Computation of Decimal Transcendental Functions Using the CORDIC Algorithm*

Álvaro Vázquez, Julio Villalba** and Elisardo Antelo
University of Santiago, SPAIN
**University of Málaga, SPAIN
alvaro.vazquez@usc.es, jvillalba@uma.es, elisardo.antelo@usc.es

Abstract

In this work we propose new decimal floating-point CORDIC algorithms for transcendental function evaluation. We show how these algorithms are mapped to a state of the art Decimal Floating-Point Unit (DFPU), both considering the use of a carry–propagate adder or a carry–save redundant adder. We compared with previous decimal CORDIC proposals and with table-driven algorithms, and we concluded that our approach have significant potential advantages for transcendental function evaluation in state of the art DFPUs with minor modifications of the hardware.

1. Introduction

Decimal floating-point is now a hot topic or research. Business applications have motivated a standard for the representation and processing of floating point decimal numbers (included in the IEEE 754–2008 floating-point standard), and the interest by microprocessor companies to include support for floating-point decimal processing. In this work we concentrate on BCD hardware implementations. A representative state of the art industrial unit provides support for floating point addition/subtraction, multiplication and division [1] [2]. Multiplication and division are performed iteratively digit-by-digit. Floating-point multiplication requires 89 cycles in the worst case for the format Decimal128. Therefore, it is very slow compared to state of the art binary floating-point implementations. In this work we are interested in the implementation of floating-point decimal transcendental functions. In [3] we considered the CORDIC algorithm for the implementation of decimal transcendentals. Specifically we proposed a constant scale factor fixed-point algorithm for computing trigonometric functions. CORDIC was used by low cost processors such as calculators, and by microprocessors when the multiplication hardware was slow (serial). 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.

In this work we extend the algorithm proposed in [3] to hyperbolic coordinates. We also extend the algorithm to deal with floating-point numbers. We map the algorithm in a state of the art DFPU unit, using a carry propagate adder, and develop the redundant version of the algorithm, to use a carry-save adder instead of the complex carry propagate adder. Early implementations of CORDIC for decimal operands can be found in [6] [4] [8] among others. More recent related implementations can be found in [5] [7]. In our proposal the number of iterations is equal to the number of bits of the input operands (each decimal digit requires four bits), keeping the CORDIC scale factor constant and the operations of each iteration simple. These are unique features not present simultaneously in previous proposals.

The paper is organized as follows. Section 2 briefly describes the CORDIC algorithm. Section 3 briefly reviews our previous work on decimal CORDIC [3] and propose the extension to hyperbolic coordinates. In Section 4 we demonstrate the convergence of the algorithm. Section 5 is devoted to the floating-point extension of the algorithm. In Section 6 we discuss the mapping of the new algorithm to a state of the art DFPU. Section 7 describes the redundant version of the algorithm at its pipelining. Section 8 compares with related works, and finally the conclusions are presented in Section 9.

2. CORDIC Algorithm

The CORDIC algorithm performs plane rotations of vectors preserving a norm for circular or hyperbolic rotation. The intended operation for the rotation mode consists of rotating a vector (x_{in}, y_{in}) by an angle $z_{in} = \theta$. Any rotation

*This work has been partially supported by the Ministry of Science and Innovation of Spain under projects TIN2007-67537-C03-01 and TIN2006-01078.

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 ($\sigma_j \in \{-1, 1\}$). Therefore, the rotation by θ can be performed as a sequence of 2D rotations

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

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. The σ_j values are computed from 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 (computation of modulus and angle of a vector), in a simple way. In this case the direction of the rotations are determined as ($(x[j], y[j])$ is the vector obtained after the $j-1$ plane rotation) $\sigma_j = -sign(y[j])$ (assuming $x[j] \geq 0$) to zero out the y coordinate.

To obtain the conditions of convergence, for the rotation mode, we consider a finite set of angles α_i and that the value of the angles is rounded to fit the wordlength of the datapath. Let I_j the angular convergence domain at iteration j, that is $I_j = \sum_{i=j}^{n} \alpha_i$. Let ϵ the upper bound of the approximation error of the input angle θ, that is $|\theta - \sum_{i=1}^{n} \sigma_i \alpha_i| < \epsilon$. Moreover, θ should be within the convergence domain, $|\theta| < I_1$. Let δ_j the bound on the truncation error on $z[j]$ due to the finite wordlength of the datapath, so that the finite wordlength value is within $[z[j] - \delta_j, z[j] + \delta_j]$. Assume that at iteration j the remainder angle is within the convergence domain, that is $|z[j]| < I_j + \epsilon$. Due to the rounding errors, the algorithm may select $\sigma_j = 1$ when $z[j] \in [-\delta_j, I_j + \epsilon]$, so that the value of $z[j+1]$ is within the interval $[-\alpha_j - \delta_j, I_{j+1} + \epsilon]$. For $\sigma_j = -1$ the resulting interval is $[-I_{j+1} - \epsilon, \alpha_j + \delta_j]$. For convergence it is required that $|z[j+1]| < I_{j+1} + \epsilon$. Therefore, the condition of convergence is $\alpha_j + \delta_j < I_{j+1} + \epsilon$ We may express $\epsilon = \alpha_n + \delta_n$, which is the error after the n iterations if the algorithm converges. Since $\delta_j \leq \delta_n$ (the bound of the truncation error increases with the number of iterations), we conclude that convergence is achieved if the sequence of elementary angles verifies:

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

This result is also valid for vectoring, taking into account that truncation is used instead of rounding.

3. Constant Factor Decimal CORDIC

Recently we have proposed a decimal constant factor circular CORDIC algorithm [3]. The algorithm performs one elementary rotation for every bit of the input operands. For each elementary rotation only two alternative angles of the same absolute value but of different sign are possible. Thus the resultant scale factor of the algorithm is constant, which is an important property to have a simple algorithm. The elementary angle set is derived taking the arctangent of the weights of the binary positions of a decimal code. We proposed an angle set derived from a decimal code with digits coded with weights 5 2 1 and 1. This results in datapaths that only require scalings by x5 and x2. To extend the set of transcendental functions, it is of interest to extend the algorithm to the hyperbolic coordinate system. The angle set derived from the code 5211 does not assure the convergence of the algorithm for hyperbolic coordinates. In this work we present an improvement of the algorithm, that allows to use the same sequence of elementary rotations for the circular and hyperbolic coordinate systems.

For hyperbolic coordinates we need to derive the angle set from a decimal code with more redundancy, so that the condition of convergence is satisfied, but restricting the necessary multiples set in the datapath to x5 and x2 to have an efficient implementation. An elegant solution is to derive the angle set from the decimal code 5221. This is a redundant decimal code, and allows the convergence of the algorithm for circular and hyperbolic coordinates (see Section 4). Moreover, using this code only x5 and x2 multiples are needed. The unified algorithm is as follows ($c = 1$ for circular coordinates and $c = -1$ for hyperbolic coordinates):

Initialization: $x[1] = x_{in} \in [1, 10)$, $y[1] = y_{in} \in (-10, 10)$ $(|y[1]| \leq |x[1]|$ for vectoring) and $z[1] = z_{in} = \theta$ ($\theta \in [-1.069.., 1.069..]$ for $c = 1$, and $\theta \in [-1.166.., 1.166..]$ for $c = -1$),

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

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

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

$$y[i+1] = y[i] + \sigma_i\, C[i]\, 10^{-\lceil \frac{i}{4} \rceil}\, x[i] \tag{3}$$

$$z[i+1] = z[i] - \sigma_i\, \alpha_{i,c} \tag{4}$$

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 $C[i] = R[\text{i mod } 4]$ with $R[0:3] = \{1, 5, 2, 2\}$. The scale factor is constant, since $\cos(\sigma_i \alpha_{i,1})$ and $\cosh(\sigma_i \alpha_{i,-1})$ are constant. We call K_c to the scale factor (c identifies the coordinate system). This algorithm may be used directly for fixed point input arguments. In Section 5 we present the extension to floating-point.

4. Convergence of the Algorithm

In this section we demonstrate that the set of elementary angles of the proposed algorithm verifies the condition of convergence (1). We first present the demonstration for circular coordinates. To show that the algorithm converges for circular coordinates, we have to demonstrate that

$$\tan^{-1}(S[j]) \le \sum_{i=j+1}^{4m} \tan^{-1}(S[i]) + \tan^{-1}(10^{-4m}) \quad (5)$$

Since the decimal code 5221 is a valid redundant decimal code (it can represent any number within a certain range), the bit weight of each position is less than the addition of the bit weights of the positions to the right (lower weight), that is

$$S[j] \le \sum_{i=j+1}^{4m} S[i] + 10^{-4m}$$

Taking the $\tan^{-1}$ function in both terms and using the property $\tan^{-1}(a+b+...+c) \le \tan^{-1}(a) + \tan^{-1}(b) + ... + \tan^{-1}(c)$ results in the condition we want to demonstrate

$$\tan^{-1}(S[j]) \le \tan^{-1}\left(\sum_{i=j+1}^{4m} S[i] + 10^{-4m}\right) \quad (6)$$

$$\le \sum_{i=j+1}^{4m} \tan^{-1}(S[i]) + \tan^{-1}(10^{-4m}) \quad (7)$$

For hyperbolic coodinates the convergence condition is similar to (5) but using the $\tanh^{-1}()$ function. We use the following bound:

$$\tanh^{-1}(S[j]) \le S[j] + \frac{1}{2}\,(S[j])^3$$

for $S[j] < 0.716....$ Since $\tanh^{-1}(s) > s$,

$$\sum_{i=j+1}^{4m} \tanh^{-1}(S[i]) + \tan^{-1}(10^{-4m}) > \sum_{i=j+1}^{4m} S[i] + 10^{-4m}$$

Therefore the condition of convergence (1) for hyperbolic coordinates is satisfied if

$$S[j] + \frac{1}{2}\,(S[j])^3 \le \sum_{i=j+1}^{4m} S[i] + 10^{-4m} \quad (8)$$

The evaluation of this expression for the different values of (j mod 4), leads to the unified condition $10^{-2\lceil \frac{j}{4} \rceil} \le \min(4/225, 19/36, 1/36, 4/36)$, which is satisfied for $j \ge 1$ and then the proposed algorithm converges for hyperbolic coordinates.

5. Floating-Point Extension

To extend the algorithm to floating point, it is necessary to perform a range reduction, and to adapt the algorithm to deal with very small values keeping the required accuracy (exceptions and other low level issues are out of the scope of this paper). 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. Although, according to the IEEE-754 2008 standard, the input operands may not be normalized, for transcendental functions the preferred exponent is the minimum possible, so a normalization stage is necessary.

5.1. Range Reduction

For range reduction we consider the methods described in [9] [10].

[sin(F)/cos(F)]: the angle is decomposed as $A\ 10^{E_A} = N\ \pi/2 + z_{in}$, with N an integer, $z_{in} = M_{zin}\ 10^{-E_{zin}} \in [-\pi/4 - \gamma, \pi/4 + \gamma]$ and $\pi/4 + \gamma \le 1.069....$ The parameter γ allows certain redundancy that may simplify the range reduction implementation [9]. The functions are computed in the circular rotation mode with input arguments (no scale factor compensation is necessary) $x_{in} = K_1, y_{in} = 0.0$ and $z_{in} = M_{zin}\ 10^{-E_{zin}}$ with $M_{zin} \in [1, 10)$ and $E_{zin} \ge 0$. After computing the sine or cosine of z_{in} (the final result of the y or x iteration), the sine or cosine of the input angle may be obtained by simple trigonometric identities.

[tan^{-1}(F/G)]: if $F \ge G$ the algorithm computes $\tan^{-1}(G/F)$ and then by trigonometric identities $\tan^{-1}(F/G)$ is obtained. The function is computed in the circular vectoring mode with $(x_{in}, y_{in}) = (A, B\ 10^{-E_{yin}})$ if $F \ge G$ or $(B, A\ 10^{-E_{yin}})$ in other case, with $E_{yin} = |E_A - E_B|$. The final result is obtained in the z coordinate.

[sinh(F)/cosh(F)]: the following decomposition is performed $S_A\ A\ 10^{E_A} = N \ln(10) + z_{in}$, with N an integer, $z_{in} = M_{zin}\ 10^{-E_{zin}} \in [-\ln(10)/2 - \gamma, \ln(10)/2 + \gamma]$, and $|\ln(10)/2 + \gamma| \le 1.166...$ As before, γ provides some redundancy to simplify the range reduction. Following the method of [10], the functions are computed in the hyperbolic rotation mode with $(x_{in}, y_{in}) = (0.5\ (1 + 10^{-2N})\ K_{-1}, 0.5\ (1 - 10^{-2N})\ K_{-1})$ and $z_{in} = M_{zin}\ 10^{-E_{zin}}$ with $M_{zin} \in [1, 10)$ and $E_{zin} \ge 0$. The result is obtained in the x or y coordinate and does not require scale factor compensation. N should be added to the exponent of the result.

[e$^{\mathbf{F}}$]: the same range reduction as for sinh/cosh is performed. The function is computed in the hyperbolic rotation mode with $x_{in} = y_{in} = K_{-1}$ and $z_{in} = M_{zin}\ 10^{-E_{zin}}$.

[10$^{\mathbf{F}}$]: the following decomposition is performed $10^{S_A\ A\ 10^{E_A}} = 10^{N+r} = 10^N\ e^{r\ \ln(10)}$, with $-0.5 - \gamma/\ln(10) \le r \le 0.5 + \gamma/\ln(10)$ and $|0.5 + \gamma/\ln(10)| \le 1.166../\ln(10)$. Then a base e exponential is computed.

[$\tanh^{-1}$(F/G)]: The domain of the function is defined for $|F| < |G|$. Since $\tanh(1.166..) = 0.8229...$, we may perform the direct computation of the function for $|F|/|G| \leq 0.8229....$ We use the range reduction method proposed in [10]. For the cases i) $E_A \leq E_B - 2$, or ii) $A\ 10^{E_A - E_B} < 0.5\ B$ when $E_A = E_B$ or $E_A = E_B - 1$, the function is computed directly in the hyperbolic rotation mode with $(x_{in}, y_{in}) = (B, A\ 10^{E_{yin}})$ with $E_{yin} = E_A - E_B$. For $A\ 10^{E_A - E_B} \geq 0.5\ B$ with $E_A - E_B = 0$ or -1 the function is computed in the hyperbolic vectoring mode with $(x_{in}, y_{in}) = ((B + A) - (B - A)\ 10^{E_s}, (B + A) + (B - A)\ 10^{E_s})$, with $(B - A).\ 10^{E_A - E_B} = S\ 10^{-E_s}$ and $S \in [1, 10)$ (see [13] for details).
[ln(F)]: We use the transformation $\ln(A\ 10^{E_A}) = E_A\ \ln(10) + \ln(A)$. Then the following computation is performed: $\ln(A) = 2\ \tanh^{-1}((A-1)/(A+1))$. Since $A < 10$ we have that $(A-1)/(A+1) < 0.8229..$ and the function is computed directly in the hyperbolic vectoring mode with $(x_{in}, y_{in}) = (A+1, A-1) = (A+1, M_{yin}\ 10^{-E_{yin}})$.
[$\log_{10}$(F)]: the following transformation is used $\log_{10}(A\ 10^{E_A}) = \log_{10}(e)\ \ln(A) + E_A$. Then $\ln(A)$ is computed as in the previous case.
[Square root]: We compute $\sqrt{A}$ for even exponent, and $\sqrt{A/10}$ for odd exponent. The square root is computed using the hyperbolic vectoring mode that allows the computation of $\sqrt{x_{in}^2 - y_{in}^2}/K_{-1}$ (obtained in the final value of the x coordinate). Specifically, for even exponent we compute $\sqrt{(A + K_{-1}^2)^2 - (A - K_{-1}^2)^2}/K_{-1} = 2\sqrt{A}$ (the final result have to be multiplied by 0.5). To avoid an overflow in the x coordinate we use $(x_{in}, y_{in}) = (A, A)$, perform the first CORDIC iteration to obtain $(x[2], y[2])$ and add to $x[2]$ (subtract to $y[2]$) the constant correction term $K_{-1}^2\ (1 + 0.5\ K_{-1}^2)$. For odd exponent we use $(x_{in}, y_{in}) = ((A/10 + K_{-1}^2/4), (A/10 - K_{-1}^2/4))$, which results in $\sqrt{A}$ in the range $[\sqrt{0.1}, 1)$, so that a final decimal left shift is needed. The hyperbolic modulus is computed with the required accuracy in about half of the iterations required for the other functions. Therefore, about $2m$ hyperbolic rotations are necessary.

5.2. Floating-Point Iterations

After range reduction, one of the inputs to the CORDIC iterations are of 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 $E_{yin}, E_{zin} \geq 0$. To produce accurate results it is necessary to modify the fixed point iteration to move E_{yin} or E_{zin} leanding zeros or nines from the computation of the corresponding y or z iteration (or both). We use a similar approach to [11]. We scale the y and/or z iterations by $10^{E_{yin}}$ or $10^{E_{zin}}$, depending on the mode of operation, and start the iterations with the index $i = J$ so that the input argument is within the range of convergence.

Specifically, the resultant floating-point algorithm is as follows:
Initialization: $x[1] = x_{in}$, $y[1] = y_{in}$, and $z[1] = z_{in}$
Iteration: for $i = J$ to $J + 4m - 1$
$\sigma_i = sign(z[i])$ (rot.) or $\sigma_i = -sign(y[i])$ (vect.)

$$\begin{aligned} x[i+1] &= x[i] - c\,\sigma_i\, C[i]\, 10^{-\lceil \frac{i}{4} \rceil - P}\, y[i] && (9) \\ y[i+1] &= y[i] + \sigma_i\, C[i]\, 10^{-\lceil \frac{i}{4} \rceil + P}\, x[i] && (10) \\ z[i+1] &= z[i] - \sigma_i\, A_{i,c}\, 10^{-\lceil \frac{i}{4} \rceil + Q} && (11) \end{aligned}$$

with $A_{i,1} = 10^{\lceil \frac{i}{4} \rceil} \tan^{-1}(S[i])$ and $A_{i,-1} = 10^{\lceil \frac{i}{4} \rceil} \tanh^{-1}(S[i])$. The value of P and Q depend on the function computed: for sin, cos, sinh (with $N = 0$) and cosh (with $N = 0$) $P = Q = E_{zin}$; for sinh and cosh with $N \neq 0$, $P = 0$ and $Q = E_{zin}$; for $\tan^{-1}$ and $\tanh^{-1}$ (and ln and $\log_{10}$) $P = Q = E_{yin}$; for square root $P = Q = 0$.

To determine the value of J, we take into account that after range reduction the following bounds result: $|z_{in}| < 10^{-E_{zin}+1}$ for rotation, and $|y_{in}|/x_{in} < 10^{-E_{yin}+1}$ for vectoring. Therefore, convergence is achieved taking $J = 4\ E_{zin} - 3$ for rotation and $J = 4\ E_{yin} - 3$ for vectoring. The implementation of these iterations require a simple control, mainly based on the value of E_{yin} or E_{zin} and on the function implemented. The control is performed over each of the shifting operations. Note that for the x iteration the shift value might be greater than the maximum shifting operation available, limited by the wordlength. In that case the shifted value underflows without affecting the accuracy of the algorithm.

Since now the initial and final index of the iteration is variable, a large number of constant would be required. However, the number of stored constant is kept small due to the linear approximation of the elementary rotation angles and the scale factors (to obtain some constant inputs). As it is shown in [13], about $4B$ angles, and $8B/3$ scale factors need to be stored, for a datapath with B fractional decimal digits.

5.3. Error analysis

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). For instance, for the computation of the cosine we have the following sources of error:
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 truncation errors in z. For the last elementary angle we use the bound 10^{-m}. For the truncations errors, we assume that the angles are stored with a rounding error bounded by $0.5\ 10^{-B}$, but during the right shifting a truncation is performed, resulting in a bound

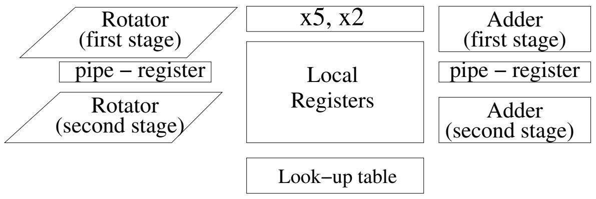

Figure 1. DFPU pipeline for mapping the CORDIC algorithm.

of 10^{-B}. However for about 2/3 of the iterations the error is much smaller than 10^{-B} ($\tan^{-1}(u) \approx u$). To compensate for this effect we use the bound $0.5\ 10^{-B}$. The resultant error bound for $4m$ iterations is $\epsilon = \pm(10^{-m} + 4m\ 0.5\ 10^{-B})$. The corresponding error in the cosine $|\cos(\theta) - \cos(\theta + \epsilon)|$ can be approximated by ϵ, since the error is very small compared to θ.

Truncation error in the x/y datapath: we assume two truncation errors (with a bound of $2\ 10^{-B}$) in each coordinate, per iteration. Therefore, the total truncation error in each one of the x/y datapaths is bounded by $8m\ 10^{-B}$.

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

The bound of error should be less than one ulp. This results in the condition $10m\ 10^{-B} + 10^{-m} \leq 0.5\ 10^{-p}$. For $B = \infty$ a lower bound of m is $m = p+1$. We take $m = p+1$, resulting in $B > p + 2.94$. Therefore we need to perform $4\ (p+1)$ elementary rotations, with a datapath of $p+4$ decimal digits (one integer and $B = p+3$ fractional). The analysis for the other functions shows that these parameters also assure one ulp accuracy.

6. Mapping to a State of the Art DFPU

In this section we propose a mapping of the proposed algorithm to compute transcendentals in a state of the art DFPU, i.e. a unit such as the DFPU of the IBM Power 6 or Z10 processors. Our intention is not to describe a mapping according to the detailed architecture of those commercial units, but to make reasonable assumptions for the mapping in a state of the art unit. We show the sequence of operations to be pipelined and specify the modifications need to suit an efficient implementation of the algorithms proposed.

We assume the hardware elements shown in Figure 1(a), and that the elements are organized so that we can configure a logical pipelined structure. Therefore the addition from register to register (including multiplexer and complementing) requires two cycles, and the shifting (using the rotator) requires also two cycles (from register to register). Moreover, we assume that the x5 or x2 scalings are included within the second cycle of the shifting operation (those multiples require a small constant time; this requires that the output of the rotator could be selected as input of the x5,x2 multiple generator).

Regarding the modification of the hardware, we need to add a look-up table for storing the elementary angles set and the scale factors. The output of the tables should input to one of the adder's multiplexers and to the rotator input multiplexer. Moreover, we need to control the operation of the adder (addition or subtraction) with the sign bit obtained in certain subtraction operations. This bit is stored and used when necessary according the algorithm. This would require a slight change in the control unit.

Basically we need to pipeline the operations for the three coordinates. For the x and y coordinates two stages are required for the shifter and x5 or x2 multiples, and two stages for the addition. For the z coordinate we need one stage for look-up table access, two stages for the shifter (in the floating-point version of the algorithm the z iteration may require a shifter), and two stages for the addition. Figure 2 shows the pipeline schedule of the different operations for some iterations of the algorithm. The same pipelining is used for rotation and vectoring (and for circular and hyperbolic coordinates) since the values of $z[i]$ or $y[i]$ are computed before σ_i is needed.

Each iteration is completed every four cycles. For $4m$ elementary rotations, a total of $16m + 2$ cycles are needed. For instance, for $p = 34$ (which would correspond to the number of elementary rotations for Decimal128) $m = p + 1 = 35$, resulting in 562 cycles.

An interesting optimization is to use a fast termination for the CORDIC algorithm [12]. This scheme is based on the fact that after half of the elementary rotations the approximations $\cos(\alpha) = \alpha$ and $\sin(\alpha) = 1$ can be used, where α is the remainder angle to be rotated (similar approximations can be used for hyperbolic coordinates). The termination consists in performing a multiplication–add (for each of the x and y coordinates) for rotation and a division–add operation (for the z iteration) for vectoring. To allow the use of the built-in multiplication and division operations for 16 decimal digits (with estimated fixed-point latencies of 20 and 67 cycles for multiplication and division respectively), we perform the elementary rotations for $m = 19$, and then perform a 16 digit multiplication or division and then a final addition. For instance, for computing the $\cos$ function, $2 + 16 \times 19 = 306$ cycles are needed for the elementary rotations, 20 cycles for the termination (multiplication) and 2 cycles for the final addition, for a total of 328 cycles. Table 1 shows the number of cycles for each of the considered functions for Decimal128, taking into account these optimizations. We also show the ratio of latency to

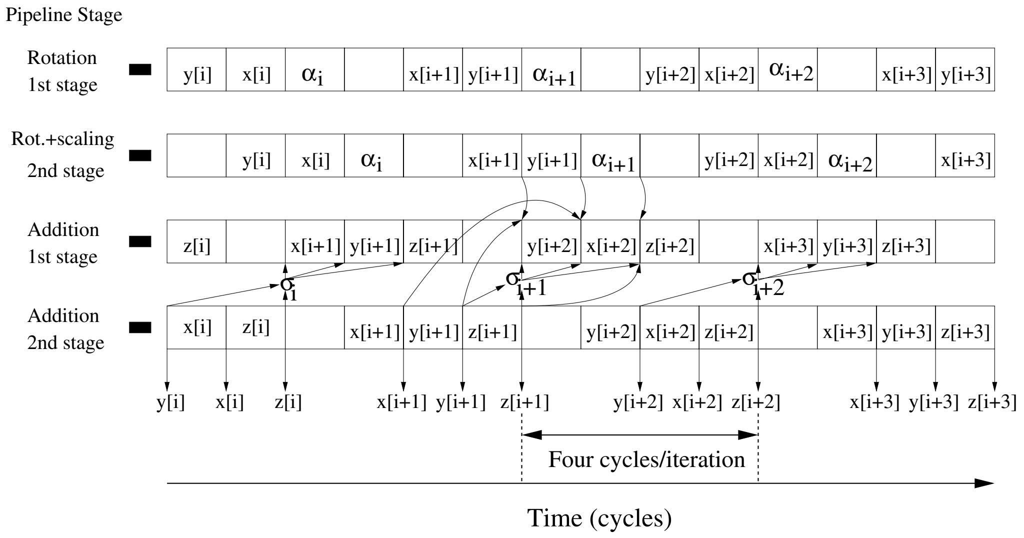

Figure 2. Pipelining of the proposed CORDIC algorithm.

Table 1. Number of cycles for the different transcendental functions.

Function	sin, cos, sinh, cosh	$\tan^{-1}(a/b)$, $\tanh^{-1}(a/b)$	ln	exp	sqrt
# cycles	328	375	378	328	292
ratio to mult.*	4.7	5.4	5.4	4.7	4.2

* Ratio with respect to fixed-point multiplication for Decimal128.

fixed-point multiplication for Decimal128 coefficients (with a estimated latency of 70 cycles). Note that the number of cycles for the computed functions does not include the pre and post processing required for floating point.

7. Algorithm for Carry-Save Representation

In this section we discuss the proposed CORDIC algorithm using a redundant decimal carry-save representation for the operands. This algorithm allows to use a carry-save decimal adder instead of the complex carry propagate adder. Since the operands are represented in decimal carry-save, and we want to keep the scale factor constant, 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 bits of the carry-save representation of the control coordinate. This scheme was already used for binary CORDIC with redundant adders (see for instance [9]). An error due to a wrong sign of the estimation may lead to convergence problems. The representation derived from the decimal code 5221 has enough redundancy so that repetition of iterations is not necessary. Therefore, we determine the number of bits of the estimation so that the basic elementary angle set assures convergence. Due to space limitations, we restrict the discussion to the rotation mode in circular coordinates.

The scheme is based on an estimation of the sign by analyzing some few bits of the remainder angle. To have the bits to be analyzed in a fixed position, a scaling of the z coordinate has to be performed. A natural choice is to scale the z coordinate by the amount $10^{\lceil i/4 \rceil}$. We perform the following change of variable $r[i] = 10^{\lceil \frac{i}{4} \rceil}\ z[i]$ and use the scaled elementary angles $A_i = 10^{\lceil \frac{i}{4} \rceil}\ \tan^{-1}(S[i])$. The resulting recurrence for $r[i]$ is

$$r[i+1] = \begin{cases} 10(r[i] - \sigma_i A_i) & \text{if } i \bmod 4 = 0 \\ r[i] - \sigma_i A_i & \text{if } i \bmod 4 \neq 0 \end{cases} \tag{12}$$

The convergence condition (5) becomes:

$$|r[i]| < A_i + 10^{\lceil \frac{i}{4} \rceil} \sum_{j=i+1}^{4m} \alpha_j + \alpha_{4m} \tag{13}$$

To perform the computation using redundant representation we carry out an estimation of the sign of $r[i]$ by truncating some leading bits. Let us denote as $\hat{r}[i]$ the truncation of $r[i]$ using t fractional bits, and $\hat{\sigma_i}$ to the estimation of the sign obtained by selecting the sign of $\hat{r}[i]$ ($\hat{\sigma_i} = sign(\hat{r}[i])$). Now we 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 bits ($\hat{r}[i]$) is:

$$\hat{r}[i] \leq r[i] < \hat{r}[i] + 2D[t]10^{-\lceil \frac{t}{4} \rceil} \tag{14}$$

where $D[t] = R[t \bmod 4]$, with $R[0:3] = \{1, 8, 4, 2\}$. A wrong estimation of the sign of $r[i]$ may be performed when $\hat{r}[i] = -D[t]\ 10^{-\lceil \frac{t}{4} \rceil}$ ($\hat{\sigma_i} = -1$). We need to determine a bound on t so that this wrong estimation still allows convergence. From expression (14) we have $-D[t]\ 10^{-\lceil \frac{t}{4} \rceil} \le r[i] < D[t]\ 10^{-\lceil \frac{t}{4} \rceil}$. Two cases are possible: i) $-D[t]\ 10^{-\lceil \frac{t}{4} \rceil} \le r[i] < 0$ and then $\sigma_i = -1$ (the algorithm converges), and ii) $0 \le r[i] < D[t]\ 10^{-\lceil \frac{t}{4} \rceil}$ and then $\sigma_i = +1 \ne \hat{\sigma_i}$, so that the convergence depends on the number of bits of the truncated estimation, as we show below.

Let us deal with the case in which $0 \le r[i] < D[t]\ 10^{-\lceil \frac{t}{4} \rceil}$ (and $\hat{\sigma_i} = -1$). Therefore, to update $r[i+1]$ using equation (12) an addition instead of a subtraction is performed. Now we prove that, in spite of wrong sign estimation, there is enough angle redundancy to ensure the convergence of the algorithm.

The maximum value of t is obtained for the case of minimum redundancy. An estimation of t bits to determine the sign of $r[i]$ may produce a residual angle bounded by $D[t]\ 10^{-\lceil \frac{t}{4} \rceil}\ 10^{-\lceil \frac{i}{4} \rceil} + \alpha_i$. This residual angle should be within the convergence bound, that is

$$D[t]\ 10^{-\lceil \frac{t}{4} \rceil}\ 10^{-\lceil \frac{i}{4} \rceil} + \alpha_i \le \sum_{j=i+1}^{4m} \alpha_j + \alpha_{4m}$$

Therefore

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

where $V[i]$ is the overlap between angle i and the addition of the remaining angles plus the bound of the final error. In [13] we show that the worst case for convergence corresponds to i mod 4 =0 and among the possible values of i that verifies this condition, the worst case is $i = 4m - 4$. Therefore, $D[t]\ 10^{-\lceil \frac{t}{4} \rceil}\ 10^{-m+1} \le (\alpha_{4m-3} + \alpha_{4m-2} + \alpha_{4m-1} + \alpha_{4m} + \alpha_{4m}) - \alpha_{4m-4}$. We show in [13] that this bound is verified for (correct for practical values of $m >> 2$)

$$D[t]\ 10^{-\lceil \frac{t}{4} \rceil} \le 0.1 + \frac{599}{30} 10^{-2m}$$

A suitable value for the number of fractional digits of the estimation of the sign of $r[i]$ is $t = 4$. The number of bits of the integer part of the estimation is obtained from the upper bound of $|r[i]|$ in (13), which corresponds to the case i mod 4=1 [13]. The upper bound is 11.11..., that requires 5 bits (two decimal digits, but one of them of one bit). 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.

7.1. Pipelining of the Redundant Algorithm

We consider a hardware unit similar to that of Figure 1 but including a decimal carry-save adder plus a sign detector (of 10 bits for circular rotation). We consider three pipeline stages. Fist stage: first part of the rotator; second stage: second part or the rotator plus scaling by x5 or x2; third stage: decimal 3–to–2 carry-save adder and sign detection (we assume that this logic fits the cycle time of 13 FO4 taking the Power 6 processor as a reference).

Since a redundant representation is used, each operand requires two words. Therefore the implementation of the iteration requires two shifts and two 3–to–2 carry-save additions for x and y (one for the z iteration). The resultant latency per iteration is 5 cycles [13], requiring 1.25 more cycles than the corresponding implementation with a carry-propagate adder. The carry-propagate adder is used at the end for conversion and rounding. Therefore the redundant algorithm requires a higher number of cycles than the non redundant version, but a simpler carry-save adder is used instead of the more costly (in hardware and power) fast carry–propagate adder.

8. Comparison with Related Works

In this section we compare our proposal with previous works on decimal CORDIC focusing on the mapping of the algorithms to a state of the art DFPU. We also present a rough comparison with table-driven polynomial methods to compute transcendentals. Decimal CORDIC for pocket calculators were proposed in [4] [6] [8]. These implementations used as rotation angles the values $\tan^{-1}(10^{-j})$. To assure 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 with our algorithm. In contrast, the implementation does not require the x2 and x5 multiples. A recent proposal [7] apply the standard base 2 CORDIC algorithm (with elementary rotation angles $\tan^{-1}(2^{-j})$) to the operands coded in decimal. They have to implement the multiplications by $\times\ 2^{-j}$ which are complex for decimal BCD operands.

Another recent work for implementing transcendental functions using a decimal digit-by-digit algorithm was proposed in [5]. 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 algorithm has a reduced convergence domain, so some range reduction computations are necessary to have a comparable range of convergence as our algorithm. They require one iteration per decimal digit of the input operands, so that m iterations are required. The main drawbacks of this algorithm are the following:

i) Although this algorithm requires 1/4 of the iterations of our algorithm (in fact about m vs. $2.5m$ iterations considering our optimization with fast termination), the mapping of 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. Moreover they have two iterations to accumulate constant values. Therefore, 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.
ii) The algorithm presents certain irregularities that make the mapping more difficult: some initial computations must be performed to reduce the range so that the selection of the digits can be done by rounding.
iii) 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 of look-up table in comparison to 14 Kbits for our algorithm (a ratio of 27), both for the fixed-point case.

Table-driven polynomial methods are a popular alternative for transcendental function evaluation. Therefore it is worth to obtain an estimate of the latency required in a state of art DFPU to compute a transcendental function in a reduced range (after the range reduction stage). We use as a reference the error of approximation of a Chebyshev polynomial (the optimum polynomial is obtained from the minimax approximation, but the result should be close), evaluation of the polynomial by using the Horner's rule and considered optimizations based on the significance of each of the coefficients of the polynomial, which allow to reduce storage space and the number of cycles for the multiplications (see [13] for details). For functions of one argument, using two digits to obtain the coefficients of the polynomial, a 10th degree polynomial is required, resulting in about 560 cycles of latency for Decimal128 (only fixed-point range reduced part) and a storage of about 120K BCD digits per function. An alternative with much less storage (4.5K BCD digits per function) is obtained, using one digit to obtain the coefficients of the polynomial (requiring a 14th degree polynomial), resulting in a latency of about 880 cycles. In contrast, our implementation requires between 300 and 400 cycles with a storage of 9.4K BCD digits for several functions (some of then of two arguments).

9. Conclusions

We presented a new decimal floating-point CORDIC algorithm for the computation of transcendental functions. As the 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. A novel coding scheme allowed us to use a unified algorithm for both circular and hyperbolic coordinates. Moreover, the same set of angles can be used for a redundant version of the algorithm. We showed how to map the algorithms (redundant and non redundant versions) to a state of the art DFPU with minor hardware modifications. A comparison with previous decimal CORDIC proposals reveals that our approach is more efficient for mapping to a state of the art DFPU. Moreover, a rough comparison with table-driven methods shows that our approach allows a significant reduction of latency and storage.

Acknowledgements

We thank the reviewers for the valuable comments, leading to improvements in this paper.

References

[1] 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.

[2] 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.

[3] 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.

[4] D.S. Cochran, "Algorithms and Accuracy in the HP-35", Hewlett-Packard Journal, pp. 10-11, 1972.

[5] L. Imbert, J.M. Muller and F. Rico, "A Radix-10 BKM Algorithm for Computing Transcendentals on Pocket Computers", Journal of VLSI Signal Proc. Sys., Vol. 25, no 2, June 2000, pp. 179–186.

[6] J.E. Meggitt, "Pseudo Division and Pseudo Multiplication Processes", IBM Journal, pp. 210-226, April 1962.

[7] A. Jimeno et al., "A BCD-based architecture for fast coordinate rotation", Journal of Systems Arch., vol 53, no 8, pp. 829–840, 2008..

[8] H. Schmid, "Decimal Computation", John Wiley & Sons, 1974.

[9] J.M. Muller, "Elementary Functions: Algorithms and Implementation". Birkhauser Verlag AG, second edition, 2007.

[10] H. Hahn et al."A Unified and Division-Free CORDIC Argument Reduction Method with Unlimited Convergence Domain Including Inverse Hyperbolic Functions," IEEE Trans. on Computers, vol. 43, no. 11, pp. 1339-1344, Nov. 1994.

[11] J.S. Walther, US patent 3766370: "Elementary Floating Point CORDIC Function Processor and Shifter", filing date May 14, 1971, issue date Oct 16, 1973.

[12] H.M. Ahmed, "Efficient Elementary Function Generation with Multipliers", in Proc. 9th IEEE Symposium on Computer Arithmetic, pp. 52–59,1989.

[13] A. Vázquez, J. Villalba and E. Antelo, "Computation of Decimal Transcendental Functions using the CORDIC algorithm – Extended Report", Int. Rep. available at http://www.ac.usc.es/biblio, 2008.