Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2008 13–17 June 2008.

A CORDIC-BASED METHOD FOR IMPROVING DECIMAL CALCULATIONS

Jose-Luis Sanchez[1], Higinio Mora[1], Jeronimo Mora[1] and Francisco-Javier Ferrandez[1]

[1] *Department of Computer Technology, University of Alicante*

emails: sanchez@dtic.ua.es, hmora@dtic.ua.es, jeronimo@dtic.ua.es, fjferran@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.

Key words: Decimal arithmetic, CORDIC, hardware design

1 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 for codifying the input 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]. This method basically works as an iterative algorithm for approximating rotation of a two-dimensional vector using only add and shift operations. It is particularly suited to hardware implementations due to the fact that it does not require any multiplication. Originally, CORDIC was applied to binary arithmetic, but later its application was proposed for decimal data, as detailed in [13] and [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 2, both the binary and the decimal CORDIC method are reviewed. In section 3, the new CORDIC method is proposed. In order for a real system to operate with our method, an architecture carried out on FPGA is proposed throughout section 4 and the results of a series of experiments with regard to precision and the required number of stages are showed. Finally, in section 5, conclusions are summarized.

2 The CORDIC Method

2.1 Reviewing the binary CORDIC Method

The rotation of a 2D point $(x,\ y)$ through an angle θ can be directly computed by means of the following equations:

$$x^R = x\cos\theta - y\sin\theta \tag{1}$$

$$y^R = x\sin\theta + y\cos\theta \tag{2}$$

The above equations involves a high computational cost due to the fact that some multiplications and the previous calculation of $\cos\theta$ and $\sin\theta$ must be performed.

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 value of 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:

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

$$y_{j+1} = y_j + \sigma_j x_j 2^{-d(j)} \tag{4}$$

$$z_{j+1} = z_j - w_{d(j)} \tag{5}$$

The parameter σ_j determines the direction of micro-rotation j. In rotation mode, σ_j is equal to 1 if z_j is positive, and σ_j is equal to -1 otherwise. 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].

The elementary angles α_j must fulfil the following condition [16]:

$$\alpha_j \leq \sum_{k=j+1}^{n} \alpha_j + \alpha_n \ , j \geq 0 \tag{6}$$

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

$$\tan^{-1}(2^{-j}) \leq \sum_{k=j+1}^{n} \tan^{-1}(2^{-k}) \ , j \geq 0 \tag{7}$$

When working with hyperbolic coordinates, carrying out each micro-rotation only once is not sufficient. Indeed, convergence can be achieved by repeating certain iterations, as shown in Table 1.

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

Table 1: Parameters for Different Coordinate Type.

Type	Results on rotation	Results on vectoring
Circular	$x_n = K_1(x \cos z - y \sin z)$ $y_n = K_1(y \cos z + x \sin z)$ $z_n = 0$	$x_n = K_1(x_0^2 + y_0^2)^{1/2}$ $y_n = 0$ $z_n = z_0 + \tan^{-1}(y_0/x_0)$
Hyperbolic	$x_n = K_{-1}(x \cosh z + y \sinh z)$ $y_n = K_{-1}(y \cosh z + x \sinh z)$ $z_n = 0$	$x_n = K_{-1}(x_0^2 - y_0^2)^{1/2}$ $y_n = 0$ $z_n = \mathrm{z}_0 + \tanh^{-1}(y_0/x_0)$
Linear	$x_n = x$ $y_n = y + xz$ $z_n = 0$	$x_n = x_0$ $y_n = 0$ $z_n = z_0 - y_0/x_0$

Table 2: Results for Different Coordinate Type.

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

$$K_{m,j} = \sqrt{1 + m\ 2^{-j}} \tag{8}$$

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:

$$K_m^{-1} = \prod_j K_{m,j} \tag{9}$$

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].

2.2 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 iterative equations:

$$x_{j+1} = x_j - \sigma_j y_j 10^{-j} \tag{10}$$

$$y_{j+1} = y_j + \sigma_j x_j 10^{-j} \tag{11}$$

$$z_{j+1} = z_j - \tan^{-1}(10^{-j}) \tag{12}$$

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 (**??**). 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. This is not specific of radix 10 representation. Recall that in binary CORDIC applied to hyperbolic coordinates, certain iterations must be repeated so as to guarantee convergence. 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:

$$\tan^{-1}(10^{-j}) \leq 9 \sum_{k=j+1}^{n} \tan^{-1}(10^{-k}) \ , j \geq 0 \tag{13}$$

References [13] and [22] show that decimal CORDIC can compute sine and cosine functions with a 5-digit accuracy if at least 30 angular steps are performed. 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. Therefore, the advantages of using the algorithm with BCD operands would be reduced to omit conversion between BCD and binary representation and, consequently, to avoid loss of precision.

3 The New Decimal CORDIC

3.1 Rotation Mode

The proposed new decimal CORDIC in rotation mode, focusing exclusively on circular coordinates, is based on the selection of successive angles α_j such that:

$$\alpha_j = \tan^{-1}([z_j]) \tag{14}$$

where $[z_j]$ is the value resulting from truncating z_j after the first digit on the left different from 0. That is,

$$[z_j] = p10^{-q}, p \in \{0, 1, ..., 9\}, q \in \mathbb{N} \wedge p10^{-q} \leq z_j \leq (p+1)10^{-q} \tag{15}$$

In this way, the z component for accumulating the remaining angle is calculated by means of the following expression:

$$z_{j+1} = z_j - \tan^{-1}([z_j]) \tag{16}$$

Since $\tan(\alpha_j) = [z_j]$, the equations for computing x and y are expressed as follows:

$$x_{j+1} = x_j - \sigma_j[z_j]y_j \tag{17}$$

$$y_{j+1} = y_j + \sigma_j[z_j]x_j \tag{18}$$

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 rotating angle is $z_0 = 0.785398$. The different values for z_j, $[z_j]$, and $tan^{-1}([z_j])$

Iteration	x_j	y_j	z_j	$[z_j]$	$\tan^{-1}([z_j])$
$j=0$	0.931420	0.538504	0.785398	0.7	0.610725
$j=1$	0.554467	1.190498	0.174672	0.1	0.099668
$j=2$	0.435417	1.245945	0.075003	0.07	0.069886
$j=3$	0.348201	1.276424	0.005117	0.005	0.004999
$j=4$	0.341822	1.278165	0.000117	0.0001	0.0001
$j=5$	0.341694	1.278199	0.000017	0.00001	0.00001
$j=6$	0.341681	1.278202	0.000007	0.000007	0.000007
$j=7$	0.341669	1.278205	0.000000	0.000000	0.000000

Table 3: Values for z_j, $[z_j]$, and $\tan^{-1}([z_j])$.

according to each iteration j are presented. Table 3 also shows the different values for x_j and y_j according to each iteration and the above mentioned rotation angle, and taking into account the initial values $x_0 = 0.931420$ and $y_0 = 0.538504$. As shown in the last row of Table 3, the final values for the rotated coordinates are $x_7 = 0.341669$, $y_7 = 1.278205$. The rotation of the original point, directly computed by means of (1) and (2), gives the values $x^R = 0.277834$ and $y^R = 1.039393$. The divergence between (x_7, y_7) and (x^R, y^R) is caused by the scaling factor that is incorporated within each ND-CORDIC iteration. The computation of the factor for compensating this scaling can be obtained by means of the following expression:

$$K_{ND}^{-1} = \prod_{j=0}^{n} \cos(\alpha_j) \tag{19}$$

In Table 4, the values for the scaling compensation factor incorporated within the first iterations of the above example are shown. For relatively small values of $[z_j]$, the scaling factor can be assumed to be equal to 1. The last row contains the value for the global scaling factor K_{ND}^{-1} as defined in (**??**). The results of the $x_7 K_{ND}^{-1}$ and $y_7 K_{ND}^{-1}$ give an error in the order of 10^{-7} for coordinates x and y. 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]. However, in ND-CORDIC this factor varies depending on the different angles chosen through the method iterations, as shown in Table 4.

A technique based on LUT (look-up tables) can be used which allows the compensation to be performed on each iteration. Equations (17) and (18) can be modified in order to include the compensation, which results in the following expression, where the superscript C indicates that the coordinates are scaling-compensated:

$$x_{j+1}^C = (x_j - \sigma_j [z_j] y_j) \cos(\tan^{-1}([z_j])) \tag{20}$$

$$y_{j+1}^C = (y_j + \sigma_j [z_j] x_j) \cos(\tan^{-1}([z_j])) \tag{21}$$

Iteration	$[z_j]$	$\cos(\tan^{-1}([z_j]))$
$j=0$	0.7	0.81923192
$j=1$	0.1	0.99503719
$j=2$	0.07	0.99755897
$j=3$	0.005	0.9999875
$j=4$	0.0001	1
$j=5$	0.00001	1
$j=6$	0.000007	1
K_{ND}^{-1}		0.81316621

Table 4: Terms Determining the scaling compensation factor.

The above equations can be rewritten in the following way:

$$x_{j+1}^{C} = x_j \cos(tan^{-1}([z_j]) - \sigma_j[z_j] y_j \cos(\tan^{-1}([z_j])) \tag{22}$$

$$y_{j+1}^{C} = y_j \cos(tan^{-1}([z_j]) + \sigma_j[z_j] x_j \cos(\tan^{-1}([z_j])) \tag{23}$$

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

$$t_{x,0} = x_j \cos(\tan^{-1}([z_j])) \tag{24}$$

$$t_{y,0} = y_j \cos(\tan^{-1}([z_j])) \tag{25}$$

$$t_{x,1} = [z_j] y_j \cos(\tan^{-1}([z_j])) \tag{26}$$

$$t_{y,1} = [z_j] x_j \cos(\tan^{-1}([z_j])) \tag{27}$$

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 and the exponent of $[z_j]$, 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 of $[z_j]$, and 3 for indicating the exponent of of $[z_j]$ (from 000 to 110). Thus, the size of a monolithic LUT for each term would be $2^{4\cdot 6+3+4} \cdot 4 \cdot 6 = 6144$ 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:

$$t_{x,0} = (x_j[5]x_j[4]x_j[3]x_j[2]x_j[1]x_j[0]) \cos(\tan^{-1}([z_j])) \tag{28}$$

Therefore, each small LUT will receive as inputs the value of a single digit of the coordinate and the mantissa and exponent of $[z_j]$. 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 KB $\cdot$ 6 $\cdot$ 4 = 144 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.

3.2 Vectoring Mode

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

$$\tan(\alpha_j) = c10^{-d} \tag{29}$$

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,

$$[x_j] = p10^{-q}, p \in 0, 1, ..., 9, q \in \mathbb{N} \wedge p10^{-q} \leq |x_j| \leq (p+1)10^{-q} \tag{30}$$

$$[y_j] = r10^{-s}, r \in 0, 1, ..., 9, s \in \mathbb{N} \wedge r10^{-s} \leq |y_j| \leq (r+1)10^{-s} \tag{31}$$

$$\alpha_j = \tan^{-1}(c10^{-d}), c \in 0, 1, ..., 9, d \in \mathbb{N} \wedge c10^{-d} < [x_j]/[y_j] \leq (c+1)10^{-d} \tag{32}$$

The recursive calculation of x, y and z is defined by means of the following equations:

$$x_{j+1} = x_j + |y_j|c10^{-d} \tag{33}$$

$$y_{j+1} = y_j + \sigma_j x_j c10^{-d} \tag{34}$$

$$z_{j+1} = z_j - \sigma_j \tan^{-1}(c10^{-d}) \tag{35}$$

In the first iteration, a default value of $c10^{-d} = 1$ is assumed so as to manage with those cases in which the value of x is lower than the value of y. In the following iterations, since the value of x always increases, y_j will always be smaller than x_j. Thus, the magnitude difference between x and y will always be 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 both digits are equal, the value $c = 1$ is assumed. The variable σ_j has the same meaning as expressed for B-CORDIC in equations (3) and (4).

Table 5 shows an example where the initial coordinates are $x_0 = 0.706479$, $y_0 = 0.304659$. After the 8^{th} iteration, the calculated value for x is 1.139611. The direct calculation of the vector (x, y) modulus is 0.769370. The divergence between both results is, again, caused by the scaling factor incorporated throughout the different iterations. The scaling compensation factors corresponding to each of the first iterations, as shown in (19), are also depicted. When the elementary angle is small enough, the scaling factor can be assumed to be 1. The product $x_8 K_{ND}^{-1}$ is 0.769370. Moreover, 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.

4 Decimal CORDIC Architecture. Experimentation.

4.1 Some Details on the Architecture Implementation

Addition on BCD operands is more complex than binary addition since the carry resulting from the sum of two digits must be propagated to the sum of the following ones [13]. Moreover, the sum of two BCD digits must be corrected adding the value 6 to this sum if it is greater than 9.

BCD X3 representation allows decimal addition/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]. Conversion from BCD to BCD X3 requires only 10 gates distributed over 3 level, and similar resources are needed when transforming BCD X3 into BCD. Therefore, the use of BCD X3 is proposed since addition, subtraction, and other operations are simpler than for BCD. The complete architecture for each one of the iterations of the proposed ND-CORDIC method is shown in Fig. 1.

j	x	y	c	d	- $\sigma_j \tan^{-1}(c10^{-d})$	z_j	$\cos(\tan^{-1}(c10^{-d}))$
0	0.706479	0.304659	1	0	0.785398	0.785398	0.707107
1	1.011138	-0.40182	3	1	- 0.291457	0.493941	0.957826
2	1.131684	-0,098479	8	2	- 0.079830	0.414111	0.996815
3	1.139562	-0.007944	6	3	- 0.006000	0.408111	0.998205
4	1.13961	-0.001107	1	3	- 0.001000	0.407111	1
5	1.139611	0.000033	2	5	0.000020	0.407131	1
6	1.139611	0.000010	1	5	0.000010	0.407141	1
7	1.139611	-0.0000010	1	6	- 0.000001	0.407140	1
8	1.139611	0.00000005	4	8	0.00000005	0.407141	1
K_{ND}^{-1}							0.675116

Table 5: Values for x_j, y_j, c, d, z_j and scaling compensation factors.

4.2 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 and also for the rotation angle θ. 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.

In the *rotation* test, 1000 random points were rotated through a random angle. The results for the maximum relative error are shown in Fig. 2. Since the rotation result is a new coordinate vector, the relative error e_{rot} on *rotation* was calculated by means of the following expression:

$$e_{rot} = 100\sqrt{\frac{(x_c - x_r)^2 + (y_c - y_r)^2}{x_r^2 + y_r^2}} \tag{36}$$

where (x_c, y_c) are the coordinate values calculated by each method, whereas the reference values directly calculated are (x_r, y_r). In the *vectoring* test, 1000 random points (x, y) were selected so as to calculate the angle of vector (x, y) towards the x-axis, as well as the vector magnitude. The results for the maximum relative error in calculating the angle and the vector magnitude are depicted in Fig. 3. As both the angle and the modulus are scalar values, the relative error for the modulus, e_{mod}, and for the angle, e_{ang}, were respectively calculated by means of the following equations, taking into account that z_c is the angle calculated by the CORDIC methods, whereas z_r is the reference value obtained by the direct operation $z_r = \tan^{-1}(y/x)$:

$$emod = 100\frac{|x_c - x_r|}{x_r} \tag{37}$$

$$eang = 100\frac{|z_c - z_r|}{z_r} \tag{38}$$

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 error is always lower for the ND-CORDIC method. The maximum relative error for ND-CORDIC seems to be always lower than that for D-CORDIC and B-CORDIC.

4.3 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. In all cases, a complete stage of the method was implemented, with the type of adder and shifter, if needed, being varied according to each method. The global method was implemented on an unfolded architecture. 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 simple additional shift-add iterations. In case of ND-CORDIC, the compensation was achieved by means of the LUT technique previously described. The initial conversion from BCD to BCD X3 and the final conversion the other way were also included for the D-CORDIC and the ND-CORDIC methods. For B-CORDIC, an initial conversion from BCD to binary and a final conversion the other way were also implemented. A homogeneous length of 28 bits was used for every number format, so six fractional digits, corresponding to 24 bits, were considered for the BCD original numbers.

The results for the delays and the FPGA resources used, when considering a single iteration and including number format conversions and scaling compensation, are shown in Table 6 for comparison. As it can be observed, 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

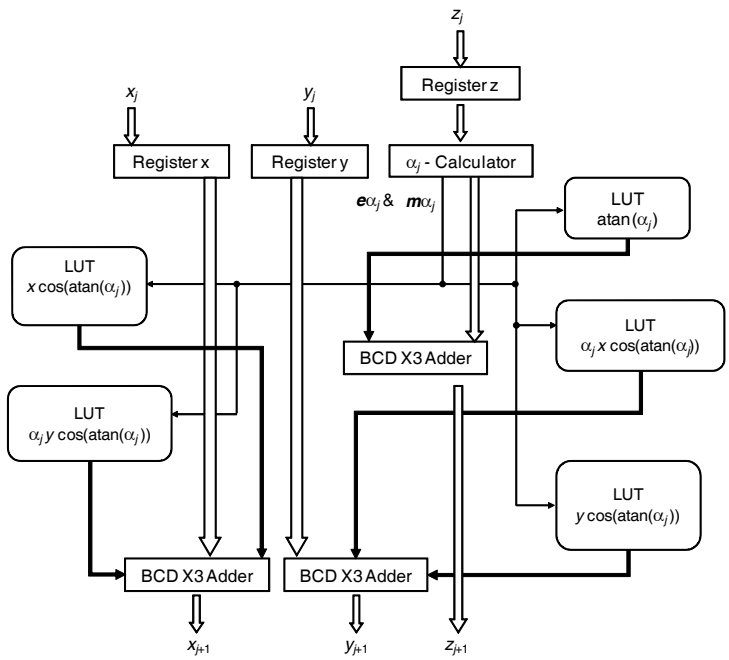

Figure 1: Architecture for a single ND-CORDIC iteration.

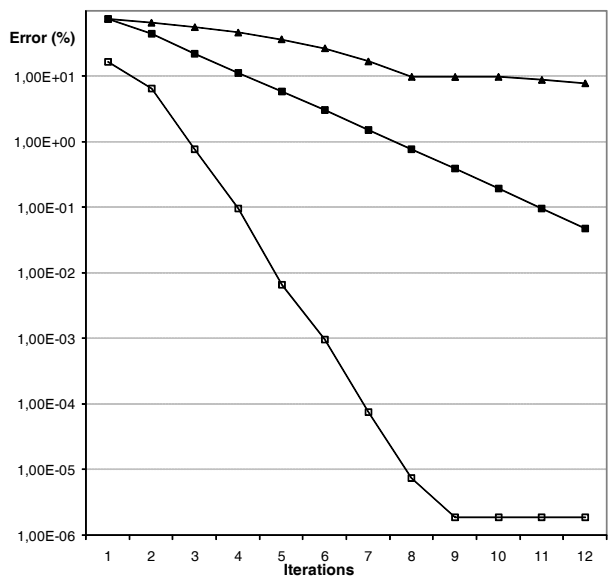

Figure 2: Maximum relative error on rotation according to the number of iterations; logarithmic scale ▲= D-CORDIC; ■= B- CORDIC; □= ND-CORDIC).

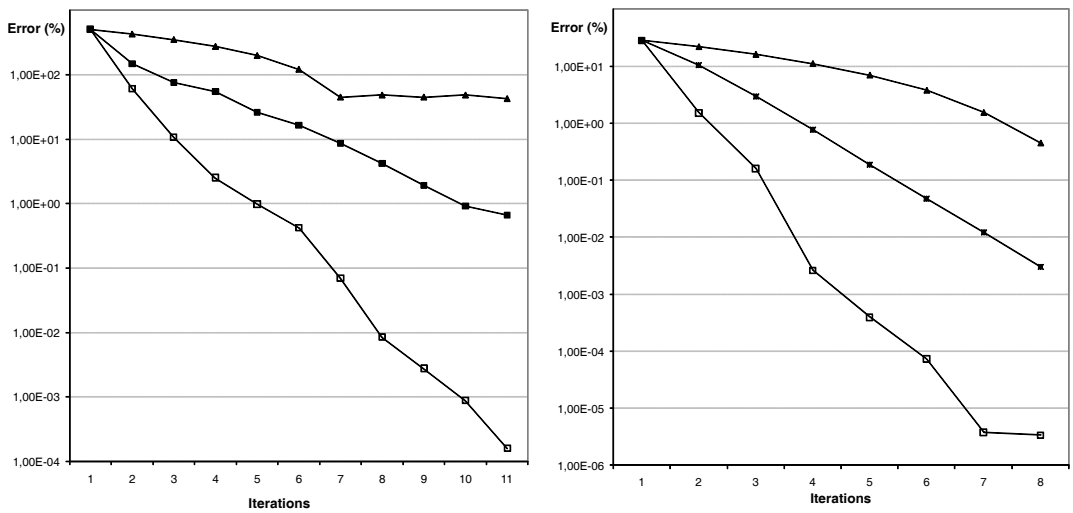

Figure 3: Maximum relative error on calculating vector angle (left) and magnitude according to the number of iterations; logarithmic scale (▲= D-CORDIC; ■= B-CORDIC; □= ND-CORDIC).

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 product of the mean relative error and the delay in nanoseconds is depicted for each architecture. It can be observed that ND-CORDIC offers better global performance than D-CORDIC and B-CORDIC for the considered iterations.

5 Conclusions

One of the most important tasks in new hardware design is to achieve high performance rates with a trade-off between precision and delays of the circuitry that forms these new embedded architectures. A growing trend towards developing new systems integrating decimal arithmetic, which is required in many practical research areas, is arisen. In this work a new CORDIC method for performing rotations 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.

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%)

Table 6: Single Stage Delay for Different CORDIC Architectures.

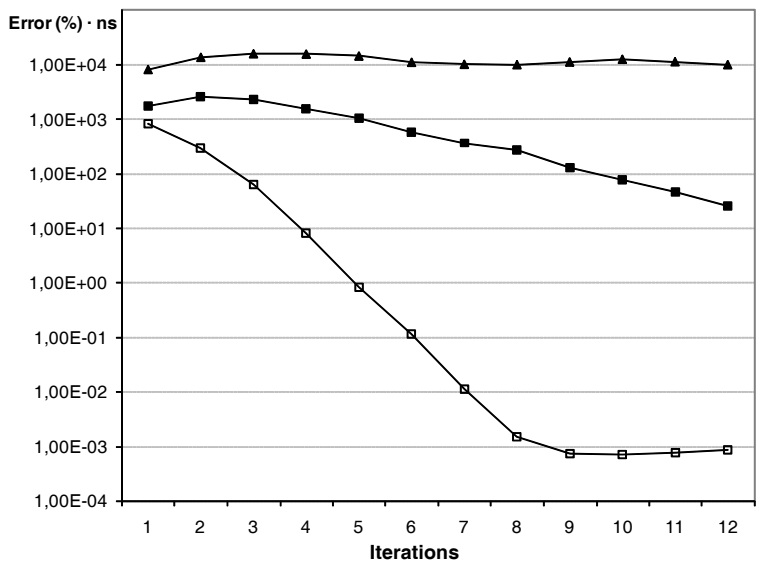

Figure 4: Error × delay, according to the number of iterations; logarithmic scale (▲= D-CORDIC; ■= B- CORDIC; □= ND-CORDIC).

References

[1] H. H. Goldstine and A. Goldstine, *The Electronic Numerical Integrator and Computer (ENIAC)*, IEEE Annals Hist. Comput. **18#1** (1996) 10–16.

[2] D. S. Cochran, *Algorithms and Accuracy in the HP-35*, HP Journal (1972) 10–11.

[3] Texas Instrument, *TI-89/TI-92 Plus Developers Guide, Beta Version .02*, 2001.

[4] M. F. Cowlishaw, *Decimal Floating Point: Algorism for Computers*, Proc. 16th IEEE Symp. Computer Arithmetic (2003) 104–111.

[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 Gratin-Tuned Fiber Laser with a Semiconductor Optical Amplifier*, IEEE Photonics Technology Letters **13.4** (2001) 350–351.

[7] S. McMains, J. Smith, and C. Séquin, *The evolution of a layered manufacturing interchange format*, Proc. DETC02, ASME Design Engineering Technical Conferences (2002) 945–953.

[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 (2001) 1335–1339.

[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. **EC-8.3** (1959) 330–334.

[13] H. Schmid, *Decimal Computation*, John Wiley & Sons, New York, 1974.

[14] J. C. Kropa, *Calculator Algorithms*, Mathematics Magazine, **51.2** (1978) 106–109.

[15] J. S. Walther, *A unified algorithm for elementary functions*, Proc. AFIPS Spring Joint Computer Conf. (1971) 379–385.

[16] J.-M. Muller, *Elementary Functions. Algorithms and Implementation*, Birkhäuser, 1997.

[17] A. Despain, *Fourier Transform Computers Using CORDIC Iterations*, IEEE Trans. Comput. **C-23.10** (1974) 993–1001.

[18] E. F. Deprettere, P. Dewilde and R. Udo, *Pipelined CORDIC architecture for fast VLSI filtering and array processing*, Proc. ICASSP'84 (1984) 41.A.6.1–41.A.6.4.

[19] G. Haviland and A. Tuszynski, *A CORDIC Arithmetic Processor Chip*, IEEE Trans. Comput. **C-29. 2** (1990) 68–79.

[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** (1991) 85–100.

[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 (1995) 258–269.

[22] H. Schmid and A. Bogacki, *Use decimal CORDIC for generation of many transcendental functions*, EDN, (1973) 64–73.