Hardware-Efficient and Performace-Enhanced MAC unit for DNN Applications: A Quantization approach

MS (Research) Thesis

By

Ashar Neha Sunil Sonal

(2104102009)

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JULY 2023

Hardware-Efficient and Performace-Enhanced MAC unit for DNN Applications: A Quantization approach

A THESIS

Submitted in fulfillment of the
requirements for the award of the degree
of

Master of Science (Research)

by

Ashar Neha Sunil Sonal

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JULY 2023

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **"Hardware-Efficient and Performace-Enhanced MAC unit for DNN Applications: A Quantization approach"** in the fulfillment of the requirements for the award of the degree of **MASTER OF SCIENCE (RESEARCH)** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from August 2021 to July 2023 under the supervision of Prof. Santosh Kumar Vishvakarma, Professor, IIT Indore.

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

23-07-2023

Signature of the student with date
(ASHAR NEHA SUNIL SONAL)

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

24-07-2023

Signature of the Supervisor of
MS (Research) thesis #1 (with date)
(PROF. SANTOSH KUMAR VISHVAKARMA)

ASHAR NEHA SUNIL SONAL has successfully given his/her MS (Research) Oral Examination held on 26/03/24

M. Shaikh 26/3/2024
Signature of Chairperson (OEB) with date

26/03/2024
Signature(s) of Thesis Supervisor(s) with date

26/3/2024 (Acting DPGC).
Signature of Convener, DPGC with date

26/3/2024
Signature of Head of Discipline with date

ACKNOWLEDGEMENTS

I am immensely grateful to my MS research thesis supervisor and mentor, Prof. Santosh Kumar Vishvakarma, for consistently encouraging and supporting me in both my research and personal growth. His unwavering belief in my abilities and his invaluable guidance have served as constant motivation, pushing me to exceed my own limits. I owe him a debt of gratitude for granting me the freedom to explore my research interests and allowing my novel ideas to flourish. I would like to extend my gratitude to my manager and mentor Mr. Srinivas Jammula, Intel Technology India, to guide me and teach me throughout my internship duration of twelve months.

I would also like to extend my sincere appreciation to the members of my PSPC, Dr. Shaibal Mukherjee and Dr. Indrasen Singh, as well as the Thesis evaluation committee. Their impartial evaluations and thought-provoking questions have contributed significantly to expanding my research perspective. Further, my sincere thanks go out to Dr. Srivathsan Vasudevan and Dr. Trapti Jain, the coordinating faculties for the MS thesis, for their invaluable support.

My family has played a major role in supporting my research work throughout the course of my master's and my internship. They have always boosted my confidence and always motivated me to push my limits. I will always be grateful to them for all their guidance, love and sacrifices. Their faith in me has brought me this far, and it will drive me further, as well, to achieve greater things.

I deeply appreciate the Nanoscale Devices, VLSI Circuit System Design Lab (NSDCS) research group, especially Dr. Gopal Raut, Mr. Narendra Dhakad, Ms. Sumiran Mehra, Ms. Vasundhara Trivedi, Mr. Varun Bhatnagar, and Mr. Sandeep Chagara, for their continuous support and guidance. I am also grateful to my friends and labmates, Ms. Khushbu Lalwani, Mr. Avikshit Khomane, Mr. Mukul Lokhande, Mr. Ratnesh Tiwari, Mr. Arghya Singha Roy, Mr. Harshit Verma, Mr. Mradul Maurya, and Mr. Digamber Gaitonde, whose camaraderie and encouragement made my time at the institute truly memorable.

Neha Sunil Ashar

This Thesis is Dedicated

to

My Parents, My Sister, My Grandparents
and the Almighty God

ABSTRACT

Artificial Intelligence (AI) has witnessed remarkable growth, garnering significant attention from various industries and researchers due to its extensive applications. Researchers continuously explore different types of Deep Neural Networks (DNNs) and their efficient implementations using diverse techniques and datasets for various use cases, encompassing software-level applications to hardware-level systems. Edge-AI Systems-on-Chip (SoCs) is a focal point for industries, tailored to specific customer applications rather than generic CPU or GPU designs. As a result, these edge-AI devices are optimized to perform particular functions, allowing customization of hardware to focus on specific performance parameters. For instance, applications that require low power, small area, and high accuracy. The MAC unit plays a crucial role in the convolution operation, making its physical parameters vital for efficient overall design implementation. Typically, the MAC unit consists of a multiplier followed by an accumulator. The multiplier's bit-width is twice that of its input, and the accumulation stage further increases the bit-width. To ensure hardware implementation efficiency, roundoff/quantization is necessary at the output port of the MAC unit. However, this approximation process often requires an additional computing block, leading to resource overhead.

In light of this challenge, this thesis addresses quantization within the computation itself, eliminating the need for extra computational elements and offering a more efficient solution. Furthermore, it focuses on image classification applications and proposes an effective approach to reduce hardware requirements while maintaining accuracy levels. The modification of the Multiply-Accumulate (MAC) unit within a Deep Neural Network (DNN) neuron allows it to produce an approximate product with the same data size as the input bit-width. The proposed approach employs the right shift-and-add method and dynamic quantization for product generation and weight elimination, while a pipelined architecture minimizes the delay of the shift-and-add process. Moreover, the design is scalable to various bit-widths. The software implementation using the LeNet-5 network model with signed fixed-point

data arithmetic shows promising results. The accuracy loss observed with an 8-bit MAC implementation using MNIST and CIFAR-10 datasets is only 1.6% and 2.1%, respectively. On the hardware front, the design was evaluated on the Xilinx Virtex-7 board, revealing a notable 37% reduction in area and a 3.6% reduction in power consumption compared to conventional implementations. This research demonstrates a practical and efficient approach to enhance hardware efficiency without significant accuracy compromise in image classification tasks, opening new avenues for optimization in AI hardware systems.

Publications

- **Journal Publications:**

1. **Neha Ashar**, Gopal Raut, Vasundhara Trivedi, Santosh Kumar Vishvakarma and Akash Kumar, "QuantMAC: Enhancing Hardware Performance in DNNs with Quantize Enabled Multiply-Accumulate Unit", *IEEE Access* (Accepted).

2. Vasundhara Trivedi, Khushbu Lalwani, Gopal Raut, Avikshit Khomane, **Neha Ashar**, Santosh Kumar Vishvakarma, "Hybrid ADDer: A Viable Solution for Efficient Design of MAC in DNNs", *Circuits, Systems and Signal Processing*, 42(12), 7596-7614.

Contents

List of Figures

List of Tables

List of Abbrevations

DL	-	Deep Learning
AI	-	Artificial Intelligence
DNN	-	Deep Neural Network
ANN	-	Artificial Neural Network
RNN	-	Recurrent Neural Network
CNN	-	Convolutional Neural Network
MAC	-	Multiply and Accumulate
MLP	-	Multi-Layer Perceptron
ReLU	-	Rectified Linear Unit
RTL	-	Register Transfer Level
FIFO	-	First In First Out
AF	-	Activation Function
MNIST	-	Modified National Institute of Standards and Technology
CIFAR	-	Center for International Financial Analysis and Research
FPGA	-	Field Programmable Gate Array
ASIC	-	Application Specific Integrated Circuit
GDSII	-	Graphic Data System II
IC	-	Integrated Circuit
STA	-	Static Time Analysis
SRAM	-	Static Random Access Memory
PE	-	Processing Element
SoC	-	System on Chip
GPU	:	Graphical Processing Unit

Chapter 1

Introduction

1.1 Overview of Artificial Intelligence

In recent years, AI has garnered immense popularity and has become a transformative force in the modern world. The AI market is on the verge of significant expansion, driven by its broad range of applications, thereby intensifying research in machine learning and neural networks. Over the past two to three decades, this field has made remarkable progress, finding applications in diverse industries, from biomedical applications to speech recognition. The impressive growth of AI is facilitated by the abundance of processing power and data available, allowing machines to learn and improve themselves through machine learning algorithms without the need for explicit programming. These algorithms have the remarkable ability to predict future outcomes.

Artificial Intelligence, or AI, involves the simulation of human intelligence in machines to mimic human mental functions, such as learning and problem-solving. Moreover, the concept of rationalizing and acting with the best chance of achieving a specific goal is a desirable trait in AI. 'Machine learning,' a subset of AI, refers to computer programs learning from and adapting to new data without human intervention. Deep learning technology, a significant aspect of machine learning, automates learning by processing vast amounts of unstructured data, such as text, images, and videos. Contrary to the notion popularized by high-budget films and

novels, AI does not necessarily involve humanoid robots bent on destroying the Earth. Instead, it aims to replicate human cognitive skills, with researchers and developers making rapid strides in emulating learning, reasoning, and cognition. The ultimate goal is to achieve a level of specification where machines can perform tasks comparable to human capabilities. While some envision the creation of systems surpassing human capabilities in learning and reasoning, others remain skeptical, pointing out that all cognitive activity is rooted in human experience. As technology advances, the boundaries of artificial intelligence continue to evolve, with tasks like basic calculations and optical character recognition no longer considered exclusive to AI.

The field of AI embraces a multidisciplinary approach, integrating knowledge from mathematics, computer science, linguistics, psychology, and other disciplines. Within AI, deep learning stands as a pivotal technology, empowering systems to self-learn and develop autonomously without explicit programming. The process involves training DNNs over data, enabling them to identify patterns and make informed decisions based on examples for future tasks. The ultimate objective is for DNNs to learn independently, adapt their behaviour, and make decisions without human intervention. Deep neural networks consist of N layers that facilitate progressive data transformation through sequential information flow. Neural network architectures typically involve a substantial number of Multiply-Accumulate (MAC) operations. For example, LeNet, AlexNet, ResNet-50, and VGG-16 each require 0.34 million, 3.9 billion, 3.9 billion, and 15.5 billion MACs, respectively. Notably, deep neural networks depends on MAC computations and large activations.

This thesis delves into the realm of AI and machine learning, focusing on the efficient hardware implementation of DNNs and exploring their applications, advancements, and potential implications across various sectors. By comprehending the intricate interplay between human intelligence and AI, researchers aim to unlock new possibilities and equip machines with learning capabilities beyond our current comprehension. The continued evolution of AI and machine learning offers exciting prospects for enhancing efficiency and problem-solving capabilities across industries,

further transforming the modern world.

1.2 Introduction to Deep Learning and Deep Neural Networks

Deep learning, an influential subset of neural networks, stands out for its distinctive architecture, comprising three or more layers, including multiple hidden layers. The number of network layers employed in deep learning today spans a vast range, from 5 to well over 1,000. In this article, the focal point revolves around the term "Deep Neural Network" and its efficient hardware implementation. DNNs specifically refer to the neural networks extensively utilized in deep learning. Notably, DNNs possess the remarkable capacity to learn intricate and abstract higher-level functions, far exceeding the capabilities of their smaller counterparts. Based on the training of the network, DNNs adeptly recognize specific objects or scenarios, demonstrating remarkable performance across diverse applications. The DNN has more parallel MAC units, demanding more hardware resources for implementation. At the same time, FPGA hardware has limited resources for logic implementations. This problem will be more dominant when the network processes the high-resolution images that need higher precision computational elements. In previous works, multi-bit precision (8,16,32 and 64-bit) data representation is used for MAC unit implementation considering the trade-off between accuracy and physical performance parameters. Moreover, when some degree of error is tolerated, it is preferred to use fixed-point computation than floating-point computation due to its resources efficient architecture.

Crucially, DNNs are essentially variants of fundamental neural network architectures, including feed-forward neural networks and recurrent networks. These encompass Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), Transformer Networks, and Gated Recurrent Unit Networks (GRUs). DNNs consist of a higher number of hidden layers arranged in diverse configurations

to create distinctive models. While DNNs have gained industry fame due to their significant contributions to deep learning, it is important to underscore that their foundation lies in the broader domain of neural networks, encompassing various architectures like ANN, CNN, RNN, LSTM, Transformer, and GRU Networks. However, this thesis has specifically focused on feed-forward DNNs, namely ANN and CNN, to explore their efficient hardware implementation and applications in deep learning.

Neural networks are a class of machine learning models inspired by the structure and function of the human brain. They are designed to process complex data and learn patterns from it. Neural networks have gained tremendous popularity and success in various fields, including computer vision, natural language processing, speech recognition, and more. Thanks to neural networks and machine learning, the standard of living has been significantly elevated. Consequently, AI has proven to be a revolutionary technology that progressively matches human capabilities. Its versatility allows it to fit into a wide range of applications. Neural networks consist of the following components: an input layer, multiple hidden layers, and an output layer. Whereas each layer consists of multiple processing elements involving MAC units and AFs.

Neural networks are recognized as a replication of the intricate neuron mesh found inside the human brain. The brain performs a series of computations to achieve various tasks such as decision-making, comparison, prediction, classification, and detection. Similarly, artificial neurons are organized to construct a network that executes a similar series of mathematical operations, emulating the human brain in an artificial system. These neurons are arranged in layers, each operating in parallel. During training, each neuron is assigned weights, enabling the system to perform human-like actions, including classification. The concept of weighted inputs in neuron computations forms the basis of neural networks, where weighted sums correspond to synaptic scaling values that are accumulated by neurons. While the cascade of neurons involves simple linear algebraic operations, the neurons themselves apply functional operations to their combined inputs, resulting in a non-linear function.

Neurons generate output only when the input exceeds a fraction of the threshold, and the neural network employs a non-linear function representing the weighted sum of input values. Ultimately, as the weighted quantities are transferred from one or more hidden layers to the outgoing layer, the user obtains the final network output, termed activation or pressure, combining the brain-inspired analogy with the neural network. Computation entails assembling fundamental building blocks as follows.

1. **Neurons and Layers**: The foundation of a neural network comprises artificial neurons, also referred to as nodes or units, which are organized into layers. The neural network structure typically includes an input layer, one or more hidden layers, and an output layer. The connections between neurons in different layers are assigned weights that are adjusted during the training process to optimize the network's performance.

2. **Activation Function**: Each neuron in the neural network applies an activation function to the weighted sum of its inputs, generating an output that is then transmitted to the subsequent layer. Common activation functions used in neural networks include the sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax (used in the output layer for classification tasks).

1.2.1 Artificial Neural Networks: Brain-Inspired Parallel Computing

An ANN is a parallel computing model inspired by the human brain's functionality. Like the interconnected neurons in the brain, the ANN consists of numerous small processors connected through weighted connections. Figure 1.1 illustrates the fully connected neural network with a network size of *784:256:128:128:10*. The basic structure of an ANN consists of an input layer, several hidden layers, and an output layer, with interconnected artificial neurons within each layer. The network is designed to classify input images from the MNIST dataset to the corresponding output layer. In the ANN model, the processing nodes can be compared to the neurons in the human brain.

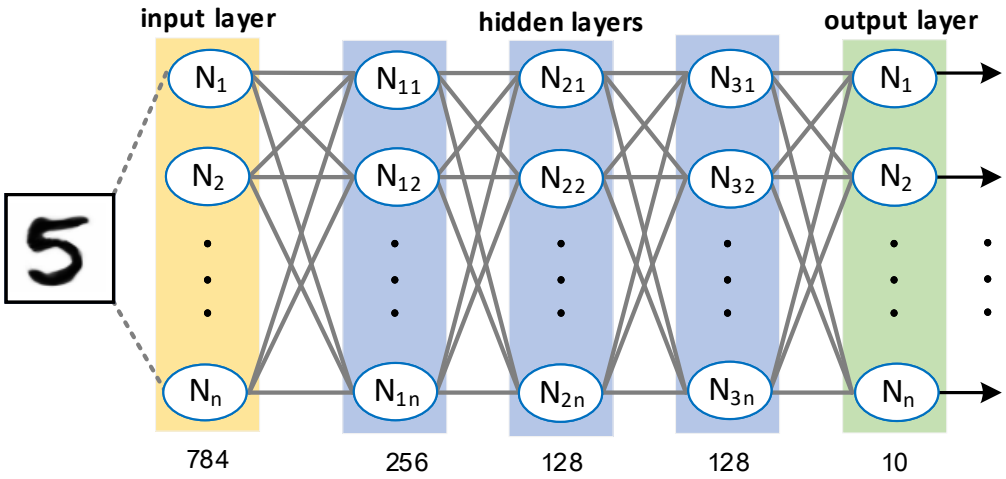

Figure 1.1: Fully connected five-layers (1-input, 3-hidden, and 1-output) artificial neural network architecture used for experimental results evaluations.

The basis of the ANN is the concept that a neuron's computation involves a weighted sum of its input values. These weighted sums are obtained after the network is trained, and they are combined within the neuron. The calculations involving a cascade of neurons can be represented as a linear algebraic process. However, neurons do not merely output the weighted total; instead, they apply a non-linear activation function to the combined inputs. This non-linear process causes neurons to produce an output only when the input surpasses a certain threshold. Consequently, artificial neurons employ non-linear functions on the sum of inputs and synaptic weights. Figure 1.2 depicts a single neuron, which serves as the fundamental building block of the ANN. The hardware implementation design of a single neuron involves a MAC unit, followed by an activation function. The figure illustrates multiple inputs, corresponding weights, and the bias associated with the neuron. Here, A_1, A_2, A_3 represent the inputs, C_1, C_2, C_3 are the weights, B denotes the bias, and Y represents the output of the neuron. Each layer in the ANN is independent and can have varying numbers of nodes or neurons, also known as bias nodes, which are consistently set to a value of 1. These bias nodes play a crucial role by providing the nodes with a constant value that can be adjusted during training. The inclusion of bias nodes is essential as they allow the activation function to be shifted to the right or left, contributing

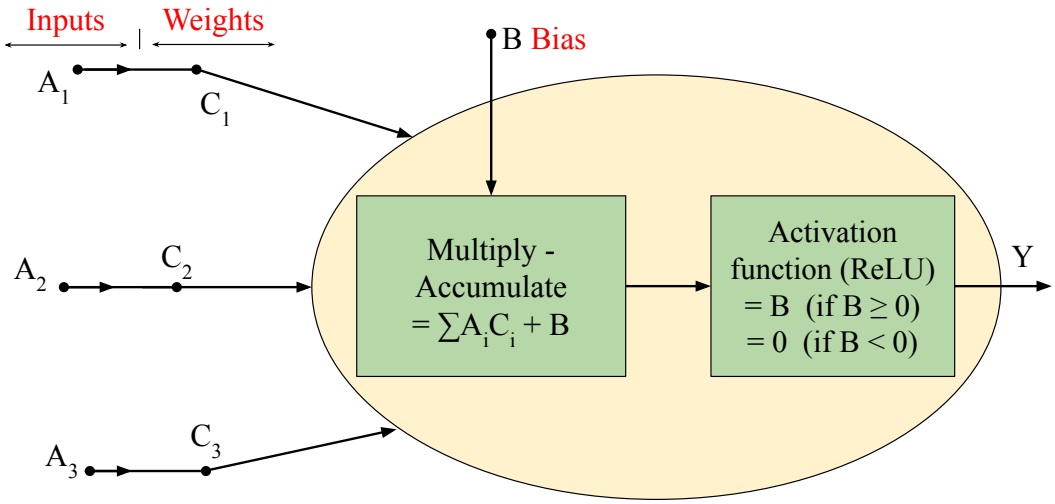

Figure 1.2: Generic structure for hardware implementation design of single neuron

significantly to the success of ANN training. The ANN's configuration may vary depending on its application. When used as a classifier, the input and output nodes correspond to input features and output classes, respectively. Conversely, for function approximation tasks, the ANN typically has both input and output nodes.

1.2.2 Convolutional Neural Networks Advancements and Components

The development of CNNs has significantly propelled the field of artificial intelligence, serving as a pivotal model for deep learning. CNNs find diverse applications in computer vision, ranging from voice recognition to self-service supermarkets, intelligent medical treatment, image categorization, and facial recognition. These networks are built upon the foundation of ANNs and are adept at simplifying parameters and addressing complex problems. CNNs can be described as 2-D network models, distinct from ANN and RNN, which operate in 1-D [2]. In CNNs, input features and filters (kernels) are utilized in both fully connected layers (1D) and convolution layers (2D) to create filter banks for feature extraction, as depicted in Figure 1.3. The fundamental components of Deep Neural Networks (DNNs) encompass Process-

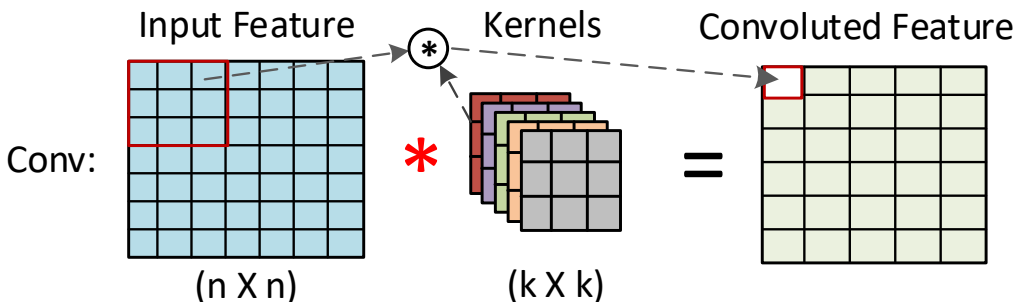

Figure 1.3: The convolution layer in deep neural networks (DNN) uses a MAC computing element during feature extraction, performing 2D matrix multiplication between the input feature map and kernel weights.

ing Engines (PEs), often including MAC units and AFs. The MAC units execute multiplication and accumulation operations, while the output undergoes non-linear transformations using different types of AFs.

Despite the capabilities of fully connected multilayer feedforward networks trained by error backpropagation algorithms, several challenges remain. To overcome these, CNNs introduce convolution for feature extraction, convolution kernels, strides, and pooling, which together form the traditional four components of a CNN model [3]. The convolution procedure yields feature maps with multiple features, often leading to overfitting. To mitigate this, pooling (such as maximum pooling and average pooling) is recommended for eliminating redundancy. Research and implementations of CNNs have explored the use of different hardware accelerators, such as FPGA, ASIC, and in-memory computing architectures [4, 5].

1.2.3 Recurrent Neural Networks for Sequence Recognition and Prediction

The RNNs are a specialized type of artificial neural network designed explicitly for handling sequential data, including time-series data and natural language sequences. What sets RNNs apart from other neural network architectures is their incorporation of feedback connections, enabling them to retain internal state information and process inputs in a temporal context. This characteristic makes RNNs exceptionally

well-suited for tasks that involve time dependencies and sequential patterns. RNNs find particular significance in natural language processing tasks, such as language translation and sentiment analysis, where context and word order are crucial for accurate understanding. In speech recognition, RNNs excel at effectively recognizing spoken words by analyzing audio signals over time. Furthermore, RNNs have demonstrated remarkable potential in time-series analysis, forecasting, and detecting anomalies in dynamic data streams. The ability of RNNs to capture temporal dynamics makes them valuable tools in various real-world applications where sequential data analysis is essential.

RNNs distinguish themselves from conventional feedforward neural networks by their capability to handle variable-length sequences, presenting a significant advantage. However, this flexibility comes at the cost of increased complexity and computational burden when processing sequential data step-by-step. A notable challenge in training RNNs is the vanishing gradient problem, which hinders the network's ability to learn long-range relationships within the data. To overcome this issue, modified RNN architectures like LSTM and GRU were introduced, effectively improving gradient flow and memory retention and thereby enhancing the performance of RNNs.

Efficient implementation and deployment of RNNs rely on sufficient computational resources, especially when dealing with lengthy sequences or large datasets. Utilizing hardware accelerators such as GPUs or specialized neural network circuits can significantly enhance the speed of RNN computations. Moreover, optimizing model design and employing techniques like batch processing and sequence truncation can help manage computational overhead [6]. In brief, RNNs have emerged as a crucial tool in various technical domains where sequential data analysis is fundamental. Their ability to recognize temporal patterns and handle variable-length sequences makes them indispensable in diverse applications. However, to fully harness the potential of RNNs, developers and researchers must consider the computing demands and tailor their implementations accordingly.

1.3 Training and Inference in Neural Networks

Neural networks involve two primary processes - Training and Inference - each relying on feed forward and backpropagation techniques. Forward propagation refers to passing input data through the network to generate an output, achieved by multiplying input data by weights and applying activation functions until predictions are obtained at the output layer. Backpropagation, on the other hand, calculates gradients of the loss function with respect to network weights, enabling weight updates during training and facilitating learning from errors to enhance predictions.

For both training and inference, frameworks follow a similar procedure. During training, a known dataset is fed into an untrained neural network, and outcomes are compared to the known dataset results. Based on the error evaluation, the framework adjusts neural network weights to improve performance on the given task. This re-evaluation and weight adjustment are essential in training to enhance the network's abilities. Retraining is also utilized to further improve inference accuracy in the model. In contrast, inference does not involve re-evaluation or weight alteration in neural network layers. Instead, it utilizes a trained model's knowledge to infer results. When new, unknown data is fed into the trained neural network, it provides predictions based on the model's predictive accuracy. Inference follows training, as it necessitates a trained neural network model to generate accurate predictions.

Continuing with the flow:

The focus of researchers has been to target both training architectures as well as inference ones, as each type of technique serves a different purpose. For achieving an accurate training model, an appropriate architecture is required that fulfills the following criteria:

- Provides meaningful feedback to facilitate weight adjustments.
- Reduces the necessity for retraining or extensive training datasets, ensuring precise inference capabilities.
- Ensures efficient training within a stipulated time frame, minimizing delays.

On the other hand, inference-specific architectures are not concerned with the intricacies of the training process. Instead, they aim to possess the following attributes:

- Deliver accurate inference specifically for the task they are trained for.
- Offer rapid response times during inference operations.

By catering to the unique requirements of both training and inference scenarios, researchers aim to optimize neural network architectures for better performance and efficiency in a wide range of applications.

1.4 Application of Neural Networks: Edge-AI

Deep neural networks (DNNs) have emerged as the cornerstone of numerous AI applications, from image classification to speech recognition, leading to a proliferation of their use. These powerful networks find applications in diverse fields, including self-driving cars and cancer detection, where they can even outperform human performance. The exceptional performance of DNNs stems from their ability to extract high-level features from raw sensory data after being statistically trained on vast datasets, effectively representing the input space. This contrasts with traditional approaches that relied on predefined rules or characteristics. However, as DNN accuracy increases, so does the computational complexity. While general-purpose computers, especially graphics processing units (GPUs), have been the backbone of DNN processing, there is a growing interest in providing DNN-specific acceleration, tailored to the unique demands of these networks.

In the pursuit of constructing intelligent machines that can achieve human-like goals, edge devices have come into play. These compact and application-specific neural network models are deployed on edge-AI devices, providing direct plug-and-play solutions for real-time environments. Among various applications, image classification stands out as an error-resilient use case, making it feasible to explore various techniques for its implementation. Additionally, the availability of a substantial

amount of training data further supports the application of image classification using a mix of optimization techniques. Therefore, neural networks, particularly deep learning-based DNNs, play a pivotal role in the realm of AI applications, empowering cutting-edge technologies and enabling tasks that surpass human capabilities. As the importance of edge-AI devices grows, the deployment of specialized neural network models becomes crucial for real-time and efficient solutions in diverse domains.

1.5 Key Contribution and Organization of the Thesis

1.5.1 Key Research Contribution

The key research contribution of this study revolves around exploring various hardware optimization techniques to enhance Edge AI performance while ensuring optimum or low consumption of area. Both computational level and structural level modifications were considered as part of the optimization strategies. The thesis introduces a novel technique aimed at reducing power consumption and time delay by scaling down resource consumption. This methodology involves the utilization of a multiplication-less multiplier utilizing the CORDIC algorithm, combined with dynamic bit quantization. The multiplier's iterative stages introduce latency, resulting in reduced throughput. To address this concern, a pipelining architecture is employed for the iterative stages.

The proposed architecture is thoroughly evaluated for both software performance metrics, such as classification accuracy, and hardware performance metrics, including area, power, and delay. The hardware implementation is carried out on the Xilinx-Virtex V7000 board. The key contributions of this work are summarized as follows:

- **Hardware Performance Enhancement:** Adapting the approximation technique of quantization, which takes care of the precision scaling. The increase of bit-width is restricted by this approximation methodology.

- **Novel Multiplication Technique:** Using CORDIC as the basis of multiplication-less product generation. The input is right-shifted and added to generate a product in an iterative manner.

- **Pipeline Adaptation:** Increasing the throughput of the MAC unit by using the concept of pipelining, as the proposed technique is iterative. Such serial-based operations can increase the path delay, but the pipeline helps to reduce this delay.

- **Focus on Accuracy:** The hardware utilization reduction and other hardware performance parameters enhancement is the goal of this work. However, inference accuracy is of utmost importance and is not to be handled leniently.

1.5.2 Organization of Thesis

The remaining sections of the thesis are structured as follows:

Chapter 2: This chapter covers the hardware architecture of conventional neurons and the structure of internal compute elements. It discusses the challenges faced by these structures, which have prompted global research efforts. Additionally, it highlights the motivation derived from the literature survey.

Chapter 3: The concept of CORDIC is thoroughly described in this section, along with its applications in various optimization techniques.

Chapter 4: This chapter presents a comprehensive solution for the MAC structure, explaining its distinctive features and implementation methodology through a numerical example.

Chapter 5: Experimental implementations are detailed in this chapter, showcasing the verification of the proposed MAC unit's design and functionality both individually and within a network model.

Chapter 6: The results obtained are discussed in this chapter, highlighting the main takeaways and demonstrating the superior performance of the proposed solution at both hardware and software levels.

Chapter 7: This section draws conclusions from the proposed iterative and quan-

tized CORDIC-based MAC unit and outlines the future scope of the work while acknowledging its limitations.

Chapter 2

Design MAC Unit for NNs: State-of-the-art Advancements, Limitations, and Motivation

The MAC unit is a fundamental building block in neural networks, responsible for making decisions and classifications based on weighted inputs and biases. Just like the biological neural networks in the human brain, where feedback and training improve decision-making, man-made neural networks utilize artificial neuron structures to perform similar tasks. An artificial neuron comprises two main components: the MAC and the AF units as depicted in Figure 2.1. These components serve as the basic units inside a neuron structure for various types of neural networks. The human brain's ability to learn and adapt through the years has inspired the development of artificial neural networks. By using different weights for classification and detection, biological neural networks achieve faster and more accurate decision-making. Following a similar principle, man-made neural networks leverage artificial neurons and their constituent blocks to classify and make decisions based on the knowledge gained from training.

In this chapter, we will delve into the design and implementation aspects of the MAC unit within neural networks. We will explore the state-of-the-art advancements that have contributed to the efficiency and performance of the MAC unit. Addi-

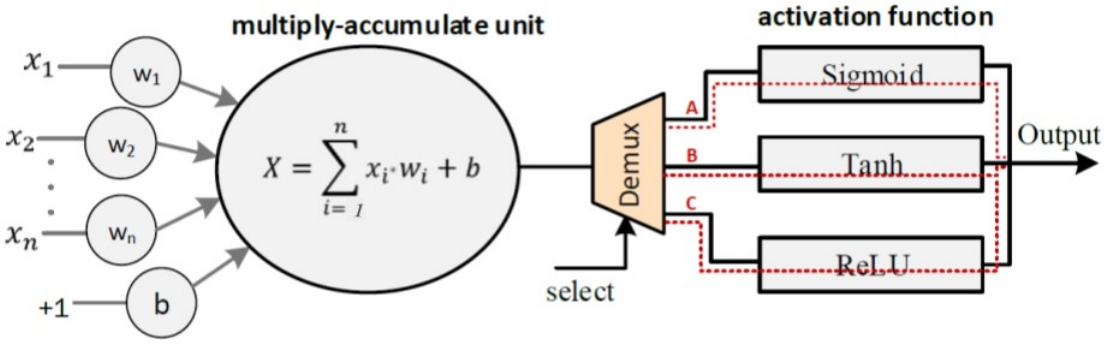

Figure 2.1: Architecture of a single neuron with MAC unit followed by a choice of activation functions.

tionally, we will address the limitations and challenges faced during its hardware implementation. Understanding the inner workings of the MAC unit is crucial for optimizing neural network architectures and unlocking their full potential in various applications.

2.1 Insights into MAC Unit and AFs for Neural Network Optimization

In this section, we delve into the internal design of the MAC unit, which serves as a fundamental building block of neural networks. The MAC unit consists of two essential sub-blocks: the Multiplier and the Accumulator. The Multiplier block plays a crucial role in the neural network's decision-making process by multiplying the input received at the input layer with the corresponding weights. These weighted inputs, acquired through the training of neurons, enable the network to make well-informed decisions based on the input signals [4, 7]. The design and optimization of the MAC unit are pivotal in unlocking the full potential of neural network architectures across various applications. It empowers neural networks to conduct complex computations and arrive at accurate decisions, making it a critical component of modern deep learning systems. In any type of neural network, whether it is an ANN or a CNN, the fundamental computational unit remains the same: the MAC unit. Its primary

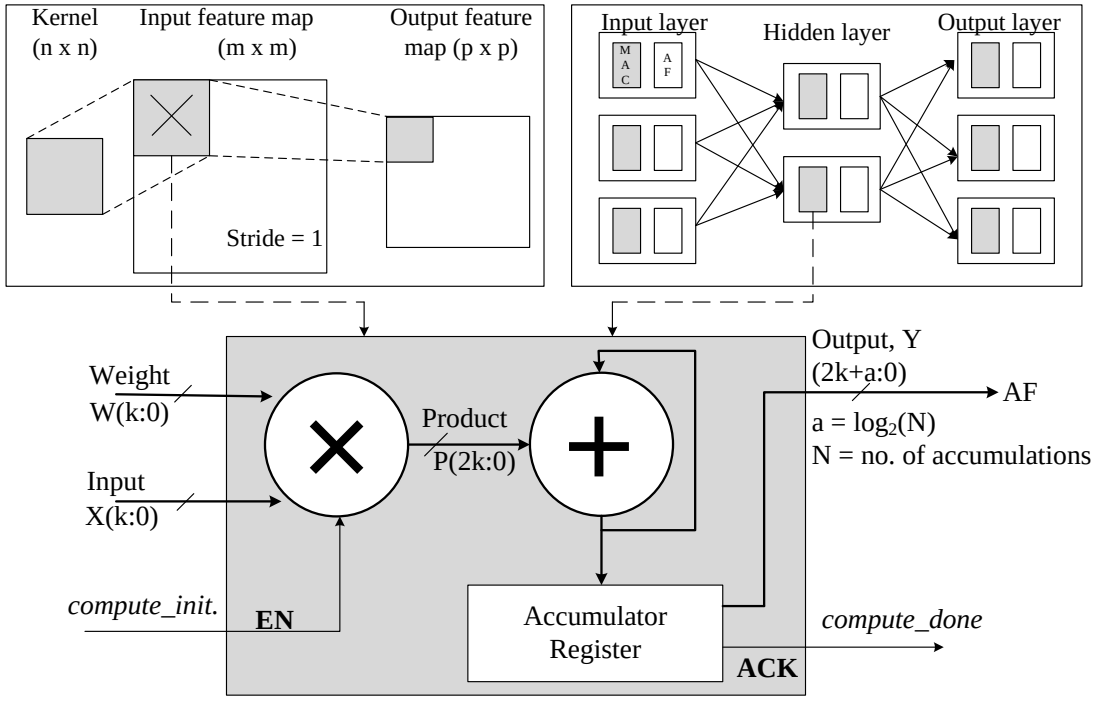

Figure 2.2: Fundamental MAC Elements across Different Networks: ANN and CNN

function is to take inputs and perform a signed product with the pre-loaded weights in each neuron. This characteristic is evident in the Figure 2.2.

The Multiplier block operates by multiplying the input signal with the corresponding weight, and the size of the output is determined by the sum of the sizes of the input and weight. The multiplications can be performed either sequentially or in parallel, depending on the specific implementation, as depicted in Figure 2.3. After the multiplication, the weighted inputs are directed to the Accumulator stage. In the Accumulator stage, the weighted inputs are summed together to generate the final output of the MAC unit. The size of the accumulator output is influenced by the number of accumulations performed. Similar to the Multiplier, the Accumulator can also operate in either serial or parallel mode, depending on the design requirements [7]. To ensure efficient data processing and prevent data loss, it is crucial to increase the accumulator size proportionally to the logarithm to the base 2 value of the number of accumulations. Neglecting to appropriately increase the accumulator size can

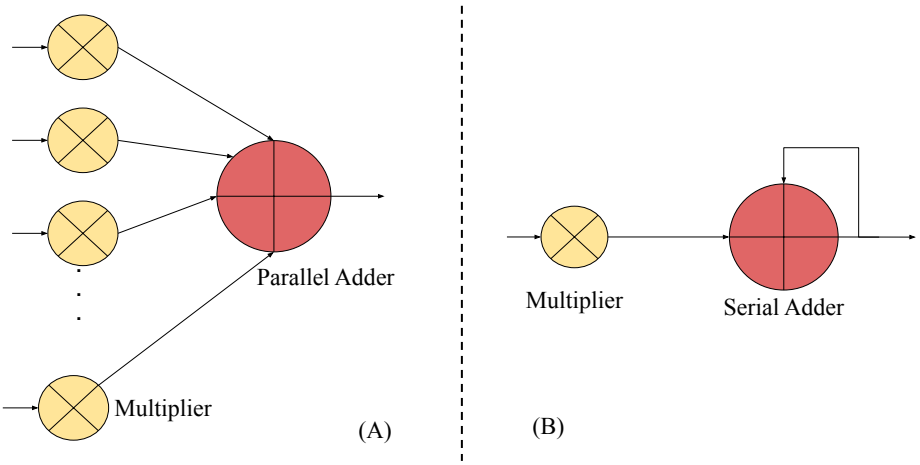

Figure 2.3: Architecture of MAC unit: (a) Parallel Summation inside a MAC Unit: The input and weight values are multiplied in parallel and then added together and (b) Serial Accumulation in a MAC Unit: The input and weight values are multiplied sequentially and the products are accumulated one after the other.

result in data loss, as the Most Significant Bit (MSB) may not be captured [4].

The AF of a neuron receives its input from the MAC unit. The MAC unit carries out the computation by multiplying the input with the corresponding weights and summing them together. The AF plays a critical role in enabling the neural network to learn and represent complex patterns and relationships in the input data by introducing non-linearity. The MAC unit's linear combination of inputs is transformed into a non-linear output using activation functions. This non-linearity allows neural networks to learn and capture more intricate and nuanced relationships in the data, enhancing their ability to handle challenging tasks such as image recognition and natural language processing. Various forms of non-linear activation functions are depicted in Figure 2.4. However, implementing these activation functions poses challenges due to their non-linear nature, and achieving accurate design and implementation often requires a substantial number of hardware resources.

The Sigmoid, ReLU (Rectified Linear Unit), Tanh (Hyperbolic Tangent), and Softmax functions are examples of frequently used conventional AFs. Each AF has

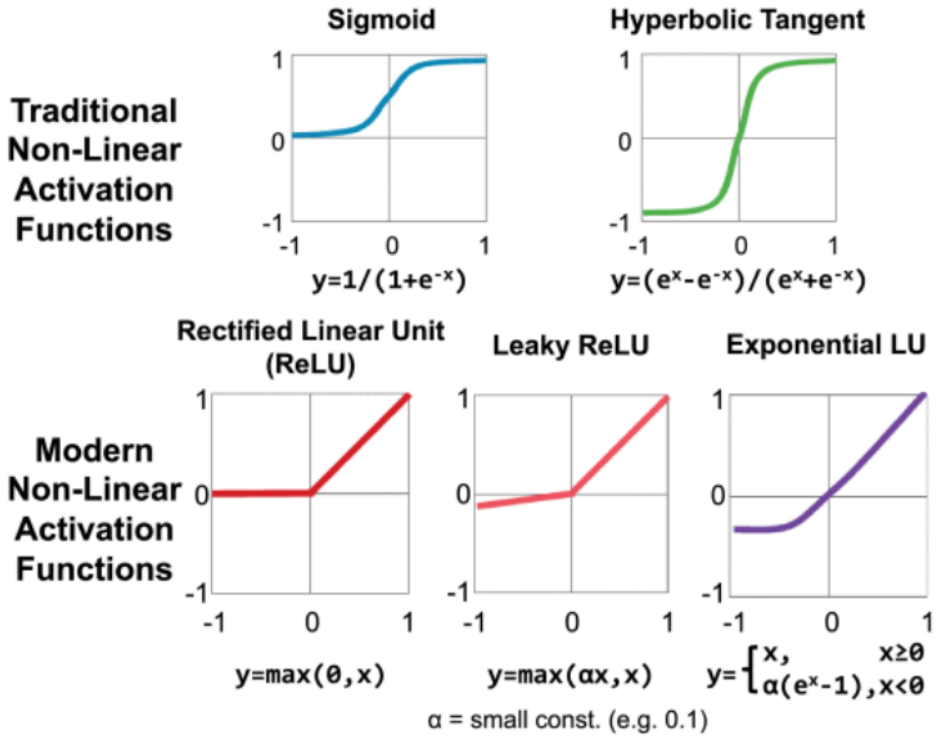

Figure 2.4: Various forms of nonlinear activation function

distinct qualities, advantages, and disadvantages that make them suited for certain neural network components and specific tasks. While Sigmoid and Tanh have found applications in specific contexts, ReLU is still widely employed due to its simplicity and ability to mitigate the vanishing gradient problem. Recent research has explored unconventional AFs to enhance network performance and address specific issues. Examples include Swish, PReLU (Parametric ReLU), and different versions of the Leaky ReLU. These non-conventional AFs aim to mitigate problems like gradient saturation and vanishing gradients while striking a balance between non-linearity and computational efficiency. The activation function selected can strongly impact the effectiveness of training, the rate of convergence, and the overall performance of the network. Designing effective deep learning models necessitates a thorough understanding of the characteristics of various AFs and their influence on neural network designs.

2.2 Challenges of hardware resources and computation complexity in NN implementations

A conventional neuron serves as the central computational unit within each layer of a neural network. In a DNN, the architecture typically comprises an input layer, an output layer, and numerous hidden layers consisting of interconnected neurons. Focusing on the input layer of a CNN, each neuron is assigned a weight of a specific bit width, for instance, 'k' bits. In this setup, all neurons in the input layer receive inputs of the same size - 'k' bits. Within each neuron, the input data is multiplied with the corresponding weight, resulting in a product of '2k' bits. This product is then passed on to the adder block, where 'N' accumulations take place.

Consequently, we now require an adder with a data capacity exceeding '2k' due to the multiple accumulations. The accumulator's capacity increases significantly to '$2k+log_2(N)$'. This highlights the increased hardware demand for bit-wise addition operations, which more than doubles the input data size. The data then progresses to the next layer, one of the many hidden layers to follow in the neural network. The input data size for the subsequent layer becomes '$2k+log_2(N)$'. Subsequently, the output data size undergoes further amplification to approximately '$4k+log_2(N)$' + $log_2(N)$. Consequently, the data size grows substantially, exceeding twice the data size present at the previous layer's neuron. This accumulation of hardware requirements at each layer can make the implementation of deeper neural networks almost infeasible, as the increased complexity and resource demands hinder the network's practicality and performance. The availability of enormous datasets for training and classification tasks has resulted in a significant demand for resources during neural network implementation. Notably, the multiplier block is known for its resource-intensive nature. Consequently, optimizing the multiplier block to reduce resource requirements and computational complexity becomes of paramount importance. A high number of bit-wise operations contributes to increased power consumption and introduces considerable latency in the path to the output. Addressing these challenges is

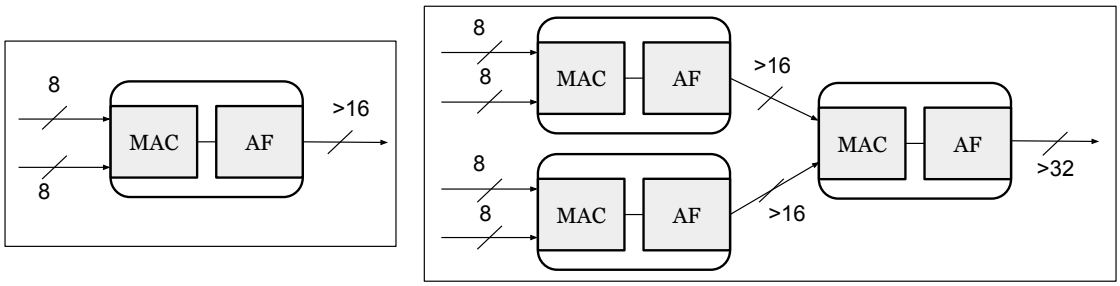

Figure 2.5: Increase in bit precision from one layer to another in the conventional neural network

crucial for achieving an efficient and accurate implementation of neural networks. To tackle these issues, researchers have explored various techniques and approaches. The following sections will discuss some of the methods proposed by researchers to mitigate the resource demands and computation complexity in neural network implementations.

2.3 State-of-the-Art techniques for optimizing Neural Network implementations and its limitations

Among neural network implementations, the MAC unit takes the spotlight as the most computationally intensive element. With focused determination, researchers are paving the way for novel architectures, addressing the complexities of both conventional and state-of-the-art neuron structures. To achieve more accurate neural network designs, diverse performance-enhancing techniques come into play. Since the performance requirements vary across different applications, a significant focus has been on exploring application-specific edge devices. A pivotal resource in this context is the article by Sze et al. [8], shedding light on neural networks' hardware-level operations and suitable accelerators for specific applications. The study delves into techniques such as precision reduction, operation minimization, model size reduction, and near-data processing, all integral to hardware-software co-design for neural

networks. State-of-the-art architectures for multipliers and adders within the MAC unit have emerged, refining the neural network's performance. Innovations have been made to traditional architectures, including Booth encoding, Wallace tree method, and Vedic method [9–11], with hybrid designs that employ efficient techniques for specific operations [12]. While these modifications have been extensively explored, researchers continue their quest for further MAC optimization for DNNs.

This quest for optimization is broadly categorized into logarithmic and non-logarithmic architectures, with a focus on computational efficiency. The hardware aspect of optimization is comprehensively discussed in the work by Armeniakos et al. [13], which presents various strategies such as computation reduction, approximation, and precision scaling. One technique, post-training quantization, emerges as a crucial precision-scaling method, central to this article's focus. Additionally, the study outlines other subcategories of hardware approximation, unveiling new possibilities for efficient neural network implementations. The optimization achieved through the use of logarithmic multipliers significantly impacts neural network performance. Researchers employ various methods in iterative logarithmic multipliers to overcome the limitations of single-stage logarithmic multipliers [14]. For instance, the two-stage operand trimming approach in iterative logarithmic multipliers improves image processing results [15]. Additionally, a hybrid multiplier with radix-4 booth encoding and logarithmic encoding applied to the MSB and LSB, respectively, has demonstrated promising results [16]. These advancements in multipliers contribute to enhanced hardware performance for neural network systems.

Non-logarithmic optimization encompasses a diverse range of techniques, including both coarse-grained and fine-grained approaches [17]. In the coarse-grained technique, modifications are made at the architecture level, treating the blocks as black boxes. On the other hand, fine-grained methodology focuses on gate-level alterations, particularly for bit-wise operations. One way to manipulate bit-precision is by changing the data format representation. The Ristretto approach simplifies data arithmetic and reduces the bit width [18]. Precision scaling is predominantly achieved through quantization [19]. For instance, the Quantized Neural Network (QNN)

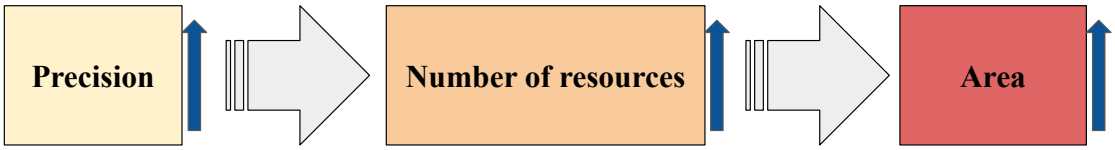

Figure 2.6: Trade-offs while achieving higher accuracy in NNs

explores quantization on a mixed precision neural network [20]. However, lower data bit-widths (e.g., 4 bits, 2 bits, and 1 bit) in architectures lead to reduced accuracy, particularly for larger datasets [21]. To address potential bit overflow issues, overflow-aware quantization scales the data down to avoid data loss [22]. Moreover, error compensation and recovery techniques have been devised for state-of-the-art approximate adders and multipliers to improve quantization accuracy. Implementing neural networks on ASIC with a mesh of resources enhances bandwidth and data reuse, optimizing computation resource utilization [23]. Figure 2.7 depicts a graphical representation of the trade-offs associated with achieving higher accuracy in neural network implementations.

Another efficient computation technique involves normalizing weights and approximating them to powers of two, leading to a 50% improvement in power and energy performance [7]. However, this design comes with a compromise on critical delay. These diverse approaches reflect researchers' focus on optimizing neural network implementations to achieve greater efficiency and accuracy in various applications. A similar technique was introduced in [24], where negative powers of two were utilized to approximate hyperbolic tangent computation for multiplication. This technique, known as CORDIC, has been applied in configurable architectures of MAC units and AFs. CORDIC is based on the Virtual Scaling Free technique [25], and its iterative nature has been deployed within MAC units [21] and AFs [26]. The CORDIC technique has emerged as a fundamental component of the proposed method, providing efficient approximations and contributing to the overall optimization of neural network implementations.

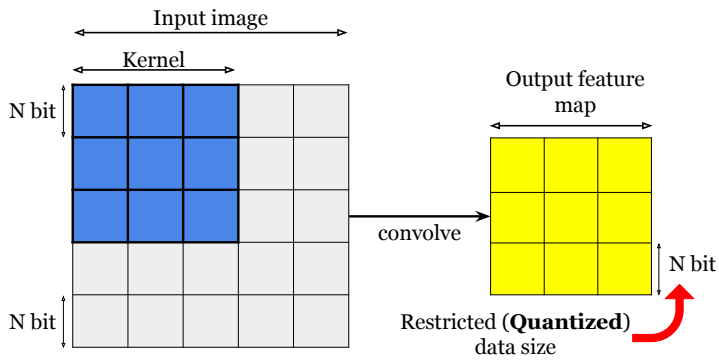

Figure 2.7: Quantized convolution operation using kernel in a CNN layer

2.4 Motivation: Advancing Deep Neural Network Implementations

The previous section (Section 2.3) demonstrates how researchers have shaped their studies and implementations to optimize neural network techniques. The success of DNN in current applications has prompted exploration in diverse domains for improved outcomes. Ongoing advancements in technology, neural network models, and data arithmetic are contributing to the increasing significance of DNNs. The challenges faced in device performance become the foundation for future research endeavours. Implementable solutions hold the potential to meet market demands and cater to a wide range of applications. The abundance of data for various applications continues to grow, and its quality is constantly improving, leading to denser data with higher bit-precision. Although higher bit precision operations offer greater accuracy, they require more resources for implementation, leading to undesirable chip area consumption (refer to Figure 2.7).

In this research, we are driven by the motivation to address the challenges highlighted in Section 2.2. The primary focus is on reducing the hardware requirements of the system by applying approximation techniques, with a specific emphasis on quantization. This technique involves truncating bits or data to maintain data size while ensuring that the CNN computation produces the same output bit precision as the input, as depicted in Figure 2.8. Quantization provides several benefits for neural

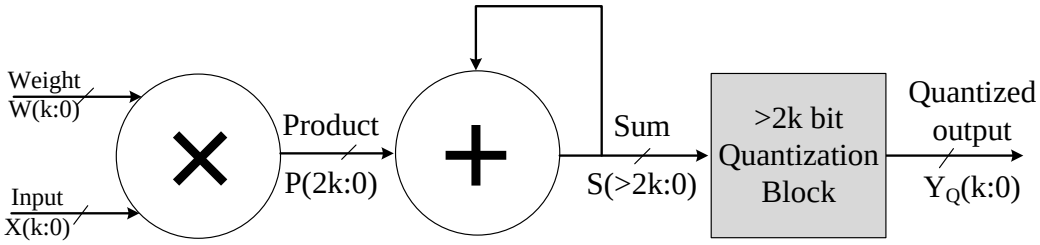

Figure 2.8: Effect of quantizer block after the accumulator stage

network implementations, as summarized in the points below. These advantages inspire us to explore quantization as a promising solution to enhance neural network implementations.

- Optimization of hardware requirements at both the MAC level and the overall network level.
- Easier implementation due to uniformity in the bit precision.
- Inference accuracy is not significantly compromised, ensuring reliable results.

The motivation behind this work lies in addressing the challenges posed by quantization in a MAC unit. Quantization can occur either after the accumulator block or after the multiplier block, and before the adder/accumulator block. In the first case, the multiplier generates a product that is twice the size of the input data size 'k'. Subsequently, the adder performs addition with $2k$ bits. To downsize the output to approximately that of the input, a quantizer block is connected after the accumulator block, as depicted in Figure 2.8. However, this solution presents two problems. Firstly, it requires twice the resources for the addition operation after the multiplier block, i.e., a $2k$ number of additions. Secondly, an external quantizer block necessitates $2k$ resources at the input to operate and compute an approximate output. Whereas, the second case involves connecting the multiplier directly to the quantizer block before passing the product to the adder block, providing a solution to the problem faced in the previous case. The quantizer block reduces the data size to the original k bits, making it compatible with the adder block, which then performs

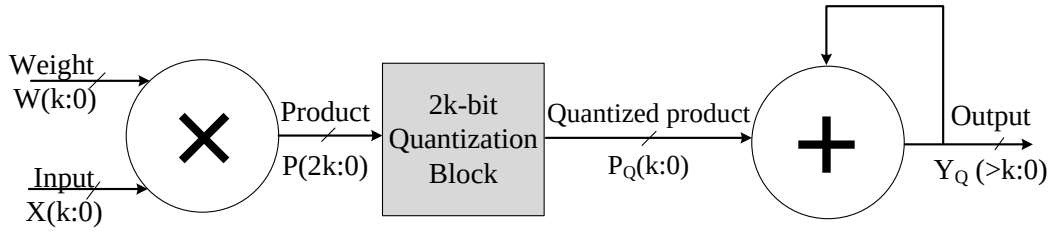

Figure 2.9: Effect of quantizer block after the multiplier stage

the operation on the original number of bits. This results in a reduction of hardware requirements for the bitwise adders to half, making it a feasible solution in terms of hardware saving. However, the challenge with this solution lies in the external quantizer block, which requires $2k$ bit operators due to $2k$ inputs, as illustrated in Figure 2.9. Additionally, an external block may increase the delay of the system and escalate the hardware requirement of the overall neuron for smaller data sizes.

In our work, we are adopting a similar technique where quantization occurs at the multiplier stage instead of the accumulator stage. This approach allows us to overcome the need for an external quantization block after the multiplier. Instead, we incorporate a dynamic quantization step within the product generation process to limit the bit-width of the product. As a result, the requirement for extra hardware to handle data size reduction, timing delay, and power consumption related to the external block is eliminated. This enables us to achieve hardware optimization and improve the overall efficiency of the system.

Chapter 3

CORDIC Algorithm: An Introduction and Its Application in MAC Solutions

CORDIC stands for **CO**ordinate **R**otation **DI**gital **C**omputation, which is used to realize trigonometric functions using rotations of a vector. The recursive behaviour of the proposed work comes from the pseudo rotation of the vector in steps. Different CORDIC modes address a variety of mathematical computational modifications. Each of these modes - Vectoring and Rotational - operate using three types of coordinate systems, that is, Linear, Circular and Hyperbolic. This work uses the hyperbolic mode of operation. The rotational mode for all three coordinate systems is described in (Table). The scalar in the circular (K_c) and hyperbolic (K_h) forms of operation have different constant values. These values converge after a number of iterations as the value decreases monotonously with each recursion. The variables X and Y are the coordinates and Z is used to know the angle of the rotation of the vector.

We consider that K is the scaling factor for the circular and hyperbolic functions, which can be taken as a common factor. The trigonometric *tan* or *tanh* are used in the further computational approximation as the negative powers of 2. T his is the exponential approximation of the CORDIC function. In a generic form, the

Table 3.1: Generalized CORDIC Algorithm

m	**Rotational**
Circular 1	$A_n = K_c(A_0 cos\theta_0 - B_0 sin\theta_0)$ $B_n = K_c(B_0 cos\theta_0 + A_0 sin\theta_0)$ $\theta_n = 0$
Linear 0	$A_n = A_0$ $B_n = B_0 + A_0 \cdot D_0$ $\theta_n = 0$
Hyperbolic -1	$A_n = K_h(A_0 cosh\theta_0 - B_0 sinh\theta_0)$ $B_n = K_h(B_0 cosh\theta_0 + A_0 sinh\theta_0)$ $\theta_n = 0$

CORDIC equations can be formulated as, for all modes of trajectories:

$$A_{i+1} = A_i - B_i \cdot d_i \cdot tan\ \alpha_i \tag{3.1a}$$

$$B_{i+1} = B_i + A_i \cdot d_i \cdot tan\ \alpha_i \tag{3.1b}$$

$$\theta_{i+1} = \theta_i - d_i \cdot \alpha_i \tag{3.1c}$$

Here, A_i, B_i and θ_i are the variables and α_i is the angle of rotation (in radians) for the i^{th} iteration, where $i = 1, 2, 3, ...n$. The angle α_i varies for the linear, circular and hyperbolic forms as 2^{-i}, $tan^{-1}(2^{-i})$ and $tanh^{-1}(2^{-i})$, respectively. the variable d_i shows the direction of rotation of the vector, so the value of d is either 1 or -1. The CORDIC equations can be re-written in the linear form for hardware implementation as:

$$A_{i+1} = A_i - m \cdot d_i \cdot B_i \cdot 2^{-i} \tag{3.2a}$$

$$B_{i+1} = B_i + d_i \cdot A_i \cdot 2^{-i} \tag{3.2b}$$

$$\theta_{i+1} = \theta_i - d_i \cdot \alpha_i \tag{3.2c}$$

In the above equations, mode m belongs to the set $0, 1, -1$ indicates the linear, circular and hyperbolic modes. The rotational or trigonometric mode operating

for hyperbolic form generates equations using *hyperbolic tangent* or *tanh* function. These *tanh* values, represented in negative powers of 2, can be implemented on the hardware using barrel shifters and stored in memory elements. The direction of rotation can be decided using the multiplexer blocks. The addition/subtraction operations can be implemented using simple adder blocks. Hence the reformulated equations show the implementation feasibility of CORDIC equations. This article uses the hyperbolic form of CORDIC equations for calculating the multiplier-less product using the basic shift-and-add method.

The single-stage CORDIC is implemented using the input X, the weight Z and output Y. Here we show how the CORDIC concept acts to generate the product. The register Y is used to store the gradual product generated in the first iteration.

$$Y = X - X \cdot 2^{-i} = X - (X >> 1) \tag{3.3a}$$

$$X_{out} = (X >> 1) \tag{3.3b}$$

$$Z_{out} = Z - 2^{-i} = Z - (1 >> 1) \tag{3.3c}$$

$$s = sign(Z_{out}) \tag{3.3d}$$

With the right shift performed to half the value of the input, the bit width of the output Y, the output value of X, $Xout$ and Z, Z_{out} increases. The value of Y that is produced can be called a raw product. We obtain almost the exact product with this technique when performed in recursions. This increases the need for resources, leading to area consumption rise. The rise in hardware requirements will ultimately increase power consumption and reduce the feasibility of the network. The sign of the operation performed for obtaining Y depends on the variable s when we move from one iteration to another. The use of a barrel shifter for shifting operations and performing iterations of the same sub-process can take up more clock cycles, hence increasing the latency of the system. We extend this solution to merge it with dynamic quantization to obtain an approximately accurate product using the right shift-and-add method on the input. The delay can be reduced by using the pipelined architecture. This will lead to an increase in the throughput of the overall solution. The solution is also focused on establishing uniformity in the data bit-width

throughout the network. It also enables flexibility in terms of the data size to be implemented in the network. The multi-stage proposed solution is discussed in the next chapter.

Chapter 4

QuantMAC: A Novel MAC Unit with Enhanced Features for DNNs

A multiplier in any MAC structure is a computationally intensive block. It is the block that increases the requirement of the resources, to store its output. Multiplication is an important block, it must be tended to and manipulated carefully. Since the multiplier is the first block of any layer, the error occurring at its output should be as minimum as possible, followed by a data accumulator block with restrictive data increase flexibility. The error might magnify at further stages like the activation function block, as it is a non-linear function. The concept of CORDIC is a versatile computational technique, which can be adapted with manipulation or approximation. The mathematical alteration of CORDIC equations is used for the implementation of a basic operation of multiplication. The error created is minimal, and performance is enhanced. The proposed multiplier is based on the concept of CORDIC along with quantization for keeping a check on the data size of the output. The strategy applied here is to generate a multiplication-less product, by simply using the shift-and-add technique. Shift-and-add techniques usually have a bit-wise shift operation and thus need to be performed in many iterations. The maximum number of iterations depends on the data size of the operands involved. Hence, it becomes mandatory for the recurring operations of shift and adds to be performed sequentially. The sequential process usually uses lesser resources or reuses

resources, as against parallel architectures. Parallel architectures usually use more resources but generate faster outputs. Since the iterations increase the path delay, the pipelining architecture is adapted to the proposed solution. The proposed MAC has an enhanced architecture with multiple unique features making it a novel strategy for better performance of a neuron structure in a neural network. The features of the proposed multiplier can be listed as below:

- **Right Shift-and-Add method of Multiplication:** The state-of-the-art works usually involve left shift-and-add method. Our MAC unit uses a right shift-and-add method of multiplication based on the manipulations using the CORDIC concept.
- **Dynamic Quantization:** Dynamic quantization during the process of product generation, eliminates the need of an external quantizer block.
- **Pipeline architecture:** The iterative process needs to have a higher throughput, which can be achieved by a pipelined architecture, adapted for all the iterations
- **Size Scalability:** The size of the system is scalable to higher or lower bits, by adjusting the number of iterations in the architecture.

These will be discussed in detail, on how each feature contributes to improving the performance of the multiplier.

4.1 Features of the Proposed Design

4.1.1 Multiplication-less Multiplier unit

The arithmetic representation used here is signed fixed point arithmetic which has an integer part and a fractional part. The representation will be signed bit plus the number of bits of the value. This work has used only 1 bit of integer and other bits will be used as fractional bits. So, for example of an 8-bit value, 1-bit is an

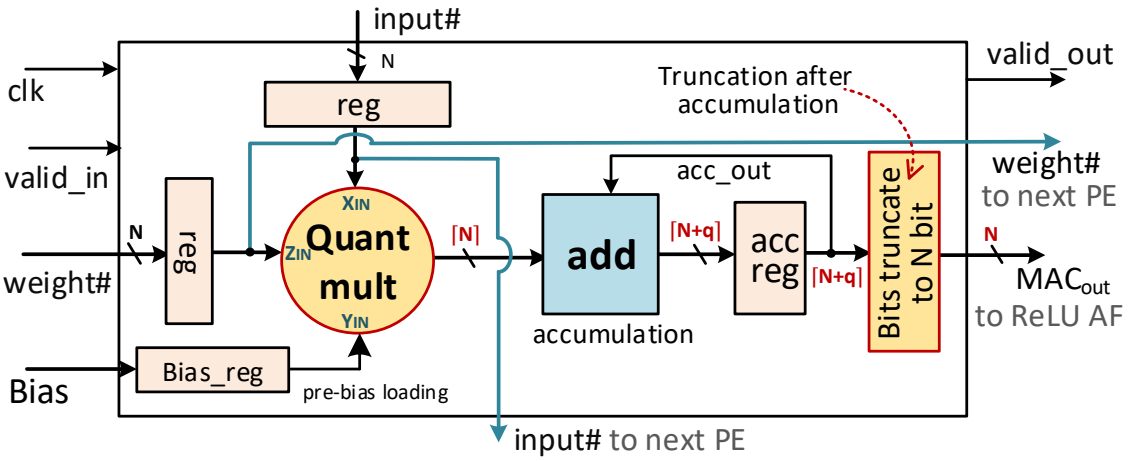

Figure 4.1: Architectural overview of the novel quantized multiplier

integer and the remaining 7 bits are the fractional part, occurring after the decimal point. The basic multiplication operation that is performed is,

$$\mathbf{X}_{product} = \mathbf{X}_{input} * \mathbf{W}_{input} \tag{4.1}$$

The multiplication operation can also be implemented with simple addition operations. The computations are manipulated in a manner such that bitwise shifting and adding or subtracting up the shifted version to the original value. The method adopted in our technique is a "right" shift-and-add method. The right shift means that the value of the operand is reduced to half of the existing value. The reduction of the value to half in each iteration can also be depicted as the multiplication of the original value with the negative powers of 2. Using the approximation of the CORDIC algorithm, we implement the following equation -

$$\begin{aligned} \mathbf{X}_p &= \mathbf{X}_i * \sum_{j=1}^{p} \mathbf{a}_j * 2^{-j} \\ &= \sum_{j=1}^{p} \mathbf{a}_j * \left(\mathbf{X_i} * \mathbf{2^{-j}}\right) \end{aligned} \tag{4.2}$$

This equation summarizes the product generated iteratively. The 'p' here is decided by the number of bits in the fractional part. The term X_p is the generated product term at the end of iterations, whereas X_i is the input to the MAC unit. The coefficient a_j is a dependent coefficient. It is driven by the W_{input} when it is reduced to zero. The coefficient manages the sign of the summation operation.

There are two subprocesses carried out in parallel in the process of multiplication. One subprocess is that of product generation and the second subprocess is weight reduction. Both sub-processes are repeated for p times along with dynamic quantization. The first subprocess is weight reduction to zero, with each iteration. The weight in this technique is considered to be maximum normalised. The weight is reduced by negative powers of 2 in each repetition, till the weight is reduced to zero ideally. This means that 2^{-1} used for reducing the weight is right-shifted in each iteration. The right-shifted factor will increase the bit-width of the data, hence the last bit of the factor is dropped off. This truncated factor will have the same bit-width as that of the weight. As the weight approaches zero, the accuracy goes on increasing. The equation that justifies the operation is,

$$\mathbf{W_{j+1}} = maxnorm\big(\mathbf{W_j}\big) \pm \mathbf{2^{-(j+1)}} \tag{4.3}$$

The second subprocess is that of product generation. The input is added/subtracted with its right-shifted version in the first step. We consider Y_j as the output from the previous step/iteration. This Y_{j+1} is the output from the current step, which is the addition/subtraction operation of the Y_j with the right-shifted version of the X_j value from the previous stage. The dropping off of the LSB of the right-shifted X_j is also performed in each iteration. By losing the trivial LSB, we restrict the bit-width of the product as well as not harming the accuracy much.

$$\begin{aligned} \mathbf{Y_{j+1}} &= \mathbf{Y_j} \pm trunc\big(\mathbf{X_j} \times \mathbf{2^{-j}}\big) \\ &= \mathbf{Y_j} \pm trunc\big(\mathbf{X_j} >> \mathbf{j}\big) \end{aligned} \tag{4.4}$$

The sign of operation of X depends on the sign of W obtained in the previous step. With each step, as the value of W approaches zero, the product generated approaches a more accurate value. The value of the Y will be lesser than or equal to X_i as the value of W is always lesser than or equal to one, since the weight is maximum normalized. Thus, the bit-width of Y is also restricted to that of the input. This is because of dynamic quantization which is carried out in the process of multiplication. Thus two parallel processes are carried out, where the accuracy of

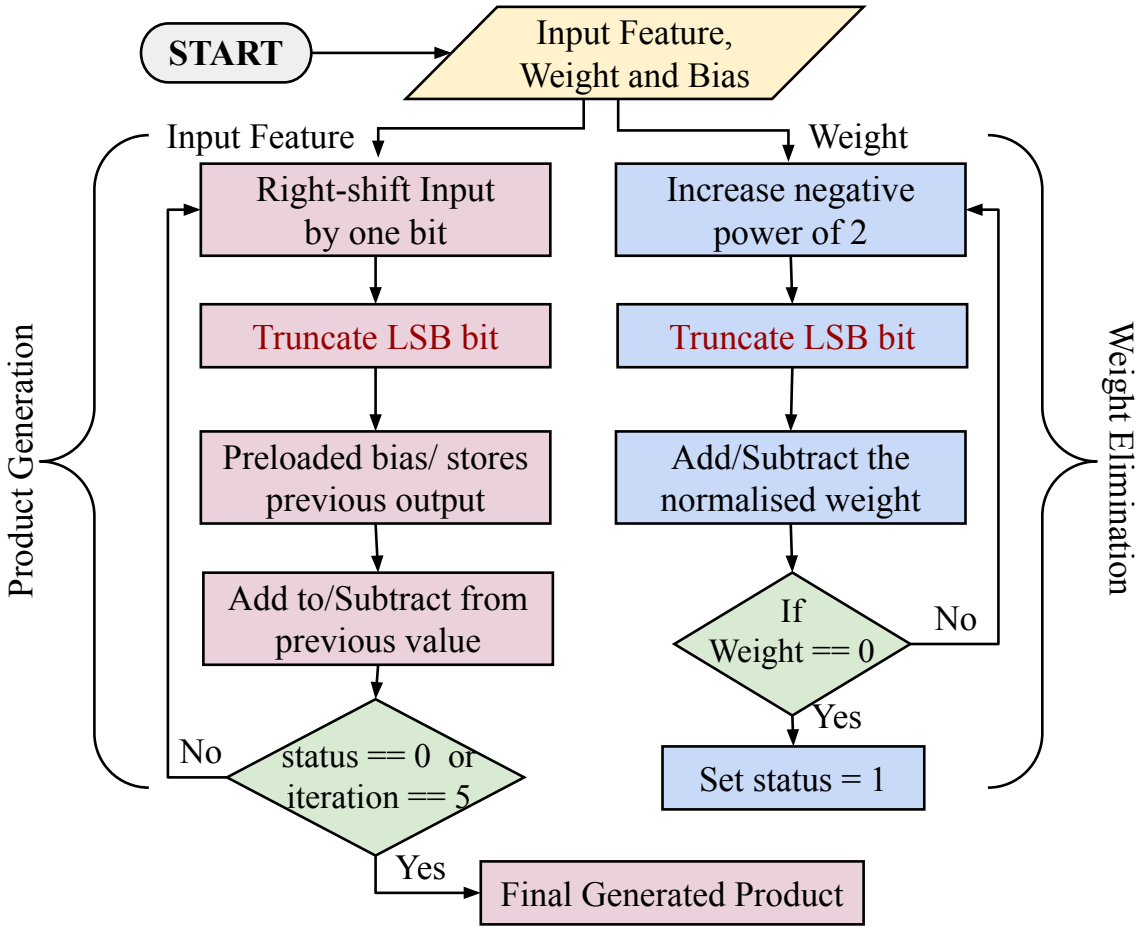

Figure 4.2: Flowchart showing two parallel subprocesses of the proposed MAC unit

the product depends on the number of iterations it goes through or the number of iterations it takes for the weight to reach absolute zero.

4.1.2 Dynamic Quantization

Multiplication generally produces output that is twice the size of the input to the multiplier. The proposed multiplier must restrict the bit-width for smoother propagation to further layers. Quantization can limit the size of the product generated. Quantization basically truncates data and checks the data size. Hence, quantization must be applied to the multiplier product to restore it to its original size.

The proposed design uses an iterative multiplier, each iteration taking a step towards product generation. In each such iteration, the bit size increases by 1 bit, which is

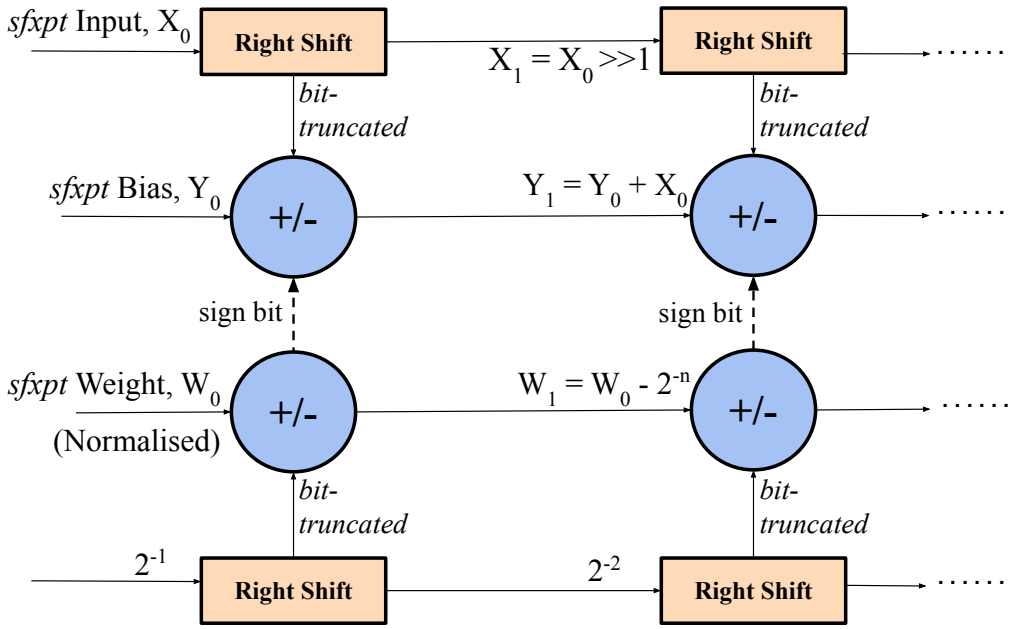

Figure 4.3: Pipelining performed inside the multiplier unit between the 'n' iterative stages

shredded. The need for an external quantization block is eliminated here. As the multiplier itself is a quantizer plus multiplier, we are saving on the external resources needed for the quantizer block and those saved in the multiplier block. This is called dynamic quantization, which keeps up with the accuracy of the multiplier but at the same time, trims down data size to original.

4.1.3 Pipeline Architecture

The repetitive behaviour of the parallel subprocesses in the algorithm leads to a delay due to a serial process. With the increase in the number of bits or number of iterations, more is the delay for producing a single product. The delay propagates to other layers in the network, making product generation a delayed process. Delays in terms of clock cycles can be reduced by pipelining the process instead of a streamlined process. With each step, the clock cycles are saved.

4.1.4 Size Scalability

The proposed multiplier restricts the size of the product in each iteration, and hence, the propagated data size will remain almost the same throughout the network. The proposed multiplier has flexibility in the bit width. It can be scaled up to a higher data size or down to a lower bit width. The system is said to be scalable when the change in the data size does not affect the architecture arrangement or the circuitry. Our proposed multiplier can be called scalable, as the architecture organization is not disturbed; only the size of the barrel shifter and the adder circuitry need to be altered, depending on the size of the data coming into the multiplier. Thus, the proposed design can be scaled to any desired data length.

4.2 Methodology of the Proposed Design

The features discussed in the previous section are the main contributing factors to the successful optimization technique that is proposed in this work. The technique incorporates a blend of empirical and structural modifications. While coming up with the proposed design, we keep in mind the accuracy of the MAC unit of a single neuron, as it will be the prominent contributing factor to the classification accuracy. The accuracy can be slightly compromised when dealing with large data, such as pixels of an image. The classification of images with the proposed quantized MAC unit is very slightly affected, as this application is very error-resilient.

4.2.1 Design architecture and State Machine control

The architecture of the proposed technique can be seen from Figure 4.3. The design is applicable for different bit-widths of data, but some prerequisites must be kept in mind before using this technique. The prerequisites of the proposed design are listed below:

1. Arithmetic representation involved in this solution is signed fixed point arithmetic

2. Inputs and weights are considered of the same bit-width - same integer and fractional parts

3. Weights are maximum normalised, which makes their values lesser than or equal to 1

These requirements will operate ideally with the given design. The neuron is loaded with weights that are obtained from training, and an input enters the neuron to meet the multiplier. The input to the multiplier is actually an input to the first iteration of the many required. This input can be denoted as X_N, and the weight as operated in each iteration can be denoted as Z_N, where $N = 0, 1, 2, 3...n$. The output of each iteration will be saved in Y_N, where $N = 0, 1, 2, 3...n$ So the final Y_n will be equal to the product $X_n * Z_n$. As we already discussed, there are two subprocesses that happen in each iteration in parallel. The subprocesses mainly include shift, dynamic truncation, and add/subtract. The product generation and weight elimination will be repeated in each layer on the signed fixed-point arithmetic. The finite state machine coordinates the data and control signals.

The state machine, as shown in Figure 4.4, exhibits the flow of the process. The signal `init` is set to 1 and will start the multiplication process using the proposed multiplier design and the input data received. The multiplicative iterations will be carried out till the `status` signal is zero. When the product is ready, the status signal is set to 1. After the signal is set, the signal is sent to the next stage of the MAC unit, which is the adder circuitry. The product is passed as data to the adder (or accumulator) unit. The accumulator will send back a signal to the multiplier to continue with multiplications in order to calculate the total of all products. This is the `index` signal which will facilitate multiplications when it is low. After each product is generated, the low `status` signal will switch to high, while the `index` signal remains low. The `index` is again reset when the next multiplication iterations start due to the low `index`. When the accumulation process is completed, the `index` is set to high. This triggers the activation function (AF) block, which starts its non-linear operations. The activation function block can be removed in the case

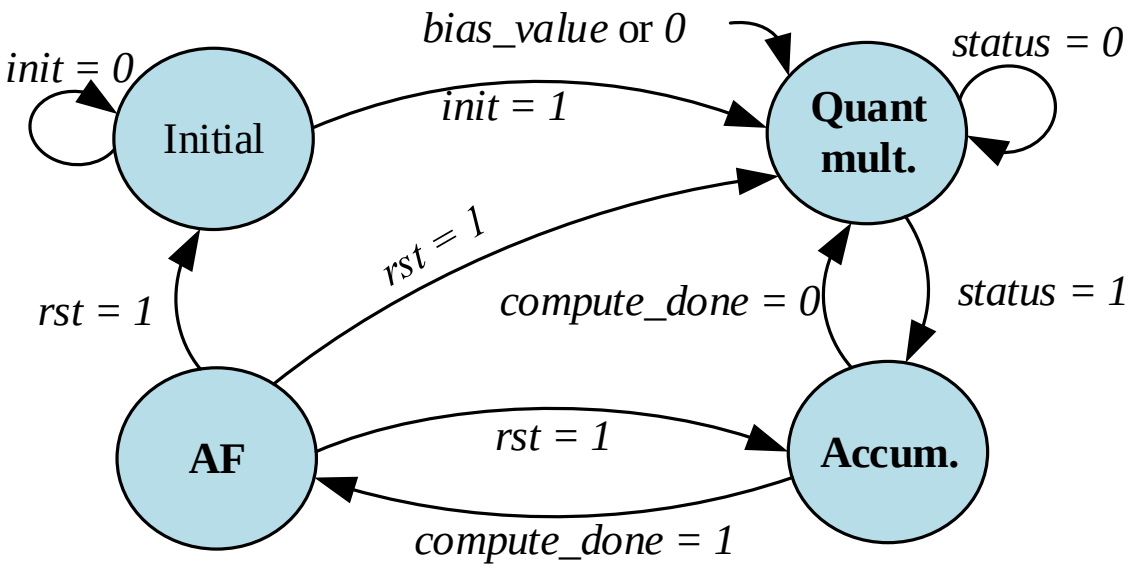

Figure 4.4: State machine to control the operations inside each neuron of a neural network using the proposed MAC design

of CNN hidden layers without AF. After the final operation, the `rst` signal is set to high and sent back to the Input, Multiplier and Accumulator blocks, where the registers are reset to 0. A `compute_done` signal is sent as a "Done" signal to the next layer for further computations. So the `compute_done` signal acts like an output signal of the Finite State Machine.

4.2.2 Empirical Calculations using an Example

The empirical realization of the solution can be seen in the following example. The example has considered a weighted sample obtained from a trained model. The data size is considered a signed 5-bit input, and the weight data is normalised to the maximum. The data is uniformly represented; that is, the data is divided into one sign bit, one integer bit, and the remaining three bits as fractional bits. The input data value is considered as +1.5, on which the operations are performed, represented in bits as {01.100}. The weight of the sample is +0.87, which can be written in bits as {00.111}. In the first iteration, the shifted version of the input value $X_0 >> 1$ is truncated by losing the LSB bit and then stored in X_1. The truncated right-shifted

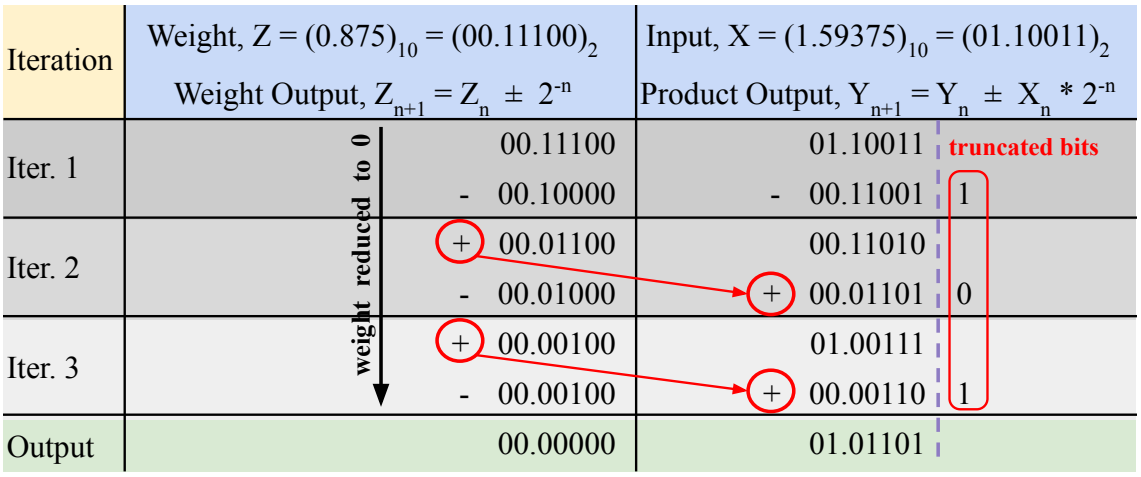

Figure 4.5: Example of product generation by providing input to a weighted sample using the novel multiplier computation

value of the input X_1 is subtracted from the original input X_0. This difference is stored in the register Y_1. The second parallel process of weight reduction also occurs simultaneously. The original value of weight Z_0 is subtracted by 2^{-1} and then stored in the Z_1. This is the result of the first iteration. After this, the process goes on till the weight is reduced to zero, which leads to the computation of the final multiplier product.

Chapter 5

Experimental Evaluation of the Novel MAC Unit in DNNs

We have investigated the parameters to enhance the performance and increase the hardware efficiency by utilizing the novel architecture of the MAC unit in DNN accelerators. The experimental evaluation has been conducted with both software and hardware-based implementations. Firstly, the proposed MAC has been validated using the Python-based QKeras library for the signed 8-bit fixed-point arithmetic at the network level. Secondly, the novel multiplier has been individually implemented to assess its performance and implementation feasibility on hardware using *Verilog-HDL* language. Lastly, the proposed design has been scripted in *Verilog-HDL* language and simulated at the RTL level using the Xilinx Vivado tool. Additionally, FPGA synthesis and implementation using the $Xilinx - Vivado$ tool have been performed. The following evaluations have been carried out:

1. We conducted an evaluation of the inference accuracy of the Lenet-5 CNN model [27] trained separately using the MNIST database and CIFAR-10 databases. The accuracy of the architecture in Python was obtained using standard *TensorFlow* libraries [28]. Additionally, the CORDIC-based bit-precise design was replicated in Python to determine the exact accuracy of this implementation.

2. The CORDIC-based proposed MAC unit was initially tested for simulation and behavior within a simple neuron structure. For hardware performance evaluation, a *Verilog* code was created and emulated on the FPGA Xilinx Virtex-7 board.

3. Building upon the previous evaluations of the MAC unit, we integrated each such unit into the Lenet-5 neural network model. Post-implementation performance parameters for the proposed CORDIC-based architecture on the Virtex-7 board were achieved and compared with previous works.

5.1 Software implementation and validation

The software implementation and validation of the proposed iterative MAC architecture have been performed in the Python platform. A pipelined CORDIC-based MAC architecture with eight pipeline stages, each performing right shift-and-add and dynamic bit truncation, was tested using the Lenet-5 model. The implemented pipelined CORDIC-based architecture was first verified in a Tensorflow-CNN model [27]. The model was initially trained using the conventional MAC unit with the MNIST dataset, at the classifier stage. Subsequently, the inference accuracy was observed using the proposed SF in the classification layer. The inference process deployed the proposed MAC unit, which is controlled using a finite-state machine, as explained in the flow shown in Figure5.1.

5.2 Hardware implementation and validation on Xilinx-FPGA board

The MAC unit was implemented on the Xilinx-FPGA board using the *Verilog Hardware Description Language* in the Xilinx Vivado Design suite. The Integrated Design Environment (IDE) of the tool provides various tools from the system level to the Integrated Circuit (IC) level. The key feature of the tool is to

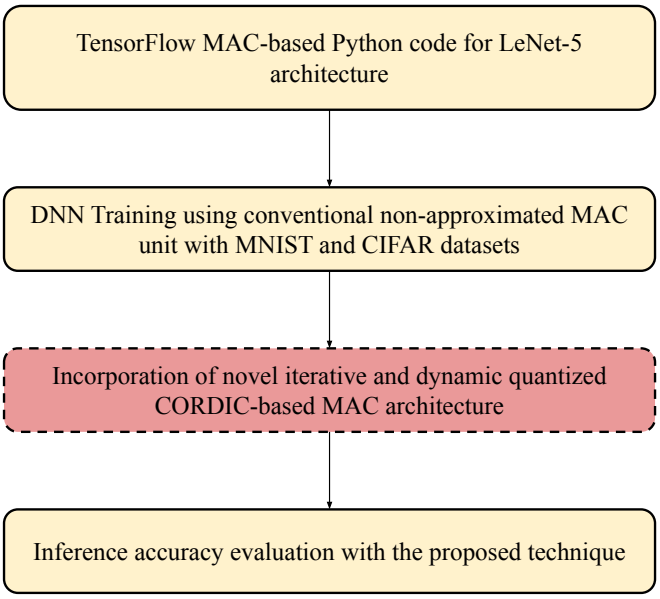

Figure 5.1: Software Evaluation Flow of QuantMAC Design

simulate, synthesize, and implement the HDL design. This implementation was carried out using the Vivado 2018.3 version. The multiplier architecture has been functionally verified (simulated), synthesized, and implemented on the Xilinx Virtex-7 FPGA.

5.2.1 Hardware implementation and performance evaluation for MAC unit

The FPGA flow initiates with the RTL coding of the MAC unit, followed by design synthesis to validate its functionality. Once the design is error-free, a behavioral simulation is conducted using a test bench to verify if the design operates as intended. Subsequently, the design implementation phase assesses the hardware performance of the design, providing crucial performance parameters such as power consumption, critical time delay, and hardware resource utilization. This comprehensive evaluation allows us to ensure that the proposed MAC unit meets the desired criteria and performs optimally in real-world scenarios. The FPGA implementation provides valuable insights into the hardware behavior of the MAC unit, enabling fine-tuning

and optimization for enhanced efficiency and effectiveness. By carefully following the FPGA flow, as shown in Figure 5.2, and evaluating key performance metrics, we can successfully integrate the MAC unit into neural network accelerators, significantly advancing the field of hardware optimization for deep learning.

5.2.2 Integration of Proposed MAC Unit in LeNet-5 CNN Model

The LeNet-5 CNN model is composed of Convolution, Pooling, and Fully connected layers, with memory elements to store the weights obtained from software-based training. To assess the performance of the model using the new design technique, the proposed MAC unit is integrated into the neurons of the LeNet-5 model. This integration follows a similar process as that of the individual proposed MAC unit. Subsequently, the performance parameters of the modified model are compared with those of the conventional MAC unit in the same CNN model. This evaluation aims to determine the impact of the novel MAC architecture on the overall performance of the LeNet-5 CNN model. By analyzing the results, we can gain insights into the efficiency and effectiveness of the proposed design technique in enhancing the neural network's performance.

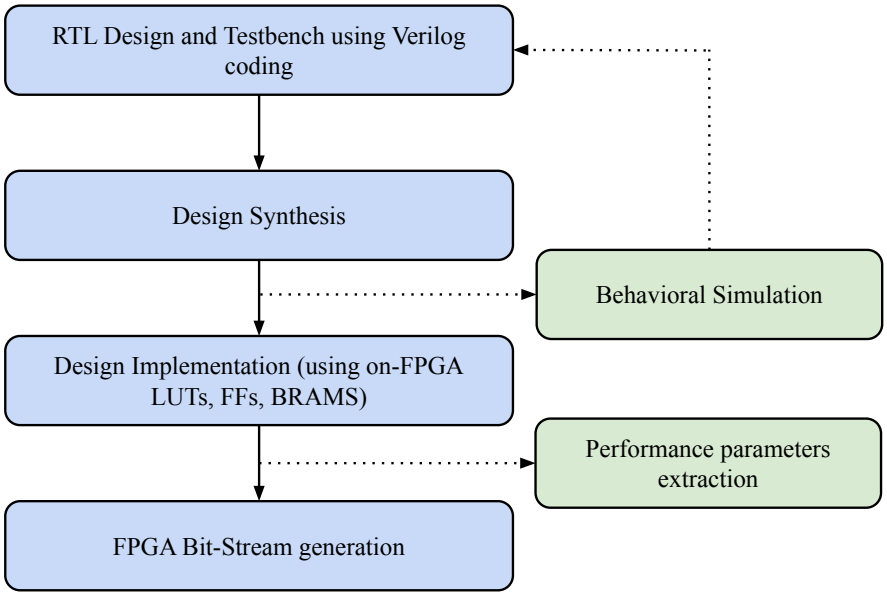

Figure 5.2: Design Flow for FPGA based Implementation

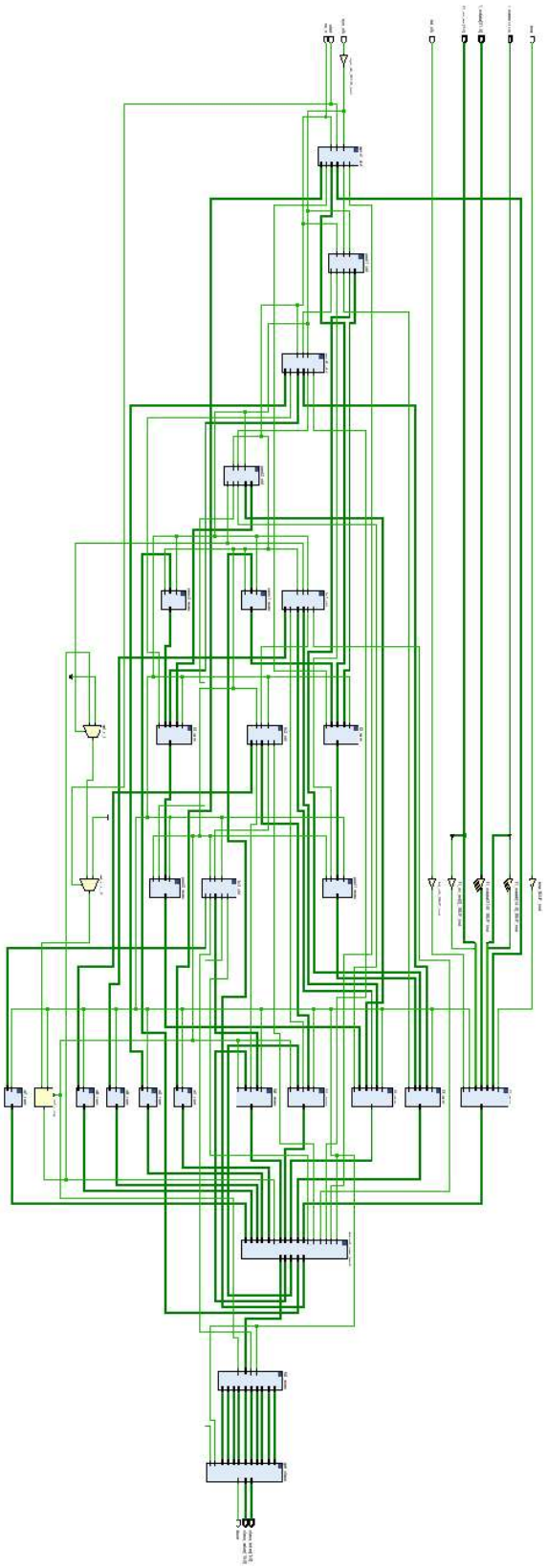

Figure 5.3: RTL Schematic of Lenet-5 using the iterative CORDIC-based MAC architecture

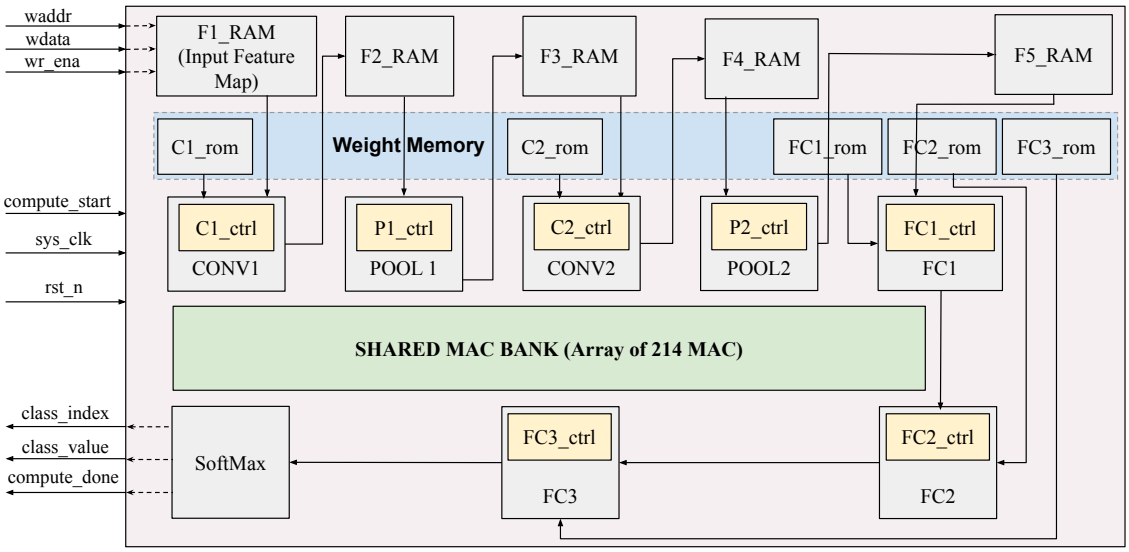

Figure 5.4: Architectural representation of the RTL implemented of Lenet-5

Chapter 6

Result and Discussion

This section provides a detailed analysis of the outcomes obtained from different implementations of the proposed quantized MAC unit. It includes the software implementation results for the CNN model accuracy, hardware implementation results for the proposed quantized MAC, and the combined assessment of the CNN model's hardware implementation using the proposed quantized MAC. These subsections offer valuable insights into the accuracy, hardware performance, and resource consumption aspects of the proposed quantized MAC, facilitating a comprehensive evaluation of its impact on the CNN model. Our novel design of the quantized MAC unit has been named '*QuantMAC*', facilitating easier reference to our proposed architecture.

6.1 Software Implementation Results and Accuracy Comparison

This section presents the emulator results obtained from the *Python* scripting of the LeNet model of CNN for image classification using fixed-point arithmetic representation. The model was trained separately for MNIST and CIFAR-10 datasets, and the accuracy was computed for various bit precisions (i.e., 8, 12, 16 bits) during the inference process using the improved quantized MAC unit. For the MNIST dataset, the accuracy of our QuantMAC architecture experiences a negligible loss

Table 6.1: LeNet-5 model inference accuracy using Proposed Quantize-enabled MAC architecture.

Bit-Precision	**Inference Accuracy@LeNet (%)**			
Dynamic	MNIST		CIFAR-10	
Fixed-Point	TensorFlow	QuantMAC	TensorFlow	QuantMAC
8-bit	98.8	97.2	80.7	78.6
12-bit	98.9	97.6	80.8	79.3
16-bit	98.9	97.8	81.2	79.8

(<1.8%) when moving from 16-bit to 8-bit precision. The 8-bit precision offers approximately a 5× reduction in computing costs and a 4× reduction in memory bandwidth requirements compared to 16-bit precision, making it a suitable fit for smaller datasets. Thus, we developed the newer MAC design for Fully Connected and Convolutional Neural Networks with 8-bit precision. Additionally, 8-bit fixed-point approaches are known to save a significant amount of hardware while maintaining reasonable accuracy [19]. This implies that 16-bit neural network implementations can better handle complex datasets. The inference accuracy comparison between the proposed MAC and conventional MAC architectures is presented in Table 6.1. The proposed design performs similarly in terms of accuracy for the digital computation technique. With 8-bit precision, the proposed MAC architecture exhibits a 1.6% higher accuracy loss than the accurate MAC computation produced by TensorFlow. However, as detailed in the next section, the design shows improvements in physical parameters, including area utilization, throughput, and power consumption.

Furthermore, the table illustrates that with increased bit precision, the deviation in accuracy between the proposed MAC and the standard MAC becomes less significant. This finding further enhances the effectiveness of the suggested MAC architecture for the hardware implementation of deep neural networks. The implementation results

and the evaluated accuracy collectively strengthen the credibility and applicability of the proposed QuantMAC architecture in real-world edge-AI devices.

6.2 Hardware Implementation Results for the Proposed Quantized MAC

The MAC hardware implementation results were conducted on the Virtex-7 FPGA VC707 Evaluation Kit using Vivado-Xilinx with HDL flow. In addition to the proposed architecture, recent and widely used multipliers were also analyzed with 8×8 bit precision for comparison. Performance parameters for various state-of-the-art architectures and the proposed architecture were extracted and compared. Specifically, we compared the MAC performance utilizing the Wallace tree multiplier, the Vedic multiplier, the Booth multiplier, the MAC with shift-and-add, and CORDIC. The extracted performance parameters are presented in Table 6.2. The results show that the state-of-the-art designs have higher utilization and higher power-delay products compared to the proposed architecture. Among them, the modern shift-and-add algorithm-based MAC architecture stands out as one of the most resource and power-efficient architectures, although it has a lower throughput.

The table provides a comprehensive comparison of various hardware utilization and performance parameters for the proposed MAC unit and several other state-of-the-art MAC architectures, all operating at a fixed-point precision of $Q_{1.7}$. The hardware utilization is quantified in terms of Look-Up Tables (LUTs) and Flip-Flops (FFs), while the critical path delay is measured in nanoseconds (ns), indicating the longest delay from inputs to outputs. Additionally, the power-delay product, measured in picojoules (pJ), represents the product of power consumption and critical path delay, offering insights into the overall energy efficiency of each architecture. Examining the results in the Table 6.2, we can observe the following details for each MAC architecture. The Vedic MAC utilizes 159 LUTs and 245 FFs, with a critical path delay of 4.48 ns and a power-delay product of 6.11 pJ. The Wallace

MAC employs 105 LUTs and 112 FFs, featuring a critical path delay of 2.59 ns and a power-delay product of 3.29 pJ. The Booth MAC uses 83 LUTs and 61 FFs, displaying a critical path delay of 3.08 ns and a power-delay product of 3.07 pJ. The Shift-add MAC utilizes 75 LUTs and 58 FFs, featuring a critical path delay of 5.44 ns and a power-delay product of 4.17 pJ. Lastly, the CORDIC MAC employs 23 LUTs and 22 FFs, with a critical path delay of 9.06 ns and a power-delay product of 1.90 pJ. Notably, the proposed MAC architecture stands out, highlighted in green, with its utilization of 52 LUTs and 88 FFs. It boasts an impressively low critical path delay of 1.53 ns, combined with a remarkably efficient power-delay product of 1.01 pJ. These results illustrate the superior hardware efficiency and energy-saving potential of the proposed MAC unit, making it a promising candidate for implementation in deep neural networks.

The proposed implemented solution showcases significant advantages over the traditional shift-and-add algorithm concerning hardware utilization, path delay, and power-delay product. Despite having a higher utilization rate compared to the CORDIC design, the proposed method is still based on the CORDIC concept. However, its data pipeline-enabled architecture enables our suggested system to surpass the CORDIC design's limitation in providing high throughput. Further, in order to assess the precision scalability of performance metrics, we designed both the conventional and novel architecture of the MAC unit for multiple precisions, namely 8-, 12-, and 16-bit. The comparison of outcomes for various fixed-point arithmetic bit-precisions is presented in Table 6.3. It is evident that the proposed method achieves space and power savings for all-bit precisions, around 37% and 3.6% less, respectively than the exact MAC for 8-bit precision. Furthermore, the observed delay for this architecture is below that of the actual MAC. Additionally, the performance metrics of the proposed design exhibit more observable outcomes for higher precisions, demonstrating its adaptability and efficiency across different precision levels.

The conventional MAC architecture shows a 5-fold increase in utilization when doubling the precision. In contrast, the proposed architecture exhibits a resource

Table 6.2: Hardware utilization and performance parameters at 'fixed-point $Q_{1.7}$' for proposed MAC and other State-of-the-art MACs

Resources Utilization	**LUT** (17600)	**FF** (35200)	**Critical Path Delay** (ns)	**Power-delay Product** (pJ)
Vedic [11]	159	245	4.48	6.11
Wallace [9]	105	112	2.59	3.29
Booth [12]	83	61	3.08	3.07
Shift-add [29]	75	58	5.44	4.17
CORDIC [21]	23	22	9.06	1.90
Proposed	52	88	1.53	1.01

usage reduction of less than 2 times while achieving increased accuracy from 8 to 16 bits, as evident from Table 6.3. This demonstrates that the proposed method is hardware-efficient for all bit precisions with fixed-point representation.

6.3 Hardware Implementation Results and Performance Comparison

The QuantMAC-based LeNet-5 architecture was implemented using HDL on the Virtex-7 FPGA VC707 Evaluation kit. Hardware parameters such as throughput, power usage, and resource utilization are obtained and analyzed. A detailed comparison was made between the performance of the proposed QuantMAC, the conventional combinational logic-based MAC, and the MAC utilizing the shift-and-add algorithm in the LeNet implementation. The Table 6.4 presents the hardware utilization results for different LeNet-5 architectures implemented on FPGA, including the proposed QuantMAC architecture and other two state-of-the-art MAC designs. The hardware utilization for Look-Up Tables (LUTs), Flip-Flops (FFs),

Table 6.3: Performance parameters comparison for Proposed QuantMAC and State-of-the-art MAC architectures [1]

Bit Precision	**LUT** (17600)	**FF** (35200)	**Critical Path Delay** (ns)	**Total Dynamic Power** (mW)
8-bit Acc. [1]	86	16	2.361	6.6
8-bit Proposed	52	88	1.53	6.36
12-bit Acc. [1]	187	24	2.587	10.26
12-bit Proposed	75	126	1.81	8.48
16-bit Acc. [1]	325	32	2.8917	14.95
16-bit Proposed	95	168	2.16	11.77

Block Random Access Memory (BRAM), and total power consumption (in Watts) are reported. The implementation of the MAC architecture with combinational logic, using the shift-and-add algorithm, is reported and compared with the proposed design. The results for the proposed QuantMAC architecture show nearly the same hardware resources but come with enhanced throughput since it has a shorter critical path delay, as discussed in Section 6.2. For each architecture, the corresponding LUT, FF, BRAM utilization, and total power consumption values are presented. These hardware utilization metrics provide valuable insights into the efficiency and resource consumption of the different MAC designs on the FPGA platform. The results revealed that for an 8-bit precision architecture, the proposed QuantMAC consumed 64% fewer resources and experienced a minor accuracy loss of less than 1.8% compared to the conventional combinational logic-based design. However, the throughput was significantly improved, showing a 3.56x increase when compared to the shift-and-add approach. This enhancement came at a cost of 13.8% resource overhead, as indicated in Table 6.4. Furthermore, an investigation into parallel processing in the MAC computation demonstrated minimal area overhead and led

Table 6.4: Hardware utilization results for LeNet-5 architectures using Proposed MAC and other State-of-the-art MAC on FPGA

LeNet-5 Implementation	**LUT** (303600)	**FF** (607200)	**BRAM (135)**	**Total Power (W)**
MAC with Comb. Logic [1]	35399	19459	40	0.185
MAC with Shift-and-Add [29]	10964	19790	15.5	0.180
Proposed QuantMAC	12480	19770	39	0.181

to achieving the least critical delay for the proposed design.

The Table 6.5 presents a bit-precision scalability report of the Quantized MAC on the LeNet-5 network using HDL. The table includes three different bit precisions: 8-bit, 12-bit, and 16-bit. For each precision, the corresponding LUT and FF utilization values are provided, along with the critical path delay in nanoseconds (ns). For the 8-bit precision QuantMAC, it was observed that it utilized 12,480 LUTs and 19,770 FFs, with a critical path delay of 7.349 ns. Moving to higher precision, the 12-bit QuantMAC required 19,985 LUTs and 26,470 FFs, resulting in a slightly increased critical path delay of 8.049 ns. For the highest precision, the 16-bit QuantMAC utilized 29,838 LUTs and 38,678 FFs, with a critical path delay of 8.964 ns.

Comparing the results, we can see that as the bit precision increases, there is an increment in resource utilization, as reflected in the increase in LUTs and FFs. These calculations show that, when transitioning from 8-bit to 16-bit precision in the Quantized MAC, there is approximately a 138.27% increase in LUT utilization (i.e 2.5×) which means the proposed MAC-based model shows a much more efficient scaling in hardware requirements, with only a 2.5× increase when doubling the bit precision, compared to the 4× to 5× increase observed in the model with conventional MAC [21]. Further 95.70% increase in FF utilization, and a 21.95% increase in critical

Table 6.5: Bit-precision scalability report of Quantized MAC on LeNet-5 network using HDL

Bit Precision	**LUT** (303600)	**FF** (607200)	**Critical Path Delay** (ns)
8-bit QuantMAC	12480	19770	7.349
12-bit QuantMAC	19985	26470	8.049
16-bit QuantMAC	29838	38678	8.964

path delay. This information is valuable in understanding the trade-offs between precision and resource usage in the design of the MAC unit. However, the critical path delay also increases, indicating a slight reduction in the performance speed. The table highlights the trade-off between precision and resource usage, enabling us to choose the appropriate bit precision that best suits the specific requirements of the LeNet-5 network in terms of hardware resources and performance. The proposed QuantMAC design exhibits minimal resource impact and critical delay when scaling bit precision. It proves to be effective for DNN acceleration applications, offering efficient scaling solutions.

Chapter 7

Conclusion and Future scope of the proposed work

The bit-truncation or approximation approach adopted in this study showcases a noteworthy reduction in hardware resources and power consumption, with only a negligible loss in accuracy due to the inherent error resilience of neural networks. For achieving an efficient MAC design, the bit-truncated architecture was implemented, incorporating data quantization, reduced area, and power overhead, as well as a right shift and accumulation architecture. By exploring the pipeline architecture, throughput was significantly improved, employing the shift-and-add iterative process in a pipelined structure without compromising accuracy. The resulting suggested architecture proved highly efficient, as the 8-bit and higher precision MAC unit demonstrated impressive results in inference accuracy for picture categorization. A comparison of the proposed QuantMAC with conventional combinational logic-based MAC and the MAC using the shift-and-add algorithm in LeNet implementation yielded compelling insights. For an 8-bit precision architecture, QuantMAC exhibited 64% fewer resource consumption and less than 1.8% accuracy loss compared to conventional combinational logic-based design. Moreover, throughput was boosted by 3.56× compared to the shift-and-add approach, albeit at a cost of 13.8% resource overhead, as indicated in Table 6.4. By exploring parallel processing in the MAC computation with minimal area overhead, we achieved the least critical delay. While

the proposed work presents notable achievements, it also has certain limitations at both the implementation and design levels. Nevertheless, these shortcomings provide opportunities for future scope and extensions of this project. Overall, the hardware-efficient quantize-enabled MAC unit holds great potential in benefiting edge computing solutions.

The future scope of this project entails the development of a solution that encompasses several key features. Firstly, it aims to implement data-type-generic hardware, facilitating easier adaptation to different data types and increasing versatility. Secondly, the solution will incorporate a dynamic error compensation process to enhance accuracy in its operations. Lastly, the design will be engineered with the potential for reuse in adder circuitry, optimizing resource utilization and promoting overall efficiency. By encompassing these attributes, the future solution aspires to offer a powerful and adaptable hardware solution with broad applicability. Key future shortcomings that could be explored are as follows:

- **ASIC aspect exploration**: Future work includes training the neural network with a standard image dataset and implementing the proposed MAC unit for image classification on an FPGA hardware accelerator. Exploring an ASIC-level implementation will offer valuable insights into system performance. Efforts can focus on deducing CMOS-level performance parameters for better feasibility on an FPGA. Comparing ASIC and FPGA implementations will aid in selecting the optimal hardware accelerator solution. Extending the design for ASIC flexibility with data size and DNN models will enhance overall performance. These endeavors will advance hardware optimization and broaden the application potential of the neural network.

- **Hardware Reusability and Hardware Optimization**: Addressing hardware consumption concerns for implementing larger neural networks on smaller edge-AI devices can be achieved through various techniques like data multiplexing, resource sharing, data reuse, and hardware reuse [30, 31]. Reusability of hardware, either at the small unit or sub-architecture level, can significantly

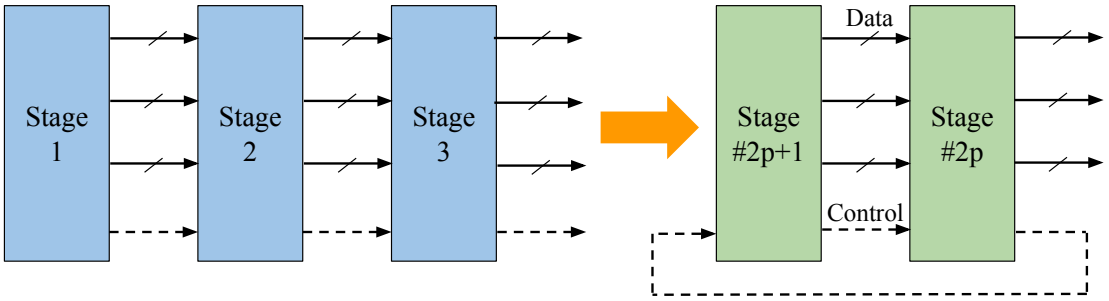

Figure 7.1: Structural reuse in a repetitive process to save hardware resources

reduce resource consumption. The iteration stage in the pipeline of a neural network is repetitive across all neurons and layers. Leveraging this iterative process, hardware reusability can be optimized by sharing the same blocks for multiple computations or merging with network pruning. Though the implementation algorithm may become more complex, substantial on-chip area savings can be achieved. However, it's essential to consider the trade-offs between hardware consumption, computation complexity, latency, and accuracy to develop an efficient hardware-reusable architecture. This can be seen from the Figure 7.1. Here $p = 0, 1, 2, 3...$, which is half the number of pipeline stages inside the MAC unit. This can be seen to reduce the number of resources for various stages at greater bit-width.

- **Extended Design in Feasibility for Industry**: An important avenue to explore is the integration of newer data arithmetic types to replace conventional fixed and floating-point representations. Efficient arithmetic representations such as Brain float, POSIT, and TensorFloat, which are gaining traction in various industries, could be incorporated into the proposed design to enhance their applicability for current industry-standard designs. Generalizing the data arithmetic used in the design would enable its feasibility for a wide range of applications and diverse databases used in neural network training. However,

researchers should carefully consider the potential increase in computation complexity when accommodating different data types. Alternatively, investigating the implementation of the novel methodology for a single newer data arithmetic type, and validating its accuracy, would demonstrate the design's flexibility in terms of data type arithmetic. This future exploration promises to open up new possibilities for more versatile and efficient neural network designs, catering to the evolving demands of practical applications.

Bibliography

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

[2] 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.

[3] N. I. Chervyakov, P. A. Lyakhov, M. A. Deryabin, N. Nagornov, M. V. Valueva, and G. V. Valuev, "Residue number system-based solution for reducing the hardware cost of a convolutional neural network," *Neurocomputing*, vol. 407, pp. 439–453, 2020.

[4] 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.

[5] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, 2018.

[6] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[7] M. Masadeh, O. Hasan, and S. Tahar, "Input-conscious approximate multiply-accumulate (MAC) unit for energy-efficiency," *IEEE Access*, vol. 7, pp. 147 129–147 142, 2019.

[8] 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.

[9] R. B. S. Kesava, B. L. Rao, K. B. Sindhuri, and N. U. Kumar, "Low power and area efficient wallace tree multiplier using carry select adder with binary to excess-1 converter," in *2016 Conference on Advances in Signal Processing (CASP)*. IEEE, 2016, pp. 248–253.

[10] V. Kunchigi, L. Kulkarni, and S. Kulkarni, "High speed and area efficient vedic multiplier," in *2012 International Conference on Devices, Circuits and Systems (ICDCS)*. IEEE, 2012, pp. 360–364.

[11] 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.

[12] F. U. D. Farrukh, C. Zhang, Y. Jiang, Z. Zhang, Z. Wang, Z. Wang, and H. Jiang, "Power efficient tiny yolo cnn using reduced hardware resources based on booth multiplier and wallace tree adders," *IEEE Open Journal of Circuits and Systems*, vol. 1, pp. 76–87, 2020.

[13] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.

[14] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[15] R. Pilipović, P. Bulić, and U. Lotrič, "A two-stage operand trimming approximate logarithmic multiplier," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 6, pp. 2535–2545, 2021.

[16] R. Pilipović and P. Bulić, "On the design of logarithmic multiplier using radix-4 booth encoding," *IEEE access*, vol. 8, pp. 64 578–64 590, 2020.

[17] M. P. Véstias, "A survey of convolutional neural networks on edge with reconfigurable computing," *Algorithms*, vol. 12, no. 8, p. 154, 2019.

[18] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5784–5789, 2018.

[19] 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.

[20] N. Bruschi, A. Garofalo, F. Conti, G. Tagliavini, and D. Rossi, "Enabling mixed-precision quantized neural networks in extreme-edge devices," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 217–220.

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

[22] H. Xie, Y. Song, L. Cai, and M. Li, "Overflow aware quantization: Accelerating neural network inference by low-bit multiply-accumulate operations," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 868–875.

[23] 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.

[24] 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.

[25] Y. Xue and Z. Ma, "Design and implementation of an efficient modified cordic algorithm," in *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*. IEEE, 2019, pp. 480–484.

[26] S. Mehra, G. Raut, R. Das, S. K. Vishvakarma, and A. Biasizzo, "An empirical evaluation of enhanced performance softmax function in deep learning," *IEEE Access*, 2023.

[27] Kaggle, "https://www.kaggle.com/kedarsai/cifar-10-88-accuracy-using-keras," 2022.

[28] F. Ertam and G. Aydın, "Data classification with deep learning using tensorflow," in *2017 international conference on computer science and engineering (UBMK)*. IEEE, 2017, pp. 755–758.

[29] D. A. Gudovskiy and L. Rigazio, "Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks," *arXiv preprint arXiv:1706.02393*, 2017.

[30] 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*, 2021.

[31] A. M. Zyarah and D. Kudithipudi, "Resource sharing in feed forward neural networks for energy efficiency," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2017, pp. 543–546.