Hardware-Software Co-Design for Edge AI Accelerator: A Hardware-Centric Perspective

A Thesis

Submitted in partial fulfillment of the

requirements for the award of the degree

of

Master of Technology (VLSI Design & Nanoelectronics)

by

OMKAR RAJESH KOKANE

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

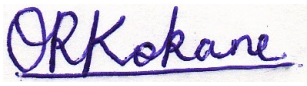
MAY 2025

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Hardware-Software Co-Design for Edge AI Accelerator: A Hardware-Centric Perspective** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology in VLSI Design & Nanoelectronics** 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 July 2023 to May 2025 under the supervision of Dr. Santosh Kumar Vishvakarma, Professor, Indian Institute of Technology Indore, Indore, India.

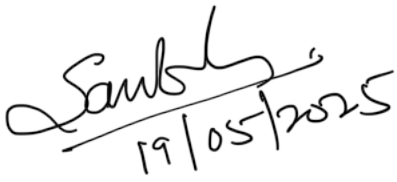
The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.



Signature of the Student

(Omkar Rajesh Kokane)

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



Signature of the Supervisor of M.Tech. Thesis with Date

(Prof. Santosh Kumar Vishvakarma)

Omkar Rajesh Kokane has successfully given his M.Tech. Oral Examination held on

05/05/2025

Signature of Supervisor of M.Tech. Thesis

Date: 19/05/2025

Saptarshi Ghosh

Signature of Convener, DPGC

Date: 19-05-2025

ACKNOWLEDGEMENTS

I am immensely grateful to my M.Tech. thesis supervisor and mentor, Prof. Santosh Kumar Vishvakarma, for consistently encouraging and supporting me in my research. His unwavering belief in my abilities and his invaluable guidance have served as constant motivation, pushing me to exceed my own limits. I would also like to extend my sincere appreciation to all of my thesis evaluation committee members. Their impartial evaluations and thought-provoking questions have contributed significantly to expanding my research perspective.

I am immensely thankful to my family for their unyielding support throughout my master's studies. Their confidence in me and their love, sacrifices, and constant encouragement have been instrumental in my academic success. I will forever be grateful for their guidance, and their faith in me will continue to propel me towards greater accomplishments.

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

I deeply appreciate the Nanoscale Devices, VLSI Circuit and System (NSDCS) Lab research group, especially Dr. Gopal Raut and Mr. Mukul Lokhande and Prof. Adam Teman form Bar-Ilan University in Ramat Gan, Israel and a co-Director of the Emerging Nanoscaled Circuits and Systems (EnICS) Labs and Dr. Ratko Pilipović from University in Ljubljana Faculty of Computer and Information Science for their unwavering support and guidance in both my technical and personal development.

I also appreciate the camaraderie and encouragement of my friends Mr. Sapan Kushwah, Mr. Athava Limaye, Mr. Ashreta Sahay, Mr. Abu Said Parvej Alam, Mr. Amit Singh and labmates, Mrs. Neha Maheshwari, Mr. Sonu Kumar, Mr. Shashank Singh Rawat, Mr. Vikash Vishwakarma, Mr. Ankit Tenwar, Mr. Vijay Sharma, Mrs Akansha Jain, Mr. Anoop R., Mr. Mohamad Faisal and my M. Tech. seniors Ms.

Komal Gupta, and Mr. Sagar Patel, Mr. Radheshyam Sharma whose friendship made my time at the institute truly remarkable...

Omkar Rajesh Kokane

This Thesis is Dedicated

to

Science , Family, Friends
and the Almighty God

ABSTRACT

This thesis presents a comprehensive co-design of hardware-efficient components tailored for edge-AI accelerator, focusing on three key innovations: the Plus-One Adder (P1A), the Logarithmic Posit-enabled Reconfigurable Engine (LPRE), and the CORDIC-based Reconfigurable Processing Engine (RPE).

First, we propose a novel Plus-One Adder (P1A), designed as an incremental unit within a ripple-carry adder (RCA) chain. It integrates a full adder with an excess-1 generator alongside inputs A, B, and Cin. The output is approximated to 2-bit values to reduce hardware complexity, significantly improving resource efficiency. The P1A is evaluated in the context of Two's complement subtraction and rounding-to-even operations, with a detailed analysis of error distance versus area and power metrics using CMOS 28nm technology. Extending this, we introduce the Hybrid Overestimating Approximate Adder (HOAA(n, m)), which enables dynamic reconfigurability between a (n–m)-bit RCA and an m-bit P1A block, based on workload requirements. This architecture achieves up to 1.33× area reduction and 0.79× power savings, making it a promising component for performance-optimized processing engines.

To address the limitations of edge-AI deployments in bandwidth-constrained environments, we propose LPRE, a Logarithmic-Posit-enabled Reconfigurable Engine that delivers high throughput with minimal resource usage. Leveraging time-multiplexed, single-layer reconfigurable hardware, LPRE supports efficient execution of multi-layer perceptrons and convolutional neural networks. Experimental results on LeNet-5 using the MNIST dataset show that LPRE achieves 4× throughput enhancement at 8-bit precision, with negligible accuracy degradation compared to the FP32 baseline. Moreover, it reduces resource utilization by up to 80% versus fixed-point designs and 50% compared to state-of-the-art accelerators, demonstrating its suitability for low-power, real-time applications such as number plate recognition and scalable IoT edge inference.

Finally, we introduce a CORDIC-based Reconfigurable Processing Engine (RPE)

that supports both linear MAC computations and nonlinear iterative activation functions such as tanh, sigmoid, and softmax. The architecture is implemented as a pipelined, systolic array-based engine, with runtime reconfigurability and precision scaling. The design incorporates five linear CORDIC stages for MAC and additional stages for activation, enabling efficient reuse of hardware. Through pruning strategies (up to 40%), the RPE achieves up to 4.64× throughput gain, with area and power reductions of 4.06× and 5.02×, respectively, on CMOS 28nm, and 2.5× resource and 3× power savings on FPGA. The SYCore (Systolic CORDIC Engine for Reconfigurability and Enhanced throughput) employs an output-stationary dataflow and is managed by a CAESAR control engine, enabling execution of diverse AI workloads including Transformers, RNNs/LSTMs, and DNNs. Use cases span image detection, LLMs, and speech recognition, validating the scalability and energy efficiency of the proposed edge-AI accelerator design.

Together, these co-designed architectural elements form a robust foundation for next-generation edge-AI accelerators, addressing key challenges in area, power, throughput, and configurability without compromising accuracy—making them well-suited for deployment in mobile, IoT, and embedded AI systems.

Contents

List of Figures

List of Tables

List of Abbrevations

DL	-	Deep Learning
ML	-	Machine Learning
AI	-	Artificial Intelligence
DNN	-	Deep Neural Network
SRAM	-	Static Random Access Memory
RCA	-	Ripple carry adder
CSkA	-	Carry skip adder
SoC	-	System-on-Chip
ASIC	-	Application Specific Integrated Circuit
STA	-	Static Time Analysis
IC	-	Integrated Circuits
P1A	-	Plus One Adder
PE	-	Processing Engine
AF	-	Activation Function
PQRE	-	Logarithmic Posit Quire Processing Engine
MAC	-	Multiply and Accumulate
LPRE	-	Logarithmic Posit-enabled Reconfigurable edge-AI Engine
SA	-	Systolic Array
CORDIC	-	Coordinate Rotation Digital Compute
HOAA	-	Hybrid Overestimating Approximate Adder
LUT	-	Look-Up-Table
FPGA	-	Field-Programmable Gate Arrays
SoTA	-	State of the Art
RPE	-	Reconfigurable Processing Engine

Chapter 1

Introduction

1.1 Introduction

Artificial Intelligence and Machine Learning (**AI/ML**) have been extensively implemented and embraced across various technological domains as shown in Fig. 1.1. Among the myriad algorithmic frameworks, **DNNs** have captured significant attention due to their vast applicability in fields like science, engineering, agriculture, healthcare, etc. DNNs have revolutionized image classification, audio recognition, and natural language processing [7]. DNNs are structured with multiple layers, starting from the input, passing through several hidden layers, and concluding with the output layers. During inference, these layers form a directed acyclic graph with nodes and edges leading to neurons in subsequent layers. Each node, or neuron, executes two core tasks: First, it employs a Multiply, Add and Accumulate(**MAC**) unit to aggregate the weighted input features and second, it applies the Activation Function(**AF**) to the resulting sum. A deep neural network (DNN) is a type of Artificial Neural Network(**ANN**) that consists of multiple hidden layers between the input and output, typically structured as either a Convolutional Neural Network (**CNN**) or Fully Connected (**FC**). Researchers have focused on developing specialized hardware solutions, including Application-Specific Integrated Circuits (**ASICs**), Graphics Processing Units (**GPUs**), Field-Programmable Gate Arrays (**FPGA**), and multi-threaded CPUs [8–10], to enhance the efficiency of DNN implementations.

Deep Neural networks (DNN) have become ubiquitous in edge-AI devices, powering applications like facial recognition, object detection in computer vision, chatbots for human-like interactions, sentiment analysis in online shopping, social media, etc. In recent years, the models have grown exponentially in size and complexity [**?**], contributing to improved prediction accuracy and effectiveness, potentially raising significant concerns for hardware platforms that are used for real-time inference. Software-level techniques like quantization and pruning help to provide smaller models for smooth inference [11]. However, to provide the best user experience, the need for hardware accelerators has rapidly evolved for resource-constrained environments [11, 12]. The edge accelerator must have special computing blocks to

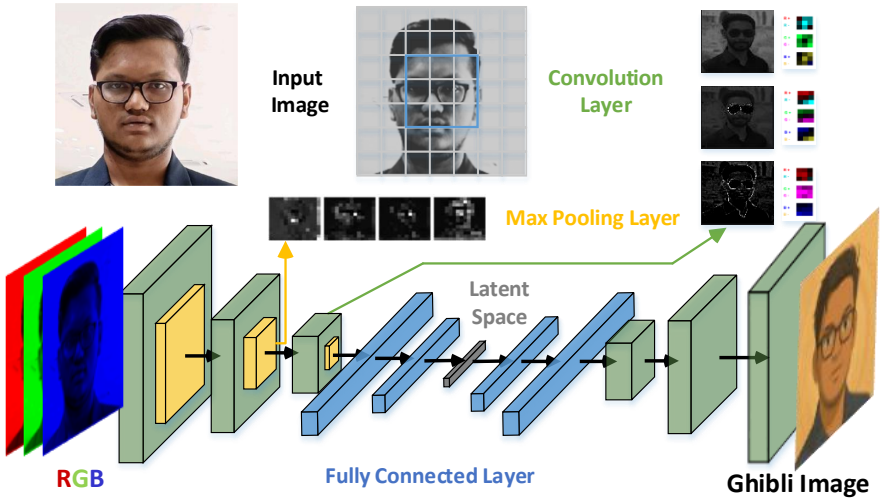


Figure 1.1 Overview of the neural network architecture for generating Ghibli-style RGB images from input images via latent space transformation.

efficiently process various Convolution, Pooling, and Fully connected Layers. At the circuit level, the fundamental parallel components are Multiply-and-Accumulate (MAC) [13] operations and Non-linear Activation Functions (AF) [12]. However, as the main processor offloads these tasks to the accelerator, the need to support less frequent operations like Subtraction, Rounding, and Pooling operations has also arisen. Otherwise, these smaller workloads would contribute to more power consumption as these operations would need to be processed by a general-purpose processor and involve significant data movement. Understanding the trade-off between the NN performance vs physical parameters of hardware resources aids in accelerating the efficient acceleration of DNN solutions. Researchers [14–16] have explored architectural optimization at various abstraction levels, including fixed-point (FxP) arithmetic, hardware reuse/reduction, and approximation.

1.1.1 Background & Motivation

1.1.1.1 Hybrid Overestimating Approximate Adder (HOAA)

The article has discussed one fundamental digital block binary adder. The adders are widely adapted in ALUs (addition, subtraction) in microprocessors, Shift-Add-based MAC units, Coordinate Rotation Digital Computer (CORDIC) based

transcendental functions, etc. The conventional accurate adders [14, 15] involve Ripple Carry Adders (RCA), Carry Look-Ahead Adders (CLA), Carry Bypass Adders (CBA), Carry Select Adders (CSA), Tree Adders, Serial Adders, Brent Kung Adders (BKA), Kogge Stone Adders (KSA), etc. However, The last decade has seen the rise of approximate adders by modifying the carry chain, half adder (HA), full adder (FA), compressor or reducing logic gates.

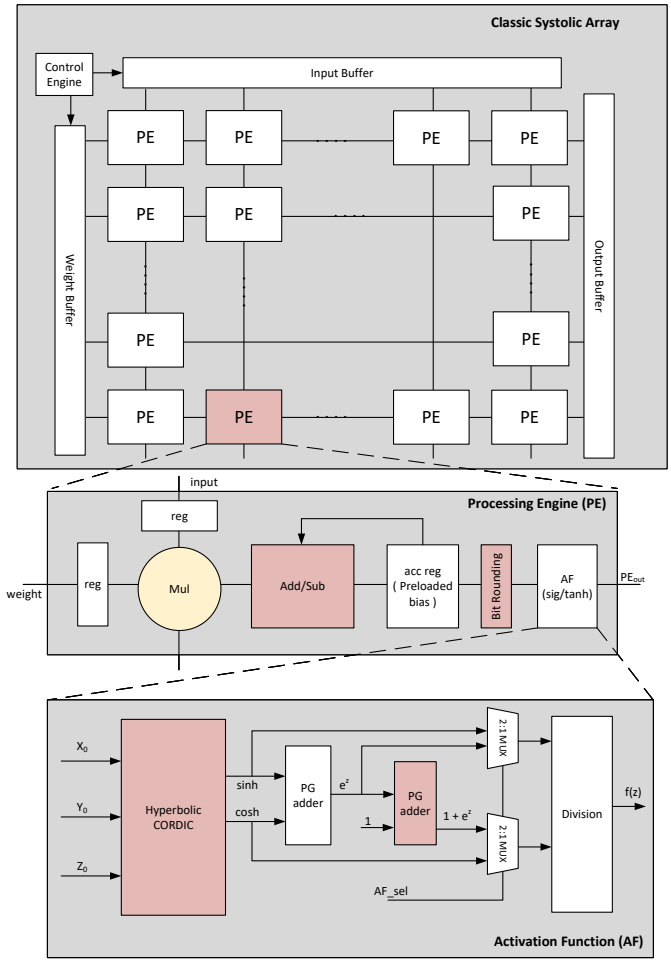


Figure 1.2 Classic 2D systolic array architecture with PE highlighting the regions to be improved.

The authors [15, 16] discussed the Area-Precision, Power-Precision, and Delay-Precision trade-off points for some of the approximate adders such as Almost Correct Adder (ACA-I/II), Quality-area optimal low-latency approximate Adder (QuAd), Equal Segmentation Adder (ESA), Error-Reduced Carry Prediction Approximate

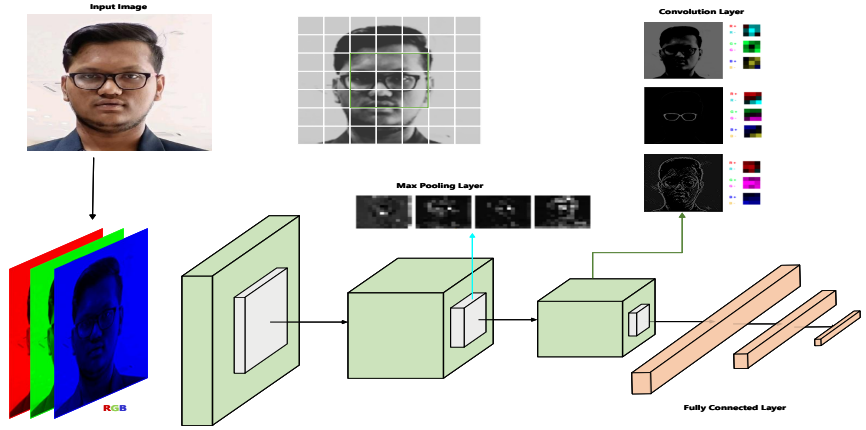
full Adder (ERCPAA), Generic Accuracy configurable adder (GeAr), Area-Power Efficient approximate adder (APEx), Error-Tolerant Adder (ETA-I/II), Lowest-cost Imprecise Adders (LIA) with error induced in the diverse application scenarios. ACA-I, ACA-II, and QuAd are designed to exploit the area-accuracy tradeoff by targeting less significant parts of an adder in an error-controlled manner. ESA, ERCPAA, and ETA-II focus on the decreasing critical path with less accurate computations while maintaining accuracy. ESA, GeAr allows the processing of additions in equal segments, configurable as per application requirements, and illustrates the basic principle behind HOAA concept. APEx, LIA focuses on the reduction of power for applications where exact results are not critical. Considering the error resiliency of neural networks, the techniques mentioned above may not affect output accuracy up to a certain point; however, beyond this threshold, enhancing physical design parameters often involves sacrificing accuracy, highlighting the importance of evaluating ideal trade-offs.

Table 1.1 State-of-the-art approximate adders and their implications on neural network inference accuracy

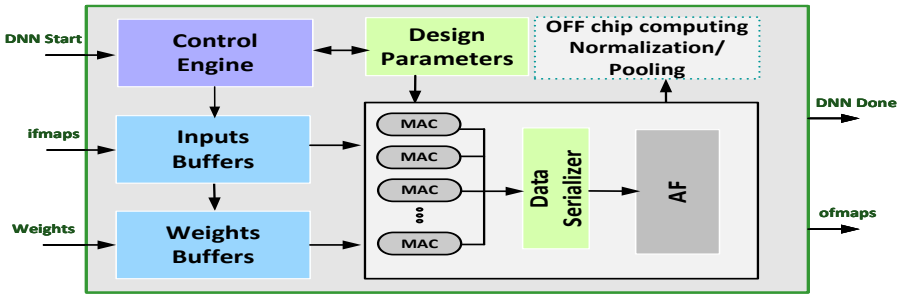
Adder Unit	**# of gates**	**Hardware Efficiency (Area-Power-Delay)**	**% Improvement**	**Accuracy Overhead(%)**
FA	40	N/A	N/A	0
HADD	32	Area, Power	50	-1.5
LOA	25	Area, Power, Delay	60	-3
ACA	32	Power	45	0
AMA	20	Power	70	N/A

1.1.1.2 Logarithmic Posit-enabled Reconfigurable edge-AI Engine (LPRE):

The rapid development of AI applications in Mobile and IoT devices has elevated a strong need for efficient data processing, as new edge computing devices suffer



(a)



(b)

Figure 1.3 Deep Neural Networks (a) A typical DNN model, (b) A classic DNN accelerator architecture showcasing data control and interface.

from many challenges when running AI workloads due to the memory wall and resource constraints [17]. To address the challenges, researchers have focused on deep neural network (DNN) accelerators to achieve speed, power, and performance improvements [18]. High-performance computing platforms, such as TPU, GPU, and ASIC/FPGA accelerators, are utilized rather than CPU-based serial execution. Significant research has been carried out in hardware-software co-design [19], architectural optimizations [3, 20], parameter reduction [21], and evolution of diverse number systems [1, 22] and data types [23].

DNNs rely heavily on billions of multiply-accumulate (MAC) operations, especially in convolutional and multi-layer perceptron layers. A typical DNN model with accelerator architecture is shown in Figure 1.3(a). Thus, optimizing the MAC units

is critical for improving system performance without creating bottlenecks in other components, including activation functions (AF), normalization layers (NORM), and pooling layers (POOL) [24]. State-of-the-art (SoTA) solutions, such as Systolic accelerators [25–27] and reconfigurable accelerators [5, 28, 29] typically focus on the resources consumed by MAC computations, while ensuring optimal data flow. Software optimization techniques, such as quantization [21,26], pruning [30], and using custom precision formats (e.g., fixed-point (FxP) [3, 27], posit [4], logarithmic [22], BF16, and TF32 [1, 6]) help significantly reduce hardware demands for efficient execution. However, these techniques may lead to accuracy degradation in some applications, thus affecting system performance [3, 5]. A comparison of SoTA methodologies (precision) with their impact on inference hardware is provided in Table 1.2.

Table 1.2 State-of-the-art precision and their impact on Hardware Accelerators for neural network inference

Precision	Data format	Max normal	Min normal	Subnormal	Resource Overhead	Accuracy Loss	Throughput
FP32	$1, 8, 23$	3.4028235×10^{38}	$1.17549435 \times 10^{-38}$	Yes	High	None	Low
FP16	$1, 5, 10$	65504	6.1035156×10^{-5}	Yes	High	Minor	Medium
FP8	$1, 4, 3$	240	0.015625	Yes	Low	Moderate	Medium
INT32	$1, 31$	2.147×10^{9}	-2.147×10^{9}	No	Medium	Minor	Low
INT16	$1, 15$	32767	-32768	No	Low	Moderate	Medium
INT8	$1, 7$	127	128	No	Low	Moderate	High
POSIT32	$1, r, es, (n-r-es-1)$	1.84467×10^{19}	5.42101×10^{-20}	No	Medium	None	Medium
POSIT16	$1, r, es, (n-r-es-1)$	4.29×10^{9}	2.33×10^{-10}	No	Low	None	High
POSIT8	$1, r, es, (n-r-es-1)$	65536	1.526×10^{-5}	No	Low	Minor	High

This work majorly focuses on a Logarithmic Posit-based arithmetic reconfigurable architecture, which maintains application accuracy (compared to baseline FP32) while saving significant hardware resources. An input-output multiplexing scheme and sparsity exploitation further enhance the memory bandwidth and computation efficiency.

1.1.1.3 CORDIC-Neuron

ASICs are specialized hardware optimized for a specific task, sacrificing the adaptability of a reconfigurable architecture for improved power efficiency compared to platforms such as FPGAs and GPUs. These latter platforms, while more versatile

and capable of executing a variety of tasks, tend to have higher power consumption [31, 32]. FPGAs provides cost efficiency, and flexibility, with limited on-chip memory and resources [32]. Thus, innovative design methodologies are required to manage the area, complexity, and reconfigurability challenges in DNNs. Additionally, a DNN comprises various layers, including CNN, feed-forward, normalization, long-term short-term memory (**LSTM)**, Gated Recurrent Unit (**GRU**), pooling, and activation layers like Rectified Linear Unit(**ReLU**), tanh, and sigmoid [8,33]. The core components of a DNN model are the Processing Engines (PEs), or artificial neurons. These involve multiplication and accumulation processes followed by a nonlinear AF. Multiplication and accumulation operations, handled by MAC units through complex adder and compressor design, are computationally intensive. The output is then subjected to nonlinear transformations using AFs such as sigmoid, tanh, and ReLU, commonly employed in DNNs [34]. Activation functions are crucial in DNN architectures to enable features like model size reduction and prevention of overfitting issues [35]. Various methods, including LookUp-Tables (**LUT**), COordinate Rotation DIgital Computer (CORDIC), and PieceWise Linear (**PWL**), can implement these Activation Functions. While some designs are resource-intensive in terms of area and power, the CORDIC design is praised for its area and power efficiency; however, a considerable limitation is its low throughput.

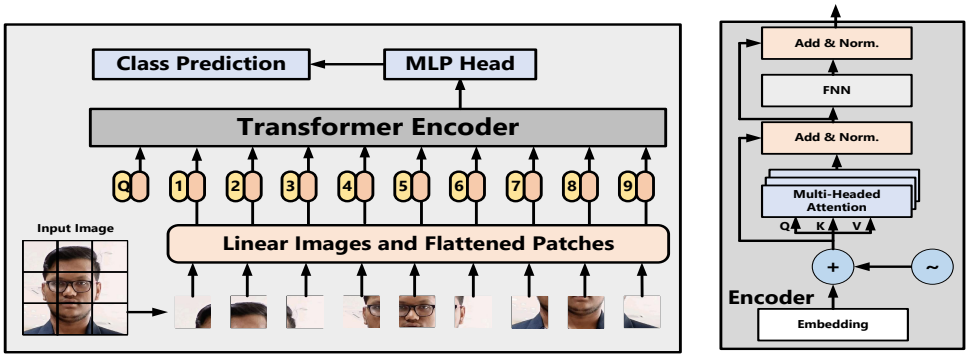


Figure 1.4 Typical demonstration of Transformers model, depicted intermediate layers such as Conv, Pooling, MHA, and FFNN

Prior AI accelerator designs prioritize area and power efficiency and focus on the

CORDIC computational algorithm [33, 36–38]. This design comprises fundamental logic components, including a shift register, adder/subtractor circuits, multiplexers, and some memory elements that facilitate a range of arithmetic operations while consuming less power. CORDIC algorithm, introduced by Jack E. Volder in 1959, has the adaptability to conduct several arithmetic operations. Mathematically, it achieves linear convergence with minimal resource use [34]. Through shift and add-sub operations, the CORDIC architecture can execute numerous tasks, such as trigonometric, hyperbolic, and logarithmic functions via vector rotation [39]. We employ CORDIC in rotation mode to achieve the tanh and sigmoid functions, akin to computing hyperbolic functions as exponential terms. Moreover, CORDIC can determine square roots and more. The widely used SoftMax function can also be implemented through CORDIC hardware, which is crucial for various classification tasks and is predominantly used in transformer architectures and capsule networks.

1.2 Research Objectives

The primary objective of this research is to investigate and develop a novel AI accelerator architecture for Edge AI applications. The specific research objectives include:

- Designing a HOAA-enabled CORDIC PE, with a focus on improving speed, energy efficiency, and area efficient compared to conventional SoTA designs.
- Developing efficient posit engine designs and performance tradeoff to boost computation within Vector Engine, leveraging the AI workload characteristics (sparsity).
- Implementing a architecture CORDIC-based RISC-V enabled Systolic array: AI SoC and conducting comprehensive performance evaluations to validate the effectiveness of the proposed architecture in real-world scenarios.
- Exploring the potential applications and practical implications for various computing tasks, such as neural networks, and AI inference.

1.3 Organization of Thesis

This thesis is organized into several chapters, each focusing on different aspects of the research conducted on AI accelerator architectures for resource efficient edge AI acceleration. The chapters are structured as follows:

Chapter 1: This chapter provides the background and motivation for the research, highlighting the increasing demands of data-intensive applications and the need for high-speed, energy-efficient solutions.

The literature review offers a comprehensive overview of DNN, including existing accelerator implementations and various types. This chapter sets the stage for the novel contributions of this research by discussing the limitations of current technologies and the potential improvements with CORDIC circuit. It introduces Systolic array architectures required for the data-intensive applications. The chapter concludes with the research objectives and a brief overview of the thesis organization.

Chapter 2: This chapter details the design and implementation of HOAA-optimized PE design to enhance the performance. The work covers the structure of the novel HOAA, its operation, incorporation into CORDIC-PE including an its PPA impacts.

Chapter 3: This chapter details the design and implementation of PQRE to incentivize the posit-based MAC, AF operations. The work covers the structure of the novel compute unit and its incorporation into LPRE structure, including an analysis of the its impacts and a performance evaluation at the FPGA prototype.

Chapter 4: This chapter details the design and implementation of resource-constrained CORDIC PE design to enhance the performance of systolic array. The work covers the structure of the novel PE, its operation, incorporation into systolic array structure and integration with RISC-V, including an performance evaluation for the edge applications.

Chapter 5: The final chapter summarizes the key findings and contributions of the research. It discusses the implications of the novel HOAA-enabled PE , LPRE and CORDIC-based systolic array design with RISC-V integration. Additionally, it

outlines potential directions for future research to further enhance the performance and applicability.

Chapter 2

HOAA: Hybrid Overestimating Approximate Adder for Enhanced Performance Processing Engine

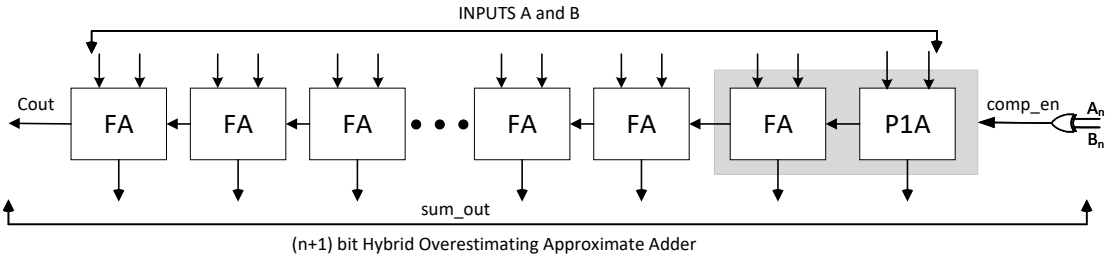


Figure 2.1 N - Bit Re-Configurable Adder, (n+1) bit HOAA.

There has been a sudden rise in the development of edge AI chips due to the high demand for running complex models and the need for optimized GEMM operations, which dominate the inference time. For instance, Google's Tensor Processing Unit (TPU) comprises 64K MAC units, and Samsung's Neural Processing Unit (NPU) has 6K MAC units [**?**,16]. The DNN workloads LeNet, AlexNet, ResNet-50, and VGG-16 require 0.34M, 724M, 3.9B, and 15.5B MAC units [**?**,11,12]. The processing element (PE) is the basic compute unit in a Convolutional Neural Network (CNN) accelerator, contributing to 90% of energy efficiency. The PE comprises a multiplier followed by an adder, combined as MAC and AFs like Sigmoid/Tanh. The fundamental block in the shift-add/sub-based CORDIC implementation [12] of PE is an adder. The equation `2.1` calculates the Sum and Carry in conventional FA.

$$\begin{aligned} \text{Sum} &= A \oplus B \oplus C_{\text{in}} \\ \text{Cout} &= (A \cdot B) + (C_{\text{in}} \cdot (A \oplus B)) \end{aligned} \tag{2.1}$$

Various Adders have been proposed with different FA and HA combinations to fit the appropriate power, performance, and area trade-offs as shown in Table. 1.1 according to various application scenarios. RCAs involve sequential carry propagation of parallel carry determination in CLA [12]. CBA selectively bypasses critical carry to optimize delay while CSA uses MUX to parallelize carry, showcasing area-speed tradeoff. While serial adders are suitable for low-power applications, Tree adders support distributed arithmetic with a hierarchical structure. BKA and KSA leverage parallel prefix carry computations suitable for high-performance, scalable applications. Further, Given the inherent error-tolerant nature of CNNs,

approximate computing methods have been explored widely for high hardware efficiency in implementing accelerators. Approximation is a vital element in enhancing physical design parameters and simplifying complex architecture by exploiting natural irregularities in real-time systems. The approach addresses resource management, with efficient circuits and helps for achieving faster and reduced computational demands. The design of approximate circuits involves using Majority-logic, modifying or removing logic gates from accurate circuits, or simplifying the Karnaugh map (K-map). From the initial application of approximate adders to boost the clock frequency of microprocessors in 2004, the AC applications have been automated with various innovative designs like the almost correct adder (ACA), the error-tolerant adder (ETA), the equal segmentation adder (ESA), and the approximate mirror adder (AMA), etc. The inherent redundancy in neuron computations and minimal degradation due to approximation errors open opportunities for trade-offs in energy efficiency, and compute density, making them ideal for resource-constrained environments like edge-AI devices. Additionally, the use of approximate adders can improve speed and throughput, leading to faster response time. Though beyond a certain error threshold, the performance of neural networks degrades noticeably, the approximation shall be tuned for efficient inference across diverse applications. A hybrid combination of conventional adders for MSBs and approximate adders for LSBs ensures minimum accuracy degradation (less than 5%) with the equations provided in `2.2`. This also ensures significant hardware efficiency in multi-bit fixed-point precision adders.

$$\begin{aligned} \text{Sum} &= (A + \text{Cin}) \oplus B \\ \text{Carry} &= (A + \text{Cin}) \cdot B \end{aligned} \tag{2.2}$$

PE has been pivotal for the efficiency of the DNN accelerator as it contributes to 90% operations. Despite huge efforts put in to enhance the performance, the PE encounters major performance fallback linked with the +1 operation. In **Signed MAC units**, +1 operation is crucial during Two's complement subtraction. In the quantization, **Rounding to even** technique employed within PE requires

+1, contributing to one cycle delay. **Configurable Activation Function** with CORDIC methodology involves frequent +1 operation affecting the performance of overall PE and, subsequently, the entire systolic array DNN accelerator. The overhead would typically be the delay of one clock cycle in serial adders or area overhead in parallel adders, along with huge power consumption, depending on the bit precision of operands involved. An approximate P1A has been proposed to address the limitation, which performs the +1 operation within the same processing cycle. Therefore, the associated area or latency could be reduced by trading off slight accuracy in the runtime. We further analyzed the impact as well. Further dynamically reconfigurable P1A within the RCA chain has been proposed as novel HOAA. Hybrid approach-based HOAA ensures the runtime reconfigurability between FA and P1A enabling future scope for OEA. Different bit-width segments can be utilized with an approximate multiplier based on an error-controllable mechanism, fundamentally improving accuracy loss due to logarithmic MAC operations.

The Chapter talks about an n-bit Hybrid Overestimating Approximate Adder to simplify advanced arithmetic operations with minor area overhead. The principal contributions of this work are :

1. **Resource efficient Plus One Adder (P1A):** An incremental approximate adder is proposed that can be incorporated into the RCA chain, which provides excess-1 output. The design approximates certain outputs to reduce hardware overhead; Hence, corresponding error metrics are also evaluated.

2. **Dynamically Re-configurable HOAA:** HOAA adder architecture is proposed to be capable of runtime interchangeability by replacing the m-LSB FAs of RCA with P1A. This saves one clock cycle and supports multiple arithmetic operations with the same hardware blocks.

3. **Evaluations for versatile test cases:** The proposed design is evaluated on Two's complement subtraction, Rounding-to-even and Configurable AF. Monte Carlo-based error distance calculations are provided. The ASIC physical design performance parameters are evaluated at `CMOS 28nm`.

2.1 Proposed Architecture

We discuss the implementation of a P1A adder and extend it to HOAA architecture. We also evaluated the error analysis for the use cases with corresponding error metrics to demonstrate resource-efficient design. [29]

2.1.1 Proposed P1A Design

Signed MAC operations use an RCA chain with FAs in the accumulator stage. Tradition FA *2.1* uses five logic gates are used to compute the sum and carry. However, reusing the adder as Two's complement-based subtraction typically consumes two-cycle latency and power consumption for N-bit operation. We identified the necessity of a +1 adder to address the issue and proposed the circuitry based on a modified truth table *2.1* with the Karnaugh Map technique. Further, to obtain the approximate P1A, we evaluated the error metrics and corresponding hardware costs. The accurate +1 adder and approximate P1A equations are shown in Eq. *2.3*, *2.4* respectively. Compared to FA, the proposed P1A uses only three logic gates, highlighting hardware efficiency. The incorrect outputs are highlighted in Table *2.1*. The truth table shows two errors in sum and carry-bit. Further, the error rate analysis is provided to justify the proposed approximations.

$$\begin{aligned} \text{Sum} &= A \cdot C_{in} + B \cdot A + B \cdot C_{in} + \overline{A} \cdot \overline{B} \cdot \overline{C_{in}}, \\ \text{Carry} &= A + B + C_{in}. \end{aligned} \tag{2.3}$$

The circuit modifies with circuitry around to provide two modes with the same HOAA(N,m) to support both operations. The FAs are utilized during normal addition while keeping the P1A block disabled with PG for normal addition. Conventional N-bit RCA was modified typically by replacing the LSB adder with P1A for subtraction operation as illustrated in Fig. *2.1*, which also ensures the direct addition of excess-1 to LSB position and faster subtraction operation within the same cycles as that of addition. Notably, with increased adder size, the error introduced by P1A vanishes, approaching better suited for scalable architectures with minimal static

Table 2.1 Truth table for accurate and approximate P1A adder designs, including all input combinations, output sum, and carry bits

A	B	Cin	Accurate P1A Output			Approx. P1A Output	
			Sum	Cout	Cout2	Sum	Cout
0	0	0	1	0	0	1	0
0	0	1	0	1	0	0	1
0	1	0	0	1	0	0	1
0	1	1	1	1	0	1	1
1	0	0	0	1	0	1*	0*
1	0	1	1	1	0	1	1
1	1	0	1	1	0	1	1
1	1	1	0	0	1	1*	1*

power consumption.

2.1.2 Proposed HOAA Design

We evaluated two approaches to fit the proposed P1A into the adder design to support subtraction within the same processing cycle adaptively. The first approach was based upon Reconfigurable Approximate CLA, which would provide the reconfigurable ability to switch between approximate and accurate modes with MUX. The second approach involved using (N,m) adders as a combination of conventional and approximate adders, which has also been standard practice for hardware efficiency with minimal accuracy degradation. The runtime selection signal comp_en has been generated based on MSBs of both operands for the selection between P1A and HA. This enables generalization and provides a template for a scalable architecture. Further, the approximate adders could be power gated when

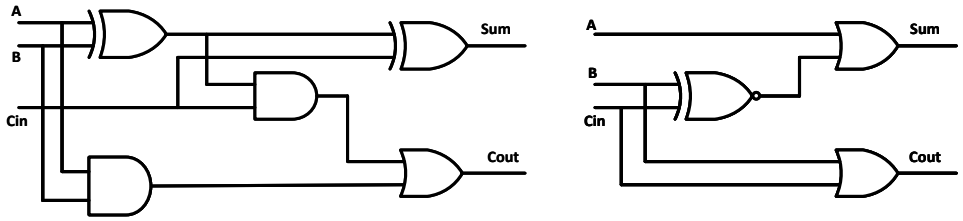


Figure 2.2 1-bit Schematic of a) Conventional Full Adder b) Plus One Adder.

not required; however, this has not been evaluated in this work. [5]

$$\begin{aligned} Sum &= A + \overline{B \oplus Cin} \\ Cout &= B + Cin \end{aligned} \tag{2.4}$$

Delay is one of the important parameters in PPA analysis. Even the SOTA studies highlight the importance of varied propagation delay per individual logic gates' properties. The propagation delay analysis for the sum and carry-bit of 1 bit P1A can be found in equations *2.5*. The further analysis of (N,m) HOAA follows the methodology per HADD/LOA adders. The equation *2.4* highlights linear area savings as the number of gates increases with bit-precision. Compared to the conventional adder, this would be an additional reduction in delay for any n-bit precision operation due to P1A.

$$\begin{aligned} T_{sum} &= T_{xnor} + T_{or} \\ T_{carry} &= T_{or} \end{aligned} \tag{2.5}$$

2.2 Methodology & Results Discussion

The Conventional full adder (FA) truth table was modified by directly adding +1 to the Cout, Sum outputs while calculating the P1A equations. This modified truth table was then converted to Boolean equations using Karnaugh maps (K-maps), producing accurate equations *2.3*. However, the decision to introduce approximation was made due to the increased hardware cost of the equation. After evaluating numerous cases of the proposed addition and the corresponding accuracy loss, an

approximate +1 adder was proposed, as per equation *2.4* and error analysis is mentioned in Table *2.2*. The timing analysis can be done for a single P1A as per equation *2.5*, replacing the combinational delay for a particular HA within RCA. Compared to conventional FA (28T), P1A requires only 16 transistors, significantly saving the CMOS area.

The proposed design has been validated with hardware-software co-design emulation flow for the experimental analysis. `Google Colab Notebook` with `Python v3.0` has been utilised for error metrics calculations as per [12, 15, 16]. The functional emulation for the proposed hardware architecture was done with the of `Python v3.0` and the error metrics were calculated as shown in Table *2.2*. The quantitative measurement of error introduced due to approximation, and its impact on performance, and quality of approximate neural networks, further to assist designers in understanding the more suitable trade-offs for particular applications, error metrics are essential. The error metrics to evaluate the accuracy loss incurred include Mean Square Error (MSE), Normalised Mean Error Distance (NMED), and Mean Relative Error Distance (MRED) [12, 15]. MSE provides overall error magnitude, as in the average squared difference between exact and approximate results. The MSE less than 2% shows the efficiency of design for all the test cases. NMED and MRED give a relative measure of error and indicate error scaling with architecture. MRED and NMED around 6% show the comparison of errors across different scale and application scenarios. Further, The Verilog RTL code for the proposed Adder was written and functionally validated using `Xilinx Vivado 2023.2` for hardware implementation. During ASIC analysis, the 8-bit HOAA is synthesized with `Synopsys Design Compiler` using `CMOS 28nm PDK` at a power supply of 0.9V for the configurations shown in Table *2.3*. The physical parameters were evaluated at the `HPC+` node, considering the better compute density, reduced area-power-dealy, and industry-adopted solution for resource-constrained scenarios. Monte-Carlo simulations for uniformly distributed 2^{n+1} times random input pattern variable. For the calculation of error metrics *2.2*, the simulations were computed on random inputs and were compared with true outputs from `Python Numpy`. Approximation trade-offs were evaluated to check the

Table 2.2 Error metrics evaluation for 8-bit HOAA design

∗ All values are in percentage scale

Error Metrics	Case-I	Case-II	Case-III
MSE	0.02444	0.02406	-0.06766
NMED	0.02444	0.02406	0.06766
MRED	0.06834	0.06729	0.06759

accuracy and compare the adder architectures. The proposed adder signifies minimal accuracy loss compared to LOA while absolute gains in PPA design parameters relative to FA, AMA, HADD, and SESA-I. P1A shows an improvement of around 5% in area compared to HADD [13] and AMA [40], while consuming approximately 15% less power than HADD [13] and SESA-I [41]. Additionally, P1A demonstrates a 21% reduction in area and a 33% reduction in power consumption compared to the conventional FA. The slack calculation is performed with a period of 10ns or an operating frequency of 100 MHz, and the value reflects a smaller improvement in timing margins compared to FA and hence, a higher operating frequency. The count and placement of reconfigurable P1A in HOAA can be error-controlled while the decision to choose HOAA can be made on the evaluation of MSE, NMED, and MRED (Refer to Table 2.2) vs physical parameters (Refer to Table 2.3). However, while the error metrics are preliminary factors, a thorough analysis should be carried out for real-world DNN workloads.

2.2.1 Case-I: Subtraction

Two's complement-based subtraction in signed computer arithmetic is the simplest and most efficient methodology followed. It simplifies subtraction between two binary numbers by modifying into the addition problem, which is convenient from a reconfigurable hardware implementation perspective and overflow handling. Negative numbers are represented with two's complement format by calculating one's complement/simply reverting the number, followed by adding one to LSB, which is a

Table 2.3 Physical Design Parameters at CMOS 28 nm, VDD 0.9V, frequency 100 MHz

Attributes	Area(μm^2)	Power(μW)	Slack(ns)
FA	8.736	1.164	1.87
HADD [13]	7.392	0.649	1.91
SESA-1 [41]	6.384	0.921	1.93
LOA [42]	4.032	0.567	1.98
AMA [40]	6.552	0.810	1.93
P1A	6.888	0.782	1.93

two-cycle process. The effective computational resources required for the process are two-cycle latency and power consumption of N-bit adders, where the second cycle is just utilized for adding "1". P1A can be effectively used to reduce the design complexity and enhance the throughput for such operations in Edge-AI applications.

The differences between accurate and approximate computations have been measured with error metrics like MSE, NMED and MRED and reported in the table *2.2*. It is also noteworthy that overall computational accuracy ain't much affected for 8-bit due to the introduction of P1A in LSB of HOAA and is reflected in overall better results. Hence, it can be concluded that the proposed HOAA can be a good choice for designing reconfigurable subtraction hardware.

2.2.2 Case-II: Rounding-to-even

In neural network computations, MAC units sum up the products of numbers, increasing precision. To support the limited precision requirement at Edge-AI, the numbers are often required to undergo quantization, resulting in rounding errors. Regular rounding accumulates the statistical bias due to long sequence operations in DNN, either in a positive direction or negative direction. Hence, the `IEEE754-2008`

roundTiesToEven technique is more suitable as rounding errors are canceled due to adding *1* when odd numbers are rounded down. However, the two-cycle process is more hardware-intensive due to the presence of shifters in two cycles, where one cycle is wasted simply for adding *1*. This can be effectively optimized by introducing P1A, which results in the further reduction of computational resources. Our results from Table *2.2*, *2.3* highlight the energy efficiency of the design with minimal accuracy loss, even less than 1% error.

2.2.3 Case-III: Configurable activation function

$$\begin{aligned} e^{Z} &= \cosh(Z) + \sinh(Z) \\ sigmoid(Z) &= \frac{e^{Z}}{e^{Z}+1} \end{aligned} \tag{2.6}$$

DNN inference models rely heavily on Non-linear AF to characterize input-output relationships. The hardware implementation methodologies for the implementation of reconfigurable activation functions are resource-intensive, and popular ones like *sigmoid* and *tanh* often require huge optimizations for Edge-AI applications. Iterative CORDIC-based methodology has been proven hardware resource-efficient yet affects critical delay. The runtime reconfigurable choice between *sigmoid* and*tanh* has been provided with AF_sel signal. The CORDIC approach involves the computation of hyperbolic sine (*sinh(Z)*) and cosine (*cosh(Z)*), requiring two-stage adders before division operation to derive the *sigmoid* function as shown in Eq. *2.6*. Our HOAA approach could easily enhance throughput by minimizing adding *1*. Furthermore, Table *2.3* discusses the impact of approximation introduced by P1A, which is negligible. Case *III* also demonstrates the scalability of our approach for fixed-point arithmetic, broadening P1A's applicability beyond the integer computations discussed earlier.

Chapter 3

LPRE:Logarithmic Posit-enabled Reconfigurable edge-AI Engine

Conventional edge-AI accelerators, based on systolic dataflow [25, 26], excel at convolutional operations but often suffer throughput loss for multi-layer perceptrons or recurrent neural network layers. Thus, reconfigurable accelerators [3, 5, 20] have been proposed, but these works are limited to fully connected layers. These architectures can be extended to convolutional layers with control engine and data flow modifications. It can benefit more from software-exploited sparsity than runtime pruning [30]. Sequential execution with time-multiplexed reconfigurable single-layer hardware enhances resource efficiency in resource-constrained environments in a resource-constrained environment, while hardware reuse (AF, POOL) [5,26] improves compute-density (TOPS/mm^2).

As an example, VGG-16 requires substantial 15.5 billion MAC operations to process a single 224 X 224 image. Optimising resource distribution (e.g., dynamic channel pruning [18, 30]) significantly improves edge-AI systems to reduce MAC workloads without affecting accuracy. While fixed-point or logarithmic computations [2, 3] are power/area-efficient for hardware implementations, reducing precision degrades output accuracy severely.

Logarithmic [22] and Posit [19,23] arithmetic present promising choices, delivering BF16/FP16-level accuracy at 8-bit and FP32 accuracy at 16-bits [1]. This reduction in data width also enables SIMD computation for enhancing the throughput [4]. The Logarithmic-Posit (LP) arithmetic unit [43] offers a formidable solution for on-device learning. Integrating a PQRE accelerator with RISC-V [29] or any general processor efficiently enhances hardware-software codesign [19] and improves existing edge-AI solutions.

The principal contributions of this work are as follows:

- **Logarithmic Posit Quire Processing Engine (PQRE):** The proposed PQRE utilises Posit arithmetic to reduce hardware resources without affecting computational accuracy. Furthermore, bit-width reduction and SIMD computations are applied, enhancing throughput by up to 4×.

- **Time Multiplexed Reconfigurable edge-AI engine:** This architecture

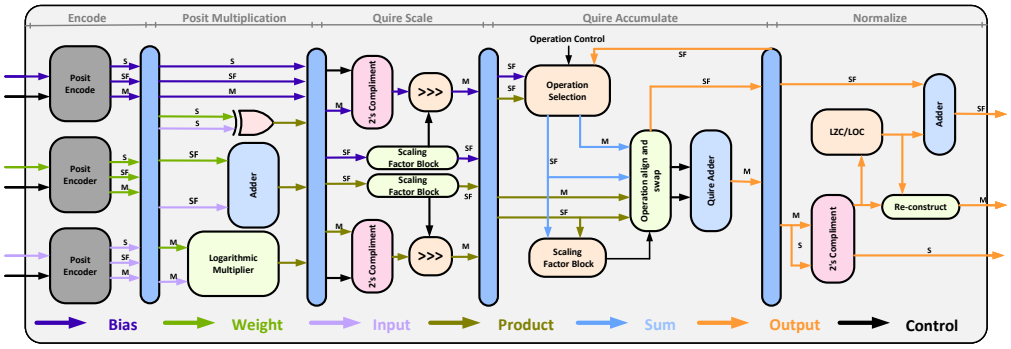


Figure 3.1 The detailed microarchitecture of 5-stage Logarithmic Posit Quire Processing Engine (PQRE).

improves compute efficiency and bandwidth by time-multiplexing the reconfigurable single-layer hardware and input-outputs by a factor of (N-1). It dynamically configures by layer to exploit sparsity and AF-reuse techniques, effectively enhancing the execution of N-layer MLPs in resource-constrained environments.

3.1 Proposed Design

The optimization of edge-AI accelerators often focuses on the design of the MAC unit. This work presents the Logarithmic Posit Quire Processing Engine designed to reduce hardware resources without compromising computational accuracy. The proposed LPRE utilizes the Posit number format, which is represented using sign, regime, exponent, and fraction, as follows:

$$-1^{s} \times 2^{e+k\times 2^{es}} \times (1.m), \tag{3.1}$$

where s is the sign bit, e is the exponent, k is the number of regime bits, es is the maximum exponent size, and m is the mantissa. The scaling factor ($2^{e+k\times 2^{es}}$) ensures a wide dynamic range with dynamic scaling from the regime and exponent values. Furthermore, Quire accumulation can be applied during intermediate stages to ensure output precision with the Quire Size calculated as follows:

$$Quire\ Size = 1 + cg + 2^{es} \times (n-2), \tag{3.2}$$

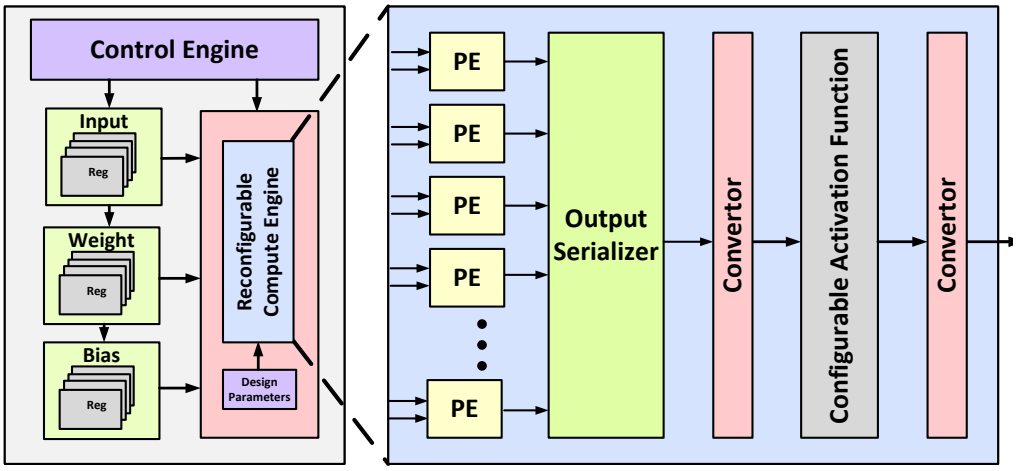


Figure 3.2 Proposed Logarithmic Posit-Enabled Reconfigurable edge-AI Engine architecture.

where *cg* are accumulation bits for preventing overflow.

Figure 3.1 depicts a 5-stage datapath designed to support 32-bit SIMD Quire MAC operations while allowing runtime switching between a single 32-bit operation, two 16-bit or four 8-bit parallel operations. The Posit arithmetic encode/decode logic block includes logic for dynamic exponent configuration with dynamic vector mode, supporting parallel computations in lower mode with output sign calculated via XOR of sign bits. The output of the preceding stage is sign-extended, while the leading zero counter (LZC) decodes the bit length of the regime. The multiply stage implements an adjustable parallel or iterative logarithmic multiplier based on overall area overhead. Run-time configurable precision and/or vectorization can be enabled with a variable Carry Propagate/Look-Ahead (CPLA) adder and configurable logarithmic barrel shifters for parallel accumulation.

The Quire Scale and Quire Accumulate stages align outputs from Quire-fixed-point format with vector shifter while preventing overflow due to accumulated bits (*cg*). The Normalise stage handles exception flags, converts Quire to unsigned values, and renormalizes output via zero/one counts (LZAC). The enhanced PQRE facilitates 4 X SIMD computations at reduced-bit width, following detailed previous work [4]. This allows more resources to be dynamically allocated for major parallel workloads, boosting application performance.

Our proposed PQRE improves accuracy loss from fixed-point-based accelerators [3, 5] without any additional area overhead. We incorporated our logarithmic-posit arithmetic-based PQRE into a layer-multiplexed accelerator and extended the work to support the convolutional layer using *im2col* software data rearrangement. The control engine dynamically allocates MAC units for a single reconfigurable layer based on data flow and can be configured at runtime based on the *Neuron-count* parameter. The other design parameters include the number of layers in the DNN model, image size ($N \times M$), kernel size ($k \times k$), stride, pooling type, etc. For instance, the first layer of LeNet-5 requires 576 MAC units for a single 5 X 5 kernel on a 28 X 28 image, followed by Max pooling.

The architecture efficiently manages the data flow through dedicated input and weight buffers, as shown in Figure 3.2. Workloads are mapped into a one-dimensional PQRE array, and the input-multiplexing scheme maximizes bandwidth utilization while fetching the input-weight feature maps to on-chip memory via AXI. Enhanced throughput PQRE significantly reduces latency, with outputs serialised to a configurable AF. To reduce area overhead and support the variable needs of different AI workloads, we incorporated a CORDIC-based runtime-configurable ReLU/Sigmoid/Tanh activation function. The dual-path architecture, a Posit-to-Fixed-Point converter compatible with CORDIC-based FxP AF and Posit-based AF, allows dynamic execution per workload type. CORDIC-based fixed-point AFs are proven to be hardware efficient [6], while Posit AFs are less explored. We incorporated AF-reuse methodology within the reconfigurable layer to reduce the area overhead since most AI workloads characterize AF operations less frequently than MAC operations. This approach saves resources consumed by (N-1) layers and (N-1) activation functions within a layer compared to Systolic-based computations. This methodology also helps us achieve better tiling opportunities whenever necessary. For larger workloads, an iterative tiling mechanism triggers the execution of the next tile via the *Compute_Done* signal. After completion of the FC/Conv layer, the *Layer_Done* flag is generated, and after execution of all layers, *DNN_Done* reports the completion of the AI workload to the master processor. Additionally, we explored

Table 3.1 Comparison of FPGA Resource Utilization for proposed PQRE with SIMD FP MAC Unit [1] and SoTA MAC works [2–4]

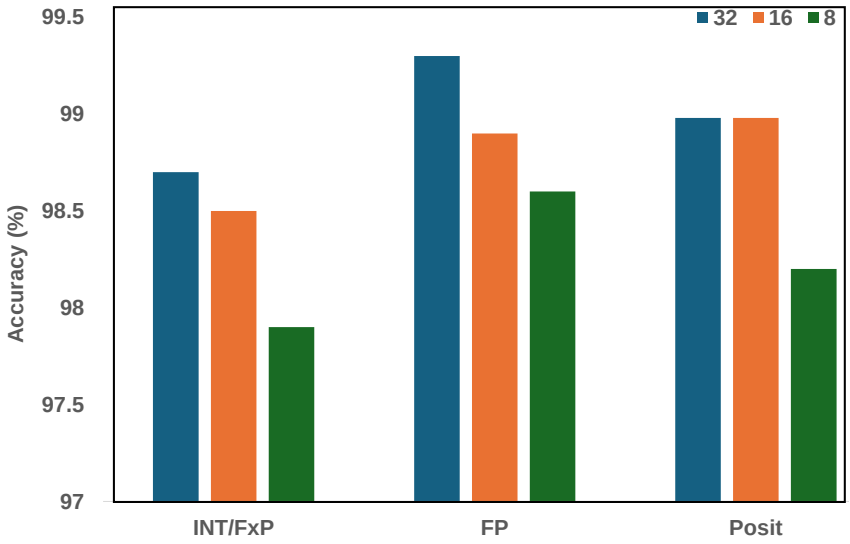
Data Type	LUT	Reg	Op. Freq (MHz)
FP32	8065	1072	250
Posit(32,2)	6813	806	125
BF16	3670	324	200
FP16	2226	1062	357
Posit(16,1)	2083	528	230
Int/FxP32	1356	208	153
FP8	645	255	434
Posit(8,0)	467	175	372
Int/FxP16	326	32	345
Int/FxP8	86	16	424

50% dynamic channel pruning [18] with modified dataflow to reduce MAC workloads by half and further enhance throughput. The overall architecture offers significant improvement over state-of-the-art works, particularly in resource-constrained edge AI accelerators, with reconfiguration, hardware reuse, and removal of redundant computations.

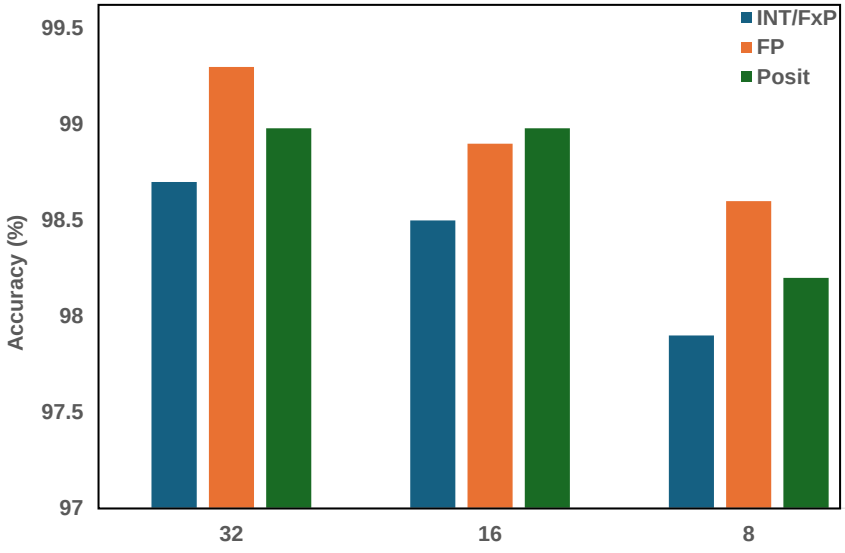
3.2 Methodology & Results

The critical requirements for edge-AI devices include quick event-triggered response, superior energy efficiency, and optimized compute density across diverse application scenarios. Key metrics, such as power consumption, latency, operating frequency, hardware utilisation, and accuracy, were evaluated to determine the efficiency of the proposed LPRE accelerator. The LPRE replaces conventional fixed-point or floating-point arithmetic-based processing engines with quantization-aware LP computations to validate the application accuracy and hardware performance at the edge node.

Inference accuracy for the proposed PQRE was compared with SoTA work [3]



(a)



(b)

Figure 3.3 Inference accuracy evaluation with the proposed Logarithmic Posit Quire Processing Engine compared to the baseline FP [1] and SoTA INT/FxP [2, 3] accuracy. (a) LeNet-5 on MNIST (b) AlexNet on CIFAR-10.

Table 3.2 Hardware Utilization Report for Various Architectures with CORDIC-based FxP MAC and Our PQRE

8-bit precision	Integer/Fixed-Point Arithmetic			RAMAN	Pruning	Posit Arithmetic		
Arch. Type	DNN-Layer Reused [3]	DNN-Fully Parallel [5]	FC Reused [2] Layer-Multiplexed	Reconfig-Sparse [25]	Sparse-Systolic [30]	DNN-Layer Reused [3]	DNN- Fully Parallel [5]	Proposed Work
LUT	112654	207908	19834	37200	40780	139561	261703	17539
FF	113648	306874	12081	8600	45250	163231	316406	14827
BRAM	32	53	13	61	257	24	57	16
Power (W)	1.097	4.07	0.78	0.137	1.4	1.05	2.08	0.93

and Tensorflow-based models [2] across 8/16/32 bit-precision using LeNet-5 on the MNIST dataset and AlexNet on the CIFAR-10 dataset, as depicted in Figure 3.3(a). The proposed PQRE achieves minimal accuracy loss at 8-bit and no accuracy loss at 16-bit compared to the FP32 baseline, demonstrating better adaptability for edge-AI applications with improved hardware efficiency. Posit arithmetic also significantly improves accuracy for larger networks, such as VGG-16, ResNet-18, and Inception-v3 [23]. Thus, further evaluations were not conducted in this work.

We implemented the PQRE, configurable AF and LPRE design on a *Xilinx Virtex-7* FPGA Board to further analyse performance. SoTA designs were also re-implemented on the same board to maintain similar parameters for a fair comparison. The performance parameters include the LUTs and FFs of the FPGA board, showcasing the resources used while operating frequency is evaluated with a focus on the response time of the edge application. The resource utilization for the proposed PQRE and SoTA MAC architectures are presented in Table 3.1. Similar evaluations for activation functions are compared with [5, 6] as presented in Table 3.3. The results show a substantial reduction in hardware resources, as compared to SoTA works [2,3,25], with enhanced throughput (4 X at 8-bit, 2 X at 16-bit). The Posit-to-Int/FxP conversion utilises 325 LUTs and 300 LUTs for Int/FxP-to-Posit conversion. Thus, the area overhead showcasing the Posit activation function seems an easier and more viable way. These outcomes showcase runtime-reconfigurability and enhanced compute efficiency of the proposed PQRE across different precisions. To ensure a fair comparison, We re-implemented the state-of-the-art fully parallel architecture [5], DNN-layer reused architecture [3], and proposed architecture with fixed-precision based QuantMAC [2] and Posit-arithmetic-based PQRE. Figure 3.2 depicts the

Table 3.3 Comparison of FPGA Resource Utilization in Different Precision and Datatype [3, 5, 6] with Proposed Posit AF

AF	Sigmoid			Tanh		
Precision	LUT	Freq (MHz)	Power (W)	LUT	Freq (MHz)	Power (W)
FP32	5101	9.12	0.121	4298	17.64	0.13
FP16	1853	16.45	0.118	1530	29.3	0.124
BF16	1856	22.39	0.083	1513	26.35	0.083
TF32	2436	16.16	0.117	1990	23.6	0.118
FxP8	156	262	0.0395	156	262	0.0395
FxP16	267	238	0.0483	267	238	0.0483
FxP32	478	154	0.0649	478	154	0.0649
Posit(16,1)	2275	16.38	0.098	1879	26.27	0.089
Posit(8,0)	1468	21.4	0.089	1548	29.8	0.078

LeNet-5 hardware architecture to demonstrate the proposed approach on a working FPGA prototype. The proposed architecture was implemented with 64 PQREs and a runtime reconfigurable control engine supporting LeNet-5 execution. The LPRE utilised approximately 50% of the resources required by SoTA-works [25, 30] and 80% less than [3, 5].

The design does not incur much power overhead compared to SoTA-works [5, 30], maintaining similar battery life for edge devices. The proposed accelerator achieves an inference time of 184 mili-seconds (42.3 GOPS), compared to 772 mili-seconds (4.95 GOPS) for DNN-Layer reused architecture [3] on *Xilinx Virtex-7* FPGA Board and 226 mili-seconds (34.38 GOPS) on *Jetson Nano* at 8-bit precision with an accuracy of 95%. Thus providing an efficient trade-off between available hardware resources and throughput latency. The detailed modeling for computation time (in

#clock cycles) for DNN execution using SoTA architecture (L_P) [5] and proposed architecture (L_R) can be expressed with:

$$Latency\_parallel(L\_P) = \sum_{k=1}^{L-1} m(k) + L - 1 \tag{3.3}$$

$$Latency\_proposed(L\_R) = \sum_{k=1}^{L} m(k) + 2L - 3, \tag{3.4}$$

where L refers to the total layers of the DNN model, and $m(k)$ is the count of PQRE utilised in the k^{th} layer. This is crucial for understanding the application inference time at edge nodes.

Overall, the LPRE consumes similar power while offering a trade-off in bandwidth and resource utilization, highlighting the efficiency of accuracy-critical IoT devices. Our design, adapted for vehicle number plate detection (facilitating automated gate control), showcases performance improvement compared to FP and Int/FxP precision-based edge-AI accelerators. This justifies our solution as better suited for resource-constrained edge-AI applications than SoTA works.

Chapter 4

CORDIC is All You Need

Enhancing the efficiency of hardware accelerators involves several techniques to increase throughput while minimizing power usage and latency. These techniques include Quantization, Pruning, Sparsity, Power and Clock Gating. Quantization entails lowering the precision of data from a higher to a lower bit-width, which helps reduce the size of adders and multipliers in both weights and inputs, ultimately decreasing delay and power consumption. This process employs methods such as Truncation and Round-to-earest-even, although such methods can compromise precision, resulting in diminished accuracy on a larger scale. Research on eliminating quantizing errors is available in sources like [44, 45]. Pruning entails minimizing computational data according to its significance, a method utilized by NVIDIA in their GPUs to enhance throughput [46–48]. Adjusting lower-valued inputs or weights to zero based on the pruning ratio effectively reduces computational requirements. Pruning coordinates with Sparsity, which converts data matrices into sparse formats by removing zeros. Although pruning can affect accuracy, it can be recovered through model fine-tuning, possibly doubling throughput gains. Some of the Latest design also Performs a Hierarchical pruning-sparsity increasing the ratio up to 3:4 reducing the 75% computation [49]. Sparsity, while complicating data flow, minimizes redundant calculations. At a more fundamental level, Power and Clock Gating are used to conserve energy by disabling power to unused logic blocks; in power gating, a transistor is placed on the power nets to effectively act as an enable pin. Clock gating reduces switching power loss by controlling the clock's frequency to certain devices. These strategies are integral to the Physical Design process during ASIC Chip Fabrication. Though in these optimization techniques, some are applicable at PE whereas some are applicable and more optimal in architecture design.

The proliferation of edge devices—from smart sensors, including intelligent cameras and soil quality sensors, to autonomous vehicles and data mining—necessitates efficient AI accelerators capable of handling complex computations. These Edge-AI Accelerators are vital for enabling real-time decision-making while minimizing latency, curbing bandwidth demands, and upholding data privacy. Contrary to cloud-based AI, edge computing reduces dependency on external servers, thereby becoming

crucial for applications with stringent latency or privacy requirements. Advances in AI and computation accelerators have been significant, particularly in systolic arrays, CORDIC processors, and neural network hardware accelerators. Data flow optimization techniques, such as data reuse and row-wise weight or output stationary flows, enhance throughput and lower computational demands. Enhanced Control Engine (**CE**) designs, along with design-level improvements in PE, have contributed significantly. SOTA hardware accelerators and high-performance computing (**HPC**) emphasize performance, albeit with substantial area and power costs, thus favouring cloud environments.

Conversely, edge devices operate under tight constraints, necessitating architectures that efficiently balance these metrics while maintaining adequate accuracy and minimizing area and power usage. In deep neural networks, each neuron executes a Multiply-and-Accumulate (MAC) operation followed by a nonlinear transformation. Different AFs serve as nonlinear transformations within the network. In a DNN, the output precision of a MAC unit determines the input precision for the AF. Typically, this precision is expressed as $2N + K$, where N is the bit-width of the MAC input, and K represents additional overhead bits for accumulation, as illustrated in 4.1(c). Enhancements in performance have been achieved by truncating MAC outputs, which allow the use of lower precision AFs, despite a compromise in accuracy. While these configurations deliver high throughput, they also incur significant area overheads in high-precision architecture implementations. Moreover, these architectures are limited by fixed design constraints, making them unsuitable for accommodating configurable AF implementations and unsupportive of direct scaling for variable bit-precision scenarios. Implementing high-precision processing elements necessitates considerable hardware resources, leading to an exponential growth in resource utilization. Various techniques have been developed, such as storing AF values and parameters in look-up tables (LUTs) [36], using piecewise linear and nonlinear transformations [50], employing the CORDIC algorithm, and approximating the AF. Each design approach offers distinct advantages. The piecewise-linear (PWL) method integrates several linear vectors to implement non-linear functions efficiently.

Utilizing a LUT for the PWL representation of an AF is optimal for minimizing hardware utilization when the number of neurons is less than the available BRAM blocks in an FPGA circuit.

Systolic array architecture is extensively applied for matrix multiplication and CNN operations, foundational in neural networks and transformers. Legacy architectures like Google's TPU utilize systolic arrays for high-throughput and energy-efficient computations. Designs such as Eyeriss [51] incorporate systolic array concepts to optimize energy efficiency through local memory reuse. This exploitation of data's temporal and spatial locality decreases unnecessary memory accesses. Additionally, it facilitates the transfer of partial sums from one PE to the next, further economizing memory use and computational cycles. This method is known as data-flow optimization, with weight-stationary or output-stationary data flows enhancing throughput and energy efficiency. Hardware architectural design strategies like layer reuse improve resource allocation by recycling hardware across different neural network layers, as discussed in [52] with Layer-reuse strategies for area-optimized DNN hardware. Reusing PE resources across multiple layers allows sequential execution, minimizing hardware underutilization. Advanced architectures also employ time multiplexing. In time multiplexing unused PEs in one computation can be repurposed for others, maximizing utilization. The data multiplexing approach also reduces memory buffer needs by loading data directly into PEs using a demultiplexer.

CORDIC blocks excel at executing trigonometric, hyperbolic, and logarithmic functions, which highlights their importance in accelerators. Past research, including [36,37,53–57], has demonstrated CORDIC's efficiency in reducing power and area during activation computations and other tasks. Nonetheless, many conventional designs cater to a single function or are not reconfigurable, presenting difficulties for varied workloads. Integrating CORDIC can effectively enhance AFs, allowing for calculating commonly used AFs within the same architecture across diverse demands. Although iterative CORDIC procedures can lead to latency issues, limiting high-throughput designs that can be mitigated via pipelining approaches. Neural network accelerators are targeting enhanced throughput, utilizing techniques like sparsity

and quantization to enhance Tensor Core performance. Despite these advancements, adapting to resource-constrained conditions and supporting emerging workloads like transformers [58], DNNs, or recurrent neural networkss (RNNs) remains challenging. Sparsity in DNNs greatly reduces computational workload, and efficient hardware support for sparsity involves using compressed data formats to lower computation and optimize memory access patterns to reduce power usage. The complexity in using sparsity arises from its control mechanism and address mapping [35, 59].

Current accelerators primarily emphasize enhancing specific computational paradigms, such as CNN or ANN, but generally lack a versatile accelerator capable of efficiently executing CNN, ANN, RNN, and Transformer models. This pursuit of a runtime-configurable accelerator, especially beneficial for devices like smartphones, AR/VR headsets, or diverse demanding applications, introduces considerable complexity. This inflexibility often results in suboptimal utilization of hardware resources. Our study aims to augment design flexibility. Upon reviewing various architectures, we have identified several techniques adeptly applied in novel designs. The study into Deep Neural Network (DNN) accelerators has yielded numerous hardware architectures designed to enhance performance, power efficiency, and area utilization. Notable accelerators, such as FEATHER [60], Efficient Inference Engine (EIE) [59], Sparse Convolutional Neural Network (SCNN) [61], and Eyeriss [51], have achieved substantial improvements by exploiting sparsity, weight pruning, and efficient memory access patterns, among other sophisticated techniques. FEATHER reduces computational demands by implementing low-bit quantization and sparsity-aware computing, thus achieving high throughput with minimal area and power, alongside slight accuracy degradation due to quantization [60]. EIE capitalizes on weight pruning during dense matrix-vector operations, thereby significantly lowering on-chip memory usage and energy consumption while boosting computational throughput [59]. SCNN enhances convolution efficiency by adopting a sparse dataflow paradigm that bypasses zero-valued computations, resulting in notable decreases in unnecessary data fetches and data calls, thereby improving memory transfer power consumption and reducing computational latency [61]. RECON (Resource-Efficient

CORDIC-Based Neuron Architecture) focuses on achieving hardware efficiency and functional flexibility using the CORDIC algorithm for proficient computation of both linear and non-linear AFs, making it suitable for diverse neural workloads, whether sparse or dense [53]. QuantMAC introduces a quantization-enabled Multiply Accumulate (MAC) unit to enhance energy efficiency by dynamically adjusting precision levels according to workload demands, employing Quire arithmetic as an extension [21]. Raut et al. introduce ReLU-Centric Design Papers, such as [52], which highlight the importance of hardware reuse to support multiple AFs (AFs), enhancing efficiency while minimizing area overhead [7]. Additionally, a variety of hardware optimization strategies, such as Booth's algorithm, Wallace tree adder, Vedic multiplier, and error-resilient logarithmic multiplier, are employed to enhance performance in high-performance computing. In near-sensor computing, the prevailing trend is towards approximate hardware circuits, with a slight accuracy trade-off deemed acceptable [53, 62–64].

AFs design are optimized for accelerators within the library's and articles [65, 66]. Gate-level designs have been developed for multiple AFs, including Hyperbolic, Tangent, SoftMax, Gaussian, Sigmoid, ReLU, GeLU, and Binary-Step. Similarly, studies concerning AFs directed us towards employing CORDIC. Previously, the LUT method was utilized to store AF matrices, while the PWL approach applied linear equations like $y = mx + c$, storing multiple values of m and c to map functions to specific non-linear AFs. Making AFs reliant on memory results in devices that consume high power [36, 67]. To circumvent such power-consuming hardware computations, we adopted the CORDIC methodology. This paper presents RPE in the form of **CORDIC(n,m)**, signifying n MAC stages and $m/2$ each for hyperbolic and linear stages of the AF. Our evaluation confirms that five pipeline stages can deliver high throughput without sacrificing accuracy with the help of Pareto analysis. Additionally, the architecture is crafted for fixed-point calculations, featuring a design that minimizes area and power usage, and it is adaptable at both architectural and clock levels.

Table 4.1 State-of-the-art AI Models and their corresponding MAC Units and parameters with the necessity for Hardware Optimization

Model	**Dataset**	**Parameters**	**MAC count** (**Billions**)	**Sparsity** (%)	**Physical Overhead** (**Area-Power-Delay**)
VGG-16	ImageNet	140 M	15.5	80	Medium
ResNet-50	ImageNet	28 M	3.7	65	High
Inception-v3	ImageNet	24.6 M	5.3	60	High
MobileNet-v1	ImageNet	4.2 M	0.6	35	High
BERT-base	SST-2	105 M	21.2	55	Medium
MobileBERT	SST-2	26 M	4.3	65	High
GPT-2	SST-2	118 M	42	35	Medium
GPT-3	Internet-scaled data	175 B	6.7	75	High
GPT-4	Internet-scaled data	1760 B	>1000	50	High

4.0.1 Why is the Need

The primary aim of this study is to devise an area-efficient architecture tailored to AI requirements by leveraging hardware resource reuse to handle a range of operations such as CNN, ANN, RNN/LSTM, and Transformers. However, significant throughput degradation occurs when DNN hardware processes Transformers, necessitating advanced mapping techniques [68]. Researchers have explored various optimization strategies for Processing Engine(**PE**) design across different abstraction levels to enhance resource efficiency and throughput, though this is often achieved at the expense of accuracy [57].

Therefore, designing circuits involves balancing area, power, latency, throughput, and accuracy. This study promotes the implementation of the Iterative and recursive CORDIC algorithm, facilitating MAC computation with minimal hardware resources and lower power consumption. The accuracy of recursive CORDIC-based MAC and AF is determined by the number of iterations; increased iterations enhance accuracy and primarily influence throughput [69]. The author has incorporated the adaptable

CORDIC architecture [38] to enable multifunctional capabilities and maximize the use of CORDIC. The proposed performance enhancement block ultimately provides increased flexibility and control over hardware design, benefiting from a Systolic array [70], which provides high throughput. In DNN and Transformers, performance metrics are limited by the MAC operation, a primary computational aspect that consumes significant resources. Typically, advancing architectural efficiency requires numerous MACs — VGG-16 involves 15.5 billion MACs, AlexNet utilizes 724 million MACs, ResNet-50 necessitates 3.9 billion MACs, while Transformers like GPT-3 and GPT-4 demand 175 billion and up to 1.76 trillion parameters, respectively, as noted in Table 4.1. Collectively, these points highlight the need for a reconfigurable design capable of supporting Transformers, DNNs, and RNNs/LSTMs, providing control at the level of individual Reconfigurable Processing Engines (RPE). A modifiable RPE block enhances hardware flexibility for diverse kernel sizes and AFs, enabling the accelerator to handle most model computations and reducing reliance on the CPU.

The noteworthy contributions of this work are outlined below:

- **Reconfigurable Processing Engine (RPE):**
  This work introduces a resource-efficient RPE that employs a Pareto-optimal CORDIC (5+2) framework characterized by minimized area overhead, facilitating key AI operations such as MAC, Sigmoid, tanh, SoftMax, GeLU, ReLU, etc. It dynamically shifts between iterative latency and pipelined throughput, optimizing hardware resource usage in response to AI task demands.

- **Systolic CORDIC engine for Reconfigurability and Enhanced Throughput (SYCore):**
  The introduced SYCore presents a systolic design incorporating the proposed RPE, delivering high throughput and scalable runtime-configurable dense and sparse matrix multiplications. It functions as a fundamental component for AI tasks, including DNNs, Transformers, and RNNs/LSTMs.

- **Configurable and Adaptive Execution Scheduler for Advanced Resource Allocation (CAESAR):**

The newly developed CAESAR control unit adopts an adaptive tiling and scheduling approach that dynamically leverages quantization and pruning co-design benefits. This ensures efficient resource utilization and adaptable workload handling in edge-AI accelerators. Several benchmark analyses are provided for detailed result assessment.

- **Empirical Analysis for Performance Enhancement:**
  We analyzed the minimal pipelined CORDIC stage requirement within RPE to augment SYCore's performance. The custom bitwise Pareto analysis of the CORDIC stages underscores their influence on error metrics, accuracy, and hardware characteristics. The outcomes reveal improved hardware efficiency with resource optimization and minimal latency impact, supported by extensive comparisons with similar FPGA and ASIC accelerator benchmarks.

4.1 Proposed Neuron Architecture (RPE)

For AI systems that are extensively used, the design must prioritize both energy efficiency and a certain level of flexibility. This would allow handling various tasks using a uniform core, eliminating the need for distinct accelerators for different functions. We have designated the proposed core as the Systolic CORDIC engine, named for its Reconfigurability and Enhanced throughput (SYCore). A review of existing literature reveals a crucial gap in hardware design: a lack of flexibility. Many existing architectures do not effectively offer PE control to users, often only limiting to features like power or clock gating and enabling signals. Nevertheless, through a sophisticated design, better PE control can be achieved via CORDIC blocks. These blocks are equipped to handle computations required by a plethora of AI processes today, including but not limited to DNNs, RNNs, and Transformers, hence they are suitable for a wide range of AI tasks. The challenge in creating a CORDIC-based Reconfigurable Processing Element (RPE) lies in managing the balance between latency and resource consumption, especially during repetitive calculations for MAC operations and AFs. Linear CORDIC stages underpin MAC

computation and necessitate multiple iterations for precise results. Although more iterations enhance precision, they concurrently increase latency. Mitigating latency either through fewer iterations or parallel processing escalates resource consumption due to the need for extra hardware, like shifters, adders, and LUTs.

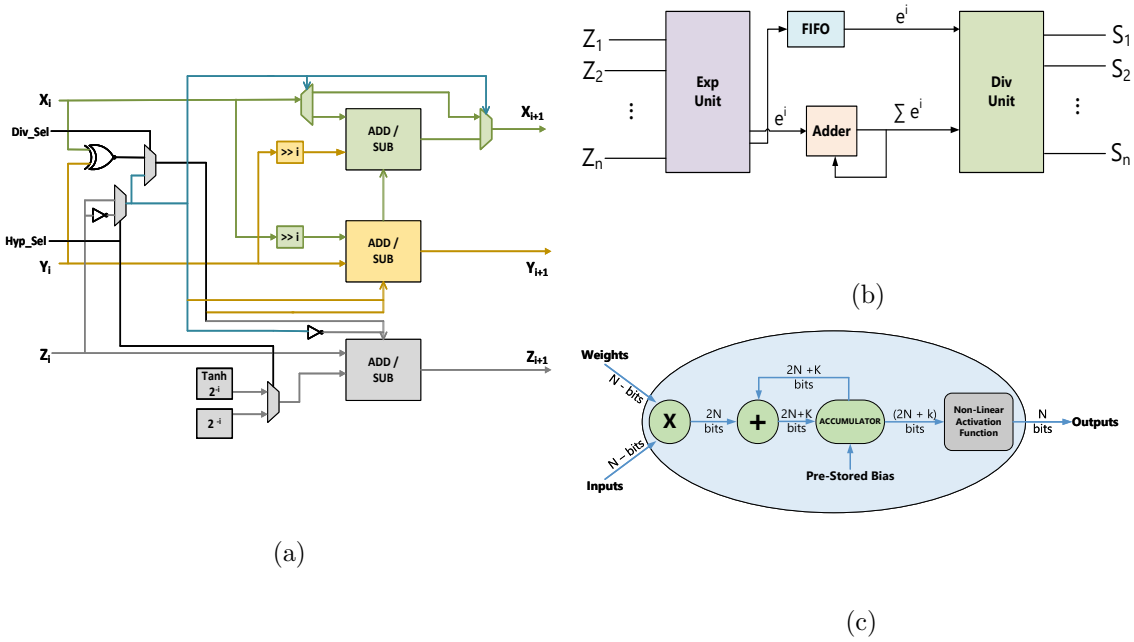


Figure 4.1 a) CORDIC Stage-1 fundamental element b) Reconfigurable Activation function using RPE c) Conventional Processing Engine.

To effectively compute hyperbolic parameters for various components, such as AFs (AF) like the sigmoid and tanh, the hyperbolic CORDIC stage is typically preferred. This stage necessitates an additional iterative element in its design. Activation functions often rely on pre-stored specific constants and angles, resulting in greater computational demands compared to linear stages. While dedicated hardware for this stage can enhance performance, it also increases resource usage. In contrast, utilizing a shared-stage design can minimize area but may introduce bottlenecks and increase latency. The division stage for AFs presents similar challenges, with iterative division methods being slower by nature. Employing existing CORDIC modules for division conserves resources but diminishes processing speed, whereas specialized division hardware boosts performance at the cost of increased area use. Pipelined hardware can address latency by facilitating parallel execution over iterative processing, though

Table 4.2 Detailed CORDIC equations: general, linear and hyperbolic

General CORDIC	Linear CORDIC	Hyperbolic CORDIC
$x_{i+1} = x_i - m\delta_i y_i 2^{-i}$	$x_{i+1} = x_i - \delta_i y_i 2^{-i}$	$x_{i+1} = x_i + \delta_i y_i 2^{-i}$
$y_{i+1} = y_i + \delta_i x_i 2^{-i}$	$y_{i+1} = y_i + \delta_i x_i 2^{-i}$	$y_{i+1} = y_i + \delta_i x_i 2^{-i}$
$z_{i+1} = z_i - \delta_i E_i$	$z_{i+1} = z_i - \delta_i E_i$	$z_{i+1} = z_i - \delta_i E_i$

this requires additional registers per iteration. Iterative designs save on hardware by employing the same resources for multiple tasks, which is advantageous for resource-constrained edge devices [71], yet they increase latency, complicating real-time application performance. On the other hand, parallel execution uses duplicate CORDIC stages for simultaneous processing, significantly cutting down latency but greatly increasing area and power demands. Precision requirements also remain a pivotal consideration.

Achieving a balance among these trade-offs depends on the specific application. Systems that require low latency, like those in robotics and healthcare, benefit from parallel task execution and reduced iteration counts to meet real-time demands. On the other hand, resource-constrained edge devices focus on iterative approaches that utilize hardware reuse, approximate computations, and scheduling that considers sparsity to conserve resources. A hybrid strategy that combines iterative, parallel, and pipelined methods can provide a balanced architecture, offering flexibility for various workloads. By dynamically optimizing iteration counts and adopting adaptive CORDIC designs, it is possible to adjust this balance to accommodate specific or mixed precision requirements within the device's performance limitations.

4.1.1 CORDIC algorithm

Coordinate Rotational Digital Computer (CORDIC) is an algorithm that mimics rotation for iterative calculations of various linear and nonlinear functions. It is engineered to enhance hardware efficiency, factoring in resource constraints and throughput needs. The quantity of CORDIC stages required depends on the fractional

bit precision in the adaptive fixed-point format, which encompasses both fractional and integer bit widths. The detailed CORDIC algorithm [34,38] is common knowledge and thus, avoided the explanation here. However, we have added appendix A. A comprehensive explanation is provided in the following subsections using Pareto analysis to identify the optimal number of stages, ensuring robustness against errors despite approximations. The AF part of the proposed RPE is illustrated in Figure 4.1(b).

$$e^{-\mathrm{AFin}} = \cosh(\mathrm{AFin}) - \sinh(\mathrm{AFin}) \tag{4.1a}$$

$$\tanh(\mathrm{AFin}) = \frac{\sinh(\mathrm{AFin})}{\cosh(\mathrm{AFin})} \tag{4.1b}$$

$$\mathrm{sigmoid}(\mathrm{AFin}) = \frac{1.0}{1.0 + e^{-\mathrm{AFin}}} \tag{4.1c}$$

4.1.1.1 CORDIC-Based Reconfigurable Processing Element (RPE)

The CORDIC-based RPE serves as the core of the processing engine, enabling efficient iterative calculations for both matrix operations and AFs. Each RPE is equipped with multiple pipelined linear CORDIC stages, which are followed by stages for hyperbolic and division computations. This setup allows for iterative MAC operations along with non-linear AFs such as sigmoid, tanh, and SoftMax. This modular structure supports runtime reconfigurability, making it adaptable to various workloads with differing computational requirements.

4.1.1.2 Modes

CORDIC, a coordinate rotation algorithm, adapts its operational mode according to the input data type. Initially, CORDIC was predominantly used for computing non-linear functions, such as tanh and sigmoid, which are integral to many deep neural networks (DNNs). The CORDIC algorithm encompasses three primary modes: hyperbolic, linear, and circular. These modes enable a broad spectrum of operations, from simple multiplication and division to intricate calculations like logarithmic

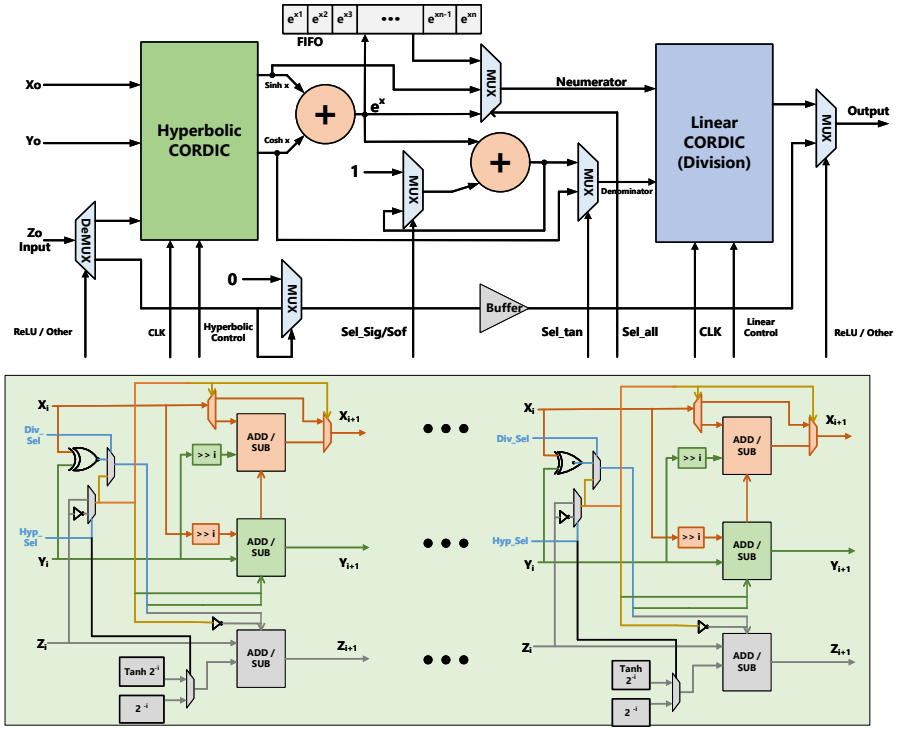


Figure 4.2 Reconfigurable Activation function using RPE

and hyperbolic operations, such as sinh and cosh, performed within the rotational CORDIC structure. Additionally, CORDIC offers a Vector Mode for computing the inverse functions of these operations. The foundational CORDIC equations represented in Table 4.2 assign the mode values of 0, -1, and 1, which correspond to circular, hyperbolic, and linear modes, respectively. These mode indicators are stored in m to identify the mode during the transition process. The inputs and outputs adjust dynamically with the selection of multiplication, division, or hyperbolic operations. In the Linear Mode, used for multiplication and division, 2^{-i} is calculated alongside variable inputs for different MAC operations. The X signal is designated for inputs, the Z signal for weights, and the Y signal can be utilized for additions, as described in (4.2) below:

$$MAC(input = X_0, weight = Z_0, bias = Y_0) = Y_0 + X_0 * Z_0 \tag{4.2}$$

For Hyperbolic Mode, Table 4.2 illustrates the process where the input, provided through the Z input, results in the computation of sinh and cosh at X and Y

outputs. Depending on the sign of Z, the variable d_i determines whether additions or subtractions are performed on the values of X, Y, and Z.

$$\text{SoftMax}(a) = \frac{e^a}{\sum_{i=0}^{n} e^i} \quad a \in (0, n) \tag{4.3}$$

4.1.1.3 Pareto Analysis

CORDIC is a pseudo-rotational framework that induces computational errors since it estimates values through an iterative process, with more stages resulting in greater accuracy. However, increasing iteration count also raises hardware latency, thereby decreasing throughput. Historically, a large number of iterations slowed computation, especially for AFs. The advent of Transformers has amplified the computation demands for AFs. To determine the optimal number of iterations across all utilized functions, we have simulated CORDIC error across various precisions: 8, 16, and 32 bits, for different iteration counts, as illustrated in the Pareto plots: Figure 4.4, Figure 4.3, and Figure 4.5. The Pareto analysis suggests that beyond a specific iteration count, error reduction becomes negligible, thus supporting the adoption of a limited number of iterative stages. The evaluation was conducted using Python within Jupyter Notebook, leveraging libraries such as fxpmath, numpy, and others[1]

The CORDIC algorithm, primarily assessed for iterative calculations in AFs and trigonometric operations, was evaluated for potential approximations balancing accuracy, resource usage, and energy efficiency. Consequently, Pareto analysis reveals that optimal configurations are contingent upon specific workload needs. This approach leads to a minimal-error design, mitigating accuracy degradation in CORDIC computation by cutting unnecessary iterations and integrating Pareto analysis with bit precision considerations. This study highlights the proposed CORDIC design's flexibility, allowing for application-specific adjustments to optimize performance, power, and accuracy constraints. Figure 4.3, Figure 4.4, Figure 4.5

[1]The code is available in the GitHub repository `https://github.com/OmkarRajeshKokane/CORDIC-Is-All-You-Need`.

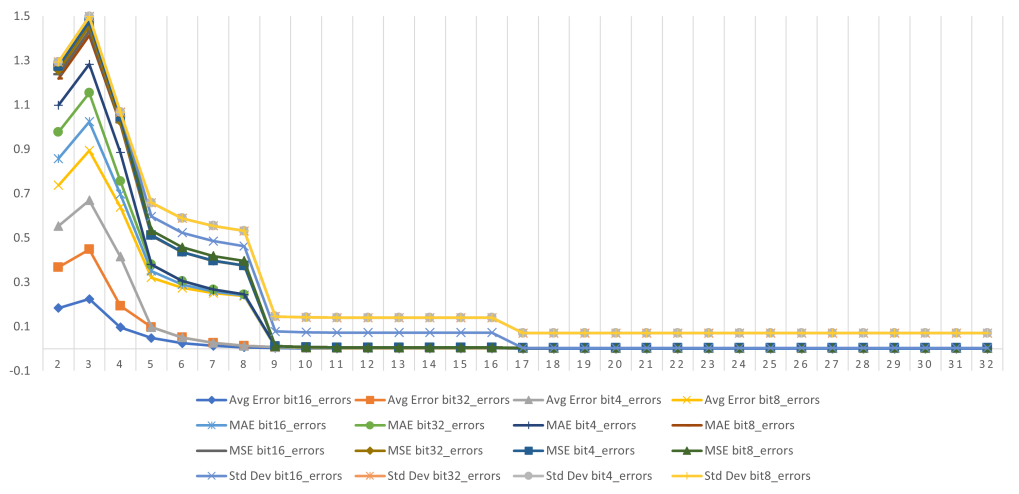


Figure 4.3 Pareto Analysis of sigmoid for various error plot.

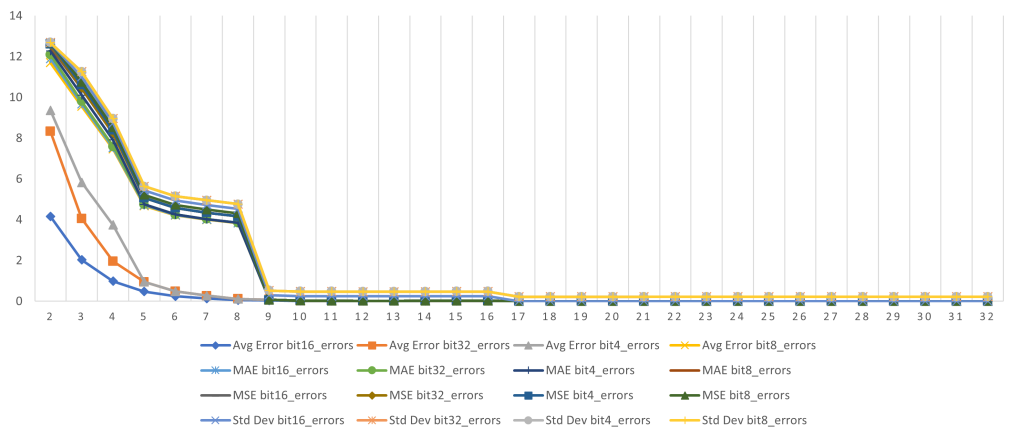


Figure 4.4 Pareto Analysis of tanh for various error plot.

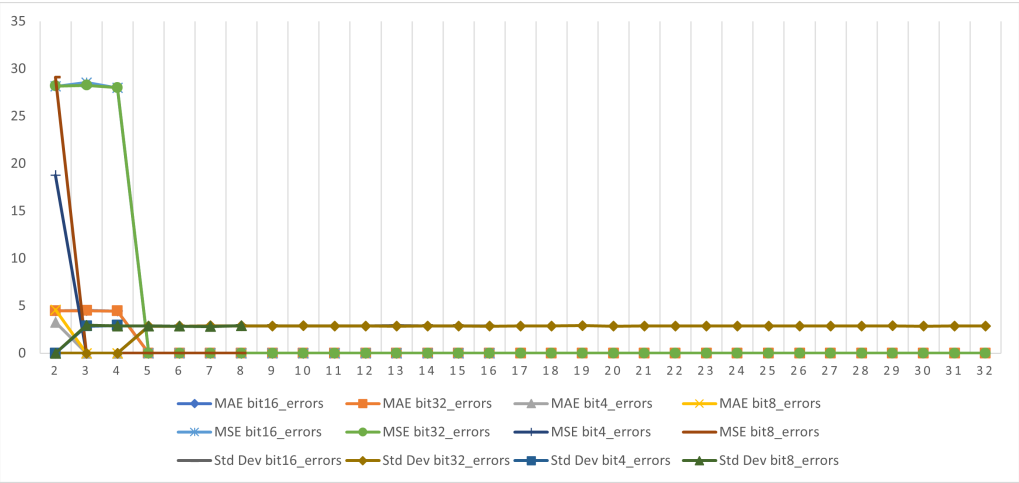


Figure 4.5 Pareto Analysis of softmax for various error plot.

illustrates the errors across varying bit precisions (4, 8, 16, 32 bits) over iterations, presenting metrics such as average error, mean absolute error (MAE), mean square error (MSE), and standard deviation (STD). The formulas for these errors are provided in the following equations:

$$\text{MSE} = \frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n} \quad (4.4) \qquad \text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \quad (4.5)$$

$$\text{Avg. Error} = \frac{\sum_{i=1}^{n} \frac{|y_i - x_i|}{|x_i|}}{n} \quad (4.6) \qquad \text{STD} = \frac{\sum_{i=1}^{n} [x_i - \text{mean}(y)]^2}{n-1} \quad (4.7)$$

These equations represent the formula utilized for determining Pareto errors, with x denoting the anticipated output and y indicating the resultant output utilizing Fixed-point CORDIC design. The error is characterized as the average discrepancy in the values. Larger values correspond to greater error magnitudes, and the same error percentage translates into larger absolute errors for higher values, as compared to those shown in Pareto plots.

4.1.2 Proposed 5+2 RPE Architecture

The Multiply-Accumulate (MAC) unit is the most frequently utilized component in AI accelerators. To improve the MAC's performance with CORDIC, we have engineered a CORDIC design that executes the MAC operation within the systolic array architecture, as delineated previously in Section 4.1.1.2. The substantial delay within CORDIC arises from its inherent iterative design. To address this, a pipelined architecture has been introduced to the MAC unit, enhancing throughput. The MAC operates in a five-stage pipeline, thereby decreasing the circuit latency, with five clock cycles required to produce an output. Initially, the multiplication process occupies five clock cycles, with each cycle computing a multiplication. This allows multiplication operations to occur at frequencies of 3 GHz or higher, enabling three billion multiplications per second. Primary operations in convolutional layers, fully connected networks, or transformers include multiplication in 2D/1D convolution and matrix multiplication formats. Additionally, the AF is implemented using a CORDIC iterative technique, prioritizing area efficiency and computational necessity.

The versatile design enables CORDIC to execute various AFs such as tanh, Sigmoid, ReLU, and the commonly used SoftMax. Implementation demands the division phase of CORDIC, along with two adders and several Register Array memory units operating in a FIFO configuration to hold SoftMax values. Multiple multiplexers and demultiplexers are utilized to select the desired AF.

4.1.2.1 CORDIC: Pipelined vs Iterative analysis

The CORDIC (COordinate Rotation DIgital Computer) algorithm is extensively used for the calculation of trigonometric, hyperbolic, and logarithmic functions. Its hardware can be realized through either pipelined or iterative methods, each presenting unique trade-offs. A pipelined approach offers advantages such as reduced delay in throughput, needing five clock cycles initially to produce the first output, and then providing output multiplied for each subsequent clock cycle. The minimized delay in the combinational path enables higher clock frequency operation, thereby enhancing the entire design's frequency capabilities. However, pipelining requires additional flip-flops between stages, which increases power consumption and physical design area. This method can be adapted for different levels of precision with minimal design alteration, primarily modifying data paths rather than control blocks. On the other hand, iterative CORDIC designs prioritize area and power efficiency, leveraging pure combinational logic at the expense of higher delay and lower throughput and are less amenable to frequency scaling. Although additional pipelining can enhance pipeline designs, a hybrid approach facilitates the optimal balance of delay, power, and area. In iterative CORDIC, storing angle rotation constants for hyperbolic and division stages in memory is necessary, unlike in pipelined designs where exact stage values are pre-determined. Iterative designs also undergo challenges related to control complexity. The proposed design uses iterative stages for hyperbolic and division functions since these operations are less frequent than multiply-accumulate (MAC) operations. Thus, a five-stage pipeline is implemented for MAC calculations, alongside an iterative design for hyperbolic and division stages.

4.1.2.2 5+2 Stage Architecture

Pipelining the MAC into five stages alongside two iterative stages for hyperbolic and division calculations enhances the throughput in our CORDIC design, without significant compromise on accuracy, as confirmed by Pareto-driven analysis. Each pipelined stage features its own 2^{-i} values that correspond to their respective iterations, facilitating logical optimization during netlist generation. Further, benefits of the CORDIC pipeline are discussed in Section 4.1.2.1. Moreover, complex Deep Neural Networks (DNNs) necessitate complex AFs, thus the employment of iterative hyperbolic and division stages enhances the functionality for AFs such as tanh, sigmoid, and SoftMax. There's also an alternative design that omits the hyperbolic and division stages, thereby lowering latency. However, the SoftMax function's integration remains essential for transformer-based architectures, enhancing SoftMax processing flow. Flex-PE and DA-VINCI [33,38] expands the range of supported AFs, including tanh, sigmoid, SoftMax, ReLU, SeLU, Swish, and GeLU, thereby enabling compatibility with diverse AI models like RNN/LSTM, DNN, and Transformers. The resulting hybrid architecture, while more intricate to manage, demands a sophisticated design and precise control mechanisms, which we will elaborate on in the next section.

4.1.3 Details of Data and Control Signals and and State Machine for Neuron Engine

To manage the design, understanding the data flow within the RPE is essential, starting from the data entry into the registers. Once the enable signal reaches the RPE, the initial stage of the MAC, synchronized with the clock, processes the data. It operates on the variables x, y, and z and relays the outcome to the subsequent register, where x receives the input values, y acquires the bias, and z obtains the weight values. Depending on the sign of z, the term δ_i determines whether to execute addition or subtraction. A shifter maintains the connections. The same procedure is repeated in the next cycle, with variation only in the angular constants E_i, which

have values of 2^{-i} based on the iteration count. This pipelined process continues until the fifth iteration, passing data to the subsequent pipelined block with each iteration. Every clock cycle triggers a new input for MAC operation, producing multiplied outputs at each cycle and capitalizing on the pipelined architecture. Upon completing multiplication and accumulation with all kernel elements, the hyp_select signal activates the hyperbolic stage flow, as depicted in Figure 4.1(b). Following this, the accumulated data proceeds to the Hyperbolic design for five clock cycles using the existing CORDIC hyperbolic stage, controlled by a counter to route the data to the next stage. The division phase of the CORDIC design commences with the Div_select signal and a similar counter to issue the output completion signal. Various signals are directed to the multiplexers and demultiplexers. To choose the AF, the $Relu/other$ signal decides whether to employ ReLU, while sel_sig/sof selects between sigmoid and SoftMax. The sel_tan signal determines whether the design operates with hyperbolic tangent, all affecting the denominator. Conversely, a three-input multiplexer sets the numerator, selecting among the three AFs: tanh, sigmoid, and SoftMax via sel_all. SoftMax computation diverges slightly: inputs stored in the FIFO memory are summed during storage. These stored inputs are then fed to the Division CORDIC block to yield the SoftMax values, concluding the RPE process with the RPE_done signal.

A suitable strategy for structuring this control flow is through the use of a finite state machine (FSM). To devise an FSM controller for the RPE, certain parameters are initially necessary, such as kernel size and the specific operation to be executed within AF, which are critical in the setup phase. Starting from the initial stage, the system proceeds to the initiation state, where it accepts data inputs and iterates over five clock cycles, continuously receiving inputs and weights each cycle while keeping the other AF signals inactive. Once the first multiplication outcome reaches the final pipeline stage, the FSM progresses to the following phase—namely, the Rigorous MAC operation stage—where inputs for multiplication are continually processed in subsequent iterations until all multiplications are finalized. Upon completing the multiplication phase, the FSM transitions to the AF, determined by the selected AF.

In the scenario where tanh or sigmoid is chosen, the FSM enters case 1, beginning with the Hyperbolic computation minor stage, followed by the Division stage in its div state. The sequence concludes with the RPdidne signal, returning to the ideal state. For case 2, if the user opts for the SoftMax state, all inputs undergo hyperbolic processing in the first stage, being stored in the SoftMax FIFO concurrently, after which the process evolves to the division state. Here, the FIFO-stored values are utilized to derive the exponent terms divided by the cumulative output generated during the hyperbolic stage, summing all input exponents. This division is executed iteratively for each FIFO input, taking n times 4 clock cycles, culminating in the completion of case 2 and dispatching the RPE complete signal. For the simplest case, 3, the FSM selects the ReLU flow, bypassing unnecessary iterations of 9 clock cycles in hyperbolic and division stages, thus conserving clock cycles and enhancing throughput. Completing from a singular AF state of ReLU, the FSM reverts to the IDLE state. When integrated into larger applications, such designs facilitate executing a broad array of tasks on the same hardware, encompassing LLMs, computer vision tasks, and more. However, to meet throughput demands, a lone RPE block is insufficient. The architectural framework is crucial in optimizing RPE's advantages, ensuring the design's applicability and maximizing efficiency in addressing these requirements.

4.1.4 Optimizations for Next-Generation Workloads: DA-VINCI

The sophisticated AF architecture of the DA-VINCI core enables seamless incorporation of AFs with minimal computational overhead. These functions can be dynamically configured at runtime to switch among Swish, SoftMax, SeLU, GeLU, Sigmoid, tanh, and ReLU, unified within a single design utilizing CORDIC. Prior studies [33] highlight the efficacy of these functions with a Quality of Results (QoR) reaching 98.5%. The unified CORDIC algorithm [36, 53] has been preferred for reconfigurable hardware for circular, linear, and hyperbolic operations combined

with rotational and vector modes. The fundamental CORDIC hardware uses simple design elements such as Add/Sub, MUX, LBS, and memory blocks. The HOAA-enabled CORDIC architecture [63] provides 21% area savings and up to 33% lower power consumption, effectively compensating for the area overhead introduced with DA-VINCI core's reconfigurability. Thus, the proposed DA-VINCI core almost matches the resource utilization for individual AF implementations. The detailed methodology for CORDIC-based activation function hardware has been explored for Swish [72], SoftMax [73], SELU [74], GELU [72], Sigmoid [75], Tanh [76], and ReLU. DA-VINCI programmable core is built on linear and hyperbolic CORDIC block, with sel_af signal, program particular AF datapath and sequence of operations. sinh and Cosh are computed with HR mode in Swish, SoftMax, SELU, GELU, Sigmoid and Tanh, marking 86% reuse factor, while division operation with LV mode in Swish, SoftMax, GELU, Sigmoid, and Tanh marking 72% reuse factor. Additional buffer in case of ReLU, FIFO in softmax and a couple of multiplication units for GELU, and fitting additional SELU and Swish contribute to extra overhead, which identifies as hardware vs reconfigurability tradeoff.

DA-VINCI surpasses current designs in resource efficiency, achieving notable reductions: up to 4.5 X in LUT usage, 3.2 X in flip-flop (FF) usage, 7.8 X in critical path delay, and 14.3 X in power consumption. Both FPGA and ASIC analyses demonstrate these advantages, with ASIC showing up to 16.2 X area reduction, 7.8 X delay reduction, and 14.3 X power reduction across different technology nodes and bit precisions, as previously discussed in our works. Integrating the DA-VINCI Design with the RPE block enhances its capability to handle diverse workloads, making it feasible to implement every non-pooling layer using a consistent design due to its support for AFs like SoftMax and GeLU in Transformers and Swish and SeLU in RNNs/LSTMs, without excluding ReLU, Sigmoid, and tanh. This broadens the design's acceleration potential across these functions. The design incorporates additional multipliers, which can also be built using CORDIC, along with an HOAA adder to address overestimated addition [63], and several multiplexers for AF selection. Initial multipliers in the AF handle scaling of the matrix multiplication (MAC)

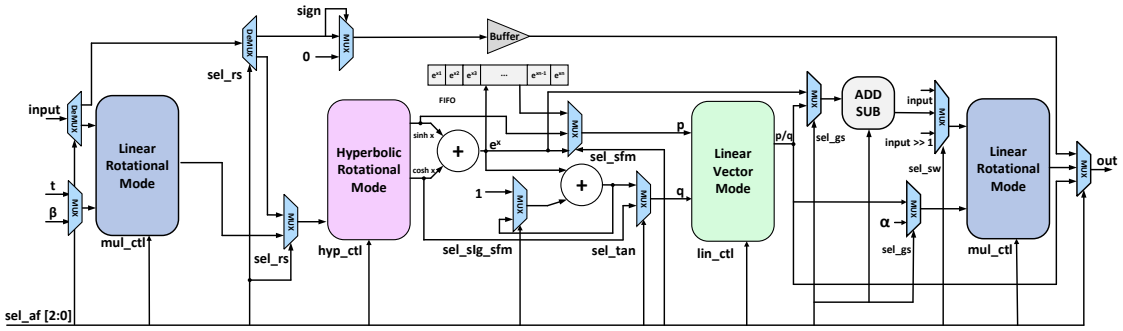


Figure 4.6 Dynamically-configurable activation function supporting GeLU, SeLU, Swish, ReLU, Tanh, Sigmoid and Softmax

outputs, which involves scaling non-linear output values. Such operations can extend across layer transitions, as observed in transformer models that contain normalization blocks. The hyperbolic stage supplies exponential and hyperbolic outputs via adders, while division tasks rely on the division capabilities of the CORDIC stage. This allows most AI-driven operations to be efficiently executed through a fundamental CORDIC block, from conventional MAC procedures to AFs like SeLU and Swish. Additionally, CORDIC is proficient in executing logarithmic and square-root functions among others, indicating that CORDIC is the sole fundamental block necessary for all workloads, validating the adage CORDIC IS ALL YOU NEED.

4.2 Parameterized and Modular Systolic Accelerator Architecture

The comprehensive analysis of a complete FPGA design flow contributes significantly to an in-depth understanding of the entire process. Starting from the initial software code, the model is developed in TensorFlow/Torch and executed on the proposed RPE using a Systolic Array architecture. In Section 4.1.2, we explored the RPE design and its advantages. We now aim to examine the Systolic array's design and evaluate its benefits across different types of flows, comparing it with other accelerator architectural designs. This includes architectures like Layer-reuse, 1D

structures, NAS-based designs for task-specific applications, parallel architectures, and sparsity-based designs such as RAMAN [77]. Additional research in architecture has spawned designs such as Eyeriss [78], EIE [59], Feather [60], Aurora, and more. The primary motivations behind various architectures are to optimize the design for particular processing elements and to harness benefits such as sparsity, pruning, SIMD, throughput demands, and other specific constraints.

Analyzing various architectures individually, starting with the Layer reuse architecture reveals the concept of utilizing the same hardware across multiple layers. This approach is predominantly applied in 1-dimensional arrays of processing element designs where, sequentially, only one layer is processed at a time. The 1-D architecture can be adapted for layer multiplexing [34] or time multiplexing [62], each offering inter-layer parallelism but requiring greater hardware resources. Although this results in lower latency, which is suitable for low-latency applications, it introduces complex data and control flows. In contrast, time multiplexing offers intra-layer parallelism by reusing hardware over time. This method is simpler in flow complexity but incurs high latency due to sequential processing. Nonetheless, its hardware efficiency makes it suitable for devices with limited resources.

Turning our attention to NAS architectures, researchers have proposed multicore devices where each core is specifically optimized for accelerating distinct tasks, such as CNNs, fully connected layers, pooling, transformers, or normalization and scaling. This approach results in straightforward designs but suffers from inefficiencies in power and area. Conversely, parallel architectures utilizing 1-dimensional PE arrays facilitate concurrent DNN model execution, thereby enhancing throughput and reducing latency, which is well-suited for multi-tenant workloads similar to NAS. However, this configuration demands substantial resources and power, making it better suited for high-performance computing (HPC) rather than edge deployments. The Systolic array, characterized by its two-dimensional arrangement of processing elements, functions by keeping input fixed at one PE and transferring weights across the array. This is advantageous for DNN applications where the input data is frequently reused. In contrast, a weight stationary approach holds weights constant

while the input data traverses the array, useful in scenarios with stable weights, thereby conserving energy on data transfers. Systems using such arrays are often referred to as vector systolic arrays. There are also other structures like output stationary and interleaved architectures. In output stationary designs, partial sums remain stationary while the inputs and weights move in an interleaved manner. Each design aligns with specific workloads, whether it be CNNs, fully connected layers, or RNNs. The limitations of traditional designs and the incorporation complexities of advanced optimization have spurred the development of solutions like Eyeriss, which continue to evolve, seen in subsequent models like EIE and Feather. These integrate various benefits into a more complex yet efficient system, enhancing throughput. Nevertheless, no existing design exhibits sufficient flexibility to perform multiple operations and run diverse AI workloads on a single architecture. We propose a new design that can manage all operation types and execute different AI workloads using the same framework. To grasp the architecture's intricacies, a comprehensive understanding of the entire system is required.

4.2.1 Host Interfacing

Analyzing the entire workflow from an application perspective enhances the comprehension of the data and control flow within any design. Starting with the sensor component, analogous to the human eye, the machine relies on the camera as its sensor. Various camera attributes are essential, such as its resolution, accuracy, and RGB configuration. Additionally, certain prerequisites are necessary, including the drivers for retrieving data from the camera and the software libraries to import the image into the codebase. The image data is subsequently processed within the flow. Camera FPS is also important. The model employed utilizes the standard quantized TensorFlow-ReLU framework [79]. This compilation of information is directed to the RISC-V compiler [80, 81], which translates the data flow and computations into suitable instructions for the required operations. With CASEAR and SYCore, a mix of custom and standard instructions is integrated. RISC-V is a processor based on a reduced instruction set computer architecture, featuring a 5-stage pipeline of

fetch, decode, execute, memory access, and write-back. These stages are part of the execution process on RISC-V, including custom instructions. To comprehend the custom instructions on RISC-V, it is necessary to understand the Instruction Set Architecture (ISA). RISC-V accommodates 32 and 64-bit instruction widths. Each instruction belongs to a specific instruction type. RISC-V includes various predefined instruction types and allows for customized instructions. The IFMA category includes the Base Integer Instruction Set (I), Integer Multiplication and Division Extension (M), Atomic Instructions Extension (A), Single-Precision Floating-Point Extension (F), and Double-Precision Floating-Point Extension (D), among other predefined instructions. Instruction Formats such as R, I, S, B, U, and J facilitate custom operations, with the possibility to introduce new formats using reserved opcode bits. Open-source tools like RISC-V GCC and LLVM compilers offer support for crafting and optimizing custom data instructions.

Once the Python code is translated into a sequence of instructions, the device determines whether the instruction should proceed to RISC-V pipelining or be directed towards CAESAR. When implementing pooling, the instruction targets the RISC-V path; however, for CNN operations and similar, it is directed to CAESAR, SYCore's control engine. Within CAESAR, the instruction functions as a co-processor, interpreting and leveraging the command to establish data handling procedures, integrating the existing ISA into a register. Subsequently, the necessary DNN parameters are secured for model implementation by setting flags. The data fetcher begins retrieving data from the shared L2 cache, while the address mapper assigns the retrieved data to appropriate locations. Concurrently, the scheduler determines the RPE requirements, optimizing efficiency in executing the complete routine within the Data-flow FSM. Upon completion, the output is transferred back to the L2 cache for storage, and a completion signal is issued. This process leads to a reduced workload on the RISC-V, allowing it to perform additional tasks. The workflow is depicted in Figure 4.7.

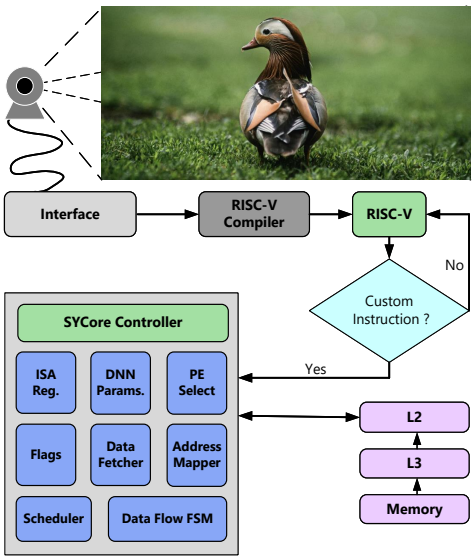


Figure 4.7 Host Interfacing

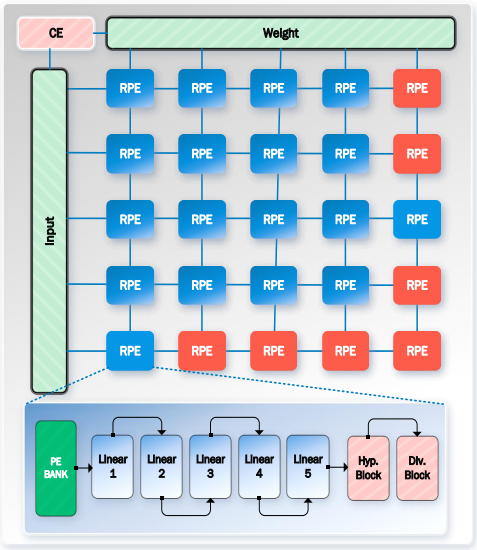


Figure 4.8 Systolic Array

4.2.2 Design and System Architecture Overview

An in-depth analysis has been conducted on the comprehensive proposed design, which incorporates SYCore, a stationary systolic array computing output controlled by CAESAR, a sophisticated engine for managing data flow. Beginning with the SYCore is an accelerator developed on a foundational architecture characterized by an output stationary systolic array. This architecture allows the inputs and weights to move continuously for computation processes while maintaining the partial sums in a fixed position. SYCore is comprised of individual RPE blocks configured in a 32 X 32 array and is subdivided into smaller 4 X 4 sub-blocks to facilitate parallel computation and optimize accelerator efficiency. When any sub-block is not in use, all sub-blocks are deactivated to conserve power. Data flow within sub-blocks is enhanced through multiplexers, enabling data transfer to neighbouring sub-blocks. In the output stationary approach, the inputs and weights are transmitted between sub-blocks, minimizing unnecessary data transfer and consequently augmenting the design's energy efficiency.

Within each sub-block, RPE units are organized in a 4 X 4 configuration. Every sub-block contains its unique input and weight buffers, facilitating computations

Table 4.3 Detailed mapping and scheduling for VGG-16/CIFAR-100 with Proposed RISC-V Enabled SYCore Architecture

VGG-16 CIFAR-100	Layer Specifications (KxK)x CinxCout x(HxW)	SYCore (MxN)	K-MAC Ops	Op. cycles	CAESAR-based Utilization (%)	Execution Time (us)	Power Consumed (mW)
C1_1	(3x3)x 3x64 x(32x32)	32x32	1728	1728	100	2.65	0.63
C1_2	(3x3)x 64x64 x(32x32)	32x32	36864	36864	100	56.7	13.55
P1	Max-Pool, (2x2), S=2	-	-	411	-	189	
C2_1	(3x3)x 64x128 x(16x16)	16x16	73728	18432	100	28.33	6.77
C2_2	(3x3)x 128x128 x(16x16)	16x16	147456	36864	100	56.7	13.55
P2	Max-Pool, (2x2), S=2	-	-	313	-	144	
C3_1	(3x3)x 128x256 x(8x8)	8x8	294912	18432	100	28.2	6.76
C3_2	(3x3)x 256x256 x(8x8)	8x8	589824	36864	100	56.9	13.5
C3_3	(3x3)x 256x256 x(8x8)	8x8	589824	36864	100	56.7	13.55
P3	Max-Pool, (2x2), S=2	-	-	515	-	237	
C4_1	(3x3)x 256x512 x(4x4)	4x4	1179648	18432	100	28.4	6.78
C4_2	(3x3)x 512x512 x(4x4)	4x4	2359296	36864	100	56.5	13.45
C4_3	(3x3)x 512x512 x(4x4)	4x4	2359296	36864	100	56.66	13.55
P4	Max-Pool, (2x2), S=2	-	-	328	-	151	
C5_1	(3x3)x 512x512 x(2x2)	2x2	2359296	36864	33.33	56.4	13.86
C5_2	(3x3)x 512x512 x(2x2)	2x2	2359296	36864	33.33	56.66	13.66
C5_3	(3x3)x 512x512 x(2x2)	2x2	2359296	36864	33.33	56.9	13.54
P5	Max-Pool, (2x2), S=2	-	-	165	-	76	
Flatten	-	-	-	1	-	55.88	
FC6	512x4096	1024	4096	2048	100	3.1	0.75
FC7	4096x4096	1024	4096	16384	100	25	6.024
FC8	4096x100	256	100	4096	40	6.3	1.506
Total					**83.75+**	**148.6 ms**	**1.524 W**

within the same sub-block while transmitting data to neighbouring sub-blocks. Executing these operations concurrently minimizes the design's output latency. By centralizing buffer procedures within sub-blocks, data management is streamlined. This method maintains the scalability of the design by adjusting the number of sub-blocks employed. A design akin to matrix multiplication is effective for diverse workloads, such as Transformers, RNN/LSTM, and traditional DNN applications. Each RPE takes 45 clock cycles to produce its first output. To govern this process, we introduce CAESAR, a Configurable and Adaptive Execution Scheduler for Advanced Resource Allocation, which efficiently manages tasks from ISA instruction reading through to execution and processing completion.

As previously mentioned in Section 4.2.1, once the CAESAR receives the custom ISA, it initially decodes the instructions required for executing a DNN. The DNN

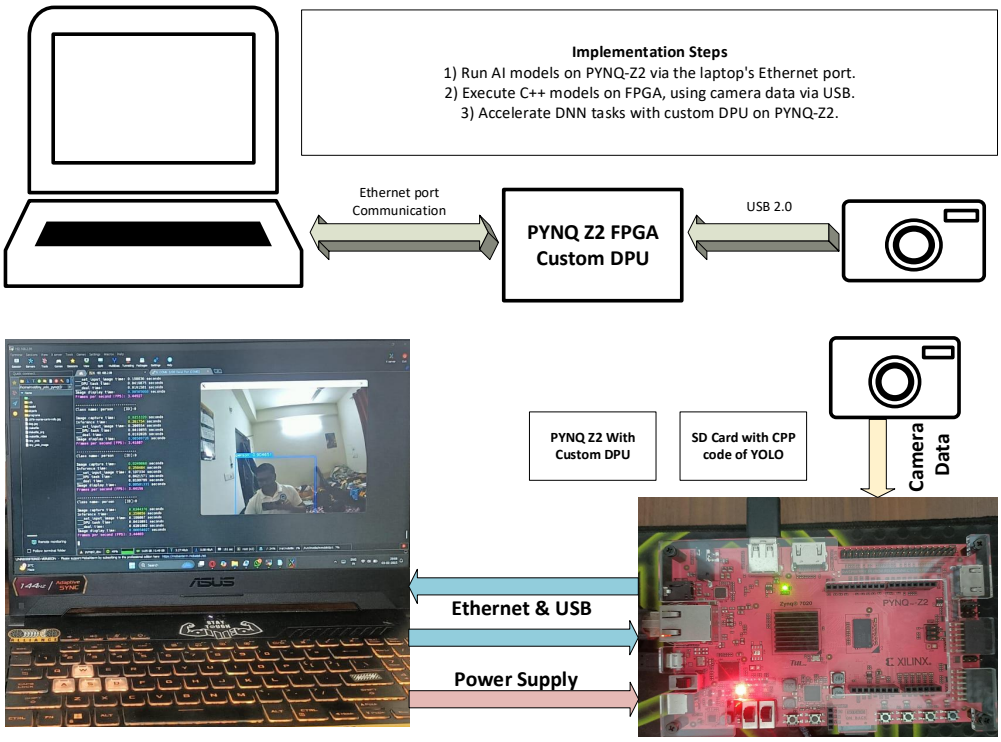


Figure 4.9 Pynq Z2 Custom DPU Prototype Implementation

parameters are established with the activation of the DNN_start signal. Multiple flag signals, such as those for task allocation and scheduler status indicating idle and busy states, along with an error handling flag, ensure the entire system is informed of any errors at any stage, allowing CAESAR to take appropriate actions. A pipeline coordination flag manages data flow, preventing data stalling while monitoring dynamic parameters. Debugging and optimization monitors utilize these flags to determine fine-tuning requirements or to identify design bottlenecks. Additionally, two primary status flags are included: the Idle FLAG, which indicates whether sub-blocks or the entire SYCore are in an idle state, and the busy status flag, which informs if a sub-block is actively processing data or if data transfer is underway.

The ISA register is tasked with storing the active instructions. Conversely, the Scheduler block is charged with selecting sub-block addresses and managing the dataflow FSM, effectively utilizing quantization, pruning, and sparsity, among other methods. The design's size is chosen by the scheduler for parallel execution,

and efficient mapping for large workloads poses a significant challenge. The final component of CAESAR is the Data Flow FSM, responsible for directing data processing, computing each layer, and signalling with *layer_done*. Once the entire DNN completes, the *DNN_Done* signal is emitted, and all outputs are stored in the L2 shared cache. A data management block oversees all data transfers between the SYCore and the L2 cache, a task included in the Address Mapper, ensuring data localization and storage.

4.2.3 Dataflow and Functioning of the array

To comprehend the data and control flow of the CAESAR controller in conjunction with the SYCore, one must gain a clearer grasp of the data flow, particularly as outlined in Section 4.3, where the data mapping is described according to computational demands. As previously mentioned, instructions for data parameters initialize the DNN sizes within the DNN parameter registers, upon which the scheduler organizes the computations executed by the SYCore. The process begins with mapping the initial Convolutional layer. In column 2, Layer Specifications (k X k) denote the kernel dimension by $C_{\text{in}} \times C_{\text{out}} \times (H \times W)$, correlating to the input image size. In the first convolution, we observe a kernel size of 3 X 3 and an input size of 32 X 32, with three input channels (C_{in}) due to RGB, and 64 output channels (C_{out}). Given the input size of 32 X 32, these can be efficiently allocated across the SYCore, which is composed of 32 X 32 RPEs, resulting in a total of 1728 MAC operations. The Op Cycles incorporate 1728+4 clock cycles, accounting for each MAC computation, plus additional cycles initially required by the RPE. This design optimizes performance, achieving a complete execution time of 2.65 µs, with a power consumption of 0.63 mW. We now proceed to analyze this flow with greater depth.

Following the storage of the DNN parameter, the scheduler evaluates the initial design to implement it and observes that the input data size completely occupies the Systolic array by utilizing all the Sub-blocks. When the SYCore reaches full capacity, memory access to the L2 cache is facilitated through the AXI interconnect. In this scenario, since no sparsity or pruning is applied that would modify data handling, an

address mapper generates a sparse data format by assigning the appropriate addresses to prevent any data issues, which is best understood with an example. Suppose pruning occurs at a 4:9 ratio, specifically weight pruning, meaning some weights are set to zero. This implies that no data is present where the smallest weight values are zeros, removing them from computation to avoid multiplying by zero, known as sparsity. In this design, such a technique is applied at the processor level, altering data patterns which must be recognized by the address mapper. Furthermore, this requires documentation in the DNN parameters to achieve a more efficient scheduling algorithm and optimize resource usage. By examining various convolution layers where each Sub-Block operates on a 4 X 4 architecture, input mapping at 4 X 4 achieves 100% utilization; however, efficiency declines as the input decreases in size. The hardware also incorporates tiling to effectively manage the solution, and a complex data reordering flow can control this issue. Additionally, the design relies on the processor to conduct the max-pooling operation.

Moreover, the integration of the RPE significantly enhances performance. Once the data is distributed into their corresponding sub-blocks, the data Fetcher retrieves all the mapped data from these specified sub-blocks. Subsequently, the DNN executions commence, with Flag variables monitoring the sub-blocks and activating the DNN_start and Layer_start signals to enable them. For each clock cycle, the subsequent inputs and weights are provided to the data, facilitating the execution of all multiplication operations. The output begins arriving after 45 clock cycles, once the initial sub-block RPEs are stored in the SYCore's L1 cache, which is necessary for the next layer's input. Upon complete data reception, the Layer_done signal is activated, transitioning the RPE state to IDLE. This process repeats until it reaches the initial pooling layer, where output data is forwarded to the L2 cache via the RISC-V processor for pooling operations. All tasks proceed similarly. In convolutional operations, the Data Flow FSM unit iterates the RPEs across sub-blocks. When the DNN processes the Flatten layer, data is transformed into a one-dimensional format using SYCore, with adjustments to address and data mapping, after which the FC layers compute the input size multiplied by 1024 SYCore computations, thus

again achieving full utilization of SYCore in the final FC as the requirement is 256 for SYCore consumption. A 40% utilization of CAESAR is observed. Consequently, the total execution time resulted in a delay of 148.6 ms per execution of an image. Thus, SYCore with a CEASER control engine can achieve up to a maximum of 29.68 inference frames per joule.

4.3 Evaluation

In the assessment of hardware parameters for optimizing DNN performance, various design strategies such as fully parallel, sparse, layer-reuse, and our proposed bit-width quantization approach have been applied. The fully parallel DNN configuration boasts significant throughput; however, it incurs considerable area overhead, thus elevating power consumption. To enhance design optimization, even at the MAC level, we have developed a refined PE design, comparing MAC results of our proposed design, as documented in Table 4.5. This table outlines foundational results from different MAC configurations, all computed using CMOS 28nm HPC+ technology and a frequency of 100 MHz. It contrasts metrics like Area, Power, Delay, ADP, Energy, and Priority, focusing on MAC/PE levels, subsequently analyzed for architectural efficiency in terms of compute density (TOPS/mm^2) and energy efficiency (TOPS/W), assessed for a 32 X 32 Systolic array. We observe an energy efficiency enhancement of up to 96.7% at a frequency of 100 MHz. The proposed design's pipelining allows operations at frequencies as high as 3 GHz, demonstrating a 92.15% improvement in compute density over [82]. Given that MAC hardware is crucial for power savings in AI applications, the proposed design is significant, achieving 31.28 TOPS/W in energy efficiency. At FPGA levels, MAC achieves notable results in terms of 17 LUTs and 26 registers for resource utilization, offering a delay of 1.46 µs at total power consumption of 0.54 µW. The power-delay product serves as a robust comparative metric due to substantial efficiency improvements introduced by the pipelined CORDIC, which enables a more efficient MAC operation at high frequencies.

Table 4.4 FPGA MAC comparison

MAC	**Xilinx IP [83]**	**Vedic MAC [37]**	**CORDIC-MAC [34]**	**Wallace MAC [84]**	**PipeMAC [7]**	**PNE MAC [85]**	**Booth MAC [86]**	**Proposed**
LUT	53	159	35	105	23	46	83	17
Reg	28	245	58	112	22	20	61	26
Delay (us)								
Logic	0.813	4.48	0.713	2.59	1.86	0.38	3.08	1.43
Signal	2.277		0.692			0.87		
Power (uW)								
Logic	0.27	1.31	0.16	1.21	0.22	0.18	0.9	0.09
Signal	0.21		0.2			0.24		0.16
Dynamic	3		4			2		0.34
PDP	9.57	5.86	5.62	3.13	0.4092	2.52	2.77	0.86

Table 4.5 ASIC MAC comparison on 28 nm

Parameter	**Area (µm²)**	**Total Power (µW)**	**Delay (ns)**	**Energy (pJ)**	**Area-Delay-Power (ADP)**	**TOPS/mm²**	**Relative (%)**	**TOPS/W**	**Relative (%)**
Liu et al. [87]	838	502	6.44	3.23	2.71E+06	1.32	8.32	2.38	7.61
Ashar et al. [21]	501	122	4.27	0.52	2.61E+05	3.34	20.99	14.77	47.21
Raut et al. [34]	307	144	3.62	0.52	1.60E+05	6.43	40.41	6.92	22.13
Ratko et al. [88]	272	195.24	2.52	0.49	1.34E+05	10.42	65.52	11.22	35.88
Ansari et al. [89]	368	260.38	2.59	0.67	2.48E+05	7.49	47.12	8.07	25.80
Yin et al. [90]	244	227.4	2.46	0.56	1.36E+05	11.90	74.82	12.82	40.98
Waris et al. [82]	771	153	7.42	1.14	8.75E+05	1.25	7.85	1.34	4.30
Warris et al. [91]	606	552.06	2.47	1.36	8.26E+05	4.77	30.00	5.14	16.43
Ratko et al. [88]	440	200.34	2.88	0.58	2.54E+05	5.64	35.44	6.07	19.41
Kim et al. [92]	297	211.64	2.71	0.57	1.70E+05	8.87	55.80	9.56	30.56
Leone et al. [93]	373	283.44	2.93	0.83	3.10E+05	6.54	41.09	7.04	22.50
Liu et al. [94]	294	224.45	2.36	0.53	1.56E+05	10.29	64.73	11.09	35.45
Reza et al. [95]	447	199.5	2.54	0.51	2.27E+05	6.29	39.56	6.78	21.66
Liu et al. [84]	815	291.66	2.93	0.85	6.96E+05	2.99	18.81	9.00	28.78
Proposed	200.5	109.8	2.24	0.25	4.93E+04	15.90	100.00	31.28	100.00

Table 4.6 ASIC MAC comparison on 7 nm

Parameter	**Area (µm²)**	**Total Power (µW)**	**Delay (ns)**	**Energy (pJ)**	**Area-Delay-Power (ADP)**	**TOPS/mm²**	**Relative (%)**	**TOPS/W**	**Relative (%)**
Liu et al. [87]	273	188	5.14	0.97	2.64E+05	5.09	106.2	7.96	150.6
Ashar et al. [21]	161	45.8	3.38	0.15	2.49E+04	13.13	273.8	49.69	940.4
Raut et al. [34]	99	54	2.86	0.15	1.53E+04	25.23	526.2	27.17	514.1
Ratko et al. [88]	255	65	2.78	0.18	4.61E+04	10.08	210.2	10.85	205.4
Ansari et al. [89]	354	81	2.04	0.17	5.85E+04	9.89	206.3	10.65	201.6
Yin et al. [90]	234	70.4	1.94	0.14	3.20E+04	15.73	328.2	16.94	320.7
Waris et al. [82]	248	57.3	5.87	0.34	8.34E+04	4.91	102.3	5.28	100.0
Warris et al. [91]	472	96.4	1.96	0.19	8.92E+04	7.72	161.1	8.31	157.4
Ratko et al. [88]	324	68	2.28	0.16	5.02E+04	9.67	201.7	10.41	197.1
Kim et al. [92]	299	87	2.13	0.19	5.54E+04	11.22	234.0	12.08	228.6
Leone et al. [93]	267	76	3.19	0.24	6.47E+04	8.39	174.9	9.03	170.9
Liu et al. [94]	246	95	1.86	0.18	4.35E+04	15.61	325.6	16.81	318.2
Reza et al. [95]	437	113	2.09	0.24	1.03E+05	7.82	163.1	8.42	159.4
Liu et al. [84]	386	102	3.86	0.39	1.52E+05	4.79	100.0	19.54	369.7
Proposed	64.6	41	1.77	0.07	4.69E+03	62.47	1303.1	106.00	2006.0

4.3.1 Experimental Setup and Validation

For Validation and Setup, we executed tests on the design. Initially, we focused on testing the CORDIC by employing Python on Google COLAB, which allowed us to determine the CORDIC requirements for implementing the necessary number of stages for operations like MAC, tanh, Sigmoid, and SoftMax, as illustrated in

Figure 4.3, Figure 4.4, and Figure 4.5. The error analysis is detailed in Section 4.1.1.3, based on various error types discussed. It was found that conducting five iterations is a reasonable approach for CORDIC computations. We then proceeded to the design phase of the RPE. We created RTL code for both single and multi-staged reconfigurable processing engines, primarily utilizing CORDIC. Using this approach, we gathered ASIC results for the design using Cadence Genus in a TSMC 28 nm technology node at a standard process corner. As shown in the table, we scripted the Tcl for the design with a clock period of 10 ns. Further scaling of the technology node can yield improved results. Prior to this, we also conducted RPE Emulation using Vivado 2024.1 software. Correct output values were achieved after adjusting select signals and proper clock mapping; following optimized synthesis results, we proceeded to the implementation and integration of the systolic array, where the design of the control engine and computational metrics such as Excel TOPS/mm^2 and TOPS/W were calculated.

4.3.2 Software-Based Evaluation and Validation of Inference Accuracy

The models' accuracy is depicted in Figure 4.10, indicating minimal variation across the different models. Notably, there is a minor decrease in accuracy attributable to the CORDIC approximation errors. Our analysis involved AI models, including custom models, VGG-16 on CIFAR-100, CaffeNet, and ResNet, with LeNet-5 evaluated on MNIST and CIFAR-10. The MNIST dataset is relatively small and consists of handwritten digits, while CIFAR-10 is a larger dataset used for image classification and is included in certain complex datasets. ImageNet, meanwhile, is an extensive dataset used for object classification and prediction in video feeds. We considered a diverse range of models, with LeNet-5 exemplifying smaller architectures, while larger models tend to attain slightly reduced accuracy that remains comparable to SOTA achievements. This evaluation also encompassed Transformers at different bit precisions. When juxtaposed with other tensor-based

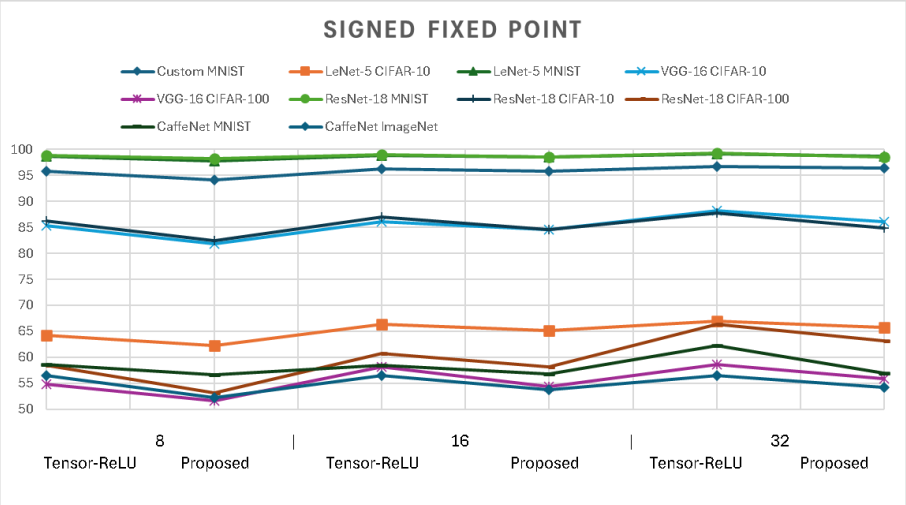


Figure 4.10 Accuracy at Several Deep Neural Nets.

models [79], the proposed solution shows less than a 2% decrease in accuracy. Accuracy results were obtained through TensorFlow using the Qkeras library to adjust precision, as illustrated in Figure 4.10, which assessed data widths of 8, 16, and 32 bits. This underscores accuracy improvements achievable through various bit precisions. Employing integer/fixed-point data formats suggests favouring lower bit widths due to the negligible accuracy gains relative to their cost. Consequently, training loops were executed for 50-80 epochs to attain optimal accuracy. Moreover, it was found that pruning up to 40% did not incur any per-layer loss, though further pruning is feasible for certain layers. Our architecture supports hardware execution using CAESER.

4.3.3 Pareto Analysis of RPE Design Parameters: Pipeline Stages, Bit-Precision, and Integer Bits, etc. other design parameters (MAC level comparison and Pareto)

The development and validation of a CORDIC PE with adjustable CORDIC MAC and AF units is conducted using a Python-based emulator to evaluate both accuracy and hardware performance. Pareto analysis is utilized to identify the ideal number of CORDIC stages that optimally balance accuracy, area, power consumption, and

delay. In the context of MAC design, this achieves enhanced performance throughput per watt, albeit with a compromise in accuracy, as evidenced by results on the MNIST dataset. Specifically, for an optimized iteration count, accuracy loss is noted in the MAC with five stages, with a normalized mean error of 6.31×10^{-5}. This corresponds to a normalized mean error distance of 0.00783, a mean relative error distance of 0.02675, and a normalized maximum error distance of 0.03131, indicating minimal error in MAC operations. Based on these metrics, it is concluded that the proposed MAC design offers approximately 23.34 X improvement in throughput per watt and up to 12.72 X improvement in computational density compared to the SOTA benchmarks, calculated by dividing throughput by total power and area, respectively. This performance advantage is evident despite processing 8-bit data in FxP format, as the pipelined MAC operation produces one multiplication per clock cycle, allowing the design to reach a frequency up to 3 GHz.

The findings summarized in Table 4.7 concern a system functioning at 100 MHz, utilizing a Systolic Array made up of 32 X 32 RPEs, thus comprising up to 1024 RPEs in total. The RPE flexibility enables various AI operations on the Accelerator, including RNNs, which handle sequential input processing within this framework. When comparing our MAC architecture with existing models and assuming our system realizes 100% efficiency, it becomes apparent that other accelerators, such as those by Yin et al. [90], achieve up to 74.82% efficiency, which remains approximately 25% less efficient than our design. Furthermore, Liu et al. [87] achieved 64.73% and Ratko et al. [88] reported 65.52%, making them the nearest competitors in efficiency. The MAC not only refines computational efficiency but also boosts AF performance, as displayed in the Pareto analysis for conventional DNN AFs like sigmoid, tanh, and ReLU in DNNs and RNNs, seen in Fig. 4.4, and 4.5. Additionally, the design accommodates Transformers by supporting SoftMax, albeit requiring added clock cycles due to expansive probability calculations.

Beyond classification, the hardware configuration, optimized for five iterations, minimizes error with a precision reduction to 8-bit. Increased precision leads to heightened error, while ReLU remains error-free irrespective of iteration count,

Table 4.7 Hardware Implementation Report with Proposed Systolic Array Architecture and State-of-the-Arts DNN Designs

	Platform	Model	Precision	LUTs (Thousands)	Registers (Thousands)	DSPs	Op. Freq (MHz)	Energy efficiency (GOPS/W)	Power (Watts)
Cong et. al [96]	Pynq-Z1	Custom	N/A	44	40	200	100	N/A	2.3
Yifan et. al [97]	ZU3EG	DiracDeltaNet	1A4W	24	30	37	250	8.5	5.5
Bi et. al [98]	ZU3EG	ResNet-50	8	41	45	250	150	45	1.5
Xiaodi et. al [90]	ZCU102	MobileNetV2	16	195	94	880	190	N/A	13.4
Liqiang et. al [99]	ZCU102	VGG16	16	130	69	365	200	12	23.5
Alessandro et. al [100]	Zynq7	VGG16	16	230	110	130	60	27	1.1
Adithya et. al [77]	Ti60	MobileNetV1	8	38	8.75	60	75	95	0.24
Xiaoru et al. [101]	Arria10	MobileNetV2	8	100	N/A	510	175	19	4.6
Gopal et. al [34]	ZC706	Custom	8	115	115	32	100	4.5	2
Bradley et.al [102]	VC707	Custom	N/A	240	200	110	170	N/A	2.2
Proposed	VC707	VGG16	8	91	126	1350	466	298	1.53

executing within a single clock cycle, similar to other AFs. This architecture supports operation at a frequency of 3 GHz, needing nine clock cycles—five for hyperbolic functions and four for division calculations. Enhanced sparsity and pruning further improve throughput, reducing latency by 1.7 X , MAC utilization by 2 X , and computational parameters by 1.8 X through commercial 4:9 pruning methods. The ASIC evaluations also cover aspect, power, and delay, with concurrent FPGA analysis in Table 4.7, respectively. The process incurs a 148 ms delay and a total power of 1.524 W at a 100 MHz operation frequency.

4.3.4 Hardware Implementation and Comparative Performance Analysis (Architectural Level of analysis)

Building on the proposition of the SYCore architecture, we proceeded to conduct a detailed analysis of its implementation on FPGA. Specifically, we assessed the VC707 Virtex-7 board, which operates at 8-bit precision and a built-in clock frequency of 500 MHz. The architecture demonstrated an energy efficiency of 298 GOPS/W. Regarding resource utilization, the architecture exhibited demands of 91k LUT, 126k registers, and 1350 DSPs. This marks as improved of approximately 1.5 ×, 2 × and 2.5 2 × over] [99], [101], and [100, 102] respectively. Furthermore, utilizing the Systolic array in conjunction with the RPE design yielded a throughput of up to 455 GOPS at 853 MHz frequency within FPGAs. Comparatively, Bradley et

al. [102] achieved a resource allocation of 240k LUTs, 200k registers, and 110 DSPs on similar hardware, which is substantially resource-intensive for any FPGA. Our design achieves the highest energy efficiency of 298 GOPS/W over SoTA works. The proposed architecture enables operation at higher frequencies for enhanced throughput. Additionally, power efficiency is critical, particularly for edge devices with limited power availability. In this context, the proposed architecture supports the implementation of complex classification models like VGG-16 on the hardware.

4.3.5 Evaluating Hardware Parameters for Optimal DNN Performance

Post-APR CMOS 28 nm HPC+ and 45 nm with Synopsys Design Compiler results for DA-VINCI design with the SOTA comparison is reported in Table 4.9, marking reduced area up to 16.2× and 4× while optimizing the critical delay to 7.8× and 1.3× compared to the SoTA works [103] and [53] respectively, while 2.1× and 14.3× power reduction than [75] and [53]. It also shows improvement up to 11.5× area, 1.75× critical delay, and 17.3× power consumption compared to ReCON [53], while reducing 2.5× power and 1.8× area compared to [75] and [104] with CMOS 28 nm.

When assessing the complete DNN for application-level purposes, throughput and latency are critical metrics. In image detection tasks, detection latency directly influences the achievable frames per second for real-time implementation. To illustrate

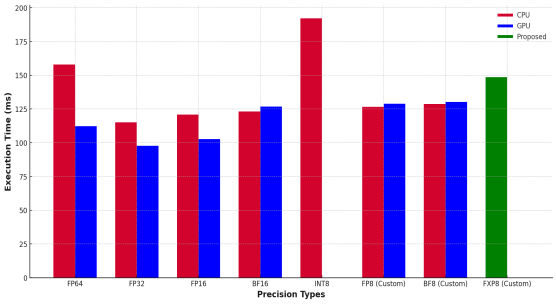


Figure 4.11 Execution time for different precision at CPU and GPU platforms compared to proposed model.

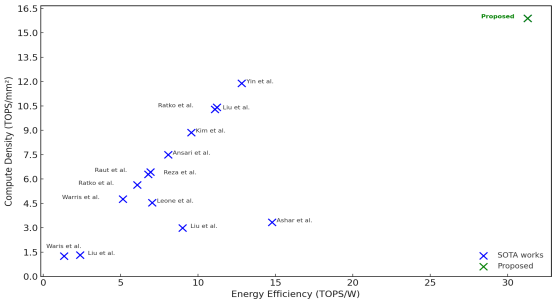


Figure 4.12 Performance analysis at CMOS 28 nm and comparison with SOTA works.

Table 4.8 FPGA Resources Utilization, compared with prior works

Design	ReAFM [72]	Shen et al. [76]	AFB [75]	AxSF [104]	DA-VINCI
AF	sigmoid-tanh-swish	tanh-sigmoid	tanh-sigmoid	softmax	configurable-AF
Precision	12-bit	-	16-bit	16-bit	8/16-bit
Mean error (%)	2.2	3.4	2.8	-	2.78
FPGA	Virtex-7	Zynq-7	PYNQ-Z1	Zynq-7	Zybo
LUTs	367	2395	36286	1215	137
FFs	298	1503	24042	1012	68
Delay (μs)	350	180	210	332	256
Power (mW)	-	0.681	125	165	27.53

Table 4.9 ASIC Resources Utilization, compared with prior works

Design	Zhang et al. [103]	AxSF [104]	AFB [75]	RECON [53]	DA-VINCI	
AF Support	softmax	softmax	tanh-sigmoid	tanh-sigmoid	configurable-AF	
Precision (bit)	32	16	16	16	8/16	
Tech. Node (nm)	28	28	45	45	45	28
Area (1000 * μm^2)	98.78	3.82	870.5	24.61	1.24	0.78
Delay (ns)	26	1.6	-	4.76	6.7	3.8
Power (mW)	24.72	1.58	151	1033	22.3	17.5

this, we have measured the latency of VGG-16 across different platforms: a CPU, a GPU, and our Proposed design. The CPU used is an AMD Ryzen 7 6800H with Radeon Graphics, operating at a base frequency of 3201 MHz with 16 logical processors manufactured in 7 nm technology. For the GPU comparison, we tested an NVIDIA GeForce RTX 3050 Ti with 4.0 GB memory, consuming 60 W and also fabricated in 7 nm, featuring 80 AI/Tensor cores. Our proposed design, using a single-core configuration at a frequency of 150 MHz, achieves similar performance despite being produced at a 28 nm node. When the SYCore operates multiple instances in parallel with a lower technology node at even higher frequencies, it surpasses conventional CPU and GPU hardware, as demonstrated in Figure 4.11. Our design is evaluated using fixed-point 8-bit (FxP 8-bit) computing, whereas the CPU and GPU are evaluated using FP64, FP32, FP16, FP8, BF16, BF8, and Int8 with the VGG-16 network, with average execution time shown on the y-axis. High latencies, in some cases, stem from a required quantization circuit in the workflow, which converts data

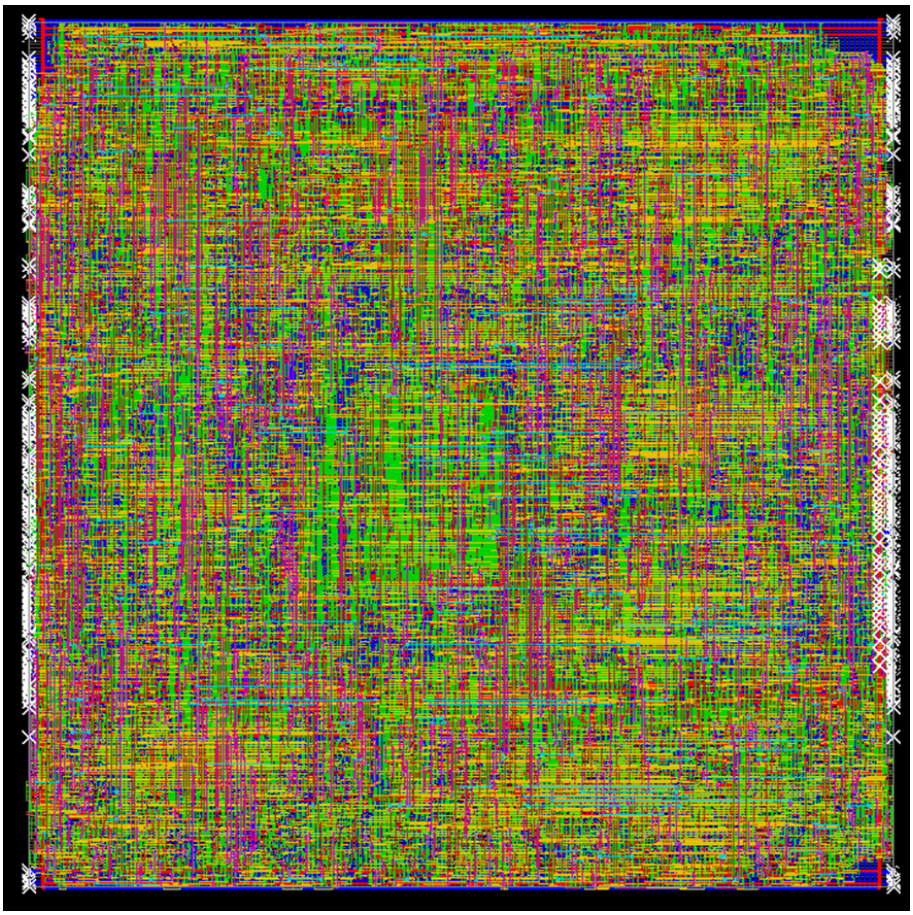


Figure 4.13 Graphic Data System II(GDSII) image of Systolic Array design using Cadence Innovus.

to 8-bit integers, introducing delays. Despite this, the proposed solution provides latency comparable to other hardware, with additional flexibility offered by the AF for overall computation. This enhances our hardware design relative to industry standards.

In conclusion, upon evaluating the throughput efficiency as depicted in Figure 4.12, the newly proposed design exhibits significant advancements in both energy efficiency and Compute Density relative to existing architectures. It achieves an Energy Efficiency exceeding 30 TOPS/W and a Compute Density surpassing 15 TOPS/mm^2, whereas comparable architectures offer a compute density not exceeding 13.5 TOPS/mm^2 and an Energy Efficiency below 15 TOPS/W. The detailed implementation approach is demonstrated in Fig. 4.9. It was noted during utilization

Table 4.10 Comparative cost performance analysis with SOTA solution

Parameter	Bit-Precision	FPGA & Cost ($)	Power (W)	MAC Units Utilisation (%)	Freq (MHz)
Proposed	FXP8	PYNQ-Z2, 140	**1.524**	**83.75+**	50
Kim et al. [105]	8-bit	A7-100T, 265	2.2	82.63	100
Jiang et al. [106]	8-bit	ZCU102, 3234	-	82.53	333
Trio et al. [107]	INT8	ZCU104, 1678	1.89	71.85	187
Thanh et al. [108]	8-bit	VC707, 5244	-	72.70	200
Raut et al. [34]	FXP8	VC707, 5244	1.12	54	466
Lee et al. [109]	Variable	ZCU102, 3234	0.82	40.43	300
Subin et al. [110]	FP8	XCVU9, 4830	5.52	34.8	150

analysis that task execution required approximately 150 ms. Table 4.10 highlights that the proposed system, when deployed on PYNQ-z2, remains a cost-effective alternative, delivering a comparable latency performance using Memory - 630 KB, and Storage - 16 MB, as previously mentioned. The SYCore, equipped with a CAESAR control engine, is capable of achieving a maximum of 29.68 inference frames per joule.

Chapter 5

Conclusion & Future Scope

In this thesis, we present a suite of reconfigurable and resource-efficient hardware solutions for edge-AI acceleration, combining the strengths of HOAA, LPRE, and a CORDIC-based RPE architecture. The proposed HOAA integrates a novel P1A adder that enhances Two's complement subtraction, rounding-to-even, and configurable activation functions by performing a +1 operation within the same cycle. This leads to a 33% area reduction and 21% power savings compared to state-of-the-art designs, with negligible accuracy trade-offs. Extending this, the LPRE accelerator leverages logarithmic posit arithmetic and time-multiplexed hardware to improve throughput by up to 80% and reduce FPGA resource usage by 50%, making it highly suitable for real-time edge deployment. Building on this efficiency stack, we introduce a reconfigurable CORDIC-based Processing Element (RPE), which incorporates five linear stages for MAC operations along with hyperbolic and division stages to enable runtime-switchable activation functions such as ReLU, sigmoid, tanh, and softmax. This architecture supports precision scaling and is validated on both FPGA (Virtex VC-707) and 28nm CMOS platforms, showing up to 2.5× throughput improvements with significantly lower area and power footprints. With support for model sparsity and pruning, the design achieves 4.57 TOPS at 3 GHz, making it suitable for edge-AI applications including LLMs, object detection, and real-time classification, while also enabling multi-tenant workloads via a scalable multi-SYCore architecture. These contributions collectively establish a high-performance, flexible hardware foundation for next-generation AI systems operating under stringent resource constraints.

Chapter 6

List of Publications

- **Omkar Kokane**, Prabhat Sati, Mukul Lokhande, Santosh Kumar Vishvakarma, **"HOAA: Hybrid Overestimating Approximate Adder for Enhanced Performance Processing Engine"** ***2024 28th International Symposium on VLSI Design and Test (VDAT)***, , Vellore, India, July 2024, pp. 1–6. DOI: 10.1109/VDAT63601.2024.10705729. **(Published)**

- **Omkar Kokane**, Mukul Lokhande, Gopal Raut, Adam Teman, Santosh Kumar Vishvakarma, **"LPRE: Logarithmic Posit-enabled Reconfigurable Edge-AI Engine"**, ***IEEE International Symposium on Circuits and Systems (ISCAS)***, London, UK, May 25–28, 2025. **(Accepted)**

- **Omkar Kokane**, Adam Teman, Anushka Jha, Guru Prasath S. L., Gopal Raut, Mukul Lokhande, S. V. Chand, Tanushree Dewangan, Santosh Kumar Vishvakarma, **"CORDIC is All You Need"**, ***ACM Transactions on Reconfigurable Technology and Systems (TRETS)***, 2025. **(Under Review)**

- Vijay Pratap Sharma, **Omkar Kokane**, Santosh Kumar Vishvakarma, **"TREA: Time-multiplexed Resource Efficient edge-AI accelerator"**, ***ACM Journal on Emerging Technologies in Computing (JETC)***, 2025. **(Under Review)**

Bibliography

[1] W. Mao, K. Li, Q. Cheng, L. Dai, B. Li, X. Xie, H. Li, L. Lin, and H. Yu, "A configurable floating-point multiple-precision processing element for HPC and AI converged computing", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, pp. 213–226, Dec 2021.

[2] N. Ashar, G. Raut, V. Treevedi, S. K. Vishvakarma, and A. Kumar, "Quant-MAC: Enhancing Hardware Performance in DNNs With Quantize Enabled Multiply-Accumulate Unit", *IEEE Access*, vol. 12, pp. 43600–43614, Mar 2024.

[3] 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, pp. 1–32, May 2023.

[4] L. Crespo, P. Tomás, N. Roma, and N. Neves, "Unified posit/IEEE-754 vector MAC unit for transprecision computing", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, pp. 2478–2482, Mar 2022.

[5] 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, Nov 2022.

[6] M. Basavaraju, V. Rayapati, and M. Rao, "Exploring Hardware Activation Function Design: CORDIC Architecture in Diverse Floating Formats", in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, Jun 2024.

[7] G. Raut *et al.*, "Designing a performance-centric mac unit with pipelined architecture for dnn accelerators", *Circuits, Systems, and Signal Processing*, vol. 42, pp. 6089–6115, May 2023.

[8] D. Zhu *et al.*, "Efficient precision-adjustable architecture for softmax function in deep learning", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, pp. 3382–3386, Dec. 2020.

[9] M. Wang *et al.*, "A high-speed and low-complexity architecture for softmax function in deep learning", in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 223–226, Sep 2018.

[10] M. Basavaraju *et al.*, "Exploring Hardware Activation Function Design: CORDIC Architecture in Diverse Floating Formats", in *25th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–8, 2024.

[11] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network", *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 243–254, Aug 2016.

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

[13] 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, pp. 7596–7614, Aug 2023.

[14] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications", *Proceedings of the IEEE*, vol. 108, pp. 2108–2135, Aug 2020.

[15] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey", *ACM Computing Surveys*, vol. 55, pp. 1–36, Mar 2022.

[16] M. H. S. Javadi, M. Faryabi, and H. R. Mahdiani, "A comprehensive model for efficient design space exploration of imprecise computational blocks", *ACM Transactions on Embedded Computing Systems*, vol. 22, pp. 1–20, Sep 2023.

[17] Y. S. Shao, "Next-Generation Domain-Specific Accelerators: From Hardware to System", in *2024 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–5, May 2024.

[18] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han, "Enable deep learning on mobile devices: Methods, systems, and applications", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, pp. 1–50, Mar 2022.

[19] A. Ramachandran, Z. Wan, G. Jeong, J. Gustafson, and T. Krishna, "Algorithm-Hardware co-Design of distribution-aware Logarithmic-Posit Encodings for Efficient DNN Inference", *61st IEEE/ACM Design Automation Conference (DAC)*, 2024.

[20] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization", *IEEE Transactions on neural networks*, vol. 18, pp. 880–888, May 2007.

[21] N. Ashar *et al.*, "Quantmac: Enhancing hardware performance in dnns with quantize enabled multiply-accumulate unit", *IEEE Access*, vol. 12, pp. 43600–43614, 2024.

[22] W. J. Dally, R. Venkatesan, and B. K. Khailany, "Neural network accelerator using logarithmic-based arithmetic", Apr. 2024.

[23] S. Walia, B. V. Tej, A. Kabra, J. Devnath, and J. Mekie, "Fast and low-power quantized fixed posit high-accuracy DNN implementation", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, pp. 108–111, Dec 2021.

[24] A. Erdem, C. Silvano, T. Boesch, A. C. Ornstein, S. P. Singh, and G. Desoli, "Runtime design space exploration and mapping of DCNNs for the ultra-low-power orlando SoC", *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, no. 2, pp. 1–25, 2020.

[25] A. Krishna, S. R. Nudurupati, D. Chandana, P. Dwivedi, A. van Schaik, M. Mehendale, and C. S. Thakur, "RAMAN: A re-configurable and sparse tinyML accelerator for inference on edge", *IEEE Internet of Things Journal*, 2024.

[26] B. George, O. J. Omer, Z. Choudhury, A. V, and S. Subramoney, "A unified programmable edge matrix processor for deep neural networks and matrix algebra", *ACM Trans. Embed. Comput. Syst.*, vol. 21, Apr 2022.

[27] M. Lokhande, G. Raut, and S. K. Vishvakarma, "Flex-PE: Flexible and SIMD Multi-Precision Processing Element for AI Workloads", *arXiv preprint arXiv:2412.11702*, 2024.

[28] J. Tong, A. Itagi, P. Chatarasi, and T. Krishna, "A reconfigurable accelerator with Data Reordering Support for Low-Cost On-Chip Dataflow Switching", in *ACM/IEEE Annual International Symposium on Computer Architecture*, 2024.

[29] S. Tiwari, N. Gala, C. Rebeiro, and V. Kamakoti, "PERI: A configurable posit enabled RISC-V core", *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 3, pp. 1–26, 2021.

[30] B. Wu, T. Yu, K. Chen, and W. Liu, "Edge-side fine-grained sparse CNN accelerator with efficient dynamic pruning scheme", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Jan 2024.

[31] R. Morcel *et al.*, "Feathernet: An accelerated convolutional neural network design for resource-constrained fpgas", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, Jun 2019.

[32] A. Boutros *et al.*, "You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, Jul 2018.

[33] O. Kokane *et al.*, "DA-VINCI: Dynamically-configurable Activation function for Versatile Neuron via CORDIC Implementation", Dec. 2024.

[34] G. Raut *et al.*, "An empirical approach to enhance performance for scalable cordic-based deep neural networks", *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, p. 32, May 2023.

[35] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", *arXiv*, Sep 2015.

[36] P. K. Meher *et al.*, "50 Years of CORDIC: Algorithms, Architectures, and Applications", *IEEE TCAS-I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.

[37] G. Raut *et al.*, "A cordic based configurable activation function for ann applications", in *2020 IEEE computer society annual symposium on VLSI (ISVLSI)*, pp. 78–83, IEEE, Mar 2020.

[38] M. Lokhande *et al.*, "Flex-PE: Flexible and SIMD Multi-Precision Processing Element for AI Workloads", Dec. 2024.

[39] S. Aggarwal *et al.*, "Reconfigurable cordic architectures for multi-mode and multi-trajectory operations", in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2490–2494, IEEE, Jul 2014.

[40] A. S. Roy, R. Biswas, and A. S. Dhar, "On fast and exact computation of error metrics in approximate lsb adders", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 876–889, 2020.

[41] C. K. Jha, A. Nandi, and J. Mekie, "Single exact single approximate adders and single exact dual approximate adders", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, pp. 907–916, May 2023.

[42] 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, no. 8, pp. 1595–1599, 2018.

[43] Y. Wang, D. Deng, L. Liu, S. Wei, and S. Yin, "PL-NPU: An energy-efficient edge-device DNN training processor with posit-based logarithm-domain computing", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, pp. 4042–4055, Jun 2022.

[44] M. A. Hanif *et al.*, "Cann: Curable approximations for high-performance deep neural network accelerators", in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, (New York, NY, USA), Association for Computing Machinery, Jan 2019.

[45] S. S. Sarwar *et al.*, "Energy efficient neural computing: A study of cross-layer approximations", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, pp. 796–809, May 2018.

[46] J. Choquette, "Nvidia hopper h100 gpu: Scaling performance", *IEEE Micro*, vol. 43, pp. 9–17, Mar 2023.

[47] J.-S. Park *et al.*, "A multi-mode 8k-mac hw-utilization-aware neural processing unit with a unified multi-precision datapath in 4-nm flagship mobile soc", *IEEE Journal of Solid-State Circuits*, vol. 58, pp. 189–202, Oct 2023.

[48] B. George *et al.*, "A unified programmable edge matrix processor for deep neural networks and matrix algebra", vol. 21, Apr 2022.

[49] Y. N. Wu *et al.*, "Highlight: Efficient and flexible dnn acceleration with hierarchical structured sparsity", MICRO '23, (New York, NY, USA), Association for Computing Machinery, Mar 2023.

[50] L. Vachhani *et al.*, "Efficient fpga realization of cordic with application to robotic exploration", *IEEE Transactions on Industrial Electronics*, vol. 56, pp. 4915–4929, Jul 2009.

[51] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks", *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 127–138, Nov 2017.

[52] G. Raut *et al.*, "Data multiplexed and hardware reused architecture for deep neural network accelerator", *Neurocomputing*, vol. 486, pp. 147–159, 2022.

[53] G. Raut *et al.*, "RECON: Resource-Efficient CORDIC-Based Neuron Architecture", *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 170–181, Jan. 2021.

[54] D. Tian *et al.*, "A low-latency power series approximate computing and architecture for co-calculation of division and square root", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Feb 2024.

[55] P. K. Meher *et al.*, "Cordic designs for fixed angle of rotation", *IEEE transactions on very large scale integration (VLSI) systems*, vol. 21, pp. 217–228, Mar 2012.

[56] S. Aggarwal *et al.*, "Scale-free hyperbolic cordic processor and its application to waveform generation", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, pp. 314–326, Jan 2012.

[57] L. Vachhani *et al.*, "Efficient cordic algorithms and architectures for low area and high throughput implementation", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, pp. 61–65, Jan 2009.

[58] P. Dong *et al.*, "Eq-vit: Algorithm-hardware co-design for end-to-end acceleration of real-time vision transformer inference on versal acap architecture", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, pp. 3949–3960, Nov 2024.

[59] S. Han *et al.*, "Eie: efficient inference engine on compressed deep neural network", in *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, p. 243–254, IEEE Press, Aug 2016.

[60] J. Tong *et al.*, "Feather: A reconfigurable accelerator with data reordering support for low-cost on-chip dataflow switching", *arXiv preprint arXiv:2405.13170*, 2024.

[61] D. Nagaraju *et al.*, "Slimmer cnns through feature approximation and kernel size reduction", *IEEE Open Journal of Circuits and Systems*, Jul 2023.

[62] O. Kokane *et al.*, "Lpre: Logarithmic posit-enabled reconfigurable edge-ai engine", Dec. 2024.

[63] O. Kokane *et al.*, "HOAA: Hybrid Overestimating Approximate Adder for Enhanced Performance Processing Engine", *VDAT*, pp. 1–6, Oct. 2024.

[64] A. Nechi *et al.*, "Fpga-based deep learning inference accelerators: Where are we standing?", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, Sep 2023.

[65] J. Zhuang *et al.*, "Charm 2.0: Composing heterogeneous accelerators for deep learning on versal acap architecture", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, Aug 2024.

[66] Y. Li *et al.*, "Fiberflex: Real-time fpga-based intelligent and distributed fiber sensor system for pedestrian recognition", *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, pp. 1–30, Aug 2024.

[67] S. Aggarwal *et al.*, "Concept, design, and implementation of reconfigurable cordic", *IEEE transactions on very large scale integration (VLSI) systems*, vol. 24, pp. 1588–1592, Jul 2015.

[68] P. Sinha *et al.*, "Fully-pipelined cordic implementation of subspace-based speech enhancement", in *2007 50th Midwest Symposium on Circuits and Systems*, pp. 97–100, IEEE, Jul 2007.

[69] S. Mehra *et al.*, "An empirical evaluation of enhanced performance softmax function in deep learning", *IEEE Access*, vol. 11, pp. 34912–34924, Apr 2023.

[70] H. Ji *et al.*, "Communication-aware and resource-efficient noc-based architecture for cnn acceleration", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 14, pp. 440–454, Aug 2024.

[71] S. Ollivier *et al.*, "Sustainable ai processing at the edge", *IEEE Micro*, vol. 43, pp. 19–28, Nov 2023.

[72] X. Wu *et al.*, "ReAFM: A Reconfigurable Nonlinear Activation Function Module for Neural Networks", *IEEE TCAS-II: Express Briefs*, vol. 70, pp. 2660–2664, July 2023.

[73] Y. Fu *et al.*, "SoftAct: A High-Precision Softmax Architecture for Transformers Supporting Nonlinear Functions", *IEEE Tran. on Circuits and Systems for Video Technology*, vol. 34, pp. 8912–8923, Apr 2024.

[74] T. Yang *et al.*, "Design Space Exploration of NN Activation Function Circuits", *IEEE TCAD*, vol. 38, pp. 1974–1978, Oct. 2019.

[75] N. A. Mohamed *et al.*, "A Unified Parallel CORDIC-Based Hardware Architecture for LSTM Network Acceleration", *IEEE Tran. on Computers*, vol. 72, pp. 2752–2766, Oct. 2023.

[76] W. Shen *et al.*, "Efficient CORDIC-Based Activation Functions for RNN Acceleration on FPGAs", *IEEE Tran. on AI*, pp. 1–11, Oct. 2024.

[77] A. Krishna, S. Rohit Nudurupati, D. G. Chandana, P. Dwivedi, A. van Schaik, M. Mehendale, and C. S. Thakur, "RAMAN: A Reconfigurable and Sparse tinyML Accelerator for Inference on Edge", *IEEE Internet of Things Journal*, vol. 11, pp. 24831–24845, July 2024.

[78] Y.-H. Chen *et al.*, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, pp. 292–308, Jun 2019.

[79] D. Pau *et al.*, "Comparing industry frameworks with deeply quantized neural networks on microcontrollers", in *2021 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–6, Mar 2021.

[80] D. Rossi *et al.*, "Energy-efficient near-threshold parallel computing: The pulpv2 cluster", *IEEE Micro*, vol. 37, pp. 20–31, Oct 2017.

[81] A. Ottaviano *et al.*, "Cheshire: A lightweight, linux-capable risc-v host platform for domain-specific accelerator plug-in", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, pp. 3777–3781, Jun 2023.

[82] H. Waris *et al.*, "Hybrid low radix encoding-based approximate booth multipliers", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, pp. 3367–3371, Feb 2020.

[83] A. M. D. Inc., "Xilinx LogiCORE IP Multiplier v12.0", Feb, 2024.

[84] Z. G. Liu *et al.*, "Systolic tensor array: An efficient structured-sparse gemm accelerator for mobile cnn inference", *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 34–37, 2020.

[85] C. Wu *et al.*, "Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, Nov 2021.

[86] R. Pilipović *et al.*, "On the design of logarithmic multiplier using radix-4 booth encoding", *IEEE Access*, vol. 8, pp. 64578–64590, Apr 2020.

[87] H. Liu *et al.*, "A 3-d multi-precision scalable systolic fma architecture", *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–12, Nov 2024.

[88] R. Pilipović *et al.*, "A two-stage operand trimming approximate logarithmic multiplier", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, pp. 2535–2545, Apr 2021.

[89] M. S. Ansari *et al.*, "An improved logarithmic multiplier for energy-efficient neural computing", *IEEE Transactions on Computers*, vol. 70, pp. 614–625, May 2021.

[90] X. Yin *et al.*, "An efficient hardware accelerator for block sparse convolutional neural networks on fpga", *IEEE Embedded Systems Letters*, 2023.

[91] P. Yin *et al.*, "Design and analysis of energy-efficient dynamic range approximate logarithmic multipliers for machine learning", *IEEE Transactions on Sustainable Computing*, vol. 6, pp. 612–625, Jun 2021.

[92] M. S. Kim *et al.*, "Efficient mitchell's approximate log multipliers for convolutional neural networks", *IEEE Transactions on Computers*, vol. 68, pp. 660–675, Nov 2023.

[93] V. Leon *et al.*, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, pp. 421–430, Nov 2018.

[94] W. Liu *et al.*, "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, 20.

[95] R. Zendegani *et al.*, "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, 20121.

[96] C. Hao *et al.* in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, (New York, NY, USA), Association for Computing Machinery, Jan 2019.

[97] Y. Yang *et al.*, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas", in *Proceedings of the 2019 ACM/SIGDA International*

Symposium on Field-Programmable Gate Arrays, FPGA '19, (New York, NY, USA), p. 23–32, Association for Computing Machinery, Jan 2019.

[98] B. Wu *et al.*, "Edge-side fine-grained sparse cnn accelerator with efficient dynamic pruning scheme", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, pp. 1285–1298, Jan 2024.

[99] L. Lu *et al.*, "Spwa: an efficient sparse winograd convolutional neural networks accelerator on fpgas", in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), Association for Computing Machinery, Apr 2018.

[100] A. Aimar *et al.*, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, pp. 644–656, Jul 2019.

[101] X. Xie *et al.*, "An efficient and flexible accelerator design for sparse convolutional neural networks", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, pp. 2936–2949, May 2021.

[102] B. McDanel *et al.*, "Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation", in *Proceedings of the ACM International Conference on Supercomputing*, ICS '19, (New York, NY, USA), p. 449–460, Association for Computing Machinery, Apr 2019.

[103] Y. Zhang *et al.*, "High-Precision Method and Architecture for Base-2 Softmax Function in DNN Training", *IEEE TCAS-I: Regular Papers*, vol. 70, pp. 3268–3279, Aug. 2023.

[104] K. Chen *et al.*, "Approximate softmax functions for energy-efficient deep neural networks", *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, vol. 31, pp. 4–16, Jan. 2023.

[105] M. Kim *et al.*, "A low-latency fpga accelerator for yolov3-tiny with flexible layerwise mapping and dataflow", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, pp. 1158–1171, Dec 2024.

[106] W. Jiang *et al.*, "A high-throughput full-dataflow mobilenetv2 accelerator on edge fpga", *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 42, p. 1532–1545, Aug 2023.

[107] T. Adiono *et al.*, "Fast and scalable multicore yolov3-tiny accelerator using input stationary systolic architecture", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, pp. 1774–1787, Aug 2023.

[108] D. T. Nguyen *et al.*, "Layer-specific optimization for mixed data flow with mixed precision in fpga design for cnn-based object detectors", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, pp. 2450–2464, Aug 2021.

[109] W. Lee *et al.*, "A real-time object detection processor with xnor-based variable-precision computing unit", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, pp. 749–761, Mar 2023.

[110] S. Ki *et al.*, "Dedicated fpga implementation of the gaussian tinyyolov3 accelerator", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, pp. 3882–3886, Jun 2023.