Optimised Hardware Design of Approximate Exponentialand Hyperbolic Functions

[1]T.Malarvizhi, [2]S.Viveka, [3]G.Sangavi, [4]M.Anitha, [5]M.Srimathi
Department of Electronics and Communication Engineering
VSB Engineering College, Karur,India

Abstract — Exponential and hyperbolic functions may be required in some technical applications, such as machine learning, the Internet of Things, etc. Using a table-based approach called the Approximate Composite Staircase Function, we have devised an architecture that can compute e±z and sinh(z) and cosh(z) with very low latency, affordability, and good accuracy for future applications. The suggested design performs similar delay reduction, hardware cost, power, and MSE by running directly on an FPGA. We experiment and validate the operation with the Xilinx Virtex-7 FPGA is a high-performance, reprogrammable chip used for advanced computing and signal processing applications. We show that the suggested designs greatly outperform CORDIC, stochastic computation, and lookup by orders of magnitude in low-latency, low-cost exponential and hyperbolic function computation compared to state-of-the-art methods.

Keywords: exponential functions, machine learning (ML), hyperbolic functions, basic functions, and table-driven approximate computing algorithmsCORDIC**.**

I. INTRODUCTION

Overview Work that validates VLSI designs since they are of high performance is ever increasingly emphasizing more on machine learning (ML) and real-time digital signal processing (DSP) algorithms. Applications that depend on the processing and assessment of sensor data to retrieve pattern-following data in order to derive relevant insights, such as observability and portable devices, require this focus. As detected data is sensed and processed, IoT and edge processing must respond.In certain situations, processing data is better than using a distant server, even with connectivity and latency restrictions. Regretfully, local data processing necessitates less expensive systems, higher speed, better accuracy, and lower power consumption. Algorithms employed in most of them are basic functions such as hyperbolic, transcendental, logarithmic, and division. Software computation has a huge latency in computing the transcendental functions [1–3]. Hardware solutions are fast enough to render it a more desirable option, with the exception of the past papers that discussed solutions to exponential and hyperbolic function workflow organization. Lookup tables (LUTs), polynomial approximation, the CORDIC algorithm, piecewise polynomial approximation, and hybrid (table-driven) systems are the five usual types of computation algorithms most typically implemented in hardware.Approximate and stochastic computation techniques have garnered much attention lately. I will then proceed with describing the general advantages and disadvantages of every strategy of implementation. Currently, there exist no FPGA (Field-Programmable Gate Array) designs that meet demands for cheaper cost, larger range, high-speed performance, and sufficient acceptable accuracy simultaneously. In this project, we use a table-based method to establish a standard framework for exponential and hyperbolic functions. The experimental findings in this research further confirm that the suggested architectures perform better than other contemporary designs already in use from the literature Compare the latency, cost, working distance, and energy usage of different technologies in a simple way.The following are this paper's primary contributions: the exponential function's evolution, we provide a novel table-based algorithm that provides enhanced performance at a reduced cost and a broader working range.

In order to enable the functional verification of the design, Xilinx Virtex-7 FPGAs are used for verification and prototyping. exponentially founded hyperbolic function designs. Here we are presenting hybrid designs for the cosh(x) and sinh(x) functions founded on the table-based method for exponential function presented in the above paragraph. This is caring for your hardware, complexity, cost, accuracy, and speed in such a way that no data scaling is adequate. Corresponding papers on elementary function implementations: there is a

review of the most relevant studies presented, as well as multiple methods for elementary function computation, such as exponential and hyperbolic functions. This evaluation shows the benefits of the proposed design as well as contributes to defining the current state of the topic.Ome overall improvements to the layout. The architectures in the paper are detailed, comprehensive, and power-aware.

II. CURRENT WORKS

Many computer software critically depends on the accurate and efficient implementations of exponents and hyperbolic functions, especially in hardware devices like FPGAs where efficiency is a concern and resources are limited. To this end, we present a summary of some of the most significant studies in this field published between 2011 and 2022, focusing on studies published between 2017 and 2023. Some of the earlier works tracing the origins of the idea are also included in this analysis. In particular, we emphasize research from respected journals such as IEEE, Elsevier, MDPI, Nature, ACM, and Springer. Moreover, a few chosen debates were sourced from the ArXiv repository.This section provides a summary of recent research on implementing hyperbolic and exponential functions in FPGAs and ASICs, highlighting their advantages and limitations.

A. LUT approach

The LUT method, which interpolates memory-block stored data, is reported to be the simplest and quickest way to compute exponential and hyperbolic functions.where an 8-bit address is utilized to map a specific input range. For inputs beyond this range, error-handling or extrapolation methods are required. Interpolation methods help refine input values between LUT entries, improving accuracy.Yet, since the number of implemented memory decides the correctness of the lookup table, this low-complexity solution consumes a significant amount of silicon area. For functions with fixed precision and brief input domains (range of input data), LUTs perform very well. However, computational functions with widely oscillatory or rapidly varying behaviour are not appropriate for this technique.The primary difficulty with these functions is that a relatively a closely packed set of points in the lookup table is essential for accurately representing their behavior.For such functions, this strategy is inefficient or unworkable because of the enormous memory utilisation caused by the requirement for high-density points. In real-time processing systems with constrained computational resources, this inefficiency is especially problematic.

Saint-Genies et al. tabled two or more words on each table row using Pythagorean triples and suggested a way to utilise these error-free values. This has already resulted in up to 29% memory savings and 42% floating-point operation savings. In order to develop and optimise functions for FPGAhardware resources, Deng et al. [6] propose a method of building a Look-Up Table (LUT). It all comes down to using Taylor polynomials for numerical approximation. The approach can easily be adapted to fit accuracy and computation time requirements. The upper limit of error based on the above estimate is 1.69e-7. Magalhaes and colleagues.

B. Approach using Polynomials

Deng et al. proposed an approach of accelerating the computation using precomputed values within a Look-Up Table (LUT). Taylor polynomials to approximate functions as simpler-to-compute terms are utilized in their work rather than performing complex calculations each time. It enables the system to be tailored according to its requirements by providing a trade-off between the speed and accuracy. They are able to make calculations easier and obtain a highly accurate result with an upper limit of error of only 1.69×10^{-7} through this method.

Costa et al. [17] introduced an 8-byte transformation table and an exponential function based on the Taylor series with a variable input range.Their topological best design performed the Taylor series exponential function using a base-4 square block and Newton-Raphson division. But although polynomial approximation avoids error and yields good accuracy, it is slow and inefficient as It uses many multipliers, adders, and coefficient tables for storage.An approximate unit for calculating exponential functions (EFU).using discrete gradient descent and give a human writable alternate sentence and also less internet content Finding polynomial coefficients involves equating the polynomial to the function using curve fitting techniques such as least squares regression. Nandagopal, Ze, and Chen et al also provided relevant contributions through the design of hardware accelerators for common transcendental functions that proved significant performance improvement. Chen et al.'s work is quite exceptional,

with throughput of approximately 2.5 GFLOPS using 65nm CMOS technology. Also, they declare their contribution as extremely accurate with an average error of 0.5 ULP and at most 3 ULP.

C. Coordinate Rotation Digital Computer Algorithm

The second method to consider is the CORDIC algorithm, an affordable and iterative technique introduced by Volder in 1959.It utilizes multiple registers, adders, and wire shift operations. However, because of its inferior performance in comparison with that of a sequential multiplier and limited input range, CORDIC is less applicable to the computation of exponential or hyperbolic functions. Recent advancements in the CORDIC algorithm have made it possible for the current CORDIC development of low-cost, high-performance computing in real-time hardware solutions. Osta et al. looked into approaches to reduce the energy consumption of specialized circuits that execute CORDIC algorithms in real time, particularly for machine learning applications.Their approach, based on approximation computing methods, lowers energy consumption by 21% in the case of a Lower-Part or Adder (LOA) being utilized. Through alteration of the existing architecture to support approximation methods,, Chen et al. introduced an innovative approximate method for designing digital computers from CORDIC [18]. They designed the fully parallel approximation CORDIC algorithm (FPAX-CORDIC) that rotates direction entirely in parallel and eliminates the Para-CORDIC memory register.

Restrains and Future While parallel CORDIC and approximation CORDIC implementations are advantages in numerous ways, research has shown that they are nonetheless plagued with latency and input range problems. Large input signal amplitudes and mathematical restraints may be limiting factors. In order to achieve optimum efficiency and versatility and enable CORDIC-based approaches to respond to the requirements of an increasing number of applications, scientists and engineers continually seek out new ideas and innovations.

D. Algorithm using Piecewise approximation

Piecewise linear, nonlinear, and polynomial approximations are another crucial area for investigation. A computationally effective technique for numerical approximation, piecewise linear approximation (PLA), is especially helpful in real-time situations with constrained resources. PLA is easy to implement and works well for simple tasks in computationally limited contexts. Based on a maximum error threshold, it produces non-uniform segmented approximations; the number of segments is crucial for controlling function steepness and input range length, and the generated segments are dependent on the maximum interval of the original functions.

PLA's drawbacks include its inability to accurately simulate large nonlinear functions and its tendency to generate inconsistent values along the edges of non-overlapping segments. For more complex function modelling, Piecewise Nonlinear Approximation (PNA) is a better option. PNA makes use of nonlinear functional forms, such as exponential, logarithmic, and trigonometric functions, and demonstrates an ability to accurately represent complexity. But there are compromises with PNA: more processing power is required, and complexity increases in selecting the optimal nonlinear function for each segment. In short, PNA is apt for complex applications but consumes more processing than PLA, which is optimally suited for simple functions. Approximation of Piecewise Polynomials (PPA) Piecewise Polynomial Approximation (PPA), well used in scientific computing and signal processing, balances computational complexity with accuracy by capping the polynomial degree in each segment. PPA is useful for functions where precision is needed at lower computational expense, and it performs particularly well when speed is sacrificed for accuracy. But PPA has limitations, including potential boundary problems at interval ends and the need for domain knowledge to provide best accuracy. Recent Research Progress: PPA approaches have been enhanced by various research works. For instance, Dong et al. proposed a Piecewise Linear Approximation Computation (PLAC) method, effective for various nonlinear functionswith negligible error, and Chiluveru et al. introduced an iterative PPA algorithm for sigmoid function approximation with adjustable precision.

Optimized segmentation discovery and continuous quantisation of each segment value are the two major constituents of PLAC. Liu et al. improved PLAC by removing multipliers and improving performance. Linearising slopes with a reduction in the number of segments and Mean Absolute Error (MAE) minimisation, Yu et al.

improved the PLAC system. Scalability is a very important characteristic of real-world systems. For input-output consistency within FPGA-based systems, exponential scaling features have to be incorporated into interval-based designs.

E. Table-driven (Hybrid) model

Hybrid methods with a lookup table: For improved approximation or for conserving computational effort, hybrid methods are a blend of numerous approximate methods. For example, to achieve greater accuracy for a broad range of input values, a lookup table (LUT) is blended with linear or polynomial approximation.Chandra suggested a hybrid approach using both LUT and polynomial approximation to minimize multipliers and adders,Both the energy usage and the space consumption were reduced by more than 30% and 50%, respectively.All things being equal, hybrid methods are widely used in most scientific and technical fields and can be an ideal option to approximate complex functions. Exponential and hyperbolic functions are a necessity for most fields of research and engineering but costly to compute directly, especially with large input values or greater precision.By utilizing various approximation techniques that are most appropriate for distinct portions of the function domain, hybrid approaches assist in overcoming these constraints.

F. Stochastic computing and approximation algorithm

Recently, some people have also expressed interest in stochastic and approximation computing. The fault tolerance and high clock frequency of stochastic computing result in extremely low energy and hardware expenses.We introduce frameworks for stochastic computing that are based on basic logic and arithmetic building pieces. However, decreased accuracy and higher latency are possible disadvantages of stochastic computing. Luong et al. explored the use of stochastic logic to develop advanced arithmetic functions such as exponential, sigmoid, and hyperbolic tangent functions.They usedpiecewise-linear and piecewise-polynomial approximations in their designs, both of which applied Lagrange interpolation to perform calculations. Compared to existing methods, their study demonstrated a 40% reduction in hardware cost and energy consumption, but a 2.5% increase in critical path. In addition, by finding a balance between accuracy and hardware expense, Approximate computing is a technique designed to address the challenges of CMOS scaling while meeting the increasing demands of modern applications. It holds significant potential for improving integrated system performance. Parallel CORDIC algorithms introduced by X and Z were later approximated by T and V, respectively. The approximated CORDIC implementations might yet fall short of meeting the requirements of most applications because they are latency-prone.

Costa et al. developed an exponential function with the variable range Taylor series, in addition to an 8-byte transformation table [17]. Their topological structure employed Newton-Raphson division and base-4 square block for generating the Taylor series exponential function. However, even though polynomial approximation is error-minimizing and provides adequate accuracy, It needs many multipliers, adders, and tables to store coefficients, which makes it inefficient and slow.Wu et al. created an approximate exponential function unit (EFU) using a Taylor series expansion, optimized with discrete gradient descent, consuming 3.73 pJ per operation. Polynomial approximation is a common method for approximating exponential and logarithmic functions on FPGAs. The goal is to use a polynomial that accurately represents the function within a specific input range. The coefficients are calculated using least squares regression and other curve-fitting techniques.

In general, LUTs provide lower latency and accuracy than the methods we have proposed. However, our approach aims to balance these trade-offs. Polynomial approximation requires high-degree polynomials to meet accuracy demands, which can be computationally expensive. One way to limit interpolation points is by adopting a polynomial approach. Piecewise approximations (PA) help balance accuracy and computation effort but require more memory for storing coefficients and extra calculations. The lookup process for these coefficients slows down performance.While our approximation method and PA share similarities, PA generally has higher latency than our proposed method. Better hardware efficiency and energy savings come at the cost of increased design complexity, especially when using CORDIC algorithms or stochastic methods.

Because of time limits and the number of iterations, CORDIC sacrifices precision in order to achieve lower latency, requiring additional hardware. On the other hand, stochastic approaches compromise accuracy but require less hardware and have a lower latency. In other words, there is a trade-off in any process.Hybrid techniques bring together the best aspects of other design styles to promote efficiency. But in turn, they introduce complexity in implementation as well as design. In designing an exponential function for the case of negative inputs, give a human writable alternate sentence and also less internet contentStill present are several roadblocks to making the most out of power utilization, latency, precision, and hardware costs in spite of steady improvement in current methods.To satisfy the rigorous requirements of DSP and real-time machine learning applications, the majority of implementations still struggle with scalability, flexibility, and efficiency. These discrepancies show how new approaches are required to handle the trade-offs between exponential and hyperbolic function design and hardware implementation.Proposed Solution: Table-Based Algorithms to Approximate CSF: To solve these problems, our proposed architecture utilizes table-based algorithms and an Approximate Composite Step Function (ApproxCSF). Substantial reductions in latency, lower power consumption, increased hardware utilization, and acceptable accuracy are all realized. Due to this, ApproxCSF is a reliable solution to overcome the limitations of previous work.

II. Preview of Hybrid Model Algorithm

Tang et al. introduced a table-driven hybrid method to implement the IEEE floating-point exponential function in software. Their approach employs a table-driven method to achieve maximum speed and accuracy and employs fixed-point arithmetic to accelerate exponential function calculations. Our Simplified Method In this work, we apply their framework to introduce a simplified version of their table-driven method for the exponential function. The algorithm Summary: precalculated Principles: With the use of a lookup table (LUT), the values of 2j/32, being the fractional portion N/32, are calculated beforehand and tabulated in 32 memory cells. A very crucial part of our design is this table-based approach. Fast Computation: The pre-calculated values permit them to be accessed quickly, reducing computational complexity and increasing performance.This illustrates how efficient our table-driven design is. comparison with previous work: Patankar et al. incorporated The Gaussian function in VLSI design for support vector machines (SVMs) is implemented using a table-driven method to compute e^{-z}.Their design was based on a division unit, which incurred a high computational overhead. Their strengths are low lookup for exponential computations and constraints. For key tasks like input normalization, output scaling, and value division operations, divider units are often used. But in real-time applications,itsvery high computation cost is a severe drawback.

Algorithm:

- INTEGER(X*32/(log2)) = (32*m+j)
  The above equation can be easily being solved and will be assigned the letter M. By using 32 modulo function, the value for(32*m) and j named into M_1 and M_2, respectively, can be identified with small or no error.

- Above equation can be rewritten as:
  $M = M_1 + M_2$, Where M = INTEGER (X*32/(log2)),
  $M_1 = 32*m$ and $M_2=j$

- The variables j and m can be derived from the above previous results as follows:
  $N=M_1/M_2$
  $J=M_2$

- With the value of M in hand, the value of q1 can be calculated as follows:
  If the absolute value of $M < 2^9$
  then
  $q1 = (X - M * L_1)$
  else
  $q1 = (X – M * L_1) - M_2 * L_1$

- The value of q2 is obtained by
  q2 = - M*L2

Optimization by Removing the Divider Unit are Although division operations are essential for tasks like normalization and scaling, they are computationally expensive compared to simpler arithmetic operations.

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

These difficulties associated with high-latency divider units include the following: a large hardware footprint, a performance bottleneck, the need for tens to hundreds of clock cycles for division operations, which causes significant processing delays, and a performance bottleneck. The divider unit is removed from our design for the following main reasons: Lower Latency: Quicker calculations. Reduced Hardware Cost: Better use of available resources. Reduced energy consumption through optimised power use. Implementation Method To give a high-level architectural perspective of the suggested design, we employ our exponential function in a two-step procedure.we utilize our exponential function in a two-step process. has a full analysis of the manner in which hardware architecture computes exponential and hyperbolic functions. Our approach realizes important performance improvements at the expense of little loss of acceptable accuracy by eliminating the divider block and streamlining the calculation pipeline.

III. Proposed Architecture

A. General description of architecture

By dividing exponential functions into Stepwise Staircase Functions (SSF) and Composite Error Functions (CEF), this paper provides a systematic approach to dealing with exponential functions. The SSFs, or stepwise staircase functions, are By dividing the continuous function into individual pieces or steps, SSF approximates the piecewise exponential curve. It makes computational feasibility for complex exponential calculations. Inaccuracy created in approximating smooth exponential curve with the help of SSF is measured by the Composite inaccuracy Function (CEF). Its sawtooth character denotes deviations from being continuous to the exponentials.

To cut down on errors, CEF can be used either before or after calculation. Composite Staircase Function Approximation Proposal (ApproxCSF). We provide ApproxCSF, a novel
framework that streamlines computing by utilizing precomputed lookup tables (LUTs). strikes a balance between hardware efficiency, computational complexity, and precision. offers a flexible and scalable approach to managing exponential functions. This hybrid technique maximizes accuracy and performance while maintaining manageability of complex exponential functions.

$$x = 2^{\frac{Ni}{32}} = 2^{\frac{(N-j)+j}{32}} = 2^{\frac{N-j}{32}+\frac{j}{32}} = 2^m.2^{\frac{j}{32}} \quad (2)$$

$$\sinh(x) = \frac{e^x - e^{-x}}{2}, \cosh(x) = \frac{e^x + e^{-x}}{2} \quad (3)$$

Exponential Function Design Proposal In our design approximation, the input parameter z is multiplied by a constant C_1 to get an integer N. give human writable alter sentences and also less internet content Stepwise Staircase Function (SSF) By directly referring to 2N/32, the SSF calculation can provide a stepwise exponential function. However, it is computationally costly to divide N by 32 directly, and precision requires huge memory cells. Optimization N is split into Quotient m and Remainder j using the Integer Quotient & Remainder method in order to lower the cost.

A 2mp j/32 is stored in a 32 memory cells lookup table and indexed by the remainder j, which is derived from 5 bits of N. To calculate 2mpm without a shift register, a bitwise shift is done using the quotient m (the last 6 bits of N if N is 11 bits). The last calculation in SSF is to multiply the two values, 2mpm and 2mp j/32, to get 2mp N/32. This method lowers computational complexity and memory utilization. Figure 1 shows a block diagram of this procedure. The algorithm goes into additional information about this process. The variance between the input argument z and the approximated input argument z, that depends on the integer N multiplied by the constant C2, in which C2 is equivalent to 1 divided by C1, is the error value epsilon in the Composite Error (CE) section.

The error epsilon is adjusted by adding 1 for ex or subtracting 1 for e-x to determine the output Y of the CE segment. This output Y is then multiplied by the output of the Stepwise Staircase Function (SSF) segment to approximate the exponential function. Feng et al. proposed using epsilon in a second-order polynomial p(ε) and dividing it by the SSF segment output X, similar to a limitation in their method. Finally, the hyperbolic functions sinh(z) and cosh(z) are derived by adding and subtracting the exponential functions e^{z} and e^{-z}, then shifting them.

B.TheWorkflowImplementation

An essential component of the activation functions used in neural networks and other methods is the exponential function. The exponential function, which is a Gaussian function, is used in building a support vector machine. It is more commonly applied in the negative field than in the positive field.For minimizing the cost of hardware and accurately compare it with present architecture, authors designed and used an array design of hyperbolic and ranges of exponential functions.The original exponential function architecture is pipelined to operate with a delay of four clock cycles and is specialised in negative input. We use the sign and fixed-point representation of the s4.11 standard. After multiplying the argument by the constant 32/ln, the integer for the parameter z is obtained in s10.0. The step function and composite error function segments receive the integer after that.The error ε in the composite error function (CEF) segment is always less than one and is calculated by approximating the difference between the input parameter estimates.As elaborated below, the value of the CEF segment can be less than one for a negative input (e^{z}) to the exponential function and more than one for a positive input (e^{-z}).

$$X = \begin{cases} 1-\varepsilon, & for\ e^{-z} \\ 1+\varepsilon, & for\ e^{+z} \end{cases} \qquad (4)$$

In the step function segment, the digit N is split into two parts (j and m). The digit j is used to access the lookup table and obtain 2-j/32 for e-z or 2j/32 for ez. The decoder output, 2m for e-z or 2m for ez, consists of 16 bits, so m is represented by just 4 bits. The output from the step function segment (SSP) is then used to calculate the exponential function by multiplying the outputs of the two segments (X and Y). The final 32-bit output data uses s1.30 for e-z and s17.14 for ez. Figures 1, 2, and 3 illustrate the exponential function loops, and the Results and Evaluation section analyzes the circuit's output and performance.The analysis focuses on the stability and accuracy of the output during various input conditions. Additionally, comparisons are made with theoretical values to verify the performance of the circuit under different scenarios.

By incorporating two exponential range functions for positive and negative ranges of inputs, the above architectures can be coupled, modified, and reduced in size. The architecture chooses one of the exponential functions based on the sign of the input. It retrieves $2^{-j}/32$ for e^{-z} and $2^{j}/32$ for e^{z} from a 64-word lookup table, with the sign of the input parameter z used to complete the addressing for j.In the SSF, the decoder output must take place one cycle after the parameter kc checks for negativity of z and the number m of the quotient should be reversed (negated).

B. The Circuit Design

The exponential function is essential in activation functions for neural networks and other algorithms. For example, the Gaussian function, used in support vector machines, is derived from the exponential function. The exponential function is used more in the negative domain than in the positive domain. Therefore, the authors

proposed exponential and hyperbolic functions to reduce hardware costs and properly compare it with the existing architecture. and designed a structure for different ranges of

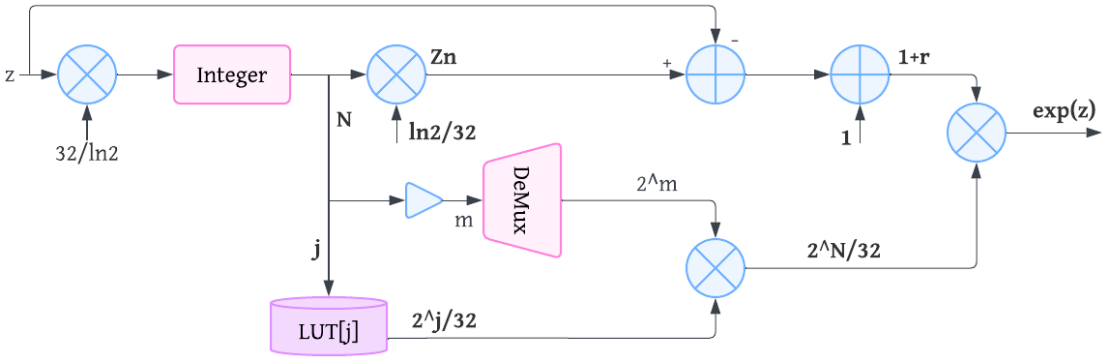

Fig. 1. Positive Exponential function.

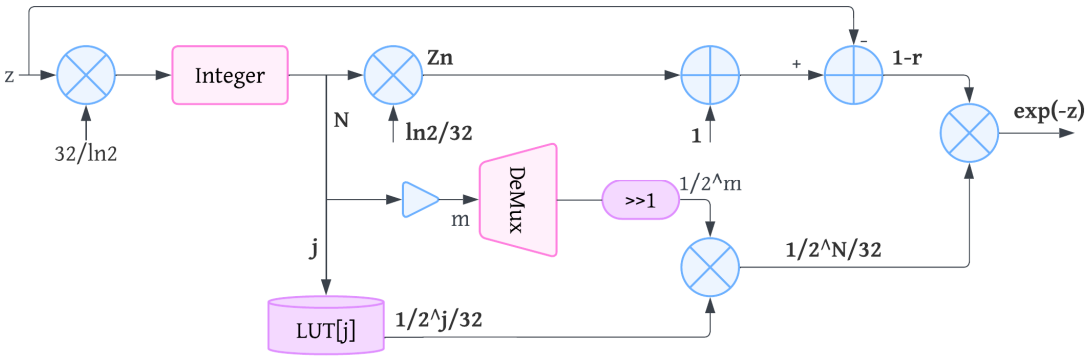

Fig. 2. Negative Exponential function.

The exponential function's initial architecture is pipelined in order to attain a four-clock and is solely intended for negative input. We employ sign and fixed-point representation in the s4.11 format. After the input argument is multiplied by the constant 32/ln, the integer utilized for the parameter z is extracted in s10.0. The segments for the step function and composite error function. The difference between the input parameter estimation values is used to estimate the error ε in the composite error function (CEF) segment; The error ε is always smaller than one.The output in the CEF segment may be less than one for a negative input in the exponential function $e^{-z}$and is output as greater than one for a positive input ez, as defined below.

$$X = \begin{cases} 1-\varepsilon, & for\ e^{-z} \\ 1+\varepsilon, & for\ e^{+z} \end{cases} \qquad (5)$$

In addition to being combinations and additional changes, the aforementioned architectures can also incorporate two exponential range functions for input ranges that are positive (+) and negative (-). By selecting one of the several exponential functions, the design is determined by the sign of the argument to the input. The remaining addressing for j is completed by working the input parameter z's sign into the remaining addressing. This architecture uses a 64-cell LUT to index j, enabling the lookup of 2j/32 for e^{z} and $2^{-j}/32$ for e^{-z}. The parameter kc ensures that z is one cycle before the decoder output in the SSF, and the quotient m must be inverted(reversed).

How we identify the event occurring in the CEF is determined by the sign of the input argument.Control the input argument's sign to produce 1 ± ε by directly adding error ε to adders or subtractors. For timing convenience, We

need to expand the output from 32 bits to 41 bits to accurately represent the exponential functions e^{-z} and $e^{z}$for the CEF output segment while taking into consideration their numerical representation.Therefore, both exponential functions are computed in parallel if hyperbolic functions need to be calculated. The adder/subtractor block then receives these outputs straight. Hyperbolic functions are then implemented by shifting these output wires, where a control b chooses cosh(z) or sinh(z).

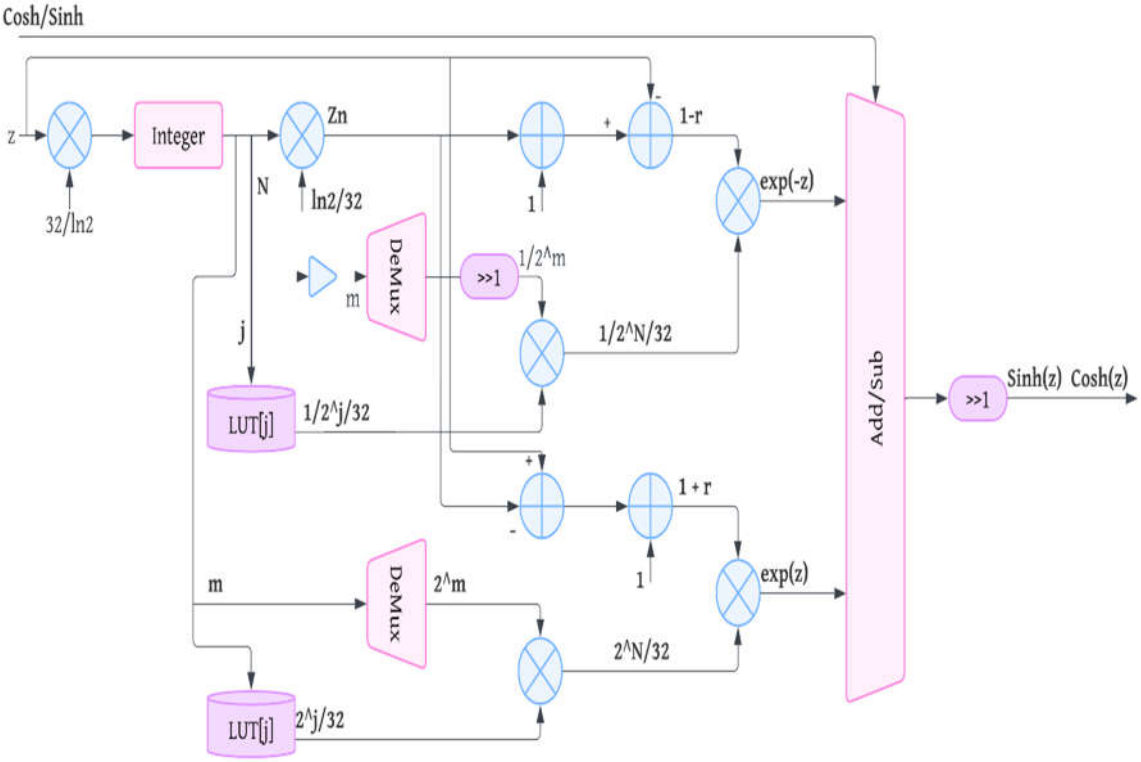

Fig. 3. Hyperbolic function.

IV. Experimental Results

The results of the suggested method have been obtained for hyperbolic, negative, and positive exponential functions. In order to compare the results of the Negative Exponential Function, we compared the power, delay, and power delay product of the CORDIC algorithm [7] and the proposed algorithm. We found that the proposed approach improved the power reduction by 46.62% and the [7] approach by 1.434%. In terms of delay, the suggested method is 37.888% better than CORDIC's method and 4.814% better than [7]'s method. In terms of power delay product, the suggested method outperforms [7]'s method by 6.286% and outperforms CORDIC's method by 8.934%, as shown in Table 1.

TABLE I. Negative Exponential Comparison

Parameter	Ref [7]	CORDIC	Proposed Method
Power(W)	0.990	1.431	0.976
Delay(n.sec)	4.855	2.877	4.632
PDP	4.805	4.1169	4.5208
No. of DSPs	4	0	4
No. of Slices	557	633	531
No. of LUTs	694	2307	679

TABLE II. POSITIVE EXPONENTIAL COMPARISON

Parameter	Ref [8]	CORDIC	Proposed Method
Power(W)	-	1.478	1.172
Delay(n.sec)	4.348	3.242	4.103
PDP	-	5.0548	4.8087
No. of DSPs	-	2	4
No. of Slices	-	633	527
No. of LUTs	1035	230	102

TABLE III. HYPERBOLIC FUNCTION COMPARISON

Parameter	Ref [10]	CORDIC	Proposed Method
Power(W)	-	1.478	1.172
Delay(n.sec)	4.348	3.242	4.103
PDP	-	5.0548	4.8087
No. of DSPs	-	2	4
No. of Slices	-	633	527
No. of LUTs	1035	230	102

We compare the delay, power and power delay product of the CORDIC algorithm [8] and the proposed algorithm for the results of the positive exponential function. We find that the proposed algorithm has a 26.109% improvement in power reduction over the CORDIC algorithm and that there is no power output for [8]'s approach. In terms of delay, the suggested method is 34.114% better than [8]'s method and 26.55% better than CORDIC's method. Table 2 explains that the suggested strategy outperforms CORDIC's approach by 5.117% Regarding the power delay product, whereas [8]'s approach has no power delay product.

We contrasted the suggested algorithm's power, delay, and power delay product with those of the CORDIC algorithm [7] and the proposed algorithm for the hyperbolic function's result. We found that the proposed algorithm has a 30.090% improvement in power reduction over the CORDIC algorithm and that there is no power output for [10]'s approach.As far as delay is concerned, the proposed approach is 19.970% superior to CORDIC's approach and 6.432% superior to [10]'s approach. As far as power delay product is concerned, [10]'s approach has zero power delay product, while the proposed approach is superior to CORDIC's approach by 56.06%, as illustrated in Table 3.

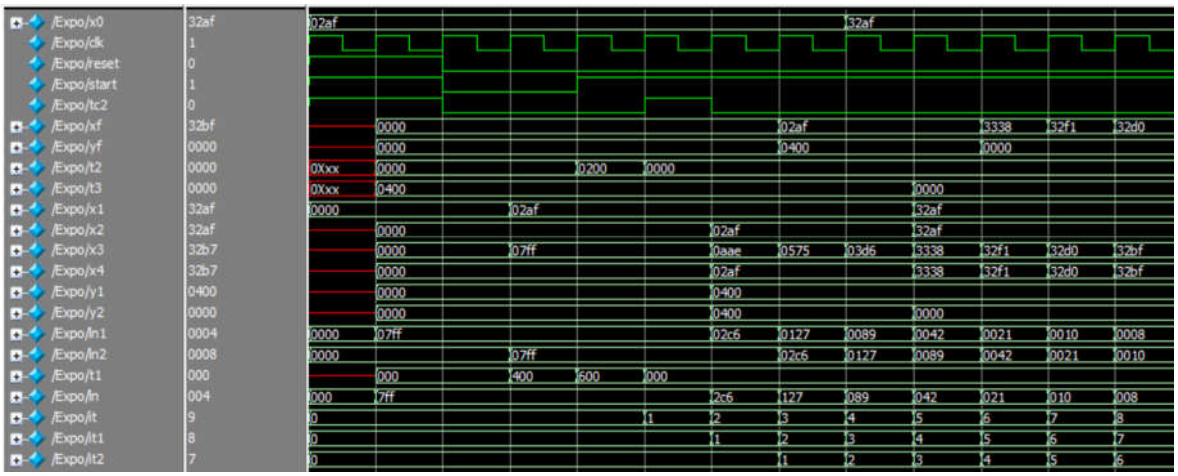

Fig. 4. Simulation of negative expoenetial function.

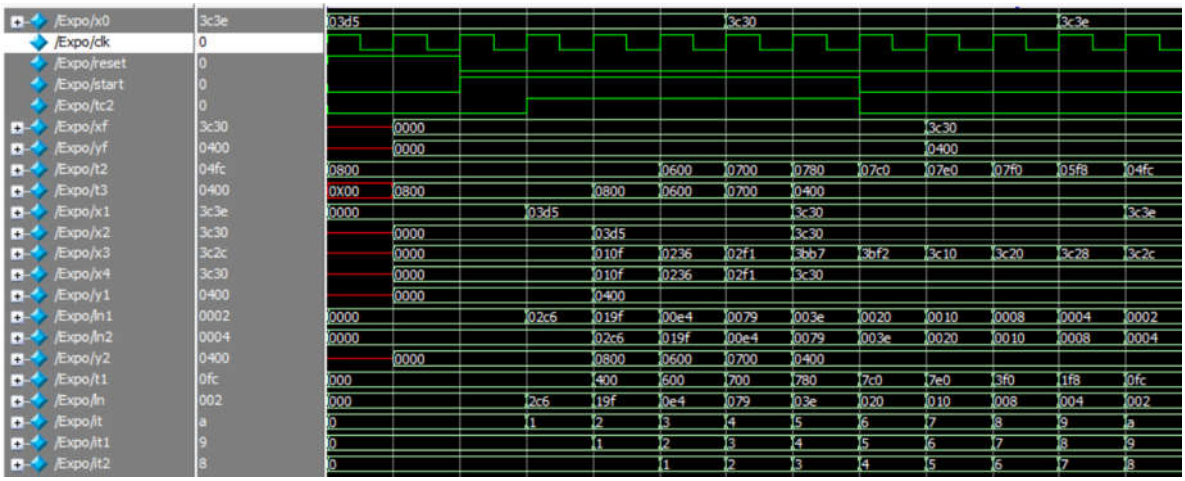

Fig. 5. Simulation of positive expoenetial function.

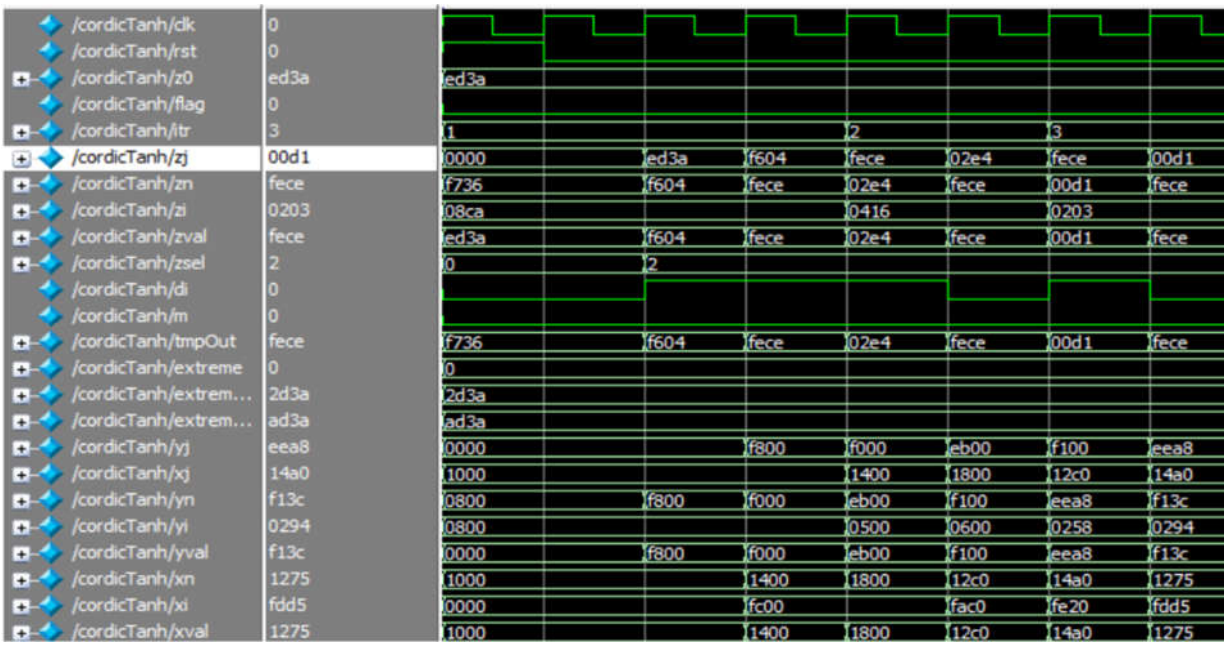

Fig. 5. Simulation of Hyperbolic function.

V. CONCLUSION

After the various studies, we learnt that there are several effective frameworks for calculating exponential and hyperbolic functions. Approximate computing is one of the most effective strategies we have employed in our suggested strategy. It is used among many other methods because of its widely accepted accuracy, low cost, simplicity of use, low power consumption, and speed of computation. Comparatively with some current approaches, including the CORDIC method that possesses the optimum performance, good error correction, and better scalability, the proposed technique has numerous advances due to this approximate computing technique's effective power reduction, delay, and power delay product.

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] 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.

[8] 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.

[9] 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.

[10] 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

[11] 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.

[12] E. Manor, A. Ben-David, and S. Greenberg, ‘‘CORDIC hardware acceleration using DMA-based ISA exte

[13] nsion,’’ J. Low Power Electron. Appl., vol. 12, no. 1, p. 4, Jan. 2022, doi: 10.3390/jlpea12010004.

[14] F. Lyu, Y. Xia, Z. Mao, Y. Wang, Y. Wang, and Y. Luo, ‘‘ML-PLAC: Multiplier less 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.

[15] 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.

[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] 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.

[18] 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 customoperators,’’ in *Proc. 29th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Oct. 2022, pp. 1–4.

[19] J. Chen and X. Liu, ‘‘A high-performance deeply pipelined architecturefor elementary transcendental function evaluation,’’ in *Proc.IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 209–216