CORDIC ALGORITHM

AND

ITS APPLICATIONS IN DSP

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

in

Electrical Engineering

By

SAMBIT KUMAR DASH
JASOBANTA SAHOO
SUNITA PATEL

Department of Electrical Engineering

National Institute of Technology

Rourkela

2007

CORDIC ALGORITHM

AND

ITS APPLICATIONS IN DSP

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology
in
Electrical Engineering

By

SAMBIT KUMAR DASH
JASOBANTA SAHOO
SUNITA PATEL

Under the Guidance of

Prof. S. Mohanty

Department of Electrical Engineering

National Institute of Technology

Rourkela

2007

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled,"**CORDIC Algorithm and it's applications in DSP**" submitted by Sri **Sambit Kumar Dash**, Sri **Jasobanta Sahoo, Sunita Patel** in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electrical Engineering at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Date : 02/05/07
Place: Rourkela

Prof. S. Mohanty
Dept. of Electrical Engg.
National Institute of technology
Rourkela-769008

ACKNOWLEDGEMENT

I would like to articulate my deep gratitude to my project guide **Prof. S. Mohanty** who has always been my motivation for carrying out the project.

I wish to extend my sincere thanks to **Prof. P. K. Nanda**, Head of our Department, for allowing us to use the facilities of the “Computing and Simulation Lab”

It is my pleasure to refer Microsoft Word exclusive of which the compilation of this report would have been impossible. Also it would not have been possible to complete the project without the simulation software”MATLAB”

A project of this nature could never have been attempted with our reference to and inspiration from the works of others whose details are mentioned in references section. I acknowledge my indebtedness to all of them. Last but not the least, my sincere thanks to all of my friends who have patiently extended all sorts of help for accomplishing this undertaking.

SAMBIT KUMAR DASH
JASOBANTA SAHOO
SUNITA PATEL

CONTENTS

ABSTRACT

OBJECTIVE:

The digital signal processing landscape has long been dominated by the microprocessors with enhancements such as single cycle multiply-accumulate instructions and special addressing modes. While these processors are low cost and offer extreme flexibility, they are often not fast enough for truly demanding DSP tasks. The advent of reconfigurable logic computers permits the higher speeds of dedicated hardware solutions at costs that are competitive with the traditional software approach. Unfortunately algorithms optimized for these microprocessors based systems do not map well into hardware. While hardware efficient solutions often exist, the dominance of the software systems has kept these solutions out of the spotlight. Among these hardware-efficient algorithms is a class of iterative solutions for trigonometric and other transcendental functions that use only shifts and adds to perform. The trigonometric functions are based on vector rotations, while other functions such as square root are implemented using an incremental expression of the desired function. The trigonometric algorithm is called CORDIC an acronym for Coordinate Rotation Digital Computer. The incremental functions are performed with a very simple extension to the hardware architecture and while not CORDIC in the strict sense, are often included because of the close similarity. The CORDIC algorithms generally produce one additional bit of accuracy for each iteration.

DESCRIPTION:

A detailed study on various modes of CORDIC algorithm is done. First of all a study is made how the CORDIC algorithm is derived from the general vector equation. Then a study is done regarding the various modes of the CORDIC algorithm and how it can be used to find the sine, cosine, tan and logarithm functions, its use in conversion of coordinate systems. An attempt is made to carry out a rigorous study of its use in DSP oriented applications AND how it has revolutionized the DSP scenario. Finally simulations are carried out using MATLAB to support the purpose of our study.

RESULTS

The results clearly bring out the advantage of using CORDIC algorithm. First of all the sine and cosine of any angle could be found out easily. Similar is the case of logarithm and hyperbolic functions. The simulation results prove the fact that the hardware complexity gets reduced by using the CORDIC algorithm. A large no of plots were obtained for different

functions. Finally the implementation in DCT was carried out and the results obtained were in line with those of the theoretical values.

CONCLUSION

The CORDIC algorithms presented in this paper are well known in the research and super computing circles. Here the basic CORDIC algorithm and a partial list of potential applications of potential applications of a CORDIC based processor array to digital signal processing is presented. The CORDIC based DCT architecture for low power design has been proposed. The proposed multiplierless CORDIC based DCT architecture produces high throughput and is easy to implementing VLSI. The proposed architecture reduced the input data range for the CORDIC processor by split and the no of compensation iterations in CORDIC based DCT computation by utilizing that most images have similar neighboring pixels. The project also shows that a tool is available for use in FPGA based computing machines, which are the likely basis for the next generation DSP systems.

Chapter 1

INTRODUCTION

Background
Objective

1.1 BACKGROUND

The digital signal processing landscape has long been dominated by the microprocessors with enhancements such as single cycle multiply-accumulate instructions and special addressing modes. While these processors are low cost and offer extreme flexibility, they are often not fast enough for truly demanding DSP tasks. The advent of reconfigurable logic computers permits the higher speeds of dedicated hardware solutions at costs that are competitive with the traditional software approach. Unfortunately algorithms optimized for these microprocessors based systems do not map well into hardware. While hardware efficient solutions often exist, the dominance of the software systems has kept these solutions out of the spotlight. Among these hardware-efficient algorithms is a class of iterative solutions for trigonometric and other transcendental functions that use only shifts and adds to perform. The trigonometric functions are based on vector rotations, while other functions such as square root are implemented using an incremental expression of the desired function. The trigonometric algorithm is called CORDIC an acronym for Coordinate Rotation Digital Computer. The incremental functions are performed with a very simple extension to the hardware architecture and while not CORDIC in the strict sense, are often included because of the close similarity. The CORDIC algorithms generally produce one additional bit of accuracy for each iteration.

The trigonometric CORDIC algorithms were originally developed as a digital solution for real time navigation problems. The original work is credited to Jack Volder .The CORDIC algorithm has found its way into diverse applications including the 8087 math coprocessor, the HP-35 calculator, radar signal processors and robotics.CORDIC rotation has also been proposed for computing Discrete Fourier[4],Discrete Cosine[4],Singular Value Decomposition[5],and solving linear systems[1].

1.2 OBJECTIVE

The project attempts to survey the existing CORDIC and CORDIC-like algorithms with an eye towards its implementation. First a brief description of the theory behind the algorithm and the derivation of several functions are presented. Then the theory is extended to the so called unified CORDIC algorithms.

Chapter 2

CORDIC ALGORITHM AND IT'S VARIOUS MODES

CORDIC THEORY: AN ALGORITHM FOR VECTOR ROTATION

All of the trigonometric functions can be computed or derived from functions using vector rotations, as will be discussed in the following sections. Vector rotation can also be used from polar to rectangular and rectangular to polar conversions, for vector magnitude, and as a building block in certain transforms such as DFT and DCT.The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds. The algorithm credited to Volder [4] is derived from the general rotation transform

$$x' = x\cos\varnothing - y\sin\varnothing$$
$$y' = y\cos\varnothing + x\sin\varnothing$$

this rotates a vector in a Cartesian plane by the angle Ø.

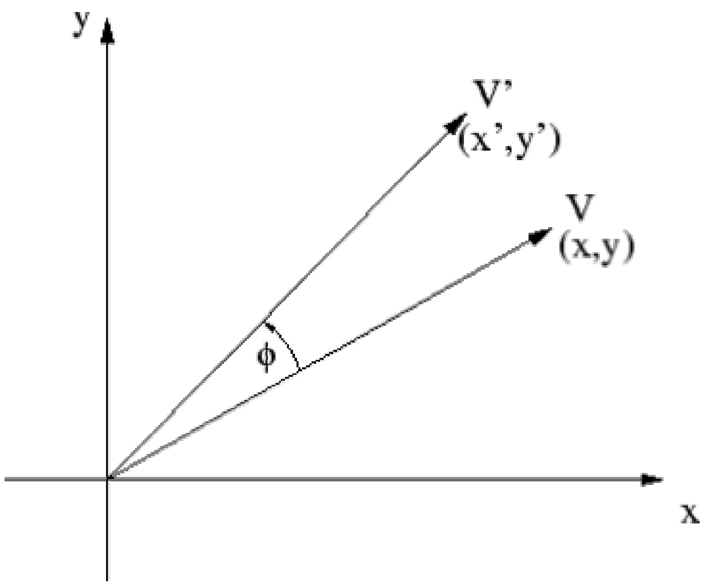


FIG 2.1: Rotation of a vector V by angle ø

These can be rearranged so that

$$x' = \cos\varnothing * [x - y\tan\varnothing]$$
$$y' = \cos\varnothing * [y + x\tan\varnothing]$$

So far nothing is simplified. However if the rotation angles are restricted so that $\tan(\varnothing) = \pm 2^{-i}$,the multiplication by the tangent term is reduced to simple shift operation.Arbitary angles of rotation are obtainable by performing a series of successively smaller elementary operations. If the decision at each iteration i ,is which direction to rotate rather than whether or not to rotate, then the $\cos(\delta_i)$ term becomes a constant because $\cos(\delta i) = \cos(-\delta_i)$.The iterative rotation can now be expressed as:

$$x_{i+1} = K_i\,[x_i - y_i \times d_i \times 2^{-i}]$$

$$y_{i+1}=K_i\,[y_i + xi\times d_i\times 2^{-i}]$$

where:

$$K_i = \cos(\tan^{-1}2^{-i}) = 1/\sqrt{1+2^{-2i}}$$

$$d_i= \pm 1$$

Removing the scale constant from the iterative equations yields a shift add algorithm for vector rotation. The product of the K_i's can be applied elsewhere in the system or treated as apart of the system processing gain. The product approaches 0.6073 as the number of iteration goes to infinity. Therefore the rotation algorithm has a gain A_n of approximately 1.647.The exact gain depends on the no of iterations and obeys the relation

$$A_n = \prod_n \sqrt{1+2^{-2i}}$$

The angle of a composite rotation is uniquely defined by the sequence of the directions of the elementary rotations. That sequence can be represented by a decision vector. The set of all possible decision vectors is an angular measurement system based on binary arctangents. Conversions between the angular systems and any other can be accomplished using a look up. A better conversion method uses an additional adder-subtract or that accumulate the elementary rotation angles at each iteration. The elementary angles can be expressed in any convenient angular unit. Those angular values are supplied by a small look up table or are hardwired depending on the application. The angle accumulator adds a third difference equation to the CORDIC algorithm:

$$z_{i+1}= z_i - d_i\times \tan^{-1}(2^{-i})$$

Obviously, in cases where the angle is useful in the arc tangent base, this extra element is not needed.

The CORDIC rotator is normally operated in one of two mode3s.The first called rotation by Volder[4]rotates the input vector by a specified angle. The second mode called vectoring rotates the input vector to the x axis while recording the angle required to make that rotation.

2.1ROTATION MODE:

In rotation mode, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator. The decision at each iteration is therefore based on the sign of the residual angle after each step.Naturally, if the input angle is already expressed in the binary arctangent base, the angle accumulator may be eliminated. For rotation mode, the CORDIC equations are:

$$x_{i+1} = x_i - y_i \times d_i \times 2^{-i}$$
$$y\ i+1 = y_i + x_i \times d_i \times 2^{-i}$$
$$z\ i+1 = z_i - d_i \times \tan^{-1}(2^{-i})$$

where

d_i = -1 if z_i<0,+1 otherwise

which provides the following result

$$x_n = A_n[x_0 \cos z0 - y_o \sin z_0]$$
$$y_n = A_n[yo \cos z0 + x_0 \sin z_0]$$
$$z_n = 0$$
$$An = \prod_n \sqrt{1 + 2^{-2i}}$$

2.2CORDIC ROTATION IN VECTORING MODE:

In the vectoring mode the CORDIC vector rotates the input vector through whatever angle is necessary to align the result vector with the x axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result. The vectoring function works by seeking to minimize the y component of the residual vector at each rotation. The sign of the residual y component is used to determine which direction is to rotate next. If the angle accumulator is initialized with zero, it will contain the traversed angle at the end of the iterations. In vectoring mode the CORDIC equations are:

$$x_{i+1} = x_i - y_i \times d_i \times 2^{-i}$$
$$y_{i+1} = y_i + x_i \times d_i \times 2^{-i}$$
$$z_{i+1} = z_i - d_i \times \tan^{-1}(2^{-i})$$

where

d_i= +1 if y_i<0,-1 otherwise

Then:

$$x_n = A_n \times \sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 + \tan^{-1}(y_o/x_0)$$

$$An = \prod_{n} \sqrt{1+2^{-2i}}$$

The CORDIC rotation and vectoring algorithms as stated are limited to rotation angles between -90^0 and 90^0. For composite rotations larger than 90^0 an additional rotation is required.
This gives the correction iteration:

$$x' = -d \times y$$
$$y' = d \times x$$
$$z' = z + d \times \prod /2$$

where

d=+1 if y<0,-1 otherwise

There is no growth for this initial rotation. Alternatively an initial rotation of either $\prod$ or 0 can be made avoiding the reassignment of the x and y components to the rotator elements. Again there is no growth due to the initial rotation

$$x' = d \times x$$
$$y' = d \times y$$
$$z' = z \text{ if } d = 1, \text{ or } z = \prod \text{ if } d = -1$$

d= -1 if x<0,+1 otherwise

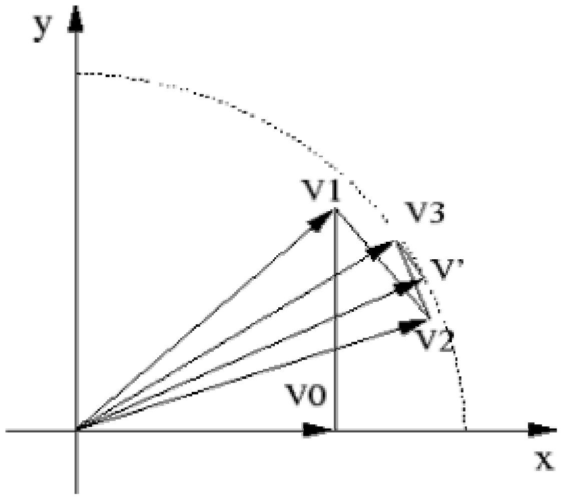


FIG 2.2: Iterative vector rotation, initialized with V0

Chapter 3

CORDIC ALGORITHM AND IT'S APPLICATIONS

3.1SINE AND COSINE:

The rotational mode CORDIC operation can simultaneously compute the sine and cosine of the input angle. Setting the y component of the input vector to zero reduces the rotation mode results to

$$x_n = An \times x_0 \cos z_0$$

$$y_n = A_n \times x_0 \sin z_0$$

By setting x_o equal to $1/A_n$ rotation produces the unscaled sine and cosine of the angle argument z_0.Very often the sine and cosine values modulate a magnitude value. Using other techniques requires a no of multipliers to obtain the modulation. The CORDIC technique performs the multiply as part of the rotation operation and therefore eliminates the need for a pair of explicit multipliers. The output of the CORDIC rotator is scaled by the rotator gain. If the gain is not acceptable, a single multiply by the reciprocal of the gain constant placed before the CORDIC rotator will yield the unscaled results. It is worth noting that the hardware complexity of the CORDIC rotator is approximately equivalent to that of a single multiplier with the same word size.

EXAMPLE:

To find the sine of 28.027^0

$Ø=0^0$

$\cos(ø)=1$ $X=1$

$\sin(ø)=1$ $Y=0$

Rotate from 0^0 to $45^0(ø_{21} = 45^0)$

$X' = X-Y/1 = 1-0/1 = 1$

$Y' = X/1+Y = 1/1+0 = 1$

Rotate from 450 to 18.4350($ø_{32} = -26.565^0$).Because this is a negative angle, and because the tangent function is odd, we change the sign of the numbers that get shifted.

$X' = X + Y/2 = 1+1/2 = 1.5$

$Y' = -X/2 + Y = -1/2+1=0.5$

The aggregate constant is not affected because it is a product of cosines, and the cosine function is even.

Rotate from 18.435^0 to 32.4710($ø_{43}=14.036^0$)

$X'=X-Y/4=1.5-0.5/4=1.375$

$Y'= X/4+Y=1.5/4+0.5=0.875$

Rotate from 32.471^0 to $25.346^0(ø_{54}= -7.125^0)$

$X'=X+Y/8=1.375+0.875/8=1.484375$

$Y'= -X/8+Y= -1.375/8+0.875=0.703125$

Rotate from 25.346^0 to $28.922^0(ø_{65} = 3.576^0)$

$X'= X-Y/16= 1.484375-0.703125/16=1.440429$

$Y'= X/16+Y=1.484375/16+0.703125=0.795898$

Rotate from 28.922^0 to $27.132^0(ø_{76} = -1.790^0)$

$X'= X+Y/32=1.440429+0.795898/32 = 1.465300$

$Y' = -X/32+Y = -1.440429/32+0.795898=0.750884$

Rotate from 27.1320 to 28.0270(ø87 = 0.8950)

$X'= X-Y/64=1.465300-0.750884/64=1.453567$

$Y'= X/64+Y=1.456300/64+0.750884=0.773779$

After that we have to just multiply by the aggregate constant

$\sin(28.027^0)=0.607253*Y=0.46988$

$\cos(28.027^0)=0.607253*X=0.88268$

3.1 POLAR TO RECTANGULAR TRANSFORMATION:

A logical extension to sine and cosine computer is a polar to Cartesian coordinate transformer. The transformation from polar to Cartesian space is defined by .

$$x = r\cos\theta$$

$$y = r\sin\theta$$

As pointed out above, the multiplication by the magnitude comes for free using the CORDIC rotator. The transformation is accomplished by selecting the rotation mode with x_0=polar phase and y_0=0.The vector result represents the polar input transformed to Cartesian space. The transform has a gain equal to the rotator gain, which needs to be accounted somewhere in the system. If the gain is unacceptable the polar magnitude may be multiplied by the reciprocal of the rotator gain before it is presented to the CORDIC rotator.

3.2GENERAL VECTOR ROTATION:

The rotation mode CORDIC rotator is also useful for performing general vector rotations, as are often encountered in motion correction and control systems. For general rotation, the 2 dimensional input vector is presented to the rotator inputs. The rotator rotates the vector through the desired angle. The output is scaled by the CORDIC rotator gain, which must be accounted for elsewhere in the system. If the scaling is unacceptable, a pair of constant multipliers is required to compensate the gain.

3.3ARCTANGENT:

The arctangent, θ=Atan(y/x) is directly computed using the vectoring mode CORDIC rotator if the angle accumulator is initialized with zero. The argument must be provided as a ratio expressed as a vector(x,y).Since the arctangent result is taken from the angle accumulator the CORDIC rotator growth does not affect the result.

$$z_n = z_0 + \tan^{-1}(y_0/x_0)$$

The vectoring mode CORDIC rotator produces the magnitude of the input vector as a byproduct of computing the arctangent. After the vectoring mode rotation the vector is aligned with the x axis. The magnitude of the vector is therefore the same as the x component of the rotated vector. This result is apparent in the result equations for the vector mode rotator:

$$x_n = A_n\sqrt{x_0^2 + y_0^2}$$

The magnitude result is scaled by the processor gain which needs to be accounted for elsewhere in the system. The CORDIC implementation represents a significant hardware savings over an equivalent Pythagoras processor. The accuracy of the magnitude result improves by 2 bits for each iteration performed.

3.4CARTESIAN TO POLAR TRANSFORMATION:

The Cartesian to polar transformation consists of finding the magnitude(r=sqrt(x2+y2)) and phase angle (ø=atan[y/x]) of the input vector(x,y).The reader will immediately recognize that both functions are provided simulnteously by the vectoring mode CORDIC rotator. The magnitude of the result will be sealed by the CORDIC rotator gain and should be accounted for elsewhere in the system. If the gain is unacceptable, it can be corrected by multiplying the resultant magnitude by the reciprocal of the gain constant.

3.5INVERSE CORDIC FUNCTIONS

In most cases if a function is generated by a CORDIC style computer, its inverse can be computed. Unless the CORDIC rotator described is usable to compute several trigonometric functions directly and others indirectly. Judicious choice of initial values and modes permits direct computation of sine, cosine, arctangent, vector magnitude and transformations between polar and Cartesian coordinates.

ARCSINE AND ARCCOSINE:

The Arcsine can be computed by starting with a unit vector on the positive x axis, then rotating it so that its y component is equal to the input argument. The arcsine is then the angle subtended to cause the y component of the rotated vector to match the argument. The decision function in this case is the result of a comparison between the input value and the y component of the rotated vector at each iteration.

$$x_{i+1} = x_i - y_i \times d_i 2^{-i}$$
$$y_{i+1} = y_i + x_i \times d_i 2^{-i}$$
$$z_{i+1} = z_i - d_i \times \tan^{-1}(2^{-i})$$

where

$d_i = +1$ if $y_i < c$, -1 otherwise, and

c= input argument

Rotation produces the following result

$$x_n = \sqrt{(A_n \times x_0)^2 - c^2}$$

$$y_n = c$$

$$z_n = z_0 + \arcsin(c/A_n\, x_0)$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

The arcsine function as stated above returns correct angles for inputs -1<c/Anx0<1, although the accuracy suffers as the input approaches ± 1(the error increases rapidly for inputs larger than about 0.98).The loss of accuracy is due to the gain of the rotator. For angles near the y axis the rotator gain causes the rotated vector to be shorter than the reference input so the decisions are made improperly.

3.7EXTENSION TO LINEAR FUNCTIONS:

A simple modification to the CORDIC equation permits the computation of linear functions

$$x_{i+1} = x_i - 0 \times d_i\, 2^{-i} = x_i$$

$$y_{i+1} = y_i + x_i \times d_i\, 2^{-i}$$

$$z_{i+1} = z_i - d_i \times (2^{-i})$$

For rotation mode (d_i = -1 if z_i<0, +1 otherwise) the linear rotation produces

$$x_n = x_0$$

$$y_n = y_0 + x_0 \times z_0$$

$$z_n = 0$$

This operation is similar to the shift add implementation of a multiplier and as multipliers go is not an optimal solution. The multiplication is handy in applications where a CORDIC structure is already available. The vectoring mode is more interesting as it provides a method for evaluating ratios

$$x_n = x_0$$

$$y_n = 0$$

$$z_n = z_0 - y_0/x_0$$

The rotations in the linear coordinate system have a unity gain, so no scaling corrections are required.

3.EXTENSION TO HYPERBOLIC FUNCTIONS:

The close relationship between the trigonometric an hyperbolic functions suggests the same architecture can be used to compute the hyperbolic functions. While there is early mention of using the CORDIC structure for hyperbolic coordinate transforms [4] the first description of the algorithm is that by Wather[1].The CORDIC equations for hyperbolic rotations are derived using the same manipulation as those used to derive the rotation in the circular coordinate system. For rotation mode these are

$$x_{i+1} = x_i + y_i \times d_i \times 2^{-i}$$
$$y_{i+1} = y_i + x_i \times d_i \times 2^{-i}$$
$$z_{i+1} = z_i - d_i \times \tanh^{-1}(2^{-i})$$

where

d_i= -1 if z_i<0, +1 otherwise

Then

$$x_n = A_n[x_0 \cosh z_o + y_0 \sinh z_0]$$
$$y_n = A_n[y_0 \cosh z_0 + x_o \sinh z_0]$$
$$z_n = 0$$
$$An = \prod_n \sqrt{1-2^{-2i}} = 0.80$$

The elemental rotations in the hyperbolic coordinate systems do not converge. However it can be shown [1] that convergence is achieved if certain iterations are repeated.

The hyperbolic equivalents for all the functions discussed for the circular coordinate system can be computed in a similar fashion. Additionally as Wather[1] points out, The following functions can be derived from the CORDIC functions

$$\tan\alpha = \sin\alpha/\cos\alpha$$
$$\tanh\alpha = \sinh\alpha/\cosh\alpha$$
$$\exp\alpha = \sinh\alpha + \cosh\alpha$$
$$\ln\alpha = 2\tanh^{-1}[y/x] \text{ where } x= \alpha+1 \text{ and } y=\alpha-1$$
$$(\alpha)^{1/2} = (x^2 - y^2)^{1/2} \text{ where } x= \alpha+1/4 \text{ and } y= \alpha-1/4$$

It is worth noting the similarities between the CORDIC equations for circular, linear and hyperbolic systems. The selection of coordinate system can be made by introducing a mode variable that takes on values 1, 0 or -1 for circular, linear and hyperbolic systems respectively. The unified [1] CORDIC iteration equations are then

$$x_{i+1} = x_i - m \times y_i \times d_i \times 2^{-i}$$
$$y_{i+1} = y_i + x_i \times d_i \times 2^{-i}$$

$$z_{i+1} = z_i - d_i \times e_i$$

where e_i is the elementary angle of rotation for iteration i in the selected coordinate system. Specifically $e_i = \tan^{-1}(2^{-i})$ for m=1, $ei = 2^{-i}$ for m=0, and $e_i = \tanh^{-1}(2^{-i})$ for m=-1.The unification due to Wather permits the design of a general purpose CORDIC processor. There are a number of ways to implement a CORDIC processor. The ideal architecture Depends on the speed versus area trade offs in the intended application. First we will examine an iterative architecture that is a direct translation from the CORDIC equations and then we will look at a minimum hardware solution and a maximum performance solution.

Chapter 4

CORDIC ALGORITHM IN DFT & DCT

4.1DISCRETE FOURIER TRANSFORM:

The sequence of *N* complex numbers $x_0, ..., x_{N-1}$ is transformed into the sequence of *N* complex numbers $X_0, ..., X_{N-1}$ by the DFT according to the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\Pi ikn/N} \qquad k=0, 1\ldots\ldots\ldots..N-1$$

where *e* is the base of the natural logarithm, i is the imaginary unit ($i^2 = -1$), and π is pi. The transform is sometimes denoted by the symbol F, as in X=F(X) or FX

The inverse discrete Fourier Transform (IDFT) is given by

$$X_n = 1/N \sum_{K=0}^{N-1} X_k e^{2\Pi ikn/N} \qquad n=0, 1 \ldots\ldots\ldots\ldots N-1$$

Note that the normalization factor multiplying the DFT and IDFT (here 1 and 1/*N*) and the signs of the exponents are merely conventions, and differ in some treatments. The only requirements of these conventions are that the DFT and IDFT have opposite-sign exponents and that the product of their normalization factors be 1/*N*. A normalization of $1/\sqrt{N}$ for both the DFT and IDFT makes the transforms unitary, which has some theoretical advantages, but it is often more practical in numerical computation to perform the scaling all at once as above (and a unit scaling can be convenient in other ways).(The convention of a negative sign in the exponent is often convenient because it means that X_k is the amplitude of a "positive frequency" $2\pi k / N$.

DISCRETE COSINE TRANSFORM

A discrete cosine transform (DCT) is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), where in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common. The most common variant of discrete cosine transform is the type-II DCT, which is often called simply "the DCT"; its inverse, the type-III DCT, is correspondingly often called simply "the inverse DCT" or "the IDCT

The DCT, and in particular the DCT-II, is often used in signal and image processing, especially for lossy data compression, because it has a strong "energy compaction" property: most of the signal information tends to be concentrated in a few low-frequency components of the DCT, approaching the Karhunen-Loève transform (which is optimal in the decorrelation sense) for signals based on certain limits of Markov processes

DCT-II

$$X_k = \sum_{n=0}^{N-1} x_n \cos[\Pi / N(n+1/2)k] \qquad k = 0, 1 \ldots\ldots\ldots\ldots N$$

4.2BASIC CORDIC PROCESSOR

A basic CORDIC processor should contain function modules which realize the CORDIC iterations specified, the angle update iteration specified and the scaling equation specified.

The modules support a single CORDIC iteration. It contains dual barrier shifters and dual adders to facilitate the updating of both x (i) and y (i) simultneously.The number of bits to be shifted is controlled by the shift sequence[s(m,i)].The shift sequence can be stored on chip using ROM (read only memory) or RAM(random access memory),or generated on chip with a simple counter and additional control devices. The add/subtract option is determined by the sequence $\{\mu_i\}$, which is determined by either the sign of z (i) or –x (i).y (i).

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

The scaling module can share the same processing unit with the CORDIC iteration module. By multiplexing the data paths the, this scaling module is able to perform the scaling iterations.

4.3PARALLEL AND PIPELINED ARRAYS

In a parallel CORDIC processor each of the n stages of the CORDIC iterations and s stages of scaling operations is realized with a dedicated basic CORDIC processor. By cascading the n+s basic CORDIC processors, we will be able to perform the entire CORDIC rotation operations including all the COIRDIC iterations and all the scaling operations in one clock cycle.

Let t_0 be the time needed for performing a single CORDIC iteration or a single scaling iteration. Then the total computational delay will be $(n+s)t_0$.Or equivalently, the signal processing throughput rate will be bounded by$(1/[(n+s)t_0]$.To increase the throughput ,one may choose to insert a latch between successive stages of a CORDIC processor and convert it into a pipelined CORDIC processor array. The latch is able to store the intermediate result after a CORDIC iteration or a scaling operation. Hence the computation of a CORDIC processor is isolated from the computation at the adjacent CORDIC processors. As a result these latches can be clocked at a period of time t0 time units which is the computational delay between adjacent stages. In other words the throughput rate is now increased (n+s) fold to 1/t0.However each individual data will take n+s clock cycles to complete. Hence the latency is $(n+s)t_0$ time units.

4.4ALGORITHM FOR DFT

```
Initiation (0, k) =0 for$0 \leq k \leq N-1$

For k = 0, 1,…… N-1 Do

For m= 0, 1,…….N-1 Do
```

```
$\begin{bmatrix} Y_r(m+1,k) \\ Y_i(m+1,k) \end{bmatrix} = K_1(n) \begin{bmatrix} \cos 2\Pi mk/N - \sin 2\Pi mk/N \\ \sin 2\Pi mk/N + \cos 2\Pi mk/N \end{bmatrix} \begin{bmatrix} x_r(m) \\ x_i(m) \end{bmatrix} + \begin{bmatrix} Y_r(m,k) \\ Y_i(m,k) \end{bmatrix}$

    End m-loop

    Y(k)=Y(N,k)/K1(n)   /* Scaling operation*/

End k loop
```

Here the sequence X(n) is taken from the left end of the processor array. Each X (n), which contains two real numbers x_r(n) and x_i(n),will be propagated from the current processor to the nearest neighbour processor in a pipelined manner. Since X (n) is not to be modified during propagation they can be piped at a rate of t_0 time unit (1 clock cycle) per stage. Each of the N CORDIC processors in this processor array will be responsible for evaluating a particular Y (k). This implies that N different rotation angles must be stored in each processor. Since each rotation angle requires n bits to store, the total storage requirement, including the s scaling iterations, will be nN +s bits.

Chapter 5

SIMULATION USING MATLAB

PROGRAM 1

```
% MATLAB CODE FOR CORDIC IN VECTOR MODE FOR TANGENT FUNCTIONS
clear all
x(1)=1;
y(1)=1;
z(1)=0;
A=1;
for i=1:25
   if y(i)<0
      d(i)=+1;
   else
      d(i)=-1;
   end
      x(i+1)=x(i)-y(i)*d(i)/2^(i-1);
   y(i+1)=y(i)+x(i)*d(i)/2^(i-1);
   z(i+1)=z(i)-d(i)*atan(2^(1-i));
   A=A*sqrt(1+2^(2*(1-i)));
   if y(i+1)==0
      break;
   end
    x(i+2)=x(i+1)/A;
    y(i+2)=y(i+1)/A;
end
for i=1:30
   plot(x,y,'*',x,y,'b-');
end
```

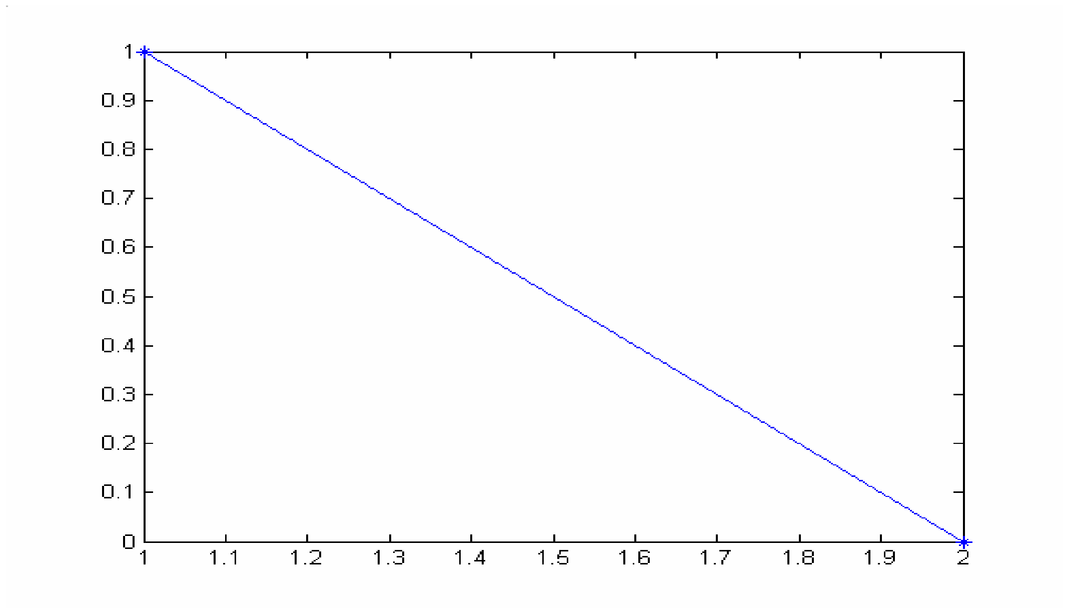


FIG 5.1 (SIMULATION RESULTR FOR CORDIC IN VECTOR MODE)

PROGRAM-2

```
% MATLAB CODE FOR CORDIC IN ROTATION MODE FOR SINE AND COSINE
FUNCTIONSl

x(1)=1;
y(1)=0;
z(1)=28;
A=1;
for i=1:100
   A=A*sqrt(1+2^(2*(1-i)));
end
   x(2)=A* (x(1)*cos(z(1))-y(1)*sin(z(1)));
   y(2)=A* (y(1)*cos (z(1))+x(1)*sin(z(1)));
   z(2)=0;
plot(x,y,'*',x,y,'b-')
```

OUTPUT

```
a =1              0
a =-1.5852    0.4461
```

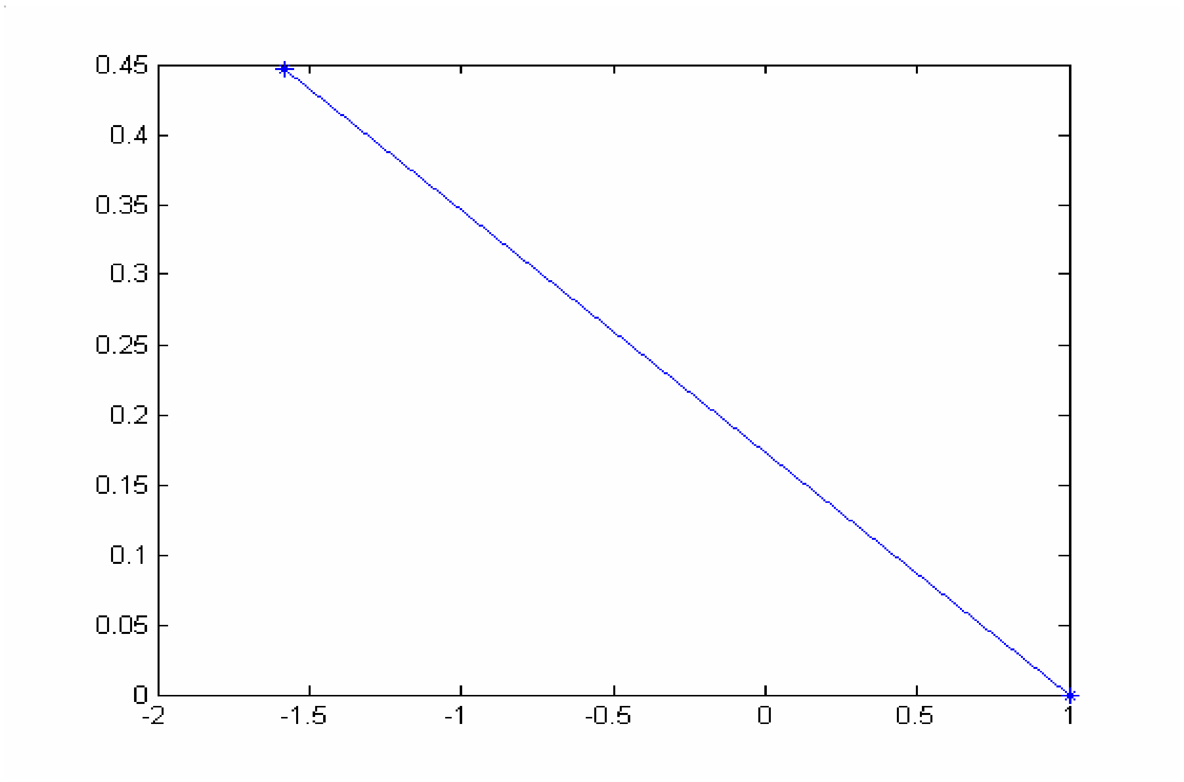


FIG 5.2(SIMULATION RESULT FOR CORDIC IN ROTATION MODE FOR SINE AND COSINE FUNCTIONS

PROGRAM 3

```
% MATLAB CODE FOR CORDIC IN ROTATION MODE
clear all
x(1)=1;
y(1)=0;
z(1)=0.5;
for i=1:25
if z(i)<0
      d(i)=-1;
 else
       d(i)=1;
   end
if i == 4 | 13 | 40
```

```
      x(i+1)=x(i)+y(i)*d(i)/2^(i);
      x(i)=x(i)+y(i)*d(i)/2^(i);
      y(i+1)=y(i)+x(i)*d(i)/2^(i);
      y(i)=y(i)+x(i)*d(i)/2^(i);
       z(i+1)=z(i)-d(i)*atanh(2^-i);
       z(i)=z(i)-d(i)*atanh(2^-i);
    else
       x(i+1)=x(i)+y(i)*d(i)/2^(i);
       y(i+1)=y(i)+x(i)*d(i)/2^(i);
       z(i+1)=z(i)-d(i)*atanh(2^-i);
    end
end
for i=1:15
         a=[x(i),y(i)]
        plot(x,y,'*',x,y,'b-')
end
```

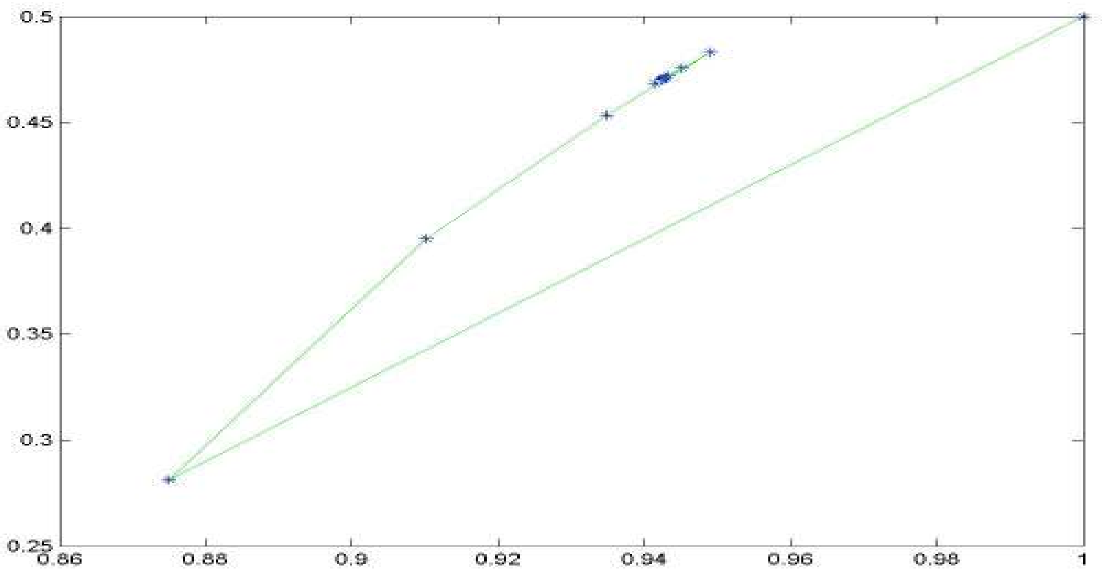


FIG 5.3(SIMULATION RESULT FOR CORDIC IN ROTATIONAL MODE)

PROGRAM 4

MATLAB CODE FOR CORDIC IN FOR HYPERBOLIC FUNCTIONS

```
x(1)=1;
y(1)=0;
z(1)=1;
for i=1:20
   if z(i)<0
      d(i)=-1;
   else
      d(i)=1;
   end
   x(i+1)=x(i)+y(i)*d(i)/2^(i);
   y(i+1)=y(i)+x(i)*d(i)/2^(i);
   z(i+1)=z(i)-d(i)*atanh(2^(-i));
   if z(i+1)==0.0
      break;
   end
end
figure
plot(x,y,'*',x,y,'b-')
```

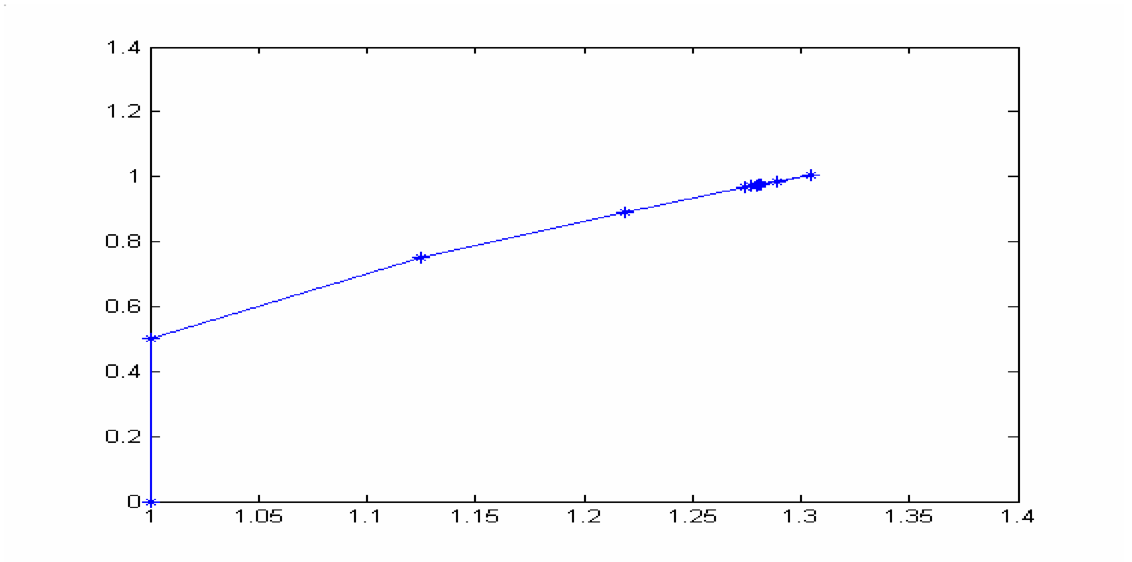


FIG 5.4(SIMULATION RESULT FOR HYPERBOLIC FUNCTIONS)

PROGRAM 5

MATLAB CODE FOR ROTATING ABOUT THE THREE AXES

```
clear all
%initialization of coordinates x1=1,y1=0,z1=3,xn=0.707,yn=0.707,zn=5;
x(1)=1;
y(1)=0;
z(1)=3;
u(1)=45;
v(1)=45;
A1=1;
A2=1;
% applying cordic equations in rotational mode
for i=1:20
   if u(i)<0
      d(i)=-1;
   else
      d(i)=1;
   end
   A1=A1*sqrt(1+2^(2*(1-i)));
   x(i+1)=x(i)-y(i)*d(i)/2^(i-1);
   y(i+1)=y(i)+x(i)*d(i)/2^(i-1);
   z(i+1)=z(i);
   u(i+1)=u(i)-d(i)*atan(2^(1-i));
end
%  scaling operation starts
x(22)=x(21)/A1;
y(22)=y(21)/A1;
z(22)=3;
% scaling operation ends
for i=22:40
   j=i-21;
   if v(j)<0
      d(i)=-1;
   else
```

```
        d(i)=1;
    end
    A2=A2*sqrt(1+2^(2*(1-j)));
    y(i+1)=y(i)+z(i)*d(i)/2^(1-j);
    z(i+1)=z(i)-y(i)*d(i)/2^(1-j);
    x(i+1)=x(i);
    v(j+1)=v(j)-d(i)*atan(2^(1-j));
end
    y(42)=y(41)/A2;
    z(42)=z(41)/A2;
    x(42)=x(41);
figure
plot3(x,y,z,'*',x,y,z,'b-')
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
figure
subplot(1,2,1)
plot3(x(1),y(1),z(1),'*',x(1),y(1),z(1),'b-')
grid on;
subplot(1,2,2)
plot3(x(42),y(42),z(42),'*',x(42),y(42),z(42),'b-')
grid on;
```

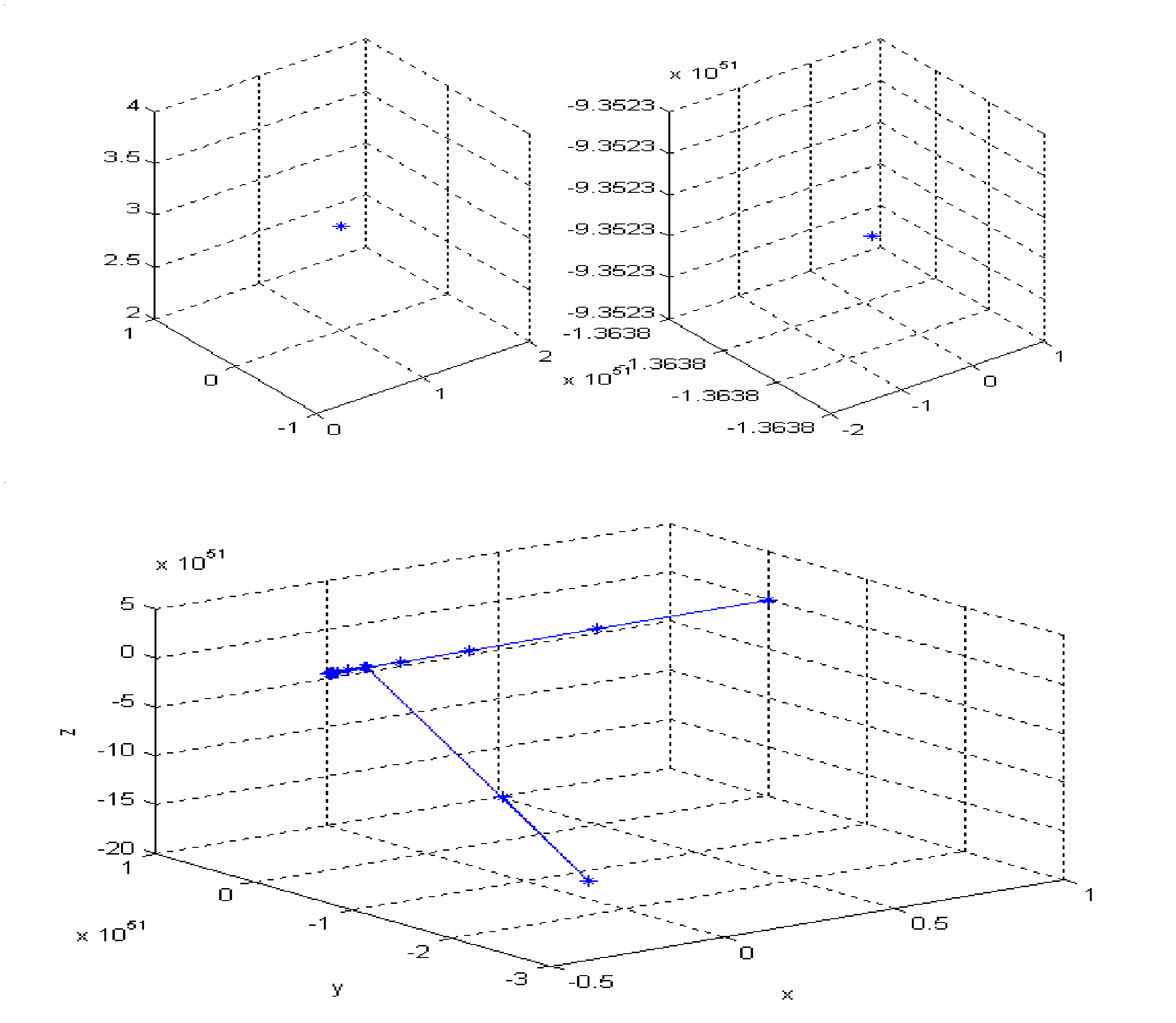


FIG 5.5(SIMULATION RESULT FOR ROTATING ABOUT THREE AXES)

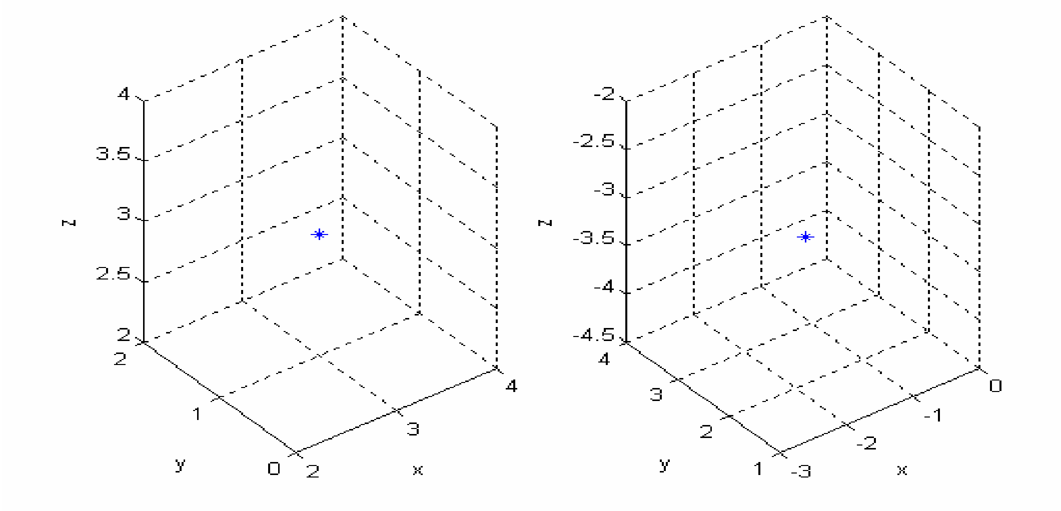
PROGRAM 6

MATLAB CODE FOR ROTATING ABOUT THREE AXES

```
clear all
x(1)=3;
y(1)=1;
z(1)=3;
u(1)=56;  %ATAND(YN/XN)-ATAND(Y1/X1)
v(1)=30;  %ATAND(SQRT(XN^2+YN^2)/ZN)-ATAND(SQRT(X1^2+Y1^2)/Z1)
A=1;
for i=1:20
   if u(i)<0
```

```
        d(i)=-1;
    else
        d(i)=1;
    end
    A=A*sqrt(1+2^(2*(1-i)));
    x(i+1)=x(i)-y(i)*d(i)/2^(i-1);
    y(i+1)=y(i)+x(i)*d(i)/2^(i-1);        %CORDIC EQUATIONS FOR VECTOR ROTATING
AROUND Z AXIS
    z(i+1)=z(i);
    u(i+1)=u(i)-d(i)*atan(2^(1-i));
end
  i=i+2;
 x(22)=x(21)/A;
 y(22)=y(21)/A;                %SCALING OPERATION
 z(22)=3;
 B=1;
for j=1:22
    if v(j)<0
        e(j)=-1;
    else
        e(j)=1;
    end
    B=B*sqrt(1+2^(2*(1-j)));
    x(j+i)=x(j+i-1);
    y(j+i)=y(j+i-1)+z(j+i-1)*e(j)/2^(j-1);      %CORDIC EQUATIONS FOR VECTOR
ROTATING AROUND X AXIS
    z(i+j)=z(j+i-1)-y(j+i-1)*e(j)/2^(j-1);
    v(j+1)=v(j)-e(j)*atan(2^(1-j));
end
z(45)=z(44)/B;
y(45)=y(44)/B;              %SCALING OPERATION
x(45)=x(44);
figure
```

```
plot3(x,y,z,'*',x,y,z,'b-')
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
figure
subplot(1,2,1);
plot3(x(1),y(1),z(1),'*',x(1),y(1),z(1),'b-')
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
subplot(1,2,2);
plot3(x(45),y(45),z(45),'*',x(45),y(45),z(45),'b-')
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
```



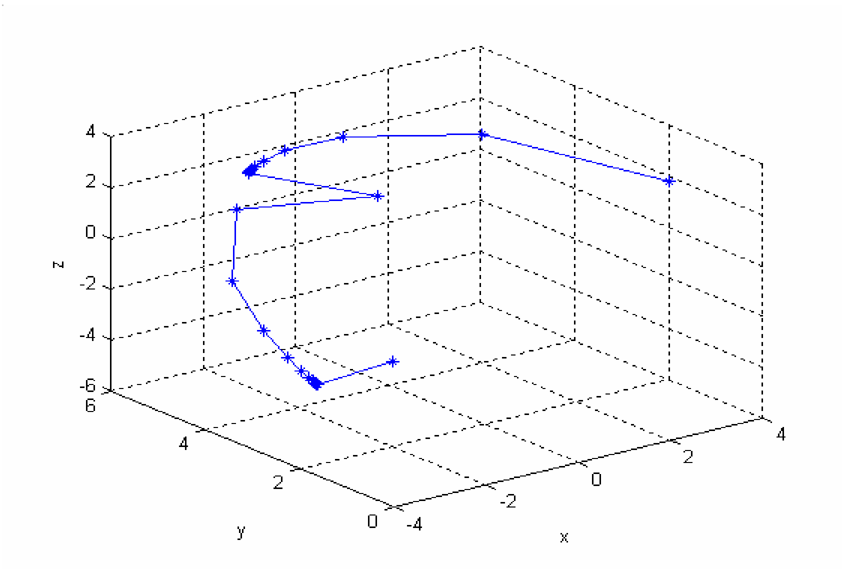


FIG 5.6(SIMULATION RESULT FOR ROTATING ABOUT THREE AXES)

PROGRAM 7

MATLAB PROGRAM FOR COMPUTING 2_DIMENSIONAL DCT

```
clc
clear all
close all
if (input(' WANT  TO ENTER MATRIX FROM COMMAND LINE(1 for Y/2 for N)   ')==1);
  x= input('ENTER THE ORDER OF THE MATRIX')
  disp('enter the elements of the matrix')
  for i=1:x
     for j=1:x
     A(i,j)=input(' ');
     end
  end
else
   x= input('ENTER THE ORDER OF THE MATRIX')
    A=rand(x);
end
disp('input matrix ')
B=A
for i=1:x                          % implementation of 1-D DCT row wise
```

```
 A(1,i)=B(1,i)+B(4,i);
 A(2,i)=B(2,i)+B(3,i);
 A(3,i)=B(1,i)-B(4,i);
 A(4,i)=B(3,i)-B(2,i);
end
B=A;
for i=1:x
   [A(3,i),A(1,i)]=cordicdct(B(1,i),B(2,i),45);
   [A(2,i),A(4,i)]=cordicdct(B(3,i),B(4,i),22.5);
end
disp('matrix after applying 1-D DCT column wise')
B=A
for i=1:x                    % implementation of 1-D DCT row wise
   A(i,1)=B(i,1)+B(i,4);
   A(i,2)=B(i,2)+B(i,3);
   A(i,3)=B(i,1)-B(i,4);
   A(i,4)=B(i,3)-B(i,2);
end
B=A;
for i=1:x
   [A(i,3),A(i,1)]=cordicdct(B(i,1),B(i,2),45);
   [A(i,2),A(i,4)]=cordicdct(B(i,3),B(i,4),22.5);
end
disp('matrix after applying 1-D DCT ROW wise')
A
function [p,q]= cordicdct(a,b,theta)
x(1)=a;
y(1)=b;
z(1)=theta;
A=1;
for i=1:25
   if z(i)<0
       d(i)=-1;
```

```
    else
        d(i)=1;
    end
    x(i+1)=x(i)-y(i)*d(i)/2^(i-1);
    y(i+1)=y(i)+x(i)*d(i)/2^(i-1);
    z(i+1)=z(i)-d(i)*atand(2^(1-i));
     A=A*sqrt(1+2^(2*(1-i)));
end
x(27)=x(26)/A;
y(27)=y(26)/A;
p=x(27);
q=y(27);
input matrix
B =
     1     2     3     4
     2     3     4     1
     3     4     1     2
     4     1     2     3
Matrix after applying 1-D DCT column wise
B =
    7.0711    7.0711    7.0711    7.0711
   -3.1543    0.5412    2.0719    0.5412
    0.0000   -2.8284    0.0000    2.8284
   -0.2242    1.3066   -2.3890    1.3066
Matrix after applying 1-D DCT ROW wise
A =
   20.0000    0.0000    0.0000   -0.0000
    0.0000   -4.0000   -3.6955    0.0000
    0.0000   -3.6955    4.0000    1.5307
   -0.0000    0.0000    1.5307   -4.0000
clc
clear all
close all
```

```
if (input(' WANT  TO ENTER MATRIX FROM COMMAND LINE(1 for Y/2 for N)   ')==1);
  x= input('ENTER THE ORDER OF THE MATRIX')
  disp('enter the elements of the matrix')
  for i=1:x
     for j=1:x
     A(i,j)=input(' ');
     end
  end
else
   x= input('ENTER THE ORDER OF THE MATRIX')
    A=rand(x);
end
disp('matrix after applying 1-D DCT column wise')
B=dct(A)
C=B';
disp('matrix after applying 1-D DCT row wise')
A=dct(C);
A=A'
```

```
Matrix after applying 1-D DCT column wise
B =
    5.0000    5.0000    5.0000    5.0000
   -2.2304    0.3827    1.4651    0.3827
    0         -2.0000   0         2.0000
   -0.1585    0.9239   -1.6892    0.9239
Matrix after applying 1-D DCT row wise
A =  10.0000     0         0         0
        0       -2.0000   -1.8478    0.0000
        0       -1.8478    2.0000    0.7654
        0.0000   -0.0000    0.7654   -2.0000
```

CONCLUSION

The CORDIC algorithms presented in this project are well known in the research and super computing circles. The trigonometric CORDIC algorithms were originally developed as a digital solution for real time navigation problems. The original work is credited to Jack Volder .The CORDIC algorithm has found its way into diverse applications including the 8087 math coprocessor, the HP-35 calculator, radar signal processors and robotics.CORDIC rotation has also been proposed for computing Discrete Fourier[4],Discrete Cosine[4],Singular Value Decomposition[5],and solving linear systems[1].

Here the basic CORDIC algorithm and a partial list of potential applications s of a CORDIC based processor array to digital signal processing is presented. The CORDIC based DCT architecture for low power design has been proposed. The proposed multiplier less CORDIC based DCT architecture produces high throughput and is easy to implementing VLSI. The proposed architecture reduced the input data range for the CORDIC processor by split and the no of compensation iterations in CORDIC based DCT computation by utilizing that most images have similar neighboring pixels. The project also shows that a tool is available for use in FPGA based computing machines, which are the likely basis for the next generation DSP systems. Its basis of application in DSP has been thoroughly investigated.

The day is not far away when many of the software algorithms will be replaced by the hardware efficient algorithms paving way for reduced complexity and faster operation.

REFERENCES

[1] J.E. Volder.”The CORDIC trigonometric computing technique,”IRE Trans.Electron.. Comput. Vol. EC-8,no 3,pp.335-339,Sept 1959

[2]B.G.Lee,"A new algorithm to compute the discrete cosine transform,"IEEE transactions on Acoustics,Speech and Signal Processing,vol ASSP-32,no. 6,pp. 1234-1245,Dec 1984

[3]Despain, A.M., "Fourier Transform Computations Using CORDIC Iterations,"IEEE Transactions On Computers,Vol.23,1974,pp. 993-1001

[4]A.Peled and B.Liu,"A newhardware realization of digital filters,"IEEE Trans. Acoust.,Speech,Signal Processing,vol, ASSP-22,PP. 456-462,Dec. 1974

[5]N. Weste and K. Eshraghian, Principles of CMOS VLSI De-sign-A Systems Perspective,2nd ed Reading,MA Addison-Wesley,1993

[6]A. P. Chandrakasan and R.W . Brodersen,Low-Oiwer CMOS Design.Piscataway. NJ: IEEE Press,1998. Ed..

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

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

[9]Andraka,R.J., "Building a High Performance Bit-Serial processor in an FPGA,"Proceedings of Design SuperCon'96,Jan 1996,pp5.1 – 5.21

[10]Duh, W.J., and Wu, J.L.,"Implementing the Discrete Cosine Transform by Using CORDIC Techniques,"Proceedings The International Symposium on VLSI Technology,Systems and Applications,Taipei,Taiwan,1989,pp. 281-285

[11]Wang,S. aned Piuri,V., "A unified View of CORDIC Processor Design",Application Specific Processors,Edited by Earl E.Swartzlander,Jr., Ch. 5, pp. 121-160,Kluwer Academic Press,November 1996

[12]Walther,J.s., "A unified algorithm for elementary functions,"Spring Joint Computer Conf.,1971,proc.,pp.379-385