CORDIC-Based Computation of Arcsine and Arccosine Functions on FPGA

Pedro Paz, and Mario Garrido, *Senior Member, IEEE*

***Abstract*—This paper presents a novel CORDIC-based approach for computing arcsine and arccosine functions. Previous approaches based on CORDIC either calculate double iterations, which increases the complexity of stages, or have a high approximation error. By contrast, the proposed approach presents a novel compensation of the gain in the rotations that allows for an accurate computation of the arcsine and arccosine that does not increase the complexity of the stages. The proposed approach has been implemented on an FPGA to demonstrate its benefits.**

***Index Terms*—CORDIC, arcsine, arccosine, sine, cosine, FPGA, trigonometric**

I. Introduction

The CORDIC algorithm [1]–[4] is based on the computation of a rotation as a series of microrotations by specific angles. These angles have the advantage that they allow to compute the microrotations by only using additions and bit-wise shifts. This is very convenient for implementations in digital systems requiring low area.

The extended version of the CORDIC algorithm [5]–[8] can be used to compute trigonometric functions, hyperbolic functions, multiplications, divisions, logarithms, and exponentials, among others. This work focuses on the CORDIC-based algorithms for computing inverse trigonometric functions. Specifically, it considers arcsine and arccosine [9]–[13].

The direct approach for arcsine and arccosine computation [6] presents large approximation errors at several points of the domain of the functions. This occurs because the gain of the microrotations needs to be compensated at each stage over the reference used to determine the direction of the microrotations. This compensation requires a real multiplication, which largely increases the resources needed to implement the algorithm. Alternatively, the double iteration CORDIC algorithm [9]–[11], [14], [15] is based on computing two microrotations by each CORDIC angle in the same direction. This allows to compensate the gain of the microrotations using bit-wise shifts and additions. This leads to computing the arcsine and arccosine functions without accuracy issues and without additional multiplications. However, this approach increases the complexity of the microrotations at each stage.

P. Paz and M. Garrido are with the Department of Electronic Engineering, ETSI de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain, e-mails: p.pazm@alumnos.upm.es, mario.garrido@upm.es

This work was supported by the Project PID2021-126991NA-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", and by the Grant RYC2018-025384-I funded by MCIN/AEI/10.13039/501100011033 and by "ESF Investing in your future".

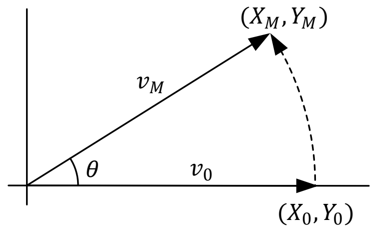

Fig. 1. Rotation of the vector v by the angle θ carried out in the CORDIC algorithm.

This brief presents a novel CORDIC-based approach for arcsine and arcosine computation that solves the issues with the direct CORDIC-based approach without increasing the complexity of the stages. The proposed approach is based on the approximation of the gain of the microrotations by an expression that can be computed only with bit-wise shifts and additions. This approach leads to accurate computations of the arcsine and arccosine functions using fewer resources than previous approaches. Additionally, an FPGA implementation is provided and compared to state-of-the-art CORDIC-based implementation for arcsine and arcosine computations.

The brief is organized as follows. In Section II, we review the CORDIC-based algorithms for computing the sine and cosine trigonometric functions and their extension to arcsine and arccosine. In Section III, the proposed algorithm is described in detail. In Section IV, implementation consideration and comparison to state-of-the-art results are provided. In Section V, the error analysis of the proposed approach is presented. Finally, the main conclusions of this work are summarized in Section VI.

II. Background

A. Sine and cosine computation using CORDIC

Fig. 1 shows the calculation of the sine and cosine of an angle θ using the CORDIC algorithm [16], [17]. The calculation is based on rotating a vector v by an angle θ. The initial value of the vector is $v_0 = (X_0, Y_0)$ with $Y_0 = 0$ and zero phase. The final value is $v_M = (X_M, Y_M)$. Once the rotation has been carried out, the sine and the cosine of the angle are obtained as $\sin(\theta) = Y_M/|v_M|$ and $\cos(\theta) = X_M/|v_M|$. When the magnitude of v_M is one, the sine and the cosine are directly Y_M and X_M, respectively.

The CORDIC algorithm for computing sine and cosine uses the variables X_i, Y_i and Z_i, which represent the real and imaginary components of the vector and its phase at stage i, respectively. The algorithm calculates the rotation by a series of M microrotatons by the set of angles

$$\alpha_i = \tan^{-1} 2^{-i}, \tag{1}$$

for $i = 0, \ldots, M-1$. Each microrotation brings the angle Z_i closer to θ. Thus, when Z_i is greater than θ, the rotation is computed in clockwise direction. Otherwise, the rotation is computed in counterclockwise direction. In the algorithm, the direction of the rotation at stage i is denoted by δ_i where

$$\delta_i = \begin{cases} 1 & \text{if } Z_i < \theta, \\ -1 & \text{otherwise,} \end{cases} \tag{2}$$

so that

$$\theta \approx \sum_{i=0}^{M-1} \delta_i \cdot \alpha_i. \tag{3}$$

The microrotations are computed as

$$\begin{aligned} X_{i+1} &= X_i - \delta_i Y_i 2^{-i}, \\ Y_{i+1} &= Y_i + \delta_i X_i 2^{-i}, \end{aligned} \tag{4}$$

which only requires bit-wise shifts and additions.

The phase Z_i also needs to be updated at each stage by either adding or subtracting the angle α_i, i.e.,

$$Z_{i+1} = Z_i + \delta_i \alpha_i. \tag{5}$$

The angles α_i are usually precomputed and stored in memory.

Each microrotation increases the magnitude of the vector by a factor

$$A_{(i+1),i} = \frac{\sqrt{X_{i+1}^2 + Y_{i+1}^2}}{\sqrt{X_i^2 + Y_i^2}} = \sqrt{1 + 2^{-2i}}. \tag{6}$$

As a result, the gain for the whole algorithm is computed as

$$A_M = \prod_{i=0}^{M-1} \sqrt{1 + 2^{-2i}}. \tag{7}$$

In order to compute correctly the sine and cosine, and avoid additional multiplications or divisions, the component X is initialized to $1/A_M$. Thus, the variables X, Y and Z are initialized as

$$\begin{aligned} X_0 &= 1/A_M, \\ Y_0 &= 0, \\ Z_0 &= 0. \end{aligned} \tag{8}$$

Consequently, after M iterations, the magnitude of the vector v will be one, its phase will be approximately θ, and its components X_M and Y_M will be $\cos(\theta)$ and $\sin(\theta)$, respectively.

B. Direct arcsine and arccosine computation using CORDIC

The calculation of the arcsine and arccosine functions using CORDIC [6] follows the same principles explained in Section II-A. In this case, either $\sin(\theta)$ or $\cos(\theta)$ is known, and the angle θ is sought. In this algorithm, the direction of the rotations is calculated by comparing X_i with $\cos(\theta)$ in case of the arccosine computation, or Y_i with $\sin(\theta)$ in case of the arcsine computation. Thus, the phase Z_M of the vector obtained after computing the algorithm will be the angle θ that is sought. The equations for computing the arcsine are

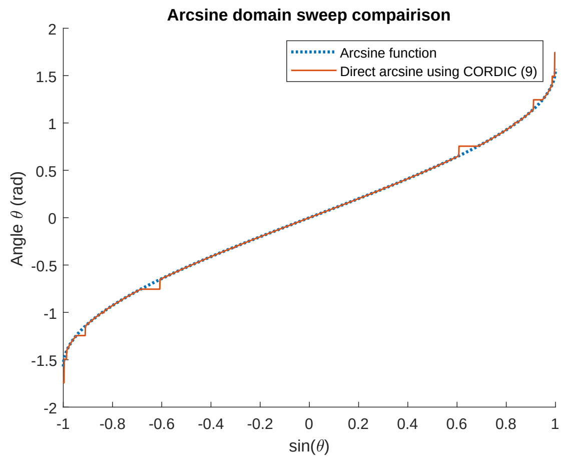

Fig. 2. Comparison between the built-in arcsine function and the direct CORDIC approximation.

$$\begin{aligned} \delta_i &= \begin{cases} 1 & \text{if } Y_i < T_0, \\ -1 & \text{otherwise,} \end{cases} \\ X_{i+1} &= X_i - \delta_i Y_i 2^{-i}, \\ Y_{i+1} &= Y_i + \delta_i X_i 2^{-i}, \\ Z_{i+1} &= Z_i + \delta_i \tan^{-1}\left(2^{-i}\right), \end{aligned} \tag{9}$$

where $T_0 = \sin(\theta)$, $X_0 = 1$, $Y_0 = 0$ and $Z_0 = 0$. Likewise, the equations for computing the arccosine are

$$\begin{aligned} \delta_i &= \begin{cases} 1 & \text{if } X_i < T_0, \\ -1 & \text{otherwise,} \end{cases} \\ X_{i+1} &= X_i - \delta_i Y_i 2^{-i}, \\ Y_{i+1} &= Y_i + \delta_i X_i 2^{-i}, \\ Z_{i+1} &= Z_i + \delta_i \tan^{-1}\left(2^{-i}\right), \end{aligned} \tag{10}$$

where $T_0 = \cos(\theta)$, $X_0 = 1$, $Y_0 = 0$ and $Z_0 = 0$.

Fig. 2 compares the direct arcsine computation using CORDIC in (9) with the exact arcsine function over the whole domain of the function. It can be observed that, at several points along the domain of the function, the CORDIC algorithm fails to obtain a good approximation. This is due to the fact that the gain of the rotations over vector v is not taken into account. At each stage, the magnitude of this vector is slightly different. However, the direction of the rotations is determined by comparing either X_i or Y_i with the value T_0, which corresponds to a vector with magnitude one. This leads to selecting the wrong direction for the rotations in some of the iterations and, thus, results in large errors in the approximation.

A way to compensate the gain of the rotations over T_0 is to compute

$$T_{i+1} = T_i \cdot \sqrt{1 + 2^{-2i}}. \tag{11}$$

However, this requires a real multiplication at each iteration.

TABLE I
COMPARISON OF THE RESOURCES REQUIRED TO IMPLEMENT A STAGE FOR AN ARCSINE COMPUTER BASED ON CORDIC.

	Adders/ subtracters	Registers	Error
Direct CORDIC	3	3	High
Double Iteration	6	4	Low
Proposed	4	4	Low

C. Double iteration CORDIC

The main solution in the literature for solving the approximation error is to apply the double iteration CORDIC algorithm [9], [10]. This algorithm calculates two rotations by each angle α_i. The equations of the algorithm for the arcsine calculation are

$$\begin{aligned} &\delta_i = \begin{cases} 1 & \text{if } Y_i < T_i, \\ -1 & \text{otherwise,} \end{cases} \\ &X_{i+1} = x_i(1+\delta_i 2^{-2i}) - y_i\delta_i 2^{-i+1}, \\ &Y_{i+1} = y_i(1+\delta_i 2^{-2i}) + x_i\delta_i 2^{-i+1}, \\ &Z_{i+1} = Z_i + 2\cdot\delta_i\cdot\tan^{-1}(2^{-i}), \\ &T_{i+1} = T_i + T_i\cdot 2^{-2i}, \end{aligned} \tag{12}$$

where , $X_0 = 1$, $Y_0 = 0$, $Z_0 = 0$ and $T_0 = \sin(\theta)$. Likewise, the arccosine equations are

$$\begin{aligned} &\delta_i = \begin{cases} 1 & \text{if } X_i < T_i, \\ -1 & \text{otherwise,} \end{cases} \\ &X_{i+1} = x_i(1+\delta_i 2^{-2i}) - y_i\delta_i 2^{-i+1}, \\ &Y_{i+1} = y_i(1+\delta_i 2^{-2i}) + x_i\delta_i 2^{-i+1}, \\ &Z_{i+1} = Z_i + 2\cdot\delta_i\cdot\tan^{-1}(2^{-i}), \\ &T_{i+1} = T_i + T_i\cdot 2^{-2i}, \end{aligned} \tag{13}$$

where , $X_0 = 1$, $Y_0 = 0$, $Z_0 = 0$ and $T_0 = \cos(\theta)$.

Note that each iteration of this algorithm can also be implemented by computing two iterations of a direct CORDIC implementation for arcsine and arcosine, including the gain compensation over T_i. This is why it is called double iteration CORDIC. This algorithm does not lead to significant errors in the approximation and it can be computed without requiring real multiplications, since the gain over T_i is computed by a bit-wise shift and an addition. This solves the issues with the direct CORDIC algorithm. However, the calculation of two microrotations per angle increases the hardware complexity.

III. PROPOSED APPROACH

A. Proposed algorithm

The main issue with the CORDIC-based arcsine and arccosine algorithms is the compensation of the gain over T_i at each iteration. Avoiding this computation leads to large approximation errors at several points across the domain of the functions, as reviewed in Section II-B. Using a real multiplier to compensate this gain is often too expensive in terms of area for hardware implementations. The double iteration CORDIC, reviewed in Section II-C solves this issue without requiring real multipliers. However, the complexity of the stages is increased. This leads to hardware implementations that require more resources.

In this work, we propose a method to compute T_i using only additions and bit-wise shifts. This approach does not require to modify the X, Y and Z paths of the original CORDIC. This leads to a simple, low-area circuit capable of computing accurately the arcsine and arcosine functions.

The proposed approach considers the following approximation:

$$\sqrt{1+2^{-2i}} \approx 1 + 2^{-2i-1}. \tag{14}$$

By squaring both terms,

$$1 + 2^{-2i} \approx 1 + 2^{-2i} + 2^{-4i-2}, \tag{15}$$

it can be observed that for $i \geq 0$ the term 2^{-4i-2} is considerably smaller than the other two terms. Thus, it can be neglected without leading to large errors in the CORDIC computation. By applying this idea, the value of T in the proposed approach is calculated as $T_{i+1} = T_i + T_i \cdot 2^{-2i-1}$.

As a result, the equations of the algorithm for the arcsine calculation are

$$\begin{aligned} &\delta_i = \begin{cases} 1 & \text{if } Y_i < T_i, \\ -1 & \text{otherwise.} \end{cases} \\ &X_{i+1} = X_i - \delta_i\cdot Y_i\cdot 2^{-i}, \\ &Y_{i+1} = Y_i + \delta_i\cdot X_i\cdot 2^{-i}, \\ &Z_{i+1} = Z_i + \delta_i\cdot\tan^{-1}(2^{-i}), \\ &T_{i+1} = T_i + T_i\cdot 2^{-2i-1}, \end{aligned} \tag{16}$$

The equations for the arccosine calculation are

$$\begin{aligned} &\delta_i = \begin{cases} 1 & \text{if } X_i < T_i, \\ -1 & \text{otherwise.} \end{cases} \\ &X_{i+1} = X_i - \delta_i\cdot Y_i\cdot 2^{-i}, \\ &Y_{i+1} = Y_i + \delta_i\cdot X_i\cdot 2^{-i}, \\ &Z_{i+1} = Z_i + \delta_i\cdot\tan^{-1}(2^{-i}), \\ &T_{i+1} = T_i + T_i\cdot 2^{-2i-1}. \end{aligned} \tag{17}$$

By following the ideas in [10], the proposed algorithm can be further optimized by avoiding the first iteration. This is due to the fact that the arcsine and arccosine functions are odd and only have outputs in the range $[-\pi/2, \pi/2]$ and $[0, \pi]$, respectively. For both of them, the input is in $[-1, 1]$, but we can only consider the absolute value of the input. This will provide a solution in $[0, \pi/2]$. Then, if the input was in the range $[0, 1]$, the result is correct, whereas if the input was in $[-1, 0)$, the output is moved to the fourth quadrant for arcsine by negating the output, and to the second quadrant for arccosine by subtracting the result from π.

The initialization of the algorithm needs to take into account two factors. First, the difference between the real gain of the microrotations and the proposed approximation is compensated as

$$A'_M = \prod_{i=0}^{M-1} \frac{1+2^{-2i-1}}{\sqrt{1+2^{-2i}}}. \tag{18}$$

Second, after the first iteration, the phase of v will be $\tan^{-1}(1)$. Thus, the components are initialized as

TABLE II
COMPARISON OF PIPELINED ARCHITECTURE FOR ARCSINE COMPUTATION BASED ON CORDIC.

	[11]	[10]	Proposed A	Proposed B	Proposed C
Algorithm	Double iteration	Double iteration	Proposed	Proposed	Proposed
Word length	25	20	25	20	20
Iterations	12	12	12	12	12
FPGA	Cyclone E	Virtex 6	Cyclone E	Virtex 6	Virtex US+
Look-up tables (LUTs)	3200	1265	1553	1012	998
Flip-flops (FFs)	3200	747	791	753	753
Clock frequency (MHz)	152	220	155	266	595
Throughput (MS/s)	152	220	155	266	595
Latency (clock cyles)	12	10	11	11	11
Latency (ns)	78.9	45.5	70.9	41.3	18.5
Power (mW)	NP	NP	216	91	131
Error (rad)	$4.88 \cdot 10^{-4}$	$2.33 \cdot 10^{-5}$	$4.90 \cdot 10^{-4}$	$2.51 \cdot 10^{-5}$	$2.51 \cdot 10^{-5}$

NP: Not provided.

$$\begin{aligned} X_1 &= A'_M \cdot \cos(\tan^{-1}(1)), \\ Y_1 &= A'_M \cdot \sin(\tan^{-1}(1)), \\ Z_1 &= \tan^{-1}(1), \end{aligned} \tag{19}$$

Note that since the first iteration is not computed, then, in (16) and (17), $i = 1, ...M-1$.

As a result, the proposed algorithm requires only bit-wise shifts, additions and subtractions to be computed and obtains an accurate approximation of the arcsine and arccosine for the whole domain of the functions. Therefore, it solves the issues in previous approaches.

B. Proposed architecture

Fig. 3 represents the architecture for a single stage of the proposed algorithm for arcsine computations. A pipelined implementation requires a stage for each microrotation in the algorithm minus the first one. The throughput of the proposed architecture is one sample per clock cycle, and its latency is $M-1$ cycles. In this figure, Z_i contains the phase information of the vector v. The angle α_i is stored in a memory and it is either added or subtracted to Z_i. The obtained result is registered. The X_i and Y_i components are right-shifted i bits. The shifted X_i signal is either added or subtracted to the original Y_i signal, and the shifted Y_i signal is added or subtracted to the original X_i signal. Finally, both results are registered. The T_i signal contains the information of the sine whose arcsine is sought, scaled to match with the magnitude of the v vector at each stage. This signal is right-shifted $(2i-1)$ bits and added to the original T_i signal in order to compensate the gain of the microrotation computed over the vector v. Afterwards, this result is registered. The direction of the rotations is calculated by comparing Y_i and T_i, as detailed in eq. (16).

IV. COMPARISON AND EXPERIMENTAL RESULTS

Table I compares the number of adders, subtracters and registers required to implement a stage for the different CORDIC-based arcsine approaches. The direct CORDIC approach requires the smallest area with 3 adders and 3 subtractors, but it has high approximation error. Conversely, both the double iteration CORDIC and the proposed approach obtain accurate results for the whole domain of the arcsine and arccosine

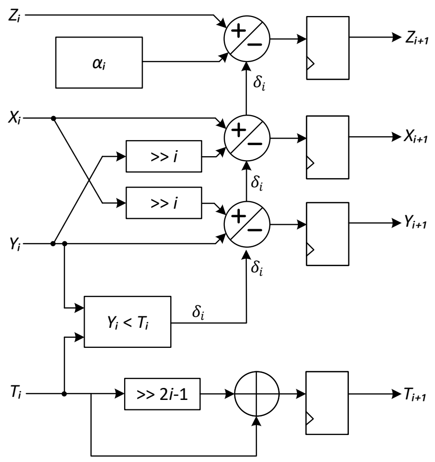

Fig. 3. Architecture for an iteration of the proposed CORDIC-based algorithm for arcsine and arccosine computations.

functions. Among them, the proposed approach saves 33% of the adders/subtracters, as it only requires 4 of them, whereas the double iteration CORDIC requires 6 of them.

The proposed architecture has been described in VHDL. Table II shows experimental results of the proposed approach and previous CORDIC-based pipelined arcsine implementations. The proposed implementations A and B have been provided with the only purpose of comparing to previous works in a fair way, whereas implementation C provides results for a state-of-the-art FPGA. In all the proposed implementations, the error has been calculated as the maximum angular error among 10^6 uniformly distributed inputs in the range $[-1, 1]$.

Implementation A and [11] use the same setup with 25-bit word length, 12 iterations and an Altera Cyclone EP4CE115F29C7 FPGA. It can be observed that both obtain similar throughput and error. However, the proposed implementation A reduces the latency by 10% and requires significantly less area, with savings of 60% in terms of LUTs and reduction of the number of FFs by a factor 4.

Implementation B and [10] have 20-bit word length, 12 iterations, and use a Xilinx Virtex-6 XC6VLX240T FPGA. Compared to [10], the proposed approach B reduces the number of LUTs by 20% and has a similar number of FFs. This agrees with the information provided in Table I. Furthermore, the proposed approach B achieves 20% higher throughput, which results in lower latency in nanoseconds, even when the implementation in [10] takes one less clock cycle to process the inputs. Finally, the error of the proposed approach is 7% higher than that in [10], but it could be reduced by increasing the number of iterations by one at the cost of a slight increase in area.

Finally, the proposed implementation C considers the same design as B, implemented on a Xilinx Virtex Ultrascale+ XCVU37P FPGA. On this FPGA, the design reaches a throughput of 595 MS/s with slightly less LUTs, same number of FFs and much better ratio between throughput and power consumption than implementation B on a Virtex-6.

V. Error analysis

The maximum error for the proposed approach has three sources: The word length used in the implementation (ε_{WL}), the number of iterations that are computed (ε_M), and the approximation error of the gain compensation over T_i, (ε_T). Thus, the maximum error can be expressed as

$$\varepsilon_{max} \leq \varepsilon_{WL} + \varepsilon_M + \varepsilon_T. \tag{20}$$

The error ε_{WL} is related to the word length as $\varepsilon_{WL} = 2^{-WL}$, where WL is the number of bits used to represent the data in the implementation. The error ε_M represents the relation between the number of iterations computed in CORDIC-based algorithms and the maximum error of the approximation. It is calculated as $\varepsilon_M \leq \tan^{-1}(2^{-M})$, where M is the number of iterations computed in the implementation. The error ε_T is the error outcome of the approximation of the gain compensation over T_i. This error is constant. We have measured this error to be $\varepsilon_T \approx 4.3382 \cdot 10^{-10}$ rad. This leads to the theoretical limit of the proposed algorithm, being ε_T the maximum resolution that can be obtained by applying this algorithm.

We can state that, since $\tan(x) \approx x$ for small values of x, then, $\varepsilon_{WL} \approx \varepsilon_M$ when $WL = M$. Therefore, the limit of the algorithm is reached when ε_T is larger than both ε_{WL} and ε_M. Each, ε_{WL} and ε_M, are equal to ε_T for a word length of 32 bits and 32 iterations. Note that a precision of ε_T is enough for most applications.

VI. Conclusion

In this paper, a novel algorithm based on CORDIC for the computation of arcsine and arccosine functions has been presented. Contrary to previous CORDIC-based approaches, the proposed algorithm does not require neither to increase the complexity of the iterations nor to compute real multiplications in order to compute the functions accurately. The hardware implementation of the proposed algorithm has been provided, proving that a better trade-off between area and error is obtained when compared to other CORDIC-based implementations.

References

[1] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Computing*, vol. EC-8, pp. 330–334, Sep. 1959.

[2] M. Garrido, P. Källström, M. Kumm, and O. Gustafsson, "CORDIC II: A new improved CORDIC algorithm," *IEEE Trans. Circuits Syst. II*, vol. 63, no. 2, pp. 186–190, Feb. 2016.

[3] L. Vachhani, K. Sridharan, and P. K. Meher, "Efficient CORDIC algorithms and architectures for low area and high throughput implementation," *IEEE Trans. Circuits Syst. II*, vol. 56, no. 1, pp. 61–65, Jan. 2009.

[4] C.-S. Wu, A.-Y. Wu, and C.-H. Lin, "A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes," *IEEE Trans. Circuits Syst. II*, vol. 50, no. 9, pp. 589–601, Sep. 2003.

[5] J. S. Walther, "A unified algorithm for elementary functions," in *AFIPS Joint Comp. Conf.*, May 1971, pp. 379–385.

[6] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. ACM/SIGDA Int. Symp. FPGAs*, Feb. 1998, pp. 191–200.

[7] E. Manor, A. Ben-David, and S. Greenberg, "CORDIC hardware acceleration using DMA-based ISA extension," *J. Low Power Electron. App.*, vol. 12, no. 1, pp. 1–12, Jan. 2022.

[8] A. Sergiyenko, L. Moroz, L. Mychuda, and V. Samotyj, "FPGA implementation of CORDIC algorithms for sine and cosine floating-point calculations," in *IEEE Int. Conf. Intell. Data Acq. Adv. Comput. Syst.: Tech. App.*, vol. 1, Sep. 2021, pp. 383–386.

[9] C. Mazenc, X. Merrheim, and J.-M. Muller, "Computing functions cos/sup -1/ and sin/sup -1/ using CORDIC," *IEEE Trans. Comput.*, vol. 42, no. 1, pp. 118–122, Jan. 1993.

[10] X. Liu, Y. Xie, H. Chen, and B. Li, "Implementation on FPGA for CORDIC-based computation of arcsine and arccosine," in *IET Int. Radar Conf.*, Oct. 2015, pp. 1–4.

[11] W. Zhu, S. Ye, Y. Huang, and Z. Xue, "Design of a precise subdivision system for gratings using a modified CORDIC algorithm," *IET Circuits, Devices and Syst.*, vol. 13, no. 8, pp. 1284–1291, Nov. 2019.

[12] A. Saha, A. Ghosh, and K. Kumar, "FPGA implementation of arcsine function using CORDIC algorithm," *Adv. Model. Anal.*, vol. 54, no. 2, pp. 196–201, Sep. 2017.

[13] T. Lang and E. Antelo, "CORDIC-based computation of arccos and arcsin," in *Proc. IEEE Int. Conf. App.-Specific Syst.*, Jul. 1997, pp. 132–143.

[14] M. Zechmeister, "Solving Kepler's equation with CORDIC double iterations," *Mon. Not. R. Astron. Soc.*, vol. 500, no. 1, pp. 109–117, Nov. 2020.

[15] W. Zhu, S. Ye, Y. Huang, and Z. Xue, "An improved CORDIC for digital subdivision of Moiré signal," *Metrol. Meas. Syst.*, vol. 27, no. 1, pp. 51–64, 2020.

[16] F. Salehi, E. Farshidi, and H. Kaabi, "Novel design for a low-latency CORDIC algorithm for sine-cosine computation and its implementation on FPGA," *Microprocess. Microsyst.*, vol. 77, pp. 1–7, Sep. 2020.

[17] J. P. Lim, M. Shachnai, and S. Nagarakatte, "Approximating trigonometric functions for posits using the CORDIC method," in *Proc. Int. Conf. Comp. Frontiers*, May 2020, p. 19–28.