Engineering and Technology Journal
Journal homepage: https://etj.uotechnology.edu.iq

Implementation of Hyperbolic Sine and Cosine Functions Based on FPGA using different Approaches

Rawasee K. Yousif*, Ivan A. Hashim, Bassam H. Abd

Electrical Engineering Dept., University of Technology-Iraq, Alsina'a street, 10066 Baghdad, Iraq.
*Corresponding author Email: eee.20.29@grad.uotechnology.edu.iq

HIGHLIGHTS

- Modifying the range of hyperbolic sine and cosine functions from the range (-π/4 to+π/4) to the range (-π to+π).
- Use three ROMs positive integers, positive values,and negative values, to design the exponential.
- System stores half wave of hyperbolic sine and cosine functions in memory, instead of storing full wave.

ARTICLE INFO

Handling editor: Ali J. Salim

Keywords:
FPGA; hyperbolic sine; hyperbolic cosine; area minimization; low latency

ABSTRACT

This paper presents the implementations of the hyperbolic sine and cosine functions, which are essential in many digital systems. In the previous eras, these functions were only implemented in software; however, hardware implementations have recently become more significant due to the performance advantages of hardware systems over software implementations. Therefore, these functions can be hardware implemented using various methods such as the CORDIC algorithm, Taylor series, polynomial approximation technique, and LUT approach. This paper focuses on reducing area utilization (logic components), low latency, and reducing power consumption. Five designs are proposed based on different techniques, in which the ROM approach achieved the best results compared to the other four proposed designs. It also achieved low area utilization, high speed and low power consumption compared to the related works where the ROM approach consumes the resource utilization are as follows: zero flip-flops, (26) occupied slices, and (43) look-up tables. The total power consumed is about (56 mW), and there is a high execution speed of one clock cycle.

1. Introduction

Hyperbolic functions such as the sinh (x) and cosh (x) can be used in different fields, such as scientific computing, and are commonly used in engineering fields, including signal processing, power transmission, aerospace, statistics, etc [1,2]. Earlier, hyperbolic functions were often only implemented in software; however, due to the performance advantages of hardware systems over software implementations, hardware implementations of these functions have gained significance. Various methods can be used to implement the hyperbolic tangent function in hardware, such as the Taylor series, CORDIC algorithm, polynomial approximation technique, and look-up table (LUT) approach. The LUT approach is deemed simple and fast since it computes functions sinh (x) and cosh (x) using stored values in memory blocks via the interpolation method [3,4]. The memory block size for this method must be carefully chosen because the computational accuracy and the resource utilization required cannot be compromised. The Maclaurin series is used to represent functions sinh (x) and cosh (x) in the polynomial approximation technique (x). The Maclaurin series is an infinite sum of derivatives generated from the Taylor series approximation at zero, which necessitates using many multipliers and adders [5, 6]. Although look-up tables can store factorial values, their design area and memory appear inefficient. Only shift and addition operations are used in the COordinate Rotation DIgital Computer (CORDIC) algorithm to compute the functions sinh (x) and cosh (x) [7]. The CORDIC algorithm is an important hardware realization technique since it requires fewer registers and clock cycles to calculate functions sinh (x) and cosh (x) [8-10]. There are no hardware methods among the four mentioned above that properly combine low area reduction, low latency, and high accuracy, which is a critical requirement for several scientific computing applications. Therefore, the primary objective of this research is to implement the sine and cosine hyperbolic function on an FPGA with high performance and low resource utilization. The reason for using the Field Programmable Generic Array (FPGA) platform is due to the following characteristics: it lacks expensive multicore Central Processing Units (CPUs) and is high density and performance.

http://doi.org/10.30684/etj.2023.139756.1440
Received 15 April 2023; Accepted 21 May 2023; Available online 28 June 2023
2412-0758/University of Technology-Iraq, Baghdad, Iraq

In contrast to General Purpuse Processor (GPP), which runs sequentially, FPGA can work in parallel and achieve orders-of-magnitude speedup [11,12]. FPGA is perfect for prototype designs where hardware testing and verification are done immediately on the chip. It also provides the advantages of software flexibility, hardware speed, re-programmability, and minimum time to market. Moreover, design flaws can be corrected without adding more hardware expenditures [13-16].

In this paper, a survey of previous studies has been made, and a lot of recent scientific research on earlier works on hyperbolic functions has been read, especially sine and cosine hyperbolic functions. In [17], the authors presented the fundamental CORDIC method, as well as its application and implementation of a greater variety of hyperbolic functions. For real-time performance, the designed architectures take advantage of high levels of pipelining and parallel processing. With a gate count utilization of 75,151, 1791 (17%), and 3014 (58%) of the flip-flop and the occupied slices, respectively, the design can fit on a single chip and be implemented using Register Transfer Level (RTL) compliant Verilog code. Xilinx's XC2V1000-6bg575 FPGA device is used to implement the design.

The researchers in [18] presented an area-efficient multiplier-less architecture to compute exponential and hyperbolic functions. Its implemented architectures have been synthesized on the Virtex-4 FPGA and Altera Quartus –II kits. Applications that demand high frequency can use this architecture. The design comprises three cases, and as iterations are increased, more hardware resources are used (N). Hence, N = 12 has approximately 40 flip-flops, 175 slices, 52 bonded IOBs, and 330 four-input LUTs. When N = 13, there are about 44 slice flip flops, 176 slices, 56 bonded IOBs, and 333 four-input LUTs; this shows the number of each component type for various values of N. Also, the proposed architecture of logical components comprises 274 elements, and the number of logical elements has been defined as an area parameter.

In the suggested work of Fu et al. [19], they present another way to compute hyperbolic functions. The Quadruple-step-ahead Hyperbolic CORDIC (QH-CORDIC) technique, which the authors proposed, is based on an enhanced CORDIC algorithm and allows for low latency and high accuracy computation of sinh (x) and cosh (x) (QH-CORDIC). Since the exponential function (e^x) mostly consists of sinh (x) and cosh(x). The study investigated Range Of Convergence (ROC) and the validity of the QH-CORDIC while computing the exponential function e^x with all-floating-point-domain inputs. The suggested circuit architecture has 32 clock cycles, 512 flip-flops, and 29172 LUTs, together with 9430 register slices. It was mapped to an FPGA device while synthesized in the Xilinx Integrated Synthesis Environment (ISE) Design Suite.

2. Theoretical Background

This section introduces the theoretical and principles of hyperbolic functions. In addition, the techniques used to compute the hyperbolic functions.

2.1 Exponential Function

The exponential function of a generic real-valued variable x is typically expressed as [20]:

$$f(x) = exp(x) = e^x > 0,\ x \in R \tag{1}$$

where, e = 2.71828 …. is the Euler number. Given a complex (therefore, including real) function $f(x)$ that has infinite derivatives at a point x0, its respective Taylor series expansion around x_0 is as follows [20]:

$$f(x) = f(x_0) + \frac{f(\dot{x}_0)}{1!}(x - x_0) + \frac{f(\ddot{x}_0)}{2!}(x - x_0) + \frac{f(\dddot{x}_0)}{3!}(x - x_0) + \cdots \tag{2}$$

where $f^{\cdot}$ is the first derivative, and so on. When $x0 = 0$, the Taylor series is often called the McLaurin series. The exponential function $f(x) = e^x$ can be expressed in terms of its McLaurin series [20]:

$$f(x) = \exp(x) = e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots = \sum_{k=0}^{\infty} \frac{x^k}{k!} \tag{3}$$

With infinite radius of convergence, i.e.,$\forall x \in R$, meaning that the exponential function is entire (i.e., holomorphic in the complex plane). The series above provides a means to numerically calculate the exponential function in terms of the simpler sum and multiplication operations. The exponential function has several useful properties. Some of them are presented as follows:

$$e^0 = 1 \tag{4}$$

$$e^x > 0\,, \forall x \tag{5}$$

$$e\alpha^{(a+b)} = e^{\alpha a}\, e^{\alpha b}, \forall \alpha, a, b \in R \tag{6}$$

$$e^{ab} = (e^b)^a = (e^a)^b, \quad \forall a, b \in R \tag{7}$$

2.2 Hyperbolic Sine (Sinh) and Cosine (Cosh) Function

Until recently, hyperbolic functions were only implemented in software. Because of the superior performance of hardware systems over software implementations, their hardware implementation has become necessary [18]. In neural networks, hyperbolic functions and exponents are used to compute activation functions, hardware realization of single-neuron units, and adaptive filtering [21-24]. Hyperbolic functions $sinhx$ and $coshx$ can be defined in terms of the exponential function (e^{θ}). Figure 1 shows the hyperbolic sine and cosine graphs. The exponential function represents the most common arithmetic operation after the addition, subtraction, multiplication, and division operations and is one of the essential fundamental components of floating-point and fixed-point applications. The exponential function is an essential operation in 3D computer graphics, digital signal processing (DSP), scientific computing, artificial neural networks, logarithmic number systems, and multimedia applications [25-27]. Many of these applications have aggressive performance goals requiring the evaluation of an exponential function in a few microseconds [28,29]. The combination of greatly enhanced FPGA architectures and a vital need for higher performance density in new applications creates a strong demand for improved algorithms [30,31]. Although these procedures produce high accuracy, they are usually too sluggish for real-time applications or numerically intensive [32,33]. In addition, exponential is used to calculate hyperbolic functions. Hyperbolic sine and cosine (sinh and cosh) can be shown utilizing (8) and (9) [31-34]:

$$Sinh\,(x) = \frac{e^{x} - e^{-\mathrm{x}}}{2} \tag{1}$$

$$\cosh(x) = \frac{e^{x} + e^{-x}}{2} \tag{2}$$

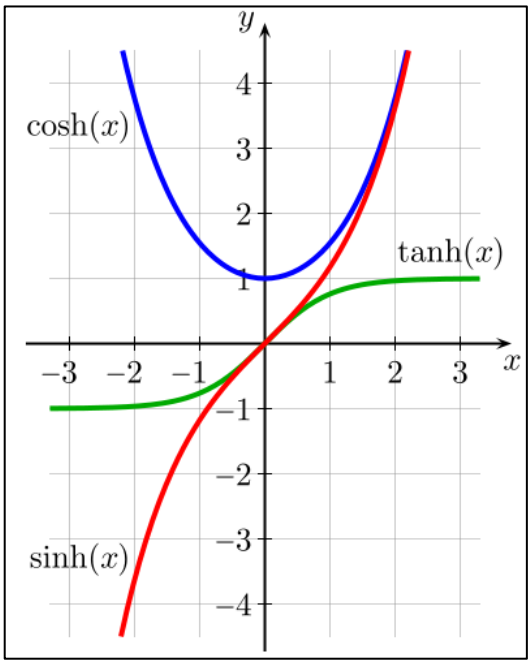

Figure 1: The Hyperbolic Sine, Cosine, And Tangent Functions Graphs [34]

2.3 Approaches to Computing the Trigonometric and Hyperbolic Functions

Many approaches can produce hyperbolic functions such as sine, cosine, tan, sinh, cosh, and tanh functions using Tylor's series, the CORDIC algorithm, polynomial curve fitting, and the LUT approach. Hyperbolic functions are used at every computation point in today's technological environment, from space navigation to the internet micro-architecture of chips. Various methods have been presented for computing hyperbolic functions to be implemented and realized in hardware; some of the most commonly used methods are introduced in the following subsections of this work.

2.3.1 Cordic algorithm

The CORDIC algorithm was first described in 1959 by Jack E. Volder as a sophisticated solution to evaluate the trigonometric function. In 1971, J. Walther extended the CORDIC algorithm to hyperbolic functions, which is now found in many applications [36]. In addition, this algorithm can be implemented using special arithmetic units such as shift registers,

adders, subtractors, and special interconnects instead of multiplication operations which are many costs in digital hardware implementation [37].

In circuit implementation practice, effective hardware design is crucial. By primarily computing addition and shift operations, the CORDIC algorithm completely meets the premise of providing an efficient and affordable hardware implementation operation [38]. This algorithm's enhancement has been a prominent research topic. Numerous developments have been created in algorithm design and architectures, particularly for high-performance and low-cost hardware solutions. Pipelined and parallel CORDIC has been advised for high throughput computations [39]. This algorithm is an iterative method for calculating rotation in 2-dimensional vectors, linear, circular, and hyperbolic coordinate systems [38-40] which is especially suitable for solving the trigonometric relationships involved in a plane coordinate rotation and conversion from rectangular to polar form. Particular predetermined angles rotate the input vector in rotation mode. The input vector is then rotated so that the summation of the rotated angles equals the desired angle to be computed. The angle accumulator stores the starting angle, which gradually approaches zero. Vectoring mode converts rectangular coordinates to polar coordinates. The angle accumulator is set to zero, and the sum of the rotated angles equals the angle to be calculated. Also, for each iteration, the y component approaches zero [17].

The mathematical representation of the hyperbolic CORDIC mode m = -1, the final contents of the X and Y registers are given by

$$X_{(N+1)} = K_h(x_{in}\ cosh(z_{in}) + y_{in}\ sinh(z_{in})) \tag{3}$$

$$y_{(N+1)} = K_h(x_{in}\ sinh(z_{in}) + y_{in}\ cosh(z_{in})) \tag{4}$$

However, the vector is rotated by small angles in the rotation mode. The summation of all angles will arrive at the desired angle. The final coordinates are obtained if (z_i) equals zero. The CORDIC hyperbolic equation is

$$X_{i+1} = k_i(x_i - \mu_i y_i d_i 2^{-1}) \tag{5}$$

$$y_{i+1} = k_i(y_i - x_i d_i 2^{-1}) \tag{6}$$

$$z_{i+1} = z_i - d_i \emptyset_i \tag{7}$$

where $i = tanh^{-1}2^{-i}$, μ = -1 for hyperbolic equations i represent the iterations (i = 1, 2, 3...N). After n iterations, the generalized hyperbolic Equation can be expressed below.

$$X_n = A_n(x\ cosh\ z\ + y\ sinh\ z) \tag{8}$$

$$y_n = A_n(y\ cosh\ z\ + x\ sinh\ z) \tag{9}$$

$$z_n = 0 \tag{10}$$

2.3.2 Taylor series

It is possible to define the Tylor series as an infinite sum of terms at a single point using its derivatives. This method produces trigonometric functions and mathematics. In order to realize the expression of a particular process, the mathematical function must be derived, and at a certain point, this produces a sequence of terms [10]. Madhava of Sangamagrama provided the earliest instances of the use of the Taylor series and closely related techniques in the 14th century, including those of the sine, cosine, tangent, and arctangent trigonometric functions [41]. Compared to standard CORDIC, the Taylor series expansion only permitted rotation in one direction [38]. Tylor series suffers from high order and utilizes enormous resources of the designed system, which are slow and require floating points. The Taylor series expansion equation of the hyperbolic terms is given as follows [42]:

$$\sinh(\theta) = \theta + \frac{\theta^3}{3!} + \frac{\theta^5}{5!} + \cdots + \frac{\theta^{2n+1}}{n!} \tag{11}$$

$$\cosh(\theta) = 1 + \frac{\theta^2}{2!} + \frac{\theta^4}{4!} + \cdots + \frac{\theta^{2n}}{2n} \tag{12}$$

This series cannot be implemented in hardware in its original form as it contains infinite terms. Thus, it needs to be approximated from the hardware implementation perspective, implying a compromise in accuracy [42]. The advantage of this method is that it is one of the oldest and most widely used and has higher accuracy [43], and includes the study of approximate calculations. While this method has the advantage of increased accuracy, higher-order factorial and numerous orders must be calculated. Although this method produces correct results at higher iterations, unnecessary memory is wasted to keep the intermediate results, and it takes longer to produce the desired result due to its iterative nature [44].

Moreover, implementing this method would at least require a multiplier, divider, adder, and subtractor. Finally, for best accuracy, it would be required to take each term in calculation till they become insignificant. Thus, this approach has many hardware requirements and is slow.

2.3.3 Polynomial approximation technique

Curve fitting is a statistical analysis approach that determines the "best fit" line or curve for a group of data points[45]. Curve fitting is the process of determining a mathematical link between a collection of scattered experimental data. As a consequence, it is very easy to use and has a clear analytical formula. It may also specify the overall design of data point acquisition [46]. So far, the used polynomial interpolation to replace the unknown function $y = f(x)$ with a polynomial of degree at most (n), resulting in Equation (20).

$$f(x_i) = p_n(x_i)(i = 0,1,2,\dots,n) \tag{13}$$

The Maclaurin series is used in the polynomial approximation approach to depict a function as an infinite accumulation of derivatives generated from the Taylor series approximation at zero.

Polynomial function fitting is a non-linear and higher-order curve fitting approach that covers the complete range of (n) data points. Two commonly utilized methods are direct polynomial fitting and indirect polynomial fitting. Moreover, the polynomial fitting has been used in several domains as a method for approximating functions. For example, implicit polynomial curves have been widely used in computer graphics, vision, and time series [47].

The use of high-degree algebraic curves and surfaces as geometric models or form descriptors for various model-based computer vision tasks has recently been developed, despite their apparent suitability for object classification and positioning applications. In most circumstances, the resulting algebraic curves or characters are unbounded [48]. The general polynomial equations can be written as follows:

$$\sum y_i = na_0 + a_1 \sum x_i + a_2 \sum x_i^2 + \cdots a_n \sum x_i^n \tag{14}$$

$$\sum x_i y_i = a_0 \sum x_i + a_1 \sum x_i^2 + \cdots + a_n \sum x_i^{n+1} \tag{15}$$

$$\sum x_i^n y_i = a_0 \sum x_i^n + a_1 \sum x_i^{n+1} + \cdots a_n \sum x_i^{2n} \tag{16}$$

Correct implicit polynomial order is necessary of the polynomial curve to roughly fit the data from the real sample. Academics have extensively researched the optimal approach to determine the appropriate order of implicit polynomials. The results show that the zero set can be either bounded or unbounded depending on the order of the implicit polynomial but that the zero set is always unbounded when the order is odd. Hence, even order is often used. This interpolation is utilized to estimate the numerical findings with the least amount of inaccuracy possible. In a data collection with points, interpolation determines functions that allow graphs to pass across each dataset point. The minimum error of the data set is obtained [49]. Basic mathematical operations are often employed in several applications, including electronic calculators, computer simulations, and crucial embedded devices. Always an approximation, their assessment often uses mathematical characteristics, precalculated tabular values, and polynomial approximations. This approximation approach requires more multipliers and adders than is practical in hardware, which results in an inefficient design. Also, because the same factorials are calculated each time, the factorial computation can be decreased Although look-up tables can store those values, the design memory is inefficient [50,51]. This approach is area consumption because it uses a lot of multipliers.

2.3.4 Look-up table approach (LUT)

Tabulated values typically embed rounding errors inherent to the transcendence of elementary functions [52]. While the second range reduction sometimes uses fast but inaccurate hardware approximations, it is often implemented using a look-up table. The interpolation method computes trigonometric and hyperbolic functions with stored values in memory blocks. The amount of memory block entries for this approach must be carefully determined since the computational accuracy and needed hardware area cannot be compromised [53], which is fast but requires memory or limited precision [54,55]. Unfortunately, the hardware efficiency of look-up tables (LUTs) deteriorates with increasing word length.

The look-up table can be big or small, but the smaller the look-up table, the more error is involved. The drawback of a larger look-up table is that it uses more memory, which is costly. Additionally, the look-up table size increases exponentially as the angle accuracy improves. Even though it yields quick results, this strategy is quite expensive to adopt.[44]. The advantage of the

LUT is that it is simple and direct and can be used for all functions as mentioned above calculations; the drawback is that when the data quantity is large, storage space will increase exponentially. Look-up tables allow blocking Random Access Memory (RAM) to store the outputs of trigonometric and hyperbolic functions. A larger amount of block RAM can be used to store more precise outputs. This is a tradeoff decision between area utilization and precision. This method gives accurate answers for certain discreet values but fails unexpectedly if inputted values are not standard. It also involves unnecessary wastage of memory in making look-up tables, and time is wasted in comparisons. Furthermore, the hardware efficiency of Look Up Tables (LUTs) deteriorates with increasing word length.

3. The Proposed Design of Hyperbolic-Sin and Cosine Functions

Before starting to design the hyperbolic functions, it is necessary to study the main core and equation that can construct these functions, and from the equation mentioned in section)3.1), the main mathematical expression to implement the hyperbolic functions is the exponential function. Therefore, the proposed exponential function will be discussed and described in the following subsection.

3.1 The Proposed Design of Exponential Function

The proposed design of the exponential function represents the core of all proposed hyperbolic functions designs such as sinh, cosh, and tanh functions. Figure 2 shows the proposed design of the exponential function implemented using hard FPGA blocks of Xilinx System Generator (XSG). The proposed system depends on the decomposition technique, which can be illustrated mathematically in Equation (24).

$$e^{int.fra} = e^{int} * e^{fra} \tag{17}$$

where, *int* is known as an integer number, and ***fra*** is the fraction number after the decimal point. The main concept of this technique is to take the exponential of the integer and the fractional terms separately and then multiply them together (according to the product rule of exponents ($\boldsymbol{a^{x1} * a^{x2} = a^{x1+x2}}$) instead of taking the exponential of the floating number directly. Theta represents the input number that needs to calculate its exponential, which has a width of 10-bit. Where the width of the integer number is 4-bit (before the binary point), and the fractional number (after the binary point) is 5-bit, one bit is assigned of the sign. According to the above detail of the input angle, the integer number occupied (locations=$2^4 = 16$) locations of one memory, and the number of locations of the fraction number is ($2^5 = 32$ locations) stored in a second memory. Whereas the negative values are stored in a third memory with a size of ($2^7 = 128$) locations. Therefore, the total number of the locations of the three memories are (total memory size=16+32+128=176 locations). In contrast, if the direct exponential is taken, it will take ($2^{10} = 1024$) locations, meaning storing these values will take many memories. The range of integer term is (-15 to +15); the reason for taking this range can be described as follows: firstly, when taking the exponential of the negative part of the range (-4 to -15), it will produce a minimal value close to zero. Secondly, not taking the range greater than (+15) because it will generate a vast number ($reslut * 10^8$).

3.2 The Proposed Hyperbolic-Sine and Cosine Design Based on Exponential System

The central concept of this design is based on the sinh and cosh equations that are depicted as follows:

$$\mathrm{Sinh}(X) = \frac{e^X - e^{-X}}{2} \tag{18}$$

$$\mathrm{Cosh}(X) = \frac{e^X + e^{-X}}{2} \tag{19}$$

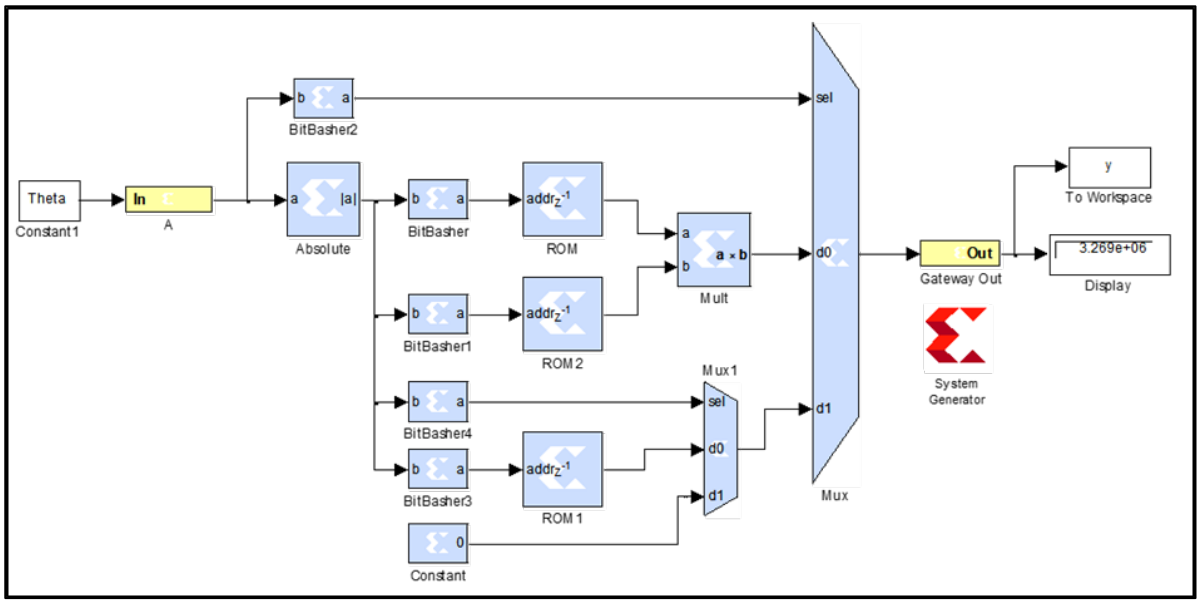

Figure 2: The Proposed Design of Exponential Function

Figure 3 shows the proposed design based on the suggested Exponential design that is explained in detail in section (3.1). The essential addition to the Exponential system is the addition of two multiplexers (Mux1 and Mux2) instead of using one multiplexer that only produces either e^{-X} or e^{X}. The main purpose of the Mux1 and Mux2 blocks are to generate both (e^{-X} and e^{X}) in same time. Consequently, the output of the Mux1 and Mux2 blocks will be driven to the AddSub and AddSub1, respectively, to acquire ($e^{X} - e^{-X}$) and ($e^{X} + e^{-X}$) where they represent numerators of the equations (25) and (26), respectively. Then, the Shift and Shift1 blocks are used to shift the output of the AddSub and AddSub1 one position to the right, in which the shifting operation represents the division of the numerator ($e^{X} - e^{-X}$) and ($e^{X} - e^{-X}$) by 2 to obtain the hyperbolic sine (Sinh (X)) and hyperbolic cosine (Cosh (X)) functions, respectively. The final results of $Sinh(X)$ and $Cosh(X)$ can be acquired by the Display and Display1 blocks, respectively.

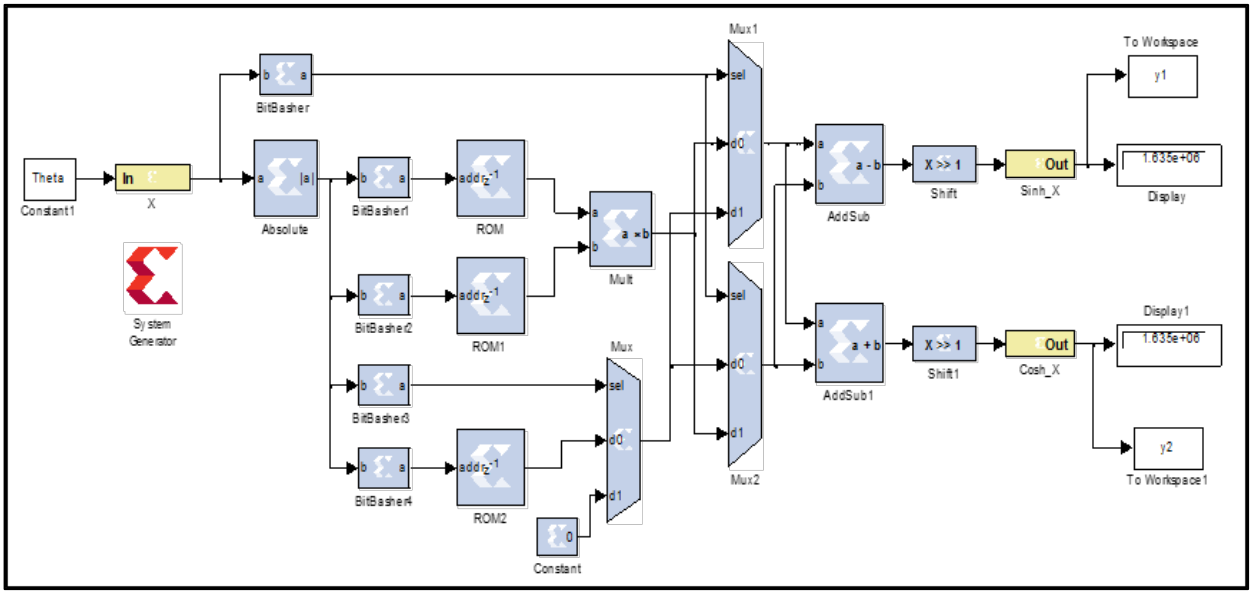

Figure 3: The Proposed Sinh and Cosh Design Based on Exponential System

3.3 The Proposed Hyperbolic-Sine and Hyperbolic-Cosine Design Based on Tylor Series Method

The suggested design employs the hard FPGA blocks created with XSG. The system proposed is based on the Tylor series method. Some modifications have occurred according to the sinh and cosh of the Tylor series equations. These modifications are represented by removing the factorial of the number from the denominator and replaced with a number to the base 2 (2^{n}). The reason for using (2^{n}) is to approximate the number that generated by the factorial function. Therefore, the factorial function can be implemented using the shift operation instead of the multiplications operation. For instance, the third term of the sinh

function of the original Tylor series equation is ($\frac{x^5}{5!}$) where the denominator (5! = 120). Whereas the third term of the modified Tylor series of sinh Equation (27) is ($\frac{x^5}{2^7}$) where the denominator ($2^7 = 128$). Also, of the fourth term ($\frac{x^7}{7!}$) where (7! = 5040), whereas the fourth term of the modified Tylor is ($\frac{2^7}{2^{12}}$) where ($2^{12} = 4096$) and so on. The same procedure is applied to the cosh Equation (28). Many reasons for using the denominator with a number to the base of 2 can be described as follows: firstly, to make the result of the denominator approach to the number produced by the factorial function. Secondly, the ability to implement the design in the Hard FPGA blocks. Finally, reduce the resource utilization devices with relatively high accuracy of the obtained results.

The hyperbolic sine and cosine Equations can be expressed as follows:

$$Sinh(x) = x + \frac{x^3}{6} + \frac{x^5}{2^7} + \frac{x^7}{2^{12}} \tag{20}$$

$$Cosh(x) = 1 + \frac{x^2}{2} + \frac{x^4}{2^4} + \frac{x^8}{2^{15}} \tag{21}$$

Relatively high accuracy results are obtained from these four terms of the sinh and cosh equations (27) and (28), respectively. Figure 4 shows that the input angle (X) is fed directly to the Mult Block and the AddSub block. The multiplication blocks are used to achieve the order value of X. where Mult, Mult1, Mult2, Mult3, Mult4, Mult5, and Mult6 are used to produce $X^2, X^3, X^4, X^5, X^6, x^7 and\ x^8$, respectively. In addition, the shift blocks are used to represent the right shifting operation therefore the division operation is obtained rather than the using of the standard division block. The Shift, Shift1, Shift2, Shift3, and Shift4 blocks are respectively equivalent to ($\frac{x^2}{2}, \frac{x^4}{2^4}, \frac{x^5}{2^7}, \frac{x^7}{2^{12}}, and\ \frac{x^8}{2^{15}}$) which are represent the terms of (27) and (28) Equations. Whereas, the first term of sinh(X) equation is represented by the CMult block, which multiplies the coefficient value (0.1667) with (X^3) instead of using the shifting block to represent ($\frac{X^3}{6}$) As can be noticed from Figure 4, the $Sinh(X)$ equation consist of four summation term that represented by AddSub, AddSub2, and AddSub3 blocks. Where, the output of the AddSub block is represented by the summation of the first and second terms ($x + \frac{x^3}{6}$). Then this output will be added to the third term by using the AddSub1 block to get ($x + \frac{x^3}{6} + \frac{x^5}{2^7}$). The AddSub2 block adds the fourth term to produce the hyperbolic sine equation. In contrast, the $Cosh(X)$ equation has also four terms that donated by AddSub3, AddSub4, and AddSub5. The Constant Xilinx block represents the first term with the value of (1), and then the second term is added to it using the AddSub3 block. Consequently, the output is used to sum the third term by using AddSub4 to get ($1 + \frac{x^2}{2} + \frac{x^4}{2^4}$). To produce the final hyperbolic cosine equation, the AddSub5 block is used. The results of $Sinh(X) and\ Cosh(X)$ is respectively obtained by the Display and Display1 Simulink blocks.

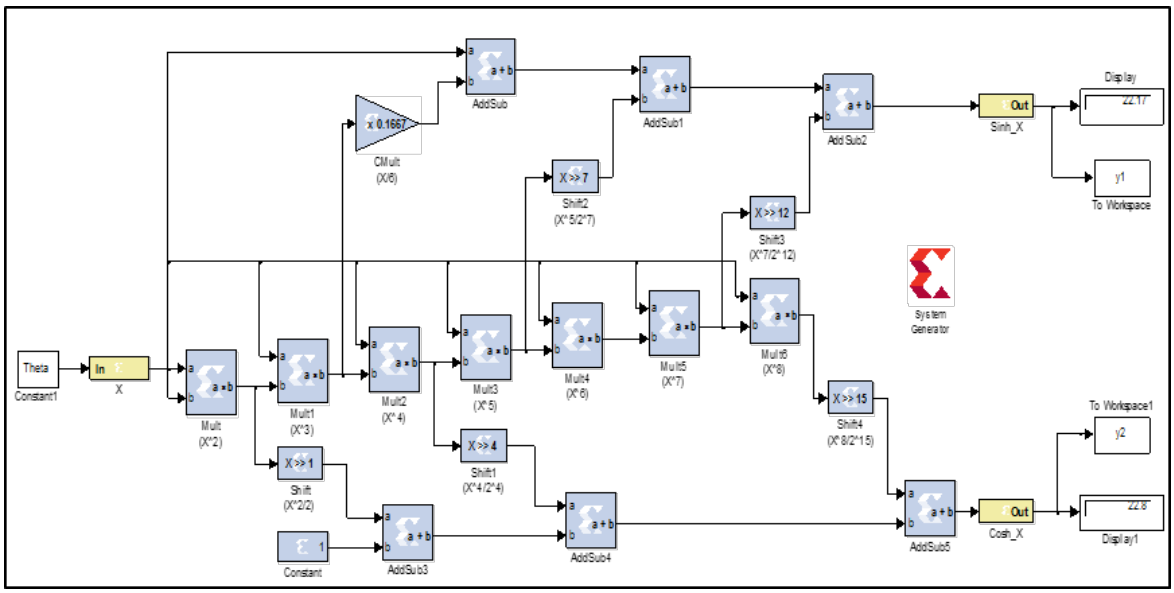

Figure 4: The Proposed Sinh and Cosh Desing Based on Tylor Series Method

3.4 The Proposed Hyperbolic-Sine and Cosine Design Based on Modified Tylor Series

The modified Tylor method design is realized to improve the previous design. The suggested design is realized according to the Tylor series equation with some modifications by adding a zero to each term that the X order is neglected. The primary concept of this design is based on Equation (23), which is covered in detail in section (2.3.2). Equation (23), which uses the distributive property of multiplication, multiplies the first coefficient by (X) before adding it to the second coefficient and so forth. According to the Equation (23), the Sinh(X) and Cosh(X) can be expressed as follows:

$$Sinh(X) = \left(\left(\left(\left(\left(\left(\frac{x}{2^{12}} + 0\right)x\right) + \frac{1}{2^7}\right)x + 0\right)x + \frac{1}{6}\right)x + 0\right)x + x \tag{22}$$

$$Cosh(X) = \left(\left(\left(\left(\left(\left(\frac{X}{2^{15}} + 0\right)X + 0\right)X + 0\right)X + \frac{1}{2^4}\right)X + 0\right)X + \frac{1}{2}\right)X + 0\Big)X + 1 \tag{23}$$

The realized design is constructed of two major parts, as shown in Figure 5. The upper part represents the hyperbolic-sine function, and the other is of the hyperbolic-cosine function. The counter block acts as a pointer to the ROM block, which holds the exact values of the coefficients, which are subsequently multiplied by the (X). When the Mult block receives the input angle(X), the counter begins counting eight counts (0 to 7), which indicate the order value of (X). According to Equation (29), the summation term will be represented by the AddSub block and will consist of seven terms. The Register block will store the current value of the coefficient in each count, and the output of the Register Xilinx block will then be multiplied by (X) through the Mult1 block. The hyperbolic-cosine will be calculated using the exact same procedure.

In contrast, there are two main differences. The first one is that the Counter block counts nine counts (0 to 8), representing the terms number of Equation (30). The coefficient values of Equation (30) will be used to store the ROM1 block, which was the second difference. And the same process will occur.

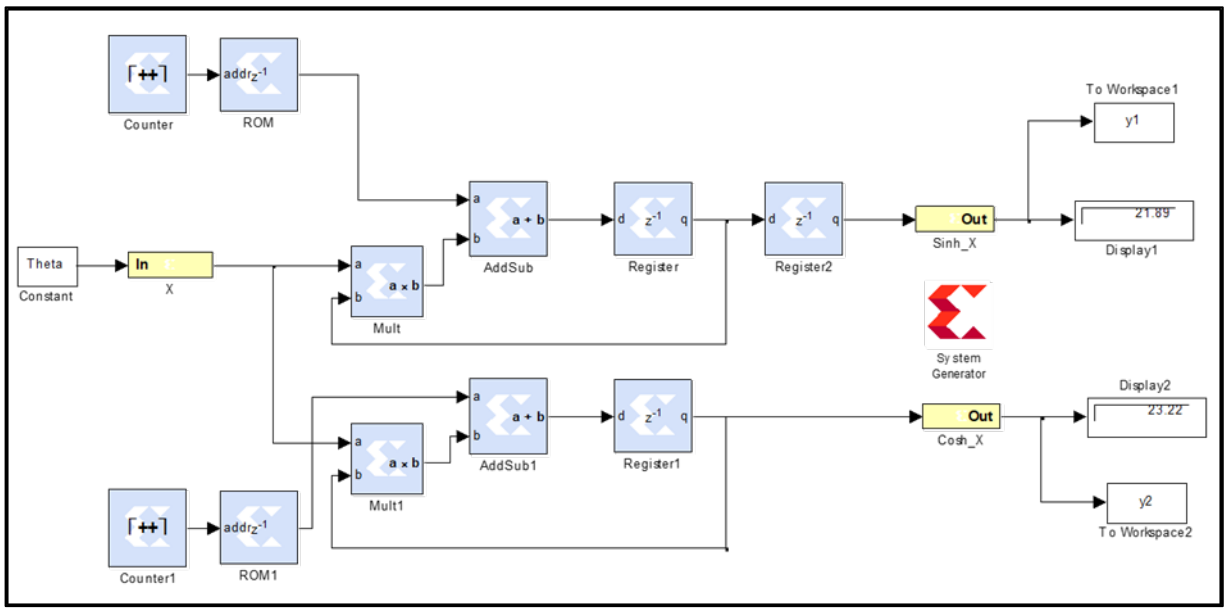

Figure 5: The Proposed Sinh and Cosh Desing Based on Modified Tylor Method

3.5 The Proposed Hyperbolic-Sine and Cosine Design Based on ROM Method

Figure 6 shows the proposed design using the Read Only Memory (ROM) approach using XSG. Since the sinh function is considered an odd function due to ($Sinh(-x) = -Sinh(x)$ and $Sinh(x) = Sinh(x)$), therefore in this design, only the positive values of the sinh signal, which represent the positive half of the sinh wave, are stored in the ROM. Figure 7 shows the generation of the positive half of the sinh wave with the red dashed circles compared to the original full sinh wave with a blue line produced by MATLAB Whereas, the negative values (the negative half of the sinh wave) are the same as the positive values but they are

multiplied by the negative sign (-1) to generate the full sinh wave. Therefore, the size of the ROM area is reduced to half instead of storing all the positive and negative values of the sinh wave in the ROM, which will increase the memory size to double.

The absolute value of the input angle (X) is taken through the Absolute Xilinx block. The Absolute block's output is driven to the Slice block, which is used to take the input width of only the seven Least Significant Bits (7-LSBs), which is the width of the location lines of the ROM and ROM1 blocks. Therefore, the resolution of this approach is $\frac{1}{2^7}$. The results of hyperbolic-sine and hyperbolic-cosine functions are already stored in the ROM and ROM1 blocks, respectively. The selection of the exact result of these two functions depends on the location number specified by the slice block. The block Xilinx Mux is utilized to select between the positive and negative values of the sinh function (ROM block), and this selection depends on the input of the select (sel) line that comes from the Slice1 block, where if the Most Significant Bit (MSB) =1, the output of the multiplexer will be negative, and if the MSB=0, the output will be positive. The sinh's result will be acquired using the Display block. In contrast, the result of the cosh function is directly driven to Display 2.

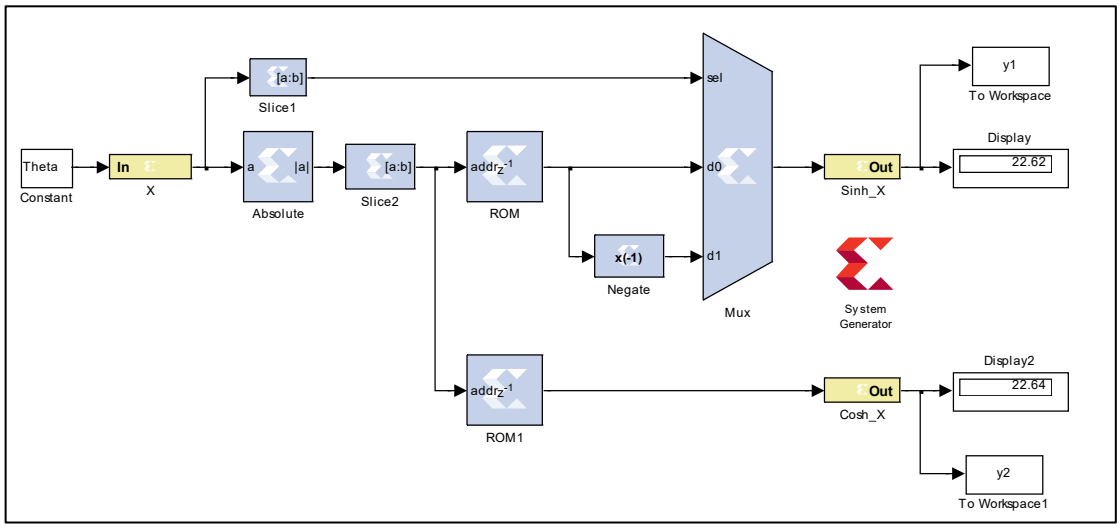

Figure 6: The Proposed Sinh and Cosh Desing Based on ROM Method

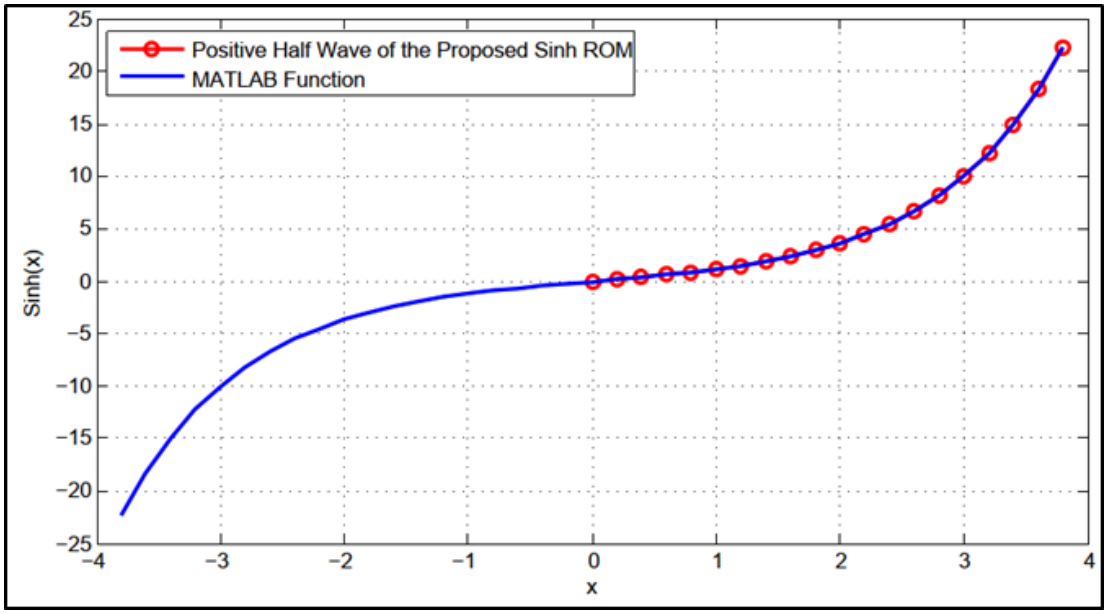

Figure 7: Generating Positive Half Sinh Wave

4. Experimental Results and Discussion

Five suggested hyperbolic function designs were realized using various methods. The Xilinx Spartan 3A-3N/XC3S700a/-4/fg484 FPGA platform is used to verify the proposed designs. These designs are implemented using XSG blocks, which can be found in MATLAB 2012a and ISE 14.7 configuration. This research aims to minimize the use of resources (area). This work includes three comparisons regarding area minimization, power consumption, and high-speed execution.

4.1 Performance Evaluation of the Proposed Hyperbolic Sine and Cosine (Sinh & Cosh) Functions

This section shows the performance of five proposed designs of sinh and cosh functions, as illustrated in Figure 8. Where the red line with circles represents the signals of the proposed designs, the blue line represents the ideal signal of these functions that are considered as a reference signal to compare it with the generated signals of the proposed designs.

Figure 8 (a, b) shows the results signals of sinh and cosh functions, respectively. As can be shown from this Figure, the cycle range of the CORDIC block designs is between $(-\frac{\pi}{4}\ to\ \frac{\pi}{4})$ which cannot cover the whole range of angles (i.e., from $-\pi$ to π). Therefore, there is no matching between the red signal and the typical blue signal at (-1 to 1) of the x-axis. In addition, this design has the lowest accuracy (high error percentage), which is equal to (352.9%) and (221.53%) for sinh and cosh functions, respectively. Another drawback of this design is taking (20) clock cycles for execution, which means it has high latency. The proposed designs that are based on the Tylor series and the modified Tylor series have relatively low accurate results due to the obtained error percentage, as shown in Figure 8:(e-f) and Figure 8:(g-h), which are as follows (sinh-error= 7.05%, cosh-error=27.13%), and (sinh-error = 6.61%, cosh-error= 43.47%), respectively. In addition, the modified Tylor series have a long delay of 10 clock cycles. The ROM approach can be considered a relatively low error percentage of (sinh-error= 5.23%, cosh-error=5.03%), as shown in Figure 8:(i-j). The most accurate results are the designs based on the exponential function, where the error percentages are (0.13%) for each sinh and cosh function as shown in Figure 8:(c-d).. But it has the main drawback: it consumes relatively high logic elements, which means it consumes a lot of area. Therefore, the ROM approach is considered the most effective method in all respects.

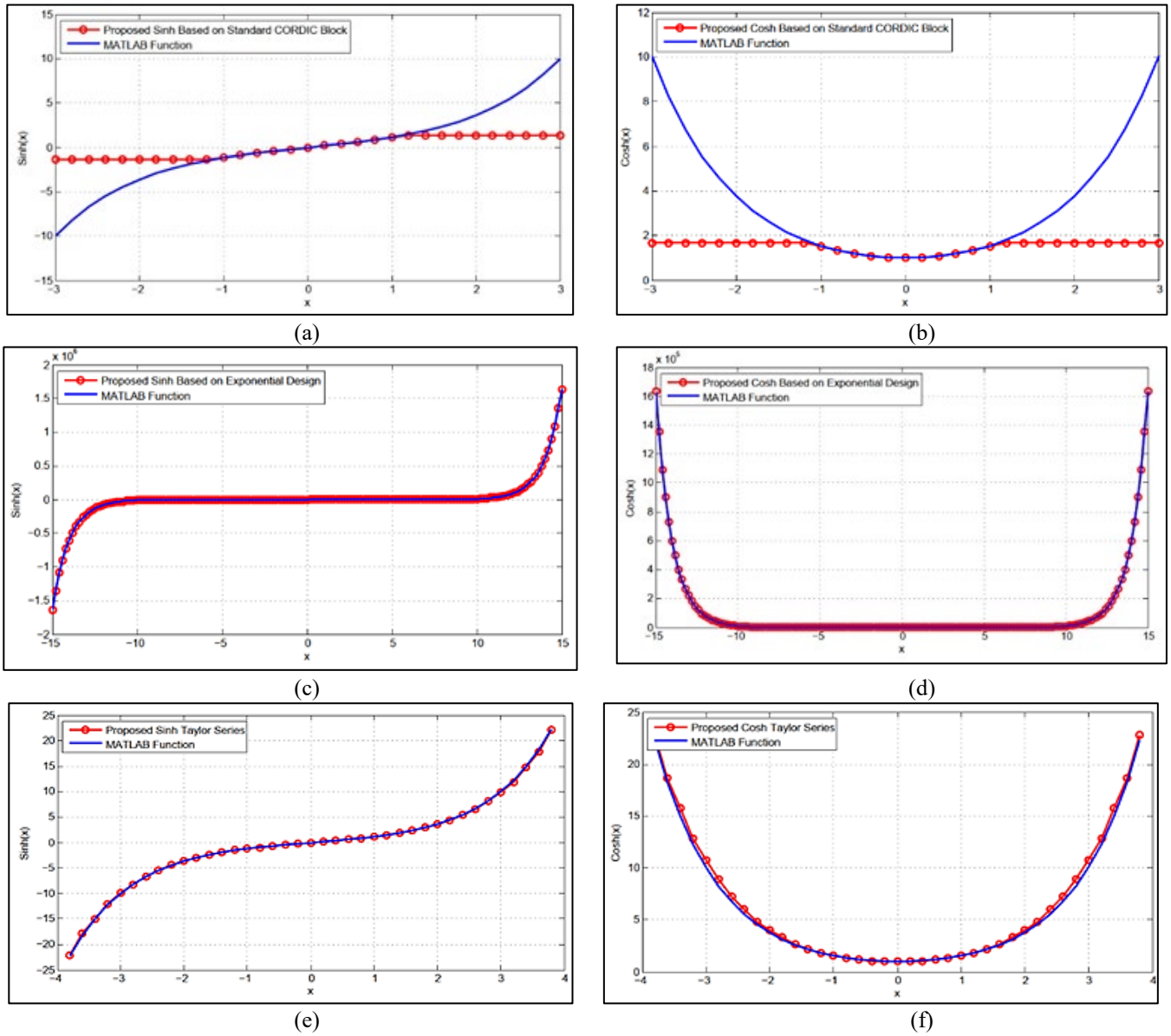

Figure 8: Plots of Sinh and Cosh Functions for (a) Standard CORDIC Hard FPGA Block, (b) Based on Exponential System, (c) Traditional Tylor Series Method, (d) Modified Tylor Series, and (e) ROM Approach

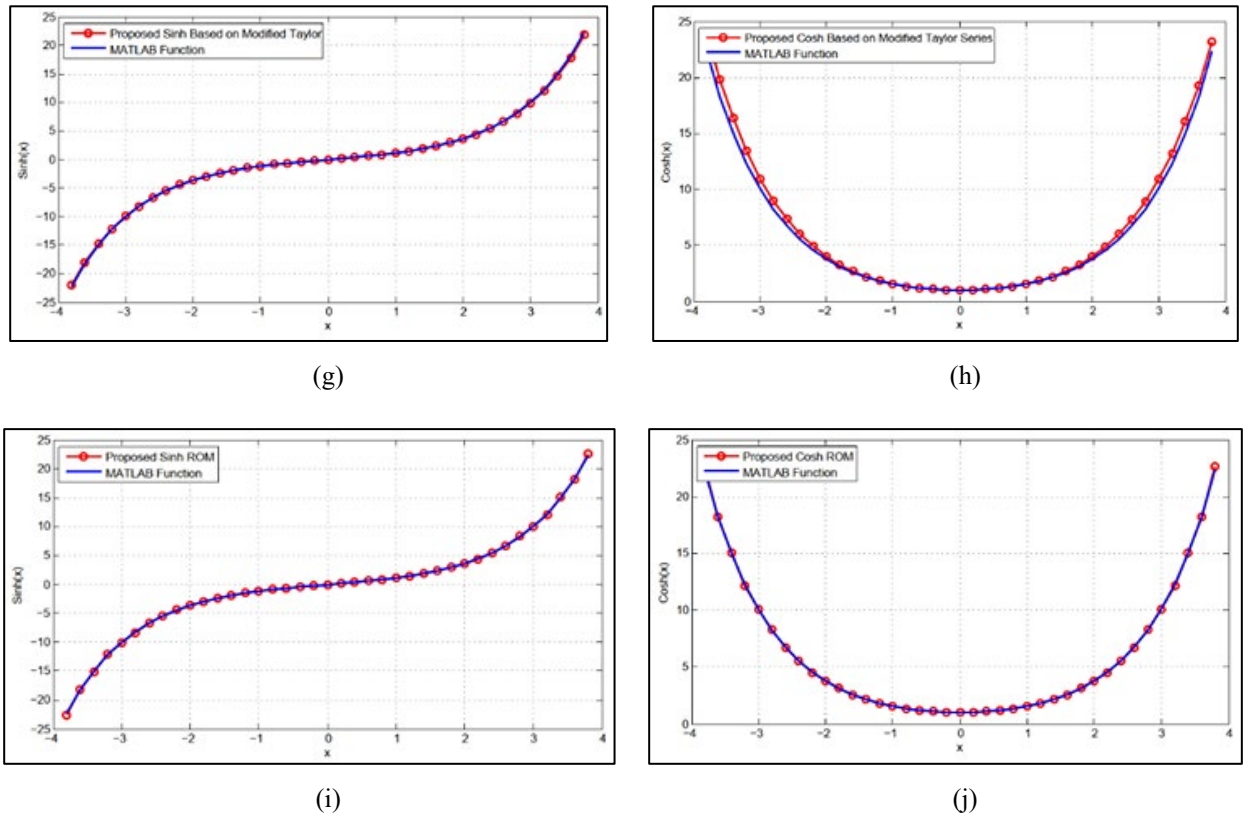

Figure 8: Continued

4.1.1 Resource utilization of hyperbolic sine and cosine (sinh & cosh) designs

Table 1 illustrates the resource utilization, where the designs consume a significant number of elements represented by the LUTs, and occupied slices are the designs based on standard CORDIC block, exponential function, and Tylor series and modified Tylor series. Also, it should be noted that the conventional Tylor series and the standard CORDIC block have flip-flops of (1145) and (70), respectively. The other three designs, in contrast, contain no flip-flops.

As depicted in Table 1 the ROM approach occupied the smallest area among the other designs. This minimizes area because the design already precomputes the angles and stores the results in the memory. This means the design determines a specific number of logic elements that are used. The resolution of the ROM design is ($\frac{1}{2^7}$), which can cover a wide range of angles.

Table 1: Resource Utilization, Max. Operating Frequency and Clock Cycle of Hyperbolic-Sine and Hyperbolic-Cosine Designs

Proposed Sinh/Cosinh Design	No. of F/F	No. of Slices	No. of LUT	No. of RAMB16BWEs	No. of MULT18X18SIO	Max. Operating Frequency (MHz)	No. of Clock Cycle
Standard CORDIC Block	1,145	632	1,067	1	0	0	20
Sinh & Cosh Based on Exponential Function	0	92	145	1	3	2	1
Tylor Series Approach	0	140	193	0	0	10	1
Modified Tylor Series Approach	70	62	77	1	0	0	10
ROM Approach	0	26	43	1	2	0	314.86	1

4.1.2 Power consumption of hyperbolic-sine and hyperbolic-cosine designs

Figure 9 shows the power consumption of the five proposed designs. The total power consumption can be introduced as follows: (116.96 mW) of the design based on standard CORDIC block, (76.75 mW) of the design based on exponential function, (61.6 mW) of traditional Tylor series, (60.5 mW) of modified Tylor series, and (56.83 mW) of the ROM approach. The suggested design based on the ROM technique significantly reduced the total power from the other designs. This reduction is because the ROM design has zero flip-flops. Therefore, the switching activities are reduced or eliminated, which they consider the main reason for dynamic power dissipation. In addition, a few numbers of the LUTs and occupied slices are consumed by this method.

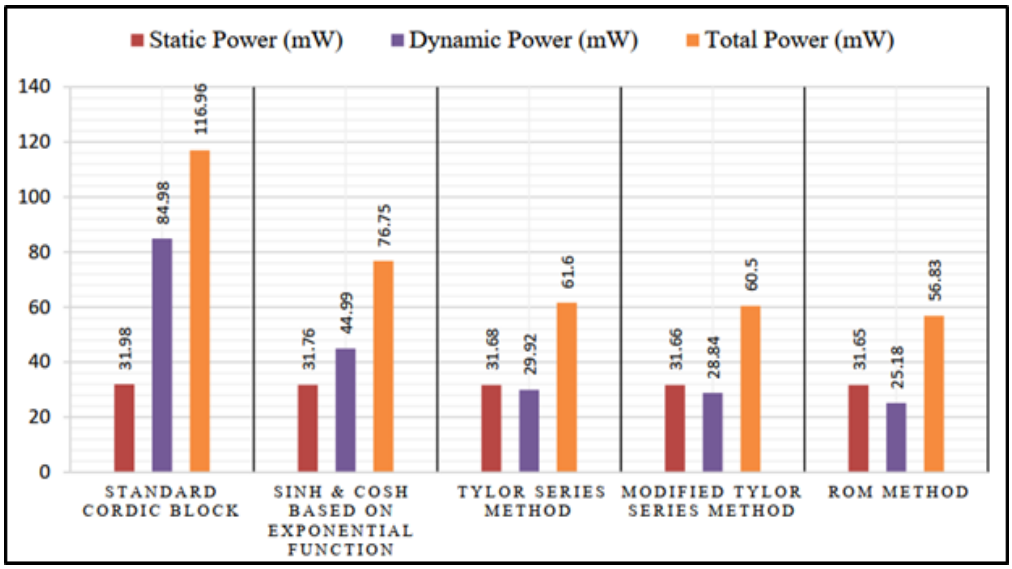

Figure 9: Power Dissipation of the Sinh and Cosh Designs

4.1.3 Area-comparison of the proposed hyperbolic-sin and hyperbolic-cosine designs with the previous works

The best results of area minimization are obtained when using the ROM approach. Various techniques for area minimization are shown in Table 2 which contrasts the suggested designs based on the ROM approach with those of earlier works. The highest number of flip-flops, LUTs, and occupied slices can be found in [17,18,19], which are as follows: (1791, 40, 512) of flip-flops, (not mentioned, 330, 29172) of LUTs, and (3014, 175, 9430) of occupied slices, respectively. In contrast, the number of flip-flops, LUTs, and occupied slices of the ROM approach are zero flip-flops, (175) LUTs, and (92) occupied slices. Therefore, the design based on the ROM approach has the highest area minimization compared to the related works.

Table 2: Comparison of Area between the Proposed Design Based on the Rom Approach and The Previous Works

Technique	No. of F/Fs	No. of LUTs	Occupied Slices	Platform	No. of Clock Cycle	Ref.
CORDIC Algorithm	1791	-	3014	Xilinx FPGA		[17]
CORDIC Algorithms	40	330	175	Altera Quartus –II kit	12	[18]
QH-CORDIC Algorithm	512	29172	9430		32	[19]
ROM Approach	0	175	92	Xilinx Spartan-3A	1	ROM Approach

5. Conclusion

In this work, five proposed designs of hyperbolic sine and hyperbolic cosine functions were implemented by XSG using different approaches. Table 1 shows the comparison between the five proposed designs; it can be concluded that the proposed design based on the ROM approach has a small area utilization, a high speed of execution, which is about one clock cycle, and low power consumption among the other four proposed designs. The other conclusion is that the ROM approach achieved the lowest number of look-up tables, occupied slices, and zero flip-flops. This means it consumes the lowest area and low latency in which it can execute the result also in one clock cycle among the related works, as shown in Table 2.

Author contributions

Conceptualization, R. Yousif, and I. Hashim; methodology, R. Yousif, and I. Hashim; software, R. Yousif, and I. Hashim; validation, R. Yousif, I. Hashim, B. Abd; formal analysis, R. Yousif, I. Hashim, B. Abd; investigation, I. Hashim, and B. Abd; resources, R. Yousif, I. Hashim, and B. Abd; data curation, R. Yousif, I. Hashim, and B. Abd; writing—original draft preparation, I. Hashim, and B. Abd; writing—review and editing, I. Hashim, and B. Abd; visualization, I. Hashim; supervision, I. Hashim ,and B. Abd; project administration, I. Hashim, and B. Abd. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability statement

The data that support the findings of this study are available on request from the corresponding author.

Conflicts of interest

The authors declare that there is no conflict of interest.

References

[1] J. E. Volder, The CORDIC trigonometric computing technique, IRE Trans. Electron. Comput., (1959) 330–334. http://dx.doi.org/10.1109/TEC.1959.5222693

[2] A. Singhal, A. Goen, T. Mohapatra, FPGA implementation and power efficient CORDIC based ADPLL for signal processing and application, Int. Conf. Commun. Syst. Netw., (2017) 325–329. http://dx.doi.org/10.1109/CSNT.2017.8418560

[3] X. Wang, Design and implementation of CORDIC algorithm based on FPGA, Int. Conf. Robots Intell. Syst., (2018) 70–71. http://dx.doi.org/10.1109/ICRIS.2018.00026

[4] P. A. Kumar, FPGA Implementation of the Trigonometric Functions Using the CORDIC Algorithm, Int. Conf. Adv. Comput. Commun. Syst., (2019) 894–900. http://dx.doi.org/10.1109/ICACCS.2019.8728315

[5] T. Meenpal, Efficient MUX based CORDIC on FPGA for signal processing application, IEEE Int. Conf. Intell. Comput. Commun., (2019) 1–6.

[6] C. Dick, CORDIC architectures for FPGA computing, in Reconfigurable Computing, Elsevier, (2008) 513–537.

[7] M. Heidarpur, A. Ahmadi, M. Ahmadi, M. R. Azghadi, CORDIC-SNN: On-FPGA STDP learning with izhikevich neurons, IEEE Trans. Circuits Syst. I Regul. Pap., 66 (2019) 2651–2661. https://doi.org/10.1109/TCSI.2019.2899356

[8] F. Salehi, E. Farshidi, H. Kaabi, Novel design for a low-latency CORDIC algorithm for sine-cosine computation and its Implementation on FPGA, Microprocess. Microsyst., 77 (2020) 103197. https://doi.org/10.1016/j.micpro.2020.103197

[9] R. K. Jain , C. Engineering, Design and FPGA Implementation of CORDIC-based 8-point 1D DCT Processor, 107 (2011) 48.

[10] A. Lashko , O. Zakaznov, VHDL implementation of CORDIC algorithm for wireless LAN. Institutionen för systemteknik, 2004.

[11] K. Paulsson, M. Hübner, J. Becker, Dynamic power optimization by exploiting self-reconfiguration in Xilinx Spartan 3-based systems, Microprocess. Microsyst., 33 (2009) 46–52. https://doi.org/10.1016/j.micpro.2008.08.006

[12] A. Muttaqin, Z. Abidin, R. A. Setyawan, I. A. Zahra, Development of advanced automated test equipment for digital system by using FPGA, Indones. J. Electr. Eng. Comput. Sci., 15 (2019) 661–670. http://doi.org/10.11591/ijeecs.v15.i2.pp661-670

[13] I. Kuon , J. Rose, Measuring the gap between FPGAs and ASICs, IEEE Trans. Comput. Des. Integr. circuits Syst., 26 (2007) 203–215. http://doi.org/10.1109/TCAD.2006.884574

[14] F. Nasser , I. A. Hashim, Power Optimization of Binary Multiplier Based on FPGA, Eng. Technol. J., 39 (2021) 1492–1505. http://doi.org/10.30684/etj.v39i10.2156

[15] F. T. Naser, S. N. Hadi, I. A. Hashim, Power Optimization of KNN Algorithm Based on FPGA, Int. Iraqi Conf. Eng.Technol. Their Appl., (2021) 168–174.

[16] F. T. Nasser, I.A. Hashim, Power optimization of binary division based on FPGA, Indo. J. Electr. Eng. Comp.Sci., 24 (2023). http://doi.org/10.11591/ijeecs.v24.i3.pp1354-1366

[17] J. Sudha, M. C. Hanumantharaju, V. Venkateswarulu, H. Jayalaxmi, A novel method for computing exponential function using CORDIC algorithm, Procedia Eng., 30 (2012) 519–528. http://doi.org/10.1016/j.proeng.2012.01.893

[18] A. Saha, K. G. Kumar, A. Ghosh, M. K. Naskar, Area efficient architecture of Hyperbolic functions for high frequency applications, Int. Conf. Circuits, Cont. Commun., 3 (2018) 139–142. http://doi.org/10.1109/CCUBE.2017.8394139

[19] W. Fu, J. Xia, X. Lin, M. Liu, M. Wang, Low-latency hardware implementation of high-precision hyperbolic functions sinhx and coshx based on improved CORDIC algorithm, Electron., 10 (2021) 2533. http://doi.org/10.3390/electronics10202533

[20] L. da Fontoura Costa, The Exponential Function: A Mathemagical Hub, 2022.

[21] B. Gisuthan, T. Srikanthan, K. V. Asari, A High speed flat CORDIC based neuron with multi-level activation function for robust pattern recognition, Proc. Fifth IEEE Int. Work. Comp. Archit. Mach. Perc., (2000) 87–94. http://doi.org/10.1109/camp.2000.875962

[22] M. Chakraborty, S. Pervin, T. S. Lamba, A hyperbolic LMS algorithm for CORDIC based realization, IEEE Work. Stat. Signal Process. Proc., (2001) 373–376. http://doi.org/10.1109/ssp.2001.955300

[23] M. Qian , A. Qing, Application of CORDIC Algorithm, 2004 (2006) 504–508.

[24] A. Meyer-Bäse, R. Watzel, U. Meyer-Bäse, and S. Foo, A parallel CORDIC architecture dedicated to compute the Gaussian potential function in neural networks, Eng. Appl. Artif. Intell., 16 (2003) 595–605. https://doi.org/10.1016/j.engappai.2003.09.010

[25] [D. M. Lewis, 114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications, IEEE J. Solid-State Circuits, 30 (1995) 1547–1553. https://doi.org/10.1109/4.482205

[26] J. A. Piñeiro, J. D. Bruguera, and J. M. Muller, Faithful powering computation using table look-up and a fused accumulation tree, Proc. - Symp. Comput. Arith., (2001) 40–47. https://doi.org/10.1109/ARITH.2001.930102

[27] A. Souto Martinez, R. Silva González, and A. Lauri Espíndola, Generalized exponential function and discrete growth models, Phys. A Stat. Mech. its Appl., 388 (2009) 2922–2930. https://doi.org/10.1016/j.physa.2009.03.035

[28] N. Kapre and A. DeHon, Accelerating SPICE model-evaluation using FPGAs, Proc. - IEEE Symp. F. Program. Cust. Comput. Mach. FCCM 2009, (2009) 37–44. https://doi.org/10.1109/FCCM.2009.14

[29] P. Echeverría and M. López-Vallejo, An FPGA implementation of the powering function with single precision floating-point arithmetic, Proc. 8th Conf. Real Numbers Comput. Santiago Compost. Spain, pp. 1–10, 2008.

[30] M. Langhammer and B. Pasca, Single precision logarithm and exponential architectures for hard floating-point enabled FPGAS, IEEE Trans. Comput., 66 (2017) 2031–2043. https://doi.org/10.1109/TC.2017.2703923

[31] C. Daramy-loirat et al., CR-LIBM A library of correctly rounded elementary functions in double-precision, 2009.

[32] F. De Dinechin and B. Pasca, Floating-point exponential functions for DSP-enabled FPGAs, Proc. - 2010 Int. Conf. Field-Programmable Technol. FPT'10, (2010) 110–117. https://doi.org/10.1109/FPT.2010.5681764

[33] B. Gostiaux, Cours de mathématiques spéciales.

[34] K. Weltner *et al.*, Exponential, Logarithmic and Hyperbolic Functions, *Math. Phys. Eng. Fundam. Interact. Study Guid.*, pp. 71–86, 2014.

[35] A. S. N. Mokhtar, M. B. I. Reaz, K. Chellappan, and M. A. Mohd Ali, Scaling free CORDIC algorithm implementation of sine and cosine function, Lect. Notes Eng. Comput. Sci., 2 (2013) 978–988.

[36] M. D. S. P. Design, Vivado Design Suite Reference Guide, 2012.

[37] J. Volder, The CORDIC computing technique, Proc. West. Jt. Comput. Conf. IRE-AIEE-ACM 1959, 257–261. https://doi.org/10.1145/1457838.1457886

[38] Digital Arithmetic - Ercegovac/Lang 2003, 2004.

[39] A. S. N. Mokhtar, M. I. Ayub, N. Ismail, and N. G. N. Daud, Implementation of Trigonometric Function using CORDIC Algorithms, AIP Conf. Proc.,1930 (2018) 020040. https://doi.org/10.1063/1.5022934

[40] Rajeev, S. G. Neither Newton nor Leibnitz : Sociology of Kerala, 2005.

[41] Claudio Canuto and Anita Tabacco, Mathematical Analysis II, Springer Cham. https://doi.org/10.1007/978-3-319-12772-9

[42] S. Kathewadi, FSCA : Fast sine calculating algorithm, 2009 IEEE Int. Adv. Comput. Conf. IACC 2009, (2009) 165–170. https://doi.org/10.1109/IADCC.2009.4809000

[43] S. Bhuria and P. Muralidhar, FPGA implementation of sine and cosine value generators using cordic algorithm for satellite attitude determination and calculators, ICPCES 2010 - Int. Conf. Power, Control Embed. Syst., pp. 1–5, 2010, https://doi.org/10.1109/ICPCES.2010.5698645

[44] Y. Song, J. Hu, X. Yang, J. Fu, and X. Xie, A method for data stream processing based on curve fitting, ICSPS 2010 - Proc. 2010 2nd Int. Conf. Signal Process. Syst., 2 (2010) 542–546. https://doi.org/10.1109/ICSPS.2010.5555670

[45] R. Banerjee and S. Das Bit, An energy efficient image compression scheme for wireless multimedia sensor network using curve fitting technique, Wirel. Networks, 25 (2019) 167–183. https://doi.org/10.1007/s11276-017-1543-9

[46] D. J. Kriegman and J. Ponce, Parameterized Families of Polynomials for Bounded Algebraic Curve and Surface Fitting, IEEE Trans. Pattern Anal. Mach. Intell., 16 (1994) 287–303. https://doi.org/10.1109/34.276128

[47] S. A. Sukri, Y. S. Hoe, and T. K. A. Khairuddin, First order polarization tensor approximation using multivariate polynomial interpolation method via least square minimization technique, J. Phys. Conf. Ser., 1988 (2021). https://doi.org/10.1088/1742-6596/1988/1/012013

[48] I. Koren and O. Zinaty, Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations, IEEE Trans. Comput., 39 (1990) 1030–1037. https://doi.org/10.1109/12.57042

[49] M. J. Schulte and E. E. Swartzlander, Hardware Designs for Exactly Rounded Elementary Functions, IEEE Trans. Comput., 43 (1994) 964–973. https://doi.org/10.1109/12.295858

[50] H. D. L. Saint-Genies, D. Defour, and G. Revy, Exact look-up tables for the evaluation of trigonometric and hyperbolic functions, IEEE Trans. Comput., 66 (2017) 2058–2071. https://doi.org/10.1109/TC.2017.2703870

[51] Muller, JM. 1997. Some Basic Things About Computer Arithmetic. In: Elementary Functions. Birkhäuser, Boston, MA. https://doi.org/10.1007/978-1-4757-2646-6_2

[52] A. A. Madi and A. Addaim, Optimized Method for Sine and Cosine Hardware Implementation Generator, using CORDIC Algorithm, 13 (2018) 21–29.

[53] R. K. Yousif, I. A. Hashim and B. H. Abd, FPGA Implementation of Polynomial Curve Fitting Approximation for Sine and Cosine Generator, 2022 5th Int. Conf. Eng. Technol. Appl., (2022) 361-366. https://doi.org/10.1109/IICETA54559.2022.9888742