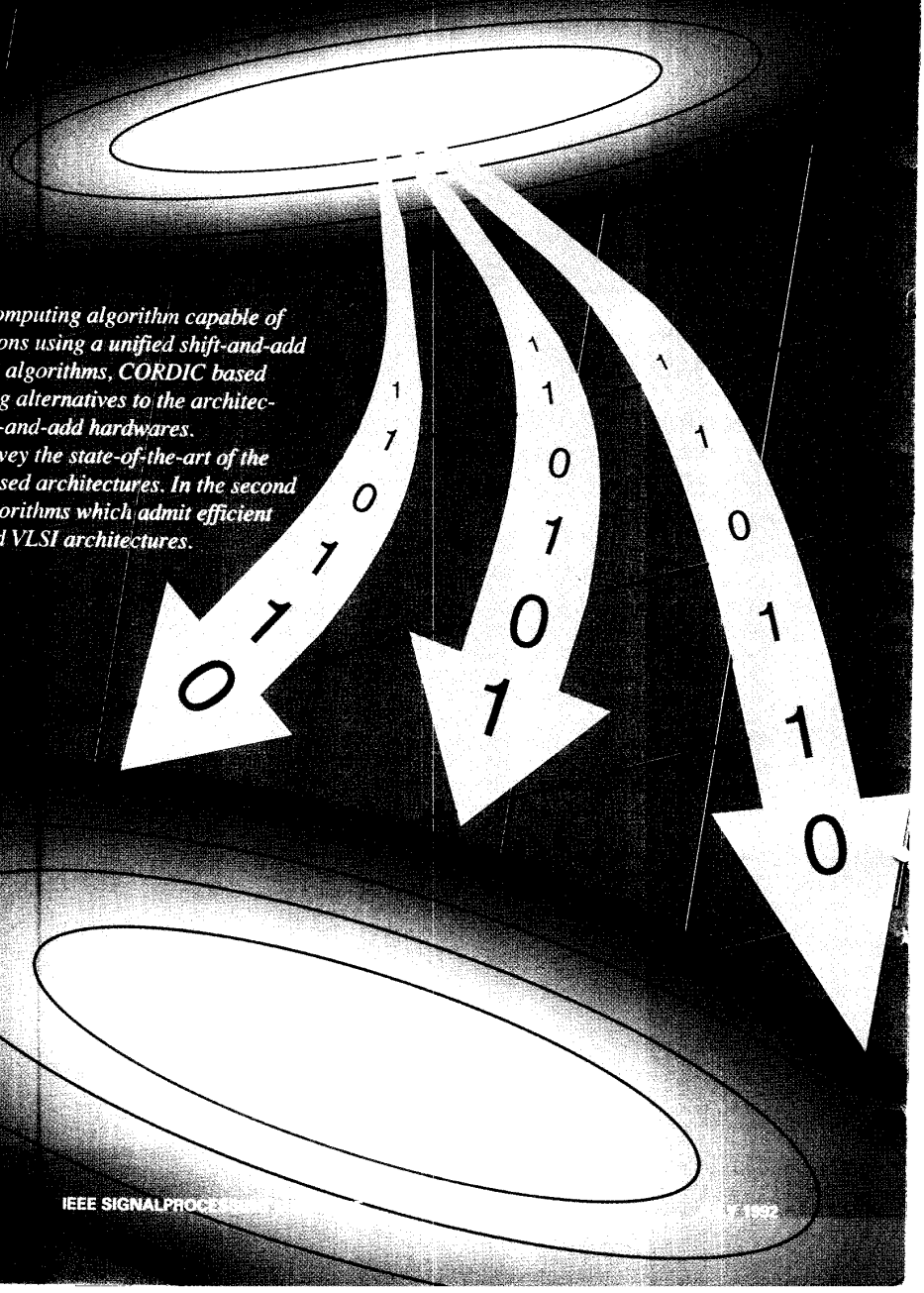
# CORDIC-Based VLSI Architectures for Digital Signal Processing

YU HEN HU



*CORDIC is an iterative arithmetic computing algorithm capable of evaluating various elementary functions using a unified shift-and-add approach. For a wide variety of DSP algorithms, CORDIC based VLSI architectures are very appealing alternatives to the architectures based on conventional multiply-and-add hardwares. In the first part of this article, we survey the state-of-the-art of the CORDIC algorithm and CORDIC based architectures. In the second part, we review a number of DSP algorithms which admit efficient implementation using CORDIC based VLSI architectures.*

1053-5888/92/$3.00©1992 IEEE

In the past decade, the unprecedented advances in VLSI technology have stimulated great interests in developing special purpose, parallel processor arrays to facilitate real time digital signal processing. Parallel computing systems such as systolic arrays [57] and wavefront arrays [59],[60] have been extensively studied. The basic arithmetic computation of these parallel VLSI arrays has often been implemented with a multiplication and accumulation (MAC) unit, because these operations arise frequently in DSP algorithms. The reduction in hardware cost also motivated the development of more sophisticated DSP algorithms to enhance the performance of modern digital signal processing systems. Many of these new algorithms require the evaluation of elementary functions, such as trigonometric, exponential, and logarithm functions, which cannot be evaluated efficiently with MAC based arithmetic units. Consequently, when DSP algorithms incorporate these elementary functions, it is not unusual to observe significant performance degradation.

On the other hand, an alternative arithmetic computing algorithm known as CORDIC (COordinate Rotation DIgital Computer) has received renewed attention, as it offers a unified iterative formulation to efficiently evaluate each of these elementary functions. Specifically, all the evaluation tasks in CORDIC are formulated as a *rotation* of a $2 \times 1$ vector in various coordinate systems. By varying a few simple parameters, the same CORDIC processor is capable of iteratively evaluating these elementary functions using the same hardware within the same amount of time. This regular, unified formulation makes the CORDIC based architecture very appealing for implementation with pipelined VLSI array processors.

In this context, many research efforts have been directed to the application of CORDIC based architectures for DSP applications [2], [3], [4], [18], [19], [21], [22], [23], [27], [41], [47], [52], [53], [63],[64] [70], [72], [74], [76], [77], [79], and [80]. The primary objective of this article is to provide a brief survey of these recent research efforts.

We will first review the state-of-the-art of the evolution of the CORDIC algorithm, and CORDIC processors. Then we will survey a number of typical DSP applications suitable for implementation with CORDIC based hardware.

## CORDIC ALGORITHM AND CORDIC BASED PROCESSOR ARRAY

CORDIC is an iterative arithmetic algorithm introduced by Volder [81] in 1956 and later refined by Walther [82] and many others [1], [4], [10], [14], [16], [17], [20], [18], [25], [30], [37], [42], [44], [45], [40], [75], [77]. The CORDIC algorithm has found a wide range of applications, including discrete transformations such as discrete Hartley transform [11], discrete cosine transform (DCT) [14], fast Fourier transform (FFT) [25], [26], Chirp Z transform (CZT) [44], solving eigenvalue and singular value problems [18], [31], [72], digital filters [21], [22], [80], Toeplitz system and linear system solvers [47], [48], [52], [74], [53], and Kalman filters [77].

### CORDIC Algorithm

The basic concept of the CORDIC computation is to decompose the desired rotation angle into the weighted sum of a set of predefined elementary rotation angles such that the rotation through each of them can be accomplished with simple shift-and-add operations. For this, we let the rotation angle, $\theta$, be represented as:

$$\theta = \sum_{i=0}^{n-1} \mu_i a_m(i) \tag{1}$$

where the $i^{-th}$ elementary rotation angle, $a_m(i)$ is defined by:

$$a_m(i) = \frac{1}{\sqrt{m}} \tan^{-1}[\sqrt{m}\,2^{-s(m,i)}] = \begin{cases} -2^{s(0,1)} & m \to 0 \\ \tan^{-1} 2^{-s(1,i)} & m = 1 \\ \tanh^{-1} 2^{-s(-1,i)} & m = -1 \end{cases} \tag{2}$$

In the above equations, $m = 1$, $-1$, and 0 corresponds to, respectively, the rotation operation in a circular coordinate system, a hyperbolic coordinate system, and a linear coordinate system. The *norm* of a vector $[\,x\ y\,]^t$ in these three coordinate systems are defined as $\sqrt{x^2 + my^2}$. The term $\{mu(i);\ 0 \le i \le n-1;\ \}$ is a sequence of $\pm 1$s which determines the rotation angle, and the modes of operations. The term $\{s(m,i)\ ;\ 0 \le i \le n-1\}$ is a non-decreasing integer *shift sequence* which is usually determined in advance.

With these definitions, the basic CORDIC algorithm can be described as follows:

*Initiation*: Given *x(0), y(0), z(0).*
*For i = 0 to n - 1, Do*
/* CORDIC iteration equation */

$$\begin{bmatrix} x(i+1) \\ y(i+1) \end{bmatrix} = \begin{bmatrix} 1 & -m\mu_i 2^{-s(m,i)} \\ \mu_i 2^{-s(m,i)} & 1 \end{bmatrix} \cdot \begin{bmatrix} x(i) \\ y(i) \end{bmatrix} \tag{3}$$

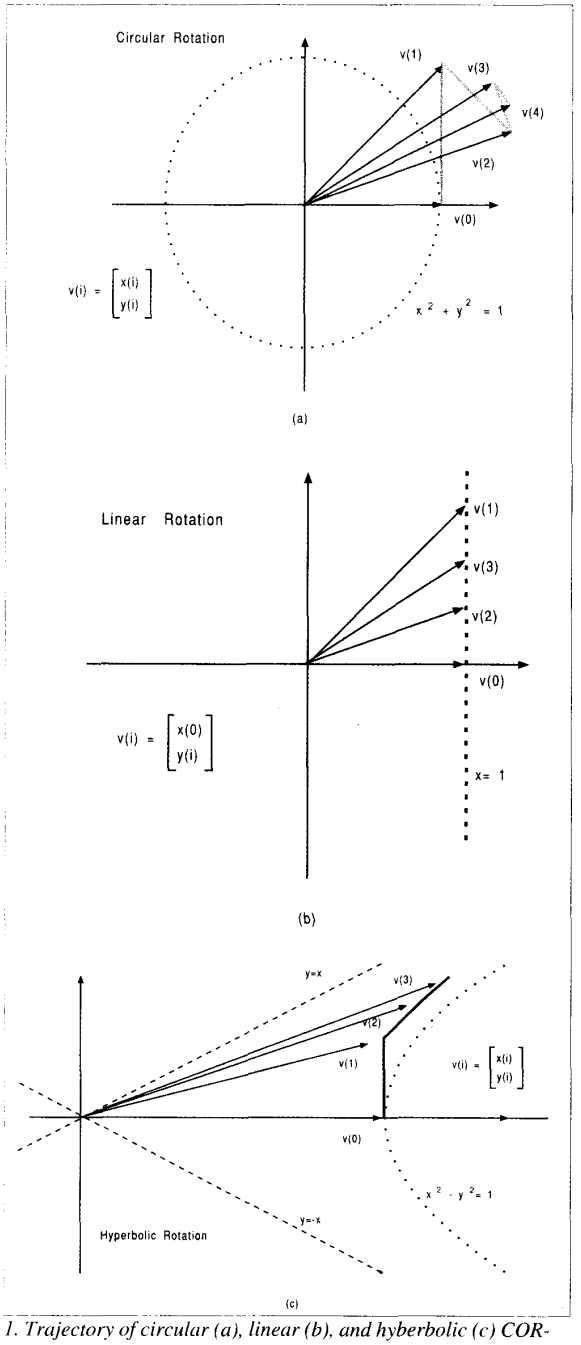
/* Angle updating equation */

$$z(i+1) = z(i) - \mu_i\, a_m(i) \tag{4}$$

*End i-loop*

/* Scaling Operation (required for $m = \pm 1$ only)*/

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \frac{1}{K_m(n)} \cdot \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} = \frac{1}{\prod_{i=0}^{n-1} \sqrt{1 + m\mu_i^2 2^{-2s(m,i)}}} \cdot \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \tag{5}$$




*1. Trajectory of circular (a), linear (b), and hyberbolic (c) CORDIC rotations.*

The purpose of the scaling operation (Eq. 5) is to make sure that after rotation, the final coordinate $[\, x_f \;\; y_f \,]^t$ has the same m-norm as the initial coordinate $[\, x(0) \;\; y(0) \,]^t$. That is,

$$(x_f)^2 + m(y_f)^2 = x^2(0) + my^2(0) \quad m = -1, \text{ or } 1$$

(For m = 0, $K_o(n) = 1$. No scaling operation is needed).

From Figure 1(a)-(c), it is clear that the rotation changes the norm of the vector. Hence this scaling operation is needed to move the end point of the final vector back to the desired trajectory (on the dotted line). A main research issue, to be discussed later, is to reduce the computation overhead due to the scaling operation.

*Modes of Operations*

The set of $\mu_i$ (=±1) in the CORDIC algorithm determine the rotation angle depending on the modes of operations. The CORDIC algorithm can be operated in either a *vector rotation mode*, or an *angle accumulation mode.*

In the vector rotation mode, which is also known as the vectoring mode [82], or forward rotation mode [44], the desired rotation angle θ is given. The objective is to compute the final coordinate $[x_f \;\; y_f]^t$. Usually, we set $z(0) = \theta$ at the beginning.

After $n$ iterations, the total angle rotated is:

$$z(0) - z(n) = \theta - z(n) = \sum_{i=0}^{n-1} \mu_i a_m(i) \tag{6}$$

Clearly, we should choose $\mu_i = \text{sign of } z(i)$ in Eq. 4 such that after the $i^{th}$ iteration, $|\, z(i+1) \,| < |\, z(i) \,|$. Eventually, we want to make $|z(n)| \rightarrow 0$.

For many DSP problems, θ is known in advance. Often, the same θ will be used over and over. In these situations, one may choose to evaluate Eq. 4 in advance (off line), and store the corresponding set of $\mu_i$, instead of θ, in the memory. A particular advantage is that there will be no need to implement the angle updating formula (Eq. 4), resulting in a one-third cost saving of hardware.

In the angle accumulation mode, which is also known as the *rotation* mode [82], *Y-reduction* mode [10], or the *backward rotation* mode [47], the desired rotation angle, θ, is not given. The objective is to rotate the given initial vector $[x(0) \; y(0)]^t$ back to the x-axis so that the angle between them can be accrued. For this purpose, we let $z(0) = 0$, and select $\mu_i = \text{sign of } x(i) \cdot y(i)$. (In the circular rotation case, an alternative criterion to select $\mu_i = -\text{ sign of } y(i)$ will guarantee $x_f > 0$ regardless the sign of $x(0)$. In summary, the selection criteria for $\mu_i$ are:

$$\mu_i = \begin{cases} \text{sign of } z(i) & \textit{vector rotation mode} \\ -\text{ sign of } x(i) \cdot y(i) & \textit{angle accumulation mode} \end{cases} \tag{7}$$

Note that the set of $\{\mu_i \,;\, i = 0 \textit{ to } n-1\}$ can be regarded as an alternative representation of the rotation angle θ. Hence, even in the angle accumulation mode, often it is not necessary to explicitly compute θ using Eq. 5.

## Shift Sequence

The shift sequence $\{s(m,i);\, 0 \le i \le n-1\}$ determines the convergence of the CORDIC iteration, as well as the magnitude of the scaling factor $K_m(n)$. Since there are only $n$ finite

elementary rotation angles $\{a_m(i); i = 0, n-1\}$, it is impossible to represent arbitrary rotation angle $\theta$ without error. Let us define an *angle approximation* error as:

$$\delta = \theta - \sum_{i=0}^{n-1} \mu(i) a_m(i) \tag{8}$$

In the vector rotation mode, since $z(0) = \theta$, from Eq. (6), we have $\delta = z(n)$. In the angle accumulation mode, one can easily show that:

$$\delta = \begin{cases} \tan^{-1} y(n)/x(n) & m = +1 \\ y(n)/x(0) & m \to 0 \\ \tanh^{-1} y(n)/x(n) & m = -1 \end{cases}$$

The effects of the angle approximation error and the rounding error on the accuracy of the CORDIC computation have been analyzed [43]. In general, it is desired that for any given rotation angle, $\theta$:

$$|\delta| \le a_m(n-1) \tag{9}$$

This condition imposes two constraints on $\theta$. First, the corresponding set of elementary rotation angles must satisfy:

$$a_m(i) \le \sum_{j=i+1}^{n-1} a_m(j) + a_m(n-1) \tag{10}$$

Next, $\theta$ must lie within the domain of convergence. That is,

$$|\theta| \le \sum_{i=0}^{n-1} a_m(i) + a_m(n-1) \equiv A_{max,m} \tag{11}$$

Walther [82] has proposed a set of shift sequence for each of the three coordinate systems. In particular, for $m = 0$ or 1, $s(m,i) = i$, $0 \le i \le n-1$. For $m = -1$, $s(-1,i) = 1, 2, 3, 4, 4, 5, \ldots, 12, 13, 13, 14, \ldots,$

The purpose of repeating elements {4, 13, 40, ...} in the hyperbolic coordinate is to assure that Eq. 10 is satisfied. Along this line of research, X. Hu et al. [40] have proposed a method to expand the range of $A_{max,m}$.

*Scaling Operation*

The multiplication of $1/K_m(n)$ in the scaling operation (Eq. 5) imposes significant computation overhead on the CORDIC algorithm. This is because to perform a multiplication, $n$ shift-and-add operations will need to be performed. Fortunately, if we constrain $|\mu_i| = 1$, and assume $\{s(m,i)\}$ is given, $K_m(n)$ can be computed in advance. As such, various techniques have been proposed to reduce computation overhead.

A first approach is to convert $1/K_m(n)$ into a canonical sign-digit representation [51]:

$$\frac{1}{K_m(n)} = \sum_{p=1}^{P} \kappa_p 2^{-i_p} \tag{12}$$

where $\kappa_p = \pm 1$, and $i_p$ are positive integers. Multiplication with $1/K_m(n)$, then will take $P-1$ shift-and-add operations. Using the canonical multiplier recoding method [51], the value of $P$ can be reduced to $b/2$, where $b$ is the number of bits in the internal registers. On the average, $P \approx b/3$.

A second approach is to represent $1/K_m(n)$ as a product of $Q$ factors such that:

$$\frac{1}{K_m(n)} = \prod_{q=1}^{Q} (1 + \kappa_q 2^{-i_q}) + \varepsilon_q \tag{13}$$

where $\kappa_q = \pm 1$, $i_q$ are positive integers, and $|\varepsilon_q| < 2^{-b}$. Given $1/K_m(n)$, similar to the first method, the design goal is to minimize $Q$. It has been observed [4], [37] that if some of the elementary rotation angles are repeated, the resulting scaling factor, $K_m(n)$ can be approximated by a power of 2. Thus the scaling operation can be accomplished with several additional (repeated) CORDIC iterations with a simple shift operation at the end. Delosme later suggested use of a simulated annealing technique to identify the best set of elementary angles to repeat [18]. Deprettere et al [22], [20] proposed a variant of that method in which some product terms have the form $(1 + \kappa_q 2^{-i_k} + \gamma_k 2^{-j_k})$ where $\gamma_k = \pm 1$ for a three-term product, and $\gamma_k = 0$ for a regular two-term product.

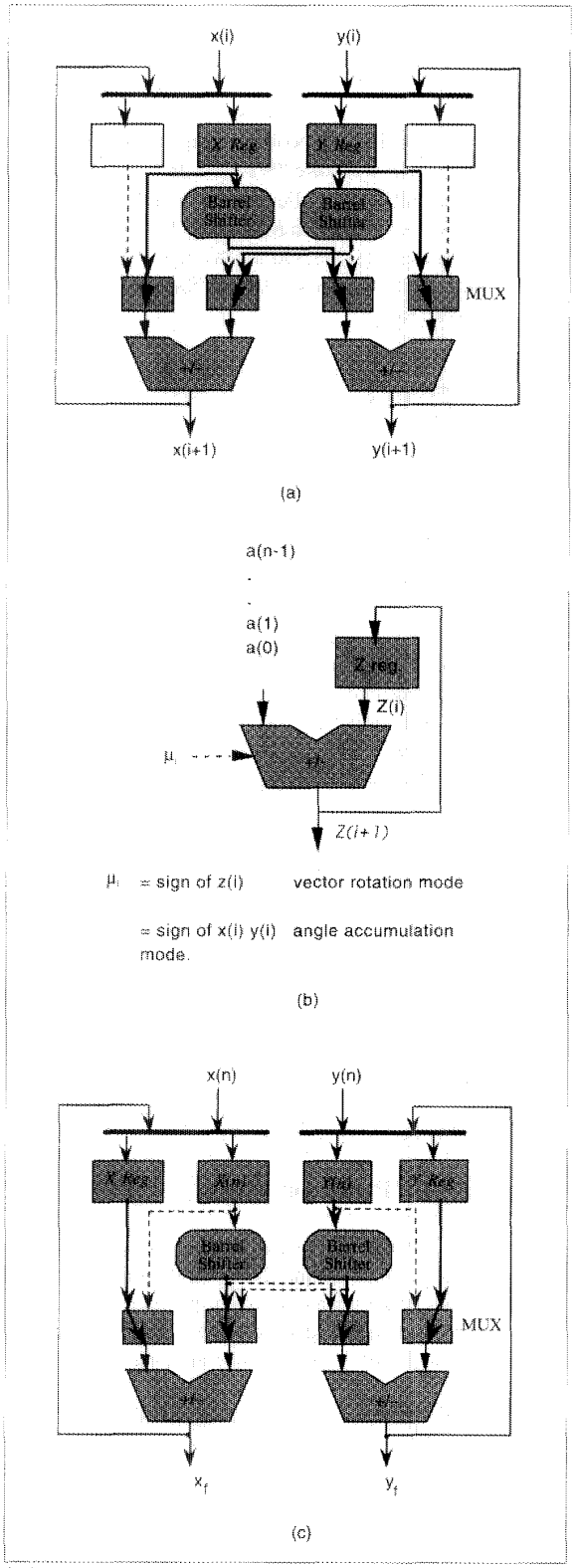
In singular value decomposition and eigenvalue decomposition problems, an important operation is to diagonalize a $2 \times 2$ matrix with simultaneous pre- and post- rotation operations. Observing that the scaling factor is $K_m^2(n)$ in this case, Cavallaro and Luk [10] have proposed a simple scheme (discussed later) to perform exact scaling. This method was recently studied and enhanced by Delosme [18], [19], and Yang and Bohme [84].

## ARCHITECTURAL DESIGN

### Basic CORDIC Processor

A basic CORDIC processor should contain function modules which realize (a) the CORDIC iterations specified in Eq. 3; (b) the angle update iteration specified in Eq. 4; and (c) the scaling operation specified in Eq. 5. The functional block diagrams for each of these operations are straightforward (Fig. 2).

The modules in Fig. 2a support a single CORDIC iteration. It contains dual barrel shifters and dual adders to facilitate the updating of both $x(i)$ and $y(i)$ simultaneously. The number of bits to be shifted is controlled by the shift sequence $\{s(m,i)\}$. The shift sequence can be stored on chip using ROM (read-only memory) or RAM (random-access memory), or generated on chip with a simple counter and additional control devices. The add/subtract operation is determined by the




*2. Basic processor for CORDIC iterations (a), angle update (b), and scaling iterations (c).*

sequence $\{\mu_i\}$, which, according to Eq. 7, is determined by either the sign of $z(i)$ or $-x(i)\cdot y(i)$.

The angle updating module (Fig. 2b) performs simple addition operations. In many DSP applications, there is no need to compute the rotation angle explicitly. In these cases, the angle updating module can be eliminated completely.

The scaling module (Fig. 2c) can share the same processing unit with the CORDIC iteration module. By multiplexing the data paths, this scaling module is able to perform the following two types of scaling iterations. Given that $x'(0) = x(n)$ , $y'(0) = y(n)$ ,

$$\begin{cases} x'(i+1) = x'(i) + 2^{-i_p} x(n) \\ y'(i+1) = y'(i) + 2^{-i_p} y(n) \end{cases} \quad \textit{Type I}$$

$$\begin{cases} x'(i+1) = x'(i) + 2^{-i_q} x'(i) \\ y'(i+1) = y'(i) + 2^{-i_q} y'(i) \end{cases} \quad \textit{Type II}$$

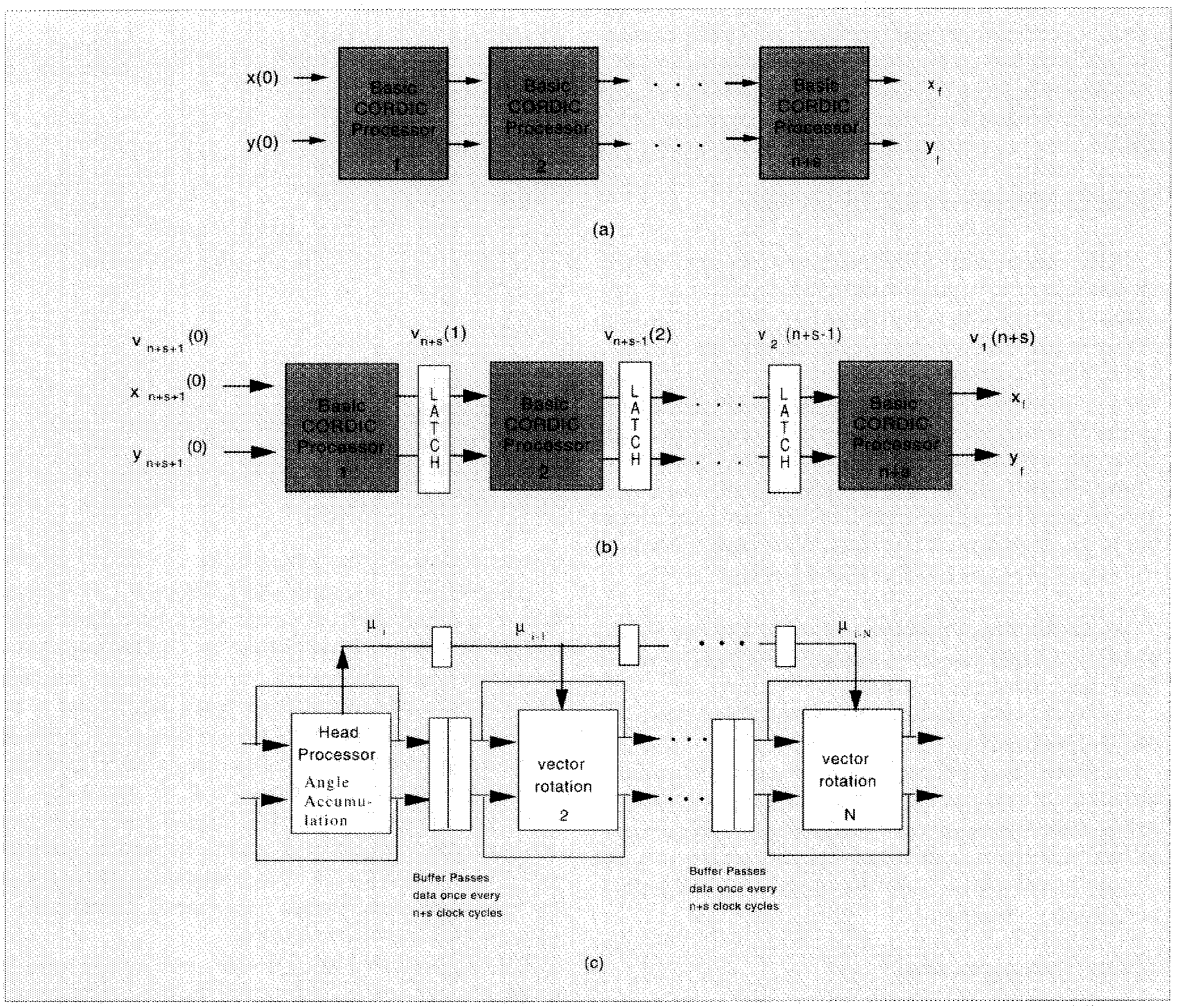
Type I scaling iterations will be invoked when the scaling factor is represented as described in Eq. (12). Type II scaling iterations will be used when the scaling factor is represented as described in Eq. 13.

Previously, Haviland and Tuszynski [37] reported a single chip CORDIC arithmetic unit. Ahmed et al. [2], [4] reported a speech analysis chip based on a micro-programmed controlled CORDIC chip set. Sung and Hu [77] has reported the design of a 32-bit CORDIC data path chip as a building block for implementing square-root Kalman filters. Deprettere et al. [20], [21], [28], [8] reported a CMOS implementation of a floating point CORDIC processor. Cavallaro [10] reported another version. Rader [69] at MIT Lincoln Labs proposed a large array of CORDIC processors for least square array processing. IMEC has also developed a CORDIC processor design using the CATHEDRAL III DSP silicon compiler [65]. A bit serial on-line arithmetic implementation of the CORDIC processor has also been proposed by Ercegovac and Lang [30], [31]. The most recent implementation was reported by Timmermann et al. [78].

## Parallel and Pipelined Arrays

In Fig. 3a, we illustrate an architecture of a *parallel* CORDIC processor. Each of the $n$ stages of the CORDIC iterations and $s$ (= $P$ in Eq. 12 or $Q$ in Eq. 13) stages of scaling operations is realized with a separate, dedicated basic CORDIC processor as specified in Figs. 2a and 2c. By cascading these n+s basic CORDIC processors, we will be able to perform the entire CORDIC rotation operations, including all the CORDIC iterations and all the scaling operations in one clock cycle.

Let $t_o$ be the time needed for performing a single CORDIC iteration or a single scaling iteration. Then the total computation delay will be $(n+s)t_o$. Or, equivalently, the signal processing throughput rate will be bounded by $1/[(n+s)t_o]$.




*3. Parallel implementation of a CORDIC processor array (a). Pipelined implementation of the array (b). Subscript indicates different data are processed at different CORDIC iterations and scaling iterations. A doubly pipelined array (c). The $\mu_i$ will be propagated every clock cycle. The input/output labels for each processor are omitted, as they depend on the specific problem at hand.*

To increase the throughput, one may choose to insert a latch (a temporary storage digital circuit) between successive stages (Fig. 3b) of this parallel CORDIC processor, and convert it into a *pipelined* CORDIC processor array. The latch is able to store the intermediate result after a CORDIC iteration or a scaling operation. Hence the computation of a CORDIC processor is isolated from the computation at the adjacent CORDIC processors. As a result, these latches can be clocked at a period of $t_o$ time units which is the computation delay between adjacent stages. In other words, the throughput rate is now increased $(n+s)$-fold to $1/t_o$. However, we note that each individual data will now take $n+s$ clock cycles to complete. Hence the latency is $(n+s)t_o$ *time units.*

A third possible configuration (Fig. 3c) will facilitate two levels of pipelining (*double pipelining*). In this configuration, a CORDIC array will consist of a *head CORDIC processor* which is operated the angle accumulation mode, and a number of vector processors operating in the vector rotation mode. The rotation angle will be computed by the head processor, and then propagated to the vector processors in a pipelined fashion. Upon completion of the rotation, the head processor as well as the vector processors will propagate the result to the neighboring processors, also in a pipelined manner. We note from Eqs. 3 and 7 a new $\mu_i$ can be computed every $t_o$ time units. Hence, the rotation angle can be piped to the vector processors at a rate of $1/t_o$. This constitutes the lower-level pipelining. On the other hand, the computation results of the CORDIC rotations will not be ready after $(n+s)t_o$ time units. Hence, the higher-level pipelining is performed at a rate of $1/(n+s)t_o$.

Doubly piped operation was first proposed by Deprettere et al. [22], [28] for implementing CORDIC-based digital

filters. It enables a fine grain pipelining, which helps reduces the pipelining latency. For Toeplitz system and linear system solvers, it also facilitates saving total computation time by almost 50 percent [41]. It was called *implicit rotation* when applied to the eigenvalue decomposition problem [18].

## IMPLEMENTING ALGORITHMS

In this section, we will illustrate how to utilize a CORDIC processor array to implement digital signal processing algorithms. Our approach is to reformulate existing DSP algorithms so that they are suitable for implementation with a CORDIC processor array performing circular, or hyperbolic rotation operations. The linear rotation operation will not be emphasized because they can also be implemented, more efficiently, with MAC based processors. There usually exists a more efficient scheme for implementing a particular function. However, the utility of the CORDIC based architecture lies in its *generality* and *flexibility*. With these in mind, we will survey three categories of DSP algorithm:

(a) *Linear transformations*, including discrete Fourier transform, Chirp-Z transform, discrete Hartley transform, and fast Fourier transform;

(b) *Digital filters*, including orthogonal digital filters, and adaptive lattice filters;

(c) *Matrix based digital signal processing algorithms*, including QR factorization, with applications to Kalman filtering, linear system solvers, Toeplitz and covariance system solvers, with applications to least square deconvolution, and eigenvalue and singular value decompositions with applications to array processing.

### Linear Transformations

*Discrete Fourier Transform (DFT)*

Given a complex-valued discrete sequence $\{X(n); 0 \le n \le N-1\}$, its discrete Fourier transform $\{Y(k); 0 \le k \le N-1\}$ is defined as:

$$Y(k) = X(0)e^{\frac{-j2\pi 0}{N}} + X(1)e^{\frac{-j2\pi k}{N}} + \ldots + X(N-1)e^{\frac{-j2\pi(N-1)k}{N}} \tag{14}$$

for $0 \le k \le N-1$. To avoid notational confusion, in this article we use $x(i)$, $y(i)$ to denote the rotation coordinates in the CORDIC algorithm, and $X(n)$, $Y(n)$ to denote the time and frequency domain sequences in the DSP algorithm. We note, however, that in most DSP literatures, the time domain sequences are denoted by lower case letters, e.g., $x(n)$. Eq. 14 can be reformulated in a recurrent algorithm format to facilitate implementation with a CORDIC processor array. The algorithm is as follows:

*Initiation*: Y(0,k) = 0 for $0 \le k \le N-1$.
For k = 0, 1, ..., N-1 Do
For m = 0, 1, ..., N-1 Do

$$\begin{bmatrix} Y_r(m+1,k) \\ Y_i(m+1,k) \end{bmatrix} = K_1(n) \cdot \begin{bmatrix} \cos\frac{2\pi mk}{N} & -\sin\frac{2\pi mk}{N} \\ \sin\frac{2\pi mk}{N} & \cos\frac{2\pi mk}{N} \end{bmatrix} \cdot \begin{bmatrix} x_r(m) \\ x_i(m) \end{bmatrix} + \begin{bmatrix} Y_r(m,k) \\ Y_i(m,k) \end{bmatrix}$$

*End m-loop*

$Y(k) = \dfrac{Y(N,k)}{K_1(n)}$ /* *Scaling operation* */

*End k-loop*

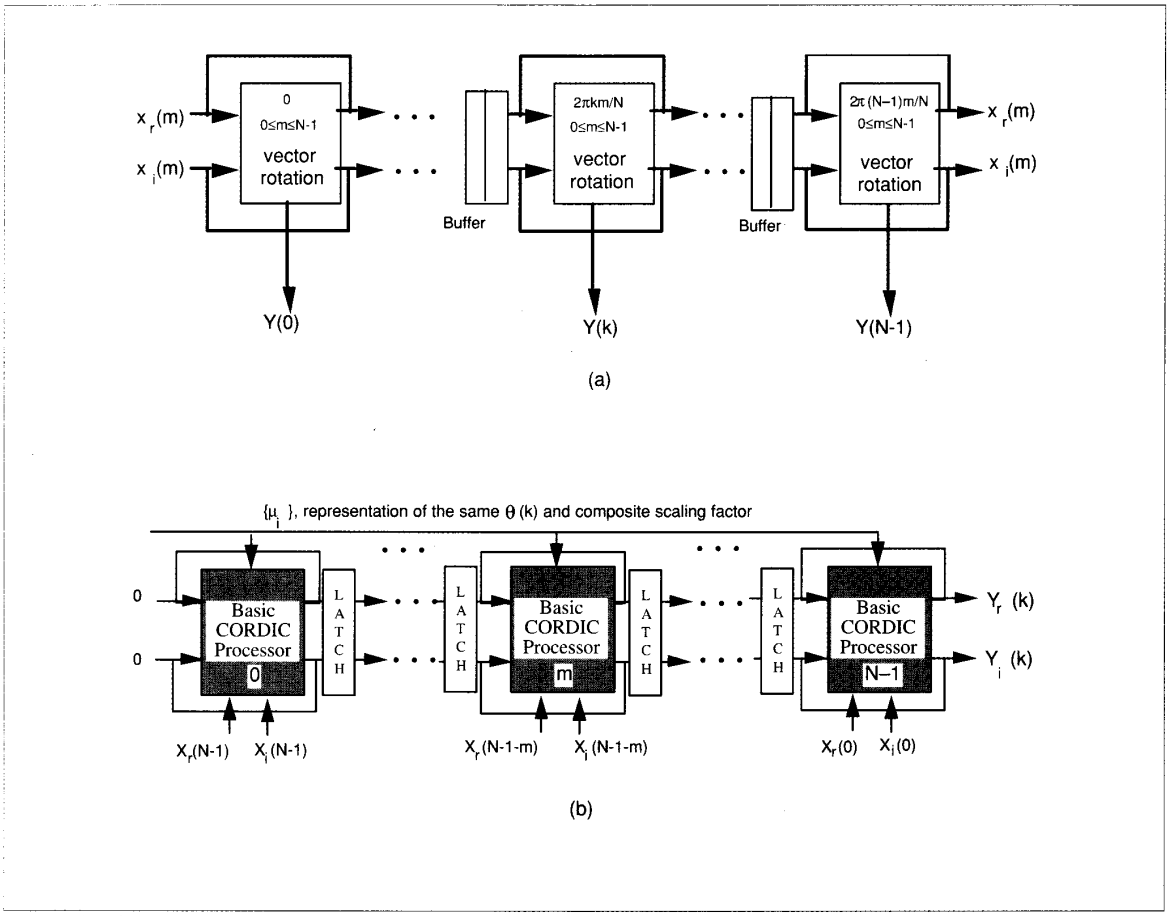
(Note that $K_1(n)$ is defined in Eq. 5.)

In Figure 4, the sequence $\{X(n); 0 \le n \le N-1\}$ is taken from the left end of this processor array. Each $X(n)$, which contains two real numbers $x_r(n)$ and $x_i(n)$, will be propagated from the current processor to the next nearest neighbor processor in a pipelined manner. Since $X(n)$ is not to be modified during propagation, they can be piped at a rate of $t_o$ time unit (1 clock cycle) per stage. Each of the N CORDIC processors in this processor array will be responsible for evaluating a particular $Y(k)$. This implies that N different rotation angles $\{2\pi mk/N;\ 0 \le m \le N-1\}$ must be stored in each processor. Since each of each rotation angle requires $n$ bits to store, the total storage requirement, including the $s$ scaling iterations, will be $nN+s$ bits.

As Despain pointed out [25], there is no need to perform scaling operation in the *DFT* algorithm. This is emphasized by placing $K_1(n)$ in front of the rotation matrix. In doing so, the output will be $K_1(n)Y(k)$, which is a scaled version of the desired output, $Y(k)$. Hence, there is no need to perform the scaling operation, which takes s iterations, for each of the N CORDIC rotation operation in the m-loop. Instead, only one final scaling operation is needed to compute $Y(k)$ at the end of the m-loop as listed in the algorithm. With this scheme, the average scaling overhead is reduced from s shift-and-add scaling iterations per CORDIC rotation down to s/N shift-and-add scaling iterations per CORDIC rotation. Consequently, the computation delay for each $Y(k)$ will be $(nN+s)t_o$ time units; and the total computing time will be $(nN+s+N-1)t_o$ time units. However, since the scaling is done at the end, the dynamic range of $Y(m,k)$ will be increased by a factor of $K_1(n)$, whose magnitude is usually less than 2. To avoid overflow and loss of accuracy, one extra bit is needed to store the intermediate result, $Y(m,k)$.

*Discrete Hartley Transform (DHT)*

The Discrete Hartley transform is proposed by Bracewell [6] as an alternative to discrete Fourier transformation. It has the following formulation:




*4. A pipelined CORDIC DFT array (a). X(m) will be propagated from processor i to processor i+1 every $nt_o$ time units. Scaling is done only once in each processor after N CORDIC rotations are all completed. Each processor will store the same set of N rotation angles in the form of $\{\mu_i\}$. Which angle to use in the mth iteration depends on k. Pipelined CZT array (b). The rotation angle and composite scaling factor will be fed into each basic CORDIC processor, one by one. Each processor will cycle through n+s cycles before feeding its result to the next processor. The input data re applied to each processor in parallel. The output is available at processor N-1.*

$$Y(k) = \sum_{n=0}^{N-1} X(n)\left[\cos(\frac{2\pi kn}{N}) + \sin(\frac{2\pi kn}{N})\right] \qquad (15)$$

where $\{X(n)\}$ and $\{Y(k)\}$ are both real sequence. To realize DHT in a CORDIC based processor array, we set the initial coordinates of the CORDIC algorithm to (X(n), X(n)). That is, $x(0) = y(0) = X(n)$, and we rotate through an angle of $2\pi^{kn}/_{N}$. Each term in the above summation will appear as $y_f$. Again, similar to the case of the *DFT*, only one scaling operation is required for every Y(k). The recurrent formulation of the *DHT* is as follows:

Initiation: Y(0,k) = 0 for $0 \le k \le N-1$.
*For k = 0, 1, ..., N-1, Do*
For m = 0, 1, ..., N-1, Do

$$\begin{bmatrix} Y(m+1,k) \\ * \end{bmatrix} = \begin{bmatrix} Y(m,k) \\ * \end{bmatrix} + K_1(n) \begin{bmatrix} \cos(\frac{2\pi km}{N}) & \sin(\frac{2\pi km}{N}) \\ -\sin(\frac{2\pi km}{N}) & \cos(\frac{2\pi km}{N}) \end{bmatrix} \begin{bmatrix} X(m) \\ X(m) \end{bmatrix}$$

*End m-loop;*

$$Y(k) = \frac{Y(N,k)}{K_1(n)} \qquad \text{/* scaling operation */}$$

*End k-loop*

Note that the *DFT* array presented in Fig. 4 can also be applied to the *DHT* with $x(0)$ and $y(0)$ tied together. A systolic array implementation of the DHT based on the CORDIC processor has been reported recently [11].

*Chirped Z-Transform (CZT)*

The Chirped Z transform evaluates the Z-transform of a discrete, complex valued sequence $\{X(n)\}$ at the points $z_k = AW^{-k}$ for $k = 0, 1, \ldots, K\text{-}1$, where $W = W_o e^{-j\varphi_o}$, and $A = A_o e^{j\theta_o}$ with $W_o$, $A_o$, $\varphi_o$, and $\theta_o$ being real numbers. Hence,

$$Y(z_k) = \sum_{n=0}^{N-1} X(n)\, z_k^{-n} = \sum_{n=0}^{N-1} X(n)\, A^{-n}\, W^{kn}$$

Traditionally, CZT can be evaluated with fast Fourier transformation using $O(N+K-1) log(N+K-1)$ operations [66]. However, when $K << N$, direct evaluation of $Y(z_k)$ would be more efficient.

A CORDIC processor array of the *CZT* was proposed [44] which is based on the following recurrence algorithm formulation:

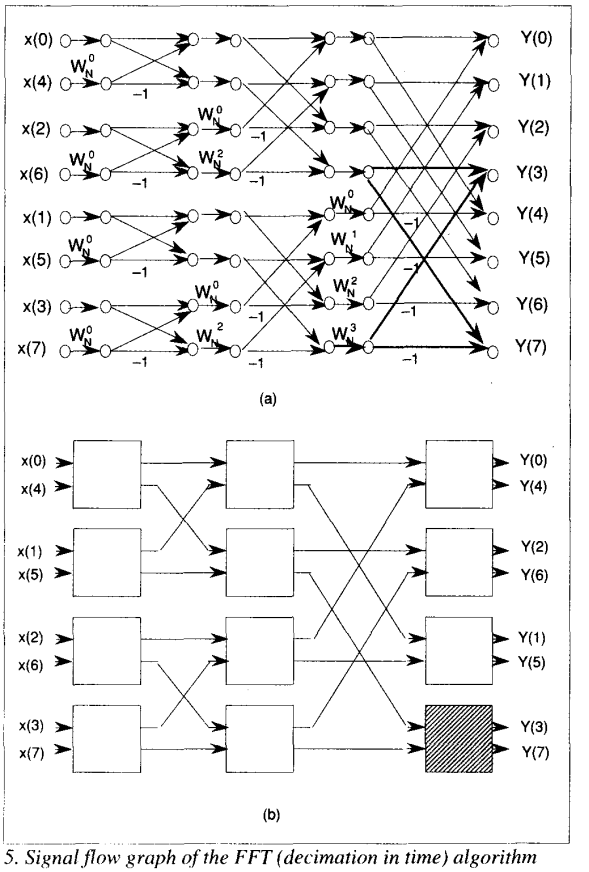
*Initiation:* $Y(0,k) = 0$ for $0 \le k \le K-1$.
*For* $k = 0, 1, \ldots K\text{-}1$ *Do*
*For* $m = 0, 1, \ldots, N\text{-}1$ Do

$$\begin{bmatrix} Y_r(m+1,k) \\ Y_i(m+1,k) \end{bmatrix} = A_o^{-1} W_o^k \begin{bmatrix} \cos(\theta_{o+k}\varphi_o) & -\sin(\theta_{o+k}\varphi_o) \\ \sin(\theta_{o+k}\varphi_o) & \cos(\theta_{o+k}\varphi_o) \end{bmatrix} \begin{bmatrix} Y_r(m,k) \\ Y_i(m,k) \end{bmatrix} + \begin{bmatrix} X_r(N-1-m) \\ X_i(N-1-m) \end{bmatrix}$$

*End m-loop*
End k-loop
Output: $Y(k) = Y(N,k)$

In each iteration of the m-loop, each $Y(m,k)$ will be multiplied with a scaling constant, $A_o^{-1} W_o^k$. This scaling constant can be combined with the scaling operation in the CORDIC algorithm. That is, instead of multiplying $1/K_1(n)$, and then followed by $A_o^{-1} W_o^k$, we will multiply a composite scaling factor, $A_o^{-1} W_o^k / K_1(n)$. A side effect, however, is that when CZT is realized with a CORDIC processor array, different $Y(k)$ may need different numbers of CORDIC iterations for scaling during each m-loop. This will cause scheduling difficulties in a pipelined implementation. On the other hand, it also presents a special opportunity to further reduce the number of CORDIC iterations using a *forward angle recoding* (FAR) method [44], [46].

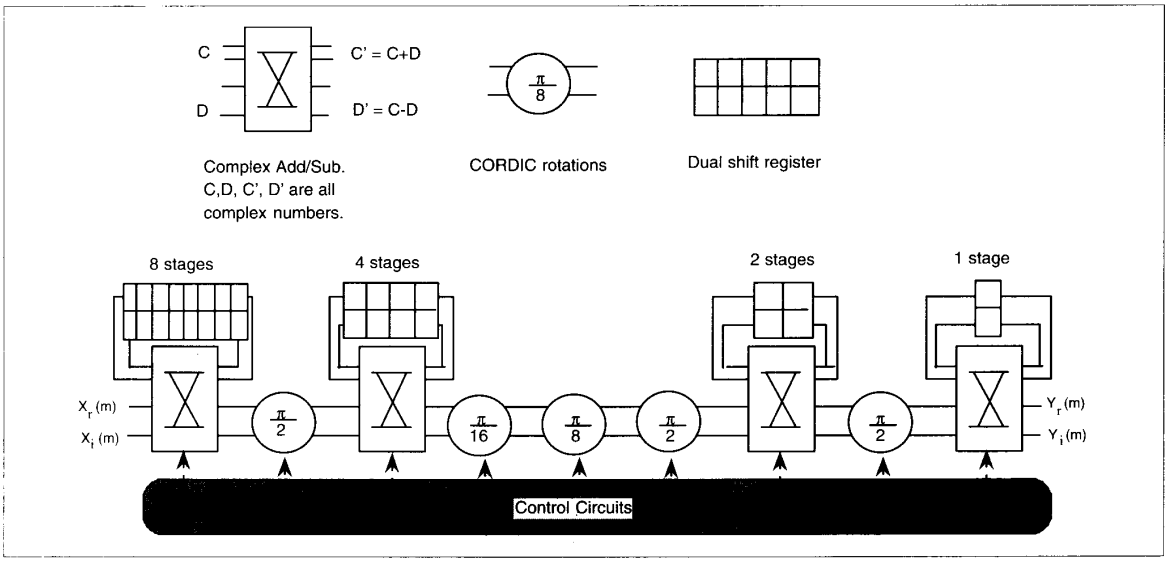
Briefly speaking, in the FAR method, the internal representation of the rotation angle $\{\mu_i\}$ can take integer values other than $\pm 1$. Whenever $\mu_i = 0$, the corresponding CORDIC iteration can be skipped. Hence, under the constraint of convergence condition (Eq. 9), the objective of FAR is to minimize $\sum_{i=0}^{n-1} |\mu_i|$. As such, the total number of CORDIC iterations can be minimized. Of course, according to Eq. 5, setting certain $\mu_i = 0$ will cause the scaling factor, $K_1(n)$, to be dependent on the rotation angle. Specifically, using FAR, each rotation angle, $\theta$, will have a different scaling factor. However, the radial scaling factor, $A_o^{-1} W_o^k$ is dependent on the rotation angle (indexed by $k$) anyway. Hence, a different $K_1(n)$ for each different index, $k$, will pose no extra computation or storage overhead. The architecture of a pipelined CORDIC processor array with forward angle recoding is illustrated in Fig. 4b.




*5. Signal flow graph of the FFT (decimation in time) algorithm (a). CORDIC implementation (b) of the DIT FFT algorithm in (a). Each box is a CORDIC processor. Thick arcs in (a) are realized by the shaded box in (b).*

*Fast Fourier Transformation* (FFT)

The fast Fourier transformation is perhaps the most frequently used DSP computing algorithm in modern digital signal processing systems. Let N be the total number of points in the DFT. If $N = N_1 \cdot N_2$, then, we may let index




*6. Pipelined CORDIC FFT array [26].*

$n = n_1 + N_1 n_2$, with $0 \le n_1 <= N_1 - 1$, and $0 \le n_2 \le N_2 - 1$. Note that

$$[e^{-j2\pi/N}]^{nk} \equiv W_N^{nk} = W_N^{N_1 n_2 k} \cdot W_N^{n_1 k} = W_{N_2}^{n_2 k} \cdot W_N^{n_1 k}.$$

Therefore, for $0 \le k \le N-1$,

$$Y(k) = \sum_{n=0}^{N-1} X(n) W_N^{nk} = \sum_{n_1=0}^{N_1-1} W_N^{n_1 k} \sum_{n_2=0}^{N_2-1} X(N_1 n_2 + n_1) \cdot W_{N_2}^{n_2 k} \tag{17}$$

If $N_1 = 2$, and $N_2 = N/2$, the above equation becomes:

$$Y(k) = \sum_{n_2=0}^{N/2-1} X(2n_2) \cdot W_{N/2}^{n_2 k} + W_N^k \sum_{n_2=0}^{N/2-1} X(2n_2 + 1) \cdot W_{N/2}^{n_2 k} \tag{18}$$

In other words, the original N-point DFT problem now becomes two N/2 point DFT sub-problems for each Y(k). Recursively applying this procedure will lead to the popular radix-2 FFT algorithm.

The basic operation in a FFT algorithm is called a *Butterfly* operation, which may have several different formats. For example, consider the decimation-in-time Butterfly operation:

$$\begin{cases} C' = C + D \cdot exp(-j\dfrac{2\pi k}{N}) \\ D' = C - D\ exp(-j\dfrac{2\pi k}{N}) \end{cases}$$

and the decimation-in-frequency Butterfly operation:

$$\begin{cases} C' = C + D \\ D' = (C - D)\ exp(-j\dfrac{2\pi k}{N}) \end{cases}$$

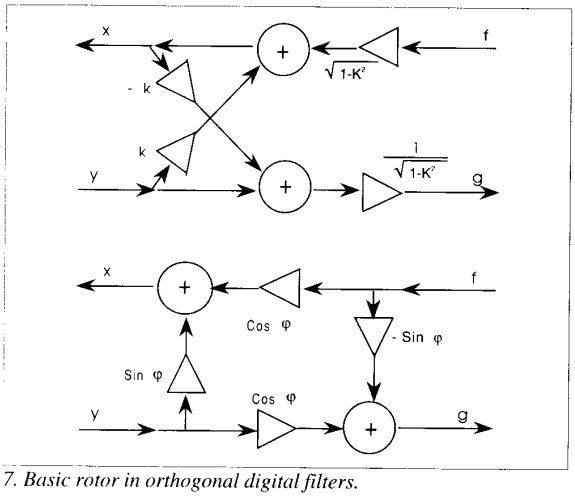
Note that $C$, $D$, $C'$ and $D'$ are all complex numbers. The above equations require a complex number multiplication (with a uni-modulus complex number), and two complex number additions. The complex number multiplication can be done with $n+s$ CORDIC iterations and scaling iterations. The two complex additions will need just one more clock cycle.

For small N, a FFT computation can be realized directly with a network of CORDIC processors (Fig. 5). A signal flow graph of an 8-point decimation-in-time FFT [66] is shown in Fig. 5a. In Figure 5b, each Butterfly operation is replaced by a rectangular box representing a CORDIC rotation followed by a complex addition or subtraction. To see how the structure in Figure 5b is obtained from Figure 5a, we highlighted the corresponding Butterfly operation in the last stage in Figure 5a, and the CORDIC processor in the shaded box in Fig. 5b.

For large N, the complicated global interconnection requirements unfortunately make it difficult to implement efficiently with a pipelined VLSI array, without severe performance penalty [60]. Despain [26] has reported an implementation of a 16-point FFT building block using a linearly array of specialized hardware to minimize the number of add/subtract stages. A block diagram of that processor is shown in Fig. 6. This processor array is able to achieve a pipelining period of 1. Also, the number of delay elements vary from stage to stage. It has $N/2$ delays in the first stage, $N/4$ in the stage, and so on.

## Digital Filtering

Most of the existing digital filtering structures are implemented with multiply-and-add operations. However, the




*7. Basic rotor in orthogonal digital filters.*

CORDIC structure is useful in implementing a large class of digital filtering functions with some unique advantages. In particular, two types of digital filtering structures will be discussed in this section: the *orthogonal digital filter* (ODF), based on circular rotations; and the *lattice filter* (LF), based on both hyperbolic and circular rotations. In fact, the basic building blocks of these two types of filter structures are intimatedly related.

*Orthogonal Digital Filter*

An orthogonal digital filter is a digital filter of which the overall numerical computation at each sample instance can be described by an orthogonal or unitary matrix. There are a number of advantages of an ODF [27] compared with other filter structures. First, it has low sensitivity both in the passband and the stopband, and is invariant under frequency transformation. In addition, it is stable in spite of parameter quantization. It is also free from the limit cycle and overflow oscillations. Finally, the pipelining, modular structure also makes it viable for VLSI implementation.

The basic building block of an orthogonal digital filter is the so-called rotor (cf. Fig. 7)— a circular rotation unit which realizes the following two-port transfer function:

$$\begin{bmatrix} x \\ g \end{bmatrix} = \begin{bmatrix} \sqrt{1-k^2} & k \\ -k & \sqrt{1-k^2} \end{bmatrix} \cdot \begin{bmatrix} f \\ y \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \cdot \begin{bmatrix} f \\ y \end{bmatrix} \qquad (19)$$

where $f$ and $y$ are inputs, and $x$, and $g$ is the output.

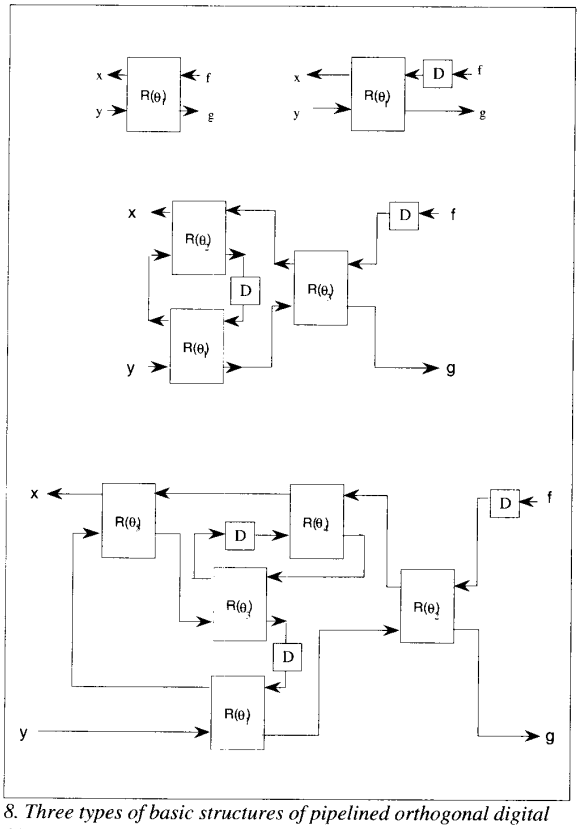
Gray and Markel [36] proposed a cascaded structure together with a tapped-sum output to realize a given transfer function. Henrot and Mullis [38] proposed a procedure based on the generalized Levinson algorithm to realize an ODF. Dewilde and Deprettere [27] based on an embedding technique, developed several pipelinable, cascaded ODF structures. In summary, three types of pipelined building blocks, based on different inter-connections of one or more rotors have been proposed [28] (cf. Fig. 8). Any transfer function can be realized by cascading a combination of these three building blocks.

Rao and Kailath [70] proposed a synthesis procedure based on the Schur algorithm to realize a uniform two-rotor-per-section ODF for stable transfer functions (Fig. 9). It was reported later [12] that the Schur algorithm is numerically sensitive to rounding error, and a remedy was developed. A different remedy using state space realization has subsequently been advanced by Desai [24]. Vaidyanathan [80] has presented a unified framework showing that a digital Lossless Bounded Real (LBR) two port pair can be realized with two plane rotations. This structure is also a normalized version of the series adaptor for wave digital filters [32].

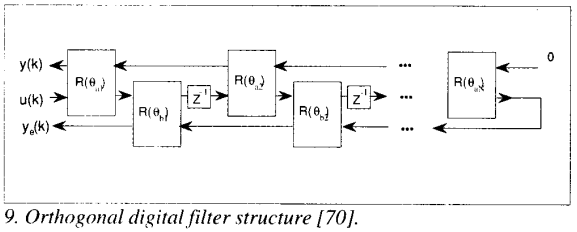
*Adaptive Lattice Filter*

Whereas an ODF is used to implement a given transfer function as a fixed-weight digital filter, a lattice filter has found most applications in adaptive signal processing [34]. In other words, most popular lattice filtering algorithms have been focused on the adaptive updating of the reflection coefficients so that it is able to keep track of the time varying parameters in the incoming signal. The building block of a lattice filter is a hyperbolic rotation unit (cf. Fig. 10), which can be described as:

$$\begin{bmatrix} f \\ g \end{bmatrix} = \frac{1}{\sqrt{1-k^2}} \begin{bmatrix} 1 & -k \\ -k & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cosh\theta & -\sinh\theta \\ -\sinh\theta & \cosh\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad (20)$$




*8. Three types of basic structures of pipelined orthogonal digital filter building blocks [28].*




*9. Orthogonal digital filter structure [70].*

In the linear prediction theory, $k$ in both of these equations are known as the *reflection coefficient*, or sometimes the PARtial CORrelation (PARCOR) coefficient. To ensure stability of the filter, the magnitude of every reflection coefficient must be less than unity. That is, $|k| < 1$. A basic cascaded lattice filter structure, implemented with Hyperbolic rotation CORDIC processors, is shown in Fig. 11. If the input $y(t)$ is an auto-regressive random process of order p, then the output of the $p^{th}$ stage $f_p(t)$ and $b_p(t)$ will be two white noise random processes.

The rotation angles $\{\theta_i;\ 1 \le i \le N\}$ can be estimated in a batch processing manner or a data adaptive manner. If the correlation function $\{R_y(n)\}$ of the input $\{y(t)\}$ is available, a Toeplitz system equation called the Yule-Walker equation can be solved to find out the N reflection coefficients; or equivalently, the N hyperbolic rotation angles. We will show later that a CORDIC processor array is able to solve a Teoplitz system of equations efficiently

If the sample data sequence $\{y(t)\}$ is to be used, we may compute the reflection coefficients using, say, the Burg's maximum entropy method [9]:

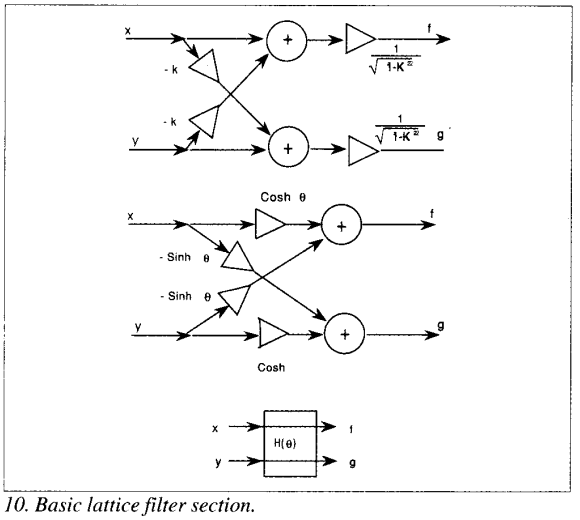
$$k_i = \frac{\sum_{t'} f_i(t') \cdot b_i(t'-1)}{\sum_{t'} f_i^2(t') + \sum_{t'} b_i^2(t')}$$

If we want to update $k_i(t)$ at each time instant, two classes of adaptive lattice filtering algorithms are available to do so: the gradient lattice filters (GLF), and the recursive least square lattice filters (RLSLF) [34]. The GLF algorithms updating the reflection coefficients use various gradient based formulations. For example, the reflection coefficient $k_i(t+1)$ at the $i^{th}$ stage at time $t+1$ can be computed as:

$$k_i(t+1) = k_i(t) + \alpha_i[f_i(t)b_{i-1}(t-1) + f_{i-1}(t)b_i(t)] \qquad (21)$$

where $\alpha_i$ is the step size. The RLSLF algorithms update the reflection coefficients using time and order updating by recursively solving a least square problem. Many fast RLSLF algorithms have been reported taking advantage of the special structure between successive data vectors. A comprehensive survey on adaptive lattice filter algorithms can be found in [34]. In Fig. 12, several building blocks of these lattice filter structures are shown.

Although the resulting prediction lattice filter structures




*10. Basic lattice filter section.*

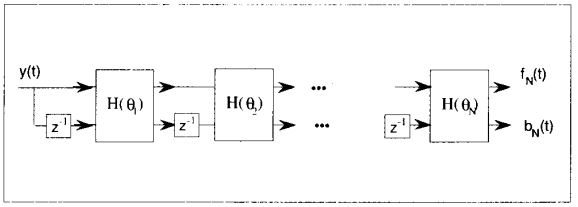
are suitable for implementation with a CORDIC processor array, the reflection coefficient adaptation algorithms themselves usually can not be easily implemented. Previously, Ahmed et al. [2] had implemented a CORDIC based ladder filter for speech analysis using a normalized lattice algorithm. Sibul and Fogelsanger [72] have proposed a modified CORDIC algorithm to implement RLSLF. Chen and Lin [13] proposed a dedicated hardware implementation of a normalized RLS lattice algorithm which they claim to be more efficient than CORDIC implementation.

Recently, Hu and Liao [49] have proposed a CORDIC-based adaptive lattice filtering (CALF) algorithm. CALF is a simplified gradient-based algorithm that is able to directly update the hyperbolic rotation angle without explicitly update the reflection coefficients. Note that there are only $2^n$ possible combinations of $\{\mu_i;\ \mu_i = \pm 1,\ 0 \le i \le n-1\}$. Hence, at most $2^n$ distinct rotation angles can be represented using the given set of shift sequence.

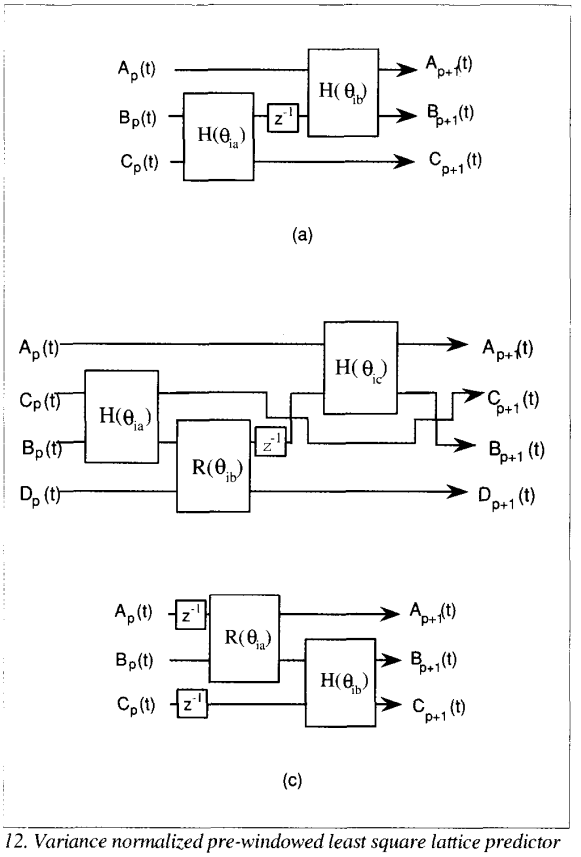
Let us order these $2^n$ angles in an ascending order, and label each of them with an index I, $0 \le I \le 2^n - 1$, such that:

$$-A_{max,-1} + a_{-1}(n-1) = \theta(0) \le \theta(I) \le \theta(I+1) \le \theta(2^n-1) = A_{max,-1} - a_{-1}(n-1)$$

With this set of indices, the updating formula of CALF can now be easily described:




*11. Basic lattice filter structure implemented with hyberbolic rotation CORDIC processors.*




*12. Variance normalized pre-windowed least square lattice predictor filter ((a), see [34]). Variance normalized sliding windowed covariance lattice predictor filter (b). Note that the second stage is a rotor implemented with circular rotation. Variance normalized pre-windowed time and order update lattice predictor filter (c).*

$$I_k(t+1) = I_k(t) + \text{sign of } (f_k(t)\cdot b_k(t)) \tag{22}$$

When $s(-1,k) = k+1$, the corresponding set of $\{\mu_i\}$ can be found directly using a simple up/down binary counter. This makes the implementation of CALF extremely simple compared with previous approaches. In Figure 13, a single section of the CALF structure is depicted where the adaptation of the rotation angle is realized with a simple up/down binary counter.

## Matrix Based DSP Computing Algorithms

Matrix based linear algebra computing algorithms such as linear system solvers, least square system solvers, QR factorization, eigenvalue or singular value decompositions arise quite often in modern digital signal processing applications. For example, linear system solvers can be used in covariance system equations in the linear prediction problem; QR factorization can be used in least square adaptive filter applications; and eigenvalue decomposition or singular value decomposition have been used in high resolution spectrum estimation, array processing and related problems. In this section, we will present CORDIC based processor array structures for implementing these matrix related DSP computing algorithms. Since we have to deal with portions of a matrix quite often, we shall adopt the following conventions to simplify the notations used in the algorithms: $A(1{:}5,:)$ will be the submatrix of A containing the first five rows, and $A(2{:}p,3{:}5)$ will be the submatrix of A containing elements in the intersection of the second to the $p^{th}$ rows with the third to the fifth columns.

*Elementary Row and Column Operations*

The purpose of the elementary row operation is to rotate corresponding pairs of elements in two row vectors so that one element in the second vector is annihilated. (For column vectors, a column operation can be performed by post-multiplying a $2 \times 2$ matrix.) Let $A(i,:)$ and $A(j,:)$ be the $i^{th}$ and the $j^{th}$ rows of the A matrix. Then, the row operation is equivalent to pre-multiplying these two row vectors by a 2 by 2 matrix **M**:

$$\boldsymbol{M}\begin{bmatrix} A(i,:) \\ A(j,:) \end{bmatrix} = \begin{bmatrix} \bar{A}(i,1)\bar{A}(i,2)\ldots\bar{A}(i,N) \\ 0 \quad \bar{A}(j,2)\ldots\bar{A}(j,N) \end{bmatrix} \tag{23}$$

We may choose **M** to be a unitary matrix so that circular rotation can be applied. In this case, we denote **M** by **C(i,j, A(j,1))** to emphasize that $A(j,1)$ is to be nullified, and to perform the row operation between the $i^{th}$ and the $j^{th}$ rows of the **A** matrix. It is easy to verify that:

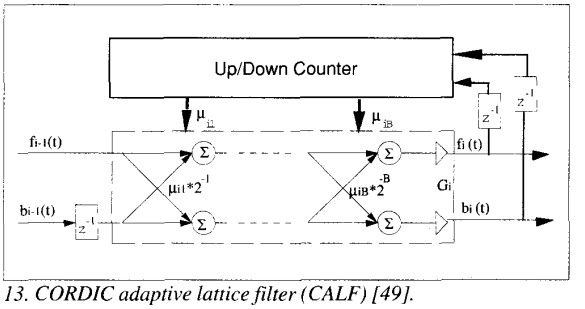
$$\boldsymbol{C}(i,j,A(j,1)) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{24}$$

with $\theta = -\tan^{-1}[A(j,1)/A(i,1)]$. Similarly, one may choose **M** so that a hyperbolic rotation between elements of $A(i,:)$ and $A(j,:)$ are performed. In this case, we denote

$$\boldsymbol{M} = \boldsymbol{H}(i,j,A(j,1)) = \begin{bmatrix} \cosh\psi & \sinh\psi \\ \sinh\psi & \cosh\psi \end{bmatrix} \tag{25}$$

with $\psi = -\tanh^{-1}[A(j,1)/A(i,1)]$.

The row operation can easily be realized with a doubly pipelined CORDIC array (Fig. 3c). $A(i,k)$ and $A(j,k)$ will be




*13. CORDIC adaptive lattice filter (CALF) [49].*

loaded to the local memory of the $k^{th}$ processor initially. The head processor will perform angle accumulation operation by evaluating $\mu_i$ in the $i^{th}$ iteration. The computed $\mu_i$ then will be propagated to each remaining processors. Note that we do not have to wait all the $\mu_i$s are computed before initiate the vector rotation operations in the remaining processors. Thus, the second processor will be able to commence its operation only one clock cycle ($t_o$) behind that of the head processor.

*QR Factorization*

In QR factorization, a matrix is factorized into the product of an orthonormal matrix and an upper triangular matrix. This operation can be accomplished by a sequence of circular row operations to nullify elements below the main diagonal of the matrix. The algorithm is described as follows:

***Initiation: Given a matrix*** $A_{p\times q} = A^{(0)}$
***For*** $l = 1$ ***to*** $\min(p-1,q)$ ***Do***
**For** i = l+1 to p **Do**

$$\begin{bmatrix} A^{(i)}(l,:) \\ A^{(l+1)}(i,:) \end{bmatrix} = C(l,i,A(i,l)) \begin{bmatrix} A^{(i-1)}(l,:) \\ A^{(l)}(i,:) \end{bmatrix}$$

***End i-loop***
***Output:*** $U(l,:) = A^{(p)}(l,:)$
***End l-loop***

A triangular array of CORDIC processors (Fig. 14) is capable of performing the QR factorization algorithm effectively.

*Example: Adaptive Antenna Nulling*

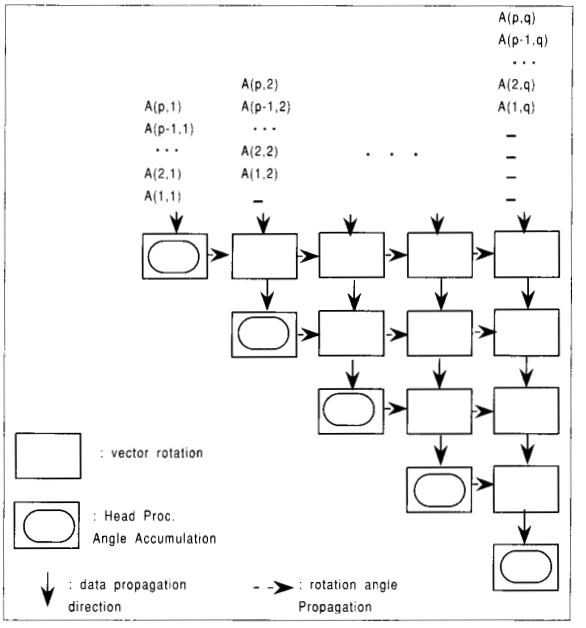
To illustrate the DSP applications of QR factorization, consider the least square adaptive antenna nulling problem: Let the row vector $\underline{x}(t) = [x_1(t), x_2(t),\ldots,x_p(t)]$ ] be the array input at time t; and $X(t) = [\ \underline{x}^t(t)\ \underline{x}^t(t-1)\ \ldots\ \underline{x}^t(0)\ ]^t$ be the data matrix. (Here, the superscript "t" refers to the matrix (vector) transpose. The index "$(t)$" refers to the $t^{th}$ time instant.) The objective is to find a p by 1 coefficient vector $\underline{W}(t)$ for each t to minimize the square error:

$$E(t) = \sum_{t'=0}^{t} |\ \underline{e}(t')\ |^2 = \sum_{t'=0}^{t} [\ \underline{d}(t') - X(t')\underline{W}(t')]^2 \qquad (26)$$

where $\underline{e}(t) = \underline{d}(t) - X(t)\underline{W}(t)$. Our approach is to apply QR factorization to X(t) such that:

$$Q(t)X(t) = \begin{bmatrix} U(t) \\ 0 \end{bmatrix}$$

where $Q(t)$ is an orthonormal matrix, $U(t)$ is a p by p upper triangular matrix, and 0 is a (t+1-p) by p zero matrix. Pre-multiplying the matrix $Q(t)$ on $\underline{e}(t)$, we have:




*14. q-by-q triangular CORDIC array for QR factorization. The matrix A is p-by-q (usually, $p \geq q$).*

$$\tilde{\underline{e}}(t) = Q(t)\underline{d}(t) - Q(t)\ X(t)\underline{W}(t) = \begin{bmatrix} \tilde{\underline{d}}(t) \\ \tilde{\underline{e}}(t) \end{bmatrix} - \begin{bmatrix} U(t) \\ 0 \end{bmatrix} \cdot \underline{W}(t). \qquad (27)$$

Thus, the least square solution of $\underline{W}(t)$ can be found as:

$$\hat{\underline{W}}(t) = U^{-1}(t) \cdot \tilde{\underline{d}}(t)$$

Assuming that a new (row) data vector $\underline{x}(t+1) = [x_1(t+1)\ x_2(t+1)\ \ldots\ x_p(t+1)]$ is now received. Our goal is to update $\hat{\underline{W}}(t)$ to $\hat{\underline{W}}(t+1)$. This can be accomplished by performing p row operations between each row of the $U(t)$ matrix and the $\underline{x}(t+1)$ data vector to nullify $\underline{x}(t+1)$. This procedure will lead to an updated upper triangular matrix $U(t+1)$, and the updated coefficient vector:

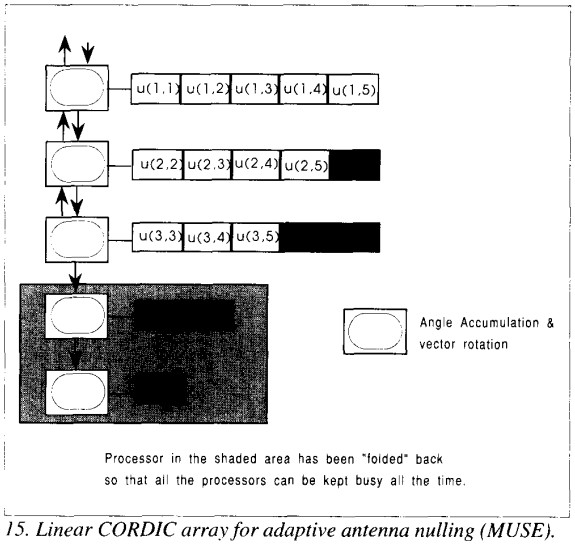
$$\hat{\underline{W}}(t+1) = U^{-1}(t+1) \cdot \tilde{\underline{d}}(t+1) \qquad (28)$$

Rader et al [69] have implemented a wafer scale linear CORDIC processor array called MUSE (Fig. 15) to implement this adaptive nulling algorithm. They incorporate a forgetting factor into the error formulation so that the scaling operation does not have to be exact. As a result, they achieve significant savings in the scaling operation.

Much research has been reported on using triangular systolic array for QR factorization [35], and recursive least square (RLS) processing [62]. Recently, Cioffi [15] reported a fast QR-RLS algorithm which requires O(N) rotations instead of $O(N^2)$ rotation.

*Example: Kalman Filtering (KF)*

QR factorization can also be applied to solve the Kalman




*15. Linear CORDIC array for adaptive antenna nulling (MUSE).*

filtering problem. A least square formula due to Paige and Saunders [67] embedded the KF recursion formula into a matrix triangularization formulation:

$$Q_{k+1}\cdot\begin{bmatrix} R(k) & 0 & b_k \\ \bar{C}(k) & 0 & \tilde{y}_{k+1} \\ \bar{F}(k) & W(k+1) & 0 \end{bmatrix}=\begin{bmatrix} R_{k,k} & R_{k,k+1} & b'_k \\ 0 & R(k+1) & b_{k+1} \\ 0 & 0 & r_k \end{bmatrix} \qquad (29)$$

where the definitions of each block entries can be found in [60]. Sung and Hu [77] has previously proposed a multiple VLSI systolic array implementation of the square-root formulation of the Kalman filter algorithm using a 32-bit CORDIC data path chip.

*Linear System Solver*

Given a $N\times N$ matrix A, and a $N\times 1$ vector $\underline{b}$, a linear systems of equations:

$$A\cdot\underline{x}=\underline{b} \qquad (30)$$

arises quite often in digital signal applications. Conventional methods involves the LU factorization of the A matrix, followed by a back-substitution step. On the other hand, Eq. (30) can be rewritten as $[A\ \underline{b}]\ [\underline{x}^t\ 1]^t=\underline{0}$. Hence the solution vector lies within the null space of the $[A\ \underline{b}]$ matrix. To find $\underline{x}$ explicitly, consider the triangularization of the $[A\ \underline{b}]^t$ matrix which is embedded in a $(2N+1)\times(N+1)$ matrix:

$$\boldsymbol{M}\cdot\begin{bmatrix} \boldsymbol{I}_N & \underline{0} & A^t \\ \underline{0}^t & 1 & \underline{b}^t \end{bmatrix}=\begin{bmatrix} * & * \\ \underline{p}^t & q \end{bmatrix}\cdot\begin{bmatrix} \boldsymbol{I}_N & \underline{0} & A^t \\ \underline{0}^t & 1 & \underline{b}^t \end{bmatrix}=\begin{bmatrix} * & * & U \\ \underline{p}^t & q & \underline{0}^t \end{bmatrix} \qquad (31)$$

where "*" are terms which are of no concern. If QR factorization is used, **M** will be an orthonormal matrix. If Gaussian elimination is used, **M** will be a lower triangular matrix. Note that the last row of the **M** matrix in above equation is such that $\underline{p}^t A^t + q\cdot\underline{b}^t=\underline{0}$. Hence, the solution to Eq. 30 can be found as: $\underline{x}=\dfrac{-1}{q}\underline{p}$.

This algorithm is a slightly modified version of the classical Fadeev algorithm. As a variation of the above algorithm, it is suggested [52] that when $A=U^t U$ and $A^{-1}=L^t L$ are both given, the above augmented matrix can be modified as:

$$\boldsymbol{M}\cdot\begin{bmatrix} \underline{0} & L & U \\ 1 & \underline{0}^t & \underline{b}^t \end{bmatrix}=\begin{bmatrix} * & * \\ \underline{p}^t & q \end{bmatrix}\cdot\begin{bmatrix} \underline{0} & L & U \\ 1 & \underline{0}^t & \underline{b}^t \end{bmatrix}=\begin{bmatrix} * & * & U_1 \\ q & \underline{r}^t & \underline{0}^t \end{bmatrix} \qquad (32)$$

Now we have $\underline{p}^t U + q\,\underline{b}^t=\underline{0}^t$, and $\underline{r}^t=\underline{p}^t L$. Therefore,

$$\underline{x}=\frac{-1}{q}\underline{r}=L^t U^{-t}\underline{b}=A^{-1}\cdot\underline{b}$$

*Toeplitz System and Covariance System Solvers*

In many advanced signal processing algorithms, the coefficient matrix A in the linear system of equations has a special low *displacement rank* [33], [55] structure. Displacement rank is a measure of closeness between a given square matrix and a certain Toeplitz matrix. A Toeplitz matrix has a structure that all elements along the diagonal direction are the same. That is,

$$t_{i,j}=t_{i-j}=t_k \quad \text{for } -N+1\le k\le N-1, \text{ and } 1\le i,j\le N$$

For example, a 4×4 Toeplitz matrix $T$ is:

$$T=[t_{ij}]=\begin{bmatrix} t_0 & t_{-1} & t_{-2} & t_{-3} \\ t_1 & t_0 & t_{-1} & t_{-2} \\ t_2 & t_1 & t_0 & t_{-1} \\ t_3 & t_2 & t_1 & t_0 \end{bmatrix}$$

A matrix has a structure which is close to a Toeplitz matrix if it has low displacement rank. Let us denote a *shift matrix* as:

$$Z=\begin{bmatrix} 0\ldots 0 & 0 \\ \boldsymbol{I}_{N-1} & \underline{0} \end{bmatrix}$$

The (+) displacement rank of a matrix A, denoted by $\alpha$, is then defined as the rank of a displaced matrix, $A-ZAZ^t$. In fact, the displacement matrix can be factorized (using, say, LDU factorization) into the form:

$$A-ZAZ^t=G\Sigma G^t=G_1G_1^t-G_2G_2^t \qquad (33)$$

where $\Sigma=\text{diag}\,[I_p - I_{\alpha-p}]$, and $G_{N\times\alpha}$ is called a *generator* matrix because the original A matrix can be recovered from $G$ according to the equation:

$$A = \sum_{i=1}^{\alpha} L(g_i)L^t(g_i) \tag{34}$$

where $L(g_i) = [\, l_{jk} \,]$ is a lower triangular Toeplitz matrix with $g_i$ being its first column. If the generator, $G$, is also *permissible*, that is, there exist an $\alpha \times 1$ vector $\rho$ such that $G\rho = [\, 1\ 0 \ldots 0 \,]^t$, then the $L\,L^t$ factorization of $A^{-1}$ can also be computed explicitly. Any generator matrix can be made permissable by adding two more redundant vectors [61]. With the generator matrix, a fast Cholesky factorization algorithm then can be applied to compute the $U^tU$ Cholesky factorization of the A matrix as well as the $L\,L^t$ Cholesky factorization of the $A^{-1}$ matrix, using $O(\alpha N^2)$ operations instead of $O(N^3)$ operations. Thus, this method will be very appealing when $\alpha << N$ ;

*Initialization: Given N, α, p, ρ, and the permissible generating matrix* $G_{N\times\alpha}$. *Let* $F^{(1)} = [\, \rho \;\; G(1{:}\alpha,1{:}N) \,]$, $U(1,1{:}N) = G(1,1{:}N)$, *and* $L(1,1{:}N) = [\rho_1\ 0 \ldots 0]$.

*For m = 1 to N Do*

/* Perform Circular rotation on the first p rows; and the remaining $\alpha$–$p$ rows of F separately.*/

$$\begin{bmatrix} \tilde{F}^{(m)}(1,1{:}N+1) \\ F^{(m+1)}(2{:}p,1{:}N+1) \end{bmatrix} = \prod_{j=2}^{p} \boldsymbol{C}(1,j,F^{(m)}(j,m+1)) \cdot F^{(m)}(1{:}p,1{:}N+1)$$

$$\begin{bmatrix} \tilde{F}^{(m)}(p+1,1{:}N+1) \\ F^{(m+1)}(p+2{:}\alpha,1{:}N+1) \end{bmatrix} = \prod_{j=p+2}^{\alpha} \boldsymbol{C}(p+1,j,F^{(m)}(j,m+1)) \cdot F^{(m)}(p+1{:}\alpha,1{:}N+1)$$

/* Perform Hyperbolic rotation on the first and the $p+1^{th}$ rows of F*/

$$\begin{bmatrix} [L(m,1{:}m)\ \ U(m,m{:}N)] \\ F^{(m+1)}(p+1,1{:}N+1) \end{bmatrix} = \boldsymbol{H}(1,p+1,\tilde{F}^{(m)}(p+1,m+1)) \cdot \begin{bmatrix} \tilde{F}^{(m)}(1,1{:}N+1) \\ \tilde{F}^{(m)}(1,1{:}N+1) \end{bmatrix}$$

*output: [L(m,1:m) U(m,m:N)]*

/* Right shift the first row of the F matrix */

$F^{(m+1)}(1,1{:}N+1) = [0\ \ L(m,1{:}m)\ \ U(m,m{:}N-1)]$

*End m-loop*

*Example: Fast Deconvolution Algorithm [47]*

Consider a discrete time, causal linear system with known finite impulse response sequence {h(k)}, random input sequence {u(k)}, and random observation noise sequence {n(k)}. Let {z(k)} be the output of this system, then:

$$z(k) = \sum_{m=0}^{N-1} h(m)u(k-m) + n(k) \tag{35}$$

for k = 1, 2, ..., N. Given {h(k)}, and {z(k)}, the objective of deconvolution is to recover the input {u(k)} in a least square sense. The solution requires the solution of a linear system of equations of the form:

$$\Gamma \boldsymbol{w} = (\nu^2 \boldsymbol{H}\boldsymbol{H}^t + \sigma^2 \boldsymbol{I}_N)\boldsymbol{w} = z \tag{36}$$

where $\nu^2 = \mathbf{E}\{u^2(k)\}$, $\sigma^2 = \mathbf{E}\{n^2(k)\}$, and $\mathbf{H}$ is a lower triangular Toeplitz matrix with its first column being $[h(0)\ h(1) \ldots h(N-1)]^t$. It is easy to verify that $\alpha = 2$, $p = 2$. Furthermore, since only one row of the U matrix ($U^tU$ factorization of $\Gamma$) and of the L matrix ($LL^t$ factorization of $A^{-1}$) are computed in each iteration, the linear system solver described in section IV.3.4 can be applied to compute $\mathbf{w}$ explicitly. Note that due to the structure of the $\boldsymbol{C}$ matrix, we have $\alpha = 2$, $p = 2$. Hence, there is no hyperbolic rotation involved. Moreover, the generator matrix

$$G^t = \begin{bmatrix} \nu h(0) & \nu h(1) & \ldots & \nu h(N-1) \\ \sigma & 0 & \ldots & 0 \end{bmatrix}$$

is permissible, with $\rho = [0\ \ 1/\sigma\ ]^t$.

These observations lead to the following fast deconvolution algorithm, where the solution, $\mathbf{w}$, is computed explicitly:

*Initiation:*

$$F^{(1)} = \begin{bmatrix} 0 & \nu h(0) & \nu h(1) & \ldots & \nu h(N-1) \\ \dfrac{1}{\sigma} & \sigma & 0 & \ldots & 0 \end{bmatrix},$$

$$\text{and } E^{(1)} = \begin{bmatrix} 0_{N+2}^t \\ 1\ 0\ z^t \end{bmatrix}$$

*For m = 1 to N Do*

$$\tilde{F}^{(m)} = \boldsymbol{C}(1,2,F^{(m)}(2,m+1)) \cdot F^{(m)}$$

$$F^{(m+1)} = \begin{bmatrix} 0 & \tilde{F}^{(m)}(1,1{:}N) \\ \tilde{F}^{(m)}(2,1{:}N+1) \end{bmatrix},$$

$$E^{(m+1)} = \boldsymbol{C}(1,2,E^{(m)}(2,m+2)) \cdot \begin{bmatrix} E^{(m)}(1,1) & \tilde{F}^{(m)}(1,1{:}N+1) \\ E^{(m)}(2,1{:}N+2) \end{bmatrix}$$

*End m-loop*

$$\textit{Output: } \boldsymbol{w} = \frac{1}{E^{(N+1)}(2,1)} \cdot E^{(N+1)}(2,2{:}N+1)$$

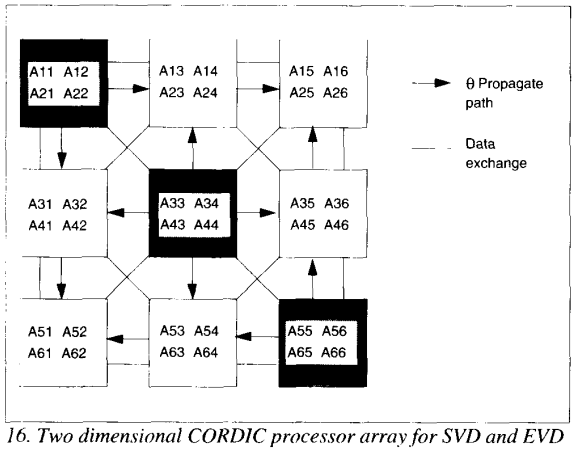
## Eigenvalue and Singular Value Decomposition

Eigenvalue decomposition (EVD) and singular value decomposition (SVD) have found extensive applications in modern digital signal processing. High resolution spectral estimation techniques, such as the Pisarenko's method [68], high resolution array processing algorithms such as the M-Ultiple SIgnal Classification (MUSIC) method [71], all require the computation of the eigenvalue decomposition or the singular value decomposition of the spatial covariance matrix of array inputs.

Given a $p \times q$ real matrix A, its singular value decomposition has the form:

$$A = U\Sigma V^t = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^t \tag{37}$$

where $U$ and $V$ are, respectively, $p \times r$ ($r < \min(p,q)$) and $r \times q$ orthonormal matrices satisfying $U^tU = V^tV = \boldsymbol{I}_r$. $\Sigma = \text{diag}$




*16. Two dimensional CORDIC processor array for SVD and EVD [10].*

$[\sigma_1, \ldots, \sigma_r]$ is a $r \times r$ diagonal matrix containing the $r$ singular values $\{\sigma_i;\ i = 1 \text{ to } r\}$. By definition, we have $\sigma_i > 0$, $1 \le i \le r$.

When A is a $p \times p$ real, symmetric square matrix, it admits an eigenvalue decomposition of the form:

$$A = E\Lambda E^t = \sum_{i=1}^{r} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^t \tag{38}$$

where $E$ is a $p \times p$ orthonormal matrix such that $E\,E^t = E^t\,E = \boldsymbol{I}_p$. $\Lambda = \text{diag}[\lambda_1, \ldots, \lambda_p]$ contains the p real eigenvalues.

Sequential algorithms for EVD and SVD has been well developed and ready to run Fortran routines can be found in public domain packages such as EISPAK [73], or LINPAK [29]. A number of efforts have also been made to map these sequential algorithms to a systolic array for parallel, pipelined processing [7], [10], [18], [39]. For real-valued matrices, a CORDIC processor can be applied to efficiently compute both singular value and eigenvalue decompositions.

Basically, a given matrix is diagonalized by pre- and post-multiplying with a sequence of unitary transformations. In other words, SVD and EVD can be accomplished with a sequence of row and column operations. Thus, a CORDIC processor array is very suitable for the implementation of SVD or EVD of real-valued matrices.

Below, we present an approach based on the classical Jacobi method [83]. This method was proposed by Brent and Luk [7], and later implemented by Delosme [18], Cavallaro [10] using CORDIC processor arrays. The Brent-Luk method is applicable both for SVD and EVD (symmetric matrix only) of real, square matrices. In the following discussion of this algorithm, we shall assume the A matrix is asymmetric, and comment on the changes need to be made when A become symmetric.

For convenience, we shall assume the dimension of the A matrix $p$ is an even number (if this is not the case, we simply append the A matrix with one row and one column of 0s so that it has even dimension).

We will partition the A matrix into $2 \times 2$ blocks. For each of these $p/2$ blocks along the diagonal of the A matrix, the following $2 \times 2$ two-sided rotation operation can be performed to nullify the two off-diagonal elements simultaneously:

$$R(\theta_l)^t \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix} \tag{39}$$

In Eq. (39), the left and right rotation angles, $\theta_l$ and $\theta_r$ can be solved from the following two equations:

$$\theta_l + \theta_r = \tan^{-1} \frac{c+b}{a-d} \tag{40a}$$

$$\theta_r - \theta_l = \tan^{-1} \frac{b-c}{d+a} \tag{40b}$$

In symmetric eigenvalue computation problems, $b = c$ on each diagonal block. Hence, Eqs. 40a and 40b reduce to:

$$\theta_r = \theta_l = \frac{1}{2} \tan^{-1} \frac{2c}{a-d}$$

Each $\theta_l$ computed in the $i^{th}$ diagonal block will be propagated to all the blocks in the $i^{th}$ row. Each $\theta_r$ computed will be propagated to all the blocks in the $i^{th}$ column. Thus, the $i,j$ off-diagonal block will receive $\theta_l$ from the $i^{th}$ diagonal block, and $\theta_j$ from the $j^{th}$ diagonal block. After every block has completed this two-side CORDIC iteration, every pair of the $2i^{th}$ and the $2i+1^{th}$ column, as well as the rows, will be interchanged. This enables the beginning of the next iteration.

Obviously, the two-sided rotation is very suitable for

CORDIC implementation [10]. Figure 16 shows a 3 × 3 CORDIC processor array for implementing this algorithm for a 6 × 6 matrix. Using the two-sided rotation method, the scaling operation can be significantly simplified: since two CORDIC operations are performed at a time, the scaling factor will become:

$$\frac{1}{K_1^2(n)} = \prod_{i=0}^{n-1} \frac{1}{1+2^{-2s(1,i)}} = \prod_{i=0}^{n-1} \frac{1}{1+2^{-2i}} \tag{41}$$

Now, if J = { 1, 3, 5, ..., $[\frac{n}{2}] - 1$ },

$$\prod_{j\in J} [\,1 - 2^{-2j}\,] \cdot \prod_{i=0}^{n-1} [\,1 + 2^{-2i}\,] = 1 + O(2^{-n-1})$$

where $O(2^{-n-1})$ are terms which are smaller than $2^{-n}$. Assuming the register length is also $n$ bits, these terms will not affect the accuracy of the result. Hence, up to n-bit accuracy, we have $1/K_1^2(n) = \prod_{j\in 0}[1 - 2^{-2j}]$. In other words, for every two CORDIC operations(left and right), approximately n/4 additional iterations are used for scaling. Thus the savings is significant.

In the original formulation, as seen in Eqs. 40a and 40b, the rotation angles $\theta_l + \theta_r$ and $\theta_r - \theta_l$ must be computed explicitly using Eq. 4 in order to compute $\theta_r$ and $\theta_l$. Thus, the double pipelining technique is not applicable. Delosme [18] has proposed some modifications to enable the use of double pipelining.

## CONCLUSION

In this article, we presented the basic CORDIC algorithm, and a partial list of potential applications of a CORDIC-based processor array to digital signal processing. Due to space limitations, many exciting on-going research projects could only be broadly addressed in the discussion. Details such as QR least square adaptive filters [15], and on-linear implementation of CORDIC algorithms [30], for example, have been left out. We hope that this article stimulates further interest in the development of CORDIC-based digital signal processing algorithms and CORDIC-based special purpose array processors.

## ACKNOWLEDGMENT

The author would like to thank Dr. J. Deller, editor-in-chief of the SP Magazine, for his patience during the preparation of this manuscript. Many useful comments from the anonymous reviewers are also deeply appreciated.

**Yu Hen Hu** received his B.S.E.E. degree from National Taiwan University, Taipei, Taiwan, ROC in 1976. He received the M.S.E.E. and Ph.D. degrees in Electrical Engineering from the University of Southern California, Los Angeles, in 1980, and 1982, respectively. From 1983 to 1987, he was assistant professor in the Department of Electrical Engineering at Southern Methodist University, Dallas, TX. He joined the Department of Electrical and Computer Engineering at the University of Wisconsin, Madison, in 1987 as an assistant professor. Currently, he serves as associate professor. His research interests include VLSI signal processing, artificial neural networks, spectrum estimation, fast algorithms and parallel computer architectures, and computer aided design tools for VLSI. He has published more than 100 journal and conference papers in these areas. He is a former associate editor (1988-90) for the *IEEE Transactions of Acoustic, Speech and Signal Processing* in the areas of system identification and fast algorithms. Dr. Hu is a member of IEEE, SIAM, and Eta Kappa Nu.

## REFERENCES

[1] Abuzzo, J., "Applicability of CORDIC algorithm to arithmetic processing", *IEEE Eighteenth Asilomar Conf. on Circuits, Systems and Computers*, Pacific Grove, CA, USA, Nov. 5-7, 1984.

[2] Ahmed, H.M., and M. Morf, "Synthesis and control of signal processing architectures based on rotations", *Proc. of the First Int'l Conf. on VLSI*, Edinburgh, Scotland, Aug. 18-21, 1981.

[3] Ahmed, H.M., M. Morf, D.T.L. Lee, and P.H. Ang, "A VLSI speech analysis chip set based on square-root normalized ladder forms". *Proc. 1981 ICASSP*, Atlanta, GA, 1981, pp. 648-653.

[4] Ahmed, H.M., "Signal Processing Algorithms and Architectures", *Ph.D dissertation*, Department of Electrical Engineering, Stanford University, Stanford, CA, June 1982.

[5] Ahmed, H.M., "Alternative arithmetic unit architectures for VLSI digital signal processors", *VLSI and modern signal processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[6] Bracewell, R. N., "Discrete Hartley Transform", *J. Opt. Soc. Amer.*, Vol. 73, No. 12, pp. 1832-1835, Dec. 1983.

[7] Brent, R. P, and F. T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM J. Sci. Stat. Comut.*, Vol. 6, No. 1, pp. 69-84, 1985.

[8] Bu,J., E.F.A. Deprettere, and F. De-Lange, "On the optimization of pipelined silicon CORDIC algorithm", *Proc. of EUSIPCO-86 Signal Processing III: Theories and Applications*, Hague, Netherlands, pp. 1227-30, Sept. 2-5, 1986.

[9] Burg, J. P., *Maximum Entropy Spectral Analysis*, Ph. D. dissertation, Stanford University, Stanford, CA, 1975.

[10] Cavallaro, J.R., and F.T. Luk, "Architectures for a CORDIC SVD processor", *Proc. SPIE Int. Soc. Opt. Eng. (USA)*, vol. 698, pp. 45-53, 1987.

[11] Chang, L.W., and S. W. Lee, "Systolic Arrays for the discrete Hartley Transform" *IEEE Trans. on Signal Processing*, Vol. 29, No. 11, Nov. 1991, 2411-2418.

[12] Chapmann, R, and M. A. Rahman, "A generalized design method for orthogonal IIR lattice filters" *Proc. ICASSP88*, Glasglow, Scotland, 1988, pp. 829-832.

[13] Chen, S. G. and J.-F. Lin, "Efficient implementation of the normalized recursive least square lattice filter", *Proc. ICASSP91*, pp. 1565-1568, Toronto, CANADA, 1991.

[14] Chen, W.H., C.H. Smith, and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform", *IEEE Trans. on Communications*, Vol. COM-25, pp. 1004-9, Sept. 1977.

[15] Cioffi, J. M., "The fast adaptive ROTOR's RLS algorithm", *IEEE Trans. on Signal Processing*, Vol. 38, No. 4, Apr. 1990, pp. 631-653.

[16] Cosnard, M.,A. Guyot, B. Hochet, J.M. Muller, H. Ouaouicha, P. Paul, and E. Zysman, "The FELIN arithmetic coprocessor chip", *IEEE Proc. of the 8th Symp. on Computer Arithmetic*, Como, Italy, May 19-21, 1987.

[17] Delosme, J.M., "VLSI implementation of rotations in pseudo-Euclidean spaces", *ICASSP Proc. of Int'l Conf. on Acoustics, Speech and Signal Processing*, Boston, MA, pp. 927-930, April 14-16, 1983.

[18] Delosme, J.M., "A processor for two-dimensional symmetric eigenvalue and singular value arrays,", *Proc.* $21^{-th}$ *Asilomar Conf. on Circ., Syst., and Compu.*, pp. 217-221, Nov. 1988.

[19] Delosme, J.M., "CORDIC algorithms: theory and extensions", *Proc. SPIE, Advanced Algorithms and Architectures for Signal Processing IV*, Vol. 1152, 1989.

[20] Delange, A.A.J., Ed.F. Deprettere, A.J. Vander Veen, and J. Bu, "Real Time Applications of the Floating Point Pipeline CORDIC Processor in Massive-Parallel Pipelined DSP Algorithms", *Proc. ICASSP Int'l Conf. on Acoustic,* Speech, and Signal Processing, pp. 1013-6, April 1990.

[21] Deprettere, Ed. F., "Synthesis and fixed point implementation of pipelinedtrue orthogonal filters", *IEEE Int'l Conf. on ASSP,* Boston, MA, Vol. 1, pp. 217-220, 1983.

[22] Deprettere, Ed. F., P. Dewilde, and R. Udo, "Pipelined CORDIC Architectures for Fast VLSI filtering," *IEEE Int'l Conf. on ASSP*, Florida, pp. 41A.6.1-4, April 1984.

[23] Deprettere, Ed.F., and K. Jainandunsing, "Orthogonal and J-orthogonal matrix inversion techniques", *IEEE Int'l Symp. on Circuits and Systems*, Philadelphia, PA, USA, May 4-7, 1987.

[24] Desai, U. B., "A state-space approach to orthogonal digital filters for VLSI implementation", *Proc. ICASSP89*, New York, NY, 1990, pp. 1255-1258.

[25] Despain, A.M., "Fourier Transform Computers using CORDIC iterations," *IEEE Trans. on Computers*, Vol. 23, pp. 993-1001, Oct. 1974.

[26] Despain, A.M., "Very Fast Fourier Transform Algorithms for Hardware Implementation", *IEEE Trans. on Computers*, Vol. C-28, pp. 333-341, May 1979.

[27] Dewilde, P, and E. F. Deprettere, and C.V. K. Prabhakara Rao, "Orthogonal digital filters", *Proc. ICASSP84*, Vol. 1., pp. 230-233, San Diego, CA, 1984.

[28] Dewilde, P., Ed.F. Deprettere and R. Nouta, "Parallel and pipelined VLSI Implementation of signal processing algorithms," in *VLSI and Modern Signal Processing*, S.Y. Kung et al, Eds., Prentice Hall Series, 1985.

[29] Dongarra, J., J. R. Bunch, C. B. Moler, and G.W. Stewart, *LINPACK User Guide*, SIAM Publications, Philadelphia, 1978.

[30] Ercegovac, M.D., and T. Lang, "Implementation of fast angle calculation and rotation using on-line CORDIC", *Proc. 1988 IEEE Int'l Symp. on Circuits and Systems*, Espoo, Finland, June 7-9, 1988.

[31] Ercegovac, M.D., and T. Lang, "Redundant and On-Line CORDIC: Applications to Matrix triangularization and SVD", *IEEE Trans. on Computers*, Vol. 39, No. 6, pp. 725-740.

[32] Fettweis, A, "Pseudopassivity, sensitivity, and stability of wave digital filter," *IEEE Trans. Circuit Theory*, vol. 19, pp. 668-673, Nov. 1972.

[33] Friedlander, B., T. Kailath, M. Morf, and L. Ljung, "Extended Levinson and Chandrasekhar equations for general discrete time linear estimation problems," *IEEE Trans. on Automatic Control*, Vol. AC-23, No. 4, pp. 653-659, August 1978.

[34] Friedlander, B., "Lattice filtering for adaptive signal processing" *IEEE Proceedings*, vol.70, No. 8, Aug. 1982, pp. 829-867.

[35] Gentleman, W. M., and H. T. Kung, "Matrix triangularization by systolic array", *Proc. SPIE, vol. 298, Real-time signal processing IV*, pp. 19-26, Bellingham, Washington, 1981.

[36] Gray, A. H. Jr., and J. D. Markel, "A normalized digital filter structure", *IEEE Trans. on ASSP*, Vol. 23, No. 3, pp. 268-277, June 1975.

[37] Haviland, G.L., and A.A. Tuszynski, "A CORDIC arithmetic processor chip", IEEE Trans. Comput. (USA), vol. C-29, no. 2, pp. 68-79, Feb. 1980.

[38] Henrot, D, and C.T. Mulles, "A modular and orthogonal digital filter structure for parallel processing", *Proc. ICASSP83*, Vol. 2, pp. 623-626, Boston, MA, 1983.

[39] Heller, D. E., and I. C. F. Ipsen, "Systolic networks for orthogonal equivalence transformations and their applications", *Proc. 1982 Conf. on Advanced Research in VLSI*, MIT, pp. 113-122, 1982.

[40] Hu, X., R. G. Harber, and S. C. Bass, "Expanding the range of convergence of the CORDIC algorithm", *IEEE Trans. on Computers*, vol. 40, No. 1, pp. 13-21, Jan. 1991.

[41] Hu, Y.H., "Pipelined CORDIC architecture for the implementation of rotation-based algorithms", *Int'l Symp. on VLSI Technology, Systems, and Algorithms*, Taipei, Taiwan, May 1985.

[42] Hu, Y.H., and T.Y. Sung, "The Optimal Design of VLSI CORDIC processor," *Proc. Int'l. Symp. on VLSI Tech., Syst., and Appl.*, Taipei, Taiwan, R.O.C., pp. 31-35, May 1987.

[43] Hu, Y.H., "The quantization effects of the CORDIC algorithm", *IEEE Trans. on Signal Processing*, Vol. 40, No.4, pp. 834-844, April 1992.

[44] Hu, Y.H., and S. Naganathan, "Efficient Implementation of the Chirp Z-Transform using a CORDIC processor", *Asilomar Conf. on Circuits, Systems and Signals*, CA, Oct. 1988. Also in *IEEE Trans. ASSP*, Vol. 38, No. 2, Feb. 1990, pp. 352-354.

[45] Hu, Y.H., "Parallel eigenvalue decomposition for Toeplitz and related matrices", *Proc. ICASSP*, Glasgow, Scotland, pp. 1107-10, May 1989.

[46] Hu, Y.H., and S. Naganathan, "An Angle Recoding method for CORDIC algorithm Implementation", *IEEE Trans. of Computers*, (to appear).

[47] Hu, Y.H., and P.H. Milenkovic, "A fast least square deconvolution algorithmfor vocal tract cross-section estimation," *IEEE Trans. on ASSP*, June 1990, pp. 921-924.

[48] Hu, Y.H., and H.M. Chern, "VLSI CORDIC array structure implementation of Toeplitz eigensystem solvers," *Proc. ICASSP*, pp. 1575-8, April 1990.

[49] Hu, Y. H., and H.-E. Liao, "CALF: a CORDIC adaptive lattice filter", to appear at *IEEE Trans. on Signal Processing*, Vol. 40, No. 4, pp. 990-993, April 1992.

[50] Hu, Y. H., H. M. Chern, "An efficient CORDIC backward angle recoding algorithm and its applications to digital signal processing," (submitted).

[51] Hwang, K., "Computer Arithmetic principles, architecture, and design", *John Wiley & Sons*, Inc., New York, 1979.

[52] Jainandunsing, K., and E.F. Deprettere, "A new class of parallel algorithm for solving systems of linear equation," SIAM J. *Sci. Stat. Comput.*, vol. 10, No. 5, pp. 880-912, Sept. 1989.

[53] Jou, I. C., T. Sung, Y. Hu, and T. Parng, "A CORDIC implementation of pipelined Toeplitz system solver", *IEEE Int'l Symp. on Circuits and Systems*, Kyoto, Japan, June 1985.

[54] Jou, I.C., Y.H. Hu, and T.M. Parng, "VLSI algorithms and pipelined architectures for solving structured linear systems", *Proc. VLSI Algorithms and Architectures. Agean Workshop* on Computing, Loutraki, Greece, July 1986.

[55] Kailath, T., S.Y. Kung, and M. Morf, "Displacement ranks of matrices and linear equations," *J. Math. Anal. Appl.*, Vol. 68, No. 2, pp. 395-407, 1979.

[56] Kalman, R.E., "A new approach to linear filtering and prediction problems", *J. of Basic Engineering*\fR, 82D, pp. 34-45, March 1960.

[57] Kung, H. T., "Why systolic architectures?", *IEEE Computer*, Vol. 15, No. 1, Jan. 1982.

[58] Kung, S. Y., "A Toeplitz approximation method and some applications", *Proc. Int. Symp. Mathematical Theory, Network Systems*, Santa Monica, CA, Aug. 1981, pp. 262-266.

[59] Kung, S.Y., and Y.H. Hu, "A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems," *IEEE Trans. on ASSP*, Vol. 31, No. 1, pp.66-76, Feb. 1983.

[60] Kung, S. Y., *VLSI Array Processors*, Prentice Hall, 1987.

[61] Lev-Ari, H., and T. Kailath, "Lattice filter parameterizations and modeling of nonstationary process," *IEEE Trans. Inform. Theory*, IT-30, pp. 2-16, 1984.

[62] McWhirter, J. G., "Recursive least squares minimization using a systolic array", *Proc. SPI 431, real time signal processing VI*, pp. 105-112, 1983.

[63] Naganathan, S., and Y.H. Hu, "Architectural design styles in the VLSI implementation of real discrete Fourier transform", *Proc. ISCAS, IEEE Symp. on Circuits and Systems*, New Orleands, LA, May 1-3, 1990.

[64] Naganathan, S., "Design Methodology For The Implementation Of Rotaion BasedAlgorithms Using A CORDIC Processor", Ph.D dissertation, Department of Electrical Engineering, Southern Methodist University, Dallas, Tx., Mar. 1990.

[65] Note, S., J. Van Meerbergen, F. Cathoor, and H. De Man, "Automated synthesis of a high speed CORDIC algorithm with the Cathedral-III compilation system", *Proc. 1988 IEEE ISCAS*, Espoo, Finland, June 7-9, 1988.

[66] Oppenheim, A.V., and R.W Schafer, "Digital Signal Processing", *Prentice Hall*, 1975.

[67] Paige, C. and M. Saunders, "Least square estimation of discrete linear dynamic systems using orthogonal transformation", *SIAM J. Numerical Analysis*, pp. 180-183, 1977.

[68] Pisarenko, V.F., "The retrieval of harmonics from a covariance function", *Geophys J. Roy. Astronom. Soc.*, Vol. 33, pp. 347-366, 1973.

[69] Rader, C.M., D.L. Allen, D. B. Glasco, and C. E. Woodward, "MUSE - A systolic array for adaptive nulling with 64 degrees of freedom, using Givens transformations and wafer scale integration", MIT Lincoln Laboratory, Tech. report 886, May, 1990, Lexington, MA.

[70] Rao, S.K., and T. Kailath, "Orthogonal digital filters for VLSI implementation," *IEEE Trans. on Cir. & Syst.*, Vol. 31, No. 11, pp. 933-945, Nov. 1984.

[71] Schmidt, R., "Multiple emitter locations and signal parameter estimation", *Proc. RADC Spectral Estimation Workshop*, ROME, NY, 1979, pp. 243-258.

[72] Sibul, L.H., and A.L. Fogelsanger, "Application of coordinate rotation algorithm to singular value decomposition", *Proc. of IEEE Int'l Symp. on Circuits and Systems*, Montreal, Que., pp. 821-4, May 1984.

[73] Smith, B.T., J. M. Boyle, Y. Ikebe, V.C. Klema, and C. B. Moler, Matrix Eigensystem Routines: EISPACK Guide, 2nd ed., Springer-Verlag, New York, 1970.

[74] Sung, T.Y., Y.H. Hu, and H.J. Yu, "Doubly pipelined CORDIC array processor for solving Toeplitz Systems", *Proc. of the Twenty-third Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, Oct. 1985.

[75] Sung, T.Y., T. Parng, Y. Hu, and P. Chou, "Design and implementation of a VLSI CORDIC processor", *IEEE Int'l Symp. on Circuits and Systems*, San Jose, CA, pp. 934-5, May 1986.

[76] Sung, T.Y., Y.H. Hu, and H.J. Yu, "Doubly pipelined CORDIC array for digital signal processing algorithms", *IEEE ICASSP Int'l Conf. on Acoustics, Speech and Signal Processing*, Tokyo, Japan, pp. 69-72, Apr. 1986.

[77] Sung, T.Y., and Y. H. Hu, "Parallel VLSI implementation of Kalman filter", *IEEE Trans. on Aerospace and Electronic Systems*, Vol. AES 23, No. 2, pp. 215-24, March 1987.

[78] Timmermann, D., H. Hahn, B. J. Hosicka, and G. Schmidt, "A programmable CORDIC chip for digital signal processing applications", *IEEE J. of Solid-State Circuits*, Vol. 26, No. 9, Sep. 1991, pp. 1317-1321.

[79] Udo, R.,E. Deprettere, and P. Dewilde, "On the implementation of orthogonal and orthogonalizing algorithms using pipelined CORDIC architectures", *IEEE EUSIPCO Proc. of 2nd. European Signal Processing*, Erlangen, Germany, Sept. 1983.

[80] Vaidyanathan, P.P., "A unified approach to orthogonal digital filters and wave digital filters based on the LBR two-pair extraction", *IEEE Trans. on Circuits and Systems*, July 1985, pp. 673-686.

[81] Volder, J.E., "The CORDIC Trigonometric computing technique," *IRE Trans. on Electronic Computers*, Vol. EC-8, No. 3, pp. 330-4, Sept. 1959.

[82] Walther, J.S., "A Unified algorithm for elementary functions" *Spring Joint Computer Conf.*, pp. 379-385, 1971.

[83] Wilkinson, J. H., *The Algebraic Eigenvalue Problem*", Oxford University Press, 1965.

[84] Yang, B., and J.F. Bohme, "Reducing the computations of the SVD array given by Brent and Luk", *Proc. SPIE Advanced algorithms and architectures for signal processing*, vol. 1152, pp. 92-102, Aug. 1989.