AREA AND LATENCY EFFICIENT CORDIC ARCHITECTURES

D. Timmermann, I. Sundsbø***

* Fraunhofer Institute of Microelectronic Circuits and Systems
Finkenstr. 61, 4100 Duisburg 1, Germany

** SINTEF DELAB
7034 Trondheim, Norway

Abstract

The CORDIC algorithm has proved to be a powerful and flexible generic architecture to implement many signal processing and image processing algorithms. In practice, however, it sometimes suffers from its comparatively excessive silicon area demands. We present hardware solutions with reduced chip area requirements and power dissipation for parallel array or pipeline implementations of the unified algorithm (providing all known CORDIC functions) as well as for specialised architectures, supporting a subset of CORDIC functions. The chip area savings lie between 20% and 50%, respectively. In addition, these architectures result in lower latencies, typically by 25-50%, when compact non-redundant addition schemes like ripple-carry adders are employed.

I. Introduction

Signal processing algorithms and their industrial applications are two fields of rapidly growing importance. The main component in practical implementations is without doubt a fast multiplier, possibly in combination with an accumulator. Many signal processing methods, especially digital filter structures, can be efficiently mapped using this basic building block. However, there exist applications that require a more powerful and flexible instruction set to choose from. In some matrix processing applications, for example, CORDIC processing units have been shown to deliver superior performance when compared with more conventional approaches. This is due to the fact that many signal processing algorithms can be interpreted as generalized vector rotations, for which CORDIC is espescially suited to.

Considering the implementation part of signal processing systems, the main drawback of CORDIC-based pipeline or array architectures is their increased hardware complexity and latency time that exceed the corresponding values of array multipliers. In this paper we address CORDIC's hardware complexity and describe architectures with considerably reduced chip area requirements. It will be shown that under certain circumstances our architectures exhibit a lower latency time, too.

The organization of this contribution is as follows. First we formulate the CORDIC algorithm and investigate the dependency between its numerical accuracy and the width of the datapath. In the following section our modifications to the standard architecture are described. We characterize a unified architecture (providing the whole functionality of CORDIC at the expense of less area savings) and specific implementations that support a subset of these funtions, but with even lower chip area. Finally we show the improved speed of our architectures when they are implemented with non-redundant adders.

II. Numerical properties of the CORDIC algorithm

The CORDIC algorithm is defined by recurrences in three coordinates /1/:

$$x_{i+1} = x_i - m\,\sigma_i\,2^{-S(m,i)}\,y_i \qquad (1)$$

$$y_{i+1} = y_i + \sigma_i\,2^{-S(m,i)}\,x_i \qquad (2)$$

$$z_{i+1} = z_i - \sigma_i\,\alpha_{m,i} \qquad i = 0,1,\dots,N-1 \qquad (3)$$

where

m	coordinate system (m=1 means circular, m=0 linear, m=-1 hyperbolic)
$S(m,i)$	shift sequence ($S(0,i) = S(1,i) = 0,1,2,\dots,n$ $S(-1,i) = 1,2,3,4,4,5,6,\dots,3i,3i+1,3i+1,3i+2,\dots n$)
$\alpha_{m,i}$	rotation angle
σ_i	rotation direction, usually $\sigma_i \in \{-1,1\}$
n	precision in bit
N	number of iterations, $N = n$ for $m = 0$ and $m = 1$ (for $m = -1$ some extra iterations are necessary, see $S(m,i)$)

The rotation angle depends on $S(m,i)$ according to

$$\alpha_{m,i} = \frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,2^{-S(m,i)}) = \begin{cases} \tanh^{-1}(2^{-S(-1,i)}) & \text{for } m = -1 \\ \tan^{-1}(2^{-S(1,i)}) & \text{for } m = 1 \\ 2^{-S(0,i)} & \text{for } m = 0 \end{cases} \qquad (4)$$

Two operational modes are possible, rotation or vectoring. The rotation direction factor σ_i is determined by the following equation:

$$\sigma_i = \begin{cases} \operatorname{sign}(z_i) & \text{for } z_i \to 0 \ \text{(i.e. rotation)} \\ -\operatorname{sign}(x_i\,y_i) & \text{for } y_i \to 0 \ \text{(i.e. vectoring).} \end{cases} \qquad (5)$$

where $\operatorname{sign}(\lambda) = 1$ for $\lambda \geq 0$, else $\operatorname{sign}(\lambda) = -1$. The algorithm converges for all input data inside the region of convergence, given by

$$C_m = \sum_{i=0}^{N-1}\alpha_{m,i} \geq \begin{cases} \left|\frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,y_0/x_0)\right| & \text{for } y_n \to 0 \\ |z_0| & \text{for } z_n \to 0 \end{cases} \qquad (6)$$

The solution of the recurrences (1-3) is given in Table I with

$$k_m = \prod_{i=0}^{N-1}(1+m2^{-2S(m,i)})^{1/2} \qquad (7)$$

being the scaling factor and x_0, y_0, and z_0 the starting values of the iterations. The internal x,y data word length of n bit (one sign bit and n-1 fraction bit) has to be increased by $g_{xy} = \log_2(n)$ guard bits to suppress two sources of error: 1) finite word length effects when calculating $a \pm 2^{-i}b$ and 2) the impact of the remaining angle error α_N on x_N and y_N. In the same way the z datapath needs $g_z = \log_2(n/3)$ guard bits. This is due to the truncation of $\alpha_{m,i}$ to the length of the datapath and, second, caused by the remaining angle error α_N which affects z_N. In order to prevent overflow, the maximum iteration values in Eqs. (1-3) have to be considered. These are also summarized in Table I. Given the shift sequences $S(m,i)$ above and $|x_0|,|y_0| \leq 1$, $|z_0| \leq C_m$, the worst case in the x and y path occurs for the rotation mode and m = -1 and m = 0, respectively. The input values $x_0 = y_0 = 1$ yield x_{max} = 2,435 and y_{max} = 3, requiring o_{xy} = 2 overflow bits. Considering the z path, $|z_{max}|$ = 4 occurs for vectoring mode and m = 0. Thus, o_z = 3 overflow bits suffice. In summary, we need a machine word length of $w_{xy} = n+g_{xy}+o_{xy} = n+\log_2(n)+2$ bit for the x and y datapath and $w_z = n+g_z+o_z = n+\log_2(n/3)+3$ bit for the z datapath. A more detailed discussion of CORDIC's inherent quantization errors can be found in reference /6/.

0-7803-0593-0/92 $3.00 1992 IEEE

	$z_n \rightarrow 0$ (rotation)	$y_n \rightarrow 0$ (vectoring)
$m = -1$ $C_{-1}=$ 1.113 $k_{-1}=0.8$	$x_n = k_{-1}(x_0\cosh(z_0)+y_0\sinh(z_0))$ $y_n= k_{-1}(x_0\cosh(z_0)+y_0\sinh(z_0))$ $x_{max} = k_{-1}\, e^{C_{-1}} = 2.435$ $y_{max} = x_{max}$	$x_n = k_{-1}\sqrt{x_0^2 - y_0^2}$ $z_n = z_0+\tanh^{-1}(y_0/x_0)$ $x_{max} = x_{0,max} = 1$ $z_{max} = 2\, C_{-1} = 2.22$
$m = 0$ $C_0=2$ $k_0=1$	$x_n = x_0$ $y_n = x_0\, z_0 + y_0$ $x_{max} = 1$ $y_{max} = 1 + C_0 = 3$	$x_n = x_0$ $z_n = z_0 + y_0 / x_0$ $x_{max} = 1$ $z_{max} = 2\, C_0 = 4$
$m = 1$ $C_1=1.74$ $k_1=1.65$	$x_n = k_1\, (x_0 \cos(z_0)-y_0 \sin(z_0))$ $y_n= k_1\, (y_0 \cos(z_0)+x_0 \sin(z_0))$ $x_{max} = \sqrt{2}\, k_1 = 2.333$ $y_{max} = x_{max}$	$x_n = k_1\sqrt{x_0^2 + y_0^2}$ $z_n = z_0+\tan^{-1}(y_0/x_0)$ $x_{max} = \sqrt{2}\, k_1 = 2.333$ $z_{max} = 2\, C_1 = 3.74$

Table I: CORDIC functions, C_m, k_m, and maximum values

III. Reduction of chip area

A typical architecture that implements the recurrences (1-3) in a spatial array manner is depicted in Fig. 1. Each iteration occupies one row of the array. As we assume $N \cong n$ iterations and three datapaths with an internal word length of at least n bit (see above) the area complexity of a CORDIC array is proportional to $3n^2$. As an example, the required silicon area of a typical implementation like the IMSCOR24 /2/, a IEEE-754 single precision floating point CORDIC pipeline, exceeds 150 mm^2 in a 1.5 µm CMOS technology. Two thirds of this area are devoted to the iteration execution. Obviously, any reduction in chip area improves the yield and, consequently, decreases production costs considerably. The motivation for this work is, therefore, to reduce the chip area but with no speed penalty. Interestingly enough, our proposed architectures exhibit less latency, too.

The key to chip area reduction lies in several observations. First, when investigating the computation carried out in the z path, we note that with increasing iteration index i the w_z bit addition/subtraction of the angles $\alpha_{m,i}$ and the current value of z_i actually reduces to the addition/subtraction of a w_z bit data word and one bit data only. Second, considering the decreasing magnitudes of y_i and z_i in vectoring and rotation mode, respectively, inspires us to also decrease the width of the corresponding datapaths.

3.1. Unified algorithm

Based on these ideas we now describe an area efficient architecture for the unified CORDIC algorithm, that provides all CORDIC functions /3/. The rotation angle can be written as

$$\alpha_{m,i} = \begin{cases} 2^{-S(-1,i)} + \frac{1}{3}2^{-3S(-1,i)} + \ldots & \text{for } m = -1 \\ 2^{-S(1,i)} - \frac{1}{3}2^{-3S(1,i)} + \ldots & \text{for } m = 1 \\ 2^{-S(0,i)} & \text{for } m = 0 \end{cases}$$

using expansion series and reduces to $\alpha_{m,i} = 2^{-S(m,i)}$ for $S(m,i) \geq n/3$ within n bit accuracy when rounding to the nearest is applied. Therefore, Eq. (3) may be written as

$$z_{i+1} = z_i - \sigma_i 2^{-S(m,i)} \quad \text{for } S(m,i) \geq n/3$$

or

$$z_N = z_j - \sum_{i=j}^{N-1} \sigma_i 2^{-S(m,i)} \qquad (8)$$

where $j = \lceil n/3 \rceil$. Eq. (8) means that the final iteration value z_N can be computed from z_j by subtraction of a N-j bit signed-digit data word. For example, with j=3, N=8, σ_3=1, σ_4=-1, σ_5=-1, σ_6=1, and σ_7=-1 we obtain $z_8 = z_3 - 2^{-3} + 2^{-4} + 2^{-5} - 2^{-6} + 2^{-7}$.

The signed-digit representation can be converted into the usual 2's - complement representation by subtraction of two data words, one containing the negative digits being subtracted from the other which contains the positive digits only. Thus, to evaluate Eq. (8) two additions/subtraction would be required. However, the same result can be achieved by the following calculation

$$z_N = z_j - \sum_{i=0}^{n-1} W_i 2^{-i} \quad \text{with } W_i = \begin{cases} 0.5(1-\sigma_j) & \text{for } 0 \leq i < j \\ 0.5(\sigma_{i+1}+1) & \text{for } j \leq i < n\text{-}1 \\ 1 & \text{for } i = n\text{-}1, \end{cases} \qquad (9)$$

which saves one addition operation. The expressions $0.5(\sigma_i+1)$ $\{0.5(1-\sigma_j)\}$ yield 0 {1} for σ_i=-1 and 1 {0} for σ_i=1 and, hence, can be implemented by simple bit flipping. Eq. (9) applied to the example above yields $z_8 = z_3$ - 0.0000101.

Now we are able to compute z_N from z_j using one subtraction, provided the σ_i are known in advance. This is not generally the case with the standard CORDIC algorithm, as the σ_i are computed one after another by evaluating the sign of the iteration values as defined in Eq. (5). In *vectoring mode* the σ_i can be obtained in the usual, sequential manner. The evaluation of z_N according to Eq. (9) commences after the iterations have been finished. The *rotation mode*, however, poses a problem, because for $i \geq j$ we no longer have any z_i to derive σ_i from. Alternatively, we use the parallelized CORDIC algorithm, presented in /4/. There, a fully parallel technique to recode the bits of

$$z_j = -Z_0 + \sum_{k=1}^{n-1} Z_k 2^{-k}$$

into $\sigma_j \ldots \sigma_{N-1}$ is described. The recoding rules are given in Table II /4/.

Z_{i-1}	Z_i	σ_i
0	0	-1
0	1	-1
1	0	1
1	1	1

Table II: Recoding rule

We note that the recoding is performed in a very small logic block with about one gate delay. This implies that the speed of our architecture will not decrease compared with the standard method.

The proposed unified CORDIC architecture, as depicted in Fig. 2, clearly leads to a chip area reduction in the z datapath by about 2/3, resulting in overall savings of about 20% without loss in design regularity. The logic block labeled R implements the recoding according to Table II and the block labeled S implements the bit conversion as defined by Eq. 8.

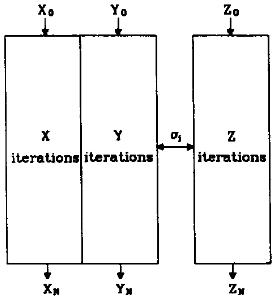

Fig. 1: Standard CORDIC array

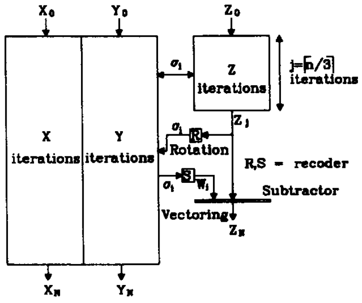

Fig. 2: Area-reduced CORDIC array

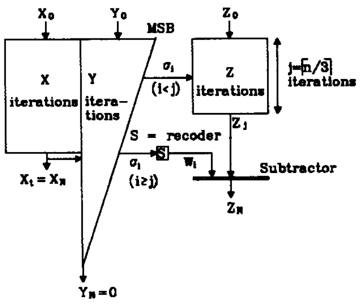

Fig. 3: Vectoring mode architecture

3.2. Vectoring mode architecture

An architecture, especially dedicated to the vectoring mode, can employ the just described technique to decrease the number of iteration stages in the z path and, in addition, benefit from the decreasing magnitude of y_i during iteration. In each iteration, the magnitude of y_i decreases by roughly one bit position. In two's complement notation $y_i = 0.0000...01xxx$ for positive y_i and $y_i = 1.1111...10xxx$ for negative y_i. Therefore, we use the minimum word length necessary to implement the recurrence (2) without loss in precision and omit the leading zeros and ones. A tight upper limit on the magnitude of y_i can be obtained from the expression

$$\left|\frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,y_i/x_i)\right| \le \sum_{k=i}^{N-1}\alpha_{m,k} \ .$$

However, it seems difficult to derive an analytical formula for y_i from this equation. Instead we use the relaxed condition

$$\left|\frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,y_i/x_i)\right| \le \alpha_{m,i-1} = \frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,2^{-S(m,i-1)})$$

which yields $|y_i| \le 2^{-S(m,i-1)}\,x_i$. Thus the following relationship holds

$$|y_{i,max}| \le \begin{cases} 2^{-S(m,i-1)} & \text{for } m = -1,0 \\ 2^{-S(m,i-1)+1.22} & \text{for } m = 1 \end{cases} \ . \qquad (10)$$

This result enables us to minimize the machine's word length in the y path. Obviously about one half of the y path can be omitted without deteriorating the performance of the algorithm. Note that tighter bounds on the magnitude of y_i exist. We used this one to illustrate the basic idea and because of its simplicity. The chip area savings will only marginally improve with tighter bounds.

It can be shown, however, that even more area reductions are possible. As has been shown in /5/, it is not necessary to iterate x_i N times in vectoring mode. Instead it suffices to abort the iteration process after $t = \lceil n/2 \rceil$ iterations (for $m = -1$ we have to take into account the extra iterations). Then it holds within n bit precision that $x_N = x_t$. This can be easily verified when refering to Eq. (1) with $S(m,i) \ge n/2$. Consequently, the second half of the x data path can be omitted, too. The resulting chip architecture is depicted in Fig. 3. A more favourable, rectangular shaped floorplan is shown in Fig. 4. The x and y paths are bitwise interleaved and folded. The resulting overall chip area savings amount to 50% compared with the conventional approach or, alternatively speaking, permit an 40% increase in word length on the same silicon area.

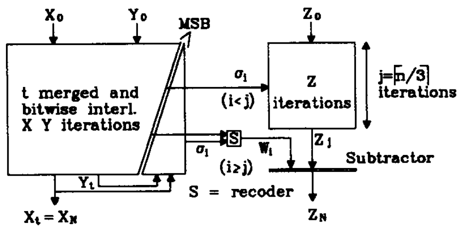

Fig. 4: Improved vectoring mode architecture

3.3. Rotation mode architecture

Two strategies are applicable in rotation mode for hardware reduction in the z path. In the same way as in the vectoring mode we can restrict the length of the z datapath to its iteration dependent minimum. The magnitude of z_i is upper-bounded by

$$|z_i| \le \alpha_{m,i-1} \le 2^{-S(m,i-1)} \ , \qquad (11)$$

requiring about 50% less hardware in the z path than before. The corresponding architecture is shown in Fig. 5.

Less z path iteration stages are necessary when we apply the z path reduction technique discussed in Section 3.1. This method /4/, characterized by the parallel generation of $\sigma_j...\sigma_{N-1}$ from the bits of z_j, needs only $j = \lceil n/3 \rceil$ iteration stages in the z path. The remaining N-j iterations can be omitted.

In addition, hardware savings in the x and y path are feasible as the parallel generation of $\sigma_j...\sigma_{N-1}$ permits a Booth-type recoding of the rotation directions to halve the number of σ_i unequal to zero. This extends the valid digit set for σ_i from $\{-1,1\}$ to $\{-1,0,1\}$. To avoid any impact on the scaling factor k_m (Eq. 7) and, consequently, an unpredictible vector norm variation, the recoding begins with the first iteration whose shift $S(m,i)$ exceeds $n/2$. In this way, in the second half of the x and y iterations two subsequent stages can be merged into one. The recoder selects the proper iteration shift by means of a 2-to-1 multiplexer.

Such strategy results in an architecture depicted in Fig. 6. We estimate the overall hardware reduction to exceed 30%, as the silicon estate for the extra components, recoder and multiplexers, can be neglected.

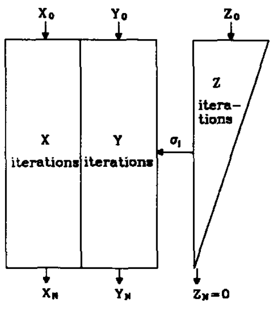

Fig. 5: Rotation mode architecture

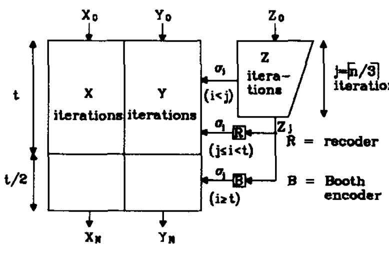

Fig. 6: Improved rotation mode architecture

IV. Latency time considerations

Until now, we have investigated the hardware reduction effect of the proposed architectures. Provided we implement the iterations using carry-dependent adders we obtain lower latency times for the special vectoring and rotation mode architectures compared with unmodified ones. This is due to the decreased word lengths in y_i or z_i which reduce the number of possible carry propagations. In this section we derive upper bounds on the carry propagation delay of our architectures. In our considerations it will be assumed that w bit ripple carry adders are employed which exhibit a latency time of at most w full adder time units (TU). To simplify the calculation we consider $m = 0$ and $m = 1$ only. The results for the hyperbolic CORDIC ($m = -1$) differ only marginally.

4.1. Latency time in rotation mode

Eq. 11 yields $|z_i| \leq 2^{-i+1}$ when using the shift sequence $S(1,i)$. In the following we do not regard the influence of σ_i as it is assumed to be available early enough due to the parallelization scheme discussed above. Thus the time for computing z_i equals $t_{Rotz,i} = w_z\text{-}i\text{-}1 = n+\log_2(n/3)+2\text{-}i$ TU. Now we have to consider the influence of the x- and y-path. In the i-th iteration the number of shifts of x_i with respect to y_i and vice versa is equal to i. Then the LSB (bit 0) of x_i can be calculated as soon as bit number i -1 from $y_{i\text{-}1}$ is determined (and similarly for y_i). The maximum carry propagation time for the i-th iteration is thus equal to the number of shifts plus 1 or $t_{Rotxy,i} = S(m,i) + 1 = i + 1$. Then the time to compute the result of the i-th iteration is given by $t_{Rot,i} = \max(t_{Rotz,i}, t_{Rotxy,i})$.

At first the latency is determined by the z-path. After $k = 0.5(n+\log_2(n/3)+1)$ iterations the latency of the x- and y-paths dominates. The total latency time amounts to

$$t_{Rot} = \sum_{i=0}^{N\text{-}1} t_{Rot,i} = \sum_{i=0}^{k} w_z\text{-}i\text{-}1 + \sum_{i=k+1}^{N\text{-}1} i+1 \cong 0.75\, n^2 + (n+1)/2 \log_2(n/3) + n\,,$$

about 25% less than with the standard architecture $w_z^2 \cong n^2+(2n+6)\log_2(n/3)$ TU).

4.2. Latency time in vectoring mode

In this mode σ_i is derived from the sign of y_i and the interdependency between the x- and y-path makes the latency considerations somewhat more complicated.

Eq. 10 yields $|y_i| \leq 2^{-i+2.22}$ when employing the shift sequence $S(1,i)$. The maximum time for computing the j-th bit (note: MSB = w_{xy}-1) of y_{i+1} and x_{i+1} equals

$$t_{Vecty_{i+1},\,j} = \max(t_{Vecty_i,\,MSB}\,;\, t_{Vectx_i,\,i}) + j - i + 1$$

$$t_{Vectx_{i+1},\,j} = \max(t_{Vectx_i,0}\,;\, t_{Vecty_i,\,MSB}) + j + 1.$$

During the first iterations it can be shown by some numerical examples that for reasonable word lengths w_{xy} the latency time $t_{Vecty_i,\,MSB}$ dominates. Therefore, the equations above reduce to

$$t_{Vecty_{i+1},\,MSB} = t_{Vecty_i,\,MSB} + w_{xy} - i = (i+1)w_{xy} - \sum_{k=1}^{i} k$$

$$t_{Vecty_{i+1},\,MSB} = (i+1)w_{xy} - i(i+1)/2$$

$$t_{Vectx_{i+1},\,j} = i\, w_{xy} - i(i\text{-}1)/2 + j + 1.$$

The prerequisite that $t_{Vecty_i,\,MSB}$ dominates is valid until $t_{Vecty_i,MSB} = t_{Vectx_{i+1}j}$ and, hence,

$$(i+1)w_{xy} - i(i+1)/2 = i\, w_{xy} - i(i\text{-}1)/2 + j + 1.$$

Due to the shift sequence given j equals $i + 1$. Then we obtain that after $i = (w_{xy} - 2)/2 = n/2+\log_2(n)/2$ iterations the latency time of the x-path would dominate. However, as has been described in Section 3.2, x_i is constant within n bit precision after $t = \lceil n/2 \rceil$ iterations. Therefore, the carry propagation in the x-path will never have an impact on the total latency time which now amounts to (note that $N = n$)

$$t_{Vect} = t_{Vecty_N,\,MSB} = n^2/2 + 5n/2 + n\log_2(n) \text{ TU},$$

and is nearly 50% faster than the original architecture ($n^2 + n\log_2(n) + 2n$ TU).

Conclusion

This contribution has treated two main issues connected with development of CORDIC hardware. The excessive hardware demands of parallel CORDIC architectures have been addressed first and modifications to the standard architectures have been introduced that reduce the hardware amount by 20% for the unified algorithm, 30% for the rotation mode, and 50% for the vectoring mode. The values are valid for implementations with both redundant as well as with non-redundant adders. The chip area savings and associated power dissipation decrease are accompanied by speedups for implementations with non-redundant adders between 25% (rotation mode) and 50% (vectoring mode) compared with the standard architecture.

References

/1/ J.S. Walther, "A unified algorithm for elementary functions," *Proc. of Spring Joint Computer Conference*, pp. 379-385, 1971

/2/ Preliminary Information, "IMSCOR24, floating point arithmetic function evaluator," Fraunhofer Institute of Microelectronic Circuits and Systems, D-4100 Duisburg, Federal Republic of Germany, Sept. 1991

/3/ D. Timmermann, H. Hahn, B.J. Hosticka, "Schaltungsanordnung und Verfahren zur Durchführung des CORDIC-Algorithmus," German Pat. pend., 1991

/4/ D. Timmermann, H. Hahn, B.J. Hosticka, "A low latency time CORDIC algorithm with increased parallelism," *Proceedings of ISCAS'91*, Singapore, pp. 2975-2978, June 1991

/5/ D. Timmermann, H. Hahn, B.J. Hosticka, "Modified CORDIC algorithm with reduced iterations," *Electronics Letters*, Vol. 25, Nr. 15, pp. 950-951, July 1989

/6/ B. Yang, "Untersuchungen zu einem 32-Bit-Gleitkomma CORDIC-Prozessor," Diploma thesis, Department of Electrical Engineering, Ruhr-University Bochum, July 1986