Wafer-Scale Integration of a Large Systolic Array for Adaptive Nulling

C.M. Rader

■ **This article describes an architecture for a highly parallel system of processors that are specialized for real-time adaptive antenna nulling computations with many degrees of freedom. This system is called the Matrix Update Systolic Experiment (MUSE), and we describe a specific realization of MUSE for 64 degrees of freedom. Each processor uses the Coordinate Rotation Digital Computation (CORDIC) algorithm and has been designed as a single integrated circuit. Ninety-six such processors working together can update the 64-element nulling weights on the basis of 300 new observations in only 6.7 msec. This update rate would require 2.88 billion instructions per second from a conventional processor. The computations support 50 dB of signal-to-interference ratio (SINR) improvement in a sidelobe canceler. The simple connectivity between processors permits MUSE to be realized on a single large wafer by using restructurable VLSI.**

When an antenna array of N elements is subject to undesired interference, such as jamming plus the thermal noise in each of the N receivers, the interference power can be reduced, relative to the power in some desired signal, by forming as the system output a suitably weighted sum of the waveforms observed on all the antenna elements. We call this process *nulling*. Usually the choice of suitable weights must be made adaptively. The choice of weights that maximizes the signal-to-interference ratio (SINR) observed in the system output is the solution to a well-studied least-squares problem. The number of arithmetic steps required to solve this least-squares problem, for almost any algorithm chosen, is proportional to the cube of the number of antenna elements. Furthermore, because the statistical characteristics of the interference change with time, the adaptive weight determination must be performed repetitively and in real time.

When the antenna array is on board a satellite, the number of antenna elements that can be nulled in practice is limited by the combination of the real-time requirement driven by satellite motion and the cubic dependence of the computational cost of adaptive weight determination. A previous study of computational cost set this limit at $N = 26$, based on an assumed conventional digital signal processing architecture. This limitation is not absolute, because it depends on the resources we are willing to allocate to a nulling processor. Using more resources, however, is not an efficient way to handle a large number of antennas.

In this article we describe a specialized adaptive-nulling processor, called the Matrix Update Systolic Experiment (MUSE), which is capable of determining the weights for $N = 64$ antenna elements. Because of its novel architecture, it can be compactly realized by using restructurable wafer-scale integration; the resulting system fits in a 4-in square. MUSE is substantially smaller and lighter than a conventional processor, and it uses substantially less power. Although the MUSE processor is specialized for $N = 64$ antennas, the MUSE design concept can be applied to the design of a similar processor for an antenna array with a different number of elements.

We briefly explain the mathematics of adaptive null-

ing and the use of Givens transformations for voltage-domain computation of a Cholesky factor. We explain Coordinate Rotation Digital Computation (CORDIC) rotations and show how they are suitable for realizing the Givens transformations needed to update a Cholesky factor. We also develop the idea of a systolic array of computing elements, each of which is composed of three CORDIC rotation cells operating together. An array of connected computing elements shares the work of updating a Cholesky factor, and this work is the largest part of the computational task. By using a technique that we call latency-controlled interleaving, we show how to modify the systolic array to make it 100% efficient for the Cholesky update task.

Once a Cholesky factor is found, the nulling weights are the solution of a set of linear equations whose constant coefficients are the Cholesky factor. Our CORDIC cells are also used for the solution of the linear equations. We describe a modification of the timing of our systolic array so that it can be used for both the Cholesky update and the solution of the linear equations.

We then consider how the MUSE systolic array can be adapted for implementation by using Restructurable Very Large-Scale Integration (RVLSI) technology [1]. We describe a design that uses 96 identical cells and the plan that provides intercell discretionary connections.

Nulling with Givens Transformations

Figure 1 illustrates a typical adaptive-nulling system. The outputs of the N antenna elements are down-converted and the N waveforms are simultaneously sampled and digitized. At the nth sampling instant a complex number x_i comes from the ith antenna. We collect the simultaneous samples together in a column vector $\mathbf{X}(n)$. These vector samples are passed to a module that performs the actual nulling, and some samples are passed to another module that adapts the weights.

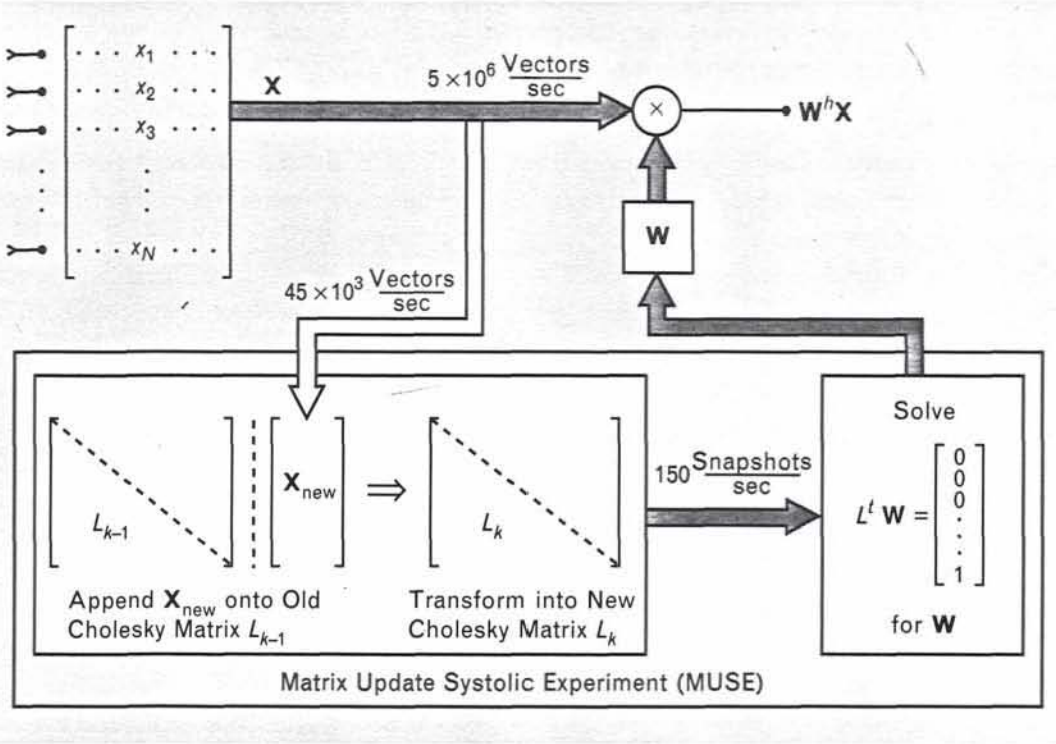

FIGURE 1. A diagram of a typical adaptive-nulling system. We simultaneously sample the waveforms on N antenna elements to give a column of complex numbers **X** for each sampling instant. The sampling rate must at least equal the radar bandwidth, which is 5 MHz in this example. We use many fewer columns of **X** per second (45,000 in this example) to characterize statistically the interference we wish to null. We take a snapshot 150 times per second of L_k, the matrix of interference statistics, and use it to compute the weights **W**.

The adapted weights can be treated as a column vector **W**. Although **W** varies with time, it is not recomputed for every sampling interval. The nulled output $y(n)$, which is a scalar, is formed by computing the dot product of the vector **W** with the vector $\mathbf{X}(n)$. We express the nulling operation by the vector equation

$$y(n) = \mathbf{W}^h \mathbf{X}(n),$$

where $(\,)^h$ is the Hermitian transpose.

The adaptation process shown in Figure 1 involves antenna samples $\mathbf{X}(n)$ and a lower triangular matrix L_k that is regularly updated and used to determine **W**. The interval within which **W** is held constant is defined as a *block*. Within each block, **W** is chosen optimally with respect to the statistics of interference appropriate to that block. The statistics that matter are the correlations of interference observed from one antenna element to another; these statistics are arranged into a matrix R, which we do not know in practice because it is a statistical expectation. Let

$$R = \mathcal{E}\left(\mathbf{J}(n)\,\mathbf{J}^h(n)\right),$$

where $\mathbf{J}(n)$ is the interference component of $\mathbf{X}(n)$. We estimate R by using samples of the vector $\mathbf{X}(n)$. The matrix R varies with time, but we have neither the time to gather enough samples to estimate R perfectly nor the computational resources to use all the samples we have. We must use a limited amount of data to obtain an estimate of R for each block, then gather another limited set of data to update R for the next block, and so on.

Let us therefore collect all the samples of $\mathbf{X}(n)$ that we intend to use to determine the optimum weights for a given block and refer to this collection of data as X. There is no implication that all the samples used come from within the given block—we actually envision using samples from within a sliding window. If the number of samples of $\mathbf{X}(n)$ is M and each sample is an N-element vector, the M vectors can be arranged into a matrix of M columns and N rows. The estimate of the correlation matrix based on this matrix of raw data is given by $\hat{R}$, where

$$\hat{R} = \frac{1}{M} X X^h . \qquad (1)$$

To optimize SINR we must first characterize the signal, given that we now have an estimate of the interference statistics. Let **S** be a known constant vector. It relates to the desired signal as follows: if all interference were absent and only a signal were present, we would expect to observe in $\mathbf{X}(n)$ some scalar time function times the vector **S**. Then the optimum choice of **W** is known from the literature [2] to satisfy the set of linear equations

$$R\mathbf{W} = \mathbf{S}.$$

These equations tell us, first, that the only information we need from the matrix X to determine the optimum weight vector is the information necessary to estimate R. Hence, if we intend to estimate R by using $\hat{R}$ in Equation 1, we can comfortably transform X in any way as long as that transformation leaves XX^h unchanged. Second, this equation tells us that we can find **W** from R by solving linear equations, and hence it suggests that we might accept as a good estimate of **W** the solution to the same equations using $\hat{R}$,

$$\hat{R}\mathbf{W} = \mathbf{S},$$

where $\hat{R}$ is the estimated correlation matrix in Equation 1 [3].

How can we transform X while leaving XX^h unchanged? Suppose we postmultiply X by a unitary matrix Q:

$$\hat{X} = XQ.$$

By definition a unitary matrix has the property $QQ^h = I$, so we can insert QQ^h between X and X^h in the equation for $\hat{R}$:

$$\hat{R} = \frac{1}{M} X X^h = \frac{1}{M} X Q Q^h X^h = \frac{1}{M} \hat{X} \hat{X}^h .$$

Therefore, we can transform X into $\hat{X}$ without changing the correlation matrix that we estimate from it. Furthermore, we can repeat this transformation with another unitary matrix, and repeat it again, as often as we like, until the final transformed version of X has a convenient form, with the nonzero elements confined to an $N \times N$ lower-triangular submatrix, which we call L, on the left. For example,

$$\begin{bmatrix} l & 0 & 0 & 0 & 0 & \cdots & 0 \\ l & l & 0 & 0 & 0 & \cdots & 0 \\ l & l & l & 0 & 0 & \cdots & 0 \\ l & l & l & l & 0 & \cdots & 0 \end{bmatrix} = [L|0].$$

The matrices L_{k-1} and L_k in Figure 1 are examples of L. We refer to lower-triangular matrices such as L as *Cholesky factors,* or *Cholesky matrices.*

Let us give L a physical interpretation. If the columns of L were the samples that arrive on the antenna, the optimum choice of weight vector $\mathbf{W}$ would be the same as for the actual data. Therefore, we are allowed to use L when we determine the weights, from which the determination of the optimum weight vector $\mathbf{W}$ is relatively simple.

By making this sequence of transformations we can write

$$\frac{1}{M} XX^h = \hat{R} = \frac{1}{M} LL^h .$$

Then the linear equations that must be solved to obtain $\mathbf{W}$ take the simpler form

$$LL^h \mathbf{W} = M\mathbf{S}.$$

The known constant M can be absorbed into the steering vector $\mathbf{S}$ and is hereafter omitted. (Because $\mathbf{W}$ is the solution to a set of linear equations, scaling $\mathbf{S}$ by any constant also scales the solution $\mathbf{W}$ by the same constant. But the SINR in $\mathbf{W}^h\mathbf{X}(n)$ remains unchanged when we multiply $\mathbf{W}$ by a scale factor, so the constant M can be ignored.)

We can solve $LL^h \mathbf{W} = \mathbf{S}$ in two steps. First define an intermediate vector variable $\mathbf{Y}$ defined by

$$L^h \mathbf{W} = \mathbf{Y}, \qquad (2)$$

so that $\mathbf{Y}$ is the solution to

$$L\mathbf{Y} = \mathbf{S}. \qquad (3)$$

We solve Equation 3 for $\mathbf{Y}$ and then, having found $\mathbf{Y}$, we are in position to solve Equation 2 for $\mathbf{W}$. Both of these equations are easier to solve than general matrix-vector equations because the matrices involved are triangular. (If the antenna array is designed so that the appropriate steering vector $\mathbf{S} = [0, 0, \ldots, 0, 1]^t$, then the triangular set of equations in Equation 3 becomes trivial and only the set in Equation 2 needs to be solved. This kind of adaptive antenna array is known as a *sidelobe canceler.*)

In summary, the determination of optimum weights for adaptive nulling can be divided into three phases:

1. Obtain interference. Form an $M \times N$ rectangular matrix X whose columns are the vector samples $\mathbf{X}(n)$ of interference observed at the antenna elements.
2. Triangularize. Use orthogonal transformations to compute a triangular Cholesky matrix L from X.
3. Back-substitute. Solve easy linear equations for the weights $\mathbf{W}$.

This process can be contrasted with a mathematically equivalent approach in which we first compute $\hat{R}$ by the equation

$$\hat{R} = XX^h ,$$

followed by a procedure called Cholesky factorization to determine L from $\hat{R}$, followed by the two back-substitutions. This approach is valid but not numerically robust. In the matrix $\hat{R}$, the largest and smallest eigenvalues are usually determined by jammer power and thermal noise power, respectively. Their ratio, if large, determines the minimum word length needed numerically to solve linear equations involving $\hat{R}$. This minimum-word-length limitation does not depend on the algorithm used to solve the linear equations. If, however, we determine L directly from the raw data X, without first computing $\hat{R}$, the minimum-word-length requirement for solving linear equations involving L or L^h is about half that needed with the approach that first computes $\hat{R}$. Because short word lengths are more economical in the design and construction of hardware, short-word-length algorithms are preferred.

Updating the Cholesky Matrix in Successive Blocks

Now let us look at the treatment of successive blocks. We must first recognize that both X and L are different in different blocks. Let the raw data X and the Cholesky matrix L in the kth block be X_k and L_k, respectively. How is X_k related to X_{k-1}?

If the statistics of interference remained constant with time, then we could expect to estimate those statistics

better and better by using more and more raw data in the successive blocks. We could append the observations that became available during the kth block, X_{new}, onto the matrix of all the observations available in earlier blocks X_{k-1}:

$$X_k = \left[X_{k-1} \middle| X_{new} \right]$$

so that

$$\hat{R}_k = \hat{R}_{k-1} + X_{new} X_{new}^h$$

and

$$L_k L_k^h = L_{k-1} L_{k-1}^h + X_{new} X_{new}^h = \left[L_{k-1} \middle| X_{new} \right] \left[L_{k-1} \middle| X_{new} \right]^h .$$

This relation suggests an efficient way to update L by using unitary transformations. If Q is a unitary transformation, then

$$L_k L_k^h = \left[L_{k-1} \middle| X_{new} \right] Q Q^h \left[L_{k-1} \middle| X_{new} \right]^h .$$

We choose Q to force zeroes into the positions occupied by X_{new}; thus

$$\left[\begin{array}{cccc|ccc} l & & & & n & \cdots & n \\ l & l & & & n & \cdots & n \\ l & l & l & & n & \cdots & n \\ l & l & l & l & n & \cdots & n \end{array}\right] \Rightarrow \left[\begin{array}{cccc|ccc} l & & & & 0 & \cdots & 0 \\ l & l & & & 0 & \cdots & 0 \\ l & l & l & & 0 & \cdots & 0 \\ l & l & l & l & 0 & \cdots & 0 \end{array}\right] .$$

There are two advantages to updating L_k by using L_{k-1}. The first advantage is that the use of prior work saves computation. The second advantage, which is more important, is that X_k involves more and more data, so that more and more storage is required as k increases. The storage of L_k requires only N^2 real numbers.

Because the interference statistics change with time, although slowly in comparison with the duration of a block, we would like to diminish the effects of the previous blocks. An often-used technique is to weight the data by an exponentially decaying window. To accomplish this diminishing window we define a *forgetting factor* called α, and substitute αL_{k-1} in place of L_{k-1} in the previous algorithm for updating L_k.

We can now update L recursively as each new vector sample becomes available, and we can preserve an after-the-update *snapshot* of $L = L_k$ at the end of the kth block. Let $\mathbf{X}$ be the new vector sample, let L_{old} be the Cholesky matrix for all previous data, and let α be the forgetting factor (α is now chosen on the basis of updating L on a per-sample basis rather than on a per-block basis). We use an $(N+1) \times (N+1)$ unitary matrix Q in the update

$$\left[\alpha L_{old} \middle| \mathbf{X} \right] Q \Rightarrow \left[L_{new} \middle| 0 \right] .$$

Next we discuss the mechanization of the unitary transformation. As we have previously explained, postmultiplication by a sequence of simple unitary transformations accomplishes the same effect as a single more elaborate transformation. Because our purpose is to introduce zeroes into the tacked-on column $\mathbf{X}$ of the matrix $\left[\alpha L_{old} \middle| \mathbf{X} \right]$, we can use a sequence of unitary transformations that each zeroes out one additional element.

The unitary transformations are of two types. The first type changes the leading nonzero element (which is typically a complex number) of the tacked-on column into a real number. A unitary matrix Q_θ that accomplishes this change is an $(N+1) \times (N+1)$ identity matrix with the lower right diagonal element replaced by $e^{j\theta}$. Postmultiplying $\left[\alpha L_{old} \middle| \mathbf{X} \right]$ by Q_θ leaves most of the matrix unchanged but multiplies the last column (the tacked-on column) by $e^{j\theta}$. We choose θ so that multiplying the leading element of the tacked-on column by $e^{j\theta}$ makes the product real.

The second type of unitary transformation is called a Givens transformation, denoted by Q_ϕ. This transformation is derived from an $(N+1) \times (N+1)$ identity matrix by changing four elements. The mth and $(N+1)$st diagonal elements become $\cos \phi$. The element in row $(N+1)$ and column m becomes $\sin \phi$. The element in column $(N+1)$ and row m becomes $-\sin \phi$. When we postmultiply by Q_ϕ we affect only the mth column and the tacked-on $(N+1)$st column. Suppose that the elements in the same row in those two columns are l and x.

After the transformation we will have

$$l' \Leftarrow l \cos\phi + x \sin\phi$$
$$x' \Leftarrow x \cos\phi - l \sin\phi .$$

If l and x are real, this transformation can be interpreted as a geometric rotation through the angle ϕ in the plane defined by l and x axes.

By an alternation of postmultiplications by unitary matrices of these two types, we can zero out the entire tacked-on column. Figure 2 illustrates this progression by using an example with $N = 4$, which requires eight transformations. Complex quantities in this figure are indicated by c and real quantities are indicated by r. For the Q_θ transformations only the tacked-on column (shown in blue) is changed, while for the Q_ϕ transformations only two matrix columns (shown in red) are changed. One of these columns is the tacked-on column and one is a column of the same length from the lower-triangular portion of the matrix. The other columns are not affected.

The mechanics of the two types of unitary transformation have much in common. The Q_θ transformation looks at the leading nonzero element of the tacked-on column and determines the angle θ that will make this leading element real. Because the column is composed of complex numbers, the determination of θ depends on two real numbers that are respectively the real part and the imaginary part of the leading element x_i:

$$\theta = \arctan \frac{-\mathrm{Im}(x_i)}{\mathrm{Re}(x_i)} .$$

When $\cos\theta$ and $\sin\theta$ are known the transformation Q_θ is applied to the remainder of the tacked-on column, in which the real and imaginary parts can be considered as two columns of real numbers. In terms of the two columns, the ith Q_θ transformation takes the form

$$\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \mathrm{Re}(x_i) & \mathrm{Im}(x_i) \\ \mathrm{Re}(x_{i+1}) & \mathrm{Im}(x_{i+1}) \\ \vdots & \vdots \\ \mathrm{Re}(x_N) & \mathrm{Im}(x_N) \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} .$$

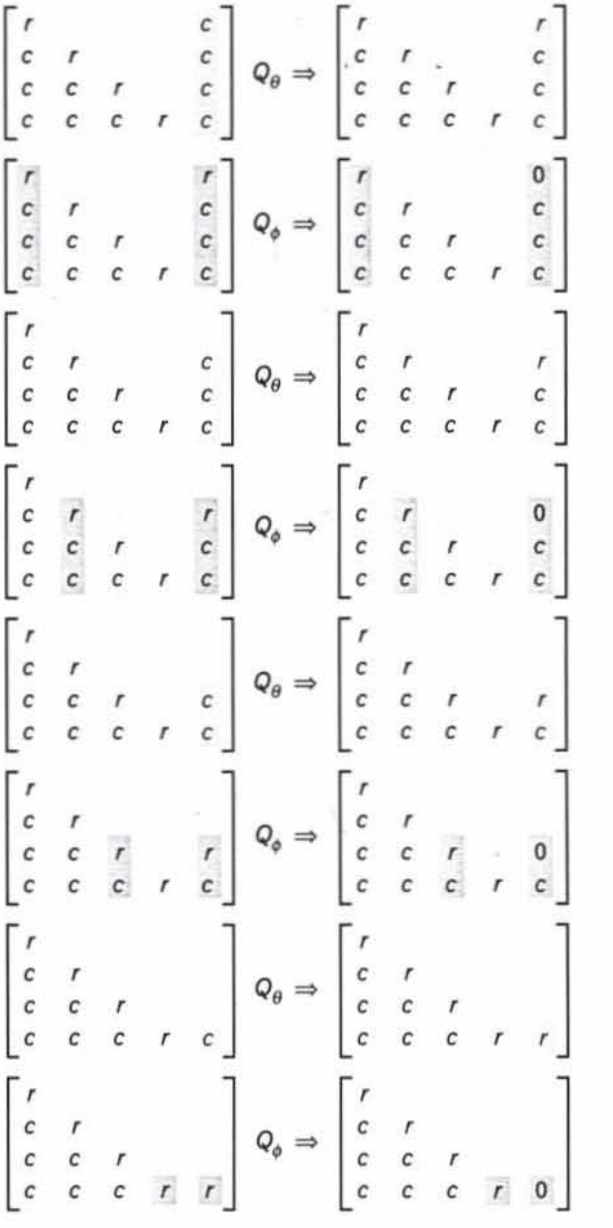

FIGURE 2. Eight successive unitary transformations that zero out the four-element tacked-on column.

On the other hand, for the ith Givens transformation Q_ϕ, the angle ϕ is determined by the leading elements of two columns. The number $l_{i,i}$ is the leading element of column i and the number x_i is the leading element of the tacked-on column. The leading element of the tacked-on column is real because of the preceding Q_θ transformation. If we assume that $l_{i,i}$ is real, then by mathematical induction it remains real because all the operations involved in updating it produce real numbers. The angle ϕ is determined by

$$\phi = \arctan \frac{-x_i}{l_{i,i}},$$

which is analogous to the determination of θ for the Q_θ transformation. Then, when $\cos\phi$ and $\sin\phi$ are available, the Givens transformation is applied only to the two columns affected by it.

We can write the Givens transformation in terms of these two columns as

$$\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ l_{i,i} & x_i \\ l_{i+1,i} & x_{i+1} \\ \vdots & \vdots \\ l_{N,i} & x_N \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}.$$

This representation shows that the mechanization of the Givens transformation is nearly identical to the mechanization of the Q_θ transformation. There is one important difference, however, between the two transformations. After we deal with the leading elements, we apply the remainder of the Givens transformation to two columns of complex numbers, whereas we apply the Q_θ transformation to two columns of real numbers. We overcome this difference by considering the Givens transformation to apply separately to the real parts of its two affected columns and to the imaginary parts of its two affected columns. In these terms a Givens transformation is identical to two Q_θ transformations. All the unitary transformations needed to zero out a tacked-on column appended to a triangular Cholesky matrix can be implemented by using identical hardware. In subsequent sections of this paper we describe a design for such hardware in detail.

Let us estimate the total workload involved in using the algorithm we have described to zero out a single 64-element vector. Each element in the Cholesky matrix is modified by using three rotations. To carry out each rotation with a conventional computer architecture requires two load instructions, four multiplications, an addition, a subtraction, and two stores, or approximately ten instructions per rotation. (The steps involved to rotate the leading elements are different and more involved, but we are counting them conservatively, as if they required the same number of steps as the other rotations.) Therefore, the total number of rotations is

$$3\big(N + (N-1) + (N-2) + \cdots + 1\big) = \frac{3}{2}N(N+1),$$

or 6240 rotations for $N = 64$. This number is equivalent to approximately 62,000 instructions per new sample.

Several authors have proposed parallel computation to compute the update of a Cholesky matrix. The best-known architecture for this computation is a systolic array of computing elements arranged in a triangular mesh [4]. For an array with N degrees of freedom this triangular array uses $N(N+1)/2$ processors, which limits practical application of the triangular array architecture to small values of N.

CORDIC Realization of Givens Transformations

In this section we describe a coordinate-rotation algorithm known as Coordinate Rotation Digital Computation (CORDIC) that is well suited for digital realization. The basic idea for this algorithm was first published in 1959 by J.E. Volder [5].

Suppose a vector from the origin, with endpoint coordinates (x, y), is rotated to new endpoint coordinates (x', y'), such that the angle between the new and old vectors is ξ. This rotation is represented by

$$x' = (\cos\xi)(x - y\tan\xi)$$
$$y' = (\cos\xi)(y + x\tan\xi),$$

which involves four multiplications. If we choose special angles, however, some of the multiplications simplify to shifts. We will concentrate on the special angles ξ_i with $i = 0, 1, 2, \ldots, \infty$, defined by

$$\tan\xi_i = \pm 2^{-i}.$$

The multiplications by $\tan\xi_i$ therefore become right shifts by i bit positions. For fixed i the two special angles have the same magnitude but opposite sign and therefore they have the same cosine.

The first step of the CORDIC algorithm is to de-

scribe an arbitrary angle ξ as a sequence of rotations either forward or backward by ξ_i, with $i = 0, 1, \ldots, \infty$. Thus

$$\xi = \sum_{i=0}^{\infty} \rho_i \xi_i .$$

Let $\rho_i = \pm 1$ determine whether a particular minirotation is forward or backward. The rotation of (x, y) through the angle ξ is therefore accomplished by the following sequence of steps:

$$\begin{aligned}
\begin{pmatrix} x \\ y \end{pmatrix} &\Leftarrow \cos \xi_0 \begin{pmatrix} x - 2^{-0}\rho_0 y \\ y + 2^{-0}\rho_0 x \end{pmatrix} \\
\begin{pmatrix} x \\ y \end{pmatrix} &\Leftarrow \cos \xi_1 \begin{pmatrix} x - 2^{-1}\rho_1 y \\ y + 2^{-1}\rho_1 x \end{pmatrix} \\
\begin{pmatrix} x \\ y \end{pmatrix} &\Leftarrow \cos \xi_2 \begin{pmatrix} x - 2^{-2}\rho_2 y \\ y + 2^{-2}\rho_2 x \end{pmatrix} \\
&\vdots
\end{aligned} \tag{4}$$

A key step in the CORDIC algorithm is to recognize that the multiplications by $\cos \xi_i$ can be collected together into a single constant K, where

$$K = \prod_{i=0}^{\infty} \cos \xi_i .$$

This constant is independent of the overall rotation angle ξ. Thus we can revise Equation 4 to the form

$$\begin{aligned}
\begin{pmatrix} x \\ y \end{pmatrix} &\Leftarrow \begin{pmatrix} x \\ y \end{pmatrix} + 2^{-0}\rho_0 \begin{pmatrix} -y \\ x \end{pmatrix} \\
\begin{pmatrix} x \\ y \end{pmatrix} &\Leftarrow \begin{pmatrix} x \\ y \end{pmatrix} + 2^{-1}\rho_1 \begin{pmatrix} -y \\ x \end{pmatrix} \\
\begin{pmatrix} x \\ y \end{pmatrix} &\Leftarrow \begin{pmatrix} x \\ y \end{pmatrix} + 2^{-2}\rho_2 \begin{pmatrix} -y \\ x \end{pmatrix} \\
&\vdots \\
\begin{pmatrix} x' \\ y' \end{pmatrix} &\leftarrow K \begin{pmatrix} x \\ y \end{pmatrix} .
\end{aligned} \tag{5}$$

We see that the CORDIC algorithm is composed of stages; most of the stages perform minirotations and the last stage performs a gain correction. For any angle ξ, all the minirotations are employed but each is employed in either a clockwise or counterclockwise direction. Although an exact realization of the CORDIC algorithm calls for an infinite number of minirotations, for obvious practical reasons we use a finite number of stages. The higher-numbered stages contribute little to the accuracy of the angle specification (a binary adder or multiplier uses a finite number of stages for the same reason). In addition, the correction stage is not a general-purpose multiplier but a multiplication by the fixed quantity K. An exact value of K depends on how many CORDIC stages are used, but when the number of stages is more than six, the appropriate correction is well approximated by 0.607253. (In the MUSE application, the fixed multiplication by K designed into each CORDIC circuit is combined with the forgetting factor α.)

The CORDIC method therefore achieves rotation without using trigonometric functions and without explicit multiplications. If the angle ξ is known in advance, it can be used to determine the set of controls

$$\left(\rho_i,\ i = 0, \ldots, i_{\max}\right),$$

each of which is represented by a single bit. This set of controls represents the angle ξ by digits ρ_i in an unconventional number system different from any of the conventional radix systems. Each ρ_i can be stored in a flip-flop in the stage whose direction of rotation it controls.

In the Givens transformation application, however, we do not know the angle of rotation in advance. We are given a coordinate pair (x, y) and we must rotate that pair through the angle such that the resulting rotated pair becomes $(x', 0)$. This operation is called *vectoring*. Then we must rotate some number of other pairs through the same angle. We have no need to know what the angle is, as long as we can rotate by that angle. Therefore, what we really need for vectoring is an algorithm that determines the CORDIC controls ρ_i. A major advantage of the CORDIC algorithm is that the same circuit used for rotating can be used for vectoring.

Consider the first CORDIC stage, for which the special angle is either 45° or –45°. Our purpose is to rotate the input (x, y) toward the x axis. If y is above the

axis we rotate down and if y is below the axis we rotate up. Therefore,

$$\rho_0 = sgn(x)sgn(y).$$

When we have determined ρ_0 we compute the effect of the first stage on x and y and pass (x, y) as modified to the second stage. Here again our rule is to rotate down if y is above the axis and up if y is below the axis.

$$\rho_1 = sgn(x)sgn(y).$$

In this way the determination of the CORDIC controls is simple.

$$\rho_i = sgn(x)sgn(y),\ i = 0, \ldots, i_{\max}.$$

We save these controls in the flip-flops of the specialized stages and use them to control those stages for the succeeding (x, y) pairs that are to be rotated through the same angle.

Figure 3 illustrates the concept of a CORDIC circuit, made up of independent stages, in which minirotations modify the input coordinates (x, y) from stage to stage. The controls ρ_i are shown as if they depend only on $sgn(y)$, but in fact they depend on the product $sgn(x) \cdot sgn(y)$. This simplified figure illustrates the natural pipelining aspect of CORDIC circuits. If registers are placed between the stages (at the positions illustrated by dashed lines), then a new rotation problem, with a new coordinate pair (x, y), can be started by the circuit as soon as the preceding pair is latched at the output of the first stage. Then another rotation can be started when the first two pairs have been latched at the output of the second and first stages, respectively, and so on. Rotation can follow vectoring in this pipelined fashion, and only an addition and a subtraction need to be performed in each stage (digital logic performs these operations rapidly). Therefore, a CORDIC circuit can begin new rotation problems at a high rate. By contrast the time required to complete any given rotation is proportional to the number of CORDIC stages provided.

CORDIC Circuits in a Systolic Array for Updating a Cholesky Matrix

The pipelined CORDIC circuit is ideally used as one of a large number of computing elements operating in parallel in a systolic array. The term *systolic* originated in

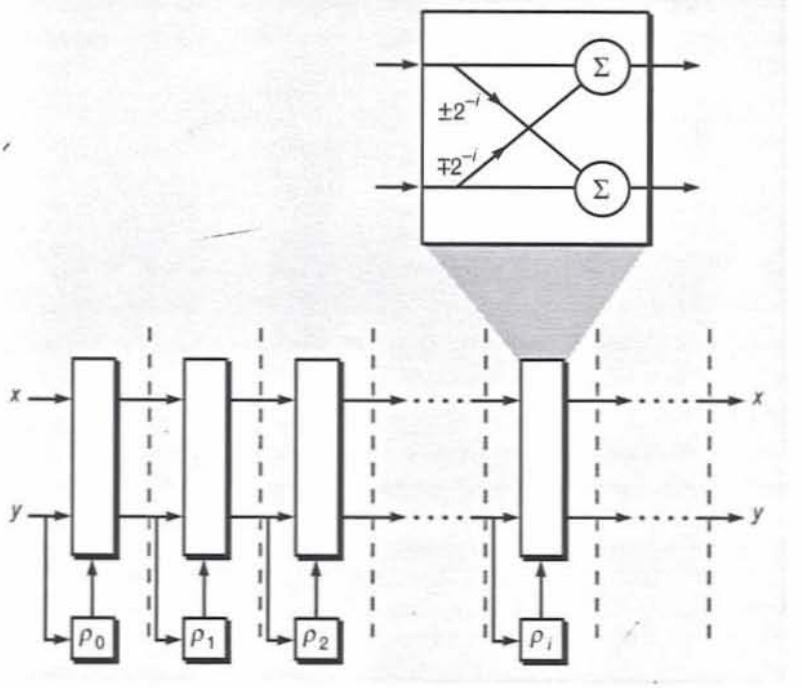

FIGURE 3. Pipelined CORDIC circuit. A rotation is accomplished as a sequence of minirotations through special angles $\xi_i = \rho_i \tan^{-1} 2^{-i}$, where ρ_i is either 1 or –1.

human biology to describe the circulatory system of blood through blood vessels and various organs. In a systolic computing system the data elements are figuratively pumped from computing organ to computing organ and processed as they move. Ideally, data elements in a systolic system should be processed as soon as they become available, so they must be available at the right place at the right time. The algorithmic and the architectural considerations are closely intertwined. This section describes how we use a large number of pipelined CORDIC circuits in a parallel systolic pipelined system to update a Cholesky matrix.

Supercell

To update one of the N columns of a Cholesky matrix L for each new vector sample, we need a Q_θ transformation that operates only on the tacked-on column X followed by a Q_ϕ transformation that operates on the tacked-on column X and a single column of L. We configure a processor for this task from three CORDIC circuits and sufficient memory to hold the one column of data. One CORDIC circuit performs the Q_θ transformation and two CORDIC circuits perform the Q_ϕ transformation. This combination of three CORDIC circuits and a column of memory is called a *supercell.*

Data vectors of complex numbers are sequentially presented to the CORDIC circuitry. The time for one complex word to enter a CORDIC circuit is called a *microcycle.* Every vector that enters a circuit has a leading element, or *leader* (its first element), and some number of following elements, or *followers* (all the other elements). The CORDIC circuits perform vectoring on the leader and rotation through the same angle for the followers. As an example, a vector composed of K elements flows into a CORDIC circuit during K consecutive microcycles. The elements that make up this vector flow out of the CORDIC circuit at the same rate they entered, which is one element per microcycle. Although passage through the CORDIC circuits modifies the elements, they retain their identity and their order, and the leader on input becomes the leader on output.

Simply passing the tacked-on column through a CORDIC circuit accomplishes the Q_θ transformation. The CORDIC circuit changes the phase of the leader so that the leader becomes a real number, and then changes the phase of the followers by the same amount. A CORDIC circuit used in this manner is called a θ-CORDIC.

A Q_ϕ (Givens) transformation is a rotation of pairs of complex numbers through a real angle. For each complex pair, two new pairs are assembled. One pair is formed from the two real parts and the other pair is formed from the two imaginary parts. These new pairs are separately rotated by the same angle. To accomplish a Q_ϕ transformation we use two CORDIC circuits, which we call the *master* ϕ-CORDIC and the *slave* ϕ-CORDIC. The master ϕ-CORDIC deals with the real parts of complex data while the slave ϕ-CORDIC deals with the corresponding imaginary parts of the data. The rotation of a column pair by either type of ϕ-CORDIC is isomorphic to a phase change in a θ-CORDIC. The columns are presented to both ϕ-CORDICs sequentially, one element per microcycle, just as is done in a θ-CORDIC. Figure 4 illustrates the architecture of a supercell.

The input vectors for ϕ-CORDICs contain complex elements from the output of a θ-CORDIC and from one column of the Cholesky matrix. The master ϕ-CORDIC gets the real part of its input vector from the real part of the Cholesky matrix column and its imaginary part from the real part of the θ-CORDIC output. The slave ϕ-CORDIC gets the real part of its input vector (except the leader) from the imaginary part of the Cholesky matrix column and the corresponding imaginary part from the imaginary part of the θ-CORDIC output. The leader of the vector that is input to a slave ϕ-CORDIC is a special case, which is discussed below. The outputs of the two types of ϕ-CORDICs are then reassembled. The real parts of the ϕ-CORDIC outputs become the real and imaginary parts, respectively, of the updated Cholesky matrix column. The imaginary parts of the ϕ-CORDIC outputs become the real and imaginary parts, respectively, of the updated tacked-on column. We explain this process in more detail below.

Note that the CORDIC circuits are used for two different meanings of rotation—one is the phase modification of a complex number and the other is the coordinate rotation of a two-element vector. In the coordinate rotation, although the two coordinates can be complex numbers, the angle through which they are rotated is real, and the same real rotation angle is used on the real and imaginary components, respectively.

To explain this process in more detail, first let us

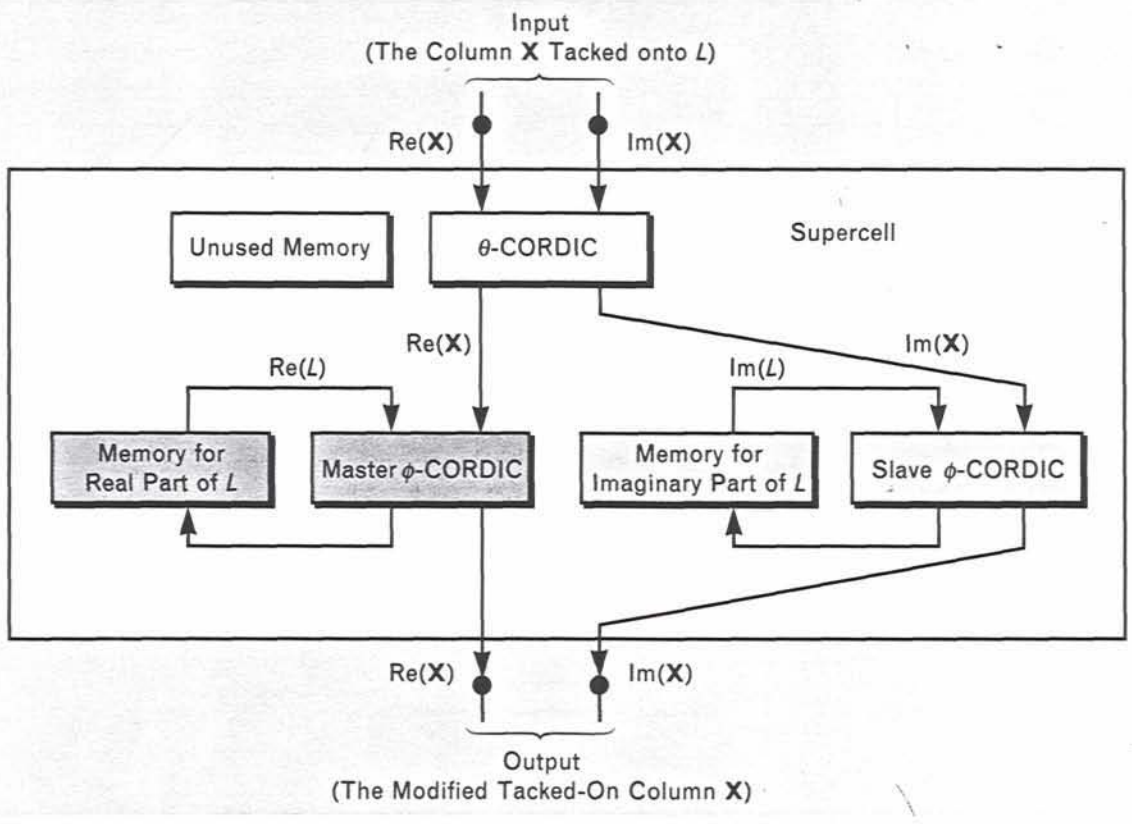

FIGURE 4. A CORDIC supercell. The θ-CORDIC determines the value of θ that makes the product $e^{j\theta}x_1$ real, and then multiplies each element in the tacked-on column **X** by $e^{j\theta}$. The ϕ-CORDIC finds the value of ϕ that rotates (l_{11}, x_1) into $(l_{11}, 0)$, and then rotates the two full (l, x) columns by that angle. The master ϕ-CORDIC works on the real parts of **X** while the slave ϕ-CORDIC works on the imaginary parts of **X**. The local memories in the ϕ-CORDIC hold the column of L to be updated.

consider the mth Q_θ transformation. Its input is the tacked-on $(N - m + 1)$-component column with complex components $[x_m, x_{m+1}, \ldots, x_N]$. These components are fed into a θ-CORDIC one pair at a time, beginning with the pair $[\mathrm{Re}(x_m), \mathrm{Im}(x_m)]$. The first pair is marked as a leader with a special indicator that causes the CORDIC circuit to determine and set the ρ_i controls as it passes through the successive rotation stages (vectoring). Thus the modified leader x_m that emerges from the θ-CORDIC is rotated in phase so that it is real—its imaginary component is zeroed. The followers $x_{m+1}, \ldots, x_N$ are rotated by the same phase as the leader but they generally emerge from the θ-CORDIC as complex numbers.

We can think of the CORDIC circuit as a pipe and visualize the tacked-on column being pumped through the θ-CORDIC one element at a time. The number of elements inside the pipe at any time is equal to the number of CORDIC stages.

In the mth Q_ϕ transformation the data pumped through the transformation are in two columns, $[l_{m,m}, l_{m+1,m}, \ldots, l_{N,m}]$ and $[x_m, x_{m+1}, \ldots, x_N]$. These columns are all complex numbers except for the real pair $(l_{m,m}, x_m)$. We arrange timing so that $l_{k,m}$ and x_k are available simultaneously. The master ϕ-CORDIC receives its two inputs from the real parts of the two input streams. The slave ϕ-CORDIC receives its two inputs (with one exception) from the imaginary parts of the two input streams. The pair $(l_{m,m}, x_m)$ is marked as a leader so that as it passes through the master ϕ-CORDIC it determines the angle controls ρ_i that rotate all the follower pairs $[\mathrm{Re}(l_{k,m}), \mathrm{Re}(x_k)]$. The same angle controls are used in the slave ϕ-CORDIC to rotate the follower pairs $[\mathrm{Im}(l_{k,m}), \mathrm{Im}(x_k)]$. The slave ϕ-CORDIC does not need to compute $[\mathrm{Im}(l_{m,m}), \mathrm{Im}(x_m)]$ because this pair already has the value (0, 0). Therefore, we trick the slave ϕ-CORDIC into setting its angle controls identical to

those of the master ϕ-CORDIC by giving it the leader pair [Re($l_{m,m}$), Re(x_m)] in place of [Im($l_{m,m}$), Im(x_m)]. The outputs of the two ϕ-CORDICs are an updated column of the Cholesky matrix and an updated tacked-on column with its mth element zeroed.

In this way, the three CORDIC circuits accomplish the two transformations needed to zero out one element in the tacked-on column. Figure 4 illustrates how the θ-CORDIC and the master and slave ϕ-CORDICs are interconnected and how the pipelining of each individual CORDIC circuit extends naturally to the pipelining of the supercell. The Cholesky matrix column [$l_{m,m}$, $l_{m+1,m}$, ..., $l_{N,m}$] can be stored within the supercell in local memory. The tacked-on column [x_{m+1}, ..., x_N], however, must be passed to another supercell where it is needed by the $(m + 1)$st Q_θ transformation and subsequent Q_ϕ transformation. Because the leader of the tacked-on column is zeroed out by passing through a supercell, the element immediately following it in the sequence becomes the new leader element.

Simple Systolic Array

Now that we have a configuration for a supercell that can update one column of a Cholesky matrix, we can configure a complete systolic array by using N such supercells. Figure 5 illustrates such a systolic array, except that the local memory in each supercell, which holds the column to be updated by that supercell, is shown separately. In

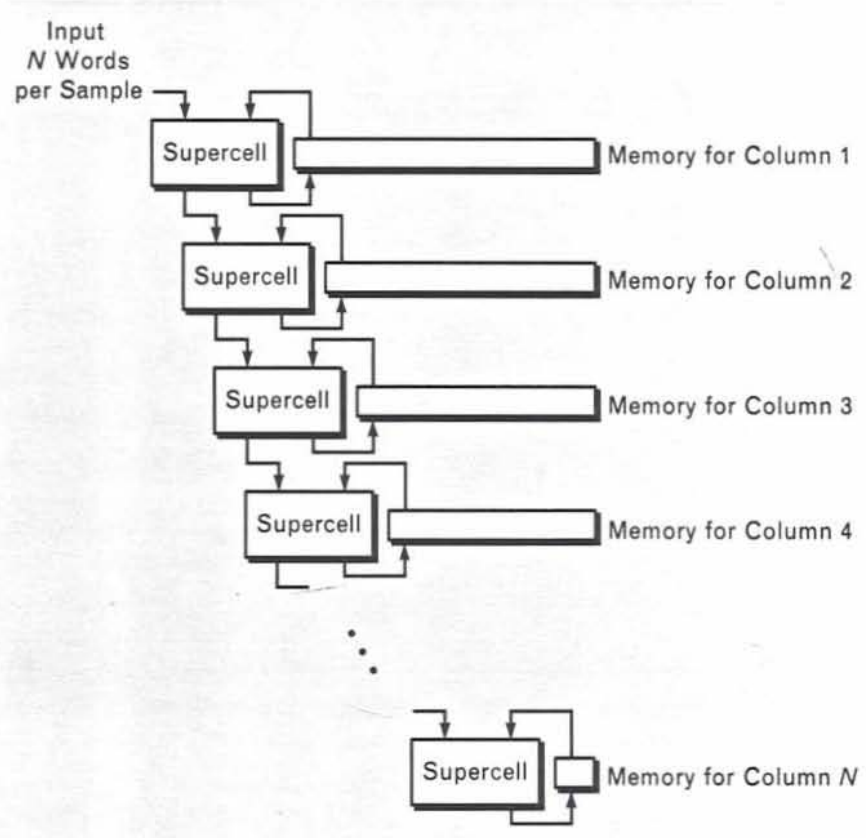

FIGURE 5. A systolic array of CORDIC supercells configured to update an $N \times N$ Cholesky matrix when a new N-element observation becomes available. Each supercell updates one column of the Cholesky matrix. We input the observation elements (the tacked-on column) one element at a time, and each supercell, as it updates its column, modifies the column, making it one element shorter, and feeds the column one element at a time to the next supercell in the chain. The memory required in each supercell to store the Cholesky matrix declines as the column proceeds forward along the chain of supercells. The computational work required of each supercell similarly declines.

this array the data passed from supercell to supercell are components of the tacked-on column, one complex word at a time. Thus, at the input to the first supercell, the data seen in order of arrival are

$$\overset{\downarrow}{\ldots}, x_1, x_2, x_3, \ldots, x_N \overset{\downarrow}{,} x_1, x_2, x_3, \ldots, x_N \overset{\downarrow}{,} \ldots.$$

The arrows mark the boundaries between successive tacked-on columns. Each tacked-on column consists of one vector sample of observed interference X. As soon as one tacked-on column has been pumped into the first supercell, that supercell is ready to receive another tacked-on column. The time interval between the first elements of two successive vectors is called a *macrocycle*.

The output of the first supercell is a modified tacked-on column. The other product of the supercell, the updated first column of the Cholesky matrix, is retained within the supercell to be used as original data for the next update. The modified tacked-on column has one fewer component than the original. Thus the output of the first supercell (which is the input to the second supercell) takes the form

$$\overset{\downarrow}{\ldots}, b, x_2, x_3, \ldots, x_N \overset{\downarrow}{,} b, x_2, x_3, \ldots, x_N \overset{\downarrow}{,} \ldots,$$

where b represents a blank, e.g., a time interval in which no data are present. The output of the second supercell (which is the input to the third supercell) takes the form

$$\overset{\downarrow}{\ldots}, b, b, x_3, \ldots, x_N \overset{\downarrow}{,} b, b, x_3, \ldots, x_N \overset{\downarrow}{,} \ldots.$$

with two blanks per tacked-on column. Strictly speaking, the blanks represent zero-valued samples. These blanks are not values that just happen to be zero—they are inevitably zero—so they need not be explicitly sent from supercell to supercell, nor involved in computation. Therefore, the first supercell is busy all the time, the second supercell is idle for one microcycle out of every N, the third supercell is idle for two microcycles out of every N, and the last supercell is idle for all but one microcycle out of every N.

Latency

The *latency*, or τ, of a pipelined digital circuit is defined as the number of inputs that a circuit accepts before the first output emerges. A supercell has the latency of two CORDICs. Therefore, each tacked-on column element not zeroed by a given supercell emerges from that supercell delayed by the supercell latency, relative to the time it entered the supercell. For example, Figure 6 illustrates the timing relationships (for $N = 5$ and $\tau = 3$) between corresponding elements of one tacked-on column as it is presented to the inputs of five supercells (numbered 0 to 4). As soon as the tacked-on column (shown in red) has entered the input supercell, that supercell begins to receive the subsequent tacked-on column (shown in blue). This example shows how a tacked-on column coexists in time with the next tacked-on column, as well as with the earlier and later tacked-on columns.

Supercell

into 0:	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	x_1	x_2
0 to 1:	x_3	x_4	x_5	b	x_2	x_3	x_4	x_5	b	x_2	x_3	x_4	x_5	b	x_2	x_3	x_4	x_5	b	x_2	x_3	x_4
1 to 2:	x_5	b	b	x_3	x_4	x_5	b	b	x_3	x_4	x_5	b	b	x_3	x_4	x_5	b	b	x_3	x_4	x_5	b
2 to 3:	b	b	x_4	x_5	b	b	b	x_4	x_5	b	b	b	x_4	x_5	b	b	b	x_4	x_5	b	b	b
3 to 4:	b	x_5	b	b	b	b	x_5	b	b	b	b	x_5	b	b	b	b	x_5	b	b	b	b	x_5

Microcycles →

FIGURE 6. The timing relationships for successive modifications of tacked-on columns as they are passed from supercell to supercell. For this example $N = 5$ and the latency is 3.

Figure 6 also illustrates an inefficiency in this systolic array. During many of the microcycles the higher-numbered supercells are not doing any computation. In addition, local memory in each ϕ-CORDIC is different because the columns of L are different lengths. The supercells must either be given different amounts of memory or, if they are all identical, most of them must waste memory.

Table 1. Pairing of Columns in the Same Supercell

Supercell	Column Number	Length	Column Number	Length
0	1	N	N	1
1	2	$N-1$	$N-1$	2
2	3	$N-2$	$N-2$	3
⋮	⋮	⋮	⋮	⋮
m	$m+1$	$N-m$	$N-m$	$m+1$
⋮	⋮	⋮	⋮	⋮
$\frac{N}{2}-1$	$\frac{N}{2}$	$\frac{N}{2}+1$	$\frac{N}{2}+1$	$\frac{N}{2}$

A solution to these problems is described in the next section.

Despite these inefficiencies, let us summarize the excellent characteristics of this architecture for updating a Cholesky matrix:

1. The CORDIC circuits are specialized for rotation; hence they are efficient.
2. The CORDIC circuits are naturally pipelined; hence they can be clocked at high speed.
3. Data are passed only locally from processor to processor; hence a global bus is not necessary.
4. Data elements arrive where they are needed exactly when they are needed; hence buffer storage is not necessary.
5. All CORDIC circuits operate in synchrony by using the same clocks.

Forgetting Factor

In practice we want to update the Cholesky matrix as if the data were weighted by an exponential time window. We accomplish this weighting if each ϕ-CORDIC attenuates its L output slightly by a forgetting factor α before reusing the L output in the next update. Each tacked-on column from the nth previous update affects the Cholesky matrix as if it had been multiplied by α^n. Because the CORDIC circuit is also designed to implement a correction gain K, we can combine the forgetting factor and correction gain into a single correction gain $K\alpha$. The choice of α is flexible; it can be chosen so that multiplication by $K\alpha$ uses minimum hardware. By selecting $\alpha = 0.99867003$, then $K\alpha = (2^6+2^2+1)(2^3+1)/2^{10}$, and the multiplication uses only three adder stages.

Ideally, the corrected gain of the θ-CORDICs, and of the ϕ-CORDICs when correcting the gain for the tacked-on column, should be unity. But we simplify the hardware design if we use only one correction gain. The attenuation of the tacked-on column due to the forgetting factor, as it passes through 128 CORDIC circuits in series, is slight enough so that insignificant difference occurs in nulling performance.

An Efficient Systolic Array with Latency-Controlled Interleaving

In the previous section we described a systolic array that updates a Cholesky matrix. This section introduces a major improvement that increases the efficiency of the systolic array. We assign each supercell the responsibility for updating not one but two columns of the Cholesky matrix. We pair the columns together so that the lengths of the two assigned columns will always add up to $N+1$, as shown in Table 1. (This technique assumes that N is even. A similar line of development can be worked out if N is odd. We would use the first supercell for the longest column and pair the remaining columns so that their lengths add up to N.)

Figure 7 illustrates a modified simple systolic array that uses the paired columns. This pairing equalizes the workload of the individual supercells, because each supercell now deals with two leaders and $N-1$ followers for every new vector sample of interference that comes along. As a result only half the number of supercells are needed. Furthermore, the memory used in a ϕ-CORDIC

to store the columns of the Cholesky matrix to be updated in that supercell is the same size—$N + 1$ words per ϕ-CORDIC—in all the supercells.

Although all the advantages of a systolic array are retained, two aspects of this improved structure need clarification. First, because the tacked-on column in the originally proposed systolic array is passed from supercell $m-1$ to supercell m, and because in the modified systolic array supercell m has the responsibilities of supercell $N-1-m$, each supercell must pass data both forward and backward. For example, supercell 2 must pass its output to supercell 3 but, later, when it is serving the role of supercell $N-3$ it must pass its output to supercell $N-2$, which is really supercell 1. Because these communications are still local, however, there is no need for a global bus.

The second consideration is more difficult to understand or to explain. Data elements in a systolic array must arrive where they are needed exactly when they are needed. It would not be suitable if supercell m were passed data from both supercell $m-1$ and supercell $m+1$ during the same microcycle. We can avoid all such possible collisions by choosing the latency τ as explained below.

Because supercell m must alternate its activity between accepting blocks of data of length $N-m$ from supercell $m-1$ and accepting blocks of data of length $m+1$ from supercell $m+1$, there is a periodicity of $N+1$ microcycles in its activity pattern. If we number microcycles consecutively, then whatever datum supercell m accepts during microcycle t, it will be accepting a corresponding datum during every microcycle whose number is congruent to t with modulus $N+1$. Thus the pattern of

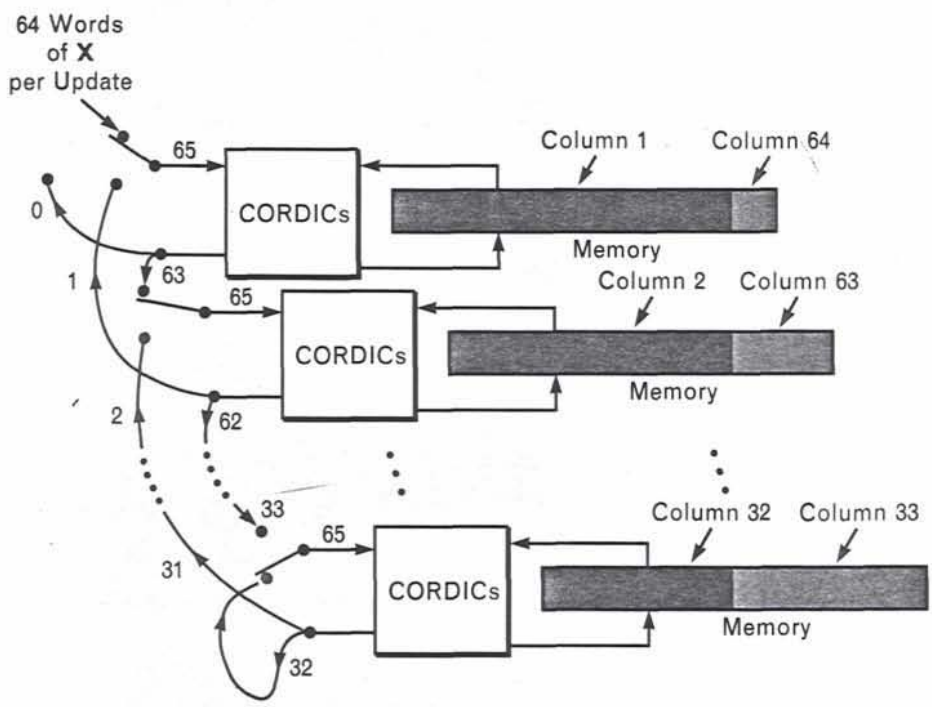

FIGURE 7. Global architecture of the folded systolic array. This architecture is derived from the simple systolic architecture illustrated in Figure 5. Each supercell in this figure plays the role of two supercells in Figure 5 and is responsible for updating two columns of the Cholesky matrix. The memory required to store one of the two columns is shown in red and the memory required for the other column is shown in blue. Each supercell, to update its red column, accepts a column from above and sends a modified column to the supercell below. To update its blue column, each supercell accepts a column from below and sends a modified column to the supercell above. These two processes are interleaved so that no supercell is required to accept data from above and from below in the same microcycle.

Table 2. Microcycles When Each Supercell Input Is Busy

Processor	When Inputs Presented	Inputs/Cycle
P_0	$0, \ldots, N-1$	N
P_1	$\tau+1, \ldots, \tau+N-1$	$N-1$
P_2	$2(\tau+1), \ldots, 2\tau+N-1$	$N-2$
$\vdots$	$\vdots$	$\vdots$
P_m	$m(\tau+1), \ldots, m\tau+N-1$	$N-m$
P_{N-1-m}	$(N-1-m)(\tau+1), \ldots, (N-1-m)\tau+N-1$	$m+1$

activity appearing at the input to any supercell is determined completely by the timing and latency of the earlier supercells.

We construct Table 2 by first listing the microcycles during which elements of an input tacked-on column arrive at the input to supercell 0. Then, taking account of the latency τ, we list the microcycles when elements of the resulting modified tacked-on column arrive at supercell 1, and so on. Soon the timing pattern of inputs for arbitrary supercell m from the logical supercell $m-1$ becomes obvious. One physical supercell serves as both logical supercell m and logical supercell $N-1-m$ (Table 2 also includes the pattern of activity for this supercell). These two patterns must follow each other indefinitely to form a periodic pattern modulo $N+1$. Thus microcycle $m\tau+N-1$ must be followed immediately by a microcycle whose number is congruent to $(N-1-m)(\tau+1)$ modulo $(N+1)$. The formal congruence

$$(m\tau+N-1)+1 \equiv (N-1-m)(\tau+1) \mod(N+1)$$

leads to

$$(m+1)(2\tau+1) \equiv 0 \mod(N+1).$$

If the same τ is to work for all m we must have

$$2\tau+1 \equiv 0 \mod(N+1).$$

The above congruence equation can be written as an equality by choosing an unknown multiplier d (a parity argument shows that if N is even then d must be odd), so that

$$2\tau+1 = d(N+1).$$

The smallest allowable latency occurs when $d=1$; namely,

$$\tau = \frac{N}{2}.$$

The idea of choosing a latency that allows all the various partially zeroed tacked-on columns of different lengths to interleave with one another without collisions is called *latency-controlled interleaving*. This concept allows us to fold a linear systolic array that is 50% efficient into a half-size linear array that is 100% efficient, without paying a penalty in complication of control.

Earlier, when we discussed the inputs to ϕ-CORDICs, we arranged that the quantities $l_{k,m}$ (which come from a local memory) and x_m (which come from a θ-CORDIC) should arrive at the ϕ-CORDIC input during the same microcycle. This timing requirement is also accomplished by latency-controlled interleaving. The delay from one update cycle to the next is $N+1$ microcycles. Therefore, as $l_{k,m}$ is updated it must be delayed by a combination of the latency of the ϕ-CORDIC and an extra delay so that it appears again at the input $N+1$ microcycles later. Because the latency of a ϕ-CORDIC is approximately $\tau/2$, we produce the extra delay by providing a small memory with approximately $N+1-\tau/2$ words.

Figures 7 and 8 illustrate the folded systolic array. Figure 7 shows the global architecture and Figure 8 shows the communication for a typical supercell and its neighbors. Two aspects are important in the control of

this systolic array. First the data passed from CORDIC cell to CORDIC cell must be marked to indicate whether it is a leader or a follower. Second, because a θ-CORDIC cell takes its input from two other supercells forward or backward in the chain, some control must tell it which supercell to choose.

A simple rule accomplishes both controls. At the input to the first supercell (supercell 0), we provide an input bit called *direction*. When *direction* = *true* the leading supercell takes its input from the given N-element tacked-on column. When *direction* changes to *false* the leading supercell takes its next input from supercell 1 (acting as supercell $N-2$). The change in *direction* from *false* to *true* or from *true* to *false* marks the arrival of a leader element, which causes the CORDIC to set its controls as that element passes through the stages.

The direction bit passes systolicly through both CORDIC cells, along with the tacked-on column, and emerges from the ϕ-CORDICs to be used in the next supercell. The leading edge must be delayed by one additional microcycle, however, relative to the tacked-on column, so that the change in direction is attached to the first nonzero element. The direction bit travels only forward along the chain (a backward-passed direction bit would be redundant and ignored). The control of the folded array is actually no more complex than the control of the original systolic array. Each supercell generates the control for the supercells that follow it in the chain.

Finite-Word-Length Effects

Many factors control the improvement in SINR achieved by adaptive nulling. These factors include the number and distribution of jammers, their strength in comparison to thermal noise in the system, the antenna element gain patterns, the mismatch of circuitry prior to digitization, the adequacy of the statistical representation of interference (principally, the number of samples that characterize the interference), and the accuracy of the digital computation.

Our goal was to demonstrate adaptive nulling with at least 50-dB improvement in SINR without relying on special knowledge of the environment or of the antenna elements. More precisely, if the dynamic range of the input data is not too much greater than 50 dB, the MUSE system should realize almost the full SINR improvement theoretically possible for that input data. To achieve this goal, we used extensive numerical simulation to select MUSE parameters that affect the accuracy of the digital computation. Each hardware configuration under consideration was simulated in software at the bit level by using simulated data sets.

Each simulated dataset was created to satisfy three constraints: (1) the number of jammers (all with equal power); (2) the ratio σ^2 of jammer power to noise power, and (3) the theoretically optimal SINR improvement possible with infinitely precise computation. In the literature on numerical analysis, σ is known as the *condition*

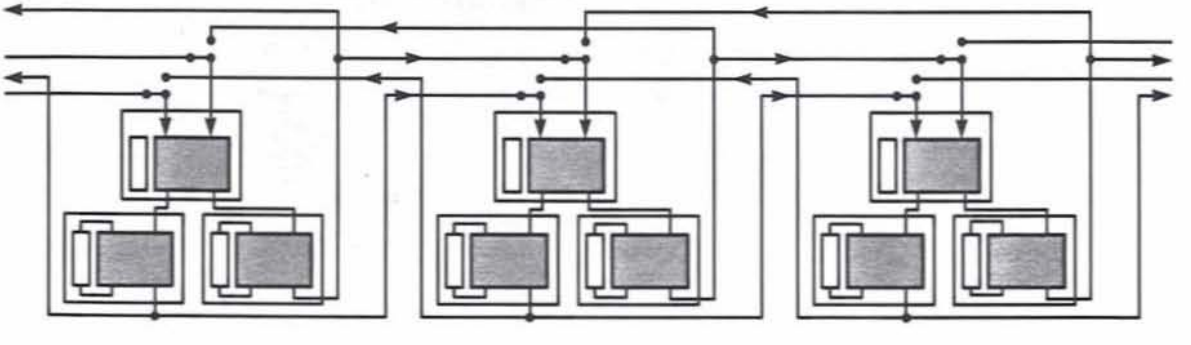

FIGURE 8. Local communication between CORDIC circuits within a folded systolic array. Within each supercell, the inputs external to the supercell come into the θ-CORDIC cell, which is shown in yellow. The output of each θ-CORDIC cell has a real part that goes to the master ϕ-CORDIC cell, shown in red, and an imaginary part that goes to the slave ϕ-CORDIC cell, shown in blue. The two ϕ-CORDIC cells also use local memory, shown in light grey, to read and write data. Outputs from the two ϕ-CORDIC cells in any supercell are transmitted to the θ-CORDIC cells in the two adjacent supercells.

number of the data [6]. The datasets were random except for these three constraints, and we tried several different datasets with the same constraints for each experiment. The Cholesky matrix determined by MUSE for each dataset was used to determine weights, and the weights were then used to determine the actual SINR improvement, which, in turn, was compared with the theoretically optimal SINR improvement. We therefore determined the loss in performance due to each hardware configuration under consideration.

On the basis of these simulations, we determined four finite-word-length parameters: the word length of the internal CORDIC registers, the word length of the tacked-on column, the word length of the Cholesky matrix, and the number of CORDIC stages. We also selected the forgetting factor α described earlier. Because the simulation of the CORDIC circuit takes exact account of the finite-word-length arithmetic process, the results it produces are bit-by-bit identical to the results expected from actual hardware.

A precise execution of Equation 5 increases the number of bits to the right of the binary point by i bits in the ith stage. Rounded arithmetic is thus needed in almost every stage. Such rounding introduces computational noise, and the total noise from all stages can be excessive. Our experiments showed that two guard bits at the low end of each word adequately reduce the rounding noise.

We also considered the problem of overflow, even though overflow in a CORDIC circuit might be unavoidable. If x and y are both near their maximum values at input and are rotated by 45°, then a component of the result can exceed the maximum by a factor of almost $\sqrt{2}$. This overflow must be prevented by an appropriate scaling of the inputs (scaling to prevent overflow is common to all fixed-point signal processing projects).

Overflow within an interior CORDIC stage is prevented in connection with realizing the combination of the CORDIC gain compensation and forgetting factor. The required fixed multiplication

$$K\alpha = \frac{621}{1024} = \frac{2^6 + 2^2 + 1}{2^7} \cdot \frac{2^3 + 1}{2^3}$$

is split between a factor of 69/128 at the input to the CORDIC (the guard bits are created at this point) and 9/8 at the output of the CORDIC (the guard bits are dropped at this point). Splitting the factor in this way insures that an overflow occurs in the interior of the CORDIC circuit only when it is an inevitable result of an accurate rotation in the available word length.

Each contrived dataset contained exactly $N = 64$ vector samples, and its ideal Cholesky matrix was determined. We computed ideal weights for each dataset by using floating-point arithmetic, and solved Equation 2 and Equation 3 by using the ideal Cholesky matrix, to confirm that the predetermined SINR improvement was correct. These 64-sample datasets were repeated as MUSE inputs several times in succession, and snapshots of L were taken after each block of 64 samples. With highly accurate arithmetic, we then used the computed Cholesky matrices to compute the MUSE weights. Any loss in SINR improvement when the MUSE weights were used in place of the ideal weights was therefore attributed to the computational error in computing L with finite-word-length arithmetic.

The section "Using CORDIC Cells to Solve Linear Equations for Weights" describes another method of computing MUSE weights. In this method the linear equations for the desired weights are solved, after L is already computed, by using the same finite-word-length CORDIC arithmetic. Nulling based on weights computed this way thus generally shows an additional loss in SINR improvement. This additional loss was also studied.

Figure 9 shows the relation between the condition number of a possible dataset and the SINR improvement possible for that dataset. The solid curve illustrates the upper bound of the set of possible scenarios. With a given condition number, no dataset is possible for which a sidelobe canceler can give an SINR improvement that lies above this curve. Some scenarios actually tested are marked as black points on the plot. The most interesting datasets are given by the points slightly below the upper bound; these points represent strong jamming that can be deeply nulled.

Individual experiments with fixed-word-length parameters produce reduced SINR improvements. The SINR improvements from the weights obtained by using MUSE's inaccurately computed Cholesky matrix to solve linear equations accurately are marked as red points on the plot. The SINR improvements for the same inaccurately computed Cholesky matrix when the weights are

computed instead by CORDIC cells are marked as blue points on the plot. When several scenarios with the same theoretical performance were tried, there is one black point and a scattering of red and blue points shown for the corresponding condition number.

The data in Figure 9 are for the final parameter choices. We can characterize the expected system performance from this data in two ways. First, the loss in SINR improvement due to parsimonious parameter choices increases with the condition number of the data. For a dataset with a condition number of 300 the loss is negligible; for a dataset with a condition number of 1000 the loss is approximately 2 dB. Second, the loss in SINR improvement does not seem to depend on the number of jammers as long as there are fewer jammers than degrees of freedom. All the experiments illustrated in Figure 9 used 35 jammers.

The final parameter choices were as follows. The number of CORDIC stages, exclusive of the three stages used for the combination of CORDIC correction gain and forgetting factor, was selected as 13 (e.g., $i_{max} = 12$). The word length of internal registers of the CORDIC cell was chosen as 24 bits. Of these 24 bits the two least significant bits are guard bits and are not passed beyond the CORDIC. The tacked-on column data passed between CORDIC cells use a 22-bit wordlength. The stored matrix elements $l_{i,j}$ are also 22-bit words. The inputs obtained from the antenna elements, although presented as 22-bit words, must not be allowed to use the full 22-bit dynamic range because the Cholesky matrix generated from many such tacked-on columns contains the energy of all of them and therefore is much larger in dynamic range than any single vector.

Realization of MUSE with Restructurable VLSI

The systolic array described in the preceding sections has

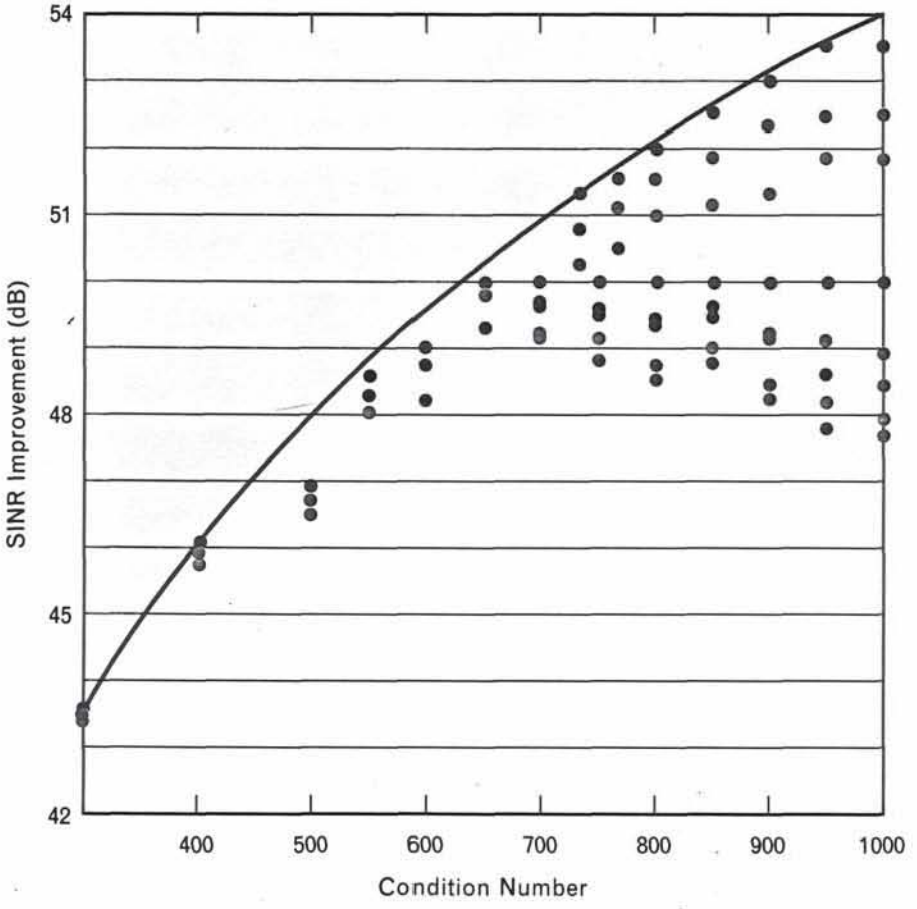

FIGURE 9. Scenarios and performance for ideal nulling and for simulated hardware parameter choices.

32 supercells, each requiring three CORDIC cells. A CORDIC cell designed as a CMOS integrated circuit requires tens of thousands of transistors, which classifies it as VLSI. Because the three types of CORDIC cells are different in function but similar in structure, we designed only one type of CORDIC cell capable of operating in three different modes. Thus MUSE can be built by using 96 identical VLSI chips. The MUSE system can be realized more efficiently, however, as a larger single chip by using an advanced technology called Restructurable VLSI (RVLSI). Lincoln Laboratory has already used this technology to realize six different systems, so the technology is reasonably mature.

RVLSI comprises a design methodology, a laser-based interconnect modification technology, and a set of CAD tools for building large-area integrated circuits [1, 7–10]. Wafers are fabricated with redundant circuits and interconnects, which are tested after fabrication. A laser is then used to build the desired system by forming and breaking connections to the operable circuits. Partitioning considerations and fabrication yield determine the size of the basic replaceable unit, or *cell;* typically, a cell comprises thousands of transistors. Experience has shown that building a wafer with twice as many cells as ultimately needed strikes a good balance between interconnect overhead and cell yield.

Several laser restructuring technologies have been developed [8]; the technique used in the MUSE application, in which the laser forms a connection between two adjacent diffusions, is completely compatible with standard integrated-circuit processing. The laser-diffused link is used for both signal and power connections. The laser breaks connections by vaporizing a metallization path, and the connections and the metal cuts are made with high yield and appear to be reliable.

Six RVLSI systems have been built on three different wafer designs [1]; the largest design has 405,000 functional transistors. One wafer-scale system (the Integrator) has been operating in a bench tester for over four years without failure. The MUSE system and another system now being built [10] are by far the largest and most logically complex wafer-scale circuits ever manufactured.

Several design choices made the MUSE system even more compatible with the RVLSI technology. First, because of the significant cost of intercell connections, the microcycle was divided into four steps so that data could be moved into and out of a CORDIC cell in four small pieces. Consider a θ cell (as shown in yellow in Figure 8) that must read two 22-bit words from either of two sources (the supercell ahead of it and the supercell behind it) while outputting two 22-bit words to the ϕ cells (as shown in red and blue in Figure 8) of its supercell. Such a θ cell would require 132 connections to at least as many metallization lines, without even counting clocks, control, jumpers, power, and ground. The four-step microcycle allows the same data to be moved with 33 connections and 33 metallization lines.

Second, the two adder-subtractors in each CORDIC stage were replaced by a single adder-subtractor time-shared between two tasks. It first computes its y output

$$y \leftarrow y + 2^i \rho_i x$$

but preserves the old y for the next computation

$$x \leftarrow x - 2^i \rho_i y .$$

The adder-subtractor is the largest part of each CORDIC stage, and is much larger than the control or the pipeline latches. Therefore, a single adder-subtractor reduces the size of the stage. In exchange, however, a minirotation, which requires essentially the time to perform the addition and the subtraction, takes twice as long. Because a CORDIC cell is relatively large (even after this change to a single adder-subtractor), and because a fast adder can be adequately designed, this modification was judged worthwhile.

Third, we decided to provide each CORDIC cell with adequate memory to store the required real or imaginary part of the two columns of the Cholesky matrix, even though the θ-CORDIC cells have no need for this memory. The memory uses only a minor portion of the silicon area of each cell, and the cost of this silicon area must be balanced against the competing cost of providing connections between the ϕ cells and a second type of cell, called *memory,* or, alternatively, against the cost of making two types of CORDIC cells, some with memory and some without. A fabrication fault in the memory portion of a CORDIC cell still leaves it useful as a θ-CORDIC cell.

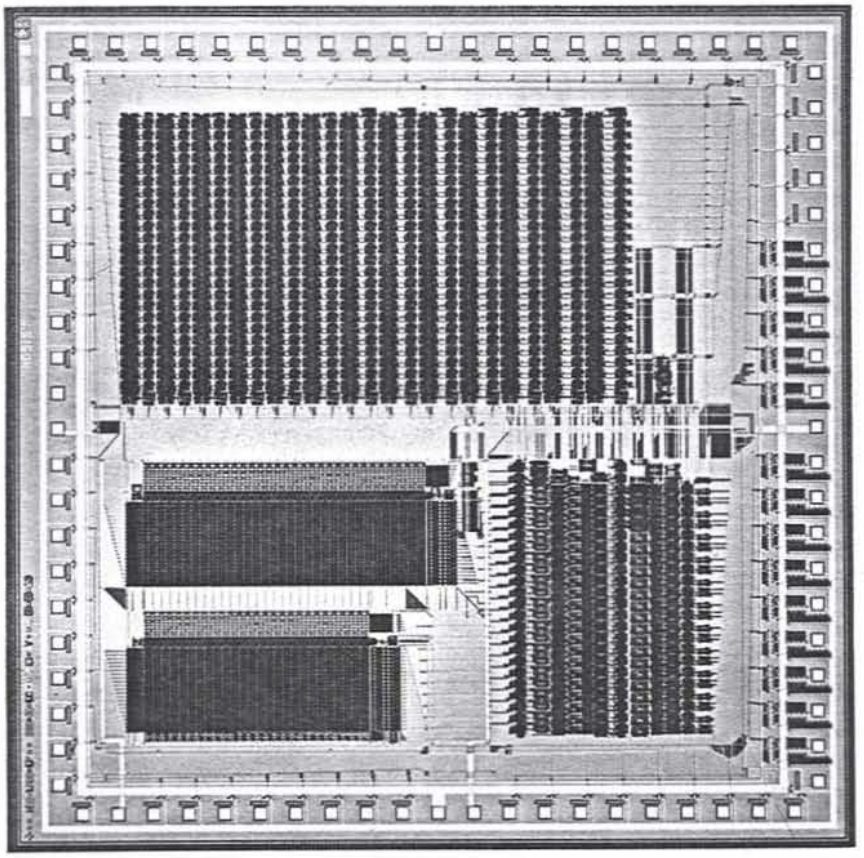

FIGURE 10. Microphotograph of one CORDIC cell and its memory. The cell is 5.5 mm wide and 5.6 mm deep, is fabricated in CMOS with 2-μm design rules, and has approximately 54,000 transistors.

The fourth change is much more complex and is described in the following section. Briefly, to use the Cholesky matrix to solve the linear equations for the nulling weights (after enough samples have updated a Cholesky matrix), we would require data paths to pass the $N(N+1)/2$ matrix elements off the wafer to their next position. Instead we found a way to use the CORDIC cells to begin the process of solving the linear equations. This method produces $2N$ intermediate results, which are the only quantities sent off the wafer, and they are moved on the paths already provided for moving the tacked-on column.

Finally, the CORDIC cells are made in two mirror-image versions and the discretionary metallization paths run between them. The ability to use CORDIC cells on either side of a bundle of discretionary connection tracks gives extra interconnection flexibility. The CORDIC cell shown in Figure 10 is fabricated in 2-μm CMOS and tested at the design speed. It contains 54,000 transistors and is 5.5×5.6 mm^2, excluding output drivers and bond pads. The CORDIC stages are 24 bits wide; 13 stages are required to meet the accuracy criterion described in the previous section on finite-word-length effects. Figure 11 is a photograph of a wafer ready for restructuring. It has 130 CORDIC cells, of which 96 must function correctly to allow us to structure a MUSE system from the single wafer.

Using CORDIC Cells to Solve Linear Equations for Weights

Consider the general $N \times N$ set of linear equations

$$A\mathbf{X} = \mathbf{B}, \tag{6}$$

where A and $\mathbf{B}$ are given and $\mathbf{X}$ is to be determined. To solve the linear equations, we introduce a seemingly unrelated second problem. First, construct an $N \times (N+1)$ matrix by appending the vector $\mathbf{B}$ onto the right edge of A:

$$\left[A|\mathbf{B}\right].$$

Next, postmultiply by an $(N+1)\times(N+1)$ unitary matrix Q that causes the last column of this matrix to become zeroes.

$$\left[A|\mathbf{B}\right]Q = \left[\hat{A}|0\right].$$

Third, partition Q as

$$\left[\begin{array}{c|c} Q_{uu} & \mathbf{Q}_r \\ \hline \mathbf{Q}_l^h & q \end{array}\right],$$

where Q_{uu} is an $N\times N$ matrix. From this partition we obtain the equation

$$A\mathbf{Q}_r + q\mathbf{B} = \mathbf{0},$$

which in turn leads to

$$A\left(\frac{-\mathbf{Q}_r}{q}\right) = \mathbf{B}.$$

This result shows that the solution for $\mathbf{X}$ in Equation 6 is hidden in the elements making up the last column of the unitary matrix Q of the second problem. If we can zero out a column $\mathbf{B}$ tacked on to a matrix A by using a transformation Q, then $A\mathbf{X} = \mathbf{B}$ has the solution $\mathbf{X} = -\mathbf{Q}_r/q$.

The collection of CORDIC cells in the MUSE system performs such a transformation. This fact suggests we can use the same CORDIC cells to solve for the weights $\mathbf{W}$ with the Cholesky matrix as the matrix of coefficients in the linear equations.

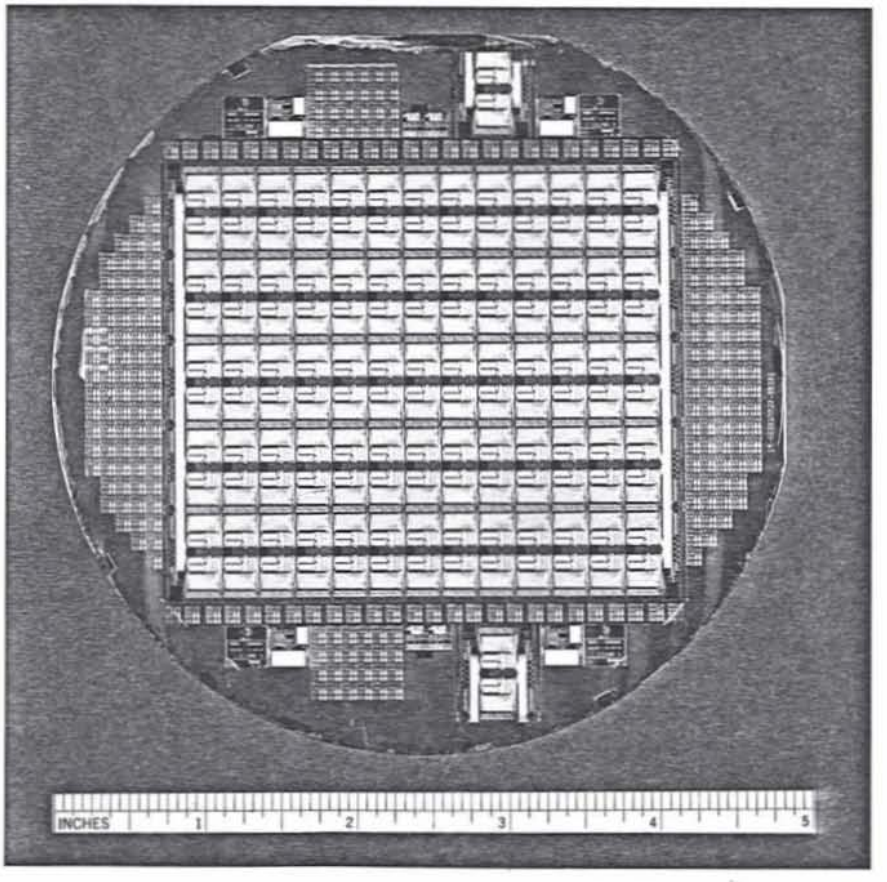

FIGURE 11. Photograph of a 5-in wafer containing 130 CORDIC cells of the type shown in Figure 10. Metallization paths running between cells can be either fused together to make connections or vaporized to break connections. If at least 96 of the cells are functional they can be connected together to form a complete MUSE system.

In the earlier section entitled "Nulling with Givens Transformations" we saw that in general two sets of linear equations must be solved. In the case of a sidelobe canceler (a particular choice of the steering vector, $\mathbf{S} = [0, 0, \ldots, 0, 1]^t$), however, the first set is solved by inspection rather than by computation, so only the second set, that involving L^h, needs to be solved by using the CORDIC cells. This second set of equations is

$$L^h\mathbf{W} = \frac{1}{l_{N,N}}\mathbf{S}.$$

In the above equation we can ignore the scale factor $1/l_{N,N}$, because, as we stated earlier, changing it only scales $\mathbf{W}$ by a constant, which cannot affect the resulting SINR.

In the following discussion, we assume that in each supercell the two columns of the snapshot of L (which we want to solve for $\mathbf{W}$) are stored in a snapshot memory. The process of solving for the weights can therefore be interleaved with the process of updating the Cholesky matrix as more antenna data are fed into MUSE.

To use CORDIC cells to solve $L^h\mathbf{W} = \mathbf{S}$, where L^h is an upper-triangular matrix, we must rewrite the equation set as one involving a lower-triangular matrix with a tacked-on column. To accomplish this transformation we use a reversal matrix J. Premultiplying a vector or matrix by J reverses it top to bottom while postmultiplying by J reverses it left to right. The product JJ is an identity matrix. Now manipulate $L^h\mathbf{W} = \mathbf{S}$ by inserting JJ between L^h and $\mathbf{W}$ and then premultiplying by J to give

$$\left(JL^hJ\right)\left(J\mathbf{W}\right) = \left(J\mathbf{S}\right),$$

and then conjugate the entire equation:

$$\left(JL^tJ\right)\left(J\mathbf{W}\right)^* = \left(J\mathbf{S}\right).$$

This equation has the form of Equation 6 with

$$A = JL^hJ$$
$$\mathbf{X} = \left(J\mathbf{W}\right)^*$$
$$\mathbf{B} = J\mathbf{S},$$

and A is the lower triangular matrix

$$\begin{bmatrix} l_{N,N} & & & & \\ l_{N,N-1} & l_{N-1,N-1} & & & \\ \vdots & \vdots & \ddots & & \\ l_{N,2} & l_{N-1,2} & \cdots & l_{2,2} & \\ l_{N,1} & l_{N-1,1} & \cdots & l_{2,1} & l_{1,1} \end{bmatrix}.$$

Solving for $\mathbf{X}$ immediately gives $\mathbf{W}$ because $\mathbf{W} = J\mathbf{X}^*$. The tacked-on column $\mathbf{B}$ is $[1, 0, 0, \ldots, 0]^t$. Changing $A\mathbf{X} = \mathbf{B}$ into $L^h\mathbf{W} = \mathbf{S}$, an equation with a lower-triangular matrix, is purely an exercise in notation. No actual computations are involved.

Description of the Q-Operation

Two complications were overcome to use the MUSE CORDIC cells to solve for weights. First, the timing was modified slightly, because a system that is 100% efficient cannot otherwise be given an extra task. Two extra idle microcycles were provided for every vector sample, which increased the number of microcycles per vector from 65 to 67. These microcycles were available to be used to solve for weights. The revised latency τ of a supercell, in units of microcycles, must be

$$\tau = \frac{N}{2} + 1.$$

Second, the matrix $A = JL^tJ$ is stored by rows in the snapshot memories of the supercells, rather than by columns. Figure 12 illustrates the pattern of storage and also shows how two elements of the tacked-on column $\mathbf{B}$ (which is initialized as $[1, 0, 0, \ldots, 0]^t$) are assigned to each supercell. We can zero out the tacked-on column $\mathbf{B}$, however, without moving the Cholesky matrix elements. This process is called the *Q-operation*, and is meant to zero out the tacked-on column $\mathbf{B}$ by using CORDIC rotations, in a manner similar to the Cholesky update process illustrated in Figure 2.

The first step of the process uses CORDIC circuits to postmultiply the two columns

$$\begin{bmatrix} l_{N,N} & b_1 \\ l_{N,N-1} & b_2 \\ l_{N,N-2} & b_3 \\ \vdots & \vdots \\ l_{N,2} & b_{N-1} \\ l_{N,1} & b_N \end{bmatrix}$$

by a unitary matrix to cause b_1 to become 0. The CORDIC controls are developed by using the pair $(l_{N,N}, b_1)$ as leaders. This pair is available in supercell 0, so the CORDIC controls are developed there. Because the other pairs are available in other supercells, we pass the CORDIC controls from one supercell to the next, moving the controls first down the chain of supercells and then back up the chain of supercells, rotating each pair in the supercell in which it is available and saving the modified tacked-on column components in that supercell for use in the next step of the Q-operation. Eventually the CORDIC controls are passed off the wafer from supercell 0 and are stored for later use in determining the weights.

As a result of all this activity, the first column of A and the tacked-on column **B** are modified. No further use is made of the first column of A. The tacked-on column **B** is modified so that its first value is zero. The remaining elements of **B** (which were zero) are now nonzero values $[b_2, b_3, \ldots, b_N]^t$ available in supercells $1, 2, \ldots, (N/2)-1, (N/2)-1, \ldots, 2, 1, 0$, respectively, where they were created by the rotations just discussed.

The next step is to postmultiply the second column of A and the modified tacked-on column **B**

$$\begin{bmatrix} l_{N-1,N-1} & b_2 \\ l_{N-1,N-2} & b_3 \\ \vdots & \vdots \\ l_{N-1,2} & b_{N-1} \\ l_{N-1,1} & b_N \end{bmatrix}$$

by a unitary matrix to zero out the next element of **B**. Both elements in the first pair $(l_{N-1,\,N-1}, b_2)$ are available in supercell 1, because b_2 was just computed there. These elements are the leaders used to set up Q-operation rotation controls in supercell 1. These Q-operation controls are moved to supercell 2, where they control the CORDICs acting on the pair $(l_{N-1,\,N-2}, b_3)$, then on to supercell 3, where they control the CORDICs acting on the pair $(l_{N-1,N-3}, b_4)$, and so on. Ultimately, these Q-operation controls also turn around at supercell $(N/2) - 1$, travel back up the chain, and are sent off the wafer at supercell 0. Like the previous set of Q-operation controls, these controls are also stored off the wafer for later use in determining weights. At the end of this step

Supercell 0	$l_{N,N}$							b_1
Supercell 1	$l_{N,N-1}$	$l_{N-1,N-1}$						b_2
Supercell 2	$l_{N,N-2}$	$l_{N-1,N-2}$	$l_{N-2,N-2}$					b_3
	⋮	⋮	⋮	⋱				⋮
Supercell 2	$l_{N,3}$	$l_{N-1,3}$	$l_{N-2,3}$	• • •	$l_{3,3}$			b_{N-2}
Supercell 1	$l_{N,2}$	$l_{N-1,2}$	$l_{N-2,2}$	• • •	$l_{3,2}$	$l_{2,2}$		b_{N-1}
Supercell 0	$l_{N,1}$	$l_{N-1,1}$	$l_{N-2,1}$	• • •	$l_{3,1}$	$l_{2,1}$	$l_{1,1}$	b_N

FIGURE 12. The lower-triangular matrix JL^tJ, where J is a reversal matrix, is a rearrangement of the elements of a snapshot of the Cholesky matrix L. We append the tacked-on column $\mathbf{B} = J\mathbf{S}$, which is obtained by reversing the elements of the steering vector **S**. To solve for the nulling weights, we zero out the elements of **B** by unitary transformations (rotations) carried out by the supercells. This figure shows how the elements making up columns of JL^tJ and the column **B** are distributed among the 32 supercells. Each rotation involves data within one supercell. To use the same rotation on an entire pair of columns, we must transmit the CORDIC controls that effect that rotation from supercell to supercell. The data rotated within any supercell remain in that supercell.

Table 3. Projected Speed of MUSE

Interval	Duration	Rate
Half-Step	$83\frac{1}{3}$ nsec	12 MHz
Step (= 2 Half-Steps)	$166\frac{2}{3}$ nsec	6 MHz
Microcycle (= 2 Steps)	$333\frac{1}{3}$ nsec	3 MHz
Macrocycle (= 67 Microcycles)	$22\frac{1}{3}$ μsec	44.78 kHz
Update Cycle (300 Macrocycles)	6.7 msec	150 Hz

the elements of the now twice-modified tacked-on column $[b_3, b_4, \ldots, b_N]^t$ are available in supercells $2, 3, \ldots, (N/2) - 1, (N/2) - 1, \ldots, 2, 1, 0$, exactly where they will be needed to carry out the third step. (The modified second column of A is no longer needed.)

Proceeding in this way, we perform the entire process of zeroing out the tacked-on column by using CORDIC rotations of data that are never moved from supercell to supercell. The only information that moves during the Q-operation is the Q-operation rotation controls. Because these controls are passed locally to the adjacent supercells, no global communication paths are needed. Each set of Q-operation controls is ultimately passed off the wafer and stored for use in the final phase of weight determination.

In the final phase we determine the elements in the last column of the unitary matrix Q. Although the CORDICs were used to postmultiply $[A|B]$ by Q, the transformation Q was never determined in a standard matrix form. The Q-operation controls that were passed off the wafer are all that is needed to instruct CORDIC cells to carry out the operations that realize a postmultiplication by Q. Therefore, any CORDIC cells with these same controls can postmultiply an identity matrix by Q. This postmultiplication gives us the matrix elements of Q, which we can use directly as the desired weights in our nulling problem.

Summary of Revised MUSE Control and Timing

The fastest clock present in the system is called a *half-step*. A half-step is the time needed to pass eleven bits between CORDIC cells. Two consecutive half-step intervals make a *step* and suffice to move a word. The reciprocal of the step interval is the rate at which new data are presented to an adder within any CORDIC stage. Two steps are called a *microcycle*. A microcycle is the time required to input one complex element into any θ-CORDIC, and its reciprocal is the rate at which new rotations can begin in any CORDIC cell.

The time interval between input of successive 64-element vectors into MUSE is 67 microcycles, called a *macrocycle*. At the input to any θ-CORDIC cell the 67 microcycles are assigned to move data, as described in the section entitled "An Efficient Systolic Array with Latency-Controlled Interleaving," except that a blank microcycle is inserted between each tacked-on column. These blank microcycles are used to move Q-operation controls.

Any macrocycle can be designated as a snapshot cycle. The Q-operation is initiated when the Cholesky factor has been updated to reflect the effect of the vector fed into MUSE during a snapshot cycle. The interval between successive snapshot cycles is called an *update cycle*.

The MUSE timing builds up from the half-step, and the CORDIC cell is designed to operate at a clock speed of 12 MHz. Table 3 summarizes the various clock speeds and periods implied by the 12-MHz half-step clock.

Statistical accuracy of the weights to within about 1 dB of optimal SINR improvement requires about 320 updates [3]. This update rate means that the weights can be updated as often as 140 times per second. MUSE, however, is physically capable of updating its weights much more frequently or less frequently because a snap-

shot cycle is designated by a special external signal.

Because MUSE has 96 CORDIC cells, and each cell performs 3 million new rotations per second, a conventional computer requiring 10 instructions per rotation would have to perform 2.88 billion instructions per second to match the operational speed of MUSE.

Summary and Status

This article describes the design of a wafer-scale adaptive-nulling processor called MUSE, whose individual cells carry out coordinate rotations by using the CORDIC algorithm. The CORDIC cells are almost 100% utilized and communicate with one another in a simple systolic fashion. The high efficiency and modularity of the algorithm is achieved by interleaving two data streams traveling in opposite directions. It also depends on a careful choice of CORDIC latency to avoid collisions.

A CORDIC cell—including memory—requires approximately 54,000 CMOS transistors and occupies a 5.5-mm × 5.6-mm rectangle. It has been clocked at 12 MHz, so that an entire system of 96 such cells can update a 64-element weight vector on the basis of 300 observations in 6.7 msec. A conventional computer would need 2.88 giga-ops to carry out the same task. Simulations have demonstrated that the system can support 50 dB of SINR improvement. A yield analysis has demonstrated that the entire MUSE system can be realized on a single wafer if cell yield averages 70%. Furthermore, the simple systolic communication permits the use of two or more wafers if cell yield is too low.

The main function of the CORDIC cells is to update the Cholesky factor of the correlation matrix of observed interference as new data are observed, with a new update approximately every 22 μsec. The solution of a set of linear equations whose coefficients are a snapshot of the Cholesky factor gives the desired nulling weights. The CORDIC cells are also used, in an interleaved fashion, to solve these linear equations.

We have demonstrated the viability of the technical concept of the MUSE system, although much work still needs to be completed. We first manufactured the CORDIC cell illustrated in Figure 10 as an integrated circuit for ease in testing. We have operated a short chain of 12 such packaged CORDIC cells as an eight-element nulling system to verify all the system aspects of the MUSE architecture, and also to check out a tester and test software, which we will use later to test and operate the wafer-based 64-element nulling system. We fabricated wafers containing 130 CORDIC cells and we have tested cells and interconnection paths on these wafers. Figure 11 shows one such wafer. We are now using a wafer (which lacks sufficient working CORDIC cells to form a complete system) to test software used to control the laser that makes and breaks connections. The final step will be to use that laser-control software on a wafer with at least 96 working cells.

Acknowledgments

The work reported here would have been impossible without the contributions of David Allen, currently with Viewlogic Corporation, who designed the CORDIC chip and simulated numerous aspects of the operation of the system. Other contributions were made by David Glasco, currently at Stanford University, and by Charles Woodward and Alan Anderson.

REFERENCES

1. J. Raffel, A.H. Anderson, and G.H. Chapman, "Laser Restructurable Technology and Design," chap. 7 in *Wafer Scale Integration,* ed. E. Swartzlander (Kluwer Academic Publishers, Boston, 1989), p. 319.
2. M. Monzingo and T. Miller, *Introduction to Adaptive Arrays* (John Wiley, New York, 1980), p. 94.
3. I.S. Reed, J.D. Mallett, and L.E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," *IEEE Trans. Aerospace Electron. Syst.* **10,** 853 (1974).
4. C.R. Ward, P.J. Hargrave, and J.G. McWhirter, "A Novel Algorithm and Architecture for Adaptive Digital Beamforming," *IEEE Trans. Antennas Propag.* **34,** 338 (Mar. 1986).
5. J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electron. Comput.* **8(3),** 330 (1959).
6. G.H. Golub and C.F. Van Loan, *Matrix Computations* (The Johns Hopkins University Press, Baltimore 1983), pp. 24–28.
7. P.W. Wyatt and J.I. Raffel, "Restructurable VLSI—A Demonstrated Wafer Scale Technology," *Proc. Intl. Conf. on Wafer Scale Integration, San Francisco, Jan. 1989,* p. 13.
8. G.H. Chapman, J.M. Canter, and S.S. Cohen, "The Technology of Laser Formed Interconnections for Wafer Scale Integration," *Proc. Intl. Conf. on Wafer Scale Integration, San Francisco, Jan. 1989,* p. 21.
9. R. Frankel, J.J. Hunt, M. Van Alstyne, and G. Young, "SLASH—An RVLSI CAD System," *Proc. Intl. Conf. on Wafer Scale Integration, San Francisco, Jan. 1989,* p. 31.
10. A.H. Anderson, R. Berger, K.H. Konkle, and F.M. Rhodes, "RVLSI Applications and Physical Design," *Proc. Intl. Conf. on Wafer Scale Integration, San Francisco, Jan. 1989,* p. 39.

CHARLES M. RADER
is a senior staff member in the Sensor Processing Technology group. He received a B.E.E. degree and an M.E.E. degree in electrical engineering from the Polytechnic Institute of Brooklyn. His many accomplishments include research on speech bandwidth compression, contributions to the field of digital signal processing, application of optical techniques to educational technology and communication, and investigation of space-based radar systems. From 1971 to 1982 he was an assistant group leader in the Spacecraft Processors group that built the LES-8 and LES-9 communications satellites, which were launched in March 1976. He is a Fellow of the IEEE and past President of the IEEE Acoustics, Speech and Signal Processing (ASSP) Society. In 1976 he received the ASSP Technical Achievement Award and in 1985 he received the ASSP Society Award. He has lectured at several universities, including MIT, University of Utah, the Polytechnic Institute of Brooklyn, Rice University, and University of California at Berkeley. He has also taught courses on advanced digital signal processing in China and Mexico. His books include *Digital Processing of Signals* (coauthored with Ben Gold), *Number Theory in Digital Signal Processing* (coauthored with James McClellan), and *Digital Signal Processing* (coedited with Lawrence Rabiner). He has been at Lincoln Laboratory since 1961.