Design and implementation of multiplier for complex numbers using CORDIC (Coordinate Rotation Digital Computer)

Architecture

1M Manimaraboopathy, 2G A Sathish Kumar, 3 M Anto Bennet

1,3 Electronics and Communication Engineering, VEL TECH, Chennai, Tamil Nadu, India.
2Electronics and Communication Engineering, Sri Venkateswara College of Engineering, Sripurempudur Chennai, Tamil Nadu, India.

Email: manimaraboopathy@veltechengg.com

Abstract

CORDIC architecture provides a simple way for calculating various trigonometric, linear and non-linear functions by using simple shift and add/subtract process. In recent days, the computation of complex numbers is much required in signal processing and related areas. A multiplier used for complex multiplication requires separate processor to compute cosine and sine values. This paper presents how CORDIC algorithm can be used to simplify the calculation of complex multiplication in a simple iterative algorithm to calculate multiplication of numbers and trigonometric functions using a single processor.

key words: CORDIC, complex, multipliers.

Introduction

CORDIC (Coordinate Rotation Digital Computer) is also known the digit-by-digit method and Volder's algorithm. This algorithm provides a simple and efficient way to calculate hyperbolic and trigonometric functions with less complexity. It is commonly used when no hardware multiplier is available. This is because of the fact that it uses only addition, subtraction, bit shift and table lookup to do operations. The modern CORDIC algorithm was first described in 1959 by Jack E. Volder [1]. It was developed at the aero electronics department of Convair to replace the analog resolver in the B-58 bomber’s navigation computer. Later the algorithm was further generalised allowing it to calculate hyperbolic and exponential functions, logarithms, multiplications, divisions, and square roots. CORDIC is particularly well-suited for handheld calculators, an application for which cost is much more important than speed (namely the chip gate count must be minimized). CORDIC is not the fastest way to perform these operations, its hardware design is simple compared to conventional multiplier circuits and uses simple iterative algorithm to calculate many functions like trigonometric, logarithmic and transcendental elementary functions, complex multiplications, Eigen value computation, matrix inversion, solution of linear equation systems and so on. The algorithm is being advanced through ages and the application reaches beyond just multiplication and minor calculation to the calculation of transforms and series by simple shift add iterative process. The complex multiplication can also be realised by using the CORDIC architecture and is as simple as the design of basic CORDIC. It needs only a minor modification that the product is calculated in polar form instead of rectangular form to enable the use of CORDIC algorithm [4,5,6].

Basic Cordic Technique

In this section, we are going to discuss the basic principle of CORDIC-based computation and the iterative algorithm used to calculate sine and cosine functions either in rotation or hyperbolic mode of operation.

Rotation Mode

A two dimensional vector \( \mathbf{v}_0 = [x_0, y_0] \) is rotated about an angle \( \beta \), to get the rotated vector \( \mathbf{v}_n = [x_n, y_n] \). This could be represented as the matrix product given as \( \mathbf{p}_n = \mathbf{R} \mathbf{p}_0 \) where \( \mathbf{R} \) is the rotation matrix: This explanation shows the usage of CORDIC in rotation mode to calculate sine and cosine of an angle, \( \beta \), and assumes the desired angle is given in radians and the angle is represented in a fixed point format. To determine the sine or cosine value for an angle \( \beta \), the x or y coordinate of a point must be on the unit circle corresponding to the desired angle to be found. Using CORDIC, we would start with the vector \( \mathbf{v}_0 \) and it is given as below:

\[
\mathbf{v}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}
\]  

(1)
In first iteration, the vector $V_0$ is rotated by $45^\circ$ counterclockwise to get the vector $V_1$. Successive iterations rotate the vector in either direction by size-decreasing steps, until it reaches the desired angle. Step $i$ size is $\arctan \left(\frac{1}{2^{i-1}}\right)$ for $i = 1, 2, 3, \ldots$.

More formally, we can say for every iteration, the algorithm calculates a rotation, which is performed by multiplying the vector $V_i$ with the rotation matrix $R_i$:

$$v_i = R_i v_{i-1} \tag{2}$$

The rotation matrix is given by:

$$R_i = \begin{bmatrix} \cos \gamma_i & \sin \gamma_i \\ -\sin \gamma_i & \cos \gamma_i \end{bmatrix}$$

By using the following two trigonometric identities:

$$\cos \phi = \frac{1}{\sqrt{1 + \tan^2 \phi}}$$
$$\sin \phi = \frac{\tan \phi}{\sqrt{1 + \tan^2 \phi}}$$

the rotation matrix $R_i$ becomes:

$$R_i = \frac{1}{\sqrt{1 + \tan^2 \gamma_i}} \begin{bmatrix} 1 & -\tan \gamma_i \\ \tan \gamma_i & 1 \end{bmatrix} \tag{3}$$

The expression for the rotated vector

$$v_i = R_i v_{i-1} \tag{4}$$

then becomes:

$$v_i = \frac{1}{\sqrt{1 + \tan^2 \gamma_i}} \begin{bmatrix} 1 & -\tan \gamma_i \\ \tan \gamma_i & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{5}$$

Where $X_{i-1}$ and $Y_{i-1}$ are the components of $V_{i-1}$. By restricting the angles $\gamma_i$, $\tan \gamma_i$ takes on the values $\pm 2^{-i}$, and the multiplication with the tangent could be replaced by a division by a power of two, which is efficiently done in digital computer hardware using a single bit shift. The expression then becomes:

$$v_i = k_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \tag{6}$$

Where

$$k_i = \frac{1}{\sqrt{1 + 2^{-2i}}}$$

and $\sigma_i$ can have the values of $-1$ or $1$, and it is used to determine the direction of rotation of the successive iterations; if the angle $\beta$ is positive then $\sigma_i$ becomes $+1$, otherwise it is $-1$. The scaling factor $k_i$ can be ignored in the iterative process and then applied afterward computing it separately using the formula given below:

$$K(n) = \prod_{i=0}^{n-1} k_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{7}$$

which is calculated in advance and stored in a table. For fixed number of iterations the value of scaling factor is also fixed. This correction could also be made in advance, by scaling $V_0$ before iterative process and hence saving a multiplication. Additionally it can be noted that:

$$k = \lim_{n \to \infty} K(n) \approx 0.6072529350088812561694$$

This allows further reduction of the algorithm’s complexity. After a sufficient number of iterations for instance say 40 iterations, the vector’s angle will be close to the wanted angle $\beta$. 40 iterations ($n = 40$) is more than enough in most cases and it obtains the correct result up to the 10th decimal place. The last task left is to determine whether the rotation should be clockwise or counterclockwise at each iteration (by choosing the value of $\sigma_i$). This can be done simply by keeping track of how much the angle was rotated in every iteration and subtracting that angle from the wanted angle; then as we want to get closer to the wanted angle $\beta$, if $\beta_{n+1}$ is positive, the rotation is in clockwise direction, otherwise it is negative and the rotation will be in counterclockwise direction.

$$\beta_i = \beta_{i-1} \cdot \sigma_i \gamma_i \cdot \gamma_i = \arctan 2^{-i}$$

Fig:1 An illustration of the CORDIC algorithm in progress.
Vectoring Mode

The vectoring-mode of operation requires only a slight modification in the algorithm of rotation mode. It starts with a vector \( x \) having positive coordinate values and the \( y \) coordinate is arbitrary. Successive rotations have the rotate the vector to the \( x \) axis (and therefore reducing the \( y \) coordinate to zero). At each step, the sign of \( y \) determines the direction of the rotation whether it is clockwise or anticlockwise. The final value of \( \beta_1 \) will contain the total angle of rotation and \( x \) the magnitude of the original vector scaled by \( K \). So, a clear use of the vectoring mode is that the transformation from rectangular to polar coordinates.

Multiplication Of Complex Numbers

The multiplication of complex numbers requires multipliers and adders to compute the final result of multiplication. Two types of multipliers are used for this computation namely parallel and serial multipliers.

Serial Multiplier

The serial multipliers are used to multiply two numbers in a bitwise fashion by an iterative shift and add process to compute the final product. It requires \( n \) number of iterations to produce the result of multiplication of \( n \)-bit numbers.

Parallel Multiplier

The parallel multipliers are inspired from serial multiplier and can compute the products of numbers in a parallel way thus computing two or more products simultaneously. A parallel multiplier can be built from combination of half and full adders and logic gates as proposed by Baugh Wooley. The fig3 shows a 4-bit multiplier.

Complex Multiplication

Consider the following complex numbers

\[ V = X + jY \]
\[ W = M + jN \]

The multiplication of the above given complex can be represented as given below and it requires four multiplications as we said earlier

\[ V\times W = (XN + MY) + j(YN - XM) \] (8)

Instead we can represent the above numbers in polar form and multiplication of two numbers in polar form is given below:

\[ V = A e^{j\theta_1} \text{ and } W = B e^{j\theta_2} \]

Product \( P = V\times W = |A||B|e^{j\theta} \)

Where, \( \theta = \theta_1 + \theta_2 \)

The above equation can be used to find the rectangular co-ordinates of a product of numbers and separately find the results of real and imaginary parts as given below

\[ \text{Real}(P) = |A||B|\cos\theta \]
\[ \text{Im}(P) = |A||B|\sin\theta \]

The sine and cosine functions and the result could be calculated directly by using CORDIC algorithm by rotating the product of \( A \) and \( B \) through an angle \( \Theta \). This results in lesser complexity as compared to conventional serial and parallel multipliers.
Proposed Model

The proposed model computes the product of complex numbers through the use of CORDIC algorithm with simple hardware unit by use of shift and add/subtract iterative process. The hardware architecture of CORDIC is given below.

![Fig: 4 basic CORDIC architecture](image)

In the above architectural diagram mux is used to initiate CORDIC by providing user input values and then after one iteration it allows previous values to pass through by changes in the selection line. There are separate blocks available for calculation of the angle value which is required to determine the direction of rotation.

This requires almost 40 iteration to get approximation up to $10^{th}$ decimal value which is necessary for majority cases. But such a long iteration makes system slower and CORDIC architecture is being redesigned with pipelining to produce almost an out per iteration when a number of calculations are involved.

![Fig: 5 pipelined CORDIC architecture](image)

The CORDIC architecture can compute the complex multiplication by providing the product $AB$ as input for $x_0$ or $y_0$ based on condition whether to find sine or cosine of the function, i.e. the real or imaginary part of the product. The final output after a considerably sufficient number of iterations gives the imaginary or real part of the product based on the input given to it.

Thus the proposed system is capable of calculating the product of two complex numbers with use of simple circuits adders, subtractors and shifters and is a most efficient way compared to complex multiplier circuits. It is a best alternative when hardware multipliers are not available.

Result

We have executed the proposed system in model sim software and obtained the following results. The fig 6 shows the analysis of waveforms generated when a basic program for multiplying complex numbers using conventional multiplier was executed in the model sim. It can be seen that the output generated takes a long time as it uses a multiple number of iterations to produce the required result of product of the given complex numbers.

![Fig: 6 output samples for normal complex multiplication](image)

The fig 7 shows the output generated when the same process is executed with the help of CORDIC instead of using multipliers. By analysing both the outputs we can say that the computation using a simple multiplier takes much complexity compared to CORDIC. It is also evident that CORDIC can execute complex multiplication through its simple iterative algorithms and with much less hardware requirement.

![Fig: 7 output samples using CORDIC](image)
Conclusion

In this brief we have proposed a model which could efficiently perform complex multiplication with less complexity in hardware design. It could also be used in signal processing and related fields as they use complex numbers for most of their applications. In future the algorithm could be extended to be used in various transforms for more efficient way of computing.

References