IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 March 2023, accepted 29 March 2023, date of publication 6 April 2023, date of current version 12 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3265327

RESEARCH ARTICLE

An Empirical Evaluation of Enhanced Performance Softmax Function in Deep Learning

SUMIRAN MEHRA[1], GOPAL RAUT[1], (Member, IEEE), RIBHU DAS PURKAYASTHA[1], SANTOSH KUMAR VISHVAKARMA[1], (Senior Member, IEEE), AND ANTON BIASIZZO[2]

[1]Department of Electrical Engineering, Indian Institute of Technology Indore, Indore 453552, India
[2]Jožef Stefan Institute, 1000 Ljubljana, Slovenia

Corresponding author: Anton Biasizzo (anton.biasizzo@ijs.si)

This work was supported in part by the Slovenian Research Agency (research core funding) under Grant P2-0098; in part by the "Electronics Components and Systems for European Leadership" Joint Undertaking (ECSEL JU) under Grant 101007273 (DAIS), Grant 876038 (InSecTT), and Grant 876659 (iRel40); and in part by the Ministry of Education, Government of India.

ABSTRACT This article present a highly efficient and performance-enhanced Softmax Function (SF) designed for a deep neural network accelerator. The SF is an essential component of deep learning models, primarily used in the classification layer, and also in hidden layers of advanced neural networks like Transformer and Capsule networks. The primary challenge of designing an efficient hardware architecture for SF is the complex exponential and division computational sub-blocks. To address this challenge, a hardware-optimized pipelined CORDIC-based architecture is proposed, leveraging the mutual exclusivity of the *CO-ordinate Rotational DIgital Computer* (CORDIC) algorithm, designed for enhanced throughput, area, and power. To maintain good accuracy in deep learning models, the proposed SF design undergoes a Pareto study that evaluates the variation of accuracy concerning the number of pipeline stages. The proposed design is quantized to 16-bit precision, and inference accuracy is validated for different datasets. The SF is prototyped using Xilinx Zynq FPGA, operating at 685MHz, and ASIC implementation is performed for 45nm technology node at 5GHz of maximum operating frequency. The design achieves a validation accuracy loss of less than 2% while reducing silicon area and Energy-Delay-Product (EDP) by 12×. Post-synthesis simulation results indicate that the proposed design outperforms state-of-the-art architectures, achieving 3× better performance in terms of area, power, and logic delay.

INDEX TERMS Softmax function (SF), CORDIC algorithm, deep learning, hardware optimization, performance enhancement, pipeline stages.

I. INTRODUCTION

The IoT-inspired world is greatly impacted by deep learning, where energy and area-efficient Deep Neural Network (DNN) computation with high throughput is required [1]. Numerous neural network accelerators have been proposed in previous works [1], [2], [3] to enhance the efficiency of DNN accelerators. Improving the neuron's architecture and layer-to-layer interconnects can optimize the runtime configurability of the DNN accelerator. Furthermore, efficient hardware implementation of non-linear functions like Softmax is also a crucial research area [2], [4], [5]. Softmax Activation Function (AF) has been used in the classification layer of present DNN models like AlexNet, ResNet, and LeNet for image, sound, audio, and other classification tasks. Meanwhile, advanced neural network co-processors such as Transformer and Capsule network use Softmax Function (SF) in their hidden layers [6], [7].

The most power and area-intensive blocks in classical Softmax architecture are the Exponential and Division

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

operations [5], [8], [9], [10]. As Softmax moves from lower to higher precision, such as 8-bit, 12-bit, and 16-bit, the number of quantization states increases exponentially, leading to an exponential rise in required memory elements. Conventional memory-based implementations that use a memory lookup table are not suitable for higher precision. To address this, the authors have improved the naive exponential operation to enable an area-efficient implementation at higher bit precision. In [11] and [12], a modified Softmax expression using the *log* unit instead of division is considered, along with a shift and subtract operation to compute division. To compute exponential, a fixed-point input value is divided into integer and fractional parts, while a log-sum exponential trick is utilized for implementation [5], [12], [13], achieving a 10× reduction in hardware area over conventional memory-based DNN accelerators. In [14], the authors have implemented an area and energy-efficient iterative CORDIC based SF by utilizing the property of **CO**-ordinate **R**otational **DI**gital **C**omputer (CORDIC) algorithm, achieving 10 to 100× area reduction. However, the design suffers from low throughput due to its recursive nature.

An investigation into an area and power-efficient computational design has led to the development of an architecture based on the CORDIC-algorithm. This architecture comprises simple logic blocks such as a multiplexer, adder/subtractor, shift register, and a few memory elements, enabling various arithmetic computations with reduced power consumption [15]. CORDIC, an iterative algorithm that performs pseudo rotations in two modes - Vectoring and Rotational, can be used to perform various arithmetic operations, including division, multiplication, trigonometric, and hyperbolic functions [16]. Research on Multiply-Accumulate (MAC) and non-linear Activation Functions (AFs) using the iterative CORDIC algorithm has shown promising results in terms of reduced area. However, a significant concern remains the throughput [14], [17], [18], [19].

The focus of this research article is a hardware-efficient and performance-centric class-adjustable SF. However, the recursive CORDIC algorithm used in the implementation has low throughput. To address this issue, we propose a pipelined CORDIC architecture that balances area, power, throughput, and accuracy with pipeline stages. The key challenge in pipelining is to minimize the area overhead while maintaining high accuracy. To overcome this challenge, we present an empirical approach that achieves a trade-off between area, power, throughput, and accuracy with pipeline stages. The proposed architecture is novel and efficient, achieving low power consumption, reduced computational time, and high throughput. The research is focused on Edge-AI applications that require low-power and high-throughput solutions, despite a reported loss in accuracy. The contributions of this study include a novel pipeline CORDIC architecture that optimizes area, power, and throughput without sacrificing accuracy significantly. The major contributions of the author's work are summarized below:

- The presented SF design enhances performance by utilizing an optimized pipeline staged CORDIC architecture with reduced area, energy consumption, and high frequency of operation. The design uses Block Random Access Memory (BRAM) memory for First-in-First-out (FIFO) buffer implementation.
- Deep pipeline: A Pareto study investigated the pipeline CORDIC architecture for SF implementation using the *LeNet* and *TensorFlow* Convolution Neural Network (CNN) model. It was determined that four pipeline stages for Hyperbolic Vectoring mode (exponential operation) and five pipeline stages for Linear Rotational mode (division operation) returned better performance than iterative CORDIC-based implementation with less than 2% accuracy loss.
- Class Adjustable: The presented SF design is class adjustable, allowing for multi-class classification. The proposed SF can be configured for various output classes ranging from 10 to 1024, enhancing the operation of many DNNs.

The proposed SF has been validated for accuracy using software simulation on various CNN models. Further, the proposed design has been implemented on a Xilinx-Zynq Field Programmable Gate Array (FPGA) board. Additionally, the design has also been verified using an Application-Specific Integrated Circuit (ASIC) implementation at the 45nm technology node. The software evaluation reveals that the proposed design achieved an accuracy of 98.83%, which is only 0.15% lower than the single-precision software-based implementation in Python tested on a CNN model for the MNIST dataset [20]. The hardware implementation results demonstrate that the proposed design has reduced the area by around 15% compared to the architecture proposed in [12], and it also shows better power efficiency. Furthermore, the proposed design exhibits nearly 5× higher throughput performance than the iterative-CORDIC evaluation and is more efficient than the state-of-the-art SF implementations. The outline of the paper is as follows. Section II explains related work and motivation, Section III discusses the realization of the activation function using CORDIC architecture. Section IV explains the performance-centric pipelined architecture of SF. Section V presents the experimental evaluations, Section VI covers the Pareto-point extraction for limiting the pipeline stages, the experimental results, and comparisons. In the end, Section VII provides the article's conclusion.

II. RELATED WORK AND MOTIVATION

This section has covered previous efforts towards achieving accurate and efficient hardware implementation of SF for use in DNN models, as well as the rationale behind its implementation. One specific type of SF, known as normalized exponential or naive probabilistic SF, is defined in Equation 1, where x' represents a j'-dimension input vector and $SF(x_i)$ represents the output probability for the i^{th} element. Therefore, for classification tasks involving 'j' classes, SF can be

utilized to classify the probability of occurrence for each class.

$$\mathrm{SF}(x_i) = \frac{\mathbf{e}^{x_i}}{\sum_{j=1}^{N} \mathbf{e}^{x_j}} \tag{1}$$

Eq. 1 mathematically demonstrates the primary segments in SF, such as exponential evaluation followed by successive addition and division. Thus, the majority of computations are required for exponential and division operations.

A. RELATED WORKS

Researchers in [8], [9], [10], [11], [12], [13], [14], [17], [19], [21], and [22] have proposed two categories of SF architecture that focus on the trade-off among hardware area, power, latency, and accuracy, with emphasis on the optimized implementation of exponential and division blocks. The authors have mainly focused on the accuracy of SF operation [8], [9], [10], [12], [21] and utilized various methods such as Piece-Wise Linear (PWL) approximation [10], Stochastic Computation (SC) [21], and Taylor series expansion [23] for accurate architecture. The maximum mean absolute error (MAE) has been observed for [21], i.e., on the order of magnitude of e^{-2}, and on the order of e^{-4} for [10]. Although a PWL approximation method to implement exponential units requires many resources at higher bit precision, SC incurs a significant propagation delay to improve accuracy. In [5], a base-split calculation method has been proposed to implement exponential unit and bit-shift and subtract for division operation, showing a remarkable decrease in logic propagation delay and MAE in the order of magnitude of e^{-8}. The main disadvantages of these approaches are their low throughput and high complexity at high precision computation.

Secondly, researchers have also investigated the optimization of SF architectures for resource efficiency and time-constraint considerations [9], [11], [12], [13], [14], [17], [19]. These works have focused on modifying SF using log-sum-exp-trick [9], [11], [12], [22] for the Exponential Unit (ExU) and using the logarithmic unit in place of the Division Unit (DiU). In [9] and [12], the authors implemented a resource-efficient yet accurate SF, but on-chip power increased exponentially with an increase in the number of inputs in [9]. In [12], a high throughput design has been implemented, but due to 4-stage pipelined computations, logic propagation delay increased. However, through pipelining, the intermediate block propagation delay can be reduced in such architectures [11] with a slight increase in the hardware area. Similarly, Wang et al. in [22] implemented the ExU using a shifter, two adders, and advanced constant multiplier blocks, and the DiU was implemented using LOD (Leading One Detector) and shift blocks. The authors reported a high throughput SF model at the cost of greater on-chip power. In addition, Marchisio et al. in [24] provided a naive approximate softmax function with an area and power-efficient architecture targeting lower precision Natural Language Unit (NLU), specifically for the CapsNet model. As the architecture occupies a small hardware area and lower dynamic power, the logic propagation delay of the overall architecture is quite high. This makes the architecture well-suited for lower precision, resource, and power constraint applications.

The iterative CORDIC algorithm is a suitable choice for implementing an area-efficient non-linear function [17], [19]. However, due to the recursive nature of CORDIC, throughput is a significant concern in such architectures. In [14], the authors implemented an SF design using an iterative CORDIC algorithm for exponential function evaluation. While the design provides an area-efficient architecture, it suffers from inferior logical propagation delay. This architecture's major drawbacks are high computational power and low throughput. Thus, a power-efficient logic architecture with high operating frequency is highly desirable.

B. MOTIVATION

SF is extensively used in several DNN models [25], [26], [27] for the classification layer, making a hardware-efficient and performance-optimized SF crucial for high-speed, power-efficient classifier models at the edge. Additionally, Transformers, a type of neural network architecture that has gained popularity, often use SF activation in each layer. However, ExU and DiU, being the most resource-intensive and power-hungry blocks in SF, increase hardware complexity when used multiple times within the model [26]. The CORDIC algorithm has been proposed to address power and area-efficient architecture in the design. ExU and DiU, which consume a lot of resources, can be realized using the iterative CORDIC algorithm in different modes. However, the recursive nature of the CORDIC algorithm results in longer computational delays and, consequently, low throughput. Parallel processing and pipelining are two popular techniques employed to enhance the throughput of any computing system [15]. This paper proposes a performance-centric pipeline CORDIC architecture for efficient and improved SF implementation. Each pipeline stage in the CORDIC architecture is identical and independent of each other; thus, pipelining can reduce overall operational latency. However, more pipeline stages are required to achieve sufficient output accuracy, which subsequently leads to area and power overhead.

Accuracy deviation has been studied by varying pipeline stages of ExU and DiU implementation. It manifested that for more than 4 and 5 stages for ExU and DiU respectively, model classification accuracy for different datasets such as MNIST and CIFAR-10 is almost optimal when implemented on the CNN model [20]. Furthermore, bit-precision plays a vital role in implementing the neural model; higher bit precision proffers higher classification accuracy yet leads to more hardware area. It has been observed that inference accuracy of SF for float16 bit precision gave optimal results [11], and a similar conclusion has been made for fixed 16-bit precision representation in [12]; thus, a similar representation is considered for the proposed work as represented in FIGURE 1. However, higher bit precision computation is desirable to achieve higher accuracy. Thus, the proposed architecture works well

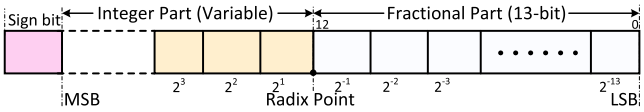

FIGURE 1. The data representation is for a signed fixed-point number that has a 13-bit fractional part and a variable-length integer part.

TABLE 1. Generalized CORDIC algorithm.

m	Rotational	Vector
Circular 1	$X_n = K_c(X_0 cosZ_0 - Y_0 sinZ_0)$ $Y_n = K_c(Y_0 cosZ_0 + X_0 sinZ_0)$ $Z_n = 0$	$X_n = K_c\sqrt{X_0^2 + Y_0^2}$ $Y_n = 0$ $Z_{0+1} = Z_0 + tan^{-1}(\frac{Y_0}{X_0})$
Linear 0	$X_n = X_0$ $Y_n = Y_0 + X_0 \cdot Z_0$ $Z_n = 0$	$X_n = X_0$ $Y_n = 0$ $Z_n = Z_0 + (\frac{Y_0}{X_0})$
Hyperbolic -1	$X_n = K_h(X_0 coshZ_0 - Y_0 sinhZ_0)$ $Y_n = K_h(Y_0 coshZ_0 + X_0 sinhZ_0)$ $Z_n = 0$	$X_n = K_h\sqrt{X_0^2 - Y_0^2}$ $Y_n = 0$ $Z_n = Z_0 + tanh^{-1}(\frac{Y_0}{X_0})$

for higher bit precision with the least computational power, delay, and resource constraint applications. In this paper, a fixed-point representation is considered with 13 bits for the fractional part and a variable integer part, depending on the computation precision as depicted in FIGURE 1.

III. PROPOSED SOFTMAX FUNCTION ARCHITECTURE USING CORDIC ALGORITHM

A. INTRODUCTION TO BASIC CORDIC ALGORITHM

The CORDIC algorithm computes various mathematical relations using trigonometric calculations by transforming planar coordinates into rotational coordinates. It performs iterative computation based on the step-by-step pseudo rotation of vector to enumerate arithmetic calculations and mathematical functions [28]. There are various modes in CORDIC to solve a variety of mathematical relationships as summarised in TABLE 1. CORDIC architecture is formulated to operate in two modes: Vectoring and Rotational, where each of these modes operates in three planar coordinates – Circular, Linear, and Hyperbolic.

The functions (including trigonometric, square root, hyperbolic and its inverse, and other basic mathematical calculations) can be performed by Rotational and Vectoring mode CORDIC algorithm in different planar coordinates as represented in TABLE 1.

The CORDIC algorithm is based on pseudo rotation, a scaled form of actual rotation involving X, Y, and Z variables. Based on the mode of operation to be performed, these variables are initiated where X and Y represent coordinates of pseudo rotation and Z keeps track of the angle at which the vector is being rotated [19]. Taking a common scaling factor K and CORDIC equations in pseudo rotation for all modes of trajectories are then formulated as:

$$X_{i+1} = X_i - Y_i \cdot d_i \cdot tan\ \alpha_i \tag{2a}$$
$$Y_{i+1} = Y_i + X_i \cdot d_i \cdot tan\ \alpha_i \tag{2b}$$
$$Z_{i+1} = Z_i - d_i \cdot \alpha_i \tag{2c}$$

where, X_i, Y_i *and* Z_i are variables values at i^{th} iterations. Further, α_i is the rotation angle in radians at each iteration i $\in \{1, 2, 3, \ldots n\}$. Also, α_i is represented as E_i considered as memory element for i^{th} iteration. In Linear, Circular and Hyperbolic coordinates E_i is 2^{-i}, $tan^{-1}(2^{-i})$ and $tanh^{-1}(2^{-i})$ respectively.

The unified form of the CORDIC algorithm is reformulated by Walther [15], which is suitable for performing circular, linear, and hyperbolic operations. CORDIC performs all mathematical operations shown in TABLE 1 using simple logic blocks such as adder/subtractor, multiplexer, barrel shifter, and memory elements [17]. The CORDIC trigonometric equations in pseudo rotation converge to linear form for the hardware implementation, as shown below in Eq. 3.

$$X_{i+1} = X_i - m \cdot d_i \cdot Y_i \cdot 2^{-i} \tag{3a}$$
$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{3b}$$
$$Z_{i+1} = Z_i - d_i \cdot E_i \tag{3c}$$

where mode $m \in \{0, 1, -1\}$ indicates a linear, circular, and hyperbolic coordinate system, respectively. Further, $d_i \in \{1, -1\}$ shows the rotation direction for i^{th} iteration which is *sign* Z_i for rotational mode and $-(sign\ X_i \wedge sign\ Y_i)$ for vectoring mode operations [19]. As used in TABLE 1, K is the scaling factor in the pseudo rotation of vector; it is a product of K_i for i iterations. Where K_i is $\cos(\alpha_i)$ and $\cosh(\alpha_i)$ for Circular mode and Hyperbolic mode, respectively, and the product converges in i^{th} iterations [17]. Scaling factors decreases monotonously at each iteration; it converges to $K_h \sim 0.8281$ for hyperbolic coordinate and $K_c \sim 1.6467$ for circular coordinate [15], [28]. The post-processing remarks of different modes of operation result in exponential, logarithmic, and hyperbolic functions. Utilizing CORDIC Hyperbolic Rotational (HR) and Linear Vectoring (LV) modes for exponential and division operations, respectively, have been performed.

This work has used HR mode for exponential function evaluation and LV mode for division operation for proposed SF implementation. Further, hardware architectural optimization has been done to make an area and power-efficient design as discussed in detail in section IV. Additionally, the details about CORDIC computations and Scaling factor evaluation can be found in [15] and [17].

B. SOFTMAX FUNCTION EVALUATION USING CORDIC ALGORITHM

The recursive CORDIC algorithm in a different mode of operation converges to perform various mathematical computations, as illustrated in TABLE 1. The CORDIC-based softmax implementation is area and power-efficient at the cost of lower throughput. We have designed an efficient softmax function using CORDIC architecture. Observing the expression represented in Eq. 1, we have realized exponential computation and division operation using CORDIC in hyperbolic and linear mode, respectively.

$X_{i+1} = X_i + d_i*Y_i*2^{-i}$;	$Y_{i+1} = Y_i + d_i*X_i*2^{-i}$;	$Z_{i+1} = Z_i - d_i*\tanh^{-1}2^{-i}$;
Initial	i = 1 (1st iteration)	
$X_0 = 001.0011010100011_2$ $= 1.2074_{10}$	$X_1 = 001.0011010100011 + di* 000.0000000000000$ $= 001.0011010100011_2 / 1.2074_{10}$	Yo right shift by i bits; Lost bit
$Y_0 = 000.0000000000000_2$ $= 0.0000_{10}$	$Y_1 = 000.0000000000000 + d_i* 000.1001101010001$ $= 000.1001101010001_2 / 0.6036_{10}$	Xo right shift by i bits; Lost bit
$Z_0 = 000.1000000000000_2$ $= 0.5000_{10}$	$Z_1 = 000.1000000000000 - d_i* 000.1000110010100$ $= -000.0000110010100_2 / -0.0493_{10}$	

FIGURE 2. The arithmetic calculations that are involved in the first iteration of CORDIC when it is operating in the hyperbolic mode.

$X_{i+1} = X_i$;	$Y_{i+1} = Y_i + d_i*X_i*2^{-i}$;	$Z_{i+1} = Z_i - d_i*2^{-i}$;
Initial	i = 1 (1st iteration)	
$X_0 = 010.1000001010001_2$ $= 2.5100_{10}$	$X1 = 010.1000001010001_2$ $= 2.5100_{10}$	
$Y_0 = 001.1001111110110_2$ $= 1.6238_{10}$	$Y_1 = 001.1001111110110 + d_i* 001.0100000101000$ $= 000.0101111001110_2 / 0.3689_{10}$	Xo right shift by i bits; Lost bit
$Z_0 = 000.0000000000000_2$ $= 0.0000_{10}$	$Z_1 = 000.0000000000000 - d_i* 000.1000000000000$ $= 000.1000000000000_2 / 0.5000_{10}$	

FIGURE 3. The arithmetic calculations that are involved in the first iteration of CORDIC when it is operating in the linear mode.

The implementation details for the Exponential Unit (ExU) and Division Unit (DiU) are examined below:

1) EXPONENTIAL FUNCTION EVALUATIONS USING CORDIC

The CORDIC algorithm in Hyperbolic Rotation Mode can be used to implement the hyperbolic trigonometric operation. The generalized equations in a hyperbolic mode of operation are shown in Eq 4. In this mode, hyperbolic trigonometric functions sinh and cosh can be evaluated using hyperbolic coordinates. The CORDIC algorithm sets variable m to -1 and divides results by scaling factor K_h, i.e., 0.8281. To calculate *sinh* and *cosh*, the hyperbolic rotational mode of the CORDIC algorithm have used. At last, division by scaling factor can be eliminated by dividing initial variables, thus setting X_0 to 1/Kh, Y_0 to 0, and setting the input to Z_0. The rotations are selected such that rotation scaling factors are negative power of 2, i.e., $\alpha_i = \tanh(2^{-i})$. Therefore, from TABLE 1, it is observed that output X_n, Y_n and Z_n converges as: X_n to $\cosh Z_i$; Y_n to $\sinh Z_i$; and Z_n to zero (i.e. $Z \rightarrow 0$).

$$X_{i+1} = X_i + d_i \cdot Y_i \cdot 2^{-i} \tag{4a}$$
$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{4b}$$
$$Z_{i+1} = Z_i - d_i \cdot \tanh^{-1} 2^{-i} \tag{4c}$$

For the first iteration, i.e. $i = 1$ CORDIC computation is shown in FIGURE 2. After n^{th} iterations, it computes $\sinh(Z)$ and $\cosh(Z)$ that can be used for further exponential function evaluation using Eq. 5 [17].

$$f(Z) = exp(Z) = e^Z = \sinh(Z) + \cosh(Z) \tag{5}$$

2) EVALUATION OF DIVISION OPERATOR USING CORDIC

To perform division using CORDIC alogrithm, the vector mode of operation with linear coordinates can be used as shown in TABLE 1. Rotation operations in linear coordinates are derived using Eq. 3 by setting the mode variable (m) to 0 and memory element $E_i = 2^{-i}$. After setting the parameter's value in the revised form, the linear vectoring mode output is expressed in Eq. 6. To perform *division* operation, this mode of CORDIC algorithm is being used. On setting X_0 as a divider, Y_0 as dividend and Z_0 as Zero, after n iterations, Z_n holds the quotient Y_0/X_0. FIGURE 3 shows the first CORDIC iteration $i = 1$ where the divider is set to X_0 as the sum of exponentials for all the neurons' output from the previous layer, dividend in Y_0 as exponential for a particular neuron's output, and Z_0 is set to 0, after n iterations, Z_n holds the quotient Y_0/X_0, as SF estimation for each neuron.

$$X_{i+1} = X_i \tag{6a}$$
$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \tag{6b}$$
$$Z_{i+1} = Z_i - d_i \cdot 2^{-i} \tag{6c}$$

The computation of the CORDIC algorithm for different coordinates is valid in a specific convergence range, as explained in [29]. For Hyperbolic coordinate input: [-1.1182,1.1182]; and for Linear coordinates input: [-1,1] are the range of convergence corresponding to Eq. 2 of CORDIC algorithm. Thus, to ensure the input converges, the SF has been normalized to the range of [-1,1] in the proposed work. Using Eq. 6, Z_{i+1} computes the Softmax output when $Y \rightarrow 0$. The detailed evaluation and consideration of the model have been discussed in Section IV.

An iterative CORDIC algorithm should iterate for (i+1) iterations for an i-bit output precision. However, each iteration incurs some propagation delay on account of adder/subtract, barrel shifter, multiplexer, and feedback register with conventional CORDIC architecture [16]. Therefore, on increasing bit-precision to maintain maximum computation accuracy, the latency and on-chip area rise. Furthermore, in the SF calculation, both exponential and division operations are resource-intensive, especially for high-precision hardware implementation. To handle this bottleneck, we proposed a performance-centric SF where we used pipeline stages of CORDIC architecture.

IV. EVALUATION OF ENHANCED PERFORMANCE CORDIC-BASED SOFTMAX FUNCTION

Any computing system's performance can be enhanced using parallel processing and pipelining techniques. Each iteration of the CORDIC algorithm is mutually independent, and thus, it can process parallel. However, the pipeline stages come with an on-chip area and power overheads. The Pareto study is perceived to determine the optimal number of pipeline stages in the ExU and DiU to improve the performance of complete SF design. The proposed performance-centric architecture for the SF is shown in FIGURE 4. The architectural block diagram delineates pipelined CORDIC-based softmax function. Exponential Unit (ExU) and Division Unit (DiU) are constructed using P-stage and Q-stage pipelined CORDIC computation in Hyperbolic Rotational and Linear Vectoring Mode, respectively.

The computations are performed in two parts manifested as part-1 and part-2. Part-1, includes ExU, two adders, a De-Multiplexer (DeMUX), and a memory element (E_i) outlined with pink color dotted line in FIGURE 4. An input of the softmax function is the initial fixed-point value of variable Z_{in}[N:0], which is fed to the ExU at the input clock rate. The hyperbolic trigonometric functions cosh and sinh obtained from pipelined CORDIC architecture, and the exponential function (of input Z_{in}) is determined by summing them as shown in Eq. 5.

All inputs of the softmax layer, which are usually the output classes of the classification network, are sequentially processed by the ExU unit, and their exponents are successively stored in FIFO as shown in FIGURE 4. At the same time, Adder2 accumulates exponential values of each input class ($Z_1, Z_2, \ldots Z_n$). The control signal `Run_in` is used to control DeMUX operation and signal `Run_out` is generated to notify the completion of part-1. `Run_in` is the select line for DeMUX. When `Run_in` is 'high,' the exponential values of successive inputs Z_{in} are summed; otherwise, the sum is fed to the DiU unit. Thus, `Run_in` is 'high' until exponential for all the inputs are calculated, thereafter `Run_out` gets 'high'. Part-2 incorporates division operation in DiU and memory block. The sum of exponential from DeMUX and the exponential of input from FIFO are the input to the DiU as shown with a green dotted line border in FIGURE 4. Here, `Run_out` signal of part-1 goes

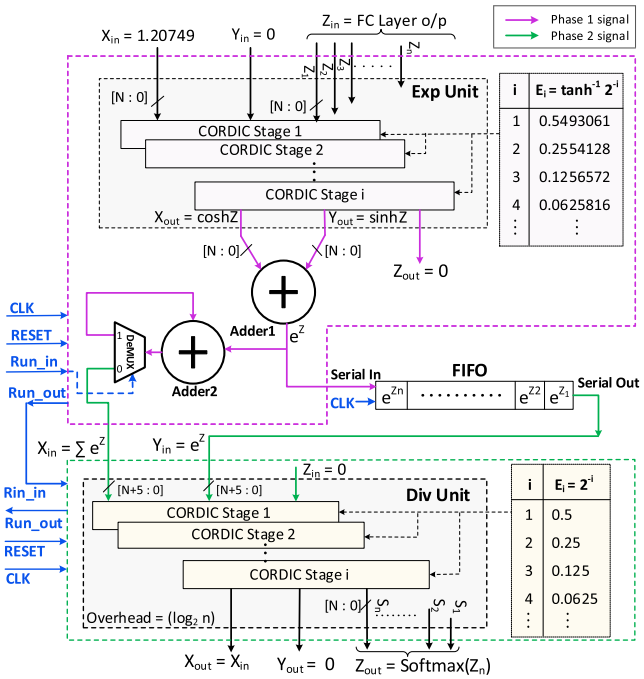

FIGURE 4. The block-level architecture of the proposed Softmax function, which utilizes the CORDIC algorithm.

to `Run_in` of phase 2. Also, until all softmax calculations are performed `Run_in` remains 'high', thereafter a 'high' `Run_out` from phase 2 shows the end of softmax operation. The performance-centric design of pipelined CORDIC for ExU and DiU is discussed in Section IV-A and IV-B.

A. PERFORMANCE-CENTRIC PIPELINED ARCHITECTURE FOR EXPONENTIAL UNIT

Exponential Unit (ExU) used in the proposed architecture depicted in FIGURE 4 consists of 'P' pipelined stages of CORDIC architecture operating in Hyperbolic Rotational mode. Each pipeline stage represents one iteration in CORDIC computation. From Hyperbolic Rotational mode CORDIC equation Eq. 4 it follows that next X_{i+1} is calculated by increasing or decreasing present X_i by the shifted Y_i, next Y_{i+1} by increasing or decreasing present Y_i by shifted X_i, and next Z_{i+1} by increasing or decreasing Z_i by $\tanh^{-1}(2^{-i})$ where the sign of Z_i control the operations. Thus, each CORDIC stage includes a fixed shifter, add/sub-block, ROM/registers to store E_i, and pipeline registers connecting two consecutive stages as shown in FIGURE 5. Keeping in mind the convergence range in Hyperbolic mode operation [29], input to SF (Z_{in}) is normalized in the range $[-1, 1]$ so sinh and cosh of input values are obtained in the range $[-e/2, e/2]$. The output of CORDIC unit after summing the sinh and cosh terms converges to $[0, e]$. Considering this, the ExU is represented in a 16-bit fixed$\langle 16, 13\rangle$ format with an extra sign bit. Overflow has been saved during the accumulation of exponentials of input; the accumulator must be widened by at least $\log_2\ n$ bits. For at most 1000 output classes, additional 10 bits in an input to the SF are

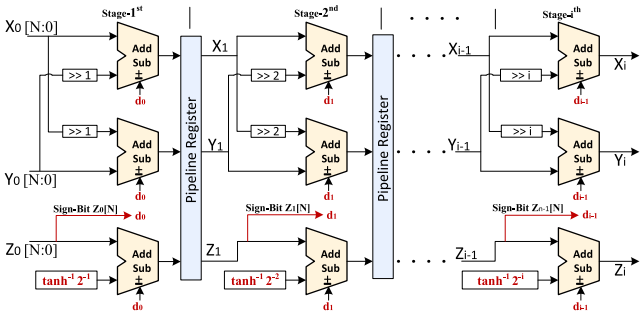

FIGURE 5. Exponential Unit including i-stage Pipeline CORDIC architecture used for N-bit precision.

TABLE 2. Four stage hyperbolic mode CORDIC calculation using fixed ⟨16, 13⟩ representation.

i	d_i	$X_{i+1} \rightarrow$ coshZ	$Y_{i+1} \rightarrow$ sinhZ	$Z_{i+1} \rightarrow 0$
Pipeline	signZ_i	Initial Conditions / Input		
Stages		X_0 = 1/K = 1.2074	Y_0 = 0	Z_0: input = 0.5
initial	-	001.0011010100011	000.0000000000000	000.1000000000000
1	1	001.0011010100011	000.1001101010000	-000.0000110010100
2	-1	001.0000111001111	000.0100110101000	000.0011010011000
3	1	001.0001100000100	000.0110111100010	000.0001010010011
4	1	001.0001111100011_2	000.1000000010011_2	000.0000010010010
		1.121459_{10}	0.502319_{10}	$0.017_{10}(\approx 0)$

required; thus, the exponential sum is represented in fixed-point ⟨26, 13⟩ format.

CORDIC in Hyperbolic mode with an initial condition evaluates the sinh and cosh function, which iterative calculation is shown in TABLE 2. A performance-centric evaluation using four pipeline stages for ExU has been demonstrated. The calculation has elaborated for the first stage for standard ⟨16, 13⟩ fixed-point as represented in FIGURE 2. Moreover, similar evaluation for subsequent stages architecture shown in FIGURE 5 have elaborated in TABLE 2, where finally evaluated cosh and sinh on output port X_{i+1}[16:0] and Y_{i+1}[16:0] respectively. The exact hyperbolic calculation for input $Z_0 = 0.5$, $sinhZ_0$ and $coshZ_0$ is 0.502337 and 1.121416 respectively. Whereas, the proposed model return $sinhZ_0$ and $coshZ_0$ are 0.502319 and 1.121459 respectively, shown in TABLE 2. Although the mean deviation for 16-bit precision is nearly 0.00012% compared to the same 64-bit floating-point calculation result. Further, design is efficient regarding physical parameters such as area, power, and critical delay, as discussed in Section VI. The entire class's exponential output and the accumulated sum are fed to the division unit to predict a multinomial probability distribution.

B. PERFORMANCE-CENTRIC PIPELINED ARCHITECTURE FOR DIVISION OPERATION

Division Unit (DiU) shown in FIGURE 4 consists of Q pipelined CORDIC stages, and the green dotted line highlights its top-level architecture. Considering Linear Vectoring mode CORDIC equations that perform division operation, the pipeline stages are constructed as shown in FIGURE 6. On utilizing Eq. 6, X_{i+1} is evaluated by increasing or decreasing current X_i by factor of the shifted Y_i, next Y_{i+1} by

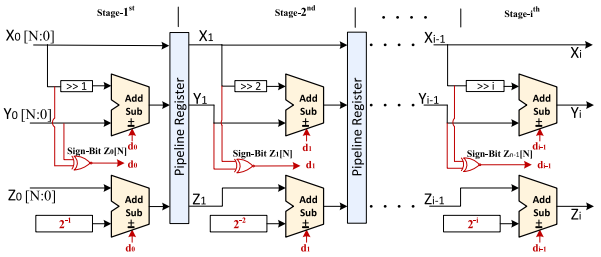

FIGURE 6. Division Unit including i-stage Pipeline CORDIC architecture used for N-bit precision.

increasing or decreasing current Y_i by factor of shifted X_i and next Z_{i+1} by increasing or decreasing Z_i by 2^{-i} where sign of X_i & Y_i control the operations. Hence, each stage of the pipeline structure includes a fixed shifter, add/sub-block, memory element to store E_i, and pipeline registers for storing the intermediate computation between the two pipeline stages. In SF implementation, the division is performed for the exponential value of each SF input and the accumulated sum of all exponential values as given in Eq. 1. Where exponential values obtained from ExU unit are in 16-bit Fixed ⟨16, 13⟩ format. The sum of exponentials requires extra bits to prevent accumulated value, thus represented in 26-bit Fixed ⟨26, 13⟩, i.e., 13-bits for the fraction part and 13 bits for the integer part. The additional overhead bits in the integer part of the sum of exponential depend on the output class, which is $\log_2(n)$, where n is the output class of the classification model.

Depending on the number of output class overhead bits decided, we have used ten overhead bits for 1024 classes; this makes the design class adjustable. Further, the sum of exponential and exponential values for each class is quantized to 21-bit Fixed ⟨21, 13⟩ format before it is given to the DiU. The resource utilization of DiU for higher bit precision is quite large. The computation in DiU is carried out in a 21-bit Fixed ⟨21, 13⟩ format.

The division computation for the first CORDIC stage in 16-bit Fixed ⟨16, 13⟩ format is described in FIGURE 3. Pipelined architecture enhances throughput, but it comes with an area overhead. As pipelining is used in this article, we have evaluated the necessary pipeline stages at which desirable accuracy is achieved. Complete illustration for 5-stages is compiled in TABLE 3. The output Z_{out}[21:0] converges to division of Y_0/X_0 after i iterations such that $Y_{i+1} \rightarrow 0$. Continuing ExU output from TABLE 2, the sum of hyperbolic trigonometric function ($sinhZ_0$ and $coshZ_0$) after four iterations returns a value 1.623778 (i.e., exponential for input $Z_0 = 0.5$). Let the overall sum exponentials for different SF inputs as 2.51.

The exact Linear mode calculation for input Y_0=1.623778 and X_0=2.51 undergoes division, and output Y_0/X_0 will be 0.64692. The output for five iterations of Linear Vectoring mode CORDIC returns a value 0.656250 in Z_{i+1} which is SF output for input 0.5, and Y_{i+1} nearly approaches 0 as shown in TABLE 3. Thus, a division operation is performed, where computed exponential values of individual SF input

are fed to Y_0 and the sum of exponential values of SF inputs to X_0; after i^{th} iterations, Z_{out} holds the SF output for given input values. The SF output Z_{out} is scaled down to Fixed $\langle 16, 13\rangle$ from Fixed $\langle 21, 13\rangle$ format. One can observe that the mean, standard deviation for 21-bit precision is nearly 1.44% compared to the exact 64-bit floating-point calculation. The proposed architecture shown in FIGURE 4 returns better physical design parameters such as area, power, and critical delay at the cost of insignificant accuracy loss, discussed in Section VI.

C. CLASS-ADJUSTABLE ARCHITECTURE

The classification class of a dataset plays a significant role in SF architecture constructed using its exact expression Eq 1. The model's memory requirement to save the exponential values varies with the input class. Generally, an SF architecture is designed for a fixed output class of dataset with limited LUT memory to keep minimum resource utilization in VLSI design. However, architecture with higher precision makes the model resource-intensive and output classes dependent. In the proposed SF architecture, input class (N) regulates the memory requirement for intermediate calculations, making it class-adjustable. The depth of FIFO gets adjusted in variation with the output class in a dataset. Considering the number of CORDIC stages and the arithmetic bit precision, the proposed SF architecture has calculated accurately up to 1024 classes of a dataset. To implement the proposed SF for beyond 1024 classes, the Pareto analysis for finding the number of CORDIC pipeline stages and a variation of fixed-point notation can be similarly analyzed.

D. TIMING ANALYSIS OF PROPOSED PIPELINED ARCHITECTURE

The overall timing analysis of complete architecture is discussed in this section. Here, for x SF inputs, we require D_{ExU} clocks delay for ExU execution and D_{DiU} clocks delay for DiU execution, and L denotes the latency of architecture. All the exponential operations for SF input, followed by accumulation, are performed in the first phase. The D_{ExU} delay incurred during this phase due to P staged ExU is shown in Eq 7. In the second phase, the division unit operates and incurs a clock delay of D_{DiU} due to Q staged DiU shown in Eq 7. The ExU and DiU units can operate in pipelined manner for two different sets of computations, thus increasing the overall throughput of the model. The latency L of the architecture is the sum of ExU and DiU clock delays. In contrast, the delay of SF output is equal to D_{DiU} due to mutual exclusivity between the two phases. The total required clock cycles (T_R) for x SF outputs is equal to the overall latency of the architecture and can be estimated using Eq. 7.

$$D_{ExU} = x + P \tag{7a}$$
$$D_{DiU} = x + Q \tag{7b}$$
$$L = D_{ExU} + D_{DiU} \tag{7c}$$
$$T_R(x) = L = (x + P) + (x + Q) \tag{7d}$$

TABLE 3. Five stage linear mode CORDIC calculation using $\langle 21, 13\rangle$ representation.

i	d_i	$X_{i+1} \rightarrow X_0$	$Y_{i+1} \rightarrow 0$	$Z_{i+1} \rightarrow Y_0/X_0$
Pipeline Stages	sign bit X_iY_i	Initial Conditions / Input		
		X_0: sum = 2.51	Y_0: exp = 1.623778	Z_0 = 0
initial	-	010.1000001010001	001.1001111110110	000.0000000000000
1	-1	010.1000001010001	000.0101111001100	000.1000000000000
2	-1	010.1000001010001	-000.0100001000111	000.1100000000000
3	1	010.1000001010001	000.0000111000010	000.1010000000000
4	-1	010.1000001010001	-000.0001101000010	000.1010100000000
5	1	010.1000001010001_2	-000.0000010111111	000.1010100000000_2
		2.51_{10}	$0.023_{10}(\approx 0)$	0.656250_{10}

* Above calculations are performed for Fixed $\langle 21, 13\rangle$ format, where 5 integer bits from left are all Zeros. Thus, $\langle 16, 13\rangle$ representation is displayed.

The SF architecture takes initial 'x+P' clocks for exponential evaluation and 'Q' clocks for the division unit to generate the first output for SF. Therefore, the first output takes 'x+P+Q' clock cycles, and after that design produces output at every clock cycle. The total required clock cycles have been enumerated in Eq. 7. The frequency,f of the complete architecture is due to the delay incurred by the single-stage operation of DiU. Thus, Throughput (T) is calculated using Eq. 8, where k is the number of operations per clock.

$$T = f \times k \tag{8}$$

V. EXPERIMENTAL EVALUATIONS

We conducted experiments to investigate the design parameters through Pareto analysis for improving the performance and efficiency of SF used in DNN accelerators. The experimental setup involved both software and hardware-based implementations. Firstly, we validated the proposed model using the QKeras Version 2.4 library for 16-bit fixed-point arithmetic in Python. Secondly, we described the proposed SF model in Verilog-HDL language and simulated it at the Resistor Transistor Logic (RTL) level using ModelSim simulator. We later synthesized it at 45nm technology using Synopsis Design_Compiler and presented the post-synthesis results. We further performed FPGA synthesis and implementation using the *Xilinx − Vivado* 17.4 tool. The evaluations included:

1) We have evaluated the accuracy using the CNN model [20] for MNIST and CIFAR-10 datasets. To evaluate whether the proposed SF is accurate enough, we compare the accuracy of CORDIC-based architecture in Python with standard *TensorFlow* computation [30]. Furthermore, we simulated fixed-point behavior by quantizing the SF's operations. Hence, our CORDIC-based python implementation replicated the hardware design in terms of accurate evaluation.
2) To enhance the throughput and energy consumption of CORDIC-based SF, pipelined CORDIC stages were used in the implementation of ExU and DiU units. In order to determine the sufficient number of pipelined stages for optimum accuracy, we extracted the Pareto points using the CNN model. The error deviation from the exact computation was also analyzed by mean

TABLE 4. Classification accuracy(%) for quantized and unquantized CNN model with Tensorflow's softmax model and proposed SF.

Tensorflow-based CNN Model				
	For MNIST dataset		For CIFAR-10 datasets	
Precision	Tensorflow	Proposed	Tensorflow	Proposed
32-bit	98.98	98.79	87.17	86.42
16-bit	98.95	98.83	85.05	84.85
LeNet Model				
16-bit	99.2	98.97	66.60	66.10

TABLE 5. Classification accuracy (%) for quantized ResNet-18 model with Tensorflow's softmax model and proposed SF.

ResNet-18 Model				
	For MNIST		For CIFAR-100 datasets	
Precision	Tensorflow	Proposed	Tensorflow	Proposed
32-bit	99.42	99.17	66.63	64.12
16-bit	99.02	98.89	60.90	57.92

average error (MAE) and mean square error (MSE) calculations.

3) Based on previous evaluations, we have selected the number of pipelined stages for each CORDIC unit. We have achieved post-implementation performance parameters for the proposed CORDIC-based architecture for the Zybo board and were compared with previous works.
4) In order to evaluate ASIC compatibility, we simulated the post-synthesis results for 45nm technology and compared the physical design parameters like area, energy, and frequency of operation with other state-of-the-art implementations.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

This section describes software validations of enhanced performance CORDIC engine-based SF and its hardware implementation performance parameter. Using a hardware description language (HDL), model simulation results have been evaluated for 16-bit Fixed $\langle 16, 13\rangle$ precision. In the *Python Version*3 platform, two CNN models, LeNet and ResNet, were implemented using Fixed $\langle 16, 13\rangle$ precision and trained on MNIST and CIFAR-10 datasets. The CNN model has been designed using Python and customized for the proposed SF at the classification layer. The network accuracy is evaluated for MNIST and CIFAR-10 datasets. Further, the SF hardware implementation was performed on a *Zybo* board with a Xilinx XC7z010 device, and the resource utilization and timing analysis were observed. The ASIC post-synthesis results for physical parameter analysis were also evaluated and compared with currently existing designs. A detailed of the experimental evaluation and explored results are given in the following subsections.

A. PARETO ANALYSIS FOR IDENTIFYING PIPELINE STAGES

In order to analyze the impact of the number of pipeline stages on the accuracy of deep neural networks, a Pareto analysis was performed. At first, training and validation were performed for a CNN-based model [20] on MNIST and CIFAR-10 datasets in the *Python* platform. The model was trained using the TensorFlow softmax function at the classification layer and validated the classification accuracy for both datasets as shown in TABLE 4. Similarly, the validated accuracy has been recorded for the proposed SF at the classification layer, considering different pipeline stages in the Exponential Unit(ExU) and Division Unit(DiU).

FIGURE 7 and FIGURE 8 depict the inference accuracy variation on unquantized and quantized models for different sets of pipeline stages in MNIST and CIFAR-10 datasets, respectively. The graph concludes that the accuracies almost converge to their maximum value after 4 and 5 pipeline stages for ExU and DiU respectively. The achieved maximum accuracy is almost equal to the accuracies of TensorFlow-based implementation. As accuracy improves with an increase in the number of pipeline stages, the amount of hardware resources, as well as energy consumption and latency, increases at a higher rate. For an area, energy, and throughput-efficient SF architecture, it was concluded that a combination of 4 and 5 pipeline stages in ExU and DiU respectively is the best choice for performance-centric, optimum architecture. Also, to analyze the effect on accuracy due to quantization, the computation was performed for unquantized and 16-bit quantized CNN models as shown in TABLE 4. Due to computing quantization, there is less than 2% loss in accuracy validation for MNIST and CIFAR-10 datasets. The SF architecture has also been verified in the 16-bit quantized LeNet model for MNIST and CIFAR-10 datasets. To further verify the SF architecture's efficacy on a dataset with more classes, the CIFAR-100 dataset has been tested on the ResNet-18 model. The inference accuracy using the proposed SF is 57.92% i.e. 3% loss with respect to TensorFlow-based implementation. Although, the significant accuracy loss of 2-3% for CIFAR-100 datasets is not desirable in applications where accuracy is of utmost importance. Nonetheless, in the context of physical performance parameters for low power and high throughput applications, such a loss may be tolerable as it can result in significant savings in hardware resources and power consumption. Also, with increased classes (beyond 1024), a higher precision implementation and more CORDIC stages will be required to increase the classification accuracy. Thus once again, Pareto analysis needs to be performed.

Here, the proposed CORDIC-based SF design is first prototyped using Python Library, and the Tensor framework is used to evaluate the performance accuracy of the CNN model. Besides, RTL of the SF design is implemented using Xilinx-Vivado for its functional verification, and, through simulation, it validated the ASIC compatibility using the Seimens-ModelSim simulator. We set the software environment similar to the hardware implementation for the

TABLE 6. Hardware utilization and performance parameter at different bit precision for pipeline CORDIC base proposed softmax function.

Fix Point	Resource Utilization			Power			Critical Delay			Energy
Precision	LUT	FF	BRAMs	Signal (mW)	Logic (mW)	Dynamic (mW)	Logic (ns)	Signal (ns)	Total (ns)	PDP (pJ)
32-bit	681	745	0.5	7.020	6.120	18	3.854	3.486	7.340	96.448
16-bit	427	369	0.5	2.240	2.520	7	2.993	3.500	6.443	30.668
8-bit	256	224	0.5	1.588	1.764	5.880	2.662	3.316	5.972	20.182

TABLE 7. Hardware implementation results of iterative and pipeline (4,5) CORDIC based SF for fixed⟨16, 13⟩.

Parameters	Exp. Unit	Div. Unit	Softmax Function	Exp. Unit	Div. Unit	Softmax Function
LUT	140	75	325	275	99	427
FF	126	89	232	138	120	369
BRAMs	–	–	0.50	–	–	0.50
Power (mW)	0.91	0.63	2.10	8.50	3.08	4.76
Delay (ns)	1.6	1.4	3.13× 9*	2.77	3.65	6.44
PDP (pJ)	0.252	0.882	59.157	27.62	14.48	31.47
Throughput (GOPs)	–	–	0.036	–	–	0.685
Performance (GOPs/W)	–	–	16.91	–	–	143.89

∗Number of clocks required to compute final output.

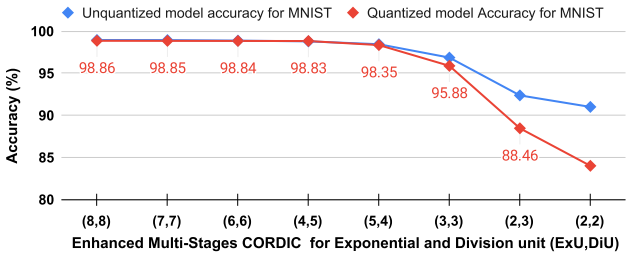

FIGURE 7. MNIST validation accuracy for CNN model developed in [20] with proposed Softmax Function with different CORDIC stages in ExU and DiU.

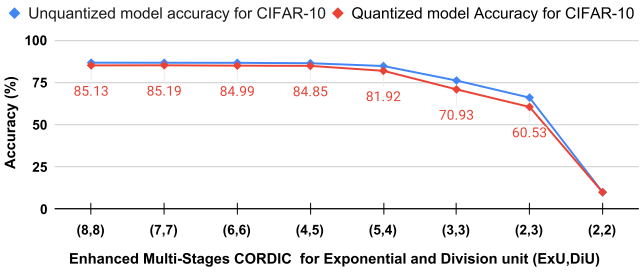

FIGURE 8. CIFAR-10 validation accuracy for CNN model developed in [20] with proposed Softmax Function with different CORDIC stages in ExU and DiU.

SF design; the python-based evaluation faithfully replicates the results for the hardware implementation.

Furthermore, the proposed SF architecture's Mean Square Error (MSE) was evaluated using 16-bit precision in which three integer bits and 13 fractional bits are considered. In the experiment, 1000 random values in the range [-1,1] were used as input; the values were uniformly distributed over the range. MSE for the proposed pipeline stage combination is in the order of e^{-5}. Increasing the number of pipeline stages in the proposed model can reduce the error by sacrificing efficient hardware area utilization and Power Delay Product (PDP).

B. HARDWARE IMPLEMENTATION RESULTS FOR PIPELINED AND ITERATIVE CORDIC-BASED SF

This section brings a comparison between pipeline and iterative CORDIC-based SF implementation. Both architectures were developed using *Verilog* HDL, and resource utilization has been reported. The hardware-implemented architectures were done on Xilinx-*Zybo* board at different bit precision. For establishing a correct comparison, iterative CORDIC-based architecture has also been implemented on hardware and results have been compared with proposed ExU, DiU, and SF architectures.

Firstly, hardware-efficient pipelined CORDIC-based SF with 4 and 5 pipeline stages for ExU and DiU, respectively, was implemented as explained in Section VI-A. SF is used as a classification function in many neural network models and is thus expected to have high classification accuracy. Whereas a higher precision computation returns better accuracy; therefore, SF implementation at a higher bit-precision representation is desirable. In order to observe the effect of bit-precision on the proposed model, physical parameter comparisons for signed 8-bit, 16-bit and 32-bit precision have been evaluated, as shown in TABLE 6. Consequently, for different precision, we have reported resource utilization, critical delay, and Power-Delay-Product(PDP), where power includes signal and logic power in TABLE 6. Resource utilization and PDP increase by 1.66× and 1.52×, respectively, when the precision increases from 8-bit to 16-bit. Furthermore, when precision is increased from 16-bit to 32-bit, resource utilization increases by 1.8×, and PDP increases by 3.065×. A significant rise in PDP when precision is from 16-bit to 32-bit is due to increased dynamic power at higher-bit precision. Considering accuracy and resource utilization to be optimum, signed 16-bit precision implementation is selected for further analysis. Similar state-of-the-art architectures [11], [12] have also considered 16-bit precision SF implementation as an efficient choice for energy, throughput, and resource-efficient design without any significant loss in accuracy. Resource utilization and energy metrics comparison of proposed SF with various state-of-the-art implementations at 16-bit precision is shown in TABLE 8.

TABLE 8. Performance parameters of proposed and state-of-the-art softmax function at different technology nodes.

Characteristics	[5]	[11]	[12]	[22]	[14]	Proposed
FPGA Parameters						
LUT	17870	1963	1858	2564	–	427
FF	16400	772	2086	2794	–	369
BRAMs	–	–	2.35	–	–	0.50
DSP	–	–	8	–	–	–
Frequency (MHz)*	–	294	500	436	–	685
ASIC Parameters						
Technology Node (nm)	65	45	28	28	90	45
Area (μm^2)	444858	220178	18392	15000	31150	58031
Throughput (G/s)	1	2	13.12	22.40	–	5
Total Logic Delay (ns)	0.50	800	0.31	–	2.39	0.10
Power (mW)	443	6.87	–	51.60	–	3.39
Energy-delay Product (pJ)	166.50	5520	–	–	–	14.04
Frequency (GHz)	1	0.25	1.64	2.80	–	5

∗ The frequency represents the speed of operation of architecture.

Secondly, we have implemented signed 16-bit precision as an optimum choice and compared the performance parameter of the proposed pipelined ExU, DiU, and CORDIC-based SF with their iterative counterparts as shown in TABLE 7. The SF with 4 and 5 pipelined CORDIC stages consumes 427 LUTs, whereas iterative CORDIC architecture utilizes 325 LUTs. Although the proposed SF design has increased resource utilization by 1.31× but achieved better performance metrics, i.e., lower PDP (1.88×) and higher throughput (19.03×) than the iterative process CORDIC-based SF implementation. The reason for 1.31× resources scaling is the absence of feedback registers, multiplexer, and barrel shifter, in the proposed architecture, unlike iterative CORDIC-based design. Thus, resource utilization has not increased proportionally with the number of pipeline stages. On the other hand, we enhanced the throughput by pipelining CORDIC stages. Consequently, the total critical delay for the proposed design has shown an almost 5× decrease than the iterative CORDIC-based design for an MNIST dataset as the iterative CORDIC-based SF design took 9× the delay for ten classes of data. TABLE 7 reports the results for iterative and pipeline CORDIC architecture. The edge AI application demands high-throughput and energy-efficient design. Thus, our proposed pipelined CORDIC-based SF function will be an optimum choice. Whereas iterative CORDIC-based SF will be useful in applications where resources are constrained.

C. PERFORMANCE PARAMETERS COMPARISON FOR FPGA AND ASIC IMPLEMENTATION

The hardware implementation depends on the targeted application, design requirements, cost, and the number of units that need to be manufactured. For that FPGA and ASIC implementation compatibility has been validated. This section discusses the proposed design's physical performance parameters and comparison with existing works in [5], [11], [12], [14], and [22]. Furthermore, the utilization impact of the proposed SF architecture is examined, and all comparisons are made for fixed 16-bit precision, as shown in TABLE 8.

The FPGA result shows the proposed design has lower LUT and FF utilization, which are 77% and 82% respectively, than the efficient method in the state of the art [11]. The proposed design has lower resource utilization due to its CORDIC-based implementation, and besides, we have limited the CORDIC stage by systematic evaluation of performance versus pipeline stages. However, limiting pipeline stages affects the insignificant accuracy loss, around 0.2% decrease in MNIST and CIFAR-10 classifications. The proposed design has used only 0.5 BRAMs as a FIFO to store the exponentials for each output class of data. Thus, we can see the benefits of the proposed SF architecture compared to previous work. Since the model is parameterized for the variable N of output classes (datasets) where the memory requirement for storing the intermediate data is variable. In order to improve the throughput performance, pipeline stages have been used, which come with area overhead compared to iterative architecture. However, it still outperforms many currently used SF models [5], [11], [12], [22] in resource utilization, PDP, and operating frequency of operation. A higher operating frequency for the proposed model depicts high computational speed.

Besides, the proposed architecture's experimental results validate the ASIC implementation. The physical parameters are evaluated at the 45*nm* technology node and compared with the state-of-the-art. The performance parameters at different technology nodes for state-of-the-art implementation are depicted in FIGURE 8. As we know by Moore's Law, on down-scaling the technology node, the silicon chip area halves successively from one technology node to another. Thus, we can make a fair comparison between different designs at various technology nodes from TABLE 8. As per the observed results at the 45nm technology node, when we scale our device from 45nm to 28nm, the chip area

will reduce by 4× approximately. It proves that the proposed design (at 45nm) outperforms the other state-of-the-art design in terms of hardware area utilization. As explained in Section IV-B, ExU and DiU utilize *four* and *five* CORDIC stages, respectively, in the proposed SF design. As a result, one Softmax output per clock is obtained in the proposed model, reducing the architecture's overall throughput. However, the loss in throughput is not significant enough compared to the iterative CORDIC-based SF design. Furthermore, it was observed that the proposed design has 3× improved total logic delay than the best state-of-the-art design [12]. Energy-Delay-Product(EDP) reduces significantly by 10× compared to the previous designs [5], [11].

The proposed design has a maximum frequency of 5GHz at 45nm ASIC implementation. Here, a high frequency of operation justified that the architecture is compatible with a wide range of deep learning applications. Furthermore, from TABLE 8, it has been concluded that the maximum frequency of operation of the proposed design is 3× and 1.8× the frequency of operation in [12] and [22] respectively, which are maximum amongst all state-of-the-art techniques. Therefore, it validates that the proposed model is advantageous in terms of area, power, delay, and speed of operation.

VII. CONCLUSION

This study introduces a novel and efficient class-adjustable SF architecture that employs a pipelined CORDIC-based design to achieve area and power-efficient operation. The proposed architecture efficiently addresses high throughput requirements by generating the final output serially at each clock after an initial computational latency. The performance-centric design is instantiated using Pareto analysis for accuracy and pipeline stages, and we have compared it with other architectures using FPGA and ASIC implementation. Our proposed model, with 16-bit precision, achieves almost lossless accuracy for MNIST and CIFAR-10 on the LeNet model and significantly for CIFAR-100 on the ResNet-18 model. We have evaluated the proposed design through simulation and synthesis on FPGA and ASIC with 45nm technology. The experimental results demonstrate that our performance-enhanced technique in SF architecture opens up possibilities for area and power-efficient designs in edge AI computing applications and iterative low-power designs for IoT applications. Furthermore, our results suggest that the proposed design is highly extensible for higher precision arithmetic computation due to its area-efficient architecture.

ACKNOWLEDGMENT

The authors would like to thank the Special Manpower Development Program Chip to System Design (SMDP), Department of Electronics and Information Technology (DeitY), through the Ministry of Communication and Information Technology, Government of India, for providing necessary research facilities.

REFERENCES

[1] M. Motamedi, D. Fong, and S. Ghiasi, "Fast and energy-efficient CNN inference on IoT devices," 2016, *arXiv:1611.07151*.

[2] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," 2021, *arXiv:2103.09301*.

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

[4] Z. Xiao, P. Xu, X. Wang, L. Chen, and F. An, "A multi-class objects detection coprocessor with dual feature space and weighted softmax," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1629–1633, Sep. 2020.

[5] Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, Q. Feng, Y. Fan, X. Yu, and W. Wang, "A high speed SoftMax VLSI architecture based on basic-split," in *Proc. 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Oct. 2018, pp. 1–3.

[6] D. Yang, J. Qin, Y. Pang, and T. Huang, "A novel double-stacked autoencoder for power transformers DGA signals with an imbalanced data structure," *IEEE Trans. Ind. Electron.*, vol. 69, no. 2, pp. 1977–1987, Feb. 2022.

[7] G. Kalyani et al., "Diabetic retinopathy detection and classification using capsule networks," *Complex Intell. Syst.*, 2021, doi: 10.1007/s40747-021-00318-9.

[8] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1045–1049, Sep. 1996.

[9] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient softmax hardware architecture for deep neural networks," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 75–80.

[10] Z. Li, H. Li, X. Jiang, B. Chen, Y. Zhang, and G. Du, "Efficient FPGA implementation of softmax function for DNN applications," in *Proc. 12th IEEE Int. Conf. Anti-Counterfeiting, Secur., Identificat. (ASID)*, Nov. 2018, pp. 212–216.

[11] Z. Wei, A. Arora, P. Patel, and L. John, "Design space exploration for softmax implementations," in *Proc. IEEE 31st Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2020, pp. 45–52.

[12] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, "Efficient precision-adjustable architecture for softmax function in deep learning," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3382–3386, Dec. 2020.

[13] Y. Gao, W. Liu, and F. Lombardi, "Design and implementation of an approximate softmax layer for deep neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Sep. 2020, pp. 1–5.

[14] A. Kagalkar and S. Raghuram, "Cordic based implementation of the softmax activation function," in *Proc. 24th Int. Symp. VLSI Design Test (VDAT)*, 2020, pp. 1–4.

[15] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.

[16] J. Duprat and J.-M. Müller, "The CORDIC algorithm: New results for fast VLSI implementation," *IEEE Trans. Comput.*, vol. 42, no. 2, pp. 168–178, Feb. 1993.

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

[18] M. Alçin, I. Pehlivan, and S. Koyuncu, "Hardware design and implementation of a novel ANN-based chaotic generator in FPGA," *Optik*, vol. 127, no. 13, pp. 5500–5505, Jul. 2016.

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

[20] *Cifar 10 88% Accuracy Using Keras*. Accessed: Sep. 9, 2022. [Online]. Available: https://www.kaggle.com/kedarsai/cifar-10-88-accuracy-using-keras

[21] R. Hu, B. Tian, S. Yin, and S. Wei, "Efficient hardware architecture of softmax layer in deep neural network," in *Proc. IEEE 23rd Int. Conf. Digital Signal Process. (DSP)*, Sep. 2018, pp. 1–5.

[22] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Oct. 2018, pp. 223–226.

[23] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using Taylor series," in *Proc. NORCHIP*, 2014, pp. 1–4.
[24] A. Marchisio, B. Bussolino, E. Salvati, M. Martina, G. Masera, and M. Shafique, "Enabling capsule networks at the edge through approximate softmax and squash operations," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 1–6, doi: 10.1145/3531437.3539717.
[25] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1764–1772.
[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[27] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[28] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. 8, no. 3, pp. 330–334, 1959.
[29] X. Hu, R. G. Harber, and S. C. Bass, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 13–21, Jan. 1991.
[30] F. Ertam and G. Aydın, "Data classification with deep learning using tensorflow," in *Proc. Int. Conf. Comput. Sci. Eng. (UBMK)*, 2017, pp. 755–758.

SUMIRAN MEHRA received the B.Tech. degree in electronics and communication engineering. She is currently pursuing the M.S. degree with the Department of Electrical Engineering, Indian Institute of Technology Indore (IIT Indore). She is with the Nanoscale Devices, VLSI Circuit and System Laboratory, Department of Electrical Engineering, IIT Indore. Her research interests include the implementation of configurable VLSI circuits and system design.

GOPAL RAUT (Member, IEEE) received the B.Eng. degree in electronic engineering and the M.Tech. degree in VLSI design from the G. H. Raisoni College of Engineering, Nagpur, India, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Indian Institute of Technology Indore (IIT Indore). He is with the Nanoscale Devices, VLSI Circuit and System Laboratory, IIT Indore. His research interests include compute-efficient and configurable VLSI circuit design for edge-AI applications.

RIBHU DAS PURKAYASTHA received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Indore. He is currently a Software Engineer, working in the field of embedded systems and software.

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

ANTON BIASIZZO received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the University of Ljubljana, in 1991, 1995, and 1998, respectively. He is currently a Researcher with the Jožef Stefan Institute. He is also an Assistant Professor with the Jožef Stefan International Postgraduate School. His research interests include the design and testing of digital systems, reconfigurable systems, and embedded systems.