Received 21 November 2023, accepted 25 January 2024, date of publication 9 February 2024, date of current version 20 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3364361

RESEARCH ARTICLE

Low-Power and Low-Latency Hardware Implementation of Approximate Hyperbolic and Exponential Functions for Embedded System Applications

AYAD M. DALLOO[1], AMJAD JALEEL HUMAIDI[2], AMMAR K. AL MHDAWI[3], AND HAMED AL-RAWESHIDY[4], (Senior Member, IEEE)
[1]Department of Communication Engineering, University of Technology, Baghdad 10066, Iraq
[2]Department of Control and System Engineering, University of Technology, Baghdad 10066, Iraq
[3]Department of Computer Science and Engineering, Edge Hill University, L39 4QP Ormskirk, U.K.
[4]Department of Electronic and Electrical Engineering, Brunel University London, UB8 3PH Uxbridge, U.K.

Corresponding author: Hamed Al-Raweshidy (hamed.al-raweshidy@brunel.ac.uk)

ABSTRACT The hyperbolic and exponential functions are widely used in various applications in engineering fields such as machine learning, Internet of Things (IOT), signal processing, etc. To fulfill the needs of future applications effectively, this paper proposes a low-latency, low-power, acceptable accuracy, and low-cost architecture for computing the approximate exponential function $e^{\pm z}$ and the hyperbolic functions sinh(z) and cosh(z) using a table-driven algorithm named Approximate Composited-Stair Function (ApproxCSF). By adopting a FPGA, the proposed design is realized and demonstrates significant improvements in terms of latency, hardware cost, power consumption, and MSE by 91%, 96%, 74%, and 99%, respectively, compared to the state-of-the-art. Xilinx Virtex-5/7 FPGAs have been employed throughout the functional verification and prototype processes. Compared to related works, it shows that the proposed architectures are much better for low-cost and low-latency computations of exponential and hyperbolic functions than CORDIC, stochastic computation, and the Look-up Table approaches. The source code is publicly available online https://github.com/AyadMDalloo/ApproxCSF.

INDEX TERMS Hyperbolic functions, exponential function, elementary functions, CORDIC, table-driven algorithm, machine learning, approximate computing.

I. INTRODUCTION

The current trend of research in the development of high-performance very large-scale integration (VLSI) designs is increasingly focused on real-time digital signal processing (DSP) and machine learning algorithms. This focus is essential for applications such as surveillance and wearable electronics, which require the analysis and evaluation of sensed data to recognize patterns [1], [2], [3]. IoT and edge processing require immediate action based on sensed data. Some prioritize local processing over cloud computing due to latency and connection limits. Unfortunately, local processing critically demands low-power, high-accuracy, low-latency, and low-cost solutions. A significant number of these algorithms utilize elementary functions such as trigonometric, hyperbolic, exponential, logarithmic, division functions, etc. The calculation of these transcendental functions via computer software always leads to significant delays. Hardware implementations have gained considerable prominence due to the performance improvements that they provide over software implementations. There is a substantial amount of published material that describes the hardware

The associate editor coordinating the review of this manuscript and approving it for publication was Nuno M. Garcia.

implementation of these functions (exponential and hyperbolic). In general, there are five common types of computing methods for implementing these functions, including the look-up table (LUT) approach [4], [5], [6], [7], the polynomial approximation methodology [7], [8], and the coordinate rotation digital computer (CORDIC) algorithm [9], [10], [11], [12], piecewise polynomial approximations [13], [14], [15], and hybrid (table-driven) approaches [16], [17]. The approximate and stochastic computing approaches [18], [19], [20], [21] have also garnered considerable interest in recent years. The benefits and drawbacks of each implementation approach will be discussed in the next section.

There are no known Field-Programmable Gate Arrays (FPGAs) designs in the literature that precisely combine the qualities of low-cost, wide-range, low-latency, acceptable high accuracy all at the same time. In this paper, we propose simple architectures of exponential and hyperbolic functions based on the table-driven approach in [22]. The proposed architectures outperform existing state-of-the-art designs in terms of latency, cost, operational range, and power consumption. The results of the experiments that are given in this paper provide more credence to this fact. Thus, the key contributions of the paper are as follows:

- Development of the Exponential Function: The paper introduces an innovative table-driven algorithm for computing the exponential function. This approach significantly improves performance and reduces cost while extending the input range. The use of Xilinx Virtex-5/7 FPGAs for verification and prototyping underscores the practical applicability of the design.
- Design of Hyperbolic Functions: The paper presents novel architectures for the hyperbolic functions sinh(x) and cosh(x), based on the table-driven approach for the exponential function. This design effectively balances hardware complexity, cost, performance, and accuracy, eliminating the need for data input scaling.
- Comprehensive Review of Elementary Function Implementations: An extensive review of key studies and various methods for implementing elementary functions (exponential and hyperbolic) is provided. This review offers valuable insights into the state of the art and highlights the advantages of the proposed approach.
- Comprehensive Architectural Advancements: The paper presents architectures that excel in accuracy and range, are energy-efficient, scalable, and flexible, and contribute to the open-source community, supporting diverse applications and technological evolution.

The rest of the paper is organized as follows: Section II surveys and outlines the most pivotal studies of different methods for implementing elementary functions. Section III elucidates the background of the computation of expansion and hyperbolic functions. Section IV describes the architecture of expansion and hyperbolic functions. Afterward, in Section V, the experimental results are used to quantify the benefits of our proposed architecture. Finally, Section VI concludes the paper.

II. LITERATURE REVIEW

Exponential and hyperbolic functions are essential for many computational tasks, requiring accurate and efficient implementation. Efficient and accurate approximations of these functions are important for FPGA-based computing, where hardware resources are limited and speed is critical. To this end, we have surveyed the most pivotal studies in the field published between 2011 and 2022, with a focus on those appearing in the last seven years between 2017 and 2023. Furthermore, this review includes some of the earliest studies documenting the concept's inception. In particular, articles published by highly regarded publishers like IEEE, Elsevier, MDPI, Nature, ACM, and Springer were prioritized. The ArXiv repository has provided the source for a few of the chosen papers. In this section, we will highlight the benefits and drawbacks of each approach and review some of the recent published studies on approximating and implementing hyperbolic and exponential functions on FPGAs and ASICs.

A. LOOK-UP TABLE APPROACH

The Look-Up Table (LUT) method [4], [5], [6], [7], is considered the easiest, most effective, and quickest method for computing exponential and hyperbolic functions by interpolating information stored in memory blocks. In Figure 1, the LUT addresses a defined range of input values with 8-bit addressing. Values beyond this range might need extrapolation or error management. Therefore, the interpolation techniques (e.g., linear interpolation) can be used to improve accuracy and generate input values between LUT entries. This low-complexity method requires ample silicon space since the accuracy is affected by the size of the memory. LUTs are particularly useful for functions that have a small input domain and a fixed output precision. Therefore, it is not suitable for highly oscillatory or rapidly changing functions. The core challenge with such functions is their need for an extremely dense set of points in the lookup table to accurately capture their behavior. This requirement for a high density of points leads to increased memory usage. This makes the lookup table approach inefficient or even impractical for these types of functions, particularly in scenarios demanding real-time processing systems with restricted computing capabilities. Saint-Genies et al. [4] proposed a method for using error-free values by tabulating two or more terms per table row using Pythagorean triples. This technique reduces memory use by up to 29% and floating-point operations by up to 42%. Deng et al. [6] introduced a framework for automatically developing of a look-up table (LUT) to generate and evaluate the functions for optimization of hardware resources in FPGAs. The evaluation of the function is conducted through a numerical approximation methodology employing Taylor polynomials. This approach is meticulously tailored to meet specific demands regarding precision and computational speed. The result of the exponential function shows the maximum error is 1.69e-7. Magalhães et al. [5] provided an optimization tool for creating accurate and efficient LUTs,

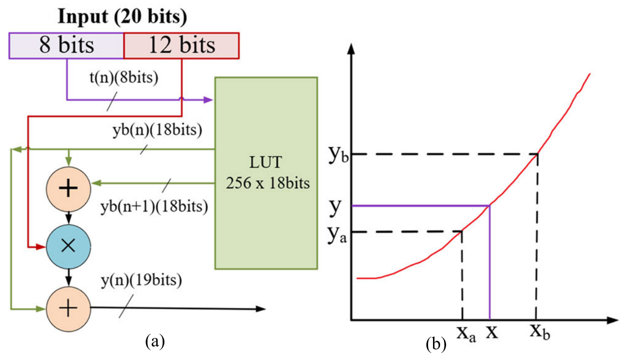


FIGURE 1. (a) Hardware implementation of exponential function using one LUT, (b) The illustration of linear interpolation method [7].

which is composed of layers with thicknesses that specify the distance between pre-calculated points.

B. POLYNOMIAL APPROACH

Another approach is polynomial approaches, which are techniques used to approximate mathematical functions by a polynomial function of a given degree. The Taylor series and polynomial approximation approaches are both techniques used to approximate mathematical functions, but they differ in their underlying mathematical principles and implementation. The implementation of exponential and hyperbolic functions frequently employs Taylor series, a renowned mathematical method for approximating functions through a sequence of derivatives at a specific point. Although these series may converge slowly for larger values, truncating them to a finite number of terms often yields a satisfactory level of accuracy. Nonetheless, this method may demand significant computational resources and is susceptible to numerical instability, especially with larger argument values.

To address these issues, various alternative methods have been proposed in recent years. Costa et al. [23] provided a Taylor Series exponential function with a variable input range and an eight-byte LUT. The proposed architecture used a Newton-Raphson division and a radix-4 Squarer unit for designing a Taylor series exponential function. To minimize error and get a reliable approximation, the polynomial approximation would like to be used [7], [8]. However, this method requires many multipliers, adders, and tables for storing coefficients. It is both slow and inefficient, as depicted in Figure 2(a). Wu et al. [8] presented an approximate exponential function unit (EFU) based on Taylor expansion and optimized using discrete gradient descent with a power consumption of 3.73 pJ/exp. Polynomial approximation is a common approach for approximating exponential and logarithm functions on FPGAs. The idea is to represent the function as a polynomial that approximates the function over a specific input range. The polynomial coefficients can be determined by fitting a polynomial to the function using least-squares regression or another curve-fitting algorithm. In their respective studies, Chen et al. [24], Nandagopal [25], and Ze [26] have each introduced a hardware accelerator specifically designed for elementary transcendental functions, demonstrating significant enhancements in computational throughput. Notably, the work of Chen et al. [24] exemplifies this advancement, achieving an approximate throughput of 2.5 GFLOPS utilizing 65nm CMOS technology. This development is further characterized by its precision, maintaining an average error of 0.5 units in the last place (ulp) and a maximum error of 3 ulp.

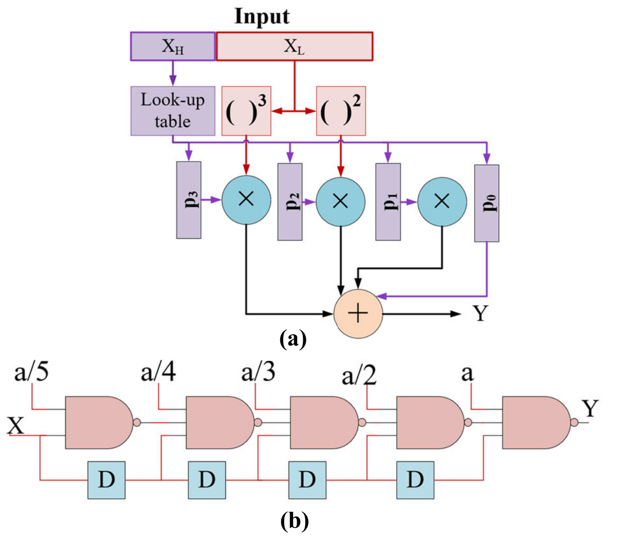


FIGURE 2. Architecture of the exponential function using (a) a 3rd-order polynomial Approach, (b) stochastic approach of a 5th-order Maclaurin polynomial [19].

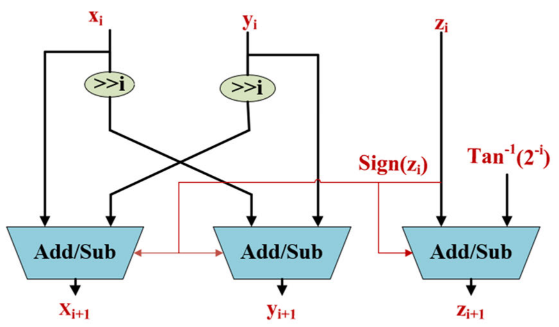


FIGURE 3. Hardware implementation of the conventional Cordic algorithm.

C. CORDIC APPROACH

The third method is the CORDIC algorithm, which is a low-cost iterative algorithm designed by Volder [12] in 1959. It uses adders, wire shift operations, and a few registers, as depicted in Figure 3. Unfortunately, it has delayed performance like a serial multiplier and a limited input range, making it not the ideal method for exponential and hyperbolic function computation.

Recent enhancements to the CORDIC algorithm offer potential for affordable, high-performance real-time computing hardware solutions [8], [9], [10], [11]. Osta et al. [18]

have conducted research with the objective of diminishing the energy usage of specialized circuits in real-time execution of CORDIC algorithms within the realm of machine learning. This is achieved through the application of approximate computing methodologies. The findings from their study indicate that the integration of Lower-Part Or adder (LOA) leads to a significant reduction in power consumption, quantified at 21%. Based on adapting the current [27] architecture with an approximation, Chen et al. [9], [28] presented a new approximate approach for coordinate rotation digital computer (CORDIC) construction. A completely parallel approximation CORDIC (FPAX-CORDIC) technique is developed, which eliminates the memory register of Para-CORDIC and makes rotation direction generation totally parallel. Although approximation CORDIC and parallel CORDIC functions have their benefits, they nevertheless have limits in terms of input range and latency. When dealing with difficult mathematical processes or big input quantities, the performance of these functions may be hampered. Hence, researchers and developers continue to explore new solutions and enhancements to meet these issues and produce more effective and adaptable algorithms for a wide range of areas.

D. PIECEWISE APPROXIMATIONS APPROACH

The fourth method is Piecewise Linear/Nonlinear/ polynomial Approximations. Piecewise Linear Approximation (PLA) is a computationally efficient method for numerical approximation, particularly suitable for real-time systems with resource constraints. PLA is characterized by its simplicity, which makes it suitable only for implementing simple functions in constrained computational environments. PLA generates non-uniform segments based on the maximum error threshold. The number of segments affects the length of the input interval. It also impacts the steepness of the function. However, PLA suffers from limitations where it has limited accuracy for complex, non-linear functions. It also introduces discontinuities at the boundaries between segments.

Moving beyond PLA's simplicity, Piecewise Nonlinear Approximation (PNA) offers a more advanced solution. PNA utilizes nonlinear functions such as exponentials, logarithms, and trigonometric functions. Its aim is to accurately represent complex functions. However, it comes at the cost of increased computational complexity and the challenge of selecting the most appropriate nonlinear function for each segment.

Lastly, Piecewise Polynomial Approximation (PPA) balances computational complexity and accuracy by using polynomials of different degrees over segmented intervals. PPA is widely used in signal processing and scientific computing. It can face boundary issues and needs domain expertise for best results. It's chosen for applications needing a good speed-accuracy trade-off. Figure 4 illustrates piecewise linear and quadratic approximations. For example, Chiluveru et al. [13] developed a novel iterative algorithm designed for the piecewise linear approximation of the sigmoid function, characterized by its controlled accuracy. Dong et al. [14] introduced an advanced,

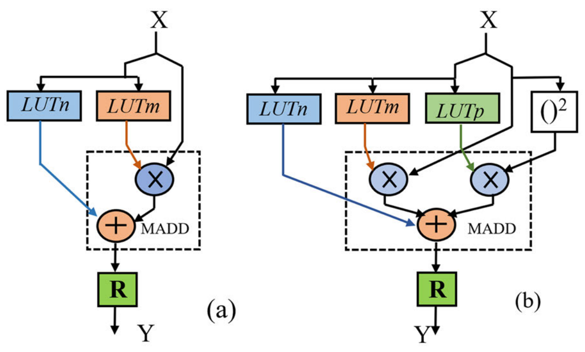


FIGURE 4. General structure of piecewise approximation approaches (a) Linear; (b) Quadratic.

universally applicable, and error-minimized piecewise linear (PLA) approximation methodology. This approach is elaborated through a comprehensive piecewise linear approximation computation (PLAC) technique, which is effectively applicable across a broad spectrum of nonlinear unary functions. The PLAC method is distinguished by its two primary components: an optimized segmentation mechanism and a refined quantization process. Then Lyu et al. [15] built PLAC without using a multiplier. This architecture is optimized by Yu et al. [29] to find the minimum number of segments and reduce the maximum absolute error (MAE). All the authors concentrated on the [0,1) interval for their circuit designs. However, these designs necessitate employing the scaling property of the exponential function for input and output processing.

E. HYBRID (TABLE-DRIVEN) APPROACH

Hybrid (Table-driven) methods involve combining multiple approximation techniques to improve accuracy or reduce computational cost [16], [22], [30]. For example, a lookup table can be combined with a linear or polynomial approximation to improve accuracy over a wider input range. To address the need for an efficient implementation of the exponential function with variable precision fixed point negative input, Chandra [16] proposed a hybrid method combining LUTs and polynomial approximation to reduce the number of multipliers and adders. The space requirements and energy consumption decreased by over 30% and 50%, respectively.

Overall, hybrid methods are a powerful tool for approximating complex functions and are widely used in many areas of science and engineering. Exponential and hyperbolic functions are important in many areas of science and engineering, but they can be computationally expensive to evaluate directly, especially for large inputs or high precision. Hybrid methods can be used to overcome these challenges by combining different approximation techniques that are well-suited to different parts of the function's domain.

F. APPROXIMATE AND STOCHASTIC COMPUTING APPROACH

In recent years, there has also been a great deal of interest in the techniques of approximation computing and stochastic computing [18], [19], [20], [21], [31]. High clock speeds and fault tolerance are two distinguishing features of stochastic computing, resulting in exceptionally low hardware costs and power consumption. Frameworks for stochastic computation based on the fundamental building blocks of arithmetic and logic are shown in Figure 2(b). However, it has drawbacks such as decreased precision and extended latency. Luong et al. [21] explored the implementation of stochastic logic in executing complex arithmetic functions, notably exponential, sigmoid, and hyperbolic tangent functions. This study utilized piecewise linear and polynomial approximations, specifically employing Lagrange interpolation, to achieve these computations. The findings show power consumption and hardware complexity reduced by 40% and increased the critical path by 2.5% compared to earlier designs.

In addition, approximate computing is a technique for reducing the gap between CMOS scaling and future application needs by utilizing the trade-off between hardware cost and accuracy, which offers significant promise for enhancing the performance of integrated systems [31]. The parallel CORDIC proposed by [27] was approximated by [9] and [18], but these approximated CORDIC versions may still not meet many applications' needs due to their delay.

In this literature review, we have examined a wide range of methodologies for implementing low-power and low-latency hardware designs for approximate hyperbolic and exponential functions, crucial in embedded system applications. Our analysis covered various approaches, including Look-Up Table (LUT), polynomial approximation, CORDIC algorithms, Hybrid (table-driven) methods, etc. Every method offers unique advantages and drawbacks. The selection relies on the needs of the particular application.

The LUT method is notable for its low latency and ability to replace complex calculations. However, it demands extensive memory for high precision, which can be a drawback, as seen in Hugues' approach [4] for exact hyperbolic functions, where high memory access and floating-point operations become a disadvantage due to delays and high-power consumption. Conversely, Magalhães [5] and Deng [6] have contributed tools for automating LUT development, optimizing hardware resources. From my experiments, LUTs are suitable for generating functions with lower accuracy or for functions like the exponential with negative inputs, which have outputs in the [0, 1] range. While LUTs offer lower latency and accuracy compared to our proposed method, our goal is to balance these aspects. Similarly, the polynomial approximation needs to high order of polynomial for satisfying high precision needs. One of methods to reduce the interpolation points is to use polynomial approach.

Piecewise approximations (PA) present a middle ground between accuracy and computational load but may need significant hardware for coefficient storage and calculations. The main time consumption in PA lies in coefficient addressing, and compared to our method, PAs have higher delays, despite sharing similar characteristics.

The CORDIC algorithm and stochastic methods are less costly in terms of hardware and power but come with their own set of challenges. The CORDIC algorithm faces higher latency and lower accuracy, whereas the stochastic method trades accuracy for lower latency. Essentially, each method compromises one design aspect. However, hybrid approaches aim to merge the advantages of different methods to boost performance, albeit at the cost of increased design and implementation complexity. For instance, Chandra [16] combined LUT and polynomial approximation in designing an exponential function for negative inputs, reducing the need for multiple multipliers and adders.

Our review highlighted that despite the advancements in these methodologies, challenges remain in achieving an optimal balance between power consumption, latency, accuracy, and hardware cost. Notably, the current implementations exhibit limitations in scalability, flexibility, and efficiency when subjected to the demanding requirements of real-time DSP and machine learning applications. These gaps underscore the necessity for innovative approaches that can adeptly navigate the trade-offs inherent in hardware design for exponential and hyperbolic function computations. Our proposed architecture, leveraging the Approximate Composited-Stair Function (ApproxCSF) and table-driven algorithms, aims to address these shortcomings by offering a design that significantly improves upon latency, power consumption, and hardware efficiency, while maintaining acceptable accuracy levels, thus presenting a viable solution to the identified gaps in the literature.

III. BACKGROUND OF TABLE-DRIVEN ALGORITHM

A table-driven (hybrid) implementation algorithm is proposed by Tang et al. [22] to provide a software implementation of the exponential function in IEEE Floating-point arithmetic. Fixed-point numbers are used to speed up the processing of exponential functions, and the table-driven technique is used to improve both speed and accuracy. In this article, we present a simplified approach to the table-driven technique for the exponential function, building upon the framework established in [22]. The algorithm, originally detailed in [22], is described in the following steps:

In this design, the values of $2^{j/32}$, corresponding to the fractional part of N/32, are pre-computed and stored in 32 memory locations, utilizing a lookup table. This setup is pivotal for the algorithm's efficiency. This approach allows for rapid retrieval of these values during computation, thereby reducing the computational complexity and underscoring the table-driven nature of our design.

As part of the VLSI design of support vector machines (SVM), Patankar et al. [32] used a table-driven algorithm to implement the exponential function e^{-z} to realize the Gaussian function. The drawback of this implementation is utilizing the divider unit, which has a high computation cost. The divider unit is employed for normal division operation between two numbers and for critical tasks such as input normalization and output scaling. Division operations,

Steps	Algorithm 1
Step 1:	The input argument X is reduced to the range [−log(2)/64, log(2)/64]. Obtain integers m, j, and r, where $\lvert r\rvert \leq \frac{log2}{64}$

$$X = \frac{(32m + j) \times \log(2)}{32} + r,$$
$$m = \frac{N_1}{32}, j = N_2,$$
$$N = Intger\ Round\left(X * \frac{32}{\log(2)}\right),$$
$$N_2 = \mathrm{N} \bmod 32,\ N_1 = \mathrm{N} - N_2 \quad (1)$$

Step 2: The function exp(r)-1 is approximated by a polynomial p(r), where

$$p(r) = r + a_1 r^2 + a_2 r^3 + \ldots + a_n r^{n+1} \quad (2)$$

Step 3: Reconnect exp(x) via

$$\exp(X) = 2^m (2^{\frac{j}{32}} + 2^{\frac{j}{32}} \times p(r)) \quad (3)$$

while necessary for these tasks, are inherently more computationally intensive than simpler arithmetic operations. Division operations are notably time-consuming, often requiring a considerable amount of clock cycles, varying from tens to hundreds. Furthermore, division operations necessitate significant area due to their complexity. Even a minimal improvement in the divider circuit, such as 1%, can significantly boost the overall system performance by up to 20% [32]. In our design, we decide to eliminate the divider circuit due to several critical considerations. The following formula for calculating the exponential function was adopted in [33]:

$$\exp(-\mathrm{z}) = \left(\frac{1}{2^{\mathrm{m}}}\right) \cdot \left(\frac{1}{2^{j/32}}\right) \cdot \left(\frac{1}{1 + \mathrm{r} + \frac{1}{2}\mathrm{r}^2}\right) \quad (4)$$

Our development is divided into two phases to acquire a comprehensive description and analysis of our architectures. We provide a broad description of the design and a breakdown of how exponential and hyperbolic functions are implemented in the circuit.

IV. THE PROPOSED ARCHITECTURES

A. GENERAL DESCRIPTION OF ARCHITECTURE

This paper presents a systematic framework for exponential functions which segments the functions into two parts: the Stair-Step Function (SSF) and the Composited-Error Function (CEF). The SSF is essentially a piecewise function that approximates the exponential curve by breaking it down into a series of discrete 'steps' or segments. This stepwise approximation simplifies the complex nature of the exponential function, making it more manageable for computational purposes. On the other hand, the CEF is designed to capture and represent the error introduced by the stepwise approximation of the SSF. It models the deviation from the actual exponential curve as a sawtooth waveform, which helps in analyzing and compensating for the approximation error in subsequent processing stages. We have named our proposed architecture the Approximate Composited-Stair Function (ApproxCSF). It employs a table-driven approach to approximate the exponential function, as detailed in Algorithm 1.

This design converts the input parameter z into an integer number N by multiplying it by a certain constant C1 and feeding N into two segments, each with its own function. The SSF segment can generate the stepped-exponential function in stair form directly by looking up $2^{\mathrm{N}/32}$, but the cost of this process will be high because dividing N by 32 yields a fractional amount. Consequently, it requires a large number of memory places to achieve the highest precision. To reduce the cost of this design, the integer number N is divided by 32 into quotient m and remainder j. Figure 5 illustrates the design's block diagram. The remainder j is generated by extracting 5 bits from N and the remaining bits are used as quotient m (e.g., m=6 bits if N is 11 bits). The fractional component of N/32 is represented by storing $2^{\mp \mathrm{j}/32}$ values in 32 memory locations. Regarding the integer quotient m, it is shifted by one position to generate $2^{\mp \mathrm{m}}$ through the use of a decoder, instead of utilizing a shift register. The last step of the SSF segment is to multiply $2^{\mp \mathrm{m}}$ by $2^{\mp j/32}$ to yield $2^{\mp N/32}$ as described in algorithm I and detailed more below.

$$X = 2^{Ni/32} = 2^{\frac{(N-j)+j}{32}} = 2^{\frac{N-j}{32} + \frac{j}{32}} = 2^m \cdot 2^{\frac{j}{32}} \quad (5)$$

In the CE segment, the error value ε is determined by subtracting the estimated input argument $\mathrm{z_n}$ from the input argument z, where $\mathrm{z_n}$ is calculated from the integer number N after multiplying by a certain constant C2 (=1/C1). The output of the segment (Y) is computed by adjusting the error (ε) through addition or subtraction of 1 to calculate $\mathrm{e^x}$ or $\mathrm{e^{-x}}$, respectively. The approximate exponential function is then computed by multiplying the segment's output, Y, by the segment's output, Feng et al. [33] entered the error ε into the second-order polynomial $\mathrm{p}(\varepsilon)$ and then divided it by the output of SSF segment X. This is a drawback of Feng's implementation.

The implementation of hyperbolic functions sinh(x) and cosh(x) can be easily developed by adding and subtracting two exponentials' functions, $\mathrm{e^{-x}}$ and $\mathrm{e^x}$, and then wire shifting them, as follows:

$$\sinh(\mathrm{x}) = \frac{\mathrm{e^x} - \mathrm{e^{-x}}}{2}, \cosh(\mathrm{x}) = \frac{\mathrm{e^x} + \mathrm{e^{-x}}}{2} \quad (6)$$

B. THE CIRCUIT DESIGN

The exponential function is a fundamental part of activation functions for neural networks and various algorithms. For instance, the exponential function is used by the Gaussian function in the construction of a support vector machine (SVM). in conscious of the fact that the exponential function for a negative domain is s more widely used than the positive domain [16]. As a result, we proposed and constructed a variety of range structures for the exponential and hyperbolic

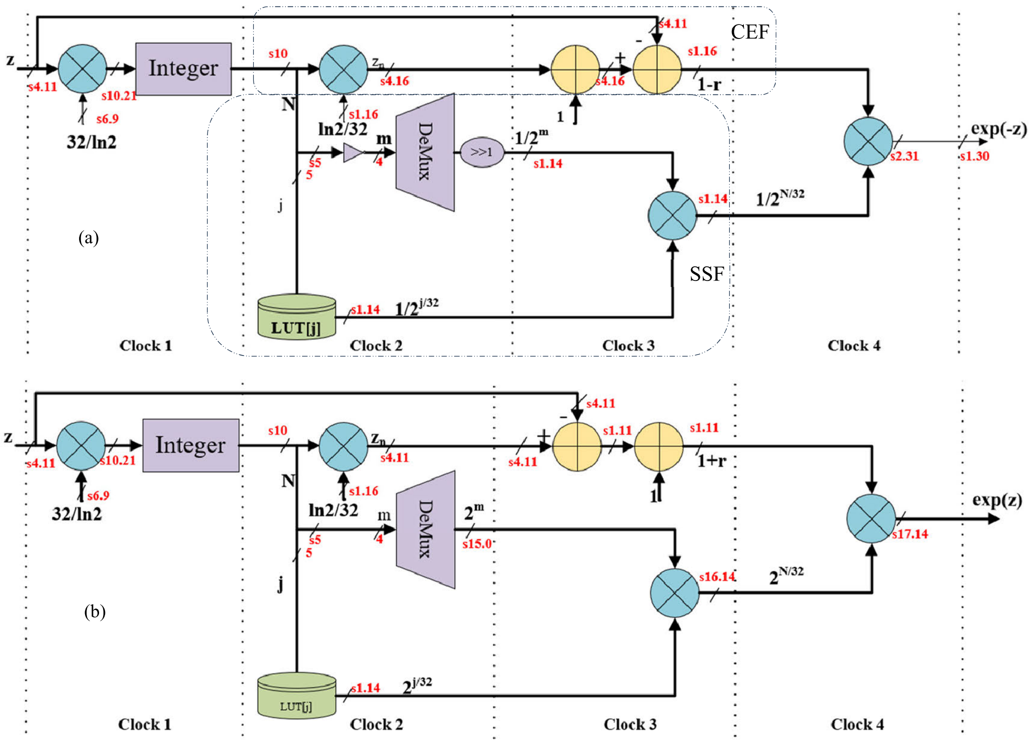


FIGURE 5. The circuit diagram of the implementation of the exponential function (a) e^{-z} (b)e^{+z}.

functions to reduce hardware costs and properly compare them to existing architectures.

The first architecture of the exponential function is limited to the negative input, as shown in Figure 5a. The design is pipelined to achieve a four-clock latency. We use the signed fixed-point s4.11 format (16 bits). The integer number s10.0 is extracted after multiplying the input argument z (s.4.11) by the constant 32/ln(2) (s6.13). Then it is fed to the stair-step function (SSF) and composited-error function (CEF) segments. In the CEF section, the error ε is computed by calculating the difference between estimated and exact input parameters, which is always less than one. The final output of the CEF segment (represented by X) may be less than or more than one used to calculate the exponential function $\boldsymbol{e}^{-z}$ or $\boldsymbol{e}^{z}$, respectively, as explained below.

$$X = \begin{cases} 1 - \boldsymbol{\varepsilon}, & \boldsymbol{for}\ \boldsymbol{e}^{-z} \\ 1 + \boldsymbol{\varepsilon}, & \boldsymbol{for}\ \boldsymbol{e}^{+z} \end{cases} \tag{7}$$

where $\varepsilon = \text{z-z}_\text{n}$, z represents the exact input argument, and z_n is the estimated input argument. Since outcome X is always around one, the outcome is always represented by s1.16.

In the stair-step function segment, the integer number N is split into two numbers (denoted j and m), and the reminder number j is used to access the lookup table in order to acquire $2^{-j/32}$ for e^{-z} or $2^{j/32}$ for e^{z}. Because the output of the decoder 2^{-m} for e^{-z} or 2^{m} for e^{z} is 16 bits, the quotient m is represented by just 4 bits here. The output of the stair-step function (SSF) segment is explained below.

For the exponential of negative input argument:

$$\begin{aligned} \text{fraction} &= 2^{-\text{j}/32} \leq 1 \rightarrow \text{s1.14} \\ \text{quotient} &= 2^{-\text{m}} \leq 1 \rightarrow \text{s1.14} \end{aligned} \tag{8}$$

For the exponential of positive input argument:

$$\begin{aligned} \text{fraction} &= 2^{\text{j}/32} \geq 1 \text{ and } \leq 2 \rightarrow \text{s1.14} \\ \text{quotient} &= 2^{\text{m}} \geq 1 \rightarrow \text{s15.0} \end{aligned} \tag{9}$$

Multiplying the outputs of two segments (X and Y) yields the final circuit output, which reflects the input argument's exponential function. The final output is represented by 32 bits, with s1.30 for $\text{e}^{-\text{z}}$ and s17.14 for e^{z}. Figures 5a and 5b depict the exponential function circuits, and the section on the results and assessment describes the aspects of these circuits.

As demonstrated in Figure 6, the aforementioned architectures may be integrated, modified further, and simplified to encompass the exponential function of the positive and negative input ranges. The design selects between the exponential functions of negative and positive input arguments based on the sign of the input argument. In this design, we utilize a 64-memory location lookup table to find $2^{-\text{j}/32}$ for $\text{e}^{-\text{z}}$ and $2^{\text{j}/32}$ for e^{z}, which are addressed by combining the remainder of j with the sign of input argument z. In the SSF segment, the quotient number m must be inverted, and the decoder's output must shift only when the parameter z is negative. The sign of the input parameter determines how the operation is performed in the CEF segment. The error ε is directly

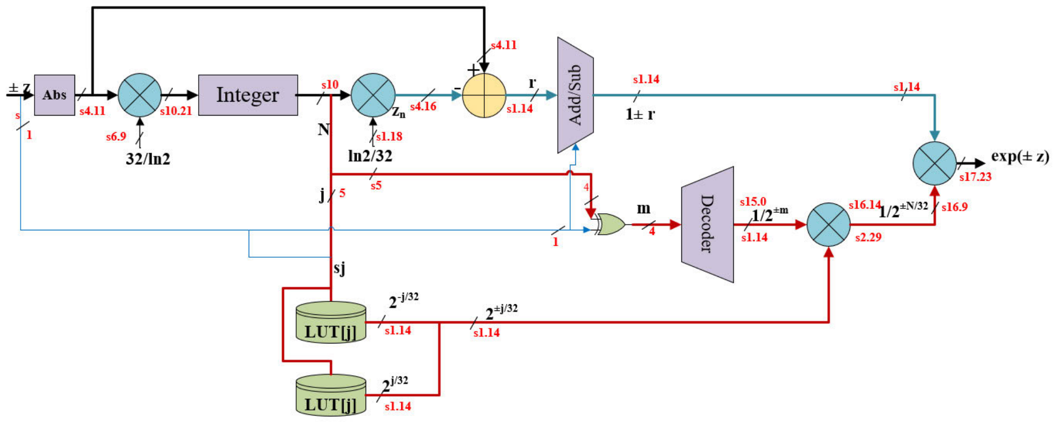


FIGURE 6. The circuit diagram of the implementation of the exponential function $e^{\pm z}$.

inserted into the adder or subtractor to obtain $1\pm\varepsilon$ and is controlled by the sign of the input argument. The output of the CEF segment has a distinct numerical format when the exponential functions e^{-z} and e^{z} are calculated. This distinction necessitates an increase in the number of output bits from 32 bits to 41 bits to avoid accuracy loss.

When hyperbolic functions are significant, it is necessary to build both exponential functions in parallel. Thus, the circuits depicted in Figure 5 operate in parallel, with their outputs fed into the adder/subtractor unit as per Equation 6. Subsequently, a wire shift is applied to generate the hyperbolic functions. The control signal is used to choose between cosh(z) and sinh(z). Figure 7 shows the architecture of sinh(z) and cosh(z) functions.

V. EXPERIMENTAL RESULTS

The hyperbolic and exponential functions have been developed in VHDL to evaluate and assess the features of the proposed architectures presented in the preceding section. Then, these designs are synthesized in the Xilinx ISE Design Suite and evaluated on the Xilinx Virtex-5 (XC5VLX110T) and Virtex-7 (XC7VX485) FPGAs, which support 16-bit operands with the fixed-point format s4.11.

In this work, we analyze the architectures based on a variety of important metrics, such as delay, area (number of LUTs, FFs, and slices), maximum frequency, latency, throughput, and error. Each proposed architecture has been functionally verified with 10^6 different uniformly distributed random input patterns. The error metrics are then calculated by comparing the hardware simulation results of the proposed designs with the floating-point results of the Matlab program. Following this, we discuss the timing analysis and hardware utilization to conclude the FPGA implementation study. In this work, we conduct three primary tests on the proposed exponential and hyperbolic function designs.

A. PERFORMANCE OF APPROXIMATE EXPONENTIAL COMPOSITED STAIR FUNCTION

To show the superiority and highlight the advantages of our architectures (ApproxCSF) for generating the exponential function compared to other best FPGA-implementing architectures. The first architecture of the exponential function is designed only for negative input. It is compared with the architecture developed by [33] and [34], and the standard CORDIC algorithm known as Xilinx CORDIC IP Core V4. All hardware was simulated with a clock of 5 ns, but the non-pipelined proposed architecture was simulated with 10 ns.

TABLE 1. Comparison of different architectures of 16-bit approximate exponential function e^{-z}.

Feature	Proposed with pipelined	[33]	Saving/ Costing	Xilinx CORDIC IPCore v4.0
Accuracy (bits) (fixed-point)	16	16	-	16
Delay	4.871ns	4.855ns	-	2.877ns
Max. frequency	205.297MHz	205.973MHz	-	347.569MHz
Latency	4clk	44clk	-91%	23clk
Throughput (Mexp/sec)	200@200MHz	200@200MHz	-	300@300MHz
LUT	39	694	-94%	2,307
FF	179	1,974	-91%	2,278
DSP48	4	4	-91%	0
BRAM	0	0		0
slices	48	557	-91%	633
Dynamic Power @200MHz	43mW	167mW	-74%	526.04mW
Total power (W)	0.865	0.990	13%	1.431
Input Range	[-10.397,0]	[-10.397,0]		[-0.56,0]
Minimum Error	-2.7e-4	-16.71e-4	-84%	0
Maximum Error	2.98e-04	19.54e-04	-85%	0.245
Mean(μ)	2.92e-05	3.42e-04	-91%	0.164
Std(σ)	5.3e-05	7.09e-04	-93%	0.070
MSE	3.68e-09	6.19e-07	-99%	0.032
MAE	3.8e-05	6.52e-04	-94%	0.164
Pe	0.994	0.979	+2%	0.954

According to Table 1, our proposed method decreased the latency from 44 to 4 clocks with the same maximum frequency compared with the method in [33]. In other words, we reduced the latency by 91% and 83% compared with the methods in [33] and the Xilinx CORDIC IP core, which is illustrated in Figure 8. The employment of the costly and latency-intensive divider unit is the cause of the high latency in [33]. The proportion of slices used by the proposed architecture is just 8% and 88% in [33] and [34], compared to the CORDIC IP core, as shown in Figure 8. In addition, the proposed architecture consumes 74% less power than the

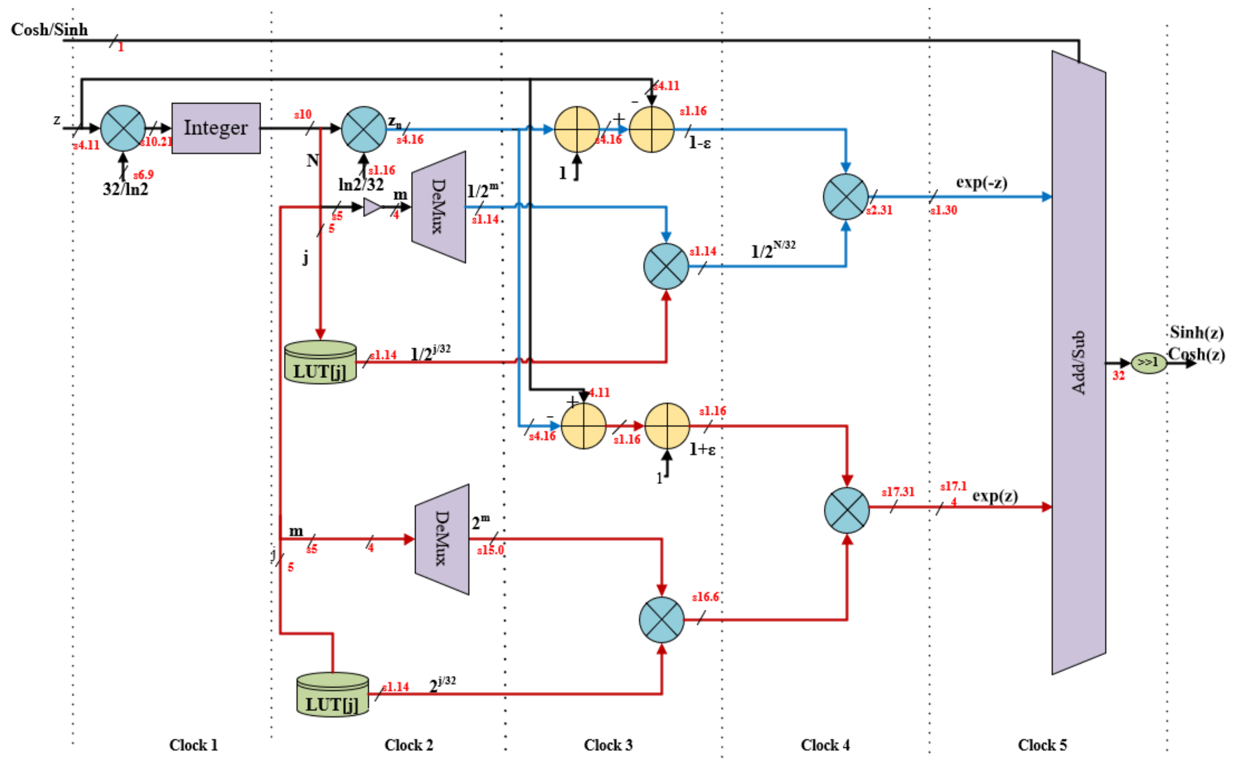


FIGURE 7. The circuit diagram of the implementation of the hyperbolic functions.

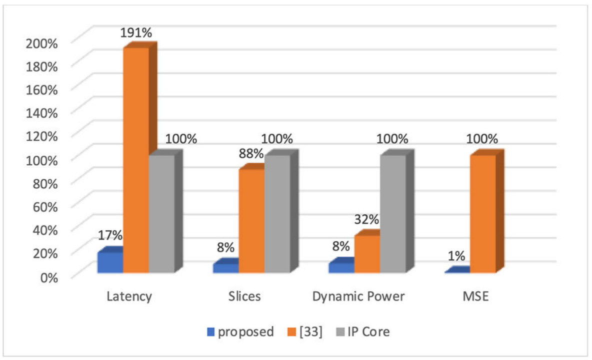


FIGURE 8. Comparative performance analysis of the proposed design against the [33] design and the IP Core in terms of Latency, Slices, Dynamic Power, and Mean Squared Error (MSE). The MSE is excluded from the IP Core comparison due to its significantly higher value.

approach described in [33] and [34] due to its usage of fewer hardware resources. However, it has the same delay as the other approaches. Figure 8 depicts a visual comparison of the advantages of the proposed architecture over alternative architectures. The CORDIC IP Core has higher error metrics and a smaller input range than the other approaches, according to Table 1. Therefore, the error metrics of IP Core are excluded from our comparison. Figure 9 illustrates the distribution of errors across the input domain, spanning from −10.397 to 0, for both the architecture proposed in and our proposed architecture. Our findings indicate that excluding the quadratic term results in a significant reduction of the maximum error value by 85%. However, this modification leads to a marginal increase in the overall error rate, specifically by 2%. Moreover, excluding this term facilitates a reduction in hardware complexity and cost by obviating the need for an additional multiplier and adder. Thereby, this simplification enhanced the system's accuracy and efficiency in terms of power consumption, hardware complexity, and resource allocation. Based on the results presented in Table 1, our design outperforms the methods described in [33] by achieving 94%, 99%, and 93% enhancements in MAE, MSE, and STD, respectively. However, it also results in a 2% higher error probability.

For handling negative inputs in exponential functions, the proposed architecture outperforms the current state-of-the-art approaches. This architecture is also employed to develop hardware implementations of exponential functions for both full and positive inputs, leveraging its distinctive features. The architectural outcomes for handling both negative and positive inputs exhibit similar performance characteristics facilitating straightforward comparison. Figure 10 illustrates the error distribution across the range [0, 10.397].

In light of the distinct qualities of the two aforementioned designs, we were inspired to create a single architecture capable of handling both the positive and negative broad ranges of exponential computation (as shown in Figure 6). Table 2 showcases a comparison of our design against various approaches in the literature. Our architecture achieves better latency and hardware efficiency. However, the method in [35] surpasses our design in terms of error metrics and range

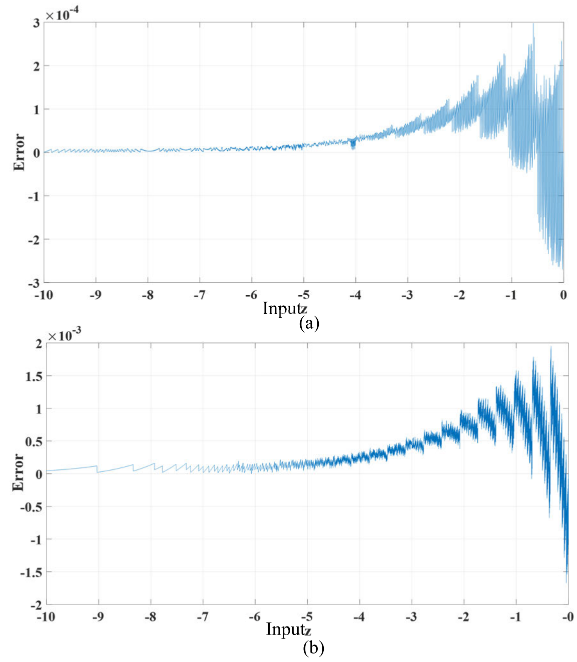


FIGURE 9. Distribution of errors in the implementation of the exponential function e^{-z} for (a) our proposed architecture and (b) the architecture referenced in [32].

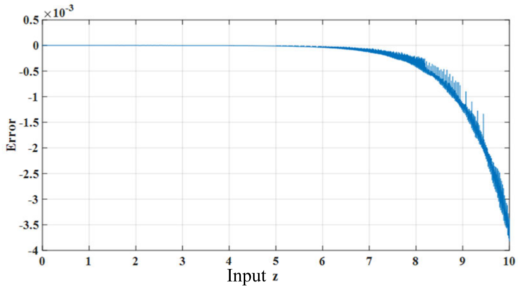


FIGURE 10. The normalized error distribution of implementing of the exponential function e^{+z}.

coverage due to its 55-bit floating-point precision, 16Kb BRAM lookup table, and high-end hardware. Our proposed architecture employs a 16-bit fixed-point format for precision, whereas the approach in [35] uses a 55-bit floating-point format. Consequently, the design presented in [35] is capable of processing a significantly broader input range compared to our proposed design. Despite identical maximum frequencies, our design significantly lowers latency, LUT, and FF usage by 82%, 91%, and 92%, respectively, compared to [35]. Our design prioritizes power efficiency with a trade-off in accuracy. Enhancing precision would require more bits and potentially separate storage for integer functions in the lookup table.

This study aims to balance different metrics, finding an equilibrium that optimizes overall performance without disproportionately favoring any single aspect.

B. PERFORMANCE OF APPROXIMATE HYPERBOLIC COMPOSITED STAIR FUNCTION

In this section, we design and implement hyperbolic functions on the FPGA Virtex-7 XC7VX485T in order to perform a fair comparison with the approach described in [11]. Our proposed circuit demonstrates significant improvements in cost, power, and performance, accompanied by a minimal 0.074% decrease in accuracy, which remains satisfactory. Table 3 depicts an evaluation and comparison of the proposed architecture with the modified CORDIC algorithms reported in [11] in terms of timing analysis, hardware utilization, and error metrics. The proposed architecture adopts a 16-bit fixed-point precision with accepting a minor decrease in accuracy to avoid the complexities and delays inherent in floating-point computations. Fu et al. [11] introduced a 128-bit floating-point version of the exponential function, emphasizing precision over other considerations. As a result, this design suffers from reduced performance and high implementation costs. The proposed architecture significantly outperforms the method described in [11] reducing latency and delay times by 81.25% and 98.45%, respectively. This leads to a remarkable 99% improvement in throughput at an operational frequency of 200 MHz. Furthermore, the architecture is characterized by minimal hardware utilization, which contributes to lower power consumption (128mW at 200 MHz), utilizing only 61 slices compared to the 9430 slices required by [11]. Table 3 demonstrates that the input ranges of the two approaches for hyperbolic functions are comparable. The findings indicate that the highest output error recorded is 7.042 at an output value of 9517.2, with this maximum error being normalized to 7.4×10^{-4}, equivalent to 0.074% of the output value. In addition, the maximum output error observed is 1.5225, representing 0.0138% of the highest output value, which is 11.013×10^{3}. As previously mentioned, the work of Fu et al. [11] emphasizes precision, achieving near-accuracy with an error probability of only 0.4%. In contrast, our architecture for hyperbolic functions prioritizes minimizing power consumption, reducing footprint and latency, and broadening the input range, albeit at the cost of some precision.

Table 4 demonstrates the comparisons of the LUT, stochastic computing, and modified CORDIC methods with Approx-CSF. Ref. [4] demonstrated that it is possible to compute trigonometric and hyperbolic functions with an accuracy of 4 bits using 77-bit look-up tables. This method requires an exponentially increasing number of look-up tables to achieve higher precision, while a larger look-up table decreases performance. Stochastic computing is a kind of computing that makes use of stochastic bitstreams; it is characterized by being both energy-efficient and cost-effective [21]. In stochastic computing, accuracy is directly proportional to the number of random numbers used [20], [21]. According to the findings in Table 4, the stochastic computing architecture [21] provides the highest latency, area, and error rate. In contrast to the others, it has a significant latency and a small input range [0, 1]. Consequently, the latency of the methods in [11] and [20], and this paper is 10240 ns, 297.472 ns, and

TABLE 2. Comparison of the different architectures of approximate exponential function $e^{\pm z}$.

Feature	[35]	Proposed
Accuracy	55 floating-point	16 fixed-point
Delay	4.348 ns	4.615ns
Max. frequency	230MHz	216.68MHz
Latency (clocks)	23	4
Throughput	200Mexp/sec @200MHz	200Mexp/sec @200MHz
Platform	Virtex-5 XC5VFX70T	Virtex-5 XC5VFX110T
LUT	1035	95
FF	1446	141
DSP48		5
BRAM	18Kb/36Kb	-
slices	-	50
Dynamic Power	-	269mW @200MHz
Total power	-	1.171W
Input Range	[-709, 709]	[-10.397,10.397]
Minimum Error	0	0
Maximum Error	1.11e-16	14.08
Std(σ)	-	1.304
MSE	-	1.855
MAE	-	0.393
Pe	-	0.08

TABLE 3. Comparison of the different architectures of approximate hyperbolic functions.

Feature	[10]	Proposed	Saving/ Costing
Accuracy (bits)	128 Floating	16 fixed	-
Delay(ns)	320	4.945	-98%
Max. frequency (MHz)	3.13	202.224	+98%
Latency (clocks)	32	6	-81%
Throughput Mhyp/sec	3@3MHz	200@200MHz	+99%
Platform	Virtex-7 XC7VX485		-
LUT	29172	157	-99%
FF	512	129	-75%
DSP48	-	6	-
slices	9430	61	-99%
Dynamic Power	-	128mW@200MHz	-
Total power	-	369mW	-
Input Range	[-11.1,11.1]	[-10.397,10.397]	-
Minimum Error	0	0	-
Maximum Error	2^{-113}	7.042	+100%
Std(σ)		1.304	-
MSE		1.855	-
MAE		0.393	-
Pe	0.004	0.16	+98%

19.78 ns, respectively. Significant delays, area requirements, and energy consumption are inevitable consequences of high-precision computations because of the massive amount of stochastically produced data.

To sum up, the use of a certain architecture depends on the applications and costs. We promote adopting stochastic computing for applications that need low power, cost, accuracy, and range and where latency is unimportant. We advocate using the method described in [4] for applications that aren't as concerned about power consumption or cost. We encourage utilizing our recommended solution for cases where all metrics are crucial.

TABLE 4. Comparison the architectures of hyperbolic functions for the proposed, LUT, stochastic computing, and cordic.

Feature	LUT method [4]	Stochastic Computing [20]	Modified CORDIC [11]	Proposed
Accuracy (bits)	16	8	128	16
LUT Volume	77 ×14	-	136×128	Entry Depth=16
Slices	-	4	9430	61
Max. Error	-	Pe=5.15%	2^{-113}	E= 7.042, Pe=16%
Range	[0,10]	[0,1]	[-11.1,11.1]	[-10.397,10.397]
Latency	-	256	32	4
Delay(ns)		1.162	320	4.945

VI. CONCLUSION AND FUTURE WORK

In this study, we introduced efficient frameworks for computing the exponential and hyperbolic functions through using a table-driven algorithm approach. The proposed designs are characterized by their acceptable accuracy, low cost, low power consumption, and low latency. The experimental findings of the proposed method show considerable improvements over the previously reported best designs in terms of performance, error, and hardware cost metrics. Moreover, the proposed architecture offers versatility and scalability, facilitating enhanced accuracy at larger scales.

In future developments, we aim to enhance both the delay and accuracy of our work through the implementation of the following four strategies.

1. The output range of the exponential and hyperbolic function can be extended by splitting the input argument into two or more parts: the integer and fraction parts to be represented in the combination of the look-up table and this table-driven method as follows:
   when $x = n_1 + n_2 + f \rightarrow$
   $$y = \exp(x) = \exp(n_1) * \exp(n_2) * \exp(f) \qquad (10)$$
   where n_1, n_2 are integers, and f is fraction of input argument x. Furthermore, by splitting the integer lookup table into multiple sections, we can significantly reduce the total storage requirement.
2. In future work, we aim to enhance computational efficiency by minimizing multiplier use, especially for constant-factor operations. We plan to replace the first two multipliers with shift operations and an adder tree, optimizing input scaling and resource usage. Furthermore, we intend to substitute the third multiplier with a barrel shifter, driven by specific "m" values, to easily adjust the outputs of the LUT in our design. This modification makes the computation less complex. It also makes the

process faster. Such improvements are valuable for real-time signal processing tasks.

3. In the proposed method, we may employ approximation computation, such approximate adders and multipliers, with a tolerable loss of precision to reduce the latency. In this case, increasing the precision (bits) of the input arguments does not result in a corresponding increase in delay or reduction in maximum frequency.
4. The exponential function exhibits a unique scaling property where input values can be scaled, allowing for calculations within a limited range to be extended through appropriate scaling. This feature allows for efficient computation, especially when dealing with a wide range of input values. This property is based on the mathematical principle that the exponential function can be scaled and shifted in a way that preserves its essential characteristics. This is particularly useful when working with hardware implementations, where computing the exponential function directly over a large input range might be computationally expensive or impractical.

REFERENCES

[1] L. Alzubaidi, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, p. 53, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[2] H. S. Ilango, M. Ma, and R. Su, "A FeedForward–convolutional neural network to detect low-rate DoS in IoT," *Eng. Appl. Artif. Intell.*, vol. 114, Sep. 2022, Art. no. 105059, doi: 10.1016/j.engappai.2022.105059.

[3] R. H. Hadi, H. N. Hady, A. M. Hasan, A. Al-Jodah, and A. J. Humaidi, "Improved fault classification for predictive maintenance in industrial IoT based on AutoML: A case study of ball-bearing faults," *Processes*, vol. 11, no. 5, p. 1507, May 2023, doi: 10.3390/pr11051507.

[4] H. de Lassus Saint-Geniès, D. Defour, and G. Revy, "Exact lookup tables for the evaluation of trigonometric and hyperbolic functions," *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 2058–2071, Dec. 2017, doi: 10.1109/TC.2017.2703870.

[5] H. Magalhães, "An optimization approach to generate accurate and efficient lookup tables for engineering applications," in *Proc. 6th Int. Conf. Eng. Optim.*, H. C. Rodrigues, J. Herskovits, C. M. Mota Soares, A. L. Araújo, J. M. Guedes, J. O. Folgado, F. Moleiro, and J. F. A. Madeira, Eds. Berlin, Germany: Springer, 2019, pp. 1446–1457, doi: 10.1007/978-3-319-97773-7_124.

[6] L. Deng, C. Chakrabarti, N. Pitsianis, and X. Sun, "Automated optimization of look-up table implementation for function evaluation on FPGAs," in *Proc. SPIE*, Sep. 2009, pp. 353–361, doi: 10.1117/12.834184.

[7] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using Taylor series," in *Proc. NORCHIP*, Oct. 2014, pp. 1–4, doi: 10.1109/NORCHIP.2014.7004740.

[8] D. Wu, T. Chen, C. Chen, O. Ahia, J. S. Miguel, M. Lipasti, and Y. Kim, "SECO: A scalable accuracy approximate exponential function via cross-layer optimization," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2019, pp. 1–6, doi: 10.1109/ISLPED.2019.8824959.

[9] L. Chen, J. Han, W. Liu, and F. Lombardi, "Algorithm and design of a fully parallel approximate coordinate rotation digital computer (CORDIC)," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 3, no. 3, pp. 139–151, Jul. 2017, doi: 10.1109/TMSCS.2017.2696003.

[10] E. Manor, A. Ben-David, and S. Greenberg, "CORDIC hardware acceleration using DMA-based ISA extension," *J. Low Power Electron. Appl.*, vol. 12, no. 1, p. 4, Jan. 2022, doi: 10.3390/jlpea12010004.

[11] W. Fu, J. Xia, X. Lin, M. Liu, and M. Wang, "Low-latency hardware implementation of high-precision hyperbolic functions Sinhx and Coshx based on improved CORDIC algorithm," *Electronics*, vol. 10, no. 20, p. 2533, Oct. 2021, doi: 10.3390/electronics10202533.

[12] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959, doi: 10.1109/TEC.1959.5222693.

[13] S. R. Chiluveru, M. Tripathy, and B. Mohapatra, "Accuracy controlled iterative method for efficient sigmoid function approximation," *Electron. Lett.*, vol. 56, no. 18, pp. 914–916, Sep. 2020, doi: 10.1049/el.2020.0854.

[14] H. Dong et al., "PLAC: Piecewise linear approximation computation for all nonlinear unary functions," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 2014–2027, Sep. 2020, doi: 10.1109/TVLSI.2020.3004602.

[15] F. Lyu, Y. Xia, Z. Mao, Y. Wang, Y. Wang, and Y. Luo, "ML-PLAC: Multiplierless piecewise linear approximation for nonlinear function evaluation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 4, pp. 1546–1559, Apr. 2022, doi: 10.1109/TCSI.2021.3133931.

[16] M. Chandra, "On the implementation of fixed-point exponential function for machine learning and signal–processing accelerators," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 39, no. 4, pp. 64–70, Aug. 2022, doi: 10.1109/MDAT.2021.3133373.

[17] P. Li, H. Jin, W. Xi, C. Xu, H. Yao, and K. Huang, "A reconfigurable hardware architecture for miscellaneous floating-point transcendental functions," *Electronics*, vol. 12, no. 1, p. 233, Jan. 2023, doi: 10.3390/electronics12010233.

[18] M. Osta, A. Ibrahim, and M. Valle, "FPGA implementation of approximate CORDIC circuits for energy efficient applications," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 127–128, doi: 10.1109/ICECS46596.2019.8964758.

[19] K. K. Parhi and Y. Liu, "Computing arithmetic functions using stochastic logic by series expansion," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 44–59, Jan. 2019, doi: 10.1109/TETC.2016.2618750.

[20] L. Huai, P. Li, G. E. Sobelman, and D. J. Lilja, "Stochastic computing implementation of trigonometric and hyperbolic functions," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Oct. 2017, pp. 553–556, doi: 10.1109/ASICON.2017.8252535.

[21] T.-K. Luong, V.-T. Nguyen, A.-T. Nguyen, and E. Popovici, "Efficient architectures and implementation of arithmetic functions approximation based stochastic computing," in *Proc. IEEE 30th Int. Conf. Application-specific Syst., Architectures Processors (ASAP)*, Jul. 2019, pp. 281–287, doi: 10.1109/ASAP.2019.00018.

[22] P.-T. P. Tang, "Table-driven implementation of the exponential function in IEEE floating-point arithmetic," *ACM Trans. Math. Softw.*, vol. 15, no. 2, pp. 144–157, 1989, doi: 10.1145/63522.214389.

[23] P. da Costa, M. da Rosa, G. Paim, E. da Costa, R. Soares, and S. Bampi, "An efficient exponential unit designed in VLSI CMOS with custom operators," in *Proc. 29th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Oct. 2022, pp. 1–4, doi: 10.1109/ICECS202256217.2022.9970960.

[24] J. Chen and X. Liu, "A high-performance deeply pipelined architecture for elementary transcendental function evaluation," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 209–216, doi: 10.1109/ICCD.2017.39.

[25] R. Nandagopal, V. Rajashree, and M. Rao, "Accelerated piece-wise-linear implementation of floating-point power function," in *Proc. 29th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Oct. 2022, pp. 1–4, doi: 10.1109/ICECS202256217.2022.9970828.

[26] T. Ze, F. Feihu, Z. Jun, R. Xianglong, and W. Yang, "High-speed transcendental function operation unit design," in *Proc. IEEE 9th Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)/ IEEE 8th Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2022, pp. 160–165, doi: 10.1109/CSCloud-EdgeCom54986.2022.00036.

[27] T.-B. Juang, S.-F. Hsiao, and M.-Y. Tsai, "Para-CORDIC: Parallel CORDIC rotation algorithm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 8, pp. 1515–1524, Aug. 2004, doi: 10.1109/TCSI.2004.832734.

[28] L. Chen, F. Lombardi, J. Han, and W. Liu, "A fully parallel approximate CORDIC design," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2016, pp. 197–202, doi: 10.1145/2950067.2950076.

[29] H. Yu, G. Yuan, D. Kong, L. Lei, and Y. He, "An optimized method for nonlinear function approximation based on multiplierless piecewise linear approximation," *Appl. Sci.*, vol. 12, no. 20, p. 10616, Oct. 2022, doi: 10.3390/app122010616.

[30] J. Partzsch et al., "A fixed point exponential function accelerator for a neuromorphic many-core system," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4, doi: 10.1109/ISCAS.2017.8050528.

[31] A. Dalloo, "Enhance the segmentation principle in approximate computing," in *Proc. Int. Conf. Circuits Syst. Digit. Enterprise Technol. (ICCSDET)*, Dec. 2018, pp. 1–7, doi: 10.1109/ICCSDET.2018.8821112.

[32] U. S. Patankar, M. E. Flores, and A. Koel, "Novel data dependent divider circuit block implementation for complex division and area critical applications," *Sci. Rep.*, vol. 13, no. 1, p. 3027, Feb. 2023, doi: 10.1038/s41598-023-28343-3.

[33] L. Feng, Z. Li, and Y. Wang, "VLSI design of SVM-based seizure detection system with on-chip learning capability," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 171–181, Feb. 2018, doi: 10.1109/TBCAS.2017.2762721.

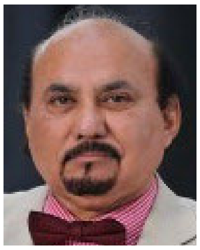
[34] L. Feng, Z. Li, Y. Wang, and C. Wang, "A fast on-chip SVM-training system with dual-mode configurable pipelines and MSMO scheduler," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4230–4241, Nov. 2019, doi: 10.1109/TCSI.2019.2929054.

[35] W. Yuan and Z. Xu, "FPGA based implementation of low-latency floating-point exponential function," in *Proc. IET Int. Conf. Smart Sustain. City (ICSSC)*, Aug. 2013, pp. 226–229, doi: 10.1049/cp.2013.2022.

AYAD M. DALLOO received the B.Sc. and M.Sc. degrees in electronic and communication engineering. He is currently pursuing the Ph.D. degree with the Electrical Engineering Department, University of Technology, Iraq. He is also a Faculty Member with the Communication Engineering Department, University of Technology. His research interests include approximate computing and machine learning.

AMJAD JALEEL HUMAIDI received the B.Sc. and M.Sc. degrees in control engineering from the Al-Rasheed College of Engineering and Science, in 1992 and 1997, respectively, and the Ph.D. degree with a specialization in control and automation, in 2006. He is a Professor with the Engineering College, University of Technology, Iraq. His fields of interests include adaptive, nonlinear and intelligent control, optimization, and real-time image processing.

AMMAR K. AL MHDAWI received the Ph.D. degree in electronic and electrical engineering from Brunel University London. He is an Assistant Professor with the Department of Computer Science and Engineering, Edge Hill University, U.K. He possesses a strong academic background and extensive expertise in the field of robotic mechatronic design (aerial, underwater, and ground systems), control engineering, machine learning, and embedded systems.



HAMED AL-RAWESHIDY (Senior Member, IEEE) is a renowned Professor of communications engineering with the University of Technology, Baghdad. He has advanced qualifications from the University of Glasgow and the University of Strathclyde, U.K. He has a rich career, including roles with the Space and Astronomy Research Centre, Iraq; PerkinElmer, USA; Carl Zeiss, Germany; British Telecom, U.K.; and various universities such as Oxford University, Manchester Metropolitan University, and Kent University. Currently, he directs the Wireless Networks and Communications Centre and Postgraduate Studies in Electronic and Computer Engineering, Brunel University London. He has published over 370 papers and edited the first book on radio over fibre technologies for mobile communications networks. He is also a consultant of global telecom companies and a principal investigator of significant research projects. His current research focuses on advanced technologies in communications engineering, including 5G and 6G developments, quantum computing, AI, and IoT applications.