IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 11, Issue 2, Ver. III (Mar-Apr .2016), PP 52-57
www.iosrjournals.org

Efficient Pipelined CORDIC Architecture for Generation of Sine and Cosine Functions

T Srinivasa Rao[1], V Bharathi Devarakonda[2]
[1]*(Department of ECE, Aditya Engineering college, Surampalem, India)*
[2]*(Department of ECE, JNTU College of Engineering, Kakinada, India)*

Abstract :*The* ***CORDIC****(***CO****ordinate* ***R****otational* ***DI****gital* ***C****omputer) algorithm which is capable of computing complex functions such as the trigonometric, hyperbolic and logarithmic functions, real and complex multiplications, division, square-root, solution of linear systems, eigenvalue estimation ,singular value decomposition, QR factorization and many others.CORDIC being used in various real time applications in the domain of Signal & Image processing, Communication Systems, Robotics and 3-D graphics and Biomedical Image & Signal Processing. The beauty of CORDIC lies in the fact that by simple shift-add operations and the CORDIC has gained momentum for decades because of its less hardware complexity. CORDIC algorithm is very simple and iterative process for performing various mathematical computations. Most of the literature lacks in calculation of resources utilized by a particular CORDIC architecture. In this paper, serial, parallel and pipelined CORDIC architecture has been implemented for computing both sine & cosine functions.*
Keywords – *CORDIC, serial and parallel pipelined CORDIC, sine and cosine functions.*

I. INTRODUCTION

CORDIC algorithm is an iterative algorithm, which can be used for the computation of trigonometric functions, multiplication and division [1]. Last half century has witnessed a lot of progress in design and development of architectures of the algorithm for high-performance and low-cost hardware solutions. CORDIC algorithm got its popularity, when [2] showed that, by varying a few simple parameters, it could be used as a single algorithm for unified implementation of a wide range of elementary transcendental functions involving logarithms, exponentials, and square. During the same time, [3] showed that CORDIC technique is a better choice for scientific calculator applications.

The popularity of CORDIC was very much enhanced thereafter primarily due to its potential for efficient and low-cost implementation. With the advent of low cost, low power FPGAs, this algorithm has shown its potential for efficient and low-cost implementation. CORDIC algorithm can be widely used in as wireless communications, Software Defined Radio and medical imaging applications, which are heavily dependent on signal processing.

Although CORDIC may not be the fastest technique to perform these operations, yet it is attractive due to the simplicity and efficient hardware implementation.

The development of CORDIC algorithm and architecture has taken place for achieving high throughput rate and reduction of hardware-complexity as well as the latency of implementation. Latency of implementation is an inherent drawback of the conventional CORDIC algorithm. Angle recoding schemes and higher radix CORDIC have been developed for reduced latency realization. Parallel and pipelined CORDIC have been suggested for high-throughput computation. CORDIC computation is inherently sequential due to two main bottlenecks firstly the micro-rotation for any iteration is performed on the intermediate vector computed by the previous iteration and secondly the $(i+1)^{th}$ iteration could be started only after the completion of the i^{th} iteration, since the value of which is required to start the $(i+1)^{th}$ iteration could be known only after the completion of the i^{th} iteration. To alleviate the second bottleneck some attempts have been made for evaluation of values corresponding to small micro-rotation angles [4]. However, the CORDIC iterations could not still be performed in parallel due to the first bottleneck. A partial parallelization has been realized in [4] by combining a pair of conventional CORDIC iterations into a single merged iteration which provides better area-delay efficiency. But the accuracy is slightly affected by such merging and cannot be extended to a higher number of conventional CORDIC iterations since the induced error becomes unacceptable [5]. Parallel realization of CORDIC iterations to handle the first bottleneck by direct unfolding of micro-rotation is possible, but that would result in increase in computational complexity and the advantage of simplicity of CORDIC algorithm gets degraded [6]. Although no popular architectures are known to us for fully parallel implementation of CORDIC, different forms of pipelined implementation of CORDIC have however been proposed for improving the computational throughput [7].To handle latency bottlenecks, various architectures have been developed and reported in this review. Most of the well-known architectures could be grouped under bit parallel iterative CORDIC, bit parallel unrolled CORDIC, bit serial iterative CORDIC architecture.

II. CORDIC ALGORITHM

Keeping the requirements and constraints of different application environments in view, the development of CORDIC algorithm and architecture has taken place for achieving high throughput rate and reduction of hardware-complexity as well as the latency of implementation. Some of the typical approaches for reduced-complexity implementation are focused on minimization of the complexity of scaling operation and the complexity of barrel-shifter in the CORDIC engine. Latency of implementation is an inherent drawback of the conventional CORDIC algorithm. Parallel and pipelined CORDIC have been suggested for high-throughput computation and efficient CORDIC algorithm.

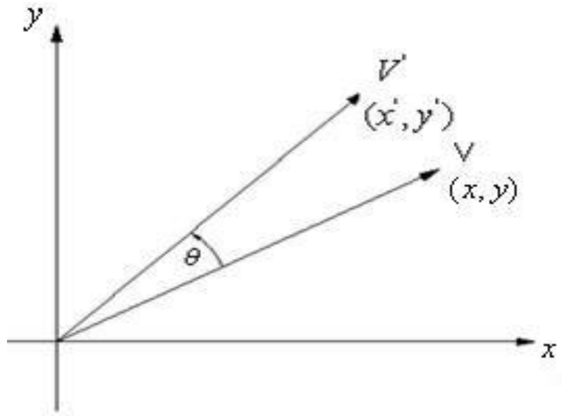

Figure 1: Vector Rotation

CORDIC algorithm has two types of computing modes Vector rotation (Rotating mode) and vector translation (Vectoring mode). The CORDIC algorithm was initially designed to perform a vector rotation, where the vector V with components (x, y) is rotated through the angle θ □ yielding a new vector V' with component (x', y') shown in Figure 1.

V′=[R][V] where R is the rotation matrix. (1)

$$R=\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad (2)$$

$$V'=\begin{bmatrix} x' \\ y' \end{bmatrix}=\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \quad (3)$$

Individual equations for x' and y' can be rewritten as:

$$x' = x.\cos(\theta) - y.\sin(\theta) \quad (4)$$

$$y' = y.\cos(\theta) + x.\sin(\theta) \quad (5)$$

And arranged so that

$$x' = \cos(\theta)[x - y.\tan(\theta)] \quad (6)$$

$$y' = \cos(\theta)[y + x.\tan(\theta)] \quad (7)$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\theta)$ are restricted so that $\tan(\theta) = 2^{-i}$ .In digital hardware this denotes a simple shift operation. Furthermore, if those rotations are performed iteratively and in both directions every value of $\tan(\theta)$ is represented. With $\theta = \arctan(2^{-i})$ the cosine term could also be simplified and since $\cos(\theta) = \cos(-\theta)$ it is a constant for a fixed number of iterations. This iterative rotation can now be expressed as:

$$x_{i+1} = k_i[x_i - y_i.d_i.2^{-i} \quad (8)$$

$$y_{i+1} = k_i[y_i + x_i.d_i.2^{-i} \quad (9)$$

where $k_i = \cos(\arctan(2^{-i}))$ and $d_i = \pm 1$

The product of the k_i represents so called k-factor

$$K = \prod_{i=0}^{n-1} K_i \quad (10)$$

This K factor can be calculated in advance and applied elsewhere in the system. Equations (8) and (9) can now be simplified to the basic CORDIC equations:

$$x_{i+1} = [x_i - y_i.d_i.2^{-i}] \quad (11)$$

$$y_{i+1} = [y_i - x_i.d_i.2^{-i}] \quad (12)$$

The direction of each rotation is defined by d_i and the sequence of all d_i's determines the final vector. Each vector V can be described by both the vector length and angle or by its coordinates x and y. Following this incident, the CORDIC algorithm knows two ways of determining the direction of rotation: the rotation mode and the vectoring mode. Both methods initialize the angle accumulator with the desired angle. The rotation mode, determines the right sequence as the angle accumulator approaches 0 while the vectoring mode minimizes the y component of the input vector.

The angle accumulator is defined by:

$$z_{i+1} = z_i - d_i.\arctan(2^{-i}) \quad (13)$$

$$\theta = \sum_{i=0}^{\infty} d_i.\arctan(2^{-i}) \quad (14)$$

Those values of $\arctan(2^{-i})$ can be stored in a small lookup table or hardwired depending on the way of implementation. Since the decision is which direction to rotate instead of whether to rotate or not, sensitive to the sign of z_i .Therefore d_i can be described as:

$$1\, d_i = \begin{cases} -1 & if\ z_i < 0 \\ +1 & if\ z_i \le 0 \end{cases} \quad (15)$$

With equation (15) the CORDIC algorithm in rotation mode is described completely. Note, that the CORDIC method as described performs rotations only within -π/2 and π/2. This limitation comes from the use of 2^0 for the tangent in the first iteration. However, since a sine wave is symmetric from quadrant to quadrant, every sine value from 0 to 2π can be represented by reflecting and/or inverting the first quadrant appropriately. In vector translation, rotates the vector V with component (X, Y) around the circle until the Y component equals zero as illustrated in Figure 2. The outputs from vector translation are the magnitude X' and phase z' of the input vector V with component (X, Y).

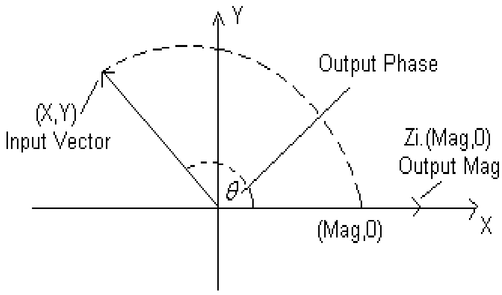

Figure2: Vector Translation

After vector translation, output equations are:

$$x' = k_i\sqrt{(x^2 + y^2)} \quad (16)$$

$$y' = 0 \quad (17)$$

$$z' = a\tan\left(\frac{y}{x}\right) \quad (18)$$

To achieve simplicity of hardware realization of the rotation, the key ideas used in CORDIC arithmetic are to decompose the rotations into a sequence of elementary rotations through predefined angles that could be implemented with minimum hardware cost and to avoid scaling, that might involve arithmetic operation, such as square-root and division. The second idea is based on the fact the scale-factor contains only the magnitude information but no information about the angle of rotation.

In 1971, John S. Walther found how CORDIC iterations could be modified to compute hyperbolic functions and reformulated the CORDIC algorithm into a generalized and unified form which is suitable to perform rotations in circular, hyperbolic and linear coordinate systems. The unified formulation includes a new variable *m*, which is assigned different values for different coordinate systems. The generalized CORDIC is formulated as follows:

$$x_{i+1} = x_i - m.\sigma_i.2^{-i}.y_i$$
$$y_{i+1} = y_i + \sigma_i.2^{-i}.x_i \qquad (19)$$
$$w_{i+1} = w_i - \sigma_i.\alpha_i$$

where

$$\sigma_i = \begin{cases} sign(w_i) & for\ rotational\ \text{mode} \\ -sign(w_i) & for\ vectoring\ \text{mode} \end{cases}$$

$$\alpha_i = \tan^{-1}(2^{-i})$$

$$\alpha_i = \begin{cases} \tan^{-1}(2^{-i}) & for\ m = 1 \\ 2^{-i} & for\ m = 0 \\ \tanh^{-1}(2^{-i}) & for\ m = -1 \end{cases}$$

Table 1: Generalized CORDIC Algorithm

m	Rotation mode	Vectoring mode
0	$x_n = k(x_o \cos w_o - y_o \sin w_o)$	$x_n = k\sqrt{x_o^2 + y_o^2}$
	$y_n = k(x_o \sin w_o + y_o \cos w_o)$	$y_n = 0$
	$w_n = 0$	$w_n = w_0 + \tan^{-1}(y_o / x_0)$
1	$x_n = x_o$	$x_n = x_o$
	$y_n = y_o + x_o w_o$	$y_n = 0$
	$w_n = 0$	$w_n = w_0 + (y_o / x_0)$
-1	$x_n = k_h(x_o \cosh w_o - y_o \sinh w_o)$	$x_n = k_h\sqrt{x_o^2 + y_o^2}$
	$y_n = k_h(x_o \sinh w_o + y_o \cosh w_o)$	$y_n = 0$
	$w_n = 0$	$w_n = w_0 + \tanh^{-1}(y_o / x_0)$

The first paragraph under each heading or subheading should be flush left, and subsequent paragraphs should have a five-space indentation. A colon is inserted before an equation is presented, but there is no punctuation following the equation. All equations are numbered and referred to in the text solely by a number enclosed in a round bracket (i.e., (3) reads as "equation 3"). Ensure that any miscellaneous numbering system you use in your paper cannot be confused with a reference [4] or an equation (3) designation.

III. FPGA IMPLEMENTATION OF CORDIC ALGORITHM FOR SIN AND COS FUNCTIONS

CORDIC can be used to compute Sin of any angle θ with little variation. The angle is given as input. A vector length 1.647 (CORDIC gain) along the x-axis is taken. The vector is then rotated in steps so as to reach the desired input angle θ. The x and y values are accumulated. After fixed number of iterations the final co- ordinates of the vector i.e. the x and y values give value of cosine and sine respectively of the given angle θ. When the Sine or Cosine functional configuration is selected, the unit vector is rotated, using the CORDIC algorithm, by input angle θ. This generates the output vector $(\cos(\theta), \sin(\theta))$.The compensation scaling module is disabled for the Sin and Cos functional configuration as it is internally pre-scaled to compensate for the CORDIC scale factor.

Table 2: Resource Utilization of Sine and Cosine functional Configuration

Configuration → Parameters ↓	**Serial**	**Parallel (No Pipeline)**	**Parallel (Pipeline)**
No. of slice f/f	349 (22%)	66(4%)	1146 (74%)
No. of 4 i/p LUT's	472 (30%)	1006(65%)	1020(66%)
No. of occupied Slices	330 (42%)	564(73%)	623(81%)
No. of slices containing Only related logic	330(100%)	564(100%)	623(100%)
Total No. of 4 i/p LUT's	595 (38%)	1087(70%)	1123 (73%)

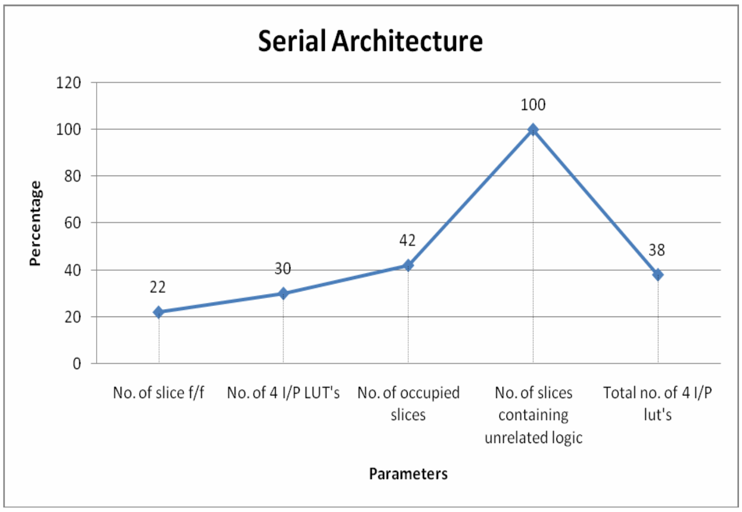

Figure 3: Resource Utilization of Sin and Cos functional Configuration in serial architecture

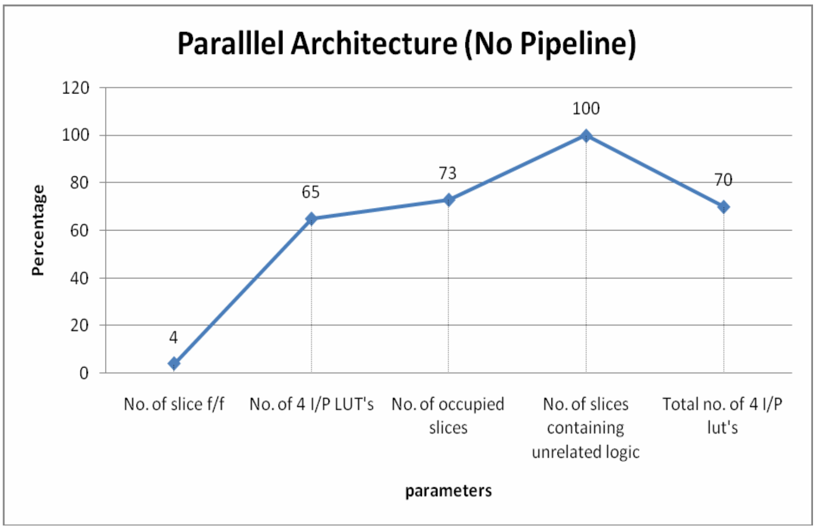

Figure 4: Resource Utilization of Sin and Cos functional Configuration in parallel architecture

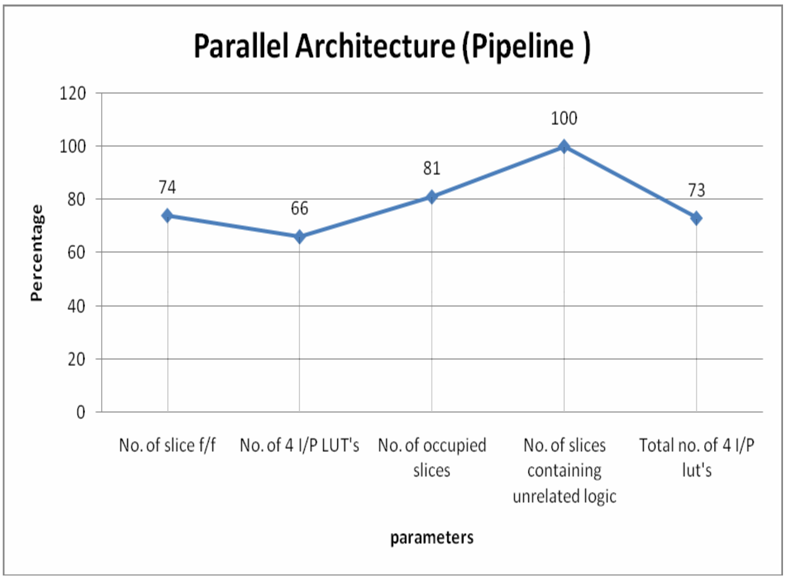

Figure 5: Resource Utilization of Sin and Cos functional Configuration in parallel architecture

From Table 2 and Figures 3-5, it has been concluded parallel architecture uses 74% no. of slices as compare to 4% no. of slices used by parallel architecture without pipelining mode and 22% no. of slices used by serial architecture. 66% no. of 4 input LUTs used by parallel architecture with pipeline mode but 65% no. of 4 input LUTs are used by parallel architecture without pipelining and 30% no. of 4 input LUTs are used by serial architecture. 81% and 73% occupied slices are used by parallel architecture with or without pipelining continuously and 42% occupied slices are used by serial architecture.

IV. CONCLUSION

From the above discussion, although parallel architecture with pipeline seems to be costlier as compare to parallel without pipelining and serial architecture, yet parallel architecture has high throughput (i.e. speed) as compare to serial architecture.

References

[1] J.E.Volder, "The CORDIC trigonometric computing technique," IRE Transactions on Electronic Computers, vol. EC- 8, pp. 330–334, Sept. 1959.
[2] J.S.Walther, "A unified algorithm for elementary functions," in Proceedings of the 38th Spring Joint Computer Conference, Atlantic City, NJ, 1971, pp.379–385.
[3] D.S.Cochran, "Algorithms and accuracy in the HP-35," Hewlett-Packard Journal, pp. 1–11, June 1972.
[4] S.Wang, V.Piuri, and J.E.E.Swartzlander, "Hybrid CORDIC algorithms, "IEEE Transactions on Computers, volume 46, no. 11,pp. 1202–1207, November1997.
[5] S.Wang and E.E.Swartzlander, "Merged CORDIC algorithm," in IEEE International Symposium on Circuits Systems (ISCAS'95),1995, volume 3, pp.1988–1991.
[6] B.Gisuthan and T.Srikanthan, "Pipelining flat CORDIC based trigonometric function generators," Microelectronics Journal, volume 33, Pp.77–89, 2002.
[7] E.Deprettere, P.Dewilde, and R. Udo, "Pipelined CORDIC architectures for fast VLSI filtering and array processing," in IEEE International Conference on Acoustic, Speech, Signal Processing, ICASSP'84, March 1984, volume 9, pp.250–253.
[8] D.E.Metafas and C.E.Goutis, "A floating point pipeline CORDIC processor with extended operation set," in IEEE International Symposium on Circuits and Systems, ISCAS'91, June 1991, volume 5, pp. 3066–3069.