neuron connection in architecture, x_i, w_i, b, and $f(.)$ are the input feature, weight, bias, non-linear function

wide range of variable. We have exploited the basic MAC unit presented in previous works to have leverage for the operations in ANN algorithm. Modification in MAC includes constant data path from BRAM and MUX to elaborate control of arithmetic for learning and run time processing as illustrated in Fig.2.5. The bit length of the internal MAC paths is depends on the amount of data transmission between on-chip BRAM and MAC. Moreover, we could exploit fixed-point bit which cover the total dynamic range of ANN computing. From the previous research analysis[18], 16-bit data width for weight /bias are sufficient to express the inherent accuracy. To obtain the minimum size of 8-bit activation function, the MAC unit representation is described as:

$$a_j^l = \sigma(\sum \omega_{jk}^l a_k^{l-1} + b_j^l) \tag{2.1}$$

Where,

k corresponds to overall neurons in $(l\text{-}1)^{th}$ layer. To formulate this equation in matrix form, we assume a weight matrix ω^l corresponding to each layer l. Elements of weight matrix ω^l are weights to the inputs of neurons from l^{th} layer with j^{th} row and k^{th} column.

When input feature enters to MAC and multiplier enable is active, multiplication is calculated first. Applying heterogeneous data path to fix-point MAC proposed a configurable MAC to design ANN core. As described in Fig. 2.5, configurable MAC is used

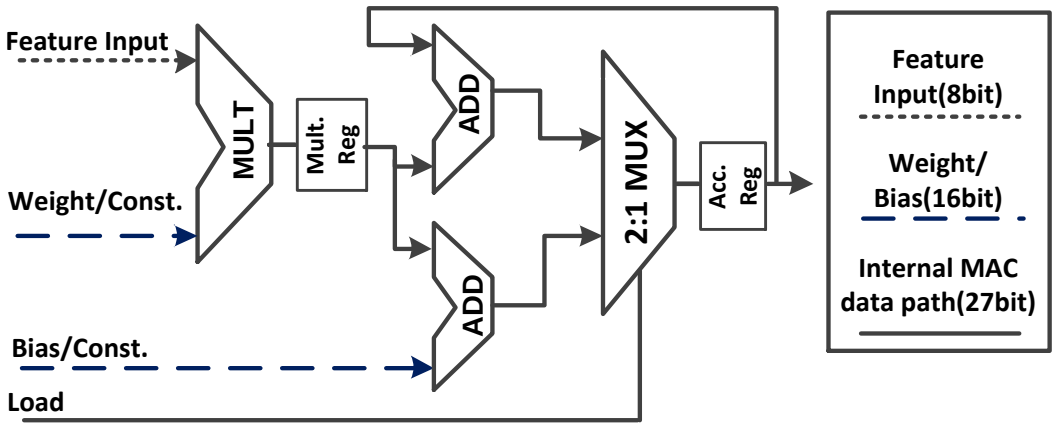

Figure 2.5: The proposed configurable fixed-point MACs with date representation schema.

for multiplication between the input feature and weight constant. A crucial advantage here is that the MAC will be used in both types of the layer which are fully connected and multiplexed. The multiplexed architecture greatly reduced power and resources utilization than the conventional technique.

The fixed point MAC is designed, as floating point MAC have a high complexity and power consumption. The limitation with fixed-point based system cannot express the wide range of variable. The Proposed MAC designed with minimum bit resolution with higher accuracy with lowest power & area utilization. Fig. 2.6 shows the internal data path with bit resolution by the configurable MAC architecture. The simple but very useful principle of multiplexing is used to minimize the interlayer connections. An optimized configurable MAC is design to suitable for both type of layers. The working of the proposed design is in two states control by the load line. The load signal is to configure the data flow path that responsible to set or reset the accumulator i.e. the accumulator data path for bias constant.

The accumulator and multiplier resisters is depends on the input data width. However for design purpose we used 8-bit feature input data-width and 16-bit for weight and Bias. The accumulator registers is taken in such a way that to secure overflow bit. We have take eight neuron in the layer1 accordingly 3-bits is used to avoid overflow. The internal MAC paths bit width is admissible to the BRAM data access by MAC unit. We have design MAC with internal data paths which supports to both layer features. To specify controls of arithmetic, constant registers and MUXes are used in configurable

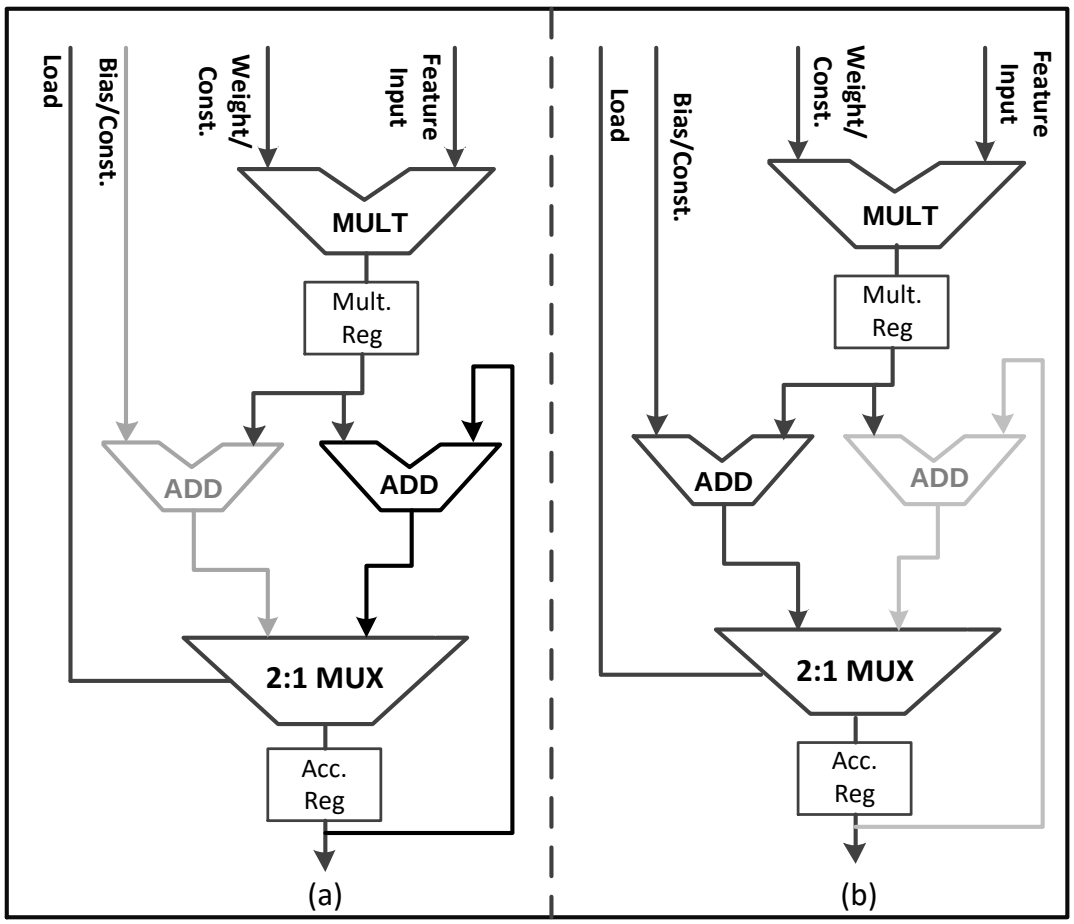

Figure 2.6: The operation of proposed configurable fixed-point MAC in FPGAs. Fig.(a) shows the operation in multiplexed single neuron in layer2 and Fig.(b) shows the operation of MAC in array of neuron in Layer1.

MAC design. Fig. 2.6(a) shows the architecture of MAC used by the ANN core in multiplexed single neuron layer2 and it fully control the parallel data in to serial flow with the help of adder tree to minimize the interlayer complex connection. The accumulator register is gating controlled by result out instruction to send serialized data to the array of next layer neurons. Fig. 2.6(b) explains the active data path in layer1, array of neurons, by multiplication and addition with bias. It is used conventionally for fully connected layer.

2.3.3 Design of LUT based Sigmoid Activation Function(AF)

Sigmoid Activation Function is a significant part of ANN literature due to its most accurate non-linear mapping among others. But when it comes to implementation of

sigmoid in FPGA, it becomes a difficult task. Solutions of the non linear problem are subject to fast computing ability of neural network.

$$y = f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The Eq. 2.2 with some modification based on the PWL approximation is realized and derived in Algorithm. 2. The implementation is best done by appropriate methods to reduced Gate Delay and chip area. The idea is to have approximate a non-linear activation function into a lookup table based piecewise linear (PWL) approximate model.

In piecewise linear approximation, the non-linear model is divided into several approximate linear segments. The approximation induces error in all neural networks. Researches done in[28] and[29] shows that non-linear activation function increase learning performance and provide higher accuracy. However, implementation of non-linear activation function will lead to high silicon area consumption and reduced the speed of operation. Therefore by using a non-linear activation function in designing ANN hardware will lower the area consumption and improves speed.

The piecewise nonlinear approximation is similar to the PWL method with the difference that nonlinear approximation is used in each segment. This method is used in[30] to approximate the sigmoid function and scheme 4 of[31] is proposed for approximating both sigmoid and hyperbolic tangent.

Table 2.1: LUT based Activation Function Implementation

Features/Resources	Utilization	Avalable
LUT	1	17600
FF	1	35200
BRAM	0.5	60

Algorithm 2 LUT Based piecewise linear Sigmoid AF (8-bit)

Input: In: Integer range $(2^n - 1)$ downto 0;
Output: Out: Encoded N-bit output

1: The excitation function is highly nonlinear, unique procedure needed to obtain an LUT of minimum size. Let n be the input bit Activation function, will having 2^n number of LUT values,

2: Let x_1 & x_2 be the lower and upper limits of the input range and the range of output (y) is between 2^{-N} and $1-2^{-N}$, It is found that x_1=$+ln(2^n-1)$ and x_2=$-ln(2^n-1)$

3: To produce change in quantize output (Δy) equals to 2^{-n} for the log-sigmoid activation function, the change in input (Δx) i.e Gap between the two input points for N-bit LUTs.

$$(\Delta x) = \frac{2 \times Input\ Margin}{2^n} \tag{2.3}$$

4: *Generation of Look-Up-Table (Slope at the origin=0.5).*

for index In -2^{n-1} to 2^{n-1}-1 **do**

$$y <= \frac{2.0}{1.0 + exp^{\left(\frac{-4.0 \times Slop}{InputRange}\right) \times x}} - 1 \tag{2.4}$$

$x := x + input\ scale(\Delta x)$; ... add scale value
end for

2.3.4 Benchmark Methodology for Hardware implementation

We have used the popular mixed national institute of standard (MNIST) handwritten digit recognition dataset [32] in test and verification of our hardware design as well as in simulation model. For hardware implementation reduced bit width as well as approximate computation, is usually adopted for efficient design implementation. Reduced bit width incorporates truncation of bits, rounding off an energy efficient output of each operator. Proposed ANN core feature is a piecewise linear log-sigmoid activation function and truncation of product obtained by multiplication. The accurate simulation model is used to verify the architecture with specified bit width, approximate customized hardware.

Table 2.2: Resource Utilization for XC7Z010clg400

Features/Resources	Utilization	Available	Utilization%
LUT	4071	17600	23.13
LUT-RAM	591	6000	9.90
FF	4591	35200	13.50
BRAM	2	60	3.33

2.3.5 Hardware Evaluation and Verification of the Proposed Accelerator

A throughput of every clock cycle produced the valid Output data. The input data is sent serially to the on-chip reconfigurable ANN core. If data are transmitted to the ANN accelerator with validation key, the core used weight and bias memory which contained in the memory block(BRAM). The fix-point format is used to performed computation and achieved good performance. In the proposed architecture overflow issues are avoided by saturating the result to the most positive or negative values when needed to reduce the bit length.

The FPGA-based approach in [23],[33] proposed an FPGA-based reconfigurable accelerator, which provides higher performance compared to a CPU-based software implementation of ANN. The previous architecture implementation performance results are compared with proposed architecture by introducing the layer multiplexing technique to serialized intermediate data. The output at the core is generated with a throughput of each clock cycle. The check-bit (validation) is asserted for every clock cycle with valid output. The internal layer signal and intermediate resister are sized to obviate overflow. The proposed ANN core has been validated through simulation and FPGA base runtime application.

2.4 Results and Discussion

FPGA platform environment (Zybo xc7z010clg400) is used to implement proposed ANN architecture. The Xilinx Zybo (xc7z010clg400) FPGA contains 4,400 logic slices, each logic cell include four 6-input LUTs and 8 flip-flops additionally it provide 240

Table 2.3: Dynamic Power Comparision for FPGA based 4 layer ANN Accelerator

Dynamic Power	Layer Multiplexed Power (mW)	Fully Connected Power (mW)	Saving power %
Clock	18	30	40
Signal	7	39	82
Logic	6	17	64
BRAM	1	2	50
Total	32	88	63

KB of BRAM.

The resource utilization by the proposed configurable 4-layer(8-1-8-10) ANNs is shown in table2.3 as it mapped into the xc7z010clg400 FPGA device. The clock frequency is optimize and operates up to $100MHz$ and worst pulse width slack for four layer network is $3.75ns$. The resource utilization by the architecture is increases proportionally with deeper neural network. It is possible to implement more deeper ANN network i.e to integrate larger nodes in the same SoC by reducing the bit width of weight and bias memory. For instance we experience the 8-bit width of weight and bias instead of 16-bit in order to get more deeper network. If we use lower precision(9-bit) for weight and bias width more deeper network can be achievable in the same resources but in the previous research[18], [34] state that lower than 16-bit width is not appropriate for the weight and bias precision in terms of accuracy.

The implemented design suitable for computing and get maximum throughput, the image size is used $14 \times 14 = 196$ pixel image which achieved by processing the original image. By comparing the results of the proposed architecture with the conventional fully connected ANN for the same layer, we have the following observation: The energy consumption of implementation configurable NN core is 01% compare to overall embedded application. The table 2.2 describes the details about resources utilization by the proposed ANN core. To save the power and get the maximum accuracy we used lowest bit precision in layer multiplexed ANN. However it gives 63% power saving by compare to fully connected with same precision architecture shown in Table 2.3.

2.5 Conclusion

The configurable, high performance and highly scalable FPGA-based layer multiplexed ANN architecture have been implemented which more efficient to accelerate the feed-forward neural network. The proposed accelerator using a full pipeline architecture dramatically accelerates the batch learning. In addition, the proposed pipeline architecture realize multilayer network with the minimum resource. To stored weight/bias we used on-chip BRAM memory, and thereby improved computational throughput and power efficiency unlike DRAM used by previous configurable ANN.

To further enhance the architecture performance, we also apply a low power approximation and LUT based technique. The proposed FPGA device based architecture implemented and operates at $100MHz$ moreover for each clock gets the output at the ANN core. The proposed architecture is 4 times more power efficient 0.5 times resource utilized as a comparison to the conventional approach. By using on-chip BRAM memory to store weight and bias and thereby improved computational throughput and power efficiency unlike DRAM used by previous state-of-the-art.

Chapter 3

Efficient Low-Precision CORDIC algorithm for Hardware Implementation of Artificial Neural Network

An efficient FPGA or ASIC based hardware implementation of deep neural networks face the challenge of limited chip area, and therefore an area efficient architecture is required to fully harness the capacity of parallel processing of FPGA and ASIC in contrast to general purpose processors. In literature, the challenges are to investigate a generalized mathematical model and architecture for neuron block in an ANN implementation. We have proposed a generalized architecture for neuron implementation based on the shift-and-add algorithm, collectively known as Coordinate Rotation Digital Computer (CORDIC) algorithm, having a wide range of application. The look-up-table (LUT) based approach with a shift-and-add algorithm is an alternative technique for polynomial approximation and implementation. This chapter explains how the CORDIC algorithm works and investigates the power and area efficient versatile computational unit for ANN application. The derived model proves that for the hyperbolic tangent function required a double pseudo-rotation and additional subtraction compares to the sigmoid function. In this reference versatile approach based optimized sigmoid activation function is implemented. The function is synthesized and validate on Xilinx zynq

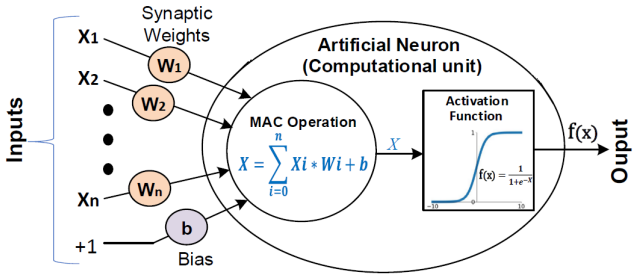

Figure 3.1: Artificial single neuron model architecture

XC7Z010clg400 SoC and result reveals the minimum resources utilization.

3.1 Introduction

Many researchers developed attractive hardware architecture for real-time inference of deep neural network. An artificial neural network (ANN) is very popular for many problems that are very difficult for the other computational model like image processing, pattern recognition, prediction, and classification [35]. The use of hardware architectures gives a more parallel structure of ANNs for the desire to optimize performance or reduce the cost of the implementation, particularly for applications demanding high parallel computation. Consequently, a significant amount of research effort has been spent on the hardware implementation of neural networks to achieve high energy and area efficient. However, many unique disadvantages are with the hardware platforms such as limitations with high data precision which has relation to hardware cost of the necessary computation, and the reconfigurability in the hardware implementation compared to software [15]. The neural network has been implemented using both software and hardware. However, researchers focused on hardware implementation as it is faster than the software part.

The implementing of digital circuits using conventional digital design procedure tends to take more chip area on Field Programmable Gate Arrays (FPGAs) than the semi-custom or full custom Application Specific Integrated Circuits (ASICs) design approach [14]. Moreover, it is efficient, flexibility to reuse and reconfigure to optimize performance for any specific type of application [16]. The main focus of this article is on accelerating ANN on small-sized FPGAs [36], each neuron contains a compu-

tational unit having multiply-accumulate (MAC) followed by activation function (AF). Whereas, each MAC unit is relatively expensive in terms of power and area of hardware floor area (i.e., large numbers of gates) in FPGA and ASIC. The architecture of the computational unit and type of AF is the choice for design. However, the mostly "sigmoid" or "squashing" functions are used which compresses an infinite input signal range to finite output signal range, e.g., [-1, +1]. Research on ANN architecture is well enough; the MAC unit is still untouched. Many methods are implemented for elementary function evaluation like look-up-table based interpolation [37], CORDIC and Polynomial approximations.

In many application such as a built-in multiplier or dedicated hardware for sigmoid function, the CORDIC algorithm is having a good compromise of accuracy versus speed. The neural network has been implemented on both digital and analog platforms, but because digital circuits are easy to design, cheaper, and have noise immunity, they are preferred over analog implementation. For many controllers and FPGAs, logic implementation using the CORDIC algorithm is over four times more efficient [38]. The size of the LUTs will decide the accuracy of the evaluated function. A CORDIC uses only adders to compute the result, with the benefit that it can, therefore, be implemented using relatively basic hardware. The CORDIC algorithm is used especially for trigonometric calculation which also can calculate other useful operation such as arithmetic operation, logarithmic and hyperbolic function calculation, and exponentiation listed by [39]. The many mathematical functions can be performed on the same hardware with efficient to reduce gate counts in FPGAs using the CORDIC algorithm.

3.2 Cordic Algorithm Explanation

The CORDIC algorithm is an iterative convergence algorithm that offers effective low-cost implementation of a all types of complex trigonometric function. A CORDIC has three inputs, x_0, y_0, and z_0. Depending on the inputs to the CORDIC, various results can be produced at the outputs x_n, y_n, and z_n. CORDIC can be used to realized the fast (digital) DSP modules with minimum resources utilization [9]-[11].CORDIC algorithm is designed in two modes,namely rotation mode and vectoring mode. The

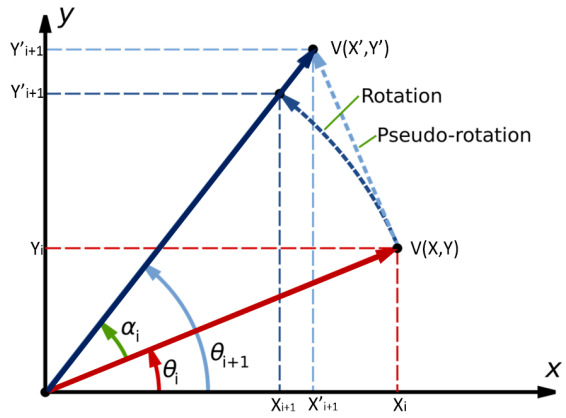

Figure 3.2: The rotation and pseudo-rotation of a vector of length about an angle with the origin. The pseudo-rotation is the key idea about the CORDIC circuit for computation and realization of mathematical functions

vector is rotated through an angle z in rotation mode, where z is decomposed of small rotating angle.The general equation [40] of CORDIC in rotation mode are as follows.

$$x_{i+1} = x_i - m \cdot y_i \cdot d_i \cdot 2^{-i} \qquad y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot tan^{-1} 2^{-i} \qquad z_n = z_0 + tan^{-1}(y_0/x_0)$$

where $d_i = +1 \quad if \quad y_i \leq 0$ -1 otherwise

The domain of convergence is$-99.7^\circ < z_0 < 99.7^\circ$ because it is the sum of all angles in the list of Table 4.2. In the rotation mode, the given vector is rotated through an angle h, where h is decomposed using a finite number of small elementary angles. The CORDIC rotation and vectoring algorithms having rotation angle between $-\pi/2$ and $\pi/2$. This limitation is due to the use of2^0 for the tangent in the first iteration.m the mode variable, is included in general equation to realize trigonometric, inverse trigonometric, linear and hyperbolic functions. Here trigonometric and inverse
trigonometric functions are realized using m = -1. Whereas,linear and hyperbolic functions are realized using m = 0 and 1 respectively.

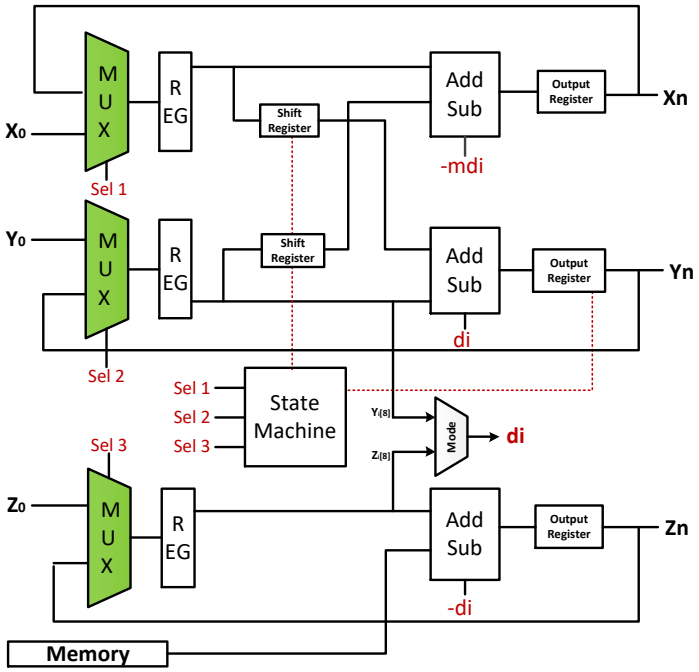

Figure 3.3: CORDIC algorithm based proposed architecture for versatile computation

3.3 Exploration of Computational Unit for ANN using CORDIC algorithm

Each neuron having a computational unit which includes the MAC and activation function. The multiplications and accumulations are performed in each MAC. Moreover, multiplier and adder can be implemented with different techniques that are accurate or approximate depending on the design objective. The objectives are area, power, design time, reliability, accuracy, etc. The energy and area improvements in operation can be achieved using the approximation technique. The fixed point Computational Unit/MAC is designed, as floating point MAC has high complexity and power consumption. The limitation with the fixed-point based system cannot express the wide range of the variable. Moreover, weight/biased precision selection is one of the important choices in resource-limited hardware implementation. The MAC operation can be realized using CORDIC architecture in Linear mode. In linear mode, the mode variable m=0 modifies the general linear mode equation as follows:

$x_n = x_0 \qquad z_n = 0$

$y_n = y_0 + x_0 * z_0$

where z_0 and y_0 represent the corresponding weight and bias value and x_0 represents the input. In Fig. 4.7 shows the input parameters and the look-up-values are stored in memory. The state machine is needed to generate select signals sel1, sel2 and sel3 to complete the feedback input after each iteration and the counter is incremented and used for shift operation in next the iteration. The d_i is generated from the $sgn(y_i)$ and $sgn(z_i)$ depending upon whether it is operating on rotation or vectoring mode. The most significant bit (MSB) will be used to decide the value of signum function which in turn will decide the whether addition or subtraction should be performed.The multiplier and accumulator resistors depends on the input data width. Applying heterogeneous data path to fix-point MAC, the power efficient architeture can be designed using CORDIC algorithm. Each computational unit is performing their computation update the outputs from the input in the preceding layer as shown here.

$$a_j^l = \sigma(\sum \omega_{jk}^l a_k^{l-1} + b_j^l) \tag{3.1}$$

where σ is the squashing function (sigmoid) of computational unit k, corresponds to overall neurons in (l-1)th layer. To formulate this equation in matrix form, we assume a weight matrix ω^l corresponding to each layer l. Elements of weight matrix ω^l are weights to the inputs of neurons from l^{th} layer with j^{th} row and k^{th} column. The CORDIC algorithm for multiplication is derived by using a series representation for weight in the l_{th} layer as shown in Eq. 4.1 as follows:

$$x_j = x_k * w_{jk} \tag{3.2}$$

$$= \omega_{jk} * \sum_{i=1}^{j} a_i * 2^{-i} \tag{3.3}$$

$$= \sum_{i=1}^{j} \omega_{jk} * a_i * 2^{-i} = \sum_{i=1}^{j} a_i * \omega_{jk} * 2^{-i} \tag{3.4}$$

The equation state that x_j is composed of a shifted version of weight ω.The unknown coefficient a_i may be found by driving x to zero one bit at a time. If the i^{th} bit of input x_k is non-zero, y_i is first right shifted by i bits and added to the current value of x_j. When x has been driven to zero all bits have been examined and x_j contains the signed product

of input vector and weight. The implementation is based on the standard shift and add multiplication.The calculation is considered for the weight ranges from -1 to $+1$.

3.4 Activation Function Design Technique

The hardware implement of activation function, various approaches have been proposed. Moreover, these methods are fall in to two categories piecewise linear approximation and Look-up table based approaches [30]. In this work, we consider the four mostly used approaches to make review concise.

3.4.1 LUTs based implementation by storing function values

The most common method used is look-up tables based hardware implementation of activation function. The approximated into discrete values of the function and store in the LUT. Moreover, LUT implementations required more storage and increasing latency.

3.4.2 LUTs based implementation by storing parameters

The activation function is a continuous function. Instead of storing the function values directly, this method keeps the function slope and the function intercept in the LUT. The outlook for implementation is storing function in piecewise with a different slope, value can be calculated using the equation.

$$y = kx + c \tag{3.5}$$

where k is the slope and c is the function intercept. The output calculation depends on the above parameters and this approach leads to higher accuracy. However, used data by the multiplier and adder to calculate the equation needs more storage.

3.4.3 Approximation in Calculation

The approximation is done in many ways, the exponential function implementation is expensive but if it is converted into base 2 then efficient to implement. In [41] used the following a formula to approximate the exponential function.

$$e^x \approx exp(x) \approx 2^{1.44x} \tag{3.6}$$

The sigmoid function can be calculated using this approximated function as:

$$Sigmoid(x) \approx \frac{1}{1+2^{-1.44x}} \approx \frac{1}{1+e^{-1.5x}}$$

similarly for tanh function can be calculated using this approximation. Approximation technique can realized sigmoid and tanh function in single block with different operation but it requires more clock cycle. Where base-2 calculation and addition/subtraction required different clock cycle which decreases the overall performance. For the tanh function required one more clock cycle for an additional block. It takes minimum hardware utilization but has more latency which affects learning period.

3.4.4 Implementation using CORDIC algorithm

The sigmoid function presents a problem for direct hardware implementation since both the division and exponential operation. The tanh function passes through zero and can be treated as y = x around to zero unlike to sigmoid function. However, the implementation of sigmoid function in piece-wise linear (PWL) approximation, the non-linear model is divided into several approximate linear segments. Research done in [28], [29] shows that non-linear activation function increases learning performance and provide higher accuracy. However, hardware implementation of non-linear activation function will lead to high silicon area consumption and reduced the speed of operation.

The sigmoid or hyperbolic tangent activation function is realized by making mode variable m= -1, and operating in vectoring mode where d_i will be decided by signum of y_i. The equation for realizing hyperbolic tangent is given by equation of Z_n from the following equation.

$$x_n = A_n\sqrt{x_0^2 - y_0^2}$$

$$y_n = 0$$

$$z_n = z_0 + tanh^{-1}(y_0/x_0)$$

$$A_n = \prod_n \sqrt{1 - 2^{-2i}}$$

The sigmoid function is realized through tan hyperbolic as follows:

$$sinh(x) = \frac{e^x - e^{-x}}{2} \;\; \& \;\; cosh(x) = \frac{e^x + e^{-x}}{2} \tag{3.7}$$

$$sinh(x) + cosh(x) = e^x \;\; \& \;\; sinh(x) - cosh(x) = e^{-x} \tag{3.8}$$

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.9}$$

$$g(x) = Sigmoid(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} \tag{3.10}$$

By solving the Equation 3.9 and 3.10, the relation between the tan hyperbolic and sigmoid is

$$tanh(x) = 1 - 2\ Sigmoid(-2x) \tag{3.11}$$

$$Sigmoid(x) = \frac{1 + tanh\,(x/2)}{2} \tag{3.12}$$

We should have generalized design for sigmoid function approximation which should be applicable for a variety of sigmoid function. The tanh function passes through zero and can be treated as y = x around to zero unlike to sigmoid function [42]. In principle, sigmoid and tanh have likely expressive ability, but in practice, sigmoid is just a activation function with constant. When the output of neuron is restricted [0, 1], the neuron activation is more likely closed to0.5. The activation range need more saturation with sigmoid than with tanh using the conventional technique [43].However, the implementation of sigmoid function requires half pseudo-rotation steps compare to the tan hyperbolic function shown in Equation 3.12.We can conclude that the sigmoid activation function is the best choice for implementation using the CORDIC algorithm.

Table 3.1

i	$\alpha_i = tan^{-1}(2^{-i})$	
	Degrees	Radians
0	45.00	0.7854
1	26.57	0.4636
2	14.04	0.2450
3	7.13	0.1244
4	3.58	0.0624
5	1.79	0.0312
6	0.90	0.0160
7	0.45	0.0080
8	0.22	0.0040
9	0.11	0.0020

Chapter 4

Low Complexity Approximate Multiply-Accumulate Unit for FPGA-Based Artificial Neural Network Accelerator

Many error resilient applications can be approximated using multi-layer perceptron (MLP) with insignificant degradation in output quality on hardware platforms. Field programmable Gate Arrays (FPGAs) and Application specification Integrated Circuit (ASICs), have edge over Graphics Processing Units (GPUs) on cost. We have discussed an error resilient MLP by approximation in computation unit. In literature, the challenge is to investigate an ANN architecture, especially in pattern recognition with less hardware and high performance. Further, the considerable computational requirements stretch the capabilities of even modern computing platforms. Hence, we have proposed, the power and area efficient novel approximate multiplier and used for approximating computation unit in the ANN Accelerator. The results are compared with state-of-the-art available approximate and accurate multipliers used in the neural accelerator. The figure-of-merits of proposed approximate multiplier is of 2.18 times, however the probability of error is 12.5% only. In this connection, we have used the approximate technique for tolerable accuracy loss in detection and classification. We have also used the look-up-table (LUT) based approach to realize the nonlinear activation

function with approximation scheme with lesser area and delay. Further, the on-chip memory with optimized technique is being used to access the weight and bias values of trained ANN network. The design of artificial neural network accelerator is validated on Xilinx Zynq XC7Z010clg400 SoC, Zybo board at 100 MHz clock frequency.

4.1 Introduction

The main focus of this chapter is on accelerating ANN on small-sized FPGAs, and mitigate circuit complexity using approximation in multiplication. In addition we focus on demonstrating reconfigurable algorithm for BRAM to access the weight and bias. The target of this application is to minimize the resources utilization and power consumption. Therefore, we design hardware Intellectual Property (IP) core for ANN. The hardware IP core are implemented with Xilinx Vivado, and in multiply-accumulate (MAC) unit exact multiplier is replace with approximate. In summary, the key contributions of this work are as follows:

- Design and analysis of the proposed power and area efficient approximate multiplier to realize the optimized and configurable MAC computation unit used in ANN accelerator.
- The use of LUT based approach for energy efficient peace-wise linear activation function implementation helps us to achieve the lesser area of delay.

4.2 Releted Work

ANN are data-flow driven and offer great possibilities to design hardware accelerator and co-processor for better parallelism. Due to the recent success of deep neural networks (DNNs), an ANN for pattern recognition application, the many researcher focusing to accelerate the ANN inference phase using FPGA bases hardware implementation. Recent research on FPGA based ANN, many architectures have been proposed [44]. The FPGA's parallel computational architecture matches the topologies of ANNs, even

their are quite limitations for the ANNs implementation. Furthermore, FPGA based implementation of ANN requires an appropriate number system to represent the weights and biases i.e. 8-bit, 16-bit, 32-bit or 64-bit floating & fixed point format is used as data width. However, fixed point is used when resources are limited and some degree of error can be tolerated [18].

The most typical computational unit in a ANNs is MAC and the hardware implementation of neural network contain innumerable MAC units. Whereas, each MAC unit is a relatively expensive in terms of power and area of hardware floor area (i.e., large numbers of gates) in FPGA and ASIC. Research on ANN architecture is well enough, the MAC unit is still untouched. Performance of MAC unit contains multiplier and adder. So, as the multiplier is more complex compare to adder overall performance and speed of DSP system will determine by the multiplier. The strict need of high precision for computation, while other parameters is also consider such as hardware complexity, power dissipation of a design and delay. The application which are error-resilient, approximate computing are naturally used to achieve great energy and area savings with minor quality degradation [45]. The approximate arithmetic unit can be employed [21]. The key points to resolve this problem is to have an efficient novel approximate multiplier based MAC framework for ANN.

Interesting FPGA implementation schemes, especially using Xilinx FPGAs are described in the book edited by Ormandi and Rajapakse [21]. The [46] describes the usage of dynamic and partial reconfiguration for re-use of chip area on FPGA. Consequently, in FPGAs our primary interest lies in their reconfigurability. Our aim to transfer the flexibility of parameter (like number of neurons, number of layers, type of quashing function, etc.) in hardware platform. To be specific, we aim to address some of these difficulties by developing and implementing a FPGA based implementation of a ANN with approximate computing. We first characterize the criticality MAC unit under approximation by jointly considering its impact on output and energy consumption, and then utilize an efficient and effective algorithm for the ANN under a given quality constraint.

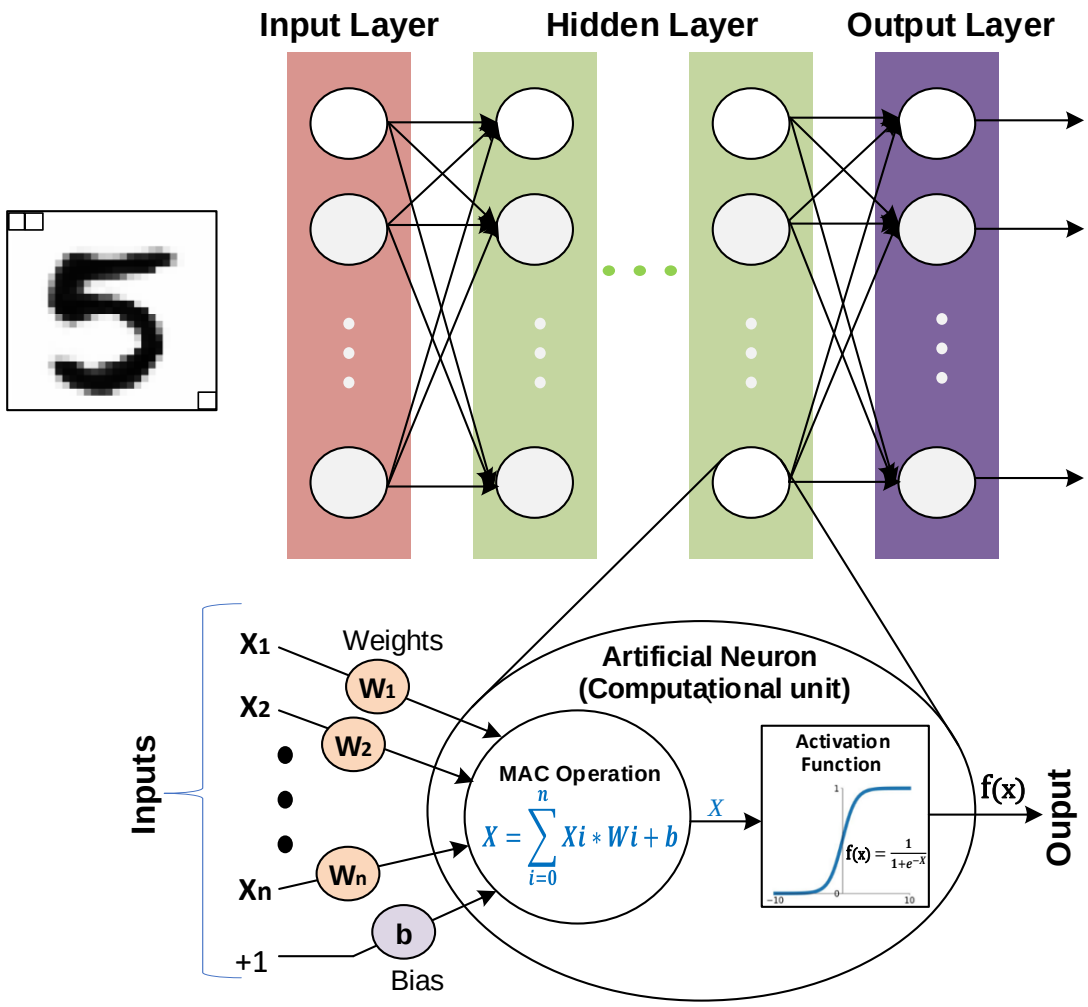

Figure 4.1: Fully connected efficient artificial neural network (ANN) architecture

4.3 An Architectural Overview

An ANN architecture contains the input, output layer and in between multiple hidden layers is shown in Fig. 4.1. Each neuron contains a computational unit having MAC followed by activation function (AF). The set of input signals, within a neural network transfer from input layer to output layer through multiple hidden layers. The mostly used ANN architecture is a multi-layered feed-forward network called MLP, i.e., the nodes (neurons) are dedicated in different layers (like input layer, hidden layers and output layer). Whereas the information flow is only between adjacent layers [47]. The architecture of computational unit and types of AF can be used which is application dependent. However, the mostly "sigmoid" or "squashing" functions is used which is compress an infinite input signal range to finite output signal range e.g., [-1, +1].

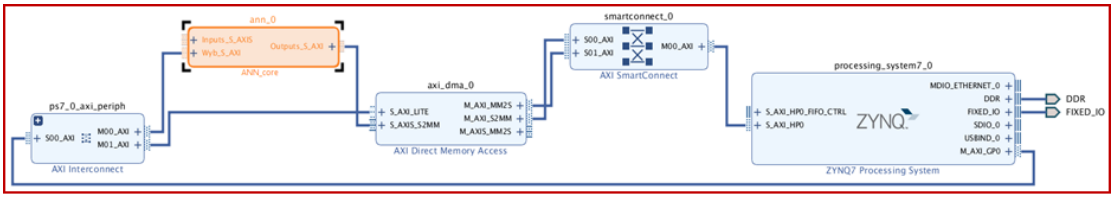

Figure 4.2: Embedded blocks design with ANN core on Vivado.

In training or testing the neural network, signals at each connection node in between the layers are updated the weights at each neuron. Those weight values are firmly updated using known back-propagation algorithm. In this process we are minimizing the calculated error through an optimization technique after each batch of inputs has finished by flowing through the network. The feed-forward neural network are connected one to another consecutive layer through the computational unit in forward direction. In this forward pass operation each computational unit is performing their computation update the outputs from the input in the preceding layer as shown here.

$$a_j^l = \sigma(\sum \omega_{jk}^l a_k^{l-1} + b_j^l) \tag{4.1}$$

where σ is the squashing function(sigmoid) of computational unit k, corresponds to overall neurons in $(l\text{-}1)^{th}$ layer. To formulate this equation in matrix form, we assume a weight matrix ω^l corresponding to each layer l. Elements of weight matrix ω^l are weights to the inputs of neurons from l^{th} layer with j^{th} row and k^{th} column.

In brief, an ANN defined by three parameters: (i) The pattern of interconnection in between the neurons of different layer; (ii) The activation function that converts indefinite to definite range of outputs; (iii) The learning process for updating the weights of the interconnections. For the hardware implementation of neural network the precision of the various blocks is an also the important part. The output resolution will depends on word length precision. However effective hardware implementation can be obtained by optimizing the resolution for arithmetic representations. Hardware platforms are, like FPGAs and ASICs, giving realizable and efficient alternative to GPUs/CPUs for ANN implementation. Especially, when the applications with strict power and performance constraints.

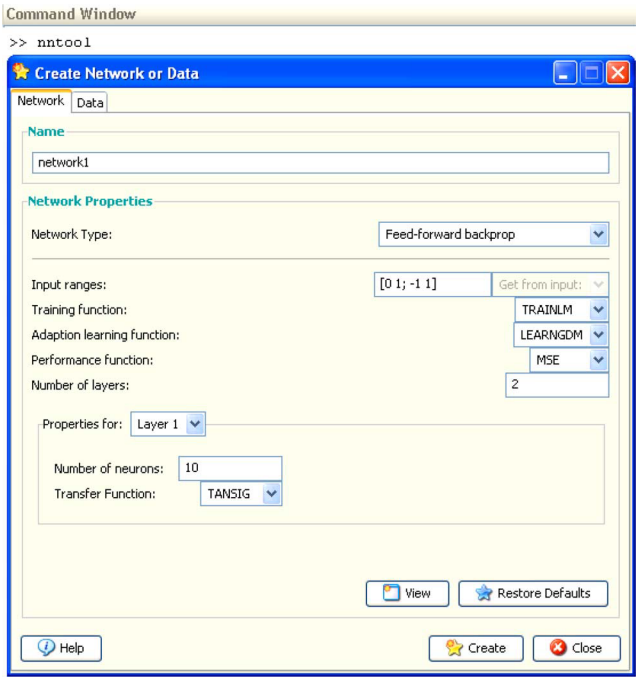

Figure 4.3: MATLAB neural network toolbox to generate graphical user interface networks [**b18**]

4.4 Hardware Implementation

This section describes the architecture of the hardware accelerator for running ANNs in feed-forward direction. The hardware implementation is done on Zybo board featuring Xilinx Zynq xc7z010clg400 SoC is shown in Fig 4.2. The FPGA has eight high-performance ports to DDR3 memory. These high-performance ports tap into DDR3 memory using the Advanced eXtensible Interface (AXI) buses. Each AXI direct memory access (DMA) is bidirectional and can transfer 128 bit data words per clock cycle per direction. The limitations and features of the design is the foremost considerations while implemented with control method. The ANN core communication is based on the AXI specification. There are three different types of interfaces. AXI4 Lite is a light-weight memory-mapped protocol, same as AXI4. Whereas, communicate and data sharing between a sender and a receiver done by AXI4 stream which is not memory-mapped, hence there is no addressing overhead. Due to the data driven flow of ANNs, we used the AXI4 stream based interfaces input to the ANN core and output from the core with PS for a high data through put. The data transfer of ANN is depends on additional parameter of processing and hence to transport the weight and bias parameters,

we utilized a memory mapped AXI4 interfaces with an adjusted burst size. While, the input data to ANN hardware IP core are connected via a simple DMA IP core.

The configurable ANN core is realized in programmable logic (PL) part of Zynq device. The input and output side of ANN core are connected to ARM host processing system (PS) of Zynq device through AXI interface protocols. The on-chip register is used to convert serial to parallel and vice versa data interface in the intermediate stage. The computational operation are performed in MAC block. The weight and bias for real time computation will access from the BRAM by the MAC unit. The control system consist of memory access architecture, finite state machine (FSM) based controlling logic and internal signal of programmable logic data path. The programmable logic ANN core is access by the processor through AXI interface at real time processing.

4.4.1 Design Architecture

In this context, it is gripping to note that at PC-based system how it can be done. The learned constants weight/bias evaluated for classification problem (MNIST), graphical user interface is used to generate networks with different feature using the MATLAB neural network toolbox [48] as shown in Fig. 4.3. The user dependent, one can choose network type, number of layers, number of neuron in each layer, activation function etc. The network is generated by considering these inputs. Of course, it is flexible in terms of data precision, network size and weight/bias constants which are given by the user configuration. The number of layers generated in a ANN network configuration as required will achieve by top level design. The ANN core is realized in VLSI hardware description language (VHDL), with fixed point package [49] proposed by IEEE. The computation unit consist of signal processing operations and storage component. These operation done by multipliers, adders, activation function, etc.

For network data representation we used a fixed point notation. Nevertheless, flexible design philosophy wants for the FPGA based implementation. Please note, this study is limited to approximate computing technique implementation in feed-forward MLP. The design is configurable in terms of design parameters like number of layers, number of neurons, input-output sizes, types of interconnection, etc. However these parameters

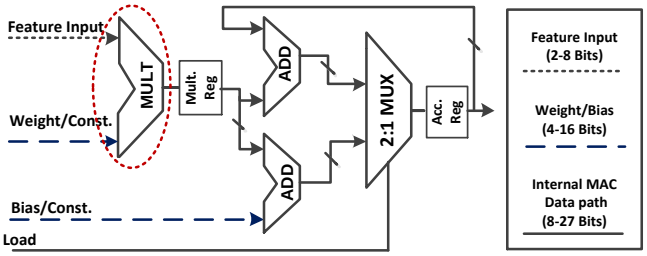

Figure 4.4: The proposed configurable fixed-point MAC with data representation schema.

are limited by the hardware resources available, depending on the type of FPGA used. ANN architecture is comprised of intensive MACs units were multiplication and accumulation operation is done and followed non-linear transformations. This non-linear transformation present along with MAC unit at each neuron.

4.4.2 Computational Unit

We say one neuron is critical, if small jitter on this neuron's computation introduces large final output quality degradation; otherwise, it is resilient. The each neurons having computational unit which includes the MAC and activation function. The multiplications and accumulations are performed in each MAC. Moreover, multiplier and adder can be implemented with different techniques that are accurate or approximate depending on design objective. The objectives are area, power, design time, reliability, accuracy etc. The energy and area improvements in operation can be achieved using approximation technique.

4.4.2.1 Multiply-Accumulate Unit

The fixed point MAC is designed, as floating point MAC have a high complexity and power consumption. The limitation with fixed-point based system cannot express the wide range of variable. Moreover, weight/biased precision selection is one of the important choices in resource limited hardware implementation. The proposed MAC designed with minimum bit resolution and approximate multiplier shown in Fig. 4.4 gives lowest power & area of utilization. The internal data path of MAC unit is depends on the sequential computation has to be perform by the single neuron and it depends on

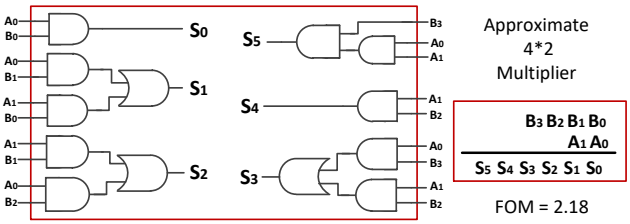

Figure 4.5: The proposed fixed-point 4×2 approximate multiplier

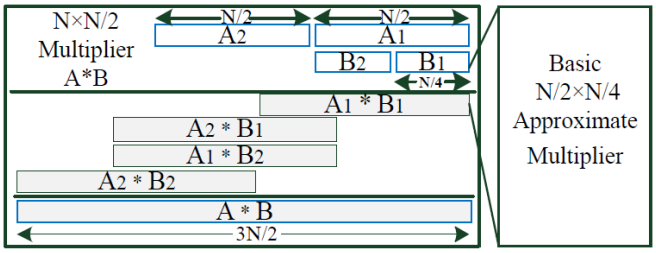

Figure 4.6: The design fixed-point 8×4 using proposed approximate 4×2 multiplier

number of neuron in the previous layer. The working of the MAC design is in two states control by the load line. The load signal is to configure the data flow path that responsible to set or reset the accumulator i.e. the accumulator data path with bias. The multiplier and accumulator resisters are depends on the input data width. However for design purpose we used 4-bit feature input data-width and 8-bit for weight and bias. When input feature enters to MAC and multiplier enable is active and multiplication have been perform.

Applying heterogeneous data path to fix-point MAC, proposed MAC unit with approximate multiplier is used. The proposed design achieved $1.5\times$ power efficient compare to accurate multiplier and better than state-of-the-art low power multiplier. We have proposed 4×2 approximate multiplier with minimum gates shown in Fig. 4.5. Selection of weight precision is used to trade-off the capabilities of the realized ANN model against the implementation cost [50]. For the higher precision we designed 8×4 using the proposed multiplier as shown in Fig. 4.6 and used in the MAC implementation. A higher weight precision means fewer quantization errors in the final implementation, while a lower weight precision leads to simpler designs, greater speed and reduction in area requirements and power consumption. Result analyzed by synthesizing the multiplier with approximate and exact multiplication technique.

Table 4.1: LUT based Activation Function Implementation

Features/Resources	Utilization	Avalable
LUT	1	17600
FF	1	35200
BRAM	0.5	60

4.4.2.2 Squashing Function

The sigmoid function presents a problem for direct hardware implementation since both the division and exponential operation. The practical option in hardware is linearly approximate the function [51]. In piece-wise linear (PWL) approximation, the non-linear model is divided into several approximate linear segments. Research done in [28], [29] shows that non-linear activation function increases learning performance and provide higher accuracy. However, hardware implementation of non-linear activation function will lead to high silicon area consumption and reduced the speed of operation. We should have generalized design for sigmoid function approximation which should be applicable for a variety of sigmoid function.

It can be concluded that the best approximation method, in terms of resources utilized and errors introduced, PWL approximation is the option especially when the number of the neurons that use sigmoid function is larger than the number of the BRAM blocks available in the FPGA circuit. When the number of neurons is lower than the total BRAM blocks available in the FPGA circuit, the best way to approximate the sigmoid function is the lookup table based method. Based on size, speed and accuracy, we have implemented the linear approximation in resulting activation function shown in algorithm 2. We include the linear interpolation along with the LUTs. LUT-based approach works much faster than piece-wise linear approximation, though LUT consumes memory. So, if there is not much concern about memory, LUT based approach is preferred e.g., real-time applications motion control and fault diagnosis of induction motor drive [52] and in our implementations. An analysis of the trade-off between the hardware cost, look-up-table based approximate activation function is also implemented and its hardware resource utilization as shown in Table. 4.1.

4.4.3 Hardware Evaluation and Verification of the ANN Accelerator

We have implemented ANN core on FPGA with featuring approximate computation and embedded block design using PS. A throughput of every clock cycle produced the valid output data. The input data is sent serially to the on-chip ANN core architecture. If input feature data are sent to the ANN accelerator with validation key, the core will access weight and bias data which contained in the memory BRAM while real time processing. The fix point format is used to performed computation by achieving good performance. In the proposed architecture overflow issues are avoided by saturating the result to the most positive or negative values when needed to reduce the bit length. The output at the core is generated with a throughput of each clock cycle. The check-bit (validation) is asserted for every clock cycle with valid output. The internal layer signals and intermediate resisters are sized to obviate overflow. The proposed ANN core has been validated through simulation and FPGA.

4.5 Results and Discussion

We have discuss this section in two parts, one is effectiveness of approximation in multiplier and activation function whereas other part is focuses on how its benefit to the MAC unit and ANN arctitecture on FPGA.

4.5.1 Low Complexity Multiplier Architecture and LUT based PWL Activation Function

The MAC unit consists of compute part and control unit which is describe in section-iii (c). Whereas, the compute part consists of multiplier and adder. The approximate multiplier circuit under test were synthesized using Xilinx Vivado. Table. 4.2 shows the energy improvements obtain for proposed 4×2 multiplier. The proposed design achieved energy saving of 4×2 with respect to the accurate multiplier and figure-of-merits(FOM) is 2.18. The FOM states that the proposed approximate design is 2.18

Table 4.2: Performance comparison of Proposed approximate comparator

Parameters	**FPGA based 4×2 multiplier performance**	
	Proposed approximate	Conventional exact
Logic Power	12.8 mW	19.2 mW
Delay	8.195 ns	26.27 ns
Gates Count	13	28
LUTs	3	5
Probability of Error	12.5 %	0.0 %

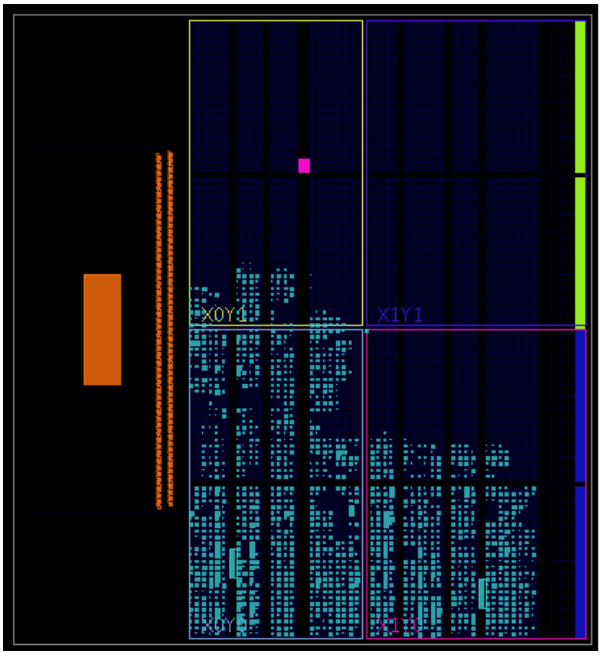

Figure 4.7: Placement of the reconfigurable ann_core area

times more efficient compares to accurate design. While considering only 12.5% probability of error. We have designed 8×4 multiplier using the proposed 4×2 multiplier for the multiply and accumulate unit as shown in Fig. 4.6. The look-up-table based approximate activation function is also implemented. Moreover its hardware resource utilization ensures that using the given technique we can implement high resolution activation function

4.5.2 Hardware Implementation using Approximate MAC

FPGA platform environment (Zybo xc7z010clg400) is used to implement proposed ANN architecture. The Xilinx Zybo (xc7z010clg400) FPGA contains 4,400 logic slices,

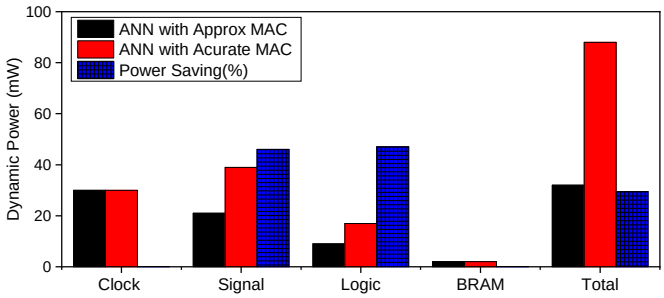

Figure 4.8: Dynamic Power Comparision for FPGA based 4(8-4-4-1) layer ANN Accelerator

each logic cell include four 6-input LUTs and 8 flip-flops additionally it provide 240 KB of BRAM. The resource utilization by the proposed configurable 4-layer(8-4-4-1) ANNs is mapped into the xc7z010clg400 FPGA device is shown in Fig. 4.7. The clock frequency is optimize and operates up to $100MHz$ and worst pulse width slack for four layer network is $3.75ns$. The resource utilization by the architecture is increases proportionally as goes deeper with the network size.

The input image is preprocessed to reduce the no of pixels on the PS before it is feed to the ANN. The implemented design suitable for computing and get maximum throughput, the image size is used $14 \times 14 = 196$ pixel image which achieved by processing the original image. By comparing the results of the proposed architecture with the conventional fully connected ANN for the same layer, we have the following observation: The energy consumption of implementation configurable NN core is 01% compare to overall embedded application. To save the power and get the maximum accuracy we used lowest bit precision in layer multiplexed ANN. However it gives 29.5% power saving by compare to fully connected with same precision architecture shown in Fig. 4.8.

4.6 Conclusion

We have proposed an approximate multiplier and used in multiply-accumulate (MAC). To further enhance the architecture performance, the LUT based power efficient algorithm is presented to implement the sigmoid. function. The achieved the figure-of-merits of the proposed multiplier is of 2.18 with 12.5% of probability of error only. We have also proposed an algorithm for configurable architecture to access the weight and

bias from on-chip BRAM memory, and thereby improved computational throughput and power efficiency unlike DRAM [53]. Several approximate multipliers are proposed in the state-of-the-art which only support for even multiplication like 2×2, 4×4, 8×8 but for odd multiplier like 4×2 the proposed design can support. The gate count and area for the approximate multiplier is significantly lower when compared with the exact multiplier. We conclude that the proposed approximate design will offer the possibility of adding more parallel MAC units in FPGA as well as ASIC based ANN accelerators. By using on-chip BRAM memory to store weight and bias thereby improved the computational throughput and power efficiency as compared to DRAM used by previous state-of-the-art [19]. The proposed FPGA based ANN core architecture operates at 100 MHz frequency.

References

[1] Howard Gardner. *Frames of mind: The theory of multiple intelligences*. Hachette UK, 2011.

[2] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai* 593598 (2002).

[3] Rodney A Brooks. "Intelligence without representation". In: *Artificial intelligence* 47.1-3 (1991), pp. 139–159.

[4] Hans Moravec. *Mind children: The future of robot and human intelligence*. Harvard University Press, 1988.

[5] John D McCarthy and Mayer N Zald. "Resource mobilization and social movements: A partial theory". In: *American journal of sociology* 82.6 (1977), pp. 1212–1241.

[6] Marvin Minsky. "A framework for representing knowledge". In: (1974).

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[8] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. "Issues in evaluation of stream learning algorithms". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 329–338.

[9] Jianming Hu, Jianzhou Wang, and Guowei Zeng. "A hybrid forecasting approach applied to wind speed time series". In: *Renewable Energy* 60 (2013), pp. 185–194.

[10] Guanwen Zhong, Smail Niar, Alok Prakash, and Tulika Mitra. “Design of multiple-target tracking system on heterogeneous system-on-chip devices”. In: *IEEE Transactions on Vehicular Technology* 65.6 (2016), pp. 4802–4812.

[11] Eric Monmasson, Lahoucine Idkhajine, Marcian N Cirstea, Imene Bahri, Alin Tisan, and Mohamed Wissem Naouar. “FPGAs in industrial control applications”. In: *IEEE Transactions on Industrial informatics* 7.2 (2011), pp. 224–243.

[12] Carlos A Parra, Khan Iftekharuddin, and Robert Kozma. “Automated brain data segmentation and pattern recognition using ANN”. In: *the Proceedings of the Computational Intelligence, Robotics and Autonomous Systems (CIRAS 03)* (2003).

[13] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, et al. “Can fpgas beat gpus in accelerating next-generation deep neural networks?” In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2017, pp. 5–14.

[14] Henry Wong, Vaughn Betz, and Jonathan Rose. “Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture”. In: *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM. 2011, pp. 5–14.

[15] Martin C Herbordt, Tom VanCourt, Yongfeng Gu, Bharat Sukhwani, Al Conti, Josh Model, and Doug DiSabello. “Achieving high performance with FPGA-based computing”. In: *Computer* 40.3 (2007).

[16] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. “Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC”. In: *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE. 2016, pp. 77–84.

[17] Jihan Zhu and Peter Sutton. “FPGA implementations of neural networks–a survey of a decade of progress”. In: *International Conference on Field Programmable Logic and Applications*. Springer. 2003, pp. 1062–1066.

[18] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. “Deep learning with limited numerical precision”. In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.

[19] Hoang Le and Viktor K Prasanna. “Scalable high throughput and power efficient ip-lookup on fpga”. In: *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE. 2009, pp. 167–174.

[20] Hiroomi Hikawa. “An efficient three-valued multilayer neural network with on-chip learning suitable for hardware implementation”. In: *Systems and computers in Japan* 31.4 (2000), pp. 43–51.

[21] Jie Han and Michael Orshansky. “Approximate computing: An emerging paradigm for energy-efficient design”. In: *Test Symposium (ETS), 2013 18th IEEE European*. IEEE. 2013, pp. 1–6.

[22] Bryan H Fletcher. “FPGA embedded processors”. In: *Embedded Systems Conference*. 2005, p. 18.

[23] S Himavathi, D Anitha, and A Muthuramalingam. “Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization”. In: *IEEE Transactions on Neural Networks* 18.3 (2007), pp. 880–888.

[24] Ravikant G Biradar, Abhishek Chatterjee, Prabhakar Mishra, and Koshy George. “FPGA implementation of a multilayer Artificial Neural Network using System-on-Chip design methodology”. In: *Cognitive Computing and Information Processing (CCIP), 2015 International Conference on*. IEEE. 2015, pp. 1–6.

[25] Murat Alçın, İhsan Pehlivan, and İsmail Koyuncu. “Hardware design and implementation of a novel ANN-based chaotic generator in FPGA”. In: *Optik-International Journal for Light and Electron Optics* 127.13 (2016), pp. 5500–5505.

[26] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. “Optimizing fpga-based accelerator design for deep convolutional neural networks”. In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2015, pp. 161–170.

[27] In: *http://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/* ().

[28] Koldo Basterretxea. “Recursive sigmoidal neurons for adaptive accuracy neural network implementations”. In: *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*. IEEE. 2012, pp. 152–158.

[29] Koldo Basterretxea, Jos Manuel Tarela, Ins del Campo, and Guillermo Bosque. “An experimental study on nonlinear function computation for neural/fuzzy hardware design”. In: *IEEE transactions on neural networks* 18.1 (2007), pp. 266–283.

[30] Ming Zhang, Stamatis Vassiliadis, and Jose G. Delgado-Frias. “Sigmoid generators for neural computing using piecewise approximations”. In: *IEEE transactions on Computers* 45.9 (1996), pp. 1045–1049.

[31] Stamatis Vassiliadis, Ming Zhang, and José G Delgado-Frias. “Elementary function generators for neural-network emulators”. In: *IEEE transactions on neural networks* 11.6 (2000), pp. 1438–1449.

[32] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998).

[33] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. “Deepx: A software accelerator for low-power deep learning inference on mobile devices”. In: *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press. 2016, p. 23.

[34] Jordan L Holi and J-N Hwang. “Finite precision error analysis of neural network hardware implementations”. In: *IEEE Transactions on Computers* 3 (1993), pp. 281–290.

[35] Mukta Paliwal and Usha A Kumar. “Neural networks and statistical techniques: A review of applications”. In: *Expert systems with applications* 36.1 (2009), pp. 2–17.

[36] Jihong Liu and Deqin Liang. “A survey of FPGA-based hardware implementation of ANNs”. In: *2005 International Conference on Neural Networks and Brain*. Vol. 2. IEEE. 2005, pp. 915–918.

[37] Masoud Sadeghian, James Stine, and E Walters. “Optimized linear, quadratic and cubic interpolators for elementary function hardware implementations”. In: *Electronics* 5.2 (2016), p. 17.

[38] Chris K Cockrum. “Implementation of the cordic algorithm in a digital downconverter”. In: (2008).

[39] Ray Andraka. “A survey of CORDIC algorithms for FPGA based computers”. In: *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*. ACM. 1998, pp. 191–200.

[40] John S Walther. “A unified algorithm for elementary functions”. In: *Proceedings of the May 18-20, 1971, spring joint computer conference*. ACM. 1971, pp. 379–385.

[41] Shaghayegh Gomar, Mitra Mirhassani, and Majid Ahmadi. “Precise digital implementations of hyperbolic tanh and sigmoid function”. In: *2016 50th Asilomar Conference on Signals, Systems and Computers*. IEEE. 2016, pp. 1586–1589.

[42] Tao Yang, Yadong Wei, Zhijun Tu, Haolun Zeng, Michel A Kinsy, Nanning Zheng, and Pengju Ren. “Design Space Exploration of Neural Network Activation Function Circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).

[43] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[44] Ravikant G Biradar, Rashmi Ugarakhod, Koshy George, and Abhishek Chatterjee. “Pipeline-design based FPGA implementation of online sequential learning algorithm”. In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE. 2017, pp. 629–634.

[45] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. “ApproxANN: An approximate computing framework for artificial neural network”. In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium. 2015, pp. 701–706.

[46] Amos R Omondi and Jagath Chandana Rajapakse. *FPGA implementations of neural networks*. Vol. 365. Springer, 2006.

[47] Abhisek Ukil. *Intelligent systems and signal processing in power engineering*. Springer Science & Business Media, 2007.

[48] Howard Demuth and Mark Beale. “Neural Network Toolbox For Use with {MATLAB}-User\’s Guide”. In: (2004).

[49] David Bishop. “Fixed point package user’s guide”. In: *Packages and bodies for the IEEE* (2010), pp. 1076–2008.

[50] Da Zhang and Hui Li. “A low cost digital implementation of feed-forward neural networks applied to a variable-speed wind turbine system”. In: *2006 37th IEEE Power Electronics Specialists Conference*. IEEE. 2006, pp. 1–6.

[51] Pedro Ferreira, Pedro Ribeiro, Ana Antunes, and Fernando Morgado Dias. “A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function”. In: *Neurocomputing* 71.1-3 (2007), pp. 71–77.

[52] Subbarao Tatikonda and Pramod Agarwal. “Field programmable gate array (FPGA) based neural network implementation of motion control and fault diagnosis of induction motor drive”. In: *2008 IEEE International Conference on Industrial Technology*. IEEE. 2008, pp. 1–6.

[53] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *IEEE Journal of Solid-State Circuits* 52.1 (2016), pp. 127–138.