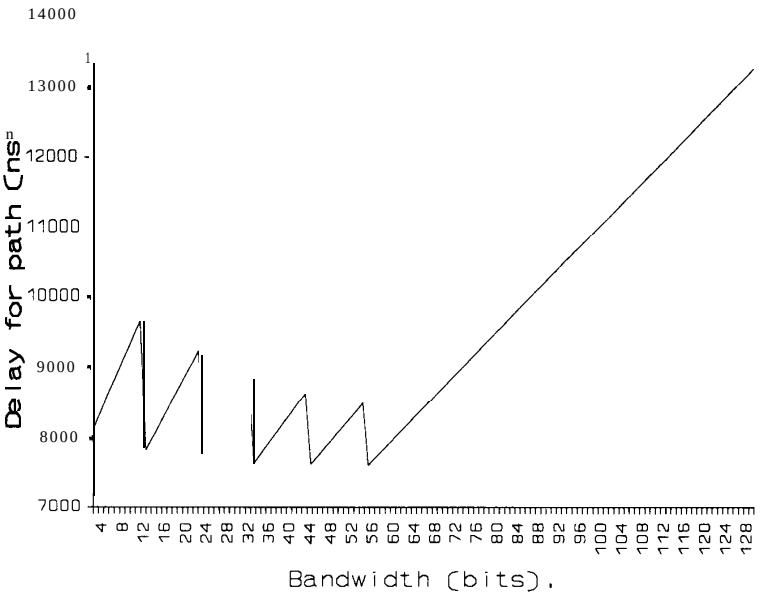

Graph 2: Peal time (throughput) for 10 PE ring.

[8] G. Mazaré and E. Payan, "A programmable highly parallel architecture for digital signal processing," in *Proc. Int. Symp. Circuits and Syst.*, 1989, pp. 1332-1335.
[9] J. Vasell and J. Vasell, "A wavefront array for functional program execution," Tech. Rep. 65, Dep. of Comput. Eng., Chalmers Univ. of Technology, Göteborg, Sweden, Apr. 1990.
[10] A. J. C. Spray and S. R. Jones, "PACE: A regular array for implementing regularly and irregularly structured algorithms," *IEE Proc.*, **vol.** 138, pt. G, no. 5, Oct. 1991.
[11] R. Melhem, "Irregular wavefronts in data-driven, data-dependent computations," in ***Systolic Array Processors,*** W. R. Moore, A. P. H. McCabe, and R. Urqhart, Eds. Bristol, U.K.: Adam Hilger, ISBN 0-85 274-826-4, July 1986, pp. 303-312.
[12] C. R. Ward and E. B. Davie, "The application and development of wavefront array processors for advanced front-end signal processing systems," in ***Systolic Arrays,*** W. R. Moore, A. P. H. McCabe, and R. Urquhart, Eds. Bristol, U.K.: Adam Hilger, ISBN O-85 274-826-4, July 1986, pp. 295-301.
[13] S.-Y. Kung and R. J. Gal-Ezer, "Synchronous versus asynchronous computation in very large scale integrated (VLSI) array processors," *SPIE*, vol. 341, ***Real Time Signal Processing V***, pp. 53-65. 1982.
[14] Praxis Electronic Design Limited, ***The ELLA Language Reference Manual,*** Issue 3v2." Praxis Electronic Design Limited, Henry Street, Bath, Avon.
[15] Texas Instruments, ***TAAC Gate Arrays Users Manual,*** E-Beam Logic Series, Texas Instruments, U.K., 1st Quarter 1988.
[16] S.-Y. Kung, S.-C. Lo, and P. S. Lewis, "Timing analysis and design optimisation of VLSI data flow arrays," in *Proc. IEEE Int. Conf.* ***Parallel Processing,*** 1986, pp. 600-607.

Computing Functions $\cos^{-1}$ and $\sin^{-1}$ Using Cordic

Christophe Mazenc, Xavier Merrheim, and Jean-Michel Mullet

Abstract-After **briefly recalling the main properties of the Cordic algorithm, we show that a slight modification of this algorithm enables the computation of the functions** $\cos^{-1}, \sin^{-1}, \sqrt{1-t^2}, \cosh^{-1}, \sinh^{-1},$ **and** $\sqrt{1+t^2}$.

Index **Terms-Computer arithmetic, Cordic, elementary functions.**

I. Introduction

The Cordic algorithm was introduced in 19.59 by Jack Volder [15]. This algorithm makes it possible to perform rotations (and therefore to compute sine, cosine, and tan-' functions) and to multiply or divide numbers, using only shift-and-add elementary steps. In 1971, John Walther [16] generalized Volder's algorithm in order to compute hyperbolic functions, logarithms, exponentials, and square roots. Cordic (or very similar algorithms) has been implemented in pocket calculators like Hewlett Packard's HP 35 [4], and in arithmetic coprocessors like the Intel 8087. Several authors have proposed to use Cordic processors for signal processing applications (DFT or filtering [9]), for image processing [3], or for solving linear systems [1], [15].

Manuscript received April 19, 1991; revised March 2, 1992.

C. Mazenc and X. Merrheim are with Laboratoire LIP-IMAG, Ecole Normale Supérieure de Lyon, France.
J.-M. Muller is with CNRS. Laboratoire LIP-IMAG, Ecole Normale Supérieure de Lyon, France.

IEEE Log Number 9200214.

In Walther's version, Cordic consists of the following iteration:

$$x_{n+1} = x_n - m d_n y_n 2^{-\sigma(n)}$$
$$y_{n+1} = y_n + d_n x_n 2^{-\sigma(n)}$$
$$z_{n+1} = z_n - d_n e_{\sigma(n)}$$

where the results (i.e., the limit values of x_n, y_n, and z_n) and the values of d_n, m, and a(n) are presented in Fig. 1 and Table I. The constants $e_{\sigma(i)}$ are precomputed and stored.

The function σ is an artefact: in a practical implementation, the iterations are performed assuming u(n) = n, and in the hyperbolic mode (m = -1), the iterations 4, 13, 40, . . . , k, $3k + 1$, are repeated. This repetition is necessary since the sequence $e_n = \tanh^{-1} 2^{-n}$ does not satisfy the relation of Theorem 1 (see below), while the sequence $e_{\sigma(n)}$ obtained from e_n by repeating the terms of indexes 4, 13, 40. . . satisfies this relation. K and K' are equal to

$$K = \prod_{n=0}^{\infty} \frac{1}{\cos(\tan^{-1} 2^{-n})} = \prod_{n=0}^{\infty} \sqrt{1 + 2^{-2n}}$$
$$= 1.646760258121\ldots$$
$$K' = \prod_{n=1}^{\infty} \frac{1}{\cosh(e_{\sigma(n)})} = \prod_{n=1}^{\infty} \sqrt{1 - 2^{-2\sigma(n)}}$$
$$= 0.8281593609602\ldots$$

Cordic is a very useful algorithm, since it allows computation of some of the most common mathematical functions [16]. For instance, e^x is obtained by adding $\cosh x$ and $\sinh x$, $\ln x$ is obtained using the relation

$$\ln(x) = 2\tanh^{-1} \frac{|1 - x|}{|1 + x|}$$

while $\sqrt{x}$ is obtained by

$$\sqrt{x} = \sqrt{\left(x + \frac{1}{4}\right)^2 - \left(x - \frac{1}{4}\right)^2}.$$

On-line implementations of Cordic have been proposed by several authors (see for instance [10], [11]). In order to explain our algorithm, let us clearly analyze Volder's version of Cordic in the rotation mode. First of all, we start from the following theorem (see [13] for proof):

Theorem 1: If (e_n) is a decreasing sequence of positive real numbers such that $\sum_{n=0}^{\infty} e_n < +\infty$, and if for any integer n, $e_n \le \sum_{i=n+1}^{\infty} e_i$, then for any $\theta \in [-\sum_{n=0}^{\infty} e_n, +\sum_{n=0}^{\infty} e_n]$, the sequences (θ_n) and (d_n) defined as

$$\theta_0 = 0$$
$$d_n = \begin{cases} 1 & \text{if } \theta_n \le \theta \\ -1 & \text{if } \theta_n > \theta \end{cases}$$
$$\theta_{n+1} = \theta_n + d_n e_n$$

satisfy $\lim_{n\to\infty} \theta_n = \sum_{i=0}^{\infty} d_i e_i = \theta$.

The sequence (e_n) is called a ***discrete basis,*** and the previous algorithm which gives (θ_n) and (d_n) is called the ***bidirectional algorithm*** [13].

Now, let us assume that we want to perform a rotation of angle θ, i.e., to compute, from an initial two-dimensional vector $(x_0, y_0)^t$, a vector $(x_\infty, y_\infty)^t$ defined as

$$\begin{pmatrix} x_\infty \\ y_\infty \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

The basic idea of Cordic is to perform this rotation as a sequence of elementary rotations. Using the bidirectional algorithm and the discrete basis $e_n = \tan^{-1} 2^{-n}$, θ is rewritten as a sum $\theta = \sum_{n=0}^{\infty} d_n e_n$ then the sequence $(x_n, y_n)^t$ defined as

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} \cos(d_n e_n) & -\sin(d_n e_n) \\ \sin(d_n e_n) & \cos(d_n e_n) \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$
$$= \cos(e_n) \begin{pmatrix} 1 & -d_n \tan e_n \\ d_n \tan e_n & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$
$$= \cos(e_n) \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \quad \text{(A)}$$

($(x_{n+1}, y_{n+1})^t$ is obtained from $(x_n, y_n)^t$ by performing a rotation of angle $d_n e_n$), satisfies

$$\begin{cases} x_n \to x_\infty \\ n \to \infty \\ y_n \to y_\infty \\ n \to \infty. \end{cases}$$

In relation (A), there is only one "true" multiplication (i.e., a multiplication which cannot be reduced to a very small number of additions and shifts), since in radix 2 a multiplication by 2^{-n} may be reduced to a shift. This "true" multiplication cannot be avoided: there is no nontrivial choice of e_n which enables one to perform a rotation of angle e_n with only a finite number of shifts (see [10] for proof). Since rotations cannot be performed without "true" multiplications, instead of (A), we perform:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

which is the basic Cordic step: it is not a rotation of angle e_n, but a similarity of angle e_n and factor $1/\cos e_n$. Hence, the final result of the iterations is no longer a rotation of angle θ, but a similarity of angle θ and factor $K = \prod_{i=0}^{\infty} 1/\cos(e_i) = \prod_{i=0}^{\infty}(1 + 2^{-2i})^{1/2}$. We then deduce:

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix}_{n\to\infty} \to K \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

The resulting algorithm is called the ***rotation mode*** of Cordic. If we define a sequence z_n as $\theta - \theta_n$, the algorithm becomes

$$z_0 = \theta$$
$$\begin{cases} x_{n+1} = x_n - d_n y_n 2^{-n} \\ y_{n+1} = y_n + d_n x_n 2^{-n} \\ z_{n+1} = z_n - d_n \tan^{-1} 2^{-n} \end{cases}$$
$$d_n = \begin{cases} 1 & \text{if } z_n \ge 0 \\ -1 & \text{if } z_n < 0 \end{cases}.$$

For instance, the sine and cosine functions are computed by taking $x_0 = 1/K$ and $y_0 = 0$. It may be shown that the error obtained if we approximate $\sin\theta$ by y_n and $\cos\theta$ by x_n is roughly equal to 2^{-n}. This algorithm yields a correct result if and only if $\theta \in [-\sum_{n=0}^{\infty} e_n, +\sum_{n=0}^{\infty} e_n] \approx [-1.743, +1.743]$.

II. Computation of $\cos^{-1}$ and $\sin^{-1}$ Functions

Now, assume that we want to compute $\theta = \cos^{-1}(t)$, $t \in [0, 1]$. When we perform a rotation of angle θ of the point $(1, 0)^t$ using Cordic, we perform:

$$\theta_0 = 0,\ x_0 = 1,\ y_0 = 0$$
$$d_n = 1 \text{ if } \theta_n \le \theta \text{ else } -1$$
$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$
$$\theta_{n+1} = \theta_n + d_n \tan^{-1} 2^{-n}. \quad \text{(B)}$$

TABLE I
VALUES OF $\sigma(n)$ AND e_n IN FIG. 1

m = 1 (circular mode)	$\sigma(n) = n$	$e_n = \tan^{-1} 2^{-n}$
m = -1 (hyperbolic mode)	$\sigma(n) = 1,2,3,4,4,5,6,\cdots,12,13,13,14,15,\cdots,39,40,40$	$e_n = \tanh^{-1} 2^{-n}$
m = 0 (linear mode)	a(n) = n	$e_n = 2^{-n}$

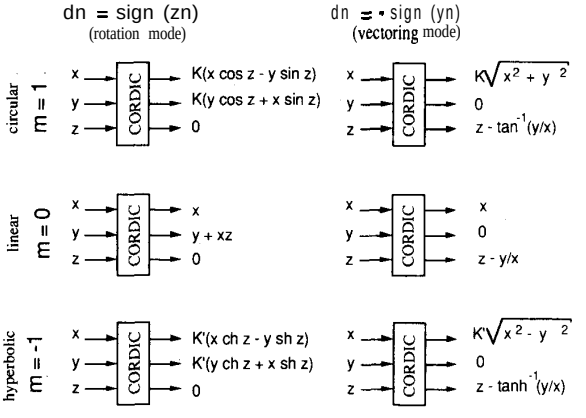

Fig. 1. Different functions computable using Cordic.

And the sequence θ_n goes to θ as n goes to infinity (this is a consequence of Theorem 1). Since the value of θ is not known (it is the value we want to compute !), we cannot perform the test (B) indicated above. However, (B) is equivalent to "$d_n = sign(y_n)$ if cos θ_n cos θ else -sign (y_n)", where $sign(y_n) = 1$ if $y_n \geq 0$, else -1. Thus, since the variables x_n and y_n obtained in step n satisfy

$$\begin{cases} x_n = \text{Ii}, \ \cos\theta_n \\ y_n = \text{Ii}, \ \sin\theta_n \end{cases}$$

with $K_n = \prod_{i=0}^{n-1} 1/\cos(e_i) = \prod_{i=0}^{n-1}(1 + 2^{-2i})^{1/2}$, (B) is equivalent to "$d_n = \text{sign}(y_n)$ if $x_n \geq K_n t$ else $-\text{sign}(y_n)$." Assume momentarily that the terms $t_n = K_n t$ are known, the algorithm

$$\left|\begin{array}{l} cos^{-\text{I}}.1 \\ \theta_0 = 0, \ x_0 = 1, \ y_0 = 0 \\ d_n = \text{sign}\ (y_n) \text{ if } x_n \geq t_n \text{ else } - \text{ sign (y,)} \\ \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \\ \theta_{n+1} = \theta_n + d_n \tan^{-1} 2^{-n} \end{array}\right.$$

gives $\theta_n \underset{n \to \infty}{\to} \cos^{-1} t$. In a very similar way, the algorithm

$$\left|\begin{array}{l} sin^{-\text{I}}.1 \\ \theta_0 = 0, x_0 = 1, y_0 = 0 \\ d_n = \text{sign}\ (x_n) \text{ if } y_n \leq t_n \text{ else } - \text{ sign } (x_n) \\ \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \\ \theta_{n+1} = \theta_n + d_n \tan^{-1} 2^{-n} \end{array}\right.$$

gives $\theta_n \underset{n \to \infty}{\to}$ sin-' t.

Obviously, the main drawback of these algorithms is the evaluation of t_n. The iterative relation $t_{n+1} = t_n / \cos(\tan^{-1} 2^{-n}) = t_n \sqrt{1 + 2^{-2n}}$ cannot be used since it involves a "true" multiplication. In order to avoid this drawback, we shall perform "double" Cordic iterations: we shall use the bidirectional algorithm with the discrete basis $e_n = 2 \tan^{-1} 2^{-n}$ instead of the discrete basis $e_n = \tan^{-1} 2^{-n}$. Therefore, at step n of the algorithm, we shall perform two similarities of angle $\tan^{-1} 2^{-n}$. Double Cordic iterations have already been used by Takagi, Asada, and Yajima [14], and by Delosme [6], [7] in quite a different context: the purpose of Takagi, Asada, and Yajima was to keep a constant scaling factor K when quickly performing Cordic iterations using a redundant number system. The purpose of Delosme was to obtain simpler scaling factors. The main advantage of doubling iterations is that in step n, the factor of the similarity becomes $1/\cos^2(\tan^{-1} 2^{-n}) = 1 + 2^{-2n}$: now, a multiplication by this term reduces to an addition and a shift. Another advantage is that the convergence domain of the algorithm becomes larger: it gives a correct result for $\theta \in [-2 \sum_{n=0}^{\infty} \tan^{-1} 2^{-n}, +2 \sum_{n=0}^{\infty} \tan^{-1} 2^{-n}] \approx [-3.48657. \ +3.48657]$, therefore, we can compute $\cos^{-1} t$ and $\sin^{-1} t$ for any $t \in [-1. \ 1]$. The algorithm for computing $\cos^{-1} t$ becomes

$$\left|\begin{array}{l} cos^{-\text{I}}.2 \\ \theta_0 = 0.1\text{'}0 = 1. \ y_0 = 0. \ t_0 = t \\ d_n = \text{sign}\ (y_n) \text{ if } x_n \geq t_n \text{ else } - \text{ sign } (y_n \\ \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix}^2 \begin{pmatrix} x_n \\ y_n \end{pmatrix} \\ \theta_{n+1} = \theta_n + 2 d_n \tan^{-1} 2^{-n} \\ t_{n+1} = t_n + t_n 2^{-2n} \end{array}\right.$$

it gives: $\theta_n \to \cos^{-1} t$. In a very similar way, the algorithm

$$\left|\begin{array}{l} sin^{-1} .2 \\ \theta_0 = 0, x_0 = 1, y_0 = 0, t_0 = t \\ d_n = \text{sign}\ (x_n) \text{ if } y_n \leq t_n \text{ else } - \text{ sign } (x_n) \\ \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix}^2 \begin{pmatrix} x_n \\ y_n \end{pmatrix} \\ \theta_{n+1} = \theta_n + 2 d_n \tan^{-1} 2^{-n} \\ t_{n+1} = t_n + t_n 2^{-2n} \end{array}\right.$$

gives $\theta_n \to \sin^{-1} t$. It is worth noting that the replacement in these algorithms of the terms $tan^{-1} 2^{-i}$ by the terms $tanh^{-1} 2^{-i}$, and the repetition of the iterations no 4, 13, 40, 121, would give algorithms for computing the functions $sinh^{-1}$ and $cosh^{-1}$, provided we replace the elementary rotations by

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix}^2 \begin{pmatrix} x_n \\ y_n \end{pmatrix}.$$

At step n of the algorithm $\cos^{-1}.2$, x_n is equal to $K_n^2 \cos\theta_n$ and y_n is equal to $K_n^2 \sin\theta_n$, where K_n has the same value as previously. Therefore, since θ_n goes to $\theta = \cos^{-1} t$ as n goes to infinity, we

TABLE II
COMPUTATION OF $\cos^{-1} 0.5$

n	x_n	y_n	t_n	d_n	θ_n	$\theta_n - \cos^{-1} t$
0	1.000000000000	0.000000000000	030000000000	1	0.000000000000	-1.047197551196
1	0.000000000000	2.000000000000	1.000000000000	- 1	1.570796326795	0.5235987755982
2	2.000000000000	1.500000000000	1.250000000000	1	0.643501108793	-0.4036964424033
3	1.125000000000	2.406250000000	1.328125000000	- 1	1.133458435047,	0.0862608838504
4	1.708984375000	2.087402343750	1.348876953125	1	0.884748445953	-0.162449105243

deduce

$$y_n \underset{n \to \infty}{\longrightarrow} K^2\sqrt{1-t^2}.$$

Thus, $\cos^{-1}.2$ may be used in order to compute $f(t) = \sqrt{1-t^2}$. The undesirable scaling factor K^2 may be avoided by performing another sequence of rotations in parallel with that of $\cos^{-1}.2$, as follows:

$$\begin{cases} \theta_n = 0,\ x_0 = 1,\ x_0' = 1/K^2,\ y_0 = 0,\ y_0' = 0,\ t_0 = t \\ d_n = 1 \text{ if } x_n \geq t_n \text{ else } -1 \\ \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix}^2 \begin{pmatrix} x_n \\ y_n \end{pmatrix} \\ \begin{pmatrix} x_{n+1}' \\ y_{n+1}' \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix}^2 \begin{pmatrix} x_n' \\ y_n' \end{pmatrix} \\ \theta_{n+1} = \theta_n + 2d_n \tan^{-1} 2^{-n} \\ t_{n+1} = t_n + t_n 2^{-2n} \end{cases}$$

We obtain: $y_n' \underset{n \to \infty}{\longrightarrow} \sqrt{1-t^2}$. If we make the same changes to this algorithm as previously (when we gave a method for computing the hyperbolic sine and cosine functions), then we obtain an algorithm for computing the function $\sqrt{1+t^2}$. It is worth noting that this function may be computed in a simpler way: in [8], Delosme shows that if

$$\begin{aligned} & x_0 = 1, y_0 = y \\ & d_n = \mathbf{sign}\,(x, \cdot\ y_n) \\ & x_{n+1} = x_n + d_n 2^{-n} \\ & y_{n+1} = y_n - 2d_n 2^{-n} x_n \quad (2^{-n})^2 \end{aligned}$$

then x_n converges to $\sqrt{y+x^2}$. As a matter of fact, this iteration may be viewed as a slight modification (change of initial values) of a classical square root iteration (see [12] for instance).

Fig. 2 shows the nth step of $cos^{-1}.2$

Example: We compute $\cos^{-1} t$, using the algorithm $cos^{-1}.2$, for $t = 1/2$. The exact value is $\pi/3 \approx 1.047197551196$. Table II displays the first 5 steps.

III. Error Analysis

We have previously shown that, if we use $cos^{-1}.2$, then $\theta_n \underset{n \to \infty}{\longrightarrow} \cos^{-1} t$ and if we use $sin^{-1}.2$, then $\theta_n \underset{n \to \infty}{\longrightarrow} \sin^{-1} t$. Our problem is to evaluate the error obtained if we stop the algorithm at step n, i.e., if we approximate the limit value θ of the sequence (θ_i) by θ_n. Since, from Theorem 1, we have

$$\theta = \sum_{i=0}^{\infty} d_i \left[2 \tan^{-1} 2^{-i}\right] = \theta_n + \prod_{i=n}^{\infty} d_i \left[2 \tan^{-1} 2^{-i}\right]$$

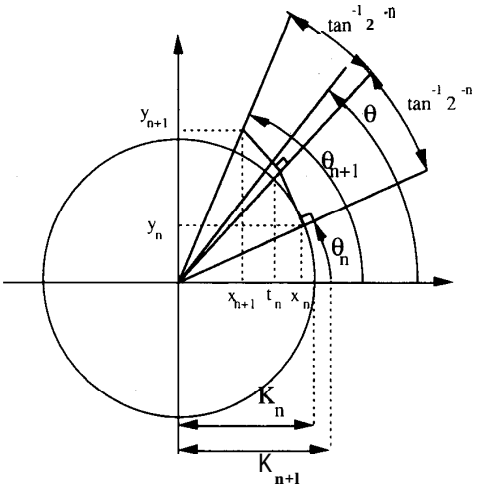

Fig. 2. nth step of the computation of $cos^{-1} t$.

therefore

$$|\theta_n - \theta| \leq 2 \sum_{i=n}^{\infty} \tan^{-1} 2^{-i} \leq 2 \sum_{i=n}^{\infty} 2^{-i} = 2^{-n+2}.$$

Hence, one needs to perform $p + 2$ iterations in order to ensure p accuracy bits.

IV. Conclusion

We have given here an extension of the Cordic algorithm which makes it possible to compute the functions cos^{-1}, sin-', $\sqrt{1-t^2}$, $sinh^{-1}$, $cosh^{-1}$, and $\sqrt{1+t^2}$. Our algorithms are suitable for VLSI implementation, and require only a slight modification of the original Cordic algorithm.

References

[1] H. M. Ahmed, J.-M. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing," *IEEE Comput. Mag.*, Jan. 1982.

[2] P. W. Baker, "Suggestion for a fast binary sine/cosine generator," *IEEE Trans. Comput.*, Nov. 1976.

[3] J. R. Cavallaro and F. T. Luk, "Cordic arithmetic for a SVD processor," in *Proc. 8th Symp. Comput. Arithmet.*, Como, Italy, May 1987, also in *J. Parallel Distributed Comput., vol. 5*, no. 3, pp. 271–290.

[4] D. Cochran, "Algorithms and accuracy in the HP35," *Hewlett Packard J.*, June 1972.

[5] M. Cosnard *et* al., "The FELIN arithmetic coprocessor chip," in *Proc. 8th Symp. Comput. Arithmet. ARITH8*, Como, Italy, May 1987, pp. 107–112.

[6] J. M. Delosme, "A processor for two-dimensional symmetric eigenvalue and singular value arrays," in *Proc. 21st Asilomar Conf. Circuits, Syst. and Comput.*, Pacific Grove, CA, Nov. 1987, pp. 217-221.

[7] ——, "Cordic algorithms: Theory and extensions," in *Advanced Algorithms and Architectures for Signal Processing IV, Proc. SPIE 1152*, Aug. 1989, pp. 131-145.

[8] ——, "The matrix exponential approach to elementary operations," in *Advanced Algorithms and Architectures* for Signal *Processing, Proc. SPIE 696*, Aug. 1986, pp. 188-195.
191 A. M. Despain, "Fourier transform computers using Cordic iterations," *IEEE Trans. Comput.*, May 1984.
[10] J. Duprat and J. M. Muller, "The Cordic algorithm: New results for fast VLSI implementation," Res. Rep. 90-04, Lab. LIP, Ecole Normale Suprieure de Lyon, France. *IEEE Trans. Comput.*, to be published.
[11] M. D. Ercegovac and T. Lang, "Implementation of fast angle calculation and rotation using online Cordic," in *Proc. ISCAS'88*, pp. 2703–2706.
[12] S. Majerski, "Square root algorithms for high-speed digital circuits," *IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 724–733, Aug. 1985.
[13] J. M. Muller, "Discrete basis and computation of elementary functions." *IEEE Trans. Comput.*, vol. C-34, no. 9, pp. 857-862, Sept. 1985.
[14] N. Takagi, T. Asada, and S. Yajima, "A hardware algorithm for computing sine and cosine using redundant binary representation," *Trans. ZECE Japan*, vol. J69-D, no. 6, June 86 (in Japanese). English translation available in *Syst. and Comput. in Japan*, vol. 18, no. 8, Aug. 1987.
[15] J. Volder, "The Cordic computing technique," *IRE Trans. Comput.*, Sept. 1959.
[16] J. Walther, "A unified algorithm for elementary functions." in *Joint Comput. Conf. Proc.*, vol. 38, 1971.

Interrupt Handling for Out-of-Order Execution Processors

H. C. Torng and Martin Day

Abstract- **Processors with multiple functional units, including the superscalars, achieve significant performance enhancement through low-level execution concurrency. In such processors, multiple instructions are often issued and definitely executed concurrently and out-of-order. Consequently, interrupt and exception handling becomes a vexing problem.**

We identify factors that must be considered in evaluating the effectiveness of interrupt and exception handling schemes: latency, cost, and performance degradation. We then briefly enumerate proposals and implementations for interrupt and exception handling on out-of-order execution processors.

Next, we present an efficient hardware mechanism, the Instruction Window (IW), and a new approach, which allows for precise, responsive, and flexible interrupt and exception handling.

The implementation of the IW is then discussed. The design of an S-cell IW has been carried out; it can work with a very short machine cycle time.

Finally, we present a comparison of all interrupt and exception handling schemes for out-of-order execution processors.

Index **Terms-Interrupt handling, interrupt latency, low-level Concurrency, modified precise interrupt, out-of-order execution.**

I. Introduction

Processors with multiple functional units issue and execute multiple instructions concurrently and possibly out-of-order; they enhance performance by extracting low-level concurrency from the instruction stream [1]–[3]. The CDC 6600, IBM 360/91, and the CRAY machines are forerunners of this class of processors; however, these processors issue at most one instruction per cycle. Due to advances in device technologies, recently announced RISC processors often issue and certainly execute multiple instructions concurrently. However, these processors have not been able to support interrupt and exception' handling efficiently and with an acceptable latency.

In this paper, we address the interrupt handling problem, which has hampered the development of processors which execute and may even issue multiple instructions. We propose an efficient hardware mechanism, which supports an interrupt handling scheme with a flexible latency, set specifically for each type of interrupts requested.

The remaining sections are organized as follows: Section II presents a discussion of interrupts and exceptions. Factors for evaluating the effectiveness of interrupt handling schemes are presented. Existing proposals and implementations for interrupt handling on out-of-order execution processors are briefly reported in Section III.

Section IV presents the Instruction Window (IW), a simple and yet versatile hardware mechanism which supports efficient and flexible interrupt handling. Basic window operations are introduced in Section V. Section VI proposes an innovative interrupt handling scheme, which makes use of the IW. In Section VII, we discuss the implementation of the IW. Section VIII gives an evaluation of all interrupt handling schemes.

II. Interrupts and Exceptions

An important and indispensable feature of any processor is its ability to handle properly interrupts and exceptions. An I/O device, a sensor, or a timer may "interrupt" a processor to perform a specific task. An executing instruction may cause a page fault or an overflow/underflow; an "exception" thus results. Finally, one may place an instruction in an instruction stream to call for a "trap," which initiates a pre-planned action. Presentations on interrupts, exceptions, and traps can be found from many sources, among them [4]–[8]. In this paper, we use the term interrupt to denote an interrupt, an exception or a trap. Our study does not treat the subject of interrupt detection; rather, we investigate how a processor responds to an interrupt request, once it has been received.

When an interrupt request is received, the processor must save its processor state, then load and execute an appropriate interrupt handler. Upon completion of the interrupt handling routine, the saved processor state is restored, and the interrupted process can then be restarted.

A processor state should contain enough and preferably only enough information so that the interrupted process can be restarted at the precise point where it was interrupted. To be able to resume an interrupted process, the processor state should consist of the contents of the general purpose registers, the program counter, the condition register, all index registers, and the relevant portion of the main memory.

The classical approach to identifying precisely the point where a process is interrupted is to save, among other vital items, the address of a specific instruction, say instruction α, when the processor state is saved. All instructions that precede instruction α have been executed. And instruction α and those that follow it have not. Instruction α thus provides a *precise* interrupt point.

For processors, which execute instructions concurrently and possibly out-of-order, the identification of a precise interrupt point when an interrupt request is made may become very costly.

'From now on, we will simply use interrupt to stand for interrupt and exception.

Manuscript received August 22, 1991; revised January 9, 1992. This work was supported in part by the joint Services Electronics Program, Contract Number F49620-90-C-0039.

H. C. Torng is with the School of Electrical Engineering, Cornell University, Ithaca, NY 14853.
M. Day is with Bell Information Systems, Toronto, Ont., Canada.

IEEE Log Number 9200216.