# AN EXPLORATION OF EFFICIENT COMPUTING TECHNIQUES FOR IMPLEMENTING DEEP NEURAL NETWORK ACCELERATORS

**Ph.D. Thesis**

By

**VASUNDHARA TRIVEDI**


DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

**February 2026**

# AN EXPLORATION OF EFFICIENT COMPUTING TECHNIQUES FOR IMPLEMENTING DEEP NEURAL NETWORK ACCELERATORS

**A THESIS**

*Submitted in partial fulfillment* of the

*requirements for the award of the degree*

***of***

**DOCTOR OF PHILOSOPHY**

by


## VASUNDHARA TRIVEDI

**DEPARTMENT OF ELECTRICAL ENGINEERING**

**INDIAN INSTITUTE OF TECHNOLOGY INDORE**


**February 2026**

# INDIAN INSTITUTE OF TECHNOLOGY INDORE

## CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **AN EXPLORATION OF EFFICIENT COMPUTING TECHNIQUES FOR IMPLEMENTING DEEP NEURAL NETWORK ACCELERATORS** in the partial fulfillment of the requirements for the award of the degree of **DOCTOR OF PHILOSOPHY** and submitted in the **DEPARTMENT OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from December 2021 to February 2026 under the supervision of Prof. Santosh Kumar Vishvakarma, Professor, Indian Institute of Technology Indore, Indore, India.

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

Vasundhara
11/02/2026

Signature of the Student with Date

**(Vasundhara Trivedi)**

---

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

Santh
11/02/2026

Signature of Thesis Supervisor with Date

**(Prof. Santosh Kumar Vishvakarma)**

---

**Vasundhara Trivedi** has successfully given his Ph.D. Oral Examination held on 15/06/2026

Santh
15/06/2026

Signature of Thesis Supervisor with Date

**(Prof. Santosh Kumar Vishvakarma)**

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my Ph.D. Thesis Supervisor, Prof. Santosh Kumar Vishvakarma, for his invaluable guidance and unwavering support throughout my doctoral tenure. His inspirational leadership and insightful direction shaped my research trajectory with purpose and excellence.

I extend my sincere gratitude to Prof. Akash Kumar, Ruhr University Bochum (RUB), Germany, for his insightful suggestions and for providing me with the opportunity to work as a visiting researcher at RUB. I am grateful to my PSPC evaluation committee members, Prof. Vivek Kanhangad and Prof. Bodhisatwa Mazumdar, for their valuable comments and suggestions during my research journey. I am grateful to to the Director of IIT Indore, faculty members, and the Indian Institute of Technology Indore staff for their immense cooperation throughout my thesis work.

I thank my senior colleague, Dr. Gopal Raut, for his continued support, guidance and constructive feedback throughout this journey. I gratefully acknowledge my friends, Mr. Akhila Gouda and Dr. Sandeep Semwal, who were constant sources of support and inspiration during my time at IIT Indore. I express my heartfelt thanks to my friends, especially Arghya, Harshit, Khushbu, Avikshit, Krutika, Madhur and Shally for their encouragement and belief in me throughout this journey.

This work would not have been possible without the endless love, encouragement, and invaluable support of my parents, Mr. Yogesh Trivedi and Prof. Preeti Trivedi. I express my deepest thanks to them, as they are my true inspirations and my greatest source of strength throughout my life. It is indeed their selfless sacrifices that made this journey possible. I am deeply thankful to my brother, Mr. Nitish Kumar Trivedi, who has been the most humorous person in my life and has always been by my side during challenging times. I am forever grateful for the blessings of my late grandparents and my Gurus, who have shaped the trajectory of my life with grace.

Lastly, I express my profound gratitude to the Almighty, who has bestowed upon me life, faith, strength, and the ability to navigate through this crucial journey.

***Vasundhara Trivedi***

*Dedicated to My Parents*

# ABSTRACT

Modern artificial intelligence (AI) applications rely heavily on deep neural networks (DNNs) to perform complex computational tasks. Although DNNs deliver high accuracy, they do so at the cost of substantial power consumption and significant hardware resource utilization, making their edge implementation a major challenge. Edge devices such as Internet of Things (IoT) nodes, mobile platforms, and embedded systems operate under stringent constraints on power, memory, and computational capability. Consequently, deploying DNNs at the edge requires highly efficient computing architectures and optimization techniques that reduce energy consumption and hardware overhead while preserving acceptable levels of inference accuracy and real-time performance.

This necessitates optimising DNNs to achieve optimal resource and power consumption while minimising accuracy degradation. Owing to the limited ability of traditional CPUs to efficiently process large volumes of data at high speed, modern DNN workloads increasingly require hardware acceleration. To meet these growing computational demands, a wide range of hardware acceleration platforms have been developed to offload compute-intensive operations from general-purpose CPUs. These platforms—including Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), and Graphics Processing Units (GPUs)—offer higher computational throughput and improved energy efficiency. Consequently, efficient utilization and architectural design of such accelerators are critical for enabling scalable, high-performance DNN processing in complex and data-intensive applications.

DNNs comprise multiple layers, each consisting of numerous neurons. Each neuron includes a multiply–accumulate (MAC) unit followed by an activation function block. MAC units serve as the fundamental computational building blocks of DNNs and account for a significant portion of power consumption and hardware resource utilization. Consequently, optimizing the MAC unit is central to the efficient implementation of DNN accelerators. At the same time, due to the large number of MAC

operations in DNN workloads, sequential execution becomes inefficient, resulting in high latency and limited throughput. As a result, operational parallelism is critical for the efficient implementation of DNNs, particularly in speed-sensitive applications.

To address these challenges, numerous techniques have been proposed in the state-of-the-art to reduce resource utilization and power consumption while enhancing throughput, typically with minimal degradation in accuracy. These techniques exploit the inherent error resilience of DNNs and include approximate arithmetic, precision scaling, and architectural optimisation. By carefully leveraging approximation and parallel execution, it is possible to achieve improved energy efficiency and performance, making MAC-level optimisation a key enabler for scalable, high-performance DNN accelerator designs.

This thesis provides a comprehensive overview of research on efficient computing techniques for DNN accelerators, with a primary focus on optimising the MAC unit and enhancing computational parallelism. Approximate computing techniques at the MAC level and the Processing Element (PE) level are explored to reduce power consumption, resource utilisation, and computational complexity while maintaining acceptable inference accuracy. In addition, architectural strategies are investigated to increase parallelism in DNN computations through SIMD (Single Instruction Multiple Data) architectures, thereby mitigating performance bottlenecks associated with large-scale MAC operations and enhancing DNN throughput.

By jointly exploiting approximation and parallel execution, this thesis presents approaches aimed at achieving a balanced trade-off among accuracy, energy efficiency, and throughput, thereby enabling high-performance and resource-efficient DNN accelerator designs.

# LIST OF PUBLICATIONS

## (A) Publications from PhD thesis work

### *A1. In refereed Journals:*

1. **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 (CSSP), Springer*, Volume 42, Pages 7596-7614, August 2023.
   D.O.I. : 10.1007/s00034-023-02469-1.(**SCI, I.F. 2.0**)

2. **Vasundhara Trivedi**, Harman Singh Bagga, Gopal Raut, Santosh Kumar Vishvakarma, "Adaptive-Precision SIMD Architecture for High-Throughput and Resource-Efficient DNN Acceleration", *Integration, Elsevier*, Volume 108, p.102666, January 2026.
   D.O.I: 10.1016/j.vlsi.2026.102666(**SCI, I.F. 2.5** ).

3. **Vasundhara Trivedi**, Gopal Raut, Baker Mohammad, Santosh Kumar Vishvakarma, Akash Kumar, "C-SIMD: CORDIC-Driven SIMD Processing Element for Resource-Efficient Multi-Precision Deep Learning Inference", *IEEE Access*, Volume 14 : 19015 - 19029. January 2026.
   DOI: 10.1109/ACCESS.2026.3653253. (**SCI, I.F. 3.6**)

4. **Vasundhara Trivedi**, Santosh Kumar Vishvakarma, "ACSAM: Accuracy-configurable Segmentation-based Approximate Multiplier for Error-resilient Edge-AI Applications", *IEEE Embedded Systems Letters*, March 2026.
   DOI: 10.1109/LES.2026.3676574 .(**SCI, I.F. 2.0**).

### *A2. In refereed conferences:*

1. **Vasundhara Trivedi**, Santosh Kumar Vishvakarma, "Design of Operand-rounding based Approximate Multiplier for Error-resilient Applications", *29th*

*International Symposium on VLSI Design and Test, 2025*, Aug 6-9, 2025 Chandigarh, India (Accepted and In Press)

## (B) Other publications during PhD

### *B1. In refereed Journals:*

1. Neha Ashar, Gopal Raut, **Vasundhara Trivedi**, Santosh Kumar Vishvakarma, Akash Kumar, "QuantMAC: Enhancing Hardware Performance in DNNs With Quantize-Enabled Multiply-Accumulate Unit", *IEEE Access*, Volume 12 (2024): 43600-43614.
   D.O.I.: 10.1109/ACCESS.2024.3379906. (**SCI, I.F. 3.6**)

# TABLE OF CONTENTS

# List of Figures

# List of Tables

# List of Abbreviations

**AI:** Artificial Intelligence

**ML:** Machine Learning

**NN:** Neural Network

**ANN:** Artificial Neural Network

**DNN:** Deep Neural Network

**CNN:** Convolutional Neural Network

**RNN:** Recurrent Neural Network

**MAC:** Multiply-Accumulate

**ASIC:** Application-Specific Integrated Circuit

**VLSI:** Very Large Scale Integration

**FPGA:** Field Programmable Gate Array

**DVFS:** Dynamic Voltage and Frequency Scaling

**SoC:** System on Chip

**CPU:** Central Processing Unit

**RISC-V:** Reduced Instruction Set Computer–V

**MSB:** Most Significant Bit

**LSB:** Least Significant Bit

**GPU:** Graphics Processing Unit

**TOPS:** Tera Operations Per Second

**GOPS:** Giga Operations Per Second

**MAC:** Multiply-Accumulate

**AF:** Activation Function

**PE:** Processing Element

**SIMD:** Single Instruction, Multiple Data

**PSNR:** Peak Signal-to-Noise Ratio

**MSE:** Mean Square Error

**SD:** Standard Deviation

**NMSE:** Normalized Mean Squared Error

**NMED:** Normalized Mean Euclidean Distance

**VED:** Vertical Edge Detection

**HED:** Horizontal Edge Detection

**LUT:** Look-Up Table

**CLB:** Configurable Logic Block

**FA:** Full Adder

**LOA:** Lower Part OR Adder (LOA

**HADD:** Hybrid Adder

**AC:** Approximate Computing

**IoT:** Internet of Things

**RAM:** Random Access Memory

**DRAM:** Dynamic Random-Access Memory

**CORDIC:** Coordinate Rotation Digital Computer

**FC:** Fully-Connected

**FFT:** Fast Fourier Transform

**FxP:** Fixed-Point

**FP:** Floating Point

**DSP:** Digital Signal Processing

**GEMM:** General Matrix–Matrix Multiplication

**MNIST:** Modified National Institute of Standards and Technology

**CIFAR:** Canadian Institute For Advanced Research

**VGG-16:** Visual Geometry Group (VGG) 16-layer network

**IEEE-754:** Institute of Electrical and Electronics Engineers Standard 754 for Floating-Point Arithmetic

# List of Symbols

$\tau_c$ Total propagation delay for computing the carry-bit of the proposed Hybrid Adder

$\tau_s$ Total propagation delay for computing the sum-bit of the proposed Hybrid Adder

$t_c$ Total propagation delay for computing the carry-bit of conventional 1-bit Adder

$t_s$ Total propagation delay for computing the sum-bit of conventional 1-bit Adder

$T_c$ Total propagation delay for computing the carry-bit of proposed 1-bit Adder

$T_s$ Total propagation delay for computing the sum-bit of the proposed 1-bit Adder

$f$ Activation Function

$N(l)$ Number of Neurons per Layer

$J(l)$ Number of Inputs per Layer

$X_n$ Input Activations

$Y_n$ Accumulated Partial Sum (including the bias term)

$\delta_n$ Control variable that determines direction of the $n^{\text{th}}$ micro-rotation

$d_n$ Direction control signal for addition

$W_n$ Weight corresponding to input activation $X_n$

# Chapter 1

# Introduction

## 1.1 Overview

In today's technology-driven society, artificial intelligence (AI) applications are ubiquitous across many aspects of daily life, fundamentally transforming and facilitating a wide range of everyday activities. Artificial Intelligence is defined as the science and engineering of creating intelligent machines capable of achieving goals in a manner comparable to human intelligence—a concept first introduced by John McCarthy in the 1950s. Within this broad field, Neural Networks (NNs) represent a subdomain and form the basis of many modern AI systems [1]. The relationship between deep learning and the overall landscape of artificial intelligence is illustrated in Fig. 1.1.

Deep learning is a specialized class of Neural Network models, consisting of multiple hidden layers. These layered architectures enable the automatic extraction of hierarchical features from raw data, eliminating the need for manual feature engineering [2]. As a result, Deep Learning has become central to recent developments in AI for analyzing complex data across multiple application areas.

Following their landmark successes in speech and image recognition, Deep Neural Networks (DNNs) have seen rapid adoption across diverse domains [3]. Today, DNNs are integral to applications such as autonomous driving, natural language processing, medical diagnosis and text-to-speech translation [2, 4]. Overall, DNN-based systems have demonstrated superior performance in many of these areas, in some cases ap-

proaching or even exceeding human-level accuracy [2].

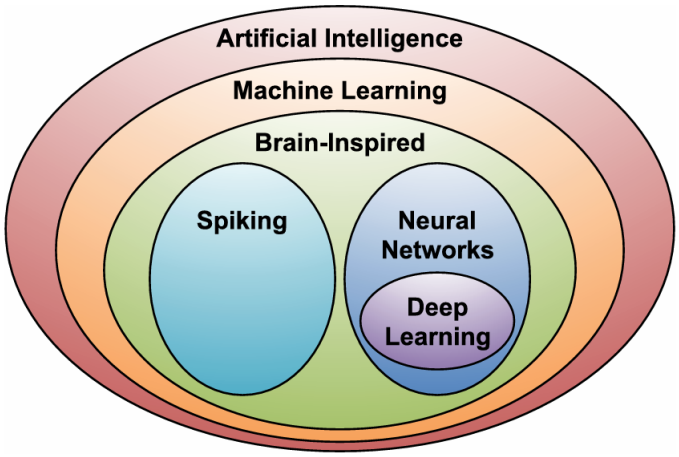



Figure 1.1: Deep Learning in the context of Artificial Intelligence [2].

## 1.2 Deep Neural Networks

Deep Neural Networks (DNNs) employ deep, multi-layered architectures and are therefore well-suited for complex, data-driven tasks. Their architectures and model sizes vary by application and continue to evolve rapidly to improve both accuracy and computational efficiency [2]. Regardless of their specific design, a DNN operates on numerical input values that encode the information to be processed by the network. Inspired by biological neural networks composed of interconnected neurons, DNNs consist of multiple layers, each containing multiple artificial neurons [5].

Like their biological counterparts, artificial neurons process incoming information [6]. An Artificial Neuron typically comprises a Multiply–Accumulate (MAC) unit followed by an activation function (AF) block. The MAC unit multiplies input values by their corresponding weights and forwards the accumulated result to the AF block for further processing. Consequently, MAC operations are fundamental to computing the outputs of convolution operations [7].

A typical convolution operation and the layers involved in a CNN architecture are illustrated in Fig. 1.2. Convolutional Neural Networks (CNNs) are a subcategory of

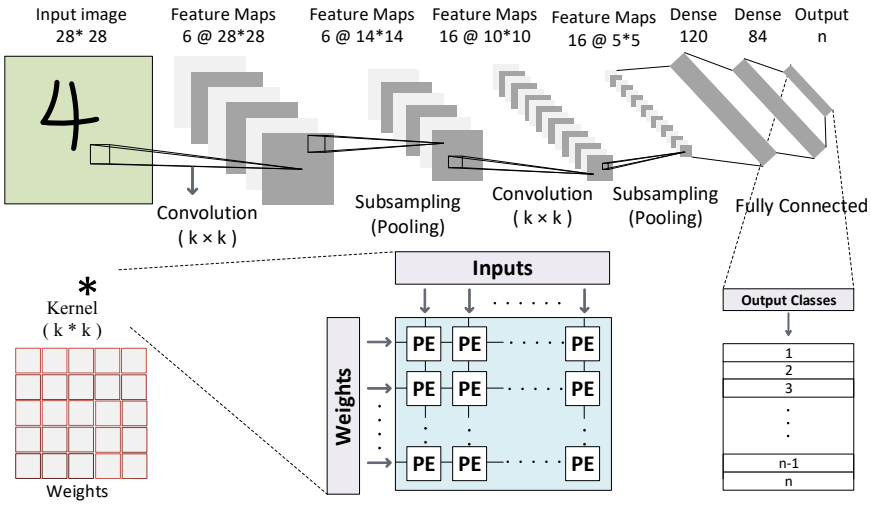



Figure 1.2: CNN architecture with convolution and fully connected layers, an input image, and an output classification layer. The Processing Elements (PEs) are responsible for performing the MAC computations.

DNNs. The convolution operation employs an array of processing engines (PEs) that execute in parallel, with each PE comprising MAC units responsible for data processing. The three types of layers in CNNs are: convolutional layers, subsampling layers, and fully connected layers. The convolution layer (CONV) uses filters that perform convolution operations, and the resulting output is called a feature map [8]. The subsampling layers, also known as pooling layers, perform downsampling, typically applied after a convolutional layer, thereby introducing spatial invariance. In particular, max and average pooling are special kinds of pooling that take the maximum and average values, respectively [9].

The fully connected (FC) layer operates on a flattened input, where each input is connected to all neurons. FC layers are usually located near the end of CNN architectures and can be used to optimise objectives such as class scores [11]. The convolution operation for an input of size $[I_x \times I_y \times M]$ using $Z$ filters of size $[K_x \times K_y \times M]$ is illustrated in Fig. 1.3. This operation produces an output feature map with a depth of $Z$, which is subsequently followed by a pooling operation.

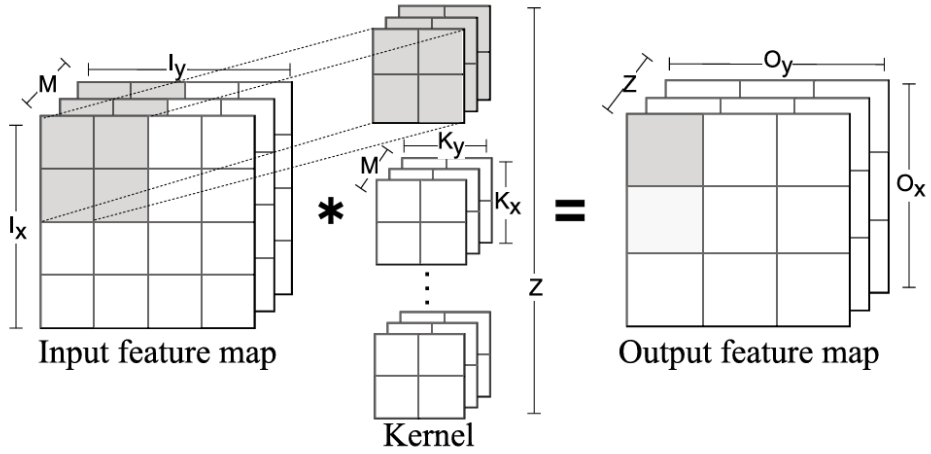


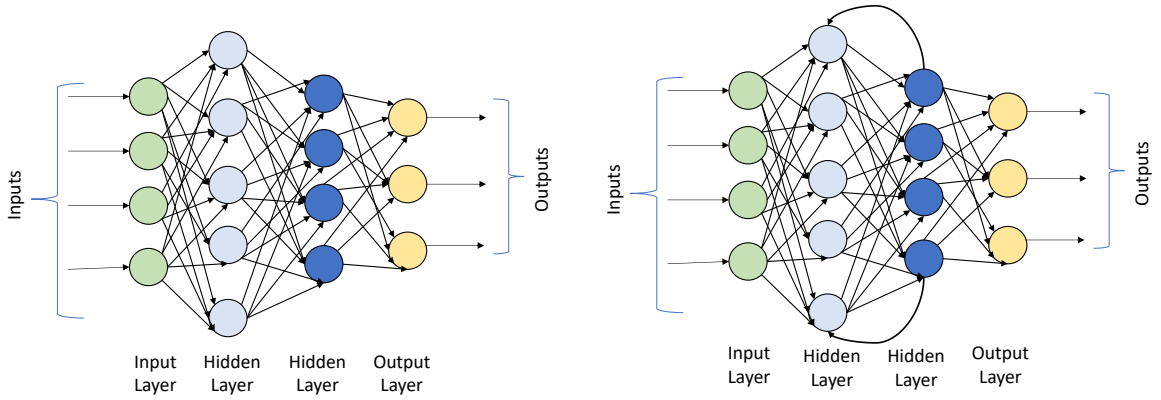

Figure 1.3: A Typical Convolution operation in a Convolutional Layer [10].

### 1.2.1 Types of Deep Neural Networks

Deep neural networks (DNNs) can be broadly categorized into three principal architectures that are most widely adopted in contemporary research and applications: Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs)[12].

- **Multi-Layer Perceptrons (MLPs)**: Multi-Layer Perceptrons (MLPs) are feed-forward artificial neural networks composed of a sequence of fully connected layers, as illustrated in Fig. 1.4(a). It is also called Fully Connected Neural Network (FCNN). In this framework, each neuron applies a nonlinear activation function to a weighted linear combination of the outputs from the previous layer. While the dense inter-layer connectivity enables MLPs to model complex nonlinear relationships, this architecture does not explicitly leverage spatial or temporal dependencies in the input data [13]. As a result, MLPs may exhibit limited scalability and reduced computational efficiency when applied to high-dimensional or structured data.

- **Convolutional Neural Networks (CNNs)**: Convolutional Neural Networks (CNNs) are specifically designed to process images. CNNs utilize convolutional layers to extract local features by performing convolution operations with shared




(a) MLP architecture (b) RNN architecture

Figure 1.4: Comparison of Multi-Layer Perceptron (MLP) and Recurrent Neural Network(RNN) architectures.

weights across different spatial regions of the input, as illustrated in Fig. 1.2 along with brief details in Section 1.2. The other layers in CNNs include subsampling (or pooling) layers along with fully connected layers. In each convolutional layer, nonlinear activation functions are applied to weighted sums of localised subsets of the outputs from the preceding layer, thereby enabling sparse connectivity and parameter sharing. As the network depth increases, successive convolutional layers learn increasingly high-level and abstract representations of the input data. This hierarchical feature-learning capability allows CNNs to automatically extract discriminative features, making them particularly effective for applications such as image classification, face recognition, and semantic segmentation [14].

- **Recurrent Neural Networks (RNNs)**: Recurrent Neural Networks (RNNs) are designed to model sequential and time-series data. Each RNN layer consists of nonlinear activation functions applied to weighted sums of the outputs from the previous layer and the network's prior hidden state, which is computed from earlier inputs and stored in the RNN's internal memory. This recurrent structure enables RNNs to capture temporal dependencies within sequential data [15]. Consequently, RNN-based models are widely used in natural language processing (NLP) tasks, including language modeling, word embedding, and machine translation [16]. The Fig. 1.4(b) illustrates a recurrent neural network in its

folded form, where feedback connections represent the recurrence of the hidden state across time steps.

## 1.3 Need for Hardware Accelerators

The superior accuracy of DNNs comes at the cost of high computational complexity, which poses significant challenges for deployment on resource-constrained edge devices. Many modern applications involve multiple tasks running concurrently, and the overall system performance can be significantly degraded if one computationally intensive task becomes a bottleneck [12]. This limitation arises largely from the sequential execution model and architectural constraints of general-purpose CPUs (Central Processing Units). In conventional systems, the CPU is responsible for executing a majority of computational workloads. While CPUs are highly flexible and capable of handling a wide range of tasks, they alone are not always well-suited for workloads that demand extensive parallelism or extremely high throughput [17, 18].

The key challenges that occur with CPUs as the computing engine include :

- **High Computational Complexity:** DNNs require a massive number of arithmetic operations, primarily MAC operations, during both training and inference. General-purpose CPUs are not optimized to handle such workloads efficiently, resulting in high latency and limited throughput [19].

- **Real-Time Performance Requirements:** Many DNN applications, such as autonomous driving, computer vision, and speech recognition, demand real-time or near-real-time processing. Hardware accelerators enable parallel execution and pipelined computation, significantly reducing inference latency compared to software-based implementations [20].

- **Energy Efficiency Constraints:** The deployment of DNNs on edge devices and embedded systems is constrained by strict power and thermal budgets. Hardware accelerators are specifically tailored to execute neural network operations with reduced energy consumption, making them more suitable than

CPUs for power-sensitive applications [18, 21].

- **Memory Bandwidth Optimization:** DNN workloads involve frequent access to large volumes of data, including weights and activations. Hardware accelerators incorporate optimized memory hierarchies and data reuse strategies, thereby reducing memory bandwidth requirements and improving overall system efficiency [22].

- **Scalability and Model Complexity:** As DNN architectures continue to increase in depth and complexity, the associated computational and memory demands grow significantly. Hardware accelerators provide scalable architectures that support large models while maintaining acceptable performance and power efficiency [18].

- **Efficient Edge and Cloud Deployment:** Hardware accelerators enable efficient deployment of DNNs across a wide range of platforms, from resource-constrained edge devices to high-performance cloud data centres, facilitating scalable and cost-effective artificial intelligence solutions [23].

Hardware acceleration addresses this challenge by offloading specific tasks to dedicated hardware units that are explicitly designed to execute those tasks in an optimized manner [18]. Hardware acceleration refers to the use of specialized hardware components within a computing system to execute particular tasks more efficiently than a general-purpose processor [24]. Few prominent examples of hardware acceleration platforms are:

- **Graphics Processing Unit (GPU):** GPUs were originally developed to accelerate graphics rendering by performing a large number of parallel operations simultaneously. Their highly parallel architecture has since made them valuable for a broader range of applications, most notably in machine learning and data-intensive computations [25]. By integrating many lightweight processing cores, GPUs achieve high computational throughput while maintaining a balance between performance and power efficiency. This inherent parallelism en-

ables substantial speedups in machine learning workloads without a significant loss in efficiency [12].

- **Application-Specific Integrated Circuits (ASICs):** ASICs are custom-designed to perform specific functions with maximum efficiency and minimal power consumption, making them particularly suitable for cryptographic acceleration in systems-on-chip (SoCs) [18, 26]. ASICs leverage dedicated hardware circuit paths to compute specific classes of tasks, achieving high energy efficiency at low power consumption, typically in the milliwatt (mW) range. However, their development cost and design cycle time are high. Consequently, ASICs are best suited for scenarios in which the AI algorithm and resource constraints are fixed. They are also an appropriate choice when low power consumption and minimal silicon area are primary design objectives [9].

- **Field-Programmable Gate Arrays (FPGAs):** FPGA offer a flexible alternative, allowing designers to implement and reconfigure specialized hardware accelerators after deployment [9]. FPGAs consist of Configurable Logic Blocks (CLBs) interconnected through programmable routing resources, enabling flexible hardware implementation of a wide range of applications. CLBs typically include memory elements, such as local RAM blocks and flip-flops, for storing intermediate data. Through reprogramming, FPGAs can implement complex arithmetic and logic functions, resulting in significantly shorter development cycles compared to ASICs. In addition, FPGAs offer substantial computational and storage resources, making them well-suited for computationally intensive applications, such as deep learning algorithms, that exploit high levels of parallelism [27].

A brief comparison of the key hardware acceleration platforms for DNNs is depicted in Table 1.1. GPUs, FPGAs, and ASICs each present distinct trade-offs as hardware acceleration platforms. GPUs provide high computational throughput through massive parallelism, while FPGAs offer a reconfigurable balance between flexibility and energy efficiency. In contrast, ASICs achieve optimal efficiency for fixed work-

Table 1.1: A Comparative Overview of Hardware Acceleration Platforms.

| **Feature** | **GPU** | **FPGA** | **ASIC** |
|---|---|---|---|
| Architecture | Parallel cores | Reconfigurable logic | Fixed-function |
| Flexibility | Low | High | Very low |
| Programmability | Software | Hardware | None(Post-Fabrication) |
| Energy efficiency | Moderate | High | Very high |
| Power consumption | High | Medium | Low |
| Performance | High throughput | Application-specific | Optimized peak |
| Development cost | Low | Medium | High |
| Design cycle | Short | Moderate | Long |
| Parallelism | Massive | High | Fixed |
| Best suited for | Training | Custom acceleration | Fixed workloads |

loads but with limited flexibility. These inherent trade-offs highlight the importance of application-specific hardware acceleration strategies that carefully balance performance, power consumption, and design complexity.

## 1.4 Multiply-Accumulate(MAC) unit

DNNs comprise of large number of layers, each with multiple artificial neurons. An artificial neuron mimics the functioning of a biological neuron [5]. It consists of a MAC (Multiply–Accumulate) unit and an AF (Activation Function) block. A few popular activation functions include RELU, Softmax, Sigmoid, Tanh, Leaky ReLU and Exponential LU [2].

A typical artificial neuron is illustrated in Fig. 1.5. Each neuron receives a set of inputs $X_0, X_1, \ldots, X_n$, where each input is associated with a corresponding weight $W_0, W_1, \ldots, W_n$. The core computational element within an artificial neuron is the MAC unit. The MAC unit performs the fundamental arithmetic operations that enable neural computation. Within this unit, each input signal is multiplied by its corresponding synaptic weight. This multiplication process is carried out iteratively across all input–weight pairs associated with the neuron.

After each multiplication, the products are successively accumulated. This iterative operation of multiplication followed by addition produces a single scalar value that represents the weighted combination of all inputs. Once the accumulation is complete,

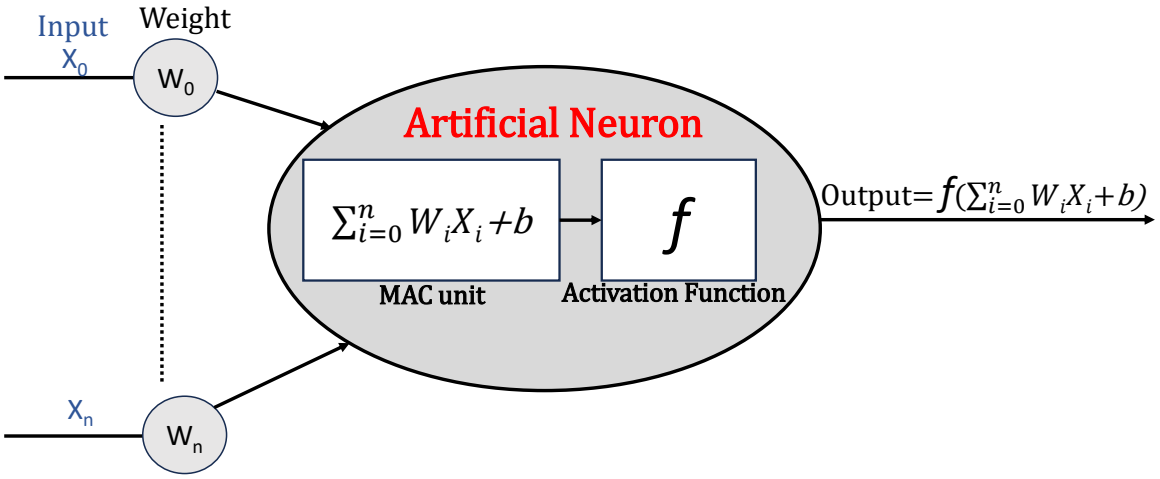



Figure 1.5: A Typical Artificial Neuron comprising MAC unit and Activation Function Block.

the output of the MAC unit is passed to an activation function block. The activation function introduces nonlinearity into the model and regulates the neuron's response, allowing the network to learn complex and non-linear relationships. A bias term $b$ is then added to the accumulated result to shift the activation threshold and improve the model's flexibility during training. The final output of the artificial neuron is therefore obtained by applying the activation function to the MAC output with the added bias, yielding the neuron's response to the given input vector,as illustrated in Fig. 1.5.

A typical MAC unit consists of a multiplier and an adder circuit. These two fundamental arithmetic circuits are resource-intensive, leading to higher power consumption. In the context of DNNs, convolution operations account for more than 90% of overall computation [28]. In other words, MAC operations dominate the computational workload of modern neural network architectures. As network depth and model complexity increase, the demand for MAC operations grows rapidly. For instance, relatively compact architectures such as LeNet require approximately 0.34 million MAC operations, whereas more sophisticated models, including AlexNet, ResNet-50, and VGG-16, require approximately 724 million, 3.9 billion, and 15.5 billion MAC operations, respectively, per individual forward inference pass [2]. This exponential increase in computational demand makes direct hardware implementation of state-of-the-art

DNNs infeasible on resource-constrained platforms without dedicated optimization strategies. Consequently, the development of efficient computation techniques is essential to alleviate the computational, energy, and area limitations associated with conventional DNN hardware implementations [29].

## 1.5 Optimisation of DNNs

Since MAC operations are highly power-hungry, there is a growing demand to optimise the MAC unit. Optimal implementation of the MAC unit can ensure efficient implementation of DNNs as a whole. Multiple techniques have been explored in the state-of-the-art works for the efficient implementation of MAC units and DNNs. These techniques can be broadly categorized into:

1. **Reduction in the Precision of the Data**: Precision can be reduced by lowering bit width, thereby reducing storage and computational costs, and is mainly achieved through uniform and non-uniform quantization. Quantization is the process of mapping continuous or high-precision data to a finite and smaller set of quantization levels. The primary objective of quantization is to minimize the error between the original data and its reconstructed representation derived from these quantization levels. Therefore, reduced-precision representation corresponds to decreasing the number of quantization levels and, hence, the bit width [9]. To guarantee no precision loss, weights and input activations represented with $N$-bit fixed-point precision require an $N$-bit $\times$ $N$-bit multiplication, producing a $2N$-bit product. This product must then be accumulated using $(2N + M)$-bit precision. After accumulation, the precision of the final output activation is typically reduced to $N$-bits, as illustrated in Fig. 1.6.

   Reduction of precision also includes shifting from floating-point to fixed-point arithmetic representation [2]. These two arithmetic representations are briefly discussed below:

   - **Fixed-Point(FxP) Arithmetic:** In the study conducted on fixed-point

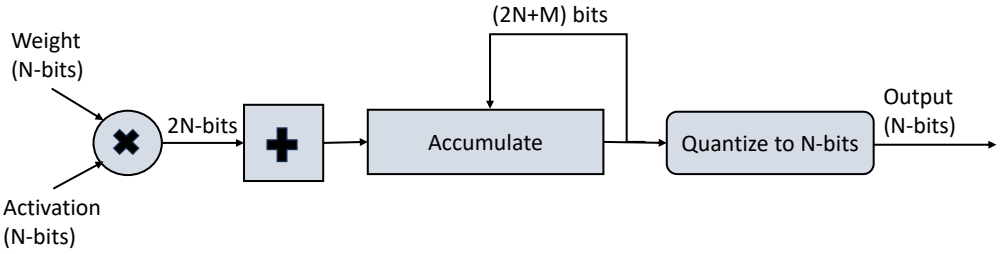



Figure 1.6: Reduction of the precision in MAC [2].

computation, it has been observed that increasing the number of integer bits does not necessarily improve the accuracy of inference. Instead, it is preferable to allocate more bits to the fractional part in the fixed-point notation [30]. Conversely, computation with the number of significant bits in the fractional part has a negligible impact on overall accuracy, suggesting the potential for approximate computation with fewer significant bits. These findings imply that fractional bits can be approximated to enhance the physical performance of fixed-point computation. The signed fixed-point $\langle n, q \rangle$ representation with a binary point implication used for implementation is depicted in Figure 1.7. The notation consists of a total of *n-bits*, in which *q-bits* are allocated to the fractional part.

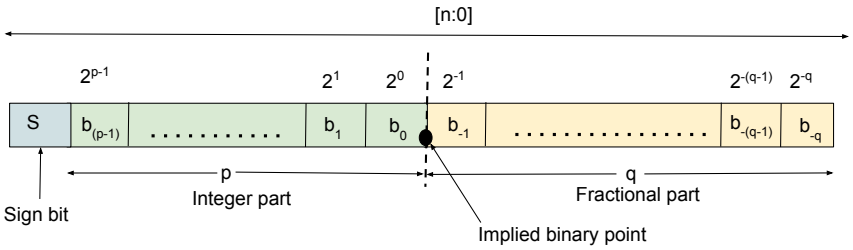



Figure 1.7: Visual representation of total and fractional bits in a signed fixed-point $\langle n, q \rangle$ format representing 'fixed $\langle n, q \rangle$' used for design implementation.

- **Floating Point (FP) Arithmetic:** A floating-point (FP) number $x$ is represented as

$$x = (-1)^s \times (1.M)_2 \times 2^{E-\text{bias}}, \tag{1.1}$$

where $s$ denotes the sign bit, $E$ is the biased exponent, and $M$ is the trailing

significand field. The trailing significand field $M$, together with the implicit leading bit for normalized numbers, forms the significand [31], as illustrated in Fig. 1.8. Floating-point representations, such as the IEEE 754 standard, provide a wide dynamic range and high numerical precision [32], but incur significant hardware complexity and energy consumption compared to Fixed point arithmetic.

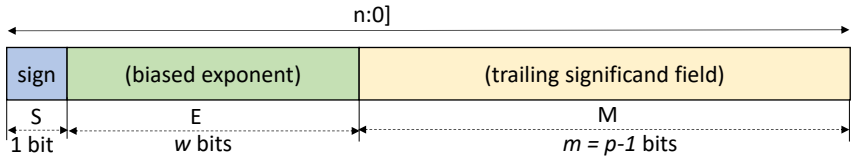



Figure 1.8: Floating-point representation with sign bit $S$, biased exponent $E$, and trailing significand field $M$ [31].

The adoption of reduced precision offers several advantages, including lower storage requirements and reduced computational complexity, which are particularly beneficial for efficient hardware implementations [2].

2. **Reduced Number of Operations and Model Size:** In addition to reducing the precision of individual operations and operands, such as weights and activations, extensive research efforts have focused on techniques aimed at reducing the overall number of operations and the model size of deep neural networks. These approaches seek to improve computational efficiency and reduce memory requirements without significantly compromising model accuracy. Broadly, such techniques can be categorized into methods that exploit activation statistics, network pruning, efficient network architecture design and knowledge distillation. Collectively, these strategies play a crucial role in enabling the deployment of deep neural networks on resource-constrained hardware platforms [2, 9].

From the DNN processing perspective, optimization can target throughput and energy efficiency without affecting model accuracy [2]. Based on these requirements, the optimization techniques can be broadly classified into:

1. **Parallelization of Operations:** Single Instruction, Multiple Data (SIMD) is a form of data-level parallelism in which the same instruction is executed simultaneously on multiple data elements. Instead of processing one data value at a time, SIMD architectures operate on vectors of data in parallel, significantly improving computational throughput. SIMD parallelization is widely used in DNN acceleration, where operations such as MAC are repeatedly applied to large sets of weights and activations. By processing multiple data values in parallel, SIMD reduces execution time and improves energy efficiency compared to scalar processing [2, 33].

2. **Energy-Efficient Dataflow:** DNN accelerators improve energy efficiency by employing multi-level memory hierarchies and specialized dataflows that minimize costly data movement. Local storage, inter-PE communication, and register files enable low-energy data access compared to DRAM(Dynamic Random-Access Memory) [34]. For DNNs, dataflows exploit three forms of input data reuse: convolutional, feature map, and filter reuse. Convolutional reuse applies the same input activations and filter weights within a channel across different weighted sums. Feature map reuse applies multiple filters to the same feature map, reusing input activations across filters. Filter reuse occurs when processing batches of input feature maps, allowing the same filter weights to be reused across multiple inputs [2, 8].

Table 1.2: Performance Metrics for DNN Evaluation [2].

| **Evaluation Category** | **Performance Metrics** |
|---|---|
| **DNN Hardware Evaluation** | Power Consumption<br>Latency and Throughput<br>Chip Area Efficiency<br>Implementation Cost |
| **DNN Software Evaluation** | Accuracy<br>Network Architecture:<br>– Number of Layers<br>– Filter Sizes<br>Number of MAC Units<br>Number of Weights |

All techniques employed for the optimization of MAC units are evaluated using a comprehensive set of metrics across both hardware and software domains, which have been illustrated in Table 1.2. These metrics are used to systematically assess and compare the strengths and limitations of different design approaches and proposed optimization techniques. For accuracy and robustness evaluation, it is important to report results on widely accepted benchmark datasets. Power and energy evaluations are crucial for Edge devices and should account for all system components, including the chip and external memory accesses. High throughput is essential for achieving efficient, real-time performance in interactive applications such as navigation and robotics. Hardware cost is largely determined by the amount of on-chip storage and the number of cores. The number of MAC operations indicates the computational workload and potential throughput of the DNN [2]. Thus, the viability of DNNs is determined by accuracy, latency and throughput, as well as energy efficiency, power consumption, and cost.

## 1.6 Approximation: An Optimisation Technique for DNNs

Among the above-mentioned optimization techniques, approximate computing has emerged as a new design paradigm for energy-efficient circuits [35]. Approximate Computing (AC) goes beyond conventional and emerging design approaches by exploiting the inherent error resilience of Neural Networks (NNs), which can maintain acceptable accuracy despite approximate computations [36]. By leveraging this property, AC applies approximations to the hardware execution of NN workloads to improve performance and energy efficiency [37]. Owing to the substantial potential for energy savings and the intrinsic tolerance of NNs to computational errors [38], research on approximate NN implementations has expanded rapidly in recent years. The growing demand for efficient DNN inference, together with the demonstrated success of approximate computing (AC) techniques, has attracted substantial research interest.

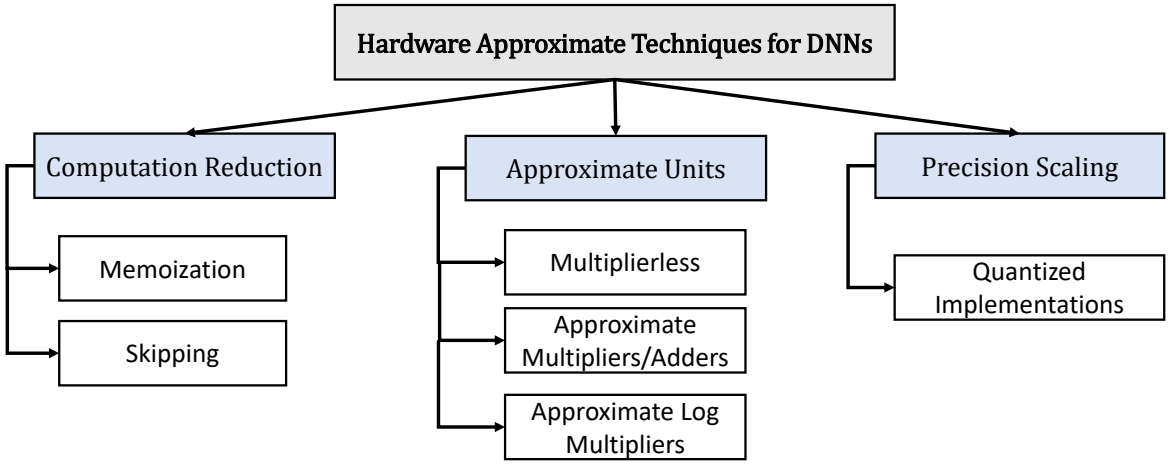



Figure 1.9: Overview of Hardware Approximation Techniques for DNNs [10].

The hardware-based DNN approximation techniques can be broadly classified into three categories: computation reduction, approximate arithmetic units, and precision scaling [10], as illustrated in Fig. 1.9. Although these categories are largely orthogonal, state-of-the-art approaches predominantly rely on approximations from a single category or combine precision scaling with techniques from another category. These techniuqes are discussed in detail in the subsections below.

### 1.6.1 Computation Reduction

The computation reduction approximation category focuses on systematically bypassing, at the hardware level, the execution of selected computations, such as multiplications and convolution operations. This approach effectively reduces the executed workload. Computation reduction is further classified into the memoization and skipping approximation families [10].

- Skipping approximations reduce the executed workload by using lightweight computations to determine whether more complex operations can be omitted at runtime. Their effectiveness depends on the frequency of skipped operations and the complexity of both the prediction logic and the omitted computations [39].

- Memoization avoids redundant computations (e.g., multiplications or convolutions) by reusing the results of previously executed similar operations. Its ef-

fectiveness depends on the degree of input similarity and the complexity of the eliminated computations [40, 41].

### 1.6.2 Precision Scaling

Precision scaling is a fundamental approximation technique that enables high compute density in DNN hardware accelerators across cloud and edge platforms. Rather than executing all operations using 32-bit or 16-bit floating-point arithmetic, as in conventional CPUs and GPUs, precision scaling employs lower-bitwidth integer representations through quantization. The details regarding quantization have been presented in Section 1.5. This approach improves computational efficiency while linearly reducing model size, thereby lowering storage requirements and memory traffic. In integer-arithmetic-only inference, weights and activations are typically quantized to low-bitwidth integers (e.g., 8-bit), whereas biases are represented using higher precision (e.g., 32-bit or lower) to preserve accuracy.

### 1.6.3 Approximate Units

The energy consumption as well as latency of DNN accelerators can be effectively improved by employing approximate circuits that replace accurate MAC units [10]. Approximate Units can be further divided into the following approximation families:

- **Approximate Multipliers/Adders:** Due to the extensive number of MAC operations performed during the inference phase of deep neural networks, a significant body of research has focused on circuit-level approximation of the MAC unit. Given that each MAC operation incurs a largely constant energy overhead, reductions at the per-operation level accumulate to substantial energy savings at the system level during inference. Consequently, approximate MAC designs constitute an effective approach for improving the energy efficiency of DNN hardware accelerators [42, 43, 44]. Architectural and algorithmic approximations with these fundamental circuits have attracted significant research interest over the last years [45, 46, 47].

Numerous works propose approximate multipliers and adders based on techniques such as evolutionary optimization, operand segmentation, truncation, and arithmetic simplification [48, 49, 50]. To limit the impact on inference accuracy, these approaches commonly employ network retraining, error-aware cost metrics, or weight–distribution–driven optimization [51]. Experimental results show that, when carefully controlled, such approximations can deliver significant power and area reductions with minimal or no loss in inference accuracy [52].

Beyond static approximation schemes, several studies investigate reconfigurable and adaptive techniques that exploit layer-wise or input-dependent error resilience [53]. Representative approaches include heterogeneous accelerators with layer-specific approximate units, dynamic precision selection mechanisms, curable approximations with runtime error compensation, and control-variate methods for on-the-fly error correction. Collectively, these approaches indicate that approximate multipliers and adders can be incorporated into either NN-specific or generic accelerator architectures, exposing trade-offs among retraining overhead, hardware complexity, and adaptability [10]. Consequently, approximate arithmetic units represent a well-established and versatile design space for enhancing the energy efficiency of DNN hardware accelerators [52, 54].

- **Multiplierless:** The Multiplierless subcategory aims to maximize efficiency by eliminating costly multiplication circuits and replacing them with hardware that implements simpler operations. By exploiting the invariance of max-pooling to input scaling, convolution weights can be constrained to the range $[-1, 1]$, enabling the use of trigonometric functions implemented via low-cost CORDIC algorithms [55, 56].

- **Approximate Log-Multipliers:** Logarithmic multipliers approximate multiplication by converting it into addition using approximate logarithmic and antilogarithmic operations. Mitchell introduced a logarithmic multiplier based on linear log–antilog approximations within power-of-two intervals [57]. Reduction of the inherent negative bias of this method through a constant correction term

and truncation has also been studied [58]. Others approaches include multi-stage logarithmic designs and fractional power-of-two quantization schemes, which replace conventional MAC units with logarithmic processing elements to improve efficiency with acceptable accuracy [59, 60].

Given the error tolerance requirements of target applications and the stringent resource constraints of edge devices, suitable approximation techniques can be systematically selected to improve the resource efficiency of AI implementations. Such techniques enable favorable trade-offs among accuracy, energy consumption, latency, and hardware complexity, making them particularly effective for resource-constrained and error-tolerant edge AI workloads.

## 1.7 SIMD Architectures for Parallel Operations

Beyond optimization, throughput is a defining factor in efficient DNN accelerator design. In practice, achieving high throughput within strict power and silicon area limits relies heavily on the use of low-to-medium-precision arithmetic. However, different DNN models react differently to reduced precision. Some experience a significant drop in accuracy at lower bit-widths, while others waste power at higher precisions due to redundant computations and underutilized hardware. This variation highlights the need for scalable architectures that can adapt to different precision needs dynamically. Ultimately, maximizing efficiency requires a granular approach, since different models—and even individual layers within the same model—often run best at entirely different bit-widths [61]. To address these challenges, multi-precision and single-instruction, multiple-data-based (SIMD) processing elements (PEs) have emerged as promising architectural solutions. These designs support flexible precision configurations, making them well-suited for heterogeneous deep learning workloads. Furthermore, they require carefully crafted arithmetic schemes to ensure both computational accuracy and hardware efficiency [62, 63, 64].

Architectures that scale precision generally fall into two categories: 1D (weight-only) and 2D (both weight and activation) scalable designs [65]. To handle flexible

bit-widths, leading designs usually rely on bit-serial computation [66, 67, 68, 69] and decomposed multipliers [70, 71, 72] built into bit-fused MAC units. These units can alter their data paths dynamically based on the operand bit-widths [70]. However, this flexibility often comes at the cost of poor hardware utilization and only minor improvements in latency.

To boost PE efficiency, optimization strategies such as loop-nest restructuring and shift-add reduction have been deployed [71]; however, processing higher-precision operands still triggers longer accumulation cycles and distinct spikes in power consumption. Another common strategy involves applying sub-word parallelism to weight tensors for multi-precision MACs [73, 74]. While this approach typically uses data gating to curb dynamic power, it simultaneously degrades overall MAC utilization. Raut et al. [64] addressed this utilization bottleneck by enhancing parallelism, yet their design is constrained to just four MAC operations for 8-bit precision and a single operation for 16-bit precision, ultimately capping maximum throughput.

Looking at floating-point alternatives, multi-precision MAC designs have been proposed, but they either suffer from poor hardware utilization [75] or induce accuracy loss due to mantissa truncation [76]. Even variable-precision multipliers paired with dynamic voltage-frequency scaling [77] fall short, as they leave sub-multipliers idle during low-precision operations, which limits efficiency. Consequently, to effectively support a broad spectrum of bit-widths (from 4-bit to 32-bit), scalable PEs must strike a more optimal trade-off between throughput, power overhead, and layer-by-layer accuracy.

## 1.8 Summary

The hardware implementation of DNNs presents significant challenges due to their high computational complexity, substantial power consumption, and stringent hardware resource constraints, particularly in edge AI and embedded intelligence systems [49]. These challenges primarily arise from the extensive number of multiply–accumulate (MAC) operations that dominate the computational workload of mod-

ern neural network architectures. As network depth and model complexity increase, the demand for MAC operations grows rapidly. This exponential increase in computational demand makes direct hardware implementation of state-of-the-art DNNs infeasible on resource-constrained platforms without dedicated optimization strategies. Consequently, the development of efficient computation techniques is essential to alleviate the computational, energy, and area limitations associated with conventional DNN hardware implementations [29].

In this thesis, efficient computational techniques for DNN acceleration, focusing on both hardware resource optimisation and throughput enhancement to enable efficient deployment on resource-constrained platforms, have been investigated. The first part of this study explores approximation techniques within the MAC unit to achieve a favourable trade-off between hardware resource consumption, power efficiency, and inference accuracy. Approximate computing in MAC units typically involves architectural or algorithmic modifications to the core arithmetic components, particularly the multiplier and adder blocks, which are known to dominate both area and energy consumption in DNN accelerators. By selectively relaxing computational accuracy in error-tolerant portions of the network, such approximate arithmetic units can significantly reduce power and hardware overhead while maintaining acceptable model performance.

In addition to approximate computing, the inherent performance bottlenecks caused by the sequential execution of MAC operations have been addressed. Given the massive number of MAC operations involved in DNN inference and training, sequential processing leads to high latency and excessive energy consumption, thereby limiting real-time deployment in edge and embedded systems. However, DNN computations exhibit a high degree of intrinsic parallelism, as operations across neurons, channels, and layers are largely independent. Exploiting this parallelism enables substantial reductions in execution latency and significant improvements in overall throughput [78]. Parallel processing architectures allow multiple MAC operations to be executed concurrently, thereby meeting stringent performance requirements within constrained power budgets. This thesis explores the use of Single Instruction Multi-

ple Data (SIMD) architectures to exploit computational parallelism in DNNs. SIMD-based designs enable simultaneous execution of multiple MAC operations using a single instruction stream, improving hardware utilisation and energy efficiency. Importantly, SIMD architectures can be extended to support variable-precision computation, which is essential for accommodating the diverse precision requirements of different DNN layers and models within a single neural network. Such flexibility allows the accelerator to dynamically balance accuracy, performance, and energy efficiency, making it well-suited for modern adaptive and heterogeneous DNN workloads. The key objectives addressed in this dissertation include:

- Resource-Efficient implementation of MAC unit by employing:
  - Approximate Adder architecture with reduced gate count
  - Operand-rounding-based Approximate Multiplier architecture
  - Accuracy-configurable Segmentation-based Approximate Multiplier architecture
- Throughput Enhancement of DNNs by employing Adaptive-Precision SIMD Processing Elements for parallel computations.
- Approximation within SIMD PE for resource optimization: Throughput Enhancement alongside Resource-Efficient implementation of DNNs by incorporating approximate multipliers within SIMD PE architectures.

## 1.9 Thesis Contributions

The major contributions to this thesis are summarised below -

1. **Approximate Hybrid Adder Design**[1]: Approximate adders enhance energy efficiency in error-tolerant workloads using topologies like Lower Part OR

[1] **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 (CSSP), Springer*, Volume 42, Pages 7596–7614, August 2023.

Adders [79], inexact hybrid architectures [80], and carry-skip FPGA designs [39]. However, standard architectures often suffer from error propagation during $n$-bit accumulation [80], necessitating flexible, carry-maskable alternatives [81]. To address these limitations, this research proposes a solution for efficiently implementing DNNs in edge-AI applications by introducing architectural approximations in the adder design within the MAC unit. Specifically, an optimal Hybrid Adder (HADD) block for accumulation in fixed-point MAC operations is developed to overcome these persistent area and power limitations. The proposed HADD design achieves a significant reduction in area and power consumption, with a tolerable loss in accuracy and a corresponding reduction in computational latency. The test results show accuracies of 96.97% and 96.64% for the MNIST and A-Z Handwritten Alphabet datasets, respectively, using the LeNet DNN model. Compared to conventional adder implementation, the proposed HADD design reduces area utilization by 44% and power consumption by 51%, with a reduction in delay of 19% for 8-bit precision at 180nm. For the same bit-precision, the proposed design reduces area by 31%, power consumption by 34%, and delay by 8.1% at 45nm. The proposed design further investigates edge detection applications, and the results for different standard images were promising. Overall, the proposed accumulator arithmetic block is a viable solution for error-tolerant applications, including image classification, object recognition, and other image-processing applications.

2. **Operand-Rounding based Approximate Multiplier Design Error-resilient AI Applications**[2]: This research presents a novel approximate multiplier design for error-tolerant applications, based on the pre-processing of input operands to achieve an energy-efficient implementation. While existing methods optimize multipliers via architectural approximations [82, 83], logarithmic reduction [57, 84], or input pre-processing like leading-one truncation [85, 86]

[2]**Vasundhara Trivedi**, Santosh Kumar Vishvakarma, "Design of Operand-rounding based Approximate Multiplier for Error-resilient Applications", *29th International Symposium on VLSI Design and Test, 2025*, Aug 6-9, 2025 Chandigarh, India (Accepted and In Press)

and power-of-two rounding [87] to reduce complexity, they often suffer from significant accuracy degradation due to aggressive truncation or rigid rounding schemes. The proposed work differs by selectively rounding 8-/7-/6-bit operands to a fixed 5-bit format (leaving lower bits unchanged), executing all higher-bit multiplications using a unified 5-bit multiplier followed by strategic shifting to achieve substantial resource savings with minimal accuracy loss. The efficacy of this design is evaluated using both error and performance metrics. Furthermore, when integrated into a MAC unit, the proposed multiplier achieves a 24.4% reduction in LUT utilization and a 37.53% reduction in critical path delay compared to the conventional implementation, exhibiting the lowest delay among the compared state-of-the-art designs. Finally, the design is validated via an edge-detection application to demonstrate its suitability for practical image-processing tasks with minor accuracy loss.

3. **Segmentation-based Approximate Multiplier Design Error-resilient AI Applications**[3]: This research focuses on the design of a segmentation-based approximate multiplier, referred to as ACSAM (Accuracy-configurable Segmentation-based Approximate Multiplier), which leverages multiple combinations of exact and newly proposed approximate 8-bit multipliers to achieve computational efficiency. While conventional dynamic segmentation schemes offer high precision by tracking the leading-one bit [86, 87, 88], they often incur significant hardware overhead, leading researchers to explore structurally simpler static segmented multipliers (SSMs) [85, 89, 90] to reduce power. The proposed ACSAM framework differs by combining configurable exact and novel approximate 8-bit units within a structured static segmentation scheme, utilizing unique shifting and rounding strategies to enable fine-grained energy–accuracy tradeoffs and flexible, application-aware deployment. The ACSAM architecture is carefully engineered to strike a balance between accuracy and hardware cost,

[3]**Vasundhara Trivedi**, Santosh Kumar Vishvakarma, "ACSAM: Accuracy-configurable Segmentation-based Approximate Multiplier for Error-resilient Edge-AI Applications", *IEEE Embedded Systems Letters*, March 2026.

making it particularly suitable for error-tolerant DSP and AI applications. By selectively integrating approximate computation in less significant segments while preserving accuracy in critical regions, the design significantly reduces hardware complexity without compromising overall performance. The proposed multiplier ACSAM achieved upto 18.9% improvement in LUT utilisation on FPGA and upto 2.85× reduction in power and upto 12.27% reduction in area for ASIC implementation on 65nm technology node compared to the state-of-the-art works. ACSAM is also validated on image blurring application to verify its utility in real-life scenarios. Furthermore, evaluation on both FPGA and ASIC platforms demonstrates the architecture's adaptability to a range of requirements.

4. **Adaptive-Precision SIMD PE Architecture for Enhanced Throughput**[4]: In this study, an adaptive-precision SIMD Processing Element (PE) architecture for signed integer and fixed-point arithmetic is presented to maximize resource utilization and MAC parallelism. Unlike multi-bit designs limited by symmetric bit-split systolic arrays [62, 91] or rigid 16-bit weight adder-trees [66], the proposed architecture enables dynamic precision scaling for both symmetric and asymmetric configurations within a single hardware instance. By reusing hardware resources during partial product accumulation, the PE concurrently processes either 16 4-bit, four 8-bit, or a single 16-bit operation. Furthermore, it supports asymmetric modes—such as parallel 16×4-bit multiplications—enhancing flexibility and throughput for mixed-precision DNN workloads. Accuracy evaluations across different DNN models and datasets show very small losses at reduced precision—less than 1% for LeNet on MNIST, 2.9% for AlexNet on CIFAR-10, 2.2% for VGG16 on CIFAR-10, and 3.5% for VGG16 on ImageNet-1000 compared to float32. Hardware synthesis yields significant improvements, including 46.2% fewer LUTs and 2.45 × less power on the FPGA compared to existing designs. The proposed architecture delivers 2× higher

[4]**Vasundhara Trivedi**, Harman Singh Bagga, Gopal Raut, Santosh Kumar Vishvakarma, "Adaptive-Precision SIMD Architecture for High-Throughput and Resource-Efficient DNN Acceleration", *Integration, Elsevier*, Volume 108, p.102666, January 2026.

throughput, upto 4.8× energy efficiency with 28.57% less area at 65nm, compared to existing works, making it ideal for applications with variable precision and limited resources. By combining multi-precision adaptability, asymmetric operation support, and efficient hardware reuse, the proposed design sets a new benchmark for precision-aware DNN acceleration.

5. **CORDIC-based Approximate SIMD PE Architecture**[5]: This study introduces C-SIMD, a CORDIC-driven, configurable SIMD Processing Element (PE) architecture designed to extend previous adaptive SIMD frameworks by substituting conventional multipliers with approximate CORDIC-based alternatives. While the CORDIC algorithm is widely explored for low-resource multiplication [55], prior recursive-CORDIC architectures incur an $n$-cycle latency penalty for $n$-bit precision, thereby constraining throughput in SIMD-based DNN accelerators [64, 92]. To address this limitation, C-SIMD implements a pipelined, throughput-optimized CORDIC methodology that integrates adaptive-precision capabilities across both symmetric and asymmetric arithmetic modes. Supporting dynamic operand precisions (4/8/16/32-bit) for signed integer and fixed-point arithmetic, the architecture leverages pipelined 8-bit CORDIC-based approximate multipliers to compute partial products, scaling efficiently to higher precisions while achieving notable area and power savings. A configurable pipeline offers tunable trade-offs between accuracy and complexity, while strategic reuse of the adder in the accumulation path maximizes resource utilization. Unlike prior designs, C-SIMD fully exploits available resources and supports configurations such as 16 parallel 8×8-bit, 4 parallel 16×16-bit, single 32×32-bit, and asymmetric 32×8-bit MACs. Hardware evaluation demonstrates up to 14.29% area savings and upto 16.17× throughput improvement. The architecture integrates custom C-SIMD processing elements within a 2D systolic array to enable highly parallel execution and efficient vector computation for

[5]**Vasundhara Trivedi**, Gopal Raut, Baker Mohammad, Santosh Kumar Vishvakarma, Akash Kumar, "C-SIMD: CORDIC-Driven SIMD Processing Element for Resource-Efficient Multi-Precision Deep Learning Inference", *IEEE Access*, Vol. 14 : 19015 - 19029, January 2026.

deep neural network models.The proposed `C-SIMD_Low (4/8/16)` achieves 7.04 GOP/s, while `C-SIMD_High (8/16/32)` attains 4.16 GOP/s, delivering a 4× performance-efficiency gain over prior MAC architectures. Inference tests indicate minimal accuracy loss—below 1% on MNIST-LeNet, under 2.9% on CIFAR-10-AlexNet, and less than 2.2% on CIFAR-10-VGG16 compared to float32 baselines—demonstrating its potential for high-throughput, energy-efficient Edge-AI systems.

## 1.10 Thesis Organization

The thesis is organized as follows:

- **Chapter 1. Introduction:** This chapter presents the fundamentals of hardware implementations for DNN accelerators. It introduces the core building blocks of DNNs and their associated computations. It also explains the need for DNN optimisation and reviews state-of-the-art techniques for optimising DNNs at the algorithmic, architectural, and hardware levels.

- **Chapter 2. Approximate Full Adder Design for Efficient Implementation of MAC unit:** Approximation is one of the promising techniques for optimizing DNNs. Approximation can be performed at various abstraction levels, including algorithmic approximation and architectural approximation. This chapter discusses the architectural approximation of the full adder design and its utilisation in designing a Hybrid Adder (HADD) inside MAC unit, suitable for error-resilient AI applications. The impact of the proposed approximation on inference accuracy is evaluated across different datasets. In addition, the effectiveness of the proposed design is demonstrated using an edge detection application. The design is synthesized using both 45 nm and 180 nm technology nodes to validate its scalability and implementation feasibility.

- **Chapter 3. Operand-rounding-based Approximate Multiplier Design:** In the previous chapter, an approximation of the adder circuit within the MAC

unit was introduced to achieve resource optimization. The MAC unit inside DNNs comprises both adder and multiplier circuits. This chapter outlines a proposed operand-rounding-based approximate multiplier that offers an optimal trade-off between accuracy and resource requirements. The design focuses on reducing operand bitwidth through a novel rounding and shifting strategy with minor accuracy degradation, making it suitable for error-resilient AI Applications. The design has been evaluated on ASIC and FPGA platforms for performance analysis. It has also been tested on an edge detection application to evaluate its utility in AI applications.

- **Chapter 4. Segmentation-based Approximate Multiplier Architecture:** This chapter introduces an approximate multiplier architecture based on operand segmentation. The proposed 16-bit approximate multiplier is constructed by hierarchically combining four 8-bit approximate multipliers presented in the previous chapter, with minor architectural modifications to enhance flexibility and performance. Building upon this structure, four distinct design variants are proposed, each representing a different combination of accurate and approximate multiplier blocks. These variants enable a tunable trade-off between accuracy and hardware efficiency, allowing the multiplier to be customized according to the precision requirements of target AI and signal-processing applications. By selectively employing accurate or approximate multipliers for specific operand segments, the design can adapt to varying error tolerance levels while maintaining computational efficiency.The proposed ACSAM architecture demonstrates optimal performance across both FPGA and ASIC platforms, combining reduced hardware cost with enhanced error characteristics. Its successful validation through an image processing application further confirms its practical viability. Overall, the proposed multiplier architecture emerges as a promising solution for resource-constrained Edge AI/DSP applications, offering an efficient and scalable approach to approximate arithmetic design.

- **Chapter 5. Adaptive-Precision SIMD PE Architecture for Enhanced**

**Throughput and Resource-Efficient DNN Acceleration:** DNNs require a large number of computational operations, and sequential execution of such operations leads to degradation in throughput and increased delay. By introducing computational parallelism in the processing elements, the associated throughput and delay can be significantly improved. While previous chapters focus on resource optimisation, this chapter explores the concept of operational parallelism in Processing Elements (PEs) by utilising SIMD concepts to enhance the throughput of DNNs. A single PE architecture supports multi-precision mode-based operations and can execute 4, 8, 16 (*SIMD-Low*) as well as 8, 16, 32-bit (*SIMD-High*) operations by employing a systolic array of 16 multipliers followed by an adder tree. The adder is effectively reused to minimise hardware, and the output is rounded after accumulation and sent to AF blocks for further processing. The SIMD PE utilises 100% of the hardware resources in all supported precision modes, enhancing throughout at lower precisions. The design has been validated on both ASIC and FPGA platforms. The architecture is also validated across various datasets and DNN models for inference accuracy evaluations.

- **Chapter 6. CORDIC-Driven Approximate SIMD PE Architecture for Resource-Efficient and Throughput-Enhanced Multi-Precision Computations within DNN models:** In the previous chapter, the adaptive SIMD processing engine (PE) employing conventional multipliers was presented to improve throughput. In this chapter, this approach is extended by incorporating approximate multipliers. Specifically, CORDIC-based multipliers are implemented in place of conventional multipliers to reduce resource utilization during parallel processing. CORDIC stands for Coordinate Rotation Digital Computer, and it is a technique that utilises vector rotation to compute various arithmetical and transcendental functions. CORDIC-based multipliers have been implemented in state-of-the-art systems, demonstrating promising results by achieving optimal trade-offs between accuracy and resource requirements at various bit-precisions. Moreover, with pipelined architectures, the latency also

improves. The proposed architecture integrates a systolic array of CORDIC multipliers, followed by an adder tree that utilises different bitwidth adders at different levels. The design has been validated through inference accuracy evaluations across various datasets and models, yielding promising results that demonstrate its utility across AI applications. The proposed C-SIMD PE design is validated at both ASIC and FPGA platforms for detailed performance analysis. The C-SIMD design delivers promising results compared to state-of-the-art works, achieving enhanced throughput, resource optimisation, and support for asymmetric precision for heterogeneous DNN workloads.

- **Chapter 7. Conclusion and Future Work:** This chapter concludes the work by summarizing the key contributions and findings. It also outlines prospective directions for future research that may further enhance and extend the proposed techniques.

# Chapter 2

# Approximate Full Adder Design for Efficient Implementation of MAC unit

Multiply–Accumulate (MAC) units account for more than 90% of the total power consumption in deep neural networks (DNNs), making their optimization critical for improving the overall energy efficiency of DNN accelerators [28]. Since neural network workloads are dominated by large volumes of multiply-and-accumulate operations, even marginal improvements in MAC unit efficiency can lead to substantial reductions in system-level power consumption, silicon area, and computational latency.

The MAC unit is primarily composed of two fundamental arithmetic components: the multiplier and the adder/accumulator. These circuits largely determine the power dissipation, critical path delay, and hardware complexity of the MAC unit [93]. Consequently, optimizing either the multiplier or the adder—or both—can significantly enhance MAC unit performance and efficiency. Techniques such as reduced-precision arithmetic, approximate computing, and architecture-aware circuit design have been widely applied to these components to achieve favorable trade-offs between accuracy and hardware cost.

As a result, extensive research efforts have focused on developing resource-efficient implementations of multipliers and adders to reduce power, area, and delay while maintaining acceptable inference accuracy. These optimized arithmetic designs form the foundation of energy-efficient DNN accelerators and are particularly important for edge computing platforms with strict power and area constraints. This chapter

investigates the proposed adder-circuit approximation, a fundamental block within the MAC unit.

## 2.1 Introduction

The Multiply–Accumulate (MAC) unit plays a fundamental role in neural network computation, as it performs the core arithmetic operations required for inference and training. A MAC unit consists of a multiplication operation followed by a sequence of additions and accumulations, enabling efficient computation of weighted sums in neural network layers. The architecture of an n-bit MAC unit is illustrated in Figure 2.1. The final bit-width equals 2n+a where a = $\log_2 W$ where W is the number of iterations. Inside a MAC unit, the trained weights are multiplied with the inputs, either iteratively or in a parallel manner, and accumulated for further processing. The final output is then produced by passing this obtained output via a non-linear activation function [49].

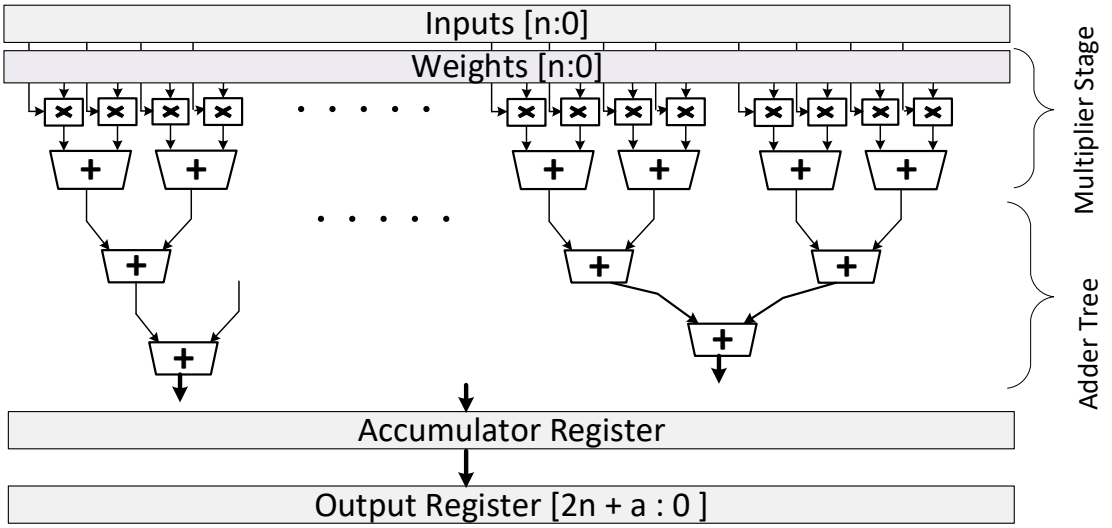



Figure 2.1: Parallel Multipliers and Adder Trees: Unleashing the Power of Multiply-Accumulate (MAC) Operations.

A wide range of optimization techniques have been explored to enhance the hardware efficiency of DNNs in terms of silicon area, power consumption, and computational latency, often at the expense of a controlled reduction in inference accuracy [2].

Among these techniques, bit quantization, weight reduction, and approximate computing have emerged as prominent strategies. Bit quantization at the output of the multiplier and accumulator stages reduces the complexity of subsequent arithmetic operations and data storage requirements; however, it necessitates additional control logic and supporting circuitry, which may introduce area overhead.

## 2.2 Background and Related Works

Approximate computing has gained widespread adoption in neural network accelerators due to the inherent error resilience of many machine learning workloads. Approximation can be applied at multiple abstraction levels—including algorithmic, architectural, and circuit levels—to achieve improved performance and energy efficiency while maintaining acceptable accuracy bounds [93, 94]. Common approximation techniques include reduced-precision arithmetic, bit truncation, network pruning, and the use of approximate multipliers and accumulators. Network pruning aims to reduce model complexity by eliminating redundant parameters and connections, thereby improving computational efficiency and arithmetic performance. However, real-time pruning implementation poses challenges such as accuracy degradation, sensitivity to initial network configurations, and the need for retraining to restore performance [95].

Since the MAC unit dominates both area and power consumption in neural network accelerators and directly affects throughput, significant research attention has been directed towards its optimization through approximation. In particular, the adder and multiplier circuits within the MAC unit are the primary targets for approximation in state-of-the-art designs, owing to their substantial contribution to delay and power dissipation. Consequently, numerous hardware-level approximation techniques have been proposed to optimize MAC units for energy-efficient DNN implementations [96].

Several studies in artificial intelligence and machine learning hardware have emphasized the critical role of approximate adders in improving energy efficiency for compute-intensive and approximation-tolerant applications [97]. Adders are fundamental arithmetic components, and their approximation can yield significant reduc-

tions in power and area with limited impact on overall system accuracy [98].

Among approximate datapath operators, adders have received significant research attention owing to their widespread use in error-tolerant applications [99, 100, 101]. Consequently, numerous approximation schemes have been proposed to reduce the critical path and hardware complexity of accurate adders. One of the earliest methodologies is based on speculative operation. In an $n$-bit speculative adder, each sum bit is predicted using its previous $k$ less significant bits (LSBs), where $k < n$. Such a speculative design makes the adder significantly faster than a conventional accurate design [102]. An alternative method for reducing the critical path delay and power consumption of a conventional adder involves approximating the full-adder units [103]. In such approaches, approximation is typically applied to the least significant bits (LSBs) while preserving accuracy in the more significant bits [102].

A representative example of this design philosophy is the Lower Part OR Adder (LOA), in which the least significant bits are computed using simple OR logic, while the most significant bits are computed accurately. By carefully optimizing the partition between accurate and approximate sections, the LOA architecture effectively minimizes error metrics such as Mean Square Error (MSE), making it suitable for error-tolerant applications [79]. This approach significantly reduces critical path delay and power consumption compared to fully accurate adders, while maintaining acceptable computational accuracy.

In addition, inexact hybrid adder architectures have been proposed to improve the efficiency of shift-and-add operations commonly employed in MAC computational units. Such designs incorporate configurable adder structures that can operate in either accurate or approximate modes, enabling dynamic trade-offs between precision and performance depending on application requirements [80]. However, in an *n-bit* accumulator stage, approximation-induced errors may propagate across successive accumulation cycles, potentially degrading output accuracy and necessitating additional corrective computation, which increases overhead.

To address error propagation in accumulation-heavy workloads, flexible approximate adder architectures have been introduced, most notably those based on carry-

maskable adders [81]. These architectures selectively suppress or enable carry propagation across specific bit positions, allowing fine-grained control over the accuracy–energy trade-off. Such designs have demonstrated effectiveness in error-resilient applications, including image sharpening algorithms, where moderate numerical errors are perceptually tolerable.

Furthermore, FPGA-based implementations of approximate arithmetic adders have been explored to achieve efficient and reconfigurable computation. Designs such as those presented in [104] exploit LUT primitives inherent to FPGA architectures to reduce delay and logic utilization. These designs often incorporate carry-skip schemes [39] to further shorten the critical path and improve energy efficiency. Although these FPGA-based approximate adders exhibit certain limitations in scalability and precision control, they have been shown to outperform application-specific exact designs across image processing tasks and DNN inference workloads.

To overcome the limitations of existing approximate adders in balancing accuracy, area, power, and delay, recent work has focused on accumulator-centric optimization in MAC units. By targeting the accumulator—the primary contributor to error propagation and hardware cost—these approaches are well-suited for DNN inference and image processing. Building on this concept, this chapter proposes an approximate adder block for use as a MAC accumulator, reviews state-of-the-art approximate MAC designs for DNNs, and presents an analysis of adder combinational logic to motivate the proposed architecture. The key contributions include :

- **Hardware Optimization**: Design of a fixed-point approximate adder to improve MAC hardware efficiency in error-resilient DNN applications.
- **Pareto Analysis**: A Pareto study to evaluate the impact of approximation in accuracy at different bit-precisions and assess different combinations of the conventional adder block and approximate adder blocks in multi-bit adders.
- **Software Integration**: Integration of the proposed adder into MAC units with DNN inference accuracy evaluation and validation on an image-processing application.

- **Performance Evaluation**: Evaluation of the efficiency of the proposed adder design through VLSI implementation and assessment of physical performance parameters such as area, power, and critical delay at 45nm and 180nm technology nodes.

## 2.3 Proposed Hybrid Adder

In MAC operations, multiple parallel multiplication and accumulation computations occur. This process involves using 1-bit full adders to build larger blocks internally during multiplication, and the accumulation stage is built from a single-bit adder. The traditional approach to designing a 1-bit full adder involves using five basic logic gates to compute the sum and the carry. Specifically, two `XOR` gates are employed to calculate the sum, and two `AND`, one `XOR`, and one `OR` gates are used in the calculation of the carry. The sum and carry bits of the conventional full adder (conventional FA) and the proposed full adder (proposed FA) are computed using the generalized equations depicted in Table 2.1.

This study introduces a 1-bit approximate adder that utilizes only three logic gates. The truth table of the proposed adder can be compared with the conventional full adder presented in Table 2.1. The incorrect outputs in the sum or carry are highlighted in the table. The truth table analysis indicates that the proposed adder produces only one error in the carry bit and two errors in the sum bit across eight possible input combinations. Maintaining a low error rate in carry-bit computation is particularly important, as errors in the carry signal propagate to higher-order bits and can significantly amplify the overall computational error. This observation highlights the critical role of carry accuracy in approximate adder design, directly motivating a careful analysis of carry generation and propagation mechanisms. Since the carry chain directly influences the correctness of subsequent sum bits, inaccuracies in carry generation or propagation can lead to substantial deviations in the final result, particularly in multi-bit arithmetic operations.

Table 2.1: Truth Table for Conventional and Proposed Full Adder Designs, including input combinations(A,B,Cin), output Sum, and Carry(Cout) bits, along with respective equations.

| **Inputs** | | | **Conventional FA** | | **Proposed FA** | |
|---|---|---|---|---|---|---|
| A | B | Cin | Sum | Cout | Sum | Cout |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| **Conventional FA** | | | Sum = A $\oplus$ B $\oplus$ Cin | | | |
| | | | Cout = (A $\oplus$ B) $\cdot$ C_in + (A $\cdot$ B) | | | |
| **Proposed FA** | | | Sum = (A + Cin) $\oplus$ B | | | |
| | | | Cout = (A + Cin) $\cdot$ B | | | |

Since the carry chain directly influences the correctness of subsequent sum bits, inaccuracies in carry generation or propagation can lead to substantial deviations in the final result, particularly in multi-bit arithmetic operations. The schematics of the conventional and proposed 1-bit approximate adders are shown in Figure 2.2.

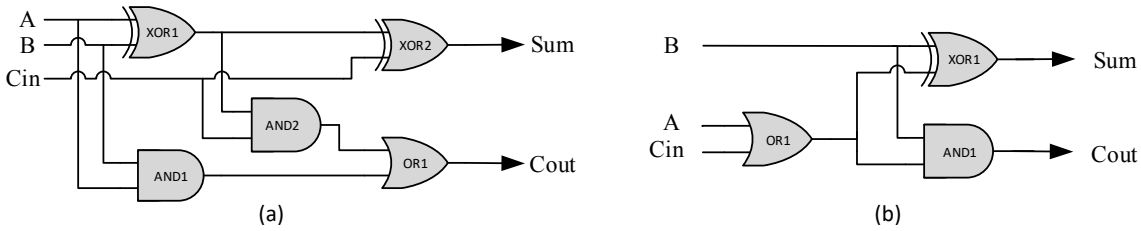



Figure 2.2: (a) Schematic of Conventional 1-bit Full Adder and (b) Schematic of the Proposed Approximate 1-bit Full Adder.

It is essential to differentiate this layout from alternative 3-gate approximate configuration such as the design proposed in [105]. While those conventional structures route and couple the primary inputs $(A + B)$ at the initial gate stage, the proposed topology intentionally pairs a primary input with the carry-in signal $(A + C_{\text{in}})$. This distinct wiring schematic alters the internal node transitions and critical-path boundaries, offering a unique structural variation within the broader classification landscape

of 3-gate approximate full adders.

### 2.3.1 Development of an n-bit Adder using the Proposed 1-bit Adder

The use of fully approximate adders across all bit positions in multi-bit arithmetic units can lead to excessive error accumulation, resulting in unacceptable degradation of computational accuracy. Consequently, contemporary adder design methodologies favor hybrid exact–approximate architectures, which strategically combine conventional and approximate adder circuits to balance accuracy and hardware efficiency.

In the state-of-the-art, accurate adders are employed for the most significant bit (MSB) positions, where numerical errors have a dominant impact on the overall output magnitude, while approximate adders are utilized for the least significant bit (LSB) positions, where errors contribute marginally to the final result [98, 79]. This selective approximation paradigm enables a controlled trade-off between precision and resource utilization, ensuring minimal accuracy degradation while achieving significant reductions in power consumption, silicon area, and critical path delay. Such hybrid adder architectures are therefore well suited for implementation in error-resilient computing domains, including deep neural network inference and digital signal processing applications.

Consequently, a combination of a conventional and a proposed approximate full adder is employed to develop a hybrid adder, HADD(x,y). The schematic of the

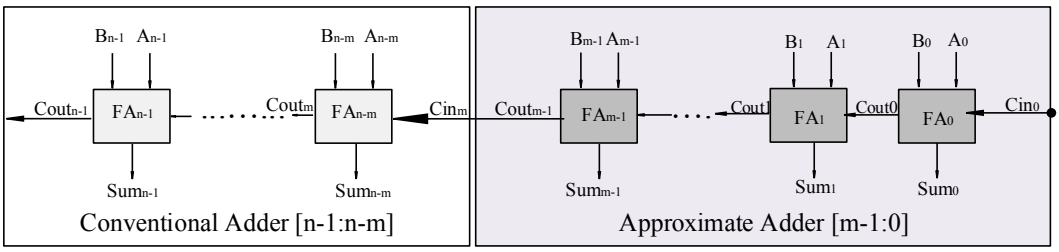



Figure 2.3: Schematic of the Proposed Approximate *n-bit* precision HADD (x,y) with the mixed-precision approach where x is the number of Conventional Adders and y is the number of Proposed Approximate Adders.

proposed approximate *n-bit* full adder, which utilizes a mixed-precision approach, is illustrated in Fig. 2.3. It can be observed that the least significant bits (LSBs) are computed using m approximate adders. In contrast, the calculation of the most significant bits (MSBs) is carried out using *(n-m)* accurate adders. This chapter introduces a new adder design named Hybrid ADDer (HADD), which is referred to by its abbreviation throughout the chapter. The HADD design combines four 1-bit accurate adders and four 1-bit approximate adders to create an 8-bit adder. Likewise, for developing a 16-bit adder, a total of 16 adders are employed, consisting of 8 accurate and eight approximate adders, using the same hybrid methodology as in HADD. The schematic provides a clear visualization of the mixed-precision approach, enabling more efficient and optimized circuit design for *n-bit* adders.

### 2.3.2 Delay Analysis of Full Adders: A Comparison of Conventional and Approximate Designs for 1-bit and *n-bit* Circuits

In assessing the physical performance of any circuit, delay is a critical parameter. According to prior studies, the properties of individual logic gates affect the propagation delay of circuits [106]. To calculate the propagation delays of the carry-bit and sum-bit in *n-bit* adder, the below Eq.2.1(a) for the carry-bit and Eq. 2.1(b) for the sum-bit can be analyzed.

$$\tau_c = \sum_{j=m}^{(n-1)} \mathbf{t}_{c_j} + \sum_{j=1}^{(m-1)} \mathbf{T}_{c_j} + \mathbf{T}_c \tag{2.1a}$$

$$\tau_s = \sum_{j=m}^{(n-1)} \mathbf{t}_{c_j} + \sum_{j=1}^{(m-1)} \mathbf{T}_{c_j} + \mathbf{T}_s \tag{2.1b}$$

Here, $\tau_c$ and $\tau_s$ represent the total propagation delays for computing the carry-bit and sum-bit in the proposed *n-bit* precision HADD. $j$ stands for the adder's index, whereas $m$ indicates the approximate number of adders used. In the case of a 1-

bit conventional adder, the propagation delays for computing the carry and sum bits are denoted by $tc$ and $ts$, respectively. These values are calculated using Eq. 2.2(a) and 2.2(b). $T_s$ and $T_c$ are the propagation delays in the computation of the sum bit and carry bit using the proposed approximate adder. Therefore, the equation can be rewritten as shown in Eq. 2.3(a) and 2.3(b).

$$\mathbf{t}_c = \mathbf{t}_{\texttt{OR}} + \max[\text{delay}(A \oplus B \cdot Cin,\ A \cdot B)] \tag{2.2a}$$

$$\mathbf{t}_s = \mathbf{t}_{\texttt{XOR}} + \max[\text{delay}(A \oplus B,\ Cin)] \tag{2.2b}$$

$$\mathbf{T}_c = \mathbf{t}_{\texttt{AND}} + \max[\text{delay}(A + C_{in},\ B)] \tag{2.3a}$$

$$\mathbf{T}_s = \mathbf{t}_{\texttt{XOR}} + \max[\text{delay}(A + C_{in},\ B)] \tag{2.3b}$$

Table 2.2: Comparison of Gate Count between the Accurate and the Proposed Approximate Adders for *n-bit* Addition.

| **Bit-precision of adder** | **XOR gates** | **AND gates** | **OR gates** | **# of gates** |
|---|---|---|---|---|
| **1-bit accurate** | 2 | 2 | 1 | 5 |
| **1-bit proposed** | 1 | 1 | 1 | 3 |
| **8-bit Conventional** | 16 | 16 | 8 | 40 |
| **8-bit proposed HADD** | 12 | 12 | 8 | 32 |
| **16-bit Conventional** | 32 | 32 | 16 | 80 |
| **16-bit proposed HADD** | 24 | 24 | 16 | 64 |

The propagation delay of a digital circuit depends on both the applied input signals and the number of logic gates used in its implementation. To evaluate the efficiency of the conventional and proposed adder architectures, a gate-count comparison is presented in Table 2.2. The analysis reveals that the difference in gate count between the conventional and proposed adders increases linearly with increasing bit precision. Notably, for any n-bit addition, the proposed adder consistently requires fewer logic gates than the conventional design. This reduction in gate count suggests that, for the same bit precision, the proposed adder is likely to exhibit a lower propagation delay compared to the conventional adder

### 2.3.3 Efficient HADD Design: Pareto-Driven Approximate Adder Selection

The primary goal is to design an $n$-bit adder that optimizes hardware utilization while maintaining acceptable accuracy. To achieve this, various combinations of `x` conventional adders and `y` approximate adders were evaluated. Approximate adders were placed in the least significant bit (LSB) positions, while conventional adders were used in the most significant bit (MSB) positions, and the resulting accuracy was analyzed. The results indicate that the overall accuracy of the adder depends strongly on the specific combination of adders employed. Consequently, different single-bit adder configurations can be selected based on the error-tolerance requirements of the target application. However, the analysis shows that at least $\frac{n}{2}$ conventional (accurate) adders are required to maintain optimal accuracy without incurring excessive hardware overhead. This configuration was therefore adopted in the HADD design, and its overall performance was subsequently evaluated. The error metrics for Edge Detection application are reported in Table 2.3, including the Standard Deviation (SD), Normalised Mean Squared Error (NMSE), and Normalised Mean Euclidean Distance (NMED), have been utilised to evaluate and quantify the dissimilarities between two images. These metrics are computed using equations 2.4, 2.5, and 2.6, with $g$ and $h$ denoting the dimensions of the images. In the equations, $c$ and $d$ correspond to the indices of the individual pixels within the images. More specifically, the image $I_1$ represents the result obtained using an accurate adder, while $I_2$ corresponds to the image acquired using the HADD (x,y) design. In equations, 2.4b and 2.4c, the variable $VAR$ denotes the variance.

$$\text{Mean} = \frac{1}{g \cdot h} \sum_{c=1}^{g} \sum_{d=1}^{h} I_2(c,d) \tag{2.4a}$$

$$\text{Variance (VAR)} = \frac{1}{g \cdot h} \sum_{c=1}^{g} \sum_{d=1}^{h} (I_2(c,d) - Mean)^2 \tag{2.4b}$$

$$\text{Standard Deviation (SD)} = \sqrt{(\text{VAR})} \tag{2.4c}$$

$$\text{Mean Square Error (MSE)} = \frac{1}{g \cdot h} \sum_{c=1}^{g} \sum_{d=1}^{h} \left(I_1(c,d) - I_2(c,d)\right)^2 \tag{2.5a}$$

$$\text{Normalized MSE (NMSE)} = \frac{1}{max\left[\left(I_1(c,d) - I_2(c,d)\right)^2\right]} \text{MSE} \tag{2.5b}$$

$$\text{Mean Euclidean Distance (MED)} = \frac{1}{g \cdot h} \sum_{c=1}^{g} \sum_{d=1}^{h} abs\left[I_1(c,d) - I_2(c,d)\right] \tag{2.6a}$$

$$\text{Normalized MED (NMED)} = \frac{1}{max\big(abs[I_1(c,d) - I_2(c,d)]\big)} \text{MED} \tag{2.6b}$$

Table 2.3: Performance Evaluation of 8-bit and 16-bit Proposed HADD designs based on Error Metrics.

| **Proposed HADD (n-bit)** | **NMED** | **SD** | **NMSE** |
|---|---|---|---|
| **8-bit HADD (4, 4)** | 0.156 | 0.450 | 0.088 |
| **16-bit HADD (8, 8)** | 0.160 | 0.014 | 0.079 |

The results presented in Table 2.3 demonstrate that the HADD design effectively reduces computational deviations for both 8-bit and 16-bit precisions when compared with accurate computations. This highlights the effectiveness of the HADD architecture in achieving high computational accuracy. Furthermore, the influence of the least significant bits (LSBs) on the overall accuracy of an $n$-bit HADD is relatively minor, which significantly contributes to the favorable results observed. Based on these findings, the deployment of the HADD configuration with a $\frac{n}{2}$ combination of conventional and approximate adders is strongly recommended. This configuration ensures a desirable balance between computational accuracy and hardware resource efficiency. Accordingly, the recommended HADD configuration has been implemented, reflecting a deliberate effort to achieve precise results while optimizing resource utilization in the HADD design.

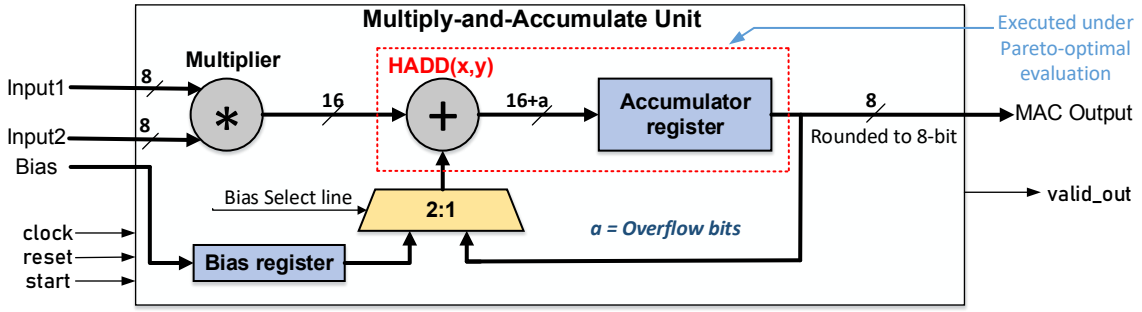



Figure 2.4: Schematic of an 8-bit MAC unit employing the proposed HADD design, with overflow bits indicated by 'a'.

For performance analysis, the proposed HADD design has been integrated into a multiply–accumulate (MAC) unit. The specific implementation of an 8-bit MAC unit is illustrated in Fig. 2.4. It is important to note that the output of the multiplier has twice the precision of its input operands and is subsequently forwarded to the accumulator stage. Although several recent studies have investigated techniques such as operand sparsity and irrelevant data suppression to reduce energy consumption in DNN accelerators, in this study, the MAC operation is initiated only upon the reception of a valid `start` signal.

The `bias_select_line` controls the selection between the bias register value and the feedback from the accumulator register, depending on the specific MAC execution. The accumulation operation is performed using the proposed HADD architecture, and the resulting output is stored in the accumulator register. The accumulator output is subsequently rounded to 8-bit precision using the `roundTiesToEven` method, consistent with the default rounding mode specified in IEEE 754-2008. Upon completion of the MAC operation, the `valid_out` signal is asserted to indicate that the final output is valid and available in the accumulator register. This concludes the MAC operation, after which the output is forwarded to the activation function block for subsequent processing

## 2.4 Experimental Analysis

The experimental analysis validates the performance of the proposed design through both software and hardware implementation flows. A Python-based emulator was employed to integrate the proposed design into MAC operations within a CNN model. The following evaluations are done:

- Pareto-study for Efficient HADD design: A Pareto study on multi-bit precision adders in the HADD design was conducted to optimize the combination of accurate and approximate adders in the MSBs and LSBs for improved physical performance and inference accuracy. The deployment of a Pareto optimal evaluation is essential for navigating the multi-dimensional design space where hardware savings (Area and Power) compete directly with algorithmic correctness (Inference Accuracy). In an $n$-bit hybrid adder, configuring the exact boundary between accurate ($x$) and approximate ($y$) bits produces numerous potential structural permutations. The Pareto frontier systematically evaluates these combinations and discards sub-optimal designs.

- Inference Accuracy: To evaluate the accuracy of the selected combination of exact and approximate adders in HADD, the design was tested on the LeNet-5 DNN model using the MNIST and A–Z Handwritten Alphabet datasets.

- Image Feature Extraction: The scope of the proposed HADD is extended to include image processing applications such as edge detection. In this application, the Sobel operator is used in convolution and HADD-based MAC operation for image feature extraction.

- Physical Performance: The HADD design was implemented with various bit precisions, and physical performance parameters were extracted. The study includes a comparison with conventional designs and state-of-the-art architectures to analyze the physical performance of HADD

The complete design methodology for developing an efficient HADD architecture is illustrated in Fig. 2.5.

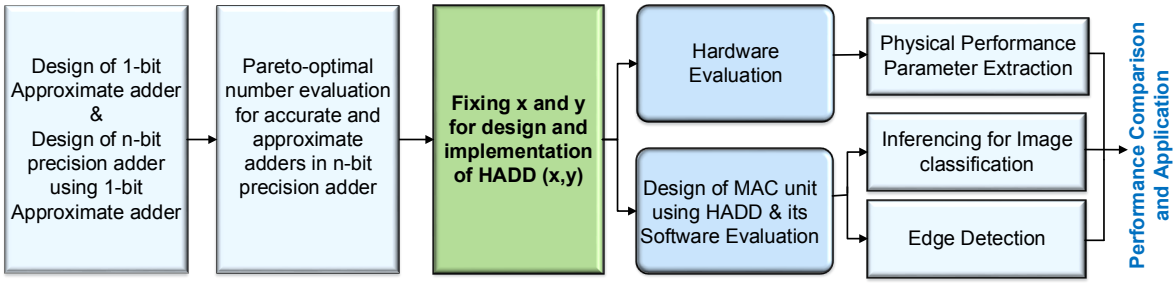



Figure 2.5: Design Process illustrating efficient HADD implementation and its integration into a MAC unit for DNNs, including Hardware Performance Evaluation and Software-Level Assessment for Edge Detection Applications.

This methodology integrates hardware performance evaluation with the subsequent deployment of the HADD within a MAC unit for DNN applications. In addition, the design flow incorporates software-level evaluations, specifically targeting edge detection applications. A key stage of the process involves Pareto-optimal analysis, which identifies the optimal combination of accurate and approximate adders for HADD implementation. Following the selection of this optimal configuration, extensive hardware and software evaluations are performed to assess the HADD architecture. The resulting performance metrics are then analyzed and reported, providing a comprehensive evaluation of the effectiveness and efficiency of the proposed HADD design.

### 2.4.1 Software Implementation and Accuracy Evaluation

In this study, multiple configurations of HADD were initially investigated using $x$ accurate adders in the most significant bit (MSB) positions and $y$ approximate adders in the least significant bit (LSB) positions, denoted as HADD($x$,$y$), to identify the optimal combination. Error metrics, as discussed in Section 2.3, were analyzed, revealing that the inclusion of at least $\frac{n}{2}$ accurate 1-bit adders in an $n$-bit adder is necessary to ensure an efficient and reliable implementation. Furthermore, Pareto-optimal analysis identified $\frac{n}{2}$ approximate adders as the optimal choice for minimizing accuracy degradation while maximizing hardware efficiency.

Based on these findings, the selected HADD configuration was integrated into MAC operations and evaluated using the LeNet DNN architecture with software-level cus-

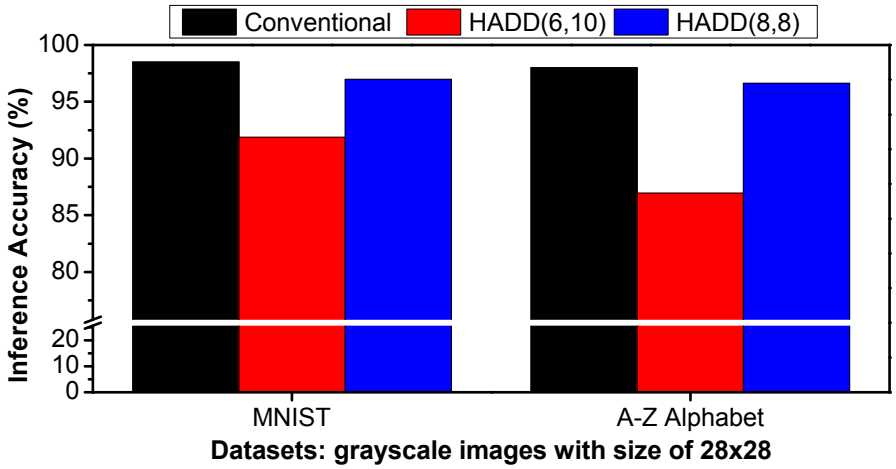



Figure 2.6: Accuracy comparison for HADD Design with combinations of Accurate and Proposed Approximate Full Adders and Conventional *n-bit* Adder Design on LeNet DNN model.

tomization. The evaluation employed grayscale images with 8-bit pixel precision and a resolution of $28 \times 28$ pixels to perform inference on the MNIST and A–Z Handwritten Alphabet datasets.

The experimental results indicate that inference accuracy varies with the proportion of accurate and approximate adders. As shown in Fig. 2.6, the conventional adder, HADD(6,10), and HADD(8,8) configurations achieved accuracy rates of 98.5%, 91.87%, and 96.97%, respectively. Although several HADD combinations were evaluated, only representative configurations are reported for clarity. Notably, HADD$\left(\frac{n}{2}, \frac{n}{2}\right)$ consistently delivered superior accuracy among the evaluated approximate configurations.

Compared to a fully accurate $n$-bit adder, the proposed HADD design incurs a modest accuracy loss of approximately 1.5% on the MNIST dataset. This evaluation was further extended to higher-class-count datasets using the A–Z Handwritten Alphabet dataset, where a similar accuracy reduction of approximately 1.3% was observed. Despite this minor degradation, the resulting accuracy remains acceptable for many applications, particularly when weighed against the substantial hardware resource savings and improvements in other physical performance parameters enabled by the HADD architecture.

(a) (b) (c) (d) (e)

Figure 2.7: (a) Original image (b) Image after $VED_{acc}$ (c) Image after $VED_{app}$ (d) Image after $HED_{acc}$ (e) Image after $HED_{app}$.

### 2.4.2 Evaluation of HADD for Edge Detection

Image and video processing applications typically involve computationally intensive convolution operations that rely on large numbers of iterative multiplications and additions, resulting in significant power consumption and hardware complexity. These operations form the core of many vision-based tasks, including filtering, feature extraction, and object detection. However, a wide range of image processing applications exhibit inherent error tolerance, as small numerical inaccuracies often have minimal impact on perceptual quality or algorithmic correctness. This tolerance makes approximate computing a promising approach for designing energy-efficient and high-performance hardware architectures [107].

In this subsection, the effectiveness of the proposed HADD architecture is evaluated in the context of edge detection applications, which are fundamental pre-processing stages in many computer vision systems. Edge detection plays a critical role in identifying object boundaries and structural features and is widely employed in practical applications such as license plate detection and recognition systems [108]. By integrating the HADD design into the convolution operations required for edge detection, the trade-off between computational accuracy and hardware efficiency is exploited, demonstrating that meaningful energy and resource savings can be achieved without significantly degrading application-level performance.

The Sobel operator is used to detect edges in various grayscale images, and the results obtained using conventional and approximate adders are compared. The experimental results for horizontal edge detection (HED) and Vertical Edge Detection

(a) (b) (c) (d) (e)

Figure 2.8: (a) Original image (b) Image after $VED_{acc}$ (c) Image after $VED_{app}$ (d) Image after $HED_{acc}$ (e) Image after $HED_{app}$.

(VED), shown in Fig. 2.7, demonstrate the promising potential of the proposed approximate accumulator for edge detection applications. Figures 2.7(b) and 2.7(c) illustrate the VED outputs obtained using the conventional adder ($VED_{acc}$) and the proposed approximate adder ($VED_{app}$), respectively. It is evident that the images do not differ visually at a large scale and exhibit comparable results, thereby demonstrating the effectiveness of the proposed HADD design. A similar observation can be made for Figs. 2.7(d) and 2.7(e), which depict clearly detected edges for HED in both cases.

Furthermore, edges are clearly detected using the proposed approximate adder design for random images, as illustrated in Fig. 2.8. A closer inspection of Figures 2.8(b) and 2.8(c), as well as Figures 2.8(d) and 2.8(e), indicates that the images obtained after both HED and VED using the proposed HADD design exhibit convincing and visually comparable results. These observations confirm the applicability and robustness of the proposed approximate adder design for edge detection applications.

### 2.4.3 Hardware Efficiency Evaluation of the Proposed HADD Architecture

The physical performance parameters of the conventional adder and the proposed HADD architectures, implemented using a 180 nm technology node across different bit precisions, are summarized in Table 2.4. The evaluated metrics include area, propagation delay, static power, and dynamic power. By employing a hybrid combination of accurate and approximate adders, the proposed HADD architecture achieves a favorable trade-off between computational accuracy and hardware efficiency when compared to the conventional adder.

For the 1-bit precision case, the proposed adder exhibits marginal improvements

Table 2.4: Physical Performance Parameters for Conventional and Proposed HADD adders at 180nm Technology Node.

| **Bit precision** | **Area**($um^2$) | **Delay (ns)** | **Power** | |
|---|---|---|---|---|
| | | | Static(nW) | Dynamic(uW) |
| **1-bit conventional** | 60.61 | 2.14 | 0.575 | 0.002 |
| **1-bit proposed** | 49.08 | 2 | 0.560 | 0.001 |
| **8-bit conventional** | 837 | 4.65 | 10.06 | 24.3 |
| **8-bit proposed HADD (4, 4)** | 456 | 3.78 | 4.5 | 11.8 |
| **16-bit conventional** | 1999 | 4.90 | 34.63 | 54.63 |
| **16-bit proposed HADD (6, 10)** | 1079 | 4.15 | 16.38 | 31.36 |
| **16-bit proposed HADD (8, 8)** | 1408 | 4.35 | 23.81 | 39.27 |

in area and dynamic power consumption while maintaining comparable delay and static power characteristics relative to the conventional adder. These results indicate that the proposed design can deliver equivalent performance with modest savings in hardware resources. At 8-bit precision, the HADD(4,4) configuration demonstrates substantial improvements across all evaluated metrics, achieving reductions of 44% in area, 19% in delay, 55% in static power, and 51% in dynamic power compared to the conventional adder. This highlights the suitability of the proposed architecture for higher-precision applications requiring enhanced performance and energy efficiency.

Similarly, for the 16-bit precision case, the HADD(6,10) configuration consistently outperforms the conventional adder, yielding reductions of up to 46% in area, 15% in delay, 53% in static power, and 43% in dynamic power. Furthermore, the proposed HADD architecture offers flexibility in navigating the accuracy–efficiency trade-off through different combinations of accurate and approximate adders. Among the evaluated configurations, HADD(6,10) and HADD(8,8) achieve the most balanced and favorable overall performance.

The performance evaluation of the proposed HADD adders, as reported in Table 2.5, indicates substantial improvements in key physical performance parameters over conventional adders implemented at the 45 nm technology node across multiple bit precisions. For the 1-bit configuration, the proposed adder achieves a 48% reduction in area, along with reductions of 9.5% in static power and 7% in dynamic power consumption, relative to the accurate 1-bit adder.

At 8-bit precision, the HADD(4,4) configuration demonstrates notable enhancements, including a 31% reduction in area, an 8.4% reduction in propagation delay, a 20% reduction in static power, and a 34% reduction in dynamic power consumption compared to the accurate adder. Similarly, for the 16-bit precision case, the HADD(6,10) configuration yields reductions of 37% in area, 6% in delay, 33% in static power, and 8% in dynamic power consumption. These results validate the scalability of the proposed HADD architecture across different bit precisions, as further corroborated by the results presented in Tables 2.4 and 2.5.

A comparative assessment of the proposed design against recent state-of-the-art 16-bit adder architectures is presented in Table 2.6. The comparison considers four critical metrics: area, delay, power consumption, and data arrival time (DAT). The data arrival time is intrinsically linked to the overall circuit delay, as it defines the maximum permissible timing at which input data becomes stable and can be correctly sampled by the receiving circuitry. Consequently, DAT serves as an essential indicator of timing robustness and functional reliability.

To further assess the suitability of the proposed HADD design for area-efficient Edge-AI applications, comparisons were conducted with several existing architectures, including SBFTA-I at 90 nm, OFTA at 90 nm, CSLA (CBL) at 90 nm, and BEC at 180 nm. The results demonstrate that, at an equivalent technology node, the proposed HADD design achieves a 26% reduction in area and a 58% reduction in power consumption relative to the BEC architecture. Moreover, when compared to the

Table 2.5: Physical Performance Parameters for Conventional and Proposed HADD Adders at 45nm Technology Node.

| **Bit precision** | **Area** ($um^2$) | **Delay(ns)** | **Power** | |
|---|---|---|---|---|
| | | | Static(nW) | Dynamic(uW) |
| **1-bit conventional** | 17.36 | 0.90 | 43.33 | 0.341 |
| **1-bit proposed** | 8.92 | 0.85 | 39.2 | 0.330 |
| **8-bit conventional** | 101 | 1.35 | 298.6 | 3.47 |
| **8-bit proposed HADD (4, 4)** | 70 | 1.19 | 240 | 2.30 |
| **16-bit conventional** | 226 | 1.82 | 735.2 | 7.5 |
| **16-bit proposed HADD (6, 10)** | 141.7 | 1.30 | 492.4 | 6.9 |
| **16-bit proposed HADD (8, 8)** | 161 | 1.71 | 515 | 7.06 |

Table 2.6: Comparison of Hardware Implementation Parameters of the Proposed Adder with 16-bit Precision State-of-the-Art Works.

| **Adder** | **SBFTA-I** [111] | **OFTA** [111] | **CSLA** [112] | **BEC** [113] | **EOHEAA**[109] | **NAA** [110] | **HADD** | **HADD** |
|---|---|---|---|---|---|---|---|---|
| **Technology node** | 90nm | 90nm | 90nm | 180nm | 90nm | 28nm | **45nm** | **180nm** |
| **Area** ($\mu$**m**$^2$) | 403.43 | 406.46 | 1722.96 | 1834 | 195.28 | 65.42 | **161** | **1408** |
| **Delay (ns)** | 1.65 | 1.75 | 7.38 | 2.97 | 1.07 | 1.10 | **1.71** | **4.35** |
| **Power** ($\mu$**W**) | 12.41 | 12.84 | 12.87 | 94.7 | 7.49 | 29.61 | **7.06** | **39.27** |
| **DAT** (**ns**·$\mu$**m**$^2$) | - | - | 10.45 | - | - | | **3.71** | **6.35** |

CSLA (CBL) design, the proposed HADD achieves reductions of 39% in DAT, 26% in area, and 41% in delay, despite being implemented at a more advanced technology node. The proposed HADD architecture was also compared with recent state-of-the-art works, namely EOHEAA [109] at 90 nm and NAA [110] at 28 nm. The HADD implementation demonstrates strong competitiveness across these diverse technology generations; it utilizes an area of 161$\mu$m$^2$ and a power profile of 7.06$\mu$W, which offers an improvement over EOHEAA's 7.49$\mu$W power consumption and a substantial power reduction compared to the 28 nm NAA design (29.61$\mu$W). This proves that HADD's 3-gate full-adder optimization offers an inherently high-efficiency hardware alternative for Edge-AI accelerators.

The proposed HADD architecture was implemented using both 45 nm and 180 nm technology nodes to evaluate its physical performance across advanced and mature process technologies. The evaluation results demonstrate that HADD outperforms existing related works in terms of area and power consumption, while maintaining comparable propagation delay. Moreover, the results obtained at both 180 nm and 45 nm technology nodes show that the proposed HADD$\left(\frac{n}{2}, \frac{n}{2}\right)$ configuration consistently achieves substantial reductions in area, delay, and power consumption compared to conventional adders across multiple bit precisions. These improvements are achieved while providing flexible trade-offs between computational accuracy and hardware efficiency, enabling the design to be tailored to diverse application requirements.

Overall, the proposed HADD architecture exhibits significant performance advantages over state-of-the-art designs for 16-bit precision, thereby establishing its suitability for area-optimized implementations in Edge-AI applications. Consequently, the HADD design is well suited for application domains that demand hardware-efficient

and energy-conscious computation.

## 2.5 Summary

This chapter proposes an approximate 1-bit adder and presents a Hybrid Adder Design, HADD$\left(\frac{n}{2}, \frac{n}{2}\right)$, that achieves accuracy comparable to conventional adders while delivering enhanced hardware efficiency. To this end, 1-bit approximate adders were developed, and a comprehensive Pareto analysis was conducted to determine the optimal combination of accurate and approximate adders for $n$-bit precision HADD designs. The analysis reveals that including at least $\frac{n}{2}$ accurate 1-bit adders is necessary to maintain acceptable accuracy, whereas selecting $\frac{n}{2}$ approximate adders optimally enables minimal accuracy degradation while achieving significant hardware savings.

The proposed HADD adders were further integrated into the MAC operations of the LeNet deep neural network architecture and evaluated on the MNIST and A–Z Handwritten Alphabet datasets. The resulting inference accuracy exhibited only marginal degradation, with losses of approximately 1.5% and 1.3% compared to implementations using fully accurate $n$-bit adders. Despite this slight loss in accuracy, the proposed HADD architecture offers flexible trade-offs between accuracy and hardware efficiency, depending on application-specific requirements. Overall, the proposed HADD adders achieve substantial reductions in area, latency, and power consumption across a wide range of bit precisions, making them a compelling solution for efficient hardware implementation in error-resilient and energy-constrained applications. This chapter emphasizes approximation in the adder, one of the key blocks of the MAC unit. By introducing approximation at the adder level, the architecture achieves a more resource-efficient implementation of the MAC unit, significantly reducing hardware complexity while maintaining acceptable computational accuracy. Building on these insights, the subsequent chapters shift attention toward the optimization of another essential arithmetic component—the multiplier—further extending the exploration of efficient design techniques for high-performance arithmetic units.

# Chapter 3

# Operand-Rounding-based Approximate Multiplier Design

In chapter 2, the adder approximation within the MAC unit was presented, reducing overall hardware complexity with only a minor degradation in accuracy. This approach resulted in savings in the hardware resources required for MAC unit processing, especially for error-resilient applications. However, the multiplier is considerably more resource-intensive and power-hungry than the adder within the MAC unit, consuming a larger portion of area and power [10]. This is mainly due to the complexity of partial product generation and accumulation involved in multiplication. Consequently, the multiplier often dominates the overall hardware cost of the MAC unit. Therefore, optimizing the multiplier is critical to improving overall resource utilization and achieving an efficient MAC unit architecture.

## 3.1 Introduction

Several prior studies have focused on optimizing multiplier architectures to achieve efficient resource utilization through a variety of approximation techniques [47, 55, 49]. In particular, researchers have investigated approximate multipliers that exploit LUT (Look-Up Table) structures and fast-carry chains to reduce both computational delay and hardware resource consumption [82]. Approximation has also been applied at the reduction tree stage of the multiplier, enabling further simplification of the

accumulation of partial products [100].

Among the most popular approximation techniques is the logarithmic multiplier proposed by Mitchell [57]. Iterative logarithmic multipliers have been shown to effectively replace conventional multipliers by reducing the number of complex arithmetic operations, thereby enabling area-efficient hardware implementations for neural networks [114]. Building upon this concept, Liu et al. analyzed logarithm-based approximate multipliers using various approximate adders and demonstrated that designs employing set-one adders (SOAs) can achieve improved accuracy [84]. Other studies have focused on mitigating the average error inherent in Mitchell's algorithm, albeit at the expense of increased power and area consumption [58, 115].

An alternative optimization strategy involves combining multipliers of different precisions to dynamically adapt to varying accuracy requirements [116]. While this method offers flexibility, it relies heavily on a high-precision multiplier, which substantially increases the overall area overhead. Since multiplication fundamentally depends on partial product generation, reducing the number of partial products has also been identified as an effective means of trading accuracy for improvements in speed and hardware efficiency [45, 28]. In this context, the use of modified Booth encoding to design approximate Wallace–Booth multipliers has been explored as a technique to reduce delay [83]. Boolean rewriting has likewise been widely adopted to simplify multiplier circuits by modifying Boolean expressions and Karnaugh maps, resulting in notable area and power savings at the cost of marginal accuracy degradation [117, 118].

Another straightforward yet effective approach to approximate multiplier design is to introduce approximation at the input level [86, 88, 87]. By reducing operand bitwidth prior to multiplication, this technique significantly lowers computational complexity and hardware requirements. For instance, in [85], leading-one detection is employed to select $m$ consecutive bits from the leading-one position of each operand, enabling an $m \times m$ multiplication instead of a full $n \times n$ operation, where $m < n$. Similarly, in [86], operands are truncated to $h$ and $t$ bits based on the position of their leading-one bits, and these truncated values are subsequently used for multiplication and accumulation. To reduce truncation-induced error, the truncated operands are

rounded to approximate their discarded portions, thereby supporting both symmetric and asymmetric multiplication with reduced power consumption and delay. In [87], rounding-based multiplication is further refined by rounding input operands to the nearest power of two, yielding higher accuracy compared to conventional rounding methods.

Overall, these input pre-processing techniques substantially reduce computational complexity while achieving significant savings in area and power consumption, thereby demonstrating strong potential as effective approximation strategies for efficient multiplier design. In this chapter,an approximate multiplier based on operand rounding is presented. The idea is to pre-process the inputs so the operand widths are reduced, then shift the product as required after multiplication. Reducing operand bitwidths reduces computational complexity, ultimately saving area and power. The induced error is minimised by shifting the approximate product obtained. This chapter focuses on developing an operand rounding-based approximate multiplier for error-tolerant AI/ML applications. The study focuses on integer and fixed-point arithmetic and is currently considered for unsigned multiplications. The key outcomes of this investigation are as follows:

- **Rounding-Based Approximate Multiplier Design:** An approximate multiplier architecture based on rounding and truncation of input operands, that supports integer and fixed-point computations, ensuring efficient resource utilisation.

- **Hardware-efficient Multiplication:** The proposed multiplier supports all bitwidths upto 8-bit multiplication. The inputs are rounded off to 5 bits following a unique proposed rounding scheme. This results in significant savings in hardware, traded off with only a little degradation in accuracy.

- **ASIC and FPGA-based implementation:** The proposed architecture is verified for both ASIC and FPGA-based implementations, highlighting its versatility.

### 3.1.1 Proposed Operand-Rounding based Multiplier Architecture

This chapter presents an approximate multiplier architecture that directly processes input operands with bitwidths ranging from 1 to 5 bits without modification. For higher bitwidth operands, the proposed design employs a rounding mechanism to reduce the input precision to 5 bits. Consequently, exact multiplication is applied to lower bit-width operands, while approximate multiplication is employed for higher bit-width operands, as detailed in the following subsections. Although the scheme is demonstrated for rounding 8-bit inputs to 5 bits, it is inherently scalable and can be extended to higher bitwidths. In general, the objective is to reduce an input operand of bitwidth $n$ to approximately $[(n/2)\ +\ 1]$ bits, thereby significantly lowering the computational complexity associated with higher-precision multiplication while maintaining acceptable accuracy.

### 3.1.2 Input Operand Rounding Scheme

The proposed approximate 8-bit multiplier architecture reduces computational complexity and improves efficiency by incorporating an input pre-processing stage that approximates the operands prior to multiplication, followed by a shift-based correction of the output. The architecture supports variable operand precisions, enabling input features and weights with bitwidths ranging from 1 to 8 bits.

Operands with bitwidths from 1 to 5 bits are processed without modification and are multiplied directly using a conventional $5 \times 5$ multiplier. For operands with higher bitwidths, a leading-one-based segmentation strategy is employed to introduce controlled approximation. As the input bitwidth increases, the approximation logic progressively reduces the operand resolution by selectively combining adjacent bits using bitwise `OR` and `AND` operations. A detailed schematic of the proposed architecture is illustrated in Fig. 3.1. For example, when an input operand $A$ lies in the range $[32, 63]$, the most significant bits $[5:2]$ are preserved, while the two least significant bits are merged into a single representative bit using the proposed rounding scheme. The re-

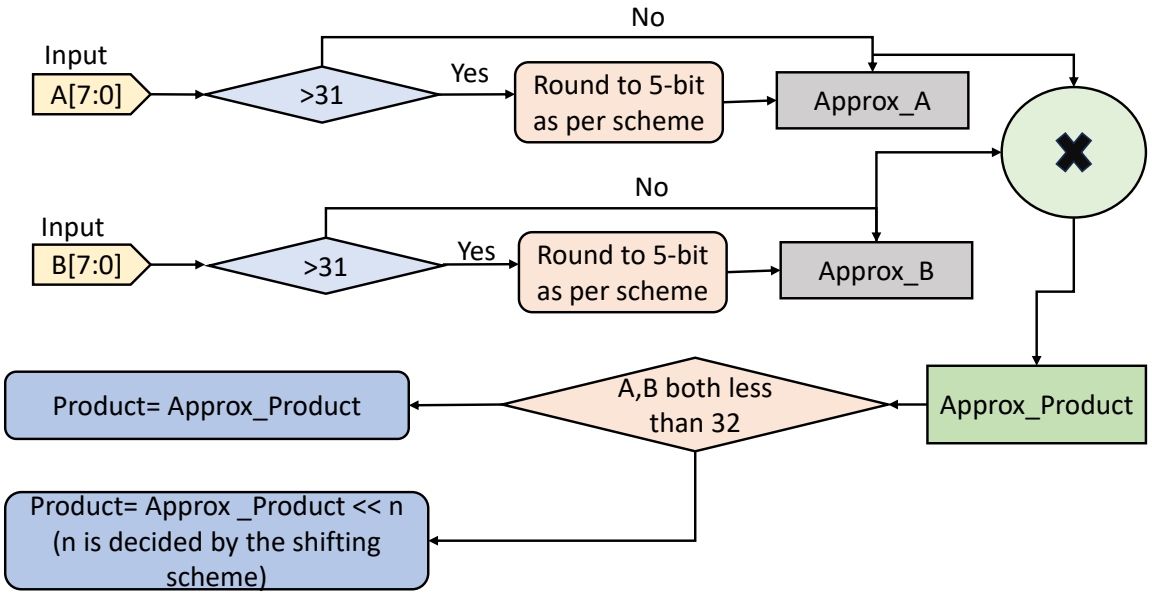



Figure 3.1: Architecture of the Proposed Rounding-based Approximate 8-bit Multiplier. The inputs are pre-processed (rounded) before Multiplication Operation.

sulting approximated operand is denoted as *approx_A*. The proposed scheme functions by partitioning the input dynamic range into multiple intervals, where each interval is associated with a distinct level of representational precision. The fundamental idea is to approximate each input value by preserving the leading bits that contribute most significantly to its magnitude, while progressively reducing the number of bits used to represent the less significant portions.

For input values in the range $[0, 31]$, all five least significant bits of the operand $A$ are retained in *approx_A*, ensuring that no approximation is introduced in this interval. When the input lies in the range $[32, 63]$, the approximation preserves the four most significant bits, $A[5:2]$, while the two least significant bits are combined into a single representative bit using a logical summary operation, specifically a bitwise `OR` of $A[1]$ and $A[0]$. For values within the range $[64, 127]$, the three most significant bits, $A[6:4]$, are maintained, and the remaining lower-order bits are further compressed by summarizing the bit pairs $A[3:2]$ and $A[1:0]$ using similar `OR`-based logic. In the highest range, $[128, 255]$, which represents the default case, only the two most significant bits, $A[7:6]$, are retained explicitly, while each of the lower 2-bit groups, namely $A[5:4]$, $A[3:2]$, and $A[1:0]$, is approximated using the same summary

technique.

Table 3.1: Illustration of Rounding Operation on the Input Operand A.

| **Step** | **Action** |
|---|---|
| **Input** | An 8-bit input $A \in [0, 255]$ |
| **Output** | $approx\_A$ |
| **If** $0 \leq A < 32$ | No change required<br>$approx\_A \leftarrow (A[4:0])$ |
| **If** $32 \leq A < 64$ | $temp \leftarrow (A[1] \& (\lvert A[1:0]))$<br>$R \leftarrow \text{Concatenate}(A[5:2], temp)$ |
| **ElseIf** $64 \leq A < 128$ | $temp1 \leftarrow (A[3] \& (\lvert A[3:2]))$<br>$temp2 \leftarrow (A[1] \& (\lvert A[1:0]))$<br>$R \leftarrow \text{Concatenate}(A[6:4], temp1, temp2)$ |
| **ElseIf** $128 \leq A \leq 255$ | $temp1 \leftarrow (A[5] \& (\lvert A[5:4]))$<br>$temp2 \leftarrow (A[3] \& (\lvert A[3:2]))$<br>$temp3 \leftarrow (A[1] \& (\lvert A[1:0]))$<br>$R \leftarrow \text{Concatenate}(A[7:6], temp1, temp2, temp3)$ |

By employing simple bitwise `OR` and `AND` operations, the proposed method effectively reduces the precision of the lower-order bits while preserving a coarse representation of their contribution. This approach significantly reduces the output bitwidth and computational complexity, resulting in lower hardware area and power consumption while maintaining the essential characteristics of the original input operand. The detailed rounding scheme employed in this study is summarized in Table 3.1. An identical approximation procedure is applied to the second input operand. It is important to note that for lower-bitwidth inputs, specifically values less than 32, the approximated operand $approx\_A$ remains identical to the original input $A$, and no approximation is introduced. Once both input operands have been approximated, the multiplication is performed using their respective approximated representations. The resulting product is then adjusted via an appropriate shift operation, determined by the magnitude of the input operands.

### 3.1.3 Shifting Scheme in the Multiplier

The product obtained after operand approximation is significantly smaller in magnitude than the exact product, particularly for large input values. To compensate

for this compression and partially restore the original dynamic range, the proposed module estimates the number of significant bits discarded during the approximation and applies a corresponding left-shift to the product.

Table 3.2: Shift Amount Based on Bit-Widths of Operands A and B.

| **Case** | **Shift Amount** |
|---|---|
| $bw_A = 5$ and $bw_B = 6$ or $bw_A = 6$ and $bw_B = 5$ | 1 |
| $bw_A = 6$ and $bw_B = 6$ | 2 |
| $bw_A = 6$ and $bw_B = 7$ or $bw_A = 7$ and $bw_B = 6$ | 3 |
| $bw_A = 7$ and $bw_B = 7$ | 4 |
| $bw_A = 7$ and $bw_B = 8$ or $bw_A = 8$ and $bw_B = 7$ | 5 |
| $bw_A = 8$ and $bw_B = 8$ | 6 |

The shift amount is determined by the position of the most significant bit (MSB) of the input operands, which indicates their relative magnitude. Based on the MSB position, the inputs are classified into distinct magnitude ranges, with each range corresponding to a specific shift factor. The total number of such ranges, therefore, directly determines the required shift amount applied to the operands. A detailed summary of the computed shift values for the different input ranges is provided in Table 3.2. This shift-based scaling mechanism allows higher bit-width operands to be efficiently normalized prior to multiplication, ensuring minimal degradation in computational accuracy while maintaining hardware efficiency.

### 3.1.4 Multiplication Operation using the Proposed Multiplier

To further demonstrate the effectiveness of the proposed approach, Table 3.3 presents a complete step-by-step illustration of the computation process. The results show that the final output generated by the proposed architecture deviates from the exact multiplication result by only 1.54%. This small deviation indicates that the introduced approximation leads to minimal accuracy degradation, thereby validating the ability of the proposed design to achieve a favourable trade-off between computational accuracy and hardware efficiency. Moreover, the observed error remains well within acceptable limits for practical and error-tolerant applications.

Table 3.3: Detailed Illustration of the Proposed Multiplication Process with elaborative steps of Operand Rounding and Shifting.

| **Steps** | **Proposed multiplication method** |
|---|---|
| **1. Inputs** | |
| $A$ | 65 (Binary: 01000001) |
| $B$ | 120 (Binary: 01111000) |
| **2. Approximate Encoding** | |
| A Range | $64 \leq A < 128$ |
| $A[6:4]$ | 100 |
| $A[3:2]$ | $00 \Rightarrow (\|A[3:2] \& A[3]) = 0$ |
| $A[1:0]$ | $01 \Rightarrow (\|A[1:0] \& A[1]) = 0$ |
| $approx\_A$ | $\{100, 0, 0\} = 5'b10000 = 16$ |
| B Range | $64 \leq B < 128$ |
| $B[6:4]$ | 111 |
| $B[3:2]$ | $10 \Rightarrow (\|B[3:2] \& B[3]) = 1$ |
| $B[1:0]$ | $00 \Rightarrow (\|B[1:0] \& B[1]) = 0$ |
| $approx\_B$ | $\{111, 1, 0\} = 5'b11110 = 30$ |
| **3. Approximate Multiplication** | |
| $approx\_Pro$ | $16 \times 30 = 480$ |
| **4. Shift Amount Calculation** | |
| $A > 127$ | False (0) |
| $A > 63$, $A > 31$ | True (1 each) |
| $B > 127$ | False (0) |
| $B > 63$, $B > 31$ | True (1 each) |
| $shift\_amount$ | $0 + 1 + 1 + 0 + 1 + 1 = 4$ |
| **5. Final Output** | |
| $P$ | $480 \ll 4 = 7680$ |
| Exact $A \times B$ | $65 \times 120 = 7800$ |
| Absolute Error | $7800 - 7680 = 120$ |
| Relative Error | $\frac{120}{7800} \times 100 \approx 1.54\%$ |

## 3.2 Performance Evaluation and Validation of Software and Hardware Implementations

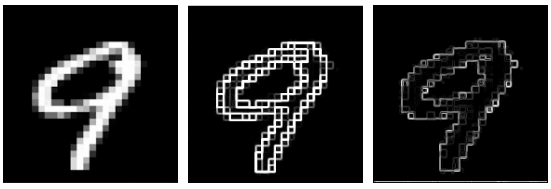
This section presents the performance evaluation and validation of the proposed approach through both software and hardware implementations. The hardware implementation is carried out on a Virtex-7 FPGA SoC, with detailed analysis of key physical metrics such as resource utilization, critical path delay, and power consumption.

Furthermore, the design is synthesised at 65 nm Technology Node, and its physical performance parameters are assessed using the Cadence Genus tool. The software-based evaluation is conducted using a Python-based simulation framework, with a primary focus on edge detection applications. A comprehensive analysis of the obtained results and their implications is presented in the following subsections.

### 3.2.1 Software Implementation and Accuracy Validation

The proposed architecture exhibits a low average relative error of 4.84% when compared to an exact multiplier, as evaluated through exhaustive testing of all 65,536 ($256 \times 256$) possible input combinations using a Python-based implementation. To assess its applicability in real-world scenarios, the architecture was further evaluated for edge detection using several test images. The experimental results are presented in Figures 3.2 and 3.3.

As a representative case study, edge detection was performed for license plate recognition. The results shown in Figures 3.2 and 3.3 indicate that, although the edges produced using the accurate multiplier ($ED_{acc}$) appear more pronounced than those obtained with the proposed approximate multiplier ($ED_{app}$), the latter still successfully identifies the digits on the license plate. These findings demonstrate that the proposed approximate multiplier is well-suited for edge detection tasks, highlighting one of its many potential application domains.



(a) Original (b) $ED_{acc}$ (c) $ED_{app}$

Figure 3.2: Edge Detection results for an image from the MNIST dataset using Accurate and the Proposed Approximate Multipliers.


(a) Original (b) $ED_{acc}$ (c) $ED_{app}$

Figure 3.3: Edge Detection results for License Plate Image using Accurate and the Proposed Approximate Multipliers.

### 3.2.2 Hardware Performance Analysis using FPGA and ASIC-based Evaluations

As summarized in Table 3.4, the FPGA implementation results offer a comprehensive comparative evaluation of the proposed resource-efficient rounding-based 8-bit MAC architecture against existing designs. The proposed architecture achieves a significant reduction of 24.4% in LUT utilization and a 37.5% decrease in critical path delay when compared to the conventional architecture reported in [119].

Furthermore, when bench-marked against other state-of-the-art implementations, the proposed design demonstrates notable improvements in timing performance, exhibiting reductions of 22.7% and 17.07% in critical path delay relative to the architectures presented in [28] and [82], respectively. Although a marginal increase in LUT utilization is observed when compared to the latter, this trade-off is offset by the substantial gains in delay reduction. Importantly, among all the architectures considered in this comparison, the proposed design achieves the lowest critical path delay. In addition to improvements in area and timing metrics, the proposed architecture also delivers enhanced energy efficiency, with a 7.8% reduction in power consumption compared to the conventional architecture [119]. These results collectively highlight the

Table 3.4: Comparative analysis of the FPGA Performance Metrics between the State-of-the-Art Works and Proposed 8-bit MAC Architectures.

| **Architecture** | **LUT** | **FF** | **Critical Delay** $(ns)$ | **Dynamic Power**$(mW)$ |
|---|---|---|---|---|
| **Conventional MAC [119]** | 86 | 16 | 3.81 | 6.6 |
| **Booth [28]** | 83 | 61 | 3.08 | – |
| **Acc_App[82]** | 62 | 59 | 2.87 | – |
| **This Work** | 65 | 16 | 2.38 | 6.01 |

Table 3.5: A Detailed Comparison of Hardware Performance Parameters of State-of-the-Art Fixed-Point Multipliers with the Proposed Work.

| **Parameter** | **VIVADO IP** [119] | **TVLSI '15** [85] | **TVLSI '17** [87] | **This work** |
|---|---|---|---|---|
| Technology node ($nm$) | 65 | 45 | 45 | 65 |
| Precision | 8 | 8 | 32 | 8 |
| Area ($um^2$) | 560 | 355 | 13224 | 392 |
| Power $uW$) | 443.8 | 328 | 5370 | 314.4 |

effectiveness of the proposed design in achieving a favorable balance between resource utilization, performance, and power efficiency.

The ASIC synthesis results summarized in Table 3.5 clearly demonstrate the hardware efficiency and competitive performance of the proposed architecture in comparison with state-of-the-art designs. When evaluated against the conventional architecture reported in [119] under the same 65-nm technology node, the proposed design achieves a substantial reduction of 30.8% in area, along with a notable 29.1% decrease in power consumption, highlighting its improved area–power efficiency. In comparison with the design presented in [87], the proposed architecture demonstrates significant advantages. Although the technology node and operand precision differ between the two implementations, a normalized evaluation indicates that the proposed design achieves approximately a 33× reduction in area and a 17× reduction in power consumption relative to the reference design. These results underscore the effectiveness of the proposed architectural choices in achieving compact and energy-efficient hardware. Furthermore, when compared with the architecture reported in [85], while the latter exhibits a marginally smaller area footprint, it incurs higher power consumption. This contrast highlights the superior power efficiency of the proposed design without a substantial compromise in area. In addition, the proposed architecture achieves lower delay and reduced resource utilization with only a minor degradation in computational accuracy, thereby offering a favorable trade-off between performance, energy efficiency, and precision.

Overall, the combination of reduced area, lower power and delay, and minimal accuracy loss positions the proposed architecture as a strong candidate for error-tolerant

AI and DSP applications. Moreover, the scalability and efficiency demonstrated by the design provide a promising foundation for future exploration across different operand precisions and application domains.

## 3.3 Summary

This chapter presents a resource-efficient approximate multiplier based on operand rounding and shifting, achieving reduced hardware complexity with acceptable accuracy. The architecture enables enhanced parallelism in MAC operations and achieves superior performance compared to state-of-the-art designs, exhibiting the lowest critical path delay and comparable or improved LUT utilization. FPGA implementation on the Virtex-7 VC707 platform demonstrates reductions of 37.5% in delay and 24.4% in LUT usage relative to a conventional 8-bit MAC. The observed average relative error of 4.84% confirms the suitability of the design for error-resilient computing. ASIC synthesis at the 65-nm technology node further validates improvements in area, power, delay, and throughput. The successful application of the proposed architecture to license plate edge detection highlights its effectiveness in image processing and error-tolerant AI applications.

# Chapter 4

# Segmentation-based Approximate Multiplier Architecture

In chapter 3, the design of an approximate 8-bit multiplier based on the operand-reduction technique was presented. This approach enabled significant optimisation of hardware resources while achieving a well-balanced trade-off between computational accuracy and efficiency. By incorporating an operand rounding technique in conjunction with appropriate bit shifting, the proposed multiplier exhibited only minimal degradation in accuracy, remaining well within acceptable limits for error-resilient artificial intelligence applications. Building upon these results, this chapter extends the proposed methodology to higher bit-width arithmetic. Specifically, it analyses the design characteristics and performance of several 16-bit multiplier variants, developed by adapting the 8-bit approximate multiplier introduced in chapter 3 with targeted modifications to accommodate increased precision requirements.

## 4.1 Introduction

This chapter focuses on the resource optimization of 16-bit multipliers, as most contemporary FPGA architectures incorporate DSP blocks that natively support 16-bit multiplication. This bit width offers a practical balance between hardware complexity and the numerical precision required by Digital Signal Processing (DSP) applications such as Digital Filters, Fast Fourier transforms (FFTs), and other signal-processing

tasks [85, 120]. The proposed architecture performs multiplication by segmenting the input operands and computing partial products using carefully selected combinations of approximate and exact multipliers. This strategy enables efficient computation with minimal accuracy degradation, making it well suited for resource-constrained Edge-AI applications.

In chapter 3, an operand-rounding-based 8-bit approximate multiplier was introduced. In this chapter, the same multiplier is reused as a fundamental building block, with specific modifications to facilitate its integration into a segmented multiplication framework. Building upon this foundation, operand segmentation is explored as an alternative approximation strategy for resource-efficient implementation.

## 4.2 Background and Related Works

Since multipliers are a key component of MAC units, several methodologies have been explored to optimize multiplier architectures, maximizing resource efficiency through distinct hardware-level adjustments [47, 49, 55]. For instance, approximate multipliers that leverage lookup table (LUT) structures and fast-carry chains have been actively deployed to minimize both propagation delays and logic overhead [82]. At the functional level, reducing partial products serves as an effective approach for balancing speed, power, and accuracy during dense multiplication workloads [28, 45]. This strategy is exemplified by techniques such as modified Booth encoding, which have been successfully integrated into approximate Wallace-Booth multipliers to alleviate critical path delays [83]. Complementary to these architectural modifications, Boolean rewriting is widely exploited to simplify internal gate counts by modifying fundamental logic equations and Karnaugh maps (K-maps). This algebraic approach achieves significant area and power savings while introducing only minor, tolerable degradation in overall computational accuracy [117, 118].

Segmented multipliers operate by extracting a set of $m$ contiguous bits (referred to as an $m$-bit segment) from each of the two $n$-bit input operands, where $m < n$. These segments are multiplied using an $m \times m$ multiplier, and the resulting $2m$-bit

partial product is subsequently expanded to produce a $2n$-bit output.

In dynamic segmentation approaches, segments are selected starting from the leading one bit of the input operands, enabling variable precision through techniques such as approximate $m \times m$ multiplication, truncation, or operand rounding [88, 87, 86]. However, this dynamic execution introduces significant architectural complexity and real-time control overhead, rendering such techniques inherently resource-intensive. In contrast, Static Segmented Multipliers (SSMs) partition input operands into fixed $m$-bit segments at design time. By eliminating runtime tracking and complex multiplexing logic, static segmentation substantially reduces hardware complexity and power consumption while simplifying the overall implementation [85]. Because SSM-based architectures inherently achieve a highly favorable trade-off between computational accuracy and hardware resource utilization, they represent an attractive framework for efficient multiplier design. Building upon these advantages, the primary novelty of the proposed design lies in a specific static-segmentation-based multiplier architecture engineered for resource-efficient edge computation. By eliminating the necessity for runtime operand tracking entirely, this work delivers an optimized, low-power multiplier topology that maximizes hardware efficiency. Consequently, it maintains a strictly bounded, deterministic accuracy-resource trade-off, offering a highly viable solution for energy-constrained edge applications.

The key contributions of this chapter are summarised as follows:

- **Accuracy-Configurable Segmentation-Based Approximate Multiplier Design (ACSAM):** A novel segmentation-based and resource-efficient 16-bit multiplier architecture that combines exact multipliers with the proposed 8-bit approximate multipliers to optimise hardware utilisation.

- **Integration of Approximate Multipliers in ACSAM:** Seamless integration of a novel operand-rounding-based 8-bit approximate multiplier within the segmented architecture to enable efficient 16-bit multiplication with reduced resource consumption.

- **Hardware Implementation and Evaluation:** Comprehensive hardware im-

plementation and performance evaluation of the proposed architecture on both ASIC and FPGA platforms, demonstrating its adaptability across different design technologies.

- **Application-Level Validation:** Validation of the proposed multiplier architecture using a Gaussian image blurring application, highlighting the effectiveness of ACSAM for error-resilient Edge-AI applications.

## 4.3 Proposed Segmentation-based Approximate Multiplier: ACSAM

The proposed 16-bit multiplier architecture, referred to as ACSAM, employs static operand segmentation to achieve improved resource efficiency without significantly compromising computational accuracy. In this approach, each 16-bit operand (for instance, $A$ and $B$) is divided into two 8-bit segments, namely the higher-order bits ($A_H$, $B_H$) and the lower-order bits ($A_L$, $B_L$). This segmentation facilitates the decomposition of the multiplication operation into four independent partial products: $P_{HH} = A_H \times B_H$, $P_{HL} = A_H \times B_L$, $P_{LH} = A_L \times B_H$, and $P_{LL} = A_L \times B_L$. These partial products are subsequently combined with appropriate bit shifting and accumulation to obtain the final 16-bit product. The ACSAM architecture leverages different combinations of exact and approximate 8-bit multipliers to compute these partial products. The primary motivation behind this segmentation-based strategy is to reduce hardware complexity, power consumption, and area overhead by selectively approximating less significant computations, while maintaining higher accuracy for the more critical higher-order terms. Such selective approximation enables an effective trade-off between accuracy and resource utilisation.

To support varying accuracy requirements across applications, four distinct variants of the segmentation-based approximate multiplier are proposed, rendering the architecture accuracy-reconfigurable. As illustrated in Figure 4.1, the 16-bit multiplication is implemented using four 8-bit multiplier blocks, denoted as M1, M2, M3, and

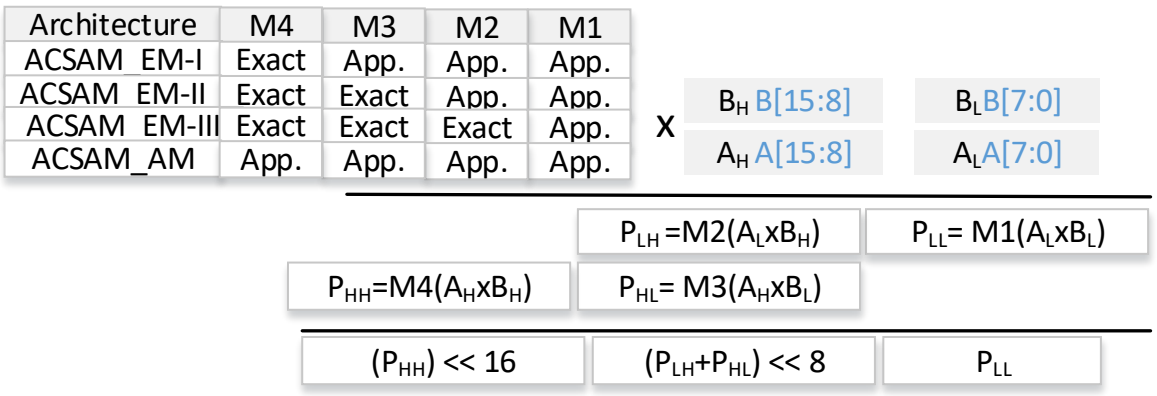



Figure 4.1: Computation of Partial Products by partitioning the input into four segments followed by corresponding multipliers: M1, M2, M3, M4. The architectures for different variants depend on the nature of these multipliers: Exact/Approximate (App.).

M4, which generate the partial products $P_{LL}$, $P_{LH}$, $P_{HL}$, and $P_{HH}$, respectively. By configuring each multiplier block as either exact or approximate, the overall accuracy and hardware cost of the multiplier can be dynamically tailored.

In the **ACSAM_EM-I** configuration, only the most significant partial product, $P_{HH}$, is computed using an exact multiplier, while the remaining partial products are generated using the proposed approximate 8-bit multiplier. The second configuration, **ACSAM_EM-II**, extends exact computation to both M4 and M3, thereby improving accuracy by precisely calculating the most significant and one of the cross-term partial products, while approximating the remaining lower-order terms using M1 and M2. In the **ACSAM_EM-III** variant, all cross-terms along with the most significant partial product are computed using exact multipliers, and only the least significant partial product, $P_{LL}$, is approximated. Finally, the **ACSAM_AM** configuration represents a fully approximate design in which all four partial products are computed using approximate multipliers, offering maximum savings in area and power at the expense of reduced accuracy.

Overall, the proposed ACSAM architecture provides a flexible and scalable design framework that can be tuned according to the error tolerance of the target application. By judiciously selecting the number and position of approximate multiplier blocks, the architecture enables a smooth trade-off between accuracy, performance, and hardware

efficiency, making it well suited for error-resilient AI applications.

### 4.3.1 Approximate Multiplier Design for Computation of Partial Products

The 8-bit rounding-based approximate multiplier, employed in the segmentation-based multiplier, adopts a multi-stage architecture that combines coarse-grain magnitude encoding with segment-wise input operand compression to achieve a low-cost, bias-balanced multiplication scheme, as illustrated in Figure 4.2. This technique builds on the multiplier from previous work [121], with modifications to the operand segmentation strategy to ensure unbiased computations. The previously proposed 8-bit multiplier [121]did not utilise balanced midpoint encoding; instead, it relied on bitwise OR and AND operations, resulting in a complex architecture and a biased output. In this study, each input operand is first partitioned into uniform 2-bit segments, and each segment is processed by a balanced midpoint encoder that indicates whether it lies in the upper or lower half of its local range. This operation forms a behavioural representation of the input while significantly reducing the bit-width required for downstream computation. To preserve fine-grain detail for small operands and to avoid excessive quantisation for larger operands, the number of encoded segments incorporated into the final operand depends on the input's magnitude.

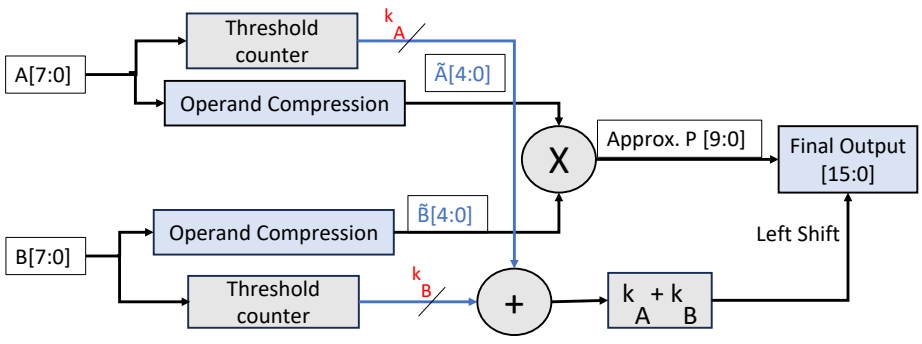



Figure 4.2: Block diagram of the Proposed Approximate 8-bit Multiplier. Here, $\tilde{A}$ and $\tilde{B}$ are the approximated 5-bit inputs after operand compression and $k_A$ and $k_B$ are the corresponding shifts required by operands A and B respectively after multiplication. The approximate product is shifted by the sum of $k_A$ and $k_B$ to get the final output.

Consequently, each operand is converted to a 5-bit approximate operand, maintaining high linearity in the low-value region while allowing for controlled compression of higher values. In the next stage, the architecture computes a lightweight estimate of the shift required by each operand by counting the number of fixed-magnitude thresholds crossed (i.e., $A > 31$, $A > 63$, $A > 127$) using a threshold counter.

Table 4.1: Operand-rounding based Approximate Multiplication Procedure.

| **Step** | **Description** |
|---|---|
| **1. Inputs** | |
| Operands | $A, B \in [0, 255]$ (8-bit unsigned) |
| **2. Midpoint Function** | |
| Definition | $\mathrm{Mid}(x) = \begin{cases} 1, & x \geq 2 \\ 0, & x < 2 \end{cases}$ |
| **3. Operand Compression** | |
| Range 0 | $0 \leq X \leq 31$ |
| | $\tilde{X} = X_{[4:0]}, \quad k_X = 0$ |
| Range 1 | $32 \leq X \leq 63$ |
| | $\tilde{X} = \{X_{[5:2]}, \mathrm{Mid}(X_{[1:0]})\}, \quad k_X = 1$ |
| Range 2 | $64 \leq X \leq 127$ |
| | $\tilde{X} = \{X_{[6:4]}, \mathrm{Mid}(X_{[3:2]}), \mathrm{Mid}(X_{[1:0]})\}, \quad k_X = 2$ |
| Range 3 | $128 \leq X \leq 255$ |
| | $\tilde{X} = \{X_{[7:6]}, \mathrm{Mid}(X_{[5:4]}), \mathrm{Mid}(X_{[3:2]}), \mathrm{Mid}(X_{[1:0]})\}, \quad k_X = 3$ |
| **4. Reduced-Precision Multiplication** | |
| Operation | $\mathrm{approx}_P = \tilde{A} \times \tilde{B}$ |
| **5. Shift Calculation** | |
| Shift amount | $\mathrm{shift} = k_A + k_B$ |
| **6. Output Reconstruction** | |
| Final output | $P = \mathrm{approx}_P \ll \mathrm{shift}$ |

*Note: During physical hardware synthesis, the underlying 2-bit midpoint function $\{x_1, x_0\} \geq 2$ used for structural algorithmic clarity mathematically reduces to a single-bit check of the most significant bit ($x_1$).*

Each satisfied threshold contributes one unit to the shift count, resulting in a value within the range of 0–3 per operand. The shifts are denoted by $k_A$ and $k_B$, respectively. These shift values are subsequently added, mimicking the exponent addition step in floating-point multiplication. The compressed operands are multiplied

using a small 5×5 multiplier, and the resulting 10-bit intermediate product is scaled by the combined shift value through a left-shift operation ($k_A + k_B$). This final recombination stage effectively reconstructs a wide dynamic range product using only low-precision arithmetic. The stepwise operation is presented in Table 4.1. Overall, this unique architecture achieves a favourable balance between hardware efficiency and computational accuracy, while maintaining performance comparable to state-of-the-art approximate multipliers.

## 4.4 Performance Evaluation and Validation on Software and Hardware Platforms

This section presents the performance evaluation and validation conducted on both software-based and hardware-based implementations. Hardware implementation utilized the Virtex-7 FPGA Board, with critical analysis focusing on physical parameters like resource utilization, critical delay and power. The physical performance parameters of the design have been evaluated at 65nm technology node. A detailed analysis of the proposed segmentation-based multiplier is provided in the subsections below.

### 4.4.1 FPGA and ASIC-based Evaluation of Approximate Rounding-based Multiplier

As shown in Table 4.2, the FPGA implementation results provide a detailed comparative analysis of the proposed resource-efficient, rounding-based 8-bit approximate multiplier integrated into a MAC architecture. The results show that the proposed design achieves a 26.74% reduction in LUT utilization and a 37.53% decrease in critical path delay compared to the conventional MAC implementation [119], while also reducing dynamic power consumption by 8.94%. Furthermore, compared with [28], the proposed architecture demonstrates a 24.10% reduction in LUT footprint and a 22.73% faster critical path delay.

In comparison with [82], a 17.07% savings in critical path delay is achieved at

Table 4.2: FPGA Performance Comparison for various 8-bit MAC Architectures

| **Architecture** | **LUT** | **FF** | **Critical Path Delay** $(ns)$ | **Total Dynamic Power** $(mW)$ |
|---|---|---|---|---|
| Conventional [119] | 86 | 16 | 3.81 | 6.6 |
| Farrukh et al. '20 [28] | 83 | 61 | 3.08 | – |
| Ullah et al. '21[82] | 62 | 59 | 2.87 | – |
| Mrazek et al. '17[47] (mul8u_1DMU) | 66 | - | 3.85 | 6.3 |
| Mrazek et al. '17[47] (mul8u_12XY) | 78 | - | 3.86 | 6.5 |
| Guo et al. '24[122] (MODA_1144) | 51 | - | 4.11 | 6.1 |
| **This work** | **63** | **16** | **2.38** | **6.01** |

the expense of only a single additional LUT. Notably, the proposed design also significantly outperforms the Pareto-optimal designs by [**?**], achieving up to a 19.23% LUT reduction and a 38.34% delay improvement over the *mul8u_12XY* variant. Even when compared against the low-power architecture of [122], our work achieves a 1.48% drop in total dynamic power and speeds up execution by 1.73× (a 42.09% reduction in delay), despite a slight increase in LUT logic overhead. Overall, the proposed architecture demonstrates the lowest propagation delay and dynamic power profile among the evaluated state-of-the-art designs, validating its exceptional efficiency for high-speed, energy-constrained arithmetic operations.

As shown by the ASIC synthesis results, summarized in Table 4.3, scaling all designs to the 65 nm node highlights the significant hardware advantages of the proposed multiplier over state-of-the-art architectures. Specifically, the proposed multiplier achieves 12.14% area reduction, 3.37% power reduction, and 10.71% delay improvement compared to the conventional design [119]. When measured against normalized state-of-the-art works, the hardware gains are even more pronounced: it yields a 1.57× and 1.78× reduction in area compared to the architectures by [89] and [123], respectively. In terms of power, the proposed work achieves a 1.68× reduction over [124] and a massive 3.98× reduction over [123]. Moreover, the proposed design operates with clock delays 1.88× and 2.44× faster than those of [124] and [123], respectively. These balanced physical optimisations directly translate into system-level performance, yielding up to a 1.87× throughput improvement (vs [124]) and a 5.69× increase in energy efficiency (vs [89]), thereby confirming its suitability for resource-constrained and high-performance AI/DSP applications.

Table 4.3: Comparison of Hardware Performance Parameters of the Proposed 8-bit Approximate Multipliers with State-of-the-Art Multipliers

| **Parameter** | **Conv.** [119] | **Strollo et al. '22** (SSM, m=6)[89] | **Nima A. et al. '22** (CMD_44) [124] | **Waris et al. '20** [123] | **Rathod et al.'25** [125] | **This work** |
|---|---|---|---|---|---|---|
| **Technology node** ($nm$) | 65 | 28 | 45 | 45 | 45 | **65** |
| **Arithmetic** | FxP | FxP | FxP | FxP | FxP | **FxP** |
| **Area** ($um^2$) | 560 | 143.5(*773.3) | 257.7(*537.7) | 419.2(*874.64) | 289 (*603) | **492** |
| **Power** ($uW$) | 68.2 | 145(*336.6) | 76.78(*110.9) | 181.65(*262.39) | 32.02 (*46.25) | **65.9** |
| **Delay(ns)** | 0.56 | 0.24 (*0.56) | 0.65(*0.94) | 0.84 (*1.22) | 0.43(*0.62) | **0.50** |
| **Throughput** (GOPS) | 1.79 | 4.17 (*1.80) | 1.54 (*1.07) | 1.19(*0.82) | 2.33(1.61*) | **2.00** |
| **Energy Efficiency** (TOPS/W) | 26.18 | 28.74 (*5.33) | 20.03 (*9.59) | 6.56(*3.14) | 72.60(*34.77) | **30.35** |

* Values in parentheses denote first-order normalization to 65 nm using fixed-supply voltage scaling.

### 4.4.2 Hardware and Software Evaluation of ACSAM

The FPGA-based synthesis results for the various 16-bit approximate multipliers are summarized in Table 4.4. As expected, scaling the performance metrics shows that increasing the number of exact 8-bit multiplier segments within the ACSAM framework results in a progressive increase in Lookup Table (LUT) consumption. Specifically, the fully approximate variant, ACSAM_AM, achieves a substantial 32.41% reduction in LUT utilization and a 12.12% savings in total dynamic power relative to the conventional exact 16-bit design [119]. Similarly, the error-masked ACSAM_EM-I variant reduces LUT utilization by 24.69% compared to the conventional baseline.

When evaluated against contemporary state-of-the-art approximate multipliers, the proposed designs showcase superior multi-objective trade-offs. Compared to [82], ACSAM_EM-I delivers a 1.56× improvement in critical path delay at the expense of a minor 1.09× increase in LUT logic overhead. Furthermore, when compared to [120], the ACSAM_EM-I and ACSAM_AM variants lower LUT requirements by 9.63% and 18.89%, while simultaneously speeding up operations by 1.44× and 1.48×, respectively. Notably, the proposed configurations also compete favourably against the architectures by [126]; for instance, ACSAM_EM-III improves the critical path delay by 1.12× over their KT-MID variant while keeping a similar logic footprint, reinforcing its high suitability for high-speed hardware environments.

The ASIC-based evaluation results, summarized in Table 4.5, demonstrate a sub-

Table 4.4: FPGA Performance Comparison of various 16-bit Approximate Multipliers

| **Architecture** | **LUT** | **FF** | **Critical Path Delay** $(ns)$ | **Total Dynamic Power** $(mW)$ |
|---|---|---|---|---|
| Conventional [119] | 324 | 96 | 4.72 | 33 |
| Ullah et al.'21 [82] | 224 | — | 4.30 | — |
| Nagar et al.'24[120] | 270 | — | 3.97 | — |
| Zhang et al.'21[126](KT-MID) | 297 | — | 2.9 | 23 |
| Zhang et al.'21[126] (KT-LSB) | 287 | — | 3.1 | 21 |
| **ACSAM_EM- I** | **244** | **116** | **2.76** | **22** |
| **ACSAM_EM- II** | **264** | **111** | **2.66** | **30** |
| **ACSAM_EM- III** | **285** | **106** | **2.58** | **30** |
| **ACSAM_AM** | **219** | **116** | **2.69** | **29** |

stantial reduction in both silicon area and power consumption when compared to existing state-of-the-art designs. For a fair comparison, all competing architectures are evaluated against their first-order normalized values at the 65 nm technology node. In direct comparison with [100], which natively operates at 65 nm, the proposed ACSAM_AM architecture achieves a 12.28% reduction in area and a 2.85× decrease in power consumption. Similarly, the ACSAM_EM-I variant exhibits a 7.97% reduction in area while delivering a 2.79× reduction in power over the same reference.When evaluated against scaled state-of-the-art architectures, the hardware advantages become even more pronounced. For instance, compared to the normalized area of [124], ACSAM_AM yields a 1.43× area reduction, while offering a 3.90× power reduction over its normalized counterpart.

Furthermore, compared to ultra-scaled technologies such as [127] normalized to 65 nm, the ACSAM_AM architecture achieves a massive 2.57× area reduction and a 6.02× power reduction. Comparing the normalised values, ACSAM architectures demonstrate competitive throughput, with ACSAM_EM-III achieving 1.06× and 1.45× higher throughput than [124] and [128], respectively. Moreover, ACSAM_EM-III attains 3.99 TOPS/W, providing close to 33% improvement over the conventional multiplier (2.99 TOPS/W) and nearly 3.8× higher efficiency than designs proposed in [124] (∼1.0 TOPS/W at 65nm). Overall, ACSAM improves throughput and energy efficiency, making it suitable for energy-constrained applications.

In addition to hardware efficiency, the error characteristics of the proposed designs are evaluated and reported in Table 4.6. The results indicate a marked improvement in

Table 4.5: Comparison of Hardware Performance Parameters of the 16-bit State-of-the-Art Works with the ACSAM variants

| Design | Tech. Node (nm) | Area ($\mu m^2$) | Power (mW) | Delay (ns) | Throughput (GOPS) | Energy Eff. (TOPS/W) |
|---|---|---|---|---|---|---|
| Nima A. et al '23 (CMD16_3000)[124] | 45 | 1299 (*2708.3) | 0.473 (*0.683) | 1.02 (*1.473) | 0.980 (*0.68) | 2.07(*0.99) |
| Nima A. et al. '23 (CMD16_0000)[124] | 45 | 1318 (*2747.9) | 0.479 (*0.692) | 1.03(*1.488) | 0.971 (*0.67) | 2.03 (*0.97) |
| Kumar et al. '22 [128] | 180 | 3947(*514.7) | 0.17 (*0.061) | 5.51 (*1.990) | 0.181(*0.50) | 1.07 (*8.20) |
| Suganthi et al. '17 [100] | 65 | 2158 | 0.500 | 0.470 | 2.128 | 4.26 |
| Zacharelos et al. '22 [127] | 14 | 226 (*4870.4) | 0.227 (*1.054) | 0.35 (*1.625) | 2.857 (*0.62) | 12.59 (*0.58) |
| Zervakis et al. '16 [44] | 65 | 2871 | 0.435 | 0.400 | 2.500 | 5.75 |
| Conventional [119] | 65 | 2767 | 0.263 | 1.270 | 0.787 | 2.99 |
| **Proposed Work: ACSAM** | | | | | | |
| ACSAM_EM-I | 65 | 1986 | 0.179 | 1.410 | 0.709 | 3.55 |
| ACSAM_EM-II | 65 | 1989 | 0.181 | 1.390 | 0.719 | 3.78 |
| ACSAM_EM-III | 65 | 1992 | 0.182 | 1.380 | 0.725 | 3.99 |
| ACSAM_AM | 65 | 1893 | 0.175 | 1.440 | 0.694 | 3.97 |

(*) Values in parentheses denote first-order normalization to 65 nm using fixed-supply voltage scaling, where Area scales with $(65/L_x)^2$, Power and Delay scale with $(65/L_x)$, Throughput scales as $(L_x/65)$, and Energy Efficiency scales with $(L_x/65)^2$.

Table 4.6: Error Metrics comparison of various 16-bit Approximate Multipliers

| Architecture | NMED | MRED |
|---|---|---|
| Suganthi et al.'17[100] | $1.78 \times 10^{-2}$ | $7.63 \times 10^{-2}$ |
| Nima et al.'23[124] | $1.91 \times 10^{-3}$ | $9.8 \times 10^{-3}$ |
| Kumar et al.'22[128] | $4.0 \times 10^{-6}$ | $1.2 \times 10^{-4}$ |
| Zervakis et al.'16[44] | $6.3 \times 10^{-6}$ | $2.3 \times 10^{-3}$ |
| ACSAM_EM-I | $1.94 \times 10^{-3}$ | $2.84 \times 10^{-2}$ |
| ACSAM_EM-II | $5.64 \times 10^{-4}$ | $1.42 \times 10^{-2}$ |
| ACSAM_EM-III | $2.20 \times 10^{-7}$ | $1.1 \times 10^{-5}$ |
| ACSAM_AM | $1.45 \times 10^{-1}$ | $0.48 \times 10^{-1}$ |

error performance over state-of-the-art approaches, with the ACSAM_EM-III variant achieving the lowest Normalized Mean Error Distance (NMED) of $2.20 \times 10^{-7}$ and Mean Relative Error Distance (MRED) of $1.1 \times 10^{-5}$ among all evaluated designs. This represents a significant leap in accuracy, outperforming [128] by over an order of magnitude in NMED. Compared to [100], ACSAM_EM-I achieves a 2.69× reduction in MRED, while ACSAM_EM-II attains an MRED ($1.42 \times 10^{-2}$) that bridges the gap with [124] while offering significantly lower hardware overhead.

Depending on the target application's error-tolerance requirements, the most suitable variant can be dynamically selected, making these architectures highly competitive across both hardware efficiency and computational accuracy.

### 4.4.3 Gaussian Image Blurring using the Proposed Multiplier

To evaluate the effectiveness of the proposed ACSAM multiplier, a Gaussian blurring filter is implemented as a benchmark application on a 16-bit grayscale image. The

filtering operation performs spatial convolution using a $3 \times 3$ integer-based Gaussian kernel. This convolution process employs a custom 16-bit tiled multiplier architecture, in which the input operands are decomposed into four $8 \times 8$ sub-blocks. All four variants of the proposed multiplier are implemented and rigorously validated within the proposed experimental framework to assess their practical effectiveness. The corresponding peak signal-to-noise ratio (PSNR) values, presented in Fig. 4.3, provide a quantitative evaluation of the reconstructed image quality. These results demonstrate that the proposed designs are capable of preserving high reconstruction accuracy while simultaneously achieving improved computational efficiency, thereby confirming their suitability for error-tolerant and performance-critical image processing applications.

(a) Original image | (b) Blur–Exact | (c) Blur–ACSAM_EM3 PSNR- 33.16 dB | (d) Blur–ACSAM_EM2 PSNR- 33.14 dB | (e) Blur–ACSAM_EM1 PSNR- 28.00 dB | (f) Blur–ACSAM_AM PSNR- 27.72 dB

Figure 4.3: Performance Comparison of the variants of the Proposed Design on Gaussian Blurring Application with corresponding PSNR values.

## 4.5 Summary

This chapter presents a segmentation-based approximate multiplier design, referred to as ACSAM, which incorporates multiple configurations of exact and proposed approximate 8-bit multipliers to enable computationally efficient arithmetic operations. In comparison with state-of-the-art designs, the ACSAM architecture demonstrates superior hardware efficiency, achieving up to 18.9% improvement in LUT utilization on FPGA platforms and up to 12.27% reduction in silicon area at the 65 nm technology node. In addition to hardware benefits, the proposed design exhibits improved error characteristics relative to existing approaches, as evidenced by favorable error metrics. The effectiveness of ACSAM is further validated through its application to an image processing task, highlighting its ability to balance computational accuracy and

hardware efficiency. The consistent performance of the proposed architecture across both FPGA and ASIC platforms, combined with enhanced error metrics, underscores its suitability for resource-constrained Edge-AI and DSP applications.

# Chapter 5

# Adaptive-Precision SIMD PE Architecture for Enhanced Throughput and Resource-Efficient DNN Acceleration

So far, the emphasis has been placed on the resource-efficient design of arithmetic units, specifically adders and multipliers, through architectural approximation of conventional implementations. These efforts demonstrated substantial improvements in area, power consumption, and error tolerance, addressing key challenges associated with low-complexity arithmetic design. While such optimizations are essential for energy- and resource-constrained systems, they alone do not fully capture the performance requirements of modern DNN accelerators. In particular, throughput emerges as a critical design metric, as it directly influences processing latency and real-time inference capability. Consequently, the following chapter shifts the focus to techniques that enhance throughput in DNN accelerator architectures, complementing the resource-efficient arithmetic designs presented earlier.

## 5.1 Introduction

As DNN models continue to grow in depth and complexity to achieve higher inference accuracy, the demand for efficient and scalable hardware implementations becomes increasingly critical. Conventional fixed-precision hardware designs often

struggle to achieve an optimal trade-off among computational performance, hardware resource utilization, and energy efficiency, particularly in resource-constrained environments. This limitation has motivated the exploration of adaptive-precision architectures that dynamically adjust computational precision in response to application-specific requirements, thereby reducing hardware overhead while maintaining acceptable inference accuracy.

Recent studies have demonstrated the effectiveness of dynamic fraction scaling and multi-precision computing paradigms in supporting 4-, 8-, 16-, and 32-bit arithmetic operations [129, 130]. These approaches enable efficient MAC operations by allowing resource sharing across multiple precision modes, thereby improving hardware utilization and flexibility. Furthermore, advances in adaptive hardware architectures underscore the importance of innovative arithmetic design techniques to meet the escalating computational demands of modern DNN accelerators. Although IEEE-754 floating-point representations, such as 16-bit, 32-bit, and bfloat16 formats [131], offer high numerical accuracy, they impose considerable area, power, and latency overheads. In contrast, integer and fixed-point arithmetic formats provide more resource-efficient alternatives, particularly suited for edge and embedded AI systems.

Building on these observations, there is a growing need for hardware architectures that simultaneously support adaptive precision and high throughput. However, existing solutions often exhibit limited flexibility, either adhering to fixed-precision designs or compromising resource efficiency to preserve numerical accuracy. This unresolved trade-off motivates the development of scalable and configurable architectures capable of dynamically adjusting precision while efficiently exploiting available hardware resources.

Modern computing systems increasingly rely on Single Instruction, Multiple Data (SIMD) processing elements to enhance performance through parallel execution, as they allow a single instruction to be applied concurrently across multiple data operands, thereby improving computational efficiency and throughput for performance-critical operations [132]. In this context, the generic RISC-V–based (Reduced Instruction Set Computing, Version 5) systolic array–driven deep neural network

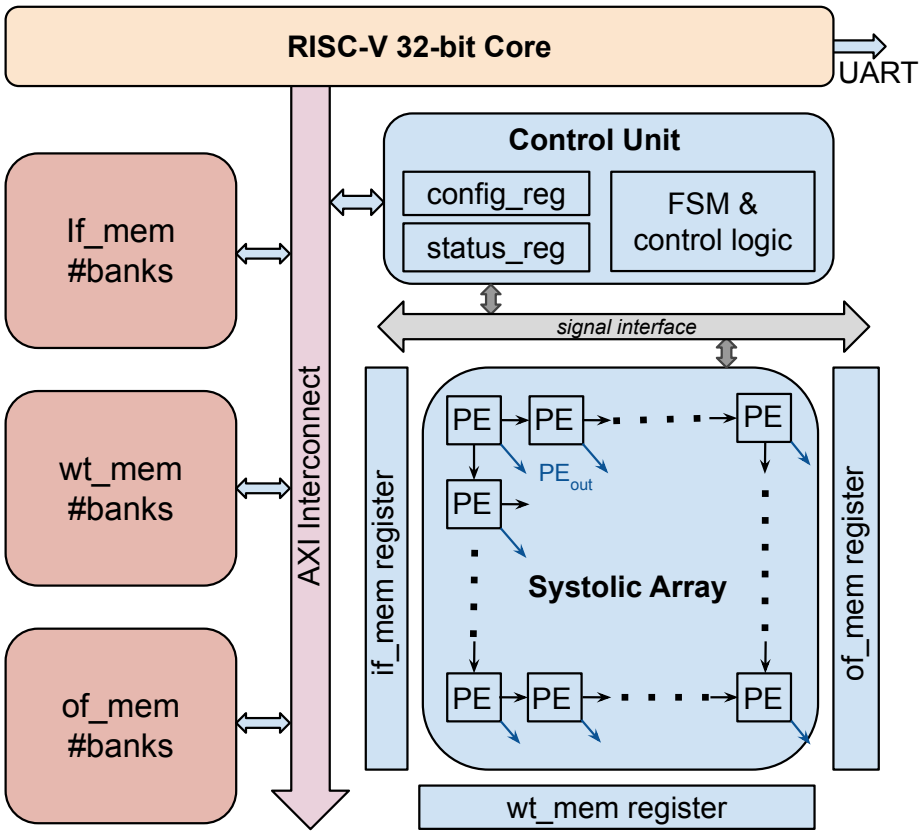



Figure 5.1: Generic RISC-V–driven Systolic Array–based DNN Accelerator Architecture incorporating SIMD Processing Elements (PEs) for Parallel and Efficient Computation.

(DNN) accelerator architecture integrates SIMD processing elements (PEs) to facilitate highly parallel and efficient computation. The overall architectural organization of this accelerator, illustrating the incorporation of SIMD PEs, is depicted in Fig. 5.1.

This architecture forms the foundation of the proposed adaptive-precision design, which aims to enhance throughput and resource utilization while supporting multiple precision modes. By leveraging SIMD-enabled PEs within a systolic array structure, the design achieves high parallelism in MAC operations, making it well-suited for edge-AI applications with stringent performance and power constraints. In this chapter, an adaptive-precision SIMD PE architecture tailored for DNN accelerators is proposed. The design supports multiple precision modes (4-bit, 8-bit, and 16-bit) and asymmetric operations (16×4), enabling efficient MAC computations with resource reuse during partial-product accumulation. This is referred to as SIMD-Low, and the

same architecture is also adaptable for SIMD-High (8/16/32-bit). Unlike conventional approaches, the proposed architecture processes up to 16 operands in low precision, four operands in medium precision, and one operand in high precision, while also supporting asymmetric computations such as 16×4-bit multiplications in parallel.

This adaptability enables fine-grained tuning of precision across network layers, optimizing resource utilization and performance while preserving accuracy. In edge-AI applications, where computational complexity and power constraints coexist, adaptive-precision computation ensures high throughput and energy efficiency [130]. The proposed architecture demonstrates significant improvements in throughput, area efficiency, and energy consumption, making the design highly suitable for resource-constrained edge-AI applications. Key contributions of this study include:

- **Adaptive Precision:** Scalable SIMD PE architecture supporting integer and fixed-point arithmetic with symmetric/asymmetric operations across 4/8/16-bit modes using dynamic resource reuse.
- **High Parallelism:** Full SIMD exploitation for efficient MAC operations under varying precision and operand asymmetry, maximizing throughput.
- **Resource Optimization:** Dynamic adder reuse in accumulation stage for reduced hardware overhead without performance loss.
- **Cross-Platform Validation:** ASIC and FPGA implementation with accuracy analysis on diverse DNN models and physical metrics (area, power, throughput).

## 5.2 Background and Motivation

A typical PE consists of a MAC unit, post-processing blocks, and an AF, as shown in Figure 5.2. Modern DNN accelerators employ large arrays of PEs, which account for over 90% of total power consumption. To meet diverse precision requirements (4/8/16/32-bit) and support dynamic fixed-point computation, scalable PE architectures are essential for improving performance and accuracy. However, scalability

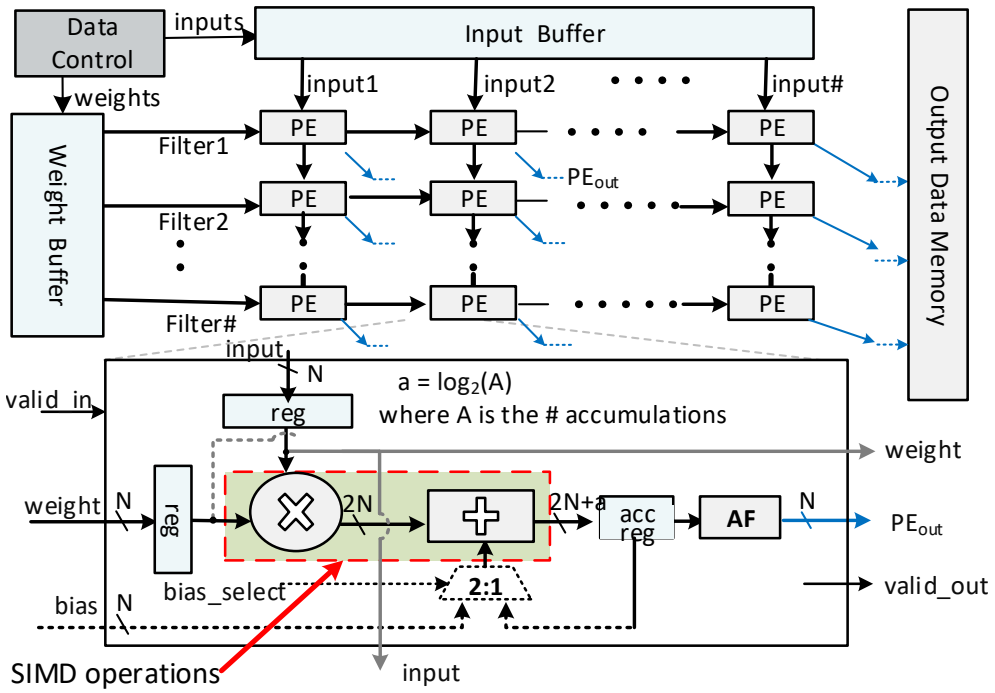



Figure 5.2: Computation Within Processing Elements (PEs) embedded in Systolic Array Architectures for Deep Neural Networks.

introduces challenges, such as increased resource overhead and trade-offs in accuracy. While low-precision computation reduces power and area, it can degrade accuracy. Moreover, different DNN models and layers require varying precision levels, necessitating flexible and adaptive architectures [63].

DNNs possess intrinsic error tolerance, which makes them well-suited for approximation-aware designs in both the multiplication and accumulation stages. While prior work has explored architectural optimizations for accumulator structures [133], efficient utilization of SIMD resources remains an open challenge. Pipelined architectures require a careful balance between throughput and hardware utilization to achieve optimal performance [134].

Recently, precision-scalable MAC architectures have gained significant attention, as optimal numerical precision can vary across models and even within layers of the same model [63, 135]. Although state-of-the-art designs demonstrate hardware-efficient PE implementations, they often do so at the expense of numerical precision and inference accuracy. Current approaches typically support precision levels ranging from 2-bit to

32-bit and investigate both one-dimensional (1D) scalability, which refers to scaling the bitwidth of weights only, and two-dimensional (2D) scalability, which enables simultaneous scaling of both weight and activation bitwidths [65].

Numerous techniques have been proposed to support multi-precision DNN design, especially at the computation level [71]. One prominent approach is the dynamic fusion of bit-flexible multipliers, which enables variable-precision MAC operations. However, this method often results in under-utilization of available MAC units, thereby constraining potential latency reductions. To address this, prior work has explored reducing shift–add computations combined with loop-nest optimizations to improve the performance of fused PEs [71]. Despite these optimizations, increasing operand precision significantly exacerbates accumulation complexity. Alternatively, scalable multiplier architectures based on sub-word parallelism have been proposed to support multi-precision computation [74]. While these designs employ operand gating techniques to reduce power consumption, such gating further contributes to MAC under-utilization. More recent efforts have focused on improving utilization efficiency [64]; however, the range of supported MAC operations remains limited, with only four operations for 8-bit precision and two operations for 16-bit precision.

Floating-point multi-precision MAC architectures face several challenges, including low hardware utilization and accuracy degradation caused by mantissa truncation [75, 76]. Accurate MAC operations are essential for accuracy-sensitive applications, as errors in the most significant bits can severely degrade overall model performance. In addition, power consumption and thermal constraints remain critical concerns, and existing mitigation techniques such as Dynamic Voltage and Frequency Scaling (DVFS) provide only limited benefits [136]. Fixed-point SIMD MAC implementations based on shared segmentation have been proposed to support multiple precision modes (e.g., eight 8-bit, four 16-bit, two 32-bit, or one 64-bit operation), but these designs typically incur high critical-path delay [137].

Moreover, to fully exploit the benefits of low-precision computation, many modern DNNs adopt different precisions for weights and activations, necessitating hardware support for asymmetric arithmetic. Asymmetric computation is particularly impor-

Table 5.1: Comparison of Hardware Performance Parameters of State-of-the-Art Designs and the Proposed Work

| Design | Arithmetic Format | Asymmetric Precision | SIMD Support (#Parallel Ops) | Adder Reuse |
|---|---|---|---|---|
| Huang et al. '22 [62] | INT | ✗ | 2 / 4 / 8 (4 / 2 / 1) | ✓ |
| Shin et al. '17 [74] | FxP | ✓ | 2 / 4 / 8 (1 / – / – ) | ✗ |
| Li et al. '22 [130] | FP | ✗ | 16 / 32 / 64 (16 / 4 / 1 ) | ✗ |
| Li et al. '22 [91] | FxP | ✗ | 2 / 4 / 8 (4 / 2 / 1 ) | ✗ |
| Neda et al. '22 [63] | INT | ✗ | 4 / 8 (8 / 2 ) | ✗ |
| Raut et al. '24 [64] | FxP | ✗ | 8 / 16 ( 4 / 1) | ✗ |
| Judd et al.'16 [66] | FxP | ✓(only 1D) | 1-16:Activation, 16:Weight | ✓ |
| Reggiani et al.'23 [141] | FxP | ✓ | 2-8(7-1) | ✗ |
| Zhao et al.'24 [142] | INT | ✗ | blue2/4/8/16(32/16/8/4) | ✗ |
| Sharma et al.'18 [70] | FxP | ✓ | 2/4/8/16(16/4/1/1) | ✗ |
| Lee et al.'18 [68] | FxP | ✗ | 1-16(16/8/4/1) | ✗ |
| Proposed: SIMD-Low | FxP | ✓ | 4 / 8 / 16 (16 / 4 / 1) | ✓ |
| Proposed: SIMD-High | FxP | ✓ | 8 / 16 / 32 (16 / 4 / 1) | ✓ |

tant because it reduces memory footprint by enabling efficient packing of low-precision operands (e.g., DNN weights) while preserving higher precision for other operands (e.g., DNN activations) [138]. Consequently, supporting asymmetric MAC operations improves both flexibility and energy efficiency, enhancing adaptability across a wide range of applications [139, 71, 140].

A comparative analysis of state-of-the-art multi-precision MAC architectures is summarized in Table 5.1. Unlike prior multi-bit-width accelerator designs such as Huang et al. [62] and Li et al. [91], which rely on bit-split systolic arrays and support only symmetric precision modes, the proposed architecture enables both symmetric and asymmetric precision while achieving higher parallelism through unified datapath reuse. Although the architecture in [66] supports asymmetric precision via adder-tree reuse, it is only one-dimensionally scalable: arbitrary precision is supported for activations, whereas weights are fixed at 16-bit, limiting overall flexibility.

While the designs in [141, 70] demonstrate energy efficiency improvements through asymmetric precision support, they do not reuse adder trees, resulting in increased resource overhead. Conversely, the architectures proposed in [142, 68] lack support for asymmetric precision, restricting their applicability to DNNs that require heteroge-

neous precisions for weights and activations.

Given the growing importance of mixed-precision inference, architectural support for asymmetric precision is essential. Furthermore, increasing operational parallelism is critical for improving throughput, while effective datapath reuse is necessary for maximizing computational efficiency. Therefore, MAC designs that jointly support asymmetric precision, high parallelism, and efficient hardware reuse are key to enabling efficient SIMD processing elements.

This chapter addresses these challenges by proposing multi-precision optimised MAC computations with symmetric and asymmetric arithmetic support for efficient DNN inference. The design enables parallel processing across all PEs, improving throughput, reducing latency, and ensuring high SIMD utilization while mitigating power and thermal issues.

## 5.3 Scalable Multi-Precision SIMD PE Design for DNN Accelerators

Multi-precision computation, encompassing 4-, 8-, 16-, and 32-bit numeric formats, has become a crucial in modern DNN accelerators to effectively balance model accuracy and computational efficiency. Such flexibility enables fine-grained precision adaptation across different network layers, allowing hardware resources to be utilized more efficiently while maximizing overall performance. At the core of DNN acceleration lies the Processing Element (PE), which executes MAC operations followed by activation functions (AFs). The proposed multi-precision signed-magnitude PE architecture, illustrated in Figure 5.3, integrates multiple lower-precision multipliers to support a wide range of precision configurations.

This design enables seamless execution of both low- and high-precision computations within a unified datapath, eliminating the need for separate specialized hardware. To ensure correct and synchronized operation within a systolic array, the PE incorporates a set of control signals, including `valid_in`, `mode`, `clk`, `accum`, `clear`,

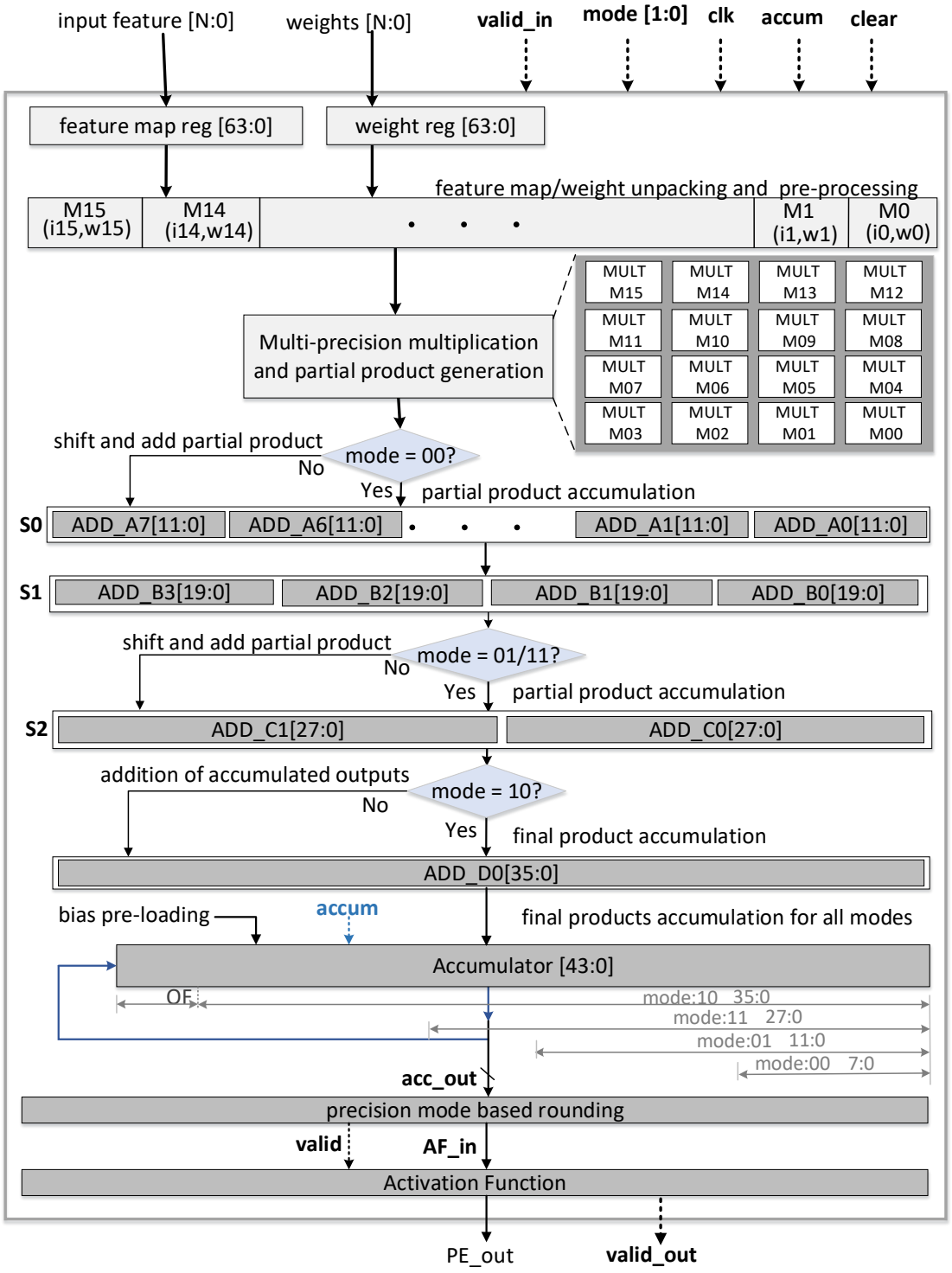



Figure 5.3: Adaptive SIMD PE Architecture handling Multiple Operand Counts and Precision Modes.

and `valid_out`. These signals coordinate data flow, precision mode selection, accumulation control, and pipeline synchronization, thereby maintaining computational

correctness and efficient throughput across the array.

The proposed design introduces two scalable SIMD configurations: **SIMD-Low** (supporting 4/8/16-bit) and **SIMD-High** (supporting 8/16/32-bit). The architecture is inherently simple and scalable, achieved by adjusting the bit-width of the logic design without structural complexity. It is worth noting that by changing the bitwidths of the unit multiplier and its corresponding adders, the architecture can be adapted for both low precision as low as 2 bits and high precision up to 64 bits and beyond. The detailed micro-architectural features and SIMD functionality are discussed in the following subsections.

### 5.3.1 Efficient Memory Addressing for Multi-Precision Weights and Biases in SIMD

The proposed architecture incorporates an efficient memory addressing scheme specifically designed for multi-precision DNN workloads, enabling high utilization and scalable operation across different precision modes. The system employs three flip-flop–based memory banks: the input feature map memory (`if_mem`), the weight memory (`wt_mem`), and the output feature map memory. Each memory is partitioned into four sub-banks with a width of 32 bits, as illustrated in Fig. 5.1.

Memory accesses are precision-aware to maximize bandwidth efficiency. In the 4-bit precision mode, all four sub-banks are accessed concurrently to retrieve a total of 64 bits, corresponding to sixteen 4-bit operands. For 8-bit precision, two sub-banks are accessed in parallel to supply 32 bits, yielding four operands. In the 16-bit precision mode, a single sub-bank is accessed to deliver one operand.

This access strategy is applied uniformly across all memory banks for both input reads and output writes, ensuring consistent and efficient data movement while eliminating idle cycles. Moreover, the proposed scheme inherently avoids bank conflicts during simultaneous multi-precision accesses and minimizes stall cycles by exploiting architectural parallelism rather than relying on decoder-based arbitration mechanisms.

Although the proposed architecture does not adopt a unified (single-bank) memory

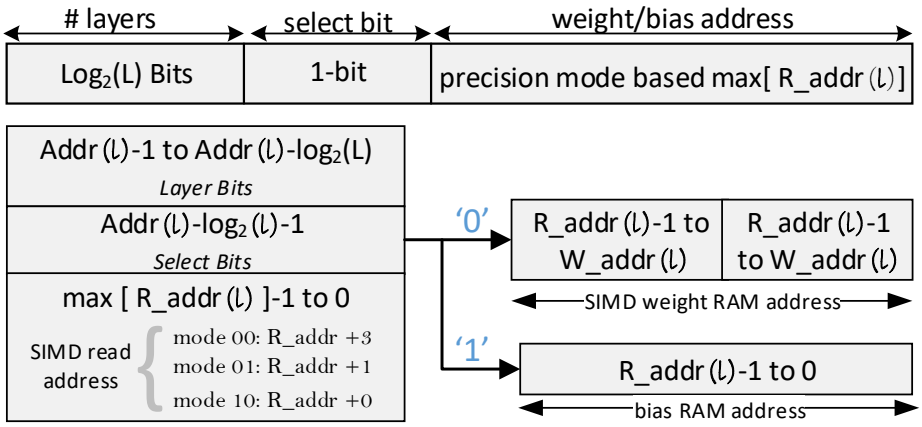



Figure 5.4: Proposed Address Mapping for DNN parameters using layer ID, Parameter Type, and Precision-Aware Indexing.

organization for storing both weights and inputs, such designs are commonly used in software-controlled and highly adaptable DNN systems [143]. In these architectures, an efficient addressing scheme is critical to avoid redundant read and write operations to inactive memory locations during SIMD execution.

For single-bank memory architectures, a scalable address-mapping strategy is typically employed to support multi-precision parameters, including 4-, 8-, and 16-bit weights and biases. This approach commonly utilizes First-In, First-Out (FIFO) buffers for temporary storage and DRAM for runtime data, enabling flexible and precision-aware memory access. In this context, MAC operations may require accessing sixteen 4-bit, four 8-bit, or one 16-bit weight per neuron, each with an effective depth of 16 bits.

Figure 5.4 illustrates an example of such a single-bank addressing scheme, in which each parameter is uniquely identified using a combination of layer ID, parameter type (weight or bias), kernel ID, and, for weights, input ID. This hierarchical identification enables precise and conflict-free memory access within a unified memory bank. It is important to note that this figure is provided solely as a conceptual reference for single-bank addressing schemes, while all experimental evaluations in this chapter are conducted using the proposed multi-bank memory organization.

The addressing format is defined as follows: the first $\lceil \log_2(L) \rceil$ MSB bits represent the layer ID, followed by a select bit indicating parameter type (‘1‘ for bias, ‘0‘ for

weight). Subsequent bits encode the RAM address based on precision mode. The length of the weight/bias address $R\_addr(l)$ is given by:

$$R\_addr(l) = \lceil \log_2 N(l) \rceil + \lceil \log_2 J(l) \rceil \tag{5.1a}$$

$$Addr(l) = \lceil \log_2 L \rceil + 1 + R\_addr(l) \tag{5.1b}$$

Here, $N(l)$ and $J(l)$ denote the number of neurons and inputs per layer, respectively. For FC layers, the bias address consists only of the neuron identifier, while the weight address includes both neuron and input identifiers. Since a fixed address length is required, the maximum address length across all layers is computed as:

$$R\_addr = \max_{l=1,2,\ldots,L} \left\{ \lceil \log_2 N(l) \rceil + \lceil \log_2 J(l) \rceil \right\} \tag{5.2a}$$

$$Addr = \lceil \log_2 L \rceil + 1 + R\_addr \tag{5.2b}$$

Importantly, new addresses in FIFO loading are generated dynamically based on precision mode: *Addr+1* for 16-bit, *Addr+2* for 8-bit, and *Addr+4* for 4-bit precision. This precision-aware addressing ensures efficient weight/bias management across layers, minimizes memory overhead, and supports runtime adaptability for diverse DNN configurations.

### 5.3.2 Precision-aware Operand Pre-processing and SIMD Computations

The proposed architecture, denoted as `SIMD-Low` and illustrated in Figure 5.3, presents a novel multi-precision processing paradigm that explicitly targets full hardware utilization across all supported precision modes—a limitation that remains insufficiently addressed in many state-of-the-art designs. For symmetric precision configurations (e.g., $4\times4$, $8\times8$, and $16\times16$), the architecture supports sixteen 4-bit operands,

four 8-bit operands, and a single 16-bit operand per processing cycle, respectively.

In addition to symmetric operation, `SIMD-Low` efficiently supports asymmetric precision modes (e.g., $4 \times 16$), in which one 16-bit operand is processed concurrently with four 4-bit operands, enabling four parallel $16 \times 4$ MAC operations. This capability is particularly important for mixed-precision DNN workloads, where activations and weights frequently employ heterogeneous precision levels. By accommodating such asymmetry, the proposed design enhances both computational flexibility and energy efficiency.

Data-path widths are dynamically adjusted based on the selected precision mode, utilizing 64 bits in 4-bit mode, 32 bits in 8-bit mode, and 16 bits in 16-bit mode. The first processing element performs precision-aware operand pre-processing before the multiplier array, ensuring that all computational resources remain fully engaged even during low-precision operations. This approach effectively eliminates the under-utilization commonly observed in conventional multi-precision architectures. The pre-processed signed-magnitude operands are subsequently supplied to an array of 4-bit multipliers, which are systematically reused to realize higher-precision computations, thereby demonstrating efficient datapath reuse and reduced hardware overhead.

The partial products generated by the multiplier array are accumulated through a set of dedicated adder stages ($S_0$, $S_1$, and $S_2$), each optimized to match the active precision mode. Full resource utilization is maintained across modes, employing direct accumulation in 4-bit operation (`mode 00`) and shift-and-add accumulation for 8-bit (`mode 01`), 16-bit (`mode 10`), and asymmetric precision (`mode 11`), as depicted in Figure 5.3. To ensure consistent bit-widths, precision-aware rounding is applied based on the selected mode.

The architecture incorporates precision-aware rounding mechanisms that adapt to the selected SIMD PE operating mode, as illustrated in Figure 5.3. Control signals such as `valid_in` and `valid_out` govern the validity of input and output data, enabling or disabling computational units as required to maintain correct dataflow. The `clear` signal is used to reset the accumulator prior to the arrival of new feature map data, thereby ensuring accurate accumulation for subsequent computations. During MAC

operations, the `accum` signal remains asserted, while bias values are preloaded into the accumulator register block to minimize initialization latency.

Operands and weights are retrieved from dedicated 64-bit buffers and undergo precision-aware pre-processing based on the selected operating mode. Specifically, in `mode 00` (4-bit precision), all 64 bits are fetched; in `mode 01` (8-bit precision), the 32 least significant bits (LSBs) are accessed; and in `mode 10` (16-bit precision), the 16 LSBs are retrieved. For asymmetric operation in `mode 11`, 16 LSBs are fetched from the input buffer and 4 LSBs from the weight buffer. This precision-aware operand mapping efficiently distributes 4-bit, 8-bit, and 16-bit data across an array of sixteen 4-bit multipliers, thereby ensuring full hardware utilization across all precision modes. In contrast to conventional multi-precision designs, where low-precision configurations often lead to substantial under-utilization of computational resources, the proposed architecture maintains consistent and optimal resource engagement.

The MAC unit achieves high throughput by performing sixteen 4×4, four 8×8, or one 16×16 multiplication and accumulation per clock cycle, depending on the selected mode. For asymmetric precision (e.g., 4×16), the architecture computes four parallel 16×4 MAC operations, further enhancing flexibility for mixed-precision DNN workloads. A multiplexer and left-shifter (zero-appending in the LSB) arrangement ensures correct alignment of partial sums across precision modes. Finally, partial products are accumulated using an optimized adder-tree structure, delivering both scalability and efficiency in multi-precision computations. The architecture supports 4-, 8-, and 16-bit multiplications, with inherent scalability for higher precisions—a key feature that has been extended and analyzed for 8/16/32-bit SIMD computations. This scalability ensures that the same design principles can be adapted for future precision requirements without structural complexity.

The adder stages are organized into a hierarchical structure comprising eight `S0` adders (12-bit), four `S1` adders (20-bit), two `S2` adders (28-bit), and one 36-bit adder, complemented by sixteen 4-bit multipliers and a 44-bit accumulator block. The accumulated result, stored in the `quire` register, is rounded according to the active precision mode, guaranteeing consistency with the input bit-width and minimizing

quantization errors. At the end of each PE operation, the `valid_out` signal asserts high, producing the output (`PE_out`). The 44-bit `quire` enables up to 256 accumulations for full-precision 16-bit operations, while additional guard bits support iterative accumulations, significantly enhancing MAC utility. For lower precisions such as 8-bit and 4-bit, accumulator sizes are dynamically adjusted to maximize resource utilization and energy efficiency. Performance is further optimized through a 5-stage pipelined architecture, which improves throughput and reduces latency across all precision modes. This combination of scalability, precision-aware rounding, and pipeline optimization positions the proposed design as a highly efficient solution for next-generation DNN accelerators.

### 5.3.3 Hardware Reuse and Optimized Adder-Tree Architecture for Multi-Precision SIMD PE

The proposed architecture introduces a novel resource optimization strategy by exploiting hardware reuse within SIMD multiply–accumulate (MAC) units. Unlike prior approaches that reuse adder stages but suffer from limited parallelism and consequent under-utilization, the proposed design ensures full utilization of arithmetic resources across all supported precision modes. By enabling extensive parallel computation and efficient resource allocation, the architecture achieves up to a fourfold performance improvement over state-of-the-art solutions.

A key novelty of the proposed architecture lies in its unified support for both symmetric precision modes (4×4, 8×8, and 16×16) and asymmetric precision modes (16×4) using a single hardware instance. This unified design maximizes computational throughput at lower precisions without incurring additional hardware overhead. The adder tree is organized into four hierarchical stages comprising adders with bit widths of 12, 20, 28, and 36 bits, instantiated in quantities of 8, 4, 2, and 1, respectively. Partial products generated by the multi-precision SIMD multiplier are first accumulated using the 12-bit adders, after which intermediate sums are progressively reduced through successive stages of the adder tree. The final accumulation is performed by

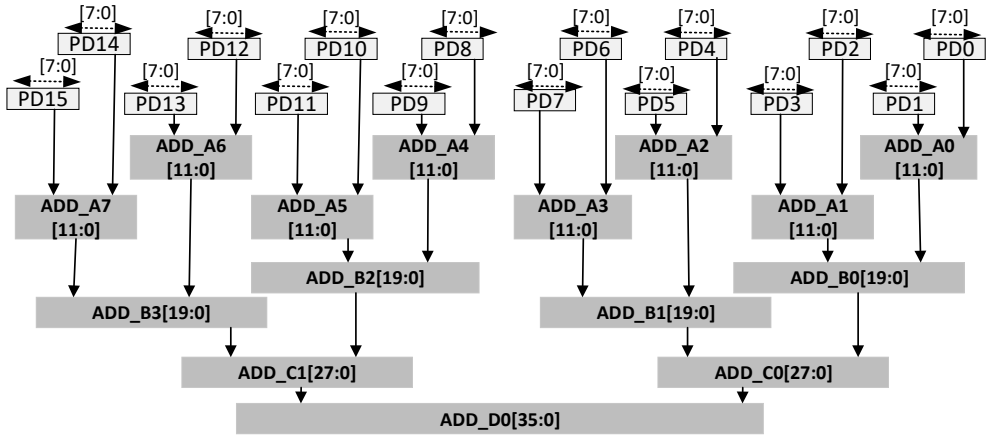



Figure 5.5: Adder Stage setup optimized for 4-bit precision computation (Mode `00`), which performs accumulation of 16 partial products (PD).

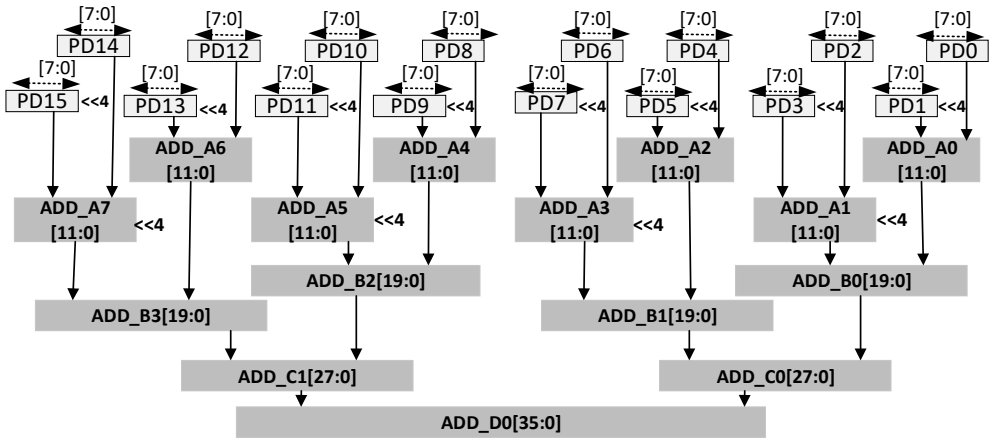



Figure 5.6: Adder Stage Setup optimized for 8-bit precision computation (Mode `01`), which performs shifting in PD, addition followed by quire accumulation.

the 36-bit adder, and the resulting output is stored in the accumulator (`quire`) to support iterative accumulation across multiple operations.

Mode-specific optimizations ensure precision-aware computation. For mode `00` (4-bit), direct accumulation is performed as depicted in Figure 5.5.

For mode `01` (8-bit), partial products undergo 8-bit shifting before addition, with alternate outputs shifted by 4 bits to maintain alignment, as depicted in Figure 5.6. For mode `10` (16-bit), all 15 adders participate in shift-and-add operations, with adder D0 performing accumulation with the `quire`, as depicted in Figure 5.7. In asymmetric mode `11`, partial products are summed using 20-bit adders, followed by final accumulation in 28-bit adders, as depicted in Figure 5.8. Notably, the 12-bit adders

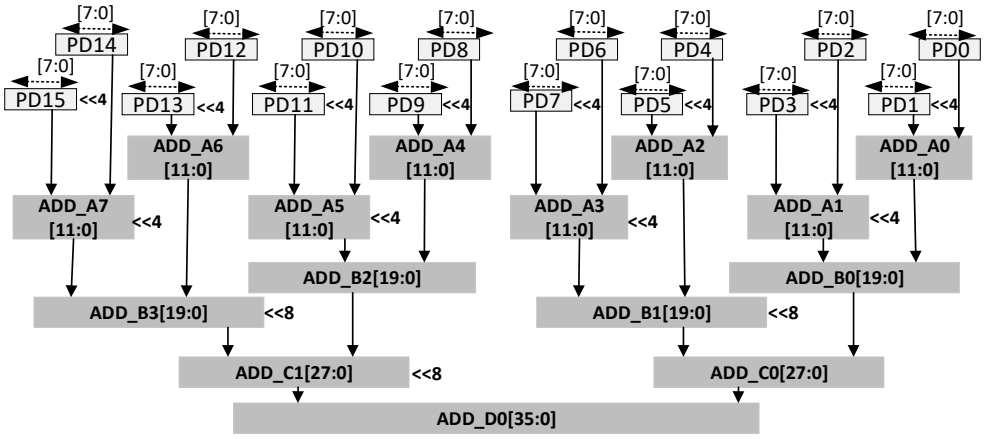



Figure 5.7: Adder Stage Setup optimized for 16-bit precision computation (Mode 10), performing shifting in PD, addition followed by quire accumulation.

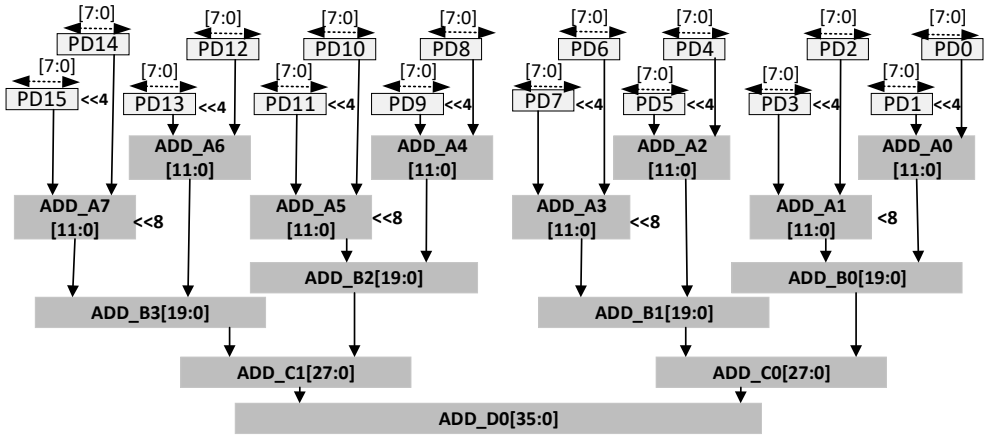



Figure 5.8: Adder Stage Setup optimized for 16x4 Asymmetric Precision Computation (Mode 11), performing shifting in PD, addition followed by quire accumulation.

perform accumulation in mode 00 but act as shift-and-add units in higher precision modes, while the 28-bit and 36-bit adders provide final outputs for modes 01, 11, and 10 respectively. This precision-aware reuse of adder stages effectively eliminates idle hardware cycles by dynamically adapting the computation path to the required operand precision. As a result, the architecture maintains high utilization across varying precision levels, avoiding the inefficiencies typically associated with fixed-precision designs. The proposed approach supports multiple precision configurations without requiring any structural modifications, enabling seamless operation across diverse computational demands. Furthermore, the design is inherently scalable, allowing straightforward extension to accommodate future DNN workloads with increased complexity

and precision requirements. Collectively, these features demonstrate the versatility and optimization capability of the proposed architecture across all precision modes, establishing a new benchmark for efficient SIMD-based MAC architectures.

### 5.3.4 Design Adaptation for SIMD-High Configuration

The proposed architecture is seamlessly extended to support 8-, 16-, and 32-bit precision in the SIMD-High configuration. It accommodates multiple operand granularities, including sixteen operands for 8-bit precision, four operands for 16-bit precision, and a single operand for 32-bit symmetric precision. In addition to symmetric modes, the architecture supports asymmetric computation, such as combining one 32-bit operand with four 8-bit operands to enable four parallel 32×8 multiply–accumulate (MAC) operations. To match these precision modes, the input data width scales proportionally, operating on 128 bits in 8-bit mode, 64 bits in 16-bit mode, and 32 bits in 32-bit mode.

At the core of the computational structure is an array of 8-bit multipliers, which generates higher-precision results through controlled accumulation of partial products. Dedicated adder stages, denoted as $S_0$, $S_1$, and $S_2$, efficiently manage the reduction of partial sums with bit widths of 24, 32, and 48 bits, respectively. The final accumulation path consists of a 64-bit adder followed by a 72-bit accumulator, enabling precise and overflow-safe accumulation across multiple operations. The execution model closely follows that of the SIMD-Low configuration, differing primarily in operand bit widths and the corresponding scaling of arithmetic unit sizes.

Operational behavior varies across precision modes. Mode `00` (8-bit) employs direct addition of partial products, while modes `01` (16-bit), `10` (32-bit), and `11` (asymmetric) apply shifted alignment of partial sums to ensure correct accumulation across varying operand precisions. Beyond the explicitly supported configurations, the architecture is inherently scalable to higher precision formats, such as 16-, 32-, and 64-bit operations, by proportionally extending the computational elements without requiring significant structural modifications. Furthermore, the design maintains full support for mixed-precision and asymmetric computation (e.g., parallel 16×64 MAC operations), thereby

ensuring adaptability to emerging DNN models that demand higher or heterogeneous precision formats.

Overall, this unified architectural approach preserves structural simplicity, minimizes idle hardware cycles, and ensures scalable performance across a wide range of precision requirements in high-performance DNN workloads.

## 5.4 Performance Evaluation and Validation of Software and Hardware Implementations

This section presents a comprehensive evaluation of the proposed architecture using both software-based emulation and hardware-level implementation. The software evaluation models the proposed design to assess numerical accuracy across diverse datasets and multiple deep neural network (DNN) models under a range of precision configurations. This analysis validates the functional correctness and precision robustness of the architecture when operating in symmetric and asymmetric modes.

For hardware validation, the architecture is implemented on an FPGA platform, enabling detailed characterization of key physical and performance metrics, including resource utilization, critical path delay, power consumption, throughput, and energy efficiency across different operand bit widths. In addition, the design is synthesized using a 65 nm fast–fast (FF) process corner and evaluated at 25°C, providing a realistic assessment of achievable performance, scalability, and technology feasibility. Both FPGA and ASIC evaluations are conducted using standard implementation and analysis flows to ensure fair and reproducible results.

The following subsections present an in-depth discussion of the experimental results, highlighting the performance, efficiency, and scalability advantages of the proposed SIMD processing element (PE) architecture compared with conventional and state-of-the-art designs.

### 5.4.1 Software Implementation and Accuracy Validation

The proposed SIMD processing element (PE) architecture is rigorously evaluated for inference accuracy across multiple dynamic fixed-point precision configurations, including 4-, 8-, 16-, and 32-bit modes. The evaluation is conducted using a diverse set of benchmark datasets—MNIST, CIFAR-10, CIFAR-100, and ImageNet-1000—and widely adopted deep neural network (DNN) models, including LeNet-5, ResNet-18, VGG-16, CaffeNet, and AlexNet. This comprehensive experimental setup enables validation of the architecture's adaptability and robustness across both lightweight and computationally intensive workloads.

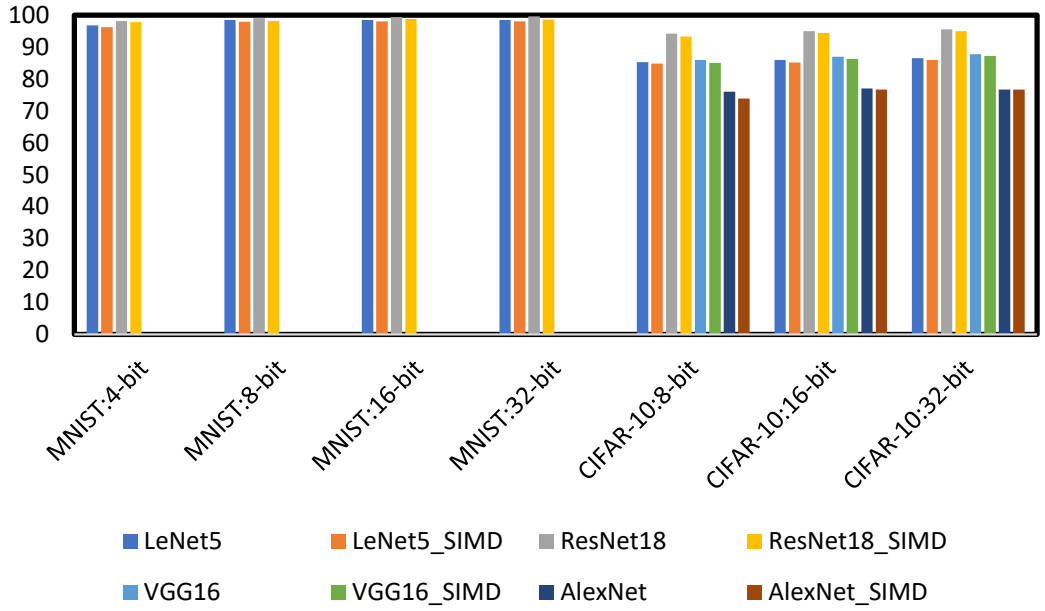



Figure 5.9: Inference Accuracy Evaluation on Various DNN models for MNIST and CIFAR-10 datasets using the Proposed SIMD PE architecture in comparison to the baseline accuracy [31].

Figures 5.9 and 5.10 compare the inference accuracy of the proposed architecture against baseline results reported in [31]. For the MNIST dataset, the proposed design achieves an accuracy of 98.01% at 32-bit precision, closely matching the baseline accuracy of 98.46%, while exhibiting almost negligible degradation at reduced precision levels (97.97% at 8-bit).

Similarly, for CIFAR-10, the architecture maintains near-baseline performance,

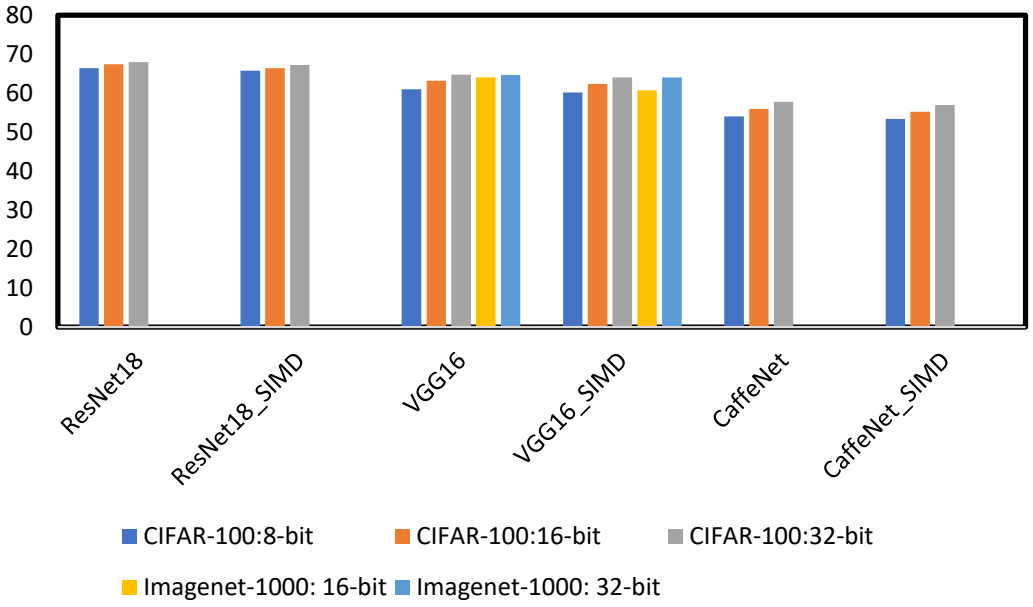



Figure 5.10: Inference Accuracy Evaluation on Various DNN models for CIFAR-100 and ImageNet-1000 datasets using the Proposed SIMD PE compared to the baseline accuracy[31].

achieving 76.75% accuracy at 32-bit precision with only marginal reductions observed in the 8-bit and 16-bit configurations. These results demonstrate that the proposed SIMD PE architecture effectively preserves inference accuracy across a wide range of precision settings, thereby validating its suitability for precision-scalable DNN inference

For more complex models, such as VGG-16 evaluated on CIFAR-10 and ImageNet-1000, the observed accuracy degradation remains within 1–2% of the baseline even when operating at reduced precision. On the CIFAR-100 dataset, both CaffeNet and ResNet-18 exhibit accuracy deviations of less than 1%, further confirming the precision-aware flexibility and robustness of the proposed architecture. These results demonstrate that the design delivers scalable multi-precision support without significantly compromising inference accuracy, making it well-suited for both resource-constrained edge devices and high-performance artificial intelligence (AI) systems.

Notably, the proposed architecture achieves near-identical accuracy to the baseline at 32-bit precision, with only minor deviations observed at 16-bit and 8-bit operating modes. For instance, under the 8-bit configuration, the accuracy deviation is approxi-

mately 2% for the VGG-16 model on CIFAR-10 and around 3.5% for ImageNet-1000. Similarly, AlexNet experiences an accuracy degradation of approximately 2%, while LeNet-5 exhibits only a 0.49% reduction. Such marginal accuracy losses are well within acceptable limits for most edge-AI applications, which are inherently tolerant to moderate numerical imprecision.

Overall, the combination of minimal accuracy degradation—even for complex datasets—together with low resource utilization, high throughput, and comprehensive multi-precision support underscores the efficiency and practicality of the proposed SIMD PE architecture. These findings confirm that the design represents an effective solution for variable-precision, resource-constrained environments, offering a favourable balance between computational efficiency and inference accuracy.

### 5.4.2 Hardware Resources Utilization and Comparison

The hardware implementation of the proposed architecture was evaluated across both FPGA and ASIC platforms to ensure a comprehensive assessment of its practicality. For the FPGA prototype, the design was mapped onto a 6-bit LUT-based fabric without using any DSP resources, and evaluated at room temperature (25°C) with a target operating frequency of 100 MHz. For the SIMD-High configuration, the design operated at 195 MHz with a supply voltage of 1.0 V. These settings establish a consistent baseline for comparing hardware performance across both implementation environments.

The FPGA implementation results, summarized in Table 5.2, demonstrate the efficiency and versatility of the proposed SIMD processing element (PE) architecture. The evaluation was carried out on AMD Virtex-VC-707 FPGA board. In contrast to a standalone PE approach with SIMD support, which instantiates separate hardware modules for each precision mode (4/8/16-bit and 8/16/32-bit) and relies on dedicated arithmetic units for all operand computations—thereby incurring substantial area overhead—the proposed design exploits dynamic resource reuse to achieve superior scalability and efficiency. Although the *SIMD-High* configuration introduces an overhead of approximately 1,590 LUTs compared to a conventional 32-bit MAC,

Table 5.2: Performance Metrics for Conventional vs. Proposed SIMD PE Across Precision Modes.

| Bit Precision | LUT | FF | Critical Path (ns) | Power (mW) | MAC Ops |
|---|---|---|---|---|---|
| **4-bit MAC** | 37 | 8 | 2.192 | 2 | – |
| **8-bit MAC** [64] | 86 | 16 | 2.361 | 6.6 | 4 |
| **16-bit MAC** [64] | 335 | 32 | 2.892 | 14.95 | 2 |
| **32-bit MAC** | 1424 | 208 | 6.513 | 22 | – |
| **Standalone PE with SIMD Support (Dedicated Hardware)** | | | | | |
| ***SIMD-Low* (4/8/16)** | 1607 | 224 | 2.892 | 24.82 | 16/4/1 |
| ***SIMD-High* (8/16/32)** | 5604 | 592 | 6.513 | 86.06 | 16/4/1 |
| **Proposed PE with SIMD Support** | | | | | |
| ***SIMD-Low* (4/8/16)** | 1279 | 576 | 2.190 | 14 | 16/4/1 |
| ***SIMD-High*(8/16/32)** | 3010 | 1121 | 5.245 | 35 | 16/4/1 |

this increase is effectively amortized by its ability to execute multiple operations concurrently. Specifically, the proposed *SIMD-High* PE can simultaneously support one 32-bit MAC, four 16-bit MACs, sixteen 8-bit MACs, and four asymmetric 32×8 MAC operations. Achieving comparable functionality with a standalone SIMD implementation using dedicated hardware would require 5,604 LUTs for 8/16/32-bit precision support and an additional 1,607 LUTs for 4/8/16-bit modes, highlighting the significant area savings enabled by the proposed approach.

Furthermore, the proposed SIMD architecture exhibits favorable and scalable area behavior across precision modes. As operand precision increases, the area growth remains moderate—approximately 2.3× in ASIC synthesis and 2.5× on FPGA when doubling precision in the SIMD-Low configuration—which is substantially lower than the 4–5× area increase typically observed in conventional multiplier-based designs. This efficiency is primarily attributed to extensive hardware reuse, with incremental area overhead largely confined to the accumulator stage required for SIMD operation. Owing to this reuse and the preservation of parallelism across precision modes, throughput scales effectively with increasing bit width, resulting in improved throughput-per-area efficiency without diminishing returns at higher precisions.

Beyond resource optimization, the proposed architecture delivers timing improve-

ments, with the *SIMD-High* PE achieving a 19.4% reduction in critical path delay compared to the conventional 32-bit MAC, and the *SIMD-Low* configuration offering a 24.27% reduction relative to the conventional 16-bit MAC. Furthermore, LUT utilization is reduced by 46.2% for 8/16/32-bit and 20.4% for 4/8/16-bit implementations, demonstrating superior resource efficiency. These reductions are coupled with enhanced throughput and expanded support for both symmetric and asymmetric operations. It is important to note that FPGA-based implementations rely on LUT-based logic, which limits the extent of performance improvements compared to ASIC designs. However, the proposed architecture shows significant gains in ASIC synthesis results, where dedicated logic and optimized pipelines enable substantial improvements in area, frequency, and throughput. This distinction underscores the scalability and practicality of the proposed design for real-world deployments.

The ASIC synthesis results, summarized in Table 5.3, demonstrate the significant advancements achieved by the proposed SIMD PE architecture. Compared to the design in [74] at the same 65 nm technology node, the proposed *SIMD-Low* configuration achieves a 28.57% reduction in on-chip area while delivering a 2.08× improvement in throughput for 4-bit precision, along with a 22.4% increase in operating frequency. Table 5.3 also highlights how the proposed SIMD-High and SIMD-Low configurations compare favorably with representative SOTA architectures across multiple precision modes. Specifically, the proposed designs provide markedly smaller area footprints (0.036 $mm^2$ and 0.015 $mm^2$) relative to prior works, competitive delays (3.57 ns and 2.27 ns), and significantly lower power consumption (3.75 mW and 1.59 mW). In addition, both configurations scale effectively with precision, achieving up to 3.12 GOPS and 4.218 GOPS, with peak energy efficiency reaching 1.104 TOPS/W. These SOTA comparisons further reinforce the hardware efficiency of the proposed multi-precision SIMD design. These results underscore the superior hardware performance of the proposed architecture, particularly under low-precision operating modes. A key distinguishing feature of the proposed design is its native support for asymmetric precision operations, which enhances flexibility and enables efficient deployment across diverse DNN workloads with varying precision requirements, while maintaining a favorable

Table 5.3: A Detailed Comparison of Hardware Performance Parameters of State-of-the-Art Works with the Proposed Work.

| **Parameter** | **Zhao et al '24** [142] | **Enrico et al.'23** [141] | **Miro et al.'20** [144] | **Shin et al.'17** [74] | **Li et al.'22** [130] | **Zhang et al.'19** [75] | **Mao et al.'21** [76] | **Raut et al.'24** [64] | **This work** *SIMD-High* | **This work** *SIMD-Low* |
|---|---|---|---|---|---|---|---|---|---|---|
| Tech. node ($nm$) | 14 | 22 | 28 | 65 | 28 | 90 | 90 | – | 65 | 65 |
| Precision | 16,8,4,2 | 2-8 | 8,16,32 | 4, 8, 16 | 16, 32, 64 | 16, 32, 64 | 16, 32, 64 | 8, 16 | 8, 16, 32 | 4,8,16 |
| Arithmetic | INT | FxP | FxP | FxP | FP | FP | FP | FxP | FxP | FxP |
| Voltage (V) | – | - | 0.9 | 0.77-1.1 | 1.0 | 1.0 | 1.0 | – | 1.0 | 1.0 |
| Area ($mm^2$) | 0.0134 | 0.0136 | 74.5 (Die Area) | 0.021 | 0.15 | 0.795 | 0.013 | – | 0.036 | 0.015 |
| Delay (ns) | – | – | 2.96 | 4.5 | 2.96 | 2.718 | 5.1 | 2.27 | | |
| Power (mW) | 0.14 | - | 96 | 279(786MACs) | 51.4 | 177 | 3.59 | - | 3.75 | 1.59 |
| Frequency (MHz) | 1160 | 1200 | 350 | 200 | 1351 | 667 | 1351 | 197.51 | 195 | 258 |
| Throughput | 0.93(16b) | 4.2(8b) | 0.375(32b) | 0.39(16b) | 0.67(64b) | 0.67 (64b) | 0.67 (64b) | 0.19(16b) | 0.195(32b) | 0.258(16b) |
| (GOPS) | – | | 0.75(16b) | | 2.7(32b) | 1.34 (32b) | 6.75(32b) | 0.78(8b) | 0.78(16b) | 1.032(8b) |
| | 2.86(2b) | 7.9(2b) | 1.5(8b) | 1.56(4b) | 10.7(16b) | 2.67 (16b) | 27 (16b) | – | 3.12(8b) | 4.218(4b) |
| Energy Efficiency | 6.73(16b) | 0.4(8b) | 0.058(32b) | – | 0.026(64b) | 0.023 (64b) | 0.026 (64b) | – | 0.069(32b) | 0.016(16b) |
| (TOPS/W) | – | – | 0.12(16b) | – | 0.105(32b) | 0.046 (32b) | 0.128(32b) | | 0.276(16b) | 0.064(8b) |
| | 20.51(2b) | 0.8(2b) | 0.23(8b) | – | 0.421(16b) | 0.092(16b) | 0.513(16b) | – | 1.104(8b) | 0.256(4b) |

balance between accuracy, performance, and energy efficiency.

Compared with prior multi-bit-width systolic accelerators, such as those proposed by Huang *et al.* [62] and Li *et al.* [91], the proposed SIMD architecture adopts a fundamentally different form of datapath reuse that enables both symmetric and asymmetric precision modes within a unified processing element. Existing bit-split systolic designs are limited to symmetric precision support and offer constrained parallelism, typically supporting four MAC operations at 4-bit precision and two MAC operations at 8-bit precision. In contrast, the proposed `SIMD-Low` configuration supports sixteen MAC operations in 4-bit mode, four MAC operations in 8-bit mode, and a single MAC operation in 16-bit mode. This results in up to a 4× throughput improvement at 4-bit precision and a 2× improvement at 8-bit precision relative to these prior approaches. Moreover, the architecture naturally extends to asymmetric precision execution (e.g., 4×16), enabling concurrent mixed-precision MAC operations—capabilities that are not supported by existing systolic-array-based designs. This enhanced flexibility, together with increased parallelism, represents a core innovation of the proposed architecture.

Table 5.4: Hardware metrics for the Multiplier and Adder units of the proposed SIMD-Low PE.

| **Metric** | **Multiplier** | **Adder** | **Total** |
|---|---|---|---|
| Area ($\mu$m$^2$) | 4467.24 | 11214.36 | 15681.6 |
| Power ($mW$) | 0.5319 | 1.0495 | 1.5814 |
| Delay($ns$) | 0.32 | 1.92 | 2.24 |
| Ports | 661 | 2861 | 3522 |
| Nets | 1683 | 6374 | 8057 |
| Cells | 1137 | 3480 | 4617 |

Further comparisons with the SAMURAI architecture [144] demonstrate that the proposed *SIMD-High* configuration achieves a 1.04× throughput improvement at 16-bit precision and a 2.08× improvement at 8-bit precision. In addition to throughput gains, the proposed architecture delivers substantial improvements in energy efficiency, achieving enhancements of 1.19× at 32-bit precision, 2.3× at 16-bit precision, and 4.8× at 8-bit precision. Furthermore, when compared with the edge-focused accelerator

presented in [142], the proposed design achieves a 1.47× throughput improvement at the lowest supported precision, further highlighting its effectiveness for low-precision, resource-constrained deployments. The architecture in [141] demonstrates superior peak performance and energy efficiency by aggressively reusing a high-frequency CPU multiplier and focusing on uniform-precision General Matrix–Matrix Multiplication (GEMM) execution. In contrast, this study targets native support for asymmetric precision and predictable execution across 4/8/16-bit modes, prioritizing effective utilization under mixed-precision inference scenarios over peak throughput. These gains, combined with multi-precision adaptability, position the proposed architecture as a highly efficient solution for next-generation DNN accelerators. Table 5.4 summarizes the hardware metrics of the multiplier and adder units in the proposed SIMD-Low processing element with dynamic adder reuse enabled, illustrating their relative contribution to area, power, and timing. Overall, these results confirm that the proposed SIMD PE architecture consistently outperforms prior designs in terms of throughput, energy efficiency, and precision flexibility, particularly in low- and mixed-precision regimes.

## 5.5 Summary

In this chapter, a SIMD processing element architecture designed for precision-aware DNN acceleration has been presented. In contrast to conventional approaches that require separate hardware for different precision configurations, the proposed design employs dynamic resource reuse to support adaptive precision across two scalable configurations: *SIMD-Low* (4/8/16-bit) and *SIMD-High* (8/16/32-bit). The architecture supports both symmetric and asymmetric operations within a unified structure, enabling efficient hardware utilization while maintaining consistent throughput scalability. Experimental results corroborate the effectiveness of the proposed approach. FPGA-based evaluations indicate notable reductions in resource usage along with improved throughput, while ASIC synthesis at 65 nm shows up to 28.57% area reduction, a 22.4% increase in operating frequency, and energy efficiency improvements of up to

4.8× for the *SIMD-Low* configuration, compared with representative state-of-the-art designs. These gains are achieved with negligible accuracy loss across a range of benchmark DNN models.

Overall, the proposed architecture provides a practical balance between precision, flexibility, hardware efficiency, and performance scalability, making it suitable for both resource-constrained edge-AI platforms and high-performance computing systems.

# Chapter 6

# CORDIC-Driven Approximate SIMD PE Architecture for Resource-Efficient and Throughput-Enhanced Multi-Precision Computations within DNN models

In Chapter 5, a SIMD Processing Element (PE) architecture was introduced that supports flexible multi-precision arithmetic, enabling operations at 4-, 8-, and 16-bit precision, as well as extended 8-, 16-, and 32-bit modes for higher-accuracy computations. This adaptability allows the architecture to efficiently handle diverse numerical requirements across different DNN layers, balancing accuracy and performance.

The SIMD PE employs accurate, conventional multipliers ensuring numerical correctness during multiplication-heavy DNN workloads such as convolutions and fully connected layers. To further improve hardware efficiency, the design leverages adder reuse, allowing the same adder resources to be shared across multiple operations and precision modes. This reuse significantly reduces hardware overhead while maintaining high computational throughput. As a result, the proposed architecture achieves high-throughput DNN acceleration, effectively combining precision flexibility, computational accuracy, and resource-efficient design. This chapter extends the discussion by exploring the integration of approximate multipliers within the SIMD PE architecture.The focus is on evaluating the trade-offs between reduced hardware complexity

and power consumption versus the resulting impact on computational accuracy and DNN inference performance.

## 6.1 Introduction

In chip-level designs for DNN accelerators, low- to medium-precision arithmetic is widely employed to satisfy strict constraints on power consumption and silicon area. DNN models exhibit varying levels of sensitivity to reduced numerical precision: while some models suffer significant accuracy degradation at lower bit widths, others incur unnecessary power and area overheads when operated at higher precision due to redundant computations and underutilized hardware resources.

This variability underscores the need for scalable, precision-adaptive hardware architectures that can accommodate diverse computational requirements. Moreover, different DNN models—and even individual layers within the same model—often demand distinct bit widths to achieve an optimal balance between accuracy, performance, and energy efficiency [61].

The adaptive PE architecture, as discussed in chapter 5, supports operand sizes of 4-, 8-, and 16-bit in the SIMD_Low configuration and 8-, 16-, and 32-bit in the SIMD_High configuration, leveraging shared computational resources. However, the detailed analysis focuses on the 8-, 16-, and 32-bit SIMD modes. The architecture uses a dynamic fractional fixed-point format, enabling efficient reuse of resources during accumulation. This enhances parallelism in Multiply-Accumulate operations and ensures 100% utilization of computational units across all precision modes.

This study investigates the architectural application of the Coordinate Rotation Digital Computer (CORDIC) algorithm in DNN accelerators. The CORDIC algorithm, originally introduced by Jack E. Volder in 1959 [145], is capable of performing a wide range of arithmetic operations, including trigonometric evaluations, vector rotations, and multiplications. Subsequently, John Walther extended the algorithm to support unified operational modes and broader functional coverage [146]. Prior research [55] proposed a neuron computational unit based on a recursive CORDIC archi-

tecture, enabling efficient implementation of MAC operations and activation functions with significant improvements in area and power efficiency. However, the recursive nature of conventional CORDIC implementations results in high latency, requiring $n$ clock cycles to achieve $n$-bit precision. This characteristic limits their applicability in high-throughput, SIMD-based DNN architectures [92, 64].

To address these limitations, this chapter explores a pipelined, throughput-optimised CORDIC architecture suitable for SIMD processing. Building on this foundation, the proposed design integrates adaptive-precision computation with both symmetric and asymmetric arithmetic capabilities, leveraging CORDIC-based operations to enhance hardware efficiency and computational throughput in DNN accelerators. The key contributions of this study are as follows:

- A partial product computation scheme was developed in a **SIMD architecture using a pipelined CORDIC approach**, with stage-level optimization guided by Pareto-front analysis to balance accuracy, latency, and area.

- The proposed **C-SIMD PE supports multi-precision operations** at 8-, 16-, and 32-bit levels, achieving parallel execution of 16, 4, and 1 operations per cycle, respectively—significantly enhancing throughput and resource efficiency.

- A **scalable, reduced-precision adder tree** is introduced to support both symmetric (8×8, 16×16, 32×32) and asymmetric (8×32) multiplication modes, enabling high hardware reuse with minimal overhead across precision configurations.

- The architecture offers **configurable computational accuracy**, supporting both approximate and exact results by tuning the CORDIC pipeline depth—making it adaptable for precision-aware inference tasks.

- The proposed design is validated through software-level simulations on multiple datasets, with hardware synthesis results demonstrating **competitive physical performance** compared to state-of-the-art solutions.

## 6.2 Related Works and Motivation

To meet the high computational demands of DNNs, recent efforts have focused on both algorithmic and architectural optimizations. Approximate computing techniques have been widely explored for hardware-level enhancements, particularly in multipliers and accumulators [133, 10, 134, 147]. In addition, algorithmic approximations such as the CORDIC method have been investigated to reduce the complexity of nonlinear function evaluations [55, 49]. Resource constraints in DNN accelerators have also been addressed through reduced-precision computation, leveraging the inherent error tolerance of neural networks [29]. While low-precision processing offers notable gains in area and energy efficiency, it can degrade model accuracy in certain architectures. Conversely, high-precision computation increases power consumption due to redundant operations and wider datapaths. These trade-offs highlight the need for scalable and adaptive architectures that support variable bit precision, as precision requirements vary across models and even across layers within the same model [63].

Precision scalability is broadly categorized into 1D (weight-only) and 2D (weight and activation) scalable architectures [65]. Leading designs employ bit-serial computation [66, 67, 68, 69] and decomposed multipliers [70, 71, 72] to enable flexible precision using bit-fused MAC units. These units dynamically configure computation paths based on operand bit widths [70], though this often results in suboptimal hardware utilization and modest latency benefits. Optimization techniques such as loop-nest restructuring and shift-add reduction have been proposed to improve PE efficiency [71], but increasing operand precision results in longer accumulation cycles and higher power consumption. Sub-word parallelism in weight tensors has also been explored for multi-precision MACs [73, 74], typically using data gating to reduce dynamic power consumption and thereby reduce MAC utilization. Raut et al. [64] addressed this issue by improving parallelism, but their design supports only 4 and 1 MAC operations for 8-bit and 16-bit precision, respectively, which limits throughput. Multi-precision floating-point MACs have been proposed, but they suffer from either low hardware utilization [75] or accuracy loss due to mantissa truncation [76]. Variable-precision

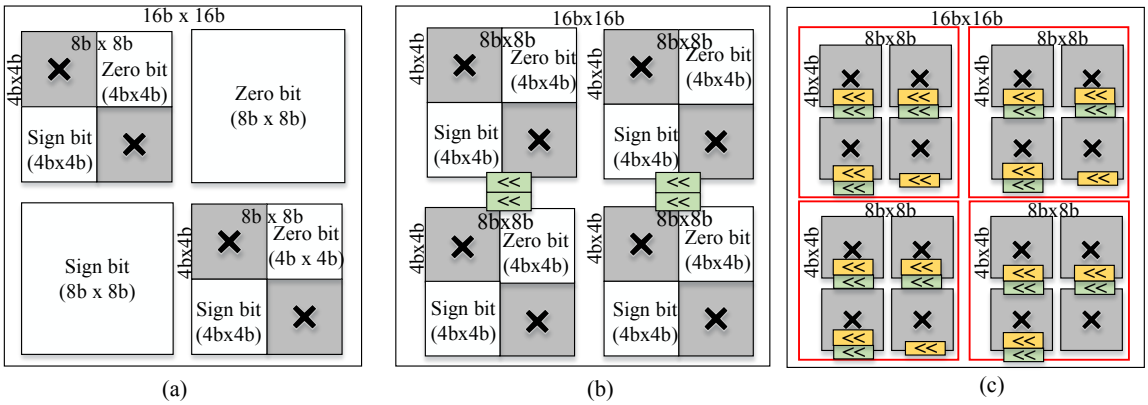



Figure 6.1: Multi-precision Architectures for DNN Inference: (a) HPS method [73], (b) BSC method [91], and (c) the Proposed Configurable C-SIMD PE, which ensures full resource utilization across all supported precision modes. Green shifters indicate shifts for 8-bit computation, while yellow markers denote shifts for 16-bit computation, illustrating the systematic shift-accumulate strategy for higher-precision operations.

multipliers with dynamic voltage-frequency scaling [77] have also been introduced; however, they leave submultipliers idle in low-precision modes, thereby constraining efficiency. Therefore, to support diverse bit widths (4/8/16/32-bit), scalable PEs must strike a balance among throughput, power, and accuracy across layers. Therefore, adaptive-precision computation emerges as a promising strategy for optimising PE design [129, 130, 148], as discussed in the previous chapter. This chapter presents a MAC unit based on the CORDIC architecture for partial product generation, supporting both symmetric and asymmetric computations and tailored for DNN workloads.

Fig.6.1 compares resource utilisation of common methods—High-Precision Split (HPS) [73] and Bit-Split-and-Combination (BSC) [91]—both limited by MAC underutilization due to logic gating. Fig. 6.1 illustrates various SIMD architectural topologies. In Fig. 6.1(a), the HPS configuration achieves 100% resource utilization at the highest precision level; however, utilization drops significantly to 25% in the lowest precision mode. In contrast, the BSC architecture shown in Fig. 6.1(b) offers better resource usage than HPS, yet still suffers from suboptimal utilization across varying precisions.

The proposed SIMD architecture for 4/8/16-bit modes (Fig. 6.1(c)) employs shift-based operand alignment to realize higher-precision operations using 4-bit partial-

product multipliers. For 8-bit computation, the four 4-bit multipliers perform shifts of 0, 1×, 1×, and 2× bits, where × = 4, effectively realizing 8-bit multiplication using 4-bit units, as indicated by the green shift markers. For 16-bit computation, each 4-bit multiplier incrementally shifts by 4 bits (0, 1×, 2×, 3×, and so on), enabling a full 16-bit product, as highlighted by the yellow markers. This systematic shift-accumulation strategy generalizes similarly across other precision blocks. Similarly, in the C-SIMD_High configuration, 16-bit precision computation requires shift offsets of 0, 1×, 1×, and 2× bits, where × = 8, as indicated by the green shift markers. For 32-bit computation, each 8-bit multiplier incrementally shifts by 8 bits (i.e., 0, 1×, 2×, 3×, and so forth), as indicated by yellow shift markers, enabling the formation of a full 32-bit product. Overall, the proposed architecture overcomes the limitations of former architectures by maintaining 100% resource utilisation across all precision modes. This configuration leverages all multipliers along with dedicated shifters, enabling precise and efficient multiplication at every supported precision level.

## 6.3 Proposed Hardware Design for MAC Computation and Quantization-Enabled Processing

Multi-precision support—covering both symmetric and asymmetric computations across 4-, 8-, 16-, and 32-bit formats—is essential for balancing computational accuracy and efficiency in deep neural network accelerators. Supporting a wide range of precisions increases architectural flexibility and enables per-layer precision tuning, improving hardware resource utilisation while maintaining high performance. This chapter proposes a configurable C-SIMD processing element (PE) architecture that supports all symmetric precision modes (4×4, 8×8, 16×16, and 32×32) as well as asymmetric modes (4×16, 8×32) via CORDIC-based partial-product generation. The proposed C-SIMD PE also supports asymmetric MAC operations, including 32×8-bit configurations, enabling flexible mixed-precision computations tailored for modern DNN layers. The bitwidth of the underlying logic blocks is fixed for each config-

Table 6.1: Unified C-SIMD operating modes illustrating precision scalability, input/weight register utilization, operand symmetry, and effective MAC parallelism per cycle.

| C-SIMD Mode | Parameter | C-SIMD_Low | C-SIMD_High |
|---|---|---|---|
| Mode_00 | Operand format | 4×4 | 8×8 |
| | Precision | 4-bit | 8-bit |
| | Input register width (bits) | 64 | 128 |
| | Weight register width (bits) | 64 | 128 |
| | Computation type | Symmetric | Symmetric |
| | MACs per cycle | 16 | 16 |
| Mode_01 | Operand format | 8×8 | 16×16 |
| | Precision | 8-bit | 16-bit |
| | Input register width (bits) | 32 | 64 |
| | Weight register width (bits) | 32 | 64 |
| | Computation type | Symmetric | Symmetric |
| | MACs per cycle | 4 | 4 |
| Mode_10 | Operand format | 16×16 | 32×32 |
| | Precision | 16-bit | 32-bit |
| | Input register width (bits) | 16 | 32 |
| | Weight register width (bits) | 16 | 32 |
| | Computation type | Symmetric | Symmetric |
| | MACs per cycle | 1 | 1 |
| Mode_11 | Operand format | 16×4 | 32×8 |
| | Precision | 16×4 | 32×8 |
| | Input register width (bits) | 16 | 32 |
| | Weight register width (bits) | 16 | 32 |
| | Computation type | Asymmetric | Asymmetric |
| | MACs per cycle | 4 | 4 |

uration, either C-SIMD_Low (4/8/16-bit) or C-SIMD_High (8/16/32-bit)—enabling runtime precision scaling depending on the selected mode. While the architecture is adaptable to both C-SIMD_Low and C-SIMD_High configurations, the detailed analysis and evaluation in this section focus on the C-SIMD_High configuration, i.e., the 8-, 16-, and 32-bit precision modes, as indicated in Table 6.1. The table distinguishes between the C-SIMD_Low and C-SIMD_High operating modes and highlights the precision configurations examined in detail in the subsequent sections. The proposed architecture accommodates multiple precision modes organized into the two configurations—C-SIMD_Low and C-SIMD_High—by systematically controlling the bitwidths of the underlying logic blocks.

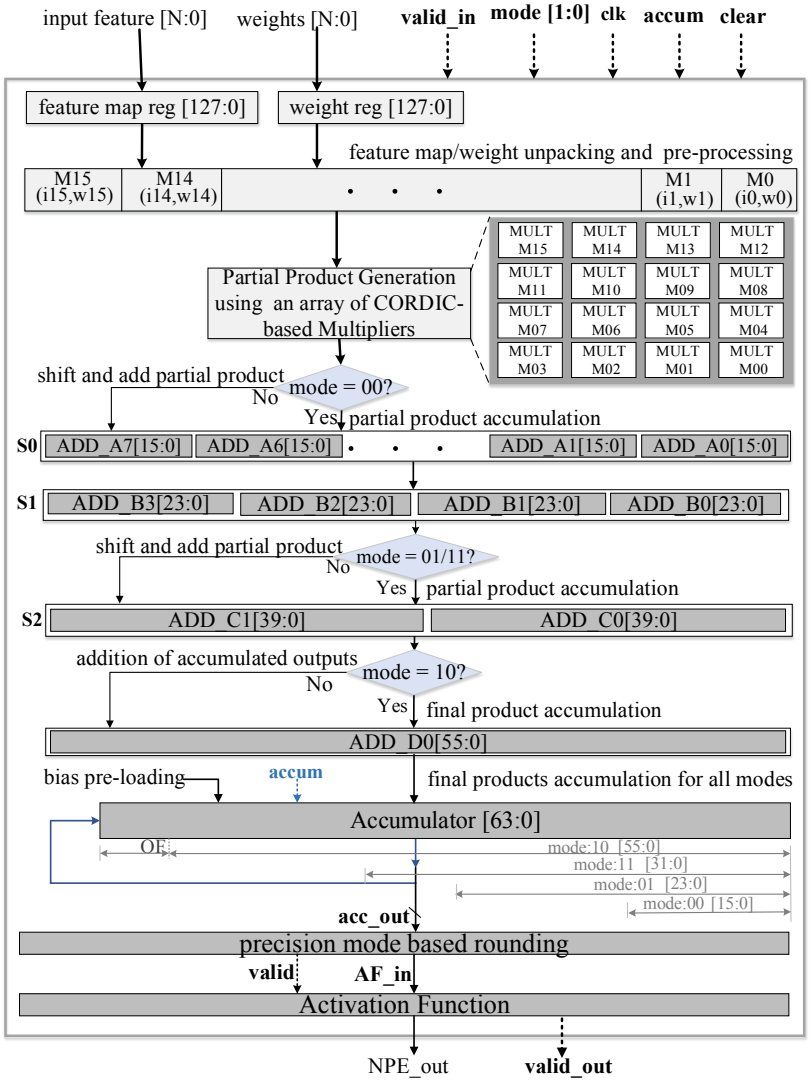



Figure 6.2: Proposed C-SIMD PE architecture supporting Configurable Multi-Precision MAC Operations. The design integrates pipelined CORDIC-based multipliers, scalable adders, and quire accumulation units to enable efficient 8-, 16-, 32-bit symmetric and 32×8-bit asymmetric computations for Deep Learning Inference.

Adaptive-precision SIMD processing elements comprise arithmetic units capable of performing both symmetric and asymmetric precision operations. These units interface with local register memory and control logic to manage data flow, and output results are produced systolically, propagating through diagonal elements according to the accumulation pattern.

The proposed C-SIMD processing element (PE) architecture, shown in Fig. 6.2, supports all symmetric precision modes (4×4, 8×8, 16×16, and 32×32) as well as

asymmetric modes (4×16 and 8×32) through CORDIC-based partial product generation. In addition, the proposed C-SIMD PE supports asymmetric MAC operations, including 32×8-bit configurations, enabling flexible mixed-precision computation suitable for modern DNN layers.

The bitwidth of the underlying logic blocks is fixed for each configuration—either C-SIMD_Low (4/8/16-bit) or C-SIMD_High (8/16/32-bit)—while allowing runtime precision scaling based on the selected operating mode. Although the architecture is compatible with both C-SIMD_Low and C-SIMD_High configurations, the detailed analysis and evaluation presented in this section focus on the C-SIMD_High configuration, namely the 8-, 16-, and 32-bit precision modes, as summarized in Table 6.1. The CORDIC implementation is described in Section 6.3.3.

The architecture is designed for two-dimensional systolic arrays to enable scalable convolution operations by adjusting multiplier and adder bitwidths in powers of two. Multiple multiplier types are integrated to facilitate performance evaluation across different SIMD configurations, supporting both approximate and accurate multiply-accumulate operations. Configurable pipeline stages are optimized using Pareto analysis to balance computational accuracy and hardware resource utilization. Unlike conventional fixed-function designs, full flexibility in multiplier selection is provided, ensuring efficient utilization of processing elements across all operating modes. During accumulation, adders are reused, and CORDIC-based partial product generation is employed to further improve hardware efficiency, scalability, and adaptability for Edge-AI applications.

### 6.3.1 Memory Bank Architecture and System Integration

The system architecture integrates three flip-flop-based memory banks to support implementation and functional validation. These memory banks include the input feature map memory (if_mem), weight memory (wt_mem), and output feature map memory. Each memory consists of four sub-banks, each with a 32-bit data width. Precision-specific access is used to optimise memory utilization. For 8-bit precision, all four sub-banks are accessed simultaneously, yielding $\lceil 128 \rceil$ bits of data that contain

sixteen 8-bit operands. For 16-bit precision, two sub-banks are accessed to retrieve 64 bits of data, corresponding to four 16-bit operands. For 32-bit precision, a single sub-bank is accessed, providing one 32-bit operand. This precision-aware memory access mechanism is uniformly applied across all three memory banks for both input reads and output writes, ensuring efficient data handling and resource utilization.

### 6.3.2 Precision-Aware Operand Handling in Symmetric and Asymmetric SIMD Modes

The proposed architecture supports a variable number of MAC operations depending on the operand precision, providing 16 MACs for 8-bit operands, 4 MACs for 16-bit operands, and a single MAC for symmetric 32-bit operands. For asymmetric precision scenarios, the architecture enables concurrent processing of one 32-bit operand with four 8-bit operands, thereby supporting four parallel 32×8 MAC operations. Data handling is optimized according to precision-specific widths, requiring $\lceil 128 \rceil$ bits for 8-bit, $\lceil 64 \rceil$ bits for 16-bit, and $\lceil 32 \rceil$ bits for 32-bit computations.

SIMD computation across multiple precision levels is enabled through the SIMD_Low configuration, which scales in parallel with the SIMD_High design. The SIMD_Low mode supports efficient execution of 4-, 8-, and 16-bit operations by employing a compact 4-bit multiplier array optimized for partial product generation. Both SIMD_Low and SIMD_High configurations are evaluated, with performance analysis primarily focused on the SIMD_High configuration.

The architecture integrates fundamental logic components, including shifters, adders, and multiplexers, together with quantization and rounding mechanisms within the computation pipeline. These enhancements enable efficient utilization of adder trees and improve overall hardware efficiency. Precision-aware quantization reduces area and power consumption while maintaining high throughput and minimal accuracy degradation. Dynamic adaptation to operand data types and precision ensures scalable and efficient DNN inference. To preserve output bitwidth consistency, precision-aware rounding is incorporated into the PE design.

The initial stage of the processing element (PE) is responsible for input pre-processing prior to the multiplier array, with the objective of ensuring full utilization of computational resources across all supported precision modes. This stage adapts the input data representation and alignment based on the selected precision configuration, thereby preventing resource under-utilization and enabling higher throughput, particularly in low-precision operating modes.

Following pre-processing, the signed-magnitude operands are distributed to an array of 8-bit multipliers, which generate partial products for subsequent accumulation. The accumulation process is managed by a set of dedicated adder stages, denoted as $S_0$, $S_1$, and $S_2$, each of which is selectively activated according to the active precision mode. In the 8-bit configuration (mode 00), partial products are accumulated through direct addition, maximizing efficiency and minimizing latency. For higher-precision configurations, including 16-bit (mode 01), 32-bit (mode 10), and asymmetric precision (mode 11), the architecture employs shifted partial sums to correctly align the partial products prior to accumulation. This approach enables accurate multi-precision accumulation while maintaining a uniform multiplier structure. The overall accumulation strategy, illustrated in Fig. 6.2, achieves resource-efficient operation and scalable performance across a wide range of precision modes.

Rounding during partial product accumulation is detailed in Section 6.3.3, where output bitwidth consistency is preserved to ensure compatibility with post-processing stages. Control signals such as `valid_in` and `valid_out` regulate data validity, enabling or suspending computation as needed. The `accum` signal remains asserted throughout MAC operations, while bias values are preloaded into the accumulator to minimize latency. Inputs and weights are retrieved from 128-bit buffers and pre-processed according to the active precision mode: mode `00` fetches 128 bits, mode `01` fetches the lower 64 bits, mode `10` fetches the lower 32 bits, and mode `11` fetches 32 LSBs from the input buffer and 8 LSBs from the weight buffer. Operands are distributed across sixteen 8-bit multipliers to ensure full resource utilization across all precision configurations. The MAC unit supports sixteen `8×8`, four `16×16`, one `32×32`, or four `32×8` operations per cycle. A multiplexer and left-shifter are employed

Table 6.2: Comparison of Hardware Performance Parameters of State-of-the-Art Designs and the Proposed Work

| Design | Arithmetic Format | Asymmetric Precision | SIMD Support (#Parallel Ops) | Adder Reuse |
|---|---|---|---|---|
| Huang et al. '22 [62] | INT | ✗ | 2 / 4 / 8 (4 / 2 / 1) | ✓ |
| Shin et al. '17 [74] | FxP | ✓ | 2 / 4 / 8 (1 / – / – ) | ✗ |
| Li et al. '22 [130] | FP | ✗ | 16 / 32 / 64 (16 / 4 / 1 ) | ✗ |
| Li et al. '22 [91] | FxP | ✓ | 2 / 4 / 8 (4 / 2 / 1 ) | ✗ |
| Neda et al. '22 [63] | INT | ✗ | 4 / 8 (8 / 2 ) | ✗ |
| Raut et al. '24 [64] | FxP | ✗ | 8 / 16 ( 4 / 1) | ✗ |
| Proposed : C-SIMD_Low | FxP | ✓ | 4 / 8 / 16 (16 / 4 / 1) | ✓ |
| Proposed : C-SIMD_High | FxP | ✓ | 8 / 16 / 32 (16 / 4 / 1) | ✓ |

to align partial products appropriately for multi-precision accumulation.

While symmetric multi-precision support significantly improves architectural adaptability, it is not sufficient by itself to satisfy the requirements of diverse real-world AI workloads [139]. To overcome this limitation, the proposed architecture extends the symmetric SIMD framework to explicitly support asymmetric precision modes. In particular, mode `11` enables mixed-precision operation by combining the highest and lowest supported bitwidths within the same processing element. Under this mode, the SIMD_High and SIMD_Low configurations support four `32×8` and four `16×4` MAC operations, respectively.

This capability facilitates efficient mixed-precision computation for weights and activations in DNNs, thereby enhancing both flexibility and energy efficiency. In conventional accelerator architectures, asymmetric operand bitwidths often lead to poor hardware utilization, as datapaths are dimensioned according to the widest operand, leaving narrower-precision lanes idle. In contrast, the proposed design maintains full (100%) computational resource utilization even under asymmetric precision configurations, directly translating into higher effective throughput.

Beyond improved utilization, the proposed C-SIMD architecture incorporates explicit asymmetric-precision support, strategic reuse of adders along the accumulation path, and scalable wide-SIMD operation. These features collectively enable superior performance and hardware efficiency compared to prior designs, many of which lack either fine-grained precision flexibility or effective resource reuse. A comparative eval-

uation against state-of-the-art accelerators is provided in Table 6.2. Although the current implementation demonstrates a single operand pair, the methodology readily extends to other bit-width combinations as needed. Besides, Table 6.1 summarizes the supported operand sizes, precision modes, and per-cycle throughput: $i$ and $w$ denote the input and weight bit widths, respectively, and # of operations reports the number of MACs executed per PE per cycle for Conventional SIMD and proposed C-SIMD architectures.

### 6.3.3 CORDIC Integration and Precision-Aware Rounding for Partial Product Generation

This chapter presents an optimized CORDIC-based multiplication architecture integrated within SIMD processing elements to enable efficient partial product generation. The proposed approach leverages the inherent shift-and-add efficiency of CORDIC operations to realize compact and scalable multiplier designs, making them particularly well suited for low-precision arithmetic in SIMD-based MAC units. To ensure numerical consistency across different precision modes, a precision-aware rounding mechanism is incorporated, allowing, for instance, 8-bit multiplications to produce 8-bit outputs. This capability enables the deployment of low-precision adder trees, thereby substantially reducing hardware resource utilization.

The proposed architecture supports both fixed-point and integer multiplication within SIMD MAC operations. Furthermore, a comprehensive design space exploration is performed to assess the trade-offs between iterative and pipelined CORDIC implementations, providing insights into performance, area, and efficiency across different architectural choices.

CORDIC architectures are widely regarded as area- and power-efficient alternatives for realizing arithmetic operations in DNN accelerators; however, their iterative nature inherently limits throughput [49]. The CORDIC algorithm computes pseudo-rotations to approximate vector rotations in circular, linear, and hyperbolic coordinate systems using a unified computational framework [145, 149]. This generality allows a wide

range of arithmetic functions to be implemented using only shift, add/subtract, and simple control logic, thereby avoiding costly multipliers. To address the throughput bottleneck associated with sequential iterations, pipelined CORDIC architectures have been proposed, enabling higher operating frequencies and improved throughput while maintaining low hardware complexity [55].

The modified CORDIC-based partial product generation architecture adopted in this chapter is illustrated in Fig. 6.3. In this architecture, $X_n$ denotes the input activation, $Y_n$ represents the accumulated partial sum (including the bias term), and $W_n$ corresponds to the residual weight at iteration $n$. Each CORDIC iteration processes one binary digit of the weight operand, such that the residual weight $W_n$ is progressively driven toward zero. Concurrently, a scaled version of the input activation $X_n$ is conditionally accumulated into $Y_n$, ensuring that the contribution of each weight bit is accurately reflected in the final result.

Multiplication between $X_n$ and $W_n$ is realized using a linear-mode CORDIC formulation based on an iterative shift-and-add/subtract process. In each iteration, the algorithm examines the sign of $W_n$ and conditionally adds or subtracts an appropriately shifted version of $X_n$ to or from $Y_n$. The shift amount corresponds to the binary position of the currently processed weight bit, enabling the accumulation of partial products in a bit-serial manner. This iterative process converges after a finite number of iterations equal to the operand bitwidth, as formalized in Eq. 6.1. At convergence, the final value of $Y_n$ represents the product of the input activation and the weight, with the bias term fully accumulated.

$$X_n = X_0 \;\Rightarrow\; X_{\text{out}} = X_{\text{in}}, \tag{6.1a}$$

$$Y_n = Y_0 + X_0 \cdot W_0 \;\Rightarrow\; Y_{\text{out}} = Y_{\text{in}} + X_{\text{in}} \cdot W_{\text{in}}, \tag{6.1b}$$

$$W_n = W_0 \;\Rightarrow\; W_{\text{out}} = 0. \tag{6.1c}$$

While this approach significantly reduces multiplier complexity, conventional CORDIC-based multiplication still relies on a barrel shifter to generate the shifted

versions of $X_{\text{in}}$ required for evaluating $X_{\text{in}} \times W_{\text{in}}$. The barrel shifter constitutes a major contributor to area, power consumption, and critical-path delay, particularly at higher precision levels. These limitations motivate the architectural refinements introduced in the proposed design, which aim to retain the computational efficiency of CORDIC while reducing hardware overhead and improving adaptability to the diverse precision requirements of SIMD-based DNN accelerators. As a result, the proposed architecture achieves a more favorable trade-off between accuracy, latency, and area compared to conventional CORDIC-based designs.

The iterative computation proceeds until convergence, at which point the final output $Y_n$ corresponds to the desired multiply-accumulate result. In the proposed implementation, the stopping criterion is explicitly defined as either (i) the residual weight $W_n$ converging to zero, or (ii) a fixed maximum of $n$ iterations equal to the operand bitwidth, whichever occurs first. This bounded iteration ensures deterministic latency, which is essential for predictable timing behavior and facilitates efficient and reliable pipeline scheduling in hardware architectures. Unlike conventional linear-mode CORDIC formulations that apply an $n$-bit right shift to the accumulated sum $Y_n$ at each micro-rotation, the proposed architecture deliberately avoids any shift operation on $Y_n$. Instead, the required scaling is implicitly realized through progressive 1-bit right shifts applied exclusively to $X_n$ across successive iterations. As a result, the update of $Y_{n+1}$ depends solely on the accumulated value unshifted $Y_n$ and the appropriately scaled $X_n$, thus preserving numerical stability while eliminating the need for costly barrel shifters. Under this formulation, the multiplication is decomposed into a sequence of micro-rotations that update the input, accumulation, and residual weight variables at each iteration. The direction of each micro-rotation is governed by the control variable $\delta_n \in \{+1, -1\}$, which determines whether $X_n$ is added to or subtracted from $Y_n$, and is explicitly defined as

$$\delta_n = \begin{cases} +1, & \text{if } W_n \geq 0, \\ -1, & \text{if } W_n < 0. \end{cases}$$

The iterative updates of the CORDIC-based multiplication are formally expressed as

$$X_{n+1} = X_n \cdot 2^{-1}, \tag{6.2a}$$

$$Y_{n+1} = Y_n + \delta_n \cdot X_n, \tag{6.2b}$$

$$W_{n+1} = W_n - \delta_n \cdot 2^{-n}. \tag{6.2c}$$

Equations 6.2(a)–(c) describe the convergent behavior of the proposed CORDIC-based multiplication process. Here, $\delta_n$ denotes the direction of the $n^{\text{th}}$ micro-rotation and controls the add/subtract operation applied to $Y_n$. The iterations terminate when either the residual weight converges to zero or the predefined iteration limit—equal to the operand bitwidth—is reached. Under these conditions, $Y_n$ converges to the product $X_{\text{in}} \cdot W_{\text{in}}$ with the bias term $Y_{\text{in}}$ accumulated. In particular, no shift operation is applied to $Y_n$ during the update of $Y_{n+1}$, which distinguishes the proposed formulation from conventional CORDIC-based multipliers and directly contributes to reduced hardware complexity and improved timing characteristics. Each iteration truncates the least significant bit (LSB) of $X_n$ due to the 1-bit right shift, introducing a controlled and bounded loss of precision. This truncation is intentional and forms the basis of the proposed precision-aware rounding strategy, enabling fixed output bitwidths across SIMD lanes without additional rounding hardware.

The design exploits the shift–add nature of the CORDIC algorithm, eliminating the need for hardware multipliers and thereby reducing area and power overhead. Each pipeline stage performs one CORDIC iteration, enabling high-throughput operation through parallelized stage-wise computation. Fig. 6.3 presents the proposed hardware-optimized n-stage pipelined CORDIC architecture tailored for fixed-point arithmetic in compute-efficient accelerators.

The computational behavior of the first pipeline stage using fixed-point $\langle 8, 6 \rangle$ precision is detailed in Table 6.3. In this stage, the $X$ operand is right-shifted by one bit, resulting in a controlled loss of the least significant bit. The $Y$ operand is conditionally updated using addition based on the direction control signal $d_n$, while the angle accumulator $W$ is modified by subtracting a power-of-two scaled term. This

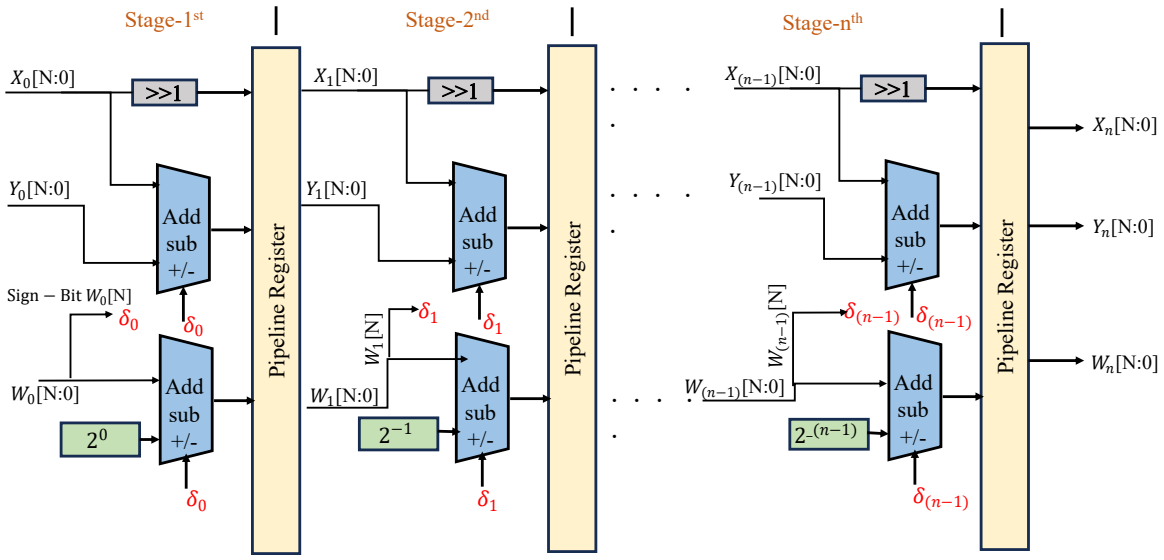



Figure 6.3: Hardware-Optimized n-stage pipeline CORDIC Architecture Implementation.

Table 6.3: Computation Process in the first stage of the Pipelined Architecture (Fixed-Point $\langle 8, 6\rangle$ precision).

| **Variable** | **Initial** | **1st stage (for $n = 0$)** |
|---|---|---|
| $X_{n+1} = X_n \times 2^{-1}$ | $X_0 = 000.101100_2$ (Bin)<br>$= 0.68750_{10}$ (Dec) | $X_1 = 000.0101100_2$ (Bin)<br>$= 000.010110_2$ (Bin)<br>(Right shift by 1-bit, lost bit) |
| $Y_{n+1} = Y_n + d_n \times X_n$ | $Y_0 = 000.001100_2$ (Bin)<br>$= 0.18750_{10}$ (Dec) | $Y_1 = 000.001100_2 + d_n \times 000.101100_2$<br>$= 000.111000_2$ (Bin) |
| $W_{n+1} = W_n - d_n \times 2^{-n}$ | $W_0 = 000.111010_2$ (Bin)<br>$= 0.90625_{10}$ (Dec) | $W_1 = 000.111010_2 - d_n \times 01.000000_2$<br>$= 111111010_2$ (Bin) |

iterative shift–add operation forms the fundamental building block of the proposed CORDIC-based computation, ensuring deterministic precision handling.

An example of signed 8-bit fixed-point multiplication using the CORDIC algorithm is shown in Fig. 6.4. The illustrated computation values are sampled from trained data, demonstrating the applicability of the approach to practical DNN workloads. The multiplication is decomposed into successive bit-wise shifts and accumulations, which improves hardware efficiency while maintaining numerical stability. Fig. 6.5 illustrates the rounding and truncation flow employed in the proposed architecture. At

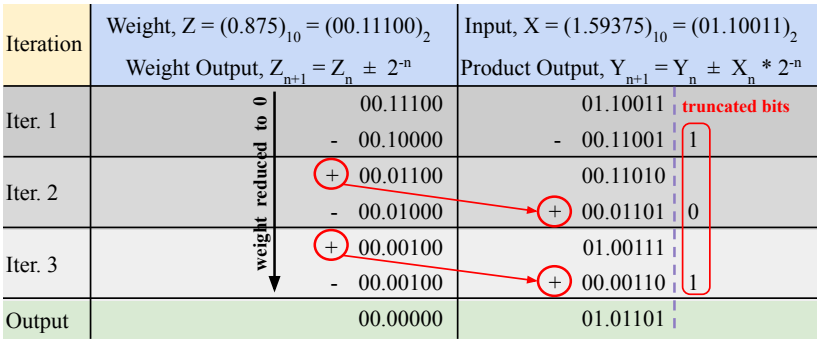



Figure 6.4: Example multiplication for the CORDIC algorithm with a Signed 8-bit Fixed-Point arithmetic scheme. Sampled calculation values have been taken from one of the trained data samples.

each iteration, the input feature map is multiplied with the corresponding weight bit using shift operations, and the weighted bit is discarded after accumulation. This iterative bit-level processing enables fine-grained precision control and supports efficient integration of the proposed CORDIC engine within DNN accelerators.

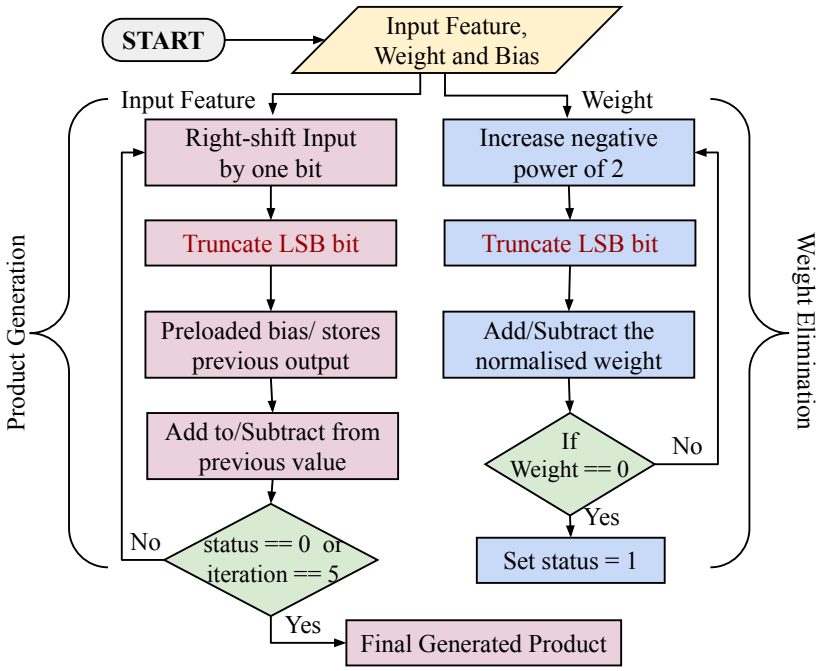



Figure 6.5: Flowchart of the rounding algorithm for the Proposed Efficient Computing Architecture. The input feature map is multiplied with weights using bitwise shifts at each binary position, with the corresponding weighted bit discarded in each iteration.

### 6.3.4 Performance-Aware Adder Reuse for Scalable C-SIMD MAC Operations

The proposed C-SIMD architecture exploits hardware reuse by sharing the same adder units for both shift-and-add and accumulation operations within the MAC units. This design maximizes the utilization of adder stages, enabling efficient pipeline scheduling and significantly enhancing throughput. As a result, the architecture achieves a performance improvement of up to 4× over prior designs, while supporting parallel computation across multiple data paths. An adder tree with precision-dependent bit-widths is employed to perform accumulation across different precision modes. In the conventional SIMD_High configuration, the hierarchy consists of eight 24-bit adders for segment S0, four 32-bit adders for S1, two 48-bit adders for S2, and a single 64-bit adder in the final stage, culminating in a 72-bit quire accumulator. This accumulator supports up to 256 accumulations in 32-bit precision mode, with proportionally scaled accumulation capacity for lower precisions. In contrast, the proposed C-SIMD PE adopts a more hardware-efficient design, deploying eight 16-bit, four 24-bit, two 40-bit and one 56-bit adders, followed by a 64-bit accumulator. The significant reduction in adder bit-widths compared to state-of-the-art SIMD architectures is attributed to the use of low-bitwidth products generated by the CORDIC-based multiplier, leading to improved area and power efficiency. The quire register in both designs implements precision-aware rounding mechanisms to ensure consistent input-output behavior across all operating modes. The final output is validated through the valid_out signal and delivered via PE_out. For lower precision operations, the conventional SIMD_Low configuration utilizes narrower adders: 13-bit (S0), 22-bit (S1), and 31-bit (S2), followed by a 40-bit ADD_D0 stage and a 44-bit accumulator. In the proposed low-precision variant of C-SIMD, the bit-widths of the adders are further reduced in accordance with the lower precision demands, leveraging the CORDIC-generated low-bitwidth products.

The unified adder stage configuration depicted in Fig.6.6 depicts the internal operation of the proposed C-SIMD PE across all precision modes. The architecture

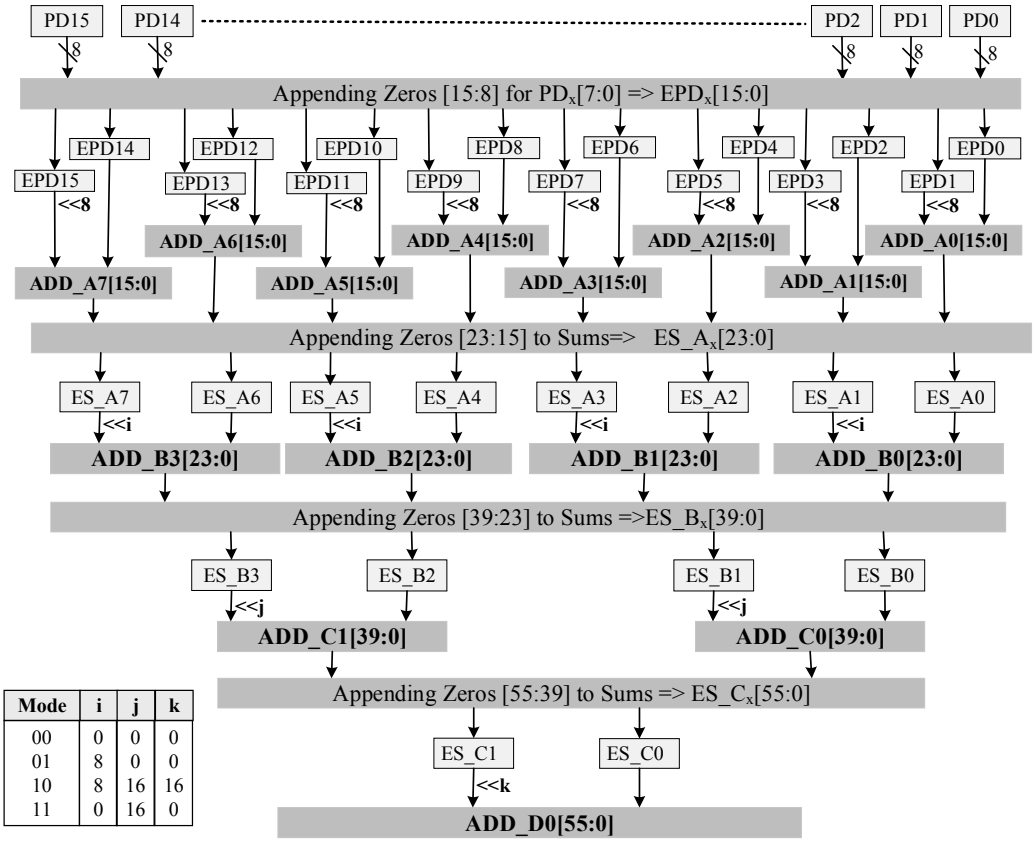



Figure 6.6: Proposed adder tree in the C-SIMD PE supporting all precision modes (00, 01, 10, 11) for symmetric and asymmetric computations. Partial product (PD) shifts are followed by addition and quire-based accumulation for efficient multi-precision MAC execution. Indices $i$, $j$, and $k$ indicate shift stages.

employs a shared adder tree structure that dynamically adapts to precision requirements, enabling efficient hardware reuse. In Mode`00`, partial products generated by the CORDIC-based SIMD multiplier are accumulated through 16-bit and 24-bit adders, with the 24-bit adders functioning as primary accumulation units. For Mode `01`, 8-bit shifted partial products are summed using a hierarchical adder network, wherein alternate outputs are shifted by 8 bits to preserve numerical fidelity prior to accumulation. Mode `10` activates all 15 adders in a pipelined shift-and-add configuration; here, adder D0 interfaces with the quire register to facilitate iterative accumulation. While Stage-I remains structurally invariant between Modes `01` and `10`, the shifter configuration in Stage-II is adapted to accommodate the differing shift requirements. In asymmetric Mode `11`, 32-bit partial products are initially processed by dedicated 32-bit adders

and subsequently accumulated using 40-bit adders. The 16-bit adders are reused across precision modes. In low-precision settings, they perform accumulation, while in higher-precision operations, they are allocated for shift-and-add computations.

This resource-efficient reuse of adder stages demonstrates the novelty of the proposed architecture, which achieves enhanced parallelism, reduced area overhead, and improved performance across a range of DNN precision requirements. The ADD_C0 and ADD_C1 adders generate the final output for Modes `01` and `11`, while a dedicated ADD_D0 adder produces the final result for Mode `10`. A similar adder configuration is employed in the C-SIMD_Low variant, as discussed previously. The proposed architecture enables complete reuse of adder resources, achieving 100% utilization across all precision modes. The shift-and-add mechanism inherently supports both partial product addition and accumulation, thereby eliminating the need for separate adder stages. This design approach leads to significant reductions in resource overhead and power consumption. The quire accumulator is dimensioned to handle up to 256 accumulation cycles for maximum bitwidth operands. Specifically, the C-SIMD_High configuration supports up to 256 32-bit operations, 1024 16-bit operations, 4096 8-bit operations, and 1024 asymmetric operations involving 32×8-bit pairs.

## 6.4 Results and Discussion

The proposed MAC architecture supports both precise and approximate computation by configuring the number of pipeline stages within the partial product generation block. In the CORDIC-based design, increasing the pipeline depth improves computational accuracy, with an eight-stage pipeline employed to achieve maximum precision. For resource-constrained Edge-AI applications, a Pareto analysis is performed to systematically explore the trade-off between pipeline depth, hardware cost, and inference accuracy in DNN workloads.

To comprehensively evaluate these trade-offs, both software-level and hardware-level assessments are conducted. At the software level, a Python-based emulator of the proposed processing element is developed to analyze inference accuracy across

a range of DNN models and benchmark datasets under different pipeline configurations. At the hardware level, performance and implementation metrics are evaluated across multiple technology nodes to assess scalability and efficiency. A conventional multiplier-based partial product generation approach is used as the baseline and is compared against the optimized CORDIC-based architecture proposed in this chapter, enabling a quantitative evaluation of accuracy, performance, and hardware efficiency.

### 6.4.1 Pareto Optimality and Evaluation of Accuracy Retention Across 8/16/32-bit Precision

Software evaluation was conducted using a Python-based emulator of the proposed C-SIMD processing element (PE) to assess inference accuracy across multiple precision modes, namely 8-, 16-, and 32-bit. Experiments were performed on standard benchmark datasets, including MNIST, CIFAR-10, CIFAR-100, and ImageNet-1000, using representative DNN models such as LeNet-5, AlexNet, VGG-16, ResNet-18, and CaffeNet. These evaluations validate the effectiveness of the proposed architecture across a diverse set of network topologies and workload characteristics.

The C-SIMD PE was evaluated under both integer and dynamic fixed-point computation models within the multi-precision framework. A 32-bit floating-point baseline model was used as a reference [31]. For the MNIST dataset, the baseline achieves an inference accuracy of 98.46%. In comparison, the proposed architecture attains accuracies of 98.06%, 98.01%, and 97.97% for the 32-bit, 16-bit, and 8-bit precision modes, respectively. These results demonstrate that the proposed multi-precision C-SIMD architecture incurs only marginal accuracy degradation when operating at reduced precision levels.

Moreover, the percentage computational error as a function of the number of CORDIC pipeline stages was evaluated. This analysis was performed for the 8-bit precision mode, as the CORDIC engine is employed for partial product generation in 8-bit computations. Accuracy degradation was observed across different pipeline depths, as illustrated in Fig. 6.7.

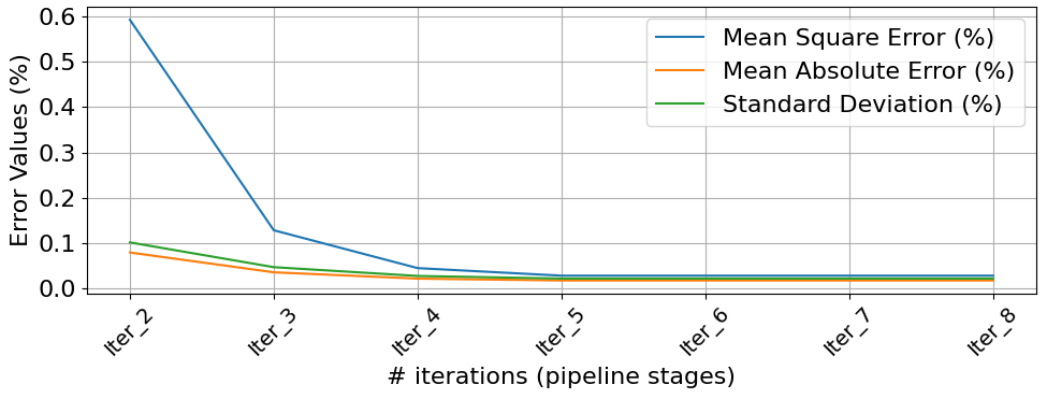
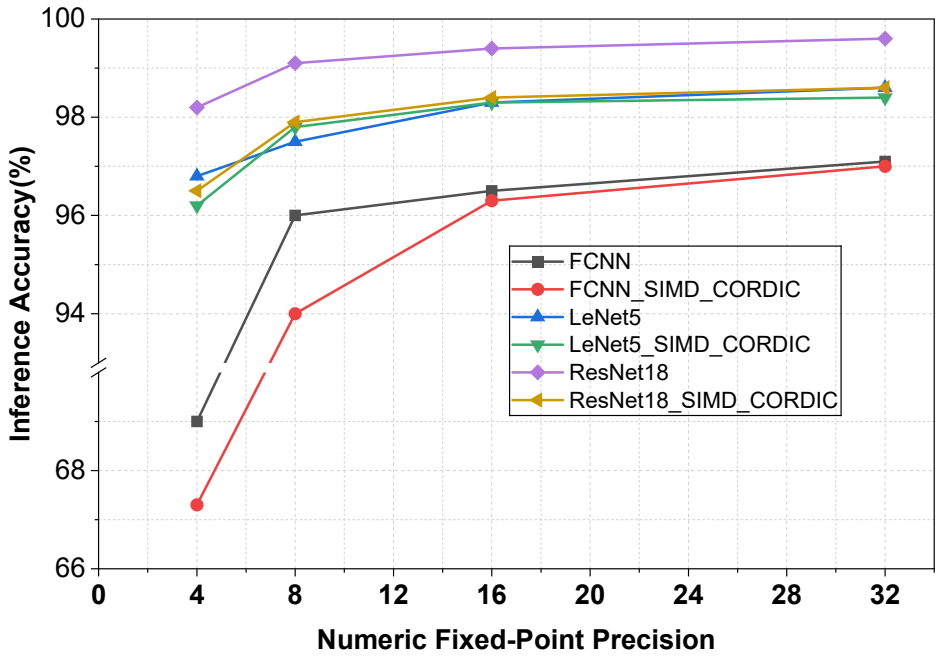



Figure 6.7: The Error Evaluation distribution of sfxpt<*8,5*> across various iterations for Pareto analysis and Pareto-Point Extraction.

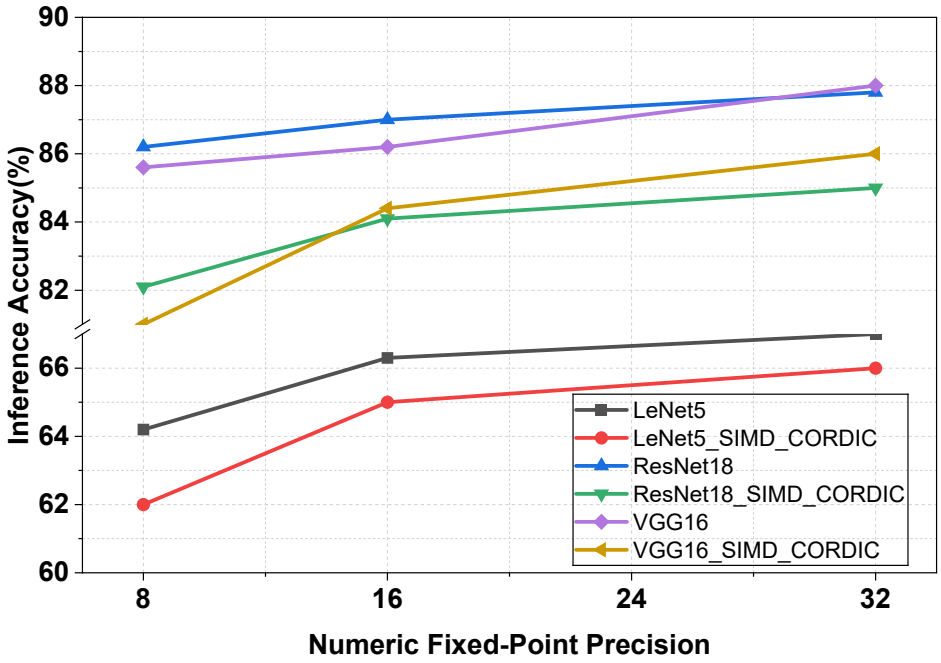
The results indicate that the computational error decreases significantly beyond four pipeline stages and becomes negligible with five pipeline stages. Based on this observation, a five-stage CORDIC pipeline is selected for subsequent evaluations to achieve improved numerical accuracy under the signed fixed-point(sfxpt) configuration $\texttt{sfxpt}\langle 8, 5\rangle$. Using this pipeline depth, the inference accuracy of baseline DNN models is evaluated.

From this empirical analysis, the five-stage CORDIC pipeline is identified as the optimal configuration for the proposed C-SIMD inference engine, offering a favourable trade-off between inference accuracy and hardware resource utilisation. This choice enables a balanced design point that enhances overall system performance while preserving hardware efficiency.

Inference accuracy across different precision modes of the proposed C-SIMD PE architecture is evaluated using multiple DNN models and benchmark datasets. For the CIFAR-10 dataset, the baseline accuracy is 76.75%. The proposed architecture achieves identical accuracy in the 32-bit mode (76.75%), while attaining 76.73% and 73.84% in the 16-bit and 8-bit modes, respectively, indicating negligible degradation at higher precision levels. Similarly, for the VGG-16 model evaluated on CIFAR-10, the baseline accuracy is 81.43%. Under the proposed C-SIMD PE, accuracies of 82.27% (32-bit), 82.24% (16-bit), and 80.01% (8-bit) are achieved, demonstrating robust performance across precision modes.




(a) MNIST dataset: 4-, 8-, 16-, and 32-bit




(b) CIFAR-10 dataset: 8-, 16-, and 32-bit

Figure 6.8: Inference Accuracy of the Proposed C-SIMD and Conventional Architectures on (a) MNIST and (b) CIFAR-10 datasets using various DNN models with Signed Fixed-Point Precision.

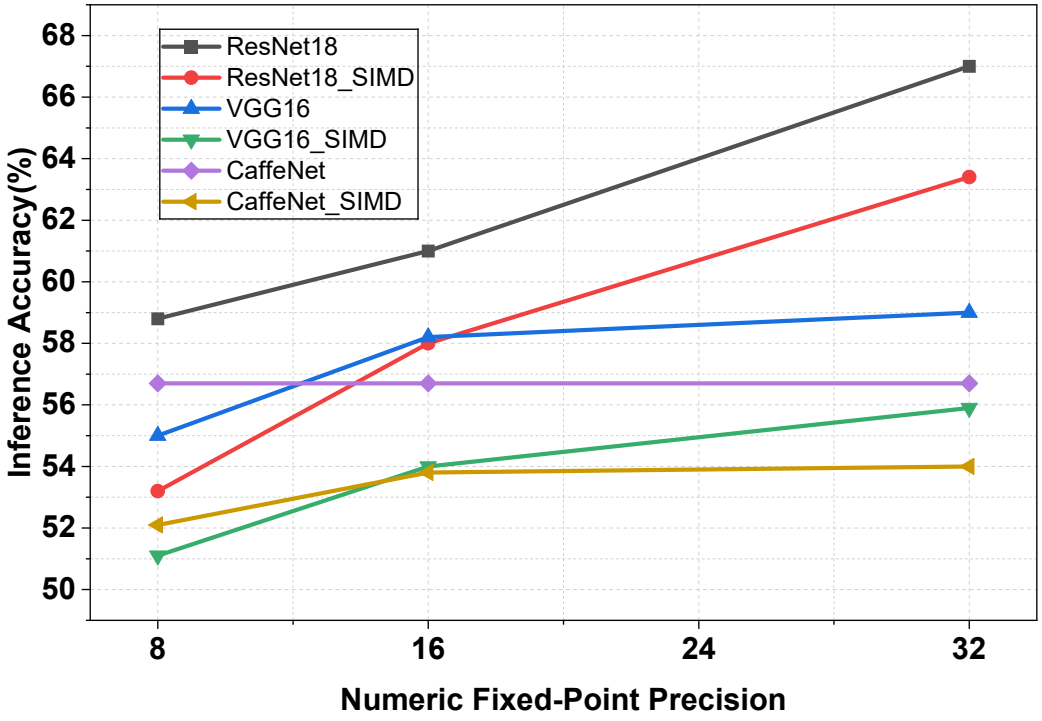



Figure 6.9: Inference Accuracy of the Proposed C-SIMD and Conventional Architectures on CIFAR-100/ImageNet datasets using various DNN models with Signed Fixed-Point Precision.

The observed accuracy degradation in the 8-bit configuration—approximately 2% for CIFAR-10 and up to 3.5% for ImageNet—can be primarily attributed to quantization effects, including rounding-to-nearest (ties-to-even) behavior in accordance with IEEE-754 standards.

It is important to note that these accuracy losses represent worst-case scenarios and can be further mitigated through optimized rounding and truncation schemes. Notably, AlexNet exhibits only about a 2% accuracy reduction in the 8-bit mode, while LeNet-5 shows a minimal degradation of 0.49%, underscoring the inherent resilience of widely used DNN architectures to reduced numerical precision.

Overall, these results demonstrate that the proposed C-SIMD architecture achieves near-baseline inference accuracy across all evaluated models while operating at substantially lower bit widths. This minimal accuracy trade-off, combined with improved energy efficiency and reduced hardware complexity, highlights the effectiveness and practicality of the proposed design for resource-constrained Edge-AI applications.

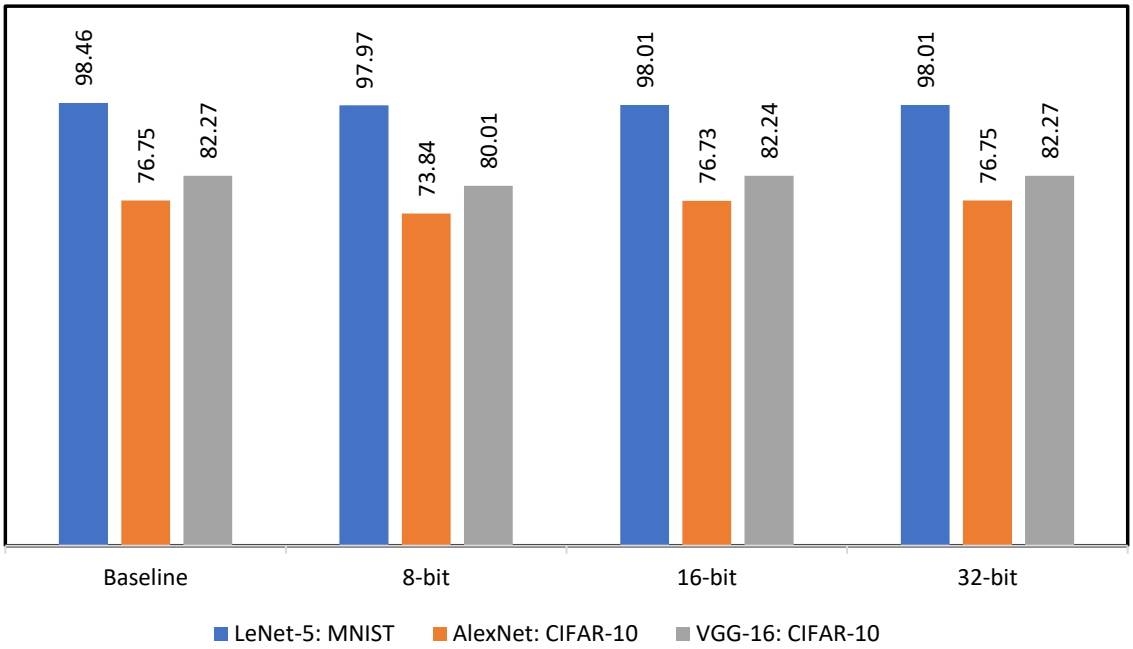



Figure 6.10: Comparison of Baseline Inference Accuracy [31] of DNN models (LeNet-5, AlexNet, and VGG-16) on various datasets (MNIST and CIFAR-10) with that of the Proposed C-SIMD PE architecture using different bit-precisions.

For the MNIST dataset, the inference accuracy of the proposed C-SIMD architecture is illustrated in Fig.6.8(a). The results demonstrate that the accuracy deviation across all evaluated DNN models is minimal, remaining below 2%, which underscores the effectiveness of the proposed approach. Recognizing that many DNNs primarily focus on feature extraction, the applicability of the C-SIMD architecture for image classification was further validated by analyzing a fully connected neural network (FCNN) with a 196:64:32:32:10 layer configuration[55]. In this case, the inference accuracy at 32-bit precision closely matches the results reported in[150], confirming the robustness of the architecture. Inference accuracies for the CIFAR-10 dataset, evaluated on LeNet-5, VGG-16, and ResNet-18 models, are presented in Fig.6.8(b). As expected, LeNet-5 exhibits relatively lower accuracy on this more complex dataset for both the proposed and conventional architectures, yet the difference between them remains small (approximately 2%). The proposed architecture's accuracy for VGG-16 deviates by less than 4% compared to[150], while for ResNet-18, the deviation is under 3%. Importantly, these minor accuracy compromises are offset by significant hardware

advantages, including an approximately fourfold reduction in LUT utilization.

Similarly, inference accuracies on the CIFAR-100 dataset, evaluated using advanced DNN models such as CaffeNet, ResNet-18, and VGG-16, are illustrated in Fig. 6.9 and exhibit trends consistent with those observed across other datasets. In particular, accuracy degradation progressively diminishes as the precision increases, a behavior that is also evident for the proposed `SIMD_High` architecture. This trend indicates that higher-precision implementations enable inference accuracy to converge toward the baseline results reported in [150].

A comparative analysis across multiple precision levels, using an $N$-bit reference and the corresponding proposed $N$-bit configuration, is presented in Figures 6.8 and 6.9. To further elucidate the impact of reduced precision, the achieved inference accuracy is separately compared against the baseline 32-bit model in Fig. 6.10. This figure also summarizes inference accuracy results for various DNN models across all precision modes supported by the proposed processing element (PE).

Overall, the proposed C-SIMD architecture demonstrates negligible accuracy loss even when evaluated on complex datasets and diverse DNN models, while simultaneously offering substantial advantages in reduced hardware resource utilization, improved throughput, and flexible variable-precision support. These characteristics underscore the strong suitability of the proposed design for resource-constrained applications that demand adaptable precision without compromising inference accuracy.

### 6.4.2 Hardware Resources Utilization and Comparison

The proposed model is implemented and evaluated using both FPGA hardware and a standard ASIC design flow to demonstrate its practicality and scalability. The C-SIMD processing element (PE) is integrated within a two-dimensional systolic array, enabling highly parallel execution of DNN workloads. Functional correctness and performance are thoroughly evaluated across multiple precision configurations. For physical design analysis and fair comparison, all evaluations are conducted using a standalone PE configuration. As discussed in Section 6.4.1, a Pareto-based design space exploration is employed to determine the optimal pipeline depth that balances

computational accuracy, throughput, and hardware efficiency.

The hardware design is deployed on a Virtex-7 VC707 FPGA SoC(System on Chip), where key physical metrics—including resource utilization, critical path delay, power consumption, throughput, and energy efficiency—are measured across different precision modes. In addition, the architecture is synthesized using a 65 nm CMOS technology to assess ASIC-level performance characteristics, providing insights into the scalability and efficiency of the proposed C-SIMD architecture across different implementation platforms.

This subsection presents a detailed evaluation of the proposed C-SIMD processing element (PE). To quantify performance gains relative to a conventional baseline—which underutilizes hardware resources— the following number of MAC operations per cycle have been considered: 4 for 8-bit, 2 for 16-bit, and 1 for 32-bit operations. The proposed adder tree architecture maximizes resource utilization, delivering up to 4× and 2× throughput improvements for 8-bit and 16-bit operations, respectively. Additionally, the design supports asymmetric precision modes, enabling flexible mixed-precision computations across different DNN layers. Pipeline stage analysis has been performed for the partial product generation logic with four, five, and eight stages. For consistent FPGA and ASIC comparisons, results are reported using four pipeline stages for the SIMD_Low and five for the SIMD_High configurations, corresponding to the optimal design points identified via Pareto analysis. These configurations ensure an effective balance between throughput, resource efficiency, and computational accuracy.

The FPGA implementation results, summarized in Table 6.4, demonstrate the efficiency and versatility of the proposed C-SIMD PE architectures. Although in C-SIMD_High PE, there is a modest increase of 381 LUTs compared to a conventional 32-bit MAC unit, this overhead is justified by the enhanced reconfigurability and resource optimisation offered by the proposed design. The reconfigurable C-SIMD_High PE supports a wide range of operations within the same hardware footprint, including one 32-bit MAC, four 16-bit MACs, sixteen 8-bit MACs, and four asymmetric 32×8 MAC operations and utilises 1805 LUTs. A non-SIMD architecture would re-

Table 6.4: Analyzing the Impact of Precision Scalability and Performance Metrics for State-of-the-Art Works and the Proposed SIMD PE Architecture.

| Bit Precision | LUT (17600) | FF (35200) | Critical Delay (ns) | Dy. Power (mW) |
|---|---|---|---|---|
| **4-bit MAC [119]** | 37 | 8 | 2.19 | 2 |
| **8-bit MAC [119]** | 86 | 16 | 2.36 | 6.6 |
| **8-bit Booth [28]** | 83 | 61 | 3.08 | - |
| **8-bit CORDIC [55]** | 54 | 88 | 1.52 | 6.6 |
| **16-bit MAC [119]** | 335 | 32 | 2.89 | 14.95 |
| **16-bit CORDIC [55]** | 95 | 162 | 2.12 | 12.21 |
| **32-bit MAC [119]** | 1424 | 208 | 6.51 | 22 |
| **Adaptive-Precision SIMD PE with Conventional Multiplier** | | | | |
| **Conv. MULT SIMD_Low** | 1279 | 576 | 2.19 | 14 |
| **Conv. MULT SIMD_High** | 3010 | 1121 | 5.25 | 35 |
| **Proposed Adaptive-Precision C-SIMD PE with CORDIC Multiplier** | | | | |
| **Exact C-SIMD_Low (4-stage)** | 948 | 745 | 4.52 | 12 |
| **Approx C-SIMD_High (4-stage)** | 1805 | 1569 | 4.18 | 22 |
| **Approx C-SIMD_High (5-stage)** | 1902 | 1697 | 4.84 | 23 |
| **Exact C-SIMD_High (8-stage)** | 2222 | 2081 | 5.21 | 35 |

quire 5,604 LUTs (Conv_High MAC) to support equivalent 8/16/32-bit operations. Thus, the proposed C-SIMD_High PE achieves nearly a fourfold reduction in resource consumption. Similarly, the C-SIMD_Low PE supports one 16-bit MAC, four 8-bit MACs, sixteen 4-bit MACs, and four asymmetric 16×4 MAC operations and utilises 948 LUTs, whereas a non-SIMD architecture (Conv_Low MAC) would utilise 1607 LUTs to support equivalent 4/8/16-bit operations. Thus, the proposed C-SIMD_Low PE utilises 1.7× fewer LUTs as compared to the non-SIMD architecture. The conventional SIMD_High architecture utilises 3010 LUTs for processing four 8-bit MACs, two 16-bit MAC operations, and one 32-bit MAC operation. Similarly, the Conventional SIMD_Low architectures require 1279 LUTs for four 4-bit MAC operations, two 8-bit MAC operations and one 16-bit MAC operation. A detailed comparison of the LUT utilisation is illustrated in Fig. 6.11.

The proposed **C-SIMD_High** PE demonstrates significant hardware efficiency improvements over the conventional SIMD_High configuration. In the prioritized five-stage pipeline implementation, the C-SIMD_High PE achieves a 36.8% reduction in LUT utilization for 8-, 16-, and 32-bit SIMD operations, while also reducing power consumption compared to the baseline design. These results highlight the effectiveness of the proposed architecture in optimizing resource usage while maintaining high

Table 6.5: A Detailed Comparison of Hardware Performance parameters of State-of-the-Art Works with the Proposed Work.

| **Parameter** | **Miro et al.'20** [144] | **Ottavi et al.'23** [151] | **Shin et al.'17** [74] | **Li et al.'22** [130] | **Zhang et al.'19** [75] | **Mach et al.'21** [148] | **Raut et al.'24** [64] | **Moons et al.'17** [77] | **Conv. MULT** SIMD_Low | **Proposed** C-SIMD_low | **Conv. MULT** SIMD_High | **Proposed** C-SIMD_High |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Process (nm)** | 28 | 65 | 65 | 28 | 90 | 22 | – | 28 | 65 | 65 | 65 | 65 |
| Precision | 8,16,32 | 2,4,8,16 | 4, 8, 16 | 16, 32, 64 | 16, 32, 64 | 16, 32, 64 | 8, 16 | 4,8,16 | 4,8,16 | 4,8,16 | 8,16,32 | 8,16,32 |
| Arithmetic | FxP | FxP | FxP | FP | FP | FP | FxP | FxP | FxP | FxP | FxP | FxP |
| Voltage (V) | 0.9V | 1.2V | 1.1 | 1.0 | 1.0 | 1.0 | – | 0.65-1.05 | 1.0 | 1.0 | 1.0 | 1.0 |
| Pipelined | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes | No | Yes | No | Yes |
| Area ($mm^2$) | 4.5 (Die Area) | 10(Die Area) | 0.021 | 0.018 | 0.795 | 0.049 | – | 1.87 | 0.018 | 0.018 | 0.036 | 0.029 |
| Delay (ns) | – | – | | 2.96 | 4.5 | 3.25 | 5.1 | - | 2.53 | 2.27 | 5.19 | 3.85 |
| Freq. (MHz) | 350 | 205 | 200 | 1351 | 667 | 923 | 197.51 | 200 | 395 | 440 | 192 | 260 |
| Power (mW) | 96 | 23 | 279 (768 MACs) | 51.4 | 177 | – | – | – | 1.88 | 2.39 | 3.75 | 4.47 |
| Throughput | 0.375(32b) | 0.47(16b) | 0.39(16b) | 0.67(64b) | 0.67 (64b) | 0.923(64) | 0.19(16b) | 0.103(16b) | 0.395(16b) | 0.44(16b) | 0.192 (32b) | 0.26(32b) |
| (GOPS) | 0.75(16b) | 0.94(8b) | – | 2.7(32b) | 1.34(32b) | 1.85(32b) | 0.78(8b) | 0.21(8b) | 0.79(8b) | 1.76(8b) | 0.384(16b) | 1.04(16b) |
| | 1.5(8b) | 1.89(4b) | 1.56(4b) | 10.7 (16b) | 2.67 (16b) | 3.69(16b) | – | 0.41(4b) | 1.58 (4b) | 7.04(4b) | 0.768 (8b) | 4.16(8b) |
| Performance | 0.058(32b) | 0.15(16b) | – | 0.026(64b) | 0.023(64b) | 0.016(64b) | – | – | 0.21(16b) | 0.184(16b) | 0.051(32b) | 0.058(32b) |
| (TOPS/W) | 0.12(16b) | 0.31(8b) | – | 0.105(32b) | 0.046(32b) | 0.032(32b) | | – | 0.42(8b) | 0.736(8b) | 0.102(16b) | 0.232(16b) |
| | 0.23(8b) | 0.57(4b) | | 0.421 (16b) | 0.092(16b) | 0.064(16b) | – | – | 0.84 (4b) | 2.945(4b) | 0.204 (8b) | 0.928(8b) |

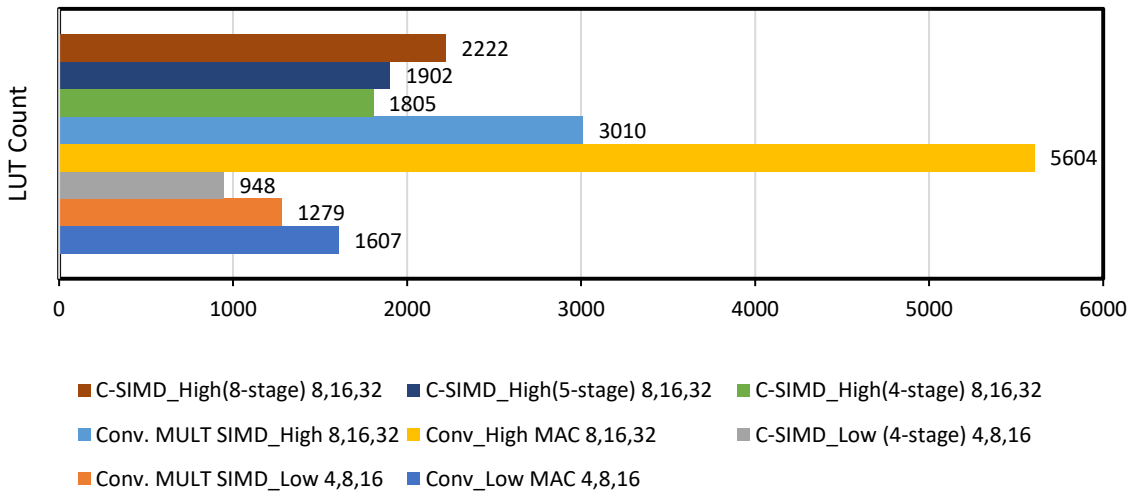



Figure 6.11: LUT count comparison of the proposed C-SIMD PE with the Conventional Architectures supporting similar precisions.

operational flexibility. The five-stage pipeline configuration strikes an optimal balance between throughput, accuracy, and hardware efficiency, making it the preferred design point for multi-precision DNN computations. For reference, four- and eight-stage pipeline implementations achieve LUT savings of 40.03% and 26.17%, respectively, but the five-stage design provides the best trade-off between performance and efficiency. This substantial reduction in LUT utilisation is accompanied by an increase in the number and diversity of supported symmetric and asymmetric MAC operations across multiple bitwidths. Moreover, the proposed C-SIMD PE architecture with five pipeline stages demonstrates a notable improvement in timing performance, reducing the critical path delay by 7.8% relative to the conventional SIMD_High architecture. These results highlight the architectural advantages of the C-SIMD PE in delivering scalable, efficient, and versatile computations for modern AI workloads by optimising resource usage while enhancing computational flexibility and throughput.

A comprehensive evaluation of the proposed **C-SIMD PE** architectures against prior state-of-the-art designs is presented in Table 6.5. The proposed design showcases its potential through significant improvements in area, delay, throughput, and performance efficiency across multiple precision modes. Compared with the design by Shin et al. [74], the proposed **C-SIMD_Low** design achieves a 14.29% reduction in silicon

area and a 120% increase in operating frequency. In terms of throughput, the proposed design delivers improvements of 1.28× and 3.51× for 16-bit and 4-bit precision modes, respectively. In comparison with the work by Moons et al. [77], the proposed **C-SIMD_Low** achieves a 120% increase in frequency. The proposed design delivers a throughput improvement of 16.17× at 4-bit, 7.38× at 8-bit, and 3.27× at 16-bit precision. Similarly, a throughput enhancement of 1.87× at 8-bit and 3.72× at 4-bit is observed in the proposed architecture in comparison to [151]. These results highlight the significant advancements in area efficiency, computational throughput, and energy-aware design, reinforcing the proposed architecture's effectiveness for ultra-low-power, high-throughput applications. Notably, post-layout simulation results are compared against post-silicon measurements, and switching power is also reported. Overall, a variation of approximately 10% is expected relative to the post-implementation results. In comparison to [144], the proposed design exhibits a significant improvement in throughput by a factor of 1.38× at 16-bit and approximately 2.77× at 8-bit precision with respect to **C-SIMD_High**. Furthermore, the performance efficiency of the proposed architecture is improved by 1.93× at 16-bit precision and by 4.03× at 8-bit precision, highlighting the performance efficiency improvement of the proposed architecture at low precisions. These enhancements underscore the architectural efficiency and suitability of the proposed design for precision-scalable, energy-efficient computing, particularly in edge AI applications.

Prior works [75, 64, 148, 130] are included in the table to contextualize the reported metrics and emphasize the relevance of the proposed solution in precision-scalable and energy-efficient computing. The proposed **C-SIMD_Low** architecture, supporting 4, 8, and 16-bit precision, demonstrates a 10.3% decrease in delay compared to the conventional SIMD_Low design. It achieves a significant throughput improvement of up to 3.45× at 4-bit precision and 1.23× at 8-bit precision. Furthermore, the performance efficiency, measured in TOPS/W, improves by 2.51× at 4-bit and 1.75× at 8-bit precision. Similarly, the **C-SIMD_High** architecture, designed for 8, 16, and 32-bit precision, achieves a 19.4% reduction in area and a 25.8% decrease in delay relative to the conventional SIMD_High implementation. It delivers a throughput im-

provement of 4.42× at 8-bit and 1.71× at 16-bit precision. The performance efficiency also increases significantly, with gains of 3.55× at 8-bit and 1.27× at 16-bit precision. These results highlight the effectiveness of the proposed architectures in enhancing computational efficiency across multiple precision modes.

Table 6.6: Post-Implementation Resource Utilization for integrate system architecture (Control Engine, Memory Unit, 8*8 2D Systolic Array) on the Virtex-7 VC709 Board.

| Resources | Utilization | | |
|---|---|---|---|
| | Utilization | Available | Utilization % |
| LUT | 112463 | 433200 | 25.96 |
| FF | 6940 | 866400 | 0.801 |
| BRAM | 2.5 | 1470 | 0.17 |
| DSP | 0 | 2560 | 0 |

A key differentiator of this study is the integration of CORDIC-based multipliers within the SIMD PE, enabling support for precision combinations of 8-, 16-, and 32-bit, including asymmetric MAC operations—capabilities not addressed by existing designs. Based on the analysis, four pipeline stages are recommended for the SIMD_Low configuration, while five pipeline stages are adopted for the SIMD_High configuration. This selection effectively balances higher throughput, tight timing constraints, and clock frequency targets, ensuring FPGA-specific implementation requirements are met without noticeable degradation in computational accuracy. The architecture exhibits reduced area, power, and delay, while also enhancing throughput, making it well-suited for real-time applications. The design has been implemented on an FPGA and functionally verified. An 8×8 systolic array has been realized using the C-SIMD_High architecture supporting 8/16/32-bit precision, integrated with a control unit and memory interface. The resource utilization details are provided in Table 6.6. Validation at the 65nm technology node has confirmed the feasibility of ASIC implementation, with superior area efficiency compared to several state-of-the-art designs. The proposed architecture delivers up to a 2× increase in throughput, particularly for lower precisions, attributed to 100% resource utilization across all supported bitwidths. This results in linear scalability of throughput for low-precision operations. The reduced delay is enabled by the pipelined architecture, and despite multiple optimizations, the design

maintains negligible accuracy loss with comparable delay metrics. These results affirm the suitability of the proposed C-SIMD PE for variable-precision, parallel, and error-tolerant AI applications.

## 6.5 Summary

This chapter introduces an adaptive-precision C-SIMD processing element (PE) designed to support both symmetric (8×8, 16×16, and 32×32) and asymmetric (8×32) MAC operations for scalable DNN inference. The proposed architecture integrates an optimized, pipelined CORDIC-based partial product generation unit, enabling high-frequency operation while maintaining low hardware complexity. By maximizing computational resource utilization across all supported precision modes, the design enhances parallelism and achieves improved throughput.

A scalable reduced-precision adder tree is employed to efficiently accommodate both symmetric and asymmetric operand configurations with minimal hardware overhead. In contrast to existing state-of-the-art designs, the proposed C-SIMD PE supports sixteen 8-bit MACs, four 16-bit MACs, one 32-bit MAC, and four asymmetric 8×32 MACs within a unified architecture. The PE is configurable to operate in both low-precision (C-SIMD_Low: 4/8/16-bit) and high-precision (C-SIMD_High: 8/16/32-bit) modes, enabling flexible precision scaling to match application requirements.

Guided by a Pareto-based analysis of accuracy–performance trade-offs, a four-stage pipelined configuration is selected to support approximate computing with minimal accuracy degradation, while the architecture remains scalable up to eight pipeline stages to enable exact computation when required. The proposed design is validated through both FPGA prototyping and a 65 nm ASIC implementation. Furthermore, a systolic array constructed using the proposed PE demonstrates functional correctness on the FPGA platform. Experimental inference results show that the proposed architecture incurs only a 2% accuracy loss compared to exact computation, demonstrating an effective balance between accuracy, performance, and hardware efficiency for DNN inference.

# Chapter 7

# Conclusion and Future Scope

## 7.1 Conclusion

This thesis proposes a comprehensive set of efficient computational techniques aimed at enhancing the performance of Deep Neural Network (DNN) accelerators, with particular emphasis on optimizing two critical and often competing performance metrics: Computational Throughput and Hardware Resource Utilisation. As DNN models continue to grow in complexity and are increasingly deployed across a wide range of applications, the demand for specialized hardware accelerators capable of delivering high inference performance under strict resource constraints has become increasingly pronounced.

The proposed work directly addresses this challenge by focusing on the design of accelerator architectures that sustain high throughput while minimising hardware resource consumption, such as logic area and power, through approximate computing and parallel architectures. Achieving this balance is especially critical in edge and embedded computing environments, where constraints on energy consumption, silicon area, and real-time latency impose significant limitations on conventional accelerator designs. By targeting these constraints, this thesis contributes towards enabling high-performance DNN inference on resource-limited platforms with minor degradation in accuracy.

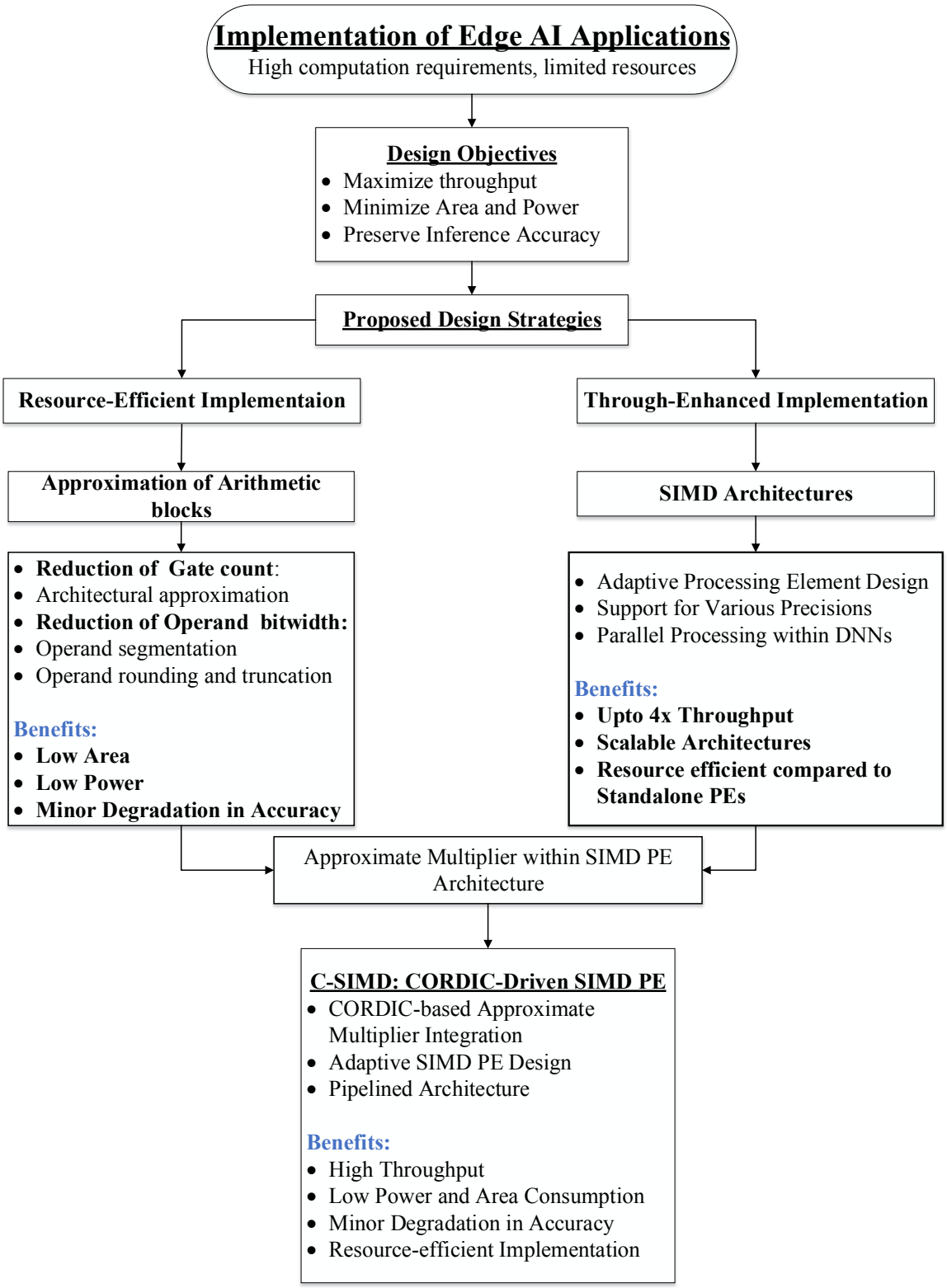



Figure 7.1: Overview of Design Techniques for Optimized Processing Element Architectures in Edge AI systems.

To achieve these objectives, this thesis investigates approximation techniques at both the architectural and algorithmic levels, leveraging the inherent error resilience of deep neural network (DNN) workloads. Since multiply–accumulate (MAC) units are the fundamental building blocks of DNN computation, optimising the arithmetic components within MAC units—namely the multiplier and adder blocks—is critical to improving the overall efficiency of DNN accelerator implementations. Accordingly, one of the central objectives of this thesis is to establish an optimal accuracy–efficiency trade-off that enables substantial reductions in hardware resource utilisation while preserving acceptable inference accuracy, thereby making the proposed architectures well-suited for error-tolerant and resource-constrained applications.

At the architectural level, approximate computing mechanisms are integrated into accelerator datapaths to reduce computational complexity and hardware overhead. At the algorithmic level, approximation techniques are employed to relax computational precision in a controlled manner, enabling further reductions in resource utilisation without incurring unacceptable degradation in inference accuracy.

In particular, the proposed approximations target fundamental arithmetic circuits, including adders and multipliers, which constitute the core computational components of DNN accelerators. Adders are approximated at the architectural level to improve hardware efficiency, resulting in reduced area and power consumption. Multipliers—typically the dominant contributors to the computational cost of DNN inference—are approximated using operand rounding and segmentation-based techniques. These methods reduce resource utilisation by limiting partial-product generation and computational complexity while maintaining acceptable accuracy levels. The proposed approximation strategies are carefully designed to avoid significant loss in inference performance, making them especially suitable for edge-oriented DNN applications.

Furthermore, the thesis also explores the Single Instruction Multiple Data (SIMD) architectural paradigm to enhance computational parallelism and improve workload mapping efficiency under limited hardware resources. The proposed Adaptive SIMD processing element design performs multiple operations at various bit precisions and supports both symmetric and asymmetric multiplication, which are crucial for hetero-

geneous DNN workloads. The proposed architecture enhances throughput upto 4 × with respect to the state-of-the-art architectures.

Moreover, within the SIMD architecture, approximation is selectively applied using a CORDIC-based multiplier to reduce the hardware required for MAC operations. By replacing conventional multipliers with CORDIC-based iterative computation, the proposed SIMD design significantly reduces logic complexity and area consumption, while still providing sufficient numerical precision for DNN inference. This approach enables efficient parallel execution of MAC operations with lower hardware overhead, making it particularly effective for resource-constrained edge AI accelerators. The various design techniques for developing optimized processing element architectures targeting Edge AI systems are illustrated in Fig. 7.1. The figure highlights the integration of arithmetic-level optimization and parallel processing strategies to achieve high throughput and improved hardware efficiency under resource constraints.

Comprehensive hardware evaluations are conducted using both FPGA- and ASIC-based implementations to validate the effectiveness, scalability, and practicality of the proposed accelerator architectures. These evaluations quantitatively analyse key metrics, including silicon area, power consumption, and throughput, demonstrating the suitability of the proposed designs for Edge AI deployments. The inference accuracy is systematically evaluated using Python-based emulation frameworks across multiple datasets and DNN models. This software-level analysis enables detailed assessment of the impact of architectural and algorithmic approximations on model accuracy, providing a thorough characterization of the trade-offs introduced by the proposed accelerator designs.

Overall, this work establishes a favourable accuracy–efficiency trade-off by systematically balancing computational precision with hardware resource optimisation across multiple design levels. Through the careful integration of approximation techniques within arithmetic units, SIMD architectures, and multiply–accumulate operations, the proposed accelerator designs demonstrate that significant reductions in area, power consumption, and computational complexity can be achieved without incurring unacceptable degradation in inference accuracy. Furthermore, the method-

ologies presented in this thesis offer a structured and scalable design framework for developing resource-efficient DNN accelerators, enabling the reliable deployment of high-performance DNN inference on energy- and area-constrained Edge AI platforms. Collectively, these contributions provide a practical pathway for translating complex DNN models into efficient hardware implementations suitable for real-time and embedded edge applications.

## 7.2 Future Scope

This thesis presented a range of optimisation strategies to enhance throughput and achieve resource-efficient acceleration of Deep Neural Networks for edge AI applications. The proposed techniques address challenges related to hardware complexity, power consumption, and computational efficiency, demonstrating the feasibility of deploying high-performance DNN inference on resource-constrained platforms. Architectural-, arithmetic-, and SIMD-level optimisations were systematically explored, providing a robust foundation for efficient edge-oriented accelerator design.

The primary objective of this research was to develop efficient computing architectures for DNN accelerators. Although the proposed approaches achieve significant improvements in throughput and hardware efficiency, certain aspects remain beyond the scope of this study and warrant further investigation. These aspects are outlined as potential directions for future work.

1. The work presented in this thesis is primarily evaluated using fixed-point and integer arithmetic. Future extensions could investigate the applicability of the proposed approximation and architectural techniques to floating-point and other numerical representations, such as posit, to broaden their usability across diverse precision requirements.

2. The proposed optimisation strategies are evaluated mainly on conventional DNN workloads. However, emerging models such as large language models (LLMs) and transformer-based architectures could also potentially benefit from the pro-

posed techniques. Evaluating the performance, accuracy, and hardware efficiency of these models under the proposed optimisation frameworks remains an important direction for future research.

3. Additional future work may explore dynamic and adaptive approximation mechanisms that can adjust precision and computational complexity at runtime based on workload characteristics, accuracy requirements, or operating conditions, thereby improving robustness and real-time adaptability in edge deployments.

# Appendix A

This appendix provides detailed numerical values and exact inference accuracy percentages for the graphical evaluations presented in Chapter 5, along with representative examples of the CORDIC equations discussed in Chapter 6.

## Inference Accuracy Data Tables

Tables 1 and 2 list the detailed performance metrics across various bit-widths, DNN architectures, and benchmark datasets, for Figures 5.9 and 5.10 respectively.

Table 1: Inference Accuracy Evaluation on Various DNN Models for MNIST and CIFAR-10 Datasets using the Proposed and Baseline architectures.

| Dataset | Model Architecture | 4-bit (%) | 8-bit (%) | 16-bit (%) | 32-bit (%) | Baseline (%) |
|---|---|---|---|---|---|---|
| **MNIST** | LeNet5 | 96.50 | 97.80 | 98.20 | 98.40 | 98.46 |
| | **LeNet5_SIMD (Proposed)** | 96.20 | 97.65 | 98.15 | 98.01 | 98.46 |
| | ResNet18 | 98.10 | 98.90 | 99.10 | 99.25 | 99.30 |
| | **ResNet18_SIMD (Proposed)** | 97.95 | 98.85 | 99.05 | 99.21 | 99.30 |
| **CIFAR-10** | LeNet5 | – | 85.50 | 86.50 | 87.00 | 87.10 |
| | **LeNet5_SIMD (Proposed)** | – | 85.00 | 86.00 | 86.50 | 87.10 |
| | ResNet18 | – | 94.00 | 95.00 | 95.50 | 95.70 |
| | **ResNet18_SIMD (Proposed)** | – | 93.50 | 94.50 | 95.00 | 95.70 |
| | VGG16 | – | 84.10 | 86.50 | 87.30 | 87.60 |
| | **VGG16_SIMD (Proposed)** | – | 83.60 | 86.15 | 87.05 | 87.60 |
| | AlexNet | – | 74.50 | 76.80 | 77.90 | 78.20 |
| | **AlexNet_SIMD (Proposed)** | – | 73.90 | 76.40 | 77.60 | 78.20 |

Table 2: Inference Accuracy Evaluation on Various DNN Models for CIFAR-100 and ImageNet-1000 Datasets using the Proposed and Baseline architectures.

| Model Architecture | CIFAR-100 (%) | | | ImageNet-1000 (%) | |
|---|---|---|---|---|---|
| | 8-bit | 16-bit | 32-bit | 16-bit | 32-bit |
| ResNet18 | 66.20 | 67.10 | 67.80 | – | – |
| **ResNet18_SIMD (Proposed)** | 65.80 | 66.75 | 67.45 | – | – |
| VGG16 | 60.50 | 62.30 | 64.10 | 63.80 | 65.50 |
| **VGG16_SIMD (Proposed)** | 60.10 | 61.90 | 63.75 | 61.40 | 64.00 |
| CaffeNet | 54.10 | 55.60 | 57.80 | – | – |
| **CaffeNet_SIMD (Proposed)** | 53.50 | 55.10 | 57.20 | – | – |

# CORDIC-Driven Linear Multiplication Algorithm

To illustrate the operation of the CORDIC-based linear multiplication algorithm, a simplified 4-bit fixed-point example is considered. Equations (6.1a)–(6.1c) define the fundamental objectives of the linear CORDIC execution block, which performs multiplication and accumulation through iterative shift-and-add operations:

$$X_{\text{out}} = X_{\text{in}} \tag{6.1a}$$

$$Y_{\text{out}} = Y_{\text{in}} + X_{\text{in}} \cdot W_{\text{in}} \tag{6.1b}$$

$$W_{\text{out}} = 0 \tag{6.1c}$$

The iterative updates governing the linear CORDIC multiplication process are given by:

$$X_{n+1} = X_n \cdot 2^{-1} \tag{6.2a}$$

$$Y_{n+1} = Y_n + \delta_n \cdot X_n \tag{6.2b}$$

$$W_{n+1} = W_n - \delta_n \cdot 2^{-n} \tag{6.2c}$$

where the control variable $\delta_n \in \{+1, -1\}$ is selected according to the sign of the residual weight:

$$\delta_n = \begin{cases} +1, & W_n \geq 0, \\ -1, & W_n < 0. \end{cases}$$

These iterative updates progressively reduce the residual weight term $W_n$ toward zero while accumulating the multiplication result within the $Y$ register using only shift and add operations.

## Setup for the 4-Bit Fixed-Point Example

To demonstrate the operation of the iterative multiplication process, the following 4-bit unsigned fixed-point values with three fractional bits are selected:

- **Input Activation ($X_{\text{in}}$):** $0.110_2 = 0.75$

- **Weight ($W_{\mathbf{in}}$):** $0.101_2 = 0.625$

- **Initial Accumulated Bias ($Y_{\mathbf{in}}$):** $0.010_2 = 0.25$

According to Equation (6.1b), the expected output is:

$$Y_{\text{out}} = Y_{\text{in}} + X_{\text{in}} W_{\text{in}} = 0.25 + (0.75 \times 0.625) = 0.71875$$

The datapath registers are initialized at $n = 0$ as:

$$X_0 = 0.75, \qquad Y_0 = 0.25, \qquad W_0 = 0.625.$$

## Step-by-Step Micro-Rotation Trace

### Iteration 0 ($n = 0$)

Since $W_0 \geq 0$, the direction control selects $\delta_0 = +1$. Applying Equations (6.2a)–(6.2c):

$$\begin{aligned} X_1 &= 0.75 \times 0.5 = 0.375 \\ Y_1 &= 0.25 + (+1)(0.75) = 1.000 \\ W_1 &= 0.625 - (+1)(2^0) = -0.375 \end{aligned}$$

### Iteration 1 ($n = 1$)

Since $W_1 < 0$, the direction control selects $\delta_1 = -1$. Applying Equations (6.2a)–(6.2c):

$$\begin{aligned} X_2 &= 0.375 \times 0.5 = 0.1875 \\ Y_2 &= 1.000 + (-1)(0.375) = 0.625 \\ W_2 &= -0.375 - (-1)(2^{-1}) = 0.125 \end{aligned}$$

**Iteration 2** ($n = 2$)

Since $W_2 \geq 0$, the direction control selects $\delta_2 = +1$. Applying Equations (6.2a)–(6.2c):

$$X_3 = 0.1875 \times 0.5 = 0.09375$$
$$Y_3 = 0.625 + (+1)(0.1875) = 0.8125$$
$$W_3 = 0.125 - (+1)(2^{-2}) = -0.125$$

**Iteration 3** ($n = 3$)

Since $W_3 < 0$, the direction control selects $\delta_3 = -1$. Applying Equations (6.2a)–(6.2c):

$$X_4 = 0.09375 \times 0.5 = 0.046875$$
$$Y_4 = 0.8125 + (-1)(0.09375) = \mathbf{0.71875}$$
$$W_4 = -0.125 - (-1)(2^{-3}) = \mathbf{0.000}$$

## Summary Table of Convergence

Table 3: 4-Bit Fixed-Point Linear CORDIC Convergence Trace

| **Step** ($n$) | $\boldsymbol{\delta_n}$ | $\boldsymbol{X_n}$ | $\boldsymbol{Y_n}$ | $\boldsymbol{W_n}$ |
|---|---|---|---|---|
| 0 | +1 | 0.750000 | 0.250000 | 0.625000 |
| 1 | -1 | 0.375000 | 1.000000 | -0.375000 |
| 2 | +1 | 0.187500 | 0.625000 | 0.125000 |
| 3 | -1 | 0.093750 | 0.812500 | -0.125000 |
| **Final** ($n = 4$) | – | 0.046875 | **0.718750** | **0.000000** |

## Hardware Convergence Verification

After four iterations, the residual weight converges to zero:

$$W_{\text{out}} = W_4 = 0,$$

which satisfies the objective specified in Equation (6.1c). Furthermore, the accumulated output value becomes:

$$Y_{\text{out}} = Y_4 = 0.71875.$$

This exactly matches the expected theoretical result obtained from Equation (6.1b):

$$Y_{\text{in}} + X_{\text{in}} W_{\text{in}} = 0.25 + (0.75)(0.625) = 0.71875.$$

Therefore, the multiplication converges exactly to the desired solution with zero residual error for this fixed-point example. This numerical validation confirms that the proposed linear CORDIC execution block accurately performs multiply-accumulate (MAC) operations using only low-cost shift-and-add hardware primitives while progressively driving the residual weight term toward zero. Consequently, the architecture eliminates the need for conventional multiplier structures and reduces datapath complexity within the critical execution path.

# Bibliography


[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[2] 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, Nov.2017.

[3] S. M. Nabavinejad, M. Baharloo, K.-C. Chen, M. Palesi, T. Kogel, and M. Ebrahimi, "An Overview of Efficient Interconnection Networks for Deep Neural Network Accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 268–282, Sep. 2020.

[4] S.-H. Noh, J. Koo, S. Lee, J. Park, and J. Kung, "Flexblock: A Flexible DNN Training Accelerator with Multi-mode Block Floating Point Support," *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2522–2535, Mar. 2023.

[5] H. T. Ünal and F. Başçiftçi, "Evolutionary Design of Neural Network Architectures: A Review of Three Decades of Research," *Artificial Intelligence Review*, vol. 55, no. 3, pp. 1723–1802, Mar. 2022.

[6] F. Foukalas, "A Survey of Artificial Neural Network Computing Systems," *Cognitive Computation*, vol. 17, no. 1, p. 4, Feb. 2025.

[7] X. Dai, J. Zhang, Z. Wang, and J. Lin, "HWSA: A High-Ratio Weight Sparse Accelerator for Efficient CNN Inference," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 10, pp. 5797–5810, Feb. 2025.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, Feb. 2015.

[9] H. Hussain, P. Tamizharasan, and C. Rahul, "Design Possibilities and Challenges of DNN models: A Review on the Perspective of End Devices," *Artificial Intelligence Review*, vol. 55, no. 7, pp. 5109–5167, Oct. 2022.

[10] 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, Nov. 2022.

[11] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-CNN: An FPGA-based Framework for training Convolutional Neural Networks," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 107–114, IEEE, Jul. 2016.

[12] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore, *et al.*, "A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms," *ACM Computing Surveys*, vol. 57, no. 11, pp. 1–39, Jun. 2025.

[13] A. Al Bataineh, D. Kaur, and S. M. J. Jalali, "Multi-Layer Perceptron Praining Optimization using Nature-inspired Computing," *IEEE Access*, vol. 10, pp. 36963–36977, Apr. 2022.

[14] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, "Deep Feature Extraction and Classification of Hyperspectral Images based on Convolutional Neural Networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6232–6251, Jul. 2016.

[15] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently Recurrent Neural Network (INDRNN): Building a Longer and Deeper RNN," in *Proceedings of the*

*IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466, 2018.

[16] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, "Understanding Hidden Memories of Recurrent Neural Networks," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 13–24, IEEE, Oct. 2017.

[17] M. Verhelst and B. Moons, "Embedded Deep Neural Network Processing: Algorithmic and Processor Techniques bring Deep learning to IoT and Edge Devices," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 55–65, Nov. 2017.

[18] D. Mahmoud, "Hardware Acceleration," *Trends in Data Protection and Encryption Technologies*, pp. 109–114, Apr. 2023.

[19] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, *et al.*, "A Configurable Cloud-scale DNN Processor for Real-time AI," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, IEEE, Jun. 2018.

[20] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The Architectural Implications of Autonomous Driving: Constraints and Acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural support for programming Languages and Operating Systems*, pp. 751–766, Mar. 2018.

[21] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, Jun. 2014.

[22] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver, "A QoS-aware Memory Controller for Dynamically balancing GPU and CPU bandwidth use in an MPSoC," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 850–855, Jun. 2012.

[23] C. Kachris and D. Soudris, "A Survey on Reconfigurable Accelerators for Cloud Computing," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–10, IEEE, Aug. 2016.

[24] W.-m. Hwu and S. Patel, "Accelerator Architectures—A Ten-Year Retrospective," *IEEE Micro*, vol. 38, no. 6, pp. 56–62, Dec. 2018.

[25] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the Graphics Processing Unit (GPU)," *IEEE Micro*, vol. 41, no. 6, pp. 42–51, Nov. 2021.

[26] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A Survey on Hardware Accelerators: Taxonomy, Trends, Challenges, and Perspectives," *Journal of Systems Architecture*, vol. 129, p. 102561, Aug. 2022.

[27] A. Boutros, S. Yazdanshenas, and V. Betz, "You Cannot Improve What You Do Not Measure: FPGA vs. ASIC Efficiency Gaps for Convolutional Neural Network Inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, Dec. 2018.

[28] 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, Jul. 2020.

[29] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran, *et al.*, "RaPiD: AI Accelerator for Ultra-low Precision Training and Inference," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 153–166, IEEE, Jun. 2021.

[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*, vol. 486, pp. 147–159, May 2022.

[31] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul. 2019.

[32] W. Kahan, "IEEE standard 754 for Binary Floating-Point Arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, May 1996.

[33] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN Pruning to the underlying Hardware Parallelism," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 548–560, Jun. 2017.

[34] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, IEEE, Feb. 2014.

[35] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm for Energy-Efficient Design," *2013 18th IEEE European Test Symposium (ETS) (pp. 1-6). IEEE*, vol. 7, p. 1–6, May 2013.

[36] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Z. Pan, M. Wolf, and J. Henkel, "MLCAD: A Survey of Research in Machine learning for CAD Keynote Paper," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3162–3181, Nov. 2021.

[37] T. W. Y. T. F. Y. Zhang, Qian and Q. Xu, "ApproxANN: An Approximate Computing Framework for Artificial Neural Network," *2015 Design, Automation and Test in Europe Conference  Exhibition (DATE)*, pp. 701–706, Mar. 2015.

[38] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-Efficient Neuromorphic Systems using Approximate Computing," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, pp. 27–32, Aug. 2014.

[39] Y. Kim, Y. Zhang, and P. Li, "An Energy-Efficient Approximate Adder with Carry Skip for Error-resilient Neuromorphic VLSI systems," in *2013 IEEE/ACM*

*International Conference on Computer-Aided Design (ICCAD)*, pp. 130–137, IEEE, Nov. 2013.

[40] C.-Y. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "Exploiting Approximate Computing for Deep Learning Acceleration," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 821–826, IEEE, Mar. 2018.

[41] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, Mar. 2020.

[42] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power-and Area-Efficient Approximate Wallace Tree Multiplier for Error-resilient Systems," in *Fifteenth International Symposium on Quality Electronic Design*, pp. 263–269, IEEE, Mar. 2014.

[43] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A Low Latency Generic Accuracy Configurable Adder," in *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6, Jun. 2015.

[44] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-Efficient Approximate Multiplication Circuits Through Partial Product Perforation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3105–3117, Mar. 2016.

[45] Y. Wu, C. Chen, W. Xiao, X. Wang, C. Wen, J. Han, X. Yin, W. Qian, and C. Zhuo, "A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 1, pp. 1–37, Jan. 2024.

[46] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 258–261, IEEE, Mar. 2017.

[47] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, Nov. 2016.

[48] I. Hammad, L. Li, K. El-Sankary, and W. M. Snelgrove, "CNN Inference using a Preprocessing Precision Controller and Approximate Multipliers with Various Precisions," *IEEE Access*, vol. 9, pp. 7220–7232, Jan. 2021.

[49] 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, Jan. 2021.

[50] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic Logic Synthesis of Approximate Circuits," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 796–801, Jun. 2012.

[51] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the Accuracy and Hardware Efficiency of Neural Networks using Approximate Multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, Oct. 2019.

[52] G. Zervakis, O. Spantidi, I. Anagnostopoulos, H. Amrouch, and J. Henkel, "Control Variate Approximation for DNN Accelerators," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 481–486, IEEE, Dec. 2021.

[53] V. Mrazek, Z. Vasícek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic Layer-Wise Approximation of Deep Neural Network Accelerators without Retraining," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, Nov. 2019.

[54] M. H. Haider, H. Zhang, and S.-B. Ko, "Decoder Reduction Approximation Scheme for Booth Multipliers," *IEEE Transactions on Computers*, vol. 73, no. 3, pp. 735–746, Dec. 2023.

[55] G. Raut, S. Karkun, and S. K. Vishvakarma, "An Empirical Approach to Enhance Performance for Scalable CORDIC-based Deep Neural Networks," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 3, pp. 1–32, Jun. 2023.

[56] Y. Parmar and K. Sridharan, "A Resource-Efficient Multiplierless Systolic Array Architecture for Convolutions in Deep Networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 2, pp. 370–374, Mar. 2019.

[57] J. N. Mitchell, "Computer Multiplication and Division using Binary Logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, Aug. 1962.

[58] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased Multipliers for Approximate Integer and Floating-Point Multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, Oct. 2018.

[59] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, "Efficient Hardware Acceleration of CNNs using Logarithmic Data representation with Arbitrary Log-base," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 1–8, Nov. 2018.

[60] 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, Apr. 2021.

[61] V. Camus, C. Enz, and M. Verhelst, "Survey of precision-scalable multiply-accumulate units for neural-network processing," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 57–61, IEEE, Mar. 2019.

[62] M. Huang, Y. Liu, C. Man, K. Li, Q. Cheng, W. Mao, and H. Yu, "A High Performance Multi-Bit-Width Booth Vector Systolic Accelerator for NAS Op-

timized Deep Learning Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3619–3631, Jun. 2022.

[63] N. Neda, S. Ullah, A. Ghanbari, H. Mahdiani, M. Modarressi, and A. Kumar, "Multi-Precision Deep Neural Network Acceleration on FPGAs," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 454–459, IEEE, Jan. 2022.

[64] G. Raut, P. J. Edavoor, D. Selvakumar, and R. Thakur, "A SIMD Dynamic Fixed Point Processing Engine for DNN Accelerators," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–8, IEEE, Apr. 2024.

[65] E. M. Ibrahim, L. Mei, and M. Verhelst, "Taxonomy and benchmarking of precision-scalable MAC arrays under enhanced DNN dataflow representation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 5, pp. 2013–2024, Jan. 2022.

[66] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, Oct. 2016.

[67] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-Pragmatic Deep Neural Network Computing," in *Proceedings of the 50th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 382–394, Oct. 2017.

[68] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Oct. 2018.

[69] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, Jun. 2018.

[70] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "BitFusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775, IEEE, Jun. 2018.

[71] S. Ryu, H. Kim, W. Yi, E. Kim, Y. Kim, T. Kim, and J.-J. Kim, "Bitblade: Energy-efficient Variable Bit-Precision Hardware Accelerator for Quantized Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 6, pp. 1924–1935, Jan. 2022.

[72] W. Liu, J. Lin, and Z. Wang, "A Precision-Scalable Energy-Efficient Convolutional Neural Network Accelerator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 10, pp. 3484–3497, May 2020.

[73] L. Mei, M. Dandekar, D. Rodopoulos, J. Constantin, P. Debacker, R. Lauwereins, and M. Verhelst, "Sub-word Parallel Precision-Scalable MAC Engines for Efficient Embedded DNN Inference," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 6–10, IEEE, Mar. 2019.

[74] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1 TOPS/W Reconfigurable CNN-RNN Processor for General-Purpose Deep Neural Networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 240–241, IEEE, Mar. 2017.

[75] H. Zhang, D. Chen, and S.-B. Ko, "Efficient Multiple-Precision Floating-point Fused Multiply-Add with Mixed-Precision support," *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1035–1048, Jan. 2019.

[76] W. Mao, K. Li, X. Xie, S. Zhao, H. Li, and H. Yu, "A Reconfigurable Multiple-Precision Floating-Point Dot Product unit for High-Performance Computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1793–1798, IEEE, Feb. 2021.

[77] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10TOPS/W Subword-Parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 246–247, IEEE, Feb. 2017.

[78] Y. E. Wang, C.-J. Wu, X. Wang, K. Hazelwood, and D. Brooks, "Exploiting Parallelism Opportunities with Deep Learning Frameworks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, pp. 1–23, Dec. 2020.

[79] L. B. Soares, M. M. A. da Rosa, C. M. Diniz, E. A. C. da Costa, and S. Bampi, "Design Methodology to Explore Hybrid Approximate Adders for Energy-Efficient Image and Video Processing Accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2137–2150, Jan. 2019.

[80] A. B. Kahng and S. Kang, "Accuracy-configurable Adder for Approximate Arithmetic Designs," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 820–825, Jun. 2012.

[81] T. Yang, T. Ukezono, and T. Sato, "A Low-Power Configurable Adder for Approximate Applications," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pp. 347–352, IEEE, Mar. 2018.

[82] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-performance Accurate and Approximate Multipliers for FPGA-based Hardware Accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 211–224, Feb. 2021.

[83] L. Qian, C. Wang, W. Liu, F. Lombardi, and J. Han, "Design and Evaluation of an Approximate Wallace-Booth multiplier," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1974–1977, IEEE, May 2016.

[84] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and Evaluation of Approximate Logarithmic Multipliers for Low Power Error-tolerant Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, Feb. 2018.

[85] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient Approximate Multiplication for Digital Signal Processing and classification applications," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, vol. 23, no. 6, pp. 1180–1184, Jul. 2014.

[86] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An Energy-Efficient Truncation-and Rounding-based Scalable Approximate Multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1161–1173, Jan. 2019.

[87] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A Rounding-based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 393–401, Jul. 2016.

[88] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 418–425, IEEE, Nov. 2015.

[89] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, G. Saggese, and G. Di Meo, "Approximate Multipliers using Static Segmentation: Error Analysis and Improvements," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2449–2462, Mar. 2022.

[90] G. Di Meo, G. Saggese, A. G. Strollo, and D. De Caro, "Approximate MAC unit using Static Segmentation," *IEEE Transactions on Emerging Topics in Computing*, Sep. 2023.

[91] K. Li, J. Zhou, Y. Wang, J. Luo, Z. Yang, S. Yang, W. Mao, M. Huang, and H. Yu, "A Precision-Scalable Energy-Efficient Bit-Split-and-Combination Vector Systolic Accelerator for NAS-optimized DNNs on Edge," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 730–735, IEEE, Mar. 2022.

[92] 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)*, pp. 78–83, IEEE, Jul. 2020.

[93] M. Masadeh, O. Hasan, and S. Tahar, "Input-Conscious Approximate Multiply-Accumulate (MAC) Unit for Energy-Efficiency," *IEEE access*, vol. 7, pp. 147129–147142, Oct. 2019.

[94] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–33, Mar. 2016.

[95] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, Oct. 2015.

[96] 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, Apr. 2022.

[97] F. Ahmadi, M. R. Semati, and H. Daryanavard, "A Low-Power Improved-Accuracy Approximate Error-Report-Propagate Adder for DSP Applications," *Circuits, Systems, and Signal Processing*, pp. 1–26, Jun. 2023.

[98] A. Dalloo, A. Najafi, and A. Garcia-Ortiz, "Systematic Design of an Approximate Adder: The Optimized Lower Part Constant-OR Adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems)*, vol. 26, pp. 1595–1599, Apr. 2018.

[99] D. Esposito, D. De Caro, E. Napoli, N. Petra, and A. G. Strollo, "On the use of approximate adders in carry-save multiplier-accumulators," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, IEEE, May 2017.

[100] S. Venkatachalam and S.-B. Ko, "Design of Power and Area Efficient Approximate Multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1782–1786, Jan. 2017.

[101] E. Napoli, E. Zacharelos, A. G. Strollo, and G. Di Meo, "Approximate Full-Adders: A Comprehensive Analysis," *IEEE Access*, vol. 12, pp. 136054–136072, Sep. 2024.

[102] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 343–348, May 2015.

[103] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 1, pp. 124–137, Dec. 2012.

[104] B. S. Prabakaran, S. Rehman, M. A. Hanif, S. Ullah, G. Mazaheri, A. Kumar, and M. Shafique, "DeMAS: An Efficient Design Methodology for building Approximate Adders for FPGA-based Systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 917–920, IEEE, Mar. 2018.

[105] N. Kumar, R. Adithyaa, T. Pavithra, and B. K. D, "Design Analysis of Wallace Tree-based Multiplier using Approximate Full Adder and Kogge Stone adder," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 612–616, IEEE, Mar. 2020.

[106] S. L. Harris and D. Harris, "Digital Design and Computer Architecture," Jul. 2015.

[107] L. B. Soares, J. Oliveira, E. A. C. da Costa, and S. Bampi, "An Energy-Efficient and Approximate Accelerator Design for Real-Time Canny Edge Detection," *Circuits, Systems, and Signal Processing*, vol. 39, pp. 6098–6120, Dec. 2020.

[108] P. Dhar, S. Guha, T. Biswas, and M. Z. Abedin, "A System Design for License Plate Recognition by using Edge Detection and Convolution Neural Network," in *2018 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2)*, pp. 1–4, IEEE, Feb. 2018.

[109] P. Goyal and S. K. Sahoo, "EOHEAA: Error-Optimized Hardware-Efficient Approximate Adder for Energy-Aware Error-Resilient Applications," *Integration*, p. 102660, May 2026.

[110] P. Balasubramanian, S. M. M. A. H. Zaheen, and D. L. Maskell, "Machine Learning Using Approximate Computing," *Journal of Low Power Electronics and Applications*, vol. 15, no. 2, p. 21, Apr. 2025.

[111] T. Mendez and S. G. Nayak, "Performance Evaluation of Fault-Tolerant Approximate Adder," in *2022 6th International Conference on Devices, Circuits and Systems (ICDCS)*, pp. 1–5, IEEE, Apr.2022.

[112] I. C. Wey, C. C. Ho, Y. S. Lin, and C. C. Peng, "An Area-Efficient Carry Select Adder Design by sharing the common Boolean Logic Term," *Proc. Imecs*, vol. 10, pp. 1–4, Mar. 2012.

[113] B. Ramkumar and H. M. Kittur, "Low-Power and Area-Efficient Carry Select Adder," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, vol. 20, no. 2, pp. 371–375, Jan. 2011.

[114] U. Lotrič and P. Bulić, "Applicability of Approximate Multipliers in Hardware Neural Networks," *Neurocomputing*, vol. 96, pp. 57–65, Nov. 2012.

[115] M. S. Ansari, B. F. Cockburn, and J. Han, "A Hardware-Efficient Logarithmic Multiplier with Improved Accuracy," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 928–931, IEEE, Mar. 2019.

[116] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "RMAC: Runtime Configurable Floating Point Multiplier for Approximate Computing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 1–6, Jul. 2018.

[117] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel, "Architectural-Space Exploration of Approximate Multipliers," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, Nov. 2016.

[118] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," in *2011 24th Internatioal Conference on VLSI Design*, pp. 346–351, Dec. 2011.

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

[120] M. S. Nagar, A. Mathuriya, S. H. Patel, and P. J. Engineer, "High-speed Energy-Efficient Fixed-Point Signed Multipliers for FPGA-based DSP applications," *IEEE Embedded Systems Letters*, vol. 16, no. 4, pp. 417–420, Feb. 2024.

[121] V. Trivedi and S. K. Vishvakarma, "Design of Operand-Rounding based Approximate Multiplier for Error-resilient Applications ," *29th International Sympo-*

*sium on VLSI Design and Test, Chandigarh, India*, August 6-9,2025 (Accepted and In Press).

[122] Y. Guo *et al.*, "High-Efficiency FPGA-based Approximate Multipliers with LUT sharing and carry switching," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–2, IEEE, Jun. 2024.

[123] H. Waris, C. Wang, W. Liu, J. Han, and F. Lombardi, "Hybrid Partial Product-based High-Performance Approximate Recursive Multipliers," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, pp. 507–513, Aug. 2020.

[124] N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini, and N. TaheriNejad, "Carry Disregard Approximate Multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 12, pp. 4840–4853, Aug. 2023.

[125] M. Rathod, A. Hazarika, A. Agrawal, N. Kumar, *et al.*, "Power Efficient Multiplier Design for Error Resilient Edge Applications," *IEEE Embedded Systems Letters*, May 2025.

[126] H. Zhang *et al.*, "FPGA-based Approximate Multiplier for Efficient Neural Computation," in *2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 1–4, IEEE, Dec. 2021.

[127] E. Zacharelos *et al.*, "Approximate recursive multipliers using low power building blocks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1315–1330, Jun. 2022.

[128] U. A. Kumar, S. K. Chatterjee, and S. E. Ahmed, "Low-Power Compressor-based Approximate Multipliers with Error Correcting Module," *IEEE Embedded Systems Letters*, vol. 14, no. 2, pp. 59–62, Sep. 2021.

[129] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–38, Oct. 2021.

[130] K. Li, W. Mao, J. Zhou, B. Li, Z. Yang, S. Yang, L. Du, S. Huang, and H. Yu, "A Vector Systolic Accelerator for Multi-Precision Floating-point High-Performance Computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 10, pp. 4123–4127, Jun. 2022.

[131] B. Zhou, G. Wang, G. Jie, Q. Liu, and Z. Wang, "A High-Speed Floating-Point Multiply-Accumulator Based on FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 10, pp. 1782–1789, Aug. 2021.

[132] D. Mustafa, R. Alkhasawneh, F. Obeidat, and A. S. Shatnawi, "MIMD Programs Execution Support on SIMD Machines: A Holistic Survey," *IEEE Access*, vol. 12, pp. 34354–34377, Mar. 2024.

[133] V. Trivedi, K. Lalwani, G. Raut, A. Khomane, N. Ashar, and S. K. Vishvakarma, "Hybrid ADDer: A Viable Solution for Efficient Design of MAC in DNNs," *Circuits, Systems, and Signal Processing*, vol. 42, no. 12, pp. 7596–7614, Dec. 2023.

[134] N. Ashar, G. Raut, V. Trivedi, S. K. Vishvakarma, and A. Kumar, "QuantMAC: Enhancing Hardware Performance in DNNs With Quantize Enabled Multiply-Accumulate Unit," *IEEE Access*, vol. 12, pp. 43600–43614, Mar. 2024.

[135] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and Benchmarking of Precision-scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697–711, Oct. 2019.

[136] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "BatchSizer: Power-performance trade-off for DNN inference," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 819–824, Jan. 2021.

[137] A. Danysh and D. Tan, "Architecture and Implementation of a Vector/SIMD Multiply-Accumulate Unit," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 284–293, Mar. 2005.

[138] C. Guo, L. Zhang, X. Zhou, G. L. Zhang, B. Li, W. Qian, X. Yin, and C. Zhuo, "A Reconfigurable Multiplier for Signed Multiplications with Asymmetric Bit-widths," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 4, pp. 1–16, Jun. 2021.

[139] H. Zhang, D. Chen, and S.-B. Ko, "New Flexible Multiple-Precision Multiply-Accumulate unit for Deep Neural Network Training and Inference," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 26–38, Sep. 2019.

[140] E. Manca, L. Urbinati, and M. R. Casu, "STAR: Sum-Together/Apart Reconfigurable Multipliers for Precision-Scalable ML Workloads," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, Mar. 2024.

[141] E. Reggiani *et al.*, "Mix-GEMM: An efficient HW-SW Architecture for Mixed-Precision Quantized Deep Neural Networks Inference on Edge Devices," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1085–1098, IEEE, Mar. 2023.

[142] X. Zhao, R. Xu, Y. Gao, V. Verma, M. R. Stan, and X. Guo, "Edge-MPQ: Layer-Wise Mixed-Precision Quantization With Tightly Integrated Versatile Inference Units for Edge Computing," *IEEE Transactions on Computers*, Aug. 2024.

[143] N. Shah, P. Chaudhari, and K. Varghese, "Runtime Programmable and Memory Bandwidth Optimized FPGA-Based Coprocessor for Deep Convolutional Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5922–5934, Apr. 2018.

[144] I. Miro-Panades, B. Tain, J.-F. Christmann, D. Coriat, R. Lemaire, C. Jany, B. Martineau, F. Chaix, A. Quelen, E. Pluchart, *et al.*, "SamurAI: A 1.7 MOPS-36GOPS Adaptive Versatile IoT Node with 15,000× Peak-to-Idle Power Reduction, 207ns Wake-Up Time and 1.3 TOPS/W ML Efficiency," in *2020 IEEE Symposium on VLSI Circuits*, pp. 1–2, IEEE, Jun. 2020.

[145] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, no. 3, pp. 330–334, Sep. 1959.

[146] J. S. Walther, "A Unified Algorithm for Elementary Functions," in *Proceedings of the May 18-20, 1971, Spring Joint Computer Conference*, pp. 379–385, May 1971.

[147] J. Gong, H. Saadat, H. Gamaarachchi, H. Javaid, X. S. Hu, and S. Parameswaran, "ApproxTrain: Fast Simulation of Approximate Multipliers for DNN Training and Inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3505–3518, Mar. 2023.

[148] S. Mach, F. Schuiki, F. Zaruba, and L. Benini, "FPnew: An Open-Source Multiformat Floating-point Unit architecture for Energy-Proportional Transprecision Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, Dec. 2020.

[149] R. Andraka, "A Survey of CORDIC Algorithms for FPGA-based Computers," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, pp. 191–200, Mar. 1998.

[150] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems," *arXiv preprint arXiv:1603.04467*, Mar. 2016.

[151] G. Ottavi, A. Garofalo, G. Tagliavini, F. Conti, A. Di Mauro, L. Benini, and D. Rossi, "Dustin: A 16-cores Parallel ultra-low-power cluster with 2b-to-32b Fully Flexible Bit-Precision and Vector Lockstep Execution Mode," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 6, pp. 2450–2463, Mar. 2023.