The CORDIC Algorithm: New Results for Fast VLSI Implementation

Jean Duprat and Jean-Michel Muller, *Member, IEEE*

***Abstract*— After a brief survey on the CORDIC algorithm, we give some new results which enable fast and easy signed-digit implementation of CORDIC, without modifying the basic iteration step. A slight modification would make it possible to use carry save representation of numbers, instead of signed-digit one. Our method, called *branching CORDIC method,* consists of performing in parallel two classic CORDIC rotations. It gives a constant normalization factor. Then, we propose an on-line implementation of our algorithm with an on-line delay equal to 5 for the sine and cosine functions.**

***Index Terms*—CORDIC, elementary functions on line, signed-digit representations.**

I. Introduction

THE CORDIC algorithm was introduced in 1959 by Volder [16]. In Volder's version, CORDIC makes it possible to perform rotations (and therefore to compute sine, cosine, and arctangent functions) and to multiply or divide numbers, using only shift-and-add elementary steps. In 1971, Walther [17] generalized this algorithm in order to compute logarithms, exponentials, and square roots. CORDIC is not the fastest way to perform multiplications or to compute logarithms and exponentials, but, since the same algorithm enables the computation of most mathematical functions using basic operations of the form $a \pm b \cdot 2^{-i}$, it is attractive for hardware implementations. CORDIC has been implemented in pocket calculators like Hewlett Packard's HP 35 [3], and in arithmetic coprocessors like the Intel 8087. Some authors proposed to use CORDIC processors for signal processing applications (DFT [6], filtering [5], SVD [9]), for image processing [2], or for solving linear systems [1], [11]. The first part of this paper is a brief survey on CORDIC. The reader familiar with this algorithm may skip this part. In the second part, we propose a new algorithm, called *branching CORDIC*, for performing rotations very quickly, using a redundant number system. In the last part, we propose an on-line implementation of our algorithm for computing sine and cosine functions. On-line arithmetic [10] is a digit-serial mode of computation where digits circulate most significant digit first. On-line implementations of CORDIC have been already proposed by Ercegovac and Lang [8], and by Lin and Sips [12].

Manuscript received March 15, 1991; revised March 27, 1992.

The authors are with CNRS, Laboratoire LIP-IMAG, Ecole Normale Superieure de Lyon, 69364 Lyon Cedex 07, France.

IEEE Log Number 9205221.

II. A Brief Survey

A. The Classical CORDIC Iteration

Volder's algorithm is based upon the following iteration:

$$x_{n+1} = x_n - d_n y_n 2^{-n}$$
$$y_{n+1} = y_n + d_n x_n 2^{-n}$$
$$z_{n=1} = z_n - d_n \arctan 2^{-n}.$$

The terms $\arctan 2^{-n}$ are precomputed and stored, and the d_i's are equal to -1 or $+1$. In the *rotation mode* of CORDIC, d_n is chosen equal to the sign of z_n ($+1$ if $z_n \geq 0$, else -1).

If $|z_0|$ is less than or equal to $\sum_{k=0}^{\infty} \arctan 2^{-k} = 1.743 \cdots$, then

$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \underset{n\to\infty}{\rightarrow} K \begin{pmatrix} x_0 \cos z_0 - y_0 \sin z_0 \\ x_0 \sin z_0 + y_0 \cos z_0 \\ 0 \end{pmatrix}$$

where the *scale factor* K is equal to $\prod_{n=0}^{\infty} 1/\cos(d_n \arctan 2^{-n})$. Since for any n, d_n equals ± 1, this factor is constant and equal to $\prod_{n=0}^{\infty} \sqrt{1 + 2^{-2n}} = 1.646760 \cdots$.

For instance, to compute the sine and cosine of a number θ, $|\theta| \leq \sum_{k=0}^{\infty} \arctan 2^{-k}$, one may take $x_0 = 1/K = 0.607252 \cdots$, $y_0 = 0$, and $z_0 = \theta$.

CORDIC may be understood as a rewriting of numbers in special bases, called *discrete bases* [4], [13]. After the iterations, z_0 is equal to $\sum_{k=0}^{\infty} d_k \arctan 2^{-k}$. The sequence $(\arctan 2^{-k})$ is the "base," and the terms d_k are the "digits" of the representation of z_0. Such a representation is possible because for any n, $\arctan 2^{-n} \leq \sum_{k=n+1}^{\infty} \arctan 2^{-k}$.

A positive sequence (e_n) satisfying

$$\forall\, n, \quad e_n \leq \sum_{k=n+1}^{\infty} e_k \tag{1}$$

is called a *discrete basis of order 1* [13], and enables such representations of numbers.

The basic idea of the rotation mode of CORDIC is to perform a rotation of angle $\theta = z_0$ as a sequence of rotations of angle $\pm e_n$, with $e_n = \arctan 2^{-n}$. We start from (x_0, y_0), and the point (x_{n+1}, y_{n+1}) is obtained from the point (x_n, y_n) by a rotation of angle $d_n e_n$ ($d_n = \pm 1$). This gives

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

In relation (A), there is only one "true" multiplication, since

in radix 2 a multiplication by 2^{-n} reduces to a shift. In order to avoid this multiplication, 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 $1/\cos e_n$. In the following, we call *c-similarity of angle* α a similarity of angle α and factor $1/\cos\alpha$. In the *vectoring mode* of CORDIC, d_n is chosen equal to the sign of $(-y_n)$ $(+1$ if $y_n \leq 0$, else $-1)$. This gives

$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \underset{n\to\infty}{\to} \begin{pmatrix} K\sqrt{x_0^2 + y_0^2} \\ 0 \\ z_0 + \arctan \frac{y_0}{x_0} \end{pmatrix}$$

where the constant K is the same as in rotation mode. Since trigonometric and hyperbolic functions are closely related, one may expect that a slight modification of Volder's algorithm could be used for calculating hyperbolic functions. In 1971, John Walther [17] found the correct modification, and obtained the *generalized CORDIC iteration*:

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

where the results and the values of d_n, m, and $\sigma(n)$ are presented in Tables I and II.

In the *hyperbolic mode* $(m = -1)$, the iterations $4, 13, 40, \cdots, k, 3k+1, \cdots$ are repeated (this is why we need to use the function σ). This is necessary since the sequence $\tanh^{-1} 2^{-n}$ does not satisfy relation (1). The sequence $\tanh^{-1} 2^{-\sigma(n)}$ satisfies this relation. K' equals $\prod_{n=1}^{\infty} \sqrt{1 - 2^{-2\sigma(n)}} = 0.82815 \cdots$.

In Walther's version CORDIC makes it possible to compute a lot of mathematical functions [17]. For instance, e^x is obtained by adding ch x and sh x, and ln x is obtained using the relation

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

B. CORDIC Iteration with Redundant Number Systems

In order to accelerate the CORDIC iterations, one can use *redundant number systems,* which enable additions without carry propagation. Here, we focus our attention on binary signed digit representations, however, the results would be similar in carry-save representation. The sequence of digits $u_0, u_1, \cdots, u_m, u_i = -1, 0, 1$, represents the number $\sum_{i=0}^{m} u_i 2^{-i}$. With redundant notations, the main problem is the evaluation of d_n. Assume that we are in *rotation* mode, and that numbers are represented with m digits. In classical CORDIC, d_n is equal to the sign of z_n. In signed-digit representation, this sign is the sign of the most significant nonzero digit: the knowledge of it requires the examination of some number of digits which may be close to m. Thus, the choice "d_n = sign (z_n)" is not satisfactory: the advantage of the redundant representation (a constant time elementary step) would be lost. An alternative is to accept the value $d_n = 0$. One examines only the most p significant digits of z_n. The number z_n^* constituted by these p digits is close to z_n. The basic idea is the following:

- If $z_n^* \neq 0$, then z_n^* and z_n have the same sign, thus the choice $d_n = \text{sign}(z_n^*)$ is convenient.
- If $z_n^* = 0$, then $|z_n|$ is very small. Therefore one can take $d_n = 0$.

Of course, this is not a proof, and the correct value of p which ensures convergence of z_n to zero must be calculated. The main drawback of such a method is that the scale factors K and K' are no longer constants. Since K is equal to $\prod_{n=0}^{\infty} 1/\cos(d_n \arctan 2^{-n})$, it is a constant if the d_i's are all equal to –1 or +1, but it is no longer a constant if the d_i's are allowed to be equal to zero. Some authors suggested different solutions to that problem. Takagi, Asada, and Yajima [14], [15] propose two methods, called *double rotation method* and *correcting rotation method.* The basic principle of the double rotation method is that at step i one performs, instead of a c-similarity of angle $d_i \arctan 2^{-i}$:

- ☐ If $d_i = \pm 1$, two c-similarities of angle $d_i \arctan 2^{-i-1}$
- ☐ If $d_i = 0$, a c-similarity of angle $+\arctan 2^{-i-1}$, then a c-similarity of angle $-\arctan 2^{-i-1}$.

With such a method, the scale factor is constant. However, it leads to more complicated iterations (or twice as many iterations if they remain unchanged).

The basic idea of the correcting rotation method is the following: as above, one examines the number z_n^* constituted by the p most significant digits of z_n. Then, one takes, in rotation mode, $d_n = \text{sign}(z_n^*)$ if $z_n^* \neq 0$, $+1$ otherwise. Sometimes, an error occurs, but it is possible to show that a repetition of the iterations $p, 2p, 3p \cdots$ is sufficient to correct this error.

Ercegovac and Lang [9] propose to evaluate the scale factor in parallel with the CORDIC iterations, and then to divide the results x_n and y_n by this factor.

TABLE I
DIFFERENT FUNCTIONS COMPUTABLE USING CORDIC

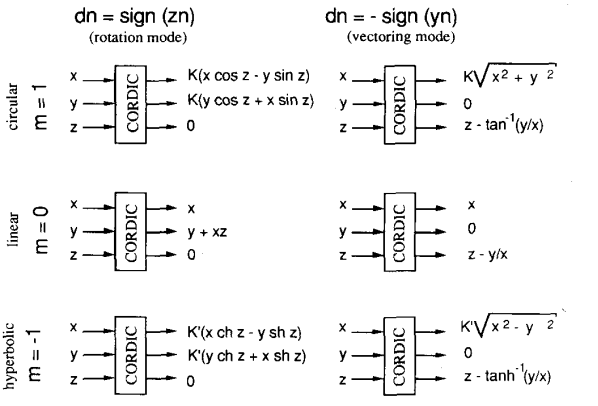

TABLE II
VALUES OF $\sigma(n)$ AND e_n IN TABLE I

$m = +1$	$\sigma(n) = n$	$e_n = \arctan 2^{-n}$
$m = -1$	$\sigma(n) = 1, 2, 3, 4, 4, 5, 6, \cdots, 12, 13, 13, 14, 15, \cdots, 39, 40, 40, 41, \cdots$	$e_n = \tanh^{-1} 2^{-n}$
$m = 0$	$\sigma(n) = n$	$e_n = 2^{-n}$

As Takagi, Asada, and Yajima's methods, our solution gives a constant scale factor. Moreover, it needs no repetition of iterations, and is therefore faster than theirs. However, it leads probably to a little more hardware, since we perform in parallel 2 classical CORDIC iterations, in a way similar to Ercegovac and Lang's *on the fly* conversion of numbers from redundant to nonredundant binary representation [7].

III. BRANCHING CORDIC ROTATION

We shall use the following properties:

(P1) $$\arctan 2^{-n} < \sum_{k=n+1}^{\infty} \arctan 2^{-k} < 2^{-n}$$

(P2) $$\arctan 2^{-n} - \sum_{k=n+1}^{n+p} \arctan 2^{-k} < 2^{-n-p}.$$

(P1) is well known (see for instance [13] for proof). (P2) is a consequence of (P1). Indeed, one can deduce from (P1):

$$\arctan 2^{-n} - \sum_{k=n+1}^{n+p} \arctan 2^{-k} \leq \sum_{k=n+p+1}^{\infty} \arctan 2^{-k} \leq \sum_{k=n+p+1}^{\infty} 2^{-k} = 2^{-n-p}.$$

A. Computation of the Sequence z_n

Let us examine the basic principle of our method. We look for a decomposition $\sum_{k=0}^{\infty} d_k \arctan 2^{-k}$ of a number θ, with $d_k = \pm 1$. In order to do that, we start like the methods presented above. We build a sequence z_n defined by $z_{n+1} = z_n - d_n \arctan 2^{-n}$. At each step, we examine p digits of z_n in order to decide the value of d_n (in practice, $p = 3$). Then

- If the examination of these p digits is sufficient to be sure that $z_n > 0$, we take $d_n = +1$.
- If the examination of these p digits is sufficient to be sure that $z_n < 0$, we take $d_n = -1$.
- If the examination of these p digits is not sufficient to know the sign of z_n, then *we try two computations in parallel:* the former assuming $z_n > 0$ (and therefore with $d_n = 1$), the latter assuming $z_n < 0$ (with $d_n = -1$). We call *branching* this operation.

One may believe that in one or both of the parallel computations, new branchings may occur, creating a lot of parallel computations. This is not true since *if a computation causes a branching, this proves that the associated value of z_n is sufficiently small to ensure the convergence ($z_n \to 0$) of at least one of its two subcomputations, therefore, all other computations performed in parallel with it can be stopped.* Thus, there are always at most two parallel computations. Another objection may be: if no new branching appears, how does one decide which computation is the correct one. We shall see that in such a case, *both* computations lead to a correct approximation of the sine and cosine.

In practice, we build in parallel two sequences z_n^+ and z_n^- (in two CORDIC modules called "+" and "−") defined by the following algorithm. We suppose that we have a function $\text{eval}(z_n)$ which returns at step n a value in $\{-1, 0, 1\}$, such that

- If $\text{eval}(z_n) \neq 0$, then $\text{eval}(z_n) = \text{sign}(z_n)$
- If $\text{eval}(z_n) = 0$ then $|z_n| < 2^{-n-1}$.

```
Algorithm branching − CORDIC
Procedure updatez (n);
  begin
    in parallel :
        z+_{n+1} := z+_n − d+_n arctan 2^{−n};      {in module "+"}
        z−_{n+1} := z−_n − d−_n arctan 2^{−n};      {in module "−"}
    in parallel :
        s+ := eval z+_{n+1}
        s− := eval z−_{n+1}
  end ;
Begin
    i := 0 ;          {initializations}
    z+_0 := z−_0 := θ :
    s+ := s− := eval(θ) ;
1 : while s+ ≠ 0 and s− ≠ 0 do {while no branching}
      begin
```

```
                d_i^+ := s^+ ;
                d_i^- := s^- ;
                updatez (i) ;
                i := i + 1 ;
              end ;
2 : {a branching is occurring}
                d_i^+ := 1 ;
                d_i^- := −1 ; {branching}
                updatez (i) ;
                i := i + 1 ;
3 : while (s^+ = −1) and (s^- = +1) do {while branching}
              begin
                d_i^+ := s^+ ;
                d_i^- := s^- ; {branching continuing}
                updatez (i) ;
                i := i + 1 ;
              end ;
            {new branching, or end of branching}
            if s^+ = 0 then {module "+" performed the good computation}
              begin
                z_i^- := z_i^+ ;
                go to 2 {branching treatment}
              end ;
            else if s^+ = +1 then {module "+" performed the good computation}
              begin
                z_i^- := z_i^+ ;
                s^- := s^+ ;
                go to 1 {branching terminated}
              end ;
            else if s^- = 0 then {module "−" performed the good computation}
              begin
                z_i^+ := z_i^- ;
                go to 2 {branching treatment}
              end ;
            else if s^- = −1 then {module "−" performed the good computation}
              begin
                z_i^+ := z_i^- ;
                s^+ := s^- ;
                go to 1 {branching terminated}
              end ;
```

Fig. 1 sums up the different cases occurring when performing the algorithm.

Theorem 1: The sequences (z_n^+) and (z_n^-) generated by the previous algorithm satisfy

(1) At step i, at least one of the terms $|z_i^+|$ and $|z_i^-|$ is lower than or equal to $\sum_{k=i}^{\infty} \arctan 2^{-k} \leq 2^{-i+1}$.

(2) Both terms are lower than or equal to 3.2^{-i+1}.

■

Proof: a) Proof of (1):

First, let us prove 1) by induction. We assume that 1) is true at step $i-1$: at least one of the terms (z_{i-1}^+) and (z_{i-1}^-) has an absolute value lower than or equal to $\sum_{k=i-1}^{\infty} \arctan 2^{-k}$, and we compute $s^+ = \text{eval}(z_{i-1}^+)$ and $s^- = \text{eval}(z_{i-1}^-)$.

1) If $s^+ \neq 0$ and $s^- \neq 0$ then

□ If we are not in a branching (both modules produce the same computation), then $|z_{i-1}^+| \leq \sum_{k=i-1}^{\infty} \arctan 2^{-k}$. Since $s^+ \neq 0$, s^+ has the sign of z_{i-1}^+, and we take $d_{i-1}^+ = s^+$. Therefore

$$|z_i^+| = |z_{i-1}^+| - \arctan 2^{-i+1}$$
$$\leq \sum_{k=i-1}^{\infty} \arctan 2^{-k} - \arctan 2^{-i+1}$$

hence:

$$-\sum_{k=i}^{\infty} \arctan 2^{-k} \leq z_i^+ \leq \sum_{k=i}^{\infty} \arctan 2^{-k}.$$

□ If we are in a branching

- if $s^+ = -1$ and $s^- = +1$ (label 3 of the algorithm:

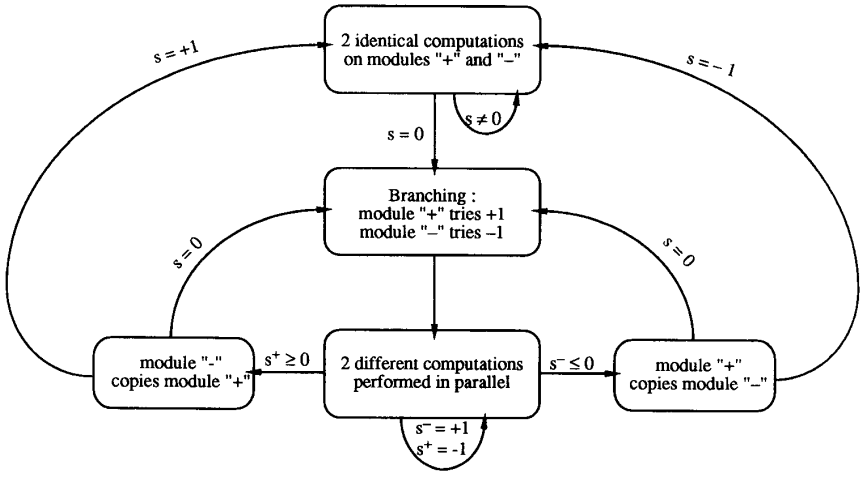

Fig. 1. Diagram of branching-CORDIC.

branching is continuing). Consider the case where z_{i-1}^{+} has an absolute value lower than or equal to $\sum_{k=i}^{\infty} \arctan 2^{-k}$ (if not, the case is absolutely symmetrical by interchanging modules "+" and "−"). As previously, we take $d_{i-1}^{+} = s^{+}$. Therefore $|z_i^{+}| \leq \sum_{k=i}^{\infty} \arctan 2^{-k}$.

- if $s^{+} = +1$ or $s^{-} = -1$ (end of branching). Assume that $s^{+} = +1$ (the case $s^{-} = -1$ is symmetrical), and that the current branching started at step $k-1$. Since no new branching or end of branching occurred before step i, all values of s^{+}, from step k to step $i-2$ have been found equal to –1. Therefore, $z_{i-2}^{+} < 0$ and $z_{i-1}^{+} > 0$. Thus, since $z_{i-1}^{+} = z_{i-2}^{+} + \arctan 2^{-i+2}$:

$$z_{i-1}^{+} \leq \arctan 2^{-i+2} \leq \sum_{k=i-1}^{\infty} \arctan 2^{-k}.$$

From this last result, we deduce as previously

$$|z_i^{+}| \leq \sum_{k=1}^{\infty} \arctan 2^{-k}.$$

2) If $s^{+} = 0$ then (the case $s^{-} = 0$ is symmetrical): Since $s^{+} = 0$, we deduce $|z_{i-1}^{+}| \leq 2^{-i}$. Now we compute

- $z_i^{+} = z_{i-1}^{+} - \arctan 2^{-i+1}$
- $z_i^{-} = z_{i-1}^{+} + \arctan 2^{-i+1}$

and deduce that $\text{Min}\left(|z_i^{+}|; |z_i^{-}|\right) \leq \text{Max}\left(2^{-i}; \arctan 2^{-i+1}\right) \leq \sum_{k=i}^{\infty} \arctan 2^{-k}$.

Thus (1) is proved.

b) Proof of (2):

- ☐ If we are not in a branching, then $|z_i^{+}| = |z_i^{-}| \leq \sum_{k=i}^{\infty} \arctan 2^{-k} < 2^{-i+1} < 3.2^{-i+1}$.
- ☐ If we are in a branching: Assume that the current branching started at step p. There exists z_p^{0} (equal to z_p^{+} or z_p^{-}) such that

$$z_i^{+} = z_p^{0} - \arctan 2^{-p} + \sum_{k=p+1}^{i-1} \arctan 2^{-k} \quad \text{and}$$

$$z_i^{-} = z_p^{0} + \arctan 2^{-p} \sum_{k=p+1}^{i-1} \arctan 2^{-k}.$$

Therefore $z_i^{-} - z_i^{+} = 2(\arctan 2^{-p} - \sum_{k=p+1}^{i-1} \arctan 2^{-k}) < 2.2^{-i+1}$ [from (P2)]. Moreover, since we are in a branching, z_{i-1}^{-} was positive and z_{i-1}^{+} was negative, thus $z_i^{-} = z_{i-1}^{-} - \arctan 2^{-i+1} \geq -\arctan 2^{-i+1}$, and $z_i^{+} = z_{i-1}^{+} + \arctan 2^{-i+1} \geq \arctan 2^{-i+1}$. Therefore, $z_i^{-} - z_i^{+} \geq -2 \arctan 2^{-i+1} \geq -2^{-i+2}$. Therefore $|z_i^{-} - z_i^{+}| \leq 2^{-i+2}$.

Therefore, since $|z_i^{+}| \leq 2^{-i+1}$ or $|z_i^{-}| \leq 2^{-i+1}$, $|z_i^{+}|$ and $|z_i^{-}|$ are less than $2^{-i+2} + 2^{-i+1} = 3.2^{-i+1}$. ■

Now, we give a convenient method enabling the computation of function eval(z_n). Let us assume that we are at step i: we want to evaluate z_i^{-} and z_i^{+} from z_{i-1}^{-} and z_{i-1}^{+}. The absolute values of z_{i-1}^{-} and z_{i-1}^{+} are less than or equal to 3.2^{-i}, and eval(z_i) must satisfy at step i:

- If eval$(z_i) \neq 0$ then eval$(z_i) =$ sign(z_i)
- If eval$(z_i) = 0$ then $|z_i| \leq 2^{-i-1}$.

Let us define the *truncation* of a digit chain as $\tau(x_{-p}x_{-p+1} \cdots x_{-1}x_0 \cdot x_1x_2 \cdots) = x_{-p} \cdots x_{-1}x_0$.

τ is not the classical truncation or integer part function, and depends on the *representations* of numbers. For instance, the numbers $10.\bar{1}101$ and $1\bar{1}.111\bar{1}$ represent the same value, however

$$\tau(10.\bar{1}101) = 10(\text{radix } 2) = 2(\text{radix } 10) \neq \tau(1\bar{1}.111\bar{1}) = 1.$$

The following lemma will help us to find the sign of a number by examining only 3 digit positions.

Lemma 1: Let $z = z_{-p}z_{-p+1} \cdots z_{-1}z_0 \cdot z_1z_2z_3 \cdots$ be a number satisfying $|z| < 3.2^{-j}$. Let C be the integer $z_{j-2}z_{j-1}z_j$ modulo $8 = \tau\left(2^{j}z\right)$ modulo 8.

- If C is equal to 1, 2, or 3 then $z \geq 0$.
- If C is equal to 5, 6, or 7 then $z \leq 0$.
- If $C = 0$ then $|z| \leq 2^{-j}$.
- the case $C = 4$ is impossible.

Proof: From $|z| < 3.2^{-j}$ we deduce $|2^j z| < 3$. Therefore, since for every $x|\tau(x) - x| \leq 1$, $|\tau(2^j z)| < 4$. Thus, $\tau(2^j z)$ is –3, –2, –1, 0, 1, 2, or 3, and $C = 4$ is impossible.

If $\tau(2^j z)$ is not equal to zero, then obviously, the signs of z and $\tau(2^j z)$ are equal. Therefore

- if $C = 1$, 2, or 3 then, since $C = \tau(2^j z)$ modulo 8 and $\tau(2^j z) \in \{-3, -2, -1, 0, 1, 2, 3\}$, $\tau(2^j z)$ is equal to 1, 2, or 3, hence $z \geq 0$.
- if $C = 5$, 6, or 7 then, since $C = \tau(2^j z)$ modulo 8 and $\tau(2^j z) \in \{-3, -2, -1, 0, 1, 2, 3\}$, $\tau(2^j z)$ is equal to –1, –2, or –3, hence $z \leq 0$.
- if $C = 0$ then, since $C = \tau(2^j z)$ modulo 8 and $\tau(2^j z) \in \{-3, -2, -1, 0, 1, 2, 3\}$, $\tau(2^j z)$ is equal to zero, hence $|2^j z| \leq 1$, i.e., $|z| \leq 2^{-j}$.

Example: If $z_{j-2} = z_{j-1} = z_j = \bar{1}$, then $z_{j-2} z_{j-1} z_j = -7$, i.e., $z_{j-2} z_{j-1} z_j = 1$ mod 8, and we deduce from the lemma that z is nonnegative.

Table III gives the sign of z for the different cases that may occur. This table may be used to implement the function eval, which may be reduced to a function of 3 digits:

function $eval(a, b, c)$: **returns** 1 if $[abc]$ mod 8 is 1, 2, or 3
$\bar{1}$ if $[abc]$ mod 8 is 5, 6, or 7
0 if $[abc]$ mod 8 is 0
anything if $[abc]$ mod 8 is 4.

B. Computation of Rotations

Now, with each sequence (z_i^-) and (z_i^+) is associated a (x, y) rotation. Module "+" performs

$$x_{i+1}^+ = x_i^+ - d_i^+ y_i^+ 2^{-i}$$
$$y_{i+1}^+ = y_i^+ + d_i^+ x_i^+ 2^{-i}$$

while module "−" performs the rotations

$$x_{i+1}^- = x_i^- - d_i^- y_i^- 2^{-i}$$
$$y_{i+1}^- = y_i^- + d_i^- x_i^- 2^{-i}.$$

In the algorithm *branching-CORDIC* presented above, one has to add the two following instructions:

☐ In parallel with *updatez* (i), perform the two (x, y) rotations.

☐ When a new branching or end of branching occurs:

- In parallel with an instruction "$z_i^+ := z_i^-$," perform $x_i^+ := x_i^-$ and $y_i^+ := y_i^-$.
- In parallel with an instruction "$z_i^- := z_i^+$," perform $x_i^- := x_i^+$ and $y_i^- := y_i^+$.

Fig. 2 presents the global architecture of a CORDIC processor that implements our algorithm.

TABLE III
SIGN OF z AT STEP $j - 1$ OBTAINED BY EXAMINING ONLY 3 DIGIT POSITIONS

z_{j-2}	z_{j-1}	z_j	sign	$[z_{j-2}z_{j-1}z_j]$ mod 8
$\bar{1}$	$\bar{1}$	$\bar{1}$	+	1
$\bar{1}$	$\bar{1}$	0	+	2
$\bar{1}$	$\bar{1}$	1	+	3
$\bar{1}$	0	$\bar{1}$	+	3
$\bar{1}$	0	0		4 impossible
$\bar{1}$	0	1	-	5
$\bar{1}$	1	$\bar{1}$	-	5
$\bar{1}$	1	0	-	6
$\bar{1}$	1	1	-	7
0	$\bar{1}$	$\bar{1}$	-	5
0	$\bar{1}$	0	-	6
0	$\bar{1}$	1	-	7
0	0	$\bar{1}$	-	7
0	0	0	??	0 branching
0	0	1	+	1
0	1	$\bar{1}$	+	1
0	1	0	+	2
0	1	1	+	3
1	$\bar{1}$	$\bar{1}$	+	1
1	$\bar{1}$	0	+	2
1	$\bar{1}$	1	+	3
1	0	$\bar{1}$	+	3
1	0	0		4 impossible
1	0	1	-	5
1	1	$\bar{1}$	-	5
1	1	0	-	6
1	1	1	-	7

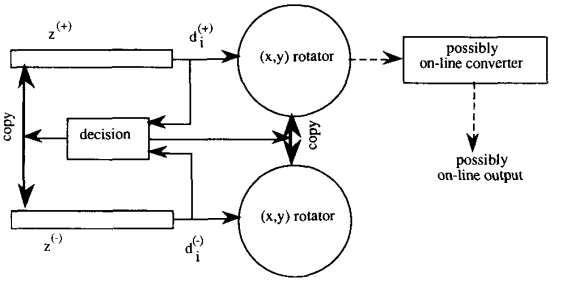

Fig. 2. Global architecture of a branching CORDIC processor.

IV. On-Line Implementation of Algorithm

Now, we propose an on-line implementation of our algorithm for sine and cosine functions. On-line algorithms receive their input data and give their results serially, most significant digit first. The *delay* of an on-line algorithm is the number δ such that ν digits of the result are deduced from $\nu + \delta$ digits of the input values. δ may be either positive or negative. The use of on-line arithmetic implies some changes in our algorithm:

1) The input value $z_0 = \theta$ of the rotation angle is carried serially. Therefore, at step i, only i digits of this value will be available.
2) x_n and y_n will be output on-line. In order to do that, these values have to be transformed: it is the *digitization* process.

Since we assume that we compute only sine and cosine functions, the angle z_0 is the only input value; $x_0 = 1/K$ and $y_0 = 0$ are internal constants. It is possible to imagine an on-line rotation using our algorithm, with x_0 and y_0 carried serially. However, it would be necessary to multiply x_0 and y_0 by the inverse of the scale factor, in on-line mode, before the rotation (or to multiply the final values of x and y after the rotation). First, we examine the modification to our algorithm, called for by the on-line input of the rotation angle.

A. Modifications Due to the On-Line Input

Let us assume that we want to compute the sine and cosine of $z_0 = \theta = \sum_{i=0}^{\infty} \theta_i 2^{-i}$. Since at the beginning z_0 is not known with full precision, instead of the sequences z_n^+ and z_n^- defined above, we manipulate two sequences $\hat{z}_n^+$ and $\hat{z}_n^-$ defined by the following *on-line branching CORDIC* algorithm. We assume that now, function eval $(\hat{z}_n)$ satisfies at step n

- If eval $(\hat{z}_n) = 1$ then $\hat{z}_n \geq 2^{-n-2}$
- If eval $(\hat{z}_n) = -1$ then $\hat{z}_n \leq -2^{-n-2}$
- If eval $(\hat{z}_n) = 0$ then $|\hat{z}_n| \leq 2^{-n-1}$.

This new function *eval* may be implemented in a way very similar to the one presented in Section III-A, examining only 4 digits of $\hat{z}_n$. The algorithm becomes

```
Algorithm On − line branching − CORDIC
procedure updatez (n) ;
  begin
    in parallel :
      ẑ+_{n+1} := ẑ+_n − d+_n arctan 2^{−n} + θ_{n+3} 2^{−n−3}; {in module "+"}
      ẑ−_{n+1} := ẑ−_n − d−_n arctan 2^{−n} + θ_{n+3} 2^{−n−3}; {in module "−"}
    in parallel :
      s+ := eval ẑ+_{n+1};
      s− := eval ẑ−_{n+1};
  end ;
Begin
    i := 0 ;                    {initializations}
    ẑ+_0 := ẑ−_0 := 0.θ1θ2 = θ1 2^{−1} + θ2 2^{−2} ;
    s+ := s− := eval (ẑ+_0) ;
1 : while s+ ≠ 0 and s− ≠ 0 do {while no branching}
      begin
        d+_i := s+ ;
        d−_i := s− ;
        updatez (i) ;
        i := i + 1 ;
      end ;
2 :          {a branching is occurring}
    d+_i := 1 ;
    d−_i := −1 ;                {branching}
    updatez (i) ;
    i := i + 1 ;
3 : while (s+ = −1) and (s− = +1) do          {while branching}
      begin
        d+_i := s+ ;
        d−_i := s− ; {branching continuing}
        updatez (i) ;
        i := i + 1 ;
      end ;
    {new branching, or end of branching}
    if s+ = 0 then          {module "+" performed the good computation}
      begin
        ẑ−_i := ẑ+_i ;
        go to 2 ; {branching treatment}
      end ;
    else if s+ = +1 then          {module "+" performed the good computation}
      begin
        ẑ−_i := ẑ+_i ;
        s− := s+ ;
```

```
    go to 1 ;                {branching terminated}
  end ;
else if s⁻ = 0 then          {module "−" performed the good computation}
  begin
    ẑ_i^+ := ẑ_i^− ;
    go to 2 ;                {branching treatment}
  end ;
else if s⁻ = −1 then         {module "−" performed the good computation}
  begin
    ẑ_i^+ := ẑ_i^− ;
    s^+ := s^− ;
    go to 1 ;                {branching terminated}
  end ;
```

Now, let us define (only for proof: they are not computable in practice) two sequences z_n^+ and z_n^- as

1) $z_0^+ = z_0^- = \theta$.
2) add to procedure *updatez* (n) in the algorithm the instructions

$$z_{n+1}^+ = z_n^+ - d_n^+ \arctan 2^{-n}$$
$$z_{n+1}^- = z_n^- - d_n^- \arctan 2^{-n}$$

3) in the algorithm, when an instruction "$\hat{z}_i^+ := \hat{z}_i^-$" is performed, perform also "$z_i^+ := z_i^-$" and when instruction $\hat{z}_i^- := \hat{z}_i^+$" is performed, perform also "$z_i^- := z_i^+$."

Obviously, in a practical implementation, we do not compute these sequences (this is impossible since at the beginning of the computation, θ is not known with full precision). They are defined only for proof. We assume that we have taken $x_0 = 1/K$ and $y_0 = 0$, in order to obtain a final scale factor equal to 1. The values x_n and y_n of the nth step of the CORDIC iteration are equal to $R_n \cos\theta_n$ and $R_n \sin\theta_n$, where R_n is equal to $\prod_{i=n}^{\infty}(1+2^{-2i})^{-1/2}$, with $\theta_n = \theta - z_n$. Therefore we need to bound $|z_n| = |\theta_n - \theta|$.

Theorem 2: 1) at step i, at least one of the terms $|z_i^+|$ and $|z_i^-|$ is lower than or equal to $\sum_{k=i}^{\infty} \arctan 2^{-k} \leq 2^{-i+1}$
2) Both terms are lower than or equal to 3.2^{-i+1}. ■

Proof: We do not give a detailed proof, since it is very similar to that of theorem 1 if we remark that

$$\begin{cases} z_i^+ = \hat{z}_i^+ + \sum_{k=i+3}^{\infty} \theta_k 2^{-k} \\ z_i^- = \hat{z}_i^- + \sum_{k=i+3}^{\infty} \theta_k 2^{-k} \end{cases}$$

which implies

$$\begin{cases} |z_i^+ - \hat{z}_i^+| \leq 2^{-i-2} \\ |z_i^- - \hat{z}_i^-| \leq 2^{-i-2} \end{cases}.$$

Therefore

□ If $\text{eval}(\hat{z}_i^+) = 1$ then $z_i^+ \geq 0$
□ If $\text{eval}(\hat{z}_i^-) = 1$ then $z_i^- \geq 0$
□ If $\text{eval}(\hat{z}_i^+) = -1$ then $z_i^+ \leq 0$
□ If $\text{eval}(\hat{z}_i^-) = -1$ then $z_i^- \leq 0$
□ If $\text{eval}(\hat{z}_i^+) = 0$ then $|z_i^+| \leq 3.2^{-i-2}$
■ If $\text{eval}(\hat{z}_i^-) = 0$ then $|z_i^-| \leq 3.2^{-i-2}$.

B. Error Evaluation

1) Use of the Outputs of One Module: Let us consider here the values x_n, y_n and z_n of any of the 2 modules "+" and "−." As previously, we assume that we want to compute the sine and/or cosine of θ, $|\theta| \leq 1$, and we denote $\theta_n = \theta - z_n$. Since we have: $|z_n| \leq 3 \cdot 2^{-n+1}$, we deduce $|\theta_n - \theta| \leq 3 \cdot 2^{-n+1}$. Therefore we have

$$\begin{cases} |\cos\theta_n - \cos\theta| \leq 3.2^{-n+1} \\ |\sin\theta_n - \sin\theta| \leq 3.2^{-n+1} \end{cases}.$$

We assume that we have taken $x_0 = \prod_{i=0}^{\infty}(1+2^{-2i})^{-1/2}$ and $y_0 = 0$, in order to obtain a final scale factor equal to 1. The values x_n and y_n of the nth step of the CORDIC iteration are equal to $R_n \cos\theta_n$ and $R_n \sin\theta_n$, where R_n is equal to $\prod_{i=n}^{\infty}(1+2^{-2i})^{-1/2}$. In order to output $\cos\theta$ and $\sin\theta$ in on-line mode, we have to estimate $|x_n - \cos\theta|$ and $|y_n - \sin\theta|$.

We have

$$\begin{aligned} |x_n - \cos\theta| &= |R_n \cos\theta_n - \cos\theta| \\ &= |\cos\theta_n - \cos\theta + (R_n - 1)\cos\theta_n| \\ &\leq |\cos\theta_n - \cos\theta| + |R_n - 1| \\ &\leq 3 \cdot 2^{-n+1} + |R_n - 1|. \end{aligned}$$

We obtain in a similar fashion

$$|x_n - \sin\theta| \leq 3 \cdot 2^{-n+1} + |R_n - 1|.$$

Therefore, we have to estimate the value $|R_n - 1| = 1 - R_n$.

Lemma: $1 - R_n \leq (2/3)\,2^{-2n}$. ■

Proof: R_n is equal to $\prod_{i=n}^{\infty}(1+2^{-2i})^{-1/2}$, therefore $\log R_n = -(1/2)\sum_{i=n}^{\infty}\log(1+4^{-i}) \geq -(1/2)\sum_{i=n}^{\infty}4^{-i}$. Therefore $\log R_n \geq -(1/2)[4^{-n+1}/3]$, thus $R_n \geq e^{-(1/2)[4^{-n+1}/3]} \geq 1 - 1/2[4^{-n+1}/3]$ thus $1 - R_n \leq (1/2)[4^{-n+1}/3] = 2^{-2n}$ and the lemma is proved. ■

From the lemma, we deduce the following result:

Theorem 3: The values $x_n = (x_n^+ \text{ or } x_n^-)$ and $y_n = (y_n^+ \text{ or } y_n^-)$ satisfy $|x_n - \cos\theta|$ and $|y_n - \sin\theta|$ are less than 2^{-n+3}. ■

Proof: From the lemma, and from the relations

$$|y_n - \sin\theta| \leq 3\cdot 2^{-n+1} + |R_n - 1|$$
$$|x_n - \cos\theta| \leq 3\cdot 2^{-n+1} + |R_n - 1|$$

we deduce that $|x_n - \cos\theta|$ and $|x_n - \sin\theta|$ are lower than or equal to

$$3\cdot 2^{-n+1} + \frac{2}{3}2^{-2n} = \left(3 + \frac{2}{3}2^{-n-1}\right)\cdot 2^{-n+1} \leq \left(3+\frac{1}{3}\right)\cdot 2^{-n+1} < 2^{-n+3}.$$

■

2) Use of the Outputs of Both Modules: Assume that $\theta \in [0, +\pi/2]$. In Section IV-B1, we used the outputs x_n and y_n of any of the 2 modules "+" and "−". It is possible to obtain a better result if we use the values $(1/2)(x_n^+ + x_n^-)$ and $(1/2)(y_n^+ + y_n^-)$. We already showed in Section IV-B1 that

$$|x_n^+ - \cos\theta| < 2^{-n+3} \qquad |y_n^+ - \sin\theta| < 2^{-n+3}$$
$$|x_n^- - \cos\theta| < 2^{-n+3} \qquad |y_n^- - \sin\theta| < 2^{-n+3}. \quad \text{(A)}$$

Moreover, using the relations given by Theorem 2, it may be easily shown (the proof is the same as in Section IV-B1) that

- at least one value $X_n \in \{x_n^+, x_n^-\}$ satisfies $|X_n - \cos\theta| < 2^{-n+2}$
- at least one value $Y_n \in \{y_n^+, y_n^-\}$ satisfies $|Y_n - \sin\theta| < 2^{-n+2}$

Let us define x_n and y_n as $x_n = (x_n^+ + x_n^-)/2$ and $y_n = (y_n^+ + y_n^-)/2$.

1) *If we are not in a branching,* then, since $x_n = x_n^+ = x_n^-$ and $y_n = y_n^+ = y_n^-$, we deduce from (B) that $|x_n - \cos\theta|$ and $|y_n - \sin\theta|$ are less than 2^{-n+2}.
2) *If we are in a branching.* Let us define θ_n^+ and θ_n^- as

$$\theta_n^+ = \theta - z_n^+ \qquad \theta_n^- = \theta - z_n^-.$$

We have $x_n^+ = R_n\cos\theta_n^+$, $y_n^+ = R_n\sin\theta_n^+$, $x_n^- = R_n\cos\theta_n^-$, and $y_n^- R_n\sin\theta_n^-$. Therefore

$$|x_n - \cos\theta| = \left|\frac{x_n^+ - \cos\theta}{2} + \frac{x_n^- - \cos\theta}{2}\right| = \left|\frac{R_n\cos\theta_n^+ - \cos\theta}{2} + \frac{R_n\cos\theta_n^- - \cos\theta}{2}\right| \leq \left|\frac{\cos\theta_n^+ - \cos\theta}{2} + \frac{\cos\theta_n^- - \cos\theta}{2}\right| + |R_n - 1|.$$

Since we are in a branching, $z_n^+ \leq 0$ and $z_n^- \geq 0$. Therefore $\theta_n^+ \geq \theta$ and $\theta_n^- \leq \theta$.

Therefore, since $\theta \in [-\pi/2, \pi/2]$, $(\cos\theta_n^+ - \cos\theta)$ and $(\cos\theta_n^- - \cos\theta)$ have opposite signs. From this we deduce

$$\left|\frac{\cos\theta_n^+ - \cos\theta}{2} + \frac{\cos\theta_n^- - \cos\theta}{2}\right| \leq \mathrm{Max}\left(\left|\frac{\cos\theta_n^+ - \cos\theta}{2}\right|; \left|\frac{\cos\theta_n^- - \cos\theta}{2}\right|\right) \leq 3.2^{-n}.$$

Thus $|x_n - \cos\theta| \leq 3.2^{-n} + |R_n - 1| \leq 3.2^{-n} + (2/3)\cdot 2^{-2n} \leq 2^{-n+2}$. Since the same proof may be used for the sine function, we deduce

Theorem 4: The values $x_n = (x_n^+ + x_n^-)/2$ and $y_n = (y_n^+ + y_n^-)/2$ satisfy $|x_n - \cos\theta|$ and $|y_n - \sin\theta|$ are less than 2^{-n+2}.

C. Digitization

In the following, we shall denote $x^{(m)}$ the ith digit of the mth iterate value $x^{(m)}$ of x. $x^{(m)}$ may be obtained as in Section III-B1 or as in Section IV-B2. Theorems 3 and 4 do not ensure that $x^{(n)}$ and $y^{(n)}$ are output-on-line. Although the value $x^{(n)}$ of each module satisfies $|x^{(n)} - \cos\theta| \leq 2^{-n+3}$, this does not prove that the $n-3$ most significant digits of $x^{(n)}$ are the $n-3$ most significant digits of an on-line result. In order to produce an on-line result, $x^{(n)}$ and $y^{(n)}$ must be transformed in order to give a new digit result at each iteration. This procedure is sometimes called *digitization* (see for instance [12]).

Assume that at step m an algorithm for computing function $f(\theta)$ gives a result $x^{(m)}$ satisfying

$$|x^{(m)} - f(\theta)| \leq 2^{-m+p}.$$

then at step m, we can give in on-line mode the $p+2-m$th digit of $f(\theta)$.

The algorithm presented below gives in on-line mode the successive digits $\dot{x}_1, \dot{x}_2, \dot{x}_3, \cdots \dot{x}_n, \cdots$ of a signed-digit representation of $f(\theta)$.

Digitization Algorithm

Assume we are at step $m+2$. We have obtained a value $x^{(m+2)}$ satisfying $|x^{(m+2)} - f(\theta)| \leq 2^{-m-2+p}$. From $x^{(m+1)}$ we have already computed $\dot{x}_1, \dot{x}_2, \dot{x}_3, \cdots \dot{x}_{m-p-1}$.

The number $x_*^{(m+2)} = 0.x_1^{(m+2)}x_2^{(m+2)}x_3^{(m+2)}\cdots x_{m+2-p}^{(m+2)}$ obtained by truncating the signed-digit representation of $x^{(m+2)}$ after its $m+2-p$th position satisfies

$$|x_*^{(m+2)} - f(\theta)| \leq 2^{-m+p-1}. \quad \text{(R)}$$

Let us denote $\phi = 0.\dot{x}_1\dot{x}_2\dot{x}_3, \cdots \dot{x}_{m-p-1}$.

The interval $I_{\bar{1}}$ of the numbers representable if we choose $\dot{x}_{m-p} = \bar{1}$ is $[\phi - 2^{-m+p+1}, \phi]$, the interval I_0 of the numbers representable with $\dot{x}_{m-p} = 0$ is $[\phi - 2^{-m+p}, \phi + 2^{-m+p}]$, and the interval I_1 of the numbers representable with $\dot{x}_{m-p} = 1$ is $[\phi, \phi + 2^{-m+p+1}]$ (see Fig. 3). From this we deduce easily (see Fig. 3):

□ If $x_*^{(m+2)} \leq \phi - 2^{-m+p-1}$ then, from (R), $f(\theta) \leq \phi$ thus $f(\theta) \in I_{\bar{1}}$: we choose $\dot{x}_{m-p} = \bar{1}$.

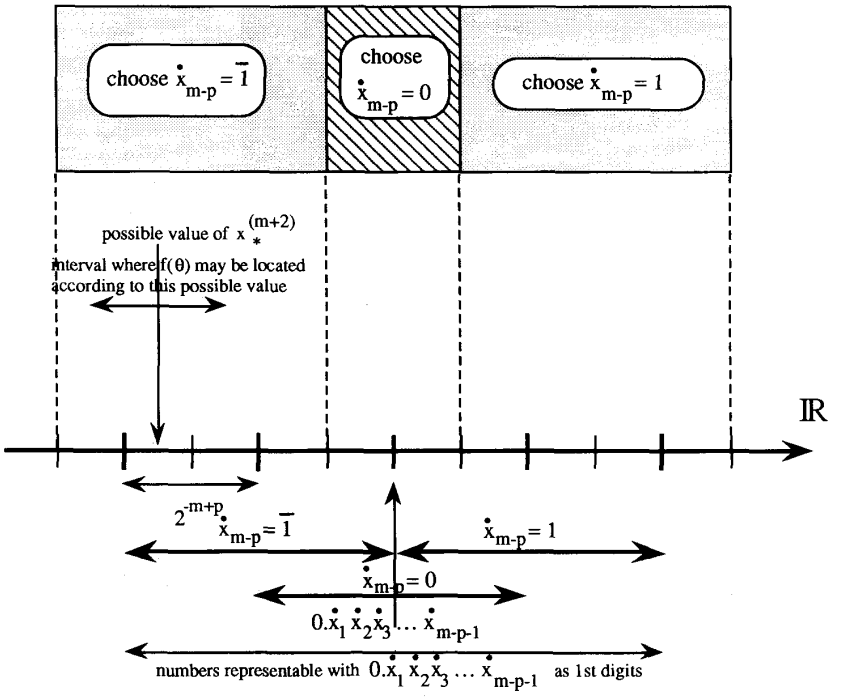

Fig. 3. The digitization process.

- If $\phi - 2^{-m+p-1} \leq x_*^{(m+2)} \leq \phi + 2^{-m+p-1}$ then, from (R), $\phi - 2^{-m+p} \leq x_*^{(m+2)} \leq \phi + 2^{-m+p}$ thus $f(\theta) \in I_0$: we choose $\dot{x}_{m-p} = 0$.
- If $x_*^{(m+2)} \geq \phi + 2^{-m+p-1}$ then, from (R), $f(\theta) \geq \phi$ thus $f(\theta) \in I_1$: we choose $\dot{x}_{m-p} = 1$.

This algorithm is easily implementable since it needs the examination of only 5 digits of $x_*^{(m+2)}$. Let us call K the integer $2^{m-p+2}\ (x_*^{(m+2)} - \phi)$. From (R) and the obvious relation $|f - f(\theta)| \leq 2^{-m+p+1}$, we deduce $|\phi - x_*^{(m+2)}| \leq 5.2^{-m+p-1}$. Thus, $|K| \leq 10$. The algorithm becomes

$$\dot{x}_{m-p} = \begin{cases} \bullet\bar{1} & \text{if } K \leq -3 \\ \bullet\bar{1} & \text{or 0 if } K = -2 \\ \bullet 0 & \text{if } -1 \leq K \leq +1 \\ \bullet 0 & \text{or 1 if } K = 2 \\ \bullet 1 & \text{if } K \geq 3 \end{cases}.$$

Since $|K| \leq 10$, it is easy to show (see the proof of Lemma 1) that if we replace K by the value K^* obtained by taking only its 5 least significant digits ($K^* = K \bmod 32$), we obtain

$$\dot{x}_{m-p} = \begin{cases} \bullet\bar{1} & \text{if } 22 \leq K^* \bmod 32 \leq 29 \\ \bullet\bar{1} & \text{or 0 if } K^* \bmod 32 = 30 \\ \bullet 0 & \text{if } K^* \bmod 32 = 31\ 0 \text{ or } 1 \\ \bullet 0 & \text{or 1 if } K^* \bmod 32 = 2 \\ \bullet 1 & \text{if } 3 \leq K^* \bmod 32 \leq 10 \end{cases}.$$

The values $K^* \bmod 32 = 11, 12, 13, \cdots, 21$ are impossible.

D. On-Line Delay of Our Algorithm

Both CORDIC modules produce a result. Assume that we want to compute the cosine function (the same applies for the sine function). If we use for digitization the output values of only one module (e.g., module "+"), then at step i of the algorithm, we have an error $|x_i - \cos\theta| \leq 2^{-i+3}$ using digits $\theta_0, \theta_1, \cdots, \theta_i, \theta_{i+1}$ of θ. Our digitization algorithm enables us to give in on-line mode the i-5th digit of $\cos\theta$. Therefore, from digits 0 to $i+1$ of θ we deduce digits 0 to i–5 of $f(\theta)$. Therefore the on-line delay of our algorithm is $(i+1) - (i-5) = 6$.

If we use for digitization the average values of the outputs of both modules (as in part Section IV-B2), since at step i, we have an error $|x_i - \cos\theta| \leq 2^{-i+2}$, the on-line delay of the algorithm becomes 5.

V. Conclusion

We have obtained a very fast version of the CORDIC algorithm, which makes it possible to perform constant-time elementary iterations, independent from the length of the operands, with a constant scale factor. The main drawback of our method is the necessity of performing two conventional CORDIC iterations in parallel, which consumes more silicon area than classical methods. However, it leads to a fast and convenient on-line implementation, with a small delay.

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] J. R. Cavallaro and F. T. Luk, "CORDIC arithmetic for a SVD processor," in *Proc. 8th Symp. Comput. Arithmet.*, Como, Italy, May 1987.
[3] D. Cochran, "Algorithms and accuracy in the HP35," *Hewlett Packard J.*, pp. 10–11, June 1972.

[4] M Cosnard *et al.*, "The FELIN arithmetic coprocessor chip," in *Proc. 8th Symp. Comput. Arithmet.*, ARITH8, Como, Italy, May 1987, pp. 107–112.
[5] E. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC architectures for fast VLSI filtering and array processing," in *Proc. ICCD'84*, pp. 41.A.6.1–.A.6.4.
[6] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Trans. Comput.*, May 1984.
[7] M. D. Ercegovac and T. Lang, "On the fly conversion of redundant into conventional representations," *IEEE Trans. Comput.*, vol. C-36, no. 7, pp. 895–897, July 1987.
[8] ——, "Implementation of fast angle calculation and rotation using online CORDIC," in *Proc. ISCAS'88*, pp. 2703–2706.
[9] ——, "Redundant and on-line CORDIC: Application to matrix triangularization and SVD," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 725–740, June 1990.
[10] M. D. Ercegovac, "On line arithmetic: An overview," in *Proc. SPIE Conf. Real Time Signal Processing*, San-Diego, CA, 1984, pp. 667–680.
[11] A. Guyot, B. Hochet, C. Mauras, J. M. Muller, and Y. Robert, "SCALA: une cellule systolique programmable pour l'algèbre linéaire et le traitement du signal," in *Proc. 2nd Symp. C^3*, Angoulême, France, May, 1987 (in French).
[12] H. Lin and H. J. Sips, "On-line CORDIC algorithms," *IEEE Trans. Comput.*, vol. 39, no. 8, Aug. 1990.
[13] J. M. Muller, "Discrete basis and computation of elementary functions," *IEEE Trans. Comput.*, Sept. 1985.
[14] N. Takagi, T. Asada, and S. Yajima, "A hardware algorithm for computing sine and cosine using redundant binary representation," *Trans. IECE Japan*, vol. J69–D, no. 6, pp. 841–847, June 1986 (in Japanese). English translation is available in *Syst. and Comput. in Japan*, vol. 18, no. 8, pp. 1–9, Aug. 1987.
[15] ——, "Redundant CORDIC methods with a constant scale factor," *IEEE Trans. Comput.*, vol. 40, no. 9, pp. 989–995, Sept. 1991.
[16] J. Volder, "The CORDIC computing technique," *IRE Trans. Comput.*, Sept. 1959.
[17] J. Walther, "A unified algorithm for elementary functions," in *Joint Comput. Conf. Proc.*, vol. 38, 1971.

Jean Duprat was born in Vic-en-Bigorre, France, in 1949. "Agrégé" in Mathematics, he taught in high school from 1973 to 1987. He studied computer science in the universities of Paris and Grenoble, then he received the Ph.D. degree in Grenoble in 1988. His Ph.D. dissertation deals with the parallel implementation of Prolog.

He has been in the Computer Science Department of the Ecole Normale Supérieure de Lyon (Lip-Imag laboratory) since 1988. He teaches Computer Architecture. His research interests include computer architecture, VLSI, computer arithmetic, and fine grain parallelism.

Jean-Michel Muller (M'87) was born in Grenoble, France, in 1961. He received the Engineer degree in applied mathematics and computer science in 1983 and the Ph.D. degree in computer science in 1985, both from the Institut National Polytechnique de Grenoble, France.

In 1986, he joined the CNRS (French national center for scientific research). He has been posted from 1986 to 1989 to Tim3-Imag laboratory. Grenoble, and then to Lip-Imag laboratory, Lyon. He teaches computer arithmetic in the Institut National Polytechnique de Grenoble and the Ecole Normale Supérieure de Lyon. His research interests include computer arithmetic and computer architecture.

Dr. Muller served as General Chairman of the 10th Symposium on Computer Arithmetic.