A Cordic-based Architecture for High Performance Decimal Calculations

Jose L. Sanchez, Antonio Jimeno, Higinio Mora, Jeronimo Mora, Francisco Pujol
Computer Technology Department
University of Alicante
Alicante, Spain
Email: sanchez@dtic.ua.es, jimeno@dtic.ua.es, hmora@dtic.ua.es, jeronimo@dtic.ua.es, fpujol@dtic.ua.es

Abstract—**Decimal arithmetic supported by digital computers has been gaining renewed importance over the last few years. However, the development of high performance radix 10-based systems is still incipient. In this paper, a modification of the CORDIC method for decimal arithmetic is proposed. The resulting algorithm works with radix 10 operands and combines decimal arithmetic with elementary angles so as to reduce the number of iterations required to achieve certain precision. Different experiments showing the advantages of the new method compared with the original decimal CORDIC method are also described. Finally, an architecture for the method implemented on FPGA is proposed.**

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 were based on radix 10 arithmetic [1]. However, due to the simplicity of binary hardware, decimal arithmetic fell into disuse and for many years it has been difficult to find new proposals of radix 10-based computers. In spite of that, one can find some examples, such as Hewlett Packard [2], Texas Instruments [3] and Casio calculators [4], and some others (see [5]-[10]).

In recent years, there has been a renewed interest in decimal arithmetic computing, 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 fraction operands are used. This loss of accuracy is also critical in Computer Aided Design (CAD), which demands high precision systems for engineering and manufacturing. As a result, the use of decimal arithmetic is growing so fast that even the IEEE 854 standard uses a radix-independent generalization of IEEE 754 and supports decimal floating point operations [11, 12].

In order to reduce the computation time of the operations involved in decimal arithmetic, which are usually implemented using binary logic, several methods have been proposed through the years. One of the most important methods to approximate mathematical functions is the CORDIC algorithm [13]. CORDIC (COordinate Rotation Digital Computer) method is an iterative algorithm for calculating trigonometric functions, including sine, cosine, magnitude and phase. It is particularly suited to hardware implementations because it does not require any product. Originally, CORDIC was applied to binary arithmetic, but later its application was proposed for decimal representation of data [14], [15].

In this work, a new CORDIC method in vectoring mode with decimal operands is proposed, based on the use of decimal arithmetic and angles of the type $\tan^{-1}(2^{-j})$. In this way, the number of iterations required to obtain suitable precision can be reduced. 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. In section 5, the results of a series of experiments with regard to precision and the required number of stages are showed. Finally, in section 6, the conclusions are given.

II. THE CORDIC METHOD

A. Reviewing the binary CORDIC method

CORDIC was developed by Volder [13] for computing the rotation of a 2D vector of circular coordinates expressed as binary numbers, exclusively using addition and shift operations. Walther [16] 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 the vector (x_0, y_0) towards the *x*-axis, whereas the final component x_n contains the vector magnitude. he algorithm is based on the following equations:

$$x_{j+1} = x_j - m \, \sigma_j \, y_j \, 2^{-d(j)} \tag{1a}$$
$$y_{j+1} = y_j + \sigma_j \, x_j \, 2^{-d(j)} \tag{1b}$$
$$z_{j+1} = z_j - w_{d(j)} \tag{1c}$$

The values for *m* are 1 for circular, -1 for hyperbolic, and 0 for linear coordinates. The value for σ_j determines the direction of micro-rotation *j*. In vectoring mode, if y_j is positive the value for σ_j is -1, whereas σ_j takes the value 1 otherwise. The values for $d(j)$ and $w_{d(j)}$ are shown in Table I, whereas Table II shows the results provided by the algorithm in vectoring mode depending on the type of coordinates. With slight modifications on the operands, CORDIC can also provide other functions such as logarithm, square root, tan^{-1}, and so on.

The elementary angles α_j must fulfil the following condition [19] [20]:

$$\alpha_j \le \sum_{k=j+1}^{n} \alpha_j + \alpha_n, \, j \ge 0 \tag{2}$$

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

$$\tan^{-1}(2^{-j}) \le \sum_{k=j+1}^{n} \tan^{-1}(2^{-k}), \, j \ge 0 \tag{3}$$

When working with hyperbolic coordinates, carrying out each micro-rotation only once is not sufficient. Indeed, convergence can be achieved by repeating certain iterations [20], as shown in Table I.

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

$$K_{m,j} = \sqrt{1 + m \, 2^{-j}} \tag{4}$$

TABLE I
PARAMETERS FOR DIFFERENT COORDINATE TYPE

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

TABLE II
RESULTS FOR DIFFERENT COORDINATE TYPE

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

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

$$K_m = \prod_j K_{m,j} \tag{5}$$

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 [18]-[22].

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 [14] and [17] 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:

$$x_{j+1} = x_j - \sigma_j y_j 10^{-j} \tag{6a}$$
$$y_{j+1} = y_j + \sigma_j x_j 10^{-j} \tag{6b}$$
$$z_{j+1} = z_j - \tan^{-1}(10^{-j}) \tag{6c}$$

The drawback of this decimal CORDIC method lies on the relation between any two consecutive elementary angles in the form $\tan^{-1}(10^{-j})$. Table 3 shows the values of these angles for radix 2 and radix 10, assuming 10 iterations. For convergence considerations, the sum of the subsequent angles for each iteration *j* is also included.

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 (3). However, in the case of decimal representation, each angle is approximately 10 times smaller than the one immediately preceding it, so convergence of the method cannot be directly guaranteed. As shown in the rightmost column of Table 3, the convergence condition in (2) is not satisfied, since the following property is obtained:

$$\tan^{-1}(10^{-j}) > \sum_{k=j+1}^{n} \tan^{-1}(10^{-k}), \, j \ge 0 \tag{7}$$

Stage	$\tan^{-1}(2^{-j})$	$\sum_{k>j}\tan^{-1}(2^{-k})$	$\tan^{-1}(10^{-j})$	$\sum_{k>j}\tan^{-1}(10^{-k})$
$j=0$	0.7853981	0.9578885	0.7853982	0.11077943
$j=1$	0.4636476	0.4942409	0.0996687	0.01111078
$j=2$	0.2449787	0.2492622	0.0099997	0.00111111
$j=3$	0.1243550	0.1249072	0.0010000	0.00011111
$j=4$	0.0624188	0.0624884	0.0001000	0.00001111
$j=5$	0.0312398	0.0312486	0.0000100	0.00000111
$j=6$	0.0156237	0.0156248	0.0000010	0.00000011
$j=7$	0.0078123	0.0078125	0.0000001	0.00000001
$j=8$	0.0039062	0.0039063	1E-08	1E-09
$j=9$	0.0019531	0.0019531	1E-09	1E-10

This fact 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, every iteration but the initial one must be repeated 9 times so as to achieve convergence [14]. In this case, the following is obtained:

$$\tan^{-1}(10^{-j}) \leq 9\sum_{k=j+1}^{n}\tan^{-1}(10^{-k}), j \geq 0$$
(8)

References [14] and [17] 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.

The results obtained with the decimal CORDIC proposed in [14] and [17] 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 derived from the use of the algorithm with BCD operands would be reduced to omit conversion between BCD and binary representation and, consequently, to avoid loss of precision.

III. THE NEW DECIMAL CORDIC METHOD

The new proposal for a decimal CORDIC method is the result of combining the performance and precision of binary CORDIC with decimal representation in BCD format. That is, the decomposition of the angle to be rotated in elementary angles that fulfil condition (3) will be used and, as a result, convergence is guaranteed without the need of repeating any stage. However, the product by 2^{-j} that appears in (1a) and (1b) can no longer be implemented as a simple shift of bits but rather, when using decimal BCD codification, some kind of correcting operation must be performed instead.

Dividing a number by 2 can be conceived as multiplying the same number by 0.5. Thus, the division can be carried out by shifting 4 times to the right the different BCD digits and then multiplying the result by 5.

In [14] multiplying a BCD number by 5 is described by means of a method which requires, for each pair of BCD digits, three inverters, ten 2-input gates and two 3-input gates, and the signal must pass through 5 gate levels until the final result is given [14]. These components must be added to the characteristics of the circuit for shifting 4 bits.

A simpler way of performing the division by 2 of a BCD operand than the method described above has been developed. Taking into account that $b^{+}_i[j]$ represents the bit in the position j within the BCD digit b_i after shifting and correction, the method can be summarized as follows:

1. If $b_{i+1}[0]$ is 0, the new digit b^{+}_i is constructed by selecting the 3 most significant bits of b_i and shifting them to the right, and setting a 0 in the most significant position.

2. If $b_{i+1}[0]$ is 1, the new digit b^{+}_i is calculated by using the following logical equations:

$$b^{+}_i[0] = \overline{b_i[1]}$$
(9a)

$$b^{+}_i[1] = b_i[2] \oplus b_i[1]$$
(9b)

$$b^{+}_i[2] = \overline{b^{+}_i[3]}$$
(9c)

$$b^{+}_i[3] = b_i[3] + b_i[2]\,b_i[1]$$
(9d)

This proposal permits the number of gates to be reduced to 3 AND gates, 2 OR gates, plus 3 inverters, and the number of gate levels the signal passes through is 3. At first glance, the proposed algorithm behaves better than the one described in [14]. The method proposed also requires a multiplexer circuit which, depending on the value of the least significant bit of the BCD digit, selects the bits of the shifted digit or the same bits after the correcting addition. A 2-to-1 multiplexer can be implemented by means of 2 AND gates, 1 OR gate, and 1 inverter. The required multiplexer causes the total number of gates to be 24 distributed over 5 levels. The configuration for this proposal can be seen in Fig. 1.

If the multiplexer is not a subsequent stage, but part of the circuit logic, the following equations are obtained:

$$b^{+}_i[0] = b_i[1]\overline{b_{i+1}[0]} + \overline{b_i}[1]\,b_{i+1}[0]$$
(10a)

$$b^{+}_i[1] = b_i[2]\overline{b_i}[1] + \overline{b_i}[2]b_{i+1}[0]b_i[1] + \\ + b_i[2]\overline{b_{i+1}[0]}$$
(10b)

$$b^{+}_i[2] = b_i[3]\overline{b_{i+1}[0]} + \overline{b_i}[3]b_{i+1}[0](\overline{b_i}[1] + \overline{b_i}[2])$$
(10c)

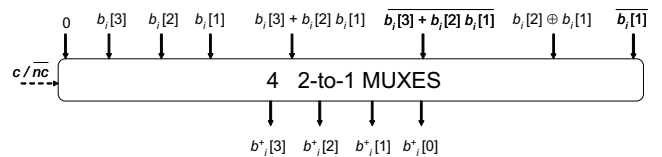$$b^{+}_i[3] = b_i[3]\,b_{i+1}[0] + b_i[2]\,b_i[1]\,b_{i+1}[0]$$
(10d)

Fig. 1. Multiplexers for selecting between b_i and b^{+}_i. The control signal c / $\overline{nc}$ (correction / no correction) comes from bit $b_{i+1}[0]$.

The total number of gates is 12 AND gates, 6 OR gates, and 4 inverters. The number of gate levels the signal passes through is 5, as in the previous case.

IV. PROPOSAL FOR A HYBRID CORDIC ARCHITECTURE

There are various alternatives for adding two BCD operands [14]. The operation 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. Moreover, the sum of two BCD digits must be corrected if it is greater than 9. The correction consists in adding the value 6 to the intermediate result.

Decimal addition/subtraction can be efficiently performed if BCD X3 representation is used. To do this, 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 [14]. Despite the advantages of using BCD X3 format for addition, the most frequent operation appeared in the proposed CORDIC method is division by 2. As in the case of natural BCD, this division can be performed as a multiplication by 5 and a 4-bit shift. The product of a BCD X3 digit by 5 can be carried out by means of a circuit with six levels of gates. The number of gates is far greater than in the different shift circuits described for natural BCD: 19 AND gates and 11 OR gates, plus 5 inverters.

As a conclusion, it must be remarked that the circuit to obtain the product by 5 in BCD X3 is significantly more complex than that for natural BCD. In addition, this would require an initial conversion from BCD to BCD X3 (10 gates distributed over 3 levels) and a final conversion in the opposite direction. Therefore, the use of natural BCD is proposed since the architectures of the adder/subtracter are not quite complex. In Fig. 2 a joint architecture for BCD addition/subtraction based on the designs in [18] is shown.

The complete architecture for each of the iterations of the proposed CORDIC method is shown in Fig 3. The BCD adder is the same as the one represented in Fig. 2. On the other hand, the 2^{j-1}-divider in Fig 4 is the shifter of the number of bits indicated by j-1, and is constructed based on the 1-bit shifter shown in Fig 1. In the first iteration ($j = 0$), the control logic selects the bits $X_i[3]$, $X_i[2]$ $X_i[1]$, and $X_{i+1}[0]$, whereas in the following iterations the bits selected are $b^+_i[3]$, $b^+_i[2]$, $b^+_i[1]$, and $b^+_{i+1}[0]$.

V. EXPERIMENTATION

A. Experiments on Precision

Different tests were carried out so as to make a complete comparison with regard to precision between decimal CORDIC and the hybrid CORDIC proposed in this work. Binary CORDIC was also included for comparison in some tests.
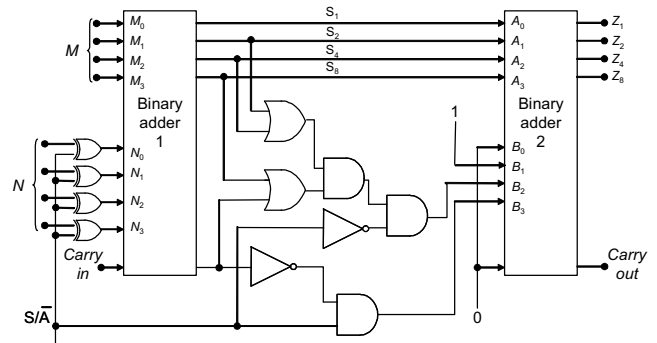

Fig. 2. BCD adder/subtracter; $Z = M \pm N$.
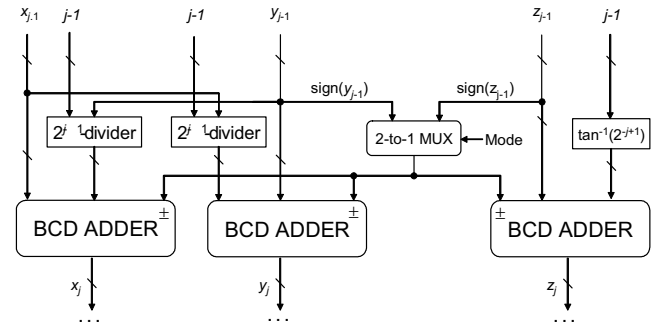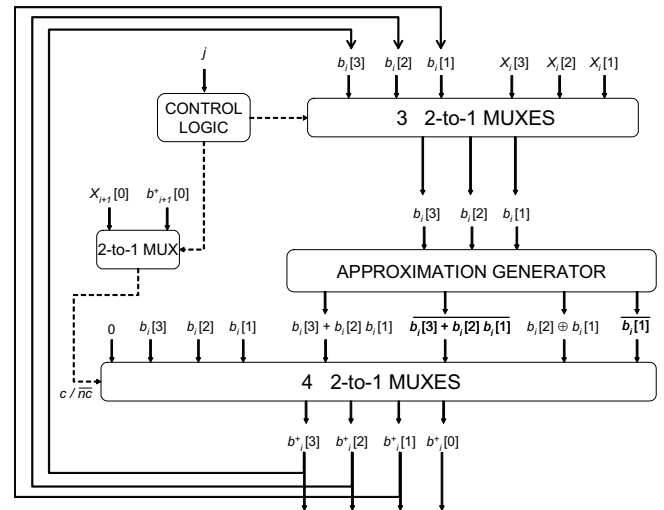

Fig. 3. The architecture for a CORDIC iteration.

Fig. 4. The 2^{j-1}-divider or iterative shifter.

Values within the range [0, 0.999] were chosen for the (x, y) coordinates. Data were represented in natural BCD with 8 fractional digits.

In the first test, the magnitudes of 30 vectors were calculated, each of them forming an angle $\theta = 0.87266463$ radians ($\approx 50°$) towards the x-axis, inside a circumference with a radius of 0.999. Fig. 5 shows that the maximum error decreases as the vector magnitude increases. Once again, the results for the maximum error are always lower for the method proposed in this work.
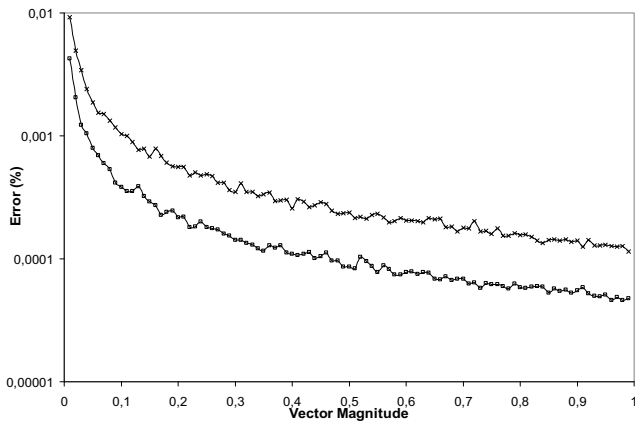
Fig. 5. Maximum relative error on calculating the magnitude of vectors within the range [0, 0.999], according to vector magnitude; logarithmic scale (× = decimal CORDIC; □ = hybrid CORDIC).

The second experiment permits the error in the calculation of the magnitudes of 30 random vectors, as described in the previous paragraph, to be determined while varying the number of digits from the input data. As shown in Fig. 6, the maximum error is never higher for the hybrid method, and with low precision the error is much lower than for decimal CORDIC.

The next experiment was aimed at comparing the number of iterations required in each method so as to achieve suitable precision. The magnitudes of 30 random vectors with an angle $\theta = 0.87266463$ radians were calculated while varying the number of iterations. Fig. 7 shows that the error decreases much faster in the hybrid CORDIC. For this method, error reaches stability in about 10 iterations whereas for decimal CORDIC this number is much greater. The maximum error for any number of iterations is always lower for the proposed method.
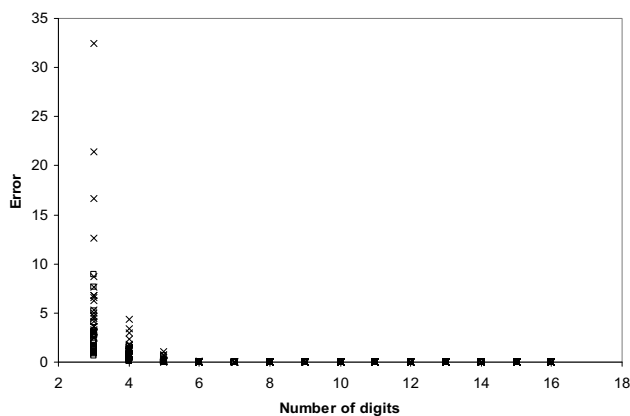

Fig. 6. Relative error when calculating the magnitudes of vectors in [0, 0.999], according to the number of digits (× = decimal CORDIC; □ = hybrid CORDIC).
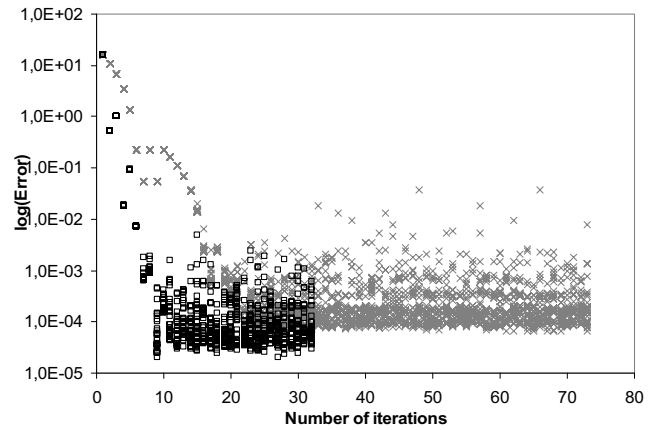

Fig.7. Relative error distribution when calculating the magnitude of vectors within the range [0, 0.999], according to the number of iterations (× = decimal CORDIC; □ = hybrid CORDIC). Logarithmic scale.

B. Experiments on Latency

The proposed architecture was implemented using the Xilinx ISE 7.1i tool. The Spartan3E XC3S100E FPGA was chosen. The architectures for binary and decimal CORDIC proposed in [14] and [17] were also implemented. In all cases, a complete stage of the method was implemented, with the type of adder and shifter being varied according to each method. The global method was implemented on an unfolded architecture. In the case of the hybrid CORDIC, the architecture was constructed by means of a 24-stage pipeline, each one according to the diagram shown in Fig. 3. Non-redundant arithmetic was used and the final scaling factor compensation stage was omitted. The delays obtained are shown in Table 4 for comparison. It shows that the proposed hybrid CORDIC has a delay per stage 4 times lower than that of decimal CORDIC.

Since each set of input data has to pass through a certain number of stages until it is fully processed, Fig. 8 shows the time per datum, in ns, obtained as the number of data supplied increases. The results show that the hybrid CORDIC architecture offers better performance than the decimal architecture, both in delay of only one stage and in latency derived from the data processing throughout the overall set of stages. In addition, these indices are not far from those obtained for the binary CORDIC architecture.

TABLE IV
SINGLE STAGE DELAY FOR DIFFERENT CORDIC ARCHITECTURES
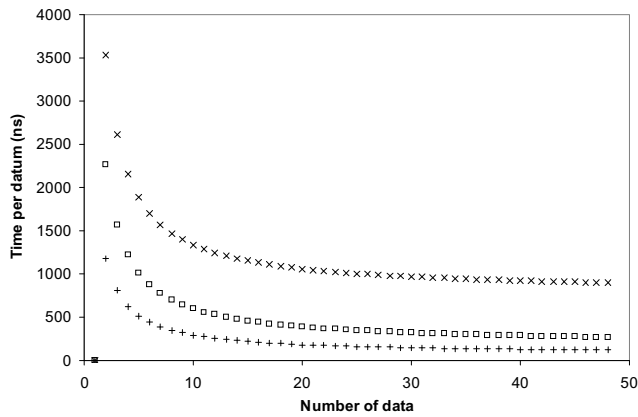
Binary	Decimal	Hybrid
71.51	784.76	181.12

Fig. 8. Time per datum for each CORDIC architecture according to the data processed, in ns ($\times$ = decimal CORDIC; + = binary CORDIC; $\square$ = hybrid CORDIC).

VI. 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 logical circuits that integrate these new embedded architectures. As shown before, it seems that there is a growing trend towards developing new systems that integrate decimal arithmetic, which is required in many practical research areas.

In this work we have described a new method to implement decimal operations using the CORDIC algorithm; this method allows using decimal input data in BCD format combined with elementary angles of the type $\tan^{-1}(2^{-j})$, which improve the convergence time of the original decimal algorithm. The experiments show that our method converges in fewer iterations than the original one and that it has a much lower delay than that for former decimal CORDIC proposals. As a future work, it would very interesting to implement in hardware the proposed hybrid architecture for a CORDIC-based ALU. It would be also important to consider a pipelined version of the new method. This and other parallel versions of our method are currently being implemented and we hope that the results will even improve the ones obtained in this work.

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: Algorism for Computers," *Proc. 16th IEEE Symp. Computer Arithmetic*, 2003.
[5] R. Perkins, "EASIAC, A Pseudo-Computer," *Communications of the ACM*, vol. 3 #2, pp. 65-72, April 1956.
[6] M. Bataille, "The Gamma 60: The computer that was ahead of its time," *Honeywell Computer Journal*, vol. 15 #3, pp. 99-105, 1971.
[7] F. B. Jones and A. Wymore, "Floating Point Feature of the IBM Type 1620," *IBM Technical Disclosure Bulletin*, 05-62, pp. 43-46, May 1962.
[8] *Burroughs B5500 Information Processing Systems Reference Manual*, Burroughs Corporation, Detroit, MI, 1964.
[9] S. Mazor, "Fairchild decimal arithmetic unit," personal communication, July-Sept., 2002.
[10] M. S. Cohen, T. E: Hull, and V. C. Hamacher, "CACAD: A Controlled-Precision Decimal Arithmetic Unit," *IEEE Trans. Comput.*, vol. 32 #4, pp. 370-377, April 1983.
[11] *IEEE 854-1987 – IEEE Standard for Radix-Independent Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., New York, 1987.
[12] *Draft IEEE Standard for Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., New York, 2005.
[13] J. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330-334, 1959.
[14] H. Schmid, *Decimal Computation*. NY: John Wiley & Sons, 1974.
[15] J. C. Kropa, "Calculator Algorithms," *Mathematics Magazine*, vol. 51, no. 2, pp. 106-109, March 1978.
[16] J. S. Walther, "A unified algorithm for elementary functions," *Proc. AFIPS Spring Joint Computer Conf.*, 1971, pp. 379-385.
[17] H. Schmid and A. Bogacki, "Use decimal CORDIC for generation of many transcendental functions," *EDN*, pp. 64-73, Feb. 1973.
[18] A. Despain, "Fourier Transform Computers Using CORDIC Iterations," *IEEE Trans. Comput.*, vol. C-23, no. 10, pp. 993-1001, Oct. 1974.
[19] 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.
[20] G. Haviland and A. Tuszynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. Comput.*, vol. C-29, no. 2, pp. 68-79, Feb. 1990.
[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, pp. 85-100, 1991.