Low Latency Time CORDIC Algorithms

Dirk Timmermann, Helmut Hahn, and Bedrich J. Hosticka, *Senior Member, IEEE*

***Abstract*— In this contribution we present several methods for increasing the speed of the CORDIC algorithm. First we develop an improved method which guarantees a constant scale factor when employing redundant addition schemes. Then an architecture with increased parallelism will be described which considerably reduces the CORDIC latency time and the amount of hardware.**

***Index Terms*—CORDIC, computer arithmetic, function generation, hardware algorithm, modified Booth's algorithm, redundant number representation.**

I. Introduction

THE CORDIC algorithm [1] is a well-known method to compute iteratively magnitude and phase or the rotation of a vector in circular, linear, and hyperbolic coordinate systems. The execution of CORDIC iterations is completely multiplier-free and requires only shift and add operations. The iteration equations are

$$x_{i+1} = x_1 - m\sigma_i 2^{-S(m,i)} y_i \tag{1}$$

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

$$z_{i+1} = z_i - \sigma_i \alpha_{m,i} \tag{3}$$

where m denotes the coordinate system, σ_i the rotation direction, $S(m,i)$ the shift sequence, and $\alpha_{m,i}$ the rotation angle. The latter directly depends on $S(m,i)$ according to

$$\alpha_{m,i} = \frac{1}{\sqrt{m}} \tan^{-1}\left(\sqrt{m} 2^{-S(m,i)}\right). \tag{4}$$

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

$$\sigma_i = \begin{cases} \operatorname{sign}(z_i) & \text{for } z_n \to 0 \text{ (rotation)} \\ -\operatorname{sign}(x_i) \cdot \operatorname{sign}(y_i) & \text{for } y_n \to 0 \text{ (vectoring)} \end{cases} \tag{5}$$

where $\operatorname{sign}(\eta) = 1$ for $\eta \geq 0$, else $\operatorname{sign}(\eta) = -1$. The solution of the iterations is given by (7), with

$$k_m = \prod_i \sqrt{1 + m\sigma_i^2 2^{-2S(m,i)}} \tag{6}$$

being the scaling factor and x_0, y_0, and z_0 the starting values of the iterations. The final values of x, y, and z are given by (7), shown at the top of the next page.

By selecting the appropriate coordinate system ($m = 0, 1, -1$ means operations in linear, circular, and hyperbolic systems, respectively) and iteration mode, we obtain a variety of computable functions, unmatched in terms of variety and simplicity by other unified algorithms. On the other hand, CORDIC belongs to the digit-by-digit algorithms with linear convergence and sequential behavior. That means for n bit precision we need approximately n iterations, with the constraint that the $(i+1)$th iteration may only commence after the ith has been completed. Equations (1)–(3) indicate that the most dominating speed factor during the iteration is the addition/subtraction operation, because its lower bound in nonredundant number systems is usually proportional to the word length n due to carry propagation. Therefore, the use of redundant addition schemes has been advocated recently, in particular, redundant binary [2] (a radix two Signed-Digit (SD) adder) and carry save adders [3] which both yield an addition time independent of n and approximately equal to 2τ where τ denotes the delay time of one full adder.

The use of redundant numbers in CORDIC arithmetics appears rather attractive but σ_i has to be estimated from the inspection of some of the most significant digits. However, when all the digits inspected are zero, the proper value of σ_i cannot be determined without the knowledge of the remaining digits. In this case, the best strategy seems to be to assign the value zero to σ_i thus freezing the iteration. A comparison with (6) demonstrates, however, that $\sigma_i = 0$ affects the value of k_m thus making it data-dependent. Several algorithms have been proposed to circumvent this problem but all these solutions increase both latency time and chip area by at least 50%. In addition, the algorithm still requires n sequential decisions to generate all σ_i thus preventing a parallelization.

This contribution will address the issue of increasing the CORDIC speed. The organization of this paper is as follows. First, we describe an improved method for CORDIC using redundant adders that yields a constant scale factor. Afterwards, an algorithm for parallelizing the determination of σ_i will be described that considerably reduces CORDIC latency time in rotation mode. The latency time may be reduced even further when applying other speedup concepts like the constant scale factor algorithm, a termination algorithm, or Booth encoding to this algorithm, as will be shown.

II. Constant Scale Factor and Redundant CORDIC

In this discussion, we concentrate on spatial array or pipelined implementations of the CORDIC algorithm. Then, in all known algorithm variations the rotation direction $\sigma_i \in \{0, 1, -1\}$ is derived from the inspection of the p ($p \ll n$) most significant digits (MSD). The choice $\sigma_i = 0$, however, is avoided in order to keep the scale factor constant. Takagi [2] suggested a halving of each iteration and executing it twice. Instead of rotating by an angle of $\tan^{-1}\left(2^{-i}\right)$ during the ith

Manuscript received October 7, 1991; revised February 24, 1992.

The authors are with the Fraunhofer Institute of Microelectronic Circuits and Systems, Finkenstr. 61, D-4100 Duisburg 1, Germany.

IEEE Log Number 9201449.

$$z_n \rightarrow 0 \text{ (rotation)} \qquad y_n \rightarrow 0 \text{ (vectoring)}$$

$$\begin{aligned} x_n &= k_m \cdot \left(x_0 \cos\left(\sqrt{m} z_0\right) - \sqrt{m} y_0 \sin\left(\sqrt{m} z_0\right)\right) & x_n &= k_m \sqrt{\left(x_0^2 + m y_0^2\right)} \\ y_n &= k_m \cdot \left(y_0 \cos\left(\sqrt{m} z_0\right) + \frac{1}{\sqrt{m}} x_0 \sin\left(\sqrt{m} z_0\right)\right) & z_n &= z_0 + \frac{1}{\sqrt{m}} \cdot \tan^{-1}\left(\sqrt{m} y_0 / x_0\right). \end{aligned} \tag{7}$$

iteration he carries out two smaller rotations (e.g., z-iteration for $m = 1$ and $S(m, i) = i$):

For $\sigma_i = \pm 1 \rightarrow \sigma_i \tan^{-1}\left(2^{-i-1}\right)$ and $\sigma_i \tan^{-1}\left(2^{-i-1}\right)$

For $\sigma_i = 0 \rightarrow +\tan^{-1}\left(2^{-i-1}\right)$ and $-\tan^{-1}\left(2^{-i-1}\right)$.

The two successive rotations per iteration can be merged into one redundant three-input addition/subtraction operation [2], requiring two 4-to-2 cells forming a 6-to-2 cell (a redundant adder with two redundant inputs (two bits each) is called a 4-to-2 cell, whereas a full-adder (carry-save adder) represents a 3-to-2 cell). Thus also the chip area doubles. The hardware and latency time increase can be reduced when employing the algorithms in [4] and [5] for $m = 1$ and $m = -1$, respectively. By inspecting at most $p + 2$ MSD's of a redundant binary representation, a repetition of each pth iteration suffices to ensure convergence. The latency time of the inspection process increases with p as it is usually implemented using a fast carry-dependent adder. Therefore, p should not exceed about four or five digits so that the MSD-inspection remains faster than the redundant addition in the iteration. Using carry-save arithmetic, a doubling of every second iteration is necessary when inspecting four MSD's [6], [7].

The comparison demonstrates that the requirement for a constant scale factor results in a chip area and latency time increase of at least 50% [6]. The algorithm that will be described now reduces considerably this overhead.

The main idea of our algorithm is to allow $\sigma_i = 0$ as a valid choice during iterations. Instead of rotating the vector we increase ($m = 1$) or decrease ($m = -1$) the length of the vector by the scale factor each time we obtain a $\sigma_i = 0$ from the inspection of the MSD's. The procedure depends on i and is as follows (assume $S(m, i) = i$ in the following):

—$0 \leq i \leq (n-3)/4$: execute regular iterations in the same manner as in the known algorithms, for example, [6].

—$(n-3)/4 < i \leq (n+1)/2$:

$\sigma_i <> 0$: execute (1)–(3)

$\sigma_i = 0$: $x_{i+1} = x_i + m2^{-2i-1} x_i$ (8)

$y_{i+1} = y_i + m2^{-2i-1} y_i$ (9)

$z_{i+1} = z_i$

—$(n+1)/2 < i$:

$\sigma_i <> 0$: execute (1)–(3)

$\sigma_i = 0$: $x_{i+1} = x_i$

$y_{i+1} = y_i$

$z_{i+1} = z_i$.

The algorithm exploits the fact that for $i > (n-3)/4$ the relation [cf. (6)] $\sqrt{1 + m2^{-2i}} = 1 + m2^{-2i-1}$ holds within n bit precision. For $i > (n+1)/2$ the scale factor does not affect the vector length any longer (within machine accuracy). The only additional component necessary to implement the different types of iteration equations is a multiplexer. The bit shifting is hard-wired. Thus, the proposed method is very hardware efficient and fast. When the algorithm in [6] is employed for the first $(n-3)/4$ iterations, we need about $1.5 \cdot (n-3)/4 + n - (n-3)/4 = (9n-3)/8$ iterations (or $(9n-3)/4\ \tau$) compared to at least $3n/2$ iterations ($3n\,\tau$) without our method. The speedup and chip area savings amount to about 25% each.

III. Parallelizing the Generation of σ_i by Prediction

The main difference between CORDIC and, for example, multipliers exists in the sequential nature of the iteration process. In each iteration the σ_i for the next rotation has to be determined. The *a priori* knowledge of σ_i could prompt an attempt to parallelize the iterations and reduce the latency time. Therefore, we propose an algorithm [8] that is based on Baker's prediction scheme [9].

To explain the underlying idea, let us consider the rotation mode and $m = 0$, yielding a multiply-and-add operation: $y_n = y_0 + z_0 x_0$. The initial value z_0 is given in binary notation

$$z_0 = \sum_i Z_i 2^{-i} \qquad Z_i \in \{0, 1\}.$$

On the other hand, z_0 is decomposed during the process of pseudodivision into [see (3) and (4)] for $m = 0$ and assume $S(m, i) = i$):

$$z_0 = \sum_i \sigma_i 2^{-i} \qquad \sigma_i \in \{-1, 1\}, m = 0.$$

This means that the original iteration process defined by (1)–(4) that is necessary for finding proper σ_i's can be replaced by *recoding* the binary representation of Z_i's in the SD notation of σ_i's. Using a simple conversion rule, we can directly obtain all σ_i from z_0. Since this can be carried out in parallel, significant speed improvements can be achieved when using this method. The bit conversion rule is given in Table I. The leftmost digit of the recoded sequence is defined to be 1 (-1) for positive (negative) numbers. One example for the decimal number 45 that should clarify the conversion is shown in Table II.

TABLE I
BIT CONVERSION RULE

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

TABLE II
CONVERSION EXAMPLE FOR 45_{10}, $(\bar{1} = -1)$

Binary z_i	0 0 1 0 1 1 0 1
Signed digit σ_i	1 $\bar{1}$ 1 $\bar{1}$ 1 1 $\bar{1}$

But in circular and hyperbolic coordinate systems the conversion is complicated due to the more complex decomposition:

$$z_0 = \sum_i \sigma_i \alpha_{m,i} = \sum_i \sigma_i \frac{1}{\sqrt{m}} \tan^{-1}\left(\sqrt{m} 2^{-S(m,i)}\right).$$

The direct application of the conversion rule is not possible as $\alpha_{m,i}$ is not equal to 2^{-i}. On the other hand, the difference between 2^{-i} and $\alpha_{m,i}$ gets smaller with increasing iteration index i. This is demonstrated in Table III (again $S(m,i) = i$).

It is possible to apply the conversion rule for $m = \pm 1$ with the constraint that the resulting prediction error will be corrected. In the following section, an upper bound for the prediction error will be derived.

Let us assume z_j to have $j-1$ leading zeros in front of a string of $k-j+1$ ones, i.e., z_j = 0.00..011..1. When we apply the conversion rule according to Table I to the iterations between j and k, the error that results from this approximation has to be smaller than or equal $2^{-S(m,k)}$. In this case, an error correction by a repetition of the kth iteration guarantees k leading zeros. These considerations yield an estimate for the maximum prediction error:

$$\sum_{i=j}^{k} \left| 2^{-i} - \frac{1}{\sqrt{m}} \tan^{-1}\left(\sqrt{m} 2^{-S(m,i)}\right) \right| \leq 2^{-S(m,k)}.$$

This equation establishes a relationship between the maximum value of j and k. Evaluating the equation above for different integer values of $j, S(m,i) = i$, and $m = \pm 1$ yields the results shown in Table IV.

Obviously, for $j > 0$ the relationship

$$k \leq 3j + 1 \tag{10}$$

holds.

Thus, it has been shown that starting from the jth iteration the fully parallel bit conversion rule for determining σ_i may be applied for the next $2j+2$ iterations (i.e., for iterations between j and $3j+1$), followed by a repetition of the last iteration in order to correct any possible errors.

In Fig. 1 an architecture for $n = 39$ bit accuracy is depicted which implements the algorithm with σ_i prediction starting with $i = 1$. The nonredundant adders serve as converters from redundant to conventional binary notation. From (10) it can be concluded that about $\log_3(n) - 1$ conversions are necessary. The recoder R generates $\sigma_i \in \{-1, 1\}$ from the bits of z_i according to the conversion rule given in Table I. Most of the additions/subtractions are executed by redundant adders, indicated by the boxes with diagonal arrows. In the x/y-data paths we employ 4-to-2 cells while in the z-path 3-to-2 adders suffice.

The main advantage of this architecture is its feasibility to incorporate more parallelism. The *additive* iteration equation for z_i [given by (3)] now can be parallelized using a Wallace tree without increasing the hardware amount and, hence, the carry-dependent additions do not affect the overall latency time behavior. The x/y-paths become the speed-dominating part of the algorithm, as they cannot be easily parallelized as the z-path without hardware increase due to the *multiplicative* nature

TABLE III
ASSIMILATION OF $\alpha_{m,i}$ WITH INCREASING i

	$\alpha_{m,i}$		
i	$m = 1$	$m = 0$	$m = -1$
0	0.785398163	1.000000000	—
1	0.463647609	0.500000000	0.549306144
2	0.244978663	0.250000000	0.255412811
3	0.124354994	0.125000000	0.125657214
4	0.062418810	0.062500000	0.062581571
5	0.031239833	0.031250000	0.031260178

TABLE IV
RELATIONSHIP BETWEEN j AND k ($*m = 1$ ONLY)

j	k
0*	1*
1	4
4	13
13	40
40	121

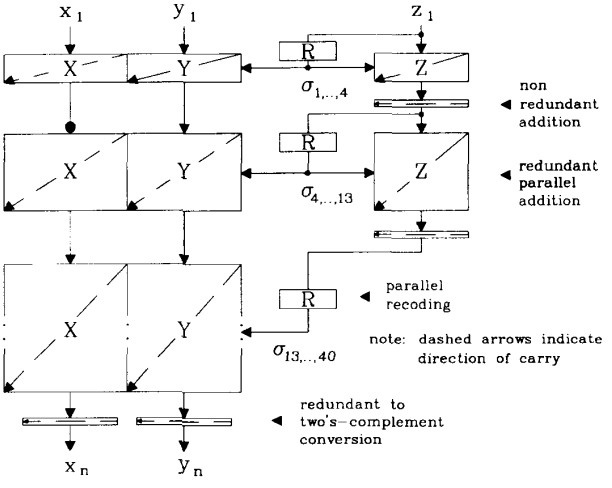


Fig. 1. Architecture of CORDIC with σ_i prediction.

TABLE V
ADDITIONAL OPERATIONS NEEDED FOR THE GENERALIZED TERMINATION ALGORITHM

Rotation mode	$x_n = x_j - mz_jy_j$ $y_n = z_jx_j + y_j$	for $j > (n+1)/2$
Vectoring mode	$x_n = x_j$ $z_n = z_j + y_j/x_j$	for $j > (n+1)/2$ for $j > (n/3) + 0.472$

of the underlying iteration:

$$\begin{bmatrix} x_{i+1} \\ y_{i+2} \end{bmatrix} = \begin{bmatrix} 1 & -m\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

If we want to compute $(x_{i+2}, y_{i+2})^T$ from $(x_i, y_i)^T$ we obtain

$$\begin{bmatrix} x_{i+2} \\ y_{i+2} \end{bmatrix} = \begin{bmatrix} 1 - m\sigma_i\sigma_{i+1}2^{-2i-1} & -m\left(\sigma_i 2^{-i} + \sigma_{i+1}2^{-i-1}\right) \\ \sigma_i 2^{-i} + \sigma_{i+1}2^{-i-1} & 1 - m\sigma_i\sigma_{i+1}2^{-2i-1} \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix}.$$

An approach to parallelize this operation would require (matrix) multipliers instead of simple adds and shifts and will not be considered here.

The recoding suppresses $\sigma_i = 0$ and any scale factor problems associated with it. About $\log_3(n) - 1$ iteration doublings are necessary. Therefore, an estimate for the latency time in terms of full adder time units yields $2n + \log_3(n) - 1 + \log_2(n)\tau$ for our architecture and $3n + \log_2(n)\ \tau$ for the previous best one. The term $\log_2(n)$ is due to the final redundant to two's-complement conversion.

IV. IMPROVEMENTS TO THE PARALLELIZATION SCHEME

Some improvements are still feasible, however. Two examples, a termination algorithm and a Booth encoding method will be described in the following.

A. Termination Algorithm

It seems favorable to quit the iteration process as early as possible. Then the "termination algorithm" originally proposed by Chen [10] and later applied to CORDIC rotation mode [11] and generalized in [12] and [13] can be used. The generalized algorithm for rotation as well as for vectoring mode is summarized in Table V.

As the prediction algorithm cannot be applied to the CORDIC vectoring mode we only need the rotation part of the termination algorithm. Basically, the second half of the n iterations in rotation mode is substituted by two multiplications in parallel. A fully parallel n bit Wallace tree multiplier exhibits a delay of about $2 \cdot \log_2(n)$ full adder time units. A combination of both algorithms, prediction and termination, results in a latency time of about $(n+1) + \log_3(n) - 1 + 2\log_2(n/2) + \log_2(n)\tau = n + \log_3(n) + 3\log_2(n) - 2\tau$, nearly 50% less than without the termination algorithm.

B. Modified Booth Encoding of σ_i

The goal of this approach is to suppress unnecessary iterations, i.e., subsequent rotations in forward and reverse direction. Booth-like techniques replace subsequent strings of 1's by $10..0\bar{1}$ and halve the number of nonzero bits. Consequently, the number of iterations in a given interval can be halved when Booth encoding of σ_i is applied. Instead of employing Table I for recoding, we use the modified Booth algorithm [14] to convert the binary representation of z_i into $\sigma_i \in \{0, 1, -1\}$. This guarantees that each bit pair contains at least one zero or, explicitly stated, $\sigma_i\sigma_{i+1} = 0$ for $i, i+2, i+4, \cdots$. Note that the recoding process is done in the parallelized z-path so it does not increase latency time. The algorithm is again dependent on i and is defined as follows; (assume $S(m,i) = i, \lambda(t) = 1$ for $|t| = 0, \lambda(t) = 0$ for $|t| = 1$):

—$0 \le i \le (n-3)/4$: use the prediction algorithm, generate σ_i from z_i using Table I in the same manner as in Fig. 1, $\sigma_i \in \{-1, 1\}$, execute iterations according to (1)–(3).

—$(n-3)/4 < i \le (n+1)/2$: use the prediction algorithm, generate σ_i and σ_{i+1} from the bits of z_i by modified Booth recoding $\sigma_i, \sigma_{i+1} \in \{0, 1, -1\}$, after each iteration increment i by 2

$$x_{i+2} = \left(x_i - m\sigma_i 2^{-i}y_i - m\sigma_{i+1}2^{-i-1}y_i\right) \left(1 + \lambda(\sigma_i)m2^{-2i-1}x_i + \lambda(\sigma_{i+1})m2^{-2i-3}x_i\right) \quad (11)$$

$$y_{i+2} = \left(y_i + \sigma_i 2^{-i}x_i + \sigma_{i+1}2^{-i-1}x_i\right) \left(1 + \lambda(\sigma_i)m2^{-2i-1}y_i + \lambda(\sigma_{i+1})m2^{-2i-3}y_i\right) \quad (12)$$

$$z_{i+2} = z_i - \sigma_i\alpha_{m,i} - \sigma_{i+1}\alpha_{m,i+1}.$$

—$(n+1)/2 < i$: use the prediction algorithm, generate σ_i from z_i by modified Booth recoding, $\sigma_i \in \{0, 1, -1\}$, after each iteration increment i by 2

$$x_{i+2} = x_i - m\sigma_i 2^{-i}y_i - m\sigma_{i+1}2^{-i-1}y_i$$
$$y_{i+2} = y_i + \sigma_i 2^{-i}x_i + \sigma_{i+1}2^{-i-1}x_i$$
$$z_{i+2} = z_i - \sigma_i\alpha_{m,i} - \sigma_{i+1}\alpha_{m,i+1}.$$

Remarks:

—$0 \le i \le (n-3)/4$: The normal prediction algorithm takes place, as discussed above and depicted in Fig. 1.

—$(n-3)/4 < i \le (n+1)/2$: The string of σ_i's is recoded in the described manner. This recoding can be done in parallel. Hence, the number of nonzero σ_i's is at most half of its largest possible value without recoding. Three cases can occur: $(|\sigma_i|, |\sigma_{i+1}|) = (0,1), (1,0), (0,0)$. Note that $\sigma_i\sigma_{i+1}$ equals zero.

The first factor in (11) and (12) executes a rotation by either $\alpha_{m,i}$ or $\alpha_{m,i+1}$. As we multiplex the different shifts, our 4-to-2 cell (3-to-2 cell in the z-path) will do. The second factor implements our constant scale factor algorithm to prevent any data-dependent variations of the vector length. It results from the relationship $\left(1 + m2^{-2i-1}\right)\left(1 + m2^{-2i-3}\right) = 1 + m2^{-2i-1} + m2^{-2i-3}$ or, generally,

$$\prod_{j=0}^{n}\left(1 + m2^{-2i-2j-1}\right) = 1 + \sum_{j=0}^{n} m2^{-2i-2j-1}$$

that holds within n bit accuracy. Therefore, the second part of (11)–(12) is parallelizable and we suggest to postpone this

part until the whole iteration process has been completed and use a Wallace tree to implement it in parallel.

—$(n+1)/2 < i$: The approach in this part is analogous to the termination algorithm. The design regularity, however, will be improved, as tree structures are not necessary in physical chip layout. The original iterations are paired so that two subsequent iterations are always merged into a new single iteration. Note that either σ_i or σ_{i+1} or even both are zero, so we need in effect a 4-to-2 adder cell (3-to-2 cell in the z-path) with a 3:1 multiplexer. The inputs to the multiplexer are the shifted iteration/angle values and zeros. As $i > (n+1)/2$ this string recoding does not affect the scale factor.

C. Comparison

For comparison purposes we have to take into account the time required to compensate for the scaling factor. A survey of the approaches known from literature has been given in [15]. Two different strategies can be identified. First, one can increase the number of iterations in order to obtain a scaling factor that is simple to compensate [11], i.e., when it equals, for example, two. This approach utilizes special search programs to find an optimal shift sequence $S(m,i)$ for each n. Therefore, as the number of extra iterations varies with n we have not included the scale factor compensation in our comparison.

However, following the argumentation in [15], we have to consider that in array or pipeline structures the scaling process can be executed in parallel because k_m^{-1} is known in advance. In view of this, the "extra iteration" methods seem less favorable as they tend to increase the latency time. Instead, in order to minimize the latency, the second strategy proves to exhibit less overall latency. Here we separate strictly the iterations and the scaling factor compensation. Then we choose the shortest possible iteration sequence $S(m,i)$ and concentrate solely on the optimized and parallel multiplication of x_n and y_n by k_m^{-1}. It has been shown in [15] that a minimal recoding (minimizing the number of 1's and -1's) of the bits of k_m^{-1} and implementing the multiplication by a Wallace tree is superior in terms of latency time compared with the first strategy.

Fig. 2 illustrates the speedup of the proposed methods in terms of full adder time units compared with the fastest CORDIC realization known to the authors, i.e., implemented using a conventional array with carry-save adders [7]. Obviously, the prediction algorithm leads to a latency time reduction by about 30%. The application of the termination algorithm and the Booth encoding results in an overall speedup by 60% and 50%, respectively. In addition, while not stressed here, the parallelized methods result in chip area savings by up to 25%. The price we have to pay is a loss in regularity. Further research should concentrate on investigating how these approaches that can be used only for rotation mode can be extended to the vectoring mode, too.

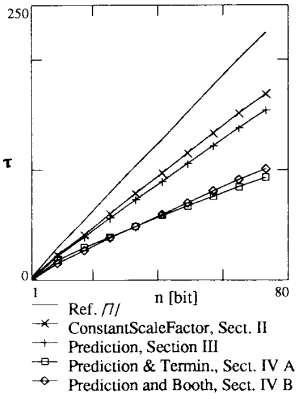


Fig. 2. Latency time comparison (excluding scale factor compensation).

V. Summary

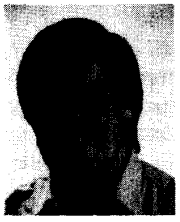
In this contribution, we first described a method to maintain a constant scale factor when employing fast redundant addition schemes. Hereby, a speedup and chip area reduction by up to 25% has been shown. Then we suggested a scheme that exploits the hidden parallelism in CORDIC by σ_i prediction. By including other concepts like the termination algorithm or Booth encoding, a considerable speed and chip area improvement has also been achieved.

References

[1] J. S. Walther, "A unified algorithm for elementary functions," in *Proc. JSSC*, 1971, pp. 379–385.
[2] N. Takagi, "Studies on hardware algorithms for arithmetic operations with a redundant binary representation," Dep. Inf. Sci., Faculty of Eng., Univ. of Kyoto, Aug. 1987.
[3] M. D. Ercegovac and T. Lang, "Redundant and on-line CORDIC: Application to matrix triangularization and SVD," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 725–740, June 1990.
[4] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," *IEEE Trans. Comput.*, vol. 40, no. 9, pp. 989–995, Sept. 1991.
[5] T. Asada, N. Takagi, and S. Yajima, "Algorithms based on CORDIC for calculating hyperbolic functions using redundant binary representation," in *34th IPSJ Nat. Conven.*, 3N-4, pp. 33–34, Mar. 1987 (in Japanese).
[6] T. G. Noll, "Carry-save arithmetic for high-speed digital signal processing," in *Proc. ISCAS '90*, New Orleans, LA, May 1990, pp. 982–986.
[7] R. Künemund *et al.*, "CORDIC processor with carry-save architecture," in *Proc. ESSCIRC '90*, Grenoble, Sept. 1990, pp. 193–196.
[8] D. Timmermann, "CORDIC Algorithmen, Architekturen und monolithische Realisierungen mit Anwendungen in der Bildverarbeitung," Ph.D. dissertation, Univ. of Duisburg, July 1990, (in German).
[9] P. W. Baker, "Suggestion for a fast binary sine/cosine generator," *IEEE Trans. Comput.*, vol. C-25, pp. 1134–1136, Nov. 1976.
[10] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios, and square roots," *IBM J. Res. Develop.*, pp. 380–388, July 1972.
[11] H. M. Ahmed, "Signal processing algorithms and architectures," Ph.D. dissertation Dep. EE, Stanford Univ., Dec. 1981.
[12] D. Timmermann, H. Hahn, and B. Hosticka, "Modified CORDIC algorithm with reduced iterations," *Electron. Lett.*, vol. 25, no. 15, pp. 950–951, July 1989.
[13] H. Hahn, "Untersuchung und Integration von Berechnungsverfahren elementarer Funktionen auf CORDIC-Basis mit Anwendungen in der adaptiven Signalverarbeitung," Ph.D. dissertation, Univ. of Duisburg, Feb. 1991, (in German).
[14] K. Hwang, *Computer Arithmetic, Principles, Architecture, and Design.* New York: Wiley, 1979.
[15] D. Timmermann, H. Hahn, B. J. Hosticka, and B. Rix, "A new addition scheme and fast scaling factor compensation methods for CORDIC algorithms," *INTEGRATION VLSI J.*, vol. 11, no. 1, pp. 85–100, Mar. 1991.

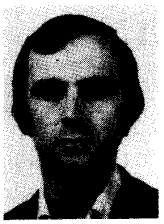
Dirk Timmermann was born in Altena, Germany, in 1957. He received the Dipl.-Ing. degree in electrical engineering from the University of Dortmund, Germany, in 1984, and the Dr.-Ing. degree in electrical engineering from the University of Duisburg, Germany, in 1990. His research interests include computer arithmetic, integrated circuit design, and digital signal processing

In 1984 he joined the Signal Processing and System Design Department at the Fraunhofer Institute of Microelectronics Circuits and Systems, Duisburg, Germany.



Helmut Hahn was born in Dortmund, Germany, in 1958. He received the Dipl.-Ing. degree in electrical engineering from the University of Dortmund, Germany, in 1984, and the Dr.-Ing. degree in electrical engineering from the University of Duisburg, Germany, in 1991. His current research interests include adaptive digital signal processing and system design.

From 1984 to 1989 he was with the University of Duisburg, Germany and since 1984 has been with the Signal Processing and System Design Department at the Fraunhofer Institute of Microelectronics Circuits and Systems, Duisburg.



Bedrich J. Hosticka (M'80–SM'84) was born in Prague, Czechoslovakia, on September 9, 1944. He received the Dipl.-Ing. degree in electrical engineering from the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, in 1972, and the Ph.D. degree from the University of California, Berkeley, in 1977.

In 1972 he joined Siemens-Albis AG, Zurich, Switzerland, where he was engaged in circuit design for the PCM-30 digital transmission system. From 1975 to 1977, he was a Research Assistant at the Electronics Research Laboratory, University of California, Berkeley. During the summer of 1976, he was with IBM T.J. Watson Research Center, Yorktown Heights, NY, working on MOS amplifiers. He joined the Institute of Telecommunications of the Swiss Federal Institute of Technology, Zurich, Switzerland in 1978, where his primary work was on the design of switched capacitor filters. From 1980 to 1985 he was with the Lehrstuhl Bauelemente der Elektrotechnik, University of Dortmund, Dortmund, Germany. He is now Professor at the University of Duisburg, Duisburg, Germany, and Head of Signal Processing and System Design Department at the Fraunhofer Institute of Microelectronic Circuits and Systems, Duisburg, Germany. His research interests include design of integrated circuits, signal processing, and neural networks.

Dr. Hosticka has authored and coauthored over 150 technical papers. From 1983 to 1986 he served as Associate Editor of the IEEE JOURNAL OF SOLID-STATE CIRCUITS.