SN Computer Science (2024) 5:494
https://doi.org/10.1007/s42979-024-02851-z

ORIGINAL RESEARCH

A Precision-Aware Neuron Engine for DNN Accelerators

Sudheer Vishwakarma[1] · Gopal Raut[2] · Sonu Jaiswal[2] · Santosh Kumar Vishvakarma[2] · Dhruva Ghai[1]

Received: 13 January 2024 / Accepted: 31 March 2024

Abstract

Deep Neural Networks (DNNs) form the backbone of contemporary deep learning, powering various artificial intelligence (AI) applications. However, their computational demands, primarily stemming from the resource-intensive Neuron Engine (NE), present a critical challenge. This NE comprises of Multiply-and-Accumulate (MAC) and Activation Function (AF) operations, contributing significantly to the overall computational overhead. To address these challenges, we propose a groundbreaking Precision-aware Neuron Engine (PNE) architecture, introducing a novel approach to low-bit and high-bit precision computations with minimal resource utilization. The PNE's MAC unit stands out for its innovative pre-loading of the accumulator register with a bias value, eliminating the need for additional components like an extra adder, multiplexer, and bias register. This design achieves significant resource savings, with an 8-bit signed fixed-point implementation demonstrating notable reductions in resource utilization, critical delay, and power-delay product compared to conventional architectures. An 8-bit sfixed < N, q > implementation of the MAC in the PNE shows 29.23% savings in resource utilization and 32.91% savings in critical delay compared with IEEE architecture, and 24.91% savings in PDP (power-delay product) compared with booth architecture. Our comprehensive evaluation showcases the PNE's efficacy in maintaining inferential accuracy across quantized and unquantized models. The proposed design not only achieves precision-awareness with a minimal increase ($\approx 10\%$) in resource overhead, but also achieves a remarkable 34.61% increase in throughput and reduction in critical delay (34.37% faster than conventional design), highlighting its efficiency gains and superior performance in PNE computations. Software emulator shows minimal accuracy losses ranging from 0.6% to 1.6%, the PNE proves its versatility across different precisions and datasets, including MNIST (on LeNet) and ImageNet (on CaffeNet). The flexibility and configurability of the PNE make it a promising solution for precision-aware neuron processing, particularly in edge AI applications with stringent hardware constraints. This research contributes a pivotal advancement towards enhancing the efficiency of DNN computations through precision-aware architecture, paving the way for more resource-efficient and high-performance AI systems.

Keywords Deep neural networks · Neuron engine · Edge-AI · Multiply-accumulate unit · Activation function · Precision-aware architecture · Approximate computing

Introduction and Motivation

The demand for efficient deep learning (DL) hardware is escalating with the increasing need for advanced AI applications. Within the realm of DL, deep neural networks (DNNs)

Sudheer Vishwakarma and Gopal Raut have contributed equally to this work.

✉ Dhruva Ghai
dhruvaghai@orientaluniversity.in

Sudheer Vishwakarma
vsudheer062@orientaluniversity.in

Gopal Raut
gopalraut05@gmail.com

Sonu Jaiswal
phd2101191002@iiti.ac.in

Santosh Kumar Vishvakarma
skvishvakarma@iiti.ac.in

[1] Department of Electronics and Communication Engineering, Oriental University, Indore, India

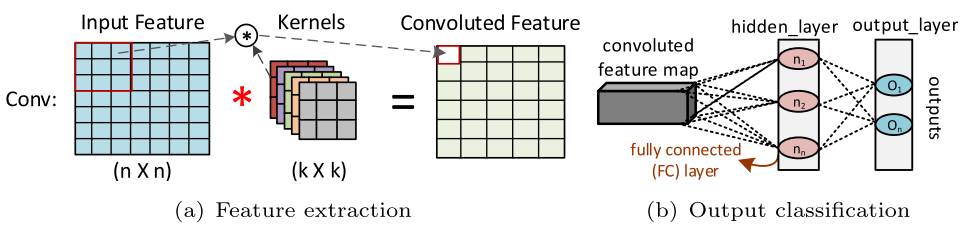
[2] Department of Electrical Engineering, Indian Institute of Technology Indore, Indore, India

Published online: 26 April 2024

have gained prominence for diverse applications, including object detection, pattern and character recognition, audio and video processing, language translation, trading, gaming, and cyber-security, as indicated by Sim et al. [1]. It's capability to map complex relationships within non-linear data bestows considerable advantages, setting it apart from other prediction techniques, as highlighted by Khalil et al. [2]. The DNN's Neuron engine plays a pivotal role in computation and accuracy, yet its hardware implementation is known for its demand for power and resources, as pointed out by Shawl et al. [3]. Optimizing the physical performance of the Neuron engine (NE) becomes crucial in addressing the heightened computational requirements of DNN. This engine adeptly handles operations like Multiply-Accumulate (MAC) and the execution of the non-linear transformation function, known as the activation function (AF). Furthermore, the MAC and AF operation are responsible for 90% of the computation in the neural network. Therefore, optimizing the computational unit architecture is essential to enhance DNN performance.

A typical DNN consists of two parts: feature extraction, as illustrated in Fig. 1a, and output classification, as depicted in Fig. 1b. In the course of this operation, input features undergo convolution with filters, requiring hundreds and thousands of parallel neuron processing engines to carry out these computations [4]. Consequently, there is a compelling need to optimize the Neuron engine responsible for executing the fundamental computations within DNN inference. Furthermore, the design of the NE, encompassing MAC and AF, involves the consideration of various design parameters. These parameters include arithmetic precision, data types, approximation in computation, data quantization, computation algorithms, and hardware implementation platforms, among others [5]. The hardware implementation platforms utilized for implementation encompass CPU, GPU, FPGAs, and ASICs, each with its respective advantages and drawbacks [6]. However, for edge-AI solutions, FPGA and ASIC-based implementations are preferred. Additionally, for power-efficient solutions, ASIC-based implementation is favored, although it lacks reconfigurability compared to FPGAs. To enhance computational efficiency and reduce architecture complexity, numerous investigations explore approximation in computation and data quantization [7]. However, these techniques often result in reduced accuracy. Hence, careful consideration of these advancements in the neuron engine becomes imperative. Scholars are presently exploring the potential of employing quantization in NE to enhance computational capacity while preserving model precision. Additionally, the use of application dependent different AFs within the same network is recommended [8]. Traditionally, this necessitates separate hardware for individual AFs and their configuration, leading to increased hardware resources and critical circuit delays.

In pursuit of reduced complexity in DNN hardware accelerators, a preference for lower arithmetic precision and integer or fixed-point data representation arises in both MAC and AF computations. During convolution, where a $k \times k$ kernel convolves with an input feature map (Fig. 1a), parallel multiplication and accumulation occur, leading to an output precision increase to 2N + M. Here, N represents input precision, and M signifies the overhead bits, dependent on the number of accumulations performed by the corresponding MAC unit. The MAC output is then provided to the AF, dictating the precision of the AF. Conventional AF implementations, such as those based on Look-Up Tables (LUTs) or Read-Only Memory (ROM), become hardware-costly for higher precision due to the increase in memory elements to 2^P, where P is the input bit precision of the AF (traditionally 2N + M). We can quantize the MAC output to N bits before applying it to the AF [8, 9]. However, this approach may result in accuracy loss, particularly when prioritizing higher accuracy for complex input features. Therefore, an efficient neuron must provide the option to select between quantized and unquantized MAC outputs. Moreover, if opting for unquantized data feed to the AF, conventional AF implementation becomes undesirable, as it is power and resource-intensive, especially for higher precision. Consequently, it becomes imperative to address a solution that accommodates both quantized and unquantized MAC outputs. To tackle this challenge, we introduce the Precision-aware Neuron Engine (PNE). The distinctive features of PNE and the primary contributions of this work are outlined below:



(a) Feature extraction (b) Output classification

Fig. 1 The convolution layer in DNN performs 2D matrix multiplication between the input feature map and kernel weights for feature extraction using MAC. FC layer in DNN performs 1D element-wise data computation. The ultimate classifying output layer is an FC layer

- A resource and power-efficient MAC architecture in PNE with a state machine design is presented, eliminating the multiplexer and utilizing pre-loaded bias for precision-aware computations, offering both quantized and unquantized outputs.
- We present an adaptable AF using ROM and Cordic, capable of producing tanh and sigmoid functions across varying bit-precisions, exhibiting minimal accuracy degradation and reduced LUT usage compared to tensor-based models.
- The PNE is designed using the proposed MAC and AF, achieving high accuracy at low-bit precision through the quantized MAC output and ROM AF output, as well as at high-bit precision using unquantized MAC output and Cordic AF output.
- The PNE's inference accuracy is assessed using Python emulation of LeNet and CaffeNet DNN models on FPGA hardware, comparing performance parameters, including resource, power utilization and delay, with state-of-the-art architectures.

This paper is an extension of our previous work presented at the IFIP-IoT conference [10] where an adaptable AF is presented. In this paper, we present a precision-aware neuron architecture using a precision-aware MAC and the adaptable AF.

Organization

This article is structured as follows: related research is presented in section "Related Research". The proposed PNE architecture, it's state machine design, and it's components i.e. the MAC and AF are discussed in section "Proposed PNE Architecture", followed by performance analysis and results discussions in section "Inference Accuracy and Hardware Performance: Evaluation and Analysis". Finally, the concluding remarks are given in section "Conclusions and Future Research".

Related Research

Various hardware accelerator architectures for neural networks have been introduced in the recent years [11]. Shallow neural networks are no longer useful, as the quantity of hardware neurons and connections in modern networks makes them outdated and unsuitable for the deep learning era. CNNs (a type of DNN) are frequently used in both video and image recognition systems, and usually employ a number of filters or convolution matrices [12]. As convolution matrices have fewer parameters than FC network layer weights, parallelism can be introduced. In order to reduce the network complexity, quantization in data representation is preferred for quantizing MAC output during inference, also rounding of weights and biases since they are fixed after training is finished [13]. In [14] and [15], a flexible, multi-precision per-layer data compression procedure is presented and implemented. Pruning aims to eliminate the subset of network units (i.e. weights or filters) which are least important for the network's intended task [16]. All of the above strategies necessitate the need for a programmable and precision-aware PNE which involves MAC computation followed by AF [13].

The MAC operation comprises of an adder, multiplier, and accumulator register. The multiplication result is transferred to an accumulator, added, and the output is stored in a register. The type of adders, multipliers, and registers lead to variations in area and delay [17]. Various articles have addressed MAC optimization by modifying the multiplication and addition techniques. Existing literature proposes different multiplication methods, such as vedic [18], array, wallace tree, booth [19], shift and add [20, 21], and modified booth [22, 23]. Researchers have also focused on optimizing the addition and quantized accumulation process using techniques such as approximation, quantization/ data resize, bits-serial, and reduced precision, as discussed in a study by Garland et al. [9]. Limited hardware resources make it challenging to implement MAC with parallel multipliers [24]. The conventional architecture of MAC (showing a single multiplier followed by accumulator along with input–output precision) is illustrated in Fig. 3.

The typical RTL view of the MAC architecture is depicted in Fig. 2, which includes a multiplier, two adders, a multiplexer, and an accumulator register [25]. The state-of-the-art revised architecture presented in Fig. 3 uses only one adder, one multiplexer, and register files, resulting in a saving of one adder compared to the typical architecture [26]. However, this design still uses a bias register file and a multiplexer, which result in more hardware resources and additional delay due to bias register loading time and delay due to the multiplexer. Therefore, the design can be further optimized to make it more efficient for DNN accelerator applications, in order to increase throughput. The conventional MAC design produces an output of 2N + M bits, where N is the input bit, and M is the overhead bits that come across the accumulation. The overflow bit depends on the number of accumulations and can be defined as 2^j, where j is the number of accumulations required in the MAC, which depends on the number of inputs.

The designs shown in Figs. 2, and 3, including our proposed design, utilize an arithmetic fixed-point $\langle N, q \rangle$ representation, which employs a binary point implication representation for the integer, signed, and fractional bits, as shown in Fig. 4. The representation comprises of N bits, comprising one sign bit, N-q integer bits, and q fractional bits. As overflow bits are necessary in the accumulation stage, the

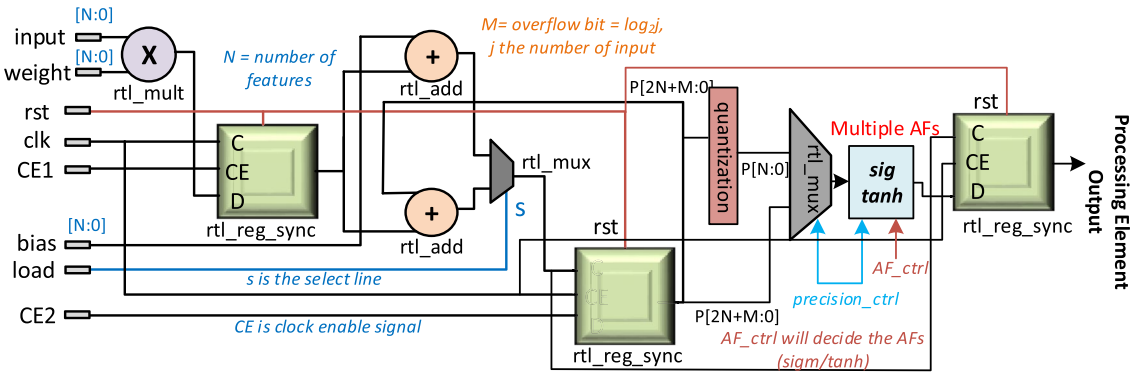


Fig. 2 Conventional RTL design for typical MAC with fixed-point representation. Here, N-bit precision is considered at the input with an output of 2N + M bits which includes overflow bits

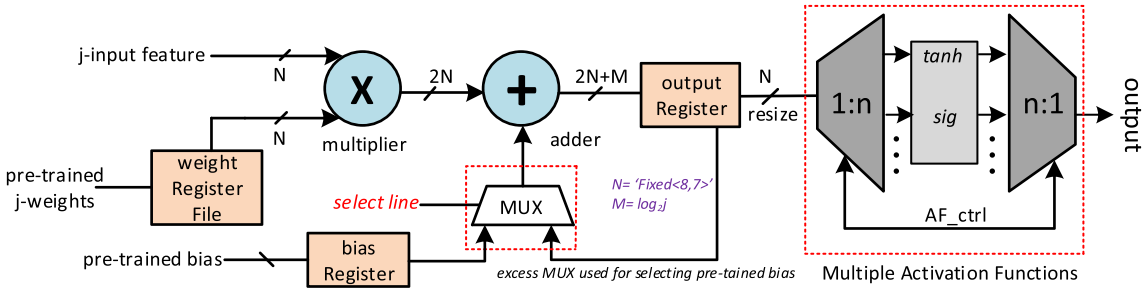


Fig. 3 Conventional MAC architecture with a single multiplier, an accumulator register with additional overhead bits, and a MUX for selecting the bias value

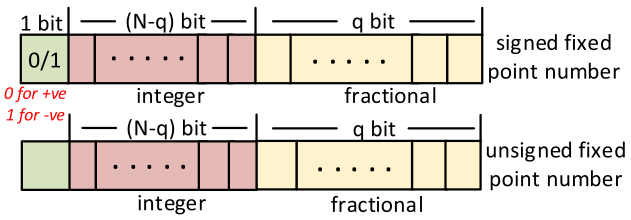


Fig. 4 The fixed-point arithmetic representation for an N-bit number, wherein q represents the fraction bit, (N - q) represents the integer bit, and MSB represents the sign bit

accumulator's bit size must be increased, and the overflow bit size is determined by the input size (i.e., the number of accumulations) in the corresponding neuron. In Fig. 3, the fixed-point N-bit numbers are depicted, as explained in Fig. 4, along with the logic elements and a 2:1 MUX for the N-bit data line. The MUX with a select line is employed to choose between pre-trained bias or accumulation process. The excessive hardware, i.e., MUX and bias register, shown in the dotted red box [1] Fig. 3, occupies additional hardware resources and increases delay. To address this issue, we optimized the design by efficiently pre-loading the bias value in the accumulator register, enabling resizing of the MAC output and handshaking with activation.

AF is the key to improving the network's learning capabilities in addition to correctly re-initializing the weights parameter. Sigmoid function is widely used for backpropagation training algorithms. It is crucial to select the right AF for machine learning training and inferencing. Type of AF can affect the convergence and accuracy of network training as well as increase the computational cost of training and inference phases [27]. Here arises the need for an adaptable AF. [28] presents a polynomial model for implementation of the fractional exponent part of tanh AF. These approaches achieve accurate approximations with minimal resource usage, but do not address configurability in AF. An energy-efficient DNN accelerator, with variable precision support, improved performance, and reduced energy consumption, is evaluated at the MAC level in [29], but the investigation of the AF is warranted for further enhancement. The Cordic method, originally introduced by Volder and later modified by Walther, performs circular, linear, and hyperbolic operations [30]. To address the issues related to additional resources and higher critical delays associated with Cordic, a resources reused Cordic-based architecture in [8] realizes sigmoid and tanh AFs using the same logic resources. This

approach has two main drawbacks: low accuracy and high LUT utilization for bit-precision ≤ 8. The adaptable AF presented in this paper combines the Cordic algorithm for high bit-precision AF and ROM for low bit-precision AF (≤ 8). For hardware implementation with the fixed-point notation, ROM-based AFs are not suitable for high-bit precision applications due to their significant resource utilization (i.e., LUT in FPGA and memory elements in ASIC). The LUT-based approach splits non-linear input ranges into regions and stores their data in LUTs as straight-line segments. FPGA-based customizable hardware designs for AFs have been proposed in [31], which are configurable, but consume more on-chip area compared to ASICs. FPGAs use BRAM to reduce computation overhead, but increased BRAM utilization trades memory usage for bit precision [26]. In [32], authors present a library of VLSI implementations for various AFs for hardware-efficient NN accelerators.

Proposed PNE Architecture

In this section, we examine the hardware architecture and state machine of the precision-aware neuron engine (PNE) proposed in this research. The design incorporates optimized MAC unit to address precision sensitivity. The PNE also includes an adaptable AF with ROM and Cordic-based implementation to improve memory efficiency in PNE processing while preserving versatile precision capabilities.

PNE and State Machine

In this section, we present the PNE architecture and its corresponding state machine. Comprising the MAC unit and the AF unit, the PNE, depicted in Fig. 5, illustrates the sequence of the MAC unit followed by the AF unit. In Fig. 6, the MAC unit within the PNE is designed to yield 2 outputs: quantized and unquantized. The MAC output serves as the input to a 2:1 MUX, which, contingent on the control signal (`precision_ctrl`), determines whether the quantized or unquantized output is to be provided as input to the AF. The (`precision_ctrl`) signal also determines the AF type based on the output produced by the MAC unit (i.e., quantized or unquantized).

The PNE has three inputs: input, weight, and bias. While the design is versatile enough to handle different bit-precision, we opt for a signed 8-bit arithmetic computation for performance evaluation and result extraction. The final output, identified as $\texttt{AF}_{out}$, is configured as an 8-bit value within the PNE. In the AF, the hardware incorporates a select pin (`precision_ctrl`) for AF processing. For lower precision, specifically quantized output, we have employed a ROM-based AF designed to support 8-bit and lower precision. However, when dealing with unquantized

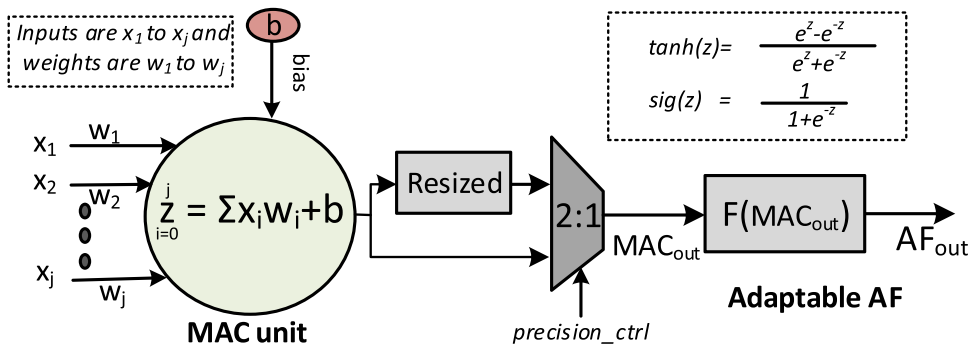


Fig. 5 PNE integrates MAC and AF. The MAC unit multiplies input features with trained weights, accumulates the result, and adds bias. The MAC output (`'z'`) is denoted as $\texttt{MAC}_{out}$. The AF is applied to $\texttt{MAC}_{out}$, resulting in the AF output ($\texttt{AF}_{out}$)

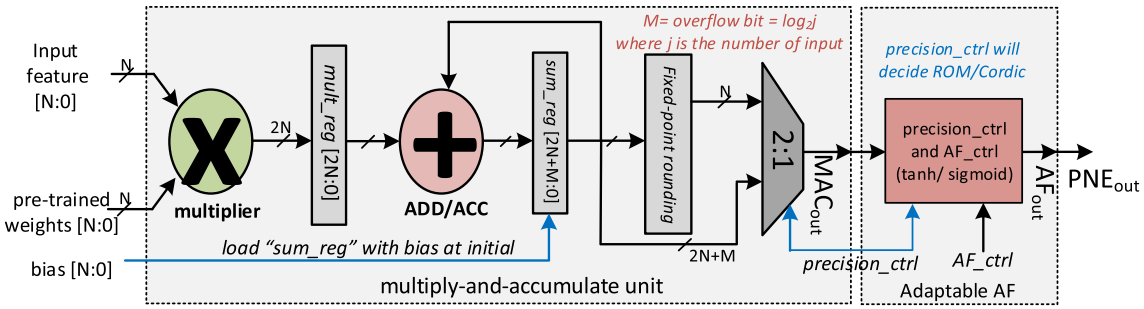


Fig. 6 PNE showcasing quantized and unquantized MAC outputs fed through 2:1 MUX to give $\texttt{MAC}_{out}$, followed by AF output: $\texttt{AF}_{out}$. $\texttt{PNE}_{out}$ is the same as $\texttt{AF}_{out}$

output featuring a higher bitwidth, a ROM-based approach might not be the most efficient choice for AF. Consequently, we introduce a novel AF function using iterative Cordic, ensuring support for higher precision computation with minimal hardware overhead compared to ROM-based implementations.

Figure 7 shows the state machine of the PNE. The state machine comprises of several states: idle, `idle`, `pre_MAC`, `MAC`, `post_MAC`, and `AF`. In the `idle` state, the machine sets various input and output signals to their initial values and waits for the `Computeinit` signal to begin processing. Upon receiving a `ComputeInit` signal, the machine transitions to the `initial` state, where input data is registered, and the initial sum value, including any bias values, is calculated. Specifically, `mult_reg` is set to 0, and `sum_reg` is initialized to the bias value (shown in Fig. 6). This state persists for a single clock cycle, after which it transitions to `pre_MAC`, initiating the first multiplication. In this state, the initial bias is loaded into the accumulator register, and the machine multiplies the input data with the appropriate weight value, adding the result to the sum (pre-loaded bias). The `index` flag is also set to the number of inputs for later use. The machine then progresses to the `MAC` state, where the multiplication and accumulation operations continue until the `index` flag reaches zero. The `index`, initialized during the `initial` state, is decremented at every clock cycle during the `MAC` state. In this state, both the multiplier and accumulator are enabled. Upon `index` reaching zero, the state transitions to `post_MAC`, where the final accumulation occurs, and the multiplier is disabled. The machine then moves to the `post_MAC` state. Here, final calculations are completed, producing two outputs: quantized and unquantized. One of these outputs is selected and passed through the MUX. Additionally, in this state, preparations are made for the AF to be applied, along with control signal `precision_ctrl`. The state then changes to `AF`, during which the output of the MAC unit (i.e., the output of the 2:1 MUX with `sum_reg`), quantized or unquantized depending on `precision_ctrl` as shown in Fig. 6, is applied to AF. Finally, the machine transitions to the `AF` state and operates based on the control signal `AF_ctrl`. In the `AF` state, the machine applies the activation function to the final sum value and sets a "done" flag to indicate that processing is complete. Here, the output of the proposed adaptable AF becomes the output of the PNE. The machine then returns to the idle state and waits for the next processing request. The detailed architecture for MAC and AF are discussed in subsections "Precision-Aware MAC Unit with Pre-Loaded Bias" and "Adaptable AF Using ROM/ Cordic". Table 1 shows the various modes of operation and outputs of the PNE.

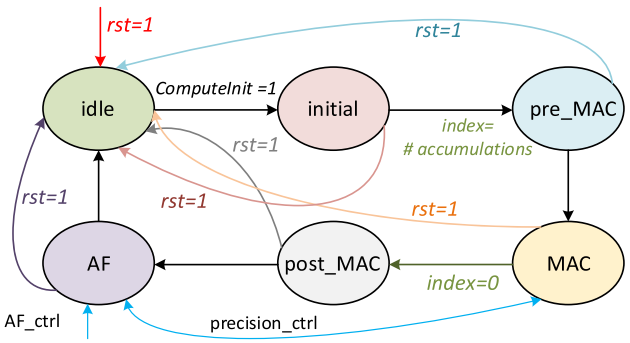


Fig. 7 State machine of the PNE showing `reset` signal and `idle` mode. The MAC computation happens during the two states: `pre_MAC` and `post_MAC`

Precision-Aware MAC Unit with Pre-Loaded Bias

The proposed design employs two inputs for the multiplication operation: the j-input feature and the pre-trained j-weight, both stored in the weight register file (shown in Fig. 6). The multiplication of these inputs yields a 2N-bit output(`mult_reg`), serving as input to the accumulator (`sum_reg`), where trained biases are pre-loaded. The adder takes the multiplier output and the feedback (accumulation) from the output register as inputs, storing the results in the output register. Iteratively accumulating the adder output, the output register produces a final unquantized output of `2N + M` bits, where M ($M = \log_2 j$) represents the overflow bit width resulting from iterative operations. The output generated at the accumulate stage is quantized into an N-bit value. Both the quantized and unquantized results are then passed to the non-linear transformation (AF) through a multiplexer based on the `precision_ctrl` signal, as depicted in Fig. 6 and shown in Table 1. Notably, the `MUX` and `bias_reg` present in the conventional design (Fig. 3)

Table 1 PNE_{out} selection using `AF_ctrl` and `precision_ctrl` signals as depicted in Fig. 6

AF_ctrl	Precision_ctrl	MAC_{out}	AF_{out}	PNE_{out}
0	0	Unquantized	Cordic[R_{in}] = sigmoid(R_{in})	sigmoid(R_{in})
1	0	Unquantized	Cordic[R_{in}] = tanh(R_{in})	tanh(R_{in})
X	1	Quantized	ROM[R_{in}]	sigmoid(R_{in}) or tanh(R_{in})

have been eliminated in the proposed design. The pre-trained bias output is now loaded directly into the accumulator register (`sum_reg`). This optimization reduces resource utilization and critical delay in the MAC operation. Also, the proposed design can accommodate any bit precision, enabling the specification of integer and fractional bits in the fixed ⟨ N, q ⟩ format representation before synthesis. However, for the purpose of design analysis and implementation, an 8-bit precision architecture has been employed. Although we have validated this proposed architecture on an FPGA using Hardware Description Language (HDL), the advantages demonstrated by this design are expected to extend to Application-Specific Integrated Circuits (ASICs) as well. In Fig. 6, the MAC architecture utilized in the PNE is emphasized with a dashed box.

In the first clock cycle, two signed fixed-point values, each with an N-bit width, are multiplied using a multiplier (`mult_reg`), and the bias value is pre-loaded into the `sum_reg`. This saves an extra clock cycle that is conventionally required for bias accumulation. The multiplication result is a 2 × N-bit value with 4 integer bits and 12 fractional bits stored in `mult_reg`. At each clock cycle, the value in `mult_reg` is accumulated in `sum_reg`. `sum_reg` is initialized with the bias value at the beginning of every layer computation, and extra bits are used to prevent overflow. Once all MAC operations are complete, the values in `sum_reg` are resized to a fixed '⟨ 8, 6 ⟩' format using the inbuilt IEEE library resize function. We have used the 'resize' function provided by *Xilinx* at the output of the MAC, which comes with rounding and inherent accuracy loss. Bit rounding can be defined as the procedure of changing a number with roughly the exact value but fewer digits with another number. This resized value is then fed into the sigmoid ROM for AF operation. In the fixed ⟨8, 6⟩' format, 2 integer bits and 6 fractional bits, along with 1 sign bit (MSB), represent the signed fixed-point representation (Fig. 4). The output of the AF is the output of the PNE for the current layer.

Adaptable AF Using ROM/ Cordic

The output generated by the MAC unit, determined by the `precision_ctrl` pin, is fed into the novel AF designed to accommodate both higher and lower precision arithmetic. In this study, we have utilized a ROM-based implementation for processing the quantized output of MAC (N-bits), and a Cordic-based implementation for processing the unquantized output (2N+M-bits). The AF design offers adaptability for selection of precision (via `precision_ctrl`) and the selection of AF (via `AF_ctrl`). The architecture supports both tanh and sigmoid AF computations. A detailed description of this performance-efficient adaptable AF's design architecture and computation techniques has been provided in [10]. We have integrated the aforementioned activation function (AF) into our proposed PNE, as depicted in red in Fig. 6. Within the PNE, the core of the adaptable AF is represented in Fig. 8. This core consists of a ROM or CORDIC Configure block and a processing block that facilitates adaptability for AF type selection. Two control signals, namely `precision_ctrl` and `AF_ctrl`, are employed for this purpose.

The ROM/ Cordic Configuration Block, shown in Fig. 8, integrates adders/ subtractors, shifters, and memory elements [10]. In the Cordic-based approach, the most significant bit (MSB) of R_{in}[N-1] (sign bit) generates the directional signal d_i [33], determining whether addition or subtraction is performed to converge R_{out} to 0. Here, $d_i \in \{0, 1\}$, representing the sign bit $R_{in}[N-1] \in \{0, 1\}$. In the ROM-based approach (Fig. 8), the value at the R_{in} address is accessed as ROM[R_{in}]. Depending on the ROM implementation and configuration of control pins, the AF's output for sigmoid or tanh is obtained. Thus, R_{out} = ROM[R_{in}] for the ROM-based approach, while R_{out} converges to 0 for the Cordic-based approach, as highlighted in the ROM/ Cordic Configuration Block in Fig. 8. The output of the Cordic block produces values cosh(R_{in}) and sinh(R_{in}) at P_{out} and Q_{out}, respectively. These outputs are used for exponential calculation, as described in Eq. 1.

$$e^{R_{in}} = cosh(R_{in}) + sinh(R_{in}) \tag{1}$$

The adaptable AF (Fig. 8) incorporates select signals `precision_ctrl` and `AF_ctrl`, which are summarized in Table 2 to determine outputs using either ROM or Cordic. The input data R_{in} serves as AF_{in} (or MAC_{out}) to the AF block and produces the output ROM[R_{in}] in the subsequent clock cycle or converges to 0 after the N^{th} Cordic iteration. One can observe that ROM/ Cordic Configuration Block provides three outputs: sinh(R_{in}), cosh(R_{in}), and 0/ ROM[R_{in}], as depicted in Fig. 8, with `AF_ctrl` controlling

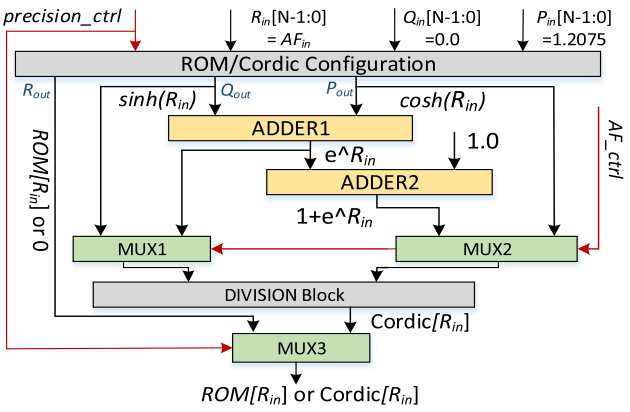


Fig. 8 The design of the adaptable AF for variable precision, consisting of the ROM/ Cordic Configuration Block and additional logic elements

MUX1 and MUX2 for Cordic-based sigmoid or tanh AF selection. sinh(R_{in}) and cosh(R_{in}) are sent to ADDER1. The output of ADDER1 is $e^{R_{in}}$ = sinh(R_{in}) + cosh(R_{in}), which serves as input to ADDER2. The output of ADDER2 is 1 + $e^{R_{in}}$. MUX1 has inputs $e^{R_{in}}$, sinh(R_{in}), and MUX2 has inputs cosh(R_{in}), 1 + $e^{R_{in}}$ with the select line as `AF_ctrl`. The outputs of MUX1 and MUX2 are processed in the divider to calculate Cordic[R_{in}] for sigmoid/ tanh evaluation. Subsequently, MUX3 is used to select ROM[R_{in}] or Cordic[R_{in}] based on the `precision_ctrl` signal. The state of control signals for generating tanh and sigmoid AFs using ROM/ Cordic approaches is presented in Table 2. Although ReLU is not implemented in this paper, Fig. 9 and Table 3 present how integration of ReLU AF is also possible in the proposed AF, showing it's adaptability. Comparing with Fig. 8, it can be observed that additional hardware requirements include 1 control signal (`AF_ctrl2`) and 2 MUXes (MUX4 and MUX5). MUX4 is used to implement the ReLU AF and MUX5 provides the selection (via the control signal `AF_ctrl2`) between ROM[R_{in}] and ReLU[R_{in}] output. The output of MUX5 is one of the inputs to MUX3, the other input being Cordic[R_{in}].

Inference Accuracy and Hardware Performance: Evaluation and Analysis

We have evaluated the inference accuracy of the proposed PNE for image classification tasks. Additionally, an analysis of resource utilization and delay of the PNE has been carried out with Cordic-based and ROM-based AF, targeting both quantized and unquantized results with precision requirements. The results demonstrate the effectiveness of our PNE, which has signed fixed-point pre-loaded bias MAC unit and an adaptable AF, enabling precision-aware DNN computations.

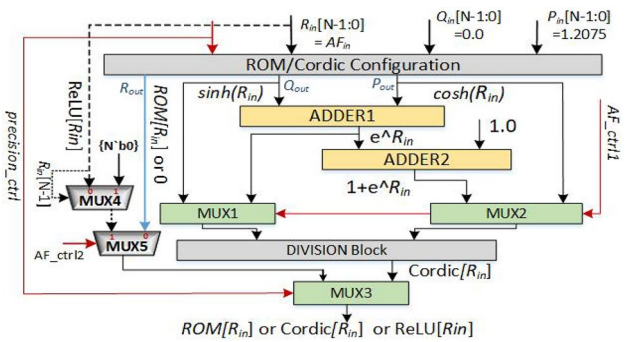


Fig. 9 ReLU AF integrated with our proposed adaptable AF design. The additional hardware requirements (MUX4, MUX5 and `AF_ctrl2`) are required to enable ReLU AF with our proposed adaptable AF

Table 2 AF selection using `AF_ctrl` and `precision_ctrl` signals for ROM/ Cordic AF as depicted in Fig. 8

AF_ctrl	precision_ctrl	R_{out}	AF_{out}
0	0	0	Cordic[R_{in}] = sigmoid(R_{in}) = $\frac{e^{R_{in}}}{1+e^{R_{in}}}$
1	0	0	Cordic[R_{in}] = tanh(R_{in}) = $\frac{sinh(R_{in})}{cosh(R_{in})}$
X	1	ROM[R_{in}]	ROM[R_{in}] = sigmoid(R_{in}) `OR` tanh(R_{in})

Experimental Validation of PNE: Quantized and Unquantized model

The experimental results presented in Table 4 provide a detailed analysis of the inferential accuracy of the PNE for both quantized and unquantized models. In this work, the precision comparison includes settings for 4-bit, 8-bit, and 16-bit across different datasets (MNIST, CIFAR-10, CIFAR-100) [34] and DNN architectures (LeNet [35], VGG-16 [36]). Further, it is to be noted that for target applications of the proposed PNE i.e. feature extraction and output classification, bit-precision upto 16-bits is optimal [37, 38]. Higher bit resolutions would lead to higher power consumption, higher critical delay and higher resource utilization, which are not desirable. The proposed PNE (P) is benchmarked against the Tensor-based MAC and AF model in the neuron engine (T) [39]. For the unquantized output, i.e., PNE with Cordic-based AF, the proposed PNE maintains competitive accuracy compared to the Tensor-based model, with a less than 2% accuracy loss compared to the TensorFlow model. The results underscore the robustness of the proposed PNE in preserving accuracy while operating under higher-precision settings. Notably, the proposed PNE showcases robust accuracy, with minimal differences ranging from 0.6% to 1.6%.

Furthermore, in the case of PNE with ROM-based AF (Quantized), the proposed PNE consistently demonstrates an insignificant accuracy loss of 1.6% compared to the Tensor-based model across different precisions and datasets. Even under lower bit-precision settings (4-bit, 8-bit, 16-bit), the proposed PNE exhibits strong performance, with accuracy difference ranging from 0.7% to 1.6%. This showcases the effectiveness of the PNE even in quantized scenarios with lower bit-precision. Overall, the proposed PNE exhibits robust performance across various precisions and DNN architectures, showcasing its effectiveness in maintaining accuracy under lower bit-precision settings. Notably, the comparison provides valuable insights into the performance of the proposed PNE across different bit precisions, highlighting its versatility and efficacy in diverse DNN architectures and datasets. It is crucial to emphasize that in evaluating accuracy, we employed a ROM-based approach for

Table 3 Table showing ReLU integration with proposed AF. 1 additional control signal (`AF_ctrl2`) and 2 MUXes are required as depicted in Fig. 9

AF_ctrl1	precision_ctrl	AF_ctrl2	R_{out}	AF_{out}
0	0	X	0	$\text{Cordic}[R_{in}] = \text{sigmoid}(R_{in}) = \frac{e^{R_{in}}}{1+e^{R_{in}}}$
1	0	X	0	$\text{Cordic}[R_{in}] = \tanh(R_{in}) = \frac{sinh(R_{in})}{cosh(R_{in})}$
X	1	0	$\text{ROM}[R_{in}]$	$\text{ROM}[R_{in}] = \text{sigmoid}(R_{in})$ OR $\tanh(R_{in})$
X	1	1	$\text{ReLU}[R_{in}]$	$\text{ReLU}[R_{\text{in}}] = \begin{cases} R_{\text{in}} & \text{if } R_{\text{in}} > 0 \\ 0 & \text{otherwise} \end{cases}$

Table 4 Comparative analysis of accuracy: PNE vs. Tensor-based neuron model having MAC and AF (T) [39] for LeNET and VGG-16 DNN models

DNN Arch.	LeNET				VGG-16			
Datasets	MNIST		CIFAR-10		CIFAR-10		CIFAR-100	
Precision	T	P	T	P	T	P	T	P
Infer. Accuracy (%) for Quantized PNE output(Quantized MAC + ROM-based AF)								
4-bit	96.1	95.6	54.8	54.2	64.6	62.8	23.8	22.7
8-bit	97.9	97.2	63.5	62.1	81.7	80.3	49.6	47.9
16-bit	98.6	97.9	64.3	62.8	82.8	81.1	51.3	49.8
Infer. Accuracy (%) for Unquantized PNE output (Unquantized MAC + Cordic-based AF)								
4-bit	96.8	88.3	55.4	48.6	65.1	52.7	24.1	22.8
8-bit	98.5	96.8	64.1	62.3	82.7	80.9	50.1	48.4
16-bit	99.1	97.9	65.2	64.3	86.1	84.8	55.3	54.1

both 8-bit and 16-bit precision. Additionally, for the Cordic-based algorithm, we conducted accuracy assessments using both 8 and 16-bit precision computations. However, for the hardware implementation, where resource constraints are a consideration, we adopted an 8-bit computation using ROM-based architecture. On the other hand, for the 16-bit computation, we utilized a Cordic-based architecture.

Hardware Implementation and Result Comparison of the Proposed PNE

This section delineates the experimental setup and methodology employed to evaluate the PNE proposed in this research. The assessment involves two distinct blocks, namely the Multiply-Accumulate (MAC) unit and the Activation Function (AF), implemented through the Zybo-Xilinx evaluation kit. Hardware performance parameters are extracted at an operating frequency of 50 MHz and an operating temperature of 25°C. The modular RTL design architecture is tailored for ASIC implementation. Simulation and performance parameters extraction consider 1 sign bit and 8 magnitude bits (2 integers and 6 fractional). Comparisons are drawn with conventional architecture, covering LUT and register utilization, critical delay, and total on-chip power. All architectures are implemented on the same FPGA platform for a fair comparison, detailed in [40]. Our proposed design, implemented on the Zybo-Xilinx FPGA SoC kit, has demonstrated noteworthy results.

Various state-of-the-art processing engines supporting definite precision were investigated, utilizing rounding or truncation between the MAC unit and AF, resulting in accuracy loss. In contrast, our proposed design provides developers the flexibility to choose quantization on the MAC result. Here, we have compared the proposed architecture with a conventional design. The proposed design supports both quantized and unquantized data. It is observed that if the same feature is incorporated into the conventional architecture, it would require 328 LUTs (102 + 226), whereas the proposed design utilizes only 248 LUTs. The specific advantages of ROM-based AF in lower precision and Cordic-based AF are discussed in section "Resource Utilization of the ROM/ Cordic-Based Adaptable AF". Our focus here is on architectural optimization and implementation. However, it can be extended for different types of MAC units and AFs, along with a state-of-the-art comparison. If higher accuracy is desired, the MAC output is processed directly through the AF. Conversely, if accuracy speed is not a significant concern, the MAC result is quantized using the `precision_ctrl` pin. It is crucial to note that quantized data configures the ROM-based AF, while unquantized data passes through Cordic-based AF. The detailed architectural design and computational arithmetic for the MAC unit and AF are elaborated in section "Precision-Aware MAC Unit with Pre-Loaded Bias" and "Adaptable AF Using ROM/ Cordic", respectively. The hardware implementation results reported for the Zybo FPGA board are presented below.

The comprehensive analysis of proposed PNE in comparison to conventional counterparts on a Zybo board at 50MHz is presented in the Table 5. This table outlines the standalone architecture supporting both unquantized and quantized MAC operations in the conventional design. Additionally, it presents the proposed precision-aware design, offering the flexibility for both operations. Notably, the proposed design can be configured through an external signal, enabling PNE and supporting adaptable AF. Precision-aware configurations, specifically the proposed precision aware pre-loaded bias MAC and adaptable Cordic AF, exhibit minor resource overhead, enabling the precision-aware design to evolve from 226 LUTs to 248 LUTs, constituting nearly a 9.7% overhead. In the conventional MAC design with only Cordic-based AF, the critical delay increases by 34.37% compared to the proposed PNE. This is due to the additional hardware resources, namely bias registers and MUX selection, utilized for bias loading. The incorporation of these elements introduces an extra clock delay, consequently reducing the throughput of the system. However, the proposed designs showcase higher throughput, with the Precision-aware pre-loaded bias MAC and adaptable Cordic AF achieving 34.61% more GOp/s than the conventional MAC with only Cordic AF. The performance metric underscores the superiority of precision-aware designs, indicating a more substantial performance improvement over the conventional MAC with only Cordic and also enables the design with quantization-enabled computation. These findings highlight the efficiency gains achievable through the proposed pre-loaded bias MAC architecture and an adaptable AF integrated into the PNE.

Optimized MAC Unit with Quantization-Enabled Output Selection and Pre-Loaded Bias

In this section, we conduct a comparative analysis of the MAC unit, which generates two distinct outputs. The first output is the quantize-enabled output, wherein the MAC unit's output is quantized from 2N + M bits to N bits through LSB bit truncation. This advancement enables the utilization of an N-bit Activation Function (AF) for subsequent computations. Additionally, the MAC unit produces a second output, namely the unquantized output, with a precision of 2N + M bits, where 'M' represents the extra overhead bits addressed earlier. The selection between these outputs is determined using a Multiplexer (MUX) based on the desired level of accuracy required for the computation. We conduct a comprehensive comparison of state-of-the-art Multiply-Accumulate (MAC) architectures, all tailored for accurate computations at 8-bit precision. We present a detailed analysis of resource utilization, power consumption, and delay for the proposed design and other existing architectures on the Zybo SoC Xilinx FPGA board as summarized in Table 6. Our proposed design demonstrates resource utilization comparable to the IEEE standard, even with the integration of the output selection advancement. Notably, our optimized architecture eliminates the need for bias registers and MUX used for bias selection systematically, resulting in no additional hardware resource overhead. This enhancement significantly improves critical computation delay and reduces a clock delay by pre-loading the bias value into the accumulator register. This also leads to reduction in power consumption. Although the introduction of an extra Multiplexer (MUX) at the MAC output incurs minimal hardware overhead, this is effectively compensated by the aforementioned advancements. Table 6 provides a comparison with architectures like Booth Multiplication, Wallace Tree Vedic, shift-and-add, Cordic, and IEEE. While each design exhibits its own merits and drawbacks, the choice depends on the specific application requirements, whether prioritizing hardware efficiency, accuracy, or performance. Our proposed design offers support for both features, aligning with diverse application needs.

The proposed MAC unit has been compared with established state-of-the-art designs, including Vedic, Wallace, Booth, Shift-Add, Cordic, and IEEE architectures, as detailed in Table 6. The reported parameters include slice LUTs and slice registers. The analysis reveals that the resources utilized by the proposed MAC design have utilized comparable hardware to state-of-the-art architectures. While

Table 5 Comparison of proposed PNE (Quantized/ Unquantized MAC + adaptable AF) for different bit-widths with conventional processing engine (conventional MAC + AF) evaluated on Zybo-board at 50MHz operating frequency

Processing Engine	Quantized MAC + AF in ROM mode	Unquantized MAC + AF in Cordic mode	Conv. MAC [41]+ ROM AF	Conv. MAC [41] + Cordic AF
Preferred AF Precision	8 bit	16 bit	8 bit	16 bit
Slice LUT	248	248	102	226
Slice Reg. (FF)	88	128	114	153
Critical Delay (ns)	3.81	4.19	4.82	5.20
Dynamic Power (mW)	3.95	4.83	4.99	6.49
Throughput (GOp/s)	0.2629	0.2073	0.2004	0.1540
Performance (MOPS/W)	66.7	42.9	40.2	23.7

Table 6 Performance evaluation: comparison of the precision-aware MAC with state-of-the-art models

Design techniques	Slise LUTs	Slice Reg	Critical Delay (ns)	On-chip static Power(μW)	Dynamic Power (mW)	Power-Delay Product (pJ)
Vedic [18]	159	245	4.48	484.8	4.93	5.86
Shift-Add [21]	75	58	5.44	119.5	2.86	3.97
Iterative Cordic [8]	23	22	9.06	139.2	4.72	–
IEEE DSP [41]	130	49	3.98	495.5	3.81	5.01
Wallace [22]	105	112	2.59	498.0	3.88	3.13
Booth [19]	83	61	3.08	146.7	6.56	2.77
Proposed	92	61	2.67	92.0	2.15	2.08

slightly exceeding the resource utilization of shift-and-add and Booth algorithm-based MAC units, our design has demonstrated superior performance in critical delay. This improvement has occurred because of the pre-loaded bias, which reduces MUX delay in the critical path and eliminates the need for an additional clock delay traditionally required to accumulate the bias value in the multiplication of input and weights, as discussed in section "Precision-Aware MAC Unit with Pre-Loaded Bias", and shown in Figs. 6, 7. To highlight the advancements of the proposed design, a comparison has been made with Vedic and IEEE DSP package MAC architectures. Specifically, the Vedic-based architecture [18] has shown to have 159 Slice LUTs, whereas our proposed design has had only 92 slice LUTs, indicating a 42.13% reduction compared to the Vedic architecture. Similarly, our design has consumed 29.23% fewer slice LUTs compared to the IEEE architecture. When compared with Wallace's architecture, our proposed design has shown a 12.38% reduction. Additionally, the proposed architecture has utilized 61 slice registers, marking a 45.53% reduction compared to Wallace's architecture, which has had 112 slice registers. In summary, the analysis suggests that the proposed design is suitable for adoption in applications where accuracy demands are application-specific, and there is requirement for reduced resource usage and critical delay.

The critical delay represents the maximum delay within a circuit, attributed to the longest combinational path. As a crucial performance metric, it dictates the circuit's maximum operating frequency. The critical delay (in nanoseconds) is presented for both the proposed and state-of-the-art architectures with 8-bit precision in Table 6. Notably, the Cordic architecture exhibits a maximum critical delay of 9.06 ns due to its iterative computation (n-iteration, i.e., n times the critical delay of each iteration), despite its superior resource utilization. In comparison, the critical delay for the IEEE standard DSP architecture [41] is lower at 3.98 ns. Conversely, our proposed design boasts of a critical delay of 2.67 ns, signifying a 32.91% improvement over the IEEE standard. Our proposed design's critical delay (calculated using Vivado) stands at 2.67 ns, the lowest among all of the architectures in literature, attributed to pre-bias loading and the elimination of multiplexing used for bias selection in conventional MAC designs.

Additionally, the Power-delay Product (PDP) is computed for 8-bit precision across all methods and detailed in Table 6. Notably, the booth-multiplier-based approach closely mirrors our proposed design's PDP at 2.77pJ compared to 2.08pJ. Our proposed design demonstrates a 24.91% higher efficiency than the booth technique [19] in terms of PDP. A comprehensive comparison in Table 6 reveals that our proposed design outperforms all techniques in the literature in terms of PDP efficiency.

Time Slack

It's important to note that the PNE simulations have been carried out at 50MHz, while the evaluation of the standalone MAC Unit slack calculation has been done at 100MHz. This difference in frequencies helps us assess the effectiveness of the design under optimum operating conditions. The slack calculation for various bit precisions has been reported in Table 7. The table includes three types of slack values: Worst Negative Slack (WNS), Worst Hold Slack (WHS), and Worst Pulse Width Slack (WPWS) for three different bit precision values: 8-Bit, 12-Bit, and 16-Bit. The equations have been presented in Eq. 2, where (T_r) and (T_a) denote the earliest start time and actual start time of a task, respectively. T_{setup} and T_{slack} refer to the time available before the earliest start time and the time available after the actual start time, respectively. The slack information provides insights into whether the design is functioning at the desired frequency, as explained in [42].

Table 7 Slack calculation at 100MHz frequency for different bit precision for optimum operating frequency

Slack Type	8-Bit	12-Bit	16-Bit
WNS (ns)	2.356	4.898	5.136
WHS (ns)	0.068	0.104	0.236
WPWS (ns)	49.500	49.500	49.500

$$\mathbf{T_{setup} = T_r - T_a} \tag{2a}$$

$$\mathbf{T_{slack} = T_a - T_r} \tag{2b}$$

$$\mathbf{T_a = T_{total} + T_{rc} + T_{cq}} \tag{2c}$$

$$\mathbf{T_a \geq T_{hold}} \tag{2d}$$

$$\mathbf{T_{slack,setup} = T_{cycle} - T_a - T_{setup}} \tag{2e}$$

$$\mathbf{T_{slack,hold} = T_a - T_{hold}} \tag{2f}$$

As the bit precision increases, WNS and WHS also increase, indicating a decrease in the timing margin. This outcome is expected because higher bit precision means more complex circuits, which results in longer propagation delays and lower timing margins. However, the WPWS values remain constant across all three bit precisions. This is because WPWS is a measure of the minimum required pulse width for the circuit to function correctly and is determined by the slowest path in the circuit. Since the slowest path does not change with bit precision, the WPWS value remains constant. Overall, the table suggests that a higher bit precision is associated with a lower timing margin, which could potentially lead to timing violations and reduced performance. However, the constant WPWS values indicate that the minimum pulse width requirement does not change with bit precision.

Resource Utilization of the ROM/ Cordic-Based Adaptable AF

In the hardware-based evaluation, resource utilization is assessed by implementing the adaptable AF using Verilog-HDL, and the corresponding parameters are extracted using the *Vivado-Xilinx* tool. The proposed design is implemented on the Zybo Evaluation Kit, with a specific focus on the sigmoid AF, which effectively utilizes all the hardware resources within the configurable architecture. The AF exhibits nonlinear behavior in artificial neurons, and its hardware implementation is particularly costly in conventional approaches. Increasing arithmetic precision leads to a rise in design complexity, as achieving higher precision necessitates exponential growth in computational complexity or memory elements. For instance, an n-bit precision requirement in ROM demands 2^n memory elements. To illustrate, 8-bit precision requires 256 elements, while 16-bit precision would need 65,536 elements, making it impractical given the dedicated AF for each neuron. Contemporary architectures aim for efficient design and implementation. However, each design favors either lower or higher bit precision and adaptability in AF types of selection. Addressing these challenges, our proposed solution involves adaptable logic for selecting types of AF and precision settings for computations.

Table 8 compares resource utilisation for ROM, Cordic, and BRAM-based techniques for various bit-precisions. The ROM-based design uses 6 LUTs for 4-bit accuracy, whereas the Cordic-based design uses 45 LUTs and 37 flip-flops (FFs), resulting in an 86.66% LUT savings for the ROM-based method. For 8-bit precision, the ROM-based solution uses 16 LUTs, compared to Cordic's 84 LUTs and 72 FFs, resulting in an 80.95% LUT savings. The ROM-based system depends just on LUTs, with no FFs required. However, as precision increases to 16-bit, the ROM-based design requires a substantial 2111 LUTs, as opposed to Cordic's 140 LUTs and 126 FFs. Thus, implementing a 32-bit ROM-based design on smaller FPGAs would be inefficient due to the exponential increase in resource requirements. We also report results for a BRAM-based approach, which shows significant rise in BRAM utilization as precision increases. Specifically, for 4, 8, and 16-bit precisions, BRAM requirements are 0.5, 0.5, and 17 BRAMs, respectively. Overall, the Cordic-based technique demonstrates better LUT utilization for higher precision computations. The ROM-based implementation of AFs exhibits superior performance at lower precision. These findings provide valuable insights for selecting appropriate AF implementations based on precision requirements and resource constraints in the PNE.

Conclusions and Future Research

In summary, our research introduces a precision-aware Neuron Processing Engine (PNE) for efficient deep neural network (DNN) computations. The PNE features a signed fixed-point pre-loaded bias Multiply-Accumulate (MAC) unit and an adaptable Activation Function (AF) supporting both ROM and Cordic implementations. Evaluating inference accuracy across quantized and unquantized models, various bit precisions, and datasets, our experimental results highlight the PNE's effectiveness. Under unquantized scenarios, the Cordic-based AF

Table 8 Resource Utilization of adaptable AF for different bit-widths evaluated on Zybo-board

AF Type	ROM		Cordic		BRAM
Precision	LUTs	FFs	LUTs	FFs	–
4-bit	6	0	45	37	0.5
8-bit	16	0	84	72	0.5
16-bit	2111	0	140	126	17

exhibits robust accuracy with less than a 2% loss compared to TensorFlow models. Even in quantized scenarios (4-bit, 8-bit, 16-bit), the PNE performs strongly with accuracy differences ranging from 0.7% to 1.6%, showcasing its versatility. Hardware implementation on the Zybo-Xilinx FPGA platform demonstrates notable results, with resource utilization comparable to state-of-the-art models and superior critical delay and throughput. The precision-aware MAC unit allows developers to choose between quantized and unquantized operations, offering flexibility in balancing accuracy and speed. The comprehensive analysis of resource utilization, power consumption, and critical delay underscores the efficiency gains achievable through our proposed architecture. The adaptable AF, implemented using ROM and Cordic, caters to diverse precision requirements. Currently, the AF supports sigmoid and tanh implementations. However, it can be adapted to implement ReLU and Gaussian AF as well with additional hardware requirements, as part of future research. In conclusion, our precision-aware Neuron Processing Engine provides a holistic solution for efficient DNN computations, contributing valuable insights to hardware-efficient neural network accelerators and advancing precision-aware computing architectures.

Acknowledgements This article is an extended version of our previous conference paper presented at [10].

Data Availability Data sharing is not applicable to this article as no data sets were generated or analyzed during the current study, and detailed circuit simulation results are given in the manuscript.

Declarations

Conflict of interest The authors declare that they have no Conflict of interest and there was no human or animal testing or participation involved in this research. All data were obtained from public domain sources.

References

1. Sim H, Lee J. Cost-Effective Stochastic MAC circuits for Deep Neural Networks. Neural Netw. 2019;117:152–62.
2. Khalil K, Eldash O, Kumar A, Bayoumi M. An efficient approach for neural network architecture. In: 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018;745–748. IEEE
3. Shawl MS, Singh A, Gaur N, Bathla S, Mehra A. Implementation of Area and Power Efficient Components of a MAC unit for DSP Processors. In: 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018;1155–1159. IEEE.
4. Machupalli R, Hossain M, Mandal M. Review of ASIC Accelerators for Deep Neural Network. Microprocess Microsyst. 2022;89:104441.
5. Merenda M, Porcaro C, Iero D. Edge machine learning for ai-enabled iot devices: A review. Sensors. 2020;20(9):2533.
6. Shantharama P, Thyagaturu AS, Reisslein M. Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies. IEEE Access. 2020;8:132021–85.
7. Hashemi S, Anthony N, Tann H, Bahar RI, Reda S. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, 2017;1474–1479. IEEE.
8. Raut G, Rai S, Vishvakarma SK, Kumar A. RECON: Resource-Efficient CORDIC-based Neuron Architecture. IEEE Open Journal of Circuits and Systems. 2021;2:170–81.
9. Garland J, Gregg D. Low Complexity Multiply-Accumulate Units for Convolutional Neural Networks with Weight-Sharing. ACM Transactions on Architecture and Code Optimization (TACO). 2018;15(3):1–24.
10. Vishwakarma S, Raut G, Dhakad NS, Vishvakarma SK, Ghai D. A Configurable Activation Function for Variable Bit-Precision DNN Hardware Accelerators. In: IFIP International Internet of Things Conference, 2023;433–441. Springer.
11. Posewsky T, Ziener D. Efficient deep neural network acceleration through fpga-based batch processing. In: 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2016;1–8. IEEE.
12. Schmidhuber J. Deep Learning in Neural Networks: An overview. Neural Netw. 2015;61:85–117.
13. Jelčicová Z, Mardari A, Andersson O, Kasapaki E, Sparsø J. A neural network engine for resource constrained embedded systems. In: 2020 54th Asilomar Conference on Signals, Systems, and Computers, 2020;125–131. IEEE
14. Qiu J, Wang J, Yao S, Guo K, Li B, Zhou E, Yu J, Tang T, Xu N, Song S, et al. Going deeper with embedded fpga platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-programmable Gate Arrays, 2016;26–35.
15. Zhang Y, Suda N, Lai L, Chandra V. Hello edge: Keyword spotting on microcontrollers. arXiv preprint arXiv:1711.07128 2017.
16. Cheng Y, Wang D, Zhou P, Zhang T. Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges. IEEE Signal Process Mag. 2018;35(1):126–36.
17. Masadeh M, Hasan O, Tahar S. Input-Conscious Approximate Multiply-Accumulate (MAC) Unit for Energy-Efficiency. IEEE Access. 2019;7:147129–42.
18. Krishna AV, Deepthi S, Nirmala Devi M. Design of 32-Bit MAC unit using Vedic Multiplier and XOR Logic. In: Proceedings of International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications, 2021;715–723. Springer.
19. Farrukh FUD, Zhang C, Jiang Y, Zhang Z, Wang Z, Wang Z, Jiang H. Power Efficient Tiny Yolo CNN using Reduced Hardware Resources based on Booth Multiplier and Wallace Tree Adders. IEEE Open Journal of Circuits and Systems. 2020;1:76–87.
20. Johansson K. Low power and Low Complexity Shift-and-Add based Computations. PhD thesis, Linköping University Electronic Press 2008.
21. Gudovskiy DA, Rigazio L. Shiftcnn: Generalized Low-Precision Architecture for inference of Convolutional Neural Networks. arXiv preprint arXiv:1706.02393 2017.
22. Janveja M, Niranjan V. High performance Wallace tree multiplier using improved adder. ICTACT j microelectron. 2017;3(01):370–4.
23. Yuvaraj M, Kailath BJ, Bhaskhar N. Design of optimized MAC unit using integrated vedic multiplier. In: 2017 International Conference on Microelectronic Devices, Circuits and Systems (ICMDCS), 2017;1–6. IEEE.
24. Sze V, Chen Y-H, Yang T-J, Emer JS. Efficient processing of deep neural networks: A tutorial and survey. Proc IEEE. 2017;105(12):2295–329.

25. Sharma VP, Vishwakarma SK. Analysis and Implementation of MAC Unit for different Precisions. signal (μW) 70(120):240
26. Raut G, Biasizzo A, Dhakad N, Gupta N, Papa G, Vishvakarma SK. Data Multiplexed and Hardware Reused Architecture for Deep Neural Network Accelerator. Neurocomputing. 2022;486:147–59.
27. Wuraola A, Patel N, Nguang SK. Efficient activation functions for embedded inference engines. Neurocomputing. 2021;442:73–88.
28. Aggarwal S, Meher PK, Khare K. Concept, design, and implementation of reconfigurable CORDIC. IEEE Trans Very Large Scale Integr VLSI Syst. 2015;24(4):1588–92.
29. Lee J, et al. Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision. IEEE J Solid-State Circuits. 2018;54(1):173–85.
30. Lin C-H, Wu A-Y. Mixed-scaling-rotation CORDIC (MSR-CORDIC) algorithm and architecture for high-performance vector rotational DSP applications. IEEE Trans Circuits Syst I Regul Pap. 2005;52(11):2385–96.
31. Mohamed SM, et al. FPGA implementation of reconfigurable CORDIC algorithm and a memristive chaotic system with transcendental nonlinearities. IEEE Trans Circuits Syst I Regul Pap. 2022;69(7):2885–92.
32. Prashanth H, Rao M. SOMALib: Library of Exact and Approximate Activation Functions for Hardware-efficient Neural Network Accelerators. In: 2022 IEEE 40th International Conference on Computer Design (ICCD), 2022;746–753. IEEE.
33. Mehra S, Raut G, Das R, Vishvakarma SK, Biasizzo A. An Empirical Evaluation of Enhanced Performance Softmax Function in Deep Learning. IEEE Access 2023.
34. Alex K. Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf 2009.
35. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE. 1998;86(11):2278–324.
36. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 2014.
37. Park J-S, Park C, Kwon S, Kim H-S, Jeon T, Kang Y, Lee H, Lee D, Kim J, Lee Y, Park S, Jang J-W, Ha S, Kim M, Bang J, Lim SH, Kang I. A Multi-Mode 8K-MAC HW-Utilization-Aware Neural Processing Unit with a Unified Multi-Precision Datapath in 4nm Flagship Mobile SoC. In: 2022 IEEE International Solid-State Circuits Conference (ISSCC), 2022;65:246–248.
38. Chang J-K, Lee H, Choi C-S. A Power-Aware Variable-Precision Multiply-Acumulate Unit. In: 2009 9th International Symposium on Communications and Information Technology, 2009;1336–1339.
39. Abadi M, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org 2015.
40. Raut G, Mukala J, Sharma V, Vishvakarma SK. Designing a Performance-Centric MAC Unit with Pipelined Architecture for DNN Accelerators. Circuits, Systems, and Signal Processing, 2023;1–27.
41. Multiplier v12.0 LogiCORE IP Product Guide. https://www.xilinx.com/support/documentation/ipdocumentation/multgen/v120/pg108-mult-gen.pdf
42. Venkataramani G, Goldstein SC. Slack Analysis in the System Design Loop. In: Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2008;231–236.