A CORDIC Argument Reduction Method with Regular Architecture and Unlimited Convergence Domain

H. Hahn, D. Timmermann, B. J. Hosticka, and B. Rix

Fraunhofer Institute of Microelectronic Circuits and Systems

Finkenstr. 61, D-4100 Duisburg 1, Germany

Abstract

One of the main problems of the CORDIC algorithm is the limited convergence domain, in which the functions can be calculated. Two different approaches can be employed to overcome this constraint: *first*, an argument reduction method and, *second*, an expansion of the CORDIC convergence domain. While the first approach requires significant processing overhead due to the need for divisions, the second technique achieves an increased but still limited convergence domain only. In this contribution we present a unified division-free argument reduction method and a regular pipeline/array architecture for floating point or fixed point implementations which results in savings of computation time. In contrast to previous methods we avoid extra CORDIC arithmetic for realization of argument reduction.

I. Introduction

One method to realize elegant computing of elementary functions is the CORDIC (Coordinate Rotation Digital Computer) algorithm /1,2/, which is becoming increasingly popular due to its functionality and regular architecture. An important aspect of elementary function evaluation is the usually limited convergence domain, in which the functions can be calculated. Two different approaches can be found in CORDIC implementations: *first*, an argument reduction method and, *second*, an expansion of the convergence domain. The first method was proposed by Walther in 1971, who used mathematical transformations for argument reduction in a recursive CORDIC add-and-shift arithmetic /2/.

The general strategy for computing elementary functions employing the argument reduction method consists of three steps /2/: *reducing the argument to a given convergence domain, evaluating the function for the reduced argument, and combining the result with the initial transformation*. The time required for argument reduction sometimes exceeds the computation time of the actual CORDIC algorithm. This also results in an increased chip area for VLSI pipeline/array implementations. An additional disadvantage is the structural irregularity when using non-unified mathematical transformations. For example, Walther´s method requires a division operation when executing the hyperbolic mode of the CORDIC algorithm.

These are the main reasons why in the last years the second concept - a limited expansion of convergence domain - has achieved more attraction, especially for fixed point (FXP) implementations in digital signal processing applications /5,11/. Recursive /3,11/ and special pipeline implementations /7,8/ do exist which use this method. Expanding the convergence range by repetition of certain iterations was proposed by Ahmed /4/ and Deprettere /5/ which is based on the so-called double-shift concept. Hu used both concepts in /9/. The advantage of this approach is its regularity and the hardware overhead is lower than the amount needed for a special argument reduction. However, all variations of the second method achieve a limited convergence range only with restrictions on the input and the output variables and, in addition, incorporate the convergence domain expansion into the basic CORDIC iterations. In this contribution we will show a unified division-free argument reduction method with an unlimited convergence domain (in a given data format) and an architecture suitable for FLP or FXP implementations /10/. The resulting architecture provides faster function computation with comparatively moderate increase of hardware amount.

II. Convergence domain and parameters of the CORDIC algorithm

CORDIC is an iterative method which performs vector rotations by using elementary add-and-shift operations only

$$\begin{aligned} x_{i+1} &= x_i - m\,\sigma_i\,2^{-S(m,i)}\,y_i \\ y_{i+1} &= y_i + \sigma_i\,2^{-S(m,i)}\,x_i \\ z_{i+1} &= z_i - \sigma_i\,\alpha_{m,i} \qquad\qquad i = 0,1,\ldots, N-1 \end{aligned} \tag{1}$$

with the shift-sequence $S(m,i)$, the iteration index i, N the number of iterations, and the rotation direction $\sigma_i \in \{-1,1\}$. The angle $\alpha_{m,i}$ is defined as $\alpha_{m,i} = m^{-½} \tan^{-1}(m^{½}\ 2^{-S(m,i)})$. Subject to the parameter m, the algorithm operates in so-called circular (m=1), hyperbolic (m=-1), and linear (m=0) mode. Further, it carries out "rotation" or "vectoring" depending on whether the final value of z or y is forced to zero during iteration process. The determination of rotation direction is described by the parameter σ_i which depends on the sign of z or y. After N+1 iterations, yielding an accuracy of at least n=N bit, the result of x and y computations must be compensated for the scaling factor K_m . Table I summarizes the computable functions using the CORDIC algorithm. The maximum convergence domain C_m defines the maximum angle by which the input vector can be rotated and is given by

$$C_m = \alpha_{m,N-1} + \sum_{i=0}^{N-1} \alpha_{m,i} \geq \begin{cases} \left| \dfrac{1}{\sqrt{m}} \tan^{-1}(\sqrt{m}\, y_0/x_0) \right| & \text{for } y_n \to 0 \\ \left| z_0 \right| & \text{for } z_n \to 0 \end{cases} .$$

Table II gives the values of the parameters $S(m,i)$, C_m, and K_m based on Ref. /2/. For m=-1 some iterations $i \in \{4,13,...,k,3k+1,... \}$ must be repeated to fulfill the convergence criteria given in /2/.

The methods which use a limited expansion of the convergence domain were proposed by Ahmed /4/ and Deprettere /5/. Ahmed introduced a repetition of the index i in $2^{-S(m,i)}$ and allowed leading negative integers for $S(1,i)$. The number of iterations increases approximately by a factor of two in comparison with the original CORDIC method. Deprettere adopted another representation of the angle, where the argument is adjusted to be close to unity in order to expand the argument range of $\tanh^{-1}(z)$ /5/, namely $\alpha_{m,i} = m^{-½} \tan^{-1}(\sqrt{m}\ (2^{-S(m,i)} + \eta_i\ 2^{-S(m,j)}))$ with $S(m,j) > S(m,i)$ and $\eta_i \in \{1,-1,0\}$. The term $2^{-S(m,i)}$ in Eq. (1) has now to be substituted by the term $(2^{-S(m,i)} + \eta_i\ 2^{-S(m,j)})$, which requires two add-and-shift operations for one iteration (also called the double-shift concept). Hu pursued the same ideas in /9/.

III. The argument reduction algorithm

Typically, a FLP number is characterized by an expression $M\ 2^E$ where M is the mantissa and E the exponent. For the FLP CORDIC operation the following steps must be executed:

1) Conversion of external FLP to internal FXP format (Preprocessing stage)

2) Reducing the argument to a given convergence domain by suitable transformations

3) Evaluating the function for the reduced argument (N+1 FXP CORDIC iterations)

4) Combining the computed function with the transformation executed in step 2 (Postprocessing stage)

5) FXP to FLP conversion

In /7/ and /12/ one can find an architecture which implements the steps 1, 3, and 5. It supports FLP multiplication and division with unlimited argument range. For trigonometric and hyperbolic vector rotations, however, the convergence domain is still limited. Fig. 1 shows the block diagram of the proposed CORDIC architecture that realizes steps 1-5 and the nomenclature used. The normalized inputs in FLP format are $x_0=Mx_0\ 2^{Ex_0}$, $y_0=My_0\ 2^{Ey_0}$, and $z_0=Mz_0\ 2^{Ez_0}$ with $0.5 \leq |M| < 1$. The reference exponent $Eref$ depends on the exponents of the input data. Here and in the following we assume that the output data of step 5 are always normalized as indicated by the function Norm().

III.1 Rotation and vectoring for *m* = 0

In linear mode no real argument reduction is required. However, there are some adjustments necessary which are summarized in Table III.

III.2 Rotation and vectoring for *m* = 1

III.2.1 *Rotation*

We introduce a function Adj() applied to a floating point number $In = M_{in}\, 2^{E_{in}} = M\, 2^{Eref}$ defined by $M = \mathrm{Adj}(M_{in}, Eref) = M_{in}\, 2^{E_{in} - Eref}$. We obtain by using the property of periodicity:

$$z_0 = Mz_0\, 2^{Ez_0} = Mz_0'' = Q\frac{\pi}{2} + D \qquad Q = \left\lfloor \frac{Mz_0''}{\pi / 2} \right\rfloor \qquad |D| < \pi/2 < C_1.$$

step 1: $Eref = \max(Ex_0, Ey_0)$ $\quad Mx_0'' = \mathrm{Adj}(Mx_0, Eref)$ $\quad My_0'' = \mathrm{Adj}(My_0, Eref)$.

step 2: Mx'_0 , My'_0 from Table IV

$$Mz'_0 = D = R\frac{\pi}{2} \qquad \text{with} \qquad Mz_0''\frac{2}{\pi} = Q + R \qquad \text{and} \qquad R = \frac{D}{\pi/2} \tag{2}$$

In contrast to Walther´s CORDIC-based implementation we use a scaler-based structure (also called multiplier by a constant and designated as constant multiplier in further context) for multiplication by $2/\pi$ or $\pi/2$, which will be introduced in Section IV (Fig. 2). The values in column 'Q mod 4' represent the two least significant bits of the integer *Q*. In the following we assume that scaling factor compensation has been already executed.

step 3: $Mx_n = Mx'_0 \cos(D) - My'_0 \sin(D)$ $\qquad My_n = My'_0 \cos(D) + Mx'_0 \sin(D)$

step 4: $Mx'_n = Mx_n$ $\qquad My'_n = My_n$

step 5: $x_n = \mathrm{Norm}(Mx'_n\ 2^{Eref})$ $\qquad y_n = \mathrm{Norm}(My'_n\ 2^{Eref})$

III.2.2 *Vectoring*

With $z_0 = 0$ we obtain the following operations in the *z* data path:

step 1: $Eref = \max(Ex_0, Ey_0)$ $\quad Mx_0'' = \mathrm{sign}(Mx_0) \cdot \mathrm{Adj}(Mx_0, Eref)$ $\quad My_0'' = \mathrm{sign}(My_0) \cdot \mathrm{Adj}(My_0, Eref)$

step 2: $Mx'_0 = Mx_0''$ $\qquad My'_0 = My_0''$

step 3: $Mz_n = \tan^{-1}(My'_0 / Mx'_0)$ $\quad$ with $\quad 0 \le Mz_n \le \pi/2$

step 4: $Mz'_n = \mathrm{sign}(My_0)\, Mz_n$ $\quad$ if $\mathrm{sign}(Mx_0) = 1$ $\quad$ else $\quad Mz'_n = \mathrm{sign}(My_0) \cdot (\pi - Mz_n)$

step 5: $z_n = \mathrm{Norm}(Mz'_n)$

Using the input data of **step 2** the norm is then given by:

step 3: $Mx_n = \sqrt{Mx'^2_0 + My'^2_0}$

step 4: $Mx'_n = Mx_n$

step 5: $x_n = \text{Norm}(Mx'_n\, 2^{Eref})$

III.3 Rotation and vectoring for m = -1

III.3.1 *Rotation*

step 1: $Eref = \max(Ex_0, Ey_0)$ $Mx_0'' = \text{Adj}(Mx_0, Eref)$ $My_0'' = \text{Adj}(My_0, Eref)$.

The convergence domain of the hyperbolic functions $\cosh(z_0)$ and $\sinh(z_0)$ is limited to the range $|z_0| \le 1.113$ (see Table II) and, hence, an argument reduction must be performed. Choosing $D \le \ln(2) < C_{-1}$ the argument z_0 can be written as $z_0 = Mz_0\, 2^{Ez_0} = Q \ln(2) + D$ with the same scheme of computation for Q and D as in the trigonometric case. According to Eq. (2) this yields $Mz_0\, 2^{Ez_0} \frac{1}{\ln(2)} = Q + R$ with $R = \frac{D}{\ln(2)}$ and $D = R \ln(2)$. That means that the same operations in the z data path have to be executed as in the trigonometric mode in order to determine Q and D (see Fig. 2). Only the values of the scaling constants differ. In the following we use the identities

$$\cosh(w) = (e^{w} + e^{-w})/2, \quad \sinh(w) = (e^{w} - e^{-w})/2, \quad \text{and} \quad e^{Q\ln(2)+D} = 2^{Q} e^{D}. \tag{3}$$

Furthermore, we insert Eqs. (3) into the hyperbolic rotation as given in Table I using the output data of step 1 and obtain

$$x_n = \{Mx_0''\,[\,(2^{Q} e^{D} + 2^{-Q} e^{-D})/2\,] + My_0''\,[\,(2^{Q} e^{D} - 2^{-Q} e^{-D})/2\,]\}\; 2^{Eref}$$

$$y_n = \{My_0''\,[\,(2^{Q} e^{D} + 2^{-Q} e^{-D})/2\,] + Mx_0''\,[\,(2^{Q} e^{D} - 2^{-Q} e^{-D})/2\,]\}\; 2^{Eref}$$

Substituting $e^{D} = \cosh(D) + \sinh(D)$ and $e^{-D} = \cosh(D) - \sinh(D)$ yields an output

$$x_n = \{[(Mx_0''+My_0'')+2^{-2Q}(Mx_0''-My_0'')]\cosh(D) + [(Mx_0''+My_0'') - 2^{-2Q}(Mx_0''-My_0'')]\sinh(D)\}\; 2^{Q-1}\, 2^{Eref}$$

$$y_n = \{[(Mx_0''+My_0'') - 2^{-2Q}(Mx_0''-My_0'')]\cosh(D) + [(Mx_0''+My_0'') + 2^{-2Q}(Mx_0''-My_0'')]\sinh(D)\}\; 2^{Q-1}\, 2^{Eref}$$

Using our notations of Fig. 1 the input data must be manipulated in the preprocessing stage before applying other steps, thus obtaining:

step 2:

$$Mx'_0 = (Mx_0'' + My_0'') + 2^{-2Q}(Mx_0'' - My_0'') \tag{4}$$

$$My'_0 = (Mx_0'' + My_0'') - 2^{-2Q}(Mx_0'' - My_0'')$$

step 3: $Mx_n = Mx'_0 \cosh(D) + My'_0 \sinh(D)$ $My_n = My'_0 \cosh(D) + Mx'_0 \sinh(D)$

step 4: $Mx'_n = 2^{Q-1}\, Mx_n$ $My'_n = 2^{Q-1}\, My_n$

step 5: $x_n = \text{Norm}(Mx'_n\, 2^{Eref})$ $y_n = \text{Norm}(My'_n\, 2^{Eref})$

III.3.2 ***Vectoring***

The function $\tanh^{-1}(v)$ is defined for $|v| < 1$ with poles at $|v| = 1$. The argument v of $\tanh(v)$ is limited by the value of $\tanh(1.113)=0.805$. For arguments in the interval [0.805, 1[the function cannot be directly calculated using the CORDIC algorithm. Because of this limited convergence domain the argument range of v is divided into two intervals to facilitate the calculation /2/. The interval is determined by testing the sign of $(Mx_0'' -2\,|My_0''|)$.

The first step is identical for both intervals.

step 1: $Eref = \max(Ex_0, Ey_0)$ $\quad Mx_0'' = \text{sign}(Mx_0) \cdot \text{Adj}(Mx_0, Eref)$ $\quad My_0'' = \text{sign}(My_0) \cdot \text{Adj}(My_0, Eref)$.

Interval 1: $0 \le |v| \le 0.5$ $\quad (Mx_0'' - 2\,My_0'' \ge 0)$

No transformation is necessary and the function can be directly computed using the CORDIC algorithm in a straightforward manner. However, we have to assure the proper sign of z_n:

step 5:

$$z_n = \text{sign}(Mx_0) \cdot \text{sign}(My_0) \cdot \text{Norm}(Mz'_n\ 2^{Ez_n}).$$

Interval 2: $0.5 < |v| < 1$ $\quad (Mx_0'' -2\,My_0'' < 0)$

The number v can be written as

$$v = 1 - \delta \quad \text{with } \delta = 1 - v = M\,2^{-E} \qquad 0.25 \le M < 1, \quad E \ge 1 \tag{5}$$

The identity $\tanh^{-1}(M\,2^{E}) = 0.5 \ln \dfrac{1 + M\,2^{E}}{1 - M\,2^{E}}$ yields the basic transformation for argument reduction, namely

$$\tanh^{-1}(1 - M\,2^{-E}) = \tanh^{-1}(T) + (E/2)\ \ln(2) \tag{6}$$

$$T = \frac{2 - M - M\,2^{-E}}{2 + M - M\,2^{-E}} \qquad \text{with } 0.2 < T < 0.78$$

This transformation treats only single argument function $\tanh^{-1}(v)$ /2/. Note that the argument T in Eq. (6) requires no division operation when using the CORDIC algorithm. But the two argument function $\tanh^{-1}(y_0/x_0)$ requires an additional division of the argument y_0/x_0 before the transformation in Eq. (6) can be employed. Now we give a division-free algorithm applicable to this two argument function. Substituting $v = My_0''/Mx_0''$ in Eq. (5), we find

$$My_0''/Mx_0'' = 1 - \delta_{new} \quad \text{with } 0.5 \le |My_0/Mx_0| < 1$$

with $\delta_{new} = (Mx_0'' - My_0'')/Mx_0'' = M_{new}\,2^{-Enew}$ $\quad$ or $\quad M_{new} = \dfrac{(Mx_0'' - My_0'')\,2^{Enew}}{Mx_0''}$.

Let γ represent the number of leading zeros in $(Mx_0'' - My_0'')$. Choosing

$$E_{new} = \begin{cases} 1 & \text{if} \quad \gamma = 1 \\ \gamma - 1 & \text{if} \quad \gamma > 1 \end{cases}$$

satisfies the conditions given by Eqs. (5) and (6). We insert the variable δ_{new} and the mantissa M_{new} into Eq. (6) and obtain

$$z_n = \tanh^{-1}\left(\frac{2 - \frac{(Mx_0'' - My_0'')\ 2^{Enew}}{Mx_0''} - \frac{(Mx_0'' - My_0'')}{Mx_0''}}{2 + \frac{(Mx_0'' - My_0'')\ 2^{Enew}}{Mx_0''} - \frac{(Mx_0'' - My_0'')}{Mx_0''}} \right) + \frac{E_{new}}{2} \ln(2) \qquad (7)$$

After some straightforward manipulations of Eq. (7) the following result can be found:

$$z_n = \tanh^{-1}\left(\frac{(Mx_0'' + My_0'') - (Mx_0'' - My_0'')\ 2^{Enew}}{(Mx_0'' + My_0'') + (Mx_0'' - My_0'')\ 2^{Enew}} \right) + \frac{E_{new}}{2} \ln(2) \qquad (8)$$

Using our notation we obtain the new CORDIC input variables My'_0 (nominator of the argument of $\tanh^{-1}$ in Eq. (8)) and Mx'_0 (denominator of the argument of $\tanh^{-1}$ in Eq. (8)) which are generated in the preprocessing stage of the argument reduction:

step 2 : $My'_0 = (Mx_0'' + My_0'') - (Mx_0'' - My_0'')\ 2^{Enew}$ (9)

$Mx'_0 = (Mx_0'' + My_0'') + (Mx_0'' - My_0'')\ 2^{Enew}$

step 3: $Mz_n = \tanh^{-1}(\frac{My'_0}{Mx'_0})$.

step 4: $Mz'_n = Mz_n + \frac{E_{new}}{2} \ln(2)$

step 5: $z_n = \text{sign}(Mx_0) \cdot \text{sign}(My_0) \cdot \text{Norm}(Mz'_n\ 2^{Ez_n})$.

When comparing this result with Eqs. (4) we can see that these equations are similar to those required for the input data Mx'_0 and My'_0 in rotation mode. The values of term $\frac{E_{new}}{2} \ln(2)$ in Eq. (8) have to be stored in a small ROM (size: E_{new} times n for n bit accuracy and $E_{new,max} = n$) which is implemented in the postprocessing stage of the argument reduction (**step 4**). Using the manipulated variables Mx'_0 and My'_0 of Eq. (9) we obtain for the hyperbolic norm:

step 3:
$$\begin{aligned} Mx_n &= \sqrt{Mx'^{\,2}_0 - My'^{\,2}_0} \\ &= \sqrt{((Mx_0'' + My_0'') + (Mx_0'' - My_0'')2^{Enew})^2 - ((Mx_0'' + My_0'') - (Mx_0'' - My_0'')2^{Enew})^2} \\ &= \sqrt{(Mx_0''^2 - My_0''^2)}\quad 2^{(Enew+2)/2} \end{aligned}$$

step 4: even exponent: $Mx'_n = Mx_n\ 2^{-(Enew+2)/2}$ if $E_{new} \mod 2 = 0$

odd exponent: $Mx'_n = (1/\sqrt{2})\ Mx_n\ 2^{-(Enew+1)/2}$ if $E_{new} \mod 2 = 1$

The term $(1/\sqrt{2})$ can be easily incorporated into the scaling factor compensation K_{-1} in the x data path. The correction by $2^{-(Enew+2)/2}$ and $2^{-(Enew+1)/2}$, resp., requires a shift operation similarly to the hyperbolic rotation mode (see Fig. 2). The operation 'E_{new} mod 2' can be realized according to the 'Q mod 4' operation in the trigonometric rotation mode. The FLP output is

step 5: $x_n = \text{Norm}(Mx'_n\ 2^{Eref})$.

IV. Architecture for pipeline or array implementations

In the proposed concept we avoid the use of CORDIC for realization of the argument reduction. This is an essential point because the use of CORDIC arithmetic for multiplication cannot achieve the same latency as a parallel array-multiplier because of its iterative, decision directed nature. Up to now all concepts use the CORDIC arithmetic for implementation of argument reduction operations.

Figure 2 depicts the floating point architecture of the proposed argument reduction process without considering the sign handling. The control variable S of the multiplexer (MUX) is a function of the parameter m and iteration mode, i.e. rotation or vectoring.

IV.1 *Area comparison*

The hatched blocks are necessary in all FLP CORDIC implementations in any case. Our proposal requires in addition the dashed functional blocks with the most chip area consuming parts being two constant multipliers, called carry save arrays in Fig. 2, and one ROM. The lower multiplier, which computes D, fits exactly the floorplan gap in the z-datapath, produced by the hardware necessary for scaling factor compensation in the x- and y-paths. The upper one produces only a moderate area overhead as it is placed besides an add-and-shift block in x and y. Therefore, the net overall chip area increase is due to one constant multiplier, five adders, three multiplexers, and one shifter in step 2 and one adder, one multiplexer, two shifters, and one ROM in step 4.

In this sense, we trade CORDIC iteration stages for constant multipliers. One row of a constant multiplier consists of full adders (3-to-2 cells) while a CORDIC x- or y- datapath built from redundant adders (i.e. carry save adders) requires two cascaded full adders (4-to-2 cells) and hardwired shifts. Therefore, the hardware in one CORDIC datapath for one add-and-shift operation consumes at least twice the area of one multiplier row. By applying Canonic Signed Digit (CSD) techniques and exploiting common sub-expressions (bit patterns) /13/ of the factors $2/\pi$, $1/\ln(2)$, etc., we have found that we need about $n/3$ full adder stages in each carry save array thus drastically reducing chip area.

It is interesting to note that the succeeding shifts in step 4 and 5 (for normalization) can be merged thus trading shifters for smaller adders/subtractors. This has not been depicted in Fig. 2 in order to keep the illustration of the basic architecture clear and simple.

Comparing the hardware amount of our concept and the FLP architectures in /7,12/ which merely implement the steps 1, 3, and 5 as mentioned above, our proposal requires one additional scaling operation only but delivers full convergence domain under all circumstances. Based on the chip presented in /12/ we estimate the chip area increase of the proposed concept to be less than 20%.

IV.2 *Latency comparison*

We assume the redundant addition scheme with constant scale factor given in Ref. /14/ for the CORDIC iterations in step 3, yielding a delay of $(2.25n + \log_2(n))\cdot\tau$ including redundant to 2´s-complement conversion (τ is the propagation delay time of an elementary full adder). A CLA (carry look ahead) adder exhibits a delay of $\log_2(n)\ \tau$ for an n bit addition. The multiplication time of the Booth-Wallace carry save array multiplier can be approximated as $2\log_2(n)\ \tau$ (approximately $\log_2(n)\ \tau$ for the Booth-Wallace array and the same for the CLA which merges the sum and the carry vector). Table V shows a comparison of Hu´s /9/ and Walther's method /2/ and our proposed concept of argument reduction. The division and the scaling operations for argument reduction require n iterations when using CORDIC arithmetic. The comparison demonstrates that the processing time can be decreased by using the proposed method.

V. Conclusion

In this contribution we described an efficient hardware algorithm for argument reduction of the CORDIC algorithm to provide unlimited convergence domain. We demonstrated a novel architecture with savings in computation time and a moderate increase in hardware complexity when compared to previous solutions. The CORDIC scaling factor compensation in the x and y data paths can be executed in parallel to one of the scaling operations in the z path and so it does not affect the overall latency time behaviour. The argument reduction algorithm in the x and y paths is nearly identical for both the hyperbolic rotation and hyperbolic vectoring modes and the trigonometric mode, resp., which facilitates hardware sharing. Interestingly enough, the hyperbolic and trigonometric vectoring mode require less hardware than the limited convergence expansion methods /4,5,9/.

References

/1/ J.E. Volder, "The CORDIC trigonometric computing technique," IRE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330-334, Sept.1959

/2/ J.S. Walther, "A unified algorithm for elementary functions," Spring Joint Computer Conference (SJCC) 1971, pp. 379-385, 1971

/3/ G.L. Haviland, A.A. Tuszynski, "A CORDIC arithmetic processor chip," IEEE J. Solid-State Circuits, vol. SC-15, no. 1, pp. 4-15, Feb. 1980

/4/ H.M. Ahmed, "Signal processing algorithms and architectures," Ph.D. dissertation, University of Stanford, June 1982

/5/ E.F. Deprettere, P. Dewilde, R. Udo, "Pipelined CORDIC architectures for fast VLSI filtering and array processing," Proceedings ICASSP 84, pp. 41.A.6.1.-41.A.6.5, 1984

/6/ N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," IEEE Trans. on Computers, vol. 40, no. 9, pp. 989-995, Sept. 1991

/7/ A.A. de Lange, E.F. Deprettere et.al., "An optimal floating-point pipeline CMOS CORDIC processor, algorithm, automated design, layout and performance," Proceedings of ISCAS'88, pp. 2043-2047, 1988

/8/ H. Yoshimura et.al., "A 50 MHz CMOS geometrical mapping processor," Digest of Technical Papers ISSCC'88, pp. 162-163 and 347-348, Febr. 1988

/9/ X. Hu, R.G. Haber, and S.C. Bass, "Expanding the range of convergence of the CORDIC algorithm," IEEE Trans. on Computers, vol. 40, no. 1, pp. 13-21, Jan. 1991

/10/ H. Hahn, "Untersuchung und Integration von Berechnungsverfahren elementarer Funktionen auf CORDIC-Basis mit Anwendungen in der adaptiven Signalverarbeitung," Fortschritt-Berichte VDI, Reihe 9, No. 125, VDI Verlag, Düsseldorf, 1991

/11/ D. Timmermann, H. Hahn, B. Hosticka, and G. Schmidt, "A Programmable CORDIC Chip for Digital Signal Processing Applications," IEEE Journal of Solid-Sate Circuits, vol. 26, no. 9, pp. 1317-1321, Sept. 1991

/12/ B. Rix, D. Timmermann, H. Hahn, and B.J. Hosticka, "A CORDIC-based floating-point arithmetic unit," IEEE Custom Integrated Circuits Conference (CICC´92), pp. 30.3.1-30.3.4, Boston, May 1992

/13/ R. Hartley, "Optimization of canonic signed digit multipliers for filter design," Proc. International Symposium on Circuits and Systems, pp. 1992-1995, Singapore, May 1991

/14/ D. Timmermann, H. Hahn, and B. Hosticka, "Low Latency Time CORDIC Algorithms," IEEE Trans. Comput., vol. 41, no. 8, pp. 1010-1015, Aug. 1992

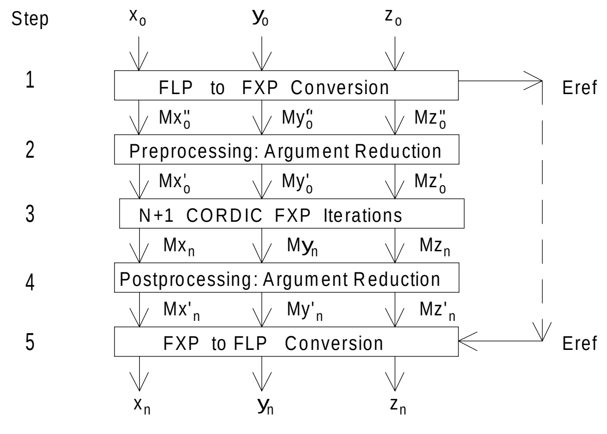

Fig. 1: Block diagram of the modified CORDIC architecture with argument reduction

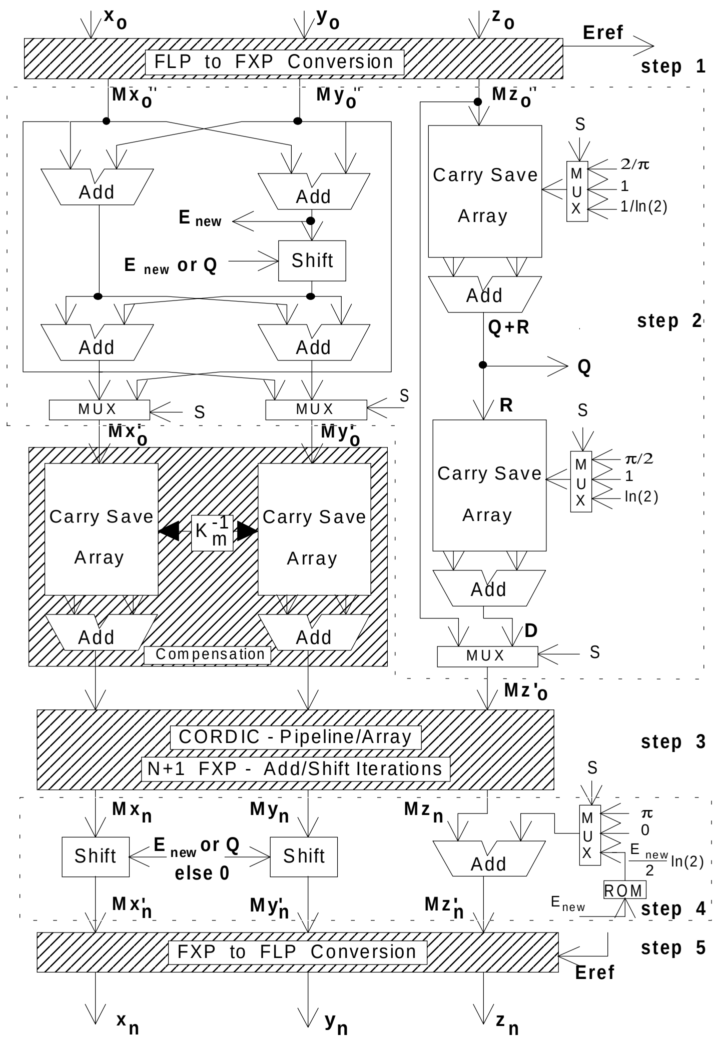

Fig. 2: Modified FLP CORDIC architecture with argument reduction

	$z_n \rightarrow 0$ (rotation)	$y_n \rightarrow 0$ (vectoring)
$m = -1$	$x_n = K_{-1}\,(x_0 \cosh(z_0) + y_0 \sinh(z_0))$ $y_n = K_{-1}\,(y_0 \cosh(z_0) + x_0 \sinh(z_0))$	$x_n = K_{-1}\sqrt{x_0^2 - y_0^2}$ $z_n = z_0 + \tanh^{-1}(y_0 / x_0)$
$m = 0$	$x_n = x_0$ $y_n = x_0\, z_0 + y_0$	$x_n = x_0$ $z_n = z_0 + y_0 / x_0$
$m = 1$	$x_n = K_1\,(x_0 \cos(z_0) - y_0 \sin(z_0))$ $y_n = K_1\,(y_0 \cos(z_0) + x_0 \sin(z_0))$	$x_n = K_1\sqrt{x_0^2 + y_0^2}$ $z_n = z_0 + \tan^{-1}(y_0 / x_0)$

Table I: CORDIC functions /2/

mode	*S(m,i)*	C_m	K_m
$m = 0$	0,1,2,3,4,5,...	2.0	1.0
$m = 1$	0,1,2,3,4,5,...	1.74	1.67
$m = -1$	1,2,3,4,4,5,...	1.113	0.83

Table II: CORDIC parameters

	Rotation	Vectoring
Input	$y_0 = 0$	$z_0 = 0$
Step 1	$Eref = Ex_0 + Ez_0$	$Eref = Ey_0 - Ex_0$
Step 3	$My_n = Mx'_0\, Mz'_0$	$Mz_n = My'_0 \,/\, Mx'_0$
Step 5	$y_n = \text{Norm}(\, My'_n\ 2^{\,Eref})$	$z_n = \text{Norm}(Mz'_n\ \ 2^{\,Eref})$

Table III: Summary of operations for $m = 0$

domain	Q mod 4	Mx'_0	My'_0
$[0, \pi/2[$	0 0	$+Mx_0''$	$+My_0''$
$[\pi/2, \pi[$	0 1	$-My_0''$	$+Mx_0''$
$[\pi, 3\pi/2[$	1 0	$-Mx_0''$	$-My_0''$
$[3\pi/2, 2\pi[$	1 1	$+My_0''$	$-Mx_0''$

Table IV: 'Modulo' operation, sign select of input data

method	C_m	block	iterations	latency [τ]	note
/17/	limited	C_m expansion	8	16	4 double add-and-shift
n= 16		CORDIC	16	$32 + \log_2(16)$	
		total	24		without K_m compensation
/2/	unlimited	argument reduction	n+3	$2n$+6	with CORDIC division
		CORDIC	n	$2n + \log_2(n)$	
		total	$2n$+3	$4n + \log_2(n) + 6$	without K_m compensation
proposed	unlimited	argument reduction	-	$2(\log_2(n/3)+\log_2(n))$	without division
		CORDIC	n	$2n + 2\log_2(n)$	includes step 4
		total	-	$2n + 6\log_2(n) - 3.17$	with K_m compensation (in parallel)

Table V: Comparison of argument reduction methods without FLP to FXP and FXP to FLP conversions

Index terms:

computer arithmetic, CORDIC, convergence domain, multiplier, VLSI

List of figure captions:

List of table captions: