Report No. NADC-91067-50

THE SQUARE ROOT CORDIC

Ronald F. Gleeson
Department of Physics
TRENTON STATE COLLEGE
Trenton, NJ 08650

James J. Davidson, Robert M. Williams and Robert G. Peck
Mission Avionics Technology Department (Code 5051)
NAVAL AIR DEVELOPMENT CENTER
Warminster, PA 18974-5000

26 JULY 1991

FINAL REPORT
Period Covering March 1991 to July 1991

Approved for Public Release; Distribution is Unlimited

Prepared for
NAVAL AIR SYSTEMS COMMAND (PMA-263)
Washington, DC 20361-0001

NOTICES

REPORT NUMBERING SYSTEM — The numbering of technical project reports issued by the Naval Air Development Center is arranged for specific identification purposes. Each number consists of the Center acronym, the calendar year in which the number was assigned, the sequence number of the report within the specific calendar year, and the official 2-digit correspondence code of the Command Officer or the Functional Department responsible for the report. For example: Report No. NADC-88020-60 indicates the twentieth Center report for the year 1988 and prepared by the Air Vehicle and Crew Systems Technology Department. The numerical codes are as follows:

CODE	OFFICE OR DEPARTMENT
00	Commander, Naval Air Development Center
01	Technical Director, Naval Air Development Center
05	Computer Department
10	AntiSubmarine Warfare Systems Department
20	Tactical Air Systems Department
30	Warfare Systems Analysis Department
40	Communication Navigation Technology Department
50	Mission Avionics Technology Department
60	Air Vehicle & Crew Systems Technology Department
70	Systems & Software Technology Department
80	Engineering Support Group
90	Test & Evaluation Group

PRODUCT ENDORSEMENT — The discussion or instructions concerning commercial products herein do not constitute an endorsement by the Government nor do they convey or imply the license or right to use such products.

Reviewed By: ______________________ Date: 8/9/91
Branch Head

Reviewed By: ______________________ Date: 9/3/91
Division Head

Reviewed By: ______________________ Date: 9/4/91
Director/Deputy Director

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 26 July 1991	3. REPORT TYPE AND DATES COVERED Final - March 1991 - July 1991

4. TITLE AND SUBTITLE

The Square Root CORDIC

5. FUNDING NUMBERS

6. AUTHOR(S)

Ronald F. Gleeson*, James J. Davidson,
Robert M. Williams, Robert G. Peck*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Mission Avionics Technology Department (Code 5051)
NAVAL AIR DEVELOPMENT CENTER
Warminster, PA 18974-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NADC-91067-50

9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES)

NAVAL AIR SYSTEMS COMMAND (PMA-263)
Washington, DC 20361-0001

10. SPONSORING MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*Ronald F. Gleeson   Trenton State College
Department of Physics   Trenton, NJ 08650

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release;
Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The CORDIC (Coordinate Rotation Digital Computer) algorithm[1] computes certain functions such as the sine, cosine, and $\sqrt{x^2 + y^2}$ using only additions and bit shifting operations.

We have implemented an integer math CORDIC algorithm on a high speed RISC processor. During the course of this work, we identified a convergence problem with the $\sqrt{x^2 + y^2}$ CORDIC. A solution to this problem is presented along with an overview of this algorithm.

14. SUBJECT TERMS

CORDIC, Integer Math

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The CORDIC (Coordinate Rotation Digital Computer) algorithm[1] computes certain functions such as the sine, cosine, and $\sqrt{x^2 + y^2}$ using only additions and bit shifting operations.

We have implemented an integer math CORDIC algorithm on a high speed RISC processor. During the course of this work, we identified a convergence problem with the $\sqrt{x^2 + y^2}$ CORDIC. A solution to this problem is presented along with an overview of this algorithm.

I. INTRODUCTION

The CORDIC algorithm[1] utilizes a series of rotations on a two dimensional vector to compute the following: sin(z), cos(z), arc tan(y/x), and $\sqrt{x^2 + y^2}$. In its generalized version it has also been shown to have the capability of performing multiplication and division, as well as computing hyperbolic functions, and $\sqrt{x^2 - y^2}$.

CORDIC has found its way into desk calculators, specifically, the HP-9100 series[2]; moreover, it has proven useful in calculating the Fourier Transform[3], and also the singular values of a matrix[4]. The algorithm can be implemented either in software or on a single digital IC[5].

We first discuss the CORDIC algorithm, and then present a problem we encountered in its use. Since our project involves real time control and requires an extremely small computer, we are using integer math in an RTX 2000 processor[6] programmed in its native FORTH language. A problem arose in the evaluation of $\sqrt{x^2 + y^2}$. using CORDIC. We characterize the problem and present our solution.

II. THEORY

The main working equations of the CORDIC algorithm can be related to the orthogonal transformation equations used to rotate a two dimensional vector. Let us assume our original vector **R** has components x and y. The transformation equations which rotate this vector through a positive clockwise angle δ are :

$$(1)\quad x' = x\cos(\delta) + y\sin(\delta)$$

$$(2)\quad y' = -x\sin(\delta) + y\cos(\delta)$$

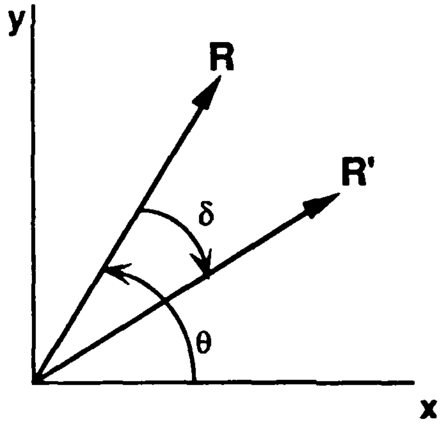

Figure 1: Orthogonal Rotation

Since the polar coordinate θ of a vector is normally defined in the counter-clockwise direction, the change in θ, that is Δθ, is the negative of this rotation angle δ (Δθ= -δ) . This is an orthogonal transformation, and the length of the rotated vector, **R'**, is the same as the length of the original vector, **R**.

For very small rotation angles $\sin(\delta) \approx \delta$, and $\cos(\delta) \approx 1$. Plugging in these approximations and reversing the order of the terms in equation (2), we have:

(3) $x' = x + y\,\delta$

(4) $y' = y - x\,\delta$

Equations (3) and (4), along with a third equation which keeps track of the cumulative angle of rotation (when this is relevant), are the main working equations of the CORDIC algorithm. The details of this procedure are discussed below in the ALGORITHM section (Section III).

The transformation equations are now no longer orthogonal, and correspond not only to a rotation, but also a stretching of the vector It is shown below that the stretch factor (K) equals $\sqrt{1+\delta^2}$

(5) $R' = \sqrt{(x')^2 + (y')^2}$

(6) $R' = \sqrt{(x + y\,\delta)^2 + (y - x\,\delta)^2}$

(7) $R' = \sqrt{x^2 + y^2\delta^2 + y^2 + x^2\delta^2}$

(8) $R' = \sqrt{(x^2 + y^2)(1 + \delta^2)}$

(9) $R' = R\sqrt{1+\delta^2}$

Furthermore, δ no longer represents the angle of rotation for the vector, but instead the vector will have been rotated clockwise through an angle α equal to the arctan(δ). The fact that α equals the arctan(δ) is proven next.

Define a vector **V** that has the same length as **R** and the same direction as **R'**.

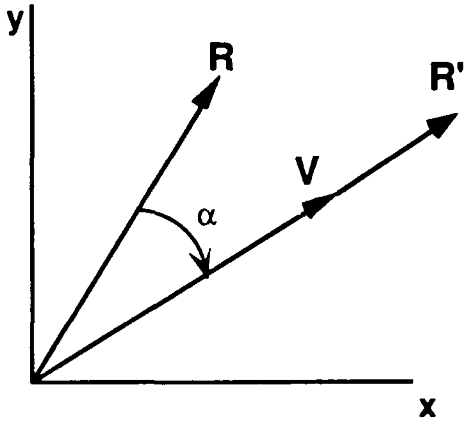

Figure 2: Cordic Rotation

Since the magnitude of V = R = R' / $\sqrt{1+\delta^2}$, the components of **V**, namely x_v and y_v ; are equal to x' / $\sqrt{1+\delta^2}$ and y' / $\sqrt{1+\delta^2}$. respectively. Since x' = x + y*δ , we have x_v = (x + y*δ) / $\sqrt{1+\delta^2}$, and therefore,

(10) $x_v = x / \sqrt{1+\delta^2} + y * \delta / \sqrt{1+\delta^2}$

The **V** vector is the **R** vector after an orthogonal clockwise rotation through an angle α, the transformation equation for x_v has the form

(11) $x_v = x * \cos(\alpha) + y * \sin(\alpha)$

Comparing equations (10) and (11) for x_v we see that

(12) $\sin(\delta) = \delta / \sqrt{1+\delta^2}$ and (13) $\cos(\alpha) = 1 / \sqrt{1+\delta^2}$

Recall that

(14) $\tan(\alpha) = \sin(\alpha) / \cos(\alpha)$

Plugging the expressions in (12) and (13) into (14) we get $\tan(\alpha) = \delta$ or $\alpha = \arctan(\delta)$. The same result can be obtained by an analysis of the y component of **V**.

III. THE ALGORITHM

There are two modes for the CORDIC algorithm. One is called vectoring; the other, rotation. The vectoring mode will be explained in detail since our problem arose in this mode when we tried to compute $\sqrt{x^2+y^2}$. For an explanation of how the rotation mode can be used to compute such functions as the sine and cosine, the reader should consult one the references[1,2,7,8].

The vectoring mode is useful when the x and y components of a vector are given and the magnitude $\sqrt{x^2+y^2}$ and/or the arctan(y/x) are desired. In this mode the successive CORDIC rotations are carried out in such a way as to eventually "force y to zero". Each iteration corresponds to a nonorthogonal rotation, and stretches the vector by a factor of $\sqrt{1+\delta_i^2}$. This stretch factor is independent of the direction of the rotation. The cumulative stretch factors are listed in Table 2. After y has been forced to zero (i.e. the vector has been rotated to align with the +x axis), the magnitude, $\sqrt{x^2+y^2}$, is obtained by dividing the value in the x variable by the cumulative stretch factor.

To compute $\sqrt{x^2+y^2}$ the working equations are:

(15) $x_{i+1} = x_i + y_i \delta_i$

(16) $y_{i+1} = y_i - x_i \delta_i$

where for the ith iteration $\delta_i = \pm (1/2)^i$ and $i = 0, 1, 2, 3 \ldots$

The ± sign is selected by checking whether y_i is positive or negative. In order to force y to zero, if y_i is positive, then δ_i is positive, and $x_i \delta_i$ is subtracted from y_i (N.B. x_i is always positive). Conversely, if y_i is negative, then δ_i is chosen to be negative also.

Multiplying x_i or y_i by δ_i is achieved by right shifting the value. For example, if i equals 3 then δ_3 equals $(1/2)^3$. The value of $y_3 \delta_3$ is then computed by simply shifting the binary value of y_3 three places to the right.

In the $\sqrt{x^2 + y^2}$ computation there is no need to keep track of the cumulative rotation angle. However, If the arctan(y/x) of the original vector is desired, then one simply sums up th9 angles of rotation (α_i) produced by each iteration (recall, α_i = arc tan (δ_i)).

IV. INTEGER ARITHMETIC PROBLEM

The RTX processor is equipped with specialized square root instructions. This routine will take the square root of any positive integer up to 31 bits long (corresponding to the decimal range of zero to 2,147,483,647). This may seem like a large range, but in the special case where x equals y in $\sqrt{x^2 + y^2}$, the maximum value for x is only 32,767. This is not adequate for our purposes. We tried using a 63 bit square root algorithm, but CORDIC executed faster. Using CORDIC we can extend the range of the input values, x and y, to 30 bits.

Unfortunately, when we tested our CORDIC square root function, we came across the difficulty illustrated in the following example.

Suppose x equals 333 and y equals 444 . We can expect $\sqrt{x^2 + y^2}$ to yield 555 since this is a 3-4-5 triangle. Below we present a table of x_i , y_i , $x_i \delta_i$, and $y_i \delta_i$ after each iteration as determined by the algorithm discussed above.

Table 1: Example

i	x_i	y_i	$x_i\delta_i$	$y_i\delta_i$
0	333	444	333	444
1	777	111	388	55
2	832	-277	208	-70
3	902	-69	112	-9
4	911	43	56	2
5	913	-13	28	-1
6	914	15	14	0
7	914	1	7	0
8	914	-6	3	-1
9	915	-3	1	-1
10	916	-2	0	-1
11	917	-2	0	-1
12	918	-2	0	-1
13	919	-2	0	-1
14	920	-2	0	-1
15	921	-2	0	-1

The reader will note that after iteration #7 the value of y is closest to zero. If the value of x after iteration #7 (namely, 914) is divided by the stretch facter (see table 2) of 1.6466932543, and then rounded to an integer, the result turns out to be the correct integer, 555. However, after iteration #9, y is stuck at -2, but x (and therefore the result) continues to grow.

Table 2: Stretch Factors

Iteration Number	Stretch Factor (K)
0	1.4142135624
1	1.5811388301
2	1.6298006013
3	1.6424840658
4	1.6456889158
5	1.6464922787
6	1.6466932543
7	1.6467435066
8	1.6467560702
9	1.6467592111
10	1.6467599964
11	1.6467601927
12	1.6467602418
13	1.6467602540
14	1.6467602571
15	1.6467602579
16	1.6467602581
17	1.6467602581
18	1.6467602581
19	1.6467602581
20	1.6467602581
21	1.6467602581
22	1.6467602581
23	1.6467602581
24	1.6467602581
25	1.6467602581
26	1.6467602581
27	1.6467602581
28	1.6467602581
29	1.6467602581
30	1.6467602581
31	1.6467602581

V. SOLUTIONS

We considered several ways to patch the algorithm. Since the y value could not always be forced exactly to zero, we needed another condition that would reliably halt the iterative process without introducing too much error in the result ($\sqrt{x^2 + y^2}$). We considered checking for small rates of change in x, y, xδ, or yδ. We decided instead to check whether the absolute value of y was less than some predetermined cutoff value as our halt condition. The values in Table 1 suggested to us that if the absolute value of y became less than three, it was time to stop. This condition was tested by looping through millions of combinations of integers that maintain the 3-4-5 proportionality and were in our range of interest. We also decided to test other limits for y. The limit for y was incremented from 0 to 127. Table 3 is a representative selection of the distribution of errors as a function of the $|y|$ cutoff. The error frequency counts were truncated to 32760 to avoid overflow. When the $|y|$ cutoff was less than three, a second peak in the error distribution appears between 10 and 14. These occurrences resulted from cases which were never halted at maturity. The drift from the correct result continued until the DO loop was completed (32 iterations).

Using the combinations of integers that maintain the 3-4-5 proportionality, the error stayed below six for a broad range of $|y|$ cutoff values. Eventually, at a sufficiently high cutoff (approximately 100) the size of the error began to rise due to premature halting of the algorithm. These cases involved small initial values of x and y. In particular, when the initial value of y was less than the cutoff, the algorithm halted immediately and returned the initial value of x as its result.

Table 3: Error Frequency Vs. Size of Error and Cutoff

Error	|y|<0*	1	2	3	28	60	80	101
0	26	665	1810	3078	32760	32760	32760	32760
1	2566	16177	32760	32760	32760	32760	32760	32760
2	23370	32760	32760	32760	32760	28259	19375	19150
3	32760	32760	32760	32760	8528	6803	5446	4359
4	21159	26561	19078	9693	1446	734	574	436
5	8043	6709	3809	2385	61	4	2	22
6	3159	1188	272	138	1	0	0	1
7	1190	432	5	1	0	0	0	0
8	812	125	0	0	0	0	0	0
9	15729	3839	0	0	0	0	0	0
10	32760	21286	8	0	0	0	0	0
11	32760	16511	11	0	0	0	0	0
12	7576	3258	3	0	0	0	0	0
13	1465	510	5	0	0	0	0	0
14	412	239	37	0	0	0	0	0
15	68	31	1	0	0	0	0	0
16	2	0	0	0	0	0	0	0
17	1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0

* This is equivalent to the standard CORDIC algorithm (no |y| cutoff).

Other combinations of integers were also tested. For example, integers that maintain the 5-12-13 proportionality, as well as integers generated randomly, were studied. The general features of the distribution of errors as a function of the |y| cutoff remained the same; however, the region where the errors were less than six moved around.

The function which we finally implemented involves a hybrid approach to evaluating $\sqrt{x^2 + y^2}$. Whenever the input values of both x and y are smaller than 32768, the RTX processor's 31 bit square root function is employed. Otherwise, CORDIC with a |y| cutoff of 100 is used. This combined the best of both worlds. The built in routine was very fast, but could not handle large numbers; whereas, CORDIC produced a much smaller per cent error for large numbers than it did for small numbers. Setting the |y| cutoff at 100 has the advantage of providing a relatively quick exit condition.

Furthermore, very little is lost with this choice of cutoff since we only use CORDIC for large values of x and y. Suppose, for example, the initial values of x and y are 30,000 and 40,000 respectively. Since one of these numbers is larger than 32768 we would utilize CORDIC. The expected result for $\sqrt{x^2 + y^2}$ is 50,000. When the vector has been rotated such that y = 100, the value of x is then 49,999.9 (ignoring the stretch factor for the sake of argument). The truncated value of 49,999 is only one less than the correct value of 50,000.

VI. CONCLUSION

While the CORDIC algorithm provides a simple method of evaluation for a wide variety of functions, we found that caution is necessary in certain circumstances. In particular, when integer arithmetic is used and $\sqrt{x^2 + y^2}$ is evaluated by CORDIC, significant errors sometimes arise. This is especially bothersome for small initial values of both x and y. One way to handle this problem is to place a cutoff condition on the absolute value of y. Usually, a built in square root function is available; however, its range may be too limited. We recommend using the built in function because of its speed and accuracy whenever it is possible, and using CORDIC with a suitable cutoff on the absolute value of y to extend the range.

REFERENCES

1. Volder, J., "The CORDIC Trigometric Computing Technique", *IRE Transactions on Electronic Computers,* EC-8,(3), pp.330-334 (Sept. 1959).

2. Walther, J.S., "A Unified Algorithm for Elementary Functions", *AFIPS Spring Joint "Computer Conference,* pp. 379-385 (1971).

3. Despain, A.M., "Fourier Transform Computers Using CORDIC Iterations", *IEEE Transactions Conference on Computers,* C-23 ,(10),pp.993-1001 (October,1974).

4. Cavallaro, J.R., and Luk,F.T., "Architectures for a CORDIC SVD Processor", *Proceedings of SPIE - The International Society for Optical Engineering,* 698, pp.45-53 (August,1986).

5. Haviland, G.L., and Tuszynski,A.A., "A CORDIC Arithmetic Processor Chip", *IEEE Transactions of Computers,* C-29,(2), pp,68-79 (Feb.,1980).

6. *RTX 2000TM Hardware Reference Manual,* Harris Corporation, Melbourne, Florida, 1990.

7. Ahmed, H.A., "Signal Processing Algorithms and Architectures", Technical Report M735-21, Stanford University, Information Systems Lab. (June, 1982).

8. Ruckdeschel, F.R., *BASIC Scientific Subroutines,* Volume 2, pp.231-242, BYTE/McGraw-Hill, Peterborough, NH, 1981.

9. Johnsson, S.L., and Krishnaswamy, V., "Floating-point Cordic", Research Report YALEU/DCS/RR-473 (April,1986).

DISTRIBUTION LIST

REPORT NO. NADC-91067-50

	No. of Copies
Defense Technical Information Center Cameron Station Alexandria, VA 22394	2
Office of Naval Technology Attn: Dr. Sherman Gee 800 Quincey St. Arlington, VA 22217	1
NAVAIRDEVCEN:	
Scientific and Technical Library,Code 8131	2
Code 01B	1
Code 301	1
Code 3011	1
Code 302	1
Code 3021	1
Code 303	1
Code 3031	1
Code 40F	1
Code 401	1
Code 402	1
Code 403	1
Code 404	1
Code 50	1
Code 501	1
Code 5012 (Dr. Jon Davis)	1
Code 5012 (Dr. Lloyd Bobb)	1
Code 502	1
Code 503	1
Code 504	1
Code 505	1
Code 505A (Dr. Robert M. Williams)	30
Code 5051 (Robert G. Peck)	10
Code 5051 (James J. Davidson)	10
Code 5032 (Anthony Passamante)	1
Code 6012	1
Code 6051 (Dr. Richard Llorens)	1