Elsevier Editorial System(tm) for Mathematical and Computer Modelling
Manuscript Draft

Manuscript Number: MCM-D-11-01529R1

Title: An Approach to the Application of Shift-and-add Algorithms on Engineering and Industrial Processes

Article Type: SI: MME&HB2011

Keywords: Shift-and-add algorithms; Engineering; CAD/CAM; Numerical Methods

Corresponding Author: Dr. Jose-Luis Sanchez,

Corresponding Author's Institution:

First Author: Jose-Luis Sanchez

Order of Authors: Jose-Luis Sanchez; Antonio Jimeno-Morenilla; Rafael Molina-Carmona; Jose Perez-Martinez

Manuscript Region of Origin: SPAIN

Cover Letter

for the submission of

"An Approach to the Application of Shift-and-add Algorithms on Engineering and Industrial Processes"

Authors: Jose-Luis Sanchez-Romero, Antonio Jimeno-Morenilla[1], Rafael Molina-Carmona[2], and Jose Perez-Martinez[1]

[1]Department of Computing Technology and Data Processing, University of Alicante, Spain

[2]Department of Computing Sciences and Artificial Intelligence, University of Alicante, Spain

Dear Sir or Madam,

Please kindly consider the manuscript

"An Approach to the Application of Shift-and-add Algorithms on Engineering and Industrial Processes"

for publication in the Special Issue corresponding to the *Mathematical Modelling in Engineering and Human Behaviour 2011 Conference* on the Mathematical and Computer Modelling Journal.

Please do not hesitate to request any additional information that might be needed in order to make your decision.

Sincerely,

Jose-Luis Sanchez-Romero

October 7, 2011

Corresponding author:

Jose-Luis Sanchez-Romero

Department of Computing Technology and Data Processing

Polytechnic University College, University of Alicante

Carretera de San Vicente, s/n

03690 San Vicente del Raspeig, Alicante (Spain)

e-mail: sanchez@dtic.ua.es

Phone number: +34 965 90 3681

Fax number: +34 965 90 9643

***Detailed Response to Reviewers**

Detailed responde to reviewers of

"An Approach to the Application of Shift-and-add Algorithms on Engineering and Industrial Processes" (MCM-D-11-01529)

The reference missing in page 8, sentence "so as to calculate the distance shown in Eq. (??)" has been corrected.

The caption under Figure 1 was wrong. It has been modified so as to indicate the correct meaning of the figure.

Jose-Luis Sanchez-Romero

November 23, 2011

Corresponding author:
Jose-Luis Sanchez-Romero
Department of Computing Technology and Data Processing
Polytechnic University College, University of Alicante
Carretera de San Vicente, s/n
03690 San Vicente del Raspeig, Alicante (Spain)

e-mail: sanchez@dtic.ua.es
Phone number: +34 965 90 3681
Fax number: +34 965 90 9643

An Approach to the Application of Shift-and-add Algorithms on Engineering and Industrial Processes

Jose-Luis Sanchez-Romero[a,*], Antonio Jimeno-Morenilla[a], Rafael Molina-Carmona[b], Jose Perez-Martinez[a]

[a] *Department of Computer Technology and Data Processing, University of Alicante, 03690 Alicante, Spain*
[b] *Department of Computing Sciences and Artificial Intelligence, University of Alicante, 03690 Alicante, Spain*

Abstract

Different kinds of algorithms can be chosen so as to compute elementary functions. Among all of them, it is worthwhile mentioning the shift-and-add algorithms due to the fact that they have been specifically designed to be very simple and to save computer resources. In fact, the almost only operations usually involved on these methods are additions and shifts, which can be easily and efficiently performed by a digital processor. Shift-and-add algorithms allow fairly good precision with low cost iterations. The most famous algorithm belonging to this type is CORDIC. CORDIC has the capability of approximating a wide variety of functions with the only help of a slight change on their iterations. In this paper, we will analyze the requirements of some engineering and industrial problems in terms of type of operands and functions to approximate. Then, we will propose the application of shift-and-add algorithms based on CORDIC to these problems. We will make a comparison between the different methods applied in terms of the precision of the results and the number of iterations required.

Keywords: Shift-and-add algorithms, Engineering, CAD/CAM

1. Introduction

CORDIC (COordinate Rotation DIgital Computer) is an iterative algorithm to approximate mathematical functions [1]. It was originally developed for approximating rotation of a two-dimensional vector using only shift and add operations. It is specially suited to hardware implementations due to the fact that it does not require any multiplication. The CORDIC algorithm has been

*Corresponding author
Email addresses: sanchez@dtic.ua.es (Jose-Luis Sanchez-Romero), jimeno@dtic.ua.es (Antonio Jimeno-Morenilla), rmolina@dccia.ua.es (Rafael Molina-Carmona), jperez@dtic.ua.es (Jose Perez-Martinez)

widely applied in the field of digital signal processing, with solutions in signal filtering, matrix processing algorithms, improvement of image processing algorithms, equation systems solvers, and so on. It has also played a significant role in telecommunications, specially in the design of processors for wireless modems. CORDIC was originally designed for working with radix-2 operands. However, in recent years, a renewed interest in decimal computing has arisen, since it is essential for many applications and scopes [2]. For instance, the need for high performance decimal systems is essential in CAD/CAM. When defining a radix-10 magnitude for an object, the 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 supply the output in BCD format, so that the user can easily monitor the evolution of some magnitudes and detect any errors [3]. The same happens with some types of actuators which use ISO-ASCII for codifying the data inputted to the manufacturing process [4]. Moreover, most financial calculations are carried out using decimal arithmetic, since binary operations often imply rounding up or down the results when working with fractional operands. In [5], it is concluded that 55% of the numerical data contained in commercial databases are represented in decimal formats. Furthermore, the European Commission specifies a certain number of decimal digits for calculating currency conversions [6].

Some facts reinforce the increasing importance given to decimal representation. The IEEE 854 standard uses a radix-independent generalization of IEEE 754 and supports decimal floating point operations [7],[8]. Recently, IBM developed the System z9 [9] and the latest System z10 [10] processors, each of them including a decimal arithmetic unit. The current trend towards decimal computation is also mirrored in the literature, mainly via the state-of-the-art in decimal addition [11], [12].

In this paper, we will analyze the requirements of some engineering and industrial problems in terms of type of operands and functions to approximate. Then, we will propose the application of different CORDIC versions to these problems. We will make a comparison between the different methods applied in terms of the precision of the results and the number of iterations required.

The paper is structured as follows. In section 2, the original, binary CORDIC, an old approach to CORDIC for decimal operands, and a more recent decimal CORDIC, are reviewed. In section 3, the application of CORDIC to a concrete engineering/industrial environment is proposed, including experimentation with regard to both precision and latency. Finally, in section 4, conclusions are summarized.

2. The CORDIC Method

2.1. Reviewing the binary CORDIC Method

CORDIC was originally developed for computing the rotation of a 2D vector of circular coordinates expressed as binary numbers, almost exclusively using

addition and shift operations [1]. The method was extended to support hyperbolic and linear coordinates [16]. 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). With regard to circular coordinates, 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. Another component, z, is used which represents the angle accumulation or decomposition. The algorithm is based on the following generalized equations:

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

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

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

The parameter σ_j determines the direction of micro-rotation j. In rotation mode, $\sigma_j = 1$ if z_j is positive, and σ_j = -1 otherwise. In vectoring mode, σ_j = 1 if y_j is negative, and σ_j = -1 otherwise. The values for t, $d(j)$ and $w_{d(j)}$ are shown in Table 1. Table 2 shows the results provided by the algorithm with regard to the type of coordinates in rotation and in vectoring mode [16]. When working with hyperbolic coordinates, carrying out each micro-rotation only once is not sufficient [17]. Indeed, convergence can be achieved by repeating certain iterations, as shown in Table 1.

Table 1: Parameters for different coordinate type

Type	t	$d(j)$	$w_{d(j)}$
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}

In iteration j, a scaling factor is added to the new coordinates (x_{j+1}, y_{j+1}). This factor is given by the following expression:

$$K_{t,j} = \sqrt{1 + t\ 2^{-2j}} \quad (4)$$

The overall scaling factor can be determined by means of the following product:

$$K_t = \prod_j K_{t,j} \quad (5)$$

Table 2: Results on rotation mode and vectoring mode for different coordinate type

Type	Rotation ($z_n = 0$)	Vectoring ($y_n = 0$)
Circular	$x_n = K_1(x \cos z - y \sin z)$	$x_n = K_1(x_0^2 + y_0^2)^{1/2}$
	$y_n = K_1(y \cos z + x \sin z)$	$z_n = z_0 + \tan^{-1}(y_0/x_0)$
Hyperbolic	$x_n = K_{-1}(x_1 \cosh z_1 + y_1 \sinh z_1)$	$x_n = K_{-1}(x_1^2 - y_1^2)^{1/2}$
	$y_n = K_{-1}(y_1 \cosh z_1 + x_1 \sinh z_1)$	$z_n = z_1 + \tanh^{-1}(y_1/x_1)$
Linear	$x_n = x_0$	$x_n = x_0$
	$y_n = y_0 + x_0 z_0$	$z_n = z_0 - y_0/x_0$

Since the coordinates obtained after the last iteration are scaled, they have to be compensated by multiplying them by K_t^{-1}. Several methods to avoid performing the final product by K_t^{-1} and carry out the scaling compensation in parallel with each of the iterations have been proposed [17]-[22].

2.2. Reviewing an old approach to decimal CORDIC Method

In [13] and [23] the use of CORDIC for BCD operands is proposed. The modification of the binary method, focusing on the case of circular coordinates, is expressed by the following iterative equations:

$$x_{j+1} = x_j - \sigma_j y_j 10^{-j} \quad (6)$$

$$y_{j+1} = y_j + \sigma_j x_j 10^{-j} \quad (7)$$

$$z_{j+1} = z_j - \tan^{-1}(10^{-j}) \quad (8)$$

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, which facilitates convergence in binary CORDIC. However, in case of the decimal algorithm, each elementary angle is about 10 times smaller than the previous one, so convergence of the method cannot be directly guaranteed. This is not specific of radix 10 representation; in binary CORDIC applied to hyperbolic coordinates, certain iterations (e.g, $j = 4, 13, 40, \ldots$) must be repeated so as to guarantee convergence. In case of decimal CORDIC, each iteration but the initial one must be repeated 9 times in order to achieve convergence [13]. References [13] and [23] show that the results achieved by decimal CORDIC are suitable in terms of precision. However, this method cannot compare to binary CORDIC with regard to 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.

2.3. ND-CORDIC : A recent Decimal CORDIC

In [25], a new CORDIC which works with decimal operands is proposed. Besides avoiding the tasks of conversion from decimal to radix-2 representation and vice versa, which implies neither conversion error nor time consumption implicit in these stages, the specific features of this recent decimal CORDIC allow a significant reduction in the number of required iterations. From this point on, the new decimal CORDIC will be referred to as ND-CORDIC, whereas binary CORDIC and the older decimal CORDIC [13] will be referred to as B-CORDIC and D-CORDIC, respectively.

2.3.1. Rotation Mode

From this point on, we will focus exclusively on circular coordinates. The ND-CORDIC in rotation mode is based on the selection of successive angles α_j such that:

$$\alpha_j = \tan^{-1}(m_j^r 10^{-e_j^r}) \tag{9}$$

where $m_j^r 10^{-e_j^r}$ is the value resulting from truncating z_j after the first digit on the left different from 0. That is,

$$m_j^r \in \{0,1,...,9\}, e_j^r \in \mathbb{N} \wedge m_j^r 10^{-e_j^r} \leq z_j \leq (m_j^r + 1)10^{-e_j^r} \tag{10}$$

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

$$z_{j+1} = z_j - \tan^{-1}(m_j^r 10^{-e_j^r}) \tag{11}$$

Since $\tan(\alpha_j) = m_j^r 10^{-e_j^r}$, the equations for computing x and y are expressed as follows:

$$x_{j+1} = x_j - \sigma_j m_j^r 10^{-e_j^r} y_j \tag{12}$$

$$y_{j+1} = y_j + \sigma_j m_j^r 10^{-e_j^r} x_j \tag{13}$$

The computation of the overall factor for compensating this scaling can be obtained by means of the following expression:

$$K_{ND}^{-1} = \prod_{j=0}^{n} \cos(\alpha_j) \tag{14}$$

2.3.2. Vectoring Mode

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

$$\tan(\alpha_j) = m_j^v 10^{-e_j^v} \tag{15}$$

where e_j^v is the magnitude difference between x_j and y_j, whereas m_j^v is the integer quotient between the first significant digit of y_j and the first significant digit of x_j. That is,

$$\alpha_j = \tan^{-1}(m_j^v 10^{-e_j^v}), m_j^v \in \{0,1,...,9\}, e_j^v \in \mathbb{N} \wedge$$
$$\wedge m_j^v 10^{-e_j^v} < [x_j]/[y_j] \leq (1 + m_j^v)10^{-e_j^v} \tag{16}$$

where $[x_j]$ and $[y_j]$ are, respectively, x_j and y_j truncated after the first significant digit, and can be formally defined in the following way:

$$[x_j] = p_j^x 10^{-q_j^x}, p_j^x \in \{0,1,...,9\}, q_j^x \in \mathbb{N} \wedge \\ \wedge p_j^x 10^{-q_j^x} \leq |x_j| \leq (1+p_j^x)10^{-q_j^x} \quad (17)$$

$$[y_j] = p_j^y 10^{-q_j^y}, p_j^y \in \{0,1,...,9\}, q_j^y \in \mathbb{N} \wedge \\ \wedge p_j^y 10^{-q_j^y} \leq |y_j| \leq (1+p_j^y)10^{-q_j^y} \quad (18)$$

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

$$x_{j+1} = x_j + |y_j| m_j^v 10^{-e_j^v} \quad (19)$$

$$y_{j+1} = y_j + \sigma_j x_j m_j^v 10^{-e_j^v} \quad (20)$$

$$z_{j+1} = z_j - \sigma_j \tan^{-1}(m_j^v 10^{-e_j^v}) \quad (21)$$

In the first iteration, a default value of $m_j^v 10^{-e_j^v} = 1$ is always assumed so as to manage with those cases in which x_0 is lower than y_0. In the following iterations, since the value of x_j always increases, y_j will always be smaller than x_j. Thus, the magnitude difference between x_j and y_j will always be positive when $j > 0$. For the method to perform smaller oscillations, the value of m_j^v is decremented by 1 if both $[x_j]$ and $[y_j]$ are different from 1. If both digits are equal, the default value $m_j^v = 1$ is used. The variable σ_j has the same meaning as expressed for B-CORDIC in equations (1) and (2).

2.3.3. Scaling compensation in ND-CORDIC

The scaling compensation by means of multiplication should be avoided due to the high computational cost of this operation. In B-CORDIC, compensation without multiplications is easy to perform because the scale factor is a constant [17]. However, in ND-CORDIC this factor varies depending on the different angles chosen throughout the different iterations.

A technique based on look-up tables (LUT) can be used which allows the compensation to be performed on each iteration. Equations (12) and (13) in rotation mode and equations (19) and (20) in vectoring mode can be modified so as to include the compensation, giving as a result the following expressions, where the superscript C indicates that the coordinates are scaling-compensated:

$$x_{j+1}^C = (x_j - \sigma_j y_j \tan(\alpha_j)) \cos(\alpha_j) \quad (22)$$

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

The above equations can be rewritten in the following way:

$$x_{j+1}^C = x_j \cos(\alpha_j) - \sigma_j y_j \tan(\alpha_j) \cos(\alpha_j) \quad (24)$$

$$y_{j+1}^C = y_j \cos(\alpha_j) + \sigma_j x_j \tan(\alpha_j) \cos(\alpha_j) \quad (25)$$

In equations (24) and (25), four different terms appear:

$$t_{x,0} = x_j \cos(\alpha_j) \quad (26)$$

$$t_{y,0} = y_j \cos(\alpha_j) \quad (27)$$

$$t_{x,1} = y_j \tan(\alpha_j) \cos(\alpha_j) \quad (28)$$

$$t_{y,1} = x_j \tan(\alpha_j) \cos(\alpha_j) \quad (29)$$

The compensation technique consists in storing the above four terms in four independent blocks of LUT. The input lines for each block of LUT consist of the one-digit mantissa $m_j^{\{r,v\}}$ and the exponent $e_j^{\{r,v\}}$, 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. Instead, a set of smaller LUT can be used for each term. As explained in [25], the overall storage size for a precision of p_a fractional digits for the angle and p_c fractional digits for each coordinate is given by $2^{4+4+\lceil \log_2(p_a) \rceil} \cdot 4p_c \cdot 4p_c$.

3. Application of CORDIC on Industrial Processes.

3.1. Specific characteristics of data in the industrial/engineering environment

A trending issue in CAD/CAM is the utilization of decimal operands, since in the earlier stages of the product design engineers work with radix-10 magnitudes, which are propagated throughout the whole manufacturing process. Therefore, decimal formats should be considered so as to represent data and operate with them. 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 in case it is greater than 9.

BCD Excess-3 (BCD XS3) format 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 XS3. More precise information on BCD XS3 adders can be found in [13]. Conversion from BCD to BCD XS3 and vice versa can be easily performed by means of a few logic gates. Therefore, the use of BCD XS3 is proposed since addition, subtraction, and other operations such as detection of zero and nine's complement are simpler than for BCD.

3.2. Computation of specific operations in industrial processes

One of the most important tasks in the manufacturing process is machining. It can be conceptually defined as the definition of an object to be manufactured by means of predefined tools. The more usual operation in machining is the removal of material by using a cutting tool. In the machining process, the main task is the tool path computation: obtaining a trajectory of tool centres that defines the required object to be machined with a given precision. Therefore, for every object point to be machined, the position of the tool centre must be determined. This task involves a high computational cost, requiring operations such as rotations and distance calculation from every object point to the tool

centre. An example of a hard-to-compute distance is the one required when using a toroidal tool for machinig [24]. A toroidal cutting tool is characterized by a major radius R, a minor radius r, and the coordinates of the torus center (T_x, T_y), as shown in Fig. 1. The distance from a point (p_x, p_y, p_z) to a toroidal tool centre is given by the following expression:

$$d(p_x, p_y, p_z) = (T_y - p_y) - \sqrt{\left(R + \sqrt{r^2 - (p_x - T_x)^2}\right)^2 - p_z^2} \tag{30}$$

In equation 30, two square roots appear:

$$sqrt_1 = \sqrt{r^2 - (p_x - T_x)^2} \tag{31}$$

$$sqrt_2 = \sqrt{(R + sqrt_1)^2 - p_z^2} \tag{32}$$

The argument of each one of these square roots is the difference of two squares. The computation of such operations has a high computational cost. However, it can be efficiently performed by the CORDIC algorithm. In fact, one of the results provided by this method working in vectoring mode with hyperbolic coordinates is precisely the square root of the diffence of two squares [17]. Therefore, an algorithm including two instances of CORDIC can be developed so as to calculate the distance shown in Eq. (30):

```
Algorithm distance (px, py, pz, Tx, Ty, R, r: real): real;
Variables srqt_1, sqrt_2, y1, x2, d: real;
y1 = px - Tx;
sqrt_1 = CORDIC_Hyp_Vec(x = r, y = y1);
x2 = R + sqrt_1;
sqrt_2 = CORDIC_Hyp_Vec(x = x2, y = pz);
d = Ty - py - sqrt_2;
return d;
End Algorithm
```

For B-CORDIC, the natural order of the iterations is altered as shown in Table 1, due to the fact that some of them must be performed twice so as to guarantee convergence. Moreover, parameters t, $d(j)$, and $w_{d(j)}$ also change, as expressed in Table 1. For D-CORDIC, repeating some iterations is also required so as to achieve convergence. In this case, the repetition are much more numerous than those for B-CORDIC. The experiments shown in [23] used 10 iterations for the initial stage, that is, for $j = 1$, and 9 iterations for the following ones. Parameters t and $w_{d(j)}$ are modified the same way as for B-CORDIC. For ND-CORDIC, parameters t and $w_{d(j)}$ adopt the same definition as for B-CORDIC and D-CORDIC. However, in case of ND-CORDIC it is not necessary to repeat any stage, due to the fact that the elementary angle chosen in each stage of the vectoring mode is not fixed, but it is specifically selected according to the value of x_j and y_j.

Data for experimentation must fulfill a set of constraints according to Eq. (30). These constraints are the following ones:

$$r \geq p_x - T_x \tag{33}$$

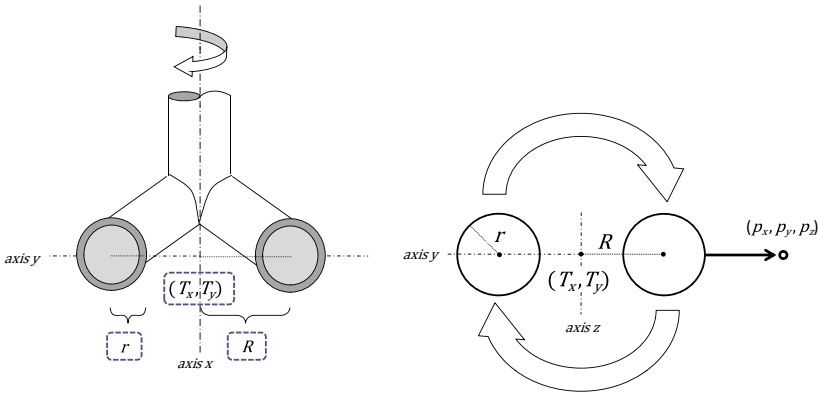

Figure 1: Structure of a toroidal tool and its main parameters.

$$R \quad +\sqrt{r^2 - (p_x - T_x)^2} \geq p_z \tag{34}$$

$$R \quad > r \tag{35}$$

$$T_y \quad > p_y \tag{36}$$

For B-CORDIC and D-CORDIC, there is a new constraint which is determined by the convergence range of both methods. In fact, the maximum hyperbolic angle supported by B-CORDIC is θ_b= 1.11817 rad, whereas for D-CORDIC this angle is θ_d= 1.102656 rad. Therefore, data were explicitly selected so as to involve a hyperbolic arctangent lower than 1.1. This constraint does not apply to ND-CORDIC, whose convergence range is wider than the one of the other two methods.

Different tests were carried out so as to make a complete comparison with regard to latency and precision between B-CORDIC, D-CORDIC, and ND-CORDIC. The architecture for the ND-CORDIC algorithm, as proposed in [25], was implemented on the hardware description language VHDL using the Xilinx ISE Design Suite 10.1 tool. The Virtex4 XC4VLX60 FPGA device was randomly chosen for simulation and synthesis. The architectures for D-CORDIC proposed in [13] and [23] and for B-CORDIC were also implemented. In all cases, a complete stage of the method was implemented.

Original data were selected in interval [0,1) and represented in BCD with 6 fractional digits. A homogeneous length of 28 bits was used for every number format, so 24 bits were used for the fractional digits of BCD and BCD XS3 numbers. The initial conversion from BCD to BCD XS3 and the final conversion the other way were also included for D-CORDIC and ND-CORDIC. For B-CORDIC, an initial conversion from BCD to binary and a final conversion the other way were also implemented. The results obtained with every algorithm were compared with those obtained by means of the direct application of Eq. (30), calculated in Matlab©. The mean relative error according to the execution time is depicted in Fig. 2. It must be taken into account that, for ND-CORDIC, experiments were performed with only six iterations of the first instance of

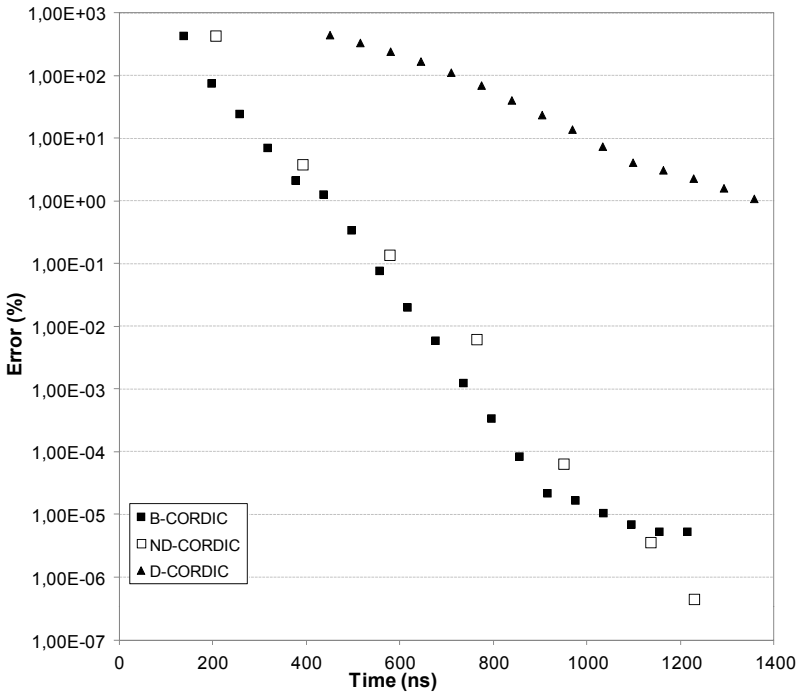

Figure 2: Mean relative error according to delay for each CORDIC architecture when calculating the toroidal tool distance; logarithmic scale.

CORDIC (calculation of $sqrt_1$). This limitation in the number of iterations is due to the fact that there is not a significant loss of precision and, moreover, the overall latency is reduced.

It can be observed that the error for ND-CORDIC is lower than that for D-CORDIC at any time interval. On the other hand, accuracy achieved by ND-CORDIC is similar to the one obtained with B-CORDIC and, indeed, ND-CORDIC provides a precision of 10^{-6} with a latency similar to the one of B-CORDIC. Besides, when a high number of iterations is performed, ND-CORDIC achieves an error lower that this limit, while B-CORDIC is not able to.

4. Conclusion

A growing trend towards developing new systems integrating decimal arithmetic, which is required in many engineering and industrial tasks, has arisen. In this paper, a proposal for the application of CORDIC on engineering and industrial processes has been shown. A manufacturing task with a high computational cost has been analyzed and, as a result, the application of CORDIC has been proposed for improving performance. Moreover, specific features of

operands involved in these tasks suggest the utilization of CORDIC versions specially developed for working with decimal operands. Experimentation has been performed for comparing the original, binary CORDIC (B-CORDIC) with two versions of CORDIC for decimal operands (D-CORDIC and ND-CORDIC). The results confirm that ND-CORDIC requires fewer iterations than both B-CORDIC and D-CORDIC so as to obtain a required precision. In addition, ND-CORDIC offers similar global performance than B-CORDIC and, in some cases, accuracy is even better than the one obtained with B-CORDIC. As a future work, the application of other shift-and-add methods on such processes, such as BKM, should also be tested and results compared with CORDIC. Moreover, more hard-to-compute operations typical of industry and engineering must be explored so as to try to speed them up with the application of shift-and-add methods.

Acknowledgments

This research was supported by the Conselleria de Educacion of the Valencia Region Government under grant number GV/2011/043.

References

[1] J. Volder, The CORDIC Trigonometric Computing Technique, *IRE Trans. Electron. Comput.*, **EC-8**(3) (1959) 330–334.

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

[3] 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.

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

[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] European Commission Directorate General II, The Introduction of the Euro and the Rounding of Currency Amounts, Note II/28/99-EN Euro Papers **22**, 32pp, Belgium, 1999.

[7] The Institute of Electrical and Electronics Engineers, Inc., *IEEE 854-1987 – IEEE Standard for Radix-Independent Floating-Point Arithmetic*, New York, 1987.

[8] The Institute of Electrical and Electronics Engineers, Inc., *754-2008 – IEEE Standard for Floating-Point Arithmetic*, New York, 2008.

[9] A.Y. Duale, M.H. Decker, H.-G. Zipperer, M. Aharoni and T.J. Bohizic, (2007) Decimal floating-point in z9: An implementation and testing perspective, *IBM J. Res. & Dev.*, **51**(1/2) (2007) 217–227.

[10] E.M. Schwarz, J. Kapernick and M. Cowlishaw, Decimal floating-point support on the IBM System z10 processor, *IBM J. Res. & Dev.*, **53**(1) (2009) 1–4.

[11] H. Thapliyal, S. Kotiyal and M. B. Srinivas. Novel BCD Adders and their Reversible Logic Implementation for IEEE 754r Format. *Proc. of 19th International Conference on VLSI Design* (2006) 387–392.

[12] S. Gorgin and G. Jaberipur, Fully Redundant Decimal Arithmetic, *Proc. 2009 19th IEEE International Symposium on Computer Arithmetic* (2009) 145–152.

[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] A. Vazquez and E. Antelo. Constant Factor CORDIC for Decimal BCD Input Operands. *8th Conference on Real Numbers and Computers*, Santiago de Compostela, Spain (2008) 83–91.

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

[17] J.-M. Muller, *Elementary Functions. Algorithms and Implementation* (Birkhäuser, 2006).

[18] A. Despain, Fourier Transform Computers Using CORDIC Iterations, *IEEE Trans. Comput.*, **C-23**(10) (1974) 993–1001.

[19] 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.

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

[21] 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.

[22] 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.

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

[24] A. Jimeno-Morenilla, *Modelado topologico del proceso de fabricacion* (*Topological modelling of the manufacturing process*), Ph.D. Thesis, University of Alicante, Alicante, Spain, 2003.

[25] J.L. Sanchez, H. Mora, J. Mora, and A. Jimeno, Function approximation on decimal operands, *Digital Signal Processing*, **11**(2) (2011), 354–366.