Fourier Transform Computers Using CORDIC Iterations

ALVIN M. DESPAIN, MEMBER, IEEE

***Abstract*—The CORDIC iteration is applied to several Fourier transform algorithms. The number of operations is found as a function of transform method and radix representation. Using these representations, several hardware configurations are examined for cost, speed, and complexity tradeoffs. A new, especially attractive FFT computer architecture is presented as an example of the utility of this technique. Compensated and modified CORDIC algorithms are also developed.**

***Index Terms*—Algorithms, array processor, computer arithmetic, CORDIC, discrete Fourier transform (DFT), fast Fourier transform (FFT), Fourier transform, function generation, real-time transform, vector rotation.**

I. INTRODUCTION

THE basic discrete Fourier transform (DFT) [1] is defined by

$$A_r = \sum_{k=0}^{N-1} B_k W^{rk} \qquad r = 0,1,2,\cdots,N-1 \tag{1}$$

where

$$W = \exp(-2\pi j/N)$$
$$j = (-1)^{1/2}.$$

This transform may be viewed as a process of rotating the vector B_k (real and imaginary components) by the angle $(2\pi rk/N)$ followed by a sum over all k for each r. A very similar process occurs in the fast Fourier transform (FFT) [1], [2], wherein the repeated application of the following FFT butterfly operation results in the same result as given in (1). The "decimation in frequency" butterfly operation [3] is

$$C' = C + D$$
$$D' = (C - D)W^l$$

where C and D are complex data points, w is defined as above, and l is a function of the location of the butterfly within the butterfly diagram, and the same sum and rotation process is evident.

The vector rotation is generally accomplished by representing the vectors in terms of real and imaginary components and performing four real multiplications and two additions. Golub's method [4] can accomplish the same result with only three real multiplications and five additions, while Buneman's method [5] requires a different treatment but only three multiplications and three additions (see Appendix I). In addition, each of the above methods requires a table of (or must generate) trigonometric coefficients of size $N/4$. In general, since multiplications are much more expensive in terms of computer operations than are additions, these latter two methods generally offer considerable savings. The object of this paper is to examine the CORDIC vector rotation method of Volder [6] for the Fourier transform problem. This appears to be attractive as a vector can be rotated in the time of a single multiply using this technique. In addition, trigonometric coefficients as distinct from the CORDIC constants are not needed in a CORDIC rotation procedure, so the expense of generating, storing, and retrieving these coefficients is avoided.

Manuscript received July 6, 1973; revised March 22, 1974.
The author is with the Department of Electrical Engineering, Utah State University, Logan, Utah.

II. CORDIC TECHNIQUE

Consider the vector $X_i = (x,y)$ to be rotated by the angle $\alpha = 2\pi rk/N$. Let a new vector X_{i+1} be obtained from X_i by the process

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

Then if

$$R_i = x_i^2 + y_i^2$$
$$\theta_i = \tan^{-1}(y_i/x_i)$$

the result will be that the magnitude and angle of the new vector

$$X_{i+1} = (x_{i+1}, y_{i+1})$$

will be

$$R_{i+1} = R_i[1 + 2^{-2i}]^{1/2}$$
$$\theta_{i+1} = \theta_i - \tan^{-1}(2^{-i}).$$

Now to rotate X by an arbitrary angle α (between $\pm\pi/2$) to an accuracy of 1 bit in n bits, the following procedure may be used. (Appendix II has a more general algorithm.) Initialize

$$i = 0$$
$$z_i = -\alpha.$$

Iterate n times

$$a_i = \operatorname{sgn}(z_i)$$
$$x_{i+1} = x_i + a_i y_i 2^{-i}$$
$$y_{i+1} = y_i - a_i x_i 2^{-i}$$
$$z_{i+1} = z_i - a_i \tan^{-1}(2^{-i})$$
$$i = i + 1.$$

These iterations are composed of addition, subtraction, shifting (multiplication by 2^{-i}), and table look-up ($\tan^{-1}(2^{-i})$) operations. Walther [7] discusses the con-

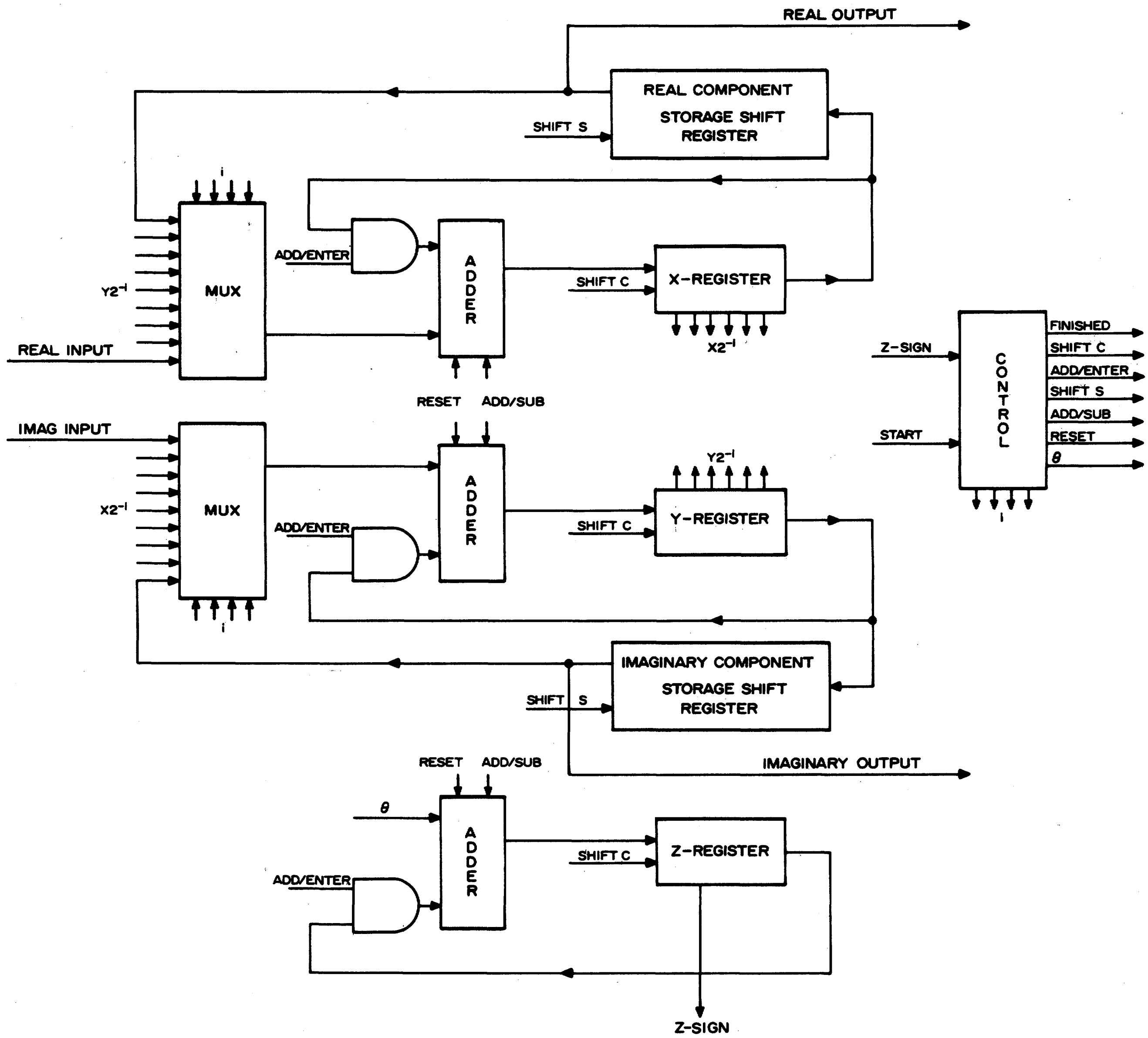

Fig. 1. Serial CORDIC DFT computer configuration.

vergence, truncation errors, and a hardware implementation of this process. If the addition (or subtraction), shifting, and table look-up are accomplished simultaneously, then the vector can be rotated in n addition times or about the time necessary to accomplish a single multiplication using the same technology.

III. APPLICATION TO DFT

The application of this technique to the DFT is now evident. One rotation will be required for each combination of input and output for N^2 total rotations. The real and imaginary components of the input vector B_k are taken as the initial values of x_i and y_i. The initial value of z_i is then set to $2\pi rk/N$. The CORDIC iteration is performed n times to produce n bits of precision and the resulting x and y values are added to the sum that will become A_r after N repetitions of this process. The process is repeated for each frequency component A_r. We also note that the resulting spectrum is scaled by a constant factor K where

$$K = \prod_{i=0}^{n-1} (1 + 2^{-2i})^{1/2} \approx 1.6.$$

The resulting spectrum can be normalized, if required, by dividing by K. The CORDIC algorithm can also be used for this division [7]. Alternatively, a "compensating" CORDIC algorithm [Appendix II] can be employed that causes K to be unity.

A very low-cost Fourier transform computer can be constructed to perform the DFT as described above. It will be most useful for those cases of modest input data rate, where only a few high resolution frequency samples are required for the output. In this case, considerable storage of trigonometric coefficients, unnecessary frequency coefficients and buffers can be saved as compared to an FFT approach. The hardware configuration for such a system is shown in Fig. 1. It should be noted that the only circuitry beyond that needed for the CORDIC rotation is the storage required for the desired output frequency coefficients. If the lowest cost approach (serial data paths and arithmetic circuits) is assumed where f_c

TABLE I
COMPARISON OF ARITHMETIC OPERATIONS REQUIRED FOR BASE 2, BASE 4, BASE 8, AND BASE 16 ALGORITHMS

Algorithm	Required Computation for	Noncompensated Rotations[a]	Compensated Rotations[a]	Modified CORDIC Vector	
				Operations	Additions
Base 2 algorithm for $N = 2^m$	Evaluating $(N/2)m$, 2 term Fourier transforms	0	0	0	mN
$m = 0,1,\cdots,$	Referencing	mN	$(m/2 - 1)N + 1$	mN	0
$l = 1$	Complete analysis	mN	$(96m/192 - 1)N + 1$	$192mN/192$	mN
Base 4 algorithm for $N = (2^2)^{m/2}$,	Evaluating $(N/4)(m/2)$, 4 term Fourier transforms	0	0	0	mN
$m/2 = 0,1,2,\cdots,$	Referencing	$mN/2$	$(3m/8 - 1)N + 1$	$mN/2$	0
$l = 2$	Complete analysis	$mN/2$	$(72m/192 - 1)N + 1$	$96mN/192$	mN
Base 8 algorithm for $N = (2^3)^{m/3}$,	Evaluating $(N/8)(m/3)$, 8 term Fourier transforms	$mN/3$	$mN/12$	$mN/12$	$13mN/12$
$m/3 = 0,1,2,\cdots,$	Referencing	$mN/3$	$(7m/24 - 1)N + 1$	$mN/3$	0
$l = 3$	Complete analysis	$2mN/3$	$(72m/192 - 1)N + 1$	$80mN/192$	$13mN/12$
Base 16 algorithm for $N = (2^4)^{m/4}$	Evaluating $(N/16)(m/4)$, 16 term Fourier transforms	$mN/4$	$mN/8$	$3mN/16$[b]	$19mN/16$
$m/4 = 0,1,2,\cdots,$	Referencing	$mN/4$	$(15m/64 - 1)N + 1$	$mN/4$	0
$l = 4$	Complete analysis	$mN/2$	$(69m/192 - 1)N + 1$	$84mN/192$	$19mN/16$
Base 2^l algorithm for $N = (2^l)^{m/l}$,	Evaluating $(N/2^l)(m/l)$, 2^l term Fourier transforms	$mN(l - 1)/2l$	$\leq 72mN/192 - mN/l + mN/l2^l$	—	$mN+$
$m/l = 0,1,2,\cdots,$	Referencing	mN/l	$[([2^l - 1]/l2^l)m - 1]N + 1$	mN/l	0
l odd	Complete analysis	$mN(l + 1)/2l$		—	$mN+$
Base 2^l algorithm for $N = (2^l)^{m/l}$,	Evaluating $(N/2^l)(m/l)$, 2^l term Fourier transforms	$mN(l - 2)/2l$	$\leq 72mN/192 - mN/l + mN/l2^l$	—	$mN+$
$m/l = 0,1,2,\cdots,$	Referencing	mN/l	$[([2^l - 1]/l2^l)m - 1]N + 1$	mN/l	0
l even	Complete analysis	$mN/2$		—	$mN+$

[a] *Note*: The number of vector (complex) additions is always mN for these cases.
[b] *Note*: The rotation by $\pi/8$ is accomplished by rotating by $\pi/2$ (trivial operation) and then two successive angle halvings, each of which require a vector add and a vector multiply by a scalar.

is the data shift rate employed in the CORDIC circuit ($\approx$20-MHz max for TTL logic), then the input bit rate f_i is limited as

$$f_i < f_c b_i / [\eta_f (b_f + (b_i + 1)(b_i + \log_2 b_i))]$$

where

- b_i the number of bits in each input sample,
- b_f the number of bits in each output frequency sample,
- η_f the number of frequency samples (complex).

Thus in a typical real-time application, we may want

- $b_i = 9$ bits per input time sample,
- $b_f = 16$ bits per output frequency element,
- $\eta_f = 128$ (complex output coefficients).

Then, $f_i < 1 \times 10^4$ bits/s, or less than 1000 input samples per second could be accepted, independent of the transform resolution (hence input transform length). If very high frequency resolution is desired, additional bits may be needed in the registers so that truncation errors will not mask the differences between closely spaced frequency elements.

Other hardware versions of the DFT algorithm are possible including parallel arithmetic and pipelining of either the serial or parallel configuration. A single fully pipelined, parallel arithmetic, CORDIC module could rotate about 2×10^7 vectors/s. Thus if M output frequency samples were required, then the system could accept $2 \times 10^7/M$ samples/s. This pipelined CORDIC DFT appears to be very useful in special applications, but is not competitive with the FFT if a full Fourier transform is required.

IV. APPLICATION TO FFT

The use of the FFT algorithm will always require as much, and often considerably more storage than the DFT approach. However, the FFT approach can produce a very much larger data throughput with the same CORDIC hardware. Many different FFT algorithms have been reported and the CORDIC method of performing the vector rotation can be used with nearly all of them. Bergland [8] has calculated the saving of computational effort that results when various radix FFT algorithms are applied. It is also of considerable interest to determine the number of vector rotations as produced by the CORDIC method that are required to produce a given size FFT and to determine this as a function of the radix of the FFT algorithm employed. The number of rotations for both the compensated CORDIC, the modified CORDIC (see Appendix II) and the simpler noncompensated CORDIC method are derived under the same assumptions employed by Bergland [8] and are expressed in Table I. In every

TABLE II
VECTOR ROTATIONS AND ADDITIONS REQUIRED IN PERFORMING BASE 2, BASE 4, BASE 8, AND BASE 16 FFT ALGORITHMS FOR $N = 4096$ ($m = 12$)

Algorithm	Number of Vector Noncompensated Rotations[a]	Number of Vector Compensated Rotations[a]	Modified CORDIC Vector Operations	Modified CORDIC Vector Additions
Base 2 ($N = (2)^{12}$)	49 152	20 481	49 152	49 152
Base 4 ($N = (2^2)^6$)	24 576	14 337	24 576	49 152
Base 8 ($N = (2^3)^4$)	32 768	14 337	20 480	53 248
Base 16 ($N = (2^4)^3$)	24 576	13 569	21 504	58 368

[a] *Note*: The number of vector additions for these cases is always $mN = 49\ 152$.

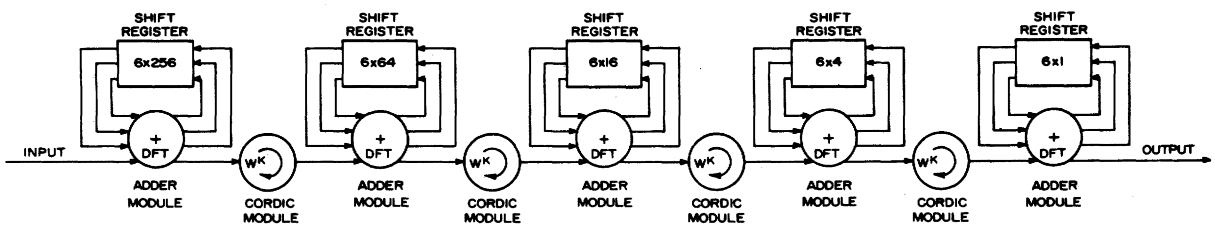

Fig. 2. Fourier transform computer, radix-4 cascade for $N = 1024$.

case compensation must occur. When the noncompensated CORDIC algorithm is applied, compensation is achieved by a dummy rotation of angle zero on those vectors that do not need to be rotated.

It is apparent from Tables I and II that for the noncompensated algorithm the base 4 case is optimal. The compensated algorithm is very near to optimal for base 4 and base 8 but some slight improvement results if a base greater than 8 is used. The modified algorithm uses a vector multiply by a scalar within the base Fourier transform and a noncompensated CORDIC vector rotation for the referencing. It is optimal for base 8. It is now apparent that a "mixed base" algorithm (such as the "4 + 2" algorithm of Gentleman and Sande [9]) of base "4 + 8" will be the best choice when CORDIC techniques are employed.

High performance Fourier transform computers can be constructed around the CORDIC iterations and the FFT algorithm. As an example, Fig. 2 illustrates a novel architecture that is well suited to the radix-4 uncompensated CORDIC technique. Either the "decimation in time" or the "decimation in frequency" base 4 algorithm can be employed in this configuration. The "decimation in time" algorithm is an attractive choice as the system can be used to process a single channel (complex) of transform length N, or 4^l ($l = 0,1,2,\cdots$) independent channels of transform length $N/4^l$ in parallel, without modifying the configuration. This tradeoff is described by Groginsky and Works [2] for the radix-2 Cooley–Tukey algorithm and the exact same technique applies to the radix-4 algorithm. Gold and Bially [3] discuss the "decimation in frequency" radix-4 algorithm but their hardware configuration, although similar in appearance, is fundamentally distinct from that presented here. The desired algorithm is easily derived from the Cooley–Tukey algorithm but can best be described by a procedure similar to that of Gold and Bially [3].

Step 1: Arrange the input data into a two-dimensional array (4 by $N/4$ for the radix-4 case).

Step 2: Multiply each element of the resulting matrix by W_l^{pq}.

Step 3: Perform a DFT (4 by 4) on each row individually.

Step 4: Transform the resultant matrix columns. This is accomplished by beginning at Step 1 and repeating the procedure for each column individually and recursively.

Step 5: The entire procedure is followed until the resulting matrices become square and the last column is processed.

This procedure results in a flow diagram (Fig. 3) similar to Fig. 3 of Gold and Bially [3] where the arrows now represent the four-point "decimation in time" DFT

$$\begin{aligned} A' &= AW^0 + BW^k + CW^{2k} + DW^{3k} \\ B' &= AW^0 + jBW^k - CW^{2k} - jDW^{3k} \\ C' &= AW^0 - BW^k + CW^{2k} - DW^{3k} \\ D' &= AW^0 - jBW^k - CW^{2k} + jDW^{3k} \end{aligned} \tag{2}$$

where p and q are the row and column indices, $j = (-1)^{1/2}$ and k is the algorithm-position dependent index above each arrow, and the numbers refer to the nodes of the corresponding "butterfly" diagram (not shown). We now note that k may be derived from a simple binary counter, incrementated at the serial word rate by masking and bit reversal, so that various sections of the counter represent the output frequency, channel number, and rotation angle (in bit-reversed form). Thus the control mechanism has the very simple form of Groginsky and Works [10], wherein data (possibly multichannel as in sonar and radar signal processing) are transformed in order of arrival with no modification of the control or arithmetic circuits when multichannel operation is desired.

The operation of the system is begun by forcing each

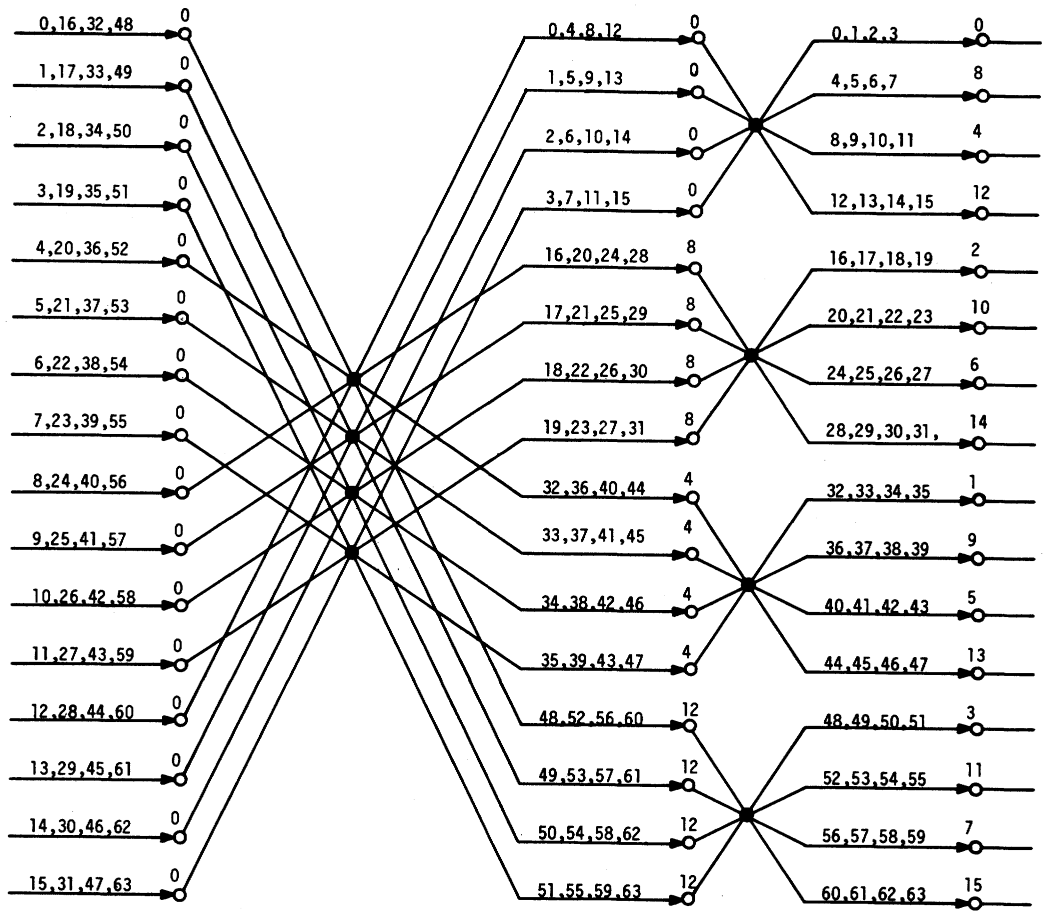

Fig. 3. 64-point FFT, radix-4 "decimation in time" flow diagram.

DFT adder module to pass input data through until its shift register is filled. That is, in the pass mode the module produces:

$$A' = A$$
$$B' = B$$
$$C' = C$$
$$D' = D$$

until the shift register is filled. Then the module is put into the add mode and the following DFT is produced:

$$A' = A + B + C + D$$
$$B' = A + jB - D - jD$$
$$C' = A - B + C - D$$
$$D' = A - jB - C + jD$$

where the indicated complex operations are simple data transpositions. After the last D vector is accepted, the adder module reverts back to pass mode. The CORDIC module rotates every complex vector it receives by the angle k supplied in sign encoded form from the control circuit. First the A' vectors are rotated while the B', C', and D' vectors of the previous stage are reentered into the shift register. The A' vectors are then followed by the B', C', and D' vectors as the output of the CORDIC module is passed to the next stage of processing. As in the DFT case the output is scaled. It may be normalized by dividing each output by K^4.

The system of Fig. 2 might be described as a "radix-4, FFT, CORDIC, cascade computer" and can be implemented with either serial or parallel data paths and with either combinational or pipelined arithmetic circuits. (Davidson and Larson [11], [12] have a good discussion of pipelined Fourier transform computers, and cost effective configurations.) The least expensive version of this system (serial TTL logic and MOS shift register storage) would process about 10^4 input samples per second while a pipelined parallel arithmetic version would process about 10^7 samples/second. In all cases the transform size is determined by the number of stages employed in the cascade. Five stages for a transform size of 1024 complex points are illustrated in Fig. 2.

V. VERY HIGH PERFORMANCE FFT PROCESSORS

The CORDIC technique has its greatest advantage in a FFT array configuration [13] (referred to by Larson and Davidson [12] as a "FFT parallel pipelined cascade"). This configuration can process about 10^7 transforms/second [12], independent of the transform size. If the radix-4 algorithm using uncompensated CORDIC iterations is applied, then from Table I, it is apparent that $(N/2)\log_2(N)$ CORDIC modules are required to produce a transform of N complex points, where one-fourth of those are "dummy" in the sense that they do not produce a vector rotation, but only a vector scaling (for compensation). Since Buneman's method [5] requires three full multiplications to perform the same rotation function as well as additional additions, the CORDIC rotation technique will always require less hardware to implement than any of the other methods (one CORDIC module has about the same part count as two full multipliers in the array case). Larson and Davidson [12] have a very good discussion of these types of FFT configurations for the radix-2 FFT algorithms and their pipelining techniques may be directly adapted to the "4 + 8" FFT CORDIC algorithm presented above.

VI. CONCLUSIONS

It appears that the CORDIC Fourier transform techniques are most useful at opposite ends of the computer power spectrum. If a very low-cost and hence low-data rate machine is needed, then the single CORDIC module with serial arithmetic is very inexpensive. The absence of trigonometric tables and auxiliary storage is often a great advantage in this case. It is not clear however, that the CORDIC method has any advantage for moderate data rates where low-cost parallel multiply circuits can perform vector rotations very inexpensively, and storage costs for the trigonometric coefficients are neglectable. For high performance systems, the CORDIC system is attractive because trigonometric coefficients need not be fetched from a relatively slow store. In the very high performance systems (array machines) the base-4 CORDIC method always requires less hardware.

APPENDIX I
VECTOR ROTATION METHODS

Let the vector $X = [x + y(-1)^{1/2}]$ be rotated by the angle θ.

Standard Method:

$$x' = x \cos\theta - y \sin\theta$$

$$y' = y \cos\theta + x \sin\theta.$$

Four multiplies and two additions are required.

Golub's Method:

$$x' = [(x + y)(\cos\theta - \sin\theta) + x \sin\theta - y \cos\theta]$$

$$y' = [y \cos\theta + x \sin\theta].$$

Three multiplies and five additions are required.

Buneman's Method:

$$x' = (1 + \cos\theta)(x - y \tan\theta/2) - x$$

$$y' = (1 + \cos\theta)(x - y \tan\theta/2) \tan\theta/2 + y.$$

Three multiplies and three additions are required as $(1 + \cos\theta)$ and $\tan\theta/2$ are stored constants.

APPENDIX II
COMPENSATED CORDIC ALGORITHMS

Introduction

The CORDIC technique for rotating a vector expressed in rectangular form as developed by Volder [6], has enjoyed considerable success in arithmetic operations [7] and special purpose function generation (as exemplified in the Hewlett-Packard HP-35 calculator) [14]. The technique is also very useful in Fourier transform computers although the change of magnitude that occurs as the vector is rotated often limits the advantages of the CORDIC technique. For example, if it is employed in calculating the FFT "butterfly" [1], either an additional multiply, divide, or CORDIC operation is required to normalize the rotated vector to its original magnitude. Several methods of avoiding this change in magnitude are herein developed.

CORDIC Technique

The CORDIC technique [6] is an iterative process similar to ordinary nonrestoring arithmetic division. Let X be the real component and Y be the imaginary component of a vector to be rotated by an angle θ.

The initialization is the following:

$$\text{if } \theta \geq 0 \quad \text{then } \alpha = +1,$$

$$\text{if } \theta < 0 \quad \text{then } \alpha = -1,$$

$$X_1 = +\alpha Y$$

$$Y_1 = -\alpha X$$

$$\theta_1 = \alpha\pi/2 - \theta$$

$$K_1 = 1.$$

Then for each iteration step i, $i = 1,2,\cdots,N$:

$$\text{if } \theta_i \geq 0 \quad \text{then } a_i = +1,$$

$$\text{if } \theta_i < 0 \quad \text{then } a_i = -1,$$

$$X_{i+1} = X_i + a_i Y_i 2^{-i+1}$$

$$Y_{i+1} = Y_i - a_i X_i 2^{-i+1}$$

$$\theta_{i+1} = \theta_i - a_i \tan^{-1}(2^{-i+1})$$

$$K_{i+1} = K_i(1 + 2^{-2i+2})^{1/2}$$

where

- X_i the real component of the vector,
- Y_i the imaginary component of the vector,
- K_i the magnitude of the vector,
- θ_i the angle of the vector,
- i the iteration index.

To rotate the vector $(X + jY)$ by the angle θ, the iteration is performed N times where N is the number of bits used to represent X, Y, and θ. Walther [7] has discussed the convergence and truncation errors of this process.

The primary difficulty in using this method is that the magnitude of the vector K_i grows with each iteration step. If a constant number of iterations (generally N) occur each time, then the vector is always scaled by the same factor (K).

Compensated CORDIC Method

The compensated CORDIC technique employs an iterative process such that at each CORDIC iteration after the first few iterations, the magnitude multiplier coefficient K_i is forced by a simple correction to approach unity. The result is that the magnitude approaches unity in the same manner that the angle approaches zero; that is, it is always within one significant bit of its correct value for that stage of iteration (1 bit out of i bits at the ith step). Truncation, if allowed, will cause some errors in the magnitude just as it does in the angle calculation. However, the guard digits usually employed for the angle [7] are also sufficient for the magnitude correction.

The compensation and rotation can be accomplished by a two-step process as the basic CORDIC rotation:

$$\begin{aligned}
&\text{if } \theta_i \geq 0 \quad \text{then } a_i = +1,\\
&\text{if } \theta_i < 0 \quad \text{then } a_i = -1,\\
&X_{i+1}' = X_i + a_i Y_i 2^{-i+1}\\
&Y_{i+1}' = Y_i - a_i X_1 2^{-i+1}\\
&\theta_{i+1}' = \theta_i - a_i \tan^{-1}(2^{-i+1})\\
&K_{i+1}' = K_i(1 + 2^{-2i+2})^{1/2}
\end{aligned}$$

followed by a magnitude correction:

$$\begin{aligned}
&\text{if } K_i \geq 0 \quad \text{then } b_i = -1,\\
&\text{if } K_i < 0 \quad \text{then } b_i = +1,\\
&\text{if } \ i = 1 \quad \text{then } b_i = 0,\\
&X_{i+1} = X_{i+1}' + b_i X_{i+1}' 2^{-i+1}\\
&Y_{i+1} = Y_{i+1}' + b_i Y_{i+1}' 2^{-i+1}\\
&\theta_{i+1} = \theta_{i+1}'\\
&K_{i+1} = K_{i+1}'(1 + b_i 2^{-i+2} + b_i^2 2^{-2i+2})^{1/2}
\end{aligned}$$

where $\{b_i\} = \{0,-1,1,-1,1,1,1,-1,1,1,-1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,1\}$ for its first 24 values. The same results can also be obtained in a single iteration as

$$X_{i+1} = X_i + a_i Y_i 2^{-i+1} + b_i X_i 2^{-i+1} + a_i b_i Y_i 2^{-2i+2}$$
$$Y_{i+1} = Y_i - a_i X_i 2^{-i+1} + b_i Y_i 2^{-i+1} - a_i b_i X_i 2^{-2i+2}.$$

These operations are considerably more complex than the corresponding ones in the uncompensated case. This may not be a serious disadvantage if serial arithmetic is used since the arithmetic portion of most serial computers is generally only a small fraction of the total system. In an effort to simplify the above calculation however, the following iteration may be considered:

$$X_{i+1} = X_i + a_i Y_i 2^{-i+1} + b_i X_i 2^{-i+1}$$
$$Y_{i+1} = Y_i - a_i X_i 2^{-i+1} + b_i Y_i 2^{-i+1}.$$

This is somewhat less complex but the vector rotation angle will change due to the truncation of the terms containing 2^{-2i+2}. Thus convergence of the angle sequence toward zero is no longer assured and must be reexamined.

The angle at any step is easily determined as

$$\theta_{i+1} = \tan^{-1}\left[\frac{Y_{i+1}}{X_{i+1}}\right] = \tan^{-1}\left[\frac{Y_i - a_i X_i 2^{-i+1} + b_i Y_i 2^{-i+1}}{X_i + a_i Y_i 2^{-i+1} + b_i X_i 2^{-i+1}}\right]$$
$$\theta_{i+1} = \theta_i - a_i \operatorname{cnt}^{-1}[b_i + 2^{i-1}]$$

while the magnitude scaling factor becomes

$$K_{i+1}^2 = [K_i^2/(X_i^2 + Y_i^2)][X_{i+1}^2 + Y_{i+1}^2]$$
$$K_{i+1} = K_i[1 + a_i^2 2^{-2i+2} + b_i 2^{-i+2} + b_i^2 2^{-2i+2}]^{1/2}.$$

In general, convergence of θ_i toward zero requires [2] that

$$\theta_i \leq \theta_{n-1} + \sum_{k=i+1}^{n-1} \theta_k$$

for all $i < n$ where n is the total number of steps. Since unity can be expanded as

$$1 = 2^{i-(n-1)} + \sum_{k=i+1}^{n-1} 2^{i-k}, \qquad \text{any } i$$

then

$$\theta_i = \theta_i 2^{i-(n-1)} + \sum_{k=i+1}^{n-1} \theta_i 2^{i-k}$$

and

$$\theta_i \leq \theta_{n-1}\gamma_{n-1} + \sum_{k=i+1}^{n-1} \theta_k \gamma_k$$

where

$$\gamma_k = (\theta_i/\theta_k) 2^{i-k}.$$

Then convergence will occur if $\gamma_k \leq 1$ since

$$\gamma_k = (\theta_i/\theta_k) 2^{i-k}$$

and $k > i$, then $\gamma_k \leq 1$ if

$$2\theta_{i+1} \geq \theta_i \qquad \text{for all } i.$$

This is more restrictive than our original convergence criteria, but will produce some useful results. Since

$$\tan(2\theta) = 2\tan\theta/[1 - \tan^2\theta]$$

the restrictive convergence criteria can be written as

$$2\tan\theta_{i+1} \geq \tan\theta_i[1 - \tan^2\theta_{i+1}]$$
$$2(b_{i+1} + 2^i)^{-1} \geq (b_i + 2^{i-1})^{-1}[1 - (b_{i+1} + 2^i)^{-2}]$$
$$2(b_{i+1} + 2^i)(b^i + 2^{i-1}) \geq (b_{i+1} + 2^i)^2 - 1$$
$$(2b_i - b_{i+1})2^i + 2b_i b_{i+1} - b_{i+1}^2 + 1 \geq 0.$$

For the special case $b_i = b_{i+1}$, then

TABLE III
COMPENSATED CORDIC ALGORITHM CALCULATION

STEP NUMBER k	SHIFT FACTOR $-i+1$	COMPENSATION COEFICIENT b_k	VECTOR ROTATION (RADIANS)	ANGLE CONVERGENCE ERROR	MAGNITUDE SCALING K_{k+1}	MAGNITUDE ERROR $(K_{k+1}-1)2^{k-1}$
1	0	0	0.7853981634D 00	0.0000000000D 00	0.7071067812D 00	-0.2928931000E 00
2	1	0	0.4636476090D 00	0.0000000000D 00	0.7905694150D 00	-0.4188611000E 00
3	2	0	0.2449786631D 00	0.0000000000D 00	0.8149003007D 00	-0.7403987000E 00
4	3	1	0.1106572212D 00	0.0000000000D 00	0.9224045089D 00	-0.6207638000E 00
5	4	1	0.5875582272D-01	0.0000000000D 00	0.9817489230D 00	-0.2920172000E 00
6	5	0	0.3123983343D-01	0.0000000000D 00	0.9822281756D 00	-0.5686983000E 00
7	6	1	0.1538340178D-01	0.0000000000D 00	0.9976935402D 00	-0.1476134000E 00
8	6	0	0.1562372862D-01	0.0000000000D 00	0.9978153215D 00	-0.1398193000E 00
9	7	0	0.7812341060D-02	0.0000000000D 00	0.9978457720D 00	-0.2757411000E 00
10	8	0	0.3906230132D-02	0.0000000000D 00	0.9978533849D 00	-0.5495334000E 00
11	9	1	0.1949315270D-02	0.0000000000D 00	0.9998042168D 00	-0.1002409000E 00
12	10	0	0.9765621896D-03	0.0000000000D 00	0.9998046936D 00	-0.1999937000E 00
13	11	0	0.4882812112D-03	0.0000000000D 00	0.9998048127D 00	-0.3997434000E 00
14	12	0	0.2441406201D-03	0.0000000000D 00	0.9998048425D 00	-0.7993649000E 00
15	13	1	0.1220554126D-03	0.0000000000D 00	0.9999268965D 00	-0.5988640000E 00
16	14	1	0.6103143111D-04	0.0000000000D 00	0.9999879290D 00	-0.1977706000E 00
17	15	0	0.3051757812D-04	0.0000000000D 00	0.9999879295D 00	-0.3955260000E 00
18	16	0	0.1525878906D-04	0.0000000000D 00	0.9999879296D 00	-0.7910445000E 00
19	17	1	0.7629336324D-05	0.0000000000D 00	0.9999955589D 00	-0.5820972000E 00
20	18	1	0.3814682714D-05	0.0000000000D 00	0.9999993736D 00	-0.1641972000E 00
21	18	0	0.3814697266D-05	-0.2424571719D-12	0.9999993736D 00	-0.1641952000E 00
22	19	0	0.1907348633D-05	-0.2424710497D-12	0.9999993736D 00	-0.3283896000E 00
23	20	0	0.9536743164D-06	-0.2424727844D-12	0.9999993736D 00	-0.6567788000E 00
24	21	1	0.4768369308D-06	0.0000000000D 00	0.9999998505D 00	-0.3135579000E 00
25	22	0	0.2384185791D-06	-0.1509946124D-13	0.9999998505D 00	-0.6271159000E 00
26	23	1	0.1192092753D-06	0.0000000000D 00	0.9999999697D 00	-0.2542319000E 00
27	24	0	0.5960464478D-07	-0.8886120162D-15	0.9999999697D 00	-0.5084638000E 00
28	25	1	0.2980232150D-07	0.0000000000D 00	0.9999999995D 00	-0.1692774000E-01
29	26	0	0.1490116119D-07	-0.4336800418D-18	0.9999999995D 00	-0.3385549000E-01
30	27	0	0.7450580597D-08	-0.4336800418D-18	0.9999999995D 00	-0.6771099000E-01
31	28	0	0.3725290298D-08	-0.4336808690D-18	0.9999999995D 00	-0.1354219000E 00
32	29	0	0.1862645149D-08	-0.4336808690D-18	0.9999999995D 00	-0.2708439000E 00
33	30	0	0.9313225746D-09	-0.4336808690D-18	0.9999999995D 00	-0.5416879000E 00
34	31	1	0.4656612871D-09	0.0000000000D 00	0.1000000000D 01	-0.8337646000E-01

$$b_i \geq -(2^{i+1})^{-1/2}.$$

Thus for the uncompensated case ($b_i = 0$) convergence is verified. Also convergence occurs if $b_i = 1$ but not if $b_i = -1$. Thus it appears unattractive to choose $b_i = -1$ for any i. However, if the magnitude is to be forced to unity, then b_i cannot be constant at either 0 or $+1$. Further if $b_i = 0$ and $b_{i+1} = 1$ at any step i, then convergence is not assured by the restricted criteria (although $b_i = 1$ and $b_{i+1} = 0$ are acceptable). Thus the series of b_i's can change from one to zero only once as i is incremented if convergence is to occur or some iterations must be repeated so that convergence will occur. This repetition of an iteration to ensure convergence is employed by Walther [7] in the CORDIC calculation of the hyperbolic functions. Clearly such repetitions must be employed here and the number of such repetitions should be minimized.

The algorithm given below was found by a heuristic search (computer program) that employed the general convergence criteria as a boundary condition and attempted to minimize the number of operations required to assure both angle convergence to zero and magnitude convergence to unity for each step.

Compensated CORDIC Algorithm

Initialization:

$$\text{if } \theta \geq 0 \quad \text{then } \alpha = +1/2,$$
$$\text{if } \theta < 0 \quad \text{then } \alpha = -1/2,$$
$$X_1 = -\alpha Y$$
$$Y_1 = +\alpha X$$
$$\theta_1 = \alpha\pi - \theta$$
$$K_1 = 1/2$$
$$i = 0.$$

Iteration:

$$k = 1,2,\cdots N+2; \qquad N+2 \leq 34$$
$$i = i+1,$$
$$\text{if } k = 8 \text{ or if } k = 21, \qquad \text{then } i = i-1,$$
$$\text{if } \theta_k \geq 0 \qquad \text{then } a_k = +1,$$
$$\text{if } \theta_k < 0 \qquad \text{then } a_k = -1,$$
$$X_{k+1} = X_k + b_k X_k 2^{-i+1} + a_k Y_k 2^{-i+1}$$
$$Y_{k+1} = Y_k + b_k Y_i 2^{-i+1} - a_k X_k 2^{-i+1}$$
$$\theta_{k+1} = \theta_k - a_k \operatorname{cnt}^{-1}[b_k + 2^{i-1}]$$
$$K_{k+1} = K_k[1 + 2^{-2i+2} + b_k^2 2^{-2i+2} + b_k 2^{-i+2}]^{1/2} \cong 1$$

where $\{b_k\}$ is given in Table III. The net result is a vector rotation by θ without a significant change in the magnitude of the vector.

Table III illustrates the performance of the algorithm up to 32 significant bits for which two repetitions are required.

As mentioned by Walther [7], to achieve a final accuracy of 1 bit in N bits, an additional $\log_2 N$ guard digits should be employed in the arithmetic operations.

Modified CORDIC Method

Occasions arise when it would be convenient to multiply the magnitude of the vector $X + jY$ by a scalar constant. This is easily accomplished by setting $a_i = 0$ in the above algorithm and encoding the scalar constant K in a suitable form.

Thus the modified algorithm for $0 < K < 2$ is the following.

Initialization:

$$X_2 = X$$
$$Y_2 = Y$$
$$K_2 = 1$$
$$i = 2.$$

Iteration:

$$\text{if } (K_i - K) \geq 0 \qquad \text{then } b_i = -1,$$
$$\text{if } (K_i - K) < 0 \qquad \text{then } b_i = +1,$$
$$X_{i+1} = X_i + b_i X_i 2^{-i+1}$$
$$Y_{i+1} = Y_i + b_i Y_i 2^{-i+1}$$
$$K_{i+1} = K_i(1 + 2^{-2i+2} + b_i 2^{-i+2})$$
$$i = i + 1.$$

As in the compensating algorithm, the b_i's for any given constant K may be precalculated. This algorithm is primarily useful for the special case of multiplying a vector $X + jY$ by a predetermined scalar. A more general vector multiply algorithm could be easily developed from the CORDIC scalar multiply algorithm [7], but would require additional variables beyond these used here. One use of the modified algorithm is in Fourier transform computers where many multiplications of a vector by a fixed constant are sometimes required.

Conclusions

Both the compensated and the modified CORDIC algorithms are useful in special purpose applications such as Fourier transform computers. The compensated algorithm may be very useful in those computers that require multiple operand addition capability for other reasons (to calculate effective addresses for example). In these cases, the additional hardware to implement the compensated algorithm as opposed to the uncompensated algorithm may be very valuable.

ACKNOWLEDGMENT

The author would like to thank Prof. A. M. Peterson for his encouragement and support and Stanford University for its hospitality.

REFERENCES

[1] W. T. Cochran *et al.*, "What is the fast Fourier transform?" *Proc. IEEE*, vol. 55, pp. 1664–1674, Oct. 1967.
[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.
[3] B. Gold and T. Bially, "Parallelism in fast Fourier transform hardware," *IEEE Trans. Audio Electroacoust.*, vol. AU-21, pp. 5–16, Feb. 1973.
[4] R. C. Singleton, "An algorithm for computing the mixed radix fast Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. AU-15, pp. 45–55, June 1967. (Golub's method is given in a footnote.)
[5] O. Buneman, "Inversion of the Helmholtz (or Laplace–Poisson) operator for slab geometry," Inst. for Plasma Res., Stanford Univ., Stanford, Calif., SUIPR Rep. 467, p. 5, Apr. 1972.
[6] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.
[7] J. S. Walther, "A unified algorithm for elementary functions," in *1971 Spring Joint Comput. Conf., AFIPS Proc.*, vol. 38. Montvale, N. J.: AFIPS Press, 1971, pp. 379–385.
[8] G. D. Bergland, "A fast Fourier transform algorithm using base 8 iterations," *Math. Comput.*, vol. 22, pp. 275–279, Apr. 1968.
[9] W. M. Gentleman and G. Sande, "Fast Fourier transforms for fun and profit," in *1966 Fall Joint Comput. Conf., AFIPS Proc.*, vol. 29. Washington, D. C.: Spartan, 1966, pp. 563–578.
[10] H. L. Groginsky and G. A. Works, "A pipeline fast Fourier transform," *IEEE Trans. Comput.*, vol. C-19, pp. 1015–1019, Nov. 1970.
[11] E. S. Davidson and A. G. Larson, "Pipelining and parallelism in cost-effective processor design," submitted to *IEEE Trans. Comput.*
[12] A. G. Larson and E. S. Davidson, "Cost-effective design of fast Fourier transform processors," submitted to *IEEE Trans. Comput.*
[13] G. D. Bergland, "Fast Fourier transform hardware implementations—An overview," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 104–108, June 1969.
[14] D. S. Cochran, "Algorithms and accuracy in the HP-35," *Hewlett-Packard J.*, pp. 10–11, June 1972.

Alvin M. Despain (S'58–M'65) was born on July 2, 1938. He received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Utah, Salt Lake City, in 1960, 1964, and 1966, respectively.

In 1957 and 1958 he was an Engineer Trainee at the Southern California Edison Company. From 1958 to 1960 he was a Research Assistant and then a Research Engineer from 1960 to 1966 for the Upper Air Research Laboratory, University of Utah. In 1966 he served as an Assistant Professor of Research at the University of Utah, and from 1966 to 1969 he served as Assistant Professor at Utah State University, Logan. In 1967 he was Assistant Director and later in 1971 an Associate Director for the Electro-Dynamics Laboratories, Utah State University. He has also been associated with Stanford University, Stanford, Calif., as a Visiting Associate Professor and is presently an Associate Professor in Electrical Engineering at Utah State University. His interests include research and teaching in computer architecture, design languages, and the concurrent development of algorithms and hardware for high-performance digital systems. He is the author of several published papers.

Dr. Despain is a member Tau Beta Pi, Sigma Xi, Sigma Pi Sigma, Eta Kappa Nu, the American Society for Engineering Education, and the American Association of University Professors. In 1970 he received the Outstanding Young Faculty Award from the American Society for Engineering Education. Also in 1966 he received the Sigma Xi award for Outstanding Research. He served as Chairman of the Utah Chapter of the IEEE Computer Society form 1971 to 1972, Vice President of the Utah State University Chapter of the American Society for Engineering Education from 1971 to 1972, and also Vice President of the Utah Chapter of Sigma Pi Sigma from 1961 to 1962.