# A SIMD Dynamic Fixed Point Processing Engine for DNN Accelerators

***Abstract*—For DNN accelerators, we introduce a novel convolution layer Processing Engine (PE) that combine multi-precision dynamic fraction fixed-point Multiply-Accumulate (MAC) and Activation Function (AF) units (8/16 bits with shared resources). This PE, functions as a Single Instruction, Multiple Data (SIMD) engine, facilitates switching the precision dynamically for 8-bit or 16-bit computations, supporting multi-precision (N=8 or 16) operations with dynamic fraction fixed-point data. Pareto analysis was performed to optimize the resources for determining the ideal number of parallel Cordic stages needed for multi-precision dynamic fraction fixed-point AF, a SIMD Engine. Our proposed approach demonstrates minimal accuracy loss, with less than 1% for LeNet with MNIST, less than 1% for AlexNet with CIFAR-10, and less than 2% for VGG16 with CIFAR-10, as compared to reference float32-based implementations. Experimental results using the Virtex-VCU118 Evaluation kit highlight that design of SIMD multi-precision dynamic fraction MAC and CORDIC AFs units with shared resources exhibits significant improvements in resource efficiency as compared to non-shared resources. Specifically, we observed a 51.83% reduction in LUTs for MAC units and a 43.50% decrease in LUTs for AFs as compared to a SIMD engine based MAC + AF without shared resources realized with separate MAC and AFs for 8 and 16 bits. Thereby, this multi-precision design with shared resources for 4x8 and 1x16-bit processing optimizes resource utilization, enhances versatility and throughput, enables efficient computations across diverse DNN applications, models, and layers with a unique SIMD-enabled PE.**

***Index Terms*—DNN, multi-precision, MAC, AF, dynamic fraction fixed-point, CORDIC, SIMD**

## I. Introduction

Deep Neural Networks (DNNs) have revolutionized various sectors, driving advances in image classification, audio recognition, and natural language processing [1]. Hardware architectures for DNN are instrumental in achieving real-time, energy-efficient inference at edge devices with the challenges posed by dynamic and wide-ranging neural network parameters like varying data representation, range and precision. Efficiently designing Single Instruction Multiple Data (SIMD) based dynamic fraction fixed-point processing engines (PEs) with shared hardware, encompassing multiply-accumulate (MAC) units and activation functions (AFs), and adapting them to diverse DNN models and layers, is a complex endeavor [2]. The implementation of dynamic fraction fixed-point data with multi-precision arithmetic (SIMD Engine) is particularly critical for non-linear AFs to allocate integer and fractional bit widths effectively, ensuring the requisite dynamic range and precision for accurate computations across various DNN layers [3] are fulfilled. In this research article, we adopt the term "SIMD dynamic fraction fixed-point" to refer "multi-precision dynamic/adaptive fraction fixed-point". This approach enhances DNN hardware performance by accommodating various operand data bitwidths and facilitating dynamic fraction fixed-point computations (MAC and AFs) within PEs. This approach effectively handles the varying operand bitwidths and dynamic fraction fixed-point requirements across different DNN layers and models [4].

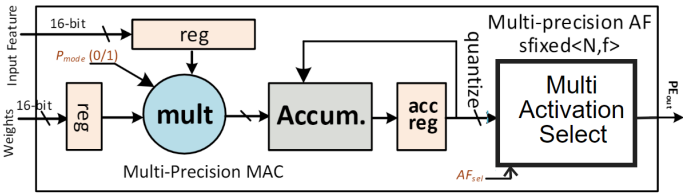


Fig. 1. Conventional PE with Multi-Precision Arithmetic (SIMD), Separate 8-bit and 16-bit MACs and AFs (separate tanh, sigmoid) Units

DNNs encompass a diverse range of layers, which include convolutional, fully connected, pooling, LSTM, GRU, normalization, and activation layers, such as $relu$, $tanh$, and $sigmoid$ [5]. In this context, the fundamental computational components of a DNN model are the PEs. The demand is for the PE to be equipped with multi-precision arithmetic, typically enabling 8 and 16-bit operand computations and supporting both MAC operations and AFs. Figure 1 illustrates a traditional PE with Multi-Precision Arithmetic featuring distinct 8-bit and 16-bit MAC and AFs. Such designs may also offer the flexibility to employ separate $tanh$ and $sigmoid$ units [6]. In a conventional setup, PEs cooperate to form filter banks for feature extraction within DNNs. These filter banks demonstrate varying dynamic ranges and susceptibility to quantization errors [7]. To address this need, a unified neural processing unit (UNPU) has been introduced in [8]. The UNPU is an energy-efficient DNN accelerator, accommodating fully variable weight bit precision and diverse layer types, viz. convolutional, recurrent, and fully connected layers. The article demonstrates and compares balancing energy efficiency and peak performance with ImageNet DNN (VGG-16). However, the mentioned article does not detail the multi-precision computations for AFs.

In the realm of DNN accelerators, PEs form the core, combining MAC units with AFs like sigmoid, tanh, and ReLU [9]. Recent studies stress dynamic fraction and multi-precision (8/16-bit) computing [10], [11]. Achieving efficient multi-precision support by sharing resources for MAC units and various AFs is a complex task. To tackle this, we propose a versatile SIMD-enabled PE for multi-precision and dynamic fraction fixed-point computations. It processes 8 or 16-bit operands using shared resources, embracing dynamic fraction fixed-point (*sfixed*<*N,f*>) representation. We focus on designing multi-precision SIMD MAC units and CORDIC-based AFs, offering operand support based on the chosen input mode and data width '`N`.' We aim to improve hardware efficiency by incorporating multi-precision computation,

allowing precision selection during runtime based on input data ranges for enhanced inference accuracy and optimized hardware performance. The key outcomes of the research include:

- A resources efficient MAC unit for multi-precision (8/16 bits) adaptive fraction fixed-point computations.
- By adopting CORDIC-based techniques and shared resources, our design enables unified, modular, and compact multi-precision adaptive fraction fixed-point computations for $sigmoid$ and $tanh$ AFs.
- SIMD PEs that are precision-aware and support adaptive fraction fixed-point *sfixed<N,f>* representation. This design empowers multi-precision capabilities, configurability, and efficient computations within DNN accelerators.

The paper's structure is as follows: Section II provides background, motivation, and related works. In Section III, we introduce our novel approach to SIMD-enabled multi-precision adaptive fraction fixed-point PEs, which ensemble both MAC and various AFs units, with a detailed illustration of their designs. Section IV presents the experimental setup, inference accuracy evaluation, hardware resources, performance metrics, and analysis. Finally, Section V concludes the paper and suggests future research directions.

## II. Background and Motivation

Utilizing the `sfixed<N,f>` notation (multi-precision: N=8/16) with a dynamic fractional bit (f) is crucial for range, precision and accuracy in both ASIC and FPGA implementations [11], [12]. Employing multi-bit precision in MAC units and AFs within PEs collectively benefits DNNs by improving efficiency and resource allocation in a fixed-point multi-precision design. Achieving efficient DNN accelerators requires hardware optimization with multi-precision dynamic fraction fixed-point architectural units [6], [13]. An FPGA DSP block architecture with SIMD support for MAC operations customized for multi-precision deep neural networks is addressed in [14], but SIMD nonlinear transformation functions (AFs) have not been explored. Therefore, further research is needed to investigate multi-precision dynamic fraction fixed-point AFs design and enhance PE adaptability across diverse neural network architectures and datasets [15]. A versatile Cordic-based architecture for AFs has been introduced in [16], providing options for precision adjustments and enhancing efficiency. Nonetheless, further validation is required, extending beyond specific contexts and multi-precision schemes.

Hardware designs of PEs equipped with SIMD-enabled MAC and AFs have exhibited remarkable efficiency, supporting either a single 16-bit operand or two 8-bit operand-based computations per clock cycle. Another notable innovation involves a multi-precision AF, offering configurability for AF selection as either $tanh$ or $sigmoid$. This configuration enhances PE performance through shared resource utilization, ultimately improving hardware resource efficiency and computational throughput. Motivated by hardware limitations such as resource constraints/optimizations, energy efficiency, computational throughput demands, and accuracy, our research presented here addresses diverse arithmetic precision requirements in DNNs. Computation with higher bit precision exponentially escalates the resource needs, while reducing bit precision may compromise inference accuracy, especially in complex tasks. To enable DNN models to excel in both simple and complex tasks, multi-precision computation with dynamic fraction fixed-point representation is essential. While prior work has explored multi-precision computation in MAC operations, the realm of multi-precision computation for non-linear transformations facilitated by AFs remains under-explored.

To bridge these gaps, our research introduces PEs for the convolution layers, featuring a novel multi-bit precision dynamic fraction fixed-point MAC unit and multi-precision AFs (8 or 16-bit with shared resources) within the PE. These PEs can dynamically switch between operational modes (Mode 0 for 8-bit operands `sfixed<8,f>` and Mode 1 for 16-bit operands `sfixed<16,f>`) as shown in Fig. 2. The PE comprises four 8-bit multipliers followed by accumulation logic and AFs. In Mode 1, it processes 16-bit operands, yielding a quantized 16-bit MAC output. This output is used as input for the AFs with 16-bit arithmetic processing. In mode 1, the final output from PE is one 16 bits, a kind of PE1 output. In Mode 0, the MAC unit handles four 8-bit operands, executing dot-product operations for the upper two as $\sum(A_m \times B_m)+C_1$ (corresponding to MAC1) and for the lower two as $\sum(A_n \times B_n) + C_2$ (corresponding to MAC2). Consequently, two quantized 8-bit outputs are produced and forwarded to the AF. The AF computes two separate 8-bit values, generating two distinct outputs corresponding to a kind of PE1 and PE2.

## III. Design of SIMD Dynamic Fraction Fixed-Point Processing Engine

The need for diverse arithmetic ranges and precision computations in neural network models and layers has been debated in various literature and affirmed. To address diverse precision needs, this article introduces a specialized SIMD architecture PE for multi-precision computation, supporting flexible, dynamic fraction fixed-point representations in the *sfixed<N,f>* format. It features an adaptable MAC unit and a unique AF unit that enables multi-precision dynamic fraction fixed-point computation, accommodating Relu, sigmoid, and tanh transformations using shared hardware resources. This approach optimizes resource usage, throughput, configurability, and energy efficiency. In the next subsection, we provide the design details of SIMD micro-architecture for multi-precision dynamic fraction fixed point MAC and AF units. Our novel architecture incorporates distinctive multi-precision Cordic-based AFs. In Table I, we compare the current state-of-the-art PE designs and our proposed architecture on the aspects of SIMD multi-precision, data types, dynamic fraction, and concurrent 8-bit

TABLE I
SIMD Processing Engine (MAC Unit and Cordic AFs)- Existing Works Vs This Work

| Design Parameters | MAC unit | | | Cordic AFs | | | |
|---|---|---|---|---|---|---|---|
| | [6] | [17] | This Work | [18] | [19] | [16] | This work |
| Multi-Precision | Yes<br>4/8/16-bit | Yes<br>4/8-bit | Yes<br>8/16-bit | *No*<br>*16 or 8 bit only* | No<br>16 or 8 bit only | No<br>16 or 8 bit only | Yes<br>16/8 bit |
| SIMD support | SIMD | SIMD | SIMD | Non-SIMD | Non-SIMD | Non-SIMD | SIMD |
| Data Types | Fixed/Float | Integer | Fixed | Fixed | Fixed | Fixed | Fixed |
| Dynamic 'f':1 to N | Yes | No | Yes | No | No | No | Yes |
| 4/8/16-bit ops/cycle | –/2/1 | 8/2/– | –/4/1 | –/1/1 | –/1/1 | –/1/1 | –/2/1 |
| # of multipliers(N-bit) | 4 (8-bit) | 8(4-bit) | 4 (8-bit) | – | – | – | – |
| #AFs Unit for SIMD | – | – | – | Non-SIMD | Non-SIMD | Non-SIMD | 1 for both 8,16-bit |

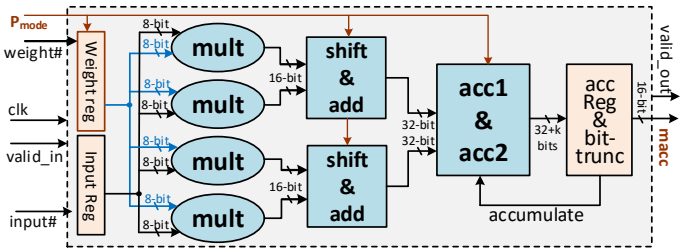


Fig. 2. Unleashing a SIMD MAC Engine for DNNs: The Multi-Precision Dynamic Fraction Fixed-Point MAC Unit

and 16-bit computations per clock cycle. In [17], they employed eight 4-bit multipliers, capable of performing either two 8-bit multiplications for 8-bit precision or eight multiplications of 4-bit data. This approach utilizes the available resources fully, similar to our method for 8-bit and 16-bit precision, although it primarily supports integer calculations.

### *A. Multi-Precision dynamic fraction fixed-point Multiply-Accumulate (MAC) unit in Processing Engine*

We developed a versatile multi-precision dynamic fraction fixed-point MAC unit for SIMD PEs, handling 16-bit and 8-bit operands. In contrast to a typical multi-precision MAC unit, which handles a single 16-bit computation or two 8-bit computations [17], the proposed design employs four 8-bit multipliers. It conducts either single 16-bit multiplication or four parallel 8-bit multiplications with accumulation per cycle (as depicted in Fig. 2), resulting in double the performance when dealing with 8-bit computations. MAC computation (Eqn. 1) includes a bias preloaded in the accumulator register. In Equation 1, $i$ ranges from 1 to $k$, representing inputs ($\mathtt{in}_i$ as input feature, $\mathtt{wt}_i$ as weight, and '$b$' as bias). MAC output ($\mathtt{macc}$) is $\mathbf{Z}$, serving as input to the AF ($\mathbf{Z}_0$). The versatile SIMD MAC unit employs 8-bit multipliers for supporting higher precision (16-bit) computations. Algorithm 1 details the data flow using 8-bit multipliers for 8-bit or 16-bit dynamic fraction fixed-point inputs. For 16-bit input, it results in a single output ($\mathtt{macc[15:0]}$), while for 8-bit input, it produces two 8-bit values (i.e., $\mathtt{macc[15:8]}$, $\mathtt{macc[7:0]}$). The MAC unit's micro-architecture is shown in Figure 2.

$$\sum_{i=1}^{k} (\mathbf{in_i} \times \mathbf{wt_i}) + \mathbf{b} = \mathbf{Z} \tag{1}$$

The algorithm has iteratively processed the shifted values of **in** and has accumulated them to compute the final result. This process depends on the sum of the iteration variable '$q$,' where its value equals the sum of the number of bits for $\mathtt{int\_bits}$ and $\mathtt{frac\_bits}$ within the for loop. Consequently, the elements within the variable **temp** are either right-shifted or left-shifted, contingent upon the number of fractional bits. Within this procedure, **wt** is expanded to match a bitwidth equal to two times the sum of $\mathtt{int\_bits}$ and $\mathtt{frac\_bits}$ through modifications to the value of $\mathbf{wt}\lceil q \rceil$. Inside a loop iterating for $q <$ fract_bits, the right-shifted values of **in_ext** and $\mathbf{wt}\lceil q \rceil$ undergo a bitwise AND operation to calculate the new values for both $\mathbf{temp}\lceil q \rceil$ and **acc_value**. Conversely, left-shift operations are conducted with the same logic. Moreover, the

**Algorithm 1** Multi-Precision (8 or 16-bit) Fixed-Point MAC Algorithm with Variable Fractional Component

**Require:** $\mathtt{in}_i$, $\mathtt{wt}_i$, b, $\mathtt{P}_{mode}$ (0 or 1): Input parameters
**Ensure:** macc, acc1, acc2: SIMD MAC outputs

**if** $\mathtt{P}_{mode}$= 0 **then**
  **for** $i \leftarrow 1$ to $k$ **do**
    $\mathtt{acc1} \leftarrow \mathtt{acc1} + \big((\mathtt{in}_i \times \mathtt{wt}_i) + (\mathtt{in}_{i+1} \times \mathtt{wt}_{i+1})\big)$
    $\mathtt{acc2} \leftarrow \mathtt{acc2} + \big((in_{i+2} \times \mathtt{wt}_{i+2}) + (\mathtt{in}_{i+3} \times \mathtt{wt}_{i+3})\big)$
  **end for**
  $\mathtt{acc\_temp1} \leftarrow \mathrm{trunc}[(\mathtt{acc1} + b),\ 8]$
  $\mathtt{acc\_temp2} \leftarrow \mathrm{trunc}[(\mathtt{acc2} + b),\ 8]$
  $\mathtt{macc} \leftarrow \{\mathtt{acc\_temp1}[7:0], \mathtt{acc\_temp2}[7:0]\}$
**else**
  **for** $i \leftarrow 1$ to $k$ **do**
    $\mathtt{acc1} \leftarrow \big(\mathtt{in}_i[7:0] \times \mathtt{wt}_i[7:0]\big) + \big((\mathtt{in}_{i+1}[15:8] \times \mathtt{wt}_{i+1}[7:0])<<8\big)$
    $\mathtt{acc2} \leftarrow \big((\mathtt{in}_i[7:0] \times \mathtt{wt}_i[15:8])<<8\big) + \big(\mathtt{in}_{i+1}[15:8] \times \mathtt{wt}_{i+1}[15:8])<<16\big)$
    $\mathtt{acc\_temp} \leftarrow \mathtt{acc\_temp} + \big(\mathtt{acc1} + \mathtt{acc2}\big)$
  **end for**
  $\mathtt{macc} \leftarrow \mathrm{trunc}[(\mathtt{acc\_temp} + b),\ 16]$
**end if**

value assigned to **result**, which corresponds to **acc_value**, is updated by adding the value of $\mathbf{temp}\lceil q \rceil$ at position '$q$' to its existing value. In this context, **in_ext** refers to the extension of **in**, achieved by adding zeros to the MSB side to align its bitwidth with that of **wt**, thus enabling consistent processing. The determination of the result's sign bit relies on the XOR operation applied to the sign bits of both **in** and **wt** parameters.

When using 16-bit operands, the MAC unit accumulates $k$ times, producing a single 16-bit output. However, with 8-bit operands, it concurrently handles the upper two 8-bit multipliers, resulting in **acc1** after $k$ accumulations and **acc2** from the remaining two 8-bit multipliers. In the latter scenario, a shift and addition operation is required for 16-bit multiplication with an 8-bit multiplier ($P_{mode} = 1$). It efficiently manages resources for parallel four 8-bit or single 16-bit computations. For 8-bit operands, two parallel outputs, **acc1** and **acc2**, each with (16+k)-bits (where k is the overflow bits calculated as $log_2(j)$), are rounded to 8 bits and assigned to **macc**, following the $\mathtt{P}_{mode}$-0 configuration. In the case of 16-bit operands, the single output (32+k)-bits are rounded to 16 bits and assigned to **macc**, which is applied to the same above AF, configurable to handle the 16 bits. Algorithm 1 details these multi-precision MAC operations supporting two precision modes: 8-bit and 16-bit, determined by the $\mathbf{P}_{mode}$ input. In 8-bit mode ($\mathtt{P}_{mode}$=0), 'k' iterations compute **acc1** and **acc2** by multiplying and adding $\mathbf{in}_i$ and $\mathbf{wt}_i$, followed by truncation for 8-bit segments combined into **macc**. In 16-bit mode ($\mathtt{P}_{mode}$=1), similar iterations involve bit shifts for **acc1** and **acc2**, resulting in a 16-bit **macc**. This versatile MAC unit accommodates various operand precisions for flexible precision computations, as shown in Algorithm 1.

### *B. Multi-Precision Cordic Algorithm and Its Hardware Implementations for Configuring Multiple AFs with SIMD Capability*

In this section, Cordic's [20] multi-precision capabilities, hardware design, and applications in configuring multiple SIMD-supported AFs for DNNs are investigated. Cordic em-

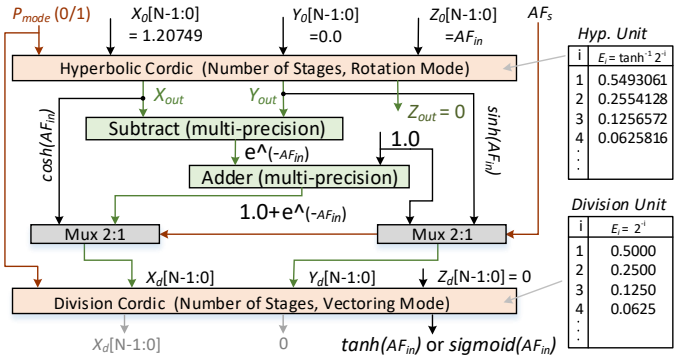


Fig. 3. Cordic-based AF Unit for Multi-Precision Dynamic Fraction Fixed Point: A SIMD AF Engine. The longest path for the sigmoid function is indicated in green. $\mathtt{P}mode$ '0' pertains to 8-bit operations, and '1' corresponds to 16-bit operations.

ploys iterative and pipeline designs, with each stage dedicated to specific computations for efficiency [19]. Each rotation ($i$-th) represents a step in the process, iteratively transforming vector coordinates ($\mathtt{X}_i$, $\mathtt{Y}_i$) into ($\mathtt{X}_{i+1}$, $\mathtt{Y}_{i+1}$). Cordic's precision and efficiency rely on scaling factor $S_i$ for adjusting vector rotations in different modes [21]. The computation $\mathrm{S}_i$ involves factoring out $\cosh(\alpha_i)$ for hyperbolic mode and $\cos(\alpha_i)$ for circular mode as the scaling factor ($\mathrm{S}_i$) in Eqn. (2). $\alpha_i$ represents the rotation angle in radians [22].

$$\begin{pmatrix}\mathtt{X}_{(i+1)}\\ \mathtt{Y}_{(i+1)}\end{pmatrix} = S_i \cdot \begin{pmatrix}1 & -m\cdot d_i\cdot 2^{-i}\\ d_i\cdot 2^{-i} & 1\end{pmatrix}\begin{pmatrix}\mathtt{X}_i\\ \mathtt{Y}_i\end{pmatrix} \tag{2}$$

A detailed analysis of Cordic's operation is provided in [19]. In Equation 2, $S_i$ is associated with $\cosh(\alpha_i)$ in hyperbolic rotation. In this mode, $S_i$ has a constant scaling factor of 0.8281 in pseudo-rotation, and $X_i[N]$ undergoes an offset adjustment of $1/S_i = 1.20749$. The Cordic algorithm performs trigonometric functions ($\sinh$, $\cosh$) and subsequent AFs ($\tanh$, $\sigma$) in hyperbolic mode [16], [19]. Hardware transformations of trigonometric equations are discussed in [22]. Equation 3 includes common equations used for Cordic hardware modification with multi-precision features. Variables $\mathtt{X}_i$, $\mathtt{Y}_i$, and $\mathtt{Z}_i$ represent values at the $i$-th iteration, and $\alpha_i$ denotes the rotation angle in radians for each iteration. $E_i$ serves as the memory element for that iteration, with values corresponding to $2^{-i}$, $\tan^{-1}(2^{-i})$, and $\tanh^{-1}(2^{-i})$ for Linear, Circular, and Hyperbolic coordinates, respectively. Equation 3 represents the generalized form adapted for hyperbolic functions by setting $\mathrm{m} = -1$ and for linear vector mode by fixing $\mathrm{m} = 0$.

$$\mathtt{X}_{i+1} = \mathtt{X}_i - m\cdot d_i\cdot \mathtt{Y}_i\cdot 2^{-i} \tag{3a}$$
$$\mathtt{Y}_{i+1} = \mathtt{Y}_i + d_i\cdot \mathtt{X}_i\cdot 2^{-i} \tag{3b}$$
$$\mathtt{Z}_{i+1} = \mathtt{Z}_i - d_i\cdot \mathrm{E}_i \tag{3c}$$

*1) **Multi-precision Hyperbolic Cordic Computation Micro-Architecture**:* We have realized the multi-precision dynamic fraction fixed-point computations in SIMD AFs using Cordic equations (Eqn. 3), involving hyperbolic functions and division operations implemented through Cordic architecture in hyperbolic and linear modes, respectively. We have implemented

**Algorithm 2** Hyperbolic `sinh` and `cosh` using Cordic

**Require:** $\mathtt{X}_{0_H}$, $\mathtt{X}_{0_L}$ $\mathtt{Y}_{0_H}$, $\mathtt{Y}_{0_L}$ $\mathtt{Z}_{0_H}$, $\mathtt{Z}_{0_L}$ & $frac(\mathtt{f})$ : input param.
**Ensure:** $\mathtt{X}_{out}$, $\mathtt{Y}_{out}$: concatenated hyperbolic Cordic output
**Require:** $\mathtt{AF}_{in}$: input applied at $\mathtt{Z}_{0_H}$ and $\mathtt{Z}_{0_H}$ with 8-bit each.
**Require:** $\mathtt{P}_{mode}$: Operation mode (0 for 8-bit or 1 for 16-bit).
1: $\mathtt{P}_{mode}$ based Initialize $\mathtt{X}_0[N]$ = 1.20749, $\mathtt{Y}_0[N]$ = 0.0, $\mathtt{Z}_0[N]$ = $\mathtt{AF}_{in}$, $m = -1$, $d_i$ = $\mathtt{Z}_i[N-1]$ (sign bit).
2: **for** $i \leftarrow 1$ to $n$ **do**
3: $\quad \mathtt{X}_{i_L} \leftarrow \mathtt{X}_{(i-1)_L}$ - $d_i \cdot(m\cdot \mathtt{Y}_{(i-1)_L}\cdot 2^{-i})$ ▷ calc. $\mathtt{X}_i$
4: $\quad \mathtt{Y}_{i_L} \leftarrow \mathtt{Y}_{(i-1)_L}$ + $d_i \cdot(m\cdot \mathtt{X}_{(i-1)_L}\cdot 2^{-i})$ ▷ calc. $\mathtt{Y}_i$
5: $\quad \mathtt{Z}_{i_L} \leftarrow \mathtt{Z}_{(i-1)_L}$ + $d_i \cdot\tanh^{-1}(2^{-i})$ ▷ calc. $\mathtt{Z}_i$
6: $\quad$ **if** $\mathtt{P}_{mode}$ = 0 **then**
7: $\quad\quad \mathtt{X}_{i_H} \leftarrow \mathtt{X}_{(i-1)_H}$ - $d_i \cdot(m\cdot \mathtt{Y}_{(i-1)_H}\cdot 2^{-i})$
8: $\quad\quad \mathtt{Y}_{i_H} \leftarrow \mathtt{Y}_{(i-1)_H}$ + $d_i \cdot(m\cdot \mathtt{X}_{(i-1)_H}\cdot 2^{-i})$
9: $\quad\quad \mathtt{Z}_{i_H} \leftarrow \mathtt{Z}_{(i-1)_H}$ + $d_i \cdot\tanh^{-1}(2^{-i})$
10: $\quad$ **else**
11: $\quad\quad \mathtt{X}_{i_H} \leftarrow \mathtt{X}_{(i-1)_H}$ - $d_i \cdot(m\cdot \mathtt{Y}_{(i-1)_H}\cdot 2^{-i})$ + OF($\mathtt{X}_{i_L}$)
12: $\quad\quad \mathtt{Y}_{i_H} \leftarrow \mathtt{Y}_{(i-1)_H}$ + $d_i \cdot(m\cdot \mathtt{X}_{(i-1)_H}\cdot 2^{-i})$ + OF($\mathtt{Y}_{i_L}$)
13: $\quad\quad \mathtt{Z}_{i_H} \leftarrow \mathtt{Z}_{(i-1)_H}$ + $d_i \cdot\tanh^{-1}(2^{-i})$ + OF($\mathtt{Z}_{i_L}$)
14: $\quad$ **end if**
15: **end for**
16: **if** $\mathtt{P}_{mode}$ = 0 **then**
17: $\quad \mathtt{X}_{out} \leftarrow \{\mathrm{trunc}(\mathtt{X}_{n_H},\ 8),\ \mathrm{trunc}(\mathtt{X}_{n_L},\ 8)\}$
18: $\quad \mathtt{Y}_{out} \leftarrow \{\mathrm{trunc}(\mathtt{Y}_{n_H},\ 8),\ \mathrm{trunc}(\mathtt{Y}_{n_L},\ 8)\}$
19: **else**
20: $\quad \mathtt{X}_{out} \leftarrow \mathrm{trunc}\big(\{\mathtt{X}_{n_H}, \mathtt{X}_{n_L}\},\ 16\big)$
21: $\quad \mathtt{Y}_{out} \leftarrow \mathrm{trunc}\big(\{\mathtt{Y}_{n_H}, \mathtt{Y}_{n_L}\},\ 16\big)$
22: **end if**
23: **return** $\mathtt{X}_{out}$, $\mathtt{Y}_{out}$

16-bit computations for AFs based on the $\mathtt{P}_{mode}$ signal, utilizing two 8-bit computational hardware blocks. Memory constants ($E_i$), essential for computing the results, are visualized in Fig. 3. Utilizing Pareto analysis, the design incorporates a 5-stage parallel Cordic for hyperbolic functions (sinh & cosh) in hyperbolic mode and a division block with a 5-stage Cordic tailored for sigmoid or tanh functions in the *sfixed*<N,f> format. The micro-architecture for multi-precision Hyperbolic Cordic computation, as shown in Fig. 4, accommodates two 8-bit operands in $\mathtt{P}_{mode}$=0 and a single 16-bit operand in $\mathtt{P}_{mode}$=1. Algorithm 2 illustrates multi-precision Cordic calculations for hyperbolic `sinh` and `cosh` using SIMD support in both iterative and pipelined modes. We have implemented a pipeline architecture for results evaluation and analysis. It produces 16-bit results, which translate to a single 16-bit output for 16-bit operands and two 8-bit packets for 8-bit operands. The output includes $\mathtt{X}out$ (cosh) and $\mathtt{Y}out$ (sinh) values, with 'OF' indicating the overflow bit for lower 8-bit computations."

We have evaluated `sinh` and `cosh` functions using multi-precision calculations, yielding two parallel outputs: `out[15:8]` and `out[7:0]` for 8-bit operands, or a single 16-bit output as `out[15:0]` for 16-bit operand. Algorithm 2 outlines the methodology for hyperbolic function calculations, accommodating both 16 and 8-bit precision and computation of one 16-bit operand and two parallel 8-bit operands with shared hardware resources. To ensure N-bit precision, we truncated the output $\mathtt{X}_n$ and $\mathtt{Y}_n$ to obtain $\mathtt{X}_{\mathrm{out}}$ and $\mathtt{Y}_{\mathrm{out}}$, representing $\cosh(Z_0)$ and $\sinh(Z_0)$. Computation of negative exponential values followed Eqn. 5(a), applied in sigmoid calculations. Figure 4 illustrates the hardware for bit-level mapping and multi-precision Cordic computation, capable of handling either

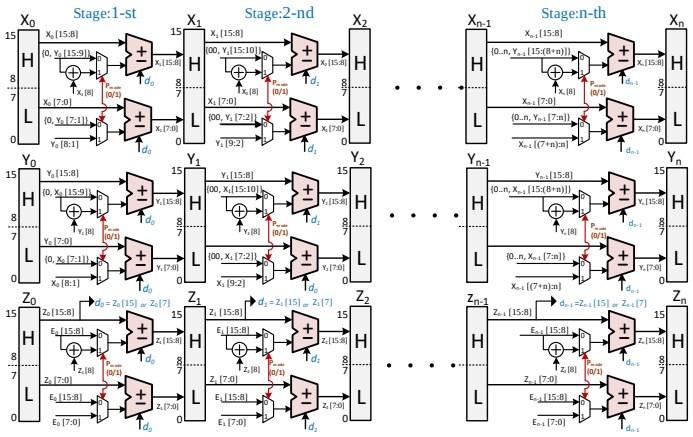


Fig. 4. Multi-Precision Hyperbolic Cordic: Using two 8-bit operands for $\mathtt{P}_{mode}$=0 and a single 16-bit operand for $\mathtt{P}_{mode}$=1. Outputs at n-th stages: $\mathtt{X}_n = \cosh(\mathtt{Z}_0)$, $\mathtt{Y}_n = \sinh(\mathtt{Z}_0)$, and $\mathtt{Z}_n \approx 0$.

a single 16-bit operand or two 8-bit operands, enabling multi-precision AFs. Output generation aligns with the procedure outlined for `out[15:0]`, featuring 8-bit adders, shifters, memory constants, and pipeline registers.

*2) **Multi-Precision Cordic Micro-Architecture for SIMD Division** :* The Cordic configured in linear vectoring mode does the division operation (`m` = 0) with memory element $E_i$ set to $2^{-i}$, the micro-architecture for the algorithm which produces output, is shown in Figure 3. It can perform multi-precision division operations by configuring the divisor ($\mathtt{X}_{d0}$), dividend ($\mathtt{Y}_{d0}$), and initial value ($\mathtt{Z}_{d0}$ = 0). After #stages pipeline stages using Eqn 4, it produces $\mathtt{Z}_n$, representing the quotient $\mathtt{Y}_{d0}/\mathtt{X}_{d0}$. The top-level hardware design in Fig. 3 has implemented multi-precision Cordic-based configurable AFs, and the sub-block used for division using Cordic is depicted in Fig. 5. In this $\mathtt{D}_i$ is considered as the result of applying the XNOR operation to the signs of variables $\mathtt{X}_i$ and $\mathtt{Y}_i$. Division operations generate outputs for the *tanh* or *sigmoid* functions, as described in Eq. 5(b) and Eq. 5(c), respectively. These AFs support either two 8-bit operands or a single 16-bit operand, enabling dynamic fixed-point representation in AF computations. The hyperbolic calculation methodology outlined in Algorithm 2 has been adapted for the division block with specific parameters in linear vectoring mode. The algorithm can be configured using either iterative stage architecture or a pipeline design for high-throughput computation. We have implemented this algorithm in Verilog HDL, optimizing it for hardware performance based on available resources and throughput requirements. Traditionally, the number of Cordic stages is determined by the fractional bits in the adaptive fixed-point representation. However, through Pareto analysis, an appropriate number of stages has been determined for error-resilient applications while allowing for approximation.

$$\mathtt{X}_{n+1} = \mathtt{X}_i \tag{4a}$$

$$\mathtt{Y}_{i+1} = \mathtt{Y}_i + d_i \cdot \mathtt{X}_i \cdot 2^{-i} \tag{4b}$$

$$\mathtt{Z}_{i+1} = \mathtt{Z}_i - d_i \cdot 2^{-i} \tag{4c}$$

The hardware design of the multi-precision Cordic includes

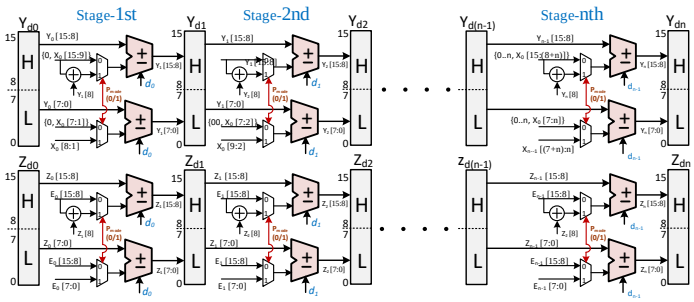


Fig. 5. Multi-precision Cordic Division: 8-bit operands ($\mathtt{P}_{mode}$=0) and 16-bit operand ($\mathtt{P}_{mode}$=1), n-th stage outputs: $\mathtt{Z}_{dn} = \mathtt{Y}_{d0}/\mathtt{X}_{d0}$.

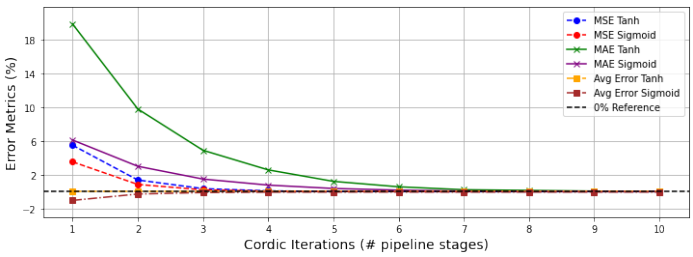


Fig. 6. Error metrics (MSE, MAE, and SD) were computed at each iteration by comparing observed data to true AF values.

sign adders, part-select logic (shifters), pipeline registers, and memory constants ($\mathtt{E}_i$). The choice of $\mathtt{E}_i$ depends on the mode of operation. For instance, in the case of an 8-bit operand, $\mathtt{E}_i$`[15:8]` is used for MSBs calculation with an i-th right-shift for both MSBs and LSBs calculations. Conversely, for 16-bit operands, it is obtained from right-shifted $\mathtt{E}_i$`[15:8]` and $\mathtt{E}_i$`[(7+i): i]`. The memory constants $\mathtt{E}_i$ are initially scaled by a factor of $2^f$, determined by the available fractional bits in the dynamic fixed-point representation (*sfixed*<N,f>). This expansion extends the computation range, resulting in two configurations: one for selecting input values and memory elements as *sfixed*<N,f>, and the other for selecting part-select bits considering shifted values used in the i-th stage of Cordic computation.

$$e^{-\mathtt{AF}_{in}} = \cosh(\mathtt{AF}_{\mathrm{in}}) - \sinh(\mathtt{AF}_{\mathrm{in}}) \tag{5a}$$

$$\tanh(\mathtt{AF}_{in}) = \sinh(\mathtt{AF}_{\mathrm{in}})/\cosh(\mathtt{AF}_{\mathrm{in}}) \tag{5b}$$

$$\mathrm{sigmoid}(\mathtt{AF}_{in}) = \frac{1.0}{1.0 + e^{-\mathtt{AF}_{\mathrm{in}}}} \tag{5c}$$

*3) **Optimal pipeline stages for Cordic-based multi-precision AFs**:* As the number of pipeline stages increases, hardware resource utilization also rises. Therefore, it is essential to investigate the optimal stage count that balances computational accuracy and hardware resource efficiency. To achieve this equilibrium, we conducted a Pareto analysis to determine the minimum number of stages necessary without significant accuracy loss. In our comprehensive Pareto analysis, we rigorously assessed error metrics while processing observed data with our proposed algorithm across varying iterations (pipeline stages). At each `i`-th iteration, we computed Mean Squared Error (MSE), Mean Absolute Error (MAE), and Standard Deviation (SD) by comparing observed data with true AF values. The

analysis, as shown in Fig. 6, reveals that error metrics quickly approach a minimal level after approximately #5 pipeline stages, indicating that using more than five stages provides negligible accuracy improvements. This selection optimizes area utilization, power consumption, and critical delay for an efficient multi-precision AF implementation using Cordic.

To optimize the algorithm's real-world hardware performance, we have observed that both $\texttt{P}_{mode}$ 0 with 8-bit precision and $\texttt{P}_{mode}$ 1 with 16-bit precision require approximately five pipeline stages. Therefore, we have implemented up to five pipeline stages for hyperbolic and division operations for the *sfixed<N,f>*. The same configuration has been subjected to inference accuracy evaluation and hardware performance assessment. Here, N corresponds to operand bit precision, specifically 8-bit and 16-bit for $\texttt{P}_{mode}$ 0 and 1, respectively. It's worth noting that in both precision settings, the number of Cordic stages can be dynamically configured to accommodate variable fractional bit requirements. To address potential overflow during addition and shifting (part-select bits), the bit representation for each stage has been increased by one bit. Additionally, the final output of the hyperbolic Cordic, encompassing `sinh` and `cosh` values, has been truncated to *N* bits based on $\texttt{P}_{mode}$ and subsequently provided to the division operation via an additional adder for *tanh* and *sigmoid* evaluation. We've implemented hyperbolic and division computations illustrated in Fig. 3 using the Cordic architecture across various combinations (variable Cordic stages) for Pareto analysis, with dynamic fixed-point representation to conclude on the pipeline stages. This architecture enables runtime computation of sigmoid and tanh AFs with dynamic fixed-point *sfixed<N,f>* and multi-precision computations. The Cordic-based AF unit is unified, supporting both sigmoid and tanh functions.

## IV. Inference Accuracy and Hardware Performance: Evaluation and Analysis

We have evaluated our multi-precision dynamic fraction fixed-point PE, designed for SIMD operations, focusing on inference accuracy for image classification tasks. Additionally, an analysis has been conducted on resource utilization and delay in Cordic-based AFs, considering both 8 and 16-bit operands. The results demonstrate the effectiveness of our PEs, having multi-precision dynamic fraction fixed-point MAC and AFs, allowing adaptive precision-aware DNN computations.

- Develop and verify a SIMD PE with configurable MAC and Cordic-based AF units using a Python-based emulator for accuracy and hardware performance assessment.
- Pareto analysis to identify the optimal number of Cordic stages to balance accuracy, area, power, and delay.

### A. Experimental Validation of multi-precision Fixed-Point PE

By modeling the proposed multi-precision dynamic fraction fixed-point PEs in a Python context, we have extensively evaluated the performance of PE. The model was created to evaluate the inference accuracy of DNN models on our proposed PE. The PE model in Python is bit-level compatible with the PE's hardware behavioral logic. This ensures that the models behave similarly to the Verilog-implemented hardware. The primary goal has been to assess the inference accuracy of the PEs in the context of the hardware. These models have been created utilizing the multi-precision dynamic *sfixed<8,f>* (four operands) and *sfixed<16,f>* (single operand) representations, which are equivalent to the hardware architecture, to ensure the same runtime inference accuracy. To ensure constant bit widths throughout the computation, we have utilized the `trunc` library function from *numpy*. No additional parameters were employed to achieve optimal accuracy. We have investigated and reported the inference accuracy for various bitwidths in Table II for dynamic fraction fixed-point.

### B. Accuracy Analysis with multi-precision PE Software (Python based) Emulator

We have performed an extensive inference accuracy analysis, comparing our proposed multi-precision dynamic fraction fixed-point MAC and Cordic-based AFs enabled PE with a reference PE model [23] which use float32 precision for MAC and AFs and ROM-based design for AFs [13]. Hence, data type and precision-dependent inference accuracy and ROM Vs. CORDIC AF effect on inference accuracy are compared and analyzed. Results in Table II show accuracy for three neural network models: LeNet (MNIST), AlexNet, and VGG16 (CIFAR-10). These networks use ReLU followed by $sigmoid$ and $tanh$ activations with 8-bit operands for dynamic *sfixed<8,f>* and with 16-bit operands for *sfixed<16,f>* representations. We explored various *sfixed<8,f>* and *sfixed<16,f>* fractional bit configurations for inference accuracy. Specific fraction fixed-point representations resulted in consistently better accuracy values in reference to the benchmark PE from Tensor [23]. For *sfixed<16,f>*, accuracy was <1% lower than Float32 across three neural network models, demonstrating the effectiveness of our PE with Cordic-based dynamic fraction fixed-point AFs for 16-bits. The results are comparable with ROM-based AFs as well. However, 8-bit precision resulted in significant accuracy drops for *sfixed<8,5>* and *sfixed<8,2>* due to reduced bit widths. VGG16 achieved the highest accuracy with 3 and 10 fraction bits for *sfixed<8,3>* and *sfixed<16,10>*, respectively.

Results show that specific fixed-point representations are comparable in accuracy to the benchmark PE from Tensor [23] and ROM-based AF model [13]. Various precision settings were evaluated for 8-bit and 16-bit dynamic fraction fixed-point representations. Dynamic fraction fixed-point AFs using multi-precision Cordic exhibit minimal accuracy loss. For `sfixed<16,f>`, the accuracy loss was <1% for LeNet with MNIST, <1% for AlexNet with CIFAR-10, and <2% for VGG16 with CIFAR-10 compared to Float32 reference PE [22]. Cordic approaches maintain accuracy similar to Tensor or ROM-based hardware, demonstrating their effectiveness for multi-precision dynamic fixed-point PEs at 8-bit and 16-bit bitwidths. Higher accuracy is observed for `sfixed<8,3>` and `sfixed<16,10>`, attributed to broader integer and fractional bit ranges. In 8-bit precision, <8,5> and <8,2> configurations show decreased accuracy due to their narrower integer and fraction scale, which is not observed in 16-bit.

We have extensively analyzed the accuracy of our PE with Cordic-based multi-precision AFs, comparing them to a reference PE model [23] using float32 precision and ROM-based design [13]. Table II presents accuracy results for three

TABLE II
COMPARISON OF INFERENCE ACCURACY: SIMD DYNAMIC FRACTION FIXED-POINT PES WITH CORDIC-BASED AFS VS. FLOAT32 [23] VS. ROM-BASED AFS [13]

| Fixed-point | Inference Accuracy(%) with ROM & Cordic AFs (sigmoid & tanh) and ReLU | | | | | |
|---|---|---|---|---|---|---|
| Variable '$f$' bits | MNIST@LeNet | | CIFAR-10@AlexNet | | CIFAR-10@VGG16 | |
| PE_float32 [23] | 98.9 | | 78.2 | | 84.8 | |
| Processing Element (PE) | $MAC + AF_{ROM}$ ROM [13] | $MAC + AF$ $Our\ SIMD$ | $MAC + AF_{ROM}$ ROM [13] | $MAC + AF$ $Our\ SIMD$ | $MAC + AF_{ROM}$ ROM [13] | $MAC + AF$ $Our\ SIMD$ |
| dynamic fixed-point sfixed<8,f> | | | | | | |
| sfixed<8,5> | 97.5 | 97.3 | 24.8 | 14.0 | 34.0 | 29.0 |
| sfixed<8,4> | 98.2 | 97.5 | 72.0 | 68.7 | 80.0 | 78.0 |
| sfixed<8,3> | 97.6 | 97.1 | 51.0 | 50.4 | 80.2 | 79.1 |
| sfixed<8,2> | 93.8 | 94.0 | 26.0 | 25.2 | 66.2 | 52.1 |
| dynamic fixed-point sfixed<16,f> | | | | | | |
| sfixed<16,12> | 98.0 | 98.0 | 77.0 | 69.2 | 83.0 | 82.0 |
| sfixed<16,11> | 98.9 | 98.2 | 77.8 | 77.0 | 83.0 | 81.0 |
| sfixed<16,10> | 98.0 | 98.0 | 78.0 | 78.0 | 84.1 | 83.7 |

neural network models: LeNet (MNIST), AlexNet, and VGG16 (CIFAR-10), employing ReLU, sigmoid, and tanh activations with 8-bit (*sfixed<8,f>*) and 16-bit (*sfixed<16,f>*) operands. Various fractional bit configurations were explored for inference accuracy. The *sfixed<16,f>* precision showed less than 1% accuracy reduction compared to Float32, highlighting the effectiveness of Cordic-based dynamic fixed-point AFs with 8-bit and 16-bit bitwidths. However, 8-bit precision led to significant accuracy drops for *sfixed<8,5>* and *sfixed<8,2>* due to reduced fractional value range. VGG16 achieved the highest accuracy with 3 and 10 fractional bits for operands with *sfixed<8,3>* and *sfixed<16,10>*, respectively.

In summary, Lenet, Alexnet, and VGG16 achieved high accuracy using multi-precision dynamic fixed-point PEs and Cordic-based AFs with `sfixed<8,3>` compared to the Tensor-based/ROM-based model. Lenet reached peak accuracy with #11 fractional bits for `sfixed<16,f>`, while Alexnet and VGG16 achieved it with #10 fractional bits. These findings emphasize the need for multi-precision dynamic fixed-point computation tailored to each layer/model, ensuring maximum accuracy for specific datasets and DNN layers/models. Our multi-precision PE designs with adaptability and flexibility in `sfixed<N,f>` demonstrate its effectiveness in accommodating dynamic fixed-point data representation addressing SIMD-supported PEs in deep learning accelerators.

### C. Hardware Design and Implementation Analysis

In this section, we provide a comprehensive analysis of the proposed SIMD multi-precision dynamic fraction fixed-point PE with shared resoources, which encompasses both the MAC unit and Cordic AFs (sigmoid and tanh). The hardware realization has been carried out using Verilog HDL, and the hardware post-implementation results were obtained using the *Vivado*-Xilinx tool. We utilized the *Virtex*-VCU118 Evaluation kit for our experiments and analysis. The design includes MAC and AFs with the same data flow as the Python-based model to maintain the same input-output precision of the PE. The study focuses on hardware utilization, critical delay, and *fmax*.

Our novel architecture incorporates a unique SIMD multi-precision Cordic-based unified AF supporting *RelU, sigmoid, and tanh*. We couldn't find such a design approach for AFs. The SIMD architecture realization, with both conventional non-shared resources and our innovative shared resource design, demonstrates superior multi-precision capabilities, as shown in Table III. The table offers a comprehensive resource utilization and delay comparison for multi-precision *sfixed<N,f>* PEs with our Cordic AFs, both with and without shared resources. The analysis includes both standalone 8-bit and 16-bit precision and proposed scenarios with SIMD support. In the standalone PEs scenarios, we used separate hardware for 8-bit and 16-bit precision while considering SIMD enablement in multi-precision cases. The proposed SIMD design efficiently reuses hardware resources without compromising performance.

We have designed standalone MAC units for both 8-bit and 16-bit precisions. We addressed the SIMD (Single Instruction, Multiple Data) data flow, necessitating a parallel hardware implementation without consuming additional hardware resources. Our analysis involved incorporating four 8-bit MAC units and a single 16-bit MAC unit for evaluation without shared resources. This configuration consumed a total of 139 CLBs (Configurable Logic Blocks), 627 LUTs (6-bit Lookup Tables), and 179 FFs (Flip-Flops). In comparison, our proposed multi-precision design with shared resources for 8/16-bit computations required only 71 CLBs, 302 LUTs, and 31 FFs. This showcases a significant improvement in resource efficiency, with reductions of 48.92%, 51.83%, and 82.68%, respectively, with a 7.94% decrease in optimum frequency performance (*fmax*).

We have designed 8-bit and 16-bit Cordic-based architectures based on [18], [19] for performance analysis with our proposed shared resources unified SIMD AF, which again supports dynamic fraction, distinguishing it from state-of-the-art designs. This work multi-precision AFs with dynamic fraction fixed-point computation, marking a novel contribution. It's essential to note that both 8-bit and 16-bit precision have separate hardware resource requirements, including two 8-bit precision AF units and one 16-bit AF unit in the non-shared resources architecture. Specifically, the proposed Cordic-based multi-precision AF design demonstrates a significant 43.50% reduction in LUTs and a 38.01% decrease in CLBs compared to the non-shared resources Cordic-based design, which follows a similar approach as [19] but with additional dynamic fraction support. It's important to emphasize that the proposed design achieves a substantial 72.27% reduction in FFs, especially in the context of SIMD support, compared to the standalone Cordic-based AF. Regarding the maximum clock frequency ($f_{max}$), the multi-precision AF exhibits a slight decrease compared to the Cordic without shared resources, with an 8.18% reduction for the 16-bit Cordic. These findings highlight the inherent trade-offs between resource utilization and performance. In comparison to the Cordic-based design in [19], which is adopted for multi-precision without shared resources, our proposed multi-precision Cordic with shared requires fewer hardware resources, including CLBs and LUTs. This design is an optimal choice for the SIMD PE and offers versatile multi-precision support.

## V. CONCLUSIONS AND FUTURE WORKS

This paper introduces an innovative architecture for designing multi-precision dynamic fraction fixed-point PEs capable of accommodating both 8 and 16-bit computations. These PEs are tailored for SIMD architectures and incorporate multi-

TABLE III
HARDWARE UTILIZATION AND DELAY ANALYSIS FOR 8-BIT AND 16-BIT ONLY DESIGNS VS MULTI-PRECISION SIMD ARCHITECTURES(NON-SHARED AND SHARED RESOURCES)

| **Design Parameters** | **PE: `sfixed<8,f>` Only 8-bit support** | | **PE: `sfixed<16,f>` Only 16-bit support** | | **4×8 & 1×16 bit PE SIMD without shared resources** | | **Proposed 4×8/1×16 bit PE SIMD with shared resources** | |
|---|---|---|---|---|---|---|---|---|
| | **This Work** | | **This Work** | | **This Work** | | **This Work** | |
| **Logic Arch.** | MAC Unit | sig & tanh | MAC Unit | sig & tanh | MAC Unit | sig & tanh | MAC Unit | sig & tanh |
| **Resources Utilisation** | | | | | | | | |
| CLBs | 18 | 44 | 67 | 83 | 139 | 171 | 71 | 106 |
| LUT | 87 | 289 | 279 | 487 | 627 | 1065 | 302 | 605 |
| FF | 32 | 225 | 51 | 422 | 179 | 972 | 31 | 116 |
| **Delay Analysis** | | | | | | | | |
| Logic delay (ns) | 1.247 | 1.273 | 1.741 | 1.402 | 1.741 | 1.402 | 1.862 | 1.471 |
| Route delay (ns) | 2.709 | 2.799 | 2.930 | 4.878 | 2.930 | 4.878 | 3.238 | 5.361 |
| $f_{max}$ (MHz) | 252.78 | 245.5 | 214.08 | 159.50 | 214.08 | 159.50 | 197.51 | 146.37 |

precision dynamic fraction fixed-point SIMD MAC units and DNN accelerator-specific AFs, offering various AF selection options. The proposed multi-precision PE incorporates state-of-the-art Cordic-based AFs, adopting the SIMD scheme to provide the flexibility of using either 8-bit or 16-bit AFs to meet varying precision requirements. We achieve inference accuracy comparable to the highest reference accuracy attainable with Tensor-based and ROM-based designs for AFs. Hence, the proposed architecture for the computational units exhibits minimal inference accuracy loss across diverse neural network models and datasets. The hardware performance parameters underscore the proposed PE's effectiveness for precision-aware DNN computations across various models and layers. Users have the flexibility to select appropriate AFs (e.g., sigmoid, tanh, ReLU) in addition to precision and fractional bitwidths for their DNN applications. Further, the architecture enables four 8 bits and one 16 bits MAC computations and two 8 bits and one 16 bits AF computations for higher throughput and low latency. Future endeavors may entail thorough testing and investigation across a wider range of DNN models and datasets, thereby reinforcing the evaluation of the proposed runtime configurable SIMD-based multi-precision dynamic fraction fixed-point computations for model and dataset-dependent DNN acceleration.

## REFERENCES


[1] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2017, pp. 1–8.

[2] M. Weißbrich, A. García-Ortiz, and G. Payá-Vayá, "Comparing vertical and horizontal simd vector processor architectures for accelerated image feature extraction," *Journal of Systems Arch.*, vol. 100, p. 101647, 2019.

[3] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[4] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 3048–3056, 2018.

[5] F. M. Shiri, T. Perumal, N. Mustapha, and R. Mohamed, "A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru," *arXiv preprint arXiv:2305.17473*, 2023.

[6] H. Zhang, D. Chen, and S.-B. Ko, "New flexible multiple-precision multiply-accumulate unit for deep neural network training and inference," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 26–38, 2019.

[7] T. Liang *et al.*, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[8] J. Lee *et al.*, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2018.

[9] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Afs in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, 2022.

[10] S. Abadal *et al.*, "Computing graph neural networks: A survey from algorithms to accelerators," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–38, 2021.

[11] K. Li *et al.*, "A vector systolic accelerator for multi-precision floating-point high-performance computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 10, pp. 4123–4127, 2022.

[12] Z. Dong *et al.*, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

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

[14] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. Leong, "Pir-dsp: An fpga dsp block architecture for multi-precision deep neural networks," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 35–44.

[15] T. Yang *et al.*, "Design space exploration of neural network activation function circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1974–1978, 2018.

[16] H. Chen *et al.*, "A cordic-based architecture with adjustable precision and flexible scalability to implement sigmoid and tanh functions," in *IEEE Inter. Sym. on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[17] N. Neda, S. Ullah, A. Ghanbari, H. Mahdiani, M. Modarressi, and A. Kumar, "Multi-precision deep neural network acceleration on fpgas," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 454–459.

[18] Y. Chang, P. Jokic, S. Emery, and L. Benini, "An ultra-low-power serial implementation for sigmoid and tanh using cordic algorithm," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–2.

[19] S. Mehra *et al.*, "An empirical evaluation of enhanced performance softmax function in deep learning," *IEEE Access*, 2023.

[20] L. Chen *et al.*, "Algorithm and design of a fully parallel approximate coordinate rotation digital computer (cordic)," *IEEE Tran. on Multi-Scale Computing Systems*, vol. 3, no. 3, pp. 139–151, 2017.

[21] P. K. Meher *et al.*, "50 years of cordic: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.

[22] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "A cordic based configurable af for ann applications," in *2020 IEEE computer society annual symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 78–83.

[23] M. Abadi, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.