IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 February 2024, accepted 29 February 2024, date of publication 20 March 2024, date of current version 28 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3379906

RESEARCH ARTICLE

QuantMAC: Enhancing Hardware Performance in DNNs With Quantize Enabled Multiply-Accumulate Unit

NEHA ASHAR[1], GOPAL RAUT[1,2], (Member, IEEE), VASUNDHARA TRIVEDI[1], (Student Member, IEEE), SANTOSH KUMAR VISHVAKARMA[1], (Senior Member, IEEE), AND AKASH KUMAR[3], (Senior Member, IEEE)

[1]Department of Electrical Engineering, Indian Institute of Technology Indore, Indore 453552, India
[2]Center for Development of Advanced Computing, Bengaluru 560100, India
[3]Chair for Processor Design, Center for Advancing Electronics Dresden, Technische Universität Dresden, 01169 Dresden, Germany

Corresponding author: Akash Kumar (akash.kumar@tu-dresden.de)

This work was supported by the Ministry of Education, India.

ABSTRACT In response to the escalating demand for hardware-efficient Deep Neural Network (DNN) architectures, we present a novel quantize-enabled multiply-accumulate (MAC) unit. Our methodology employs a right shift-and-add computation for MAC operation, enabling runtime truncation without additional hardware. This architecture optimally utilizes hardware resources, enhancing throughput performance while reducing computational complexity through bit-truncation techniques. Our key methodology involves designing a hardware-efficient MAC computational algorithm that supports both iterative and pipeline implementations, catering to diverse hardware efficiency or enhanced throughput requirements in accelerators. Additionally, we introduce a processing element (PE) with a pre-loading bias scheme, reducing one clock delay and eliminating the need for conventional extra resources in PE implementation. The PE facilitates quantization-based MAC calculations through an efficient bit-truncation method, removing the necessity for extra hardware logic. This versatile PE accommodates variable bit-precision with a dynamic fraction part within the sfxpt$<N,f>$ representation, meeting specific model or layer demands. Through software emulation, our proposed approach demonstrates minimal accuracy loss, revealing under 1.6% loss for LeNet-5 using MNIST and around 4% for ResNet-18 and VGG-16 with CIFAR-10 in the sfxpt$<8,5>$ format compared to conventional float32-based implementations. Hardware performance parameters on the Xilinx-Virtex-7 board unveil a 37% reduction in area utilization and a 45% reduction in power consumption compared to the best state-of-the-art MAC architecture. Extending the proposed MAC to a LeNet DNN model results in a 42% reduction in resource requirements and a significant 27% reduction in delay. This architecture provides notable advantages for resource-efficient, high-throughput edge-AI applications.

共8位元，其中5個是小數

INDEX TERMS Approximate compute, bit-truncation, CORDIC, deep neural network, hardware accelerator, quantize processing element.

I. INTRODUCTION

Artificial Intelligence (AI) has become an integral part of our daily lives, enhancing the intelligence of various devices. AI solutions use a two-stage process for finding the best DNN: network training and inference accuracy evaluation. DNN is trained using application datasets, and model accuracy is validated with the test set. The use of collected data for training Deep Neural Network (DNN) models has significantly improved object recognition and

The associate editor coordinating the review of this manuscript and approving it for publication was Ludovico Minati.

classification capabilities [1]. DNNs typically comprise an input layer, multiple hidden layers, and an output layer, with the depth of hidden layers varying based on the complexity of the input data or the task to be performed. As data quality advances, with images having higher pixel density, the need for larger and more layered DNNs arises to classify abundant high-quality input data effectively. For complex datasets, precision computations with additional hidden layers are essential to achieve greater accuracy. In essence, neural networks either require more hidden layers or an equivalent number of neurons in fewer hidden layers to maintain high efficiency and accuracy [2]. As the demand for AI applications grows, optimizing DNN architecture becomes crucial for handling intricate datasets effectively.

To achieve accelerated DNN computations, a large number of resources are needed. For instance, VGG-16 requires 138 million weights and 15.5 billion multiply-accumulate (MAC) operations to process a single 224 × 224 input image [3]. Over 90% of the MAC operations are typically dedicated to DNN models. This research focuses on analyzing the resource distribution among key DNN layers: Convolution Layer, Fully-Connected (FC) Layer, and Recurrent Layer. Notably, the Convolution Layer stands out with its relatively smaller number of weights per filter compared to the FC and Recurrent Layers. Understanding these resource allocations can aid in optimizing DNN performance and accelerating AI solutions efficiently. In the domain of DNNs, a prevalent approach for achieving better performance involves employing 2D systolic architectures with processing elements for parallel computations [4]. Figure 1 depicts the basic 2D systolic architecture used for convolution computation, along with the internal architecture of the processing element (PE), which is integral to the array. Conventionally, the PE consists of a MAC unit where the multiplier and accumulator blocks are included, performing fundamental arithmetic operations followed by nonlinear activation functions (AFs) with higher utilization. We employ fixed-point or floating-point arithmetic computation based on the nature of the information and the system's requirements. Nevertheless, fixed-point arithmetic schemes are the preferred choice for hardware implementation, albeit with some trade-off in accuracy [5]. Considering the error resilience of neural networks, reducing precision may not significantly impact output accuracy.

The PE, being resource-intensive, plays a vital role in neural network computations, impacting the model's physical performance, throughput, and accuracy. It is versatile, catering to multiplication and accumulation for computations in convolution and fully connected layers. It can dynamically select bias values and add them to the accumulated result. However, achieving this compatibility requires an additional multiplexer for bias selection and addition, leading to increased hardware resources and introducing extra critical delay as depicted in Figure 1. Additionally, the multiplier significantly consumes hardware resources, necessitating double the output bit precision compared to the input,

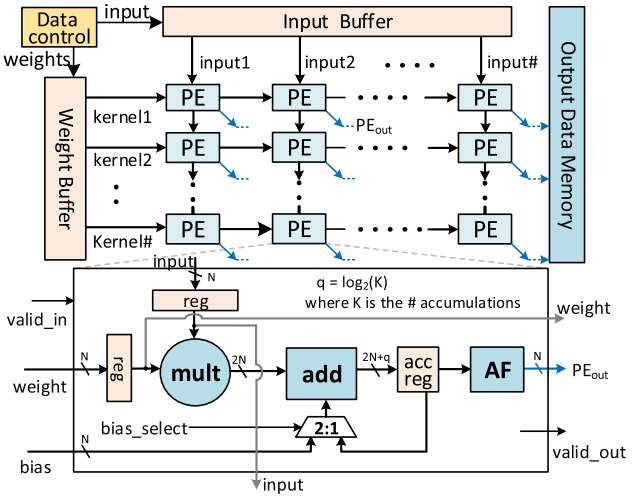

FIGURE 1. Classic 2D systolic array architecture for processing elements (PEs) employed in DNN accelerators includes on-chip memory, data control, and interface.

resulting in higher resource requirements in subsequent blocks. Researchers explored architecture optimization at various abstraction levels, including different arithmetic computation types, quantization/approximation in computation to reduce precision while maintaining accuracy, and hardware reuse/reduction. However, improving one parameter often entails compromising another, resulting in trade-offs among performance parameters [6].

Diverse hardware architectures are essential for various applications, particularly in computing elements like MAC units. These encompass DSPs, FPGA-based systems, and taped-out ASIC designs. DSP-based structures excel at accommodating multiple multiplication-based algorithms, ensuring functional scalability. However, this comes at the expense of heightened hardware utilization, and underutilized hardcore architecture can result in power wastage. In contrast, ASICs provide limited logic flexibility. While researchers have explored reconfigurable ASIC designs, they exhibit restricted functional scalability and logic configuration [7].

FPGA-based implementations offer both logic flexibility and optimized performance, despite hardware constraints when increasing bit-precision [8]. To address hardware complexity, we present an efficient MAC computing architecture for fixed-point computation across all bit precisions. The flexible fixed-point representation, denoted as sfxpt<*N*,*f*>, is depicted in Figure 2. It illustrates that for a given bit-width *N*, the data can have a variable number of *f* fractional bits. This study highlights streamlined arithmetic multiplication, leading to reduced hardware, improved throughput, and the ability to quantize output precision without additional components. Moreover, a pre-bias loading scheme introduces customizable bias values for fully connected layer calculations while maintaining a zero value during convolution layer computations. The principal contributions of this work are as follows:

- Design a hardware-efficient multiply-accumulate architecture with limited adders and shifters for signed fixed-point computation. It supports both iterative and pipeline implementations.
- A processing element, with a pre-bias loading scheme, that supports storing bias values in the accumulation register, no extra hardware is required; enabling convolution and fully connected computations.
- Introduce a processing element that enables quantization-based multiply-accumulate calculations through an efficient bit truncation method, eliminating the need for additional hardware logic. Consequently, we appropriately name it 'QuantMAC.'
- Development of versatile processing element capable of accommodating variable bit-precision within the sfxpt<*N,f*> representation, based on the specific demands of the model or layer.

The design of the quantize-aware MAC (QuantMAC) unit offers key benefits, including reduced resource utilization and minimal power consumption, while incurring only insignificant accuracy loss. The proposed PE, based on a signed fixed-point N-bit design, demonstrates significant advantages, particularly with the 8-bit implementation, requiring 37% fewer resources and incurring a mere 1-2% accuracy loss with MNIST compared to state-of-the-art accurate PE designs for the sfxpt<*8,5*> representation.

The remainder of this paper is organized as follows: Section II covers related work on quantize-enabled MAC architectures and comparative analysis of efficient design techniques. Section III elaborates on the proposed PE architecture along with its characteristics and the state machine that is controlling the architecture. Section IV discusses the impact of QuantMAC on convolutional neural networks, emphasizing precision overhead, scalability and efficiency. Section V presents the software emulation and hardware implementation flow used for the proposed architecture evaluation. The inference accuracy on the DNN model through software emulation for the proposed PE architecture and comparison of physical performance parameters by implementing on FPGA hardware is analyzed in Section VI. Finally, Section VII provides the conclusion of the article and future work.

II. RELATED WORK AND MOTIVATION

This section provides a comprehensive overview of DNNs in the context of AI, focusing on strategies to enhance computational efficiency through MAC optimization. A thorough examination of DNN fundamentals, hardware platforms, and cost-reduction techniques has been undertaken for efficient AI processing in [9]. Research on ASIC, FPGA, GPU, TPU, and CPU for DNN deployment in AI-driven mobile devices has intensified [10]. To cope with area and power constraints in computation-intensive DNN layers [11], efficient architectures at different abstraction levels are being explored [12]. ASIC designs like TPUs are being adopted for on-chip acceleration [13]. Cutting-edge frameworks, such

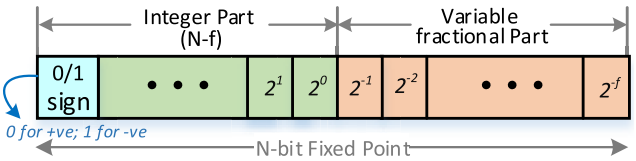

FIGURE 2. Visual representation illustrating the total and fractional bits in a signed fixed-point sfxpt<*N,f*>, employed for design implementation and parameter extraction.

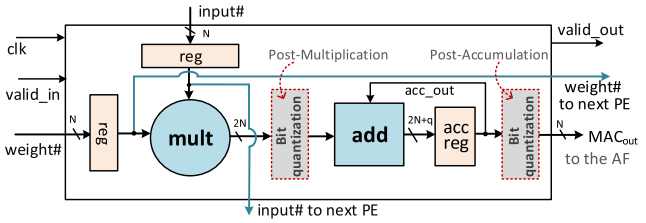

FIGURE 3. The conventional approach to bit quantization in the MAC unit involves carrying it out either after multiplication or accumulation, with the goal of reducing precision while maintaining accuracy.

as Eyeriss [14], Gemmini [15], and Simba [16], utilize ASIC/FPGA implementations, with a focus on optimizing arithmetic computation, data precision, and design architecture. In this research article, efficient MAC designs in DNN were evolved and extensively explored. Additionally, the research aims at hardware efficiency to reduce area utilization [17] in AI-enabled IoT and mobile applications. Further, there are research articles that explore various techniques to achieve an optimum performance in MAC design. One such technique involves using the Wallace tree with a carry-select adder, providing high throughput but at the cost of increased resource utilization [18]. Another approach is the use of Modified Booth's algorithm, aiming to achieve multiplication with minimal resource consumption [3]. Additionally, the paper considers the Vedic multiplier-based approach, addressing efficient MAC computation with lower bit-precision implementation [19]. However, it is essential to note that increasing accuracy through higher bit precision in PE computation leads to more complex architectures and introduces critical delays.

The architecture of a DNN is heavily influenced by the number of MAC computations performed by PEs and the number of weights in fully connected layers. Consequently, researchers have extensively explored application-targeted MAC designs to achieve efficient performance. One such technique is the use of approximation in Ristretto for the DNN framework proposed by the authors [23]. This model simplifies number representation and reduces word data width. Studies recommend using 8-bit and higher fixed-point arithmetic for complex datasets [22]. While higher precision increases hardware cost, the PE architecture should be scalable for any precision with fixed-point representation. To enhance hardware performance, quantized DNN models have been explored [24]. The performance analysis of this technique was conducted in [24]. On the other hand, lower

TABLE 1. State-of-the-art methodologies and their features in fixed-point multiply-accumulate units for neural network inference.

Fixed-Point Arithmetic Multiply-Accumulate Unit	Dynamic fraction (f)	Variable bit-precision	Data Pipelining	Physical Overhead (Area-Power-Delay)	Accuracy Support Inference	Throughput
Accurate Xilinx DSP [20]	Yes	Yes (DSP size limit)	No	High	High	Medium
Accurate Modified Booth [3]	No	Yes	Yes	Medium	High	Medium
Reconfigurable Truncation [21]	No	Yes (not scalable)	No	High	Variable	Medium
Iterative MAC [22]	No	Yes	No	Low	Medium	Low
CORDIC MAC [6]	Yes	Yes (Pipeline Stages)	Yes	Medium	Medium	High
This Work:QuantMAC	Yes	Yes (Any precision)	Yes	Low	Medium	High

precision architectures (e.g., 4-bit, 2-bit, or 1-bit) come with minimal resources but sacrifice accuracy for complex datasets.

The research article discusses the effectiveness of Binary Neural Networks (BNNs) in reducing precision and conserving resources and memory access [25]. BNNs utilize a binary format for both weights and activations, resulting in faster computations with minimal impact on classification accuracy. In some datasets, augmenting one more bit to constrain weights and activations to $+1$ and -1 yields accuracy comparable to 8- or 16-bit counterparts [26], albeit with limitations for complex datasets. Continuing investigations in recent times explore data reuse techniques and quantization in numbers [27]. These techniques have been integrated into coarse-grained and fine-grained architectures. Coarse-grained architectures treat PE arrays and memory blocks as black boxes, while fine-grained gate-level implementations allow for a more flexible architecture, enabling a deeper understanding of individual blocks of interest [28]. An example of such an architecture is Eyeriss v2, where data compression is applied at global buffers and off-chip DRAM, which can also be extended to the processing elements. This technique significantly improves energy efficiency and enhances the throughput of the network [14]. A novel data truncation technique has emerged to reduce data bit-width, exhibiting 25× improved performance on multi-core GAP-8 SoC compared to certain STM micro-controllers [24]. Furthermore, overflow-aware quantization using floating-point arithmetic benefits the inference process, as explored in [29].

Table 1 summarizes the design features and performance support adopted by the different approaches for MAC unit design and highlights some features on which the performance of architectures can be evaluated. The accurate approaches mentioned in the first two rows of the table deliver high accuracy but at the cost of decreased physical performance and lesser throughput [3], [20]. These architectures also have the liberty to handle scalable bit-precision but within the limits of the hardware used. The case is quite different in the case of approximation-friendly MAC units like the CORDIC MAC [6], Iterative MAC [22] and MAC with reconfigurable truncation [21]. The physical performance of such MAC techniques is higher, but there is a minor dip in the accuracy delivered by these algorithms. Few designs like the MAC with reconfigurable truncation [21], have higher area overhead due to an increase in the circuitry required to incorporate reconfigurability. However, this overhead is quite less than the area overhead of accurate MAC units. Some architectures provide higher throughput but exhibit degraded accuracy. Various techniques for MAC computation and normalization of weights are explored in one of the research articles, wherein by normalizing the weights and representing them as powers of two, MAC operations can be efficiently performed using the shift-and-add algorithm [30]. The authors reported that this design improved power and energy by over 50%, but it also led to a noticeable increase in critical delay due to its iterative architecture. A new approach investigates the benefits of using CORDIC-based configurable architecture for MAC and AF computations [22]. This approach allows for iterative and pipeline architectures, targeting low area-power and high throughput applications. The iterative computation technique has been explored within the CORDIC-based computation [31]. Unlike the left shift algorithm used in a previous work [32], the proposed technique uses the right shift and add algorithm. These methods reduce hardware utilization but come with a trade-off in throughput. However, quantized-aware techniques have not been explored to reduce hardware complexity in such architectures.

The importance of quantization is highlighted by considering input feature maps and weights with the same data bit precision, denoted by 'N' bits. Using a conventional multiplier will result in an output size of more than '$2N$' bits, increasing the complexity of subsequent computing elements within the DNN as shown in Figure 1. The output of the conventional MAC requires a bit-width of '$2N + q$', where the bit overhead 'q' is determined by $q = log_2(K)$, with 'K' representing the accumulation count within the MAC computation. The output precision doubles the input bit width with overflow bits in MAC. Since the MAC output is the input to non-linear AFs, a high precision AF block is conventionally required [33]. Alternatively, to maintain a minimum precision of AF (N-bit), the output of the MAC unit needs to be quantized to N-bit. Moreover, the bit precision increases significantly as the unquantized model moves from the first layer to subsequent multiple layers within the DNNs. To mitigate this increase in data precision, state-of-the-art MAC architectures with quantization features have been illustrated in Figure 3.

However, these designs require additional hardware overhead and computational delay. State-of-the-art approaches have attempted to minimize certain parameters at the expense of others among various performance metrics, such as computations, power, delay, and hardware utilization. Previous works have mainly focused on optimizing one parameter, leading to bottlenecks in other aspects. In contrast, unlike the state-of-the-art designs, this work addresses quantize-aware MAC computation in PE, ensuring the same input-output bandwidth without any hardware overhead. The proposed design features low hardware cost, minimal critical delay, and quantization-based computations for all precision fixed-point computations. This comprehensive approach aims to achieve a well-balanced optimization across multiple performance metrics, thus offering a promising solution for efficient and effective MAC architectures.

III. NOVEL HARDWARE DESIGN FOR MAC COMPUTATION AND QUANTIZATION-ENABLED PROCESSING

The section showcases contributions in hardware design, with a primary focus on enhancing MAC computations for improved efficiency and enabling quantization processing, as exemplified by the proposed QuantMAC architecture. It introduces a hardware-efficient MAC architecture that supports iterative and pipeline implementations, utilizing limited adders and shifters. A pre-bias loading scheme is also introduced, allowing for bias storage in the accumulation register. This, in turn, facilitates convolution and fully connected computations without the need for additional hardware. In the context of efficient hardware design for DNNs, FPGA and ASIC-based implementations require an effective quantization scheme for MAC computations. The section also outlines a novel PE design that enables quantized MAC computation through an efficient bit truncation method. Furthermore, a versatile PE is developed for all-bit fixed-point computation, catering to applications that demand variable precision. This section provides an overview of advancements in hardware architecture, specifically focusing on improved fixed-point computations and quantization techniques.

The exploration of optimizing computing architecture for multiplication and addition has significantly impacted the design of MAC units. These algorithms can be executed either sequentially or in a pipeline manner. While the iterative approach conserves resources and reduces power consumption, it often introduces delays that impact the throughput. Conversely, parallel or pipelined logic reduces latency and increases throughput, although it has higher resource utilization [30]. This section delves into quantization within MAC units to formulate a resource-efficient design. The primary objective of quantization is to decrease the bit width propagated through the network while minimizing accuracy loss. Quantization can be implemented before or after the accumulator stage [24]. The data can be quantized before accumulation and after accumulation, or simply post-accumulation, depicted in Figure 3. However, the latter case requires a larger accumulator size of $(2N + q)$, while in the former case, the accumulator size would be $(N + q)$. Additional hardware components are necessary in both cases, leading to area overhead and increased critical delay. We propose a MAC with an innovative quantization scheme to address these challenges and enhance efficiency.

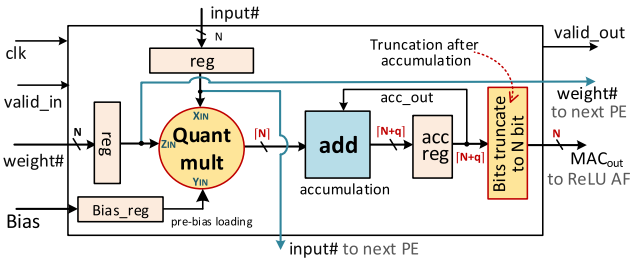

FIGURE 4. Integrate quantization-aware data flow and control signals into the proposed design of the multiply-and-accumulate unit.

A. DYNAMIC QUANTIZATION-ENABLED MULTIPLICATION FOR MAC UNITS

The proposed technique for quantize-enabled multiplication has the inherent capability of employing the shift-to-right and add method. The design of the multiply-and-accumulate unit leverages a quantized-aware data-flow and control signal scheme. This scheme is vividly illustrated in Figure 4, situated towards the end. This algorithm accommodates both iterative and pipelined architectures. In the iterative architecture, a supplementary delay is introduced based on the number of iterations entailed in the computation. This framework empowers the output to uphold the same bit precision as the input, thereby negating the necessity for a distinct quantization block subsequent to the multiplier block. The dynamic quantization unfolds during each iteration of product generation. This dynamic process is visualized in Figure 4, providing a comprehensive representation of the data flow and control signals orchestrating bit-truncated multiplication within the MAC unit.

The computational equation, executed by the *quant_mult* function, is succinctly expressed in Eq. 1. Here, the product of the multiplication $(X_{in} \times W_{in})$ and the bias addition (Y_{in}) is achieved by amalgamating the input X_{in} with a right-shifted variant of itself. This procedure is exhaustively elucidated in Eq. 1. The foundation of this implementation rests upon the established technique of multiplication via right shift and accumulation. The coefficient a_i, which remains unknown, is meticulously determined by progressively driving the weight matrix W towards zero, bit by bit. In instances where the i^{th} bit of input X_{in} is non-zero, the input X_{in} is subjected to a one-bit right-shift and is then added to the prevailing value of Y_i within the proposed architecture. This iterative procedure persists until W inevitably converges to zero, thereby culminating in the signed product of the input feature X_{in} and the weight W_{in}, ultimately denoted

as X_j. Notably, the iterative computation is underpinned by a solitary MAC operation, yielding a granularity of 1-bit accuracy per iteration. Notably, the output's precision is intricately entwined with the precision of the input feature. Consequently, in a computation characterized by n-bit precision, an inherent error of approximately 2^{-n} arises due to the incomplete convergence of W_n towards zero.

$$X_j = X_{in} \times W_{in} = X_{in} \times \sum_{i=1}^{j} a_i \times 2^{-i} = \sum_{i=1}^{j} a_i \times \left(X_{in} \times 2^{-i}\right) \quad (1)$$

The research article aims to enhance the performance of hardware implementation in the MAC unit by integrating a pipeline architecture. The primary goal has been to increase the computational throughput, thus expediting the evaluation of Eq. 1, as depicted in Figure 5. The algorithm can be designed for both a single-stage iterative architecture and a pipeline architecture, further amplifying throughput capabilities. Our architectural depiction has the inherent flexibility to be extended across multiple pipeline stages denoted as #k, or alternatively, the architecture can be repurposed effectively to function as a single stage, usable within #k clocks. The proposed method computes the MAC operation following Eq. 1. A truncated computation approach has been adopted for the same equations to enhance efficiency in terms of bit-width.

The innovative technique employs the shift-and-add method to facilitate multiplication, requiring several iterations or spanning multiple pipeline stages before yielding the final output. The objective of enhancing computational throughput has resulted in the serialization of the computational architecture, where the data pipeline concept is embraced, as depicted in Figure 5. This pipeline framework streamlines the design, resulting in significant savings in clock cycles. Notably, the design has the capacity to complete an N-bit multiplication in fewer than 'k' clock cycles. Conventionally, 'k' corresponds to N. Each stage within this pipeline framework encompasses dedicated bit-shifter and adder/subtractor blocks, which have been thoughtfully orchestrated to optimize efficiency. The algorithm is adaptable for utilization in both DSPs and Look-Up Tables (LUTs). Furthermore, it also has compatibility with ASICs, employing gate-level netlist. This versatile resource utilization allows for a wider spectrum of implementation options within the FPGA domain.

We have initially considered the bias value Y_0, input feature X_0, where W_0 represents the corresponding weight, and Y_n denotes the final desired output. During intermediate iterations, X_i has been employed as the input for the i^{th} pipeline stage, and Y_i has functioned as the temporary variable within the output register, enabling addition and subtraction operations. Subsequently, W_i has corresponded to the processed weight at the i^{th} stage. To evaluate the accuracy, the corresponding neuron weights

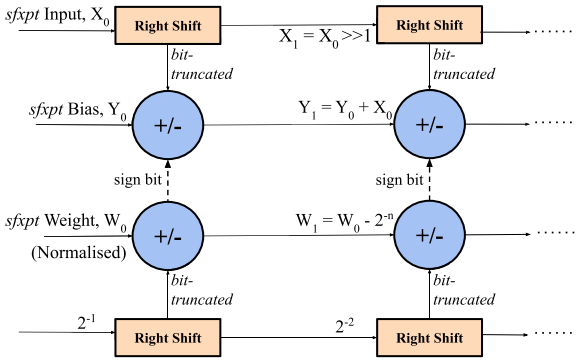

FIGURE 5. Enhance performance pipeline architecture to increase computational throughput. The design can be configured for both single-stage iterative architecture and pipeline architecture to enhance throughput.

have been generated in a maximum normalized (mn) form throughout the neural network's training process. The accomplishment of weight normalization has been achieved using Eq. 2.

$$mn(W) = \frac{W_i}{max(W)} \quad (2)$$

Here, W_i represents the weight at the current stage, while $max(W_i)$ denotes the upper limit of the weight's potential values. This limit is determined by considering its precision and the characteristics of fixed-point representations.

The first sub-process has entailed the reduction of the weight (W_i) by utilizing negative powers of two (i.e., 2^{-i}) during each iteration, aiming to achieve a weight that is nearly zero as per Eq. 3. In this context, 'i' signifies an iteration encompassing a range from 0 to n (i = 0, 1, 2,..., n). Through every successive iteration, the weight value steadily converges towards zero. Simultaneously, as the weight approaches zero, the level of accuracy experiences enhancement. To assess performance parameters and accuracy comprehensively, we have adopted a notation involving a 1-bit whole integer and fractional bits in fixed-point representation. The least significant bits (LSBs) of the right-shifted weights bear an insignificantly small value, which is either disregarded or truncated prior to the subsequent addition step. Conversely, the second sub-process has encompassed the operation of adding or subtracting X_i and Y_i, wherein Y_i encapsulates the output from the preceding step. In contrast, X_i comprises the value of the previous layer X_i that has undergone both right-shifting and bit-truncation within the prior stage. This modified X_i is then subjected to addition with the new Y_i in cases where the subtraction yields a negative result. Conversely, for positive outcomes, the reverse process is executed. In Eq. 4, the term *trunc* signifies the bit-truncated output derived from the input function. The detailed computation elucidated within Eq. 4 encompasses multiplication and bias addition procedures. The shift operations do not impose additional bit

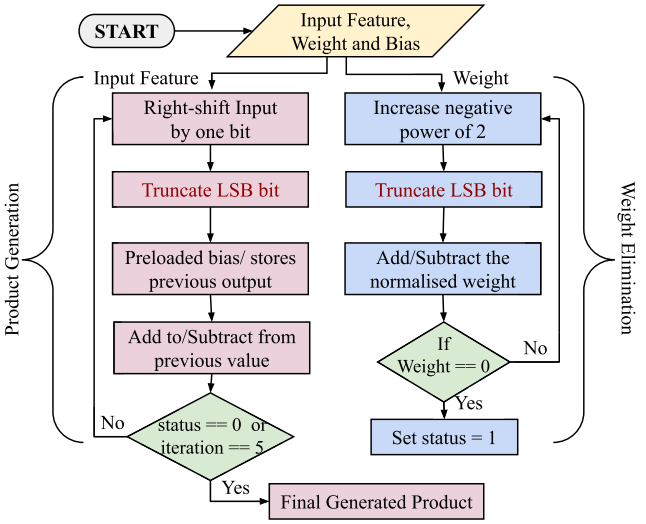

FIGURE 6. Flowchart of the proposed algorithm for QuantMAC architecture. Here, the input feature map multiplies with the weights simply by shifting for each binary position and eliminates the weighted positioned bit in every iteration.

requirements or require extra clock cycles.

$$W_{i+1} = mn(W_i) \pm 2^{-(i+1)} \tag{3}$$

$$\begin{aligned} Y_{i+1} &= Y_i \pm trunc(X_i \times 2^{-i}) \\ &= Y_i \pm trunc(X_i \gg 1) \end{aligned} \tag{4}$$

B. EFFICIENT ITERATIVE ARCHITECTURE AND RESOURCE-CONSCIOUS COMPUTING

The algorithm's implementation involves iterative computation, conserving resources, and minimizing power consumption, albeit introducing delays impacting throughput. This architecture employs shifters, adder/subtractors, and multiplexers to choose a feedback path for input, utilizing the previous output of the same hardware. The feedback mechanism and single-stage iterative calculation architecture are depicted in Figure 6. In this setup, the primary process encompasses shift-and-add operations for quantized-enabled product generation and weight elimination, where weights can be positive, negative, or zero. The proposed architecture employs a signed 8-bit fixed-point arithmetic scheme. An example magnitude calculation spanning three iterations, with sampled data for performance insights, is illustrated in Figure 7. In this example, values are subjected to computation to yield a 8-bit signed product, consisting of one signed bit, two integer bits, and five fractional bits. In this instance, the input considered is 1.593, which undergoes iterative addition and subtraction with its shifted counterpart. The determination of the next operation's sign hinges on the summation or difference of the weights derived from the preceding stage. The least significant bit (LSBs) is discarded with each iteration, resulting in output truncation. The number of iterations is predefined based on bit precision, or they continue until the weight reaches zero, with higher bit precision demanding a greater number of iterations.

Iteration	Weight, $Z = (0.875)_{10} = (00.11100)_2$ Weight Output, $Z_{n+1} = Z_n \pm 2^{-n}$	Input, $X = (1.59375)_{10} = (01.10011)_2$ Product Output, $Y_{n+1} = Y_n \pm X_n * 2^{-n}$
Iter. 1	00.11100 - 00.10000	01.10011 (truncated bits) - 00.11001 | 1
Iter. 2	+ 00.01100 - 00.01000	00.11010 + 00.01101 | 0
Iter. 3	+ 00.00100 - 00.00100	01.00111 + 00.00110 | 1
Output	00.00000	01.01101

(weight reduced to 0)

FIGURE 7. Example multiplication for the proposed algorithm with a signed 8-bit fixed-point arithmetic scheme. Sampled calculation values have been taken from one of the trained data samples.

The hardware implementation leverages either DSP or LUT. The algorithm truncates the insignificant LSB during each shift operation, which is applicable to various bit-precision fixed-point arithmetic. Dynamic truncation of the LSB occurs with each shift, influencing shift and adding iterations. Two parallel operations occur per stage: the first right-shifts the input by one bit and adds/subtracts it with the previous iteration's output, truncating the LSB before addition. The second operation focuses on weight and memory constant addition/subtraction, driving the weight towards zero. This technique maintains output precision similar to the input and eliminates the need for a separate quantization block, dynamically quantizing during product generation iterations. The data flow and control signals of bit-truncated multiplication within a MAC unit are depicted in Figure 4.

C. STATE MACHINE-DIRECTED QUANTIZED MULTIPLICATION AND PE REALIZATION

The utilization of the Shift-and-add technique plays a crucial role in the process of generating products and quantize-enable multiplication computation. The weights involved in this process can assume positive, negative, or zero values, as illustrated in Figure 7. In order to orchestrate the various computational components within a neuron, control signals are harnessed to determine the initiation of the AF. Each computational block or stage within the neuron is steered by corresponding signals, which dictate their operational behaviour. To illustrate this mechanism, we present the state-machine framework for the processing engine, encompassing both the MAC unit and the AF unit, as depicted in Figure 8. The initial phase of this state machine entails the emission of an `init` signal, which serves as a trigger to initiate the multiplication process. Conversely, when the `init` signal is set low, the machine remains idle.

Central to our proposed approach is introducing a `status` signal within the multiplier block. This signal remains low throughout the iterative pipeline operations, signifying ongoing calculations. Once all iterations are successfully completed, the `status` signal transitions to a high state, thus enabling the accumulation phase. Simultaneously, the accumulator block employs an `index` signal to sustain

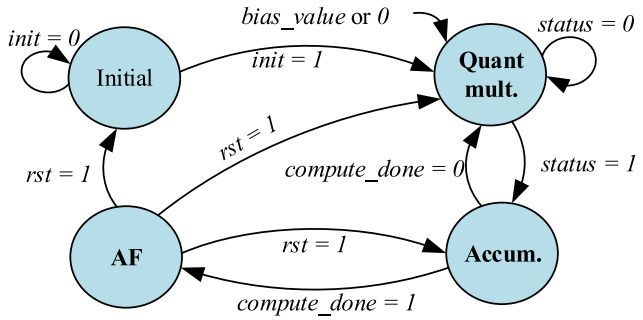

FIGURE 8. Finite State-machine of a proposed MAC-based neuron control engine, featuring the QuantMAC module followed by an activation function, in coordination with control signals.

the activation of the multiplier block until all accumulation operations are concluded. Subsequently, as accumulations finalize, the `index` signal is elevated to a high state, thereby facilitating the engagement of the AF block for subsequent computations. The newly devised MAC unit encompasses input pins that receive input features and corresponding weights. These inputs are supplemented by control signals, namely `compute_init` and `compute_done`, which govern the initiation and termination of computation cycles. The multiplication outcomes are funnelled into the accumulator, where they undergo recursive addition and are stored within the accumulator register. Upon completion of all accumulation tasks, the state machine manipulates the `compute_done` signal to a high state. This elevated state triggers the AF transformation within the neuron, culminating in the execution of the desired computational processes.

We have conducted an exhaustive analysis of error metrics by processing observed data using the proposed algorithm across varying numbers of iterations (pipeline stages). For each iteration, the Mean Squared Error (MSE), Mean Absolute Error (MAE), and Standard Deviation (SD) have been computed by comparing the observed data with the true accumulated unit. As specified, normalized weights have been utilized to evaluate error metrics. We have quantified shift-and-add operations through a pre-computation Pareto analysis, which considers input precision and the desired output accuracy. This Pareto analysis has enabled us to discern the impact of different pipeline stages on the accuracy of the algorithm. The results of our Pareto analysis have been graphically presented in Figure 9, illustrating

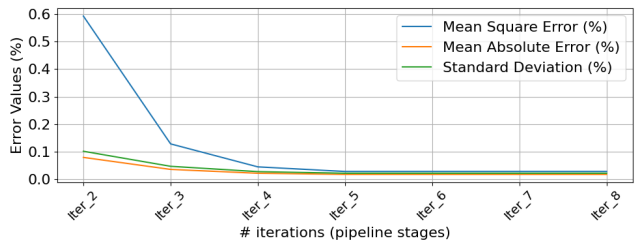

FIGURE 9. The error evaluation distribution of sfxpt<*8,5*> across various iterations for Pareto analysis and Pareto-point extraction.

the evolution of error metrics as the number of pipeline stages has increased. Notably, the graph has revealed that the error values rapidly converge to a minimal level after approximately five pipeline stages. This observation suggests that employing more than five stages would yield negligible improvements in accuracy as the error metrics plateau beyond this point. Furthermore, a crucial insight has been gained by analyzing the graphical trends. It has become evident that maintaining five pipeline stages balances computational accuracy and resource utilization. This choice optimally balances factors such as area, power consumption, and critical delay, ensuring an efficient and effective implementation of the processing element. This insight supports our broader objective of optimizing the algorithm's performance within real-world hardware constraints.

IV. EXTENDING CONVOLUTION OPERATION WITH PE INTEGRATION

This analysis holds particular relevance for individual MAC units. The implications of these changes extend to the broader network context, especially as more QuantMAC units are integrated. Let's consider a convolutional neural network to provide a comprehensive understanding, as illustrated in Figure 1. We will start with a specific example extracted from a trained module to demonstrate the integration of PEs within the convolutional operation. The accompanying MAC calculation, shown in Figure 7, showcases 8-bit signed values representing weights and inputs. The accumulated product of each multiplication, denoted as Y_n, is computed using a $(N + q)$-bit adder, resulting in a final accumulated sum denoted as R, as described by Eq (5). Here, N signifies the bit precision of the i^{th} convolutional layer and $q \geq 2 \times \log_2(K)$, accounting for the bit overhead due to K accumulations.

$$R_i = \sum_{a=1}^{n} (Y_n)_a \tag{5}$$

The convolutional network involves an input feature matrix with dimensions of $m \times m$ for grayscale images, denoted as $(m \times m)^{(N)}$, and for colour images, the dimensions are $m \times m \times 3$, denoted as $(m \times m \times 3)^{(N)}$. The kernel matrices are of size $p \times p$ denoted as $(p \times p)^{(i,N)}$, with a stride value of s applied over the image. For instance, considering a grayscale input image containing values from X_{11}, X_{12}, and so on up to X_{mm}, while the weight values range from W_{11}, W_{12}, extending to W_{pp}. In this context, let $(r \times r)_i^{(2N+q)}$ denote the output dimensions of convolutional layer i. Each of these accumulations corresponds to a single pixel in the output feature image of the convolutional layer. After each convolutional layer, pooling is performed; max-pooling is applied in this case. The pooled images have dimensions of $t \times t$. Comparing the effect of the proposed quantize-enabled MAC with the conventional MAC computation is elucidated below through equations. The equations for convolution with the conventional MAC are as follows:

$$(r \times r)_1^{(2N+q)} = \sum_{a=1}^{n} [(m \times m)^{(N)} * (p \times p)_1^{(N)}]_a \tag{6a}$$

$$(t \times t)_1^{(2N+q)} = \texttt{maxpool}\left[(r \times r)_1^{(2N+q)}\right] \tag{6b}$$

Here, the asterisk ($*$) denotes convolution. From Eq. 8, it is evident that the output precision of convolution conventionally is $2N + q$. Furthermore, the impact of unquantized values increasing the input feature precision affects the network as it progresses from one layer to the next. Hence, the extent of scalability is limited in networks employing the conventional MAC design. It's noticeable from Eq. 6 that when moving from the first layer to the next, the bit precision increases by more than twice that of the input bit width. This limitation contrasts with the proposed MAC design, which is capable of scalability to higher or lower bit precision. The Equation 7 describes the relationship between the dimensions of the output feature maps at different layers in the network. Specifically, it expresses that the dimensions of the output feature maps denoted as $\texttt{dim}\big(r \times r, t \times t\big)_i$, for layer i are twice the dimensions of the input feature maps $\texttt{dim}\big(m \times m, p \times p\big)_{i-1}$ from the previous layer. This equation highlights the hierarchical nature of feature extraction in the network, where each subsequent layer's output dimensions are expanded by a factor of two compared to the previous layer's input dimensions.

$$\texttt{dim}\big(r \times r, t \times t\big)_i = 2 \times \texttt{dim}\big(m \times m, p \times p\big)_{i-1} \tag{7}$$

Hence, quantifying the computed data to match the input precision is necessary for uniform computation with minimal resource overhead. Consequently, extra hardware is needed to quantize the $2N + q$ output precision to the N input precision, a challenge addressed by the proposed quantize-enabled MAC, as elucidated through Equation (8) which illustrates quantize enabled convolution and subsequent max-pooling:

$$\big(r \times r\big)_1^{(N+q)} = \textstyle\sum_{a=1}^{n} [\big(m \times m\big)^{(N)} * \big(p \times p\big)_1^{(N)}]_a \tag{8a}$$

$$\big(t \times t\big)_1^{(N)} = \texttt{maxpool}\big[\big(r \times r\big)_1^{(N+q)}\big] \tag{8b}$$

In Equation 8, part (a) depicts the convolution between the input feature matrix $\big(m \times m\big)^{(N)}$ and the kernel matrix $\big(p \times p\big)_1^{(N)}$. The outcome is $\big(r \times r\big)_1^{(N+q)}$, representing convolutional layer 1 with dimensions $(r \times r)$ and bit precision $(N + q)$. In Equation 8, part (b) involves max-pooling applied to the output of convolutional layer 1, yielding $\big(t \times t\big)_1^{(N)}$. Max-pooling selects the maximum value from a neighbourhood, reducing dimensions while preserving crucial information. When the bit-precision of conventional MAC computation is considered for the convolution operation, it requires additional hardware to quantize the output to N. However, it is observable that the output precision after multiplication and accumulation is $N + q$ bits, achieved using the proposed quantize-enabled MAC unit. Subsequently, this output is resized to N bits before pooling. The q overhead bits are truncated for smoother data propagation through the network.

$$\big(r \times r\big)_2^{(N+q)} = \textstyle\sum_{a=1}^{n} [\big(t \times t\big)_1^{(N)} * \big(p \times p\big)_2^{(N)}]_a \tag{9a}$$

$$\big(t \times t\big)_2^{(N)} = \texttt{maxpool}\big[\big(r \times r\big)_2^{(N+q)}\big] \ldots \tag{9b}$$

In Equation 9, part (a) explains how we combine $\big(t \times t\big)_1^{(N+q)}$ from the first layer with $\big(p \times p\big)_2^{(N)}$ in a specific way, creating $\big(r \times r\big)_2^{(N)}$. On the other hand, in Equation 9, part (b) illustrates how we take the highest values from the outcome of the second layer's convolution, leading to $\big(t \times t\big)_2^{(N)}$. By comparing these steps to how we traditionally calculate results, we see how our new quantize-enabled MAC process stands out. This comparison highlights how our approach is more adaptable and efficient. These actions are crucial for making neural networks more skilled at recognizing complex patterns in input information, improving their learning ability.

V. EXPERIMENT

We conducted a comprehensive evaluation of the MAC unit architecture, incorporating quantization-aware computation. The experiments were designed to validate both the accuracy of deep learning models and the hardware performance for compatibility with edge-AI applications. In assessing the accuracy of the proposed QuantMAC, various experiments were conducted. Firstly, the DNN application was equipped with an error-resilient feature. Given the design's support for pipeline computation, we analyzed the error matrix for QuantMAC, reporting metrics such as MSE, MAE, and SD across different pipeline stages/iterations. To understand the impact of increasing pipeline stages on computational accuracy, we applied Pareto analysis, revealing that error parameters approach nearly zero values with five or more pipeline stages, as illustrated in Figure 9. Moreover, an extensive inference accuracy analysis was performed using a software emulator. The MAC unit, mirroring the hardware architecture's features, was implemented in Python and validated on various DNN models, including LeNet, ResNet, and VGG16.

The focus was on dynamic signed fixed-point sfxpt<*8,5*> arithmetic representation, employing five pipeline stages/iterations based on the error matrix observations. This strategic choice aimed to minimize hardware overhead while optimizing the benefits of a pipeline architecture. Accuracy calculations were conducted using MNIST and CIFAR-10 datasets. To ensure a fair comparison, we implemented the state-of-the-art MAC unit on the same FPGA board, maintaining similar arithmetic precision for MAC computation. The performance parameters were meticulously evaluated for both the proposed MAC unit and the state-of-the-art MAC units. Further, we implement LeNet-5 using QuantMAC for analyzing the performance. The implementation took place on the Xilinx Virtex-7. The LeNet Hardware, featuring different types of MAC units, underwent comprehensive performance evaluation and comparison. These experiments aimed to provide insights into the standalone MAC unit's hardware performance, compare it with existing state-of-the-art designs and assess its effectiveness in the context of DNN models, particularly LeNet-5. Following key evaluations were conducted to validate the design performance:

- **Pareto Analysis:** Conducted a Pareto analysis to calculate error metrics, determining optimal pipeline stages for hardware utilization.
- **Inference Accuracy Validation:** Utilized software emulation to validate inference accuracy for diverse DNN models incorporating the QuantMAC unit across various bit precisions and image datasets (MNIST and CIFAR-10).
- **Hardware Implementation & Performance Evaluation:** Evaluated the performance of hardware implementations, encompassing both a standalone MAC unit and the LeNet DNN design featuring the enabled MAC unit. The architecture was crafted using a high-level design (HLD) language, with performance parameter extraction executed at selected design parameters.

VI. RESULTS AND DISCUSSION

A. INFERENCE ACCURACY COMPARISON FOR PROPOSED QUANTIZED MAC ARCHITECTURE

A comparison of inference accuracy between the proposed quantized MAC architecture and TensorFlow-based models across various bit-precision settings and DNN models has been depicted in Table 2. The evaluation focuses on the LeNet-5, ResNet-18, and VGG-16 architectures using MNIST and CIFAR-10 datasets. For the 8-bit bit-precision setting, the proposed QuantMAC architecture achieves slightly lower accuracy compared to TensorFlow-based models. Specifically, in MNIST, QuantMAC achieves 97.2% accuracy, representing a 1.6% decrease compared to TensorFlow's 98.8%. In CIFAR-10, QuantMAC achieves 78.6% accuracy, indicating a 2.1% decrease compared to TensorFlow's 80.7%. Despite these decrements, the observed accuracy loss is considered insignificant. For the 12-bit and 16-bit bit-precision settings, similar trends are observed, with QuantMAC achieving slightly lower accuracy compared to TensorFlow. It is crucial to focus on computation approximation and the associated compromise in accuracy. While acknowledging accuracy losses ranging from 1% to less than 4% across various datasets and DNN models for 8-bit precision, it is noteworthy that these losses diminish with increasing precision, being less than 1% in 16-bit precision. The proposed approach results in significant hardware resource savings compared to accurate computing architectures with 16-bit precision, as elaborated in the subsections below in Table 4 and Table 5. These minor accuracy reductions demonstrate that the QuantMAC architecture maintains a high level of accuracy, making it a viable choice for DNN model inference while remarkably improving hardware efficiency.

It is important to note that we observed a nearly 4% accuracy loss for CIFAR-10, signalling concerns about accuracy degradation in 8-bit precision. However, the article addresses these accuracy concerns in 8-bit precision, emphasizing the crucial role of pipeline stages in MAC computation accuracy. QuantMAC employs five stages for error resilience, optimizing accuracy based on Pareto analysis. Higher precision mitigates accuracy loss, emphasizing the need for an efficient design balancing area and performance. The proposed architecture is scalable for precise computations, allowing flexibility in pipeline stages. While enhancing accuracy in complex feature maps is feasible with more stages, it comes at the cost of reduced throughput or increased area overhead. The trade-off between accuracy and throughput is central, with scalability's impact minimal. Notably, accuracy differences between proposed and conventional designs decrease at higher bit-precision, showcasing the proposed design's merits.

B. QUANTMAC COMPARATIVE ANALYSIS OF HARDWARE IMPLEMENTATION PERFORMANCE

A resource utilization and performance parameters comparison at sfxpt$<8,5>$ for the proposed quantized MAC architecture and state-of-the-art MAC architectures are presented in Table 3. The resources under consideration include LUTs and FFs of the FPGA board, while the performance parameters encompass critical path delay and power-delay product. The proposed quantized MAC architecture outperforms existing solutions, demonstrating significantly reduced resource utilization. Specifically, it has employed 52 LUTs and 88 FFs, resulting in a remarkable reduction in hardware resources compared to alternative architectures extracted from different research works. For example, the proposed design has achieved a 37.35% improvement in LUT utilization and a significant 44.26% enhancement in FF utilization compared to Booth's architecture, showcasing its superior resource efficiency. Furthermore, the proposed design excels in terms of critical path delay, with a minimal value of 1.53 ns, signifying superior speed and efficiency. In addition to resource efficiency, the proposed architecture boasts an impressively low power-delay product of 1.01 pJ, further solidifying its position as a hardware-efficient solution. These findings underscore the scalability and efficiency of the proposed design, which is capable of achieving exceptional performance across different bit-precision settings while minimizing hardware requirements.

This code should render the table correctly in Overleaf with the specified formatting and content.

Furthermore, we compared the Accurate/Approximate (acc_app) MAC architecture, which consumes 62 LUTs, with our proposed design showcasing a more efficient LUT utilization, requiring only 52. The proposed design demonstrates superior critical path delay and power-delay-product compared to the acc_app MAC unit. However, it excels in computation time, as the number of clock cycles needed for its execution is merely 1. In contrast, the proposed design requires an initial extra 4 cycles due to its five-stage pipeline, but subsequent computations necessitate only a single clock delay to produce the final result. On the other hand, the shift-and-add method demands 5 clock cycles for each computation, resulting in an overall requirement of 5 times as many clocks compared to the acc_app MAC unit.

TABLE 2. Performance comparison of the proposed quantized MAC architecture across different bit-precision settings and DNN models in terms of inference accuracy.

Bit-Precision	LeNet-5 (%)				ResNet-18 (%)		VGG-16 (%)	
Dynamic Fixed-Point	MNIST		CIFAR-10					
	TensorFlow [34]	QuantMAC	TensorFlow [34]	QuantMAC	TensorFlow [34]	QuantMAC	TensorFlow [34]	QuantMAC
8-bit	98.8	97.2	80.7	78.6	86.1	82.3	85.1	81.6
12-bit	98.9	97.6	80.8	79.3	86.2	83.7	85.9	83.01
16-bit	98.9	97.8	81.2	79.8	87.0	86.4	86.3	85.0

TABLE 3. Comparative analysis of multiply-accumulate units with sfxpt<8,5>: Revealing resource utilization and performance metrics.

Resources	LUT	FF	Critical Delay (ns)	Total Clocks	Power-delay Product (pJ)
Vedic [35]	159	245	4.48	1	6.11
Wallace [18]	105	112	2.59	1	3.29
Booth [3]	83	61	3.08	1	3.07
Shift-add [32]	75	58	1.12	5 (approx)	4.17
Acc_App [36]	62	59	2.87	1	2.04
CORDIC [22]	23	22	1.82	5	1.90
QuantMAC	52	88	1.53	1 (initial 4)	1.01

TABLE 4. Analyzing the impact of precision scalability and performance metrics for both conventional and proposed QuantMAC architectures.

Bit Precision	LUT	FF	Total Delay (ns)	Total Dynamic Power (mW)
8-bit Comb. Logic [20]	86	16	3.816	6.6
8-bit Proposed	52	88	1.57	6.36
12-bit Comb. Logic [20]	187	24	4.451	10.26
12-bit Proposed	75	126	1.86	8.48
16-bit Comb. Logic [20]	325	32	9.051	14.95
16-bit Proposed	106	168	2.21	11.77

Here, for a conventional shift-and-add MAC, the number of clock cycles (n) depends on the bit precision (n-bit). In this case, we consider 5 clock cycles as we are comparing it with the 5-stage QuantMAC, which produces approximate results. While remaining state-of-the-art MAC units require a single clock delay, the iterative CORDIC method demands 5 clocks for every computation, reflecting a longer execution time. Overall, the proposed design outperforms state-of-the-art designs, boasting better performance with lower resource utilization and considerable throughput. The execution time remains at one clock cycle, except for an additional 4 clocks in the initial execution.

Table 4 presents a comprehensive evaluation of the performance and scalability of the proposed QuantMAC Architecture across various bit-precision settings, comparing it to the accurate compute MAC [20]. Critical hardware utilization parameters, including LUTs, FFs, latency, and total dynamic power consumption, are meticulously scrutinized. For the 8-bit bit-precision setting, the proposed design stands out with just 52 LUTs, a stark contrast to the 86 LUTs utilized by the accurate MAC [20] for LUT utilization. It is crucial to note that the precise implementation of the conventional MAC design entails the incorporation of a multiplier, an accumulator, an additional adder for bias addition, a multiplexer for adder selection, and an overall utilization of 86 LUTs. Advancing to the 12-bit bit-precision level, the proposed design outperforms the conventional accurate MAC, employing only 75 LUTs and 126 FFs compared to 187 LUTs and 24 FFs. Furthermore, the proposed design exhibits a critical path delay of 1.81 ns, whereas the accurate MAC [20] lags behind at 2.587 ns. At the 16-bit bit-precision level, the proposed design maintains its scalability and efficiency. It utilizes a mere 95 LUTs and 168 FFs in contrast to the accurate MAC architecture's 325 LUTs and 32 FFs.

These findings unequivocally demonstrate that the Proposed Quantize-Enabled MAC Architecture outperforms the accurate compute MAC [20], significantly reducing hardware utilization, critical path latency, and power consumption. The hardware performance parameters have been evaluated across bit-precision settings from 8-bit to 16-bit.The results indicate that, with increasing bit precision, the proposed QuantMAC design's hardware parameters show only a marginal increment of approximately 2.1%. This increase is notably lower compared to conventional designs, which often experience a four to five-fold resource increase when doubling precision. For example, one can observe the resource utilization in Table 4 for conventional accurate designs when increasing precision from 8-bit to 16-bit. This analysis underscores the optimal utilization of hardware resources by the proposed architecture across all precision levels, showcasing enhanced performance as computational precision grows. These findings strongly motivate the adoption of QuantMAC in larger systems to attain superior benefits. Therefore, the analysis of the scalability and hardware performance of the proposed QuantMAC architecture across different bit precisions. The proposed arithmetic solution for the MAC architecture offers a more efficient approach, delivering enhanced performance without excessive resource overhead, making it a promising solution for a wide range of applications.

This code should render the table correctly in Overleaf with the specified formatting and content.

C. PERFORMANCE EVALUATION AND RESOURCE COMPARISON OF QUANTMAC-BASED LENET-5 ARCHITECTURE

The quantize-aware MAC architecture, QuantMAC, demonstrates remarkable enhancements in resource utilization and performance metrics in comparison to two other prominent designs: the Shift-and-Add algorithm-based design and the

Accurate MAC design. In terms of LUTs, QuantMAC outperforms the Accurate MAC, utilizing 42% fewer LUTs, which signifies a substantial improvement in LUT efficiency as reported in Table 4. While QuantMAC does require approximately 14% more LUTs than the Shift-and-Add design, this slight increase in usage is justified by the significant performance gains it delivers. Regarding Flip-Flops (FFs), QuantMAC utilizes an equivalent number to the Accurate MAC, showcasing parity in FF utilization. However, the Shift-and-Add algorithm-based design requires a substantial 79% more FFs compared to QuantMAC, underscoring QuantMAC's efficiency in FF utilization.

We have implemented the fully parallel LeNet-5 architecture using FPGA hardware, incorporating sfxpt<*8,5*> precision. Figure 10 presents a detailed depiction of the LeNet-5 hardware architecture, highlighting key components such as 5 weight ROMs (C_ROM and FC_ROM), an input feature memory (F1_RAM), and four intermediate memory banks (F_RAM) situated between the layers. The shared MAC bank, housing a total of 214 MAC units, plays a pivotal role in both convolution and fully connected layers. In each layer, we employed distinct control units (C_ctrl and FC_ctrl) for the selection of MAC units in the shared MAC bank and for weight/input feature selection. The figure showcases the integration of various input signals (write address, write data, wr_en) facilitating data loading into the input feature memory. Global signals (sys_clk, rst_n, start) are also illustrated, contributing to the synchronization of operations. Notably, the architecture's final fully connected layer incorporates 10 neurons corresponding to the 10 output classes, passing to a softmax layer for probabilistic index verification. The LeNet-5 model is designed with a feature for input grayscale images of size 28×28 and features 6 kernels of size 5×5, yielding an output from the first convolution layer of size 24×24 when the stride is set to one. Subsequent max-pooling of 2×2 reduces its size to 12×12, which is then convoluted with 16 filters of size 5×5 each, resulting in an 8×8 image size. A subsequent max-pooling operation of 2×2 produces 16 feature maps with a size of 4×4. In CONV_1, we utilized 6 parallel MAC units for the 6 filters, while in CONV_2, we employed 16 parallel MAC units for the 16 filters, facilitating parallel MAC operations for both CONV_1 and CONV_2 layers. This data, considering the flattened layer, serves as input to the FC_1 layer, which is then fed sequentially through layers FC_2 and FC_3, with a configuration of 120:84:10. For the fully connected layer, we incorporated parallel MAC units in each layer, i.e., 120, 84, and 10, respectively.

The performance parameters for the proposed architecture, as well as state-of-the-art architectures, are provided in Table 5. In the case of the Combinational logic MAC unit, the MAC operations of the total LeNet-5 model amount to 16,460 clock cycles for processing a single image. In contrast, for QuantMAC, our proposed solution requires 20 extra clocks (4 extra clock per layer) due to our 5-stage pipelined MAC architecture, necessitating 4 extra clocks

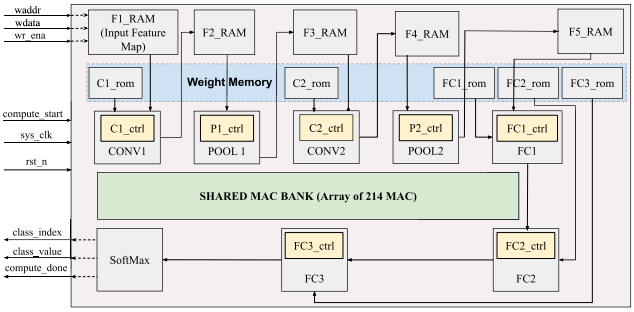

FIGURE 10. Comprehensive illustration of LeNet-5 hardware architecture.

to produce the initial result; thereafter, it takes the same number of clocks as in a combinational logic-based design. Additionally, shift-add-based computation logic requires $5\times$ clocks when considering the five-time shift and add processes, compared to the conventional design, as it involves iterative computation. Overall, as illustrated in Table 5, the proposed QuantMAC-based DNN model implementation yields superior results in terms of throughput and resource utilization. Insights into resource utilization, total delay, and dynamic power for MAC units with 8-, 12-, and 16-bit precisions are provided in Table 4. Table 5 offers DNN model performance metrics. The proposed MAC-based LeNet-5 operates at 460 MHz, while the conventional Xilinx MAC IP-based LeNet-5 runs at 262 MHz with 8-bit precision. Notably, increasing precision to 16-bit in conventional designs significantly amplifies critical path delay, as presented in Table 4, leading to the DNN model supporting 66 MHz for 16-bit precision. However, the conventional MAC-based DNN model at 16-bit precision operates at 50 MHz for a better slack margin. Therefore, the proposed MAC-based DNN model also operates at 50 MHz, reporting performance parameters for a fair analysis alongside state-of-the-art designs under identical conditions and highlighting its potential. It is important to note that the proposed MAC-based DNN model with 8-bit precision achieves 460 MHz, surpassing the 262 MHz of the conventional MAC-based DNN model, showcasing its efficiency and superiority in handling various precision scenarios. Considering the LeNet-5 model and a 28×28 image, a total of 1.125 lakhs MAC operations are required. The total number of clocks per image is calculated based on the reported number of MAC operations in Table 5, taking one clock per MAC operation. The operations performed per second (in MOPs) are presented in Table 5. Hence, the overall results indicate that the proposed design achieves nearly equal throughput compared to the conventional design, with the added advantage of 64.7% lower LUT resource utilization.

One of the most noteworthy improvements is in the critical path delay, where QuantMAC excels. It achieves an impressively low critical path delay of 7.349 ns, which is significantly faster than the Accurate MAC's delay of 10.04 ns, representing a remarkable 27% reduction in

TABLE 5. Analysis of resource utilization and performance metrics for MAC implementations in LeNet-5 at a frequency of 50 MHz.

LeNet-5	LUT	FF	BRAM	Itr. Delay (ns)	Avg. Power (W)	Throughput (MOPs)	Clocks/Img
Comb. Logic [20]	35399	19459	40	10.04	0.185	683.2	16460
Shift-and-Add [32]	10964	19790	15.5	19.63	0.180	136.6	16460×5
QuantMAC	12480	19770	39	7.349	0.181	682.4	16460+20

TABLE 6. FPGA-based hardware performance parameters for QuantMAC with 5 pipeline stages on the LeNet-5 network using HDL.

Bit Precision	LUT	FF	Critical Delay (ns)	Total Clocks	Throughput (MOPs)
8-bit QuantMAC	12480	19770	1.5698	16460	682.4
12-bit QuantMAC	19985	26470	1.641	–	–
16-bit QuantMAC	29838	38678	1.813	–	–

delay. Moreover, when compared to the Shift-and-Add algorithm-based design, QuantMAC's critical path delay is approximately 12.7 ns faster, signifying an outstanding 65% boost in performance. QuantMAC emerges as an exceptionally efficient MAC architecture, achieving substantial reductions in LUT utilization and critical path delay when compared to both the Accurate MAC and the Shift-and-Add algorithm-based designs. These improvements underscore QuantMAC's effectiveness in optimizing resource utilization and enhancing computational speed, making it a highly advantageous choice for MAC computations within the LeNet-5 architecture.

This corrected code should render the table properly in LaTeX with the specified formatting and content.

This code should render the table correctly in Overleaf with the specified formatting and content.

When transitioning from 8-bit to 16-bit precision in the QuantMAC architecture, there is a noticeable increase in resource utilization as reported in Table Table 6. At the 8-bit precision level, QuantMAC efficiently utilizes 12,480 LUTs and 19,770 FFs to perform its computations. However, as the bit precision doubles to 16 bits, the resource requirements experience a significant rise. The number of LUTs increases by approximately 139%, reaching a total of 29,838 LUTs, while the utilization of FFs sees a 95% increase, totalling 38,678 FFs. Additionally, we projected throughput at 50MHz, which remains constant for all precisions (8, 12, 16-bit) architectures. However, increasing the precision also increases the critical delay of the circuit, potentially impacting the optimal higher operating frequency.

Overall, the conventional accurate design exhibits a substantial increase in resources when doubling the precision, often necessitating four to five times more resources, QuantMAC's resource growth remains relatively modest (2.39×). This low margin of increase in resources for doubled bit-precision is a key advantage of the proposed architecture. It demonstrates that while achieving higher bit precision is associated with increased resource utilization, the QuantMAC design optimally manages this escalation, ensuring that the resource overhead is significantly lower compared to conventional designs. This not only highlights the efficiency of the QuantMAC architecture but also underscores its practicality and suitability for hardware implementations where resource constraints and power efficiency are vital considerations. As the input feature map complexity grows, achieving higher inference accuracy requires deeper and higher-precision computational neural networks. For higher bit-precision of data, the selection can move towards the complex network models. Higher complexity networks will also be able to handle larger data sets for training. Also, the data processing will be more accurate for higher bit-widths. However, limited hardware resources pose a challenge. In such scenarios, our proposed design becomes preferable, utilizing minimal resources compared to state-of-the-art methods. It exhibits reduced design complexity, especially when dealing with increased computation precision within the DNN model. The ability to strike a balance between precision and resource utilization positions QuantMAC as an attractive choice for a wide range of applications in the field of deep learning and AI applications.

VII. CONCLUSION

In essence, QuantMAC, our quantize-enabled MAC architecture, exhibits substantial improvements in resource utilization and performance metrics compared to alternative designs. Despite utilizing quantized computation and incurring minimal accuracy loss, it remains comparable to TensorFlow models across various bit-precision settings and DNN models. With minor accuracy decreases of 1-4%, QuantMAC maintains high accuracy levels, validating its suitability for DNN model inference while significantly improving hardware efficiency. The architecture outperforms existing designs, utilizing 42% fewer LUTs and achieving a remarkable 27% reduction in critical path delay compared to the Accurate MAC design. It also excels in both LUT efficiency and critical path delay (65% performance boost) when compared to the Shift-and-Add design. Furthermore, QuantMAC exhibits a modest resource increase (only 2.1%) when transitioning from 8-bit to 16-bit precision, emphasizing its efficiency. This optimized architecture, extendable to different processors like RISC-V, proves to be a compelling solution for diverse applications in deep learning and artificial intelligence, particularly in resource-constrained edge-AI solutions. The design's impact is evident in error-resilient and hardware-restrictive image detection and classification applications, laying the groundwork for future investigations into more complex models such as transformers, LSTM, and recurrent models.

ACKNOWLEDGMENT

The authors acknowledge the Ministry of Electronics and Information Technology (MeitY), India, for providing research facilities. Special thanks go to the DST-funded Indo-South Korea Joint Network Center (JNC) for Environmental Cyber-Physical Systems for providing essential research equipment.

REFERENCES

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

[2] M. Uzair and N. Jamil, "Effects of hidden layers on the efficiency of neural networks," in *Proc. IEEE 23rd Int. Multitopic Conf. (INMIC)*, Nov. 2020, pp. 1–6.

[3] F. U. D. Farrukh, C. Zhang, Y. Jiang, Z. Zhang, Z. Wang, Z. Wang, and H. Jiang, "Power efficient tiny YOLO CNN using reduced hardware resources based on booth multiplier and Wallace tree adders," *IEEE Open J. Circuits Syst.*, vol. 1, pp. 76–87, 2020.

[4] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 58–70.

[5] H. F. Langroudi, Z. Carmichael, D. Pastuch, and D. Kudithipudi, "Cheetah: Mixed low-precision hardware & software co-design framework for DNNs on the edge," 1908, *arXiv:1908.02386*.

[6] G. Raut, S. Karkun, and S. K. Vishvakarma, "An empirical approach to enhance performance for scalable CORDIC-based deep neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, pp. 1–32, Sep. 2023.

[7] S. Nelson, S. Y. Kim, J. Di, Z. Zhou, Z. Yuan, and G. Sun, "Reconfigurable ASIC implementation of asynchronous recurrent neural networks," in *Proc. 27th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Sep. 2021, pp. 48–54.

[8] K. Rajesh and G. U. Reddy, "FPGA implementation of multiplier-accumulator unit using vedic multiplier and reversible gates," in *Proc. 3rd Int. Conf. Inventive Syst. Control (ICISC)*, Jan. 2019, pp. 467–471.

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

[10] H. Zhu, C. Chen, S. Liu, Q. Zou, M. Wang, L. Zhang, X. Zeng, and C.-J. R. Shi, "A communication-aware DNN accelerator on ImageNet using in-memory entry-counting based algorithm-circuit-architecture co-design in 65-nm CMOS," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 283–294, Sep. 2020.

[11] Y. Zhang, C. Hu, and B. Jiang, "Embedded atom neural network potentials: Efficient and accurate machine learning with a physically inspired representation," *J. Phys. Chem. Lett.*, vol. 10, no. 17, pp. 4962–4967, Sep. 2019.

[12] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, Jan. 2019.

[13] N. P. Jouppi, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[14] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[15] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 769–774.

[16] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 14–27.

[17] Y. Qian, D. Wu, W. Bao, and P. Lorenz, "The Internet of Things for smart cities: Technologies and applications," *IEEE Netw.*, vol. 33, no. 2, pp. 4–5, Mar. 2019.

[18] R. B. S. Kesava, B. L. Rao, K. B. Sindhuri, and N. U. Kumar, "Low power and area efficient Wallace tree multiplier using carry select adder with binary to excess-1 converter," in *Proc. Conf. Adv. Signal Process. (CASP)*, Jun. 2016, pp. 248–253.

[19] V. Kunchigi, L. Kulkarni, and S. Kulkarni, "High speed and area efficient vedic multiplier," in *Proc. Int. Conf. Devices, Circuits Syst. (ICDCS)*, Mar. 2012, pp. 360–364.

[20] *Xilinx LogiCORE IP v12.0*. Accessed: Feb. 1, 2024. [Online]. Available: https://www.xilinx.com/support/documentation/ipdocumentation/multgen/v120/pg108-mult-gen.pdf

[21] F.-Y. Gu, I.-C. Lin, and J.-W. Lin, "A low-power and high-accuracy approximate multiplier with reconfigurable truncation," *IEEE Access*, vol. 10, pp. 60447–60458, 2022.

[22] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "RECON: Resource-efficient CORDIC-based neuron architecture," *IEEE Open J. Circuits Syst.*, vol. 2, pp. 170–181, 2021.

[23] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, Nov. 2018.

[24] N. Bruschi, A. Garofalo, F. Conti, G. Tagliavini, and D. Rossi, "Enabling mixed-precision quantized neural networks in extreme-edge devices," in *Proc. 17th ACM Int. Conf. Comput. Frontiers*, May 2020, pp. 217–220.

[25] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–11.

[26] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," 2016, *arXiv:1602.02830*.

[27] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surv.*, vol. 55, no. 4, pp. 1–36, Apr. 2023.

[28] 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, Syst., Signal Process.*, vol. 42, no. 12, pp. 7596–7614, Dec. 2023.

[29] H. Xie, Y. Song, L. Cai, and M. Li, "Overflow aware quantization: Accelerating neural network inference by low-bit multiply-accumulate operations," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 868–875.

[30] M. Masadeh, O. Hasan, and S. Tahar, "Input-conscious approximate multiply-accumulate (MAC) unit for energy-efficiency," *IEEE Access*, vol. 7, pp. 147129–147142, 2019.

[31] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "A CORDIC based configurable activation function for ANN applications," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 78–83.

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

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

[34] M. Abadi, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.

[35] A. S. Krishna Vamsi and S. R. Ramesh, "An efficient design of 16 bit MAC unit using vedic mathematics," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2019, pp. 0319–0322.

[36] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-performance accurate and approximate multipliers for FPGA-based hardware accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 2, pp. 211–224, Feb. 2022.

NEHA ASHAR received the Bachelor of Engineering (B.E.) degree in electronics engineering from the Sardar Patel Institute of Technology, India. She is currently pursuing the M.S. degree with the Department of Electrical Engineering, Indian Institute of Technology Indore, India, with a focus on hardware optimization using quantized-enabled architectures for deep neural network accelerators. Her research interests include RTL design and hardware implementation.

GOPAL RAUT (Member, IEEE) received the master's degree in VLSI design from the G. H. Raisoni College of Engineering, Nagpur, India, in 2015, and the Ph.D. degree from the Indian Institute of Technology Indore, India, in 2022. He is currently a Knowledge Associate with the Centre for Development of Advanced Computing (C-DAC), Bengaluru, India. His research interests include compute-efficient design and hardware implementation of DNN accelerators for the IoT and edge-AI applications.

VASUNDHARA TRIVEDI (Student Member, IEEE) received the Bachelor of Engineering (B.E.) degree in electronics and telecommunication engineering from the Institute of Engineering and Technology, DAVV, Indore, India, in 2019. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Indian Institute of Technology Indore, India. Her research interests include hardware optimization of deep neural network accelerators and environmental cyber-physical systems (E-CPS). She is working on approximate computing for neural networks and image processing applications.

SANTOSH KUMAR VISHVAKARMA (Senior Member, IEEE) received the Ph.D. degree from the Indian Institute of Technology Roorkee, India, in 2010. From 2009 to 2010, he was with University Graduate Center at Kjeller, Norway, as a Post-Doctoral Fellow, under the European Union Project "COMON." He is currently a Professor with the Department of Electrical Engineering, Indian Institute of Technology Indore, India, where he is also leading the Nanoscale Devices and VLSI Circuit and System Design Laboratory. His current research interests include nanoscale devices, reliable SRAM memory designs, and configurable circuit designs for the IoT applications.

AKASH KUMAR (Senior Member, IEEE) received the joint Ph.D. degree in electrical engineering in embedded systems from the University of Technology (TUe), Eindhoven, and the National University of Singapore (NUS), in 2009. From 2009 to 2015, he was with the National University of Singapore, Singapore. He is currently a Professor with Technische Universität Dresden (TUD), Germany, where he is also directing the Chair for Processor Design. His current research interests include the design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems.