Architecture Implementation of an improved Decimal CORDIC Method

Jose-Luis Sanchez, Higinio Mora, Jeronimo Mora, and Antonio Jimeno
Computer Technology Department, University of Alicante, Spain
{sanchez, hmora, jeronimo, jimeno}@dtic.ua.es

Abstract—Since radix-10 arithmetic has been gaining renewed importance over the last few years, high performance decimal systems and techniques are highly demanded. In this paper, a modification of the CORDIC method for decimal arithmetic is proposed so as to improve calculations. The algorithm works with BCD operands and no conversion to binary is needed. A significant reduction in the number of iterations in comparison to the original decimal CORDIC method is achieved. The experiments showing the advantages of the new method are described. Also, the results with regard to delay obtained by means of an FPGA implementation of the method are shown.

I. INTRODUCTION

Numbers are commonly expressed by human beings using decimal representation; as a consequence, in the early days of computing, most of the first computers worked with decimal operands [1]. Due to the greater simplicity of binary arithmetic unit and the compactness of binary numbers, decimal arithmetic fell into disuse and for many years it has been difficult to find new proposals of radix 10-based computers. This fact has finally led to a preponderance of binary systems over decimal ones. In spite of that, some examples of decimal architectures can be found, such as Hewlett Packard [2], Texas Instruments [3] and Casio calculators, and some others [4].

In recent years, a renewed interest in decimal arithmetic computing has arisen, since it is essential for many applications. For instance, financial calculations are carried out using decimal arithmetic, as binary operations often imply rounding up or down the results when working with fractional operands. Several studies involving financial and business-oriented applications have revealed that 55% of the numerical data contained in commercial databases are in decimal format [5]. The need for high precision engineering and manufacturing systems is also essential in CAD/CAM. When defining a radix-10 magnitude for an object, the internal use of radix-2 usually implies loss of precision, since the equivalent binary number is likely to have an infinite amount of fractional digits. On the other hand, there are currently optic and magnetic sensors which directly provide the output in BCD format, so that the user can easily monitor the evolution of certain magnitudes and detect any errors [6]. The same happens with some types of actuators

which use ISO-ASCII as the code for inputting data to the manufacturing process [7].

Proof of the importance recently given to decimal representation is the fact that even the IEEE 854 standard uses a radix-independent generalization of IEEE 754 and supports decimal floating point operations [8] [9]. Recently the IBM z900 microprocessor has been developed [10], with a decimal arithmetic unit. Furthermore, the European Commission specifies a certain number of decimal digits for calculating currency conversions [11].

CORDIC (COordinate Rotation Digital Computer) is a relevant method to approximate mathematical functions [12]. It works as an iterative algorithm for approximating rotation of a two-dimensional vector using only add and shift operations. CORDIC is particularly suited to hardware implementations due to its simplicity. Originally, CORDIC was applied to binary arithmetic, but later its application was proposed for decimal data [13] [14].

In this paper, a new CORDIC method for decimal operands is proposed, based on the use of decimal arithmetic and on the selection of adequate angles so as to reduce the number of iterations required to obtain a suitable precision. In section II, both binary and decimal CORDIC are reviewed. In section III, the new decimal CORDIC method is proposed. An architecture carried out on FPGA is proposed throughout section IV and the results of a series of experiments with regard to precision and the required number of stages are showed. Finally, in section V, conclusions are given.

II. REVIEWING THE CORDIC METHOD

A. Reviewing the binary CORDIC method

Rotating a 2D point (x, y) through an angle θ can be directly performed by means of the following equations:

$$x^R = x \cos \theta - y \sin \theta \quad (1)$$

$$y^R = x \sin \theta + y \cos \theta \quad (2)$$

The above equations involve a high computational cost due to the fact that some products and also the values of $\cos \theta$ and $\sin \theta$ must be calculated.

CORDIC was developed by Volder [12] for computing the rotation of a 2D vector of circular coordinates expressed

as binary numbers, exclusively using addition and shift operations. Walther [15] extended the method to hyperbolic and linear coordinates. CORDIC works in two different modes. In *rotation mode*, a vector (x_0, y_0) is rotated through an angle θ in order to obtain a new vector (x_n, y_n) . The overall rotation is divided into micro-rotation such that, in micro-rotation j , an angle $\alpha_j = \tan^{-1}(2^j)$ is added to or subtracted from the remaining angle θ_j . In this way, this angle approaches zero. In *vectoring mode*, the vector (x_0, y_0) is progressively rotated towards the x -axis by means of angles such as those previously mentioned, so that the component y approaches 0. As a result, the sum of all the angles applied gives the angle of vector (x_0, y_0) towards the x -axis, whereas the final component x_n is the vector magnitude. The algorithm is based on the following equations:

$$x_{j+1} = x_j - m \sigma_j y_j 2^{-d(j)} \quad (3a)$$

$$y_{j+1} = y_j + \sigma_j x_j 2^{-d(j)} \quad (3b)$$

$$z_{j+1} = z_j - w_{d(j)} \quad (3c)$$

The variable σ_j determines the direction of micro-rotation j . In vectoring mode, σ_j is equal to 1 if y_j is negative, and σ_j is equal to -1 otherwise. The values for m , $d(j)$ and $w_{d(j)}$ are shown in Table 1, whereas Table 2 shows the results provided by the algorithm with regard to the type of coordinates [16]. With slight variations on the operands, additional functions can be calculated such as $\ln(x)$, $x^{1/2}$, and some others. The elementary angles α_j must fulfil the following condition:

$$\alpha_j \leq \sum_{k=j+1}^n \alpha_k + \alpha_n, j \geq 0 \quad (4)$$

With regard to the elementary angles chosen for circular coordinates, convergence is guaranteed since the following property is accomplished:

$$\tan^{-1}(2^{-j}) \leq \sum_{k=j+1}^n \tan^{-1}(2^{-k}), j \geq 0 \quad (5)$$

When working with hyperbolic coordinates, carrying out each micro-rotation only once is not sufficient. Indeed, convergence can be achieved by repeating certain iterations,

Table 1. Parameters for Different Coordinate Type.

Type	$d(j)$	$w_{d(j)}$	m
Circular	j	$\tan^{-1}(2^j)$	1
Hyperbolic	$j - k$, k is the largest integer such that $3^{k+1} + 2k - 1 \leq 2j$	$\tanh^{-1}(2^j)$	-1
Linear	j	2^j	0

Table 2. Results for Different Coordinate Type

Type	Results on vectoring
Circular	$x_n = K_1(x_0^2 + y_0^2)^{1/2}$ $z_n = z_0 + \tan^{-1}(y_0/x_0)$
Hyperbolic	$x_n = K_{-1}(x_0^2 - y_0^2)^{1/2}$ $y_n = 0$ $z_n = z_0 + \tanh^{-1}(y_0/x_0)$
Linear	$x_n = x_0$ $y_n = 0$ $z_n = z_0 - y_0/x_0$

as shown in Table 1.

In iteration j , a scaling factor is added to the new coordinates (x_j, y_j) . This factor is given by the following expression:

$$K_{m,j} = \sqrt{1 + m 2^j} \quad (6)$$

The coordinates obtained after the last iteration have to be compensated by multiplying them by K_m^{-1} , taking into account that K_m results from the following product:

$$K_m = \prod_j K_{m,j} \quad (7)$$

Several methods to avoid performing the final product by K_m^{-1} and carry out the scaling compensation in parallel with each of the iterations have been proposed [17]-[21].

B. Reviewing the decimal CORDIC method

The CORDIC method is flexible and simple, so it is suitable for environments in which a small number of hardware resources are available. One of these environments is that of portable calculators [2]. However, these devices usually work with numbers in decimal format and, therefore, binary CORDIC cannot be directly used. In [13] and [22] the use of CORDIC for BCD operands is proposed. The modification of the method, focusing on the case of circular coordinates, is expressed by the following equations:

$$x_{j+1} = x_j - \sigma_j y_j 10^j \quad (8)$$

$$y_{j+1} = y_j + \sigma_j x_j 10^j \quad (9)$$

$$z_{j+1} = z_j - \tan^{-1}(10^j) \quad (10)$$

The drawback of this decimal CORDIC method lies on the relation between any two consecutive elementary angles in the form $\tan^{-1}(10^j)$. The relation between any two consecutive angles in the form $\tan^{-1}(2^j)$ is approximately 2. This fact facilitates convergence in binary CORDIC, as expressed in (5). However, in the case of decimal representation, each angle is approximately 10 times smaller than the previous one, so convergence of the method cannot be directly guaranteed. According to decimal CORDIC, each iteration but the initial one must be repeated 9 times so as to

achieve convergence [13]. In this case, the following condition is fulfilled:

$$\tan^{-1}(10^{-j}) \leq 9 \sum_{k=j+1}^n \tan^{-1}(10^{-k}), \quad j \geq 0 \quad (11)$$

References [13] and [22] show the results for different experiments carried out using 30 angular steps. For both the vector magnitude $(x^2 + y^2)^{1/2}$ and the function $\tan^{-1}(y/x)$, a precision of 5 digits is obtained. These results are suitable in terms of precision. However, this method cannot compete with binary CORDIC with regard of latency, since the binary method requires a smaller number of iterations so as to obtain the same precision. Thus, the advantages of using CORDIC on BCD operands would be reduced to omit conversion between BCD and binary representation and, consequently, to avoid loss of precision.

In the next section, the vectoring mode of a new decimal CORDIC method with reduced number of iterations is described.

III. THE NEW DECIMAL CORDIC METHOD

The new decimal CORDIC method in vectoring mode is based on the selection of elementary angles α_j such that

$$\alpha_j = \tan^{-1}(c10^{-d}) \quad (12)$$

where d is the magnitude difference between x_j and y_j , whereas c is the integer quotient between the first significant digit of y_j and the first significant digit of x_j . That is, if the subset of positive integers from 0 to 9 is referred to as $\mathbb{N}_9$,

$$[x_j] = p10^{-q}, p \in \mathbb{N}_9, q \in \mathbb{N} \wedge p10^{-q} \leq |x_j| \leq (p+1)10^{-q} \quad (13)$$

$$[y_j] = r10^{-s}, r \in \mathbb{N}_9, s \in \mathbb{N} \wedge r10^{-s} \leq |y_j| \leq (r+1)10^{-s} \quad (14)$$

$$c \in \mathbb{N}_9, d \in \mathbb{N} \wedge c10^{-d} < [x_j]/[y_j] \leq (c+1)10^{-d} \quad (15)$$

Since $\tan(\alpha_j) = c10^{-d}$, the recursive calculation of x , y , and z is defined by means of the following equations:

$$x_{j+1} = x_j + [y_j] c10^{-d} \quad (16)$$

$$y_{j+1} = y_j + \sigma_j x_j c10^{-d} \quad (17)$$

$$z_{j+1} = z_j - \sigma_j \tan^{-1}(c10^{-d}) \quad (18)$$

In the first iteration, a default value of $c10^{-d} = 1$ is assumed so as to cope with those cases in which the value of x is lower than the value of y . In the following iterations, as the value of x always increases, y_j will always be smaller than x_j . Thus, the magnitude difference between x and y will be always positive. For the method to perform smaller oscillations, the value of c is decremented by 1 if both $[x_j]$ and $[y_j]$ are different from 1. If $[x_j] = [y_j]$, then the value $c = 1$ is assumed. The variable σ_j is the same as explained for B-CORDIC.

From this point on, the new decimal CORDIC will be referred to as ND-CORDIC, whereas binary CORDIC and the previous decimal CORDIC will be referred to as B-CORDIC and D-CORDIC, respectively. Table 3 shows an example where the initial coordinates are $x_0 = 0.706479$, $y_0 = 0.304659$. Table 4 shows the angles added in the different iterations to accumulate the final z . After the 8th iteration, the calculated value for x is 1.139611, as shown in table 3. The direct calculation of the vector (x, y) modulus is 0.769370. The divergence between both results is caused by the scaling factor incorporated throughout the different iterations. The computation of the factor for compensating this scaling can be obtained by means of the following expression:

$$K_{ND}^{-1} = \prod_{j=0}^n \cos(\tan^{-1}(c10^{-d})) \quad (19)$$

In table 3, the values for the scaling compensation factors incorporated within the first iterations of the above example are also shown. For relatively small values of α_j , the scaling factor can be assumed to be to 1. The product of the partial scaling compensation factors, as defined in (19), gives an overall value of $K_{ND}^{-1} = 0.675116$. The product $x_8 K_{ND}^{-1}$ is 0.769370. The angle value directly calculated gives the result $\Theta = \tan^{-1}(y/x) = 0.407141$, which is the same as the one obtained with the ND-CORDIC, as shown in table 4.

Table 3. Values for x_j , y_j , c , d , and scaling compensation factors.

j	x	y	c	d	$\cos(\tan^{-1}(c10^{-d}))$
0	0.706479	0.304659	1	0	0.707107
1	1.011138	-0.40182	3	1	0.957826
2	1.131684	-0.098479	8	2	0.996815
3	1.139562	-0.007944	6	3	0.998205
4	1.13961	-0.001107	1	3	1
5	1.139611	0.000033	2	5	1
6	1.139611	0.00001	1	5	1
7	1.139611	-0.000001	1	6	1
8	1.139611	0.00000005	4	8	1

Table 4. Values for z_j

j	$c10^{-d}$	$-\sigma_j \tan^{-1}(c10^{-d})$	z_j
0	1	0.785398	0.785398
1	0.3	-0.291457	0.493941
2	0.08	-0.079830	0.414111
3	0.006	-0.006000	0.408111
4	0.001	-0.001000	0.407111
5	0.00002	0.000020	0.407131
6	0.00001	0.000010	0.407141
7	0.000001	-0.000001	0.407140
8	0.00000004	0.00000004	0.407141

The scale factor compensation by means of multiplication should be avoided due to the high computational cost of this operation. In B-CORDIC, compensation without products is easy to perform due to the fact that the scale factor is a constant [12]. In ND-CORDIC this factor varies depending on the different angles chosen through the method iterations.

A technique based on LUT (Look-Up Tables) can be used which allows the compensation to be performed on each iteration. Equations (16) and (17) can be modified so as to include the compensation, which results in the following expression, where the superscript C indicates that the coordinates are scaling-compensated:

$$x_{j+1}^C = (x_j + |y_j| c 10^{-d}) \cos(\tan^{-1}(c 10^{-d})) \quad (20)$$

$$y_{j+1}^C = (y_j + \sigma_j x_j c 10^{-d}) \cos(\tan^{-1}(c 10^{-d})) \quad (21)$$

Taking into account (12), the above equations can be rewritten in the following way:

$$x_{j+1}^C = x_j \cos(\alpha_j) + |y_j| c 10^{-d} \cos(\alpha_j) \quad (22)$$

$$y_{j+1}^C = y_j \cos(\alpha_j) + \sigma_j x_j c 10^{-d} \cos(\alpha_j) \quad (23)$$

In equations (22) and (23), four terms appear:

$$t_{x,0} = x_j \cos(\alpha_j) \quad (24)$$

$$t_{y,0} = y_j \cos(\alpha_j) \quad (25)$$

$$t_{x,1} = c 10^{-d} y_j \cos(\alpha_j) \quad (26)$$

$$t_{y,1} = c 10^{-d} x_j \cos(\alpha_j) \quad (27)$$

The proposed compensation technique consists in storing the above four terms in four independent blocks of LUT. The entries for each block of LUT consist of the one-digit mantissa c and the exponent d , and also the value of x_j or y_j . If each term was stored on a single LUT, the size of each LUT would be excessive. For instance, when a precision of 6 fractional digits is required, then 24 bits are needed for each coordinate, 4 for indicating the mantissa c , and 3 for indicating the exponent d (1 to 8). Thus, the size of a monolithic LUT for each term would be $2^{4+3+4} \cdot 4 \cdot 6 = 6144$ MBytes. Instead, much smaller LUT can be used. If the different BCD X3 digits of x_j are considered, the term $t_{x,0}$ can be expressed as:

$$t_{x,0} = (x_j[5] x_j[4] x_j[3] x_j[2] x_j[1] x_j[0]) \cos(\alpha_j) \quad (28)$$

Therefore, each small LUT will receive as inputs the value of a single digit of the coordinate, the mantissa c and the exponent d . For 6 fractional digits, the size of each LUT would be $2^{4+3+4} \cdot 4 \cdot 6 = 6$ KB. Since 6 fractional digits and four terms must be considered, the overall memory size would be $6 \text{ KB} \cdot 6 \cdot 4 = 144 \text{ KB}$. In this case, 6 LUT would constitute the storage block for $t_{x,0}$, other 6 LUT would compound the LUT block for $t_{x,1}$, and so on.

IV. DECIMAL CORDIC ARCHITECTURE

A. Some Details on the Architecture Implementation

Addition on BCD operands is more complex than binary addition since the sum of two BCD digits must be corrected adding the value 6 to this sum if it is greater than 9. BCD Excess 3 (BCD X3) representation allows decimal addition and subtraction to be more efficiently performed, since only two 4-bit binary adders are required for each pair of digits. The final result is directly obtained in BCD X3. More detailed information on BCD X3 adders can be found in [13]. Conversions from BCD to BCD X3 and from BCD X3 to BCD both require only 10 gates distributed throughout 3 level. Therefore, the use of BCD X3 is proposed since addition, subtraction, and other operations are simpler than for BCD. The overall architecture for each one of the iterations of the proposed ND-CORDIC is shown in Fig 1. The α_j -calculator extracts the exponent and mantissa from z_j so as to address the LUT.

B. Experiments on Precision

Different tests were carried out so as to make a complete comparison with regard to precision between B-CORDIC, D-CORDIC, and the ND-CORDIC proposed in this work. Values within the range $[0, 1)$ were chosen for the (x, y) coordinates. Original data were represented in BCD with 6 fractional digits. For D-CORDIC and ND-CORDIC, a conversion stage from BCD to BCD X3 was included, whereas for B-CORDIC the BCD operands were converted into binary numbers. In any case, 28-bit operands were considered. The experiments were aimed at comparing the number of iterations required for each method so as to achieve suitable precision.

A benchmark of 1000 random points (x, y) was selected so as to calculate the angle of vector (x, y) towards the x -

Fig. 1. The architecture for a single ND-CORDIC iteration.

axis, as well as the vector magnitude. The results for the maximum relative error in calculating the vector modulus are depicted in Fig. 2, whereas Fig. 3 shows the maximum relative error in computing the vector angle. In both cases, a logarithmic scale was chosen. The relative error for the modulus, e_{mod} , and for the angle, e_{ang} , were respectively calculated by means of the following equations:

$$e_{mod} = 100 (x_c - x_r) / x_r \quad (29)$$

$$e_{ang} = 100 (z_c - z_r) / z_r \quad (30)$$

In equations (29) and (30), x_c and z_c are respectively the modulus and the angle calculated by the CORDIC methods, whereas x_r and z_r are the reference values obtained respectively by the direct calculations $z_r = \tan^{-1}(y/x)$ and $x_r = (x^2 + y^2)^{1/2}$.

A decreasing tendency can be observed for every method as the number of iterations increases. However, the error decreases much faster for ND-CORDIC. For this method, the error reaches stability in about 10 iterations, whereas for D-CORDIC and B-CORDIC much more iterations are required. In addition, the maximum relative error is always lower for the ND-CORDIC method.

C. Experiments on Latency and Hardware resources

The proposed architecture was implemented on VHDL using the Xilinx ISE 7.1i tool. The Virtex4 XC4VLX60 FPGA was chosen for simulation. The architectures for D-CORDIC proposed in [13] and [22] and for B-CORDIC were also implemented. For every method, a complete stage was implemented, with the type of adder and shifter, if needed, being varied according to each method. Conventional arithmetic was used. In case of B-CORDIC, the scaling factor was compensated by means of the method proposed in [20], which allows the compensating product to be transformed into additional shift-add iterations. In case of ND-CORDIC, the compensation was achieved by means of the LUT technique previously described. An initial conversion from BCD to BCD X3 and a final conversion the other way were also included for D-CORDIC and for ND-CORDIC. For B-CORDIC, an initial conversion from BCD to binary and a final conversion the other way were implemented. A homogeneous length of 28 bits was used for every number format, so six fractional digits, that is, 24 bits, were considered for the BCD original numbers.

The results for delay and FPGA resources used, when considering a single iteration and including number format conversions and scaling compensation, are shown in Table 5 for comparison. The delay for the proposed ND-CORDIC is less than half the delay for the conventional D-CORDIC. Although B-CORDIC has the least delay per iteration, it must be taken into account that a higher number of iterations must be performed so as to achieve a precision similar to the one obtained with ND-CORDIC.

An overall comparison between the three methods including precision and delay, has been performed. The results are shown in Fig. 4, where the mean relative error according to delay in nanoseconds is depicted for each architecture. ND-CORDIC offers better global performance than D-CORDIC and B-CORDIC for the considered time lapse.

Fig. 2. Maximum relative error on calculating the modulus of random vectors ($\blacktriangle$ = D-CORDIC; $\blacksquare$ = B-CORDIC; $\square$ = ND-CORDIC).

Table 5. Single Stage Delay for Different CORDIC Architectures

Results	B-CORDIC	D-CORDIC	ND-CORDIC
Delay (ns)	48.413	225.182	103.042
Slices	909 (3%)	5785 (21%)	8585 (32%)
4-input LUT	1626 (3%)	5416 (10%)	9746 (18%)
Bonded IOB	172 (26%)	172 (26%)	172 (26%)

Fig. 3. Maximum relative error on calculating the angle of random vectors ($\blacktriangle$ = D-CORDIC; $\blacksquare$ = B-CORDIC; $\square$ = ND-CORDIC).

Fig. 4. Mean relative error according to delay for each CORDIC architecture ($\blacktriangle$ = D-CORDIC; $\blacksquare$ = B-CORDIC; $\square$ = ND-CORDIC).

D. Conclusions

One of the most important tasks in new hardware design is to achieve high performance with a trade-off between precision and delay. There is a growing trend towards developing new systems integrating decimal arithmetic, which is required in many areas.

In this work a new CORDIC method for performing calculations on decimal coordinates has been proposed. The tests performed confirm that the proposed method requires fewer iterations so as to obtain a required precision. Moreover, the maximum error obtained is always lower for the proposed method than for binary and decimal CORDIC.

On the other hand, with regard to latency, the experiments show that the proposed method has a much lower delay than that for decimal CORDIC. In addition, the indices obtained are not very far from those obtained by binary CORDIC.

As a future work, an interesting task consists in developing a hardware implementation of a specific CORDIC-based calculator embedded on a decimal architecture. At this point, new scaling compensation techniques must be studied and developed so as to improve delay and resources utilization.

ACKNOWLEDGMENT

This work has been supported by the Valencia Region Government (Spain) under Grant No. GV/2007/173.

REFERENCES

- [1] H. H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)", *IEEE Annals Hist. Comput.*, vol. 18 #1, pp. 10-16, 1996.
- [2] D. S. Cochran, "Algorithms and Accuracy in the HP-35", *HP Journal*, pp. 10-11, July 1972.

- [3] TI-89/TI-92 Plus Developers Guide, Beta Version .02, Texas Instruments, Jan. 2001.
- [4] M. F. Cowlishaw, "Decimal Floating Point: Algorithm for Computers", *Proc. 16th IEEE Symp. Computer Arithmetic*, 2003.
- [5] A. Tsang and M. Olschanowsky, "A Study of Database 2 Customer Queries", IBM Santa Teresa Laboratory, San Jose, CA, Technical Report TR-03-413, 1991.
- [6] S. Kim, J. Kwon, S. Kim, and B. Lee, "Multiplexed Strain Sensor using Fiber Grating-Tuned Fiber Laser with a Semiconductor Optical Amplifier", *IEEE Photonics Technology Letters*, vol. 13, no. 4, pp. 350-351, 2001.
- [7] S. McMains, J. Smith, and C. Séquin, "The evolution of a layered manufacturing interchange format", *Proc. DETC02, ASME Design Engineering Technical Conferences*, pp.945-953, Sept. 2002.
- [8] *IEEE 854-1987 – IEEE Standard for Radix-Independent Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., New York, 1987.
- [9] *Draft IEEE Standard for Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., New York, 2005.
- [10] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, "The IBM z900 Decimal Arithmetic Unit", *Proc.35th Asilomar Conf. Signals, Systems and Computers*, pp. 1335-1339, 2001.
- [11] European Commission Directorate General II, "The Introduction of the Euro and the Rounding of Currency Amounts", Note II/28/99-EN Euro Papers no. 22., 32pp, Belgium, 1999.
- [12] J. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330-334, 1959.
- [13] H. Schmid, *Decimal Computation*. NY: John Wiley & Sons, 1974.
- [14] J. C. Kropa, "Calculator Algorithms", *Mathematics Magazine*, vol. 51, no. 2, pp. 106-109, March 1978.
- [15] J. S. Walther, "A unified algorithm for elementary functions", *Proc. AFIPS Spring Joint Computer Conf.*, 1971, pp. 379-385.
- [16] J.-M. Muller. *Elementary Functions. Algorithms and Implementation*. Birkhäuser, 2006.
- [17] A. Despain, "Fourier Transform Computers Using CORDIC Iterations", *IEEE Trans. Comput.*, vol. C-23, no. 10, pp. 993-1001, Oct. 1974.
- [18] E. F. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC architecture for fast VLSI filtering and array processing", *Proc. ICASSP'84*, 1984, pp. 41.A.6.1-41.A.6.4.
- [19] G. Haviland and A. Tuszynski, "A CORDIC Arithmetic Processor Chip", *IEEE Trans. Comput.*, vol. C-29, no. 2, pp. 68-79, Feb. 1990.
- [20] D. Timmermann, H. Hahn, B. J. Hosticka, and B. Rix, "A new addition scheme and fast scaling factor compensation methods for CORDIC algorithms", *INTEGRATION, the VLSI Journal*, 11, pp. 85-100, 1991.
- [21] J. Villalba, J. A. Hidalgo, E. L. Zapata, E. Antelo, and J. D. Bruguera, "CORDIC Architectures with Parallel Compensation of the Scale Factor", *Proc. IEEE Int. Conf. Application-Specific Array Processors*, July 1995, pp. 258-269.
- [22] H. Schmid and A. Bogacki, "Use decimal CORDIC for generation of many transcendental functions", *EDN*, pp. 64-73, Feb. 1973.