Advance Access publication on 11 March 2011
doi:10.1093/comjnl/bxr012

Decimal CORDIC Rotation based on Selection by Rounding: Algorithm and Architecture

AMIR KAIVANI[1,*] AND GHASSEM JABERIPUR[1,2]

[1]*Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran*
[2]*School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*
[*]*Corresponding author: a_kaivani@sbu.ac.ir*

Hardware implementation of decimal floating-point arithmetic is a topic of great interest among the researchers in computer arithmetic and also the digital processor industry. Software packages for decimal arithmetic are actually being challenged by decimal hardware units. This spreading trend seems to include hardware implementation of elementary functions. The (Coordinate Rotation Digital Computer) CORDIC algorithm, due to its simplicity, is one of the most efficient methods for computing elementary functions. In this work, we develop a decimal CORDIC scheme with almost half number of equally long cycles with respect to the best previous design. This is achieved via retiming of the conventional CORDIC architecture and selection of the microrotation factors by rounding. However, the proposed design does not lead to a predetermined constant scaling factor. The solution that we use is to iteratively compute the logarithm of the scaling factor followed by a decimal exponentiation. The same CORDIC hardware is reused for performing the latter. The proposed CORDIC method requires $2n + 3$ cycles for n-digit decimal operands vs. 4n cycles of the previous methods. Evaluations with 16-digit operands based on logical effort analysis conclude that the proposed architecture shows 82% speed advantage, at the cost of 60% more area and 2.5 KB more ROM.

Keywords: decimal computer arithmetic; decimal CORDIC; selection by rounding; retiming

Received 8 September 2010; revised 11 December 2010
Handling editor: Mark Josephs

1. INTRODUCTION

Mirroring the manual decimal calculations is required by many commercial, financial and internet-based applications of radix-10 computer arithmetic [1]. Unfortunately, the relatively fast binary floating-point arithmetic, with due forward and reverse conversions, cannot fully satisfy such requirements for exact representation of decimal values [2]. Consequently, the industry's growing preference toward the implementation of radix-10 computer arithmetic is probably the main reason for inclusion of the decimal arithmetic specifications (e.g. decimal number representation, operations, rounding and exceptions) in the IEEE 754-2008 standard [3] for floating-point arithmetic.

The ever-increasing computing power provided by the advanced VLSI technologies has motivated researchers in the field of computer arithmetic and also the digital processor industry to focus on the hardware implementation of decimal arithmetic operations [4, 5].

It is a longstanding practice that commercial digital processors are augmented with decimal adder/subtractor circuitry [6]. The advances in hardware industry have led the manufacturers to incorporate fully hardware radix-10 sequential multipliers and dividers in decimal arithmetic units [4]. Furthermore, the state-of-the-art research on decimal hardware is mainly devoted to fast division [7], parallel multiplication [8, 9], square root [10] and CORDIC (COordinate Rotation DIgital Computer) algorithms [11, 12]. Following this trend, it can be expected that the industry will, sooner or later, take advantage of such research findings for realizing more powerful decimal hardware. This can include the replacement of commercialized software packages for decimal transcendental functions [13] with the relevant hardware circuitry.

Fast hardware realization of polynomial approximation for transcendental functions is often relying on fast parallel multipliers [14]. However, given the high area and power consumption of parallel decimal multipliers [8], the relatively low area/power CORDIC algorithm [15] seems to be an attractive alternative for implementing decimal transcendental functions [11, 12, 16].

The variants of CORDIC algorithm describe the displacement of a vector via micromovements on circular, hyperbolic or linear coordinates in rotation or vectoring modes. For example, the original CORDIC [17] operates on circular coordinates via microrotations, where the determination of the amount of successive rotation angles, for high-radix implementation [18, 19], appears to be the most difficult task in every iteration. The solutions that have so-far been proposed for radix-10 CORDIC (e.g. [11, 16]) are more complex, as expected, than the power-of-two-radix cases [20]. For example, rotation angles that are easily determined for the ith iteration as $\tan^{-1}(2^{-i})$ (instead of micro-angles with the power of 10 tangents) lead to faster, but more numerous, iterations [11].

In this work, we handle rotation angles that are multiples of $\tan^{-1}(10^{-i})$ each within a single iteration. However, we show that the selection of these multiples can be considerably simplified, with due speed-up, via adapting the well-known technique of *selection by rounding* for radix-10. This technique was first introduced in [21] with the intention of rarefying the complexity of the selection function in high-radix binary algorithms such as division [22], square-root [23] and CORDIC [18].

One drawback of the proposed method, in this work, is that the scaling factor of the CORDIC algorithm [24] cannot be determined in advance, while the previous decimal CORDIC designs are based on constant predetermined scaling factor (e.g. [11, 16]). However, to remedy this drawback, we adapt for radix- 10—a known technique [20] for computing the scaling factor in parallel with the main CORDIC computation.

The rest of this paper is organized as follows. Section 2 presents a background on the decimal CORDIC algorithm. The proposed solution, based on the *selection by rounding* technique, is presented in Section 3. Next, in Section 4, a retiming technique for reducing the latency of the angle-selection function is introduced. The applied algorithm for computing and compensating the scaling factor is discussed in Section 5. The architecture and timing of the proposed CORDIC scheme are presented in Section 6. Evaluation and comparison with the relevant previous works is taken up in Section 7, and finally, Section 8 draws our conclusions.

2. DECIMAL CORDIC ALGORITHM

In this work, we intend to focus on the decimal CORDIC in rotation mode for circular coordinates, which is briefly described here. Given a vector $v = (x_{\text{in}}, y_{\text{in}})$ and an angle Θ, the CORDIC algorithm relocates v toward $u = (x_{\text{out}}, y_{\text{out}})$ via n microrotations. The size of the ith microrotation is equal to an elementary tiny angle $\theta_i = \tan^{-1}(10^{-i}\sigma_i)$, such that $\Theta = \sum_{i=1}^{n} \theta_i$, where σ_i is the micro rotation factor. Equation set (1) (repeated for n iterations), whose derivation can be found in the relevant text books (e.g. [24, 25]), describes the ith iteration that simulates the ith microrotation, where $x[1] = x_{\text{in}}$, $y[1] = y_{\text{in}}$, $z[1] = \Theta$, $x_{\text{out}} = x[n+1]/K$, $y_{\text{out}} = y[n+1]/K$, σ_i is usually selected based on an estimate on the value of $z[i]$, and $K = \prod_{i=1}^{n}(1 + 10^{-2i}\sigma_i^2)^{0.5}$ is the scaling factor.

$$\begin{aligned} x[i+1] &= x[i] - 10^{-i}\sigma_i \times y[i], \\ y[i+1] &= y[i] + 10^{-i}\sigma_i \times x[i], \\ z[i+1] &= z[i] - \tan^{-1}(10^{-i}\sigma_i). \end{aligned} \tag{1}$$

The condition for the latter recurrence to converge is given by Equation (2), adopted from [24] for radix-10, where the required precision is specified by n (i.e. $ulp = 10^{-n}$) and α determines the range of microrotation factor $\sigma_i \in [-\alpha, \alpha]$. The negative range is necessary in order to correct previous possible overestimates. Moreover, based on the results in [20], it is required that $\alpha \geq 5$.

$$|z[i]| \leq \sum_{j=i}^{n} \tan^{-1}(\alpha \times 10^{-j}) + \tan^{-1}(10^{-n}). \tag{2}$$

After n iterations, on Equation set (1), we arrive at $(x[n+1] = Kx_{\text{out}}, y[n+1] = Ky_{\text{out}})$. In order to compensate for the scaling factor K, two common methods have been used in binary CORDIC such as incremental multiplications throughout the iterations [26] or a final multiplication by a predetermined constant K^{-1} [27].

The estimation on the value of $z[i]$, to decide on σ_i, is based on a few of its most significant non-zero digits [24]. However, more and more leading zero digits show up in the most significant positions as $z[i]$ approaches smaller values throughout the successive iterations. This calls for dynamically locating the leading non-zero digits and thus complicating the required circuitry for selection of σ_i. In order to avoid this complication, the z component is usually scaled up as in Equation (3). Accordingly therefore, the angle recurrence of the CORDIC iteration (i.e. the third in Equation set (1)) is modified as in Equation (4), where $A[\sigma_i] = 10^i \tan^{-1}(10^{-i}\sigma_i)$. Consequently, we rewrite the convergence condition (i.e. Equation (2)) as in Equation (5).

$$w[i] = 10^i \times z[i], \tag{3}$$

$$w[i+1] = 10 \times (w[i] - A[\sigma_i]), \tag{4}$$

$$|w[i]| \leq 10^i \times \left[\sum_{j=i}^{n} \tan^{-1}(\alpha \times 10^{-j}) + \tan^{-1}(10^{-n})\right]. \tag{5}$$

The bulk of work in the angle recurrence (i.e. Equation (4)) is to determine $\sigma_i \in [-\alpha, \alpha]$ such that the convergence condition (i.e. Equation (5)) holds. The relevant works (i.e. those based on micro-angles with power of 10 tangents) that we have encountered are due to [16, 28]. These works, both with $\alpha = 9$, use several sub-iterations with predetermined σ_i^j such that $\sigma_i = \sum_j \sigma_i^j$. The former work uses $\sigma_i^j \in \{0, 1\}(1 \leq j \leq 9)$. However, $\sigma_i^j \in \{1, 2, 5\}(1 \leq j \leq 4)$ in the latter. For improved speed and controlled increase of area, we opt to choose $\alpha = 5$ and directly (i.e. without any sub-iteration) select $\sigma_i \in [-5, 5]$ via the well-known method of selection by rounding [22].

3. SELECTION OF THE MICROROTATION FACTOR BY ROUNDING

The straightforward method for selection of σ_i is to compare $w[i]$ with 10 precomputed constants. There is a similar practice in decimal division algorithms, but it suffers from high area cost [7]. There is, however, another less complex method known as selection by rounding that is used for radix-2^h ($h > 1$) division [22] and CORDIC algorithm [18, 19]. We adapt this method for radix-10 and select σ_i as $w[i]$ rounded to the nearest integer ($\sigma_i = \text{round}(w[i])$).

The value of $A[\sigma_i] = 10^i \tan^{-1}(10^{-i}\sigma_i)$, needed for the angle recurrence ruled by Equation (4), is best obtained by looking it up in an appropriate ROM. It is not difficult to show that the smaller the digit set of σ_i the smaller the ROM size. This will be discussed in more detail, along with other relevant influential factors, in Section 6. Therefore, given that $\alpha \geq 5$ [20], we tend to decide on the digit set $[-5, 5]$ for σ_i, which is a valid choice if and only if $|w[i+1]| < \min(R_{i+1}, 5.5)$, where $R_{i+1} = 10^{i+1} \times [\sum_{j=i+1}^{n} \tan^{-1}(5 \times 10^{-j}) + \tan^{-1}(10^{-n})]$, for $i \geq 1$.

The reason is that $|w[i+1]| < R_{i+1}$ due to Equation (5) for $\alpha = 5$ and $|w[i+1]| < 5.5$ due to the selection by rounding (i.e. $\sigma_{i+1} = \text{round}(w[i+1])$). Since $R_{i+1} > 5.5$ for $i \geq 1$, as we prove in the Appendix 1, the convergence condition is reduced to Equation (6).

$$|w[i+1]| = |10 \times (w[i] - A[\sigma_i])| < 5.5. \tag{6}$$

We now examine whether Equation (6) holds for $i \geq 1$. It is obviously the case that $-0.5 \leq w[i] - \sigma_i < 0.5$, due to selection of σ_i by rounding $w[i]$. Applying the latter bounds into Equation (4) leads to Equation (7). Therefore, for Equation (6) to hold, the inequalities (8) and (9) must also hold.

$$-5 + 10(\sigma_i - A[\sigma_i]) \leq w[i+1] < 5 + 10(\sigma_i - A[\sigma_i]), \tag{7}$$

$$5 + 10(\sigma_i - A[\sigma_i]) < 5.5, \tag{8}$$

$$-5.5 \leq -5 + 10(\sigma_i - A[\sigma_i]). \tag{9}$$

TABLE 1. Selection table for the preprocessing iteration ($i = 0$).

$w[0]$	σ_0	Max of $\lvert w[0] - A[\sigma_0]\rvert$
$\pm$ [0.0, 0.3)	± 1	0.21
$\pm$ [0.3, 0.6)	± 5	0.17
$\pm$ [0.6, 0.8)	± 10	0.22

Replacing $A[\sigma_i]$ by $10^i \tan^{-1}(10^{-i}\sigma_i)$ in Equations (8) and (9) takes us to Equation (10) to hold for $i \geq 1$.

$$-0.05 \leq \sigma_i - 10^i \tan^{-1}(10^{-i}\sigma_i) < 0.05. \tag{10}$$

The expression $\sigma - 10^i \tan^{-1}(10^{-i}\sigma_i)$ is a monotonically increasing (decreasing) function of σ_i, for $\sigma_i \geq 0$ ($\sigma_i < 0$). Therefore, if Equation (10) holds for the maximum (minimum) value of σ_i (i.e. $\sigma_i = 5(-5)$), it will be satisfied for all values of $\sigma_i \in [-5, 5]$. This implies Equation (11) for $i \geq 1$.

$$5 - 10^i \tan^{-1}(5 \times 10^{-i}) < 0.05. \tag{11}$$

The latter discussion can be summarized as $6 \Rightarrow (8\&9) \Rightarrow 10 \Rightarrow 11$. Unfortunately, however, it turns out that Equation (11) does not hold for $i = 1$, but it does for $i \geq 2$ (the proof is found in the Appendix 2). Therefore, Equation (6) does not hold for $i = 1$. This means that *selection by rounding* works fine for all iterations, except for the first one. Consequently, selection of σ_1 requires a special treatment to be discussed below.

3.1. Selecting the microrotation factor σ_1

Solving Equation (10) for $i = 1$ leads to $|\sigma_1| \leq 2$. The latter binding, that $\sigma_1 = \text{round}(w[1])$ and $w[1] = 10z[1] = 10\Theta$, take us to $|10\Theta| < 2.5 \Rightarrow |\Theta| < \pi/13$. This tight range on the input angles is certainly not desirable. A way out of this problem is via a common technique that installs a preprocessing iteration [20] that cannot be directly applied for radix-10. Therefore, in the way of adapting it, we define iteration 0 based on Equation (12), where the input angle is $w[0] = z[0] = \Theta$, and $A[\sigma_0] = \tan^{-1}(10^{-1}\sigma_0)$.

$$-2.5 \leq w[1] = 10 \times (w[0] - A[\sigma_0]) < 2.5. \tag{12}$$

To determine σ_0, for a given input $w[0](|w[0]| = |z[0]| < \pi/4)$[1], we use Table 1, which is derived based on Equation (12). This table can be translated to a simple combinational logic for selection of σ_0, as is depicted by Fig. 1, where the input (i.e. the sign and most significant fractional digit of $w[0]$) and output are represented in 10's complement BCD format.

[1]To widen the range of the input angle to $[-\pi/2, \pi/2]$, one can repeat iteration $i = 0$.

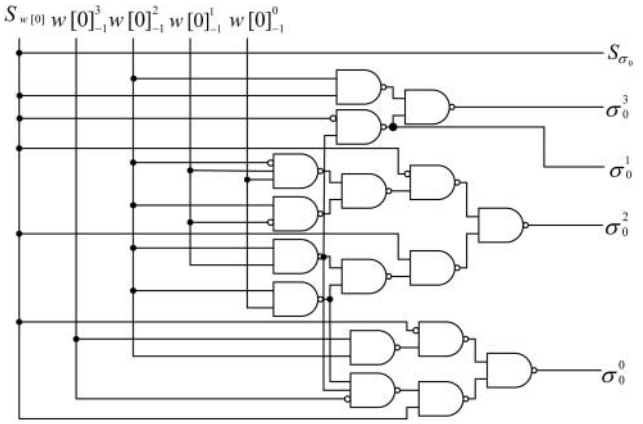

FIGURE 1. The circuit for selection of σ_0.

4. RETIMING THE SELECTION OF MICROROTATION FACTOR

In order to arrive at the microrotation factor $\sigma_i = \text{round}(w[i])$, the rounding operation on $w[i]$ is normally done after execution of Equation (13), which is the same as Equation (4), but for the $(i-1)$th iteration.

$$w[i] = 10 \times (w[i-1] - A[\sigma_{i-1}]). \tag{13}$$

One way to bring forward the computation of σ_i is to perform the rounding process in parallel with the normal execution of Equation (13), as is described below. This technique (known as retiming) is used, for instance, in division algorithms to do quotient digit selection in parallel with partial remainder computation [7].

Figure 2 depicts a dot representation of the constituent terms of $w[i]$. This is indeed a double-BCD number as a redundant representation of $w[i]$. The idea is to use the minimum number t of the most significant double bits of this representation to compute $\sigma_i = \text{round}(w[i])$ within some acceptable lower and upper error bounds $e_l(t)$ and $e_u(t)$, respectively, such that

$$e_l(t) \le w[i] - \sigma_i < e_u(t). \tag{14}$$

The convergence condition for the CORDIC algorithm in the case of selection of $\sigma_i \in [-5, 5]$ by rounding is represented by Equation (15) as a reproduction of Equation (6). By combining Equations (14) and (15), after some manipulations similar to Equations (7)–(11), we conclude that Equations (16) and (17) should hold in order to satisfy Equation (15), but for $i \ge 2$. The case of $i = 1$ will be dealt with later.

$$A[\sigma_i] - 0.55 < w[i] < A[\sigma_i] + 0.55, \tag{15}$$

$$e_l(t) \ge -0.55; \tag{16}$$

$$e_u(t) < 0.54. \tag{17}$$

We now explore possible values for t that satisfy Equations (16) and (17). The maximum round-up error is -0.5, which occurs when the most significant fractional double-BCD digit is equal to 5 and the rest of the digits are 0. Therefore, since $e_l(t) = -0.5 > -0.55$, Equation (16) is satisfied for all values of t. On the other hand, the maximum round-off error occurs when the collective value of the fractional part of the double-BCD representation of $w[i]$ is closest to 0.5, from below. This calls for the most significant double-BCD digit to be equal to 4. A rounding decision based only on this double-BCD digit (implying $t = 4$) can easily violate Equation (17).

For $t > 4$, let $v(t)$ denote the maximum collective value of the t most significant fractional double-bits and $v'(t)$ denote the maximum collective value of the rest of the fractional part. It is easy to see that $v(5) = v(6) = v(7) = 0.48 < 0.5$, and thus rounding off could be correct. However, $v'(5) = (0.1 - 2\ ulp)$ leads to $e_u(5) = v(5) + v'(5) = (0.58 - 2\ ulp)$, which violates Equation (17). Similar analysis for $t = 6$ leads to $e_u(6) = (0.54 - 2\ ulp)$, which does not violate Equation (17). However, recalling the method of Section 3.1 for selection of σ_1, the very small 2 *ulp* difference with the upper bound leads to $\sigma_1 = 0$ and thus very small range for the input angles. To avoid this undesirable situation, we try $t = 7$, where $e_u(7) = (0.52 - 2\ ulp)$. In this case, Equation (15) holds for $i = 1$ only if $|\sigma_1| \le 2$. Consequently, selection of σ_1 can be done exactly as in Section 3.1.

5. COMPUTATION AND COMPENSATION OF THE SCALING FACTOR

Recalling the formula of the scaling factor from Section 2 (i.e. $\prod_{i\ge1}^{n}(1 + 10^{-2i}\sigma_i^2)^{0.5}$), note that for the proposed method of Section 3, this formula should be augmented with an extra factor reflecting the effect of the preprocessing iteration for σ_0. This is described by Equation (18).

$$K = (1 + 10^{-2}\sigma_0^2)^{0.5} \times \prod_{i\ge1}^{n}(1 + 10^{-2i}\sigma_i^2)^{0.5}. \tag{18}$$

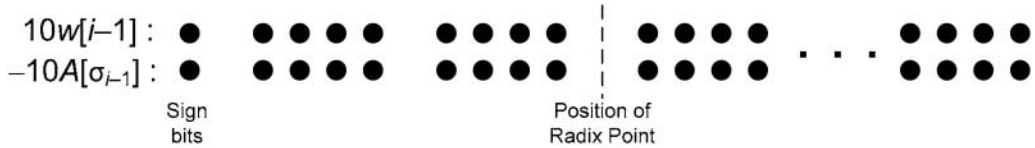

FIGURE 2. Double-BCD representation of $w[i]$.

Since the microrotation factors σ_i are not predetermined, the compensating factor K^{-1} is not constant. Therefore, it should be computed anew each time the CORDIC algorithm is executed. To avoid n cascaded multiplications embedded in Equation (18), the method used for high power-of-two radix CORDIC algorithms in [20] can be adapted for radix-10. Therefore, we use Equation (19) to first compute $\ln(K^{-1})$. This will prevent the scaling factor computation from lagging behind the main CORDIC process.

$$\begin{aligned} g &= \ln(K^{-1}) \\ &= -0.5\left[\ln(1+10^{-2}\sigma_0^2) + \sum_{i\geq 1}^{n} \ln(1+10^{-2i}\sigma_i^2)\right]. \end{aligned} \quad (19)$$

The ln terms embedded in Equation (19) are obtained via a small look-up table (LUT) (see Section 6 for the exact size). Once g is computed, the final coordinates $(x_{\text{out}}, y_{\text{out}})$ are obtained as in Equation set (20).

$$x_{\text{out}} = x[n+1] \times e^g, \quad y_{\text{out}} = y[n+1] \times e^g. \quad (20)$$

To compute e^g, we use Equation set (21) (for $1 \leq i \leq n$) reproduced from [29], where $B[d_i] = 10^i \ln(1+10^{-i}d_i)$ and d_i is selected exactly in the same way as σ_i was in Section 3. This equation set has exactly the same operator structure as in Equation set (1) (as modified by Equation (4)). Therefore, the e^g computation can be carried out via the proposed CORDIC hardware of Section 3, initialized as $X[1] = x[n+1]$, $Y[1] = y[n+1]$, $W[1] = 10\,g$. Note that we have used capital letters X, Y and W in Equation set (21) in place of lowercase letters x, y and w in Equation set (1) and Equation (4).

$$\begin{aligned} X[i+1] &= X[i] + 10^{-i}d_i X[i], \\ Y[i+1] &= Y[i] + 10^{-i}d_i Y[i], \\ W[i+1] &= 10 \times (W[i] - B[d_i]). \end{aligned} \quad (21)$$

Given that $d_i \in [-5, 5]$, the convergence condition is described by Equation (22) (for $i \geq 1$) [14], where $S_{i+1} = 10^{i+1} \times \left[\sum_{j=i+1}^{n} \ln(1+5\times 10^{-j}) + \ln(1+10^{-n})\right]$.

$$|W[i+1]| \leq S_{i+1}. \quad (22)$$

Therefore, assuming selection by rounding and redundant representation for $W[i+1]$ (like that of $w[i+1]$, discussed in Section 4), the following should be satisfied, where the rounding error bounds are defined similar to those in Section 4 as $\delta_l(t) \leq W[i] - d_i < \delta_u(t)$ such that $\delta_l(t) = -0.5$ and $\delta_u(t) = v(t) + v'(t)$.

$$10\delta_u(t) + 10(d_i - B[d_i]) < \min(S_{i+1}, 5.5), \quad (23)$$

$$-\min(S_{i+1}, 5.5) \leq -5 + 10(d_i - B[d_i]). \quad (24)$$

We show in the Appendix 3 that $S_{i+1} > 5.5$, for $i \geq 2$. Therefore, Equations (23) and (24) can be reduced to Equations (25) and (26), for $i \geq 2$, and to (27) and (28), for $i = 1$, respectively.

$$10\delta_u(t) + 10(d_i - B[d_i]) < 5.5, \quad (25)$$

$$-5.5 \leq -5 + 10(d_i - B[d_i]), \quad (26)$$

$$10\delta_u(t) + 10(d_1 - B[d_1]) < S_2 \approx 5.43, \quad (27)$$

$$-5.43 \approx -S_2 \leq -5 + 10(d_1 - B[d_1]). \quad (28)$$

The appropriate value for t is again 7 via the same analysis that led to the same value for t, in Section 4. Therefore, given that $B[d_i] = 10^i \ln(1+10^{-i}d_i)$, Equations (25) and (26) ((27) and (28)) can be further reduced to Equation 29 (30). However, given that $5 - 10\ln(1.5) = 0.945$, Equation (30) never holds. This means that the convergence condition (Equation (22)) is not satisfied for $i = 1$. Furthermore, we show in Appendix 4 that Equation (29) does not hold for $i = 2$, while it does for $i > 2$. Therefore, special treatment is required for selection of d_1 and d_2.

$$5 - 10^i \ln(1+5\times 10^{-i}) < 0.03, \quad (29)$$

$$5 - 10\ln(1.5) < 0.023. \quad (30)$$

5.1. Selecting d_2 and d_1

Solving Equation (25) for $i = 2$ leads to $|d_2| \leq 2$. For correct selection of d_2, recalling that $d_2 = \text{round}(W[2])$, we only need to enforce $-2.5 \leq W[2] < 2.5$. For the latter to hold, using Equation set (21) to replace for $W[2]$ results in the Equation (31) to hold, which is similar to Equation (12) for σ_0. Therefore, d_1 can be similarly computed via a combinational logic (Fig. 3) that is based on Table 2. This table is built similar to Table 1, but with only negative entries. This is understood by recalling Equation (19) and that simple manipulations lead to $-4.594 < W[1] = 10\,g \leq 0$.

$$-0.25 \leq W[1] - B[d_1] < 0.25. \quad (31)$$

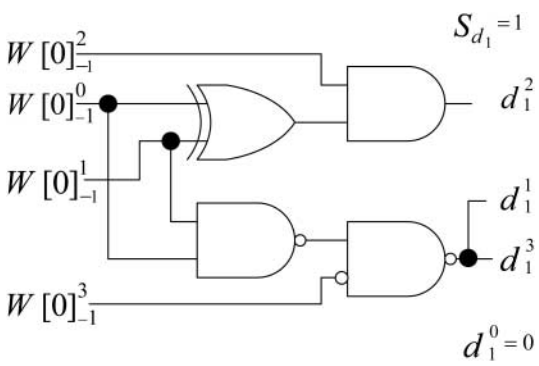

FIGURE 3. The circuit for selection of d_1.

TABLE 2. Selection table for the iteration ($i = 1$) of the exponential function.

$W[1]$	d_1	Max of $\lvert W[1] - B[d_1]\rvert$
$(-5, -3)$	-4	<0.22
$[-3, 0)$	-2	<0.23

6. ARCHITECTURE AND TIMING

The following three data paths are required for the implementation of the proposed CORDIC algorithm, whose timing is displayed in Table 3.

(i) The w/W data path, shared for the w recurrence (i.e. CORDIC rotation based on Equation (4) and for the W recurrence (i.e. third one in Equation set (21)) in the compensation process for the scaling factor.
(ii) The x/X and y/Y data path, for the x and y recurrences in the CORDIC rotation and for the X and Y recurrences in the compensation process for the scaling factor. The x_s/X_s and y_s/Y_s variables represent the shifted terms. The multiply and accumulate operations for the ith iteration take place in parallel with the shift operations for the $(i + 1)$th iteration. This retiming will be shown to be quite influential in reducing the cycle time.
(iii) The g data path to implement Equation (19).

The hardware implementation of the proposed architecture, for 16-digit operands, is shown in Fig. 4, where the round block, LUTs and decimal multiplier-accumulators (MACs) are described in separate subsections below. The 16-digit decimal CPAs are designed based on the decimal adder/subtractor of [30].

6.1. Round block

The round block in the leftmost partition of Fig. 4 computes $m = \text{round}(T + Q)$ without actually performing the addition operation therein, where T is either $A[\sigma_i]$ or $B[d_i]$, Q is either $w[i]$ or $W[i]$ and m represents σ_{i+1} or d_{i+1}. All the operands and the result are represented in 10's complement BCD format. The rounding mode is round to nearest such that $m = \lfloor L \rfloor$, where $L = T + Q + 0.5$ as is illustrated in Fig. 5, with detailed explanations for each stage to follow.

We recall Equation (6) to reiterate the fact that $-5.5 \leq L < 5.5$ and $m \in [-5.5]$. Therefore, $S_L L_1 \in \{00, \bar{1}9\}$, such that the arithmetic value of $S_L L_1$ is equal to that of S_L. This sign compression leads to $m = S_L L_0$. To compute m, we first extract, the decimal carry into position 0, out of the fractional digits of T and Q (i.e. $C_0 = \lfloor .T_{-1}T_{-2} + .Q_{-1}Q_{-2} + .5 \rfloor \in \{0, 1, 2\}$). We encode C_0 as two equally weighted bits c_0 (defined by Equation (32)) and c_0' (defined by Equation (33)) such that

TABLE 3. Timing of the proposed algorithm.

	w/W	x/X and y/Y	g
Cycle	CORDIC rotation		Computation of $\ln(K^{-1})$
Initialization	$w[0] = \Theta$; Set σ_0; $A[\sigma_0] = \tan^{-1}(10^{-1}\sigma_0)$	$x[0] = x_{\text{in}}$; $y[0] = y_{\text{in}}$ $x_s[0] = 10^{-1}y_{\text{in}}$; $y_s[0] = 10^{-1}x_{\text{in}}$	$g[0] = 0$
$i = 0$	$w[1] = 10(w[0] - A[\sigma_0])$; Select σ_1; $A[\sigma_1] = 10\tan^{-1}(10^{-1}\sigma_1)$;	$x[1] = x[0] - \sigma_0 x_s[0]$; $y[1] = y[0] + \sigma_0 y_s[0]$; $x_s[1] = 10^{-1}y[0]$; $y_s[1] = 10^{-1}x[0]$	$g[1] = g[0] + \ln(1 + 10^{-2}\sigma_0^2)$
$i = 1$ to n	$w[i + 1] = 10(w[i] - A[\sigma_1])$; Select σ_{i+1}; $A[\sigma_{i+1}] = 10^{i+1}\tan^{-1}(10^{-i-1}\sigma_{i+1})$	$x[i + 1] = x[i] - \sigma_i x_s[i]$; $y[i + 1] = y[i] + \sigma_i y_s[i]$ $x_s[i + 1] = 10^{-i-1}y[i]$; $y_s[i + 1] = 10^{-i-1}x[i]$	$g[i + 1] = g[i] + \ln(1 + 10^{-2i}\sigma_i^2)$
Compensation of the scaling factor			
Initialization	$W[1] = 10\,g[n + 1]$; Set d_1; $B[d_1] = 10\ln(1 + 10^{-1}d_1)$	$X[1] = x[n + 1]$; $Y[1] = y[n + 1]$; $X_s[0] = 10^{-1}x[n + 1]$; $Y_s[0] = 10^{-1}y[n + 1]$	
$i = 1$ to n	$W[i + 1] = 10(W[i] - B[d_i])$; Select d_{i+1}; $B[d_{i+1}] = 10^{i+1}\ln(1 + 10^{-i-1}d_{i+1})$	$X[i + 1] = X[i] + d_i X_s[i]$; $Y[i + 1] = Y[i] + d_i Y_s[i]$; $X_s[i + 1] = 10^{-i-1}X[i]$; $Y_s[i + 1] = 10^{-i-1}Y[i]$	

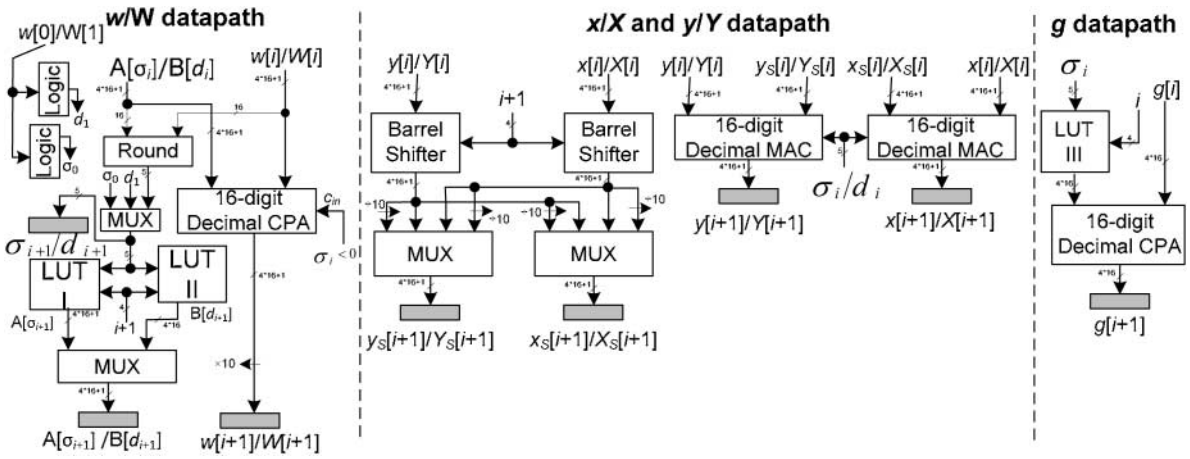

FIGURE 4. The proposed decimal CORDIC architecture (16-digit).

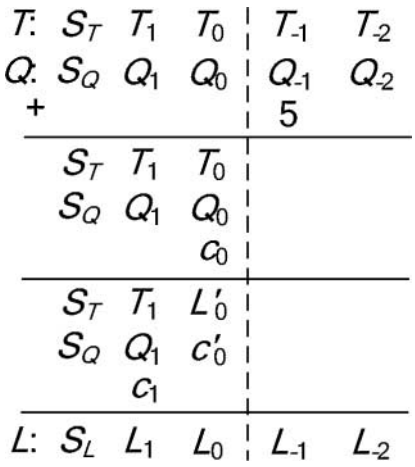

FIGURE 5. Straightforward rounding method.

$C_0 = c_0 + c'_0$, where $c_{-1} = \lfloor (T_{-2} + Q_{-2})/10 \rfloor$. The latter is computed by Equation (34), where $T^3_{-2}T^2_{-2}T^1_{-2}$ and $Q^3_{-2}Q^2_{-2}Q^1_{-2}$ represent and T_{-2} and Q_{-2}, respectively.

$$c_0 = \begin{Bmatrix} 0 & \text{if } (T_{-1} + Q_{-1} + c_{-1} + 5) < 20 \\ 1 & \text{if } (T_{-1} + Q_{-1} + c_{-1} + 5) \geq 20 \end{Bmatrix} = \left\lfloor \frac{T_{-1} + Q_{-1} + c_{-1} + 1}{16} \right\rfloor, \tag{32}$$

$$c'_0 = \begin{Bmatrix} 0 & \text{if } (T_{-1} + Q_{-1} + c_{-1} + 5) < 10 \\ 1 & \text{if } (T_{-1} + Q_{-1} + c_{-1} + 5) \geq 10 \end{Bmatrix} = \left\lfloor \frac{T_{-1} + Q_{-1} + c_{-1} + 11}{16} \right\rfloor, \tag{33}$$

$$c_{-1} = T^3_{-2}Q^3_{-2} + (T^3_{-2} + T^2_{-2})Q^2_{-2}Q^1_{-2} + (Q^3_{-2} + Q^2_{-2})T^2_{-2}T^1_{-2}. \tag{34}$$

We use 4-bit binary carry look-ahead logic to compute c_0 and c'_0 via Equations (35) and (36), where the relevant generate and propagate signals are shown in Equation sets (37) and (38), respectively. Note that in deriving the Equations (34) and (35), the following relations are taken into account that hold on the bits $bcde$ of a BCD digit: $bc = bd = 0$, $b\bar{c} = b\bar{d} = b$, $\bar{b}c = c$ and $\bar{b}d = d$.

$$c_0 = (g_3 + g_0 p_1 p_2 p_3) + c_{-1}(p_1 p_2 p_3), \tag{35}$$

$$c'_0 = (T^3_{-1} + Q^3_{-1}) - g'_3 + g'_2 p'_3 + g'_1 p'_2 p'_3 + g'_0 p'_1 p'_2 p'_3, \tag{36}$$

$$g_0 = Q^0_{-1} + T^0_{-1}; \quad p_0 = 1;$$
$$(g_i = Q^i_{-1}T^i_{-1}; \quad p_i = Q^i_{-1} + T^i_{-1} \quad \text{for} = 1, 2, 3) \tag{37}$$

$$g'_0 = \overline{(Q^0_{-1} \oplus T^0_{-1})}(c_{-1});$$
$$g'_1 = \overline{(Q^1_{-1} \oplus T^1_{-1})}(Q^0_{-1} + T^0_{-1});$$
$$p'_1 = \overline{(Q^1_{-1} \oplus T^1_{-1})} + (Q^0_{-1} + T^0_{-1}),$$
$$g'_2 = (Q^2_{-1} \oplus T^2_{-1})(Q^1_{-1} + T^1_{-1});$$
$$p'_2 = (Q^2_{-1} \oplus T^2_{-1}) + (Q^1_{-1} + T^1_{-1}),$$
$$g'_3 = \overline{(Q^3_{-1} \oplus T^3_{-1})}(Q^2_{-1} + T^2_{-1});$$
$$p'_3 = \overline{(Q^3_{-1} \oplus T^3_{-1})} + (Q^2_{-1} + T^2_{-1}). \tag{38}$$

Since c_0 is delivered quite in advance of c'_0, we use a BCD digit adder [31] with c_0 as the carry-in and L'_0 as the sum, followed by a BCD conditional incrementor described by Equation set (39),

where $L'_0 = L'^3_0 L'^2_0 L'^1_0 L'^0_0$.

$$L^0_0 = L'^0_0 \oplus c'_0, \quad L^1_0 = \overline{L'^1_0 \oplus \overline{(L'^0_0 c'_0)}},$$
$$L^2_0 = \overline{L'^2_0 \oplus \overline{(L'^0_0 L'^1_0 c'_0)}}, \quad L^3_0 = \overline{L'^3_0 \overline{(L'^0_0 L'^1_0 L'^2_0 c'_0)}}. \qquad (39)$$

Figure 6 depicts the overall block diagram for the round block.

6.2. Look-up tables

There are three LUTs embedded in the proposed architecture in Fig. 4. LUT I stores $A[\sigma_{i+1}] = \tan^{-1}(\sigma_{i+1}10^{-i-1})$ and in the actual implementation, for $n = 16$, holds 64 data words of 65 bits each, where there are nine input lines (i.e. five lines for the microrotation factor σ_{i+1} and four lines for the iteration index).

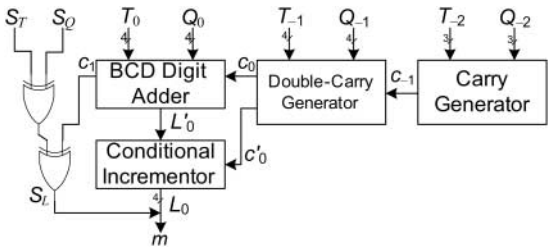

FIGURE 6. Block diagram of the rounding operation.

However, due to the symmetry of the tangent function, negative values are not stored. Moreover, for $i \geq 8$, $A[\sigma_{i+1}]$ can be approximated by σ_{i+1}, where the error is less than *ulp*. The LUTs II and III are of size 256×64 and hold $10^{i+1} \ln(1+d_{i+1}10^{-i-1})$ and $\ln(1 + (\sigma_i)^2 10^{-2i})$, respectively.

6.3. Simplified decimal MAC

To implement the main component of the CORDIC algorithm and that of the exponential function (i.e. the first two recurrences in Equations (1) and (21)), we design a simplified decimal MAC circuit, two copies of which are used in the architecture of Fig. 4. These MAC blocks compute $S = d \times M + H$, where S, M and H are 16-digit 10's complement BCD numbers and d is a 5-bit redundant decimal digit ($d \in \{-10, -5, \ldots, 5, 10\}$).

To compute $d \times M$, we first generate *easy* multiples of M (i.e. $M, 2M$ and $5M$), represented in 4-2-2-1 encoding. The corresponding circuit, based on the work in [9], is shown in Fig. 7. The required M multiples can be obtained as $U + V = d \times M$, with the details shown in Table 4. Also, the actual circuitry for this product generator is illustrated in Fig. 8, where N_U and N_V are the negating signals.

To complete the MAC computation $S = d \times M + H$, we use a 4-2-2-1 (3:2) compressor–convertor [32] followed by a 16-digit decimal add/subtract unit [30] that results in $S = U + V + H$. The details are illustrated in Fig. 9.

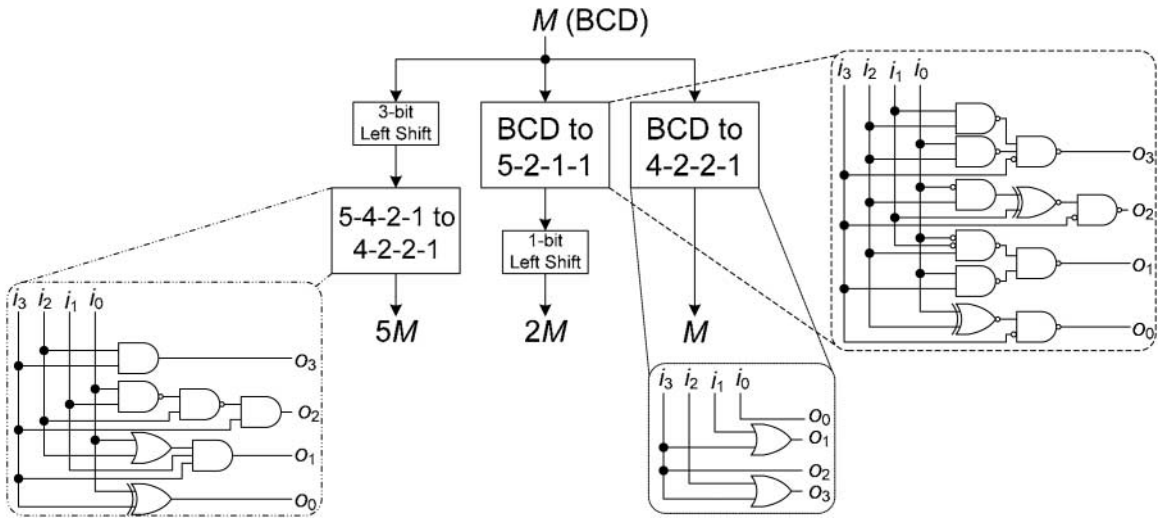

FIGURE 7. Generation of *easy* multiples (a digit-slice).

TABLE 4. Selection of the *easy* multiples.

d	-10	-5	-4	-3	-2	-1	0	1	2	3	4	5	10
U	$-5M$	$-5M$	$-5M$	$-2M$	$-2M$	0	0	0	$2M$	$2M$	$5M$	$5M$	$5M$
V	$-5M$	0	M	$-M$	0	$-M$	0	M	0	M	$-M$	0	$5M$

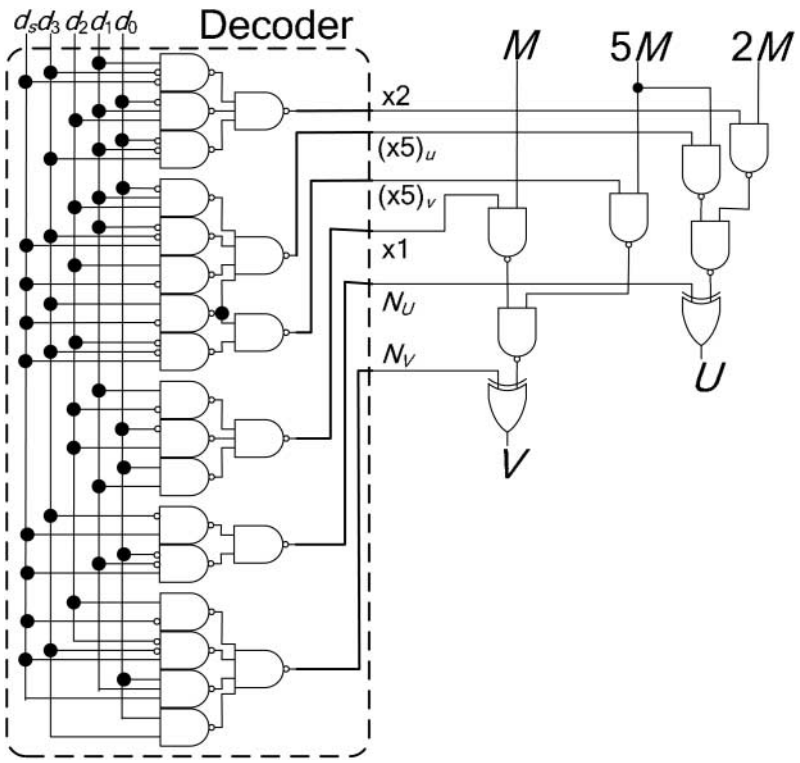

FIGURE 8. The decoder and a bit-slice of the logic for computing $U + V = d \times M$.

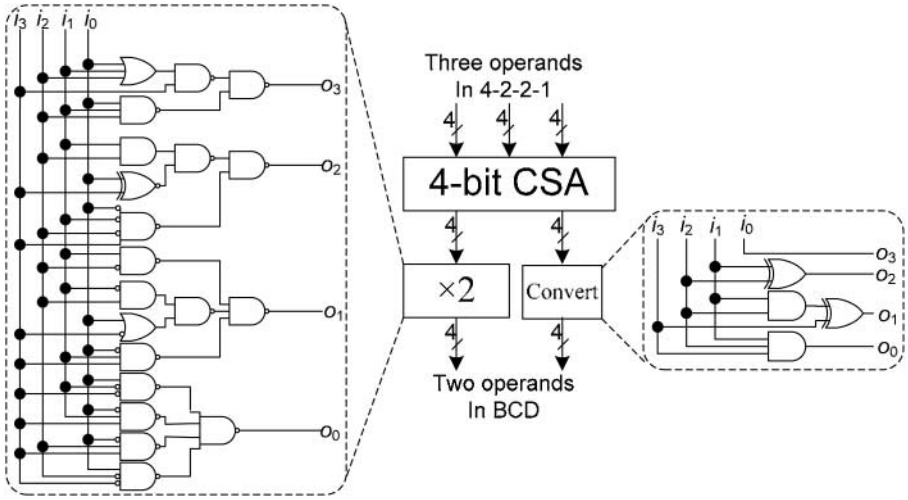

FIGURE 9. A 4-2-2-1 (3:2) compressor–convertor (a digit-slice).

7. EVALUATIONS AND COMPARISON WITH PREVIOUS WORKS

The most recent work on decimal CORDIC that we have encountered is due to [12], where the main design basis is the same as in [16], but the concentration is on area saving and using the same hardware for both hyperbolic and circular coordinates as required by the IBM POWER6 architecture [4]. Therefore, given that in the proposed design the emphasis is on time efficiency, we use the work of [16] as the main comparison reference. Although there are other interesting decimal CORDIC-like methods (e.g. [33]), given that the works in [12, 16] are shown to be the best state-of-the-art designs [16], we decide to evaluate the performance of each data path of the proposed architecture and compare the results with the corresponding results in [16]. The differences in the evaluation results are high enough to allow a fair comparison based on

the logical effort (LE) model [34]. Therefore, based on the following assumptions, we evaluate delays in terms of FO4 units (i.e. the delay of an inverter with a fan-out of four inverters) and area in minimum size NAND2 gate units.

The LE method is ideal for evaluating alternatives in the early design stages and provides a good starting point for more intricate optimizations [34]. Therefore, neither we undertake optimizing techniques such as gate sizing, nor consider the effect of interconnections. We rather allow gates with the drive strength of the minimum-sized inverter and assume equal input and output loads.

7.1. Evaluation of the proposed architecture

Recalling the timing of the proposed algorithm (Table 3), the corresponding architecture, as is shown in Fig. 4, includes three data paths, each of which is separately examined and evaluated below. The conclusion is that the x/X and y/Y data path is the critical one (34.62 FO4). Therefore, it is the latency of this data path that determines the cycle time, where the number of cycles for 16-digit operands is figured out to be 35.

The total area of the proposed CORDIC architecture is evaluated to be equal to that of 18 826 NAND2 and that of 4.5 KB ROM.

The area and delay of the components used in the proposed architecture are reported in Table 5.

7.1.1. w/W data path

The critical delay path consists of the round block, a 3-to-1 MUX, the LUT II, a 2-to-1 MUX and a latch (Fig. 4).

The critical delay path for the round block (Fig. 6) consists of the carry generator, the double-carry generator, the BCD digit adder and the conditional incrementor. The BCD digit adder is the same as that introduced in [31].

For the 9-bit input LUT, we assume multiplexer implementation. The critical path, as such, consists of nine levels of 64-bit 2-to-1 MUXes.

TABLE 5. Area delay of the components.

#	Component	Area	Delay (FO4)
1	The σ_0 logic (Fig. 1)	21 NAND2	Not in critical path
2	The d_1 logic (Fig. 2)	7 NAND2	Not in critical path
3	The round block (Fig. 6)	67 NAND2	10.70
4	The 5-bit 3-to-1 MUX	34 NAND2	4.39
5	LUT I	≈0.5 Kbytes	Not in critical path
6	LUT II, III	2 Kbytes	14.11
7	16-digit BCD 9's compl. [31]	80 NAND2	Not in critical path
8	The 65-bit 2-to-1 MUX	220 NAND2	1.45
9	The 16-digit BCD adder	2655 NAND2	Not in critical path
10	Barrel Shifter	877 NAND2	Not in critical path
11	65-bit 4-to-1 MUX	658 NAND2	Not in critical path
12	16-digit MAC	5136 NAND2	29.14

The delay for the latch, as in [22], is assumed to be equal to that of three XOR gates.

With the above assumptions, the overall latency of the w/W data path is evaluated to be 32.13 FO4. The area, as the sum of the areas of the components (#1–#9 from Table 5), is evaluated to be equal to that of 3084 NAND2 plus that of 2.5 KB ROM.

7.1.2. x/X and y/Y data path

The critical path consists of a 16-digit simplified decimal MAC and a latch.

The critical path of the MAC consists of a BCD-to-5211 convertor (Fig. 7), the circuit for generating U and V (Fig. 8), a binary full-adder followed by the $\times 2$ block (Fig. 9) and a decimal adder/subtractor. Therefore, the latency of this data path is evaluated to be 34.62 FO4 and the area, which is the sum of the areas of the components (#10–12 from Table 5), is equal to that of 13342 NAND2.

7.1.3. The g data path

The critical delay path consists of the LUT III (with the same size and latency as in LUT II), a 16-digit BCD adder and a latch. Therefore, the overall latency of this data path is 31.71 FO4 and the corresponding area is evaluated to be equal to that of 2400 NAND2 and that of a 2 KB ROM.

7.2. Evaluation of the constant scaling factor method [16]

Conventional CORDIC algorithms usually lead to constant scaling factor. This helps in saving area and time. Our major comparison reference for decimal CORDIC is the work of [16], which relies on keeping the scaling factor constant. Therefore, the architecture has only two data paths:

(i) *Z data path*: The critical path consists of an 8×64 LUT, a decimal adder/subtractor and a latch. Due to the lack of any concrete delay and area measures in [16], we have done our own LE evaluation of their work with the same assumptions as in the preamble of Section 7. The result shows 33.16 FO4 for the latency and 2655 NAND2 for area as well as a 2 KB ROM.

(ii) *X/Y data path*: The critical path consists of a 16-digit barrel shifter, a $\times 5$ block, a 4-bit 3-to-1 MUX, a 16-digit decimal adder/subtractor and a latch. The latency and area of this data path are evaluated to be 34.59 FO4 and 9054 NAND2, respectively.

Consequently, the latency of the architecture introduced in [16] is 34.59 FO4 (i.e. the cycle time is almost equal to that of the proposed method) and the consumed area is equal to that of 11 709 NAND2 plus 2 KB ROM. The total number of cycles required for 16-digit implementation is 64 (equal to the number of bits of the operands) vs. the 35 cycles in the proposed method. The full comparison result is tabulated in Table 6, where the speed advantage of the proposed design is ∼82% at the cost of 60% more area and 2.5 KB more ROM. The major drawback of

TABLE 6. Comparison with the best previous work (16-digit architecture).

	Latency of each cycle (FO4)	# of cycles	Total delay (FO4)	Ratio	Total area (NAND2)	Ratio	ROM (KB)
Proposed	34.62	35	1211.70	1	18826	1	4.5
[16]	34.59	64	2213.76	1.82	11709	0.62	2.0

the proposed architecture is the high area cost, which makes this design not suitable for applications in which area cost is critical.

It should be noted that no evaluation has been made in [16] about the method used for compensating the constant scaling factor. Although, one may use the standard compensation algorithms [24], this would add extra area/delay to the CORDIC hardware of [16]. The proposed architecture shows delay advantage over that of [16], even though we have not taken into account the compensation overhead in the analysis of [16].

8. CONCLUSIONS

Hardware implementation of the decimal CORDIC algorithm has been addressed, apparently for the first time, in [28] and recently in [11, 12, 16]. All the latter works can be categorized as sacrificing the speed for reduced complexity of the selection function in the angle recurrence, where the number of iterations is at least four times the number of digits of the input operand (i.e. $4n$). Also the scaling factor is kept constant in all the latter works. However, via relaxing the latter criteria and adapting for radix-10 the well-known *selection by rounding* technique, we have managed to keep the number of iterations in the main CORDIC rotation to $n + 2$. This reduction in the number of iterations is achieved at the cost of another $n + 1$ iterations that is needed to compensate for the scaling factor. However, the latter is executed on the same rotation hardware. Despite this unpleasant obligation, our method shows 82% more speed at the cost of 60% more area and 2.5 KB more ROM with respect to the best previous work due to [16].

The major specifications of the proposed method are:

(i) *Selection by rounding:* This technique helps in reducing the number of iterations and the complexity of the selection function.

(ii) *Retiming:* Retiming the angle recurrence data path and the main CORDIC recurrence data path have resulted in reducing the cycle time.

(iii) *Decimal round block:* This architecture extracts the rounded integer part of the remaining rotation angle out of its redundant double-BCD representation.

(iv) *Simplified decimal MAC:* This component helps in reducing the cycle time.

(v) *Reusing the CORDIC rotation hardware for compensation of the scaling factor:* The many decimal multiplications required for computing the variable scaling factor are reduced to addition operations via logarithmic representation of the scaling factor. This necessitates a final decimal exponentiation for compensating the scaling factor. To perform the latter, the CORDIC rotation hardware is reused.

Further research is ongoing for extending the proposed method for hyperbolic coordinate and vectoring mode in order to design a unified decimal CORDIC hardware unit based on selection by rounding.

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for their valuable comments on the original manuscript.

FUNDING

This research was supported in part by IPM under Grant CS1389-2-03, and in part by Shahid Beheshti University.

REFERENCES

[1] Wang, L.K., Schulte, M.J., Thompson, J.D. and Jairam, N. (2009) Hardware designs for decimal floating-point addition and related operations, *IEEE Trans. Comput.*, **58**, 322–335.

[2] Cowlishaw, M.F. (2003) Decimal Floating-Point: Algorism for Computers, *Proc. 16th IEEE Symp. Computer Arithmetic (ARITH16)*, Santiago de Compostela, Spain, pp. 104–111.

[3] IEEE Standards Committee, (2008) *754-2008 IEEE Standard for Floating-Point Arithmetic*, IEEE Computer Society Std., pp. 1–58.

[4] Eisen, L. *et al.* (2007) IBM POWER6 accelerators: VMX and DFU, *IBM J. Res. Dev.*, **51**, 663–684.

[5] Webb, C.F. (2008) IBM z10: the next-generation mainframe microprocessor, *IEEE Micro*, **28**, 19–29.

[6] Goldstine, H.H. and Goldstine, A. (1996) The electronic numerical integrator and computer (ENIAC), *IEEE Ann. Hist. Comput.*, **18**, 10–16.

[7] Lang, T. and Nannarelli, A. (2007) A radix-10 digit-recurrence division unit: algorithm and architecture, *IEEE Trans. Comput.*, **56**, 727–739.

[8] Jaberipur, G. and Kaivani, A. (2009) Improving the speed of parallel decimal multiplication, *IEEE Trans. Comput.*, **58**, 1539–1552.

[9] Vazquez, A., Antelo, E. and Montuschi, P. (2010) Improved design of high-performance parallel decimal multipliers, *IEEE Trans. Comput.*, **59**, 679–693.

[10] Wang, L.K. and Schulte, M.J. (2005) Decimal Floating-Point Square Root Using Newton-Raphson Iteration, *16th IEEE Int. Conf. Application-Specific Systems, Architecture Processors (ASAP'05)*, Samos, Greece, pp. 309–315.

[11] Jimeno, A., Mora, H., Sanchez, J.L. and Pojul, F. (2008) A BCD-based architecture for fast coordinate rotation, *J. Syst. Archit.*, **54**, 829–840.

[12] Vazquez, A., Villalba, J. and Antelo, E. (2009) Computation of Decimal Transcendental Functions Using the CORDIC Algorithm, *Proc. 19th IEEE Symp. Computer Arithmetic (ARITH19)*, Portland, OR, USA, pp. 179–186.

[13] Harrison, J. (2009) Decimal Transcendentals via Binary, *Proc. 19th IEEE Symp. Computer Arithmetic (ARITH19)*, Portland, OR, USA, pp. 187–194.

[14] Muller, J.M. (2007) *Elementary Functions: Algorithms and Implementation* (2nd edn). Birkhauser Verlag AG.

[15] Meher, P.K., Valls, J., Juang, T.B., Sridharan, K. and Maharatna, K. (2009) 50 years of CORDIC: algorithms, architectures and applications, *IEEE Trans. Circuits and Syst. I: Regular Papers*, **56**, 1893–1907.

[16] Vazquez, A. and Antelo, E. (2008) Constant Factor CORDIC for Decimal BCD Input Operands, *8th Real Numbers and Computers Conf. (RNC8)*, Santiago de Compostela, Spain, pp. 83–91.

[17] Volder, J.E. (1959) The CORDIC trigonometric computing technique, *IRE Trans. Electron. Comput.*, **EC-8**, 330–334.

[18] Antelo, E., Lang, T. and Bruguera, J.D. (2000) Very-high radix circular CORDIC: vectoring and unified rotation/vectoring, *IEEE Trans. Comp.—Spec. Issue Comput. Arith.*, **49**, 727–739.

[19] Antelo, E., Lang, T. and Bruguera, J.D. (2000) Very-high radix circular CORDIC based on selection by rounding, *J. VLSI Signal Process.*, **25**, 141–153.

[20] Antelo, E., Bruguera, J.D., Lang, T., Villalba, J. and Zapata, E.L. (1996) High Radix CORDIC Rotation based on Selection by Rounding, *Proc. Euro-Par'96*, Lyon, France, pp. 155–164.

[21] Svoboda, A. (1963) An Algorithm for Division, *Inform. Process. Mach.*, **9**, 25–34.

[22] Ercegovac, M.D., Lang, T. and Montuschi, P. (1994) Very-high radix division with prescaling and selection by rounding, *IEEE Trans. Comput.*, **43**, 909–918.

[23] Lang, T. and Montuschi, P. (1995) Very-high Radix Combined Division and Square Root with Prescaling and Selection by Rounding, *Proc. 12th IEEE Symp. Computer Arithmetic (ARITH12)*, Bath, UK, pp. 124–131.

[24] Ercegovac, M.D. and Lang, T. (2004) *Digital Arithmetic*, Morgan Kaufmann Publishers.

[25] Parhami, B. (2010) *Computer Arithmetic: Algorithms and Hardware Designs* (2nd ed). Oxford University Press, New York.

[26] Haviland, G.L. and Tuszynski, A.A. (1980) A CORDIC arithmetic processor chip, *IEEE Trans. Comput.*, **C-29**, 68–79.

[27] Timmermann, D., Hahn, H. and Hosticka, B.J. (1992) Low latency time CORDIC algorithms, *IEEE Trans. Comput.*, **41**, 1010–1015.

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

[29] Baker, P.W. (1975) Parallel multiplicative algorithms for some elementary functions, *IEEE Trans. Comput.*, **C-24**, 322–325.

[30] Vazquez, A. and Antelo, E. (2006) Conditional Speculative Decimal Addition, *Proc. 7th Conf. Real Numbers and Computers (RNC7)*, Loria, Nancy, France, pp. 47–57.

[31] Schmookler, M. and Weinberger, A. (1971) High speed decimal addition, *IEEE Trans. Comput.*, **C-20**, 862–866.

[32] Vazquez, A., Antelo, E. and Montuschi, P. (2007) A New Family of High-performance Parallel Decimal Multipliers, *Proc. 18th IEEE Symp. Computer Arithmetic (ARITH18)*, Montpellier, France, pp. 195–204.

[33] Imbert, L., Muller, J.M. and Rico, F. (2000) A radix-10 BKM algorithm for computing transcendentals on pocket computers, *J. VLSI Signal Process.*, **25**, 179–186.

[34] Sutherland, I., Sproull, R. and Harris, D. (1999) *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann.

APPENDIX 1: PROOF OF $R_{i+1} > 5.5$ FOR $i \geq 1$

Recall the definition $R_{i+1} = 10^{i+1} \times [\sum_{j=i+1}^{n} \tan^{-1}(5 \times 10^{-j}) + \tan^{-1}(10^{-n})]$. The latter can be simplified, as follows, based on the fact that $\tan^{-1}(5 \times 10^{-j}) > 4.99 \times 10^{-j}$, for $j \geq 2$.

$$R_{i+1} > 10^{i+1} \times 4.99 \times \sum_{j=i+1}^{n} 10^{-j} = 4.99 \times \sum_{j=0}^{n-(i+1)} 10^{-j} > 4.99 \times 1.11 > 5.5 \quad (i \geq 1).$$

APPENDIX 2: PROOF OF $5 - 10^{i}\tan^{-1}(5 \times 10^{-i}) < 0.05$ FOR $i \geq 2$

It can be easily verified that $[\tan^{-1}(5 \times 10^{-i}) > 4.99 \times 10^{-i}$ for $i \geq 2]$, which leads to the following desired results:

$$10^{i} \tan^{-1}(5 \times 10^{-i}) > 4.99 \Rightarrow 5 - 10^{i} \tan^{-1}(5 \times 10^{-i}) < 0.01 < 0.05 \quad (i \geq 2).$$

APPENDIX 3: PROOF OF $S_{i+1} > 5.5$ FOR $i \geq 2$

Recall the definition $S_{i+1} = 10^{i+1} \times [\sum_{j=i+1}^{n} \ln(1+5\times10^{-j}) + \ln(1+10^{-n})]$. The latter can be simplified, as follows, based on the fact that $4.98 \times 10^{-j} < \ln(1 + 5 \times 10^{-j})$, for $j \geq 3$.

$$S_{i+1} > 10^{i+1} \times 4.98 \times \sum_{j=i+1}^{n} 10^{-j} = 4.98 \times \sum_{j=0}^{n-(i+1)} 10^{-j} > 4.98 \times 1.11 > 5.5 \quad (i \geq 2).$$

APPENDIX 4: PROOF OF $[5 - 10^{i}\ln(1 + 5 \times 10^{-i}) < 0.03]$ FOR $i \geq 3$

It can be easily verified that $[\ln(1 + 5 \times 10^{-i}) > 4.98 \times 10^{-i}$ for $i \geq 3]$, which leads to the following desired results:

$$10^{i} \ln(1 + 5 \times 10^{-i}) > 4.98 \Rightarrow 5 - 10^{i} \ln(1 + 5 \times 10^{-i}) < 0.02 < 0.03 \quad (i \geq 3).$$