DESIGN EXPLORATION AND VLSI IMPLEMENTATION OF DEEP NEURAL NETWORK ACCELERATOR WITH CONFIGURABLE ARCHITECTURE

Ph.D. Thesis

By

GOPAL R. RAUT

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

MAY 2022

DESIGN EXPLORATION AND VLSI IMPLEMENTATION OF DEEP NEURAL NETWORK ACCELERATOR WITH CONFIGURABLE ARCHITECTURE

A THESIS

Submitted in partial fulfillment of the
requirements for the award of the degree
of
DOCTOR OF PHILOSOPHY

by

GOPAL R. RAUT

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
MAY 2022

INDIAN INSTITUTE OF TECHNOLOGY INDORE

I hereby certify that the work which is being presented in the thesis entitled **DESIGN EXPLORATION AND VLSI IMPLEMENTATION OF DEEP NEURAL NETWORK ACCELERATOR WITH CONFIGURABLE ARCHITECTURE** in the partial fulfillment of the requirements for the award of the degree of **DOCTOR OF PHILOSOPHY** and submitted in the **DEPARTMENT/SCHOOL 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 May 2022 under the supervision of Prof. Santosh Kumar Vishvakarma, Indian Institute of Technology Indore, India.

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.

20/09/22

signature of the student with date

(GOPAL R. RAUT)

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

21/09/2022

Signature of Thesis Supervisor with date

(PROF. SANTOSH KUMAR VISHVAKARMA)

GOPAL R. RAUT has successfully given his/her Ph.D. Oral Examination held on **21 Sept. 2022.**

21/09/2022

Signature of Thesis Supervisor with date

(PROF. SANTOSH KUMAR VISHVAKARMA)

ACKNOWLEDGMENTS

This dissertation is based on the work done at the NSDCS lab of IIT Indore and Cfaed, TU Dresden, Germany. This work is accomplished in the project funded by the University Grant Commission, Government of India. I sincerely appreciate the scholarship's financial support from the HiPEAC grant-Europian Union and ASA ENGAGEMENT program-Germany during my research. First and foremost, I thank my Ph.D. supervisor Prof. Santosh Kumar Vishvakarma, for his invaluable guidance, continuous support, and patience during the entire Ph.D. I feel very fortunate to have an opportunity to work under their patience and supervision. He constantly encouraged me to focus on the most critical and challenging problems during the entire research process and stimulated me to find innovative and creative solutions that build a strong publication record for myself.

I extend my sincere gratitude to Prof. Akash Kumar from Technical University Dresden, Germany, Dr. Anton Biasizzo, and Prof Gregor Papa from Jožef Stefan Institute, Slovenia, who thoroughly reviewed the research work/publications and provided insightful comments that helped improve the quality of this thesis. I am also sincerely thankful to Prof. Koen De Bosschere, The HiPEAC Coordinator from Ghent University, Belgium, for providing fellowship to work at TU Dresden, Germany. I am grateful to my PSPC evaluation committee members, Prof. Vivek Kanhangad and Prof. Aruna Tiwari, for their valuable comments and suggestions during my research. Special thanks to the Director of IIT Indore, faculty members, and the Indian Institute of Technology Indore staff for their cooperation throughout my thesis work.

Many thanks to my colleagues at the NSDCS lab, IIT Indore, and Center for Advancing Electronics Dresden, TU Dresden, Germany, with whom I have worked on projects. I appreciate the technical support and patience during my weird technical discussion and for making the lab a great place to work for me. Especially, I would like to thank Mr. Shubham Rai from Cfaed Dresden, Dr. Vishal Sharma, Dr. Ambika Prasad Shah, Mr. Saurabh Karkun, Mr. Deepak Sunny and Mr. Ribhu Das. I am also thankful to Dr. Ankur Beohar, Dr. Sajid Khan, Dr. Gunjan Rajput, Dr. Neha Gupta, Mr. Narendra Dhakad, Ms. Sumiran Mehra, Mr. Varun Bhatnagar, Ms. Vasundhara Trivedi, Ms. Neha Ashar, Mr. Sonu Jaiswal, Mr. Jogesh Mukala, Mr. Avikshit Komane, and Ms. Khushbu Lalvani for many interesting discussions on technical and non-technical issues during tea time. Further, I thank all my seniors and colleagues who helped me during my thesis work in several ways.

I could not be the person that I am without endless love and the invaluable support of my family and friends throughout life. With their confidence in me and their great expectations, I am climbing up the stairs of a successful life. A special acknowledgment to my elder brother Vilas Raut 'painkiller' for offering invaluable support and humor over the years. I thank my father, Mr. R. S. Raut, and mother, Mrs. S. R. Raut, for always encouraging me to explore new horizons. I also thanks to my uncle Mr. Dilip Raut who was always by my side and was my biggest supporter.

MAY, 2022 GOPAL R. RAUT

Dedicated

to

God Almighty

Shri Chakradhar Swami

& My Family

ABSTRACT

Deep Learning is a subset of Artificial Intelligence involves deep neural networks. Despite many decades of research on high-performance deep neural network accelerators, their massive computational demand still requires resources, efficient architecture, parallel processing, and high memory bandwidth for computational acceleration. The implementation of DNNs faces the burden of excess area requirements due to resource-intensive elements such as multiply-and-accumulate (MAC) units and activation function (AF). Moreover, Edge-AI applications demand a high throughput DNN accelerator with more area utilization and high power consumption. In order to design an area-power efficient DNN accelerator at the cost of insignificant loss in throughput requires optimization in MAC design, AF, and network complexity for efficient data flow. Further, the ASIC-based hardware design of DNN faces the challenge of offering functional configurability and limited chip area.

Addressing the hardware implementation of DNN and targeting the low-power and resource-constrained applications, this dissertation investigates the low-power and efficient VLSI architecture of DNN accelerators. We explore and optimize the *Co-ordinate Rotation Digital Computer* (CORDIC) architecture for the evaluation of MAC and non-linear AF operations. Despite being area and power-efficient, one of the significant drawbacks of CORDIC-based designs is their low throughput. Therefore, we propose a performance-centric pipelined architecture for CORDIC-based MAC and AF. Since pipeline stages come with more hardware resource utilization, we explore the mutual exclusivity between CORDIC stages and conduct a detailed study of accuracy variation concerning the number of stages required to achieve high throughput. We have given different topologies for the CORDIC-based MAC and AF with iterative and pipeline architecture. The proposed designs can be configured to compute both MAC and AF using the same hardware that allows for saving enormous hardware resources.

In order to design system-level architecture, we explored two different designs. First, we propose layer reused architecture. An empirical approach is presented for pipeline MAC to enhance the performance of the DNN engine, which is modular and easily adaptable for any network configuration. Secondly, we proposed DNN with reused hardware-costly AF by multiplexing data using shift-register. The on-chip quantized log_2 based memory addressing with an optimized technique is used to access input features, weights, and biases. The external memory bandwidth requirement is reduced and dynamically adjusted for DNNs.

The proposed design's system parameters and hardware architecture can be configured to application-specific targets. Iterative architecture is useful for AI-enabled IoT applications with insignificant throughput loss. Furthermore, pipeline architecture is useful for Edge-AI with high throughput at the cost of area and power overhead. Further, we worked on a digital solution to design and implement a deep neural network for ASIC and FPGA platforms; however, DNN has succeeded in numerous applications, including digital and analog input information. An ADC is required as an intermediate between the analog input and digital hardware accelerator that enable the process of the analog information since the design network processes the data in digital format.

Therefore Therefore, the dissertation also addressed the efficient design of a low-power, high-speed 4-Bit Flash ADC is designed to process both analog and digital input. The ADC design effectively works at a sampling rate of $2.4GS/s$ and is used in analog-digital interface accelerator. Overall, proposed designs with lower hardware resources and power consumption are especially important for increasingly important edge computing solutions.

List of publications

(A) Publications from PhD thesis work

A1. In refereed Journals:

1. **Raut, G.**, Biasizzo, A., Dhakad, N., Gupta, N., Papa, G. and Vishvakarma, S.K., "Data multiplexed and hardware reused architecture for deep neural network accelerator," *Neurocomputing*, Elsevier, 486, pp.147-159, 2022.

   DOI: 10.1016/j.neucom.2021.11.018 **(SCI, IF 6.523)**

2. **Raut, G.**, Shah, A.P., Sharma, V., Rajput, G. and Vishvakarma, S.K., "A 2.4-GS/s Power-Efficient, High-Resolution Reconfigurable Dynamic Comparator for ADC Architecture," *SCI, Circuits, Systems, and Signal Processing*, Springer 39(9), pp.4681-4694, 2020.

   DOI: 10.1007/s00034-020-01371-4. **(SCI, IF 2.28)**

3. **Raut, G.**, Rai, S., Vishvakarma, S.K. and Kumar, A., "RECON: resource-efficient CORDIC-based neuron architecture," *IEEE Open Journal of Circuits and Systems*, 2, pp.170-181, 2021.

   DOI: 10.1109/OJCAS.2020.3042743.

4. **Raut, G.**, Mukal, J., Sharma, V., and Vishvakarma, S.K., "Performance Enhanced Multiply-Accumulate unit for DNN accelator," *Circuits, Systems, and Signal Processing (CSSP), Springer*, (minor revision received, **SCI, IF 2.225)**

5. **Raut, G.**, Rai, S., Das, R., Kumar, A. and Vishvakarma, S.K., "Enhancing Performance for a CORDIC-based Configurable Activation Function Implementation," *IEEE Access*, (under review **SCI, IF 3.367)**

6. **Raut, G.**, Karkun, S., and Vishvakarma, S.K., "An Empirical Approach to Enhance Performance for CORDIC-based Deep Neural Networks.," *ACM Transactions on Reconfigurable Technology and Systems* , (under revision, **SCI, IF 3.33)**

A2. In refereed conferences:

1. **Raut, G.**, Rai, S., Vishvakarma, S.K. and Kumar, A., "A CORDIC based configurable activation function for ANN applications," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Cyprus*, 2020, pp. 78–83. DOI: 10.1109/ISVLSI49217.2020.00024.

2. **Raut, G.**, Bhartiy, V., Rajput, G., Khan, S., Beohar, A. and Vishvakarma, S.K., "Efficient low-precision cordic algorithm for hardware implementation of artificial neural network." in 23^{rd} *International Symposium on VLSI Design and Test*, Springer, 2019, pp. 321–333. DOI:10.1007/978-981-32-9767-8_28

(B) Other publications during PhD

B1. In refereed Journals:

1. Mehra S., **Raut, G.**, Das R., Biasizzo, A., and Vishvakarma, S.K., "An Empirical Evaluation of Enhanced Performance Softmax Function in Deep Learning" IEEE Access (minor revision) **(SCI, IF 3.367)**

2. Rajput, G., **Raut, G.**, Chandra, M. and Vishvakarma, S.K., "VLSI implementation of transcendental function hyperbolic tangent for deep neural network accelerators," *Microprocessors and Microsystems,*, 84, p.104270, 2021. **(SCI, IF 2.091)**

3. Gupta, N., Shah, A.P., Kumar, R.S., **Raut, G.**, Dhakad, N.S. and Vishvakarma, S.K., "Soft error hardened voltage bootstrapped Schmitt trigger design for reliable circuits. Microelectronics Reliability," *Microelectronics Reliability*, 117, p.114013, 2021. **(SCI, IF 1.947)**

4. Chhajed, H., **Raut, G.**, Dhakad, N., Vishwakarma, S. and Vishvakarma, S.K., "BitMAC: Bit-Serial Computation-Based Efficient Multiply-Accumulate Unit for DNN Accelerator," *Circuits, Systems, & Signal Processing*, pp.1-16, 2022. **(SCI, IF 2.28)**

5. Rajput, G., Agrawal, S., **Raut, G.** and Vishvakarma, S.K., "An accurate and noninvasive skin cancer screening based on imaging technique," *International Journal of Imaging Systems & Technology.*, 2022. **(SCI, IF 2.275)**

B2. In refereed conferences:

1. Knechtel, J., **Raut, G.**, Vishvakarma, S.K., et al. "SCRAMBLE: A Secure and Configurable, Memristor-Based Neuromorphic Hardware Leveraging 3D Architecture" in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, (Accepted).

2. Bhatnagar, V., **Raut, G.**, and Vishvakarma, S.K.,"Loading Effect Free MOS-only Voltage Reference Ladder for ADC in RRAM-crossbar Array" In Proceedings of the 2021 on Great Lakes Symposium on VLSI, 2022, (Accepted).

3. Beohar, A., **Raut, G.**, Rajput, G., Vishwakarma, A., Shah, A.P., Renewal, B.S. and Vishvakarma, S.K., "Compact spiking neural network system with SiGe based cylindrical tunneling transistor for low power applications" in 23^{rd} *International Symposium on VLSI Design and Test (VDAT-2019)*, Springer, 2019, pp. 655-663.

4. Khan, S., Gupta, N., **Raut, G.**, Rajput, G., Pandey, J.G. and Vishvakarma, S.K.,"An ultra low power AES architecture for IoT" in 23^{rd} *International Symposium on VLSI Design and Test (VDAT-2019)*, Springer, 2019, pp. 334-344.

5. G. Rajpur, **Raut, G.**, S. Khan, N. Gupta, A. Beohar, and S. K. Vishvakarma,"ASIC Implementation Of Biologically Inspired Spiking Neural Network" in 9^{th} *International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-19)*, IEEE, 2019, pp. 1–5.

Contents

List of Figures

List of Tables

List of Abbreviations

IC	:	Integrated circuit
VLSI	:	Very large scale integration
AI	:	Artificial intelligence
DNN	:	Deep neural network
CNN	:	Convolutional neural network
ANN	:	Artificial neural network
FPGA	:	Field programmable gate array
ASIC	:	Application specific integrated circuit
GPU	:	Graphics processing unit
CPU	:	Central processing unit
DSP	:	Digital signal processing
LUT	:	Lookup table
RRAM	:	Resistive random access memory
IMC	:	In-memory computating
ADC	:	Analog to digital converter
DAC	:	Digital to analog converter
MAC	:	Multiply-and-accumulate
AF	:	Activation function
ReLU	:	Rectified linear unit
ROM	:	Read only memory
RAM	:	Random access memory
BRAM	:	Block random access memory
HDL	:	Hardware Description Language
RTL	:	Register-transfer level
GOPs	:	Giga [billion] operations per second
FIFO	:	First-In, First-Out
DMA	:	Direct memory access
AXI	:	Advanced eXtensible interface
PISO	:	parallel-in serial-out
SRAM	:	Static random-access memory
CORDIC	:	coordinate rotation digital computer

I/O	:	Input/output
MOSFET	:	Metal oxide semiconductor field effect transistor
FinFET	:	Fin field effect transistor
PMOS	:	P-type metal oxide semiconductor
NMOS	:	N-type metal oxide semiconductor
CMOS	:	Complimentry metal oxide semiconductor
PG	:	Power Gating
DRC	:	Design Rule Check
IOT	:	Internet of things
OTP	:	One time programmable
MTP	:	Multiple time programmable
FF	:	Flip-flop
XOR	:	Exclusive OR
IP	:	Intellectual property
SD	:	Standard deviation
VTC	:	Voltage transfer characteristics
FOM	:	Figure of merit
FS	:	Fast NMOS slow PMOS
FF	:	Fast NMOS fast PMOS
SF	:	Slow NMOS fast PMOS
SS	:	Slow NMOS slow PMOS
TT	:	Typical NMOS typical PMOS
PVT	:	Process-voltage-temperature
FSM	:	Finite state machine
ALP	:	Area-latency-power
PWL	:	Piecewise linear
MSB	:	Most significant bit
MC	:	Monte Carlor
MSE	:	Mean squared error
MAE	:	Mean absolute error
EDP	:	energy–delay product
MNIST	:	Modified national institute of standards and technology
CIFAR	:	Canadian institute for advanced research
GDSII	:	Graphic Data System II
PDK	:	Process Design Kit

List of Symbols

V_{th}	:	Threshold voltage
V_{thn}	:	NMOS Threshold voltage
V_{thp}	:	PMOS Threshold voltage
t_{ox}	:	Oxide thickness
g_m	:	Transconductance
V_{dd} or V_{DD}	:	Supply voltage
T	:	Temperature
W/L	:	Transistor width to length ratio

Chapter 1

Introduction

1.1 Overview

Artificial Intelligence (AI) has been gaining momentum in recent years. Artificial intelligence (AI) and machine learning (ML) have transformed the modern world. There has been significant improvement in this field during the last decade. In recent years, self-driving vehicles, digital assistants, robotic factory staff, and intelligent cities have proven that intelligent machines are possible. AI has transformed most industry sectors like retail, manufacturing, finance, healthcare, and media and continues to invade new territories. Machine learning is a branch of artificial intelligence that allows systems to learn and develop without explicit programming. Machine learning is essential in developing computer programs that can access data and learn for themselves. Like the human brain, machine learning relies on input, such as training data or knowledge graphs, to understand entities, domains, and their connections. Deep learning can begin once entities have been defined.

1.1.1 Accelerators for deep learning

Deep learning is a subset of Machine Learning inspired by the human brain and learns from large amounts of data. Likewise, on how we learn from experience, the Deep Learning algorithm would complete a task repeatedly, each time squeezing it a little to enhance the result. We refer to 'deep learning' because deep neural networks have numerous layers that promote learning. Many recent AI applications are built on deep neural networks (DNNs). DNN (known as deep learning) is a wide branch of AI, science, and technology that aims to create intelligent machines that can attain human-made goals, according to computer scientist John McCarthy. Figure 1.1 shows how deep learning links with all artificial intelligence. DNNs can now outperform humans in many of these domains. DNN's excellent performance stems from its capacity to extract high-level features from raw sensory data after statistical training on vast volumes of data in order to efficiently represent the input space. Further, DNN is a popular choice due to its diverse application and is applied to various non-linear detection problems, some of which are lane detection, pattern recognition, fault

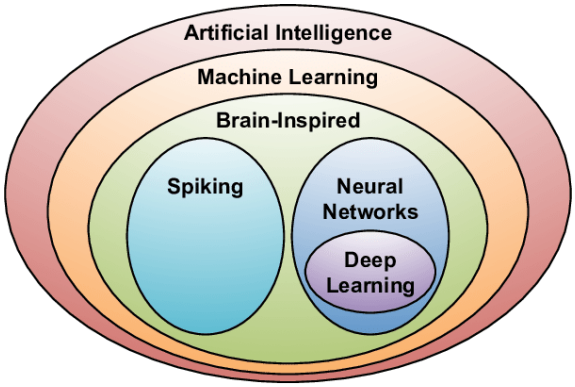


Figure 1.1: Deep learning in the context of artificial intelligence [1]

detection, and monitoring in the industry [8, 9]. However, these applications often require a DNN and real-time processing, requiring high computational power.

Deep learning opens up a need for different platforms like GPU, CPU, ASIC, or FPGA to accelerate the computation of the DNN algorithm. The CPU and GPU-based DNN implementations are general-purpose platforms with specialized hardware supporting various operations, including Multiply-Accumulate (MAC) operation. However, the drawback of CPU and GPU is the low utilization of their resources that, reflects in high power consumption [10]. In comparison, ASIC and FPGA devices provide fast multiplication operations with minimal resource utilization and lower power consumption than streaming pixels and learning features [11]. Further, FPGA also has a specialized hardware structure for MAC operation, but their design is customized to DNN implementation; thus, they achieve high resource utilization and lower power consumption. At the same time, ASIC-based hardware accelerators are the fastest and most energy-efficient. However, they are constrained by their inability to reconfigure neural networks with different features and the inferred network's limited size [12].

The convolutional neural networks (CNNs) such as `VGG-16` require 138 million weights and 15.5G MACs to process one 224× 224 input image. Whereas over 90% is the MAC operations in the total computation model. The overall performance for `VGG-16` on different hardware accelerator have shown in Table 1.1 [13]. Table 1.1 shows the throughput, power budget, and energy performance for VGG-16 by different hardware accelerator platforms. Here we have shown the comparison for implementation with GPU, CPU, FPGA, and ASIC. It can be observed that the throughput performance of the NeuroFlow chip is much higher than the GPU and CPU but takes more development time and cost. Further, ASIC/FPGA has less power consumption and better operation performance per watt. Besides, FPGAs have considerable performance and easily prototype the DNN in minimum time and cost. The CPU and GPU require significantly less development time, and very easy to deploy the DNN accelerator at the cost of performance loss. In order to address the edge-AI devices, the hardware should have high throughput and minimum energy cost. Therefore such devices require FPGA or ASIC-based solutions for better performance.

The features of hardware for AI accelerators is depending on required performance such as speed of computation, parallel processing data bandwidth, and reconfigurability. The comparative

Table 1.1: Hardware Accelerator (CPU, GPU, FPGA, ASIC) performance for Deep neural network.

Performance	CPU (Intel DuoCore 2.7GHz)	GPU (GTX480)	mGPU (GT335m)	NeuroFlow on Xlinx Virtex 7	NeuroFlow on IBM 45nm process
Real GOPs	1.1	294	54	147	1164
Power (W)	30	220	30	10	5
GOPs/W	0.04	1.34	1.8	14.7	230

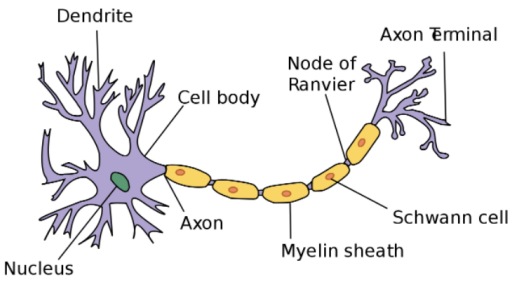
Hardware	Adaptability	Availability	Price	User friendly	Power efficiency
ASIC		✓			High
GPU/CPU	✓	✓	✓	✓	Low
FPGA	✓	✓		✓	Medium

Figure 1.2: Design requirements and performance analysis of AI hardware accelerators [1]

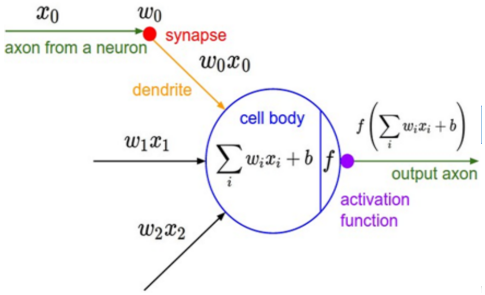
analysis of the hardware types have been reported in Figure 1.2. It can be observed that the performance parameter for FPGA and ASIC shows better results than CPU and GPU. In a fully parallel implementation of the Deep Neural Network, each layer needs a distinct memory space for its weight and bias in order to maximize performance efficiency. The anatomy of a brain neuron is depicted in Figure 1.3 (a). Neurons are connected to several elements called dendrites and contain output elements called axons. Neurons receive signals from the dendrites and process them in order to generate signals in the axons. Activation is the term for these inputs and outputs. Neuron axons spread out and connect to the dendrites of numerous other neurons. A synapse is a connection between the axon's branches and the dendrites. The typical number of synapses in the human brain is 10^{14} to 10^{15}. Whereas artificial neurons mimic the behavior of biological neurons as shown in Figure 1.3 (b).

1.1.2 Neuron Architecutre

The neuron computation entails multiply-and accumulate units (MAC) followed by a nonlinear mathematical transformation called AF. The single neuron with multiple inputs and weights for MAC operation followed by a nonlinear transformation is shown in Fig. 1.4. The various forms



(a) Anatomy of a brain neuron [1]



(b) Artificial neuron having multi inputs at multiply-and-accumulate unit followed by activation function.

Figure 1.3: A biological and an artificial neuron

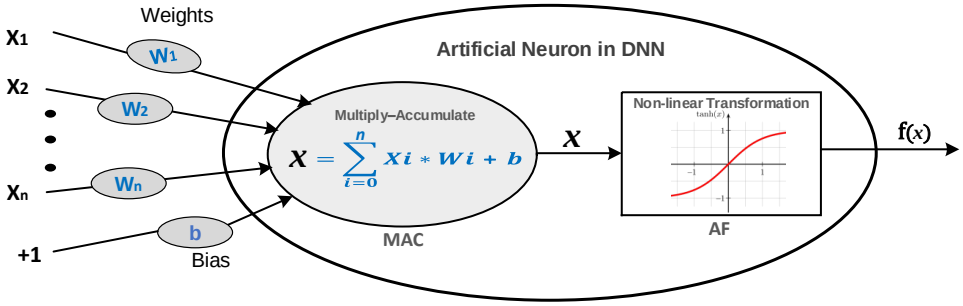


Figure 1.4: Single neuron having multi inputs with MAC followed by activation function.

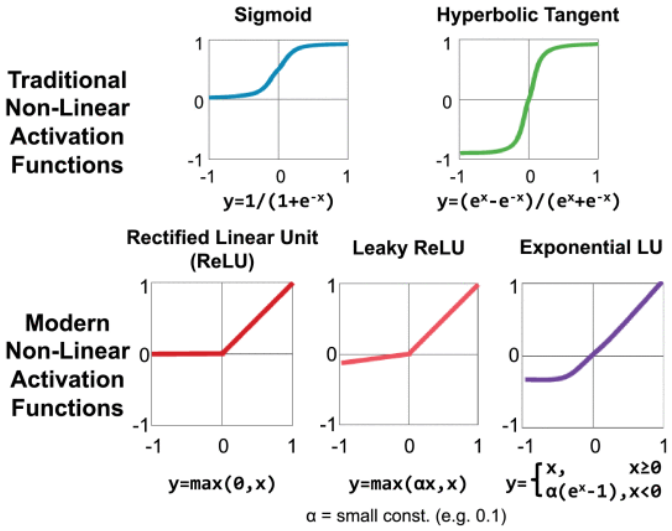


Figure 1.5: Various forms of nonlinear activation function [1]

of nonlinear activation functions have shown in Figure 1.5. Moreover, implementing the AF is challenging due to its nonlinear nature. Therefore, its accurate design and implementation require a large number of hardware resources.

In a deep neural network (DNN), each neuron performs two basic functions: sums weighted input features using MAC and evaluates Activation Function (AF) from calculated sum as shown in Figure 1.3 (b). In this dissertation, the authors refer to a fully connected feed-forward Artificial Neural Network (ANN) with multiple layers between the input and output layers as a DNN. The deep Neural Networks have more (several) layers that contain one hidden layer, multiple hidden layers, and one output layer followed by the softmax function. As an example, DNN has demonstrated having a five-layer fully connected neural network (784:256:128:128:10) for MNIST data classification as depicted in Figure 1.6, and more MACs (within a neuron) are a consequence that demands more hardware resources for implementation. This problem will be more dominant when the network processes the high-resolution images that need higher precision computational elements. Therefore, DNN implementation for mobile or edge devices on resource-constrained hardware attracts much attention. In previous works, multi-bit precision (8, 16, 32, and 64-bit) data representation is used for MAC unit arithmetic computation, considering the trade-off between accuracy and physical performance parameters. Furthermore, the fixed point is preferred over the

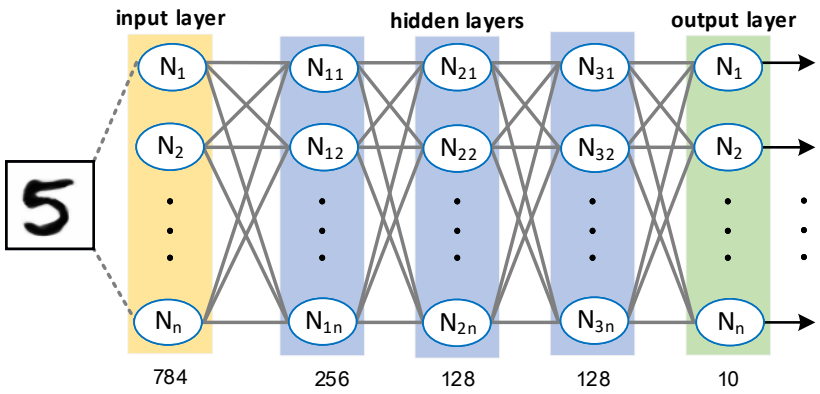


Figure 1.6: Fully connected multi-layer (1-input, 3-hidden, and 1-output) neural network. Each layer has n-number of neurons that connected to each neuron of next layer

floating-point, maximizing utilization density and throughput. However, it is also preferred when the resources are limited, and some degree of error is tolerated [14]. Therefore, a neural network design with 8-bit precision operands is preferable [15] which has also varified in this dissertation and used for hardware implementations. In this proposed efficient architecture of the DNN accelerator, we have used signed 8-bit fixed-point arithmetic.

The neuron computation entails multiply-and accumulate units (MAC) followed by a nonlinear mathematical transformation called AF. The area-efficient architecture is made using different design techniques for MAC unit, AF, and layer architecture [12, 16, 17, 18, 19]. However, The main bottlenecks of the FPGA implementations are limited hardware resources and limited Programmable System (PS)-Programmable Logic (PL) data transmission bandwidth. In order to reduce the memory bandwidth of the DNN implementations, the weights & bias constants should be stored as close to the processing element as possible. Further, lesser hardware resources and lower power consumption are significant for increasingly important edge computing solutions. In this respect, DNN design should have an efficient architecture with better resource utilization and other performance parameters. Some of the previous works have investigated the efficient architecture for a MAC unit that can use minimum resources without performance loss [12, 17]. However, network implementation demands a more efficient design for hardware utilization, power consumption, and throughput performance.

1.1.3 Deep Neural Network

The Deep neural networks (DNNs) have a hierarchical organization of neurons similar to the human brain. Deep neural networks (DNN) are computationally expensive for data-intensive applications that may contain many neurons and several parameters. Typically DNN contains both convolution layer and fully connected layers. In such a network, each layer successively produces a higher abstraction of input data called a function map (fmap). It keeps important but unique information. Modern CNNs can achieve good performance using very deep hierarchies. In CNN, each convolution layer is composed of multidimensional beliefs. In this calculation, the

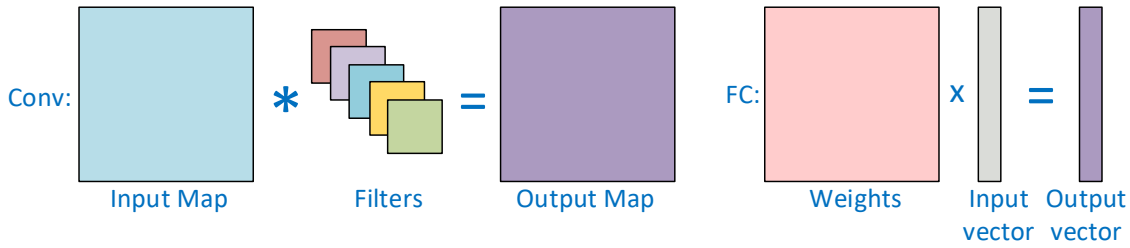


Figure 1.7: convolution and fully connected layers feature

layer input operation consists of two-dimensional input function maps (input map), filters called a channel as shown in Figure 1.7. Each channel is mixed into a filter bank and another 2D filter, one for each channel. This set of 2D filters is often referred to as a single 3D filter. At each point, conviction findings are summarized in each channel. We can also add a one-dimensional offset to the filter results. The convolution layers work for image feature extraction, whereas for image classification and probabilistic distribution, a fully connected layer is used, which has neuron to neuron connection between all neurons, a large number of weights, and data processing involved. The dimension and FC calculation is shown in Figure 1.7. The overall computation involves mainly MAC and AF

In recent years, researchers have focused on the efficient hardware implementation of DNNs. The implementation of DNN is done on GPUs, ASICs, and FPGAs. An area and power-efficient designs are desirable for tiny FPGAs/hardware implementation [20, 21]. The GPU gives flexibility in application development and requires much lower design efforts, but the design is area and power-inefficient. The GPU and CPU use has benefits in process development but inefficiently use the resources, whereas ASIC and FPGA has complex development process but have good performance and efficiently use the resources. Each development platform for the DNN accelerator has its benefits and drawbacks. However, for edge AI applications ASICs are more power-efficient than the GPUs at the cost of compromising design flexibility. In contrast, FPGAs have a good trade-off for power efficiency, parallel processing, and reconfigurability. Further, FPGA designs can easily be transformed into ASIC designs. Hence FPGA is the befitting platform for the DNN implementation. However, FPGA has lower on-chip memory bandwidth and limited hardware resources. An efficient method is required to overcome the area-overhead issue. In the last decades, hardware optimization techniques in DNN design and implementation have been presented by researchers [22, 23]. The hardware efficient design architectures for DNN have been inspected in the state-of-the-art [24, 25, 26, 27, 28]. Moreover, hardware efficient design has been made through reusing the single-layer reuse concept to compute the entire network computation at the cost of lower throughput [25]. These state-of-the-art works have focused either on the system architecture or efficient computational design techniques for achieving better physical parameters.

The conventional cloud-based Artificial Intelligence method used today is insufficient to cover bandwidth, protect data privacy, or support low latency applications due to the increasing demand for real-time deep learning workloads. Therefore, edge computing technology is required to bring

AI jobs to the edge (Edge AI). Due to the recent advances in edge AI, customized AI hardware is required for on-device machine learning inference. Thus, the work presented in this thesis provides insights related to an efficient implementation of deep neural networks for edge-AI applications. We worked on configurable architecture for multiple activation functions such as sigmoid, tanh, and ReLU using the same hardware. At the same time, the design is extended for multiply-and-accumulate computations. Since due to the iterative nature of computing, algorithm design comes with low throughput, which is further explored with hardware pipelining in order to enhance the throughput. The related work and motivation of each work have reported in the individual Chapters. The system-level fully connected and convolution neural network has been designed, which used features of enhancing the performance design of multiply-and-accumulate and activation function computing elements. The system design is also explored for the layer reused and data multiplexed architecture to make it more efficient for edge computing applications. The research can be extended to in-memory computing architecture to make more and more performance-efficient designs. We used targeted hardware for logic implementation, and performance evaluation process have elaborated in Appendix A. The proposed work focuses on efficient architecture in terms of area and power overhead without performance loss. The thesis contributes to the DNN accelerator for FPGA and ASICs have summarized below in Section 1.2.

1.2 Thesis Contributions

The thesis has majorly investigated the CORDIC-based DNN computation by implementing efficient MAC and AFs in the accelerator. In the case of a hardware implementation of a DNN, increasing computational complexity has an unfavorable impact on area and performance. Moreover, an ASIC design is not adaptable, and any configurability such as flexibility in bit precision, or configurability in the type of activation function such as *exponential*, *sigmoid*, hyperbolic tangent (*tanh*), comes with additional overhead. Computation with higher bit precision (32-bit or 64-bit) is also expensive in terms of area and power. Thus, a reduction in hardware complexity (i.e., bit precision) and faster response without compromising accuracy is highly desirable.

To overcome the above limitations, we propose CORDIC-based designs of processing element and AFs, which uses the iterative and pipelined CORDIC algorithm. It uses a fixed-point signed arithmetic computation. The proposed logic has two benefits– Firstly, it is scalable in terms of area and power as a single block can compute both MAC and AFs. Secondly, it allows configurable activation function *(sigmoid/tanh)* and hence has reduced critical path delay and lesser power dissipation. The design also maximizes hardware reuse, unlike conventional ASIC-based designs, which use *MUX*, which leaves out hardware blocks unused. Further, thesis talked about non linear AFs such as exponential sigmoid and tanh AFs since they have higher values of gradient during training and higher updates in the weights of the network. Further, Fully connected neural nets and recurrent neural networks have good performance with sigmoid and tanh due to its less vanishing gradient problem.

Thesis major contributions are summarized below-

1. FPGA and ASIC Implementation of Configurable Activation Function using CORDIC and its Throughput Enhancement.

The multiple AF design is implemented using the CORDIC architecture at the 45nm technology node. The power-gating technique is explored to reduce static power dissipation to make ASIC implementation efficient. Further, the design is implemented on FPGA, and Performance parameters have been evaluated. In the second part, to enhance the throughput of the design, we have proposed the pipeline architecture. We address this trade-off and propose an enhanced CORDIC-based hardware design to compute configurable non-linear activation functions at high throughput for neural network applications. A Pareto study between accuracy and mutually exclusive CORDIC pipeline stages is presented. This study gives an insight on how to define the number of pipeline stages to be used in CORDIC-based design for non-linear activation function, as pipeline stages lead to area and power overheads.

Publications

Gopal Raut, Shubham Rai, Santosh Kumar Vishvakarma, and Akash Kumar. "A CORDIC based Configurable Activation Function for ANN Applications." In 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 78-83. IEEE, 2020.

Gopal Raut, Shubham Rai, Ribhu Das, Santosh Kumar Vishvakarma, and Akash Kumar. "Enhancing Performance for a CORDIC-based Configurable Activation Function Implementation" IEEE Access (Under review)

2. CORDIC-based Configurable Architecture for MAC and AF Implementation within the Neuron.

In this work, we have proposed a resource-efficient and configurable CORDIC-based design, RECON, for a neuron. The CORDIC-based design allows configuration, and the same block can compute both MAC and multiple activation functions. Additionally, our proposed design can compute both signed and unsigned computations. The proposed architecture is area and power-efficient compared to the state-of-the-art designs for both MAC and activation function computation. Using extensive evaluation, we show that our proposed design gives better returns at higher bit precision as compared to other designs.

Publications

Gopal Raut, Shubham Rai, Santosh Kumar Vishvakarma, and Akash Kumar. "RECON: resource-efficient CORDIC-based neuron architecture." IEEE Open Journal of Circuits and Systems 2 (2021): 170-181.

3. Empirical Evaluations of CORDIC Pipeline Stages for High Throughput MAC and Implementation of Layer Multiplexed DNN Accelerator.

To enhance the performance of Deep Neural Network (DNN) accelerators, both compute efficiency and operating frequency need to be maximized. Despite being area and power-efficient, one of the significant drawbacks of CORDIC-based design is its low throughput. This work proposes a performance-centric pipelined architecture of MAC that mitigates low throughput. We conduct a detailed Pareto study of accuracy variation at different precision and the required pipeline stages to achieve high performance. Implementation reports are evaluated and extract performance parameters for pipeline architecture. Further, We implemented layer multiplexing in DNN for efficient usage of resources. The proposed architecture is modular and easily adaptable for any network configuration. The ANN engine is prototyped using the Xilinx Virtex-7 VC707 FPGA board and operates at 50MHz.

Publications

Gopal Raut, Jogesh Mukala, Vishal Sharma and Santosh K. Vishvakarma, "Enhancing Performance for a CORDIC-based Configurable Activation Function Implementation," *Circuits, Systems, and Signal Processing (CSSP), Springer*, (Minor Revision recieved)

Gopal Raut, Saurabh Karkun and Santosh Kumar Vishvakarma,"An Empirical Approach to Enhance Performance for CORDIC-based Deep Neural Networks." ACM Transactions on Reconfigurable Technology and Systems (TRETS), (Revision Submitted).

4. Data Multiplexed and Hardware Efficient Design of Fully Connected Neural Network Accelerator for Tiny FPGAs.

We presented a novel DNN architecture that reduces the hardware-resource requirements and on-chip power consumption without losing computational rate. Our architecture has struck a demand of resource overhead in a DNN by exploiting the hardware-reused context. The contribution of the work is twofold. Firstly, the activation function is designed efficiently for higher accuracy, and quantized memory elements are stored in BRAM for better utilization and higher throughput. Secondly, our experimental results demonstrate the reuse of AF by serializing the output of the array of MAC in each layer. The efficient memory addressing scheme is used to overcome the SoC throughput limits.

Publications

Gopal Raut, Anton Biasizzo, Narendra Dhakad, Neha Gupta, Gregor Papa, and Santosh Kumar Vishvakarma. "Data multiplexed and hardware reused architecture for deep neural network accelerator." Neurocomputing 486 (2022): 147-159.

5. Application to Analog In-Memory Computing: Configurable Design of Dynamic Comparator for ADC.

A 4-bit energy-efficient flash ADC is presented at the 180nm CMOS process. A power-efficient, high-resolution reconfigurable comparator is designed using SCL 180 nm CMOS technology at the supply voltage of 1.8V. We present a runtime reconfigurable differential amplifier circuit design for a low-power application that operates at 2.4 GS/s and significantly reduces the active power in the circuit by using the configurable technique. The proposed architecture is divided into two steps of algorithm implementation depending on the analog input to the circuit. The power-hungry preamplifier stage is bypassed in the multistage comparator for the higher input signal ($> V_{th}$), which results in huge power savings and does not require any kind of extra processing blocks. This reconfigurable technique with the proposed dynamic latch-based comparator gives a very high-speed conversion with low power consumption; hence, it is very useful in Flash-type ADC applications.

Publications

Gopal Raut, Ambika Prasad Shah, Vishal Sharma, Gunjan Rajput, and Santosh Kumar Vishvakarma. "A 2.4-GS/s Power-Efficient, High-Resolution Reconfigurable Dynamic Comparator for ADC Architecture." Circuits, Systems, and Signal Processing 39, no. 9 (2020): 4681-4694.

1.3 Thesis Organization

The remaining portion of this thesis is organized in the following way:

- In Chapter 2, presented the configurable architecture for AF based on the shift-and-add algorithm, collectively known as Co-ordinate Rotation Digital Computer (CORDIC) algorithm. The proposed versatile, configurable AF is designed using CORDIC architecture and implements ReLU, tan hyperbolic, and sigmoid AF using the same hardware resources. Further, presented the performance-centric pipelined design for CORDIC-based AF to mitigate the issue of low throughput.

- In Chapter 3, we extend the Architecture of Chapter 2 for the exploration of RECON. Contemporary hardware implementations of artificial neural networks face the burden of excess area requirement due to resource-intensive elements such as multiplier and non-linear activation functions. We address the above challenge with the proposed resource-efficient *Co-ordinate Rotation Digital Computer* (CORDIC)-based neuron architecture (RECON). It is configured to compute both multiply-accumulate (MAC) and non-linear activation function (AF) operations. The CORDIC-based architecture uses linear and trigonometric relationships to realize MAC and AF operations. The proposed design is synthesized and verified at 45nm technology and explores a semi-custom design approach for *cadence-virtuoso* in order to extract physical performance parameters.

- In Chapter 4, we address the enhanced performance technique for MAC throughput optimization. We design a MAC unit using *Co-ordinate Rotation DIgital Computer* (CORDIC)-based architecture in Chapter 3. Despite being area and power-efficient, one of the significant drawbacks of CORDIC-based design is its low throughput. This work proposes a performance-centric pipelined architecture of MAC that mitigates low throughput. We conduct a detailed Pareto study of accuracy variation at different precision and the required pipeline stages to achieve high performance.

- The Chapter 5 explores the DNN for AI-enabled IoT applications with minimum power and area utilization. We propose hardware reused architecture, and an empirical approach is presented to enhance the performance of the DNN engine. Network implement multiplexed DNN architecture which shows efficient reuse single neuron's layer. The proposed architecture is modular and easily adaptable for any network configuration. The proposed architecture is scalable for any bit-precision. We conduct a detailed Pareto study of accuracy variation at CORDIC pipeline stages that explore the impact of design parameters on accuracy and throughput performance. The DNN is prototyped using the Xilinx Virtex-7 VC707 FPGA board and operates at 50MHz.

- In Chapter 6, we propose DNN with reused hardware-costly AF by multiplexing data using shift-register. The on-chip quantized log_2 based memory addressing with an optimized technique is used to access input features, weights, and biases. The external memory bandwidth requirement is reduced and dynamically adjusted for DNNs. Further, high-throughput and resource-efficient memory elements for the sigmoid activation function are extracted using the Taylor series, and its order expansion has been tuned for better test accuracy.

- In Chapter 7, we design 4-bit energy-efficient flash ADC at 180nm CMOS process. In ADC architecture, we have used the proposed reconfigurable multistage comparator with less power consumption at the Nyquist rate. The proposed schema with a modified dynamic latch is efficient at high speed with a higher slew rate. The pre-amplifier stage is optional to use in the proposed circuit, and it can bypass voltages more than the specified voltage range. The transmission gate-based thermometer to gray Encoder is designed for power-efficient bubble-free conversion.

- Finally, the thesis is concluded in Chapter 8. The direction of future research work is also provided in this chapter.

Chapter 2

Configurable AF design and Throughput enhancement via Pareto analysis

The utility of types of activation functions varies with applications. However, conventional architecture does not support the on-chip configurable architecture of the multi-activation function. This chapter investigates design strategies and optimizations of activation functions (AF) for a DNN architecture. An efficient ASIC-based hardware design of activation function (AF) in neural networks faces the challenge of offering functional configurability and limited chip area. Therefore an area-efficient configurable architecture for an AF is imperative to fully harness the parallel processing capacity of an ASIC in contrast to a general-purpose processor. To address this, we propose a configurable AF based on the shift-and-add algorithm, collectively known as the *Coordinate Rotation Digital Computer* (CORDIC) algorithm. The proposed versatile, configurable activation function uses CORDIC architecture and implements both tan hyperbolic and sigmoid functions.

2.1 Introduction to Activation Function

A deep neural network (DNN) is famous for computational models like pattern recognition, prediction, and classification. A DNN implementation has major components: a computational unit, an activation function (AF), arithmetic precision, data types, and a design platform. Furthermore, hardware realization of DNNs can be classified into three groups– Application-Specific Integrated Circuits (ASIC), Digital Signal Processing (DSP), and Field Programmable Gate Arrays (FPGA) implementations. An ASIC implementation has a performance and area advantage over the other choices [29]. However, the ASIC design is fixed and can not be modified to accommodate different applications. Additionally, the ASIC implementation of a large deep neural network is

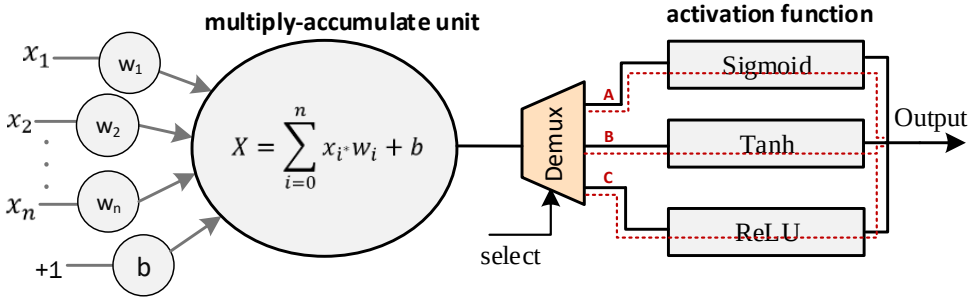


Figure 2.1: Typical design architecture of single neuron with configurable activation function.

resource-hungry. Hence, realizing high precision multiple activation functions for a neural network is generally avoided. Furthermore, an efficient design technique for lowering the supply voltage is essential for power-efficient applications [30, 31].

Hardware implementation of a DNN can exploit the underlying parallelism of the network to further optimize performance. Efficient VLSI architectures have been proposed, targeting diverse applications based on DNN [32], [15]. However, from an application point of view, the performance and accuracy of a neural network primarily depend upon the data precision required [32] [15]. Specifically, in the case of a hardware implementation of neural networks, higher precision generally comes with the high area and power overheads. This trade-off gets even more complicated when configurable architectures are required.

While FPGAs offer configurable hardware designs, they often require more chip area as compared to ASICs [33]. From the design point of view, a neural network demands configurable activation function implementations. It performs various types of non-linear transformation such as *sigmoid*, *hyperbolic tangent* (*tanh*), *exponential* is often desirable [34]. The conventional architecture of a neuron with multiple activation functions, proposed in [35] is shown in Fig 2.1. It has some major drawbacks like area overhead by using separate hardware paths (path-A, path-B, and path-C) for individual activation functions as well as increased data propagation delay (due to MUX) and power dissipation (static power dissipation) due to the unused hardware. The dedicated hardware for the individual activation function is not the desired choice as only one activation function is activated at one time. We propose an optimized CORDIC-based architecture to enable efficient yet configurable computations required in the neural networks to address this trade-off.

2.2 Types of Activation Functions and Design Techniques

In DNN, MAC output is the input for AF's non-linear transformation function. Due to the non-linear nature of the AF, it demands more hardware resources for implementation, and resource utilization increases exponentially for higher precision. In addition, the AF with 16-bit and higher precision requires more memory elements for PWL implementation. Therefore, it is very costly for hardware utilization as the number of required memory elements increases exponentially with the increasing precision. Different types of non-linear transformations are used in the DNN application.

The type of activation functions is Linear, Sigmoid, Tanh, ReLU, Leaky ReLU, Parameterised ReLU, Exponential, Linear Unit, Swish, Softmax [36]. However, AF should be differentiable to compute gradients that can perform back-propagation from output to the input layer weights. An activation function should be differentiable, non-linear, and easy to handle. There are two types of activation functions; those are linear activation functions and nonlinear activation functions. We should select a non-linear activation function. More popularly, nonlinear activation functions are used, such as Exponential, Sigmoid, Tanh, ReLU, etc. However, other nonlinear functions, such as Leaky ReLU, SeLU, Softplus, etc., are also used based on the application. We discussed a few (sigmoid, tanh, ReLU) nonlinear activation functions that have implemented with configurable architecture. In this chapter, we have implemented **Co**-ordinate **R**otation **DI**gital **C**omputer (CORDIC) algorithm based configurable architecture that performs a sigmoid, tanh, and ReLU activation using the same hardware.

Exploration of hardware implementation of activation functions

Some popular AFs are addressed and elaborated on here, such as sigmoid, Tanh, and ReLU. Furthermore, those AF uses different mathematical approaches for hardware implementations in the state-of-the-art as summarized in Table 2.1. One can notice that all these functions are of different forms of the same function but vary in terms of the requirement of arithmetic operations. Further, it can use different on-chip memory resources such as Lookup Tables (LUTs), Block RAM (BRAM), Distributed FPGA memory or external DRAM for local storage of AF parameters. Some of the implementation methods are summarized below as

1. LUTs based implementation by storing function [37]
2. LUTs based implementation by storing parameters [38]
3. Approximation in calculation into *base*-2 exploration
4. Coordinate Rotation Digital Computer (CORDIC) algorithm [39, 40]
5. Combinational logic based implementation [41]

An exact calculation of the AF using hardware implementation is complex due to its continuous and non-linear nature. Therefore, Piece-Wise Linear (PWL) technique is used for these functions' implementation [37, 23]. Moving from lower precision to higher precision, such as 8-bit, 12-bit, and 16-bit, the number of quantization states increases exponentially, leading to an exponential rise in required memory elements. The error contribution due to PWL activation functions at different precision is expressed in [42]. Moreover, the non-linear AF such as sigmoid/tanh cannot be approximated efficiently using only combinational logic [18]. However, using purely combinational logic provides low latency with a small area overhead compared to conventional ROM-based approaches. In [43], an approximation scheme for tanh AF implementation has been proposed using combinational logic design to explore sigmoid function evaluation further. However, circuit design

Table 2.1: Computational equations with different representation of activation function for its design exploration and evaluation

Preliminary Work	Activation Function Implementation	
	Sigmoid	Tanh
[41, 44, 45]	$\frac{1}{(1+e^{-z})}$	$1 - 2\ Sigmoid(-2z)$
[46, 47]	$\frac{1}{(1+e^{-z})}$	$\frac{(e^{2z}-1)}{(e^{2z}+1)}$
[39, 35]	$\frac{1+tanh(z/2)}{2}$	$\frac{(e^{z}-e^{-z})}{(e^{z}+e^{-z})}$
[10, 18]	$\frac{e^{z}}{(1+e^{z})}$	$\frac{sinh(z)}{cosh(z)}$

complexity will increase for higher precision AF implementation. The reuse of hardware resources with an improved architecture for configurable AF implementation is investigated in [35]. This technique achieves high utilization of the FPGA and remarkably improves the physical parameters of FPGA but suffers from low throughput.

In [17], the authors used a single multiplier and adder with iterative accumulation for MAC computation, and AF has been used in every neuron. In DNN, each layer has many neurons, and it requires more hardware resources due to parallel architecture and consumes more on-chip power. Based on the concise review, we address the hardware-efficient and performance-centric solution. We have designed a signed 8-bit dynamic fixed-point hardware-reused DNN accelerator with insignificant loss in throughput. In addition, we have resized the MAC output that allows efficient use of AF with lower precision implementation. Further, area-efficient Taylor series expansion is used for Piece-Wise Linear AF implementation. Finally, we used Block RAM to store pre-calculated values of sigmoid AF, which is beneficial at higher precision.

2.2.1 CORDIC Modes of Operation

Circular rotation mode of operation

We elaborate the working principle of CORDIC architecture for trigonometric calculations (same follows for hyperbolic trigonometric) in Appendix A. Here, CORDIC operates in circular mode. In this case, the scale-factor, K converges. The constants used for circular trajectory in the above matrix calculation are $K_c = 1.6467$ and $m = 1$. The i^{th} iteration with each step to force the angle to converge to the desired final rotation, uses constants $tan^{-1}(2^{-i})$ for the trigonometric calculations as shown in Table 2.3. Moreover, the constant $tan^{-1}(2^{-i})$ in Table 2.3 is first stored in the *ROM* for circular calculations. The calculation is carried out circular mode of operation for *sin(30)* and *cos(30)*. From Table 2.2, inputs X_0[8:0] and Y_0[8:0] are applied as $1/K_c$=0.6072 and 0 respectively. Taking a common scaling factor K and CORDIC equations in pseudo rotation for

circular modes of trajectories are then formulated as:

$$X_{i+1} = X_i - Y_i \cdot d_i \cdot tan\ \alpha_i \tag{2.1a}$$

$$Y_{i+1} = Y_i + X_i \cdot d_i \cdot tan\ \alpha_i \tag{2.1b}$$

$$Z_{i+1} = Z_i - d_i \cdot \alpha_i \tag{2.1c}$$

where, X_i, Y_i *and* Z_i are input variables at i^{th} iterations. Further, α_i is the rotation angle in radians at each iteration i $\in \{1, 2, 3, ...n\}$. Also, α_i is represented as E_i considered as memory element for i^{th} iteration. The linear converge form of Eq. 2.1 is shown in Eq. 2.2. Here, constants K_c correctness have shown for up to 4 decimal points which requires approximately 13-bit precision after decimal point. However, we choose quantized values for the K_c based on the fractional bits in arithmetic precision. Further, for K_c higher bit precision more accurate value considered for the results evaluation.

$$X_{i+1} = X_i + d_i \cdot Y_i \cdot 2^{-i} \tag{2.2a}$$

$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{2.2b}$$

$$Z_{i+1} = Z_i - d_i \cdot \tan^{-1} 2^{-i} \tag{2.2c}$$

Here, $di = -1$ if $z_i < 0$ and $m = 1$

We have given the calculation for trigonometric calculations *sin(30)* and *cos(30)* and is calculated in 5 clock cycles using Eq. 2.2; shown grayed out in Table 2.2.

2.2.2 Hyperbolic rotation mode of operation

The CORDIC algorithm in Hyperbolic Rotation Mode can be used to implement the hyperbolic trigonometric operation. The generalized equations in a hyperbolic mode of operation are shown in Eq 2.3. In this mode, hyperbolic trigonometric functions sinh and cosh can be evaluated using hyperbolic coordinates with scaling factor K_h to 0.8281. The CORDIC algorithm set variable m to -1 and used a division block for the final results. In order to calculate sinh and cosh, the rotational mode of the CORDIC algorithm is used. The final division by scaling factor can be eliminated by dividing initial variables, thus setting X_0 to $1/K_h$, Y_0 to 0, and setting the input to Z_0. The rotations are selected such that rotation scaling factors are negative power of 2, i.e. $\alpha_i = \tanh(2^{-i})$. Therefore, from Table 2.4, it is observed that output X_n, Y_n and Z_n converges as: X_n to $\cosh Z_i$; Y_n to $\sinh Z_i$; and Z_n to zero (Z $\rightarrow$ 0).

Table 2.2: The i^{th} iteration (clock) steps to force the angle to converge to the desired final cos & sin in modified CORDIC architecture in rotation mode.

i	**d_i**	**$X_{i+1} \rightarrow$ cos Z**	**$Y_{i+1} \rightarrow$ sin Z**	**$Z_{i+1} \rightarrow$ 0**
Clock	Direction	$1/K_c$=0.6072	0.00	Input = 30.00
0	1	0.6072	0.6072	-15.00
1	-1	0.9108	0.3036	11.56
2	1	0.8349	0.5313	-2.47
3	-1	0.9013	0.4269	4.65
4	1	0.8747	0.4833	1.07

Table 2.3: The constants used in CORDIC calculation at n^{th} iteration that force the input Z_{in} to converge towards zero

Calculation for n^{th} stage	1	2	3	4
$E_n = \tanh^{-1}(2^{-n})$	0.54931_{10}	0.25542_{10}	0.12566_{10}	0.06258_{10}
Binary Conversion	0.1000110_2	0.0100001_2	0.0010000_2	0.0001000_2

$$X_{i+1} = X_i + d_i \cdot Y_i \cdot 2^{-i} \tag{2.3a}$$

$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{2.3b}$$

$$Z_{i+1} = Z_i - d_i \cdot \tanh^{-1} 2^{-i} \tag{2.3c}$$

After n^{th} iterations, it computes $\sinh(Z)$ and $\cosh(Z)$ that can be used for further exponential function evaluation using Eq. 2.4 [10] have elaborated in Table 2.2.

Here, the results in Table 2.2 are still a bit far from sin(30)=0.5 or Cos(30)=0.8660. The computed values of 0.4833 and 0.8747 are 0.015 +1/64 away from the real value, i.e., The accuracy is only valid upto 6 bits. From the CORDIC computation, we have shown sin/cos values representing 8-bit fixed points having 2 bits as an integer. The values are not precisely the actual values. However, since the neural network is error-resilient, the approximate evaluation also has the desirable accuracy. Furthermore, by increasing the bit precision, the representation of constants values will be more precise. Therefore, with higher bit width representation, accuracy will be increased.

$$f(Z) = exp(Z) = e^Z = \sinh(Z) + \cosh(Z) \tag{2.4}$$

2.3 Configurable architecture for Multiple Activation Function using Iterative CORDIC

This work investigates design strategies and optimizations of activation functions (AF) for a DNN architecture. We employ CORDIC architecture in hyperbolic rotation mode to realize `tanh` and `sigmoid` and `ReLU` functions. The CORDIC-based architecture requires only a shift-and-add op-

Table 2.4: Iteration-level calculation shown for activation function using CORDIC in hyperbolic mode

i	**d_i**	**$X_{i+1} \rightarrow$ cosh Z**	**$Y_{i+1} \rightarrow$ sinh Z**	**$Z_{i+1} \rightarrow$ 0**
clock	for $(i+1)^{th}$	Initial Conditions/Inputs		
iteration	iteration	$1/K_h$	reset	Input AF
initial	–	1.2075	0.0000	0.78125
1	1	1.2075	0.6037	0.2319
2	1	1.3584	0.9056	-0.0208
3	-1	1.2452	0.7358	0.1022
4	1	1.2911	0.8136	0.0397
5	1	1.3172	0.8554	0.0084

eration, leading to area and power savings with better data precision. The proposed architecture uses the same hardware resources to realize the sigmoid and tanh function. The major contribution is summarized in the following points:

- Design of a configurable architecture for multiple activation function using the CORDIC-based algorithm.
- Optimization of exiting CORDIC-based architecture in terms of area, power, and delay.
- Analyze the impact of technology scaling on circuit's physical parameters like area and power, and proposed power-gating approach for additional power savings.

The multiple AF design is realized using the CORDIC architecture at 45nm technology node. The configurable activation function using CORDIC is designed for low-cost hardware requirements. The power gating technique is further explored to reduce static power dissipation.

2.3.1 Iterative CORDIC architecture

The CORDIC architecture for all modes of operation is shown in Fig. 2.2. Conventionally for N-bit precision arithmetic computation required maximum 'N' iteration in CORDIC computation. Here, in Fig. 2.2 we referred n= 0,1,2,.....N. In archtiecture, by setting the CORDIC configuration parameters in the hyperbolic mode, we evaluate sinh(z) and cosh(z) through which we evaluated the sigmoid and tanh AF in neural network. The inputs and constants for hyperbolic calcualtions have discused in Section 8. The propagation delay of a conventional CORDIC is hence the sum of the delay of MUX, adder/subtractor, barrel-shifter, and feedback resistor blocks are involves in one CORDIC unit for each micro-rotation. Here, the output of the current iteration will be the input for the following iteration in the CORDIC architecture, which computes iteratively. As a result, the result must be saved in the intermediate step for iterative processing. The value is kept in the intermediary register. We referred to the storage component as a feedback register. The modified architecture is iterative in nature as it uses only one CORDIC unit for all micro-rotations.

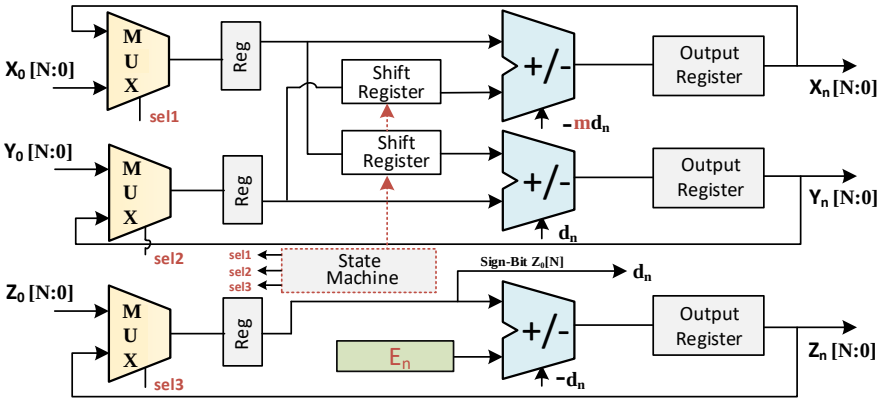


Figure 2.2: Signed N-bit iterative CORDIC architecture

2.3.2 Configurable activation function

The hyperbolic tangent and sigmoid are generally the most used nonlinear activation functions. Activation functions like sigmoid or tanh provide a smooth transition between excitation and inhibition, which improves the neural response [48]. The authors in [49] proposed the direct computation of a single sigmoid activation function with CORDIC architecture. For negative values, they implemented it with $2's$ complement arithmetic computation. Similarly, the authors employed approximation of log-sigmoid transfer function with CORDIC algorithm in [47] for e^{-x} for sigmoid function realization. However, these techniques require multiple stages for AF realization, which lead to high area overheads and longer critical paths. To overcome the double multipliers overheads with state-of-the-art, we have systematically investigated the configurable architecture for both *sigmoid* and *tanh* AF design. The proposed architecture for configurable activation function block design is shown in Figure 2.3 and Figure 2.4. It can be noticed that sigmoid (shown by green color lines in Figure 2.4) uses the maximum components, leading to the largest critical path. Here, n-stage Pipeline CORDIC architecture is used with N-bit precision. The CORDIC module is used in hyperbolic rotation mode for the generation of *sinh* and *cosh* functions. The signed 8-bit precision CORDIC architecture is designed. The most significant bit (MSB) of 1-bit inputs, $Y[8]$ and $Z[8]$, is used for generating the '*dn*' signal. The signal '*dn*' is fed to the adder/subtractor block that decides whether addition or subtraction has to be done, the signals are depicted in Figure 2.4.

The generated trigonometric hyperbolic functions is used for producing exponential function as shown below. The 8-bit CORDIC output is applied to the adder for producing exponential output.

$$sinh(z) + cosh(z) = e^{z} \tag{2.5}$$

The CORDIC-based design outputs trigonometric hyperbolic functions – *sinh* and *cosh* functions to compute the exponential function as shown in Eq. 2.6a. Once, *exp(z)* is calculated, it can be used to compute *sigmoid* and *tanh* function as shown in Eq. 2.6b and Eq. 2.6c respectively. Additionally, using a simple MUX-based extension, the proposed architecture can easily be

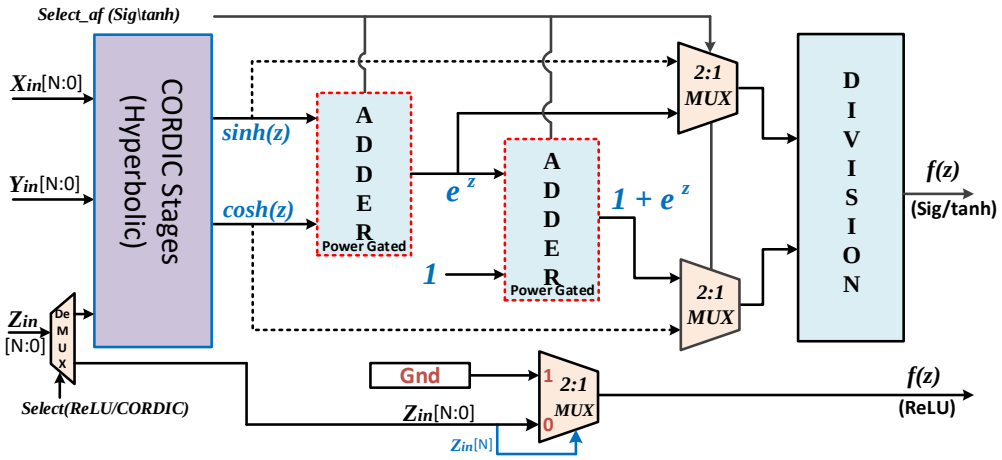


Figure 2.3: N-bit precision, CORDIC-based configurable AF with 'select' and 'select_af' signals.

configured to realize a *rectified linear unit* (ReLU) function using Eq. 2.6d. Figure 2.3 shows the hardware design for computing *exponential*, *sigmoid*, *tanh* and *ReLU* function using CORDIC. Whereas, the control signals for AF configuration are elaborated in Table 2.4.

$$f_1(z) = exp(z) = e^z = sinh(z) + cosh(z) \tag{2.6a}$$

$$f_2(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{sinh(z)}{cosh(z)} \tag{2.6b}$$

$$f_3(z) = sigmoid(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \tag{2.6c}$$

$$f_4(z) = ReLU(z) = \begin{cases} z, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases} \tag{2.6d}$$

Additionally, in the proposed architecture, the execution of the *tanh* function does not require the additional adder block as compared to [50]. However, this unused block can dissipate static power. Addressing this issue and considering the trade-off between leakage current and speed of operation, the *Power Gating* (PG) technique is used to minimize the leakage power and improve the performance. We have implemented adders with the PG technique used in the proposed architecture as shown in Fig. 2.3. The *select* line is used for the power supply connection to the adder logic block. We used a logic gate with wide-gate size transistors so that the performance is not compromised. The coarse-grain technique is used in the design for better efficiency, less circuit complexity, and moderate switching time.

From Eq. 2.6(a), the output of the adder is fed to the division block. The subtraction and the shift operation are the two basic operations that are used within the divider circuit [51]. The proposed work further explores the relationship between the tanh and sigmoid using Eq. 2.6(b) and 2.6 (b), and the design architecture allows to implement of both *tanh* and *sigmoid* using the same RTL design based on the select line. The input from the MAC unit will be directly applied to the proposed design for the realization of the activation function. This is the key difference

Table 2.5: The constants used in CORDIC calculation at n^{th} iteration that force the input Z_{in} to converge towards zero

AF Input			Select signal for AF f(Z)		
X_{in}	Y_{in}	Z_{in}	Configure AF	select_c	select_af
–	–	Z	ReLU	0	X
K	0	Z	Sigmoid	1	0
K	0	Z	Tanh	1	1

Table 2.6: Hardware implementation result for configurable activation function

Architecture Design	**Resources Utilization**		**On-chip Power (mW)**			**Critical Delay(ns)**	**PDP**
	LUT	**FF**	**Logic**	**Signal**	**Clock**		
Available Res.	17600	35200	–	–	–	–	–
CORDIC Arch.	17	22	0.10	0.12	0.09	2.49	0.78
Conf. AF	27	38	0.22	0.28	0.31	3.09	2.23

between the previous approaches and the proposed architecture, leading to significant performance enhancement.

The configurable archtiecture have implemented on the FPGA and evaluated the performance parameters , it is implemented on FPGA *ZyboXC7z010-board*.

2.4 Enhancing Performance of AF using Pipelining

The hardware designs have employed the CORDIC algorithm's hyperbolic rotation mode to realize configurable activation functions as discussed in Section 2.3. These designs have lower area and power overheads than the conventional memory-based designs but suffer from low throughput due to the inherently iterative nature of the CORDIC algorithm. This section addresses this trade-off and proposes an enhanced CORDIC-based hardware design to compute configurable non-linear activation functions at high throughput for neural network applications. The major contributions of the work are summarized in the following points:

- A Pareto study between accuracy and mutually exclusive CORDIC stages is presented. This study gives an insight on how to define the number of pipeline stages to be used in CORDIC-based design for non-linear activation function, as pipeline stages lead to area and power overheads. Our study shows that using four pipeline stages achieves higher throughput without compromising accuracy.

- The error cost functions for the AF model are extracted to analyze the impact of approximation in computation which comes with a reduced number of pipeline stages.

- We analyze and discuss the circuit's physical parameters like area, power, and critical delay for extracted Pareto points, evaluate it with the CMOS 45nm technology node and compare it with the state-of-the-art designs.

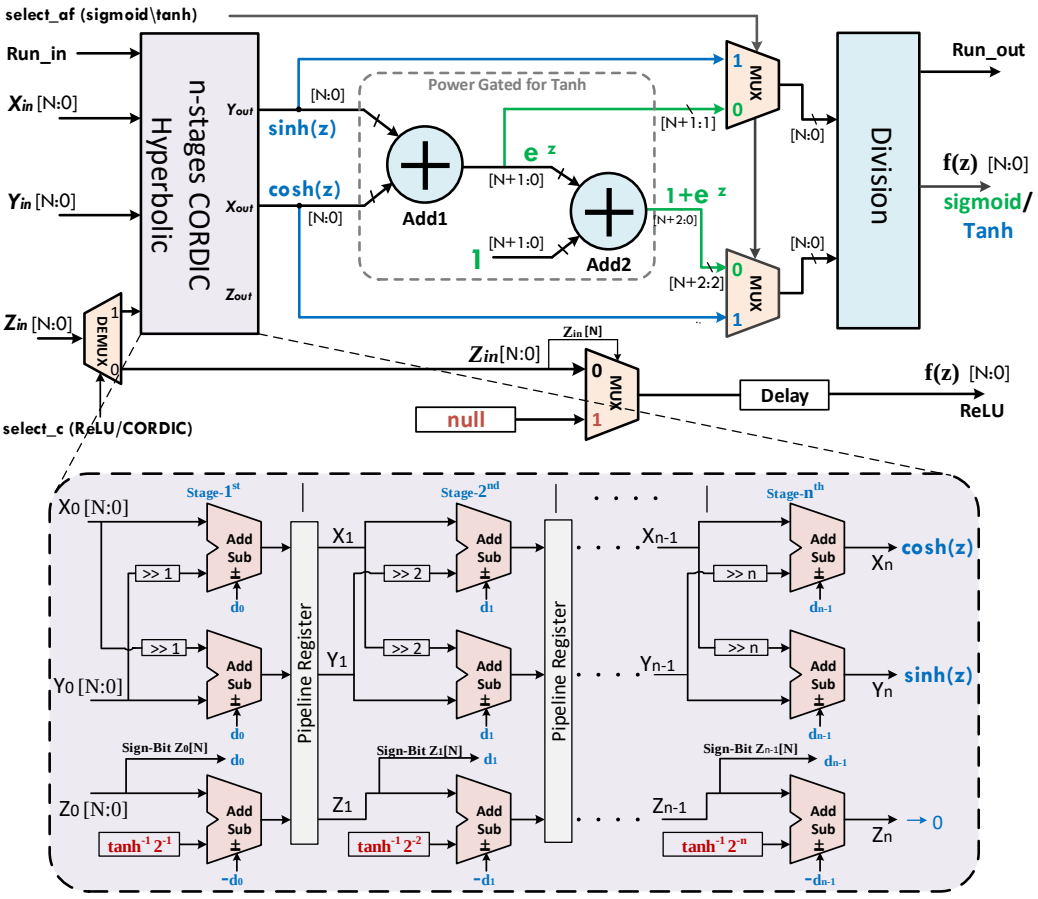


Figure 2.4: N-bit precision, CORDIC-based configurable AF with 'select_c' and 'select_af' signals [18]. It can be noticed that sigmoid (shown by green colour lines) uses the maximum components and hence leads to largest critical path. Here, n-stage Pipeline CORDIC architecture is used with N-bit precision.

2.4.1 Finding Pareto points to enhance hardware performance

If we consider the iterative hardware design (Figure 2.2) for the CORDIC algorithm, we can see that every iteration is mutually exclusive. Hence, these iterative stages can be pipelined for high throughput. However, the number of pipeline stages directly impacts the chip area and static power consumption overhead. Hence, a minimum number of stages must be determined, giving an achievable accuracy comparable to the maximum. Moreover, this work has evaluated the performance parameters at different precision. For our experiments, we use four pipeline stages. In the ReLU mode, our design does not suffer from throughput loss because, unlike CORDIC-based architecture, the activation is computed in a single cycle. Hence, the pipeline stages are not needed for ReLU computation.

The proposed design supports fixed-point representations with different precision, datasets, and neural network sizes. Hence, multiple bit representations such as 8-bit, 12-bit, 16-bit, and 32-bit can be used. We take an extra bit for the sign bit and only 1 bit for the integer part in all these representations. For example, in the case of 8-bit precision, we have used a 9-bit number format representation. In this representation, 1 bit is reserved for the sign, 1 bit for the integer part, and 7 bits for the fractional part. Symbolically, this is written as 'fixed $\langle 8, 7 \rangle$'. The CORDIC

Variable	n=0 (Initial)	n=1 (1st stage)
$X_{n+1} = X_n + d_n * Y_n * 2^{-n}$	X_0 = 1.0011010 (Bin) = 1.20312(Dec)	X_1 = 1.0011010 + dn * 0.0000000 0 = 1.0011010 (Bin) (Y0 right shift by n bits; lost bits)
$Y_{n+1} = Y_n + d_n * X_n * 2^{-n}$	Y_0 = 0.0000000 (Bin) = 0.000 (Dec)	Y_1 = 0.0000000 + dn * 0.1001101 0 = 0.1001101 (Bin) (X0 right shift by n bits)
$Z_{n+1} = Z_n - d_n * \tanh^{-1}(2^{-n})$	Z_0 = 0.1010010 (Bin) = 0.640625 (Dec)	Z_1 = 0.1010010 - dn * 0.1000110 = 0.0001100 (Bin)

Figure 2.5: Dynamic fixed-point data representation and arithmetic calculation for the $n = 1$ stage. Similar subsequent calculation used in enhance CORDIC calculations are shown in Table 2.7.

Table 2.7: The calculation for each n^{th} stage in hyperbolic mode. (The Z_0 i.e input for AF calculation is shown for first neuron in the fatten Layer).

n	$\mathbf{d_n}$	$\mathbf{X_{n+1} \rightarrow \cosh Z}$	$\mathbf{Y_{n+1} \rightarrow \sinh Z}$	$\mathbf{Z_{n+1} \rightarrow 0}$
Pipelined stage	$(n+1)^{th}$ iteration	Initial Conditions/Inputs		
		$X_0 = 1/K$	Y_0 = reset	Z_0 = input: 0.640625_{10}
initial	—	1.0011010 (set)	0.0000000 (set)	0.1010010 (input)
1	+1	1.0011010	0.1001101	0.0001100
2	+1	1.0101101	0.1110011	– 0.0010101
3	–1	1.0011111	0.1011110	– 0.0000101
4	–1	1.0011010_2 1.203125_{10}	0.1010101_2 0.664063_{10}	0.0000011 (≈ 0)

calculation for the single-stage is reported in Figure 2.5. Furthermore, in most cases it is beneficial to have the same input and output format. This implies $i_{b\_in} = i_{b\_out} = i_b$, $f_{b\_in} = f_{b\_out} = f_b$ and $N_{in} = N_{out} = N$, where i_b represents the integer bits excluding the sign bit, f_b represents the fractional bits and N represents the total number of bits. The experimental results evaluation merely shows that a 1-bit integer yields the maximum accuracy. The architecture accuracy has been evaluated for one integer bit in fixed-point representation. However, the integer bits are user-defined and set for specific requirements. The permissible range for the AFs is considered between the -8 to +8 and can be achieved in the proposed model by keeping the three integer bits, i.e., the number of integer and fractional bits is user-dependent. Therefore, in the case of fixed-point format final choice must be based on the target application that gives maximum accuracy with desired precision [14].

The required memory cells for AF implementation increase exponentially with precision. Conventionally, the CORDIC produces 1-bit accuracy at each iteration. However, the overall computational output accuracy depends on the precision of input numbers. The binary representation and CORDIC calculations for the four pipeline stages are shown in Table 2.7 Therefore, simply four-bit precision ROM can't achieve like four pipeline stages CORDIC computation. Similarly, the LUT-based approach is not appropriate for higher precision implementation in FPGA implementation due to the large memory size.

2.4.2 Timing analysis

Considering T_{clock} time required for the memory element (LUT) based activation function. In reference to the T_{clock}, the timing analysis have explored using Eq. 2.7. Here, T_M, T_C, T_i and T_p are the total time required for memory element based AF, CORDIC computation, proposed iterative CORDIC based AF and pipelined CORDIC based implementation respectivelly. In Eq. 2.9(b), n represents the number of CORDIC iteration.

$$T_M = T_{clock} \tag{2.7a}$$

$$T_C = n \times T_{clock} \tag{2.7b}$$

$$T_i = T_C \ + \ T_{Add} \ + \ T_{Div} \tag{2.7c}$$

$$T_p = T_{clock} \ + \ T_{Add} \ + \ T_{Div} \tag{2.7d}$$

2.5 Results and Discussion

2.5.1 Accuracy evaluation for mult-bit precision CORDIC-based AF

To validate the compatibility of our proposed architecture in a NN model, we use a CNN model from *TensorFlow* [5] for the MNIST and CIFAR-10 datasets using CORDIC-based non-linear activation function. Accuracies at different bit-precision and different fixed point representations were evaluated. Inference accuracy comparison for AFs at different bit precision using CORDIC-based model, piece-wise linear (PWL) model [46] and the *TensorFlow* implementation [5] is shown in Table 2.8. The TensorFlow model has three convolutional later followed by two fully connected layers. As input, a CNN takes tensors of shape (*image_height*, *image_width*, *color_channels*), ignoring the batch size. The detailed model description has given below Algorithm. CAF is a CORDIC-based configurable activation function with Sigmoid and Tanh realization.

Algorithm 1: TensorFlow CNN model description

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='CAF', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='CAF'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='CAF'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='CAF'))
model.add(layers.Dense(10))
```

Here, we can observe that there is only $< 1\%$ accuracy loss using the proposed CORDIC-based design as compared to the computation using TensorFlow [5] sigmoid and tanh activation functions.

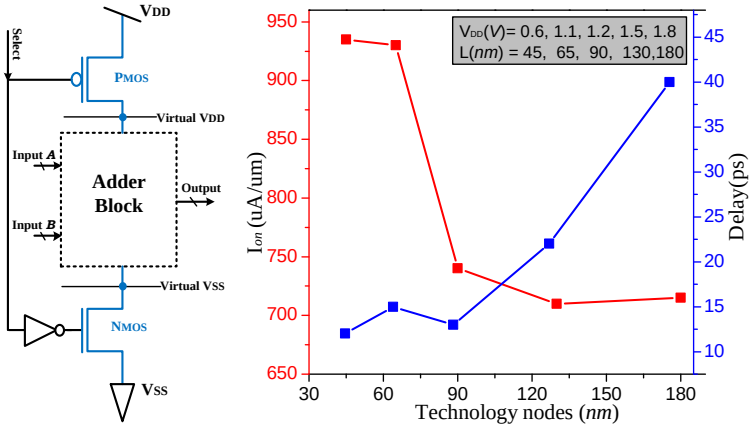
Table 2.8: Network inference accuracy of CNN with CORDIC based, piece-wise linear AF and TenserFlow library based AF [5] at different fixed-point bit-precision

Fx-Point	**Sigmoid**			**tanh**		
Precision	**Tensor[5]**	**PWL[46]**	**Proposed**	**Tensor[5]**	**PWL[46]**	**Proposed**
Inference Accuracy (%) for Configurable Activation Functions for MNIST Dataset						
32-bit	99.14	98.77	98.23	98.95	98.83	98.92
24-bit	99.09	98.73	98.18	98.71	98.68	98.81
16-bit	99.04	98.70	98.16	98.61	98.60	98.79
12-bit	98.96	98.67	97.72	98.46	98.58	98.67
8-bit	98.63	98.52	97.5	97.90	98.44	98.22
Inference Accuracy (%) for Configurable Activation Functions for CIFAR-10 Dataset						
32-bit	64.6	62.6	62.2	66.4	66.5	65.1
24-bit	61.8	59.4	58.1	65.4	66.2	64.8
16-bit	57.8	57.7	56.1	63.9	64.8	63.8
12-bit	55.6	54.2	52.2	63.4	64.2	62.8
8-bit	52.6	51.3	49.0	60.9	61.4	60.5

2.5.2 Impact of technology scaling and power-gating

At lower technology node, area and dynamic power savings are observed but there is an increases in static power dispassion. These savings are due to scaling in technology model which has an impact on physical parameters such as mobility (μ_0) and saturation velocity (V_{sat}). At lower technology nodes, static power dissipation is the biggest concern. For the inverter with coarse-grain power gating circuit, *ON* current and delay variations with respect to technology scaling is shown in Fig.2.6 (b). Based on simulation results and merit, the power gate size should be large as compared to its normal size for handling the amount of switching time. We determine the power gate size for a larger slew rate with a lower response time and the same is used for circuit simulations. Based on simulation results and merit, the power gate size chosen is $3\times$ larger compared to its standard length in 45nm technology to maintain good output performance. At 45nm, we used standard sizes for PMOS and NMOS as $3.3um$ and $1.5um$, respectively. The proposed architecture using the coarse-grain technique is shown in Fig. 2.6 (a). In addition, the circuit design with lower power supply is beneficial as it minimizes power dissipation. However, it comes with an increased propagation delay. The increasing power supply dominates the large leakage current flows which means there is a trade-off between leakage current and speed of operation. The leakage power increases with the supply voltage and it exceeds the dynamic power for supply greater than $0.7V$ at 45nm technology. Moreover, the relation between the power supply and propagation delay in the CMOS circuit is given in Eq. 2.8.

$$T_d = \frac{C_L \cdot V_{dd}}{(V_{dd} - V_t)^{\alpha}} \tag{2.8}$$



(a) (b)

Figure 2.6: (a) Basic cells circuit design using power gating technique and *select* pin is use to isolate the circuits from power supply when not in used. (b) Inverter Circuit ON current and Delay variations with respect to technology scaling.

2.5.3 Performance parameters of iterative CORDIC-based configurable AF

To evaluate the proposed design architecture, the configurable activation function design is represented in HDL using Verilog hardware description language. The synthesis results are produced by *Design Compiler*-Synopsys. The netlist file extracted from *cadence-encounter* and generated .cdl used for RTL digital design extraction into CMOS design using the *v2lvs*-Mentor Graphics. The extracted design is simulated in *cadence-virtuoso* for performance parameter validation at different process corner, temperature and mismatch. In order to evaluate the impact of process variation and mismatch (which increases significantly in $65nm$ and lower CMOS technology), the circuit is also simulated at all PVT variations. Further, we carry out Monte-Carlo simulation to model the probability of different outcomes of dynamic power. It helps to calculate random variations in dynamic power dissipation due to device-mismatch in the characteristics of identically designed devices, occurring during the manufacturing of ICs.

The CMOS logic based circuit design is extracted in *virtuoso-cadence* and post-layout circuit simulation of proposed configurable AF design is carried out for current and voltage variation at different process corners and supply variation. The parameters calculated at three different FF, TT and SS corners are shown in Fig. 2.7 (a). It is observed that the static power is more than a dynamic power in the fast-fast (FF) process corner with temperature 85^{o}C. The circuit simulated at different supply voltages with typical-typical (TT) process corner and observed variations in simulation results are shown in Fig. 2.7 (b). We design and use the adder with the power gating technique in adder block as shown in Fig. 5. We have designed and simulated the circuit designs from [50, 47] at 45nm process technology. Results and comparison of proposed architecture with and without power gating and state-of-the-art is shown in Table 2.9.

The proposed architecture is synthesized for different precision to have a fair comparison between CORDIC based proposed architecture and combinational logic [46] design for *sigmoid*

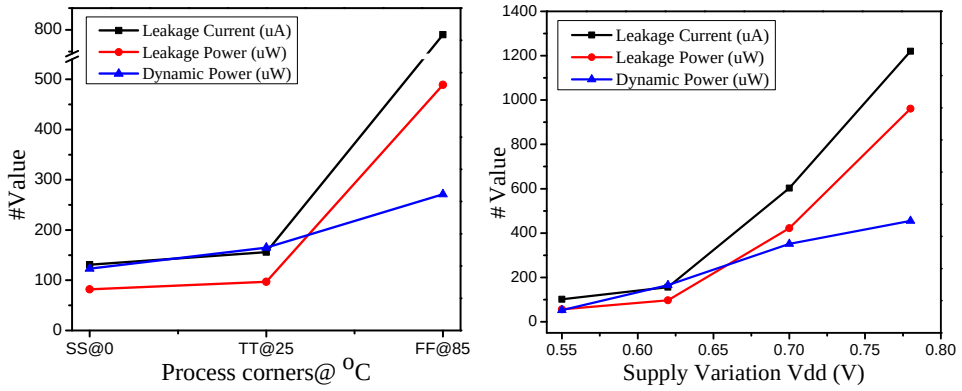


Figure 2.7: Performance parameter variation of proposed architecture for 45nm technology node (a) the different process corner and temperature at power supply = 0.62V (b) for TT corner with power supply variation.

Table 2.9: Performance parameter metrics of activation function design using CORDIC architecture for proposed and state-of-the-art at 45nm TT Process Corner

Sigmoid AF Design Arch. Parameters	**Rotation Scaling [50]**	**Inverted Input [47]**	**Proposed without PG**	**Proposed with PG**
Area (μm^2)	2983	2735	2145	2280
St. Power (μW)	176	121.7	96.7	72.94
Dy. Power (μW)	299.4	209.5	176	176
Delay (ns)	-	5.93	4.30	4.71
Precision ($bits$)	8	8	8	8

function. The synthesis results at *180nm* and *45nm* technology node are shown in Table 2.9 and Table 2.10 respectively, for 8-bit, 12-bit and 16-bit precision digital design. The results comparison shown in tables concludes that CORDIC based architecture is having linear increments in physical parameters such as area and power with respect to increment in bit precision. However, in combinational logic design, this increase is exponential. The proposed technique with CORDIC architecture offers a substantial saving of area over the combinational logic-based design proposed in [6]. It concludes that CORDIC based architecture is an admirable choice for higher precision AF implementation. However, the number of *clk* edges required is more as compared to combinational design. Hence, CORDIC based design is favorable for applications where the energy and area requirements are less. Hence, they offer an excellent choice for IoT devices where there is a tight area and power budget.

At lower technology, process and mismatch is also an important issue in terms of power dissipation, stability, and reliability. We have simulated the circuit at *45nm* technology node. We observed area saving as compared to the previous work as shown in Table 2.9. The Monte-Carlo simulation for dynamic power variation due to process and mismatch is carried out for 1000 samples for the proposed architecture and the state-of-the-art and shown in Fig. 2.8. The mean dynamic

Table 2.10: parameter metrics @45nm TT Process Corner for activation function design architecture using Combinational Logic [6] and CORDIC architecture

Sigmoid Function Bit Precision	Combinational Area (μm^2)	Dynamic Power (μW)	Leakage Power (μW)	Critical Path Delay (ns)	Required No. of Clocks
8_bit Combinational	631.7	33.73	19.12	4.76	1
8_bit CORDIC	307.4	176.60	96.77	4.30	5
12_bit Combinational	4899.5	198.3	110.9	5.10	1
12_bit CORDIC	658.4	252.04	130.3	4.92	5
16_bit Combinational	23408.2	8542.7	3960.00	9.53	1
16_bit CORDIC	782.3	257.4	141.5	5.12	5

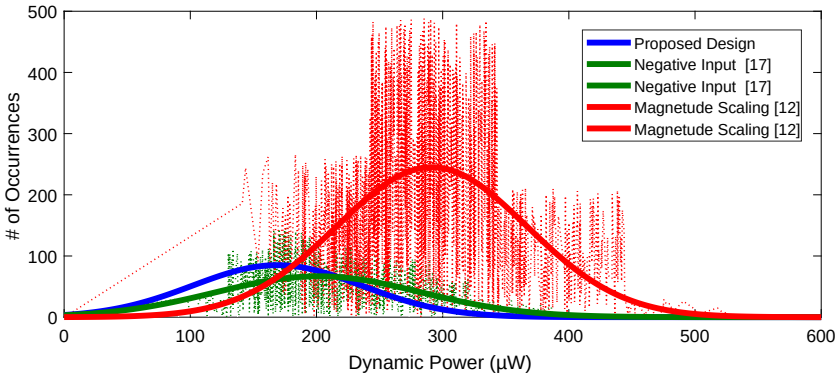


Figure 2.8: 1000 Monte Carlo simulation with process variation and mismatch for mean dynamic power variation of signed 8-bit activation function.

power and σ deviation of proposed architecture are $180.73\mu W$ and $51.7\mu W$ respectively. The mean dynamic power of proposed work is 79% compared to architecture proposed by [47] and 60% compared to [50]. The σ deviation is $51.7\mu W$ in power variation for proposed design which is less compared to the state-of-the-art with a similar approach as it is $66.15\mu W$ and $78\mu W$ in [47] and [50] respectively. It shows the proposed architecture is more reliable in terms of power variation due to process and mismatch.

2.5.4 Enhance performance pipelined CORDIC architecture

We have used the multi-stage CORDIC-based proposed architecture for *sigmoid* and *tanh* activation functions to generate the prediction for both MNIST and CIFAR-10 datasets. We have obtained the final output for different number of stages of the CORDIC architecture and observed the accuracy. The observed inference accuracy plot is shown in Figure 2.9. Further, we have evaluated the physical performance metrics for sigmoid activation function as it makes use of all the computational blocks shown in Figure 2.4. It is observed that for both MNIST and CIFAR-10 data sets, four pipeline stages are adequate for getting comparable accuracy for 8-bit and higher bit precisions. An important point to be noted, is that even with two stages, the maximum drop in accuracy is just 2% in case of MNIST and upto 4% in CIFAR-10. Hence, depending upon the application accuracy requirement and area (and power) overhead, we can see that a designer can use variable pipeline stages for activation function calculation.

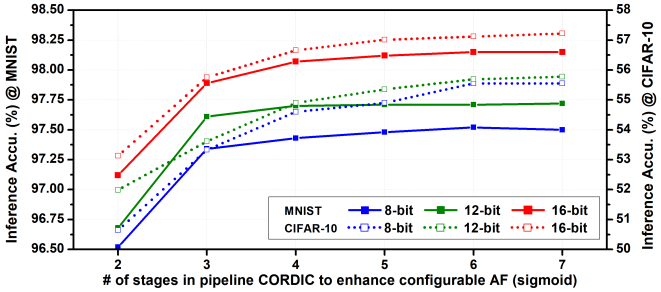


Figure 2.9: Inference accuracy for used Tensor CNN model with proposed sigmoid AF that used n-stage pipeline CORDIC architecture.

Table 2.11: Error Estimation

Sigmoid AF	**Mean Square**		**Mean Absolute**		**Standard Deviation**	
Pipeline Stages	4	8	4	8	4	8
8-bit	10.9×10^{-5}	1.91×10^{-5}	8.59×10^{-3}	3.63×10^{-3}	0.1398	0.13783
12-bit	8.77×10^{-5}	2.63×10^{-6}	7.85×10^{-3}	2.10×10^{-3}	0.1382	0.13768
16-bit	8.75×10^{-5}	2.45×10^{-6}	7.73×10^{-3}	1.91×10^{-3}	0.1381	0.13765

2.5.5 Error estimation for AF with # pipeline stages

We took the input range as [-1, 1] for all the CORDIC iteration-precision combinations. Then, we have divided the input range into an appropriate number of samples based on precision. For example, for an 8-bit precision, we took evenly spaced 2^8 samples from the input range. After choosing the pieces, the sigmoid AF (comes with all hardware components) has been evaluated for the various values of CORDIC iterations and precisions. Finally, based on the CORDIC and NumPy (accurate) sigmoid values, we have calculated the different measures of error, such as Mean Squared Error, Mean Absolute Error and the Standard Deviation. In N-bit precision, one integer bit and rest fractional bits are considered. The mean square error (MSE) for the proposed architecture with four and eight pipeline stages have been evaluated, and it is in the order of 10^{-5}. The Mean Absolute Error (MAE) is in the order of 10^{-3} and minor variations for the Four and Eight stage based evaluations. Therefore, it is observed that the Standard Deviation (SD) for the same is nearly similar, and it persuades to use of four pipeline stages in the arithmetic evaluation of AF. Further, one can note that by increasing the number of pipeline stages in the proposed model, the error can be reduced by sacrificing efficient hardware area utilization and Power-Delay-Product (PDP).

2.5.6 Physical parameters for enhancing performance AF

It can be observed from Table 2.12, that despite higher power consumption as compared to some of the designs, our proposed pipelined method for 8-bit precision returns the best *Energy-Delay-product* (EDP) numbers (up to 3.12× as compared to the previous best [18]) across all the other CORDIC proposed techniques. This is primarily due to the reduced number of clock cycles and reduced critical delay per cycle. The overall delay per cycle is 14% less as compared to the AF

Table 2.12: Performance parameter metrics comparison at 8-bit precision between the proposed and state-of-the-art CORDIC-based designs @45nm TT Process Corner

Config. AF Physical Parameters	**Magnitude Scaled [50]**	**Negative Input [47]**	**Iterative CORDIC [18]**	**Pipelined CORDIC Proposed**
Chip Area (μm^2)	541	483	377	794
St. Power (μW)	176.7	121.7	96.77	198.4
Dy. Power (μW)	299.5	209.4	189.3	327.2
Delay/Cycle (ns)	6.21	5.93	4.34	3.73
Energy-Delay Product (mW/s)	17,134	11,780	6,264	1,958
# clock cycles	6	6	5	1
Inference Accuracy (%)	98	99	98.8	98.8

design proposed in [18] (shown in Table 2.12). This is due to the lower number of registers required in our proposed design (shown in Figure 2.2). The reduced number of clock cycles (and increased throughput) can be ascertained as a direct outcome of pipelining. In our proposed design, while each output is generated four clock cycles after the input is fed, the delay between consecutive results is just one clock cycle. This is unlike the design proposed in [18] which used multiple iterations to compute activation. Considering that the method proposed in [18] takes four iterations to add an output (which means that output produces after every four cycles) to achieve similar accuracy, the number of outcomes generated per clock cycle in this work is still 4× that in the design presented in [18]. Hence, the overall throughput performance of our architecture is 4.65× that of [18].

2.5.7 Performance parameters at different bit precision AF

While in the previous section, we observe that our design shows less delay and better EDP numbers than the other works on CORDIC-based designs, in this subsection, we demonstrate how our work scales with higher bit-precisions as compared to the conventional memory-based designs. Higher bit-precisions have direct impact on the accuracy as can be seen in Table 2.8 where approximately 13% inference accuracy is lost when the bit precision changes from 32-bit to 8-bit for CIFAR-10 datasets. We show, that due to our proposed modifications, we achieve similar (even better for higher precisions) throughput as compared to the memory-based activation function implementation. We compare our design with the memory logic-based sigmoid activation function proposed in[41] and the CORDIC-based design as proposed in [18] for different precision bit-width. The experimental results are shown in Table 2.13. One of the first things we can observe is that our design shows a good balance between area and EDP calculation as compared to [41, 18]. We can also notice, that the memory-based design as proposed in [52, 53, 41, 54] scales poorly with higher bit-precisions. This is due to the fact, that state-of-the-art designs consist of memory elements and a selection logic consisting of $MUX/DeMUX$ and hence, have high area and power overheads.

Table 2.13: Performance parameter matrix for configurable activation function using proposed pipeline CORDIC based architecture and conventional memory-based implementation @45nm TT Process Corner

Configurable AF Precision	**Chip Area** (μm^2)	**Dy. Power** (μW)	**St. Power** (μW)	**Total Logic Delay**(ns)	**Energy-delay Product**
12_bit digital[41]	5220.2	351.7	172.4	8.10	4,245
12_bit CORDIC[18]	658.4	252.0	130.3	20.27	7,749
12_bit Proposed	1439.2	584.1	347.3	3.97	3,697
16_bit Digital [41]	24608.2	8842.7	4230.0	15.04	1.19×10^5
16_bit CORDIC [18]	782.3	257.4	141.5	25.30	10,092
16_bit Proposed	1669.7	671.3	362.4	4.76	4,920
24_bit Digital[41]	–	–	–	–	–
24_bit CORDIC [18]	2226.8	820.3	431.6	29.58	37,031
24_bit Proposed	4948.1	2109.0	965.8	5.02	15,431

2.6 Summary

The semi-custom ASIC approach is investigated for CORDIC architecture. The contribution of the work is two-fold. First, the proposed configurable activation function is able to realize both tanh and sigmoid AF, using the same VLSI architecture. The simulated and synthesized results prove that the proposed CORDIC based AF model is the desirable choice in ASIC based high precision artificial neural network. To draw the quantitative comparison, with different precision performance parameters were calculated. Secondly, the power gating technique is used for saving static power dissipation. This model can be used for the different types of activation function by changing the select signal value, producing multi activation function in neural networks: tanh, sigmoid, etc. Further, throuput enhancement issue addresed by proposing hte pipeline CORDIC architecutre that takes only one clock cycle. We conduct a comparative study of the proposed architecture with other architectures, i.e., iterative CORDIC-based and using memory-based combinational logic. The proposed configurable architecture for multiple activation functions i.e *ReLU*, *sigmoid*, *tanh*, *exponential* is simulated and synthesised at *4*5nm technology to further validate efficiency of proposed architecture in terms of performance parameters. This allows enabling of CORDIC-based architecture for higher precision networks without compromising on the throughput.

Chapter 3

Hardware Efficient and Configurable Design of Neuron using CORDIC Architecture

3.1 Introduction

In the case of hardware implementation of an ANN, increasing computational complexity has an unfavorable impact on area and performance. Moreover, an ASIC design is not adaptable, and any configurability such as flexibility in bit precision, or configurability in the type of activation function such as *sigmoid*, hyperbolic tangent (*tanh*), comes with additional overhead. Computation with higher bit precision (32-bit or 64 bit) is also expensive in terms of an area and power [55]. Thus, a reduction in hardware complexity (i.e bit precision) and faster response without compromising accuracy is highly desirable [56].

The conventional ASIC-based configurable architecture of neurons with multiply-accumulate unit and multiple activation functions proposed in [57, 52, 58] is shown in Figure 3.1. Each MAC unit consists of several multipliers followed by an adder tree. The output of the MAC unit is then fed to the multiplexer which selectively feeds to a particular activation function depending upon an application. The architecture shown in Figure 3.1 has some major drawbacks like area overhead due to array of multipliers and use of separate hardware path (path-A and path-B) for individual activation function. It also leads to increased data propagation delay (due to the MUX) and power dissipation (static power dissipation) due to the unused hardware because only one activation function is activated at a particular time.

An area and power-efficient configurable architecture are desirable at all technology nodes [59]. Keeping this in mind, we propose a *Resource Efficient Coordinate Rotation Digital Computer (CORDIC)-based neuron architecture* (RECON) that provides compelling application opportunities and enables efficient yet configurable computations required in a neural networks. It uses a

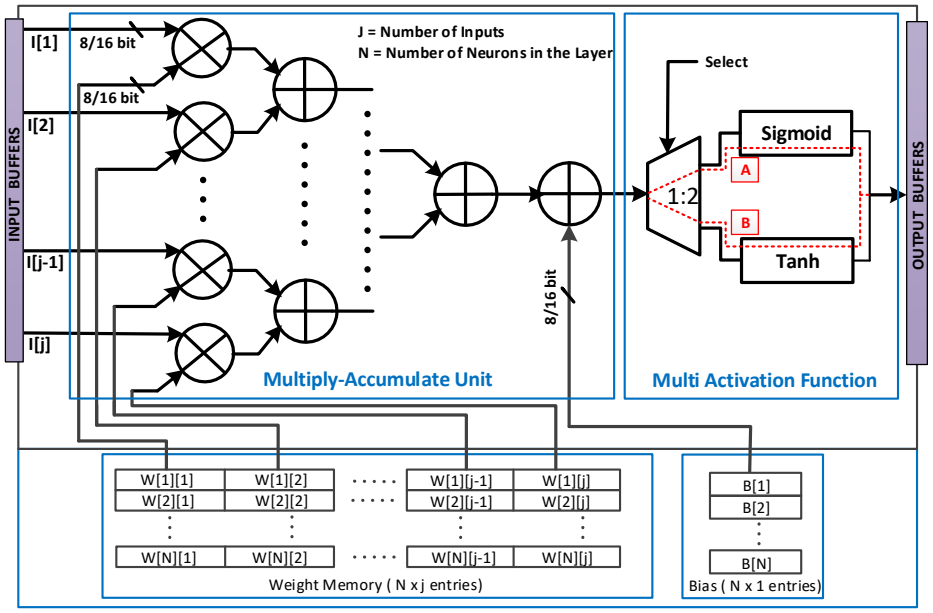


Figure 3.1: The neuron architecture having multiply-accumulate unit with parallel multiplier for j^{th} input and n neurons in the layer followed by multiple activation functions (path-A and path-B)

fixed-point signed arithmetic computation. To overcome the above limitations, we propose RECON which uses CORDIC algorithm. The proposed logic has two benefits– Firstly, it is scalable in terms of area and power as a single block can compute both MAC and activation function. Secondly, it allows configurable activation function *(sigmoid/tanh)* and hence has reduced critical path delay and lesser power dissipation. The proposed design also maximizes hardware reuse as unlike conventional ASIC-based designs which uses *MUX* which leaves out hardware blocks unused.

Contributions

In this chapter, we investigate design strategies and optimizations for RECON which can realize both MAC and activation function computation. The major contributions are summarized in the following points:

- We propose a CORDIC-based design of an unsigned/signed computational unit which can compute both MAC and non-linear activation function.
- We demonstrate how CORDIC is configured within RECON to operate in linear or hyperbolic rotation mode to solve arithmetic (for MAC) and trigonometric operations (for AF) respectively.
- Optimization of the proposed CORDIC-based architecture in terms of area, power, and delay. We further employ power-gating approach and evaluate it with 45nm technology node to demonstrate power-savings. We analyze and discuss the impact of technology scaling on circuit's physical parameters like area, power and delay.

Experimental evaluations show that RECON as a MAC unit, has a lower area and power footprint as compared to the contemporary state-of-the-art designs proposed in [60, 61, 62, 35]. Moreover, our proposed design architecture can support both signed and unsigned number representations. Similarly, in an activation function configuration, RECON returns the best area and power numbers as compared to other designs [50, 47].

3.2 State-of-the-art MAC Implementation Techniques

The basic multiply-accumulate (MAC) unit consists of multiplier, adder, and accumulator blocks. The multiply-accumulate computational unit is shown in Fig. 3.1. The input and output arithmetic relation in the preceding layer of a neural network is shown below:

$$a_j^l = f(\sum \omega_{jN}^l a_N^{l-1} + b_j^l) \tag{3.1}$$

where f is the activation function ($simoid/tanh$) of computational unit k, corresponding to overall neurons in $(l-1)^{th}$ layer. To formulate this equation in matrix form, we assume a weight matrix ω^l corresponding to layer l. Elements of weight matrix ω^l are weights to the inputs of neurons from l^{th} layer with j^{th} row and N^{th} column. The term, b_j^l is the bias here. Most of the earlier work proposed a dedicated ASIC design for the MAC unit. In [63, 60] authors designed a MAC unit using a *Vedic-multiplier* and register-based accumulator. Such kind of architecture is effective for low precision. However, it is not scalable as increasing bit-precision for higher accuracy, leads to a substantial increase in the critical path and the propagation delay. In order to carry out inference for a deep neural network, the authors in [64, 65, 62] resort to a shift-and-add-based multiplication instead of bulky multipliers for MAC computation. Though the architecture was area and power-efficient, it suffered due to low throughput. In [62], the authors proposed calculating one multiplication using shift and addition operation. The proposed approach requires n-shift left and $n-1$ addition for n-bit precision calculation. However, the design is limited for unsigned number calculation as it uses left shift operation.

Configurable n-bit (where n can be any value between 1 and 16) precision MAC unit was proposed using a digital in-memory computing concept [66]. The architecture used an XNOR-based bit-wise multiplier, a full-adder, and an SRAM cell for computation. The circuit design was efficient in terms of area and power but throughput was very low as it needs n-clocks for n-bit precision. Hence, it was not efficient for high-precision architecture. The *Wallace Tree* multiplier based MAC design was analyzed in terms of area, delay, and power in [61]. The proposed architecture was designed using only *AND* and *OR* basic gates. However, it leads to an increase in the critical path and high area utilization. The *double MAC* design proposed in [67, 57], implements two multiplication in a single clock cycle but suffers due to high resource utilization.

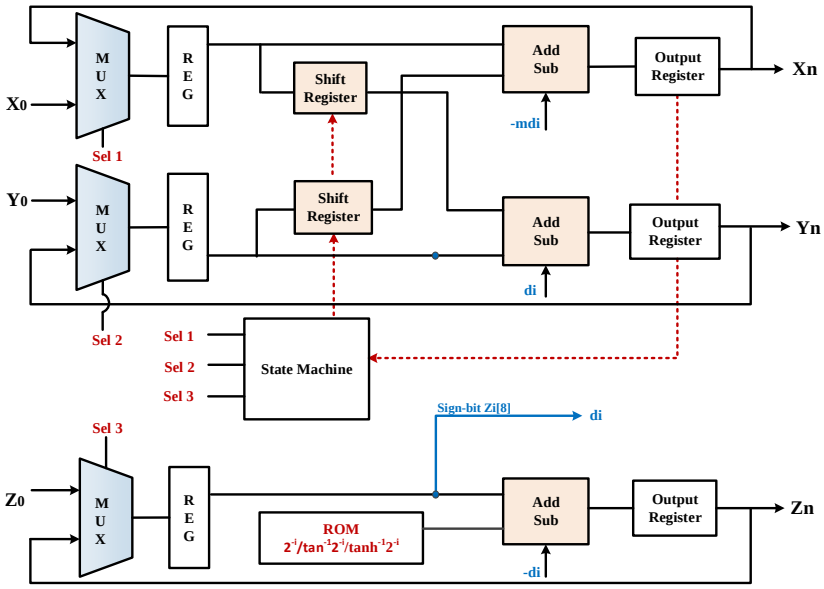


Figure 3.2: Signed 8-bit precision for the basic CORDIC-based design

Table 3.1: Used adder and subtractor functional similarity in CORDIC design architecture

Components	Sum / Difference	C_{out} / B_{out}
Adder	$S = X \oplus Y \oplus C_{in}$	$C_{out} = X \oplus Y \cdot C_{in} + XY$
Subtractor	$D = X \oplus Y \oplus B_{in}$	$B_{out} = \overline{X \oplus Y} \cdot B_{in} + \overline{X}\, Y$

3.3 CORDIC in Linear Mode of Operation and MAC Computation

The genralized linear convenges CORDIC equation is discussed in Section 3.2 and show below in Eq. 3.2 for an individual mode of operation. Here mode $m \in \{1, 0, -1\}$ indicates a circular, linear, and hyperbolic coordinate system, respectively. In this connection, adder/subtractor block is designed for output calculation using the equations shown in Table 3.1. Here, α_i is the memory constant at each i^{th} iteration which is equal to 2^{-i}, $tan^{-1}(2^{-i})$ and $tanh^{-1}(2^{-i})$ for linear, circular and hyperbolic rotation mode respectively.

$$X_{i+1} = X_i + d_i \cdot Y_i \cdot 2^{-i} \tag{3.2a}$$

$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{3.2b}$$

$$Z_{i+1} = Z_i - d_i \cdot \alpha_i \tag{3.2c}$$

In general, the CORDIC algorithm needs n iterations for n significant digits of the fractional part. For higher precision, higher rate of shifting is required which demands more clock cycles for maintaining the computation accuracy. The optimized CORDIC architecture for all modes of operation is shown in Figure 3.2. Further it shows three separate flows for calculation of X_n, Y_n and Z_n. Shift registers are used to right-shift X_n and Y_n. The sign bit of Z_i determines the

direction signal d_i. The functionality of the add/sub-block used in CORDIC architecture depends on the d_i direction. The direction signal is important as it helps to converge the computation iteratively. The d_i represents the rotation direction for i^{th} iteration such that the output at Z converges to 0.

3.4 Design of Neuron Architecture using CORDIC Algorithm :RECON

We proposed a design for configurable activation function in Chapter 2. Design implements a multi-activation function such as *tanh*, *sigmoid* and ReLU etc. using the same hardware resources. Furthermore, design of neural network accelerators requires calculations that involve multiplication, accumulation. We extend the Chapter 2 design for MAC evaluation. The proposed CORDIC-based design enables such computation by using pre-computed constants with shift-and-add operation for fast computation and minimum resources utilization. As compared to a conventional architecture which has dedicated blocks for MAC and activation function calculation, *Resource Efficient Coordinate Rotation Digital Computer (CORDIC)-based neuron architecture* (RECON) focuses on re-utilization of logic architecture (via iteration) for both MAC as well as activation function calculation. The architecture uses linear and hyperbolic rotation mode for multiply-accumulation and non-linear activation function calculation respectively. RECON support for both signed and unsigned computations. The optimized CORDIC architecture with power gating technique is shown in Figure 3.3. The proposed architecture and its operation are described in the following subsections.

3.4.1 Multiply-and-Accumulate computation using RECON

The multiply-accumulation computation technique used in this work depends on the shift-and-add multiplication approach. It has two principal features. Firstly, arithmetic right-shift is used instead of the left shift operation. This technique allows RECON to support both signed and unsigned numbers. Secondly, for mathematical computations (such as addition and multiplication) for an n-bit precision, the iterative convergence using the CORDIC algorithm returns the same accuracy as one gets using conventional combinational logic.

In this work, we focus on fixed-point number representation for multiply-accumulate computation, as the floating-point representation (IEEE-754 notation) in MAC has a higher complexity and power consumption [56]. Moreover, the choice of bit-precision in case of weight/bias constants is one of the important considerations as it directly impacts the area and power of the hardware implementation. In order to reach an acceptable precision that gives a good trade-off between area and accuracy, we have trained the neural network (as shown in Appendix A) for different precision. We got the best accuracy-area trade-off for 8-bit precision among the possible combinations. We have used a 9-bit format that reserve 1 bit for the sign, 3 bits for the integer part, and 5 bits for the

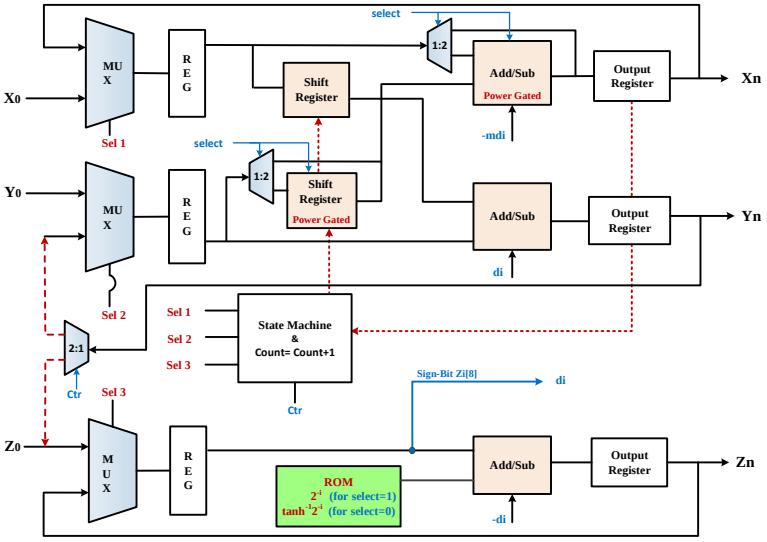


Figure 3.3: The efficient recursive sign 8-bit precision RECON architecture configured by *select* and *ctr* line for MAC and AF computation

fractional part. Fixed $\langle 8, 5\rangle$ represents an 8-bit fixed-point number of which five rightmost bits are fractional. The 8-bit signed fixed-point representation with binary point is shown in Figure 3.4 (a). The most significant bits (MSB) [7:5] are assigned for the integer part and hence the maximum multiplication output range of -7.968 to +7.968 is achieved using this representation.

The MAC operation is realized using RECON as shown in Figure 3.3. In linear mode, the mode variable, m is considered as 0 and E_i is considered as 2^{-i}. The equation 3.2 then translates the general output as shown below

$$X_{i+1} = X_i \tag{3.3a}$$

$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{3.3b}$$

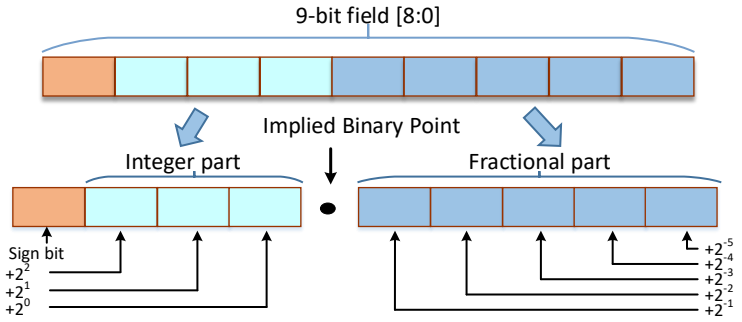
$$Z_{i+1} = Z_i - d_i \cdot 2^{-i} \tag{3.3c}$$

Here, Y_{i+1} computes the multiply-accumulate operation when $Z_0 \to 0$. The computation is valid as long as the binary point is allowed to float. We have considered fixed $\langle 8, 5\rangle$ representation hence binary point is floated for 5 iterations. This calculation assumes that both X_i (input) and Z_i (weight) are fractions within the range of $\{-1, +1\}$. After 5 iterations, Eq. 3.3 translates as follows:

$$\begin{aligned} x_n &= x_0 \quad \& \quad z_n \cong 0 \quad whereas, \\ y_n &= y_0 + x_0 * z_0 \end{aligned} \tag{3.4}$$

where x_0, y_0 and z_0 represent the input, bias value and corresponding weight respectively. This is depicted in Figure 3.3. This implementation is similar to the standard shift-and-add multiplication.

From Eq. 3.4, the CORDIC algorithm for multiplication $x_0 \times z_0$ is derived using a series



(a) Signed Fixed $\langle 8, 5 \rangle$ precision representation used for the calculation

Calculation in Table-II for each iteration:

for i=0: X_i = 000.10110 (Bin) = 0.68750 (Dec) & Y_i = 000.11100 (Bin) = 0.87500 (Dec)

for i=1: $X_{i+1} = X_i$ = 000.10110 (Bin) = 0.68750 (Dec) & $Y_{i+1} = Y_i + d_i * X_i * 2^{-i}$ = 000.11100 + di * 000.01011 (lost bit) 0 (Xi right shift by one bit) = 0.87500 (Dec)+ (-1)* 0.34375 (Dec) = 0.53125 (Dec)

(b) Calculation for i^{th} iteration for multiply-accumulate operation

Figure 3.4: Data representation with binary point and arithmetic calculation

representation for weights shown below:

$$\begin{aligned} x_j = x_{jN} * \omega_N \quad &= x_{jN} * \sum_{i=1}^{j} a_i * 2^{-i} \\ = \sum_{i=1}^{j} x_{jN} * a_i * 2^{-i} \quad &= \sum_{i=1}^{j} a_i * x_{jN} * 2^{-i} \end{aligned} \tag{3.5}$$

The equation states that x_j is composed of a shifted version of input x_0 with respect to weight z_0. This implementation is based on the standard shift and add multiplication. The unknown coefficient a_i may be found by driving ω to zero one bit at a time. If the i^{th} bit of input ω_N is non-zero, x_i is first right-shifted by i bits and added to the current value of y_j. When ω has been driven to zero all bits have been examined and x_j contains the signed product of input vector and weight. The calculation as shown in Eq. 7.2 is carried out using RECON in linear mode. The sign bit of Z is used to calculate the value of d_i. The computation has to perform till $Z_0 \rightarrow 0$ for exact calculation that demands high bit precision computation for approaching zero [68]. The inputs and output of each iteration of a MAC operation is shown in Table 3.2. Calculation for one of the iteration is shown in Figure 3.4 (b). A single MAC operation, thus takes 5 iterations to compute[1]

The output at $Y_{i+1}[8{:}0]$ is the MAC output after 5 iterations which is then used for activation function calculation. The exact calculation using Eq. 3.4, returns a value of Y_n as 0.81054 as shown in Table 3.2. Since we are using quantization for 8-bits precision, the value returned after 5 iterations is 0.78125. We can see that the value returned is just 3% less than the exact(64-bit floating-point calculation) results. In order to have an additional saving in static power dissipation,

[1]Since the binary point is allowed to float for five iteration of right shift, we stop calculating for the value for Y after five iterations.

Table 3.2: Iteration-level calculation shown for MAC computation using CORDIC in linear mode for fixed $\langle 8, 5 \rangle$ representation

i	$\mathbf{d_i}$	$\mathbf{X_{i+1} \rightarrow X_0}$	$\mathbf{Y_{i+1} \rightarrow Y_i + d_i * X_i * 2^{-i}}$	$\mathbf{Z_{i+1} \rightarrow 0}$
clock iteration	d_i for $(i+1)^{th}$ iteration	Initial Conditions/Inputs		
		input data	bias	weight
initial	–	0.68750	0.18750	0.90625
0	+1	0.68750	0.87500	-0.09375
1	-1	0.68750	0.53125	0.40625
2	+1	0.68750	0.68750	0.15625
3	+1	0.68750	0.75000	0.03125
4	+1	0.68750	0.78125 (0.81054)	-0.03125

the add/sub and the shift register blocks are power-gated as they are not required for the MAC calculation (shown in Figure 3.3).

3.4.2 Activation function computation using RECON

The generated MAC output after 5 iterations at Y_{out} is considered as the input for activation function at Z_{in} through feedback and it is controlled by *ctr* pin shown in Figure 3.3. In order to evaluate the *sinh(Z)* and *cosh(Z)* using Eq. 3.6 for exponential function evaluation expressed in Eq. 3.7, we choose m =-1 and K_i=0.8281 as the scaling factors in pseudo rotation which is compensated by applying (i) $1/K_i$=1.2075 at X_0[8:0], (ii) Y_0[8:0] = 0 and (iii) MAC output as input at Z_0[8:0]. Most significant bits (MSBs), Y[8] and Z[8] are used for generating '*di*' signal. The direction, d_i is chosen in the range $\in$ {-1,1} based on the sign of the previous output i.e current input in the each iteration. The signal '*di*' is fed to the adder/subtractor block which decides whether addition or subtraction has to be done. The hyperbolic calculation as shown in Eq. 3.6, hence, transforms as shown below:

$$X_{i+1} = X_i + d_i \cdot Y_i \cdot 2^{-i} \tag{3.6a}$$

$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{3.6b}$$

$$Z_{i+1} = Z_i - d_i \cdot tanh^{-i}(2^{-i}) \tag{3.6c}$$

The CORDIC module is used in hyperbolic rotation mode for realizing *sinh* and *cosh* functions using Eq.3.6.The final desired outputs for hyperbolic calculations (*sinh(Z)* and *cosh(Z)*) are calculated in another 5 clock cycles as we discudded in previous section 3.4.1. After 5 iterations, Y is again reset at the 6th clock cycle for next evaluation. This gives X_n and Y_n as *cosh* and *sinh* functions respectively of the previous evaluation. The *sinh* and *cosh* functions are further used to calculate *tanh* or *sigmoid* function as activation function calculation. Z_n gives the output of activation function applied to the MAC output of the succeeding neuron which is the final resultant.

The generated trigonometric hyperbolic functions are used for producing exponential function as required from Eq. 3.6 are shown in Eq. 3.7. The 8-bit CORDIC output is applied to the adder for producing exponential output as shown in Figure 3.5. The realization of the *tanh* function in the proposed architecture can be represented in terms of *sigmoid* function as shown in the below equations.

$$e^z = sinh(z) + cosh(z) \tag{3.7a}$$

$$f_1(z) = tanh(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.7b}$$

$$f_2(z) = sigmoid(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \tag{3.7c}$$

The proposed work further explores the relationship between *tanh* and *sigmoid* functions using Eq. 3.7. The configuration between *sigmoid* or *tanh* activation function is based on the *select_af* line. The *tanh* function is realized using the Eq. 3.7. When *select_af = 1*, the CORDIC output is directly transferred to the divider through the *MUX* to generate *tanh* function. The subtraction and the shift operation are the two basic operations that are used within the divider circuit [51] for calculating *tanh*. Additionally, in the proposed architecture, the execution of *tanh* function does not require the additional adder block as compared to [50]. However, this unused block can dissipate static power. Addressing this issue and considering the trade-off between leakage current and the speed of operation, *Power Gating* (PG) technique is used to minimize the leakage power and to improve the performance. We have implemented adders with the PG technique used in the proposed architecture as shown in Figure 3.5.

The *sigmoid* function is realized using the Eq. 3.7 when *select_af = 0*. For calculation of *sigmoid*, additional adders and dividers are used as given in Eq. 3.7. The *select_af = 0* is used to activate the adder logic block for *sigmoid* calculation. In comparison to [49], which used an additional step for calculating 2's complement for calculating negative exponents, our *sigmoid* function does not need negative exponents as shown in Eq. 3.7. This gives an additional saving in terms of area and delay. The output from the MAC unit is then applied as feedback along with the initialization of the predefined parameters. The overall design allows reconfiguration, and hence the user can configure the activation function depending upon application requirement. This is the key difference between the previous approaches and the proposed architecture, leading to significant performance enhancement.

3.4.3 Neuron computation using RECON

Figure 3.5 shows the complete RECON architecture. Three partitions can be seen depending upon the configuration. The red encircle represents the CORDIC unit which is used in linear mode for MAC computation. The blue and green encircle represent the blocks required for *tanh* and *sigmoid* function respectively. It can be noticed that only in the case of sigmoid function calculation, all blocks are needed. While, RECON provides a configurable design to implement both MAC and

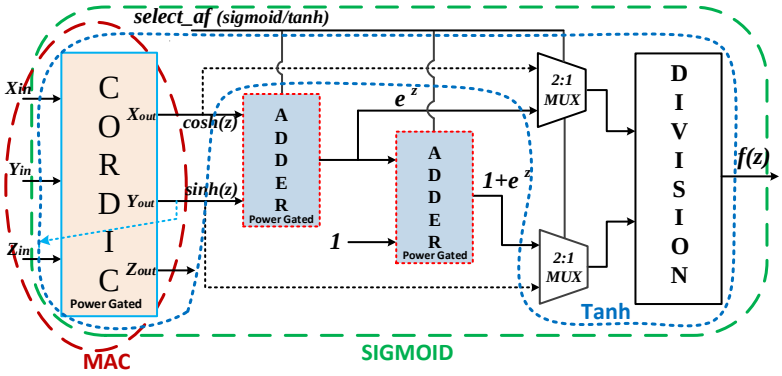


Figure 3.5: Block-level architecture for RECON. The red dotted line shows blocks required for MAC computation. The blue and green dotted lines represent the *tanh* and *sigmoid* function computation

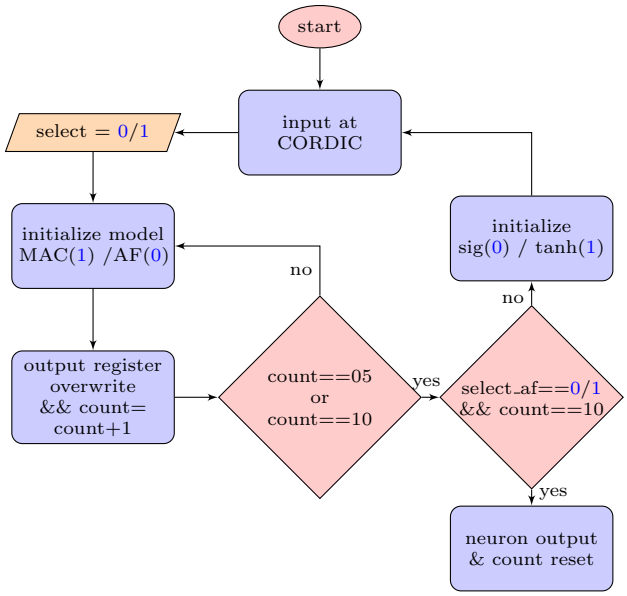


Figure 3.6: Complete flow for RECON architecture to iteratively compute MAC and AFs

activation function, the proposed design components can also be used independently depending upon the user requirement.

The overall flow for a neuron is shown in Figure 3.6. In this CORDIC-based architecture, the *select* signal is set to 0 to compute the multiply-and-accumulate operation. The power gated blocks (add/sub and shift register) connected with the *select* signal, is isolated from the power supply for saving static power dissipation. When the value of *select* is 0, m is set to 0 so that the CORDIC operates in linear mode. The value of the MAC operation is calculated after the five clock cycle as explained before. This is represented by the count variable in Figure 3.6. The output after the MAC computation is then fed as a feedback to the same architecture for initialization to compute the activation function. The generated MAC output at Y_n is the input for activation function at Z_0. Here, *select_af* signal is initialized to the type of activation function to be used. The architecture employs a configurable activation function based on the application requirement. The output of the activation function is calculated in next five clock i.e *count=10*. At *count=10*, the complete computation of a single MAC followed by activation function is done. At *count=10*,

the count register is reset and the final output of a neuron is the input for the next layer neuron. The complete iterative computation for the neuron architecture is shown in Appendix B.

3.5 Results and Discussion

3.5.1 Experimental setup

To evaluate the proposed design architecture, the neuron unit having a multiply-accumulate unit and configurable activation function are represented in *HDL*. The RTL for our RECON architecture is synthesized and results are produced by *Design Compiler*-Synopsys [69]. The netlist file is extracted from *cadence-encounter* and the generated .cdl is used for RTL digital design extraction into CMOS design using the *mentor graphics-v2lvs* [70]. The extracted design is simulated in *cadence-virtuoso* [71] for performance-parameter validation at different process corners, temperature, and mismatch. Following experiments are done to validate our proposed design:

1. The first experiment shows the results for the FPGA prototyping of the proposed design.
2. In the second experiment, we evaluate the effect of power gating technique. We use a logic gate with wide-gate sized transistors so that the performance is not compromised. The coarse-grain technique is used in the design for better efficiency, less circuit complexity, and moderate switching time.
3. In order to evaluate the impact of process variation and mismatch (which increases significantly in $65nm$ and lower CMOS technology), in the third experiment, the circuit is simulated at all PVT variations.
4. In the fourth experiment, the proposed design is compared with the state-of-the-art for both MAC and AF configuration at technology nodes of 45nm.
5. In the last experiment, we carry out Monte-Carlo simulation to model the probability of different outcomes of dynamic power. It helps to calculate random variations in dynamic power dissipation due to device-mismatch in the characteristics of identically designed devices, occurring during the manufacturing of ICs.

3.5.2 RECON waveform and training accuracy for bit precision selection

In order to show the iterative computation the complete neuron architecture, complete waveform is shown in Figure 3.7. Here, we have selected the input (X_i), weight (Z_i), bias (Y_i) for the MAC computation and $Tanh$ function as an activation function.

In order to finalize the number of digits after the decimal point in fixed-point number representation, we carry out experiments using different artificial neural networks. Accuracy comparison at different bit-precision is shown in Table 3.3. It can be seen that their is a very low ($\leq$2%) loss of accuracy between usage of 8-bit and 32-bit fixed-point computation. This leads to reduction in

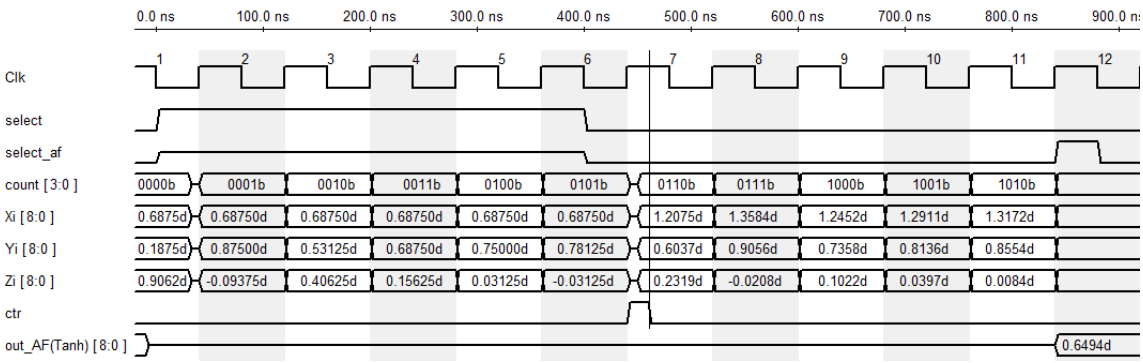


Figure 3.7: Waveform for complete neuron architecture computation with each iteration

memory consumption by factor of 4. Hence, we have taken 8-bit fixed-point with $\langle 8, 5 \rangle$ representation in computation with fractional for RECON. We have given the inference accuracy for the lower bit precision as well as higher bit precision. The output accuracy of the network depends on the application data complexity. Most cases have good accuracy for up to 8-bit precision. Moreover, complex and large datasets require higher precision. However, for 4-bit and lower-bit precision, CORDIC-based DNN implementation is not the appropriate choice. However, for higher precision implementation, using proposed CORDIC-based processing elements and Configurable activation in the DNN framework saves huge resources.

Table 3.3: Accuracy comparison of different DNN architecture at different fixed-point bit-precision computation for MNIST and CIFAR-10 data-set

Fixed Point Data Precision Representation	Training Accuracy (%)		
	LeNet		CaffeNet
	MNIST	CIFAR-10	ImageNet
32-bit	99.1	81.7	56.9
16-bit	98.7	81.2	56.8
8-bit	98.2	80.7	56.7
4-bit	97.6	79.6	06.0
2-bit	85.9	48.0	00.1

3.5.3 Hardware implementation

In order to validate the proposed design, it is implemented on FPGA *ZyboXC7z010-board*. We implement a three layer *4:4:2* fully connected artificial neural network using 8-bit precision (i.e 8-bit weight, bias, input). The single neuron consists of a MAC followed by sigmoid AF design using the proposed CORDIC based architecture. The post implementation hardware resources utilization is given in Table 3.4.

3.5.4 Gate sizing

At lower technology nodes, static power dissipation is the biggest concern. In order to implement power-gating technique, appropriate gate-sizing is an important step. For CMOS-based inverter

circuit with *coarse-grain* power gating technique, the *ON* current and delay variations with respect to technology scaling is shown in Figure 2.6 (b). We determine the power gate size for a larger slew rate with a lower response time and the same is used for our RECON circuit simulations. The add/sub, shift register logic blocks used in Figure 3.3 is designed using the same power getting technique. These logic blocks are isolated when they are not in use in a specific computation task such as multiply-accumulation or *tanh* calculation. We save 30% static power dissipation with minimal area overhead compared to non power-gated architecture. Based on simulation results and merit, the power gate size should be around 3× large as compared to its standard size to maintain similar performance.

3.5.5 RECON in MAC configuration

We compare our RECON architecture in MAC configuration with the state-of-the-art designs [72, 73, 62]. For RECON, the area of the corresponding module performing the MAC is shown in Figure 3.5. For the sake of comparison, we have implemented the designs proposed in [72, 73, 62] with 8-bit precision at 45nm since they were proposed at different technology node. The post-synthesis performance parameters are populated in Table 3.5.

It can be observed from the Table 3.5 that our proposed design has the least area as compared to other proposed designs [73, 72, 62]. While the design in [62] shows the least static and dynamic power, it requires more number of clock cycles along with a larger delay to compute the results as compared to our proposed design. Hence, our proposed design shows a lower *power-delay-product* (PDP) as compared to the design proposed in [62]. To evaluate the overall hardware overheads, we adopt a figure of merit, which is characterized by area, latency, and power product (ALP = area × latency × power) and is 60% less as compared to the architecture proposed in state-of-the-art [62]. An important point to note here is that while other designs focused primarily on unsigned values, our proposed design architecture can support both signed and unsigned numbers. It is particularly relevant considering hardware implementations for neural architecture, as it has a direct impact on the accuracy. Further In the fully connected neural nets, each neuron has hundreds of MAC operations which is not feasible to implement physically on the hardware. However, in the case of lower bit precision, i.e., 2-bit or 4-bit, the parallel MAC units can be deployed but needs to be sacrificed with the accuracy of the application inference with complex datasets. The proposed

Table 3.4: Hardware implementation result for RECON on FPGA Zybo SoC

Architecture	**Resources Utilization**		**On-chip Power** (mW)			**Critical**	
Design	**LUT**	**FF**	**Logic**	**Signal**	**Clock**	**Delay**(ns)	**PDP**
Available Res.	17600	35200	–	–	–	–	–
CORDIC Arch.	17	22	0.10	0.12	0.09	2.49	0.78
Single Neuron	29	41	0.23	0.31	0.28	3.28	2.69
Network 4:4:2	262	318	1.86	2.47	2.12	11.06	71.34

Table 3.5: Comparison for the proposed design and the state-of-the-art for MAC computation @45nm TT Process Corner for 8-bit precision

Parameters	**Wallace tree** [72]	**shift-add** [62]	**Vedic mult.** [73]	**Proposed (CORDIC)**
Area (μm^2)	838	501	1144	307
St. Power (μW)	3.88	2.86	4.93	4.72
Dy. Power (μW)	496.34	119.53	484.83	139.19
Delay (ns)	6.54	4.27	7.66	3.62
Power-Delay-Product	3246.06	510.39	3713.79	503.87
No. of Clocks	2	8	2	5
Integer	unsigned	unsigned	signed	signed/unsigned

Table 3.6: Comparison for the proposed design and the state-of-the-art CORDIC-based design for sigmoid activation computation @45nm TT Process Corner for 8-bit precision

Configurable AF design simulation Parameters	**Magnitude Scaling [50]**	**Negative Input [47]**	**Proposed without PG**	**Proposed with PG**
Area (μm^2)	541	483	377	428
St. Power (μW)	176.00	121.70	101.74	77.94
Dy. Power (μW)	299.40	209.50	189.30	189.30
Delay (ns)	6.21	5.93	4.83	5.12
Power-delay-Product	1,859.27	1,242.33	829.10	969.22
No. of Clock	6	6	5	5

MAC unit has a good design implementation for 8-bit and higher precision. Since increasing the precision of MAC by double, it increases the hardware resources by 4-5 times. In conventional combinational designs, splitting two 16-bit operations into 8-bit operands requires 4 (not 2) 8x8 multiplications and added circuitry for the addition. Hence, the area for a 16-bit multiplier is also 4-5 times that of an 8-bit multiplier. In contrast, proposed design required very few resources (2 to 2.5 times).

3.5.6 RECON in AF configuration

Just as we have evaluated RECON in MAC configuration, we compare our design in *Sigmoid* AF configuration with other works proposed in [50, 47] which employed CORDIC-based architecture for AF computation at both 45nm. Results and comparison of proposed architecture with and without power gating and state-of-the-art are shown in Table 3.6. While, it is to be noted that power gating technique is not applicable for *Sigmoid* AF calculation, the overheads of using power gating technique are applicable, as shown in Table 3.6. It can be observed that our design achieves better numbers for area, power, and delay as compared to other CORDIC based designs [50, 47].

3.5.7 Process variation and mismatch

At lower technology nodes, process variation and mismatch is also an important issue in addition to power dissipation, stability, and reliability. We have simulated the circuit at *45nm* technology node. The *Monte-Carlo* simulation for dynamic power variation due to process and mismatch is

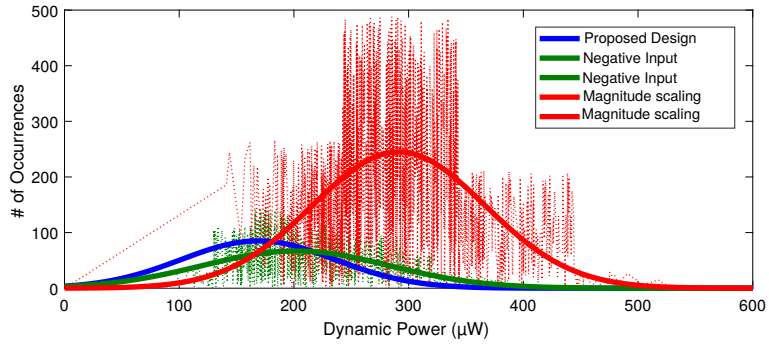


Figure 3.8: 1000 Monte Carlo simulation with process variation and mismatch for mean dynamic power variation of signed 8-bit activation function.

carried out for 1000 samples for the proposed architecture (with *select=0 &* select_af=0) and the state-of-the-art the state-of-the-art as shown in Fig. 3.8.. The mean dynamic power and standard deviation of the proposed architecture are 189.30μW and 58.2μW respectively. The mean dynamic power of the proposed work is 90% compared to architecture proposed by [55] (shown in green color) and 63% as proposed by [50] (shown in red color). The standard deviation for power variation in the case of RECON is 58.2μW which is less compared to 66.15μW [55] (shown in green color) and 78μW [50] (shown in red color) respectively. It shows the proposed architecture is more reliable in terms of power variation due to process and mismatch.

3.6 Summary

To address hardware resourcess limitations for both MAC and AF, the proposed design, RECON uses a single CORDIC architecture with a power gating technique for the realization of an individual neuron. The proposed a resource-efficient and configurable CORDIC-based design, RECON for a neuron architecture performs the computation iteratively. The CORDIC-based design allows configuration and the same block can compute both MAC and multiple activation functions. Additionally, our proposed design can also compute both signed and unsigned computations. The proposed architecture is area and power efficient as compared to the state-of-the-art designs for both MAC and activation function computation. Using extensive evaluation, we show that our proposed design gives better returns at higher bit precision as compared to other designs.

The proposed design is synthesized and verified at 45nm technology using *cadence-virtuoso* for all physical parameters. Implementation of the signed fixed-point 8-bit MAC using our design, shows 60% less area, latency, and power product (ALP) and shows improvement by 38% in area, 27% in power dissipation, and 15% in latency with respect to the state-of-the-art MAC design. Further, *Monte-Carlo* simulations for process-variations and device-mismatch are performed for both the proposed model and the state-of-the-art to evaluate expectations of functions of randomness in dynamic power variation. The dynamic power variation for our design shows that worst-case mean and standard deviations are 189.73μW and 58.7μW respectively which is 63% of the state-of-the-art.

Chapter 4

An Empirical Approach to Enhance the Performance of MAC Unit using Pipelining

Contemporary DNNs implementation faces the burden of excess area requirement due to resources intensive Multiply-Accumulate (MAC) unit. Further, in order to enhance the performance of Deep Neural Network (DNN) accelerators, both compute efficiency and throughput need to be maximized. We design a MAC unit using *Co-ordinate Rotation DIgital Computer* (CORDIC)-based architecture in Chapter 3. Despite being area and power-efficient, one of the significant drawbacks of CORDIC-based design is its low throughput. In this chapter, we propose a performance-centric pipelined architecture of MAC that mitigates low throughput. We conduct a detailed Pareto study of accuracy variation at different precision and the required pipeline stages to achieve high performance.

4.1 Introduction to MAC and its Performance Enhancing Techniques

In DNN, the resources-intensive MAC block has a power-hungry multiplication operation and is more intensive for higher bit-precision arithmetic. The earlier works have proposed efficient customize designs for ASICs and FPGAs. However, state-of-the-art techniques have trade-offs between physical performance parameters, throughput, and accuracy. In addition, hardware implementation of the accelerator has the challenge of bandwidth limitation. Therefore, multi-bit precision (8, 16, 24, or 32-bit) performance parameters have been analyzed for hardware-based acceleration in [74, 18]. The iterative computation shift-and-add based multiplication method simplified the logic arithmetic complexity and required less hardware resources [65]. The efficient design of MAC has been proposed using CORDIC architecture in [18]. However, the iterative CORDIC-based

computations suffer from lower throughput. Furthermore, it requires an n-bit barrel shifter and n-1 additions for n-bit precision computation. The Vedic multiplier-based efficient design for MAC at low precision is presented in [75]. However, the algorithm is not scalable for high-precision architectures due to its increasing critical path and propagation delay. The modified booths algorithm for multiplication has been investigated to save the resources utilization in [76]. Wallace tree-based MAC design is implemented and analyzed the physical performance parameters [77]. Its architecture design is with AND and OR planes which are efficient for lower precision. However, with increasing bit precision, such architecture gets more and more complex.

An approximate multiplier in MAC computation reduces the circuit delay and on-chip power consumption [78]. The error-resilient applications favor approximate computation. An approximate partial product accumulation tree has been presented in [79]. Furthermore, the inaccurate logic compressors in the multiplier also show promising results for hardware implementation [80]. As a result, there is an improvement in power and energy efficiency. The weight sharing scheme reduces the MAC computational complexity and constant storage capacity [81]. A high-performance array multiplier with reversible logic structure has been investigated that increases the throughput performance [82]. However, these improvement techniques do not have generality for multi-precision signed/unsigned computation.

The area, power, and throughput trade-off are addressed in this article by evaluating an enhanced performance CORDIC-based MAC unit for the DNN accelerators. We used pipeline CORDIC stages to address the throughput issue. The pipeline stages come with an area overhead, and the case is managed by evaluating the necessary pipeline stages. We have considered various pipeline stages of the CORDIC-based design to understand the trade-off between accuracy and the number of pipeline stages required as an individual iteration are mutually exclusive. Further, we have evaluated performance accuracy at different arithmetic precision. Compared with 16-bit precision, 8-bit precision computation shows significant performance and saves $4\times$ memory bandwidth at the cost of insignificant accuracy loss. Further, the Pareto study for evaluating the required number of stages helped us save the area utilization with high throughput performance at the cost of insignificant accuracy loss. Next, the proposed design is synthesized and discusses the circuit's physical parameters like area, power, and critical delay using a 45nm technology node.

4.2 Iterative Architecture Benefits and their Limitations in MAC

In the last decade, to improve the MAC performance, research has been focused on efficient design techniques such as [83, 84, 65, 76, 85]. In order to improve the architecture, approximate computing techniques have been primarily focused, especially in the field of DNNs. The method allows reducing the area and power consumption by the DNN accelerator at the cost of acceptable accuracy loss. The on-chip area and power reduction at the expense of throughput have been investigated using recursive Coordinate Rotation DIgital Computer (CORDIC) based architecture in [18]. The

recursive CORDIC architecture is used for the MAC implementation, which performs computation iteratively. CORDIC algorithm uses shift-and-add operation, which needs minimum area and power. The iterative CORDIC architecture requires N+1 clocks for N-bit precision computation, whereas each iterative calculation is mutually exclusive. Hence, pipeline architecture can enhance the throughput performance instead of using iterative architecture in CORDIC.

DNNs have resources intensive architecture due to their heavy computational demands. The throughput performance of DNN is also the major parameter that comes with parallelism. In the edge AI and mobile applications, area and power are on a tight budget. Therefore, we have proposed the efficient design of MAC using CORDIC architecture and enhanced the performance by applying it to the pipeline. Although MAC has pipeline stages that improve the throughput, it comes up with area overhead. Moreover, the neural network is an error-resilient where approximation can introduce. Therefore, we have given a performance-centric analysis to fix the optimum number of CORDIC pipeline stages to make it area and power efficient. In the evaluation process, we primarily analyzed the accuracy performance variation concerning different pipeline stages. Through, a minimum number of required pipeline stages allowed to save a significant area.

The LeNet architecture with the proposed CORDIC-based MAC has been designed. The accuracy has been validated for different pipeline stages. We trained the network until the maximum accuracy gets achieved, and $\approx$ 1% accuracy loss is observed compared to the accurate `Tensor` libraries for MNIST and CIFAR-10. Further, fully connected NN implemented on Vertex-7 shows $\approx 2\times$ better performance than Xilinx MAC-based computation. Additionally, the design's physical parameters are extracted for the 45nm node and compared with previous work. Experimental results show that area-latency-power (ALP) is 12% better than iterative CORDIC and 21% than other best of state-of-the-art. Besides, the proposed pipeline architecture design has $5\times$ higher throughput than iterative architecture.

4.3 Empirical evaluation for enhancing performance of CORDIC-based MAC.

Pseudo rotation coordinate equations converged to linear CORDIC equations form as shown in Eq.4.1 [18] and also described in details in previous chapter.

$$\begin{aligned} \mathbf{X}_{n+1} &= \mathbf{X}_n - m\cdot d_n\cdot \mathbf{Y}_n\cdot 2^{-n} \\ \mathbf{Y}_{n+1} &= \mathbf{Y}_n + \mathbf{d}_n\cdot \mathbf{X}_n\cdot 2^{-n} \\ \mathbf{Z}_{n+1} &= \mathbf{Z}_n - d_n\cdot E_n \\ d_n \in \{-1,+1\};& \quad n = 0,1,2,.....,N \end{aligned} \tag{4.1}$$

Here, mode $m \in \{1,0,-1\}$ indicates a circular, linear, and hyperbolic coordinate system, respectively. The d_n signal generates from $sign(Z_n)$ which give the direction for either addition or subtraction. Further, E_n is the memory elements and for different modes is equal to $\quad tan^{-1}(2^{-n}), \quad 2^{-n},$

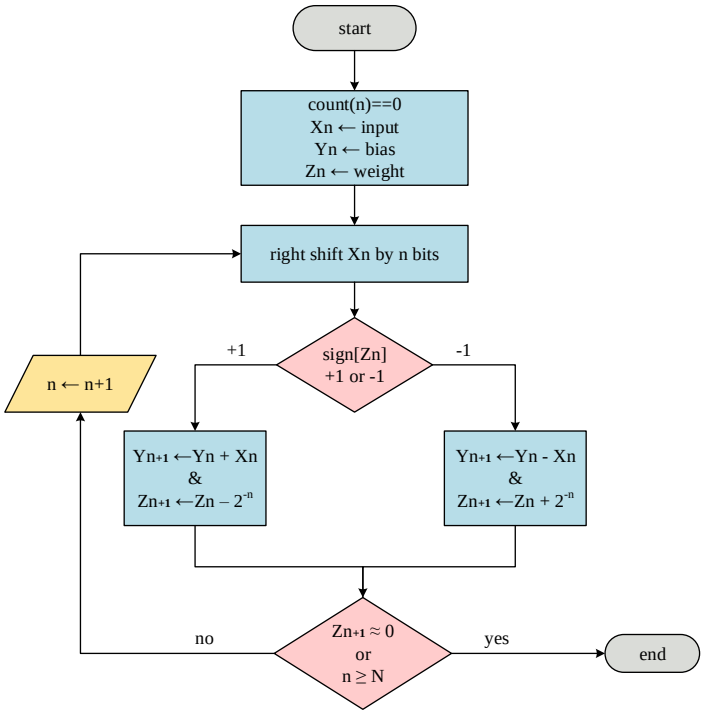


Figure 4.1: CORDIC-based MAC computation design flow. Two conditions have checked to stop the computation i.e either $Z_{n+1} \approx 0$ or n $\geq$ N.

and $tanh^{-1}(2^{-n})$ for circular, linear and hyperbolic rotation computation mode respectively for n^{th} iterations.

One can observed from Eq. 4.1 that hardware implementation requires an addition/subtraction, bit-shift operation and memory elements. In linear mode, set as $m \in \{0\}$ and E_n is 2^{-n} i.e pre-calculated memory element at each iteration. The simplified form of Eq. 4.1 for the linear mode of operation is expressed in Eq 4.2. These Eq. 4.2 perform the MAC computation that is primarily used in the DNN accelerator. The further discussion addressed the linear mode of operation for MAC computation.

$$\begin{aligned} \mathbf{X}_{n+1} &= \mathbf{X}_n \\ \mathbf{Y}_{n+1} &= \mathbf{Y}_n + d_n \cdot \mathbf{X}_n \cdot 2^{-n} \\ \mathbf{Z}_{n+1} &= \mathbf{Z}_n - d_n \cdot 2^{-n} \\ d_n \in \{-1, +1\}; \quad n &= 0, 1, 2, ..., N-1 \end{aligned} \tag{4.2}$$

4.3.1 Multiply-and-Accumulate arithmetic using CORDIC architechure

The iterative CORDIC-based MAC architecture has been proposed in RECON [18], in which majorly used multiplexer, shift register, adder/subtractor, and memory constants. However, from Eq. 4.2 one can be observed that the output at Y_{n+1} performs the accumulation of Y_n and the shifted version of X_n for n^{th} iteration. The iterative computation has to be performed until the output at $Z_n \rightarrow 0$, which means the number of iterations is purely dependent on Z_{in} and the maximum is N+1, where N is the input bit-precision. In discussion we assume X_{in}, Y_{in} and Z_{in}

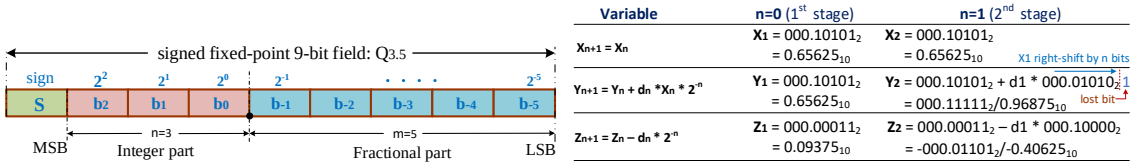
represent the input, bias value and corresponding weight respectively.

In CORDIC computations, multiplication is performed between the input at X_{in} and Z_{in} and thereby accumulating with Y_{in}. In calculation we assumed all X_n, Y_n and Z_n are fractions with 8-bit dynamic fixed-point representation. For signed 8-bit precision MAC computation, Eq. 4.2 needs to be computed maximum for 9-iterations or until $Z_n \rightarrow 0$. Therefore, the generic CORDIC equations in linear mode perform MAC computation, and the final arithmetic operation to be realized is shown in Eq. 4.3. The CORDIC computation evaluation flowchart in linear mode of operation is shown in Figure 4.1. It observed that X_n right-shift and add iteratively with itself until $Z_n \rightarrow 0$. Finally, computation performs multiplication operation between X_n and Z_n.

$$\begin{aligned} & \mathbf{X}_{out} = \mathbf{X}_{in} \quad \& \quad when, \quad \mathbf{Z}_{out} \rightarrow 0 \\ \Rightarrow \quad & \mathbf{Y}_{out} = \mathbf{Y}_{in} + \mathbf{X}_{in} * \mathbf{Z}_{in} \end{aligned} \tag{4.3}$$

The multiplication at Y_{out} i.e $(X_{in} * Z_{in})$ have been achieved through addition of input X_{in} with shifted version of X_{in} with shifted version of X_{in} elaborated below using Eq.4.4. The Eq. 4.4 states that $\mathbf{X}_j$ is composed of a shifted version of input $\mathbf{X}_{in}$ with respect to weight $\mathbf{Z}_{in}$. This implementation is based on the standard right-shift and accumulates technique of multiplication. The unknown coefficient a_i may be found by driving $\mathbf{W}$ to zero single bit at a time. If the i^{th} bit of input $\mathbf{W}_N$ is non-zero, X_i is first right-shifted by i bits and added to the current value of Y_i in CORDIC architecture. When $\mathbf{W}$ has been driven to zero, all bits have been examined, and X_j (Y_n in Eq. 4.2) contains the signed product of input feature X_{in} and weight Z_{in}. Here, we have performed a single MAC operation that computes the final desired output and takes i^{th} iterations due to its iterative computation. This implementation technique is used to the standard shift-and-add operation in a linear mode of operation. Although each CORDIC iteration produces 1-bit accuracy, it is important to notice that output accuracy depends on the input feature precision, i.e., n-bit precision CORDIC produces error around 2^{-n} at $Z_n \rightarrow 0$. Here, Eq.4.4 have elaborated how CORIDIC computation achieved the multiplication for Zin and Xin. However there is no Zin, it is just the Xin shift by bit in each iteration and add with Xin till the Zin goes to zero. Here Zin approaches zero in Zout computation.

$$\begin{aligned} \mathbf{X}_j = \mathbf{X}_{jN} * \mathbf{W}_N \quad &= \mathbf{X}_{jN} * \sum_{i=1}^{j} \mathbf{a}_i * 2^{-i} \\ = \sum_{i=1}^{j} \mathbf{X}_{jN} * \mathbf{a}_i * 2^{-i} \quad &= \sum_{i=1}^{j} \mathbf{a}_i * \mathbf{X}_{jN} * 2^{-i} \end{aligned} \tag{4.4}$$



(a) Signed fixed-point $Q_{3.5}$ precision representation used for the calculation

Variable	n=0 (1st stage)	n=1 (2nd stage)
$X_{n+1} = X_n$	$X_1 = 000.10101_2$ = 0.65625_{10}	$X_2 = 000.10101_2$ = 0.65625_{10} X1 right-shift by n bits
$Y_{n+1} = Y_n + d_n * X_n * 2^{-n}$	$Y_1 = 000.10101_2$ = 0.65625_{10}	$Y_2 = 000.10101_2 + d1 * 000.01010_2$ = $000.11111_2/0.96875_{10}$ lost bit
$Z_{n+1} = Z_n - d_n * 2^{-n}$	$Z_1 = 000.00011_2$ = 0.09375_{10}	$Z_2 = 000.00011_2 - d1 * 000.10000_2$ = $-000.01101_2/-0.40625_{10}$

(b) Calculation for i^{th} iteration approach in linear mode

Figure 4.2: Data representation with decimal point implication used for arithmetic calculation

4.3.2 Enhance performance MAC with CORDIC pipeline stages

The CORDIC iterative computations are mutually exclusive, i.e., each iteration computations are independent of each other. Although iterative CORDIC architecture is area and power-efficient, it requires N+1 clock cycles to evaluate the final output for N-bit precision input. Thus, it is desirable to use pipeline stages to achieve high throughput at the cost of the area overhead. However, area and power are the primary concern in edge-AI and mobile applications. Therefore, to enhance the performance, pipeline CORDIC stages have been designed that improve the MAC performance at the cost of area and power overhead. Furthermore, Pareto points are evaluated to make it performance-centric and discussed in a subsequent section. Although n-stage pipeline architecture requires $n\times$ arithmetic computation of iterative architecture, it is relatively small as a multiplexer, feedback registers, and barrel shifters are not needed in the pipeline design. Therefore area overhead is relatively small as many hardware components are unused, unlike iterative CORDIC architecture with a high delay and require more logic resources.

The signed fixed-point $Q_{3.5}$ arithmetic notation has been used for the design and the implementation as shown in Figure 4.2 (a). The arithmetic computation for the first CORDIC stage with a fixed-point arithmetic nation is shown in Figure 4.2 (b). Here, MSB of the fixed-point computation is a signed bit used for generating d_n in Eq. 4.2. The proposed enhanced performance MAC using pipeline CORDIC architecture shown in Figure 4.3. It is essential to notice that pipeline architecture is relatively efficient for in area and power as it does not require feedback registers and multiplexers, unlike iterative architecture [18]. Further, the design has used an n-bit shift register instead of a barrel shifter. As a result, it generates the desired output at each clock just after the first N-clock cycles as it takes initially by the pipeline stages. Therefore it gives n-times higher throughput computation compared to iterative CORDIC architecture implemented in [18].

The parallel multipliers and adder tree implementation for MAC in neuron computation are costly and burdensome for ASICs and tiny FPGAs. Although single multiplier followed accumulation architecture is preferred due to the utilization of its minimal resources, it suffers from low throughput [18]. We have used a single multiplier followed accumulator in the proposed MAC architecture of **w**-bit precision as shown in Figure 4.3. In DNN, the current layer output is an input to the subsequent next layer. Therefore in practice, the same input and output format is preferred, and that we use implies $\mathbf{i}_{b\_in} = \mathbf{i}_{b\_out} = \mathbf{i}_b$, $\mathbf{f}_{b\_in} = \mathbf{f}_{b\_out} = \mathbf{f}_b$ and $\mathbf{w}_{in} = \mathbf{w}_{out} = \mathbf{w}$,

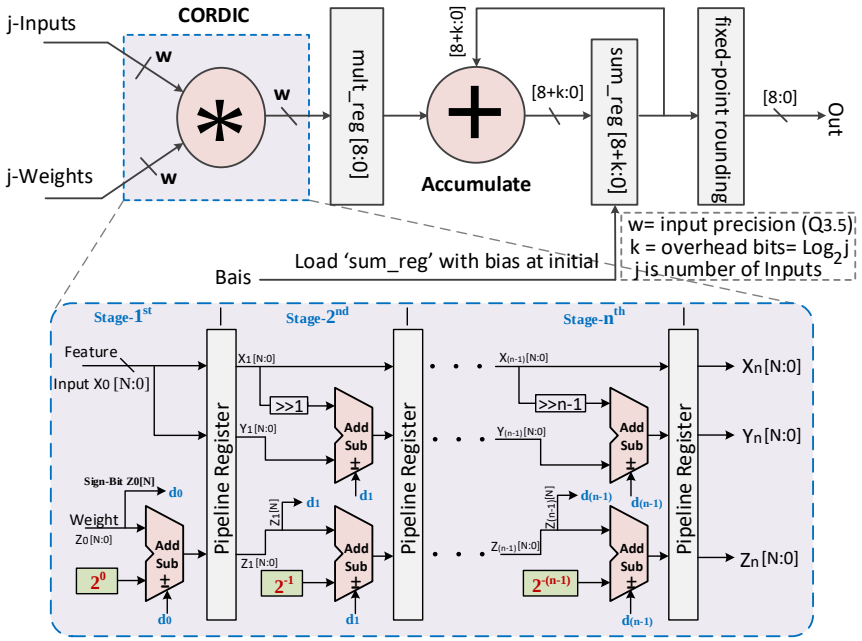


Figure 4.3: Proposed performance enhance pipeline CORDIC-based MAC architecture where each MAC has a single multiplier and adder. The bias constant has to be loaded before computation in the case of a fully connected DNN accelerator, whereas sum_reg will be empty for the first computation in case of the convolution operation. Further, signed N-bit, n-stage pipeline CORDIC architecture for enhancing performance MAC unit is depicted.

where $\mathbf{i}_b$ represents the integer bits excluding the sign bit, $\mathbf{f}_b$ represents the fractional bits and $\mathbf{w}$ represents the total number of bits at input/output. However, in the case of fixed-point format final choice must be based on a target application that gives maximum accuracy with desired precision. The conventional w-bit precision MAC has $\lceil 2w + k \rceil$ output bits in which the w-bit multiplier generates the 2w-bit output bit-width. Whereas CORDIC-based design has the same input and output precision (w) as shown in Figure 4.3. Due to the accumulation in the MAC, we have used extra overhead bits, i.e., k=$\lceil log_2 J(l) \rceil$. Where $J(l)$ is the number of inputs to the MAC unit in the corresponding l^{th} layer of DNN.

The resources-efficient iterative accumulation in MAC has been implemented. It requires $j + n$ clocks to compute weighted sum, where j is the number of inputs at the MAC computation and n is the number of pipeline stages in the CORDIC computation. The rounding scheme has been used at MAC output to resize the output bit size into **w**-bits (9-bits) using *numeric_std* library package of VHDL. It allows us to keep the same input and output precision for MAC implementation with dynamic fixed-point arithmetic representation $Q_{3,5}$ as shown in Figure 4.3. The proposed enhanced performance MAC is workable for both convolution layers and fully connected layers. In fully connected layers, bias has to accumulate in the MAC computation, which is achieved by loading in the sum register for the first clock of MAC computation. The total clock overhead in the l^{th} layer is due to the performance Enhance CORDIC MAC architecture (T_E) in comparison with Conventional single clock multiplication evaluated architecture (T_C) [86] is express using Eq 6.11

where n is the number of CORDIC pipeline stages.

$$T_E(l) = T_C(l) + (n-1) \tag{4.5}$$

Therefore, the total clock overhead for complete DNN acceleration due to pipelined CORDIC-based MAC is further dependent on overall layers $l = 1, 2, ..., L$, available in the DNN accelerator. It is essential to notice that enhanced performance CORDIC architecture has clock cycles overhead. Still, the critical delay of the CORDIC-based logic architecture has comparatively very small, i.e., comparatively, it can operate at a higher frequency and therefore give high throughput. By applying the property of relative clocks evaluation, the simplified equation for comparable clock computation of for the DNN having L-layers is expressed using Eq 4.6.

$$\begin{aligned} T_E = \sum_{l=1}^{L} T_E(l) \quad &= \sum_{l=1}^{L} \Big(T_P(l) + n \Big) \\ &= \sum_{l=1}^{L} T_P(l) + L \cdot (n-1) \end{aligned} \tag{4.6}$$

4.3.3 Pareto analysis for finding bit precision and integer bits in Fixed-point arithmetic

We have used different bit-precision representations such as 8-bit, 12-bit, 16-bit, and 32-bit for our experiments. We take an extra bit for the sign bit and 3 bits for the integer part in all these representations. For example, in the case of 8-bit precision, we have used a 9-bit number format representation. In this representation, 1 bit is reserved for the sign, 3 bits for the integer part, and 5 bits for the fractional part. Symbolically, this is written as $Q_{3.5}$. The $Q_{3.5}$ arithmetic computation format is used in CORDIC algorithm-based modeled MAC architecture. The benchmark MNIST dataset and LeNet network have been used for performance evaluations and Pareto studies. For different arithmetic precision, we have varied the position of the binary point implication, and observed accuracy variation has shown in Figure 4.4.

Further, one can observe two things. Firstly, higher bit precisions return nearly the same classification accuracy, which is apparent for the dataset. However, this factor is less pronounced in simple datasets such as MNIST, where even a 4-bit and the 8-bit fixed-point precision model returns a comparable accuracy. Secondly, since the graph shows inference accuracy for bits used for integer parts, up to 2-3 significant digits to the left of the decimal point are sufficient, and more integer bits don't really help.

4.3.4 Identification of a number of pipelining stages

Due to the iterative nature of CORDIC architecture, it suffers from low throughput issues. Therefore throughput is enhanced by implementing CORDIC pipeline stages. Conventionally, performing the N-bit precision multiplication operation requires N+1 pipeline stages. However, it is challen-

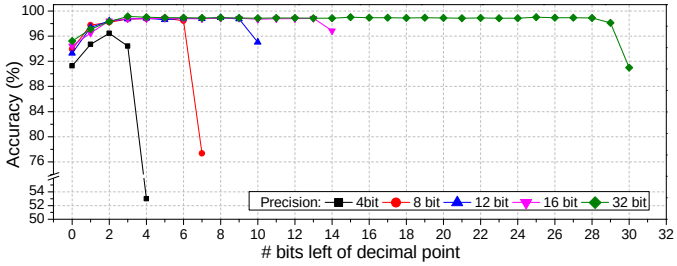


Figure 4.4: Inference accuracy for used Tensor CNN model with proposed sigmoid AF for different precision and different positioning of dynamic Fixed-point.

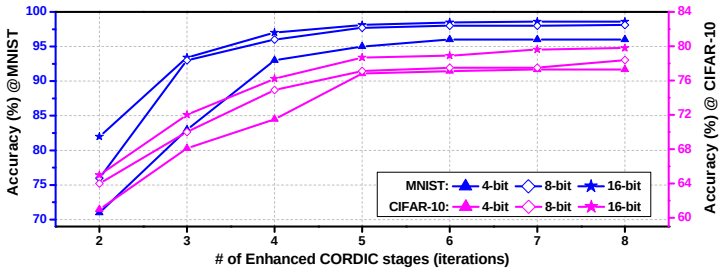


Figure 4.5: Inference accuracy for used Tensor CNN model with proposed sigmoid AF for different precision and different number of pipeline stages.

ging to implement the N+1 pipeline stages for 8-bit and higher precision computation. Initially, we have fixed the dynamic fixed point implication for better performance, i.e., the number of fractional bits and integer bits are found by the Pareto study, which is $Q_{3.5}$. We have designed a CORDIC-based MAC unit for the DNN accelerator. The throughput is enhanced by implementing the pipeline stages. However, Neural Networks are error-resilient, and approximation in computation can help to improve the performance. Therefore, we perform the Pareto study between the number of pipeline stages and validate the accuracy. The study has helped us to fix the minimum pipeline stage with optimum accuracy.

The pipeline architecture gives high-performance computation at the cost of hardware resources overhead. Therefore, it needs to be a balance between accuracy and the number of pipeline stages. In $Q_{3.5}$, the pipeline architecture gives the highest accuracy. As we see higher the pipeline stages, more accurate computation performs until $Z_{out} \rightarrow 0$. Therefore, we systematically Pareto points are extracted for the necessary CORDIC pipeline stages. We experimentally observed that for all 8-bit and higher precision, MAC computation with five and higher pipeline stages returns nearly the same accuracy as shown in Figure 4.5. Therefore, enhanced performance and efficient MAC have been implemented using five pipeline CORDIC stages. The results have been compared with the state-of-the-art. For single-input CORDIC calculation using five pipeline stages is shown in Table 4.1. One can be observed that at 4^{th} and 5^{th} stage results on Z_n closely approach to the zero.

Table 4.1: Iteration-level calculation shown for MAC computation using CORDIC in linear mode for fixed $Q_{3.5}$ representation Processing at each n^{th} stage in pipeline architecture for high-performance MAC computation.

n	$\mathbf{d_n}$	$\mathbf{X_{n+1} = X_n}$	$\mathbf{Y_{n+1} = Y_n + d_n X_n * 2^{-n}}$	$\mathbf{Z_{n+1} = Z_n - d_n * 2^{-n}}$
Pipelined	$(n+1)^{th}$	Initial Conditions/Inputs		
stage	iteration	Input = 0.65625_{10}	bias = 0.00_{10}	weight = 1.09375_{10}
initial	–	000.10101_2	000.00000_2	001.00011_2
0	+1	000.10101_2	000.10101_2	000.00011_2
1	+1	000.10101_2	000.11111_2	-000.01101_2
2	-1	000.10101_2	000.11010_2	-000.00101_2
3	-1	000.10101_2	000.11000_2	-000.00001_2
4	-1	000.10101_2	000.10111_2 / 0.7187_{10}	000.00001_2 (≈ 0)

4.4 Results and Discussion

Based on the Pareto points, we have fixed the hardware design parameters for results extraction. The proposed MAC accuracy is validated on LeNet architecture through software-based evaluation and presented using Verilog hardware description language for assessment and verification on FPGA. Addressing ASIC design, the RTL for our proposed 8-bit precision MAC architecture is synthesized, and results are obtained by *synopsys-design_compiler* [69]. The experimental evaluation is summarised below.

- Inference accuracy for proposed MAC is evaluated for different fixed-point precision numeric representations. The accuracy was assessed using MNIST and CIFAR-10 datasets on LeNet architecture as shown in Table 4.2.
- Through Pareto studies, design parameters for hardware implementation are evaluated. Further different errors are estimated. The physical synthesis parameters are also evaluated at 45nm technology node.
- Functional verification have evaluated for fully connected neural network `196:64:32:32:10` using Vivado-Xilinx. Further throughput and performance are observed.
- MAC RTL digital design convert into CMOS design using *mentor_graphics-v2lvs*[70]. The *Monte-Carlo* simulation for dynamic power variation due to process variations and mismatch is carried out using cadence-virtuoso.
- Sake of comparision; Resources utilization and performacne parameters are reported for LeNET convolutional neural network.

The MAC is a dominant part of the deep neural network that requires more chip area and high computational power. Further, the multiplier circuit has a sizeable critical delay, and it is increasing in large scale for high precision arithmetic. For analyzing the proposed MAC performance, results are extracted at different abstraction levels. We have addressed the enhanced performance MAC

Table 4.2: Network inference accuracy for CORDIC based enhance performance MAC architecture.

Bit-Precision	**Inference Accuracy@LeNet (%)**			
Dynamic	MNIST		CIFAR-10	
Fixed-Point	Tensor[18]	PipeMAC	Tensor[18]	PipeMAC
32-bit	99.3	98.7	81.7	81.2
16-bit	98.9	98.5	81.2	79.8
12-bit	98.9	98.1	80.8	79.1
8-bit	98.8	97.9	80.7	78.4
4-bit	97.4	96.2	77.9	77.4

unit using CORDIC architecture which is efficient in physical parameters while compromising with insignificant accuracy loss. The results evaluation and observations for both ASIC and FPGA are discussed in the following sections.

4.4.1 Performance metric (error measures) and accuracy validation

The compatibility of the proposed architecture in a DNN model is verified by customizing with the LeNet model using which inference accuracy is evaluated for different dynamic fixed-point representations (i.e., 4,8,16,32-bit) for MNIST and CIFAR-10 datasets. It is observed insignificant accuracy loss (<1.5%) for 32-bit to 8-bit precision. Whereas, in the case of 8-bit precision, one can save nearly 5× computational cost and 4× memory bandwidth requirement as compared to 32-bit precision. Therefore, the proposed MAC design has been implemented for 8-bit precision and evaluated for both CNN and FCNN. The inference accuracy comparison for the proposed MAC and state-of-the-art is shown in Table 4.2. One can observe that the proposed design has comparative accuracy performance in comparison with the digital computation technique.

The VLSI design should be efficient in terms of physical performance parameters. On the other hand, DNNs are error-resilient, and learning features can be approximate. Therefore, computation approximation is one of the choices. Area-efficient design has been investigated by evaluating the required pipeline CORDIC stages for MAC implementation. The Pareto study shows that five-stage pipeline architecture is sufficient, and therefore five pipeline stages have been used that reduce the on-chip area utilization and power consumption. For example, we efficiently evaluated pipeline stages' impact on accuracy and fixed minimum pipeline stages, which are five for efficient MAC implementation. The output calculation is validated with different error metric equations. The error metric equations calculation for mean square error, mean absolute error, average error, and standard deviation are given in Table 4.3. The analysis made in Table 4.3 is for the 8-bit precision CORDIC computation with five pipeline stages. The errors are considered for fixed-point $Q_{3.5}$ representation with five pipelines CORDIC stages based on MAC computation.

Table 4.3: Error Estimation.

Performance Metrics (Error Measure)				
Error Metrics	**Mean Square Error (MSE)**	**Mean Absolute Error (MAE)**	**Average Error (AE)**	**Standard Deviation (SD)**
Metric Equation	$\frac{1}{N}\sum_{i=1}^{N}(A-E)^2$	$\frac{1}{N}\sum_{i=1}^{N}\lvert(A-E)\rvert$	$\sum_{i=1}^{N}\frac{\frac{\lvert(A-E)\rvert}{\lvert E\rvert}}{N}$	$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(A-A')^2}$
Measured Value	0.055%	1.82%	5.31%	0.434187

N=samples; *A=approx value*; *E=accurate value*; *A '=mean approx value*;

4.4.2 Hardware resources utilization and comparison

The physical parameters such as resources utilization, power consumption, and delay analysis have been evaluated on Virtex-7 FPGA VC707 Evaluation Kit and implemented using Vivado-Xilinx. In order to compare the proposed architecture with state-of-the-art approaches, we have used a fixed-point $Q_{3.5}$ arithmetic representation and evaluation kit for the implementation of all the previous and proposed architecture. The Verilog HDL is adopted to implement the proposed design. The extracted post-implementation results on the *Virtex-7* FPGA board are shown in Table 4.4.

The performance parameters have been evaluated for various MAC architecture with different design techniques such as Vedic multiplier, Wallace tree, Booth algorithm, shift-add, and CORDIC-based implementation listed in Table 4.4. The proposed design has less hardware utilization compared to the state-of-the-art techniques. Besides, the proposed method has a minimum critical delay, which inherently gives high throughput. Therefore, the proposed enhanced performance MAC has a lower Power-Delay product than the state-of-the-art. In proposed improved performance MAC having CORDIC computation that used two basic operations n-bit *right-shift* and *addition* and used pre-calculated memory constants in the calculation. The proposed design with the right-shift function allows performing both sign and unsigned computation, unlike left shift and add operation [65]. Further, the conventional shift-and-add calculation requires n-clocks to generate the final desired output for n-bit precision computation.

We have extracted the implementation report for the proposed architecture depicted in Figure 4.3. We mentioned before, for 8-bit precision, CORDIC-based computation conventionally required nine pipeline stages. However, more pipeline stages come with an area overhead. Therefore we have evaluated through a Pareto analysis which shows required pipelining is five stages. The pipelined architecture requires five clocks for the very first output, and just onward for every clock, it gives the output as we discussed in Section 4.3. We have shown the results for 8-bit precision (i.e $w=8$) computation unit. Although iterative CORDIC-based architecture uses a minimum area and power budget compared to other state-of-the-art designs, it requires five clocks for every final computation.

The proposed design made area and power efficient by implementing the novel CORDIC-based approach for MAC implementation. The proposed method reduced the critical delay by implementing pipeline CORDIC stages. As we mentioned, results are extracted for signed fixed-point

Table 4.4: Resource utilization and performance parameters at 'fixed-point $Q_{3.5}$' for MAC

Resources Utilization	**LUT** (17600)	**FF** (35200)	**Critical Path Delay** (ns)	**Power-delay Product** (pJ)
Vedic [75]	159	245	4.48	6.11
IEEE [86]	130	49	3.98	5.63
Wallace [77]	105	112	2.59	3.29
Booth [76]	83	61	3.08	3.07
Shift-add [65]	75	58	5.44	4.17
CORDIC [18]	23	22	9.06	1.90
Proposed	58	74	1.86	1.01

Table 4.5: Performance parameters for fully connected NN `196:64:32:32:10` at 'fixed $\langle 8, 5\rangle$' with different MAC unit implementation. Results are produce using Vertex-7 FPGA.

Parameters	**Xilinx IP MAC [86]**	**Iterative CORDIC [18]**	**Proposed pipeline CORDIC**
On-chip Power (W)	2.194	0.67	1.13
Computational Time (in # clock cycles)	344	1640	360
Throughput (GOPS)	4.650	0.977	4.450
Performance (GOPs/W)	2.11	1.45	3.94
Accuracy (%)	95.41	95.06	95.06

8-bit precision representation. The MAC with pipelined five stages consumes 58 LUTs, whereas recursive CORDIC architecture utilizes 23 LUTs. The reason for 2.3× resources scaling is we have not used feedback registers, multiplexer, and barrel shifter, unlike recursive CORDIC architecture. Therefore, resources utilization has not increased proportionally with pipeline stages. Although MAC with pipeline CORDIC used 2.3× resources utilization in comparison with [18], it improved the throughput performance by five times. The critical delay of recursive CORDIC architecture is reported for five iterations in Table 5.2. Further, the power-delay product has been calculated to evaluate the overall performance; it is 30% less as compared to iterative CORDIC-based architecture design in [18] and 20% less as compared to best of the state of the art design.

The performance throughput per watt for the proposed MAC is higher. The proposed design achieved all the benefits at the cost of insignificant accuracy loss, as we observed for the MNIST dataset shown in Figure 4.5. Conventional designs consume 344 clock cycles to complete the computation. In comparison, pipeline CORDIC-based architecture requires four extra clocks in each layer. Therefore, the proposed CORDIC-based algorithm involves a total of 360 clocks cycles for final evaluation. However, the proposed MAC-based DNNs can be operated at a higher clock rate, with nearly half the critical path delay. In these 344 cycles, the proposed DNN design architecture computes 15,963 Multiplication operations, 15,963 addition operations, and 138 AF operations, at 50MHz giving about 4.65 GOPs. Based on these values, we can see the proposed design performs nearly 1.89× better than conventional MAC-based fully connected deep neural network in terms of Throughput/W.

Table 4.6: Resource utilization and performance parameters at 'fixed $\langle 8, 5 \rangle$' for MAC

Performance Parameters	Area (μm^2)	Dy. Power (μW)	St. Power (μW)	Delay (ns)	Clock cycles	Area-Delay-Power(ADP)
Vedic [75]	1164	484.8	4.93	7.66	N	6.10×
IEEE [86]	832	495.5	3.81	6.71	N	3.89×
Wallace [77]	838	498.0	3.88	6.44	N	3.79×
Booth [76]	771	146.7	6.56	7.42	N	1.23×
Shift-add [65]	501	119.5	2.86	4.27	$8N$	2.92×
CORDIC [18]	307	139.2	4.72	3.62	$5N$	1.13×
Proposed	749	322.7	10.5	2.83	N	1.00×

The performance throughput per watt for the proposed MAC is higher. The proposed design achieved all the benefits at the cost of insignificant accuracy loss, as we observed for the MNIST dataset shown in Figure 4.5. Conventional designs consume 344 clock cycles to complete the computation. In comparison, pipeline CORDIC-based architecture requires four extra clocks in each layer. Therefore, the proposed CORDIC-based algorithm involves a total of 360 clocks cycles for final evaluation. However, the proposed MAC-based DNNs can be operated at a higher clock rate, with nearly half the critical path delay. In these 344 cycles, the proposed DNN design architecture computes 15,963 Multiplication operations, 15,963 addition operations, and 138 AF operations, at 50MHz giving about 4.65 GOPs. Based on these values, we can see the proposed design performs nearly 1.89× better than conventional MAC-based fully connected deep neural network in terms of Throughput/W.

4.4.3 Physical performance parameters evaluation and comparison for proposed MAC with state-of-the-art.

The proposed 8-bit MAC architecture's performance parameters such as area, on-chip power, and critical logic delay are compared with the various state-of-the-art designs. The proposed architecture is validated for the ASIC design development process, and physical parameters are evaluated and compared with state-of-the-art methods at $45nm$ technology node. The post-synthesis parameters are evaluated using *Design Vision*-Synopsys. The proposed work and state-of-the-art designs have been synthesized at *45nm* technology, and extracted performance parameters are shown in Table 4.6. In this table, N is the number of input features as discussed in Section 3.2. In this design, we have used an n-bit shifter instead of a barrel-shifter that further lowers the area overhead as compared to the iterative architecture [18]. Although shift-and-add method [65] is having less physical parameters, it needs N-clock cycles for N-bit *(8-clocks for 8-bits)* precision computation which diminish the throughput performance and make incompetent in edge computing application.

The high precision computation in DNN returns better accuracy for more extensive and complex datasets. The CORDIC-based MAC in DNN implementation is the admirable choice for 8-bit and

higher precision computation [18]. Whereas, at worst case 8-bit precision, physical parameters are evaluated as shown in Table 4.6. In this table, N-represents the number of accumulations performed by MAC, which further depends on the number of inputs features at the input of the MAC unit. To evaluate the overall chip-cost, we have used figure-of-merit, the area, latency, and power product *(ALP= area* $\times$ *latency* $\times$ *power)*. The ALP is 12% and 21% less compared to the iterative CORDIC-based architecture proposed in [18] and other best of state-of-the-art work [65] respectively. It is essential to notice that iterative CORDIC-based MAC requires 5-clock cycles for each multiplication, and therefore in each neuron, we need $5n$ clock cycles for n multiplications and accumulation.

Consequently, such designs can be helpful where area and power are in a tight budget and throughput can be compromised. The proposed architecture uses 5-clocks for the first multiplication due to its pipelined nature, and afterward, it computes each multiplication at every clock cycle as depicted in Eq. 4.6. Therefore, one can say MAC is useful in the high throughput edge AI and mobile application.

4.4.4 Process variation and mismatch analysis

Process variation is the device and peripherals characteristics such as length, widths, oxide thickness when the integrated circuits are fabricated. The amount of process variation becomes particularly pronounced at smaller process nodes (below $65nm$) as the variation becomes a more significant percentage of the total length or width of the device and as feature sizes approach the fundamental dimensions during the lithography masks. Therefore, process variation and device mismatch have a significant role in circuit physical parameters' stability and reliability at lower technology. There is a substantial variation in static current due to process variation and mismatch. Monte-Carlo simulation calculates the probabilistic distribution of dynamic power variation due to process variation and device mismatch in the characteristics of similar design devices, which occur during the manufacturing of IC's. Power-Delay product at lower technology is more sensitive for naturally occurring variation at lower technology note. The Monte-Carlo simulation is carried out for 10,000 samples to validate the power variation due to process and mismatch.

The RTL design is extracted into the custom-CMOS circuit design using *v2lvs-Mentor Graphics*. The Mont-Carlo simulation is performed in *cadence-virtuoso* for 2500 samples as shown in Figure 4.6. The simulation result is extracted at *45nm* technology node. The proposed design has less dynamic power variation and standard deviation. The mean dynamic power and σ deviation at $45nm$ node is $319.7uW$ and $2.45uW$.

4.4.5 Design and Implementation of LeNET using CORDIC-based MAC

In order to comare the propsoed archtiecture resources utilization and other performance parameters with conventional vivado based MAC we have given the comparision table

We can observe that the state-of-the-art MAC based LeNet-5 architecture takes a huge resource-

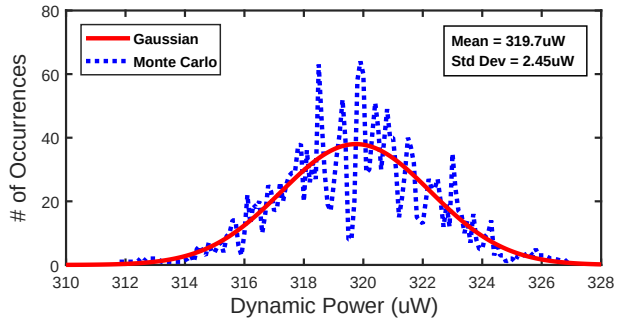


Figure 4.6: Dynamic Power consumption by CORDIC based MAC unit having five pipeline stages. The Monte-Carlo with process variation and mismatch simulation is perform for 2500 samples.

Table 4.7: Hardware utilization of LeNet-5 architectures in FPGA

LeNet-5	LUT(53200)	FF(106400)	BRAM	Power(W)
Network with MAC [7]	35399	19459	40	0.185
Using Iterative CORDIC MAC	9733	17490	15.5	0.180
Using pipeline CORDIC MAC	11855	17063	39	0.181

sas compared to propsoed CORDIC MAC based implementation. Iterative CORDIC MAC-based LeNet-5 architecture consumes less resources (area) and less power, however it comes with significant throughput loss as discussed in Chapter 3. Therefore the design can will be very much useful for AI-enabled IoT applications. The proposed pipeline MAC-based Lenet-5 architecture consumes a nominal resource utilization and power with high throughput (5× that of iterative MAC). The proposed pipeline MAC-based LeNet-5 architecture takes fewer resources and power than state of art MAC-based LeNet-5 architecture but consumes more resources and more power when compared with iterative CORDIC MAC-based LeNet-5 architecture. However the design has hogh throughput and therefore architecture is well suitable for the edge-AI applications. The observed results for conventional MAC and proposed arhctiecture infered that MAC plays a key role in hardware implementation of DNN Accelerator.

4.5 Summary

To enhance the performance of Deep Neural Network (DNN) accelerators, both compute efficiency and operating frequency need to be maximized. Contemporary DNNs implementation faces the burden of excess area requirement due to resources intensive Multiply-Accumulate (MAC) unit. We design a MAC unit using *Co-ordinate Rotation DIgital Computer* (CORDIC)-based architecture. Despite being area and power-efficient, one of the significant drawbacks of CORDIC-based design is its low throughput. In this work, we propose a performance-centric pipelined architecture of MAC that mitigates low throughput. We conduct a detailed Pareto study of accuracy variation at different precision and the required pipeline stages to achieve high performance. Implementation reports are evaluated, and using the Vertex-7 FPGA board extracts performance parameters for DNN. This paper proposes enhancing performance MAC computational elements using a pipelined

CORDIC-based scheme that takes only one clock cycle and has a minimum critical delay. The proposed MAC design will compute for both signed and unsigned computations. We conduct a comparative study of the proposed architecture with the other design technique. A detailed analysis shows selecting circuit design parameters can adapt to different architectures in the efficient design of the DNN accelerator. Using extensive evaluation for FPGA and ASIC, we show that our proposed method gives better returns than previous work.

Further, the MAC design explores for ASIC, and post-synthesis results are extracted at $45nm$ technology node. The proposed architecture is scalable for any bit-precision. As a result, the proposed MAC with 8-bit precision achieves a better performance metric - lower area-delay-product (ADP) (1.11×) and DNN have enhanced throughput (2.73×) as compared to the recursive CORDIC-based MAC and nearly 1.89× better than conventional MAC-based fully connected Neural Network. The proposed performance enhancement technique in system architecture opens new opportunities for low area and power with enhanced performance design, mainly in the Edge-AI applications. We will make our project code open-source after acceptance to assist reproducible results and hardware implementation, i.e., for FPGA and ASIC.

Chapter 5

Hardware Implementation Layer-Multiplexed High-performance DNN Accelerator

Implementation of deep neural networks (DNN) in real-world problems needs huge hardware resources, high computational power, and more memory bandwidth. The AI-enabled IoT applications require resource-efficient DNN engines. The requirement of hardware resources for DNN implementation is proportional to the depth of the neural network. In this chapter, we proposed an enhanced performance and resource-efficient design architecture for the DNN engine. The design have implemented on system level proposed layer-multiplexed DNN accelerator. Since proposed architecture have reused the single layer within DNN comes with throughput limitations, which overcome by implementing the pipelined MAC design that proposed in Chapter 4.

5.1 Introduction

In DNN, the multiply-accumulate (MAC) unit is the most hardware expensive computational element in an any deep neural networks [74]. In addition, it consumes more than 90% computation time in the DNN accelerators [76]. The issue has been addressed using *Coordinate Rotation DIgital Computer* (CORDIC) algorithm [18]. It can perform different arithmetic computations such as trigonometric functions, multiplication, and many, which Jack E. Volder invented in 1959 [87]. Later, an efficient algorithm for a configurable mode of operations and its applications were expanded by John Walter in 1971 [88]. Algorithm can be used in hardware design for carrying out arithmetic operations of a neuron [18]. An efficient novel neuron computational unit has been proposed using recursive-CORDIC architecture [18]. As compared to the state-of-the-art, CORDIC-based neuron

architecture outperforms in the area and power reports compared to previous work but suffers from low throughput due to its recursive architecture [18, 89]. The CORDIC-based architecture inherently uses the concept of the recursive approach, which leads to low throughout performance. The issue will becomes serious for Layer multiplexed architecture. Conventionally CORDIC requires n clock cycles to the final results evaluation for n-bit precision computation evaluations.

The state-of-the-art designs have addressed design optimization at different abstraction levels. An important arithmetic operation multiplication is efficiently computed in the previous work based on booth's algorithm, Wallace tree adder, Vedic multiplier, etc., [90, 76, 75, 77]. Besides, designs have optimized by using techniques such as hardware reused [91], bit-serial computing [92, 93], bit rounding [94, 84, 95, 96], using CORDIC arithmetic [89, 18] and computational approximation in error resilient applications [97, 98, 99, 100]. All the state-of-the-art techniques has trade-off in performance parameters. Out of these techniques, recursive CORDIC engine-based implementations offer an area and power-efficient implementations of MAC unit in DNN [18]. The CORDIC-based design have many advantages. First, it significantly reduce the area and power utilization. Secondly, the design have more benefits at higher precision arithmetic due to it computational simplicity. The numerical accuracy of recursive CORDIC-based MAC is determined by the n-iterations; the higher the iterations, the better the accuracy, but it dominates the throughput by n times. Secondly, the hardware design of general CORDIC architecture uses a barrel shifter which comes with more area overhead and has a high critical delay.

Though the recursive architecture of CORDIC is area and power-efficient, it is not desirable for high throughput arithmetic computation application e.g DNNs. In order to implement area and power MAC unit along with high throughput, we have evaluated the required pipeline stages using the Pareto analysis. Therefore, the pipeline architecture of CORDIC has been used in the to enhance the throughput at the cost of minimal area overhead. The proposed pipeline MAC based DNN design returns better performance. Studies also confirm the decimal-point implication for better accuracy and efficient hardware implementation. The pipeline CORDIC-based MAC has enhanced the DNN accelerator throughput. Furthermore, to make the efficient hardware design on tiny FPGAs, we have reused the single layer of a fully connected neural network to implement any deeper neural network. The proposed dynamically configurable layer-multiplexed DNN architecture is efficient interns of both hardware utilization due to resources reused technique and throughput as pipeline stages has been used in the MAC. The major contributions are summarised in the following points:

- Enhance Performance: The enhanced performance CORDIC-based MAC is proposed. CORDIC pipeline stages have been used to improve the performance. Further, the design has used a 1-bit shift element to minimize the critical delay, unlike conventional architecture that uses a barrel shifter.

- Hardware Implementation: Dynamically reconfigurable and layer-reused DNN accelerator have implemented. The architecture can be configure on an FPGA with any depth. However,

benchmark `196:64:32:32:10` configuration have presented for the evaluation and comparison.

- Pareto Studies: An empirical approach is presented to enhance the system performance that evaluates the required CORDIC pipeline stage in the Neuron unit. Further, Pareto point extracted for one between the number of bits used in the integer part and its impact on accuracy, and second between *max_norm* and optimized # CORDIC stages by applying their implications for accuracy.

This work implemented DNN `196:64:32:32:10` on Virtex-7 VC707, proposed design validation results evaluation, and comparison of performance parameters with the state-of-the-art. The network has trained using software implementation, and weights are extracted for the MNIST dataset to validate the hardware implementation. The proposed enhanced performance MAC-based layer-reused DNN has achieved 95.06% accuracy where observed 0.35% accuracy loss against architecture with accurate fully connected DNN [7]. At the cost of insignificant accuracy loss, the proposed architecture has a better performance than state-of-the-art layer-reused architecture. Against the fully connected neural network, the proposed design saved 45.81% slice LUTs relatively without significantly affecting the throughput [101].

5.2 Related Work on DNN Design Techniques and Motivation

The mobile devices with resource-constrained hardware for DNN deployment are attracting much attention. Further, the success of AI drives application-specific circuit design for the area and performance-efficient DNN computation [102]. The FPGA and ASICs have elevated for custom hardware implementation of DNNs. DNNs compose computation performed for many layers which comes with the area and power overheads [103], and hence in DNN area and power constraint challenge increasingly prominent especially in mobile devices and edge-AI applications. An ASIC design as Tensor processing unit (TPU) [104] has the scope for further improving the performance through the proposed enhanced performance MAC. In the state-of-the-art, dynamically configurable ASIC/FPGA DNN framework have presented such as Eyeriss [105], Gemmini [106], Simba [107] etc. The proposed performance-centric pipelined MAC has good performance parameters and can deploy for such frameworks; However, pipeline MAC needs a state machine to control.

Primarily MAC unit has a multiplier, adder, and accumulate register as shown in Figure 5.1. Parallel multipliers and adder trees can be used for high-throughput MAC implementation at the cost of area and power overhead. Whereas iterative MAC architecture is resources-efficient, but throughput is sacrificed [108]. The MAC operation have investigated at different abstraction level such as hardware reused [91], bit-serial computing [92], data quantization for lower precision logic implementations [94, 84, 95, 96], functional configurability [89, 18] and computational approximation can be used in error resilient applications [97, 98, 99, 100]. Furthermore, MAC have efficiently

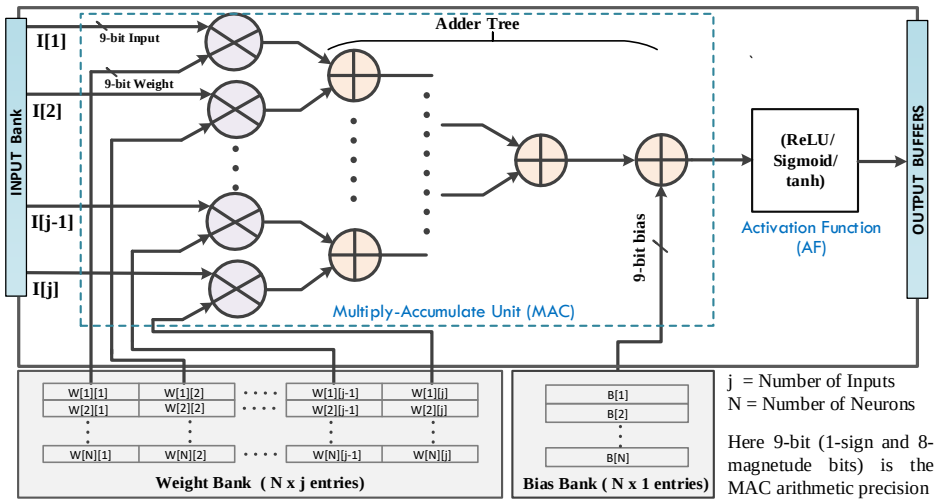


Figure 5.1: Neuron's internal computational units having CORDIC-based iterative MAC unit followed by activation function.

design using booth's algorithm, Wallace tree adder, Vedic mathematics, etc., [90, 76, 75, 77]. Each design technique is good for its individual tasks, but not efficient for others i.e circuit designs are in trade-off for area, power, throughput, and accuracy.

In DNN, limited MAC units on FPGA cause a problem to implement the deeper NN and DSP blocks become the bottleneck in case the direct hardware mapping on FPGA have required [109]. Therefore, explorations of hardware efficient, such as layer reused, have prompted more attention to reduce area utilization [110]. However, the throughput loss on layer-reused technique is significant due to iterative layer by layer computation. Various design techniques have been proposed for optimization of the performance and physical parameters in DNNs [15, 111]. As DNNs has high computational demand, custom design architecture are vital for enhancing the performance. The FPGA is the more efficient choice for the DNN implementation and it has high computational demand, and custom DNN architectures are vital to enhancing the performance [112] . However, tiny FPGA has limited resources and memory bandwidth that limits the performance [113].

In the last decade, the DNNs with a higher number of layers were studied in order to improve the accuracy [114, 101, 26, 39, 115, 19]. Such architectures substantially increase the number of neurons that increase the number of MAC utilization, which in turn increase the hardware resources demand, processing time and power consumption. However, hardware-overhead reduction scheme have used for implementation of high cost computational block such as MAC and AF. Further many efficient hardware design techniques have presented in the state-of-the-art in [20, 57, 84, 116]. The DNN development on FPGA takes few months for an expert hardware (HW) designer. Hence, dynamically configurable design of DNN is feasible to model efficiently.

In this connection, resources-efficient layer-reused DNN have designed but it leads to lower throughput due to iterative layer-by-layer computation [117, 19]. The iterative CORDIC architecture is efficient for area and power, and can perform the MAC operation [18]. Though the design is efficient for area and power, it is suffered from lower throughput due to its iterative

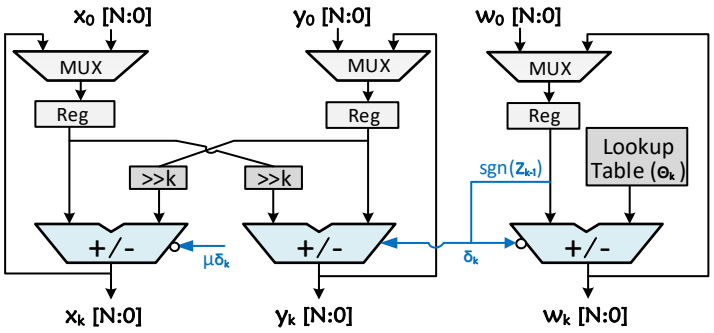


Figure 5.2: Signed N-bit precision recursive CORDIC architecture that configure to linear, circular and hyperbolic, where lookup table is $\Theta_k = 2^{-k}$, $tan^{-1}(2^{-k})$ & $tanh^{-1}(2^{-k})$ respectively [18].

architecture i.e, first, design computes the final desired output in n^{th} iteration, which accountable for $n\times$ lower throughput. Second, the hardware design of CORDIC architecture used a barrel shifter which leads to more hardware resources utilization and increases the computational logic critical delay. We presented the performance centric CORDIC-based neuron architecture that area and power-efficient and significantly less critical delay which leads to higher throughput in ANN engine computation. on the other hand, we have designed a layer-reused system architecture with pipeline CORDIC based MAC which is efficient for area and power with enhanced performance on the same platform. In MAC, Pareto points analysis evaluated design parameters for better performance. Moreover, the proposed design is feasible for both ASIC and FPGA implementation.

5.3 CORDIC Optimization & Hardware Efficient Multiply-and-Accumulate unit Implementation

The MAC is the key resource-hungry computational element in Neuron and limits the ANN computation throughput. Further, MAC consumes 90% of the overall computational power of the neural network [118]. The CORDIC computation have used for the MAC implementation as it gives area and power-efficient architecture, but it suffers from low throughput due to its iterative nature [18]. This section discuss the performance centric CORDIC-based MAC architecture within the neuron and state machine have discuss in the following subsection.

5.3.1 Introduction to CORDIC architecture

The CORDIC equation for all mode of trajectory in pseudo rotation computation is converged for hardware implementation to the following equations:

$$x_{k+1} = x_k - \mu \cdot \delta_k \cdot (y_k \cdot 2^{-k}) \tag{5.1a}$$

$$y_{k+1} = y_k + \delta_k \cdot (x_k \cdot 2^{-k}) \tag{5.1b}$$

$$w_{k+1} = w_k - \delta_k \cdot \Theta_k \tag{5.1c}$$

Whereas, δ_k = -1 if $w_k < 0$, +1 otherwise. Further mode $\mu \in \{0, 1, -1\}$ indicates a linear, circular, and hyperbolic coordinate system, respectively. Θ_k is the convergence angle memory constant at each k^{th} iteration which is equal to 2^{-k}, $tan^{-1}(2^{-k})$ and $tanh^{-1}(2^{-k})$ for linear, circular and hyperbolic rotation mode respectively. It can notice that in linear mode of operation i.e for $\mu = 0$, the Eq. 5.1 (a) becomes independent on the any variable parameters involved in the iterative calculation.

The CORDIC calculations in linear mode of operation uses $\mu = 0$. Therefore, the Eq. 5.1 have been revised as shown in Eq. 5.2. In linear mode of operation the lookup table constants are used to be Θ_k=2^{-k} for every k^{th} iterations. The hardware implementation of CORDIC equation in linear mode of operation adder/subtractor, barrel-shifter, multiplexer and feedback registers are requires for y_k and w_k computations. Further, it can be notice from Eq. 5.2, no logic operations performed at x_k in linear mode, whereas y_k evaluation requires k-bit barrel shifter in each CORDIC stage to compute the final desired output. The signal δ_k represents the rotation direction for k^{th} iteration decides the addition or subtraction arithmetic computation at y and also for w converges to 0 as shown in Figure 5.2.

$$x_{k+1} = x_k - 0 \cdot \delta_k \cdot (y_k \cdot 2^{-k}) = x_k \tag{5.2a}$$

$$y_{k+1} = y_k + \delta_k \cdot (x_k \cdot 2^{-k}) \tag{5.2b}$$

$$w_{k+1} = w_k - \delta_k \cdot \Theta_k \tag{5.2c}$$

5.3.2 Enhanced performance multiply-and-accumulate unit for high-throughput applications

We assumes x_i is the input feature and w_i will be the corresponding weight, whereas y_i is the neurons' bias constant. The CORDIC Eq. 5.2 for multiplication of x_0 and w_0 and addition with bias y_0 have realized and elaborated below using Eq. 5.3. The multiplication will be performed by the CORDIC architecture at the y_{out} is $x_{in} \times w_{in}$ with bias addition. It is achieved through addition of shifted version of w_0 with input x_0 as explained below using Eq.5.3. The Eq. 5.3 states that x_j is composed of a shifted version of input x_0 with respect to weight w_0. This implementation is based on the standard right-shift and add based multiplication. The unknown coefficient a_n may be found by driving ω to zero one bit at a time. If the n^{th} bit of input ω_N is non-zero, x_n is first right-shifted by n bits and added to the current value of y_n in CORDIC architecture. When ω driven to zero, all bits have examined and x_j (y_n in Eq. 5.2) contains the signed product of input

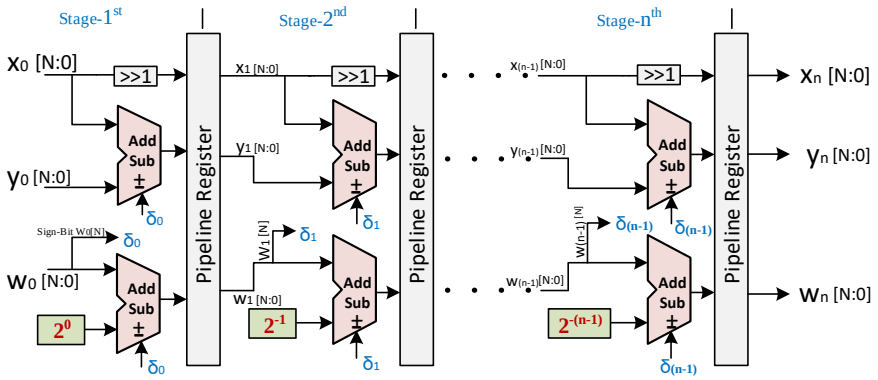


Figure 5.3: Hardware optimized n-stage pipeline CORDIC architecture for enhancing multiplication and bias addition in MAC.

feature (x_0) and weight (w_0).

$$\begin{aligned} x_j = x_{jN} * \omega_N \quad &= x_{jN} * \sum_{n=1}^{j} a_n * 2^{-n} \\ = \sum_{n=1}^{j} x_{jN} * a_n * 2^{-n} \quad &= \sum_{n=1}^{j} a_n * x_{jN} * 2^{-n} \end{aligned} \tag{5.3}$$

Conventionally 8-bit precision computation needs to iterate a set of equations until we get *Z=0* or a maximum of 8-iterations [18]. Thus, implementation is efficient for area and power at the cost of lower throughput. In this work, we have investigated the CORDIC-based MAC by implementing the pipeline CORDIC stages. However, pipeline architecture comes with more hardware utilization. This work have proposed performance-centric pipeline CORDIC architecture as shown in Figure 5.3. The proposed hardware architecture realizes MAC operation at n^{th} stage. The optimized CORDIC architecture in linear mode have realized using Figure 5.3 with Eq. 6.6. One can be observed that hardware implementation for Eq 6.6 needs a single bit shift instead barrel shifter which gave an two benefits i.e lower hardware utilization and minimum delay. The MAC computation at y_k needs a k-bit right shift of input y_0. We have k-bit shift achieved by 1-bit shift operation at every iteration, which inherently comes with k^{th}-bit shifted input for k^{th} iteration.

$$x_{n+1} = x_n \cdot 2^{-1} \tag{5.4a}$$

$$y_{n+1} = y_n + \delta_n \cdot x_n \tag{5.4b}$$

$$w_{n+1} = w_n - \delta_n \cdot \Theta_n \tag{5.4c}$$

The arithmetic computation by the proposed hardware for 1^{st} iteration is shown in Figure 5.4. The signed dynamic fixed-point arithmetic precision have been used for the implementation and example calculation. In representation we have used MSB one bit as a signed, two bits for integer,

Variable	Initial	1st stage (for n=0)
$X_{n+1} = X_n * 2^{-1}$;	x_0 = 000.101100 (Bin) = 0.68750 (Dec)	x_1 = 000.010110 0 (X0 right shift by 1-bit; lost bit) = 000.010110 (Bin)
$Y_{n+1} = Y_n + d_n * X_n$;	y_0 = 000.001100 (Bin) = 0.18750 (Dec)	y_1 = 000.001100 + dn * 000.101100 = 000.111000 (Bin)
$W_{n+1} = W_n - d_n * 2^{-n}$;	w_0 = 000.111010 (Bin) = 0.90625 (Dec)	w_1 = 000.111010 - dn * 01.0000000 = 111111010 (Bin)

Figure 5.4: Example calculations for first stage in the pipeline architecture for 'fixed $\langle 8, 6\rangle$' and similarly for subsequent stages shown in Figure5.6

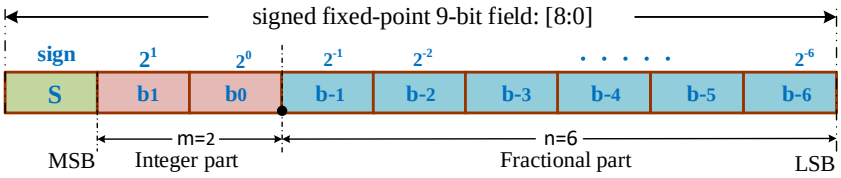


Figure 5.5: Signed dynamic 'fixed $\langle 8, 6\rangle$' precision representation with binary point used for computation in multiply-accumulate unit and system architecture

and six bits for the fractional part as shown in Figure 5.5. In further discussion, we have used a representation scheme is 'fixed $\langle 8, 6\rangle$'. In Figure 5.4, we have represented both decimal and it's binary computation. The x_0 right shift by one bit in first iteration which comes with lost of LSB of the number. Similarly, in each subsequent stage x_i right shift by one bit and lost LSB at each stage. Whereas, in this architecture y_n has not used any shift element, unlike traditional CORDIC computation used in linear mode.

Though more pipeline stages return better accuracy, it comes with more area and power overhead. Therefore, it is desirable to find the optimum pipeline stages at which a better performance parameter and significant accuracy could achieve. The design evaluates the first output after 'n' clocks, and after that, the delay between consecutive MAC outputs is just one clock cycle. In comparison, iterative CORDIC-based MAC used n-clock cycles for every output as discussed in [18]. Therefore, the proposed MAC has $n\times$ higher throughput with similar output accuracy in comparison with [18]. However, the proposed design has considerable area overhead targeting the high throughput application.

5.4 Characterization of CORDIC-based Neuron Engine Architecture

The neuron engine mainly presents the Multiply-Accumulate (MAC) computation and Activation Function (AF) computational units. Furthermore, the size of AF typically depends on the output bit-width of the MAC unit. Primarily compute logic design will have the same input and output precision, and dynamic fixed-point representation [74]. This implies $i_{b\_in} = i_{b\_out} = i_b$, $f_{b\_in} = f_{b\_out} = f_b$ and $N_{in} = N_{out} = N$, where i_b represents the integer bits excluding the sign bit, f_b represents the fractional bits and N represents the total number of bits. However, arithmetic

format must be based on target application that gives maximum accuracy with desired precision.

5.4.1 Pareto analysis, evaluation, and characterization for pipeline stages in MAC within the neuron

From Figure 5.2, it can observe that every iteration in the CORDIC computation is mutually exclusive. Therefore, pipeline stages of CORDIC have been used in MAC implementation with minimum output delay, which comes with high throughput performance. It is noticed that we may achieve significant accuracy and save hardware utilization with limited pipeline stages. Hardware has a trade-off between the number of pipeline stages and system output accuracy, and hence the performance-centric design must be determined. Therefore, Pareto points evaluation is essential to fix the necessary pipeline stages with insignificant accuracy loss within the neuron. Experimentally, we evaluated the output accuracy for different pipeline stages and observed MAC with more than five pipeline stages based on DNN accelerator returns with nearly the same accuracy. The MAC computation with five CORDIC pipeline stages is shown in Figure 5.6. The proposed pipeline architecture generates the input weighted multiplication for every clock cycle after the initial five clocks. Furthermore, one can be observed that implementation complexity is very low. The design is easily scalable for the higher precision arithmetic computational in DNN as it provides a nice balance between resource utilization and scalability.

The computation has significantly used six fractional bits as discussed in Section 5.3. However, important to notice that the 8-bit precision for the input feature-map has been used to perform the arithmetic computation. Furthermore, as mentioned, unlike iterative nature architecture [18], the proposed design evaluates the output at every clock, which comes with 5× high throughput. Conventionally, for w-bit, the input precision of MAC has 2w+n bit output precision, whereas n is the number of overhead bits depending on the accumulation operations within the MAC. Since the proposed MAC architecture has w-bit input and the same output precision, AF with w-bit precision is required within the neuron. The AF has a non-linear nature that needs huge hardware implementation resources; therefore, reduced precision AF implementation has saved significant resource utilization. Further, calculation conversion and accuracy for four pipeline stages have been verified for dynamic fixed-point arithmetic.

The architecture accuracy has been evaluated for two integer bits in fixed-point representation. However, the integer bits are user-defined. However, the number of integer and fractional bits is user-dependent. Therefore, during the network training, weight should also be limited to a specific range for better accuracy. Hence Max_Norm must be determined as a kernel and bias constraint during the training process in order to avoid overflow, following the quantization of parameters into fixed-point notation. Hence, deciding on the # of integer bits is necessary for better system accuracy in the dynamic fixed-point arithmetic computation. It is observed that 2-integer bits are sufficient for gaining the maximum accuracy. The permissible range for the weight is bounded between the -4 to +4.

Time Clock	Initial Value			Output at each n^{th} stage: $X_{n+1} = X_n * 2^{-1}$; $Y_{n+1} = Y_n + \delta_n * X_n$; $Z_{n+1} = Z_n - \delta_n * 2^n$														
	Input	Bias	Weight	Stage 1			Stage 2			Stage 3			Stage 4			Stage 5 (Final output)		
	x_i	y_i	w_i	x_1	y_1	w_1	x_2	y_2	w_2	x_3	y_3	w_3	x_4	y_4	w_4	x_5	y_5	w_5
0	000101100	000001100	000111010															
1	000101100	000001100	000000000	000010110	000111000	111111010												
2	111101100	000001100	110111010	000010110	000111000	111000000	000001011	000100010	000011010									
3	000101101	000000000	110111010	011110110	000100000	111111010	000001011	000100010	111100000	000000101	000101101	000001010						
4	000000000	000001100	110111010	000010110	111010011	111111010	001111011	000101010	000011010	000000101	000010111	111110000	000000010	000110010	000000010			
5				000000000	000001100	111111010	000001011	110111101	000011010	000111101	000100101	000001010	000001110	000100010	000000010	000000001	000110100	111111110
6							000000000	000001100	000011010	000000101	111001000	000001010	000011110	000100010	000000010	000000001	000011001	111111100
7										000000000	000001100	000001010	000000010	111001101	000000010	000001111	111011100	111111110
8													000000000	000001100	000000010	000000001	111001111	111111110
9																000000000	000001100	111111110

Figure 5.6: Computation chart at each pipeline stage for all the possible combination of ± inputs and weights. Here values are chosen from the set of inputs and respective weights at the first neuron in the first hidden layer.

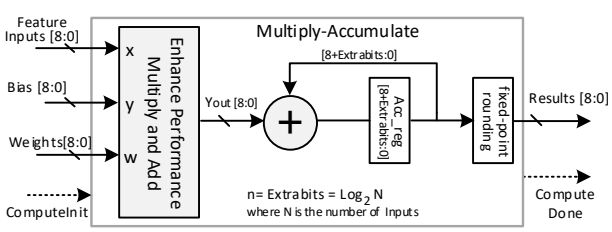
The Pareto points have eventuated bit-precision, binary point implications in dynamic fixed-point representation, and the number of pipeline stages is selected based on the experimental results. We have designed a DNN using a software platform and performed different experiments to extract the optimum design parameters. We have used the 8-bit precision to design all internal computation elements of the neural network. The data format for the 8-bit fixed-point representation with binary point implication has depicted in Figure 5.5. Regarding design parameters, the proposed enhanced performance MAC generates the output for every clock just after the five clock cycles (since we have a five pipeline stage). In continuity, the complete DNN has used the same input-output data format.

5.4.2 Neuron hardware architecture and dataflow

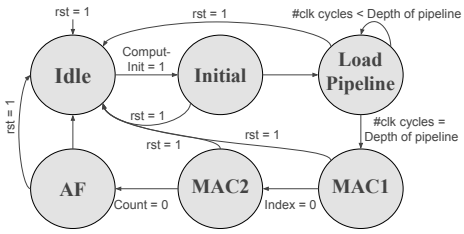
The enhanced performance MAC design computes the multiplication of input feature and weight along with bias accumulation using pipeline architecture. The proposed MAC architecture used in the neuron engine has shown in Figure 5.7. The conventional multiplication has the input precision's $2\times$ output bit width. Whereas the proposed design has n-bit input, and output precision is shown in Figure 5.6. Hence, the proposed technique has saved huge resources in the MAC implementations and subsequent arithmetic blocks such as the accumulator. The output of the multiplier gets accumulated for N times, where N is the number of inputs at the MAC unit. Therefore we have used k overhead bits (k=$log_2 N$) during the accumulation to save the overflow shown in Figure 5.7.

Each neuron in ANN consists of a MAC unit followed by AF, whereas the AF area depends on the size of the MAC output. Besides, AF such as sigmoid/tanh implementation is challenging for higher bit precision due to its non-linear nature. We have used ROM-based sigmoid AF's input and output ports as 9 bits filed addressing the efficient hardware architecture in this design. Therefore in Figure 5.7 (a), we have used fixed-point rounding at MAC output to follow the input-output precision uniformity.

We have used the in-built function call resize in HDL design. The resize function at MAC output represents the model with 'fixed $\langle 8, 6 \rangle$' at the AF input, which reduces AF's memory size and simplifies arithmetic hardware. For, high `ComputeInit`, neuron engine initiate the computation.



(a) Proposed enhance performance MAC design and Datapath for 9 -bit field 'fixed $\langle 8, 6 \rangle$'



(b) State transition diagram of Neuron computation includes proposed pipelined MAC followed by AF.

Figure 5.7: Single neuron's design architecture and state machine used in system architecture.

Once all the MAC operations in a current layer are done, the output from sigmoid activation ROM is given on output of neuron unit along with `ComputeDone` signal.

5.4.3 State-machine for controlling neuron engine

We have used pipeline architecture to enhance the performance; however, it requires a control mechanism for signal acknowledge and valid data flow. Therefore neuron engine has a state machine that ensures the loading of the pipeline as shown in Figure 5.7 (b). In `Idle` state, the entire neuron engine is on standby mode, and all the computation units are in an idle state. The `ComputeInit` signal initiates the neural computation units by changing the state to `Initial`. All the necessary control and status signals like `Index`, `Count` and the registers used in the computation pipelined proposed MAC and accumulator registers are assigned an appropriate value in this state. At the subsequent next clock cycle, the state changes to `LoadPipeline`; wherein the pipeline in the proposed MAC is loaded. In this state, the proposed MAC is loaded with values, but the accumulator is disabled. The state is to compensate for the delay in getting output due to pipelining.

After the pipeline gets loaded, the current state gets changed to `MAC1`. The accumulator is turned on, and the pipeline in the proposed MAC is loaded into the local register simultaneously. The Neuron holds in this state till all the weights and inputs are loaded into the proposed MAC. When all the weights and inputs for the current layer operation are loaded, the loading of the pipeline needs to be disabled at the next clock cycle, but the accumulation needs to be kept enabled. This is handled by `MAC2` state. The Neuron remains in this state till the entire pipeline is unloaded, an event marked by `Count` equal to zero. After which, the AF is applied in state `AF` and the output of the Neuron is provided, then the Neuron is put back in `Idle` state.

5.5 DNN with Layer Reused Architecture and Throughput Analysis

The DNN network has many layers that demand more hardware resources. However, tiny FPGAs required efficient design for implementations in Edge AI applications. The paper has presented

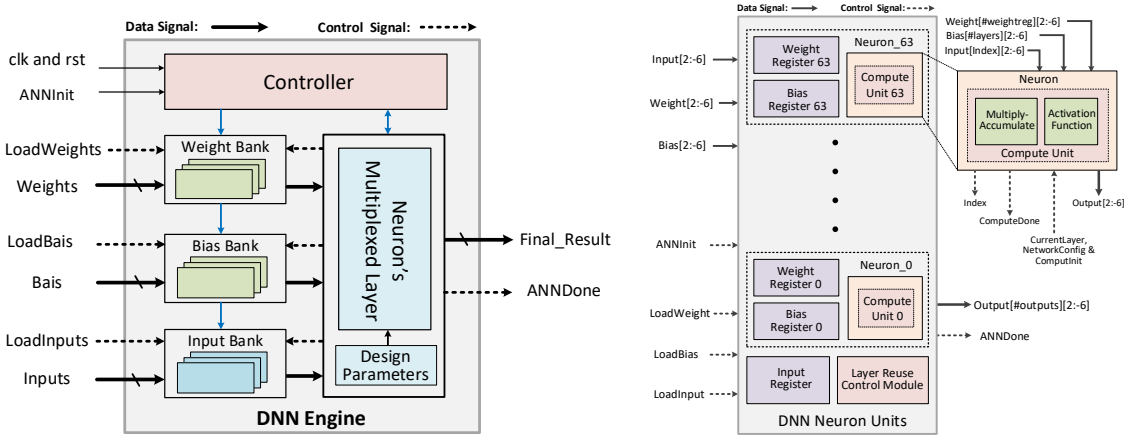
Table 5.1: Hierarchy of entities and user-defined pre-synthesis constants for compute DNN architecture

Hierarchy	Entity	Pre-synthesis constants	Value (user defined)
1	Layers	Layer configuration	196 : 64 : 32 : 32 : 10
2	Max Neurons	#Neurons - layer reuse	64
3	AF	Number of Layers	5
4	Proposed MAC	Width	9
#	Types	Integer bits	3 (include sign bit)

the hardware reused architecture to address resource-efficient design and implementation. In this proposed design, we have reused the single layer of a fully connected neural network, which realizes any desired neural network. This section has elaborated an overview of the performance-centric layer-multiplexed design system architecture. The proposed RTL design of system architecture is implementable for FPGA and ASIC. It can be configured to any size of deep neural network. The complete hardware is written using *VHDL*-hardware description language. The pre-synthesis design constants and their value for the DNN design implementation are shown in Table 5.1.

The first column shows the hierarchy of the entities used in the system design. Here, *Types.vhd* is a configuration file that contains all the pre-synthesis constants, data types, and other necessary manual functions. Thus, entity *Types.vhd* is not considered as a part of the hierarchy as it is a package. Furthermore, the input design parameters are shown in the last column. We have enhanced the performance of the neuron unit by implementing the pipelined architecture. The signed 8-bit precision fixed-point notation used for hardware implementation of the DNN have shown in Figure 5.5. The dynamic fixed-point notation used in the network is systematically selected through experimental evaluations. Moreover, by changing the pre-synthesis constants network user can be modified the DNN configuration, bit width, and fixed-point resolution. The following subsections describe a key feature of the layer-reused computational unit.

Hardware implementation of deep neural networks has faced the challenges of high resource utilization and memory bandwidth limits. Therefore, layer-reused architecture has been proposed, which is very resource-efficient but often has low throughput. Further, MAC has been implemented with minimum critical delay to enhance the throughput pipeline. In this layer, reused DNN architecture on-chip register banks have been used to avoid bandwidth limitations. The top-level system architecture of the DNN design is shown in Figure 5.8 (a). The dotted arrow indicates control signals for both input and output that ensure the correct functioning of the layer-reused DNN implementation, and the bold black arrow represents data signals. In addition, an `DNNInit` signal is used to initiate the computation, and we also have a clock (`clk`) and reset (`rst`) pins that are assigned to all computational blocks available in the engine. The proposed DNN design has `Input`, `Weight`, and `Bias` input data pins along with their control signals. The `Final_Result` is the output data pin, and `DNNDone` is a valid data pin control signal which gets high when computing the final desired output.



(a) Dynamically configurable proposed layer-reused DNN system architecture. Registers banks are used for the storage parameters.

(b) Detailed representation of all the units and sub-units of the layer-reuse DNN with CORDIC MAC

Figure 5.8: Design implementation of layer-reused performance enhance system architecture

5.5.1 DNN block architecture and system overview

The top-level system design is divided into the storage part and computation part. The different register banks have been used to store the Weights, Biases, and Inputs. The pre-synthesis parameters discussed in Table 5.1 that fixed as a DNN system design parameters shown in Figure 5.8 (a). Based on these parameters engine generates the resister banks depth during the synthesis for the weight and bias constant. These register banks are read by the neuron computation module available in Neuron multiplexed layer as and when necessary to compute the DNN. These banks are divided into different sub-banks to decrease placement and routing congestion, as explained in detail. The specific memory addressing scheme to load the parameter has been addressed in the below Section 5.6.

The FPGAs and ASICs have elevated for custom hardware implementation of DNNs. DNNs perform computations for many layers with higher area and power demands due to their parallel architecture. It generally consists of one input layer, one output layer, and many hidden layers. Each neuron consists of a MAC computational unit in each layer, followed by an AF. Thus, in an entirely similar architecture, the layer with N neurons has an N number of MACs and AFs. Furthermore, the current computing layer's output in DNN will be the input for the next subsequent layer, which produces the possibility of reusing the same logic resources in sequential order. It allowed us to reduce the hardware cost of considerable computational delay. Therefore, in an address to efficient design, the configurable layer-reused DNN is designed for minimalist hardware architecture and simplifies data movement between memory compute. We developed an efficient architecture that gives the required performance within the overall system requirement (size, weight, power, cost, etc.). Further, DNN hyperparameters vary with different applications in a fully connected neural network. Hence, architecture should be upgradable regarding the number of layers in the neural network and the number of neurons in each layer. Furthermore, the weights/bias addressing should have a viable scheme for memory allocation.

This work has implemented an efficient design of a layer-reused scheme that can support the dynamically reconfigurable architecture of DNN. The circuit design parameters are in trade-off with the depth of the neural network. This work has addressed the trade-off by proposing layer-reused architecture that can implement any deeper neural network with minimum resources where we used the single neuron's layer. The layer-reused DNN internal design architecture and all the I/O ports along with their signed 8-bit precision 'Fixed $\langle 8, 6 \rangle$' as shown in Figure 5.8 (a). In these representations, we have used a 9-bit number format representation. In this representation, 1 bit is reserved for the sign, 2 bit for the integer part, and 6 bits for the fractional part. Symbolically, this is written as 'fixed $\langle 8, 6 \rangle$'. The DNN system output is a 2D vector as the output class is for #Output Neurons (10 for MNIST). Moreover, each output is signed 'fixed $\langle 8, 6 \rangle$'.

Implementing a DNN with fully parallel layers is a very inefficient way of hardware utilization as each layer's computations are sequential. We overcome the resource limitation bottleneck by implementing a single layer that can use any number of times. The neuron-multiplexed layer's internal architecture has 64 neurons shown in Figure 5.5 (b). Here, count 64 will be the highest number of neurons in any hidden layer (user-defined). The weight and bias register banks are divided into parts to improve performance in place and route. The FPGA suffers from the memory bandwidth problem, and hence the weights and the bias register banks are broken down and clubbed along with a neural computation unit called *Neuron*. The entire network and state machine control module are designed on programmable logic (PL). The input layer engine has been provided via an I/O port.

The depth of register banks has been decided according to an algorithm that minimizes the synthesis of redundant registers. We only assign the number of registers needed for specific Neurons during computation. As mentioned network configuration, the *Neuron_63* to *Neuron_54* will be active during the computation of every layer and therefor needs a total 324 (i.e = 196 + 64 + 32 + 32) weight registers where each register is of 9-bits. Furthermore, the *Neuron_31* to *Neuron_0* are only active during the first hidden layer's computation, and hence it will require only 196 registers of 9-bit(signed + magnitude) each. Similarly, #weight registers in *Neuron_53* to *Neuron_32* will have 292 (i.e = 196 + 64 + 32) weight registers. The bias register banks are also assigned similarly.

The sub-modules MAC and AF within the Neuron are presented with a green color in Figure 5.8 (b). The proposed performance enhanced, pipelined MAC and ROM-based AF has been used in the Neuron. In ROM-based AF, the values of the sigmoid function corresponding to the input are stored. Both input and output of this ROM have signed 'fixed $\langle 8, 6 \rangle$' resolution. In the neuron module, I/O ports have an `index`, `ComputeDone`, `Currentlayer`, `ComputeInit` as a status/control signal, and the module is also provided with the pre-synthesis constants. In these I/O ports, the `Index` signal represents the count of MAC operation in the current layer computation, `CurrentLayer` keeps track of the layer computations done, and `ComputeInit` indicates the Neuron to begin calculation of a layer. Further, `ComputeDone` is the status flag indicating that computation is completed for the current layer. The input register bank and the control module for the layered reuse part have a separate blocks in the system.

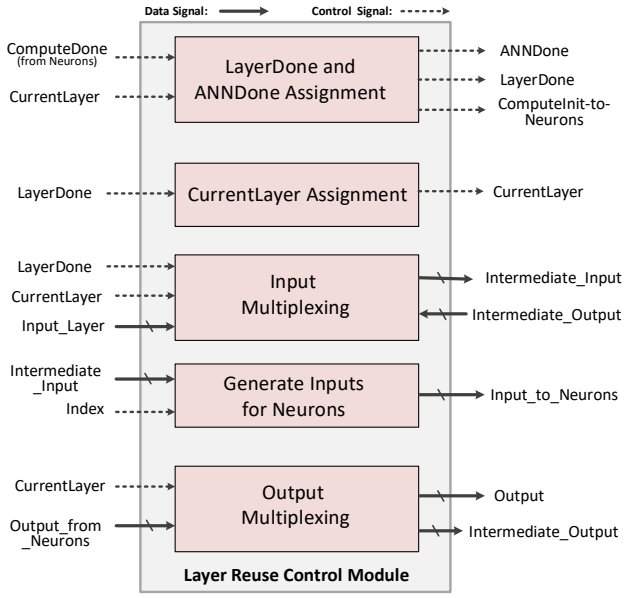


Figure 5.9: Finite state controller to efficiently reused single layer in enhance performance DNN architecture

To run a layer-reused DNN architecture, it the essential to have a control module divided into five sub-blocks shown in Figure 5.9. Each block functionally controls a status signal or generates a control signal based on the status signals. The control module has status signals, which are `LayerDone`, `DNNDone`, `CurrentLayer`, `ComputeInit`, `Index` and `ComputeDone`; out of which, `Index` and `ComputeDone` are Neuron generated, and the control module generates the rest. The `Index` is a status signal that indicates the number of MAC operations done in the current layer, responsible for telling the Index of input to be supplied for MAC operation. Further, the `ComputeDone` signal gives information about the Neuron computation that has been done in the current layer, and the valid output is ready for further subsequent computation. The `ComputeDone` of all the neural computation unit clubbed together in an array is given an alias `ComputeDoneArray`. These two signals are used to set the status signal mentioned above. These status signals control the set of data signals to operate the layer-reused architecture correctly.

5.5.2 Control engine Design for Data & Control Signals

The proposed enhanced performance layer-reused DNN architecture is dynamically configurable. For example, the generation of register banks for weights and bias constants is dynamically mapped according to the number of layers and neurons in each layer. Further, layer-reused mechanisms have performed efficiently that required signal controlling. The controller design architecture for these operations in the system architecture is shown in Figure 5.9. In this control module, each control signal has its specific applications. The `ComputInit` signal is mapped to the individual Neurons to initiate the layer computation. It is triggered by `DNNInit`, an external input signal set by the user to initiate the computation of the DNN's first hidden layer. This `ComputInit` signal is controlled based on internal status signals for the computation of further layers. For the

first hidden layer computation, all the `ComputInit` signals will be active because all 64 neurons are required to calculate the first hidden layer. In the subsequent layer computation, only the 32 Neurons must compute the layer that is made active, i.e., for computation of the second hidden layer, which is ComputeInit for 32 neurons for Neurons_63 to Neuron_32 is activated, and the rest are inactive. The scheme can be implicated in any bigger neural network. It helped save the overall dynamic power, as the pipeline register in the computational unit is disabled in an idle state. The `LayerDone` signal is turned on for one clock just after every layer's computation is done. The all Neuron units `ComputeDone` signals will be active during the current layer computation are connected to an `AND` gate; therefore, inactive neurons are not considered.

Algorithm 2: HDL code for Input Stream Generation

```
Notation– M: maximum(Layers, number_of_layers)
          N: number_of_layers
for i in N–1 downto 0 do
    if currentstate1 = compute then
        intermediate_input_to_neurons(i) <= intermediate_input – (M – 1 – index_array(i));
    else
        (others=>'0');
    end
end
```

Algorithm 3: HDL code for Input Multiplexing

```
Notation– M: maximum(Layers, number_of_layers)
          N: number_of_layers
if currentstate1 = idle or current_layer ¡0 then
    intermediate_input <= (others => (others =>'0'));
else
    if currentstate1 = compute then
        if current_layer = N–1 then
            for i in N–1 downto 0 do
                if i >= M – layers(current_layer+1) then
                    intermediate_input(i) <= input(i–M+layers(N–1));
                else
                    intermediate_input(i) <= (others =>'0');
                end
            end
        else
            for i in N–1 downto 0 do
                if i >= M – N then
                    if ComputeDonearray(i–M+N)='1' then
                        intermediate_input(i) <= intermediate_output(i-M+N);
                    else
                        intermediate_input(i) <= (others =>'0');
                    end
                else
                    intermediate_input(i) <= (others =>'0');
                end
            end
        end
    end
end
```

Furthermore, `CurrentLayer` status signal is also important that keeps track of the layer which is being computed. In this proposed benchmark design, we need to calculate the four layers, so the current layer is initialized at three, and after every `LayerDone`, the current layer count is reduced by one. After `CurrentLayer` count reaches zero, the next layer done signal triggers the `DNNDone` signal, and the entire computation module is put back into the idle mode. We use input multiplexer

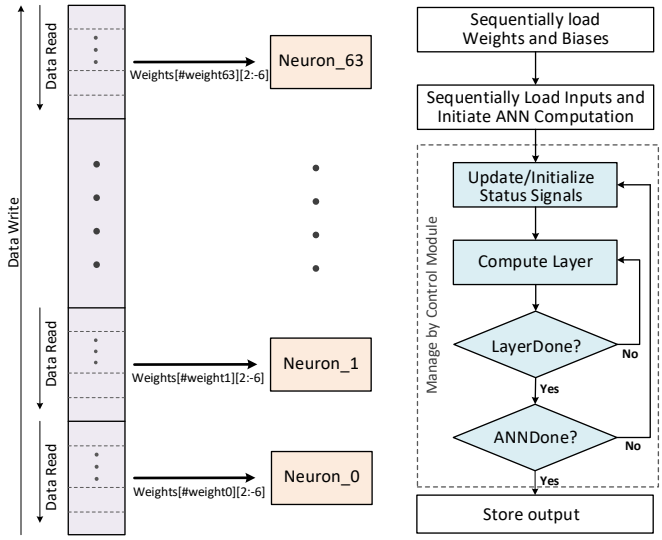
Algorithm 4: HDL code for Output Multiplexing

```
Notation– M: maximum(Layers, number_of_layers)
          N: number_of_layers
process (CurrentState1, current_layer)
begin
if currentstate1 = idle then
  Outputs <= (others =>(others =>'0'));;
else
  if currentstate1 = compute then
    if current_layer = –1 then
      for i in N–1 downto 0 do
        if ComputeDonearray(i) = 1 then
          if i > N – layers(0)-1 then
            Outputs(i–N+layers(0))<= intermediate_output(i);
          end
        end
      end
    end
  end
end
```

and output de-multiplexer for muxing the intermediate signals. Input to multiplexer are the `Input_Layer` and `Intermediate_Output` where output of this multiplexer is `Intermediate_Input`. The control signals generated through the layer reused control model is shown in Algorithm 2. The sequential stream of input signals sent to Neurons from the `Intermediate_Input` based on the `Index`; a status signal generated by the neurons. The detailed working principle is shown in Algorithm 3. We have output De-multiplexer is to get the output from the Neurons after the layer computation is done. The decoder is used for current layer value whether to send Output from Neurons to `Output_Layer` or `Intermediate_Output`. If the Current Layer count is 0, i.e., the layer computed is the output layer, then the Output from Neurons is directed to `Output_Layer`. Else, it is directed to `Intermediate_Output` as shown in Algorithm 4.

5.6 Input Data Loading Scheme and DNN System Processing

We mentioned in the previous section that the weight bank is broken into 64 groups and paired with the corresponding neuron computation unit is illustrated in Figure 5.10 (a). Although self-explanatory, we need to understand one important thing in this flow: memory addressing for loading the model's parameters. It can be observed that the write and read orders are reversed. The data flow scheme is beneficial for accessing the weights/bias constants with minimum routing delay. So, *LIFO* (*L*ast *I*n *F*irst *O*ut) scheme have to be followed while loading the weights, and similarly for bias and input loading. The I/O protocol for loading and reading is followed a standard synchronous communication protocol with a valid data pin which in design is `load_param_weight` for loading weights. In the case of valid output, we have a data-ready pin which in design is `DNNDone`. In the case of input, the value gets loaded at every clock when the valid pin gets high. For output, we get the output for all ten neurons in parallel when the `DNNDone` signal gets high that needs to be stored on the software side. The software flow for DNN accelerator implementation as



(a) Data read/write order (b) Operation flowchart

Figure 5.10: DNN engine working flow and order to initialized the loading data

shown in Figure 5.10 (b).

5.7 Experimental Results and Discussion

This section has discussed the hardware performance and implementation results for complete layer-reused system architecture and enhanced CORDIC engine-based MAC unit performance. The efficient layer-reused DNN configuration 196 : 64 : 32 : 32 : 10 with enhanced performance MAC unit is designed using a hardware description language. The accuracy performance of the proposed design has been validated using Python with the standard TensorFlow computation [5] for MNIST. Furthermore, we have simulated fixed-point behavior by quantizing all the operations and activation. Hence, our CORDIC-based python implementation[1] is framework-independent and faithfully replicates the hardware design to evaluate accuracy. The MNIST input image 28×28 is resized to 14×14 that increased our DNN performance, and the model has trained for signed 8-bit precision fixed-point, i.e., `[8:0]` arithmetic and noted the inference accuracy. The extracted weights/bias is used to verify the hardware implementation. The Virtex-7 VC707 FPGA has targeted DNN implementation and evaluated results. The following experimental evaluation is performed for design validation and extracted results.

5.7.1 Neurons Pareto Analysis for bit-precision, #integer bits, and pipeline-stages

In order to evaluate the arithmetic bit precision impact on accuracy for the deep neural network, we experimentally verified. The accuracy has been calculated for 8-, 12-, and 16-bit precision fixed-point exact computation arithmetic for MNIST dataset on benchmark network 196 : 64 : 32 : 32 : 10

[1] get the accuracy-test code

where observed accuracy is 95.41%, 96.70%, and 96.80% respectively. At the cost of 1.4% accuracy loss between 8-bit and 16-bit fixed-point arithmetic, we can save the memory bandwidth by a factor of 2.

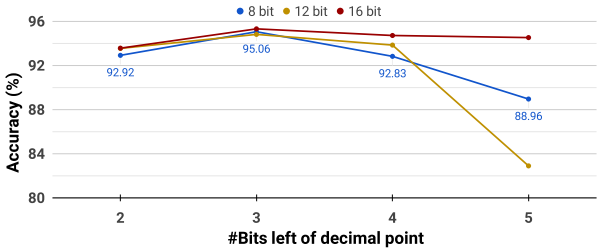
As we saw, recursive CORDIC architecture performs the computation iteratively, and every iteration is mutually exclusive. We leveraged this exclusivity to develop the proposed enhanced CORDIC-based computational engine (multiply-add-accumulate). The compute iteration in the CORDIC algorithm is equal to the number of stages in the proposed MAC design. The neural network is error-resilient and has the scope for computational approximation, i.e., we can limit the number of iterations at which tolerable accuracy has occurred. The proposed CORDIC-based neuron engine contains three parameters that need to optimize with hardware architecture and determine Pareto points against the test accuracy. These constants are the bit precision(bit-width) of the parameters, the number of bits left of the decimal point, and the number of pipeline stages by which design will make resource-efficient and performance-enhancing design.

In order to evaluate Pareto points for determining the number of pipeline stages in the MAC computation, we have used max norm kernel constraint and bias constraint, which bounds the l_2 norm of the parameters. We also need to optimize the parameter of this constraint, i.e., Max_Norm. Now, from Figure 5.11 (a) and (b), and 5.12 we can observe that with Max_Norm = 5.5 the #integer bits = 3 , #pipeline stages = 5 and bit-precision of 8-bit fixed-point arithmetic produces 95.06% accuracy[1]. Producing a vaguely better accuracy is possible by increasing the number of pipeline stages in the proposed enhanced MAC. However, the increase in utilization for accuracy is too significant for the minuscule increase in accuracy after the number of pipeline stages = 5 in the proposed MAC. Hence, the Pareto points are fixed at Max_Norm = 5.5 the #integer bits = 3 , #pipeline stages = 5 and bit-precision = 9 (1 sign bit and 8 magnitude bits).

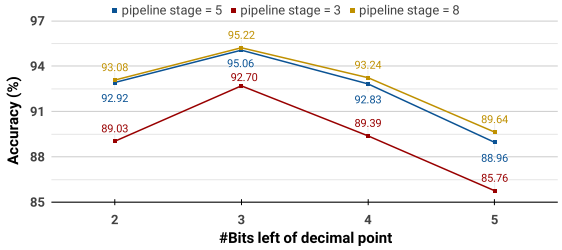
5.7.2 MAC performance evaluation for previously determined Pareto points and comparison

To enhance the performance of MAC, we optimized the architecture by introducing the pipeline concept. The number of pipeline stages will directly impact resource utilization. Hence we analyzed it through the experimental evaluation and the number of pipeline stages, which are *five*-a pipelined stage we used for performance parameters evaluation. Based on observed Pareto points, proposed enhanced performance MAC have been implemented, and results are extracted for comparison with the state-of-the-art designs. In this experiment, first, we compared the performance parameters for signed 8-bit precision with the state-of-the-art designs as shown in Table 5.2. In this table, we reported the resources utilization, circuit critical delay per clock, and Power-Delay-Product (PDP). In this table, we used to signal and logic power for PDP calculation. It can be observed that the proposed design has 2.2× lower PDP compared to the best design in the state-of-the-art.

Secondly, we analyzed the impact of bit precision on hardware parameters. We compared our MAC implementation results with the multiply and accumulation IP from Xilinx [7] shown in



(a) Analysing the effect of number of integer bits for different bit-precision, at Max_Norm = 5.5 for proposed enhanced MAC.



(b) Accuracy vs number of bits left of decimal point for 8,12,16 bit-precision, and Max_Norm = 5.5 for proposed enhanced MAC

Figure 5.11: Pareto point evaluation for hardware design and implementation of System architecture

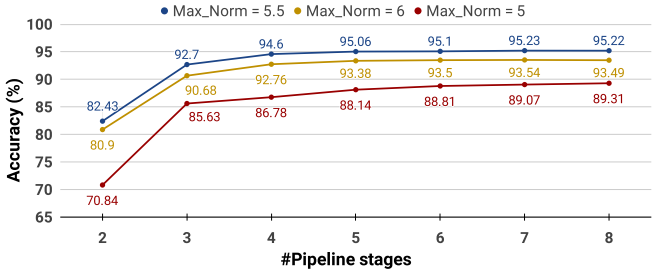


Figure 5.12: Analysing the effect of number of pipeline stages on accuracy for different Max_Norm for proposed enhanced MAC.

Table 5.3. Here, we compared the results for different signed dynamic fixed-point representations where we used 6-, 9-, 13-, and 17-bit, i.e., for the 5-, 8-, 12-, and 16-bit precision, respectively. It can see that one extra bit is used as a signed bit in the actual computation, as we discussed in Section 5.4. As the proposed MAC used five pipeline stages, the possible minimum bit precision is five magnitude bits. It can be observed that the proposed design has very few increments in resource utilization for higher precision. The rate of increase in FF utilization is almost similar for both MACs. However, as the proposed MAC is pipelined structure, it has higher utilization of FF, but the difference increases more or less linearly.

The proposed MAC and Xilinx MAC IP share almost the same latency for every bit of precision. For 8-bit, our design saved relatively save 65.4% LUT resources utilization, whereas for 12- and 16-bit saved 69.3% and 74.3%. Further, it is also observed that lower *Power-Delay-Product* (PDP), which is for 8-bit precision, is 2.51× whereas for 12-bit and 16-bit precision is 2.73× and 5.11× respectively. The results are concluded that the proposed architecture more benefits from the higher precision computation.

Table 5.2: Resource utilization and performance parameters comparison of MAC with state-of-the-art technique at 'fixed $\langle 8, 6 \rangle$' precision

Resources Utilization	Slice LUTs	Slice Registers	Critical Path Delay (ns)	Power-delay Product (pJ)
Vedic [75]	159	245	4.48	5.86
IEEE [7]	130	45	3.98	5.01
Wallace [77]	105	112	2.59	3.13
Booth [76]	83	61	3.08	2.77
Shift-add [65]	75	58	5.44	3.97
Acc_App [97]	62	59	2.87	2.04
Proposed	54	88	1.52	0.91

Table 5.3: Resource utilization and performance parameters for MAC architecture at 'fixed $\langle 8, 6 \rangle$' precision.

Bit-precision signed	MAC Design	Slice LUTs	Slice Registers	DSP Util.	Delay (ns)		Power (mW)			PDP (pJ)
					Logic	Signal	Logic	Signal	Dynamic	
16-bit	Xilinx IP [7]	369	76	–	2.268	6.783	1.95	1.95	13	117.7
	Proposed	95	162	–	0.805	1.319	0.44	0.77	11	23.7
12-bit	Xilinx IP [7]	244	60		0.966	3.485	1.26	1.26	9	40.1
	Proposed	75	126	–	0.731	1.108	0.32	0.48	8	14.7
8-bit	Xilinx IP [7]	130	44	–	0.921	2.895	0.66	0.60	6	22.9
	Proposed	54	88	–	0.755	0.768	0.24	0.36	6	9.1
5-bit	Xilinx IP [7]	53	28	–	0.813	2.277	0.27	0.21	3	9.37
	Proposed	35	58	-	0.713	0.692	0.16	0.20	4	5.62

5.7.3 Hardware parameters evaluation for proposed and state-of-the-art DNN design techniques

The fully parallel deep neural network has high throughput but has area overhead which is further accountable for the higher power consumption. In every neuron, the output bit-width of the MAC unit depends on the number of accumulations in the neuron. Further, MAC output will be the input for the AF inside the neuron. The resources utilization (memory elements) by the AF has increased exponentially with the bit precision, i.e., 2^N factor, where N is the AF bit-precision. Hence, bit quantization is immensely required, especially at the MAC unit's output, making the resource-efficient design. The implementation results for different design techniques i.e fully parallel [101], layer-reused [25] and our layer reused with bit-width quantization technique is shown in Table 5.4. We used Xilinx multiplier and adder IP for the sack of comparison at the architecture level and observed the physical hardware parameters.

The deep neural network parallel computing demands high memory bandwidth. In contrast, FPGA suffers from memory bandwidth issues. Therefore, the proposed design has used the on-chip register banks for weight/bias constants in the proposed architecture. It can be observed from Figure 5.4 that the proposed design has saved 30% LUTs and 48% FFs relative to the fully parallel layer architecture in the benchmark network with acceptable throughput loss. Furthermore, as our layer-reused DNN architecture utilized fewer hardware resources, it has saved the 22.7% on-chip dynamic power. Further, it can be observed that no quantization at MAC output exceeds the BRAM utilization (>2K) as it needs 2^N memory elements for the AF implementation [25]. The

Table 5.4: Hardware utilization report for fully parallel and layer-multiplexed architecture with Xilinx IP [7] based multiply-accumulate unit and our enhance performance MAC

Hardware Parameters		**Fully parallel Archi. [101]**	**DNN Layer -Reused [25]**	**Proposed Layer -Reused**
Resources Utilization	Slice LUTs	207908	145474	144283
	Slice Reg.	306874	156117	155811
	Slices	-	-	61966
	BRAM	53	exceed (>2K)	23
On-chip Power (W)	Logic	0.914	0.241	0.133
	Signal	0.956	0.317	0.235
	Dynamic	1.933	0.622	0.481
	Static	0.261	0.252	0.248

proposed design has utilized only 23 BRAM for the complete system design, where we used a bit-rounding scheme at the MAC output.

An optimized layer-reused architecture is efficient for area and power but has lower throughput. Hence, this work investigates and proposes an efficient MAC with a lower area and power utilization. We implement the CORDIC-based MAC design. However, it suffers from lower throughput, which enhances in this article using the pipeline stage of the CORDIC engine. Further, this article's CORDIC design has been optimized for lower area utilization and critical logic delay. The number of pipeline stages will increase the computational accuracy but come up with the area overhead. Hence we have characterized the neuron architecture for different design parameters. As discussed in Section 5.7.1 and hardware utilization for the proposed model and fully parallel neural network using Native MAC [7] is shown in Table 5.5. Here we examine the utilization impact of the proposed MAC in DNN system architecture.

The reported benchmark network we use for the results extraction. It is observed that there is an almost 42.92% decrease in resource utilization and a 62% reduction in FF utilization. The improvement in LUT utilization is the overall decreased consumption of LUTs per MAC unit, as we previously observed. The staggering decrease in FF results from Xilinx Vivado's internal optimization because we are using a 5-stage computation pipeline, so Vivado optimized redundancies in parameter storage, decreasing FF utilization. However, the accuracy remained almost unaffected (0.35% decrease in accuracy for MNIST classification). The BRAM usage can be explained using a similar argument that when a native MAC was used, Vivado optimized the design because of the use of native MAC. When analyzing a neural unit in standalone mode, it is observed that a single neural unit contains a 512 deep sigmoid ROM. Each AF has 0.5 BRAM utilization which is entirely used for the activation function. In line with this observation, in the case of design with proposed MAC where Vivado performed no optimizations, BRAM utilization was 69, exactly half of the number of neural units.

Table 5.5: Hardware implementation results for fully connected neural network. The layer-wised resources utilization compared with Xilinx-IP [7] based DNN design

Number of Neurons in each layer	Utilization Report for DNN (196:64:32:32:10)							
	Design with Native MAC [7]				Design with proposed MAC			
Resources	Slice LUTs	Slice Reg	Slices	BRAM	Slice LUTs	Slice Reg	Slices	BRAM
1st hidden layer (64)	153951	235462	-	32	79194	78229	34311	32
2nd hidden layer (32)	29705	40911	-	9	19834	17539	8015	16
3rd hidden layer (32)	18309	21684	-	9	14827	12081	5399	16
Output layer (10)	5717	7018	-	3	4606	4017	1802	5
Total DNN Utilization	207908	306874	-	53	118684	113665	49547	69
Total Utilization (%)	68.48	50.54	-	5.15	39.09	18.72	65.28	6.70

Table 5.6: Hardware implementation report for 'fixed $\langle 8, 6 \rangle$' benchmark network with proposed DNN architecture and state-of-the-art.

Hardware Parameters	**Fully Parallel DNN**	**Layer Reused with Quantization**	**AF Reused in layer**	**Multiplexed Proposed**	**% Util. save Relative**
Resources Utilization Report for 6-bit LUT Size					
Slice LUTs	207908	144283	208608	112654	45.81
Slice Reg.	306874	155811	308374	113648	62.96
Slices	-	61966	-	45439	-
BRAM	53	23	3	32	39.62
Network's On-chip power utilization report (W)					
Logic	0.914	0.256	0.843	0.133	85.44
Signal	0.956	0.585	0.911	0.235	75.41
Dynamic	1.933	1.020	1.792	0.481	72.85
Static	0.261	0.252	0.58	0.248	04.98

5.7.4 Comparison for power estimation, performance evaluation and accuracy of proposed DNN accelerator

This subsection examines the benefits of using the proposed MAC unit with optimization techniques like layer reuse and bit-rounding. The utilization results after implementation with two techniques for layers multiplexed designs, one with proposed MAC and another with native MAC [25], are given in Table 6.5.2. We can observe a 21.91% decrease in LUT utilization for the same reasons as fully parallel DNN. FF utilization was a 27.04% decrease in FF utilization and a 26.67% improvement in overall slice utilization. The BRAM usage is observed as 32 (half of 64, i.e., the number of neural units). The BRAM utilization in the case of native MAC-based design is 23, resulting from optimization from Vivado. We can now clearly see the benefit of a proposed MAC-based layer-reused DNN implementation. The layer reused technique reduces resource utilization and has a few more benefits.

We enhance the throughput of the Layer reused DNN by implementing the pipelined CORDIC stages in the proposed MAC. Moreover, the total critical delay of the proposed design has almost 2.6× less as compared to Native MAC [7] and 1.8× of the best design from the state-of-the-art, which implies that the proposed design has achieved 2.6× throughput compared to Native MAC [7]. Table 5.7 shows implementation results in terms of other performance parameters. Firstly, we have

Table 5.7: On-chip power summary and other performance parameters for DNN implementation with Xilinx IP MAC and proposed MAC @50MHz clock frequency for 'Fixed $\langle 8, 6 \rangle$'

Parameter		Fully Connected Architecture		Proposed Architecture	
		Xilinx IP [7]	**Proposed MAC**	**Xilinx IP [7]**	**Proposed MAC**
Power (W)	**logic**	0.914	0.263	0.256	0.133
	Signal	0.956	0.417	0.585	0.235
	Dynamic	1.933	0.879	1.020	0.481
	Static	0.261	0.254	0.252	0.248
Computation Time (in #clock cycles)		201	205	344	360
Throughput (GOPS)		4.65	4.45	4.65	4.45
Performance (GOPs/W)		2.41	5.06	9.17	9.25
Accuracy (%)		95.41	95.06	95.41	95.06

power. Note that the power report for Fully Connected DNN with conventional MAC is generated from synthesis. We can see the benefit of both the Layer reused technique and the proposed MAC on logic power. We see a 49.43% reduction in logic power by introducing Layer multiplexing in designs with the proposed MAC. Similarly, in Layer multiplexed architectures, we can see a 15.04% decrease in logic power by replacing native MAC with proposed MAC in Layer multiplexed design. Similarly, signal power also shows 43.65% benefit and 13.92% benefit correspondingly. However, this minor increase in logic power can be attributed to more flops in MAC architecture, which is compensated by the signal power. This leads to a total dynamic power saving of 45.28% decrease in dynamic power consumption in Fully Connected vs. Layer multiplexed designs with proposed MAC and 5.13% benefit in total dynamic power. The static power is more or less the same for all designs as it is seen to design independently and dependent on the FPGA. The previous results show that the proposed design is very efficient for area and power. Moreover, the other performance parameters are also addressed.

The performance parameters of DNN implementation with different technique is shown in Table 5.7. The parameters are compared for the proposed enhanced MAC-based design and MAC design using Xilinx multiplication and addition IP [7]. The throughput we calculated by applying the equal input clock frequency, i.e., at 50MHz, to all the designs. It can be observed that the proposed enhanced performance pipeline architecture of MAC takes four more clock cycles in a state-of-the-art layer-multiplexed design. Therefore, 16 clock cycles more than the four Layer conventional fully connected neural network. It is caused by the additional clock cycles required to load and unload the pipeline in the proposed MAC unit for every Layer. For Depth of pipeline = 5, we need four additional clock cycles per Layer, leading to a net of 16 extra clock cycles for computation. However, the throughput loss is not significant and can easily be justified by reducing resource utilization. Moreover, for signed 8-bit precision after observing from Table 5.2, the critical path delay in the DNN implementation with the proposed design architecture can operate at 1.8× higher frequency compared to the best one in the state-of-the-art.

The performance throughput per watt for the proposed design is higher. The proposed design

achieved all the benefits at the cost of 0.35% accuracy loss we observed for the MNIST dataset. Both designs consume 344 clock cycles to complete the computation. In these 344 cycles, the proposed DNN design architecture computes 15,963 Multiplication operations, 15,963 addition operations, and 138 AF operations at 50MHz, giving about 4.65 GOPs. Based on these values, the proposed design performs nearly 1.89× better than a fully connected deep neural network in terms of Throughput/W.

5.8 Summary

This paper proposes and implements efficient techniques for layer-multiplexed ANN engines for reduced hardware resources and better performance. The conventional multiplexed technique sufferers from low throughput, which address by proposing efficient enhanced performance neuron architecture. The MAC in neurons is implemented using pipeline CORDIC stages which enhance the throughput, and CORDIC architecture is optimized to make it area-efficient with lesser critical delay. The ANN engine architecture is written in HDL language, and it is implementable on both ASIC and FPGA. The user can implement any deeper ANN for object classification by fixing the user-defined parameters. A detailed analysis has been given for selecting circuit design parameters, which can adapt to other architecture for the efficient design of the DNN engine.

The proposed performance enhancement technique in system architecture opens new opportunities for low area and power with enhanced performance design, mainly in the Edge-AI applications. The Virtex-7 FPGA platform is selected to implement the benchmark configuration for MNIST classification and verify the proposed architecture's functioning. The experimental evaluation validates that the proposed enhanced performance layer-multiplexed solution will help us gain better area, power, and throughput performance for future system implementations. After acceptance, we will make our project code open-source to assist the reproducible results and hardware implementation, i.e., for ASIC and FPGA.

Chapter 6

FPGA Implementation of Novel Data Multiplexed DNN Accelerator

Despite many decades of research on high-performance Deep Neural Network (DNN) accelerators, their massive computational demand still requires resource-efficient, optimized, and parallel architecture for computational acceleration. In Chapter 5, layer reused architecture comes with huge hardware resources saving at the cost of throughput loss which is further resolved using pipeline MAC. In this chapter, we have reused the hardware of AF within the layer at the cost of no throughput loss. In DNN, every neuron has an activation function which takes huge hardware resources due to its non-linear nature. We propose DNN with reused hardware-costly AF by multiplexing data using shift-register. The on-chip quantized log_2 based memory addressing with an optimized technique is used to access input features, weights, and biases. This way, the external memory bandwidth requirement is reduced and dynamically adjusted for DNNs.

6.1 Introduction

The Deep Neural Networks (DNNs) have become a popular algorithm in pattern recognition since the recent evolution of computer hardware, which fulfilled the high computational power requirement of learning algorithms [119, 120]. The main advantage of the DNN over other prediction techniques is its capability to learn hidden relationships in data with unequal variability [121]. Further, DNN is a popular choice due to its diverse application and is applied to various non-linear detection problems, some of which are lane detection, pattern recognition, fault detection, and monitoring in the industry [8, 9]. However, these applications often require a DNN and real-time processing, which need high computational power.

Deep learning opens up a need for different platforms like GPU, CPU, ASIC, or FPGA to ac-

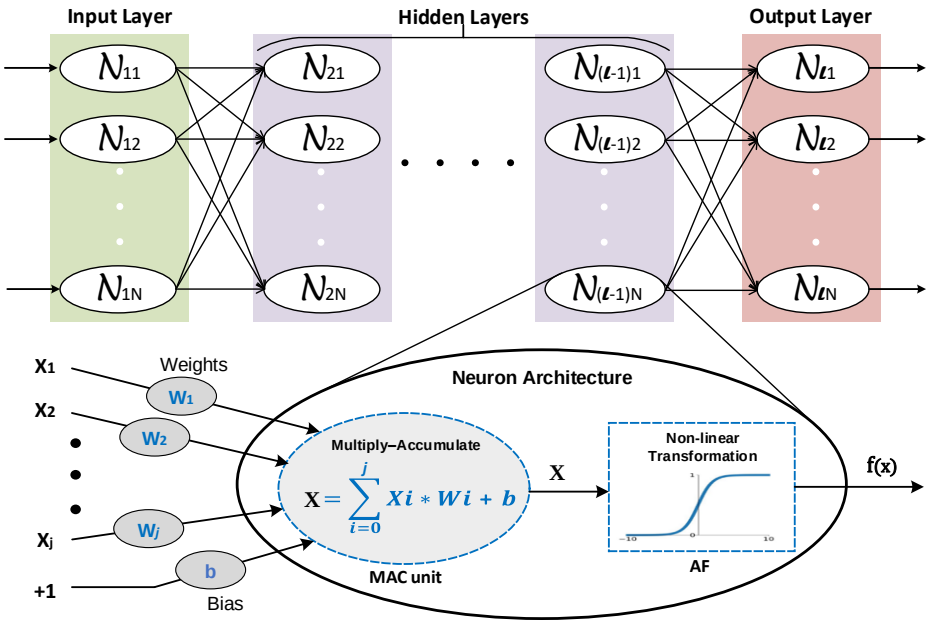


Figure 6.1: Fully parallel feed-forward Deep Neural Network architecture

celerate the computation of the DNN algorithm. Here, we refer to a fully connected feed-forward Artificial Neural Network (ANN) with multiple layers between the input and output layers as a DNN. The CPU and GPU-based DNN implementations are general-purpose platforms with specialized hardware supporting various operations, including Multiply-Accumulate (MAC) operation. However, the drawback of CPU and GPU is the low utilization of their resources that reflects in high power consumption [10]. In comparison, ASIC and FPGA devices provide fast multiplication operations with minimal resources utilization and lower power consumption than streaming pixels and learning features [11]. Further, FPGA also has a specialized hardware structure for MAC operation, but their design is customized to DNN implementation; thus, they achieve high resource utilization and lower power consumption. At the same time, ASIC-based hardware accelerators are the fastest and most energy-efficient. However, they are constrained by their inability to reconfigure neural networks with different features and the inferred network's limited size [12]. Attractive FPGA implementation schemes, focusing on the usage of Xilinx families, are described in the book edited by Ormandi and Rajapakse [122]. Besides, FPGA has hardware efficiency, resources utilization flexibility, and reconfigurable logic design architecture to optimize performance for any specific type of application [16].

In DNN, each neuron performs two basic functions: sums weighted input features using MAC and evaluates Activation Function (AF) from calculated sum as shown in Figure 6.1. DNNs are computationally expensive for data-intensive applications that may contain many neurons and several parameters. In a fully parallel implementation of the deep neural network, each layer needs a distinct memory space for its weight and bias in order to maximize performance efficiency. Previous works have rarely investigated flexible hardware architecture that scales well with the size of a neural network without compromising accuracy. The area-efficient architecture is made using different design techniques for MAC unit, AF, and layer architecture [12, 16, 17, 18, 19].

Moreover, the implementation of the AF is challenging due to its non-linear nature. Therefore for its accurate design and implementation it requires a large number of hardware resources. However, an area-efficient design can be made by configuring many arithmetic operations implementation such as MAC and AF on the same hardware at the cost of lower throughput [18].

Despite the configurability, the design must scale in terms of hardware due to a profound DNN realization. Our goal is to develop an area-efficient architecture of a DNN that can reuse the AF within the DNN to make it suitable for implementation on small FPGAs. Considering the demands on trade-off in area, power, and speed of the computational block, we propose a data-multiplexed and hardware-reused DNN architecture with an efficient way of memory mapping. Further, compute efficient AF is optimized using Taylor series expansion, and we reuse the AF within each layer by multiplexing and serializing the data-flow using the Parallel-in Serial-out (PISO) mechanism.

Contribution

The proposed work focuses on efficient architecture in terms of area and power overhead without performance loss. The contribution of the FPGA based DNN architecture is summarized in the following three points:

- In order to compute a more popular sigmoid AF, we implement its computation through a memory element that uses a positive exponential function with Taylor series expansion and uses BRAM utilization to store the memory element that benefits at higher precision. The AF model physical parameters are also evaluated at the 45 nm technology node.
- In order to find the Pareto point in the Taylor series order expansion for sigmoid AF, accuracy is analyzed. Another, accuracy versus bit precision, is analyzed for different AF models.
- The performance-centric, resources-efficient fully connected DNN is presented that reuses AF by multiplexing the data-flow. The throughput and other physical parameters impact are analyzed. Further, we presented efficient memory addressing scheme for reading/writing of weights and biases.

This work confirms that DNN architecture with data multiplexing that reuses hardware resources improves area utilization and decreases energy consumption. Furthermore, the authors have verified the proposed architecture by implementing the `16:16:10:4` Deep Neural Network on Zybo xc7z010clg400 and evaluating its performance parameters. The performance validation has been done using the 16 level thermometer to four binary-level classification applications.

6.2 State-of-the-art Hardware Optimization Techniques

Conventionally a DNN can either be efficient in terms of area/power or accuracy/throughput. Therefore, research has been done primarily on computational arithmetic blocks such as MAC,

AF, and neuron layer interconnection within the network to optimize neural network acceleration performance. DNNs contain one input layer, multiple hidden layers, and one output layer.

6.2.1 Related work for DNNs implementation and MAC optimization

The mobile devices with resource-constrained hardware for DNN deployment are attracting much attention. Further, the success of AI drives Application-Specific Integrated Circuit (ASIC) design for the area and performance-efficient DNN implementation [102]. As a result, the FPGA and ASICs have elevated for custom hardware implementation of DNNs. DNNs perform computations for many layers, which come with higher area and power demands [103], and hence the area and power constraint in DNNs represent an increasingly major challenge, especially in mobile devices and AI-enabled IoT applications. Significant research has scrutinized the efficient design architectures at different abstraction layers to enhance the performance of DNNs [123]. An ASIC design as Tensor Processing Unit (TPU) [104] has been implemented for on-chip acceleration. An ASIC/FPGA-based DNN framework such as Eyeriss [105], Gemmini [106], Simba [107] have been proposed in the state of the art. However, the reused hardware resources with no throughput loss have not been addressed. Therefore, explorations of hardware efficiency, such as hardware reuse, have prompted more attention to reduce area utilization [110].

The fully connected neural network with reconfigurable nodes within the layer has been designed in [24]. Furthermore, the configurable layer reused DNN is designed for minimalist hardware architecture and simplifies data movement between memory compute [12]. In [124, 125], authors have implemented a binarized neural network which drastically cut down the hardware consumption at the cost of insignificant accuracy loss. Consequently, a significant amount of research is focused on the lower area utilization and power-efficient hardware-based neural networks while compromised with the throughput and accuracy performance..

DNNs with a higher number of layers have been studied in the last decade to improve DNN accuracy. Many architectures have been proposed in the state-of-the-art FPGA-based neural network implementations [114, 45, 39, 126]. Such architectures substantially increase the number of neurons that increase the MAC utilization, increasing the hardware resources demand, processing time, and power consumption. Further, previous works have investigated the efficient architecture for a MAC unit that can use minimum resources without performance loss [12, 17]. In [127], the authors overcome this limitation by using dynamic partial reconfiguration to reuse the FPGA resources. While this technique does improve the resource utilization of the FPGA device, it significantly increases the delay of the circuit. Another way to cope with this increased resource utilization is to use approximate computing techniques to implement MAC [128, 129] if some degree of error is tolerable.

The application-specific reconfigurable DNNs have been implemented in [130]. The number of neurons in each hidden layer can be configured within the DNN through which on-chip power has been significantly saved. However, design can further explore hardware reused architecture. An area-efficient method using layer multiplexing has been addressed in which physical implementa-

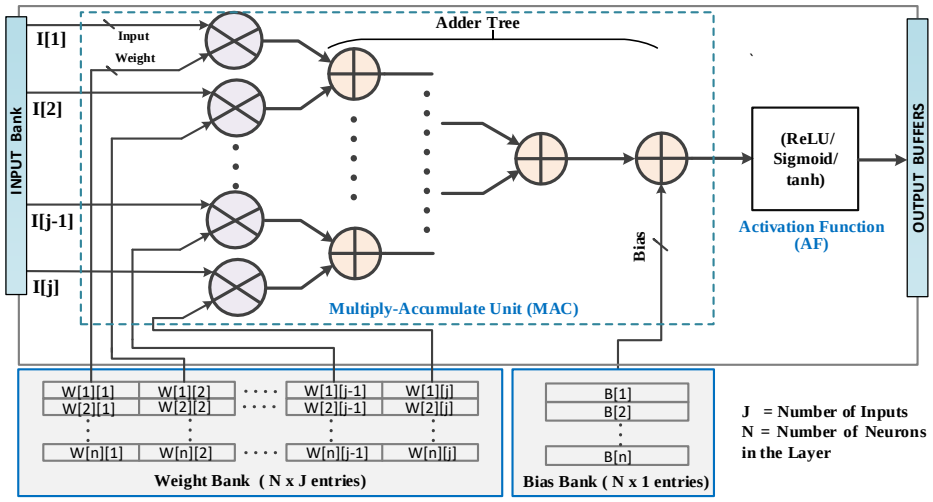


Figure 6.2: The memory mapped neuron computational element design having parallel multiplier for J inputs, adder tree followed by AF. Considering N be the number of neurons in corresponding l^{th} Layers.

tion of the number of hidden layers is halved [19]. Although network implementation has lower area utilization, design suffers from lower throughput, halved compared to the fully parallel layer architecture. The resource-hungry non-linear AF has been reused by implementing the multiplexers between the parallel MAC units and the AF within the layer [91]. This kind of architecture is efficient for very low precision, and tiny neural networks as multiplexers are hardware costly for higher precision and increase the complexity in the data-flow, which comes with a higher critical delay. However, we have presented an efficient hardware design that reuses the AF with an insignificant throughput performance loss in DNN, and the design technique can be efficient for all fixed-point arithmetic precision implementation. Furthermore, our design architecture is adequate in the hardware design of DNN with runtime configurable for many AFs [25].

The high-throughput MAC unit with a fully parallel array of multipliers is shown in Figure 6.2. Usually, the bit-width of MAC output is the sum of input bits, trained weight bits, and extra overhead bits to save the overflow bits generated during the accumulation. The performance-efficient state-of-the-art neuron computational unit having a MAC unit with a parallel multiplier and adder tree followed by AF is shown in Figure 6.2. However, it is impossible to instantiate parallel multipliers in MACs for every neuron due to the limited hardware resources. The hardware-costly multiply and accumulate unit can be shared by the multiple consecutive hidden layers at the cost of throughput loss [131, 101]. Furthermore, dedicated AF in each neuron is accountable for area overhead due to its underlying non-linear nature. Primarily MAC unit consists of multiplier and accumulator block, whereas arithmetic relation between input and output of a n^{th} neuron in the l^{th} layer is given by Equation (6.1),

$$\mathbf{a}_n^l = f(\sum_{j=1}^{J} \mathbf{W}_{n,j}^l \cdot \ {\mathbf{a}_j}^{l-1} + \mathbf{b}_n^l) \tag{6.1}$$

where f is the Activation Function (e.g. *simoid*, *tanh*, *ReLU*), $w_{n,j}^l$ is the weight of the j^{th} input

a_j^{l-1}, and b_n^l is a bias of the n^{th} neuron. Note that the output of the layer $(l-1)^{\text{th}}$ is the input to the layer l. Computation of a whole layer formulated in matrix form is given in Equation (6.2).

$$\mathbf{a}^l = f(\mathbf{W}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l) \tag{6.2}$$

The Deep Neural Networks have more (several) layers, and more MACs are a consequence that demands more hardware resources for implementation. This problem will be more dominant when the network processes the high-resolution images that need higher precision computational elements. Therefore, DNN implementation on resource-constrained hardware mobile or edge devices is attracting much attention. In previous works, multi-bit precision (8, 16, 32, and 64-bit) data representation is used for MAC unit arithmetic computation, considering the trade-off between accuracy and physical performance parameters. Furthermore, compared to the floating-point, the fixed-point is preferred which has maximized utilization density and throughput. However, it also prefers when the resources are limited, and some degree of error is tolerated [14]. Therefore, a neural network design with 8-bit precision operands is preferable [18]. In this proposed efficient architecture of the DNN accelerator, we have used signed 8-bit fixed-point arithmetic precision.

The he non-linear AF such as sigmoid/tanh cannot be approximated efficiently using only combinational logic [18]. However, using purely combinational logic has the benefits of providing low latency with small area overhead compared to conventional ROM-based approaches. In [43], an approximation scheme for tanh AF implementation has been proposed using combinational logic design to explore sigmoid function evaluation further. However, the complexity of circuit design will increase for higher precision AF implementation. The reuse of hardware resources with an improved architecture for configurable AF implementation is investigated in [35]. This technique achieves high utilization of the FPGA and remarkably improves the physical parameters of FPGA but suffers from low throughput.

In [17], the authors used a single multiplier and adder with iterative accumulation for MAC computation, and AF has been used in every neuron. In DNN, each layer has many neurons, and it requires more hardware resources due to parallel architecture and consumes more on-chip power. Based on the concise review, we address the hardware-efficient and performance-centric solution. We have designed a signed 8-bit dynamic fixed-point hardware-reused DNN accelerator with insignificant loss in throughput. In addition, we have resized the MAC output that allows efficient use of AF with lower precision implementation. Further, area-efficient Taylor series expansion is used for Piece-Wise Linear AF implementation. Finally, we used Block RAM to store pre-calculated values of sigmoid AF, which is beneficial at higher precision.

6.2.2 Motivation

The main bottlenecks of the FPGA implementations are limited hardware resources and limited Programmable System (PS)-Programmable Logic (PL) data transmission bandwidth. In order to reduce the memory bandwidth of the DNN implementations, the weights and bias constants should

be stored as close to the processing element as possible. Further, neural network MAC featuring a single multiplier in each neuron will access the data serially and compute iteratively. The final weighted sum of neuron's j-inputs is calculated by MAC unit after j^{th} clocks considering that j accumulative iterations are required for the final desired output. At the same time, implemented sigmoid AF at each neuron is unused for j clocks and gets active at $(j+1)^{th}$ clock. Furthermore, the parallel AF implementation in each layer takes more hardware resources, and inherently unused hardware of the AF during MAC accumulation for j clocks comes with static power dissipation. We address some of these difficulties by developing and implementing an FPGA-based DNN implementation that reuses the AF in particular and is suited for systems where hardware resources are limited.

The lesser hardware resources and lower power consumption are significant for increasingly important edge computing solutions. In this respect, we have designed an efficient DNN architecture with better resource utilization and other performance parameters. The proposed hardware implementation technique uses multiplexed and serialized data path that allows reusing the AF. We have serialized output of the MAC array within the layer using shift register in each layer and efficiently reused single AF for excitation. The embedded design approach is divided into two parts: firstly, a DNN core is designed for proposed hardware-reused architecture with a control unit for weight/bias access that includes optimized AF. An efficient log_2 quantized scheme is used for serial data access using FIFO. Secondly, an embedded block design is implemented on the Zybo FPGA board to verify the design and compare physical performance parameters. Additionally, to address the ASIC DNN implementation, our AF is synthesized at 45 nm technology, and physical performance parameters are compared with the state of the art.

6.3 System Architecture and FPGA PS-PL Integration

This section explains the methodology for the efficient design and implementation of the DNN accelerator. The FPGA-based embedded approach is the best option because of its good performance, fast implementation, energy efficiency, and reconfigurability [23, 132]. In this context, we have developed an efficient hardware architecture of the DNN that employs the AF reuse technique that reduces the hardware resource requirements. The proposed system architecture is validated on a 16-level thermometer to 4-bit binary code converter using four layers *16:16:10:4* DNN with a configuration of 1-input, 2-hidden, and 1-output layers. In each layer, output values of an array of MACs are serialized using a shift register (PISO) and iteratively passed through a single AF.

We developed an efficient architecture that gives the required performance within the overall system requirement (size, weight, power, cost, etc.). The complete embedded architecture with the DNN hardware accelerator is implemented on the Xilinx Zybo board. The FPGA implementation was done using Xilinx Vivado HLx and Xilinx SDK for hardware design and data initialization, respectively. The Advanced eXtensible Interface (AXI) bus is used to connect the custom DNN core to the controlling CPU. The AXI bus supports DMA data transfer and achieves high data

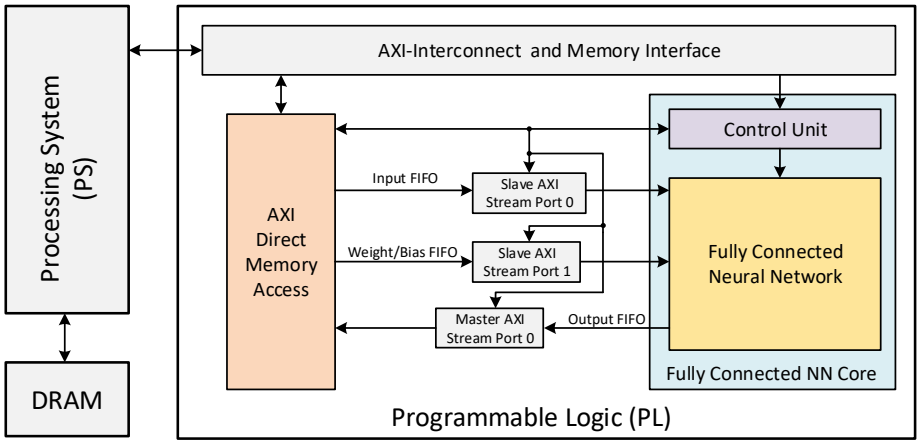


Figure 6.3: The DNN co-processor architecture in which DNN architecture and on-chip BRAM memory banks, used to cache weights/biases, are implemented using programmable logic.

throughput. The hardware implementation is done using VHDL hardware description language, and the external communication through the Processing System (PS) is developed in 'C' for an embedded architecture.

6.3.1 System overview

This work demonstrates the four-layer neural network on an affordable Xilinx FPGA SoC. Since BRAM memory has low latency, our architecture uses BRAM memory for the FIFO buffer implementation, for storing weights, and for storing pre-calculated values of the AF. The software control of the embedded processing system has been implemented using Xilinx SDK. The software control module is responsible for loading input features and weights into FIFO buffers. The design of the embedded architecture DNN co-processor is shown in Figure 6.3.

FIFO buffers are efficiently implemented using internal BRAM since it is dual-port memory and has low latency [133]. The weights that are stored in the BRAM are loaded through FIFO buffers. The data is serialized before AF, allowing the reuse of AF, and the internal hardware architecture is customized for the data-flow. The weights are loaded serially into the FIFO buffer and out from it into the MAC local registers. After processing all layers of the neural network, the output of the last layer is stored in output FIFO buffers. Thus, the runtime access of weights and biases, i.e., loading and accessing them, is efficiently designed. We devised an efficient technique for loading all necessary weights into FIFO buffers in sequence for all MAC computational elements. It allows the MACs not to wait to load the weights and access the BRAM, which has already been loaded in the local registers of the computational element.

The $\langle 8, 7 \rangle$ fixed-point format shown in Figure 6.4 demonstrates the arithmetic computations used in developed architecture. This 9-bit field representation uses 1-bit as a sign bit, 1-bit for the integer part, and 7-bits for the fractional part. Symbolically, this is written as a signed 'Fixed $\langle 8, 7 \rangle$'. Thus, it represents a signed 8-bit fixed-point number of which seven rightmost bits are fractional. The HDL hardware coded DNN architecture *16:16:10:4* is validated for thermometer

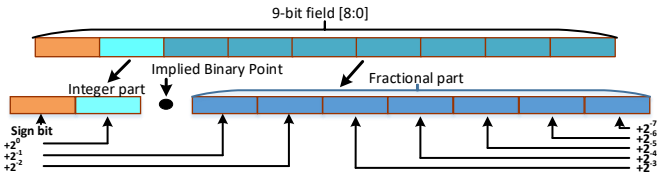


Figure 6.4: Signed Fixed $\langle 8, 7 \rangle$ precision representation used for the data representation with binary point and arithmetic calculation

level to binary value conversion using 'fixed $\langle 8, 7 \rangle$' representation, and it gives 98.9% accuracy with insignificant throughput loss to fully parallel architecture.

6.3.2 Memory addressing scheme for reading/writing of weights and biases

In a fully connected neural network, DNN hyperparameters vary with different applications. Hence, architecture should be upgradable regarding the number of layers in the neural network and the number of neurons in each layer. The weights/bias addressing should have a viable scheme for memory allocation. In fixed dedicated memory allocation in adaptable Deep Neural Networks such as [134], several memory locations do not correspond to any physical implementation. Hence it should avoid read or write operation on these unimplemented addresses. Thus an efficient scheme of memory addressing is required for the weight/bias access. Here, we use FIFO for the temporary data storage, and BRAM is used for the on-chip storage. An efficient address mapping scheme for weight/bias memory for each neuron is shown in Figure 6.5. The scheme is derived using the following labels:

- L is the total number of layers of the DNN model,
- $N(l)$ is the number of neurons or biases in l^{th} layer,
- $J(l)$ is the number of inputs to the l^{th} layer.

Note that the number of neurons in current layer is a number of inputs to the following neuron as given in Eq 6.3.

$$J(l+1) = N(l) \tag{6.3}$$

The address mapping scheme is illustrated in more detail in Figure 6.5. Figure 6.5(a) gives an addressing scheme to access the DNN parameters (weights and biases). In this scheme, a parameter is identified by its layer ID, by its type (weight/bias), by its neuron ID, and in the case of weights by the corresponding input ID. First $\lceil log_2(L) \rceil$ MSB bits represent the layer ID of the addressed parameter. The next bit is called a *select bit*, and it determines the type of the parameter given in subsequent bits. *Select bit*='1' denotes that the following bits represents bias address, whereas '0' denotes that the following bits represents the weight address. Bits following the *select bit* represent weight or bias RAM address as shown in Figure 6.5(b). The length of the weight or bias address $R\_addr(l)$ is the maximum of length of the weight address and the bias address and is given by

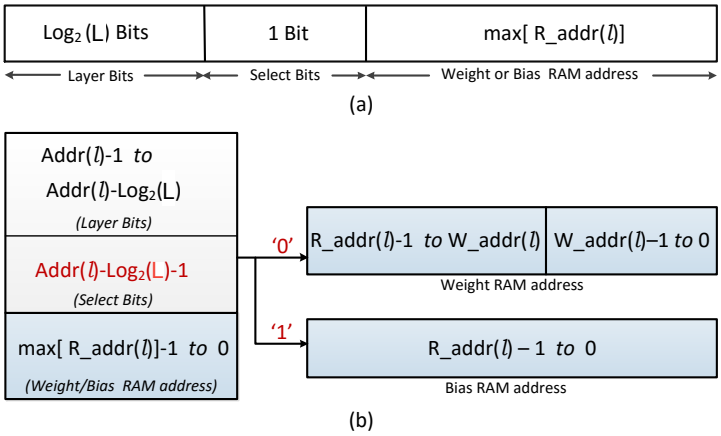


Figure 6.5: Mapping scheme for address bits that requires to address weights and bias for the individual neurons.

Eq 6.4(a). Whereas the bias address consist only of neuron identifier with length $\lceil log_2 N(l) \rceil$, the weight address includes also input identifier with the length $W\_addr(l) = \lceil log_2 J(l) \rceil$. The required address length including layer ID and parameter type $Addr(l)$ for layer l is given by Eq 6.4(b).

$$R\_addr(l) = \lceil log_2 N(l) \rceil + \lceil log_2 J(l) \rceil \tag{6.4a}$$

$$Addr(l) = \lceil log_2 L \rceil \ + 1 + R\_addr(l) \tag{6.4b}$$

Since the required address length varies depending on the layer and since fixed address length is used in the addressing scheme, the address length $Addr$ is the maximum required address length over all layers $l = 1, 2, ..., L$, and is given by Eq 6.5.

$$R\_addr = \max_{l=1,2,...,L} \left\{ \lceil log_2 N(l) \rceil + \lceil log_2 J(l) \rceil \right\} \tag{6.5a}$$

$$Addr = \lceil log_2 L \rceil + 1 + R\_addr \tag{6.5b}$$

6.4 Proposed Data Multiplexed Architecture

This section presents the proposed performance-centric hardware-reused DNN implementation. The proposed architecture reuses the resource-hungry non-linear AF using data multiplexing and is area and power-efficient, and with minimal performance loss compared to fully parallel implementation. It was implemented and tested on FPGA; however, the proposed DNN core can also be efficiently implemented in ASIC. The complete DNN core is written in *VHDL*-hardware description language, and it was integrated using block design of *Xilinx Vivado Design Suite*.

The dynamic $\langle 8, 7 \rangle$ fixed-point arithmetic used in the developed DNN architecture was selec-

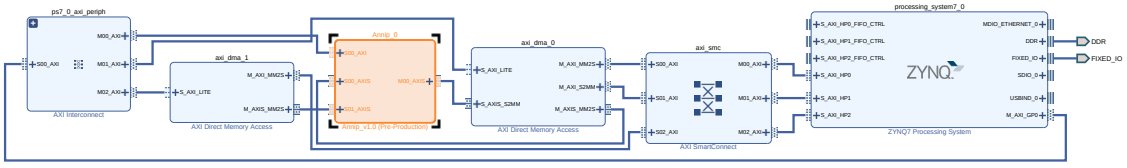


Figure 6.6: Block design of system architecture with use of AXI interconnects. The proposed design DNN IP (Annip_0) is shown in orange color. Processing system (PL) is used for the external communication.

ted through experimental evaluations as a compromise between resource usage, performance, and accuracy. Compared to the floating-point format, the $\langle 8, 7 \rangle$ fixed-point format comes with two advantages. Firstly, fixed-point computational units are usually fast and consume minimal hardware resources and power, i.e., the logic design with fixed-point arithmetic will allow more instantiating for the given area. Secondly, memory footprint will be reduced, thus allowing larger models in given memory capacity. In addition, it dramatically increases data parallelism. The following subsections describe key features of the accelerator design and implementation.

6.4.1 Integrated block design of DNN accelerator

Hardware-based implementations of Deep Neural Networks face the challenges of memory bandwidth limits. Therefore, the efficient addressing scheme for dynamic loading of the weights and the bias constants in the local registers of the MAC unit is required. All FIFO registers used to store trained weight/bias constants are loaded using the scheme as discussed in Section 6.3.2. The block design of the embedded architecture is shown in Figure 6.6. Annip_0 block, which is marked in orange, is proposed DNN core and is connected to PS by three slave and one master AXI interconnects. The two slave AXI streams are used to access the input features and weight/bias constants. The third slave interface is an AXI lite interface used by the DNN core control module. The control module manages data exchange, handshake signals, and other configurable signals.

`Ready`, `DNNInit`, and `DNNDone` are three control signals at the top-level module that control data exchange, initiate computing, and acknowledge completed calculations to the top-level module. The `Ready` signal initiates the loading of the weight/bias into the local registers of the DNN core. The `DNNInit` signal triggers the computation once the weight/bias loading is done. The `DNNDone` is a control signal which indicates that the computation has finished and the DNN output is ready.

6.4.2 Computation mechanism for each fully connected layer and DNN IP architecture

In order to initiate the DNN computation, the `DNNInit` signal become active once the weight and bias constants are loaded in the local registers of the MAC units. Each layer has the `LayerInit` input signal used to initiate the layers computation, whereas `LayerDone` signal is generated by the layer which is used to acknowledge that the output of the layer is ready. The top-level DNN Core has two additional sub-modules which interacts with the control module:

- Weight loading module: Due to the limited number of DMA channels available in Zynq,

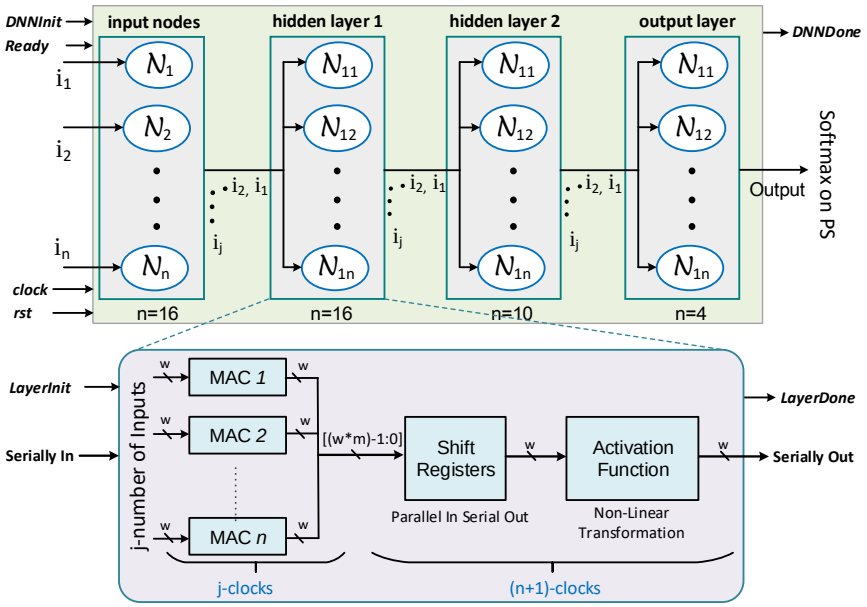


Figure 6.7: DNN architecture with efficient design of hidden layer's architecture that reuses resources hungry Activation Function. The parallel output of array of MAC is serialised using shift register and pass it in packets to the AF.

we found a workaround for loading the weight in sequence into corresponding FIFOs by multiplexing the handshake signals based on the index of the DMA data transfer request, i.e., neuron's FIFO that accepts the data is selected based on index of loading DMA request.

- Computational module: It is build from Layer modules which perform calculations for individual DNN layers and are daisy-chained by data and handshake signals. Each layer module is controlled by two handshake signals: `LayerInit` and `LayerDone`, first is input signal of the layer module that initiates layer computation while later is output signal of the layer module and indicates that the layer computation has finished. Since the computation of subsequent layer has to start after the computation of current layer has finished the `LayerInit` signal of the subsequent layer is connected to the `LayerDone` signal of the current layer. Similarly the output of the current layer is connected to the input of the subsequent layer as depicted in Figure 6.7. The `LayerInit` signal of the first layer is connected to the top-level `DNNInit` control signal, which initiates the DNN computation. On the other hand the `LayerDone` signal of the last layer is connected to the top-level `DNNDone` control signal, which acknowledges the master *AXI stream*, which is used to access the output of the DNN core as shown in Figure 6.6, that the DNN computation is finished.

In general DNNs consists of one input layer, one output layer, and many hidden layers. Each layer may have many parallel neurons. Each neuron consists of a MAC computational unit followed by an AF. Thus, in a fully parallel architecture the layer with N neurons has an N number of MACs and AFs. In contrast to the fully parallel architecture, the proposed architecture uses N number of MAC units and a single AF unit shared through Parallel-In Serial-Out (PISO) unit placed in between the MACs and AF. The serial output stream of the current layer is the input

feature i_1, i_2, ... i_j for the subsequent layer, i.e number of inputs of the next layer is the number of neurons in the current layer as shown in Figure 6.7. In order to achieve better performance, the proposed design for each layer has the following key features:

1. FIFO Buffer: Each multiply-accumulate unit has an input buffer and an output buffer to receive and send the input, weight, and output data in FIFO.

2. Pipeline Accelerator: We use a stream-like data passing mechanism (AXI stream) to transfer data between the input data and multiply-accumulate unit.

3. Neural Unit: Contains one FIFO and one MAC unit that is used for computation of one neuron in correspondence with its assigned index.

4. Data Quantization: The AF is implemented for limited precision to address minimum resources utilization. Hence the output of the processing unit is resized into 'Fixed $\langle 8, 7 \rangle$'.

5. Data Serialize: The output of the array of MAC unit available in the single-layer are serialized to reused the single AF within the layer.

The proposed DNN core's timing complexity has been elaborated by determining the number of clock cycles needed to perform the computation. Let T_R and T_P represent the number of clock cycles delay required to compute outputs for proposed AF *Reused* and fully *Parallel* DNN architecture, respectively. The proposed data multiplexed layer architecture computes the MAC output in j clocks, where j is the number of layer's input features, and computes the first neuron output at $j+2$ clock cycles while the outputs of the following neurons are computed by one per clock cycle. One can notice that it is due to the sharing of a single AF in each layer. Whereas in fully parallel layer architecture, MAC computes outputs for all MAC in j clock cycles and computes outputs of all neurons in $j+1$ clock cycles. It is important to note that while parallel architecture computes all outputs at $j+1$ clock cycle, the next layer reads them serially one per clock cycle due to the iterative computational nature of the used MAC unit. This implies that in the case of hidden layers the data multiplexed architecture introduces the delay of the PISO module, which is one clock cycle as described in Eq. 6.6.

$$T_R(l) = T_{MAC}(l) + T_{PISO} + T_{AF} = j(l) + 1 + 1 \tag{6.6a}$$

$$T_P(l) = T_{MAC}(l) + T_{AF} = j(l) + 1 \tag{6.6b}$$

On the other hand, the output layer does not have the next layer and the serial computation of the AF unit must be taken into account. Therefore, the computation time of the output layer of the proposed architecture output requires additional $n-1$ clock cycles to compute all outputs.

The computation time of the output layer for both architectures is given in Eq 6.7.

$$T_R(L) = T_{MAC}(L) \; + \; T_{PISO} + \; T_{AF} = j(L) + 1 + n(L) \tag{6.7a}$$

$$T_P(L) = T_{MAC}(L) \qquad\qquad + \; T_{AF} = j(L) + 1 \tag{6.7b}$$

The computation time T_R of the whole AF reused DNN architecture is computed by summing the computational time of the individual layers. The first layer is input layer and does not perform computations and is omitted from the sum illustrated in Eq. 6.8.

$$\begin{aligned} T_R &= \sum_{l=2}^{L-1} T_R(l) + T_R(L) \\ &= \sum_{l=2}^{L-1} \Big(j(l) + 2) \Big) + j(L) + 1 + n(L) \end{aligned} \tag{6.8}$$

By applying the property that number of neurons in a layer is equal to the number of inputs in next the layer (Eq 6.3) the equation can be simplified to Eq. 6.9.

$$\begin{aligned} T_R &= \sum_{l=1}^{L-2} n(l) + 2(L-2) + n(L-1) + 1 + n(L) \\ &= \sum_{l=1}^{L} n(l) + 2L - 3 \end{aligned} \tag{6.9}$$

Similarly, the computation time T_P of the fully parallel DNN architecture is

$$T_P = \sum_{l=1}^{L-1} n(l) + L - 1 \tag{6.10}$$

By comparing T_R and T_P computation time it can easily be seen that the computation time of the AF Reused DNN architecture is longer by the number of hidden layers $h = L - 2$ and by the number of DNN outputs $n(L)$ as given in Eq 6.11. Further, time delay of accelerator output can be compute by simply multiplying clock time t_{CLK} with the number of clock periods T_R evaluated using Eq 6.9.

$$T_R = T_P + h + n(L) \tag{6.11}$$

Let us illustrate how the computation time can be determined on the evaluation example of the `16:16:10:4` feed-forward DNN using proposed DNN architecture. First hidden layer with 16 neurons has $j(2) = 16$ inputs and its MAC units require 16 clock cycles to compute weighted sum. The processing of the output of the first neuron requires two additional clock periods: one by PISO unit and the other for AF unit, hence the output of the first neuron is computed in 18^{th} clock cycle as given in Eq 6.6a. Second hidden layer has 10 neurons and 16 inputs and can start processing after 18 clock cycles. By the analogy it requires additional 18 clock cycles thus the output of its

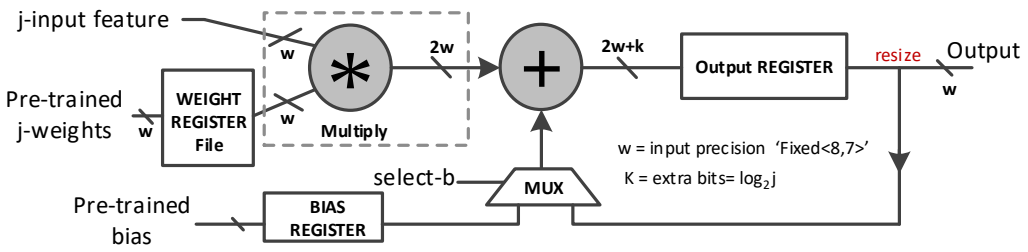


Figure 6.8: Iterative multiply-and-accumulate (MAC) computating unit with resized output data.

first neuron is available after 36 clock cycles. The output layer has four neurons and 10 inputs. 10 clock cycles are used for MAC units to compute weighted sum, one clock is used by PISO unit, however to complete the DNN computation the outputs of all neurons must be determined. Due to serial AF computation this requires four clock cycles. Thus 15 clock cycles are required for the output layer. For this example 51 clock cycles are required for DNN computation in this example. Similarly, 45 clocks (i.e., 17+17+11) are required to compute the output of the fully parallel DNN.

6.4.3 Architecture design and implementation of w-bit precision multiply-and-accumulate unit

Each neuron has MAC computational unit followed by an AF unit. The MAC unit with parallel multipliers and adder tree in each neuron are hardware costly and burdensome on tiny FPGAs. Hence, iterative MAC architecture that uses minimal resources utilization at the cost of throughput loss [18] is preferred. The resources-efficient iterative MAC unit used in this work is shown in Figure 6.8. It requires j clocks for the computation of weighted sum, where j is the number of inputs at the MAC computation. The standard multiplier and adder defined in IEEE library package are used, and implemented using logic slices (CLBs) on FPGA. Here, we used fixed-point 8-bit precision 'Fixed $\langle 8,7 \rangle$' arithmetic for input feature and weights/biases i.e, `input[8:0]`, `weight[8:0]` and `bias[8:0]`. The output of the MAC unit is the sum of input bits, weight memory bits, and overflow bits required for the weighted sum accumulations. Therefor, output of multiplier gets accumulate using adder with extra overhead bits (k), i.e., k=$\lceil log_2\ j \rceil$. We use `select_b` to load the bias constant that gets added in first multiplication, i.e., at the first clock initiating the MAC computation. Afterward, it selects the accumulation path using the same `select_b` signal.

The neurons have MAC followed by non-linear transformation over the weighted accumulated sum, and therefore the precision of an AF depends on the output bit-width of the MAC unit within the neuron. The MAC used in the proposed design is shown in Figure 6.8. The output of the MAC unit $[2w+k]$-bits, whereas practically it is not desirable to keep the $[2w+k]$ bits precision AF for hardware implementation. Hence, we have resized MAC output into w-bits (9-bits) using *numeric_std* library package of VHDL. It allows us to keep the same input and output precision for MAC implementation with dynamic fixed-point arithmetic 'fixed $\langle 8,7 \rangle$' as shown in Figure 6.8. Besides, in most cases it is beneficial to have the same input and output format that we use and implies $i_{b\_in} = \ i_{b\_out} = i_b$, $f_{b\_in} = f_{b\_out} = f_b$ and $w_{in} = w_{out} = w$, where i_b represents the integer

bits excluding the sign bit, f_b represents the fractional bits and w represents the total number of bits at input/output. However, in the case of fixed-point format final choice must be based on a target application that gives maximum accuracy with desired precision [14].

6.4.4 Design methodology and implementation of activation function

In DNN, MAC output is the input for a non-linear transformation function AF. Due to the non-linear nature of the AF, it demands more hardware resources for implementation, and resources utilization increases exponentially for higher precision. In addition, the AF with 16-bit and higher precision requires more memory elements for PWL implementation, and therefore, it is very costly for hardware utilization as the number of required memory elements increases exponentially with the quantization precision at the input of the AF. The sigmoid function is the extended version of an exponential function, and the exponential function can be evaluated using Taylor series expansion. For example, the Taylor series expansion for a hyperbolic and exponential function is shown in Eq. 6.12. Whereas sigmoid AF can be implement using this Taylor series expansions. The higher order of Taylor series expansions returns better accuracy but comes with high precision representation. Hence, performance-centric evaluations have been done between the Taylor series order used in AF evaluation and DNN accuracy. The analysis of the impact of Taylor series order on inference accuracy is shown in Figure 6.9.

$$sinh\ (z) = z + \frac{z^3}{3!} + \frac{z^5}{5!} + \frac{z^7}{7!} + \frac{z^9}{9!} + \ \ldots . \tag{6.12a}$$

$$cosh\ (z) = 1 + \frac{z^2}{2!} + \frac{z^4}{4!} + \frac{z^6}{6!} + \frac{z^8}{8!} + \ \ldots . \tag{6.12b}$$

$$e^z = sinh\ (z) + cosh\ (z) \tag{6.12c}$$

An exponential function has been evaluated using the sum of the Taylor series for hyperbolic *sin* and *cos* function, as shown in Eq. 6.12 (c). We have used an optimized sigmoid AF evaluation technique for both negative and positive inputs. In this work sigmoid function is represented using positive exponential function as given *f(z)* for positive and negative inputs. The elaborated equations for the positive and negative half are shown in Eq. 6.13. The Pareto study between series expansion order and accuracy shows that accuracy gets saturated after the fifth-order of series expansion as shown in Figure 6.9. Hence, fifth-order Taylor series expansion has been used for the memory elements extraction in the 'Fixed $\langle 8, 7\rangle$' representation. Further extracted memory elements for sigmoid AF are stored in the BRAM for real-time function evaluation. The proposed approach can easily be extended for tanh AF implementation simply by dividing the *sinh* and *cosh*. However, sigmoid and tanh AF can also be implemented using a different approach [21].

$$f(z) = sigmoid(z) = \begin{cases} \dfrac{e^z}{1+e^z}, & \text{if } z \geq 0 \\ \dfrac{1}{1+e^{-z}}, & \text{otherwise} \end{cases} \tag{6.13}$$

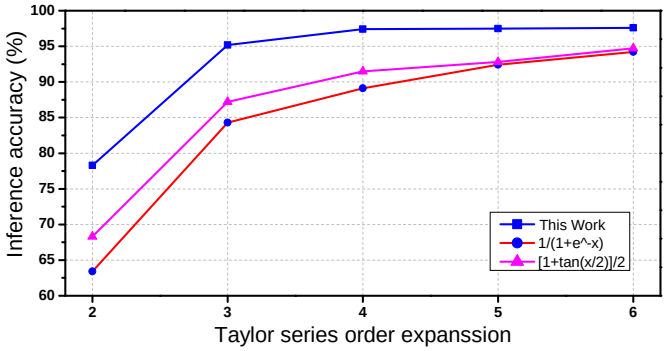


Figure 6.9: Inference accuracy for fully connected neural network using sigmoid activation function. The network 784:256:128:128:10 is trained for MNIST dataset and used sigmoid with different order of Taylor expansion. The results shown for 'fixed $\langle 8, 7 \rangle$' precision.

The implemented sigmoid AF design is efficient in terms of physical performance parameters such as area, power, and delay, where we have used only positive exponential function for sigmoid AF evaluation. We have analyzed the order representation of Taylor series expansion for sigmoid AF. The order of Taylor series expansion for AF versus inference accuracy for the proposed scheme and state of the art is shown in Figure 6.9. The accuracy results are evaluated for the MNIST dataset on DNN configuration `784:256:256:128:10`. The arithmetic representation of the integer bits and the fractional bit have been presented using the 'Fixed $\langle 8, 7 \rangle$' scheme. Here, we trained the model for 20 epochs and stopped the training if the validation accuracy decreased consecutively for two epochs. Then we evaluated the trained model on a test set of MNIST and observed the accuracy. Conventionally these equations computations have shown in Figure 6.9 represent the same output; however, due to 8-bit fixed-point representation, numbers get quantized at every math operation involves in AF computations. Thus results show the sigmoid with the proposed technique has better accuracy compared with equations used in previous works [47, 135].

6.5 Experimental Results and Discussion

We have evaluated the system architecture at different abstraction levels for setting the design parameter for hardware implementation. Finally, the embedded design of the DNN has been implemented on hardware. The detailed analysis and evaluation results are given in the following subsections.

Experimental setup

The system architecture is represented at the RTL level using VHDL language to evaluate the performance of the proposed design. The RTL for our system architecture is synthesized using Vivado-Xilinx, and implementation results are produced for Zybo FPGA hardware. Inference accuracy for sigmoid AF on DNN is evaluated through software evaluation. Further for ASIC design, synthesis results for an optimized sigmoid AF are extracted using *Design_Vision- Synopsys*.

Table 6.1: Network inference accuracy of DNN size 784:256:128:128:10 with Piece-Wise Linear (PWL) AF [46] and TenserFlow library based AF [5] at different bit-precision with dynamic fixed-point representation @MNIST data-set

Activation function precision	Inference accuracy (%) for different AFs					
	ReLU	Sigmoid			tanh	
	Tensor [5]	Proposed	PWL [46]	Tensor [5]	PWL [46]	Tensor [5]
32-bit	97.51	98.20	97.94	98.30	97.14	98.58
16-bit	96.71	97.63	97.86	97.58	96.90	98.53
8-bit	97.50	97.06	97.58	96.68	95.54	97.93

Following experiments are done to validate our proposed design:

1. The inference accuracy has been evaluated for different precision 8, 16, and 32-bit fixed-point arithmetic representations of sigmoid AF implementation with different approaches to finalize the precision selection for hardware implementation.

2. The DNN configuration `784:256:128:128:10` is designed for 'Fixed $\langle 8, 7 \rangle$' precision using QKeras. The Pareto study tuned the Taylor series order expansion for the sigmoid AF implementation, which returns better accuracy and physical performance parameters. MNIST dataset is used for evaluation and validation of AF. Further ASIC synthesis results for 8-bit precision are presented at *45 nm* technology.

3. The proposed hardware efficient data multiplexed DNN architecture implantation is validated on the Zybo board for network size `16:16:10:4`. Here, we have designed a custom IP for the proposed architecture and show the implementation results for the FPGA prototyping of the proposed resources-reused (i.e AF) design architecture.

6.5.1 Accuracy for multi-bit precision and Pareto study for Taylor series expansion

The QKeras model is used for the training of the DNN as it allows drop-in replacement for the quantized versions of the conventional *Dense* and *Activation* layers. The accuracy has been observed for the MNIST dataset on DNN configuration `784:256:256:128:10` with 8, 16, and 32-bit precision, as shown in Table 6.1. Furthermore, our sigmoid AF design's evaluated accuracy using MNIST dataset has been compared with the accurate AF function of the Tensor library [5] and Piece-Wise Linear Activation Function [46]. From Table 6.1, it can be observed that there is an insignificant accuracy loss ($\leq$2%) between 8-bit and 32-bit precision computation. However, 8-bit precision computation leads to reduce memory computation demand by a factor of 4. Hence we have chosen the fixed-point 8-bit precision for hardware implementation. We have also evaluated the optimal position of the decimal point, i.e., one integer bit and the rest fractional bits in the dynamic fixed-point arithmetic.

Table 6.2: Resource utilization and performance parameters for 4-layer (*16:16:10:4*) on xc7z010clg400 at $100MHz$

Hardware Avail. Logic Resources	DNN with AF Reused [91]	DNN with Layer Multiplexed [101]	DNN with Layer Reused [17]	DNN with Fully Parallel AF [24]	Hardware Reused (AF) Proposed	% Utilized Relative [24]
Slice LUTs (17600)	10439	6733	6127	9968	8858	88.86
Slice FFs (35200)	7573	6018	5904	10638	6219	58.46
BRAM (60)	6.5	9	13.5	11	6.5	59.09
BUFG (32)	8	8	8	8	8	–
On-chip Power (mW)	158.5	$88.12 \times$ L	$74.04 \times$ L	120.46	106.05	88.33
I/O Critical Delay (ns)	11.6	$4.07 \times$ L	$3.28 \times$ L	9.23	9.86	-6.82
Throughput ($samples/s$)	T	T/L^1	T/L^1	T	T	–

[1]L is the number of layers available in the implemented DNN

6.5.2 Data multiplexed DNN implementation and results comparison

The Xilinx Zybo (xc7z010clg400) FPGA hardware contains 4,400 logic slices; each logic slice includes four 6-input LUTs and 8-FF. Additionally, it provides 240 KB of BRAM. In order to evaluate the resources utilization and power consumption, we used Xilinx-Vivado tool. Through an efficient addressing scheme, local memory registers with weights and bases are loaded. Architecture has been implemented on Zybo. Furthermore, the DSP blocks have not been used for logic implementation due to their fixed architecture for higher precision and power consumption constraint.

We have implemented data multiplexed hardware reused architecture and results are extracted. For the sake of comparison with state-of-the-art approaches, we used the same design parameters which are 'Fixed $\langle 8, 7 \rangle$' arithmetic, DNN size, and evaluation kit for the implementation of all the previous and proposed work. We used DNN configuration `16:16:10:4` for the implementation and results comparison. The hardware architecture has been designed using *VHDL* and for DNN core extracted post-implementation results on the *Zybo* FPGA board are shown in Table 6.2.

In [24], authors have design DNN with fully parallel layers. The network has high throughput and flexibility to change the number of nodes in the network, allowing us to configure the network for different applications. However, the design architecture has further scope to make it hardware efficient using the data multiplexed and AF reused scheme. An efficient hardware design has been proposed where authors have reused the AF [91]. This work has used a multiplexer between the parallel MAC units and AF in which MAC address is the input select line for the MUX. The hardware implementation results show less LUT utilization as authors have reused the AF through multiplexing shown in Table 6.2. However, techniques are not efficient for a larger size of a DNN as multiplexer increases the time complexity of the logic architecture, and further bigger size multiplexer requires more hardware resources. We have addressed this issue and proposed an efficient AF reused technique using shift register following the Parallel-In Serial-Out's mechanism.

The proposed technique is efficient for any deep neural network at very insignificant throughput loss, as we discussed in Section 6.4. For the performance parameter comparison, we have synthesized the different DNN architectures and proposed DNN core for `16:16:10:4`. The implementation report for DNN core is shown in Table 6.2. The hardware implementation results show

that our method uses *25%* fewer resources and requires *12%* less power, without performance loss compared to the work proposed in [24] at the cost of insignificant delay overhead compared to parallel architecture in DNN as discussed in Section 6.4. Here, L is the number of layers available in the DNN. Therefore, L-1 clocks period delay occurs for the first output in comparison with the fully parallel architecture. However, after the first output, the network computes the output at every clock, and hence reused architecture has insignificant throughput loss, as we have discussed in Section 6.4.2. Compared with AF reused architecture design in [91], implementation results show 17.8% and 21.7% fewer LUT and FF utilization. Further, the proposed design has 17.6% less critical delay. For the proposed design, the comparison number will be adequate for the bigger sizes of DNNs.

In order to design resource-efficient architecture, state-of-the-art works have proposed layer-reused architectures [17, 101]. Though the design is hardware efficient, from Table 6.2, one can see that layer reused designs have $L\times$ lower throughput as compared to fully parallel architecture. That architecture implementation will be efficient for less resources utilization at the cost of low throughput. In [101], the author has used the multiplexer for passing the output data of the current layer to the input of the same layer; using this one can reuse the single layer at any number of times within the DNN. However, design suffers from low throughput. Moreover, the design has used parallel AFs within the layer. Therefore, this design has further scope to reduce resources utilization using the AF reused technique by compromising throughput loss of one clock period in each layer computation. The resources utilization for fixed parameters is shown in Table 6.2. It can be observed that [17] has higher BRAM utilization compared the [101]; as no quantization scheme was used in [17]. However, these designs can adopt the proposed AF reused scheme to further optimize hardware utilization.

The design's performance parameters are in the trade-off for area/power and throughput. However, the proposed design will be the best choice for area/power and throughput-centered applications where resources and power are on a tight budget, such as Mobile, edge-AI, IoT applications, etc.

6.5.3 Implementation results and comparison of proposed AF at multi-bit precision

In an AF, moving from 8-bit to 12-bit and 16-bit, the number of quantization states is equal to $2^8 = 256$, $2^{12} = 4096$, and $2^{16} = 65536$ which leads to an exponential increase in the memory elements required. Therefore, we used the pre-calculated memory element for AF implementation using the Taylor series expansion approach and efficiently stored it in BRAM. Table 6.3 shows the resource utilization comparison at different precision for our optimized architecture with the utilization of BRAM and LUT-based approach [42]. Generally, tiny FPGA have limited slice resources utilization, whereas it can be seen that 16-bit precision sigmoid AF needs *2111* LUTs. Here, it can be observed that higher precision AF implementation for LUT based approach is not

Table 6.3: Activation function implementation using LUT based and proposed technique for Zybo xc7z010clg400

Resources Utilization (Available)	Sigmoid Activation Function Resources Utilization					
	8-bit		12-bit		16-bit	
	LUT [44]	**Proposed**	**LUT** [44]	**Proposed**	**LUT** [44]	**Proposed**
LUT	16	1	370	1	2111	20
FF	0	1	0	1	0	5
BRAM	0	0.5	0	1.5	0	32
BUBG	0	1	0	1	0	1

Table 6.4: Performance parameter metrics of 8-bit precision proposed design for sigmoid AF and state-of-the-art @45nm TT Process Corner

Sigmoid AF Physical Parameters	**Digital Implement [41]**	**Negative Input [47]**	**Iterative CORDIC [18]**	**Memory Elements Proposed**
Chip Area (μm^2)	846	483	377	671
St. Power (μW)	27.6	121.7	96.77	21.6
Dy. Power (μW)	121.3	209.4	189.3	38.2
Delay (ns)	7.41	5.93	4.38	4.86
Energy-Delay Product (EDP)	1105	11,780	6,264	287
No. of Clock	1	6	5	1

the appropriate method. Hence, our design uses BRAM, which is scalable for higher precision, has better access time, and saves LUTs for other logic computations.

6.5.4 Comparison of AFs physical performance parameters at 45nm technology node

In order to evaluate the design performance of our optimized AF for ASICs, we have synthesized design for fixed-point 8-bit precision at *45 nm* TT Process Corner. The efficiently extracted memory elements for sigmoid AF have been compared with sigmoid using different design techniques. The *RTL* for our AF architecture is synthesized, and results are produced by *Design Vision*-Synopsys. The physical performance parameters comparison is shown in Table 6.4. Results show that compared to CORDIC-based AF, we have *4.8*× higher throughput at the cost of *1.9*× of area overhead. Further, it can be seen that the memory elements-based proposed method has better physical parameters compared to the digital design technique [41].

6.6 Summary

We presented a novel DNN architecture that reduces the hardware-resource requirements and on-chip power consumption without the loss in computational rate. Our architecture has struck a demand of resource overhead in a Deep Neural Network by exploiting the hardware-reused context. The contribution of the work is two-fold. Firstly, the activation function is designed

efficiently for higher accuracy, and quantized memory elements are stored in BRAM for better utilization and higher throughput. Secondly, our experimental results demonstrate reused AF by serializing the output of the array of MAC in each layer. The efficient memory addressing scheme is used to overcome the SoC throughput limits. The implementation was verified at 100 MHz for thermometer level to digital conversion. The performance is validated and compared with previous work for the MNIST dataset. Besides, the digital design of AF is synthesized at $45\ nm$ technology node and physical parameters are compared with previous work. The proposed hardware reused architecture is verified for neural network `16:16:10:4` using 8-bit dynamic fixed-point arithmetic and implemented on Xilinx Zynq xc7z010clg400 SoC using 100 MHz clock. The implemented architecture uses *25%* less hardware resources and consumes *12%* less power without performance loss, compared to other state-of-the-art implementations. Using extensive evaluation, we show that our proposed design gives better results as compared to other designs. Thus, the embedded system design with lower hardware resources and power consumption can significantly benefit edge computing solutions.

Chapter 7

Multi-stage Configurable Dynamic Comparator in ADCs for Analog In-memory Computing Applications

In chapter 6 we have used the application with input data as thermometer code (digital format). However, practically information is in analog form before processing. Therefore, ADC will be part of the integrated system to enable the accelerator for both analog and digital information processing. For the design and implementation of a deep neural network for ASIC and FPGA platforms, we worked on a digital solution. However, DNN has succeeded in numerous applications, including digital and analog input information. In ASIC integrated system design, an Analog to Digital Converter (ADC) is required to process the analog information since the design network processes the data in digital format. In the future, we are working on the ASIC system design for the DNN accelerator, where ADC is required as an intermediate between the analog input and digital hardware accelerator. The proposed 4-bit flash ADC supports high-speed data conversion. In order to represent the integrated system accelerator for both analog and digital applications, an ADC interface is required. Furthermore, design has scope for In-memory computing applications since ADC architecture is low power and has efficient physical performance.

Further explore machine learning in multiple applications such as edge computing, the internet of things(IoT) demands more and more power area-efficient computing elements. Therefore, in-memory computing architecture using SRAM, ReRAM, is evolved. The energy-efficient meromorphic application with emerging resistive random access memory(ReRAM) and SRAM crossbar have excellent potential. However, the max-pooling in CNN is challenging to be implemented in the analog domain. Hence it needs to be converted into digital. The low bit-level ADC/DAC interface in crossbar system design has better energy efficiency and high-speed operation to bridge the gap

between analog to digital interfaces. Furthermore, power-efficient and high-speed analog-to-digital converters demand high-speed and power-efficient architecture of dynamic comparators with good performance.

7.1 Flash type ADC with High Throughput Performance

The major drawback with flash type ADC, as the resolution increases, there is exponential growth in overall cost. The cost in input capacitance, comparator kickback noise, power consumption, chip area, and complexity in routing the signal. The crossbar-based accelerator system provides the MAC unit in the analog domain. Hence, bridging the crossbar-based processing unit with the digital peripherals like ADC & DAC is required. The peripheral circuit and memory access consume the power of the complete system architecture. In a hybrid structure, two convolution layers can directly exchange data between output and input buffers, bypassing memory access and thus increasing energy efficiency. ADC and DAC do the ReRAM crossbar array and peripheral digital circuit interfacing.

In ReRAM-based CNN, 98% area and power are consumed through ADCs and DACs, even in the case of small crossbars of size 512*512. A flash-type ADC is the choice for fast operation and low to medium resolution. Still, the comparator ladder in a flash type associated with several latches at the output of comparator stages consumes a big chunk of power. The high power consumption problem associated with the peripherals and reuse scheme [136] is used to adopt the distinct power budget. Since the weights are defined by the conductance of ReRAM and the input data is defined by input voltage signals, the ReRAM crossbar will perform the convolution kernels. The input-output parameter is related to the following Eq. 7.1.

$$i_{out,k} = \sum_{j=1}^{N} g_{k,j} \times v_{in,j} \tag{7.1}$$

where $\overrightarrow{v}_{in}$ is the input voltage (denoted by j=1,2,...,N), $\overrightarrow{i}_{out}$ is the output current denoted by (k=1,2,...,M), and $g_{k,j}$ is conductivity of the RRAM device representing the matrix data.

In future technologies, the power consumption of matching-dominated high-speed ADCs will increase to achieve the same accuracy and speed. A high-speed fash ADC needs $2N-1$ comparators for an $N-bits$ conversion. The conventional technique is not feasible in RRAM based CNN classifier. The flash-type ADC is very costly in terms of power for higher resolution. Addressing the problem of high power overhead of peripheral circuit units, especially by ADCs/DACs and memory accesses, this paper proposes a reconfigurable scheme in comparator stages of ADC to meet low power budgets. We proposed the reconfigurable comparator-based 4bit flash ADC for crossbar-based CNN shown in Figure 7.2. The data compression scheme and approximation technique can in the case of higher resolution computing architecture. The underlying idea is to put the expensive ADCs/DACs in the spotlight.

Moreover, the cascaded latch interpolation technique can be reduced by one-fourth in the num-

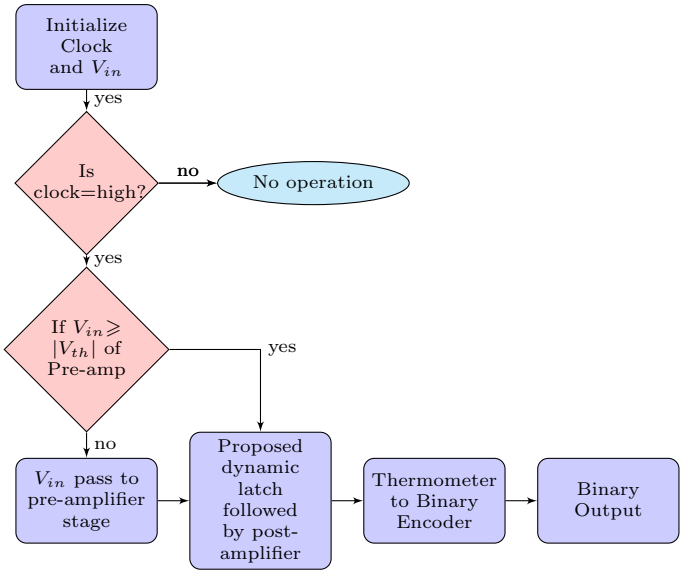


Figure 7.1: Operation flow with different stages of runtime reconfigurable differential comparator.

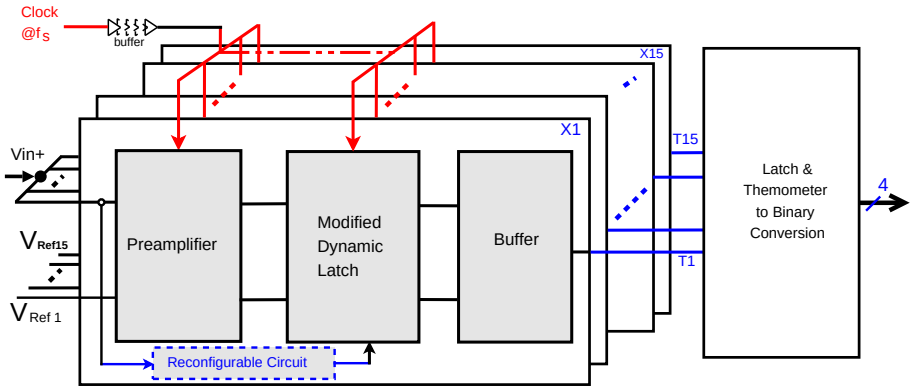


Figure 7.2: Block architecture of proposed 4-bit flash ADC

ber of first-stage latches for flash ADC. The cascaded latch interpolation is liable to be influenced by PVT variation, and a dedicated circuit that can shield robust performance is needed [137]; the resolution depends on the operating link condition in the benchmark circuit [138]. The multi-stage comparator is based on the cascade structure in a pipelined arrangement of the modified input offset storage amplifier, and the output offset storage amplifier. The topology maintains a good input common-mode range and improves speed due to reducing capacitive loads. However, this technique increases the overall power consumption. The reduced comparator count in the ladder can be significant for higher resolution to reduce the power consumption for T/H circuit load capacitance. This work has addressed the issue in the proposed 4-bit flash ADC.

7.2 Design and Implementation of Proposed 4-bit Flash ADC

In on-chip Internet of Things (IoT) applications demanding performance is required to push the limit for a solution over the power consumption in the analog front-end circuits. However, it needs system considerations like flexibility in modulation and analog-based solutions in architecture.

Power consumption is one of the major research areas for digital and analog circuit design. The solution to the limitations is application-based reconfigurability in the circuit architecture. As the number of active transistors in a chip increases, the power dissipation also goes on increase rapidly [139]. The input to the ADC from the ReRAM crossbar array will depend on the weight of the conductance and the input feature map. An inclusive architecture is required at all levels of the system design aspect, which means architectures with the logic styles and the underlying technology. An application like ADCs, especially flash type ADC, is the ideal choice in ADCs for high-speed applications (GS/s). However, it consumes more power compared to the other ADC architectures. Since flash-type ADC operates in parallel, the number of comparators increases exponentially with higher resolution, leading to significant amounts of power consumption and a large area. Unfortunately, the modern ADCs have a trade-off between power, speed, and accuracy with low complexity [140]. In very high sampling speed superconductor ADCs utilizing periodic comparators that reduce hardware complexity [141] [142], it has a perceptive significance for highly parallel large bandwidth applications.

The ADC circuit differential amplifier is the dominant agent, especially the preamplifier. If possible to bypass the preamplifier stage, then there is again the scope of power reduction and, with this motivation proposed, a design of flash ADC with reconfigurable architecture as shown in Fig. 7.2. The reference is generated using the resistive ladder and thermometer to binary code conversion using a transmission gate-based thermometer to the binary encoder. The conversion accuracy depends on the accurate definition of the reference voltages sensed by each comparator and its offset voltage. The ADC's differential and integral nonlinearity (INL) characteristics directly influence because of the offset.

7.3 Multi-stage Configurable Dynamic Comparator for Voltage Comparision

This chapter discussed the state-of-the-art comparator designs and proposed efficient reconfigurable comparator for N-bit flash ADC, which requires 2^N-1 comparators along with the resistive ladder. However, investigation shows that the comparator consumes maximum area and power. The conventional comparator includes a preamplifier, dynamic latch, and post-amplifier stages. The observation and cascade stages in the comparator can be bypassed with a reconfigurable technique. The differential comparator for low power applications is designed using a reconfigurable technique that bypasses power-hungry active stages in the comparator. However, the static power consumption by the remaining amplifier stages is still not desirable given the recent low power demand. The conversion of continuous to discrete-time is realized within the comparator. Apart from technological advancement, developing new circuit architecture that avoids stacking too many transistors between the supply rails is preferable for low-voltage, high-speed operation, significantly if it does not increase circuit complexity.

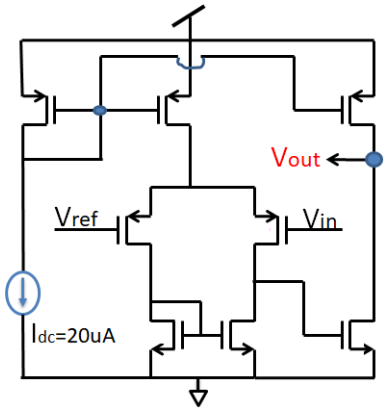


Figure 7.3: Amplifier with a current-mirror active load [2].

7.3.1 State-of-the-art comparator design architectures

The circuit with a basic model to derive design conditions for improving the speed and resolution in clockwise CMOS trans-impedance comparators as shown in Figure 7.3. The analysis gives two different architectures for dynamic comparators, depending on whether the input sensing node is capacitive or resistive. The sensing node reports that both different yields ranges of the input current. The observed input differential comparison range, i.e., the voltage gain of a prototype is 80dB and offset below $10pA$. However, a single stage with a minimum number of MOS cannot be used for the high-speed application. Simulation results show that it can be operated up to 100MHz and the average power consumption is $160\mu W$. From the above discussions, it is observed that power dissipation is very high against the operating speed.

The offset in the reference voltage comparison is due to device mismatch. The comparator circuit directly creates static nonlinearities due to a mismatch in the transfer characteristics of ADC that reflects in the degradation of the integral nonlinearity (INL) and differential nonlinearity (DNL). The multistage comparator is based on the cascade structure in a pipelined arrangement of modified input offset storage amplifier, and the output offset storage amplifier. The topology maintains a good input common-mode range and improves speed due to reducing capacitive load. However, this technique increases the overall power consumption. The multistage consist of a track-and-hold stage and subsequent buffer in the input stage, succeeded by the comparator and further amplifier & latches.

The TIQ technique uses two cascade inverters for the threshold comparison, as shown in Fig. 7.4. The first stage is an inverter that generates switching voltage internally, and the second stage acts as a gain booster [143]. For the required reference voltage, the aspect ratio of MOSFETs (W/L) used in the inverter circuit which can be adjusted and modifies the threshold voltage given in Eq. 7.2. Likewise, the CMOS-Linear tunable Transconductance Element (LTE) comparator used additional clock circuitry to control the static power dissipation of the TIQ comparator. The reference voltages are internally generated by systematically varying the transistor sizing of the CMOS-LTE comparator [3]. The limitation with all transistor sizes of these elements should be identical. The mid-point voltage (Vm) for the inverter is described as $Vin = Vout1$ in the voltage

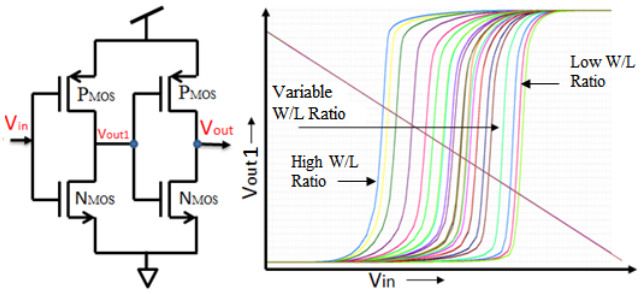


Figure 7.4: TIQ comparator design and response for its variable W/L ratio [3].

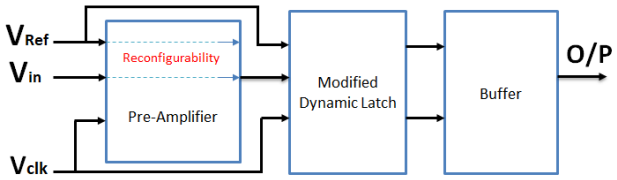


Figure 7.5: Block level architecture of runtime reconfigurable 3/2-stage differential comparator. The same V_{Ref} pin is applied to both stages of the proposed comparator. Whereas, V_{in} goes out through analog switch.

transfer characteristic (VTC) of an inverter and can be expressed as:

$$V_m = \frac{r(V_{DD} - |V_{T0,p}|) + V_{T0,n}}{1 + r} \tag{7.2}$$

where, $r = \sqrt{\frac{K_p}{K_n}}$, $K_n{=}\mu_n C_{ox}(\frac{W}{L})_n$ & $K_p{=}\mu_p C_{ox}(\frac{W}{L})_p$

The output performance comparison for TIQ-based and CMOS-LTE comparator for ±5% power supply voltage variation claims that with the use of CMOS-LTE comparator power supply variation problem can be reduced. However, practical implantation of CMOS-LTE circuit comparator design fabrication will come with a possibility of mismatch, and with this assumption, the comparator's threshold point may vary. Hence, such a prototype will not be practical due to its high mismatch and offset for high resolution.

7.3.2 Proposed configurable dynamic comparator circuit design

This section describes the different block levels of the low-power reconfigurable comparator. The principal drawback with flash type ADC is as the resolution increases; there is exponential growth in overall cost. The cost in input capacitance, comparator kickback noise, power consumption, chip area, and complexity in routing the signal. In flash type ADC, the count in the comparator is doubled, with resolution increasing by one bit. Moreover, the more active area effect is on the input capacitance, which is raised eight times. The proposed reconfigurable multi-stage comparator has stages, namely (i) Preamplifier with bypass switch circuitry, (ii) Modified Dynamic Latch, and (iii) Post-amplifier, as shown in Fig. 7.5. The trend for 4-bit and above resolution, flash type ADC need better offset cancellation with a low noise sensitive comparator and high bandwidth. The 3-dB bandwidth for the amplifier can be described as

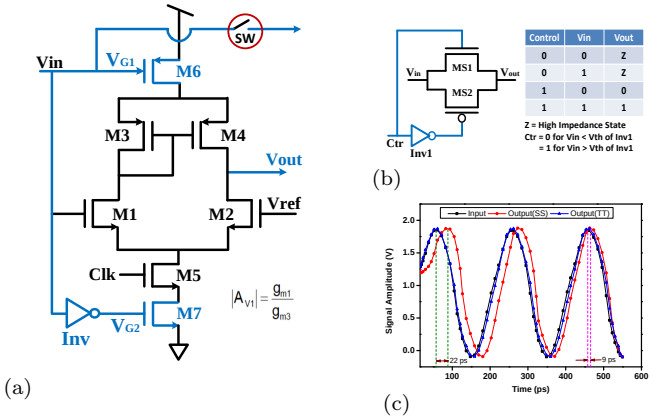


Figure 7.6: (a) Preamplifier with controlled analog switch (SW) for threshold detection and bypass the analog signal (b) MOS architecture of switch with control pin (c) Analog switch output response.

$$f_{3dB} = \frac{1}{2\pi \times R_{Load} \times C_{Load}} \tag{7.3}$$

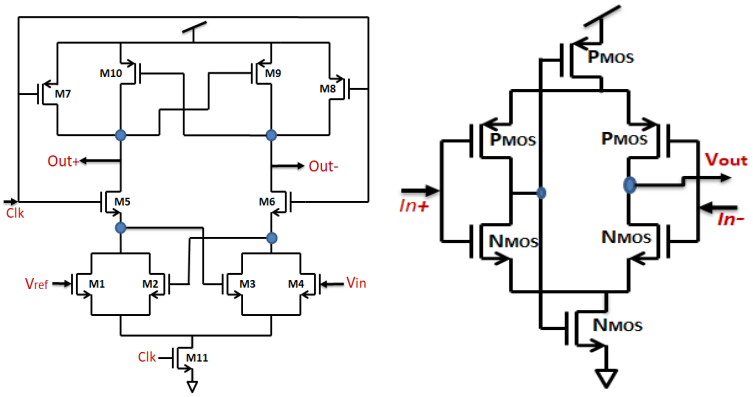
Where R_{Load} and C_{Load} are the resistive and capacitive load for the amplifier, respectively; moreover, Eq. 7.3 shows with an increase of amplifier bandwidth, the value of R_{Load} must be reduced. The preamplifier is often used to relax the effects of comparator input offset voltage for better matching and metastability, but it increases the total power consumption. In addition, the parasitic input capacitance by the preamplifier remains a bottleneck for high-speed and low-power applications.

Addressing the above problems, application-specific users can bypass the preamplifier stage. To do this, two important design aspects are considered. First is the proposed dynamic latch, and another is application-dependent reconfigurability using an analog switch. The detailed working flow-chart of a reconfigurable differential comparator is shown in Fig. 7.1. The following section gives a detailed description of the reconfigurable comparator.

7.3.3 Configurable design for with and without preamplifier in multi-stage dynamic comparator

Figure 7.6(a) shows the preamplifier circuit with a controlled analog switch which achieved the sampling rate of 2.4GS/s. Depending on the input analog signal and user applications, it contains two steps. If input signal $V_{in} \leq (\mid V_{th} \mid)_{M_6,M_7}$ then preamplifier stage is active in differential amplifier. The transistors with linear tunable transconductance, $M6$ and $M7$ sizes are identical, and V_{G1} and V_{G2} are at fixed switching voltage. However, channel length will decide the switching potential. The MOS level architecture of the switch to bypass the input signal is shown in Figure 7.6(b), input is passed through both PMOS and NMOS to the output terminal, and its switching control by the gate terminal.

The analog switch is used to bypass the input signal, and the concept behind this advancement is to save the power consumed by the preamplifier stage. If an input signal is greater than V_{th} of



(a) (b)

Figure 7.7: (a) Proposed dynamic latch (b) Post-amplifier circuit.

$M6$ and $M7$, mask the preamplifier stage and bypass the input analog signal. We have simulated switch circuits at different process corners for the worst delay and power. The worst-case delay is calculated for the slow-slow (SS) process corner, and in the worst case, power consumption in the fast-fast (FF) corner. However, the switch output response and delay at the worst process corner (mismatch) are shown in Fig. 7.6(c). The maximum delay generated by the MOS-based switch at the output for the first pick and a third pick is at the worse corner (SS) is $22ps$ and $9ps$ respectively. Thus, settling time to within 2% is the T_s=$4/\zeta\omega_n$ where, ζ is damping ratio =1 and natural frequency ω_n, and it gets at $550ps$. The observation can ensure the zero switching delay up to 2.4 GS/s sampling frequency at a worse corner.

7.4 Design Proposed Dynamic Latch and Post-Amplifier (Buffer)

The behavior of a first-order system having a single dominant pole shows a stable phase margin without complicated frequency compensation. Self-bias circuit automatically generates bias voltages to sustain performance over a wide supply voltage range. The circuit is designed and tested with post-layout simulation for the $1.8V \pm 10\%$ supply. The three-stage comparator consists of a preamplifier followed by the proposed dynamic latch and post-amplifier. A proposed dynamic latch follows the preamplifier with good offset cancellation and driving strength. The proposed circuit is implemented to differentiate the signals without a preamplifier, which is why configurability. In order to maintain the stable DC operating point, the transconductance at the input pair has to increase with equal proportionality. It implies raising current and power consumption. The differential comparator without a preamplifier will say a two-stage comparator.

The two-stage circuit consists of a proposed dynamic latch followed by post-amplifier and architecture as shown in Fig. 7.7(a) and 7.7(b), respectively. The proposed dynamic latch design contributes to the overall gain of a comparator. The proposed dynamic latch has an excellent driving strength with a high voltage gain. For the analog input $V_{in} \geqslant 0.5V$, the performance of

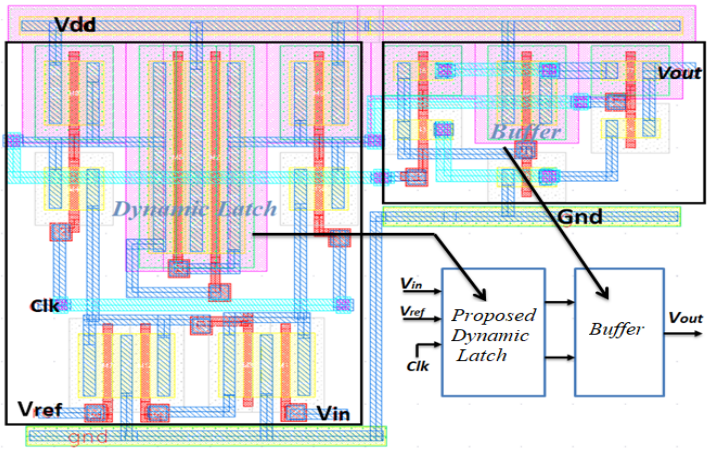


Figure 7.8: Layout for the two-stage comparator: proposed dynamic latch followed by post-amplifier.

the comparator is as good as the three-stage comparator at the given technology node. Using a two-stages comparator to reduce overall power consumption and lower input offset voltage improves the gain for better matching. The two-stage comparator circuit layout, which is proposed dynamic latch followed by post-amplifier, is shown in Figure 7.8 while it has circuit RC delay. The total propagation delay (t_P) of a differential amplifier depends on the number of stages used, and it is calculated as:

$$t_P = t_{Latch} + t_{Preamp} \tag{7.4}$$

$$t_{Latch} = \frac{C_{Load}}{g_m}\ ln\left(\frac{\Delta Vout}{\Delta Vin}\right) \tag{7.5}$$

Where ΔVin, $\Delta Vout$, C_L and g_m are the differential input voltage, output voltage, load capacitance, and transconductance of latch respectively.

The overall delay contribution by the differential comparator in real-time performance is shown in Eq. 7.4. However, we focused in this paper on the latch, and the delay contribution by the latch is shown in Eq. 7.5. We have designed a dynamic latch with the minimum area and less delay by considering the delay parameters. The design principle and operation of a proposed differential latch are operated in the preset and regeneration phase. During the preset/equalization phase, when the clock is low, $M5, M6, M9$ & $M10$ are in cut-off mode and $M7$ & $M8$ are in ON state. The drain of $M9$ & $M10$ transistors will charge towards V_{dd} and $M7$ & $M8$ is an equalize the differential output. Therefore, its output does not change in the equalization phase. When the clock is low, the output of the latch does not evaluate the input change. The MOS $M9$ & $M10$ are majorly responsible for the parasitic capacitance, limiting the bandwidth.

While current starts to flow during the regeneration phase, i.e., the clock is high, the drain of $M5$ & $M6$ starts to discharge. The $M5$ & $M6$ are then used to pass the differential voltage from the input node to the regenerating stage. The cross-coupled transistors start to regenerate the differential voltage as $M5$ & $M6$ can't claim the output to the ground. Hence the output node $Out+$ and $Out-$ is discharged toward the ground. The input and reference voltage is connected to the $M1 - M4$ in the linear region and acts like voltage-controlled registers. Therefore, as the clock

Table 7.1: Parameter Specification of the Proposed Dynamic Latch

Process File	**scl 180nm**
Power Supply	V_{dd} = 1.8V, V_{ss} = 0V
MOS Sizing	(W/L) $M_{1,2,3,4} = (1.5\mu/0.180\mu)$
	(W/L)$M_{5,6} = (1.0\mu/0.180\mu)$
	(W/L)$M_{9,10} = (4.5\mu/0.180\mu)$
Reference Voltage	V_{ref} = 0 to 1.8V
Clock Signal (CLK)	High = 1.8V; Low = 0V
	Rise and Fall time = 10ps
	Speed = 2.4GHz
Switch (PMOS)	(W/L)$M_{7,8} = (1.5\mu/0.180\mu)$

is high, we will get differential cross-coupled pair $M1 - M4$ connecting the input differential signal to the output terminal. The $M9$ & $M10$ are for boosting magnitude towards the rail voltages, and $M11$ isolates the ground path when the clock is low *i.e.* in the preset phase. The coupled pair $M1 - M2$ and $M3 - M4$ gives the large input impedance. The large area of PMOS, i.e., $M9$ & $M10$ taken for the design, gives the low output impedance. The above property of the design supports an increase in the comparator's overall voltage gain.

The comparator metastability occurs when very small signals appear at the input of the comparator and close to the comparator decision point, which is reduced using the $M7$, $M8$ and $M2$,$M3$. Whereas $M2$ and $M3$ MOS support offset compensation and ground path during regeneration. These MOSFETs support a comparison of the differential signal at the input. The small-signal (mV) needs low noise sensitivity and high amplification gain, which is difficult to achieve on the same path. The three-stage comparator has high input impedance and very little variation at the reference voltage generation point compared to the two-stage comparator. The second stage is the post-amplifier used to increase the slew rate. Overall, the proposed design increases the gain of the comparator and minimizes the error specifically, which gives the logic signal (i.e., 0 or 1.8V) from the output of the decision circuit. The output stage buffer/post-amplifier should accept a differential input signal and not have slew rate limitations for high-speed performance. The MOS width-length fixed for design have reported in Table 7.1 The basic architecture of post-amplifier used for the design is shown in Fig. 7.7(b). The layout for the two-stage comparator in cascade connection is shown in Fig. 7.8 whereas the post-layout simulated output waveform for a proposed circuit is shown in Fig. 7.9. The circuit operates at $2.4GS/s$ with tolerable delay, and the observed voltage gain is 100dB.

Offset Voltage Compensation

The input offset voltage in the comparator is due to its positive feedback and transient response. However, it is difficult to analytically predict the effect of offset. The fully configurable dynamic

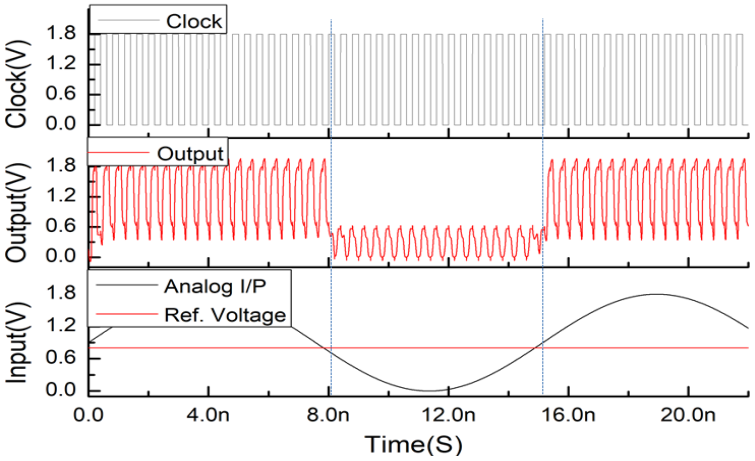


Figure 7.9: Post-layout simulation waveform for configurable differential Comparator for two-stage at clock $2.4 GS/s$.

comparator with no mismatch reached a balanced state. The time during transient Fig. 7.7 response is $V_{out+} = V_{out-}$ in the preset phase, and the voltage $\triangle V_{in}$ can be applied to the minimize the mismatch effect, and this voltage represents the input offset voltage. The mismatch and $\triangle V_{in}$ will affect the bias magnitude of the comparator. In order to calculate offset voltage, a balanced state need to be found to compensate for mismatch. The mismatch in threshold voltage and current through $M5$ & $M6$ are the dominant factors responsible for the process variation. First we have consider overall variation due to current factor β=$\mu C_{ox} W/L$ mismatch in transistor $M5$ & $M6$. However, $M2$ and $M3$ MOS is used in cross-manner for the compensation. The comparator to work at the balanced condition, $\triangle V_{in} = (V_{OS})_{M_5 M_6}$ which assure the offset compensation. The mismatch at the input is compensated using coupled transistor $M_{1,2,3,4}$. The offset voltage for M5 & M6 can be calculated as

$$(V_{OS})_{M_5 M_6} = \frac{(\mu_n + \triangle\mu_{M_5})(W_5/W_3)(V_{out+} - V_{s5} - V_{tn})^2}{2\mu_n V_{s5}} - \frac{(\mu_n + \triangle\mu_{M_6})(W_6/W_2)(V_{out-} - V_{s6} - V_{tn})^2}{2\mu_n V_{s6}}$$

Where V_{s5}, V_{s6} are the source node voltage. $\triangle\mu$ is the respective mobility variation during mismatch. μ and V_{tn} are nominal mobility and threshold voltage for NMOS.

7.5 Thermometer to Binary Encoder for Flash ADC

A different approach is used to convert thermometer code into gray or binary code. Bubble error is the major problem at high-speed conversion by the encoder, as shown in Fig. 7.2. The ROM-based decoder is proposed[144] which has a two-stage conversion. The ROM decoder-based architecture has a regular structure that is easy to design. However, more bubble errors are introduced with increasing conversion speed, and hence more advanced bubble error correction technique is required. The circuit counts the number of ones in the thermometer code as an encoder called ones-counter, and it gives complete bubble error suppression [145]. The Wallace tree topology [146] is proposed in, which uses a tree of full adders (FAs). The Wallace tree-based decoder can use for high-speed converters but consumes more power.

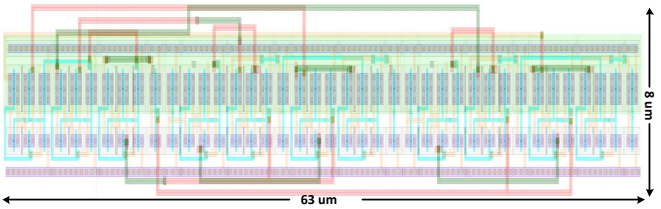


Figure 7.10: Implementation of Encoder with Different Technique

We have design a code transmission gate based encoder which takes the run time minimum delay and dynamic power. The proposed transmission gate based encoder and comparison with state of the art is shown in Fig. 7.10

7.6 Results and Discussion

Referring to the effectiveness of the Flash-type ADC using the proposed reconfigurable dynamic comparator, 180nm CMOS technology is used. All the simulations are performed considering 1.8V of supply voltage at $T = 27°C$ operating temperature unless specified.

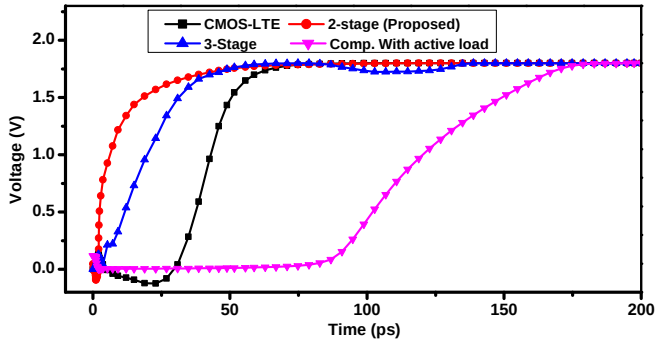
7.6.1 Comparator response and delay analysis of comparator

The proposed prototype can achieve competitive energy efficiency compared with other voltage domain comparators. The voltage response concerning time is demonstrated in Fig. 7.11 (a), which refers to the slew rate of a comparator. It helps us identify the maximum input frequency and amplitude applicable to the amplifier such that the output is not significantly distorted. The simulation results show the performance of different types of comparators at threshold crossing. The proposed dynamic latch area for M9 & M10 is more, as shown in Fig. 7.7 gives low output impedance. The comparator transient response simulation results give a distinct idea of the speed of the comparator. However, in line with reconfigurability views, the result demonstrates that the two-stage comparator provides better performance with low power and high resolution. The result shows that the two-stage comparator has a faster response than the state of the art.

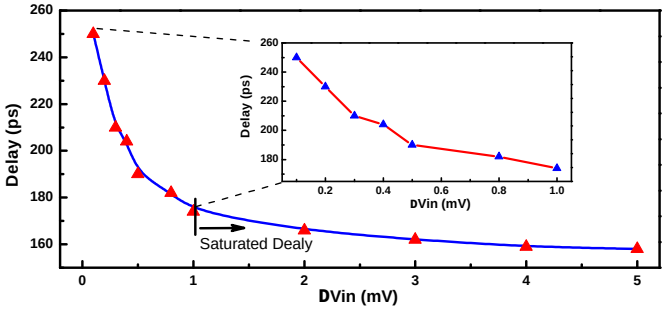
The proposed two-stage comparator is simulated, and measured the delay for the differential input voltage is shown in Fig.7.11(b). At the same time, it proves that the comparator's measured delay decreases with increasing the differential input voltage. Moreover, at a given V_{dd} supply, the larger differential voltage at the input provides a smaller comparator delay. The results also concluded that the proposed comparator delay is inversely proportional to the input voltage difference between the input node and reference node voltages. From the above discussions, we find that a higher input voltage difference results in a minor output delay and vice versa.

7.6.2 Circuit performance for PVT variation and comparison

The impact of the process, voltage, and temperature (PVT) variations on the circuit performance are important for stable circuit performance. The effect of PVT variations on the comparator



(a) Settling and slewing behavior response of proposed and state-of-art comparators during threshold detection

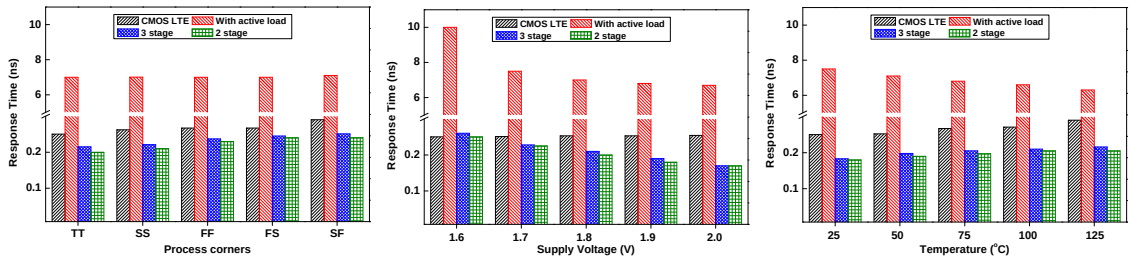


(b) Delay variation of proposed comparator with change in input voltage difference (ΔVin) i.e the reference node voltage and the input voltage.

Figure 7.11: Timing responce & voltage variation of proposed comparator and comparision with state-of-the-art.

response time is shown in Fig. 7.12. Fig. The response time of different considered circuits at the different process corners, which are slow-slow (SS), fast-fast (FF), Slow-Fast (SF), Fast-Slow (FS), and typical-typical (TT), is shown in Fig. 7.12(a). Results show that the response time of the proposed 2-stage comparator is less as compared to all other considered circuits for various process corners. The supply voltage of the design can vary from its fixed ideal value with day-to-day operation and environmental conditions. Moreover, the supply voltage is responsible for the saturation current and indirectly responsible for the signal propagation delay. The circuit response time with supply variation for circuit design is shown in Fig. 7.12 (b), and it observes that the response time decreases with increasing supply voltage for all the considered comparators.

The result shows that the effect of supply variation on the response time of the proposed 2-stage comparator is less as compared to all other considered circuits. The response time sensitivity with supply voltage is less for the proposed 2-stage comparator than for a comparator with active load and a 3-stage comparator. The temperature variation also affects the circuit performance mostly as a linear scaling effect at the 180nm silicon process. We have simulated the response behavior for the proposed circuit with the different operating temperatures at 1.8V supply voltage and TT process corner, as shown in Fig. 7.12 (c). The result shows that the response time increases with the increasing operating temperature. The change in response time with temperature for the proposed 2-stage comparator is less than the CMOS-LTE and 3-stage comparator.



(a) (b) (c)

Figure 7.12: Effect on comparator response time with the PVT variations (a) Response time with process Variation (b) Responce time with supply variation (c) Response time with temperature variation

7.6.3 Proposed circuit parameters and comparison

Table 7.2 summarizes the differential comparator performance matrics and compares them with state-of-the-art. The proposed comparator utilizes clock edges to generate voltage differences and directly latch this difference to the post-amplifier, which enhances the speed of the comparator. The hysteresis is the two different threshold voltages in the differential comparator observing rising and falling threshold detection. In a differential comparator for a very slowly varying input, output switching can be slow. For the calculation of hysteresis, the input signal must exceed the upper threshold (V_H) to transition low or below the lower threshold (V_L) to transition high. The power supply rejection ratio (PSRR) is the ability of a dynamic comparator to maintain its output voltage as its varying DC power supply. The PSRR is calculated using PSRR = (change in V_{dd})/(change in V_{out}). The input is driven with a voltage larger than the required and finds the proposed circuit's response time and state-of-the-art. The settling time is considered 90% of its saturation value, and the response time is to reach 100% of its saturation value. However, the response time depends on the value of the overdriven voltage for a given input amplitude.

Results demonstrate that the sensitivity of the proposed 2-stage comparator is significantly less compared to other considered comparators. The $0.13\mu V$ of sensitivity for the proposed 2-stage comparator provides less voltage difference to sense. The proposed comparator has 3.09× less overdrive recovery time than the CMOS-LTE comparator. The proposed 2-stage comparator also has the latching capability, which provides better performance of the circuit. Table ref shows the performance comparison of the different comparator circuits is shown in Table 7.3. The observation shows that the proposed two-stage comparator consumes less power than previous works, including a three-stage comparator. The power consumption by the proposed design is 83.15μW at $2.4GS/s$. The significant amount of power is reduced by bypassing the preamplifier stage in a cascaded connection using an analog switch that controls the input signal. The proposed two-stage comparator can operate faster and consume low power while using the reconfigurable technique; we can use a two-stage comparator and a preamplifier.

Table 7.2: Performance parameter metrics of differential comparators at 1.8 V Power Supply

Performance Parameter	**Comparators**			
	CMOS-LTE [3]	**with active load [2]**	**3-Stage**	**2-Stage**
Technology CMOS	180nm	180nm	180nm	180nm
Sensitivity (μV) @27^oC	1000	10	0.1	0.13
Overdrive Recovery Time (pS)	71	175	35	23
PSRR (dB) @100KHz	—	—	-43	-41
Hysteresis (mV)	15	85	10	15
Latching compatibility	No	Yes	Yes	Yes

Table 7.3: Performance comparison of various comparators at 180nm process at 1.8V with ±10% Supply Variation.

Comparators	**Max. Sampling Frequency** (GS/s)	**Average Power @500MHz** (μW)	**Voltage Gain** (dB)
CMOS-LTE [3]	1	560	60
Comparator with Active Load [2]	0.5	172	80
Double-tail Dynamic Comparator [147]	1.8	160	—
Three-stage Dynamic Comparator	2.4	554	100
Proposed Reconfigurable with Two-stage	2.4	83.15	100

7.6.4 Performance summary of 4-bit flash ADC

The total power consumption is 0.95mW at a 1.8V supply. The analog power, including the comparators and the sampling network, is 72mW, and the digital, including the encoder, is 0.23mW. The reference voltage for the comparison is generated using the MOS ladder. The performance summary of the circuit design is shown in Table. 7.4. The proposed architecture achieves circuits with low power and high-speed requirement. We have simulated the Flash ADC circuit using three stages and a Reconfigurable three-stage comparator individually. We have simulated the circuit at all PVT Variation and the mismatch under worse conditions. The post-layout simulation states that the proposed design architecture is suited for low-power applications without compromising the other parameters. We have designed a circuit at 180nm technology as we have the fabrication support from the foundry.

7.7 Summary

In this Chapter, a power-efficient, configurable dynamic comparator and 4-bit flash type ADC is designed using SCL 180 nm CMOS technology at the supply voltage of 1.8 V. The proposed run-time configurable differential amplifier circuit design operates at 2.4GS/s. It significantly reduces the circuit's active power by using the configurable technique. The proposed architec-

Table 7.4: ADC Performance summary

Physical Parameter	**Value**
Technology	$180nm$
supply Voltage	$1.8V$
Power Consumption @2.4GHz	$0.95mW$
Comp. offset Voltage (σ_{os})	$4.8mV$
Sampling Rate	$2.4\ GS/s$
Nominal Resolution	$4 - bits$
Nominal LSB Size	$112.5mV$

ture is divided into two-step algorithm implementation depending on the analog input to the circuit. Compared with previous state-of-the-art, the proposed architecture results are better and 83.2% power-efficient compared to the three-stage comparator. This reconfigurable technique with the proposed dynamic latch-based comparator gives very high-speed conversion with low power consumption; hence it is very useful in flash type ADC. The power-hungry preamplifier stage is bypassed in the multi-stage comparator for the higher input signal ($\geq V_{th}$), which results in huge power saving and does not require any kind of extra processing blocks. The results of the proposed circuit claim that the use of the reconfigurable technique will have the option of power-saving mode.

Further, 4-bit flash ADC with a reconfigurable technique in a modified dynamic latch-based comparator has been designed, which can be used in RRAM-based CNN applications needed high-speed conversion with low power consumption. ADC effectively works at a sampling rate of $2.4GS/s$. The average power consumption for comparator and total ADC architecture is $0.118mW$ and $0.95mW$, respectively, which is very less compared to other proposed techniques. The design shows better results in significantly reduced power consumption. Therefore, the proposed technique can achieve low power consumption without compromising speed compared to the other voltage domain comparator designs for analog conversion. Therefore, the proposed 2.4GS/s reconfigurable Flash ADC can be a significant choice where the input peak to peak voltage range is variable. The applications such as SRAM and RRAM-based Convolutional neural networks with analog computing techniques.

Chapter 8

Conclusion and Future Work

The presented DNN architecture reduces the hardware-resource requirements and on-chip power consumption without losing computational rate. Our architecture has struck a demand of resource overhead in a Deep Neural Network by exploiting the hardware-reused context. The semi-custom ASIC and FPGA approach is investigated for design and implementation. The proposed architectures have struck a demand of resource overhead in a Deep Neural Network by exploiting the hardware-reused context and novel computational units such as MAC and AF. The proposed block designs support a power gating technique that saves static power dissipation. The hardware design parameters such as signed/unsigned, arithmetic computation, bit-precision, and approximation limit have been extensively evaluated for efficient DNN hardware implementation through Pareto analysis.

The contribution of the work is twofold. Firstly, a novel MAC unit and nonlinear AF are designed using CORDIC iterative and pipelined architecture. The CORDIC-based design allows configuration, and the same block can compute both MAC and multiple activation functions. Additionally, the proposed design can compute both signed and unsigned computations. Further, using extensive evaluation, we show that our proposed design gives better returns at higher bit precision than the state-of-the-art. The pipeline architecture comes with an area overhead. Therefore, we explore the mutual exclusivity between CORDIC stages and conduct a detailed study of accuracy variation concerning # the number of stages required to achieve high throughput that shows 4 *to* 5 pipeline CORDIC stages are sufficient for achieving maximum accuracy in MAC and AF evaluation. Further, the sigmoid AF function design using Taylor series expansion shows better results without circuit configurability. The proposed configurable architecture for MAC and multiple activation functions i.e *ReLU*, *sigmoid*, *tanh*, *exponential* is simulated and synthesised at *4*5nm technology to further validate efficiency of proposed architecture in terms of performance parameters. The proposed design architecture is scalable for any precision. However, it shows better results for 8-bit and other higher bit-precision implementations.

Secondly, System-level Performance efficient DNN design architecture reported better performance. The design has explored the hardware-reused techniques within the DNN architecture with

insignificant throughput loss. The hardware-reused techniques have been investigated at different abstraction levels in the system. 1. Reused hardware resources to iteratively compute MAC and AF using the same hardware reported above. 2. The designed fully connected neural network and reused resources hungry AF within the neural network layer. The presented technique saved huge hardware resources with no throughput loss. Further, The efficient memory addressing scheme is used to overcome the SoC throughput limits. Thus, the embedded system design with lower hardware resources and power consumption can significantly benefit edge computing solutions. 3. Layer-multiplexed architecture presented that reused single neuron layer for any deeper neural network. This technique saves huge hardware resources but comes with low throughput. Therefore, pipeline MAC shows good results for area/power and throughput. In order to verify the behavior of the proposed architectures, the Xilinx FPGA platform is selected to implement the benchmark configuration and tested for MNIST classification. Thus, the embedded system design with lower hardware resources and power consumption can significantly benefit edge computing solutions. The experimental evaluation validates that the proposed enhanced performance layer-multiplexed solution will help us gain better area, power, and throughput performance for future system implementations in ASICs and FPGAs.

The proposed 4-bit flash ADC supports high-speed data conversion. In order to represent the integrated system accelerator for both analog and digital applications, an ADC interface is required. Furthermore, design has scope for In-memory computing applications since ADC architecture is low power and has efficient physical performance. The ADC design effectively works at a sampling rate of $2.4GS/s$ and is used in analog-digital interface accelerator. Further, proposed ADC architecture can be use in analog computing In-memory computing DNN accelerators and the importance of ADCs in IMC accelerators. The power-efficient design of flash ADC is presented with configurable power gated dynamic comparator.

Exploring our proposed DNN implementation method with a near-memory computing technique will be exciting. Besides, this article does not discuss some well-known machine learning algorithms like recurrent neural network (RNN), Long short-term memory (LSTM), Capsule Neural Network (CapsNet), etc.; nonetheless, we will explore them in future work. The CORDIC solution can be explored further for $softmax$ activation function implementation to compute complex exponential and division operations.

Limitations

We have introduced a DNN framework for low-power, area-efficient, and high throughput applications. However, there are some limitations since these parameters are counterparts. Therefore, a future course of action can be used to remove these limitations and further improve the quality of the proposed work. In this concern, some of the salient points are as follows:

1. In this thesis, we presented Pareto analysis in MAC which comes with arithmetic approximation. Moreover, architecture can further be investigated for data quantization and ap-

proximation at CORDIC primary computing elements to make it efficient for AI-enabled IoT applications.

2. This thesis presented the novel MAC architecture for digital design; Moreover, the CORDIC-based MAC design technique will further investigate in-memory computing.

3. The hardware reused DNN architecture can explore runtime programmable features that can dynamically configure iterative and pipeline MAC.

Finally, we can conclude that the intended objectives of designing and hardware implementing DNN accelerator with the functionally configurable processing element and nonlinear activation function architecture for edge computing applications have been successfully achieved.

Appendix A

Hardware features and utility

The targeted boards are one is the ZYBO Evaluation kit and another is Virtex-7 VC707 hardware kit is shown in Figure 8.1. The ZYBO development board has a hard core Zynq processor (PS) that allows the development of an integrated system with programmable logic(PL). It gives good features for PS-PL integration. The board has many hardware interfaces such as UART, SPI, and HDMI. Further, the board has sufficient hardware resources that are required for the hardware accelerator deployment. The Virtex-7 board is mostly preferable for the PL based acceleration. The board has huge hardware resources and can deploy a deeper neural net accelerator and apply parallel processing for high throughput applications. The board has more RAM (4GB) that we required to store the weights and input features of the DNN accelerator. Furthermore, it has a PCI express interface for high speed communication with large data width. The SoC do not have the hardcore processing system, however it can configure with Microblatz soft-core IP and program it through SDK development kit.

Hardware implementation flow for proposed design

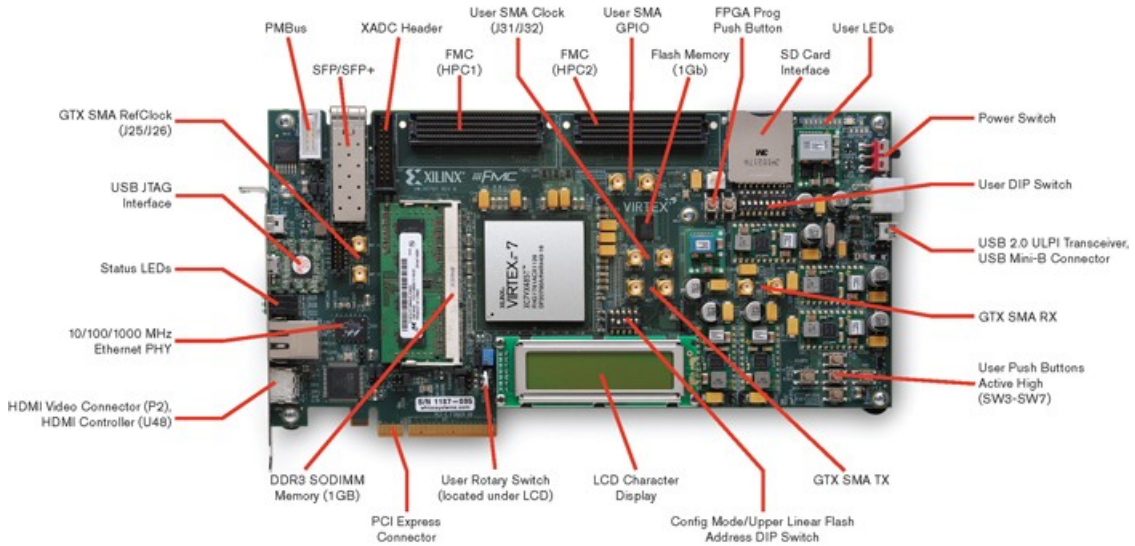
Figure 8.2 shows the FPGA design flow for hardware implementation and varification. At first the design has been realized and functionally verified using Behavioural Simulation. The simulation results of Proposed arhctiectures, i.e., MAC AF, fully connected neural network and convolutional neural network are provided with the flow of FPGA and ASIC implementation flow. The verified design was synthesized and resource utilization with other perforamcne parameters were reported. A successfully synthesized proposed architectures have been implemented where planning, placing and routing of the design Layout were performed. Once the design was implemented, Static Timing Analysis (STA) has been conducted. Here the max path delay of the comprehended architecture were documented. Accordingly the maximum frequency of operation is concluded.

Appendix B

Introduction to CORDIC Algorithm and Modes of Operation

The **CO**ordinate **R**otation **DI**gital **C**omputer (CORDIC) algorithm realizes various mathematical functions by rotating a vector. The underlying principle allows solving the trigonometric relationships involved in plane coordinate rotation and conversion from rectangular to polar coordinates [87]. The CORDIC architecture is configured to operate in three rotation modes – circular, linear, or hyperbolic rotation. In this connection, a unified algorithm for linear and hyperbolic CORDIC is an extension of the basic CORDIC algorithm for a circular trajectory as explained in

(a) Zybo xc7zclg400 evalautation board



(b) Virtex7 VC707 SoC

Figure 8.1: FPGA SoCs used for hardware implementation of DNN [4]

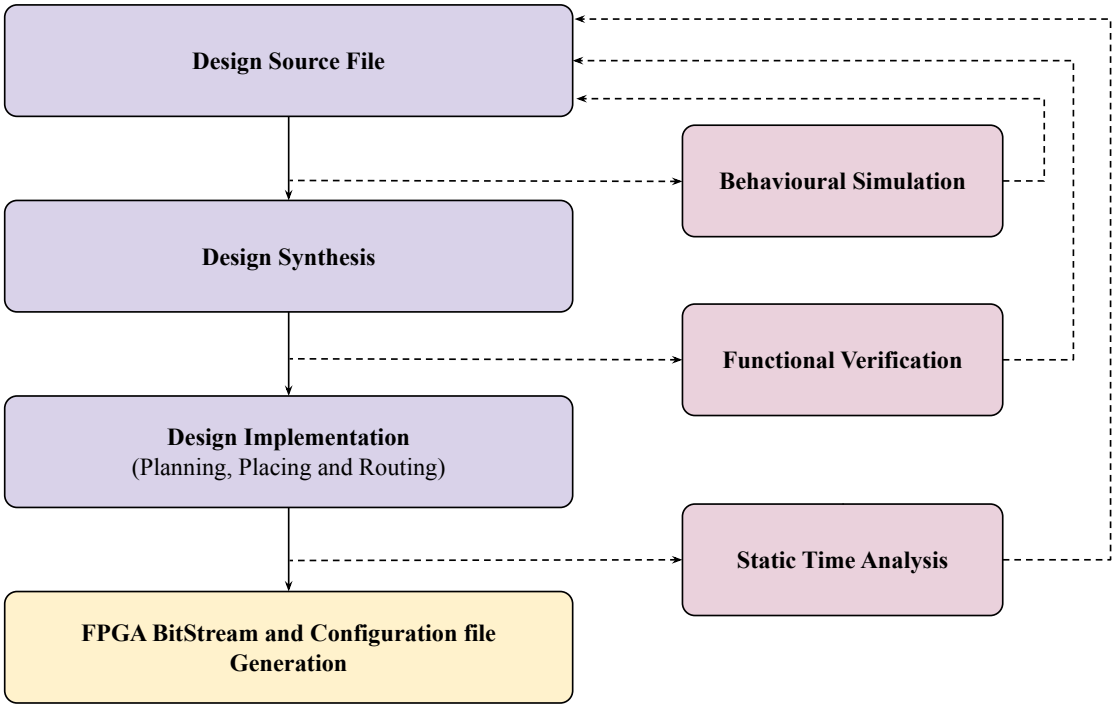


Figure 8.2: Design Flow for FPGA based Implementation

[148]. CORDIC algorithm is based on pseudo rotation shown in Figure 8.3, which is a scaled form of real rotation using variables X, Y, and Z. The pseudo-rotation is the key idea of the CORDIC circuit for computation and realization of mathematical functions. Based on the operation to be performed, these variables are initiated where X and Y represent coordinates of pseudo rotation and Z keeps track of the angle at which the vector is being rotated [18]. The scaling factor, K in pseudo rotation of vector is a product of K_i, where K_i is cos α_i and cosh α_i for Circular mode and Hyperbolic mode, respectively, and the product converges in i number of iterations [10]. The generalized CORDIC architecture for all modes of operation is shown in Figure 8.4

CORDIC modes of operation and linear equations convergence

The functions (including trigonometric, square root, hyperbolic and its inverse, and other basic mathematical calculations) can be performed by Rotational and Vectoring mode CORDIC algorithm in different planar coordinates as represented in Table 8.1. We have worked on the CORDIC Rotational plane includes circular, linear, and hyperbolic modes. The functional validation of the designed architecture is done for trigonometric and hyperbolic trigonometric.

In this connection, a unified algorithm for linear and hyperbolic CORDIC is an extension of the basic CORDIC algorithm for a circular trajectory as explained in [148]. The equations for hyperbolic coordinate computations includes a rotation matrix and is given in the equation below.

$$\begin{bmatrix} X_{(i+1)} \\ Y_{(i+1)} \end{bmatrix} = \begin{bmatrix} cosh\ \alpha_i & sinh\ \alpha_i \\ sinh\ \alpha_i & cosh\ \alpha_i \end{bmatrix} \begin{bmatrix} X_{(i)} \\ Y_{(i)} \end{bmatrix} \tag{8.1}$$

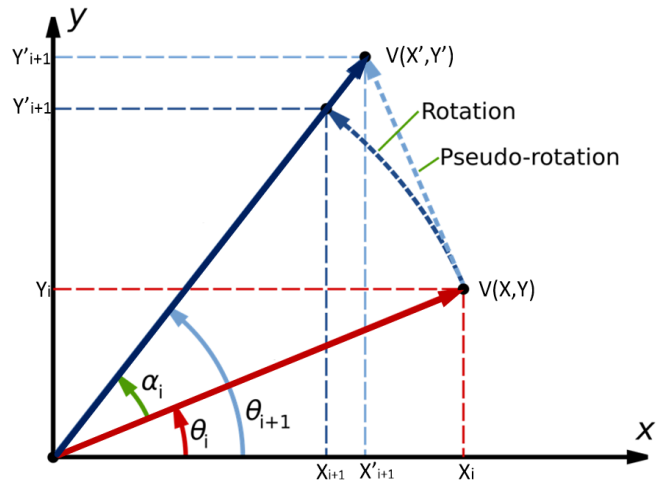


Figure 8.3: The circular-rotation and pseudo-rotation of a vector of length about an angle with the origin.

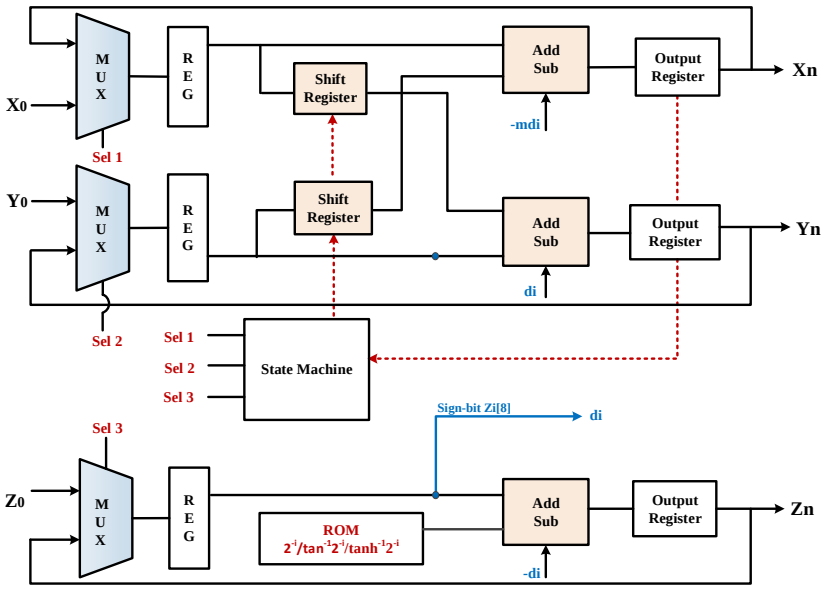


Figure 8.4: Signed 8-bit precision for the basic CORDIC-based design

Here, (X_i,Y_i) are set of coordinate components representing an i^{th} state. In terms of polar co-ordinates, an angle α is used where the new coordinates (X_{i+1},Y_{i+1}) can be easily reached. The above equations describes a rotation with scaling factor of the plane vector v_i to v_{i+1} at each iteration. In above equations if we take $cosh\ \alpha_i$ term as common (called as K_i scaling factor) in CORDIC calculation, the equations for all mode of trajectories are thus formulated in unified CORDIC algorithm as:

$$\begin{bmatrix} X_{(i+1)} \\ Y_{(i+1)} \end{bmatrix} = K_i \cdot \begin{bmatrix} 1 & -m \cdot d_i \cdot 2^{-i} \\ d_i \cdot 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} X_{(i)} \\ Y_{(i)} \end{bmatrix} \tag{8.2}$$

where $\alpha_i = 2^{-i}$, $tan^{-1}(2^{-i})$ or $tanh^{-1}(2^{-i})$

Here mode $m \in \{1, 0, -1\}$ indicates a circular, linear, and hyperbolic coordinate system, respect-

Table 8.1: Generalized CORDIC Algorithm

m	Rotational	Vector
Circular 1	$X_n = K_c(X_0 cosZ_0 - Y_0 sinZ_0)$ $Y_n = K_c(Y_0 cosZ_0 + X_0 sinZ_0)$ $Z_n = 0$	$X_n = K_c\sqrt{X_0^2 + Y_0^2}$ $Y_n = 0$ $Z_{0+1} = Z_0 + tan^{-1}(\frac{Y_0}{X_0})$
Linear 0	$X_n = X_0$ $Y_n = Y_0 + X_0 \cdot Z_0$ $Z_n = 0$	$X_n = X_0$ $Y_n = 0$ $Z_n = Z_0 + (\frac{Y_0}{X_0})$
Hyperbolic -1	$X_n = K_h(X_0 coshZ_0 - Y_0 sinhZ_0)$ $Y_n = K_h(Y_0 coshZ_0 + X_0 sinhZ_0)$ $Z_n = 0$	$X_n = K_h\sqrt{X_0^2 - Y_0^2}$ $Y_n = 0$ $Z_n = Z_0 + tanh^{-1}(\frac{Y_0}{X_0})$

ively. The d_i shows the rotation direction for i^{th} iteration. The α_i rotation angle in radians for the each iteration. K_i is scaling factor and it is different for different mode of operations (linear, circular or hyperbolic).

Bibliography

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] A. Rodríguez-Vázquez, R. Domínguez-Castro, F. Medeiro, and M. Delgado-Restituto, "High resolution cmos current comparators: Design and applications to current-mode function generation," *Analog integrated circuits and signal processing*, vol. 7, no. 2, pp. 149–165, 1995.

[3] M. Kulkarni, V. Sridhar, and G. H. Kulkarni, "4-bit flash analog to digital converter design using cmos-lte comparator," in *2010 IEEE Asia Pacific Conference on Circuits and Systems*. IEEE, 2010, pp. 772–775.

[4] *https://www.xilinx.com/products/boards-and-kits/see-all-evaluation-boards.html.*

[5] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015.

[6] S. J. V. Rani and P. Kanagasabapathy, "Multilayer perceptron neural network architecture using vhdl with combinational logic sigmoid function," in *2007 International Conference on Signal Processing, Communications and Networking*. IEEE, 2007, pp. 404–409.

[7] *Xilinx LogiCORE IP v12.0 https://www.xilinx.com/support/ documentation/ip documentation/mult gen/v12 0/pg108-mult-gen.pdf.*

[8] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[9] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "Fpgas in industrial control applications," *IEEE Transactions on Industrial informatics*, vol. 7, no. 2, pp. 224–243, 2011.

[10] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "A CORDIC based configurable activation function for ANN applications," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 78–83.

[11] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *Computer*, vol. 40, no. 3, 2007.

[12] E. Wu, X. Zhang, D. Berman, I. Cho, and J. Thendean, "Compute-efficient neural-network acceleration," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 191–200.

[13] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPR 2011 workshops*. IEEE, 2011, pp. 109–116.

[14] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[15] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[16] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. 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] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 880–888, 2007.

[18] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "Recon: Resource-efficient cordic-based neuron architecture," *IEEE Open Journal of Circuits and Systems (OJCAS)*, vol. 2, pp. 170–181, 2021. [Online]. Available: doi:10.1109/OJCAS.2020.3042743

[19] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "An efficient approach for neural network architecture," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2018, pp. 745–748.

[20] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[21] G. Raut, A. Biasizzo, N. Dhakad, N. Gupta, G. Papa, and S. K. Vishvakarma, "Data multiplexed and hardware reused architecture for deep neural network accelerator," *Neurocomputing*, vol. 486, pp. 147–159, 2022.

[22] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J.-s. Seo, "Alamo: Fpga acceleration of deep learning algorithms with a modularized rtl compiler," *Integration*, vol. 62, pp. 14–23, 2018.

[23] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. 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, pp. 161–170.

[24] K. Khalil, O. Eldash, B. Dey, A. Kumar, and M. Bayoumi, "A novel reconfigurable hardware architecture of neural network," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2019, pp. 618–621.

[25] J. G. Oliveira, R. L. Moreno, O. de Oliveira Dutra, and T. C. Pimenta, "Implementation of a reconfigurable neural network in fpga," in *2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS)*. IEEE, 2017, pp. 41–44.

[26] X. Zhou, S. Li, F. Tang, S. Hu, Z. Lin, and L. Zhang, "Danoc: An efficient algorithm and hardware codesign of deep neural networks on chip," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3176–3187, 2017.

[27] J. Park, T. Yu, S. Joshi, C. Maier, and G. Cauwenberghs, "Hierarchical address event routing for reconfigurable large-scale neuromorphic systems," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2408–2422, 2016.

[28] S. M. Nabavinejad, M. Baharloo, K.-C. Chen, M. Palesi, T. Kogel, and M. Ebrahimi, "An overview of efficient interconnection networks for deep neural network accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 268–282, 2020.

[29] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[30] M. Eisele, J. Berthold, D. Schmitt-Landsiedel, and R. Mahnkopf, "The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 360–368, 1997.

[31] V. A. Pedroni, *Circuit design with VHDL*. MIT press, 2020.

[32] K.-L. Du and M. Swamy, "Neural network circuits and parallel implementations," in *Neural Networks and Statistical Learning*. Springer, 2019, pp. 829–851.

[33] H. Wong, V. Betz, and J. 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*, 2011, pp. 5–14.

[34] F. Yu, L. Liu, L. Xiao, K. Li, and S. Cai, "A robust and fixed-time zeroing neural dynamics for computing time-variant nonlinear equation using a novel nonlinear activation function," *Neurocomputing*, vol. 350, pp. 108–116, 2019.

[35] G. Raut, V. Bhartiy, G. Rajput, S. Khan, A. Beohar, and S. K. Vishvakarma, "Efficient low-precision cordic algorithm for hardware implementation of artificial neural network," in *International Symposium on VLSI Design and Test*. Springer, 2019, pp. 321–333.

[36] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[37] I. Tsmots, O. Skorokhoda, and V. Rabyk, "Hardware implementation of sigmoid activation functions using fpga," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*. IEEE, 2019, pp. 34–38.

[38] I. del Campo, R. Finker, J. Echanobe, and K. Basterretxea, "Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons," *Electronics Letters*, vol. 49, no. 25, pp. 1598–1600, 2013.

[39] T. Yang, Y. Wei, Z. Tu, H. Zeng, M. A. Kinsy, N. Zheng, and P. Ren, "Design space exploration of neural network activation function circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1974–1978, 2018.

[40] K. 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.

[41] S. Gomar, M. Mirhassani, and M. 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] V. Saichand, S. Arumugam, N. Mohankumar *et al.*, "Fpga realization of activation function for artificial neural networks," in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, vol. 3. IEEE, 2008, pp. 159–164.

[43] G. Rajput, G. Raut, M. Chandra, and S. K. Vishvakarma, "VLSI implementation of transcendental function hyperbolic tangent for deep neural network accelerators," *Microprocessors and Microsystems*, vol. 84, p. 104270, 2021.

[44] K. Leboeuf, A. H. Namin, R. Muscedere, H. Wu, and M. Ahmadi, "High speed VLSI implementation of the hyperbolic tangent sigmoid function," in *2008 Third international conference on convergence and hybrid information technology*, vol. 1. IEEE, 2008, pp. 1070–1073.

[45] G. Baccelli, D. Stathis, A. Hemani, and M. Martina, "NACU: a non-linear arithmetic unit for neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[46] A. Armato, L. Fanucci, E. P. Scilingo, and D. De Rossi, "Low-error digital hardware implementation of artificial neuron activation functions and their derivative," *Microprocessors and Microsystems*, vol. 35, no. 6, pp. 557–567, 2011.

[47] M. Alçın, İ. Pehlivan, and İ. Koyuncu, "Hardware design and implementation of a novel ann-based chaotic generator in fpga," *Optik*, vol. 127, no. 13, pp. 5500–5505, 2016.

[48] M. Wedlake and H. L. Kwok, "A cordic implementation of a digital artificial neuron," in *1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997*, vol. 2. IEEE, 1997, pp. 798–801.

[49] R. Pottathuparambil and R. Sass, "An fpga-based neural network for computer vision applications," in *Proc. Workshop Comput. Vis. Low Power Reconfig. Architect.(FPL)*, 2011, pp. 1–7.

[50] V. Tiwari and N. Khare, "Hardware implementation of neural network with sigmoidal activation functions using cordic," *Microprocessors and Microsystems*, vol. 39, no. 6, pp. 373–381, 2015.

[51] N. Sorokin, "Implementation of high-speed fixed-point dividers on fpga," *Journal of Computer Science & Technology*, vol. 6, 2006.

[52] S. J. V. Rani and P. Kanagasabapathy, "Multilayer perceptron neural network architecture using vhdl with combinational logic sigmoid function," in *2007 International Conference on Signal Processing, Communications and Networking*. IEEE, 2007, pp. 404–409.

[53] B. Zamanlooy and M. Mirhassani, "Efficient vlsi implementation of neural networks with hyperbolic tangent activation function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 39–48, 2013.

[54] C.-H. Chang, H.-Y. Kao, and S.-H. Huang, "Hardware implementation for multiple activation functions," in *International Conference on Consumer Electronics-Taiwan*. IEEE, 2019.

[55] M. Ercegovac, D. Kirovski, and M. Potkonjak, "Low-power behavioral synthesis optimization using multiple precision arithmetic," in *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. IEEE, 1999, pp. 568–573.

[56] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[57] D. Nguyen, D. Kim, and J. Lee, "Double mac: Doubling the performance of convolutional neural networks on modern fpgas," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 890–893.

[58] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[59] A. Arthurs, J. Roark, and J. Di, "Ultra-low voltage digital circuit design: A comparative study," in *2012 IEEE Faible Tension Faible Consommation*. IEEE, 2012, pp. 1–4.

[60] M. Yuvaraj, B. J. Kailath, and N. Bhaskhar, "Design of optimized mac unit using integrated vedic multiplier," in *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*. IEEE, 2017, pp. 1–6.

[61] M. S. Kumar, D. A. Kumar, and P. Samundiswary, "Design and performance analysis of multiply-accumulate (mac) unit," in *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*. IEEE, 2014, pp. 1084–1089.

[62] G. B. Joseph and R. Devanathan, "Algorithms for multiplierless multiple constant multiplication in online arithmetic," *Circuits, Systems, and Signal Processing*, vol. 37, no. 11, pp. 5127–5142, 2018.

[63] N. M. Kumar, G. Saravanan, D. S. Ganesh, and S. Kanimozi, "An efficient design of 16 bit mac unit using vedic mathematics," *International Journal*, vol. 1, no. 1, pp. 1–4, 2021.

[64] W. Xu, Z. Zhang, X. You, and C. Zhang, "Efficient deep convolutional neural networks accelerator without multiplication and retraining," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 1100–1104.

[65] D. A. Gudovskiy and L. Rigazio, "Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks," *arXiv preprint arXiv:1706.02393*, 2017.

[66] H. Kim, Q. Chen, T. Yoo, T. T.-H. Kim, and B. Kim, "A 1-16b precision reconfigurable digital in-memory computing macro featuring column-mac architecture and bit-serial computation," in *ESSCIRC 2019-IEEE 45th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2019, pp. 345–348.

[67] A. S. Abraham and S. Anand, "An asic design of an optimized multiplication using twin precision," in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2017, pp. 455–461.

[68] T.-Y. Sung and H.-C. Hsin, "Fixed-point error analysis of cordic arithmetic for special-purpose signal processors," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 90, no. 9, pp. 2006–2013, 2007.

[69] "https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html."

[70] "https://communities.mentor.com/docs/doc-3114."

[71] "https://www.cadence.com/en_us/home/tools/custom-ic-analog-rf-design/circuit-design/virtuoso-schematic-editor.html."

[72] Y. Ni, W. Chen, W. Cui, Y. Zhou, and K. Qiu, "Power optimization through peripheral circuit reusing integrated with loop tiling for rram crossbar-based cnn," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1183–1186.

[73] A. S. K. Vamsi and S. Ramesh, "An efficient design of 16 bit mac unit using vedic mathematics," in *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019, pp. 0319–0322.

[74] H. Zhang, D. Chen, and S.-B. Ko, "New flexible multiple-precision multiply-accumulate unit for deep neural network training and inference," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 26–38, 2019.

[75] A. S. K. Vamsi and S. Ramesh, "An efficient design of 16 bit mac unit using vedic mathematics," in *International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019.

[76] F. U. D. Farrukh, C. Zhang *et al.*, "Power efficient tiny yolo cnn using reduced hardware resources based on booth multiplier and wallace tree adders," *IEEE Open Journal of Circuits and Systems*, 2020.

[77] R. B. S. Kesava *et al.*, "Low power and area efficient wallace tree multiplier using carry select adder with binary to excess-1 converter," in *Conference on Advances in Signal Processing (CASP)*. IEEE, 2016.

[78] T. Sato and T. Ukezono, "A dynamically configurable approximate array multiplier with exact mode," in *2020 5th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2020, pp. 917–921.

[79] H. Jiang, C. Liu, F. Lombardi, and J. Han, "Low-power approximate unsigned multipliers with configurable error recovery," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 189–202, 2018.

[80] N. Van Toan and J.-G. Lee, "Fpga-based multi-level approximate multipliers for high-performance error-resilient applications," *IEEE Access*, vol. 8, pp. 25 481–25 497, 2020.

[81] J. Garland and D. Gregg, "Low complexity multiply-accumulate units for convolutional neural networks with weight-sharing," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2018.

[82] K. Yugandhar, V. G. Raja, M. Tejkumar, and D. Siva, "High performance array multiplier using reversible logic structure," in *International conference on current trends towards converging technologies (ICCTCT)*. IEEE, 2018.

[83] V. Mrazek, L. Sekanina, and Z. Vasicek, "Libraries of approximate circuits: Automated design and application in cnn accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2020.

[84] Y. Umuroglu, N. J. Fraser *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[85] L. Mei, M. Dandekar *et al.*, "Sub-word parallel precision-scalable mac engines for efficient embedded dnn inference," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019.

[86] "Iso/iec/ieee international standard - floating-point arithmetic," *ISO/IEC 60559:2020(E) IEEE Std 754-2019*, 2020.

[87] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[88] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, spring joint computer conference*, 1971, pp. 379–385.

[89] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "A cordic based configurable activation function for ann applications," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 78–83.

[90] H. D. Tiwari, G. Gankhuyag, C. M. Kim, and Y. B. Cho, "Multiplier design based on ancient indian vedic mathematics," in *2008 International SoC Design Conference*, vol. 2. IEEE, 2008, pp. II–65.

[91] L. Prono, A. Marchioni, M. Mangia, F. Pareschi, R. Rovatti, and G. Setti, "A high-level implementation framework for non-recurrent artificial neural networks on fpga," in *2019 15th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*. IEEE, 2019, pp. 77–80.

[92] H. Kim, Q. Chen, T. Yoo, T. T.-H. Kim, and B. Kim, "A 1-16b precision reconfigurable digital in-memory computing macro featuring column-mac architecture and bit-serial computation," in *ESSCIRC 2019-IEEE 45th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2019, pp. 345–348.

[93] H. Chhajed, G. Raut, N. Dhakad, S. Vishwakarma, and S. K. Vishvakarma, "Bitmac: Bit-serial computation-based efficient multiply-accumulate unit for dnn accelerator," *Circuits, Systems, and Signal Processing*, pp. 1–16, 2022.

[94] P. C. Knag, *et al.*, "A 617-tops/w all-digital binary neural network accelerator in 10-nm finfet cmos," *IEEE Journal of Solid-State Circuits*, 2020.

[95] Z. Deng, C. Xu, Q. Cai, and P. Faraboschi, "Reduced-precision memory value approximation for deep learning," *Hewlett Packard Labs, HPL-2015-100*, 2015.

[96] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," *arXiv preprint arXiv:1511.05236*, 2015.

[97] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-performance accurate and approximate multipliers for fpga-based hardware accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[98] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, "Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[99] Z.-G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel, "Weight-oriented approximation for energy-efficient neural network inference accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4670–4683, 2020.

[100] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Cross-layer approximate computing: From logic to architectures," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[101] T. V. Huynh, "Deep neural network accelerator based on fpga," in *2017 4th NAFOSTED Conference on Information and Computer Science*. IEEE, 2017, pp. 254–257.

[102] H. Zhu, C. Chen, S. Liu, Q. Zou, M. Wang, L. Zhang, X. Zeng, and C.-J. R. Shi, "A communication-aware dnn accelerator on imagenet using in-memory entry-counting based algorithm-circuit-architecture co-design in 65-nm cmos," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 283–294, 2020.

[103] Y. Zhang, C. Hu, and B. Jiang, "Embedded atom neural network potentials: Efficient and accurate machine learning with a physically inspired representation," *The journal of physical chemistry letters*, vol. 10, no. 17, pp. 4962–4967, 2019.

[104] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[105] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[106] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[107] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[108] C. Zhang, H. Wang, J. Wen, and L. Peng, "Deeper siamese network with stronger feature representation for visual tracking," *IEEE Access*, vol. 8, pp. 119 094–119 104, 2020.

[109] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, and F. Berry, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, vol. 9, no. 4, pp. 113–116, 2017.

[110] Y. Qian, D. Wu, W. Bao, and P. Lorenz, "The internet of things for smart cities: Technologies and applications," *IEEE Network*, vol. 33, no. 2, pp. 4–5, 2019.

[111] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin *et al.*, "A hardware–software blueprint for flexible deep learning specialization," *IEEE Micro*, vol. 39, no. 5, pp. 8–16, 2019.

[112] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[113] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[114] J. Han, Z. Li, W. Zheng, and Y. Zhang, "Hardware implementation of spiking neural networks on fpga," *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 479–486, 2020.

[115] J. Skodzik, V. Altmann, B. Wagner, P. Danielis, and D. Timmermann, "A highly integrable fpga-based runtime-configurable multilayer perceptron," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2013, pp. 429–436.

[116] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *IEEE Micro*, vol. 33, no. 3, pp. 16–27, 2013.

[117] F. Ortega-Zamorano, J. M. Jerez, I. Gómez, and L. Franco, "Layer multiplexing fpga implementation for deep back-propagation learning," *Integrated Computer-Aided Engineering*, vol. 24, no. 2, pp. 171–185, 2017.

[118] T. Yang, T. Sato, and T. Ukezono, "An approximate multiply-accumulate unit with low power and reduced area," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 385–390.

[119] J. Gama, R. Sebastião, and P. P. 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.

[120] K. Sun, J. Zhang, C. Zhang, and J. Hu, "Generalized extreme learning machine autoencoder and a new deep neural network," *Neurocomputing*, vol. 230, pp. 374–381, 2017.

[121] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structure–activity relationships," *Journal of chemical information and modeling*, vol. 55, no. 2, pp. 263–274, 2015.

[122] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*. Springer, 2006, vol. 365.

[123] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019.

[124] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[125] J. Vreča, I. Ivanov, G. Papa, and A. Biasizzo, "Detecting network intrusion using binarized neural networks," in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 2021, pp. 1–6, in press.

[126] X. Fan, S. Zhang, and T. Gemmeke, "Approximation of transcendental functions with guaranteed algorithmic QoS by multilayer pareto optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2495–2508, 2020.

[127] F. Kästner, B. Janßen, F. Kautz, M. Hübner, and G. Corradi, "Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 154–161.

[128] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*. IEEE, 2013, pp. 1–6.

[129] V. Pejovic, "Towards approximate mobile computing," *CoRR*, vol. abs/1901.08972, 2019. [Online]. Available: http://arxiv.org/abs/1901.08972

[130] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "N2OC: Neural-network-on-chip architecture," in *32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2019, pp. 272–277.

[131] K. Khalil, O. Eldash, B. Dey, A. Kumar, and M. Bayoumi, "Architecture of a novel low-cost hardware neural network," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2020, pp. 1060–1063.

[132] U. Legat, A. Biasizzo, and F. Novak, "SEU recovery mechanism for SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 59, no. 5, pp. 2562–2571, 2012.

[133] H. Le, W. Jiang, and V. K. Prasanna, "Scalable high-throughput sram-based architecture for IP-lookup using FPGA," *International Conference on Field Programmable Logic and Applications*, pp. 137–142, 2008.

[134] N. Shah, P. Chaudhari, and K. Varghese, "Runtime programmable and memory bandwidth optimized FPGA-based coprocessor for deep convolutional neural network," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 5922–5934, 2018.

[135] A. Kundu, A. Heinecke, D. Kalamkar, S. Srinivasan, E. C. Qin, N. K. Mellempudi, D. Das, K. Banerjee, B. Kaul, and P. Dubey, "K-tanh: Efficient tanh for deep learning," *arXiv preprint arXiv:1909.07729*, 2019.

[136] K. Qiu, W. Chen, Y. Xu, L. Xia, Y. Wang, and Z. Shao, "A peripheral circuit reuse structure integrated with a retimed data flow for low power rram crossbar-based cnn," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1057–1062.

[137] J.-I. Kim, D.-R. Oh, D.-S. Jo, S.-T. Ryu *et al.*, "A 65 nm cmos 7b 2 gs/s 20.7 mw flash adc with cascaded latch interpolation," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 10, pp. 2319–2330, 2015.

[138] L. Wang, M.-A. LaCroix, and A. C. Carusone, "A 4-gs/s single channel reconfigurable folding flash adc for wireline applications in 16-nm finfet," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 12, pp. 1367–1371, 2017.

[139] A.-J. Annema, B. Nauta, R. Van Langevelde, and H. Tuinhout, "Analog circuits in ultra-deep-submicron cmos," *IEEE journal of solid-state circuits*, vol. 40, no. 1, pp. 132–143, 2005.

[140] L. Danial, N. Wainstein, S. Kraus, and S. Kvatinsky, "Breaking through the speed-power-accuracy tradeoff in adcs using a memristive neuromorphic architecture," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 396–409, 2018.

[141] A. Inamdar, A. Sahu, J. Ren, A. Dayalu, and D. Gupta, "Flash adc comparators and techniques for their evaluation," *IEEE transactions on applied superconductivity*, vol. 23, no. 3, pp. 1 400 308–1 400 308, 2013.

[142] B. Razavi, "The strongarm latch [a circuit for all seasons]," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 2, pp. 12–17, 2015.

[143] J. Yoo, "A tiq based flash a/d converter for system-on-chip applications. phd diss," Ph.D. dissertation, Ph. D. Thesis, The Pennsylvania State University, The Graduate School ..., 2003.

[144] Y.-J. Chuang, H.-H. Ou, and B.-D. Liu, "A novel bubble tolerant thermometer-to-binary encoder for flash a/d converter," in *2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005.(VLSI-TSA-DAT)*. IEEE, 2005, pp. 315–318.

[145] E. Sall and M. Vesterbacka, "Comparison of two thermometer-to-binary decoders for high-performance flash adcs," in *2005 NORCHIP*. IEEE, 2005, pp. 253–256.

[146] P. Pereira, J. R. Fernandes, and M. M. Silva, "Wallace tree encoding in folding and interpolation adcs," in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353)*, vol. 1. IEEE, 2002, pp. I–I.

[147] S. Babayan-Mashhadi and R. Lotfi, "Analysis and design of a low-voltage low-power double-tail comparator," *IEEE transactions on very large scale integration (vlsi) systems*, vol. 22, no. 2, pp. 343–352, 2013.

[148] T. Lang and E. Antelo, "Cordic-based computation of arccos and arcsin," in *Proceedings IEEE International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 1997, pp. 132–143.