THE MODIFIED CORDIC ALGORITHM*

by

Asif Naseem+, Student Member, IEEE

and

P. David Fisher, Senior Member, IEEE
Department of Electrical Engineering and Systems Science
Michigan State University
East Lansing, Michigan 48824

ABSTRACT

A Modified CORDIC Algorithm (MCA) has been developed for the evaluation of elementary arithmetic functions. MCA incorporates increased parallelism over the original CORDIC algorithm, thus, resulting in an enhanced speed of computation. This has been accomplished by decoupling the CORDIC interation equations and transforming the sequential nature of these equations. n-bit fixed point data operands are assumed and the parameter k relates to the level of parallelism in the algorithm. The modified algorithm employs a $[k+1]n^2$-bit ROM for lookup tables that enable elementary arithmetic functions to be evluated in no more than [3n+2] and no less than 2 time steps. The two bounds correspond to a pipelined and a parallel implementation, respectively. The formulation of the MCA can be manipulated to obtain implementations with various speed/cost characteristics. This compares to n(3n+1)/2 time steps for the original CORDIC algorithm. For example, 32-bit ALU has k=12 and n=32; so, a 13,312-bit ROM is required to store the lookup tables, and the computation requires 98 time steps for a pipelined implementation.

I. INTRODUCTION

Circuit design criteria has changed because of VLSI. We must now re-examine older algorithms to see if they can be improved and matched with architectures that meet their special data-flow requirements in the light of the current IC technology. With this motivation, the CORDIC algorithm [19] has been re-visited to see if more parallelism can be embeded in it. In this paper, we present the Modified Cordic Algorithm (MCA). It incorporates increased parallelism, thereby resulting in an enhanced speed of computation. A matching architecture for the MCA has also been developed and is presented here.

The concept of CORDIC computing technique was first presented in 1956 by Volder [19]. Later, in 1971 Walther [20] unified the algorithms developed by Volder and showed that these algorithms can be described by a single set of iterative equations. The basis of Walther's algorithm is coordinate rotation in the three coordinate systems. He showed that by rotating a vector in the linear, the circular and the hyperbolic coordinate systems, a wide variety of elementary arithmetic functions can be evaluated. One attractive feature of the CORDIC algorithm is that it may be implemented in hardware using adders and shift registers. Here, we relax the requirement that hardware costs be minimized in order to increase the speed of execution of algorithms on the CORDIC computer.

The CORDIC iteration equations in the circular coordinate system are:

$$X_{i+1} = X_i + b_i 2^{-i} Y_i \tag{1a}$$

$$Y_{i+1} = Y_i - b_i 2^{-i} X_i \tag{1b}$$

$$Z_{i+1} = Z_i - b_i \tan^{-1} 2^{-i}, \tag{1c}$$

where

$$b_i = \operatorname{sgn} Z_i.$$

It is important to note that X and Y are coupled in the sense that X_{i+1} can not be computed until X_i and Y_i are known and Y_{i+1} can not be computed until Y_i and X_i are known. Furthermore, the evaluation of the b_i's is sequential. Thus, as a result, the algorithm is sequential. Also, there is only one way to rotate a vector through a given angle and the number of iterations is equal to the word length, n, of the machine. The algorithm must complete all n iterations even if the rotation through the desired angle has been performed exactly in fewer than n iterations [5]. We address here, the problem of transforming the sequential CORDIC algorithm into a more parallel algorithm. It is shown that X and Y can be decoupled if the b_i's can be predicted before the computation of X and Y. The constraint that the b_i's can take up values only from the set [-1,1], has been relaxed and an algorithm to compute all the b_i's in parallel is developed. The theory for this new algorithm is presented more fully elsewhere [14,15] and is only outlined here.

In the next section we decouple the CORDIC iteration equations. Next, an algorithm for predicting B is presented. And, finally, a Modified CORDIC Algorithm (MCA) computational module is described along with its timing characteristics.

*This research was supported in part by NSF under Grant No. MCS 79-09216.

+Current address is AT&T Bell Laboratories; 1100 East Warrenville Road; Naperville, Illinois 60566.

II. DECOUPLING X AND Y

Equation (1c) is not a linear recursion in the sense that Z_{i+1} can not be expressed as Z = FZ + G. However, b_i and Z_i can be evaluated independently of X_i and Y_i. Once all the b_i's and Z_i's have been evaluated, Equations (1a) and (1b) become linear recursions and X and Y can be expressed as

$$X = AX + C,$$

and

$$Y = DY + E.$$

From (1a) and (1b), general expressions for X_i and Y_i are:

$$X_i = X_i + \sum_{j=1}^{i-1} b_j 2^{-j} Y_j. \tag{2}$$

$$Y_i = Y_1 + \sum_{k-1}^{i-1} b_k 2^{-k} X_k. \tag{3}$$

It can be shown that (2) and (3) can be expressed as follows [14]:

$$X_i = \xi_i - \sum_{j=1}^{i-2} a_{ij} X_j; \tag{4}$$

$$Y_i = \delta_i - \sum_{j=1}^{1-2} d_{ij} Y_j; \tag{5}$$

where

$$\xi_i = X_i + Y_1 \sum_{j=1}^{i-1} b_j 2^{-j}, \tag{6}$$

$$a_{ij} = b_j 2^{-j} \sum_{1=j+1}^{i-1} b_1 2^{-1}, \tag{7}$$

$$\delta_i = Y_1 - X_1 \sum_{k=1}^{i-1} b_k 2^{-k}, \tag{8}$$

$$d_{ij} = b_j 2^{-j} \sum_{1=j+1}^{i-1} b_1 2^{-1}. \tag{9}$$

Equations (4) and (5) may be concisely expressed as

$$X = C - AX, \tag{10}$$

$$Y = E - DY, \tag{11}$$

where A and D are strictly lower-triangular matrices with diagonal elements and all elements immediately below the diagonal equalling zero.

Solving Equations (10) and (11) for X and Y yields

$$LX = C, \tag{12}$$

$$MY = E, \tag{13}$$

where

$$L = A + I,$$

and

$$M = D + I$$

and I is an nxn identity matrix.

Equations (12) and (13) are key results and have the following important characteristics:

1. X and Y have been completely decoupled and can ben evaluated independently of each other, whereas the X and Y in the original algorithm are coupled (Equations (1a) and (1b).
2. The form of Equations (12) and (13) is such that they lend themselves to be implemented on a highly concurrent architecture, whereas the form of Equations (1a) and (1b) dictates a sequential evaluation process.

However, before X and Y of Equations (12) and (13) can be evaluated using a parallel architecture, the matrices L and M and the vectors C and D must be known which in turn require that all the b_i's be known. The determination of the b_i's is discussed in the next section.

III. PREDICITNG B

It was shown in the last section the Equations (1a) and (1b) become linear recursions once all the b_1's are known; also, it was shown that the b_i's and Z_i's can be evaluated independently of X_i and Y_i. Equation (1c) can be used to iterate Z_i and b_i until all the b_i's are known forming the vector B. Unfortunately, this is a highly sequential procedure and the length of the process is directly proportional to n, the size of the vector B. A parallel algorithm for the prediction of the vector B is required to overcome this timing problem. One such algorithm is provided below.

The basis of this algorithm is given in Table I. The first column lists the index variable i, which runs from zero through n-1. For the case illustrated in Table I, n is 32. The second column in the table lists the values of 2^{-i}. The arctan of 2^{-i} are listed in the third column. The fourth column contains the difference of the corresponding entries in the second and third columns. These differences are termed the OFFSETs. The entries in the second and the third columns are listed under the headings α_i and γ_i, respectively. We will refer to γ_i's as the 'CORDIC angles' due to the fact that these are the fundamental angles used in vector rotation in the CORDIC algorithm.

I	α_i	γ_i	$OFFSET_i$
0	+057.454544067382	+045.095985412597	+012.358557701110
1	+028.727272033691	+026.638664245605	+002.088606834411
2	+014.363636016845	+014.075137138366	+000.288498431444
3	+007.181818008422	+007.144759654998	+000.037058424949
4	+003.590909004211	+003.586244344711	+000.004664675891
5	+001.795454502105	+001.794870376586	+000.000584101140
6	+000.897727251052	+000.897654235363	+000.000073039400
7	+000.448863625526	+000.448854506015	+000.000009123235
8	+000.224431812763	+000.224430680274	+000.000001137060
9	+000.112215906381	+000.112215764820	+000.000000140460
10	+000.056107956171	+000.056107938289	+000.000000016721
11	+000.028053978085	+000.028053975105	+000.000000001672
12	+000.014026989042	+000.014026989042	+000.000000000000
13	+000.007013494372	+000.007013494372	+000.000000000000
14	+000.003506747186	+000.003506747186	+000.000000000000
15	+000.001753373622	+000.001753373622	+000.000000000000
16	+000.000876686811	+000.000876686811	+000.000000000000
17	+000.000438343405	+000.000438343405	+000.000000000000
18	+000.000219171702	+000.000219171702	+000.000000000000
19	+000.000109585851	+000.000109585851	+000.000000000000
20	+000.000054792928	+000.000054792928	+000.000000000000
21	+000.000027396464	+000.000027396464	+000.000000000000
22	+000.000013698232	+000.000013698232	+000.000000000000
23	+000.000006849116	+000.000006849116	+000.000000000000
24	+000.000003424558	+000.000003424558	+000.000000000000
25	+000.000001712279	+000.000001712279	+000.000000000000
26	+000.000000856139	+000.000000856139	+000.000000000000
27	+000.000000428069	+000.000000428069	+000.000000000000
28	+000.000000214034	+000.000000214034	+000.000000000000
29	+000.000000107017	+000.000000107017	+000.000000000000
30	+000.000000053508	+000.000000053508	+000.000000000000
31	+000.000000026754	+000.000000026754	+000.000000000000

Table I. 2^{-I}, $\mathrm{Tan}^{-1}2^{-i}$ and the Offsets Computed Using 32-bit Fixed-Point Arithmetic

V(1)	V(2)	V(3)	V(4)	V(5)	V(6)	V(7)	V(8)	V(9)	V(10)	V(11)	V(12)
+1	+1	+1	0	0	0	0	0	0	0	0	0
-1	-1	-1	0	0	0	0	0	+1	0	0	0
-1	-1	-1	0	+1	+1	+1	0	-1	0	0	0
+1	-1	-1	0	-1	-1	-1	0	-1	0	0	+1
+1	+1	+1	+1	-1	-1	-1	0	-1	0	0	-1
-1	+1	-1	-1	-1	-1	-1	0	-1	0	0	-1
-1	-1	+1	-1	-1	-1	-1	0	-1	0	0	-1
-1	+1	+1	-1	-1	-1	-1	0	+1	0	0	-1
+1	-1	-1	-1	-1	-1	-1	0	+1	0	+1	-1
+1	+1	+1	-1	+1	+1	+1	0	+1	0	-1	-1
+1	+1	+1	-1	-1	-1	-1	0	+1	0	-1	-1
-1	-1	-1	-1	-1	-1	-1	0	+1	0	-1	-1
+1	+1	-1	+1	-1	-1	-1	+1	-1	+1	-1	+1
+1	-1	+1	-1	+1	+1	+1	-1	+1	-1	-1	-1
+1	-1	+1	+1	+1	+1	+1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	-1	-1	-1	+1	-1	-1	+1
-1	+1	-1	+1	+1	-1	-1	-1	-1	-1	-1	-1
+1	+1	-1	-1	-1	+1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	-1	-1	+1	-1	+1	-1	-1	-1
-1	+1	+1	+1	-1	+1	+1	-1	+1	-1	-1	-1
+1	+1	+1	-1	-1	+1	-1	-1	+1	-1	-1	+1
-1	+1	+1	+1	-1	-1	-1	-1	+1	-1	-1	+1
-1	-1	+1	-1	+1	-1	+1	-1	+1	-1	-1	-1
+1	+1	-1	+1	-1	+1	+1	-1	-1	-1	-1	-1
+1	-1	+1	-1	+1	-1	+1	+1	+1	-1	-1	-1
-1	-1	-1	-1	+1	-1	+1	-1	+1	-1	-1	-1
0	-1	+1	+1	+1	-1	+1	-1	-1	-1	-1	-1
0	-1	+1	-1	+1	+1	-1	-1	-1	-1	+1	+1
0	0	-1	-1	+1	-1	+1	+1	+1	-1	-1	-1
0	0	+1	0	-1	+1	-1	-1	-1	-1	-1	-1
0	0	0	0	+1	+1	-1	+1	-1	+1	+1	+1
0	0	0	0	+1	+1	-1	-1	-1	-1	-1	+1

Figure 1. Coefficient matrix for the offsets of Table I.

Notice that α_i and γ_i are in radians whereas the entries in Table I have been converted into degrees. The following observations are made from Table I.

1. γ_i is always contained in α_i i.e., $(\alpha_i - \gamma_i) \geq 0$, for all i;
2. $\alpha_{i+1} = 1/2\alpha_i$, for all i;
3. $\gamma_{i+1} = \gamma_i$, for $i \geq 12$;
4. $\alpha_i = \gamma_i$, for $i \geq 12$;
5. $OFFSET_i = 0$, for $i \geq 12$.

Now let us consider an angle whose binary representation is of the form:

$$\phi = \phi_0 \phi_1 \phi_2 \phi_3 \cdot \cdot \cdot \cdot \phi_n.$$

The value of ϕ, given in degrees, is

$$\phi = \sum_{i=1}^{n} \phi_i \alpha_i$$

$$= \sum_{i=1}^{n} \phi_i \left[\gamma_i + OFFSET_i\right], \qquad (14)$$

where α_i is the angle contribution to ϕ by the ith bit of ϕ, i.e., ϕ_i. Our goal is to express ϕ as a sum of the CORDIC angles (γ_i's).

In Table I, only twelve of the OFFSETS are nonzero. For generality, let k be the number of nonzero OFFSETs, where k is less than n. Each of the k OFFSETs can be expressed as a finite sum of the CORDIC angles,

$$OFFSET_i = \sum_{j=1}^{n} v_{ji} \gamma_j \qquad (15)$$

The v_{ji}'s in (15) determine the sign of γ_j and constitute the elements of a coefficient vector V(i) associated with $OFFSET_i$. Since there are k nonzero offsets, there are k coefficient vectors, V(1) through V(k), associated with $OFFSET_i$ through $OFFSET_k$, respectively. We will refer to the matrix formed by the k Coefficient Vectors as the Coefficient Matrix. The Coefficient Matrix is of the dimension nxk, where n is the work length of the machine. A detailed algorithm for generating this Coefficient Matrix is presented elsewhere [14]. This algorithm was used to generate the Coefficient Matrix illustrated in Figure 1. Substituting Equation (15) in (14),

$$\phi = \sum_{j=1}^{n} \phi_i \left[\gamma_i + \sum_{j=1}^{n} v_{ji} \gamma_j\right]. \qquad (16)$$

Since only k of the OFFSETs are nonzero, we may express Equation (16) as follows [14].

$$\phi = \sum_{i=1}^{n} b_i \gamma_i, \qquad (17)$$

where

$$b_i = \phi_i + \sum_{j=1}^{k} \phi_j v_{ij}. \qquad (18)$$

In vector form, Equation (17) can be written as

$$\phi = B\gamma, \tag{19}$$

where

$$B = [b_1 b_2 b_3 \ . \ . \ . \ b_n]^T,$$

and

$$\gamma = [\gamma_1 \gamma_2 \gamma_3 \ . \ . \ . \ \gamma_n]^T.$$

And, so, we have developed a means for concurrently generating each element in B for a given angle ϕ. Thus, we have achieved our goal of expressing the given angle ϕ in terms of the CORDIC angles.

The three vectors, α, γ and OFFSET are always constant for a given n-bit machine. Also, all three vectors are independent of the angle ϕ and, thus, are known constants for a given machine. It is to be noted that the vector OFFSET does not directly enter the computation of the vector B; hence, it need not be stored in the machine's memory. Furthermore, the elements of the vector appear only in the product terms in the algorithm. The multiplication by 2^{-i} means simply shifting the multiplicand i-bit places to the right. This shifting can be implemented by a shifter/scaler with multiple-bit shift capability, thus, eliminating the need for storing the vector in the machine's memory. The only vector that actually enters into the computation of the vector B is ; it can be stored in a read only memory (ROM) look up table. As stated earlier, all the OFFSETS are constant for a given n-bit machine and are independent of the angle ϕ. It follows that all the coefficient vectors, V(i)'s, are also constants, which can be stored in a ROM look up table. The total memory space, M, taken up by the look up tables is

$$M = [k + 1]n, \tag{20}$$

where M is taken to be an n-bit word. For example, a 32-bit ALU that employs the MCA algorithm would require 13,312 bits for these lookup tables since n=32 and k=12.

The most significant feature of the preceding algorithm is that all the bits of the angle ϕ can be operated upon in parallel. The angle contribution due to each bit of ϕ in terms of the CORDIC angles can be computed in parallel and, consequently, all the b_i's can be evaluated concurrently.

An example computation of the vector B for a given angle is presented next. Consider a 32-bit angle ϕ whose binary representation is given as

$$\phi = \cdot\phi_1 \ \phi_2 \ . \ . \ . \ \phi_{32}. \tag{21}$$

The ith element of the vector B is computed using Equation (18). All the elements of the vector B are computed in parallel. Let θ represent the value of the angle computed using the vector B. Then θ is given by

$$\theta = \sum_{i=1}^{32} b_i \cdot \gamma_i. \tag{22}$$

How many of the twelve OFFSETS are considered actually significant depends upon the accuracy

θ		0	1	0	1	1	1	0	1	1	1	0	0	B
	γ_i	COEFFICIENT MATRIX												
0	+045.095985412597	+1	+1	+1	0	0	0	0	0	0	0	0	0	+ 1
1	+026.638664245605	-1	-1	-1	0	0	0	0	0	+1	0	0	0	+ 1
0	+014.075137138366	-1	-1	-1	0	+1	+1	+1	0	-1	0	0	0	+ 0
1	+007.144759654998	+1	-1	-1	0	-1	-1	-1	0	-1	0	0	+1	- 3
1	+003.582244344711	+1	+1	+1	+1	-1	-1	-1	0	-1	0	0	-1	+ 0
1	+001.794870376586	-1	+1	-1	-1	-1	-1	-1	0	-1	0	0	-1	- 2
0	+000.897654235363	-1	-1	+1	-1	-1	-1	-1	0	-1	0	0	-1	- 3
1	+000.448854506015	-1	+1	+1	-1	-1	-1	-1	0	+1	0	0	-1	+ 0
1	+000.224430680274	+1	-1	-1	-1	-1	-1	-1	0	+1	0	+1	-1	- 2
1	+000.112215764820	+1	+1	+1	-1	+1	+1	+1	0	+1	0	-1	-1	+ 4
0	+000.056107958289	+1	+1	+1	-1	-1	-1	-1	0	+1	0	-1	-1	- 1
0	+000.028053975105	-1	-1	-1	-1	-1	-1	-1	0	+1	0	-1	-1	- 3
0	+000.014026989042	+1	+1	-1	+1	-1	-1	-1	+1	-1	+1	-1	+1	+ 1
0	+000.007013494372	+1	-1	+1	-1	+1	+1	+1	-1	+1	-1	-1	-1	- 1
0	+000.003506747186	+1	-1	+1	+1	+1	+1	+1	-1	-1	-1	-1	-1	- 1
0	+000.001753373622	-1	-1	+1	+1	+1	-1	-1	-1	+1	-1	-1	+1	- 1
1	+000.000876686811	-1	+1	-1	+1	+1	-1	-1	-1	-1	-1	-1	-1	+ 0
1	+000.000438343405	+1	+1	-1	-1	-1	+1	-1	-1	-1	-1	-1	-1	- 2
1	+000.000219171702	-1	-1	+1	+1	-1	-1	+1	-1	+1	-1	-1	-1	- 2
1	+000.000109585851	-1	+1	+1	+1	-1	+1	+1	-1	+1	-1	-1	-1	+ 2
1	+000.000054792928	+1	+1	+1	-1	-1	+1	-1	-1	+1	-1	-1	+1	+ 0
1	+000.000027396464	-1	+1	+1	+1	-1	-1	-1	-1	+1	-1	-1	+1	+ 0
1	+000.000013698232	-1	-1	+1	-1	+1	-1	+1	-1	+1	-1	-1	-1	- 2
1	+000.000006849116	+1	+1	-1	+1	-1	+1	+1	-1	-1	-1	-1	-1	+ 0
1	+000.000003424558	+1	-1	+1	-1	+1	-1	+1	+1	+1	-1	-1	-1	+ 0
1	+000.000001712279	-1	-1	-1	-1	+1	-1	+1	-1	+1	-1	-1	-1	- 2
1	+000.000000856139	0	-1	+1	+1	+1	-1	-1	+1	-1	-1	-1	-1	+ 0
1	+000.000000428069	0	-1	+1	-1	+1	+1	-1	-1	-1	-1	+1	+1	- 2
1	+000.000000214034	0	0	-1	-1	+1	-1	+1	+1	+1	-1	-1	-1	+ 1
1	+000.000000107017	0	0	+1	0	-1	+1	-1	-1	-1	-1	-1	-1	- 2
1	+000.000000053508	0	0	0	0	+1	+1	-1	+1	-1	+1	+1	+1	+ 4
1	+000.000000026754	0	0	0	0	+1	+1	-1	-1	-1	-1	-1	+1	+ 0

Figure 2. An illustrative example, where θ = +042.082717895507, ϕ = +042.082717895507, ERROR = +000.000000000000, and K = 12.

requirements imposed on the algorithm. If the accuracy requirements are relaxed such that the smaller values of the OFFSETS have negligible effect on the accuracy of the final results, then fewer than twelve OFFSETS are significant. Therefore, fewer coefficient vectors would contribute to the vector B, thus reducing the computational complexity and, consequently, reducing the cost of the hardware required to implement the algorithm. An illustrative example of computation of the vector B and the angle θ for k=12, using 32-bit fixed-point arithmetic, is presented in Figure 2. The bits of the given angle ϕ are listed in the first column of Figure 2. The second column lists the CORDIC angles. The twelve most significant bits of ϕ are listed in the top row. Listed beneath each of these twelve bits, are the corresponding coefficient vectors. The last column lists the resulting vector B. The value of ϕ and θ in degrees and the error are also listed in Figure 2, where

$$\text{ERROR} = \phi - \theta. \tag{23}$$

IV. THE MCA COMPUTING MODULE

This section deals with the implementation of the Modified CORDIC Algorithm. The mathematical model of the MCA can be manipulated in a variety of ways to obtain representations that would yield hardware implementations with various speed/cost characteristics. At one end of the speed/cost tradeoff spectrum is a pipelined implementation offering the advantage of low cost realization, whereas at the other end is the high-speed parallel implementation. Implementations with moderate speed/cost characteristics can also be

obtained through manipulation of the mathematical formulation of the MCA [14].

A PIPELINED IMPLEMENTATION

As such, the formulation of the MCA can be used to obtain a pipelined hardware implementation. One such implementation is presented here and involves realization of the vectors B, X, Y and Z. The hardware design for the realization of each of these vectors is presented below.

IMPLEMENTATION OF B

Recall that the vector B is given by the following:

$$B = [b_1\ b_2\ b_3\ \ldots\ b_n]^T,$$

where

$$b_i = \phi_i + \sum_{j=i}^{k} \phi_j V_{ij}. \qquad (24)$$

Examination of the formulation of the MCA reveals that the elements of the vector B appear in the computations either as b_i or $2^{-i}b_1$. Therefore, instead of employing a more expensive general purpose i-bit shifter to generate $2^{-i}b_i$ sequentially, it would be cheaper and faster to generate $2^{-i}b_i$ sequentially, it would be cheaper and faster to generate $2^{-i}b_i$ simultaneously with b_i in the same hardware. The hardware structure for the generation of b_i along with b_i^* is shown in Figure 3, where

$$b_i^* = 2^{-i}b_i. \qquad (25)$$

The circles on the cross-points represent a logical AND function.

IMPLEMENTATION OF X

In order to find a hardware realization for X, we are faced with designing a hardware structure that would implement the linear system of equations expressed by Equation (12). We must have the vector C and the matrix L before we can realize Equation (12). The elements of the vector C, ξ_i, are given by Equation (6). A pipelined hardware realization of the ξ_i's is given in Figure 4, which employs a two-operand adder and an Inner-Product-Step Processor (IPSP). The elements of the matrix A, a_{ij}, are given by Equation (7). A pipelined implementation employing two-operand adders and inner-product-step processors for the computation of the matrix L is given in Figure 5. Feedback loops are incorporated to accumulate the successive values. It is to be noted that the elements of the matrix L are not computed row-wise or column-wise but are computed with a particular skew. The advantage of this will become clear shortly.

Now we are ready to realize the linear system of Equations (12). Mead and Conway [13] have proposed a VLSI architecture employing inner-product step processors for the solution of the linear system of equations of the type given in Equation (12). This structure is shown in Figure 6.

IMPLEMENTATION OF Y

A comparison of Equations (8), (9) and (13) with Equations (6), (7) and (12) shows immediately that the computation of Y can be carried out in hardware similar to that required by the computation of X.

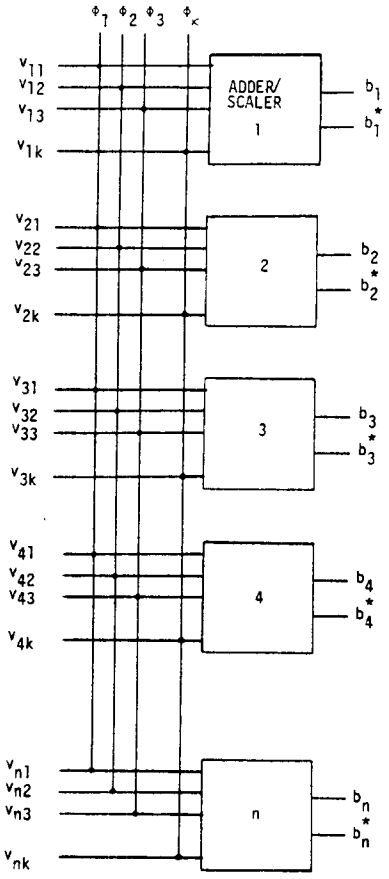

Figure 3. Functional realization of Vector B.

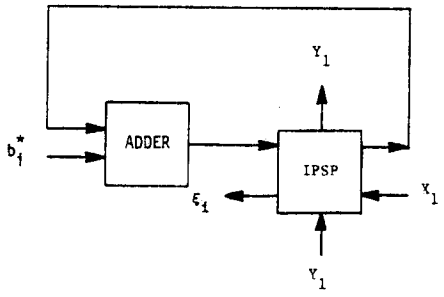

Figure 4. A pipelined realization of ξ_i.

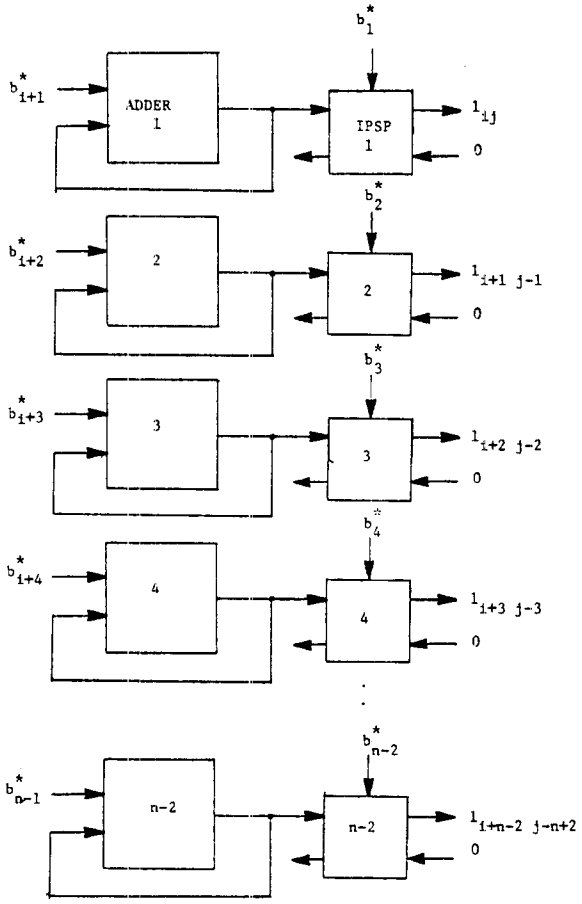

Figure 5. A pipelined realization of the matrix L.

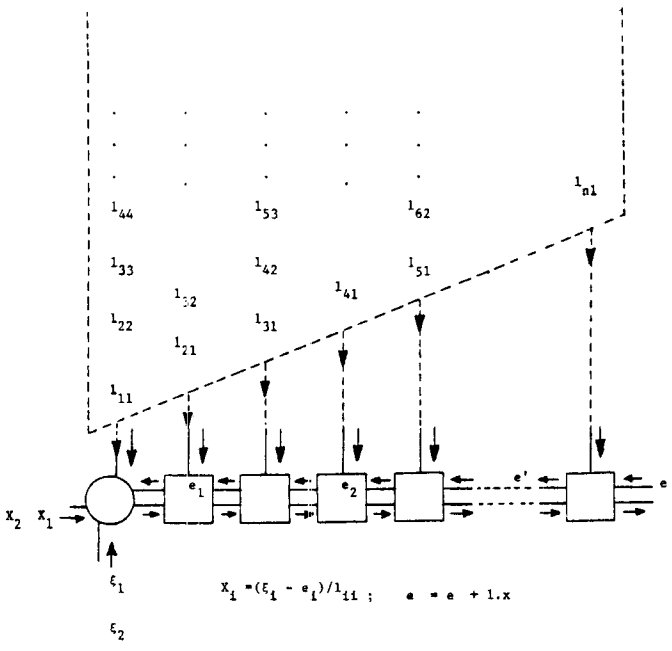

Figure 6. Realization of the linear system LX = C.

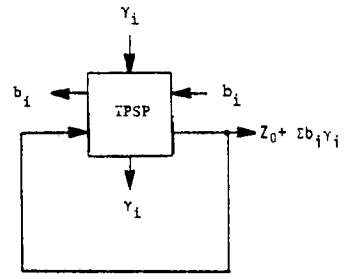

Figure 7. Realization of Z.

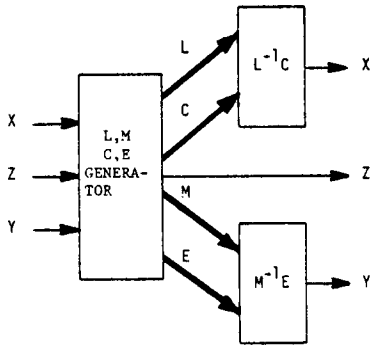

Figure 8. CORDIC module.

IMPLEMENTATION OF Z

The pipelined realization of z is straightforward. This is the only instance in which the elements of the vector B appear as b_i rather than b_i^*. This angle accumulation may be carried out in one inner-product step processor as shown in Figure 7.

The mathematical modeling of the MCA presented here is done for the circular coordinate system only. The extension of the algorithm to linear and hyperbolic coordinate systems is presented elsewhere [14]. The resulting CORDIC module is shown in Figure 8. The initial values of X, Y and Z serve as input to the block that generates the matrices L and M and the vectors C and E, which in turn are processed by two linear arrays of processors of Figure 6. The final values of X, Y and Z are shown as outputs of the module. A key result to be noted here is that since X and Y have been decoupled, it is not necessary to compute Y while computing X or X while cmputing Y. And, since the computation of X and Y require a similar hardware structure, in low cost applications it may be desirble not to duplicate this hardware but rather to compute either X or Y at a time depending upon the type of function desired at the output of the module. The input-output function block diagram of the MCA module is shown in Figure 9.

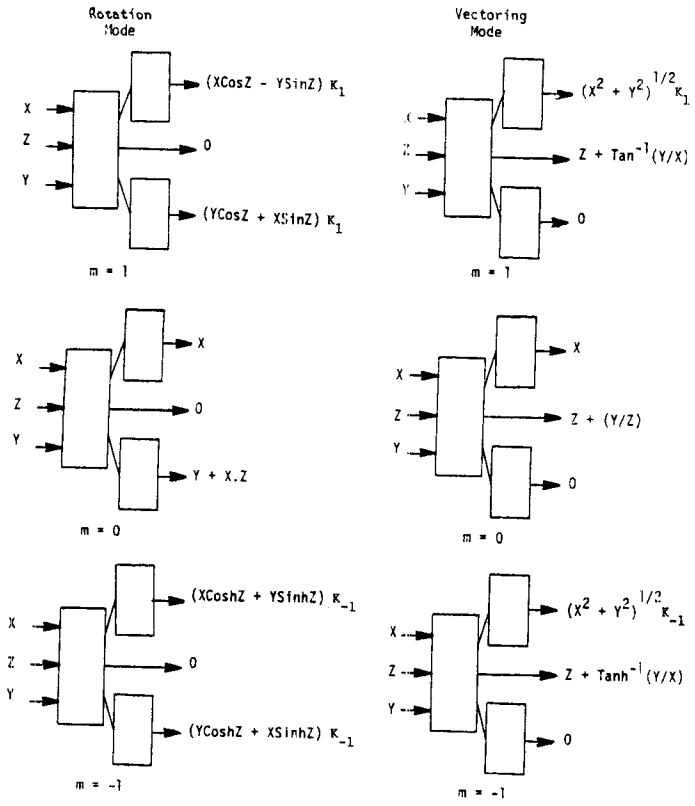

Figure 9. Block diagram of the input output functions of the two CORDIC modes.

Time Step	B	L	M	ξ	δ	LX - C	MY - E	Z
1	b_1–b_n							
2		l_{11}	m_{11}	ξ_1	δ_1			z_1
3		l_{21}	m_{21}					z_2
4		l_{22},l_{31}	m_{22},m_{31}	ξ_2	δ_2			z_3
5		l_{32},l_{41}	m_{32},m_{41}					z_4
6		l_{33},l_{42},l_{51}	m_{33},m_{42},m_{51}	ξ_3	δ_3			z_5
7 | n+1		l_{ij} & m_{ij} are computed with a skew shown in Figure 6						
n+2						X_1	Y_1	z_n
n+3 | 3n+1								
3n+2						X_n	Y_n	

Table II. Timing of the Pipelined CORDIC Module

n	CORDIC	MODIFIED CORDIC (PIPELINED)
8	100	26
16	392	50
32	1552	98

Table III. Total Number of Time Steps Required to Compute a Function Using the Two Algorithms

V. SYSTEM TIMING

The system timing of the pipelined implementation of the module based on the MCA is discussed in this section. A comparison of the timing characteristics of the pipelined MCA module and the implementation of the original CORDIC algorithm are also presented in this section.

Table II depicts the timing of the pipelined MCA module. The first column lists the time steps, while the corresponding activities in various sections of the module are listed in eight different columns. The eight sections are indicated at the top of each column according to the function performed therein. For example, the columns under the headings of B, L and M list the activities performed in the sections that compute the vector B, the matrix L and the matrix M respectively. The crosses in the boxes indicate that the corresponding section is active during that time step. It is clear from the table that, for an n-bit machine, 3n+2 time steps are required to obtain the final result. The table also shows that the latency of the pipe is n. After the pipe is filled, successive results emerge from the pipe every alternate time step. Thus, the total number of time stps, N_T, required by the MCA module to compute a function is

$$N_T = 3n + 2. \tag{25}$$

The timing characteristics of the implementation of the original CORDIC algorithm will be analyzed next. The number of additional steps required in an n-bit machine is n^2. The number of time steps, N, needed for i-bit shifting of the operands required at the ith step is

$$N = \sum_{i=1}^{n} (n-1) = n(n+1)/2.$$

Thus, the total number of time steps, N_T, required for the computation of a function is

$$N_T = n^2 + \sum_{i=1}^{n} (n-i) = n(3n + 1)/2. \tag{26}$$

The total number of time steps required by the two algorithms for values of n=8, 16 and 32 are listed in Table III. It should be emphasized here that the entries in Table III are presented for compar-

n	AREA OF THE CORDIC MODULE
8	$\sim 2.9 \times 10^4 \lambda^2$
16	$\sim 2.2 \times 10^5 \lambda^2$
32	$\sim 1.6 \times 10^6 \lambda^2$

Table IV. Area of the Pipelined CORDIC Module for Different Word Lengths

ing the total number of time steps required by the two algorithms to evaluate a given function, and that the two time step sizes associated with CORDIC and MCA, respectively, are not necessarily equal. This is to be expected due to the increased hardware complexity of the MCA.

A Stick Diagram was used to estimate the total silicon area taken up by the MCA module and is expressed as

$$\text{Total Silicon Area} = n^2\lambda^2 [48n + 4k + 40]. \quad (27)$$

This expression is used to compute the area of the MCA module in terms of the minimum feature width, λ, for n = 8, 16, 32 and are presented in Table IV. Chip areas of $10^7\lambda^2$ to $10^8\lambda^2$ are possible [13].

VI. CONCLUSIONS

The CORDIC algorithm has been modified, and a new algorithm devised that incorporates increased parallelism and, consequently, possesses superior timing characteristics. The modified algorithm, known as the Modified Cordic Algorithm, employs a $[k+1]n^2$-bit ROM for lookup tables that enable elementary arithmetic functions to be evaluated in no more than $[3n+2]$ and no less than 2 time steps. The two bounds correspond to a pipelined and a parallel implementation, respectively. The formulation of the MCA can be mathematically manipulated to obtain hardware implementations with various speed/cost characteristics.

Furthermore, the algorithm has been matched with an architecture to achieve optimum performance based upon both pre-established hardware complexity and computational error requirements [12]. The result is an algorithm and architecture match that is quite attractive for a variety of applications. How favorably does MCA compare to other algorithms such as Chen's [6] and Newton's [16,17] etc., with respect to cost, speed and accuracy, is a question that can be answered only after the algorithm in question is matched with an architecture that meets the particular data flow requirements of the algorithm. The foundation for comparing the Modified Cordic Algorithm with other similar matches has been established.

BIBLIOGRAPHY

1. H. M. Ahmed, M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," Computer, January, 1982, pp. 65-82.

2. H. M. Ahmed, M. Morf, "A VLSI Speech Analysis Chip Set Based on Square Root Normalized Ladder Forms," Proc. 1981 Int'l Conf. Acoustic Speech and Signal Processing, March-April, 1981, Atlanta, GA., pp. 648-653.

3. M. Andrews, "Mathematical Microprocessor Software; A x Comparison," IEEE Micro, May, 1982, pp. 63-76.

4. P. W. Baker, "More Efficient Radix-2 Algorithms for some Elementary Functions," IEEE Tran. on Computers, Vol. C-24, No. 11, November, 1975, pp. 1049-1054.

5. C. L. Bridge, P. D. Fisher, R. G. Reynolds, "Asynchronous Arithmetic Algorithms for Data-Driven Machines," Proc. 5th Symposium on Computer Arithmetic, May 1981, pp. 56-62.

6. T. C. Chen, "The Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots," IBM Research Report, RJ970, 1972.

7. D. H. Daggett, "Decimal-Binary Conversions in CORDIC," IRE Tran. on Electronic Computers, Vol. EC-8, September 1959, pp. 335-339.

8. A. M. Despain, "Very Fast Fourier Transform Algorithms for Hardware Implementation," IEEE Tran. on Computers, Vol. C-28, No. 5, May, 1979, pp. 333-341.

9. A. M. Despain, "Fourier Transform Computers Using CORDIC Iterations," IEEE Tran. on Computers, Vol. C-23, No. 10, October, 1974, pp. 993-1001.

10. G. L. Haviland, A. A. Tuszynski, " A CORDIC Arithmetic Processor Chip,"IEEE Journal of Solid-State Circuits, Vol. SC-15, No. 1, February 1980, pp. 4-15.

11. P. M. Kogge, The Architecture of Pipelined Computers, McGraw-Hill Book Company, 1981.

12. H. T. Kung, "Let's Design Algorithms for VLSI Systems," Proc. Caltech Conf. on VLSI, January, 1979, pp. 65-90.

13. C. A. Mead, L. A. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980.

14. A. Naseem, P. D. Fisher, Implementation of Parallel Algorithms on a Modified Cordic ALU, Division of Engineering Technical Report No. MSU-ENGR-84-001, Michigan State University, 1984.

15. A. Naseem, P. D. Fisher, "A Modified CORDIC Algorithm," Proc. IEEE Int'l. Conf. on Computer Design: VLSI in Computers, pp. 584-688, October, 1984.

16. D. E. Smith, History of Mathematics, Ginn, 1973.

17. P. H. Sterbenz, C. T. Fike, "Optimal Starting Approximations for Newton's Method," Math. Computation, Vol. 23, 1969, pp. 313-318.

18. G. D. Taylor, "Optimal Starting Approximations for Newton's Method," J. Approximation Theory, Vol. 3, 1970, pp. 156-163.

19. J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Tran. on Electronic Computers, Vol. EC-8, September, 1959, pp. 330-334.

20. J. S. Walther, "A Unified Algorithm for Elementary Functions," AFIPS Conf. Proc., Vol. 38, 1971 SJCC, pp. 379-385.