Redundant Floating-Point Decimal CORDIC Algorithm

Álvaro Vázquez, Julio Villalba-Moreno, Elisardo Antelo, and Emilio L. Zapata

doi: 10.1109/TC.2011.217

Redundant Floating-point Decimal CORDIC Algorithm

Álvaro Vázquez, Julio Villalba**, Elisardo Antelo, Emilio L. Zapata**
Dept. Electronic and Computer Science, University of Santiago, SPAIN
**Dept. Computer Architecture, University of Málaga, SPAIN
alvaro.vazquez@usc.es, jvillalba@uma.es, elisardo.antelo@usc.es, zapata@uma.es

Abstract—In this work we propose a new decimal redundant CORDIC algorithm to manage transcendental functions, using floating-point representation. The algorithms determine the direction of the elementary rotation using sign estimations. Unlike binary redundant CORDIC, repetition of iterations are not required to ensure convergence since novel decimal codes have been carefully selected with sufficient redundancy to prevent any repetition. The algorithms are mapped to a low-cost unit based on a decimal 3-2 carry-save adder which can also be used as a floating-point decimal division unit. Compared to current decimal floating-point units, the implementation of our algorithm involves minor modifications of the native hardware, while providing a huge set of elementary functions.

Index Terms—CORDIC algorithm, Decimal arithmetic, carry-save arithmetic

1 Introduction

The processing of numbers represented in the floating-point decimal format is driven by two increasingly important areas-legal issues and standard practices in financial data processing-as well as by the growing complexity of financial applications (e-commerce, banking, etc). This has motivated the standardisation of floating-point decimal numbers in representation and processing (included in the IEEE 754–2008 floating-point standard [1]), and increased interest in support (hardware, software or a mix of both) for floating-point decimal processing on the part of microprocessor companies.

The IEEE 754–2008 standard allows two different storage formats, DPD (Densely Packed Decimal) and BID (Binary Integer Decimal). The DPD format is in fact a compressed version of BCD encoding. Current hardware support in commercial microprocessors uses BCD for the implementation of decimal floating-point units [2] [3] [4]. Regarding BID, current implementations rely on a highly optimized Decimal Floating-Point Math Library [5].

In this study we focus on BCD hardware implementations. A representative state-of-the-art industrial unit [2] provides support for both Decimal64 and Decimal128, with a 144-bit-wide datapath. Its main components are a fast decimal adder, a parallel left/right rotator (for shifting operations), x2 and x5 multiple generators and support registers and multiplexers. This unit directly supports floating-point addition/subtraction, multiplication and division. Multiplication and division are performed iteratively digit-by-digit. In the worst case, floating-point multiplication requires 89 cycles for the Decimal128 format. This is very slow compared to a state-of-the-art binary double precision floating-point implementation (about 4-8 cycles a singly cycle throughput). The main reason for such limitations appears to be the high cost (in area and power) involved in supporting the Decimal128 format. If we look at the layout of the Z10 processor [4] we observe that the area devoted to the decimal (serial) floating-point unit is similar to the area of the binary double precision (parallel) floating-point unit. Therefore, we may expect significantly greater decimal floating-point multiplication latencies than in the corresponding binary operation during the next few years.

In this study we concentrate on the implementation of floating-point decimal transcendental functions using redundant arithmetic (carry–save). Our aim is to provide an alternative implementation to support the floating-point decimal number system defined by the IEEE 754–2008 standard for the computation of transcendentals. Similarly, Intel provided a software implementation for transcendentals in its Intel Decimal Floating-Point Math Library [6].

We have already proposed floating-point decimal transcendental functions based on CORDIC in [7] and [8]. Specifically, in [7] we considered the CORDIC algorithm for the implementation of decimal transcendentals, involving a constant scale factor fixed-point algorithm for computing trigonometric functions. In [8] we extended the algorithm to hyperbolic coordinates to increase the number of functions computed, dealing with floating-point numbers and mapping the algorithm onto a state-of-the-art DFPU unit, using a carry propagate adder. We also developed a preliminary redundant version of the algorithm (using fast carry–save adders) for one

This work has been partially supported by the Ministry of Science and Innovation of Spain under projects TIN2007-67537-C03-01, TIN2006-01078, TIN2010-16144 and TIN 2010-17541

Digital Object Indentifier 10.1109/TC.2011.190

of the four possible combination of modes and coordinates: the rotation mode in circular coordinates (four options are possible: circular rotation, circular vectoring, hyperbolic rotation and hyperbolic vectoring). The main contribution of this study is the full extension of the redundant algorithm to the remaining three cases; this is not straightforward due to convergence problems. We also map the redundant algorithm onto a simple unit based on a 3-2 carry-save adder that may also be used for operations such as floating-point decimal division, avoiding use of the floating-point adder/multiplier for the computation of transcendentals. Moreover, in contrast to our previous early implementation [8], the sign estimation logic necessary to determine the direction of each elementary rotation is outside the critical path, and fully overlapped with datapath computations. CORDIC was used by low-cost processors, such as calculators, and by microprocessors when multiplication hardware was slow (serial). This algorithm was no longer implemented in microprocessors when high-performance parallel multipliers became available, since the elementary functions are efficiently implemented by means of table–based polynomial approximation methods. We consider CORDIC an attractive approach to compute decimal transcendentals, due to its potential low cost, and the fact that a fully parallel decimal multiplier is too costly in terms of area and power. The current stage of development of decimal floating-point units resembles the binary units in their early stages of development with slow floating-point multipliers, and when CORDIC was considered an attractive approach for transcendental functions.

The paper is organized as follows: Section 2 briefly describes the CORDIC algorithm and reviews our previous work on decimal CORDIC [7] [8]. In Section 3 we present the redundant version of the algorithm for all modes of operation, and determine the number of digits required for the sign estimations, such that convergence is ensured without any repeated iterations. Section 4 is devoted to the floating-point extension of the redundant algorithm. In Section 5 we discuss reducing the latency of the algorithm by using a linear approximation of the rotation. Section 6 discusses mapping the algorithm to a low-cost unit that can be shared with a floating-point division algorithm. In this section we also evaluate the number of cycles required for the different functions computed. Section 7 compares related studies, and the conclusions are presented in Section 8. In order to make the work more readable and not distract the reader from the main flow, some auxiliary proofs have been made available through the report [15].

2 Unified Constant Factor Decimal CORDIC

2.1 Fundamentals of the CORDIC algorithm

The CORDIC algorithm performs plane rotations of vectors preserving a norm for circular or hyperbolic rotation. We describe the algorithm for circular coordinates as the basis for computing vector rotation, then we extend it to the vectoring operation that computes the modulus and angle of a vector. The intended operation consists in rotating a vector (x_{in}, y_{in}) by an angle $z_{in} = \theta$. Any rotation angle may be represented as a sequence of n angles of the form

$$\theta \approx \sum_{j=1}^{n} \sigma_j \alpha_j$$

where σ_j indicates the direction of rotation ($\sigma_j = -1$ for a clockwise rotation and $+1$ for counterclockwise rotation). Therefore, the rotation by θ can be performed as a sequence of 2D elementary rotations

$$\cos(\sigma_j\ \alpha_j) \begin{pmatrix} 1 & -\tan(\sigma_j\ \alpha_j) \\ \tan(\sigma_j\ \alpha_j) & 1 \end{pmatrix}$$

All the cosine factors are precomputed as one factor $K = \prod_{j=1}^{n} \cos(\sigma_j \alpha_j)$. Note that K is a constant independent of the rotation angle since $\cos(\sigma_j \alpha_j)$ does not depend on the value of $\sigma_j \in \{-1, 1\}$. The multiplication by the precomputed factor K can be done before or after the 2D elementary rotations. The σ_j values are computed by implementing the recurrence

$$z[j+1] = z[j] - \sigma_j \alpha_j$$

To have a remainder that converges to zero, $\sigma_j = sign(z[j])$. The algorithm is extended to the vectoring mode in a simple way. In this case, the direction of the rotations is calculated as $\sigma_j = -sign(y[j])$ (assuming $x[j] \geq 0$) to zero out the y coordinate ($(x[j], [j])$ is the vector obtained after the $j-1$ plane rotation).

The algorithm is easily extended to hyperbolic rotations by using the following elementary step

$$\cosh(\alpha_j) \begin{pmatrix} 1 & \tanh(\sigma_j \alpha_j) \\ \tanh(\sigma_j \alpha_j) & 1 \end{pmatrix}$$

To obtain the convergence conditions, for the rotation mode we consider a finite set of angles α_j and that the value of the angles is rounded to fit the wordlength of the datapath. Let I_j be the angular convergence domain at iteration j, that is $I_j = \sum_{i=j}^{n} \alpha_i$.

For the CORDIC algorithm convergence is achieved if the sequence of elementary angles verifies [9]:

$$\alpha_j < I_{j+1} + \alpha_n \tag{1}$$

This result is valid both for rotation and vectoring (the same reasoning may be performed using $\tan^{-1}(y[j]/x[j])$ instead of $z[j]$), taking into account that truncation is used instead of rounding.

For the z coordinate, the convergence condition results in:

$$z[j] \leq I_j + \alpha_n \tag{2}$$

For the conventional CORDIC algorithm, the elementary angles are taken as $\alpha_{j,1} = \tan^{-1}(2^{-j})$ for circular

coordinates and $\alpha_{j,-1} = \tanh^{-1}(2^{-j})$ for hyperbolic coordinates. This results in the following well-known basic CORDIC iteration, where $c = 1$ for circular coordinates and $c = -1$ for hyperbolic coordinates and $\sigma_j = sign(z[j])$ (rotation mode) or $\sigma_j = -sign(y[j])$ (vectoring mode):

$$x[j+1] = x[j] - c\ \sigma_j\ 2^{-j}\ y[j] \quad (3)$$
$$y[j+1] = y[j] + \sigma_j\ 2^{-j}\ x[j] \quad (4)$$
$$z[j+1] = z[j] - \sigma_j\ \alpha_{j,c} \quad (5)$$

This ensures that the implementation of the basic elementary step requires additions and binary shiftings (since for circular (hyperbolic) coordinates, the tangent (hyperbolic tangent) of the angle is a power of two). Moreover, for circular coordinates the condition of convergence (1) is ensured. For hyperbolic coordinates some elementary angles need to be repeated for convergence [9].

2.2 Unified Constant Factor Decimal CORDIC

After reviewing the conventional CORDIC algorithm, we conclude that in order to obtain a simple constant factor decimal CORDIC algorithm the following constraints apply:

- To have a constant scale factor, for each elementary rotation we should have two possible angles of the same magnitude but with a different sign.
- The tangent (or hyperbolic tangent) of each of the elementary angles should be a simple multiple of a power of ten, in order to have simple operations in the implementation of the elementary step.

Next, we briefly present the unified CORCIC algorithm (circular and hyperbolic) to deal with decimal BCD numbers [8]. Let us denominate $(a_3a_2a_1a_0)$ as the binary weights of the four bits of a BCD number. Inspired by what is done for the binary CORDIC and taking into account the above constraints we select the angles as the arctangent (or hyperbolic arctangent) of the binary weights of a valid binary coded decimal representation[1]. For instance, for circular coordinates and the code $(a_3a_2a_1a_0)$=(8421), we take angles of the form: $\tan^{-1}(8\ 10^{-1})$, $\tan^{-1}(4\ 10^{-1})$, $\tan^{-1}(2\ 10^{-1})$, $\tan^{-1}(10^{-1})$, $\tan^{-1}(8\ 10^{-2})$, and so on. In this way, the basic iteration step requires a decimal shifting (which is an easy operation for BCD operands) and a simple multiplication by a_3, a_2, a_1 or a_0.

The unified algorithm for a $(a_3a_2a_1a_0)$ code and m decimal digits is as follows ($c = 1$ for circular coordinates and $c = -1$ for hyperbolic coordinates):

Initialization: $x[1] = x_{in} \in [1, 10), y[1] = y_{in} \in (-10, 10)$ ($|y[1]| \leq |x[1]|$) and $z[1] = z_{in} = \theta$ ($\theta \in [-D_c, D_c]$),

Iteration: for $i = 1$ to $4m$

$\sigma_i = sign(z[i])$ (rotation mode) or $\sigma_i = -sign(y[i])$ (vectoring mode)

$$x[i+1] = x[i] - c\ \sigma_i\ C[i]\ 10^{-\lceil \frac{i}{4} \rceil}\ y[i] \quad (6)$$
$$y[i+1] = y[i] + \sigma_i\ C[i]\ 10^{-\lceil \frac{i}{4} \rceil}\ x[i] \quad (7)$$
$$z[i+1] = z[i] - \sigma_i\ \alpha_{i,c} \quad (8)$$

where $\alpha_{i,1} = \tan^{-1}(S[i])$, $\alpha_{i,-1} = \tanh^{-1}(S[i])$, with $S[i] = C[i]\ 10^{-\lceil \frac{i}{4} \rceil}$, and $C[i] = R[i \bmod 4]$ with $R[0] = a_0, R[1] = a_3, R[2] = a_2, R[3] = a_1$. The scale factor K_c is constant, since $\cos(\sigma_i\alpha_{i,1})$ and $\cosh(\sigma_i\alpha_{i,-1})$, with $\sigma_i \in \{-1, 1\}$, are constant. The angular convergence domain is given by $[-D_c, D_c]$, and depends somewhat on the $(a_3a_2a_1a_0)$ decimal code selected to generate the elementary angles. This algorithm may be directly used for fixed point input arguments within the range of convergence. The number of iterations, determined by the parameter m ($4m$ iterations), depends on the desired accuracy.

The convergence of the algorithm for both circular and hyperbolic coordinates is proved in [8] for the (5,2,1,1) code. This code is of interest since x5 and x2 are easy multiples in decimal, generated without any carry propagation [10].

For the unified algorithm the convergence given by equation (2) becomes:

$$RA[i] \leq \sum_{j=i}^{4m} \alpha_{j,c} + \alpha_{4m,c} \quad (9)$$

where $RA[i]$ is the remaining angle to be rotated from iteration i, i.e., $RA[i] = z[i]$ for the rotation mode (circular and hyperbolic), $RA[i] = \tan^{-1}(y[i]/x[i])$ for circular vectoring and $RA[i] = \tanh^{-1}(y[i]/x[i])$ for hyperbolic vectoring.

3 Algorithm for Carry-Save Representation

In this section, we discuss the proposed CORDIC algorithm using a redundant decimal carry-save representation (two full digits for each decimal position coded in standard BCD) for the operands[2].

The unified constant factor decimal CORDIC proposed in Section 2.2 uses the sign of a control coordinate. In a redundant representation a full conversion to conventional representation is involved to obtain the sign. This is a very slow operation which should be avoided. To do this, we use the sign of an estimation of the control coordinate to determine the σ_i values. The sign of the estimation is obtained by determining the sign of some leading digits of the carry-save representation of the control coordinate. Thus, only a few digits have to be converted from redundant to conventional representation. This scheme has already been used for binary CORDIC with redundant adders (see for instance [12]).

1. Usually binary coded decimal representations use four bits for each decimal digit of weights 8, 4, 2 and 1, but other alternative representations are possible (different binary weights).

2. Alternatively, decimal signed-digit representations could be used [11].

An error due to a wrong sign of the estimation may lead to convergence problems, which can be solved by repeating some iterations [13] [14] in the classic binary CORDIC.

An important issue is the selection of the decimal code to generate the elementary angles, since this may avoid repeating iterations (as required in binary CORDIC). In this section, we prove that for circular coordinates the representation derived from the decimal code 5221 has sufficient redundancy such that no repeated iterations are needed, whereas for the hyperbolic case code 5421 is necessary to ensure sufficient redundancy. Therefore, we determine the number of bits of the estimation such that the basic elementary angle set ensures convergence without any repeated iterations. The number of bits of the estimation is limited by the cycle time and the pipelining performed.

The scheme is based on an estimation of the sign by analyzing a few leading digits of the control coordinate. To have the digits analyzed in a fixed position, the control coordinate has to be scaled. A natural choice is to scale the control coordinate by the amount $10^{\lceil i/4 \rceil}$. For the rotation mode, we perform the following change of variable $r[i] = 10^{\lceil \frac{i}{4} \rceil}\ z[i]$ and use the scaled elementary angles $A_{i,c} = 10^{\lceil i/4 \rceil} \alpha_{i,c}$. The resulting recurrence for $r[i]$ is

$$r[i+1] = \begin{cases} 10(r[i] - \sigma_i\ A_{i,c}) & \text{if } \ i \bmod 4 = 0 \\ r[i] - \sigma_i\ A_{i,c} & \text{if } \ i \bmod 4 \neq 0 \end{cases} \quad (10)$$

The convergence condition (9) becomes:

$$|r[i]| < 10^{\lceil \frac{i}{4} \rceil} \sum_{j=i}^{4m} \alpha_{j,c} + A_{4m,c} \quad (11)$$

The proposed scaling does not affect the x and y coordinates.

For the vectoring mode, we perform the following scaling $w[i] = 10^{\lceil \frac{i}{4} \rceil}\ y[i]$. The resulting iterations are as follows:

$$w[i+1] = \begin{cases} 10(w[i] + \sigma_i\ C[i]\ x[i]) & \text{if } \ i \bmod 4 = 0 \\ w[i] + \sigma_i\ C[i]\ x[i] & \text{if } \ i \bmod 4 \neq 0 \end{cases} \quad (12)$$

and

$$x[i+1] \ = \ x[i] - \ c\ \sigma_i\ C[i]\ 10^{-2\lceil \frac{i}{4} \rceil}\ w[i] \quad (13)$$

The z coordinate is not affected by this scaling (see (8)).

In the next subsection we determine the number of digits of the estimation for both circular and hyperbolic coordinates and both rotation and vectoring modes. To make it easier to follow the main flow, some auxiliary proofs have been made available through the report [15],

3.1 Number of digits of the Estimation for Circular Coordinates

3.1.1 Rotation mode

To perform the computation using redundant representation, we estimate the sign of $r[i]$ by truncating some leading digits of the carry-save representation.

Let us denote as $\hat{r}[i]$ the truncation of $r[i]$ using t fractional digits, and $\hat{\sigma}_i$ the estimation of the sign obtained by selecting the sign of $\hat{r}[i]$ ($\hat{\sigma}_i = sign(\hat{r}[i])$). We now analyze the case of an incorrect estimation of the sign of $r[i]$ (that is, $\hat{\sigma}_i \neq \sigma_i$)). Taking into account that $r[i]$ is a carry-save number, the relationship between $r[i]$ and its truncated value using t fractional digits is:

$$\hat{r}[i] \leq r[i] < \hat{r}[i] + 2\ 10^{-t} \quad (14)$$

Therefore, if we have a positive $r[i]$ value of less than 10^{-t} (illustrated in Figure 1), according to equation (14) the corresponding truncated value $\hat{r}[i]$ can be -10^{-t}. Thus, the sign of the actual coordinate $r[i]$ is positive and the sign of its truncated value $\hat{r}[i]$ is negative, which leads to a microrotation in the wrong direction. In spite of this erroneous direction, it is possible to maintain convergence if the number of truncated digits is large enough due to redundancy.

Our aim is to determine a bound on t such that this erroneous estimation still allows convergence. In reference to the situation illustrated in Figure 1, and from Expression (14), we obtain $-10^{-t} \leq r[i] < 10^{-t}$. Two cases are possible: i) $-10^{-t} \leq r[i] < 0$ and then $\sigma_i = -1$ (the algorithm converges); and ii) $0 \leq r[i] < 10^{-t}$ and then $\sigma_i = +1 \neq \hat{\sigma}_i$ (as shown in Figure 1). Therefore, the convergence depends on the number of digits of the truncated estimation, as shown in the following.

Let us deal with the case in which $0 \leq r[i] < 10^{-t}$ (and $\hat{\sigma}_i = -1$). Thus, to update $r[i+1]$, an addition is performed instead of a subtraction. Now we prove that, despite the erroneously estimated sign, there is sufficient angle redundancy to ensure the convergence of the algorithm.

The minimum value of t is obtained for the case of minimum angular redundancy. Taking into account the recurrence (10) and the change of variable $r[i] = 10^{\lceil \frac{i}{4} \rceil}\ z[i]$, an estimation of t digits to determine the sign of $r[i]$ may produce a residual angle bounded by

$$z[i+1] < 10^{-t}\ 10^{-\lceil \frac{i}{4} \rceil} + \alpha_{i,1} \quad (15)$$

This is the maximum residual angle after erroneously estimating the sign, and should be within the convergence bound, that is[3]

$$10^{-t}\ 10^{-\lceil \frac{i}{4} \rceil} + \alpha_{i,1} \leq \sum_{j=i+1}^{4m} \alpha_{j,1} + \alpha_{4m,1}$$

Therefore, for convergence, the resulting condition is

$$10^{-t} 10^{-\lceil \frac{i}{4} \rceil} \leq V[i] = \left(\sum_{j=i+1}^{4m} \alpha_{j,1} + \alpha_{4m,1} \right) - \alpha_{i,1}$$

where $V[i]$ is the overlap between angle i and the addition of the remaining angles plus the bound of the final

3. We consider an infinite wordlength datapath to find t; therefore the datapath width should have enough guard digits to ensure the final required bound in the angle.

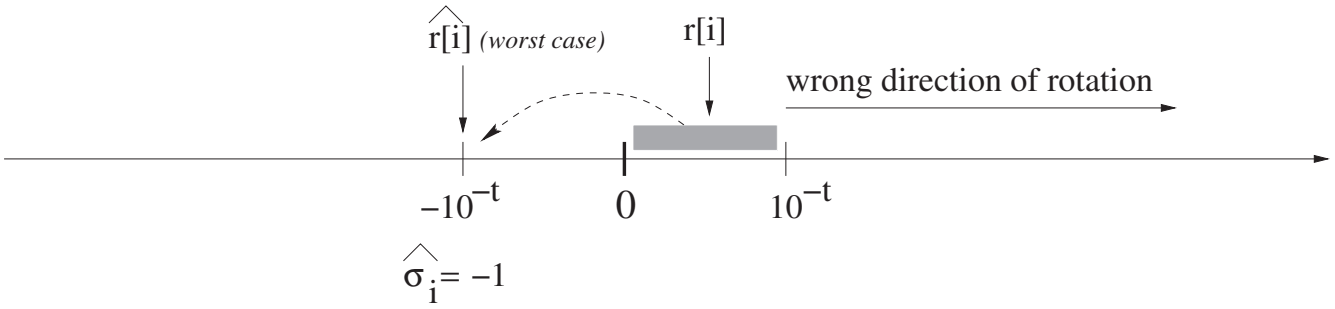

Figure 1. Wrong estimation of the sign of $r[i]$.

error. To obtain the number of digits of the estimation we need a lower bound of the scaled overlap, that is

$$10^{-t} \leq \min(10^{\lceil \frac{i}{4} \rceil} V[i])$$

In [15] (Section 1) we prove that the worst case for convergence (minimum overlap) corresponds to i mod 4 =0; among the possible values of i that verify this condition, the worst case is $i = 4m - 4$. In [15] (Section 1) we also prove the following bound:

$$\min(10^k V[4k]) \geq 10^{m-1} V[4(m-1)] \geq 0.1 + \frac{599}{30}\ 10^{-2m}$$

This condition has to be true to ensure that an erroneous estimation in the last iteration is within the bound of the final error:

$$10^{-t} \leq 0.1 + \frac{599}{30}\ 10^{-2m}$$

From this expression, and for practical values of m, we can select $t = 1$ as a suitable value. Thus, the number of fractional digits of the estimation of the sign of $r[i]$ is one.

On the other hand, the number of digits of the integer part of the estimation can be obtained from the upper bound of $|r[i]|$. In [15] (Section 2) we prove that

$$|r[i]| < \frac{100}{9} - \frac{1}{9}\ 10^{k-m} < \frac{100}{9} = 11.111...$$

Since the truncated value satisfies the bound

$$r[i] - 10^{-t} < \hat{r}[i] \leq r[i]$$

The bound of the estimation is (as determined above, we use $t = 1$)

$$-\frac{100}{9} - \frac{1}{10} < \hat{r}[i] \leq \frac{100}{9}$$

Therefore, the estimation requires two decimal digits, one of one bit and one sign bit (we use a sign bit with weight -20). Thus, a total of 10 bits have to be assimilated to ensure the convergence of the algorithm in redundant carry-save arithmetic: one sign bit, five integer bits and four fractional bits.

3.1.2 Vectoring mode

In the case of having $w[i]$ in the interval $[0, 10^{-t})$, it is possible to have a truncated value of $\hat{w}[i] = -10^{-t}$ and therefore an error occurs in the estimation of the sign of $w[i]$. Thus, instead of having $w[i+1] = w[i] + C[i]x[i]$ we perform an actual microrotation in the wrong direction: $w[i+1] = w[i] - C[i]x[i]$. Accordingly, in the z coordinate we have $z[i+1] = z[i] + \alpha_{i,0}$. The biggest error occurs when the actual value of $w[i]$ is close to 10^{-t}. Thus, considering that in vectoring mode the remainder angle is $\tan^{-1}\left(\frac{y[i]}{x[i]}\right)$ and the change of variable $w[i] = 10^{\lceil i/4 \rceil} y[i]$, an error in the estimation on the sign of $w[i]$ with t fractional digit leads to a residual angle bounded by

$$\tan^{-1}\left(\frac{y[i]}{x[i]}\right) \leq \tan^{-1}\left(\frac{10^{-t}\ 10^{-\lceil i/4 \rceil}}{x[i]}\right) + \alpha_{i,0}$$

Since for the vectoring mode $x[i] \geq 1$, and $\tan^{-1}(x) < x$, a bound for the residual angle results in

$$\tan^{-1}\left(\frac{10^{-t}\ 10^{-\lceil i/4 \rceil}}{x[i]}\right) + \alpha_{i,0} \leq 10^{-t}\ 10^{-\lceil i/4 \rceil} + \alpha_{i,0}$$

which is the same bound in the residual angle obtained for the rotation mode (see (15)). Therefore, as for rotation, an estimation with one fractional digit is sufficient to ensure convergence for the vectoring mode. As for the number of integer digits, the maximum value of $|w[i+1]|$ is obtained when $w[i] = 0$ and $(i+1)$ mod $4 = 3$, resulting in $|w[i+1]| \leq 10\ x[i]$. A bound for $x[i]$ is given by (obtained by taking into account that $|y_{in}| \leq x_{in} < 10$)

$$x[i] \leq \frac{1}{K_1}\ \sqrt{x_{in}^2 + y_{in}^2\ 10^{-2E_{yin}}} < 17.67...$$

Therefore, $|w[i+1]| < 176.7...$, resulting in three decimal digits, one of them of one bit, plus the sign bit. Therefore, for the vectoring mode a total of 14 bits have to be assimilated: one sign bit, nine integer bits and four fractional bits.

3.2 Number of digits of the Estimation for Hyperbolic Coordinates

We analyze the residual angle produced after a wrong estimation of the sign for rotation and vectoring mode and then compare them to determine the worst case.

Given the same assumptions as in the case of circular coordinates and rotation mode (Section 3.1.1), the residual angle produced after a wrong estimation of the sign for the hyperbolic case is

$$10^{-t}\ 10^{-\lceil \frac{i}{4} \rceil} + \alpha_{i,-1} \tag{16}$$

Similar to the circular coordinate case, the estimation error for the vectoring mode on w (scaled value of y) may lead to the following residual angle

$$\tanh^{-1}\left(\frac{10^{-t}\ 10^{-\lceil \frac{i}{4} \rceil}}{x[i]}\right) + \alpha_{i,-1}$$

A lower bound on $x[i]$ is obtained as follows. The x coordinate converges to the hyperbolic modulus ($\sqrt{x_{in}^2 - y_{in}^2}$) scaled by $1/K_{-1}$, and this is the minimum value that the x coordinate may take. On the other hand, the minimum value of the hyperbolic modulus is obtained for the highest possible ratio between the initial values of the y and x coordinates (0.8229.. for a range of convergence of $D_{-1} = 1.166..$). Moreover, $x_{in} \geq 1$. Therefore,

$$x[i] \geq \frac{x_{in}}{K_{-1}}\ \sqrt{1 - \tanh(1.166..)^2} \geq 0.4369...$$

Since $\tanh^{-1}(x) \leq x + (1/2)x^3$, the residual angle for vectoring is bounded by

$$\frac{10^{-t}}{0.4369...}\ 10^{-\lceil \frac{i}{4} \rceil} + \frac{1}{2}\left(\frac{10^{-3t}}{(0.4369...)^3}\right)\ 10^{-3\lceil \frac{i}{4} \rceil} + \alpha_{i,-1}$$

Comparing this expression to the corresponding expression of the rotation mode (16), we observe that the vectoring operation produces the worst case for the residual angle. Thus, we focus on the vectoring mode.

For hyperbolic coordinates with angles derived from the 5221 decimal code, we have verified that it is not possible to ensure the convergence of the algorithm using an estimation with one decimal digit, as done for circular coordinates.

One solution is to use one additional fractional digit for the estimation. We have examined an alternative that allows the use of sign estimations with just one fractional digit for both circular and hyperbolic coordinates. This alternative uses angles derived from the decimal code 5421, which has greater redundancy. Specifically, we show in [15] (Section 3) that convergence is achieved by using the following scheme for hyperbolic coordinates using a one fractional digit sign estimation:

- To use angles derived from the code 5221 for $i \leq 4$, that is, to use the following sequence of angles for the leading four iterations: $\tanh^{-1}(5\ 10^{-1})$, $\tanh^{-1}(2\ 10^{-1})$, $\tanh^{-1}(2\ 10^{-1})$ and $\tanh^{-1}(10^{-1})$.
- To use the code 5421 for $i > 4$, that is angles of the form $\tanh^{-1}(S[i])$, with $S[i] = C[i]\ 10^{-\lceil \frac{i}{4} \rceil}$, and $C[i] = R[i \bmod 4]$ with $R[0] = 1, R[1] = 5, R[2] = 4, R[3] = 2$.

The drawback of the 5421 code is that it is necessary to generate the x4 decimal multiple. This can be done by adding an additional x2 module to perform two consecutive x2 multiples for the iterations that require the x4 scaling.

4 Redundant Floating-point CORDIC algorithm

It is necessary to perform a range reduction and to adapt the algorithm to deal with very small values while maintaining the required accuracy (exceptions and other low-level issues are beyond the scope of our paper) to extend the algorithm. We consider the computation of the following transcendental functions: $\cos(F)$, $\sin(F)$, $\tan^{-1}(F/G)$, $\sinh(F)$, $\cosh(F)$, $\tanh^{-1}(F/G)$, e^F, 10^F, $\ln(F)$, $\log_{10}(F)$ and $\sqrt{F}$ where $F = S_A\ A\ 10^{E_A}$ and $G = S_B\ B\ 10^{E_B}$, with, S_A, S_B the sign bits, $A,\ B \in [1, 10)$ coded in BCD and $E_A,\ E_B$ the exponents. According to the IEEE-754 2008 standard, the input operands do not have to be normalized. However, for transcendental functions, the preferred exponent is the minimum possible, so a normalization stage is necessary.

In [8] we presented the extension of the non-redundant algorithm, described the high-level steps required for the range reduction for each function and showed the resulting CORDIC iterations (in [15] (Section 4) the high-level steps required for range reduction are presnted in full detail). After range reduction, one of the inputs to the CORDIC iterations takes the form $y_{in} = M_{yin}\ 10^{-E_{yin}}$ or $z_{in} = M_{zin}\ 10^{-E_{zin}}$, with $|M_{yin}|, |M_{zin}| \in [1, 10)$ and $E_{yin}, E_{zin} \geq 0$. Therefore, the control variable has absolute terms $P = \max\{E_{yin} - 1, 0\}$ or $Q = \max\{E_{zin} - 1, 0\}$ fractional leading zeros or nines and a total of p significant digits. To produce accurate results it is necessary to modify the fixed point iteration to move P or Q leading zeros or nines from the computation of the corresponding y or z iteration (or both). We used a similar approach to [16] (floating-point binary CORDIC). We scale the y and/or z iterations by 10^P or 10^Q, and start the iterations at the index J ($J = 4Q+1$ for rotation and $J = 4P + 1$ for vectoring [8]), so that the input argument is within the range of convergence covered from that iteration. For the carry-save algorithm we have to take into account that the z (rotation mode) or y (vectoring mode) iterations are scaled by $10^{\lceil i/4 \rceil}$ to have the digits analyzed for the estimation in a fixed position. The resulting equations for the carry-save algorithm are the following (to simplify the presentation we use the same variable names for the x, y, r, w and z iterations as before):

Rotation Mode

Initialization: $x[1] = x_{in}$, $y[1] = y_{in}$, and $r[1] = 10z_{in}$

Iteration: for $i = 1$ to $4m$

$\sigma_i = \text{sign}(\hat{r}[i])$

$$x[i+1] = x[i] -\ c\ \sigma_i\ C[i]\ 10^{-\lceil \frac{i}{4} \rceil - T}\ y[i]$$

$$y[i+1] = y[i] +\ \sigma_i\ C[i]\ 10^{-\lceil \frac{i}{4} \rceil - U}\ x[i]$$

$$r[i+1] = \begin{cases} 10(r[i] - \sigma_i\ A_{i-1+J,c}) & \text{if } i \bmod 4 = 0 \\ r[i] - \sigma_i\ A_{i-1+J,c} & \text{if } i \bmod 4 \neq 0 \end{cases}$$

The values of T and U determine the shifts required after scaling the iterations, and these depend on the function computed: $T = 2\ Q$ and $U = 0$ for sin, cos, reduced range sinh and cosh (with an absolute value of the input range less then or equal to $\ln(10)/2$), and $T = U = Q$ for sinh and cosh with large ranges.

Vectoring Mode

Initialization: $x[1] = x_{in}$, $w[1] = 10y_{in}$, and $z[1] = z_{in}$

Iteration: for $i = 1$ to $4m$

$\sigma_i = \text{sign}(\hat{w}[i])$

$$x[i+1] = x[i] - c\ \sigma_i\ C[i]\ 10^{-2\lceil \frac{i}{4} \rceil - T}\ w[i]$$

$$w[i+1] = \begin{cases} 10(w[i] + \sigma_i\ C[i]\ x[i]) & \text{if } i \bmod 4 = 0 \\ w[i] + \sigma_i\ C[i]\ x[i] & \text{if } i \bmod 4 \neq 0 \end{cases}$$

$$z[i+1] = z[i] - \sigma_i\ A_{i-1+J,c}\ 10^{-\lceil \frac{i}{4} \rceil}$$

with $T = 2P$ for $\tan^{-1}$ and $\tanh^{-1}$ (and $\ln$ and $\log_{10}$), and $T = 0$ for square root. Note that although the iteration index i takes values in the range 1 to 4m, the rotation angles used start at index J due to the scaling performed.

4.1 Number of Iterations and Datapath Width

For our implementation, we want normalized significand results in the interval $[1, 10)$ with p decimal digits and an accuracy of one ulp (a maximum error of $\pm 10^{-(p-1)}$ in the significand). We assume a datapath with one integer digit and B fractional digits. We shall discuss the computation of the sine. To have most of the results normalized and to avoid having two guard digits, we scale the input vector by 10. For the computation of the sine, we have the following sources of error (we use errors relative to 10^{-Q}, i.e. the decimal weight to the left of the most significand digit of the input argument):

Approximation error of the angle: for $4m$ iterations a bound for the error in the angle is given by the addition of the last elementary angle plus the estimation error of the sign of r (in the last iteration). For the last elementary angle we use the bound 10^{-m}. For the estimation error of the sign of r in the last iteration, we take the bound $(1/10)\ 10^{-m}$, since the estimation is performed using one fractional digit of r. The resulting error bound after $4m$ iterations is $\epsilon = \pm((11/10)\ 10^{-m}$. The corresponding error in the sine $|\sin(z_{in}) - \sin(z_{in} + 10^{-Q}\ \epsilon)|$ can be bounded by ϵ (relative error), since the error (expressed absolute terms) is very small compared to z_{in}.

Truncation error in the x/y datapath: we assume two truncation errors (with a bound of $2\ 10^{-B}$ for each truncation error of a carry-save operand) in each coordinate per iteration. Therefore, the total truncation error in each one of the x/y datapaths is bounded by $4m \times 2 \times 2\ 10^{-B} = 16\ m\ 10^{-B}$.

Rounding error and guard digits: the result of the computation is rounded, resulting in an error bounded by $\pm 0.5\ 10^{-(p-1)}$. Moreover, the result may need to be normalized before rounding by one decimal left shift, and thus the approximation and truncation errors are multiplied by 10.

The resulting expression for the bound of error should be less than one ulp. Therefore, adding up all the previous sources of error results in the following condition:

$$0.5\ 10^{-(p-1)} + 10 \times \left(\frac{11}{10}\ 10^{-m} + 16m\ 10^{-B} \right) < 10^{-(p-1)}$$

which is simplified to $16\ m\ 10^{-B} + (11/10)\ 10^{-m} < 0.5\ 10^{-p}$. For $B = \infty$ the lower bound of m is $m = p+1$. We take $m = p+1$, resulting in $B > p+3.15...$ Therefore we need to perform $4m = 4(p+1)$ elementary rotations, with a datapath of $p+5$ decimal digits (one integer and $B = p+4$ fractional). Analysis of other functions shows that these parameters also ensure one ulp accuracy.

An overflow problem may arise when computing the arctangent of a quotient for a datapath with only one integer digit. This is due to the function being computed in the vectoring mode for circular coordinates. In this case the x coordinate converges to the scaled value of the magnitude of the input vector, which may take values greater than 10. To avoid this problem, the input x and y coordinates are scaled by 10^{-1} (note that there is no loss of information due to the downscaling since the datapath has some guard digits). This operation does not modify the value of the arctangent of the quotient, but avoids the overflow of the the x coordinate (note that we do not need to correctly scale the modulus of the vector).

5 Improving Latency with Fast Termination

An interesting optimization is to use a fast termination for the CORDIC algorithm [17]. This scheme is based on the fact that after a certain number of elementary rotations, the approximations $\cos(\alpha) = 1$ and $\sin(\alpha) = \alpha$ can be used, where α is the remainder angle to be rotated (similar approximations can be used for hyperbolic coordinates). Termination consists in performing a multiplication–add for rotation (for the coordinate of interest; for instance, for the cosine computation, y is multiplied by z and then added to x) and a division–add operation for vectoring (in this case y is divided by x and the resulting quotient added to z). The key idea is that the multiplication or division operation involved requires fewer iterations than the corresponding CORDIC iterations. This is because current implementations of multiplication and division use radix-10 and therefore perform one iteration per digit (plus some initial overhead), whereas the CORDIC algorithm needs four iterations per digit. Thus, the termination should lead to a reduction in the total number of cycles.

We now calculate the number of CORDIC iterations required before fast termination as well as the number of digits involved in the final termination process.

Let us denominate $4m'$ as the number of CORDIC iterations needed before fast termination such that the desired degree of precision is achieved. Therefore, we perform the CORDIC iterations from J to $(J + 4m' - 1)$. The error condition used is that the relative error of

the approximation of the function should be less than $10^{-(p+1)}$ (similar to the error in the approximation of the angle using all the CORDIC iterations). This means that the allowable absolute error of the approximation depends on the function computed:

- When the result is obtained in the x iteration, the absolute allowable error is $10^{-(p+1)}$, since this iteration is not scaled.
- When the result is obtained in the y (z) iteration, the absolute allowable error is $10^{-P-(p+1)}$ ($10^{-Q-(p+1)}$), since this iteration is scaled by 10^P (10^Q).

We determined m' (this determines the number of CORDIC iterations before termination) for the representative cases of cosine, sine and arctangent:

Cosine: the result is obtained in the x iteration. It is well-known [9] that the absolute value of the bound of the remainder angle after iteration j is the last rotated angle $\alpha_j = \tan^{-1}(10^{-(J+4m'-1)/4}) = \tan^{-1}(10^{-(Q+m')})$. Taking into account that for small α, $\cos(\alpha) \approx 1-\alpha^2/2+...$ we may use the linear approximation to the rotation by the remaining angle when $\alpha^2/2$ is less than the allowable error of the approximation (i.e. the value of $\cos(\alpha)$ is approximated by 1). On the other hand, we know the bound $\tan^{-1}(a) < a$. Therefore, the condition for an accurate rapid termination is:

$$\frac{1}{2}(\tan^{-1}(10^{-(Q+m')})^2 < \frac{1}{2}(10^{-(Q+m')})^2 < 10^{-(p+1)}$$

Thus, the following condition is sufficient:

$$m' \geq \frac{p}{2} - Q + 1$$

This means that in the worst case ($Q = 0$), about half of the CORDIC iterations are needed before performing the linear approximation. This number of iterations decreases as Q (E_{zin}) increases. The linear approximation is performed by multiplication, using the value of z obtained after the $4m'$ CORDIC iterations. Due to the convergence of the algorithm, the remaining angle after the CORDIC iterations has $p - m' + 2$ significant digits. Therefore, for the linear approximation we need a decimal fixed-point multiplication by $p - m' + 2$ digits (with one iteration per digit) plus a fixed-point addition.

A similar result is obtained for the cosh and the exponential functions (base e and base 10).

Sine: the result is obtained in the y iteration. Taking into account that for small α, $\sin(\alpha) \approx \alpha - \alpha^3/3+ ...$ and $\alpha^3/3 < \alpha^2/2$, we may use the linear approximation to the rotation by the remaining angle when $\alpha^2/2$ is less than the allowable bound of error for the approximation. Therefore, the resulting condition for this case is

$$\frac{1}{2}(\tan^{-1}(10^{-(Q+m')}))^2 < 10^{-Q-(p+1)}$$

Proceeding as before, this results in $m' \geq \frac{p}{2} - \frac{Q}{2} + 1$. As before, the linear approximation requires a fixed-point decimal multiplication by $p - m' + 2$ digits plus a fixed-point addition. A similar result is obtained for sinh.

Arctangent: the result is obtained in the z iteration. After a certain number of CORDIC iterations the remaining angle can be approximated by $\tan^{-1}(q) = q - q^3/3+ ....$ The remaining angle is the arctangent of the quotient q between the values of the y and x iterations (note that we actually implement the scaled iteration w). We may use the linear approximation to the rotation by the remaining angle when $q^3/3$ is less than the allowable bound for the error of approximation. After $4m'$ CORDIC iterations, the y coordinate is bounded by (worst case obtained when $y[4m'-1] = 0$):

$$y[4m'] \leq 10^{-(J+4m'-1)/4}\ x[4m'-1] = 10^{-(P+m')}\ x[4m'-1]$$

Therefore,

$$q = \frac{y[4m']}{x[4m']} \leq 10^{-(P+m')}\frac{x[4m'-1]}{x[4m']}$$

From CORDIC theory, we know that for circular coordinates $x[4m'-1] \leq x[4m']$ [9]. Moreover, due to scaling to avoid overflow in the x coordinate, we know that $x[4m'] \geq 0.1$. Therefore, the resulting condition for this case is

$$\frac{1}{3} \times (10^{-(P+m')}))^3 < 10^{-P-(p+1)}$$

Proceeding as before results in $m' \geq \frac{p}{3} - \frac{2P}{3} + 1$. This means that in the worst case about one-third of the CORDIC iterations are needed before performing the linear approximation. Due to the convergence of the CORDIC iterations, the y coordinate has $p - m' + 2$ significant digits after 4m' iterations. Therefore, for the linear approximation, we need a decimal fixed-point division to obtain $p + m' - 2$ quotient digits, plus an addition.

A similar result is obtained for hyperbolic arctangents and logarithms.

6 Architecture

In this section we present an example of the architecture needed to implement the algorithm. Since the frequency of transcendentals is usually low, we mapped the algorithm onto a low-cost unit that can be shared with other operations. We avoid using the decimal multiplier and use the decimal carry propagate adder only at the end the computation for conversion from carry-save to standard BCD. We propose a unit based on a single 3-2 decimal carry-save adder, x2, x4 and x5 scalers (which are very simple for decimal) and the reuse of the rotator that is required for the initial alignments for other floating-point operations (which in current implementations is not used intensively). This unit could be used to implement the division operation as described in [3] (based on prescaling with selection by rounding).

Figure 2 shows the resulting architecture. We pipelined the architecture to schedule the three iterations of the algorithm using the same datapath. The pipeline consists of two stages for the rotator, one stage for x5, x4 (performed as x2 x2), x2, x1 scalers and carry-save addition,

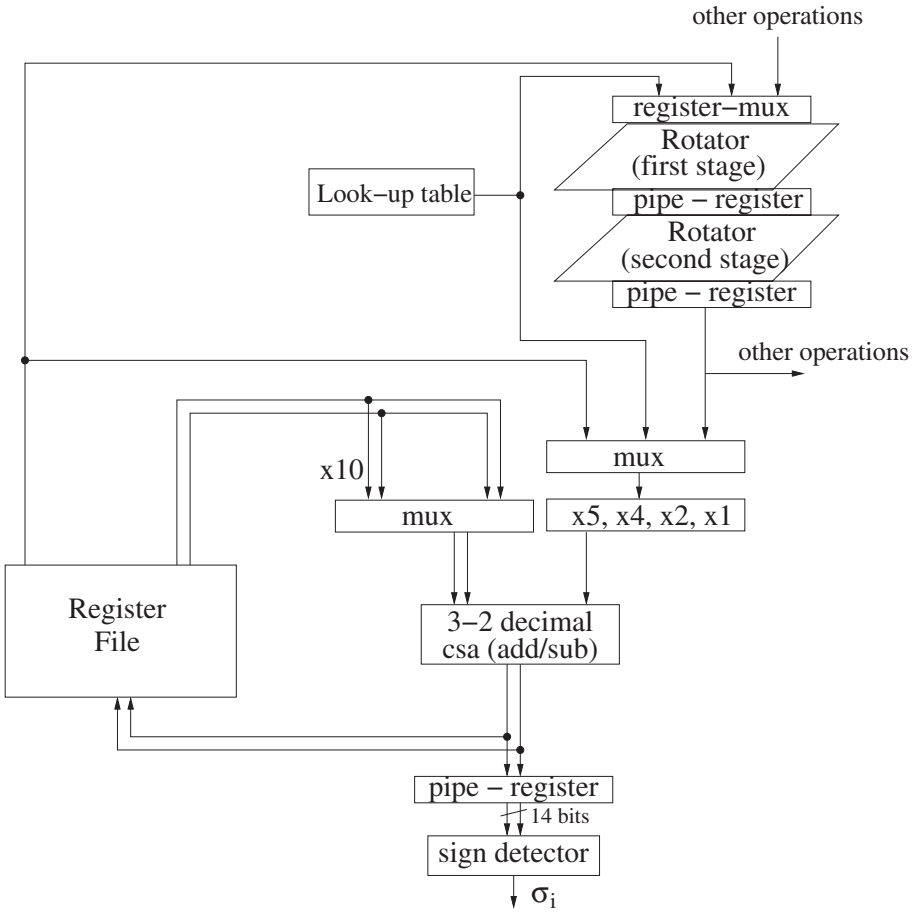

Figure 2. Pipeline for mapping the CORDIC algorithm.

and one stage for the sign detector to obtain the values of σ_i. For the pipelining we have taken into account the reference data provided for the Power 6 implementation [2], where a rotator requires two cycles and a decimal carry-propagate addition of 144 bits (36 decimal digits) also requires two cycles.

Figure 3 shows the pipelining of the algorithm for the rotation mode. For each of the x and y iterations we need to perform two shifting operations and two carry-save additions due to the carry-save representation of the operands. The r iteration requires only one carry-save addition since the angles are represented in non-redundant form; furthermore, these iterations do not require the use of the rotator. This scheduling requires five cycles per CORDIC iteration, achieving a full utilization of the carry-save adder. Sign detection is required every five cycles, with a maximum latency for the sign detector of four cycles.

Figure 4 shows the pipelining for the vectoring mode. In this case shifting is necessary for the x iteration (it requires two shifts for the carry-save value of w) and the z iteration (only one shift, since the angle is represented in non-redundant form). As before, the x and w iterations require two carry-save additions. Iteration of z requires only one carry-save addition. Sign detection is performed every five cycles, but in this case the maximum allowable latency is three cycles. Therefore, the area/power characteristics of the sign detector can be optimized for a latency of three cycles to meet the timing requirements of both two modes of operation. The latency per CORDIC iteration is also of five cycles for the vectoring mode.

6.1 Latency of Computation

In this subsection we determine the number of cycles required for computing the different elementary functions after range reduction. Each iteration is completed every five cycles. For $4m$ elementary rotations, a total of $20m+2$ cycles are needed (two cycles for pipeline filling). For instance, for $p = 34$ (Decimal128 format) $m = p+1 = 35$, resulting in 702 cycles.

To reduce latency, we may use the linear approximation scheme (fast termination). In this study, we assume that the multiplication and the division operations needed for the linear approximation are implemented using the same unit used for the CORDIC iterations. For multiplication, the extra hardware required is a one-digit recoder, using the carry-propagate adder for odd multiple generation (i.e. x3). As mentioned, the division algorithm could also be implemented within the same unit. The multiplication operation can perform

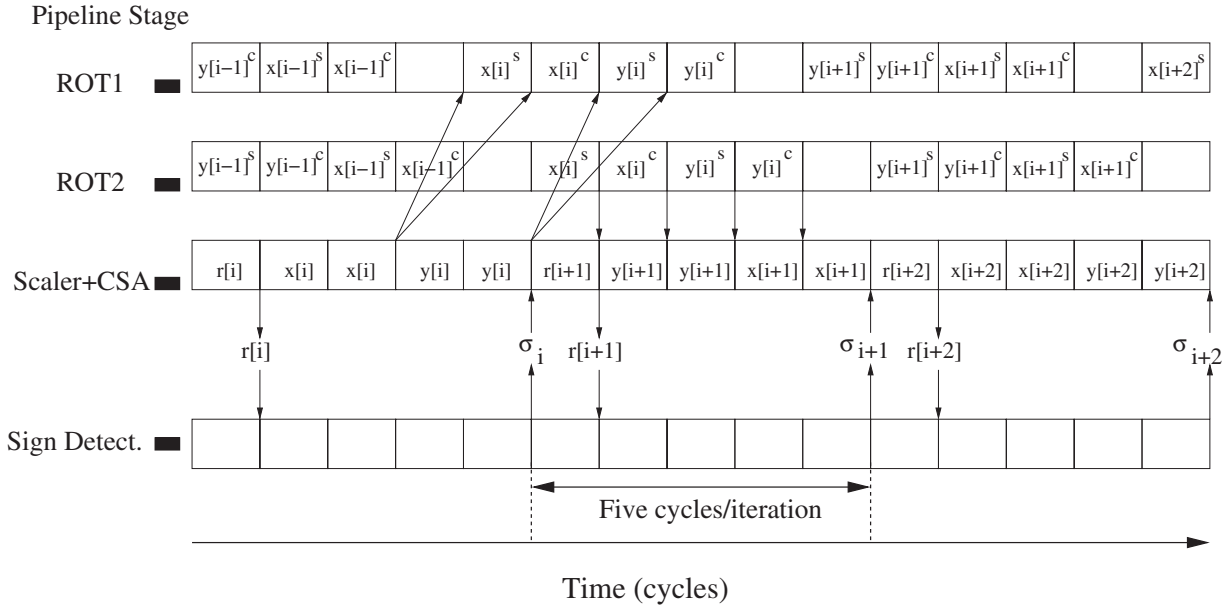

Figure 3. Pipelining with a redundant adder for the rotation mode (superscript s (c) indicates sum word (carry word) of the carry-save representation).

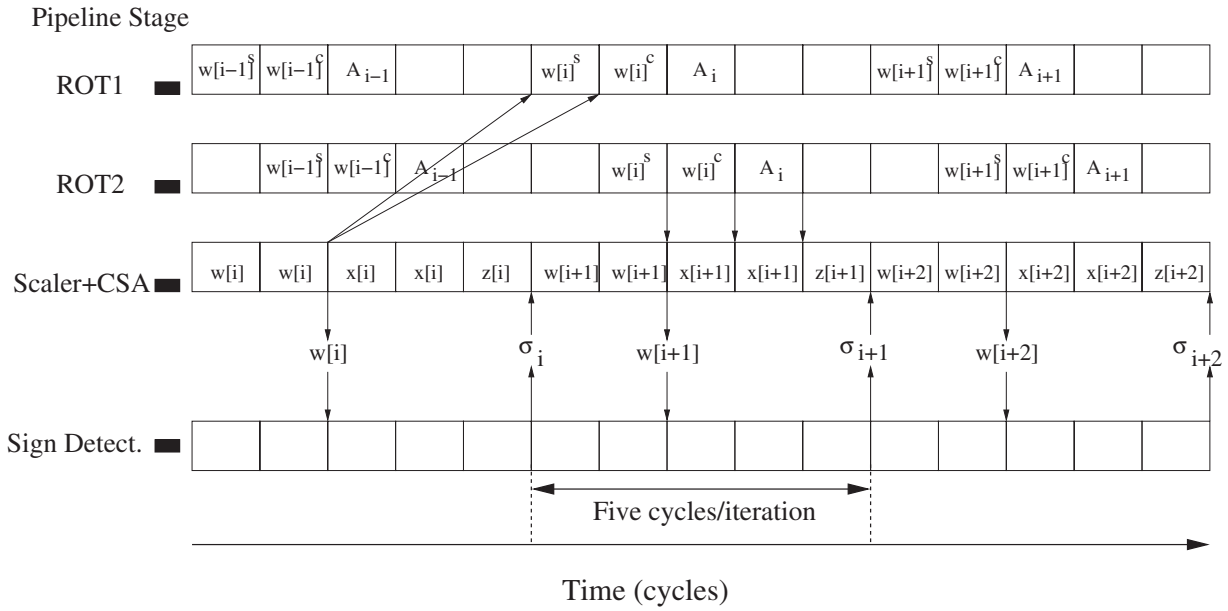

Figure 4. Pipelining with redundant adder for the vectoring mode (superscript s (c) indicates sum word (carry word) of the carry-save representation).

multiplication of one digit per cycle, plus some initial overhead. We assume that a fixed-point multiplication requires $T + 4$ cycles for a multiplier of T digits; for fixed-point division using a carry-save adder we assume $13 + 2T$ cycles to obtain T quotient digits (based on the implementation suggested in [3]). Moreover, four additional cycles are required, two for a carry-save addition to complete the linear approximation and two for the conversion to BCD.

Some operations are required to set up the input operands for the CORDIC iterations for some functions after range reduction. This requires a few cycles (we call Δ to the number of overhead cycles).

Therefore, the number of cycles for the functions that perform a linear approximation in the rotation mode (cos, sin, cosh, sinh and exponential (base e and 10)) is ($4m$ is the number of CORDIC rotations before the linear approximation):

$$\Delta + 2 + 5 \times 4m + 4 + (p - m + 2) \times 1 + 4$$

For the worst case ($Q = 0$) $m = p/2 + 1$, and the number of cycles results in $(21/2)p + 31 + \Delta$. The value of Δ is zero for sin, cos and exponentials, and 7 cycles for cosh and sinh (see [15], Section 4).

Table 1
Number of cycles for the different transcendental functions computed(without pre- and post-processing due to range reduction).

Function	sin, cos, e^a, 10^a	sinh, cosh	$\tan^{-1}(a/b)$	$\tanh^{-1}(a/b)$	ln, $\log_{10}$	sqrt
# cycles* (p=16)	200	205	170	175	170	190
# cycles* (p=34)	390	395	315	320	315	370

* Rounded values to multiples of five.

For the functions that perform the linear approximation in the vectoring mode (arctangent, hyperbolic arctangent and the logarithms), the number of cycles is:

$$\Delta + 2 + 5 \times 4m + 13 + (p - m + 2) \times 2 + 4$$

For the worst case ($P = 0$) $m = p/3 + 1$, and the number of cycles results in $8p + 41 + \Delta$. The value of Δ is zero for arctangent, 7 for hyperbolic arctangent and 3 for logarithms (see [15], Section 4).

For square root, the result corresponds to the hyperbolic modulus of the vector obtained in the vectoring mode. The modulus converges faster than the angle, and does not require final linear approximation. Therefore, for the square root case, the number of cycles is $\Delta + 2 + 5 \times 4m + 2$, with $m = p/2 + 1$ and $\Delta = 4$.

Table 1 shows the total number of cycles (without the operations related to range reduction) required for the computation of the different functions for $p = 16$ and 34. For $p = 16$ the number of cycles ranges from 170 to 205. For $p = 34$ the range is 315 to 395 cycles.

7 Comparison

In this section we compare our proposal to table-driven polynomial methods to compute transcendental functions. We also compare our study to previous works on decimal CORDIC, focusing on mapping the algorithms to a state-of-the-art DFPU.

Table-driven polynomial methods are a popular alternative for transcendental function evaluation. Thus, it is relevant to obtain an estimate of the latency and look-up table size required using the same hardware unit as the proposed CORDIC algorithm. We present the results for the Decimal128 format (p=34) in Table 2. We use the approximation error of a Chebyshev polynomial as a reference (the optimum polynomial is obtained by the minimax approximation, but the result should be close), evaluate the polynomial by using Horner's rule and consider optimizations based on the significance of each of the coefficients of the polynomial, which allow reducing storage space and the number of cycles for the multiplications. Table 2 shows that for one-argument functions using two digits to obtain the coefficients of the polynomial, a 10th-degree polynomial is required, resulting in about 290 cycles of latency (using the carry-save adder to perform the multiplications) and storage of about 103 Kbits per function. An alternative with far fewer storage requirements (about 15.5 Kbits per function) is obtained by using one digit to obtain the coefficients of the polynomial (requiring a 14th-degree polynomial), resulting in a latency of about 450 cycles. In contrast, our implementation requires 315-390 cycles with storage of about 32 Kbits for several functions (some of which have two arguments).

Table 2
Comparison with table-driven polynomial methods

Method	Cycles	Memory (Kbits)
This work (several functions)	315-390	32
10^{th} degree polinomial (one function of one argument)	290	103
14^{th} degree polinomial (one function of one argument)	450	15.5

Decimal CORDIC for pocket calculators were proposed in [18] [19] [20]. These implementations used the values $\tan^{-1}(10^{-j})$ as rotation angles. To ensure convergence, each elementary rotation must be repeated 9 times. Therefore, the algorithm requires $9m$ iterations for the rotations, a factor of 2.25 more iterations than our algorithm. In contrast, it does not require the x2 and x5 multiples.

A recent proposal [22] applies the standard base 2 CORDIC algorithm (with elementary rotation angles $\tan^{-1}(2^{-j})$) to operands coded in decimal. They have to implement the multiplication by x2^{-j} for each iteration, which is very complex for decimal BCD operands.

Another recent proposal for implementing transcendental functions using a decimal digit-by-digit algorithm was proposed in [21]. The algorithm is an extension to radix 10 of the binary BKM algorithm, and it is based on the computation of complex logarithms and exponentials. The iterations of the algorithm are the following:

$$\begin{aligned}
S[i+1]^x &= 10^{1-h}(S[i]^x + (1-h)d_i^x + \\
&+ (d_i^x S[i]^x - d_i^y S[i]^y)\ 10^{-i}) \\
S[i+1]^y &= 10^{1-h}(S[i]^y + (1-h)d_i^y + \\
&+ (d_i^x S[i]^y + d_i^y S[i]^x)\ 10^{-i}) \\
T[i+1]^x &= 10^h(T[i]^x - Log_Const(d_i^x, (d_i^x)^2, (d_i^y)^2)) \\
T[i+1]^y &= 10^h(T[i]^y - Atan_Const(d_i^x, d_i^y))
\end{aligned}$$

where d_i^x, d_i^y take values in $\{-6, -5, .., 0, ..., 5, 6\}$, Log_Const and Atan_Const are stored constants that depend on the values of d_i^x and d_i^y. The values of d_i^x

and d_i^y are obtained by rounding the value of $S[i]^x$ and $S[i]^y$, or $T[i]^x$ and $T[i]^y$, depending on the mode of operation. Moreover, $h = 1$ for the E-mode, and 0 for the L-mode. The algorithm has a reduced convergence domain, so some range reduction computations are needed to have a range of convergence comparable to our algorithm. They require one iteration per decimal digit of the input operands, such that m iterations are required. The main drawbacks of this algorithm are the following:

- Although this algorithm requires 1/4 of the iterations of our algorithm (in fact about p vs. $2.5p$ iterations considering the optimization of fast termination with radix-10 multiplication and division), mapping the iteration is more complex. Specifically, they have two iterations with three additions and four multiples have to be generated on the fly, which can take values x0, x1, x2, x3 (requires an addition), x4 (computed as x2 x2), x5, x6 (x2 x3). Therefore, in the worst case they require four additions per iteration for S^x and S^y. Moreover they have two iterations to accumulate constant values. Thus, in the worst case, they require 10 additions for each digit iteration. Our algorithm requires 12 for a group of 4 bit-by-bit iterations.
- The algorithm presents certain irregularities that make mapping more difficult: some initial computations must be performed to reduce the range such that the selection of digits can be done by rounding.
- The main limitation of the algorithm is the large number of constants that need to be stored. We estimated that they require about 380 Kbits in BDC (317 Kbits in DPD) of look-up table (comparable to the size of a level 1 cache) in comparison to 14 Kbits in BCD (11.7 in DPD) for our algorithm (a ratio of 27), both for the fixed-point case.

8 Conclusions

We present an efficient alternative to provide support, in state-of-the-art decimal hardware, for transcendental functions for the IEEE-754 2008 floating-point decimal number system in DPD format. We presented a new decimal floating-point redundant CORDIC algorithm for the computation of transcendental functions. As with standard binary CORDIC, the proposed algorithm has a constant scale factor and the number of iterations corresponds to the number of bits of the input operands. We demonstrate the convergence of the redundant algorithm for all modes of operations, allowing computation of a rich set of elementary functions. We map the algorithm to a low-cost unit based on a decimal 3-2 carry-save adder, which can be shared with a floating-point decimal division algorithm. We show that our design is more efficient for mapping onto a state-of-the-art DFPU than previous designs. A comparison with table-driven methods shows that our approach allows significant improvement in latency and/or storage requirements. In addition to the immediate practical use of the algorithm, the paper also contributes to expand CORDIC algorithm theory.

References

[1] IEEE, "IEEE Standard for Floating-Point Arithmetic," IEEE Std 754-2008 , Aug. 29 2008.
[2] L. Eisen et al., "IBM Power 6 Accelerators: VMX and DFU", IBM J. Res. and Dev., vol 51, no 6, pp. 663–684, Nov. 2007.
[3] E.M. Schwarz and S.R. Carlough, "Power 6 Decimal Divide", in Proc. IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 128-133, 2007.
[4] E.M. Schwarz et al., "Decimal Floating-Point Support on the IBM System Z10 Processor", IBM J. Res. and Dev., vol 53, no 1, 2009.
[5] M. Cornea, J. Harrison, C. Anderson, P.T.P. Tang, E. Schneider, E. Gvozdev, "A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format," IEEE Transactions on Computers, vol. 58, no. 2, pp. 148-162, Feb. 2009.
[6] J. Harrison, "Decimal Transcendentals Via Binary", in 19th IEEE Symposium on Computer Arithmetic, pp. 187–194, 2009.
[7] A. Vázquez and E. Antelo, "Constant Factor CORDIC Algorithm for Decimal BCD Input Operands", 8th Real Numbers and Computers Conference, Santiago de Compostela, July 2008.
[8] A. Vázquez, J. Villalba and E. Antelo, "Computation of Decimal Transcendental Functions Using the CORDIC Algorithm", in Proc. 19th IEEE Symposium on Computer Arithmetic, pp. 179–186.
[9] M. Ercegovac and T. Lang, "Digital Arithmetic", Morgan Kaufmann, 2004.
[10] R.K. Richards, "Arithmetic Operations in Digital Computers", D. Van Nostrand Co., NY, 1955.
[11] S. Gorgin and G. Jaberipur, "Fully Redundant Decimal Arithmetic", in Proc. 19th IEEE Symposium on Computer Arithmetic, pp. 145–152, 2009.
[12] J.M. Muller, "Elementary Functions: Algorithms and Implementation". Birkhauser Verlag AG, second edition, 2007.
[13] N. Takagi and T. Asada and S. Yajima, "Redundant CORDIC Methods with a Constant Scale Factor", IEEE Transactions on Computers, pp. 989-995, vol. 40, num. 9, 1991
[14] J. A. Lee and T. Lang, "Constant-factor redundant CORDIC for angle calculation and rotation", IEEE Transactions on Computers, pp. 1016-1025, vol. 41, num. 8, 1992.
[15] A. Vázquez, J. Villalba, E. Antelo and E.L. Zapata, "Calculations for Redundant Floating-point Decimal CORDIC Algorithm". Internal Report, University of Santiago de Compostela (www.ac.usc.es/system/files/additional_material_CORDIC4.pdf), 2011.
[16] J.S. Walther, US patent 3766370: "Elementary Floating Point CORDIC Function Processor and Shifter", filing date May 14, 1971, issue date Oct 16, 1973.
[17] H.M. Ahmed, "Efficient Elementary Function Generation with Multipliers", in Proc. 9th IEEE Symposium on Computer Arithmetic, pp. 52–59, 1989.
[18] J.E. Meggitt, "Pseudo Division and Pseudo Multiplication Processes", IBM Journal, pp. 210-226, April 1962.
[19] D.S. Cochran, "Algorithms and Accuracy in the HP-35", Hewlett-Packard Journal, pp. 10-11, 1972.
[20] H. Schmid, "Decimal Computation", John Wiley & Sons, 1974.
[21] L. Imbert, J.M. Muller and F. Rico, "A Radix-10 BKM Algorithm for Computing Transcendentals on Pocket Computers", Journal of VLSI Signal Processing Systems, Vol. 25, no 2, June 2000, pp. 179–186.
[22] A. Jimeno et al., "A BCD-based architecture for fast coordinate rotation", Journal of Systems Architecture, vol 53, no 8, pp. 829–840, 2008.

Alvaro Vazquez graduated in Physics in 1997 and received the PhD. degree in Electronic and Computer Engineering in 2009 from the University of Santiago de Compostela, Spain. In 1998 he joined the Departamento de Electronica e Computacion at the University of Santiago de Compostela, where he worked on different research projects in the Computer Architecture Group. From October 2009 to March 2011 he was an INRIA postdoctoral fellow at the Laboratoire de l'Informatique du Parallelisme, ENS Lyon, France. His research interests include different aspects of computer arithmetic and computer architecture, such as decimal floating-point arithmetic, design of high-speed and low-power numerical processors, FPGA-specific floating-point operators for reconfigurable computing, and algorithms and architectures for computing elementary functions. Dr. Vazquez is member of the IEEE and the IEEE Computer Society.

Julio Villalba-Moreno received the BS degree in Physics in 1986 (Univ. of Granada, Spain) and the PhD in Computer Engineering in 1995 (Univ. of Malaga, Spain). From mid 1986 to late 1991 he worked as a design engineer in the Research and Development Department of Fujitsu Spain ($R\&D$ Digital Signal Processing group) and was an assistant professor in the University of Malaga. Since 2007, he has been a Full Professor in the Department of Computer Architecture at the Univ. of Malaga. He was a research visitor at the Department of Electrical Engineering and Computer Science of the University of California at Irvine in 1996, 2003, 2004 and 2005 for a total amount of one year. Since 2006 he belongs to the Program Committee of the IEEE Symposium on Computer Arithmetic. From July 2011 he is an Associated Editor of IEEE Transaction on Computers. His research interest includes computer arithmetic and specific application architectures.

Elisardo Antelo Elisardo Antelo graduated with a degree in Physics in 1991 and received the Ph. D. in computer engineering in 1995 from the University of Santiago de Compostela, Spain. In 1992, he joined the Departamento de Electronica e Computacion at the University of Santiago de Compostela. From 1992 to 1998, he was an assistant professor and, since 1998 he has been a tenured associate professor in this Department. He was a research visitor at the University of California at Irvine several times between 1996 and 2000. Dr. Antelo is a member of the Computer Architecture group at the University of Santiago de Compostela. Since 2001, he has been involved in the Program Committee of the IEEE Symposium on Computer Arithmetic (program cochair in the 2011 edition). He also was involved with the Program Committees of the Real Numbers and Computers Conference since 2006 and EUROSIPCO since 2008. He is Associate Editor of the IEEE Transactions on Computers (since 2007), and of Integration, the VLSI Journal (since 2011). His primary research and teaching interest are in digital design and computer architecture with current emphasis on high-speed and low-power numerical processors, application-specific modules, computer arithmetic and design issues related to multi-core processors. Dr. Antelo is member of the IEEE and the IEEE Computer Society.

Emilio L. Zapata has got his degree in Physics from the University of Granada in 1978, and his Ph.D. in Physics from the University of Santiago de Compostela in 1983. From 1978 to 1982 he was assistant professor at the University of Granada. In 1982 he joined the University of Santiago de Compostela where he becames full professor at 1990. Since 1991 he is full professor at the University of Malaga. Currently he is head of the Computers Architecture Department, at the University of Malaga, Spain.

Dr. Zapata has published over 90 journal and 200 conference papers on the parallel computing field (applications, compilers and architectures). His main research topics include: application fields, compilers, computer arithmetic and application-specific array processors. Dr. Zapata is a member of the Editorial Board of the Journal of Parallel Computing and Journal of Systems Architecture. He has also been a guest editor of special issues of the Journal of Parallel Computing.