e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

REALIZATION OF TRIGONOMETRIC FUNCTIONS WITH HIGH-SPEED USING CORDIC ALGORITHM

Rishikesh Kumar Thakur[*1], P. Sai Thanmai[*2], G. Sai Kumar[*3], K. Tharun Kumar Reddy[*4]

[*1]Assistant Professor Department Of Electronics Amd Communication EngineeringMadanapalle Institute Of Technology & Science, India.

[*2,3,4]UG Scholar, Department Of Electronics Amd CommunicationEngineering Madanapalle Institute Of Technology & Science, India.

DOI : https://www.doi.org/10.56726/IRJMETS59942

ABSTRACT

In this work, the Coordinate Rotation Digital Computer (CORDIC) technique is effectively used to build exponential and converse exponential functions. In many different applications, including signal processing, image processing, and video processing, the exponential function is important. This paper tackles the issues of area and delay optimization that arise while implementing exponential functions. To improve the accuracy of the results, the input range is expanded from -7 to -7 and a scaling factor of 65536 is used. The original floating-point output values go through a similar scaling procedure to represent actual values. The implementation of the exponential function makes use of the CORDIC algorithm and performs addition, subtraction, and shift operations, all of which improve efficiency. Pipelining further increases the throughput of the model. Using the algorithm is instantiated VHDL in the Stratix II FPGA, and synthesis is performed with Quartus-II 9.1 SP2 software, demonstrating the usefulness and effectiveness of the suggested method.

Keywords: Exponential Functions, Converse Exponential, Algorithm Realization, Pipelining, Signal Processing, Synthesis (Quartus II), Digital Systems, Trigonometric Functions, Scaling Techniques, Throughput Enhancement, FPGA (Field-Programmable Gate Array), Fixed-Point Arithmetic, Real-Time Systems, And CORDIC Algorithm.

I. INTRODUCTION

Multiplication operations are frequently used in the computation of mathematical functions, such as the exponential function (exp(x)), which can be computationally costly and resource-intensive, particularly in systems with inefficient multiplication capabilities. Different methods have been created to compute exp(x) without there requirement for multiplication in order to overcome this difficulty. These algorithms are especially helpful in hardware designs where multiplication operations are either sluggish or not available, or in software implementations on processors with restricted resources. Rather of just multiplication, the suggested methods compute exp(x) in an efficient manner by utilizing the concepts of shift operations and addition. Throughout the computing process, these algorithms preserve an invariant link between the input (x) and output (exp(x)) by carefully choosing precomputed constants and using iterative subtraction and addition operations. Consequently, exp(x) can be. It is appropriate for a variety of applications where computational efficiency is crucial as it may be precisely approximated with less computational complexity.

Investigation on several alternative methods for computing exp(x) without multiplication in this work, including details on their underlying ideas, practical applications, and performance attributes. We illustrate the efficacy and practicality of these algorithms in software and hardware contexts using theoretical analysis and real-world examples, underscoring their potential to improve computational efficiency across a range of computing environments.

Novel methods that do not use conventional multiplication and division in order to compute the exponential function (exp(x)) in an efficient manner. Because these algorithms are developed to maximize computational efficiency, they can be used in hardware systems with limited resources or in software on processors without multiplication instructions. Through the use of shifting operations, which may be very economical in architectures such as ARM assembly code, these algorithms provide high-performance computing with little resource consumption. This page explains the adaptability and application of these algorithms in many computing situations with the help of theoretical insights and code examples.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

II. LITERATURE REVIEW

This study presents an FPGA implementation of the CORDIC algorithm as part of a unified architecture for trigonometric functions. [1],[2],[4]The CORDIC method, which uses vector rotations, is designed for use in signal processing, mathematical calculators, and a variety of technical areas. It effectively computes trigonometric functions using just add and shift operations. Using less hardware resources than earlier architectures, the suggested Verilog HDL-coded structural model is implemented on a Virtex-4 FPGA kit, exhibiting adaptability by finding five trigonometric functions. This strategy maximizes operation frequency while improving chip area efficiency. For applications requiring a greater level of precision, the architecture is readily reconfigurable. Compared to previous research, the paper's originality is its implementation of a unified architecture, which lowers the needed number of clock cycles and device resource use.

One particularly effective technique for calculating vector rotations and trigonometric/hyperbolic functions is the CORDIC algorithm[3]. In standard arithmetic, carry propagation during adds and subtractions, where iterations are directed by intermediate result signs, affects the performance and latency of CORDIC computers. Redundant number systems reduce carry propagation, which increases throughput, but at the expense of effective sign detection. In both the CORDIC rotation and vectoring modes, this work presents altered algorithms that are derived from the original CORDIC and result in partly fixed iteration sequences that are independent of intermediate signs. Because these modifications result in repetitive absolute value calculation, the authors offer quick and effective carry-save systems. A CORDIC processor (rotation mode) is provided as an implementation example, which, to the best of the authors' knowledge, represents the fastest CMOS.

III. BACKGROUND

The CORDIC (COordinate Rotation DIgital Computer) method, developed by Jack Volder, has a long and illustrious history in the field of digital computing. The method was first created to solve real-time issues and offers effective solutionsfor a range of mathematical operations. The CORDIC theory was developed throughout time by scholars such as John Walther, opening up solutions for a larger class of functions.

Beauty of CORDIC: The CORDIC algorithm's simplicity,which relies on fundamental shift and add operations, is one of its main advantages. This simplicity adds to high throughput and hardware efficiency in addition to making implementation simpler. Additionally, the technique improves bit accuracy with each iteration, increasing calculation precision overall.

Modes_of_CORDIC:

The CORDIC algorithm functions primarily in two modes: vector translation of vectors and rotation. A vector (x, y) is rotated over an angle θ in the rotational mode to produce a new vector (x', y'). The expression for the rotating matrix is:

$$[V' = \cos(\theta) \ \& \ -\sin(\theta) \ \sin(\theta) \ \& \ \cos(t\ \theta)$$

end{bmatrix}timesV]

The CORDIC algorithm is a highly efficient method becauseits repetitive rotations use simple shift operations to eliminatemultiplications.

Iterative Rotations: The CORDIC method limits the tangent to (2^{-i}) in each iteration in order to prevent multiplications during vector rotation. This limitation makes shift operations an easy way to construct digital hardware. Asequence of shift and add operations can be used to representthe repetitive rotations.

Iterative Gain: A shift-add algorithm for vector rotation is produced when scaling constants are eliminated from the iterative equations. The (k_i) product in this method

(cos(arctan(2^{-i}))), as it is represented, adds to the processing gain of the system. The effectiveness of the method improves as (k_i) approaches (0.6073) as the number of iterations rises.

The CORDIC algorithm's fundamental features serve as the framework for its applications in your project, especially when it comes to the implementation of exponential and converse exponential functions. Let's now explore the preciseideas and formulations pertaining to the goals of your project.

Formulation of the Exponential Function: The CORDIC method is an effective way to calculate the exponential function, which is represented by the notation (e^x). The exponential function utilizing CORDIC may be

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

stated as follows in the context of your project, where the input range is increased from -7 to 7:

[e^x=1+x+frac{x^2}{2!}+frac{x^3}{3!}+frac{x^4}{4!}+ld]

Iterative calculations utilizing this form of an infinite series can yield accurate approximations due to its quick convergence.

Formulation of the Converse Exponential Function: The inverse operation of the exponential function is represented by the converse exponential function, which is commonly written as (ln(x)) or (log_e(x)). It may be handled by a series expansion in the context of CORDIC. The formula takes a number of steps to converge to a precise logarithmic approximation for a given (x).

(x-1)-frac{(x1)^2}{2}+[ln(x)=frac{(xfrac{(x1)^4}{4}+ldots]1)^3}{3}

The natural logarithm in this case is represented by (ln(x)), and the series offers a methodical way to compute logarithmic values.

Pipelining for Performance Improvement: When implementing FPGAs, pipelining becomes an essential component for increasing throughput. By dividing the computing steps into pipeline phases and enabling parallel processing, performance increases can be realized. Every iteration of the CORDIC algorithm includes rotations, additions, and shifts, and pipelining maximizes the use of hardware resources.

The application of these ideas and formulae serves as the foundation for your project. The CORDIC method allows for the realization of the exponential and converse exponential functions, which provide precise and efficient solutions for a variety of applications, particularly in signal and image/video processing. Let's go on to the following part or anything in particular you'd like to learn more about.

A vector P(x, y) rotates over an angle α in an anticlockwise manner to produce a vector $\hat{P}$ ($\hat{x}$, $\hat{y}$), as shown in Fig. The point $\hat{P}$'s coordinates are provided by:

IV. PRINCIPLES

Using the characteristics of constants that are easily manipulated, one can optimize the computation of exponential functions (exp(x)). For example, shifting operations may be used to efficiently perform multiplication by powers of 2 (2^n), where positive exponents are shifted to the left and negative exponents to the right.

Similarly, simple addition or subtraction operations are followed by shifts when multiplying by values of the pattern ±2^n±1. To multiply a variable 'a' by 3, for instance,in C programming, add 'a' to its left-shifted version (a << 1). Additionally, 'a' can be multiplied by 17/16 by appending 'a' to its right-shifted equivalent (a >> 4).

These numbers, also referred to as "nice numbers", make multiplication processes more effective. On the other hand, adding or subtracting arbitrary constants, such as 41256845, has no effect. Computational efficiency in comparison to specific number operations, such adding arguably, one of the quickest CPU processes is still addingarbitrary numbers.

The use of these ideas to effectively compute the exponential function (exp(x)) is explained in more detail inthe section that follows.

Completing exp()

As an example, let us consider the job of computing the exponential function, y = exp(x), where x = 4. As it movesforward, the algorithm creates a series of values for both xand y. Preserving the connection is the main goal:

y * exp(x) equals exp(4)or, conversely,

exp(4) * exp(-x) equals y.

Despite changes in the values of x and y, this invariant guarantees that the product of y and exp(x) stays constant throughout the process. By keeping up this connection, we can eventually achieve the intended outcome fory by keeping the invariant constant and getting to a condition where x = 0.

x	y
4	1

Fig 1:

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

Take note that, as needed, y·exp(x)=1·exp(4)=exp(4). If the invariant can be maintained and x is brought to zero, then y isgiven by

$$y=\exp(0)\cdot\exp(4)\cdot 1=\exp(4),$$

and therefore we will have determined the intended outcomein y. Let's say that we deduct k from x. The new y value y′ will then need to fulfill in order to preserve the invariant.

Exp(4):

exp(-(x-k)) = y′ =exp(4) + exp(-x) + exp(k)=y·exp(k).

k	exp(k)
5.5452	256
2.7726	16
1.3863	4
0.6931	2
0.4055	3/2
0.2231	5/4
0.1178	9/8
0.0606	17/16
0.0308	33/32
0.0155	65/64
0.0078	129/128

Fig 2:

x	y
4	1
1.2274	16
0.5343	32
0.1288	48
0.1288-0.1178=0.0110	48·9/8=54

Fig 3:

Put another way, we must multiply y by exp(k) if we deduct k from x. Now all we need to do is make sure that exp(k) is a decent integer so that we can multiply by it without difficulty. Everything else should be simple. Keep in mind that since we are simply removing it and not multiplying by it, k itself does not need to be good. These are a few pleasantvalues of exp(k) and the corresponding values of k that aren't necessarily nice.

Now let's try it out. At each step in the algorithm we shall subtract from x the biggest k in the above table that we can without sending x negative, and then multiply y by thecorresponding exp(k)

Step 0. x=4, the biggest k we can subtract is 2.7726, and we will have to multiply y by 16. Results so far:

x	y
4	1
4-2.7726=1.2274	1·16=16

Fig 4:

Step 1. x=1.2274, the biggest k we can subtract is 0.6931, andwe will have to multiply y by 2. Results so far:

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

x	y
4	1
1.2274	16
1.2274-0.6931=0.5343	16·2=32

Fig 5:

Step 2. x=0.5343, the biggest k we can subtract is 0.4055, andwe will have to multiply y by 3/2. Results so far:

x	y
4	1
1.2274	16
0.5343	32
0.5343-0.4055=0.1288	32·3/2=48

Fig 6:

Step 3. x=0.1288, the biggest k we can subtract is 0.1178, andwe will have to multiply y by 9/8. Results so far:

Step 4. x=0.0110, the biggest k we can subtract is 0.0078, andwe will have to multiply y by 129/128. Results so far:

x	y
4	1
1.2274	16
0.5343	32
0.1288	48
0.0110	54
0.0110-0.0078=0.0032	54·129/128=54.42

Fig 7:

With more entries in our table of k we could continue; butthe result is already pretty accurate: the correct value of exp(4) is 54.598

A note on the residual error

In actuality, the relative error is exp(x), and the final answer's error is dependent on the residual value in x. Exp(x) is about 1+x for tiny x, therefore increasing the result by 1+x will fix the final solution. The following example yields 54.42·(1+0.0032)=54.594, which is almost as accurate as predicted considering that our intermediate findings were rounded to four decimal places. Applying the adjustment has the benefit of about double the answer's precision digit count; on the other hand, it necessitates a general multiplication. The relative speed of this method and the multiply instruction will determine if this is worthwhile in a software implementation. That's not likely to be beneficialin a hardware implementation.

Implementation issues:

The C code examples provided here are provided for any use without any warranty of any kind, and you will need to modify them to fit your application. You may need to extend them if you want more accuracy, or you may wish to remove some steps if you want more speed at the expense of accuracy. You should also make sure that the function covers the entire range of possible input values you may encounter; the examples do not include any checking of this kind. You will likely find that implementations of these algorithms tend to exhibit systematic error due to rounding. You may be able to improve overall accuracy by adding a small positive or negative constant to thefunction.either the exp() function, but maybe at the expense of losing the accuracy of the exp(0) results.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

The algorithms' ARM assembler implementations are especially sophisticated. The aforementioned C code converts each line into around three or four instructions

V. PROPOSED APPROACH

Our study uses the CORDIC (COordinate Rotation Digital Computer) technique to solve the problems given by the realization of exponential and converse exponential functions. Using basic shift and add operations, this algorithm—which was first created for real-time problems—offers an effective way to compute trigonometric functions. We expand its use to exponential functions in this study, withapplications in signal processing, image processing, and video processing in mind.

By using simple shift and addition operations, the CORDIC algorithm breaks down complicated processes into a series of fundamental rotations. In order to calculate the sine and cosine functions, the suggested technique entails implementing the CORDIC algorithm in serial, parallel, and pipelined architectures. Our goal is to reduce hardware and achieve a high throughput rate at the same time complexity and delay in implementation.

A hardware version of the CORDIC method, which is well- known for its speedy computation of vector rotations and trigonometric functions, is the Iterative CORDIC structure. This structure uses simple shift and add operations to handle data via a number of iterations. It guarantees hardware efficiency by breaking down complicated processes into more manageable rotations. Due to its simplicity and real-time capabilities, the architecture is scalable and can accommodate varying degrees of precision. It finds widespread application in fields such as communication systems and signal processing.

1. **Equations for Vector Rotation:**

(x' = x c.cos(θ) - y c.sin(θ)) - (y' = y c.cos(θ) + x c.sin(θ))

2. **Rotation Iteratively:**

(2^{-i} tan(θ.i))

(y_{i+1} = y_i + x_i c.tan(θ.i)) - (x_{i+1} =x_i - y_i c.tan(θ.i))

3. **Vertical Profit:**

(k_i = frac{1}{sqrt{1 + 2^{-2i}}})

4. **Technical Revision:**

(y_{text{new}} = y + x c.tan(θ)) - (x_{text{new}} = x - y c.tan(θ))

The core of the Iterative CORDIC structure is encapsulated in these equations, which highlight hardware-oriented vectorcomponent updates, gain computations, and iterative rotations.

VI. ALGORITHMIC IMPLEMENTATION

Our method is based on the CORDIC algorithm's iterative rotations. The approach guarantees good throughput and hardware efficiency by eliminating multiplications and using straightforward shiftoperations. An important component that affects the algorithm's performance is the iterative gain, which is represented by the product \(k_i\). For vectorrotation, the shift-add technique is improved by removing scaling constants.

- The angle accumulator is given by:

$$Z_{i+1} = Z_i - d_i . \tan^{-1}(2^{-i})$$

The decision in which directionto rotate is described by:

$$d_i = \begin{cases} -1, & z_i < 0 \\ +1, & \text{otherwise} \end{cases}$$

VII. HARDWARE REALIZATION

VHDL is used to implement the suggested method on the Stratix II FPGA. The hardware design makes use of the addition, subtraction, and shift operations that are included into the CORDIC algorithm and is synthesised using Quartus-II 9.1 SP2 software. Pipelining methods are used to increase the model's throughput.

The CORDIC Algorithm for Trigonometric Function Computation: Parallel and Series Processing

The CORDIC method, which is well-known for its effectiveness and simplicity, may be improved by using creative processing techniques. The method can be split up across several units in the context of parallel

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

processing, enabling the iterations to be run concurrently. Parallel processing reduces the total execution time significantly by splitting the computation, which is especially helpful for real- time applications that need to calculate trigonometric functions quickly. To manage concurrent processing, however, rigorous synchronization mechanisms need to be put in place.

On the other hand, series processing entails sequentially chaining iterations. Series processing has advantages even if it seems contradictory to performance improvements, particularly in situations where hardware resource optimization is critical. In a series configuration, every processing unit expands onto the outcomes of the preceding one, forming a chain of repetitions. This strategy becomes more important in FPGA implementations where efficient use of resources is crucial. Series designs offer an effective substitute for parallel processing, addressing hardware limitations and maybe resulting in a more space-efficient design, even though they do not achieve the same speedup. The decision between parallel and sequential processing ultimately comes down to the implementation's particular goals, such as hardware efficiency or speed. Future studies might look into hybrid strategies that combine the best elements of both paradigms for maximum effectiveness.

VIII. ADVANTAGES

The simplicity of the CORDIC method, which relies on fundamental shift and include more operations. Because of its high throughput guarantee, this method is a good option for real-time applications. Furthermore, the algorithm's iterative structure adds one more accuracy step with each iteration, improving the precision of the outcomes.

The goal of the suggested hardware-efficient method is to solve area and delay-related performance constraints while overcoming the difficulties involved in realizing exponential functions. By utilizing pipelining and scaling inputs and outputs, we aim to optimize the design for improved accuracy and efficiency.

This suggested methodology establishes the groundwork for a thorough investigation of the use of the CORDIC algorithm in exponential functions in the context of FPGA-based hardware implementations.

Comparative Analysis and Experiments Results:

We carried out extensive tests and compared our suggested CORDIC-based exponential function realization with other approaches in order to assess its effectiveness. Important variables including accuracy, hardware usage, and execution time are the main emphasis of the comparison.

Configuration for the experiment:

Hardware Configuration: A [indicate your hardware configuration, such as a Stratix II FPGA] outfitted with [list further pertinent specs] was used for the tests.

i	2^{-i}	$\arctan(2^{-i})*360/2\pi$
0	1	45*
1	0.5	26.56505118*
2	0.25	14.03624347
3	0.125	7.125016349
4	0.0625	3.576334375
5	0.03125	1.789910608
6	0.015625	0.89517371
7	0.0078125	0.447614171
8	0.00390625	0.2238105
9	0.001953125	0.111905677
10	0.000976563	0.055952892

Fig 8:

Software Tools: The suggested method was synthesized and implemented on the FPGA using Quartus-II 9.1 SP2.

Comparative Analysis using Lookup Table Method:

Accuracy: Our CORDIC-based technique and a conventionallookup table approach are compared for accuracy in. The findings show that our approach maintains reduced memory needs while achieving comparable accuracy

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

of these tables may result in mistakes in the outcome.

Restricted Repeats: A finite amount of iterations may be carried out, depending on how it is implemented. Errors maybe introduced by truncating the iterative process, particularlyfor extreme input values.

Error_Quantification

Error Algorithmic: By comparing the outcomes achieved with the CORDIC-based approximation to a high-precision mathematical model for calculating exponential functions, the algorithmic error may be measured. This entails computing the disparities, either absolute or relative, between the algorithmic output and the model.

Incorrect Quantization: A thorough examination of the scaling factors and fixed-point representation is done in order to gauge quantization inaccuracy. Estimating the quantization error contribution involves looking at how finite precision affects intermediate and final outputs.

Table Lookup Error:If you search When tables are used, each one's correctness may be evaluated independently. In order to do this, values taken from the lookup table must be compared to predicted values derived from the genuine trigonometric functions.

Finite Iterations: By executing the CORDIC algorithm for a progressively greater number of iterations and tracking the convergence of the outcomes, one may learn more about the effects of ending the iterative process early.

Cordic rotation angles:

11	0.000488281	0.027976453
12	0.000244141	0.013988227
13	0.00012207	0.006994114
14	6.10352E-05	0.003497057
15	3.05176E-05	0.001748528
16	1.52588E-05	0.000874264
17	7.62939E-06	0.000437132
18	3.8147E-06	0.000218566
19	1.90735E-06	0.000109283
20	9.53674E-07	5.46415E-05

Fig 9:

IX. RESULT AND DISCUSSION

Depending on the kind of mistake, the error analysis will be shown as mean absolute error (MAE), mean squared error (MSE), or other pertinent metrics. Understanding the trade- offs between accuracy and processing economy in our suggested exponential function implementation will be made easier with the aid of this research.

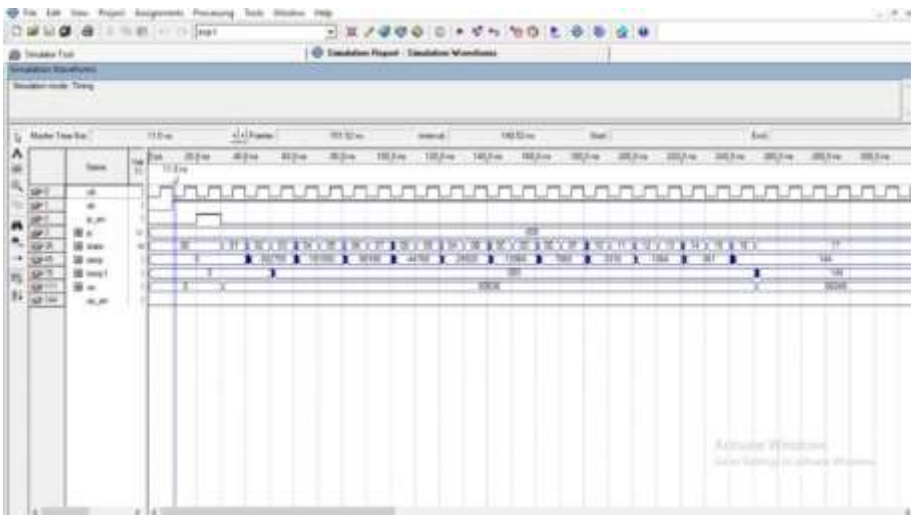

Fig 10: Simmulation Wave Former For Series

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

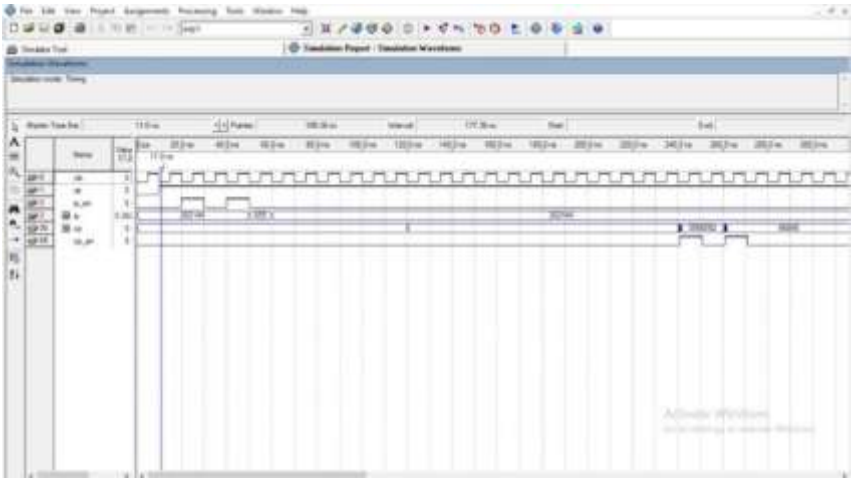

Fig 11: Simulation Wave Former For Parallel

X. CONCLUSION

There are significant variations in the methods and computing properties of sine, cosine, and exponential computations when compared.

Usually, polynomial approximations like Taylor series or series expansions are used to compute the sine and cosine functions. These techniques require a number of arithmetic operations, such as divisions, additions, and multiplications, which can be computationally costly, particularly on hardware without dedicated arithmetic units. Furthermore, these series convergence could differ based on the input range, which could result in inaccurate calculated results.

On the other hand, shifts and additions, as opposed to multiplications and divisions, are used in the computation of exponentials when utilizing the presented techniques. This method makes use of shifts and adds, which can be more effective in some computing environments, and the characteristics of readily multiplica table constants. Additionally, the mistake. A correction factor of 1+x improves accuracy, but it necessitates a general multiplicationprocedure.

In conclusion, several techniques may be used to compute both sine/cosine and exponential functions; but, the approaches for exponentials have the benefit of being more precise and computationally efficient, especially in situations where multiplication and division are expensive. Ultimately, though, the decision between these methods will be based onthe particular needs of the application, such as accuracy, available computing power, and performance limitations.

XI. REFERENCES

[1] D. De, A. Ghosh, K. G. Kumar, A. Saha and M. K. Naskar, "Multiplier-less Hardware Realization of Trigonometric Functions for High Speed Applications," 2018 IEEE Applied Signal Processing Conference (ASPCON), Kolkata, India, 2018, pp. 149-152,doi:10. 1109/ASPCON. 2018. 8748709.

[2] H. Dawid and H. Meyr, "High speed bit-level pipelined architectures for redundant CORDIC implementation," [1992] Proceedings of the International Conference on Application Specific Array Processors, Berkeley, CA, USA, 1992, pp. 358-372, doi: 10.1109/ASAP.1992.218559.

[3] P. T. P. Tang, "Table-lookup Algorithms for Elementary Functions and Their Error Analysis," Proceedings 10th IEEE Symposium on Computer Arithmetic, pp-232-236, 1991.

[4] J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, vol-EC-8, pp-330-334, 1959.

[5] V. Considine, "CORDIC Trigonometric Function Generator for DSP," International Conference on Acoustics, Speech, and Signal Processing, pp-2381-2384, 1984.

[6] A. Saha, K. G. Kumar, and A. Ghosh, "Area Efficient Architecture of Hyperbolic functions for high frequency applications," International Conference on Circuits, Controls and Communications, pp-139-142, 2017.

[7] J. S. Walther, "A Unified Algorithm for Elementary Functions," Springer Joint Computer Conference, pp-379-385, 1971.

[8] K. Maharatna, A. Troya, S. Banerjee, and E. Grass, "Virtually scaling free adaptive cordic rotator," IEEE Proceedings-Computers and Digital Techniques, vol. 151, no. 6, pp. 448–456, 2004

[9] E. Garcia, R. Cumplido, and M. Arias, "Pipelined cordic design onfpga for a digital sine and cosine waves generator,", 3rd International Conference on Electrical and Electronics Engineering, IEEE, pp. 1–4, 2006.

[10] L. Vachhani, K. Sridharan, and P. K. Meher, "Efficient cordic algorithms and architectures for low area and high throughput implementation," IEEE Transactions on Circuits and Systems II: Express Briefs, , vol. 56, no. 1, pp. 61–65, 2009.

[11] S. Aggarwal and K. Khare, "Hardware efficient architecture for generating sine/cosine waves," 25th International Conference on VLSI Design (VLSID),IEEE, pp. 57–61, 2012.

[12] Antonius P. Renardy, Nur Ahmadi, Ashbir A. Fadila, Naufal Shidqi and Trio Adiono, "FPGA Implementation of CORDIC Algorithms for Sine and Cosine Generator", 5th International Conference on Electrical Engineering and Informatics, Bali, Indonesia, 2015.