

Simulation of Trigonometric Signals Using Area-Time Efficient Algorithm

Beera Jaya Bharathi¹ Nutakki Rajeswari² Dr. SaiTeja Chopparapu³ B Sraavan Kumar⁴ Sandiri Swetha⁵

^{1,2,3,4,5} Assistant Professor, Department of ECE, St. Peters Engineering College Hyderabad, India

Corresponding Author: beerajayabharathi@stpetershyd.com

ABSTRACT

In this paper, a parametric Co-ordinate Rotation Digital Computer (CORDIC) algorithm is presented, simulating fixed point arithmetic (sine, cosine, and arctangent) trigonometric functions evaluation. The Important design options for hardware implementation include iterative, unrolled and unrolled pipelined architectures of the CORDIC module. The design uses the VHDL '93 backwards-compatible version of the fixed point package, as defined according to the verilog 2008 standard. Hardware performance analysis results are presented in this paper for more than 75 circuit variations implemented on a Spartan 3 Xilinx FPGA, each for

different parameter values of the proposed CORDIC module. A maximum 47% increase of speed and 57% area reduction are accomplished, in comparison with other designs. In our simulation design we are using circular co-ordinate system for producing digital trigonometric functions and these are calculated in the two main modes in CORDIC algorithm which are rotation mode and vectoring mode.

Keywords -Coordinate Rotation Digital Computer (CORDIC), Cosine/Sine, Recursive Architecture, XILINX.

I. INTRODUCTION

Trigonometric functions are widely used in almost every application and there are many algorithms for producing digital trigonometric functions. CORDIC algorithm is capable of generating digital trigonometric functions by shifting and adding procedure. The abbreviation for CORDIC is CO-ordinate Rotation Digital Computer. Here the computation of trigonometric functions can be done in digital binary format by performing rotation of vectors in co-ordinate axis. The basic idea is embedding of elementary function evaluation as a generalized rotation operation and then the rotation operation is decomposed into successive basic rotations and then these basic rotations is implemented by using shift and add operations. This trigonometric iteration-based approach relies on vector rotations for performing successive mapping between polar and rectangular co-ordinates. Although not a LUT based strategy, the CORDIC still relies on predetermined phase, amplitude and frequency values for calculating points on a sine wave. The particular architecture hereby presented generates the phase of the sine by self on enable and performs the CORDIC vector rotation in order to produce sine wave values at the rate of 4096 samples per cycle. The

implementation of DSWG (Digital sinusoidal wave generation) was partitioned into two main blocks: a Sine magnitude generator (SMG) block and a CORDIC logic processor (CLP) block. The SMG produces the phase increments which drives the CLP, while performing replication in order to obtain the complete cycle of sine wave. Meanwhile, the CLP block performs the vector rotation and generates the sine magnitude. The CORDIC algorithm involves rotation of a vector on the XY-plane in circular, linear and hyperbolic coordinate systems depending on the function to be evaluated. CORDIC algorithm can be used for computing wide range of functions like trigonometric, logarithmic, hyperbolic and linear functions. CORDIC algorithm can be implemented in two modes, Rotation mode and Vectoring mode.

II. Basic equations of CORDIC algorithm

All the trigonometric functions can be computed or derived from functions using vector rotations. Vector rotation can also be used for 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 shift and add.

If a vector V with coordinates (x, y) is rotated through an angle ϕ then a new vector V' can be obtained with coordinates (x', y') where x' and y' can be obtained using x, y and ϕ by the following method.

$$X = r \cos \theta, Y = r \sin \theta \quad (1)$$

The algorithm, credited by Volder, is derived from the general rotation transform:

$$V' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cos \phi - y \sin \phi \\ y \cos \phi + x \sin \phi \end{pmatrix} \quad (2)$$

Which rotates a vector in a Cartesian plane by an angle ϕ .

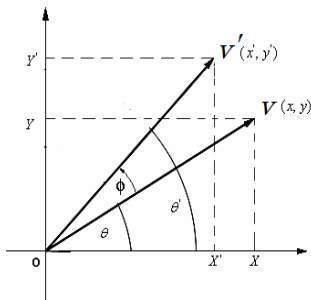


Figure 1: Rotation of a vector V by an angle ϕ

Let's find how the above equations came into picture. As shown in the figure 1, a vector $V(x, y)$ can be resolved in two parts along the x -axis and y -axis as $r \cos \theta$ and $r \sin \theta$ respectively. Figure 2 illustrates the rotation of a vector $V(x, y)$ by an angle ϕ .

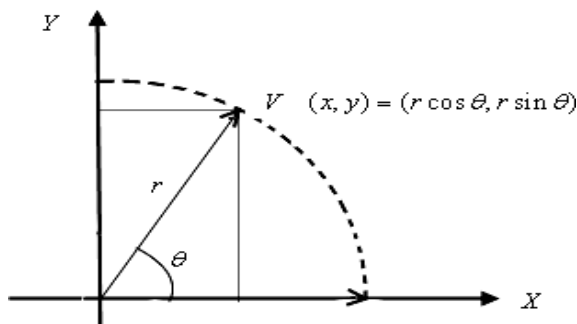


Figure 2: Vector V with magnitude r and phase θ
i.e., $x = r \cos \theta, y = r \sin \theta$

Similarly from figure 2 it can be seen that vector V and V' can be resolved into two parts. Let V has its magnitude and phase as r and θ respectively and V' has its magnitude and phase as r and θ' where V' came into picture after anticlockwise rotation of vector V by an angle ϕ . From figure 2.1 it can be observed:

$$\theta' - \theta = \phi \quad \text{then } \theta' = \theta + \phi$$

$$\begin{aligned} OX' = x' &= r \cos \theta' \\ &= r \cos(\theta + \phi) \\ &= r(\cos \theta \cos \phi - \sin \theta \sin \phi) \\ &= (r \cos \theta) \cos \phi - (r \sin \theta) \sin \phi \end{aligned} \quad (3)$$

Using the figure 2 OX' can be represented as:

$$OX' = x' = x \cos \phi - y \sin \phi \quad (4)$$

Similarly,

$$OY' = y' = y \cos \phi + x \sin \phi \quad (5)$$

The vector V' in the clockwise direction rotating the vector V by the angle ϕ and the equations obtain in this case be

$$x' = x \cos \phi + y \sin \phi \quad (6)$$

$$y' = x \sin \phi - y \cos \phi \quad (7)$$

The above equations can be represented in the matrix form as

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} \cos \phi & \mp \sin \phi \\ \pm \sin \phi & \cos \phi \end{pmatrix} \quad (8)$$

The individual equations for x and y' can be rewritten as:

$$x' = x \cos \phi \mp y \sin \phi \quad (9)$$

$$y' = y \cos \phi \pm x \sin \phi \quad (10)$$

Volder observed that by factoring out a $\cos \phi$ from both sides, resulting equation be in terms of the tangent of the angle ϕ , the angle of which we want to find the sin and cos. Next if it is assumed that the angle θ is being an aggregate of small angles and composite angles is chosen such that their tangents are all inverse powers of two, then this equation can be rewritten as an iterative formula.

$$x' = \cos \phi (x \mp y \tan \phi)$$

$$y' = \cos \phi (y \pm x \tan \phi)$$

$z' = z \pm \phi$, here ϕ is the angle of rotation (\pm sign is showing the direction of rotation) and z is the argument. For the ease of calculation here only rotation in anticlockwise direction is observed first.

$$x' = \cos \phi [x - y \tan \phi] \quad (11)$$

$$y' = \cos \phi [y + x \tan \phi] \quad (12)$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\phi)$ are restricted so that $\tan(\phi) = 2^{-i}$. In digital hardware this denotes a simple shift operation. Furthermore, if those rotations are performed iteratively and in both directions every value of $\tan(\phi)$ is representable. With $\phi = \arctan(2^{-i})$ the cosine term could also be simplified and since $\cos(\phi)$

= cos(-φ) it is a constant for a fixed number of iterations. This iterative rotation can now be expressed as:

$$x_{i+1} = k_i[x_i - y_i \cdot d_i \cdot 2^{-i}] \quad (13)$$

$$y_{i+1} = k_i[y_i + x_i \cdot d_i \cdot 2^{-i}] \quad (14)$$

Where, i denotes the number of rotations required to reach the required angle of the required vector, $k_i = \cos(\arctan(2^{-i}))$ and $d_i = \pm 1$. The product of the k_i 's represents the so called K factor:

$$K = \prod_{i=0}^{n-1} k_i \quad (15)$$

and θ is the angle of rotation for n times

TABLE 1: For 8-bit CORDIC hardware

i	$d^{-i} = 2^{-i} = \tan \phi_i$	$\phi_i = \arctan(2^{-i})$	ϕ_i in radian
0	1	45°	0.7854
1	0.5	26.565°	0.4636
2	0.25	14.036°	0.2450
3	0.125	7.125°	0.1244
4	0.0625	3.576°	0.0624
5	0.03125	1.7876°	0.0312
6	0.015625	0.8938°	0.0156
7	0.0078125	0.4469°	0.0078

k_i is the gain and its value changes as the number of iteration increases. For 8-bit hardware CORDIC approximation method the value of k_i is given as

$$k_i = \prod_{i=0}^7 = \cos \theta_1 \cdot \cos \theta_2 \dots \dots \dots \cos \theta_7 \quad (16)$$

$$= \cos 45^\circ \cdot \cos 26.565^\circ \dots \dots \dots \cos 0.4469^\circ$$

$$= 0.6073$$

From the above table it can be seen that precision up to 0.4469° is possible for 8-bit CORDIC hardware. These θ_i are stored in the ROM of the hardware of the CORDIC hardware as the look up table. Now by taking an example of balance it can be understood that how the CORDIC algorithm works.

B. Basic CORDIC iterations

To simplify each rotation, picking α_i (angle of rotation in i th iteration) such that $\alpha_i = d_i \cdot 2^{-i} \cdot d_i$ is such that it has value +1 or -1 depending upon the rotation i.e. $d_i \in \{+1, -1\}$. Then

$$x_{i+1} = x_i - d_i y_i 2^{-i} \quad (17)$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \quad (18)$$

$$z_{i+1} = z_i - d_i \tan^{-1} 2^{-i} \quad (19)$$

The computation of x_{i+1} or y_{i+1} requires an i -bit right shift and an add/subtract. If the function $\tan^{-1} 2^{-i}$ is pre-computed and stored in table (Table 2) for different values of i , a single add/subtract sufficient to compute z_{i+1} . Each CORDIC iteration thus involves two shifts, a table lookup and three additions. If the rotation is done by the same set of angles (with + or signs), then the expansion factor K , is a constant, and can be precomputed. For example to rotate by 30 degrees, the following sequence of angles be followed that add up to 30° degree.

$$30.0 \approx 45.0 - 26.6 + 14.0 - 7.1 + 3.6 + 1.8 - 0.9 + 0.4 - 0.2 + 0.1 = 30.1$$

In effect, what actually happens in CORDIC is that z is initialized to 30 degree and then, in each step, the sign of the next rotation angle is selected to try to change the sign of z ; that is, $d_i = \text{sign}(z_i)$ is chosen, where the sign function is defined to be -1 or 1 depending on whether the argument is negative or non-negative.

TABLE 2: Approximate value of the function $\alpha_i = \arctan(2^{-i})$, in degree, for $0 \leq i \leq 9$

i	α_i
0	45
1	26.6
2	14
3	7.1
4	3.6
5	1.8
6	0.9
7	0.4
8	0.2
9	0.1

In CORDIC terminology the preceding selection rule for d_i , which makes z converge to zero, is known as rotation mode.

III. Modes of operation

The CORDIC rotator is normally operated in one of two modes. The first called rotation by Volder 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 i.e. in the first mode the rotator is aware of the angle and in the second mode the rotator will find the angle to which the

vector needs to rotate to get into the same alignment of the given vector.

3.1. Rotating Mode

In rotation mode 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:

$$x_{i+1} = x_i - d_i y_i 2^{-i} \quad (20)$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \quad (21)$$

$$z_{i+1} = z_i - d_i \alpha_i \quad (22)$$

where $\alpha_i = \tan^{-1} 2^{-i}$,

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

After m iteration in rotation mode, when z (m) is sufficiently close to zero. we have $\sum \alpha_i = z$, and the CORDIC equations become:

$$x_m = k(x \cos z - y \sin z) \quad (23)$$

$$y_m = k(y \cos z + x \sin z) \quad (23)$$

$$z_m = 0 \quad (24)$$

Choose $d_i \in \{-1, 1\}$ such that $z \rightarrow 0$

The constant K in the preceding equation is $k = 1.646760258121$. Thus, to compute $\cos z$ and $\sin z$, one can start with $x = 1/K = 0.607252935\dots$ and $y = 0$. Then, as z_m tends to 0 with CORDIC iterations in rotation mode, x_m and y_m converge to $\cos z$ and $\sin z$, respectively. Once $\sin z$ and $\cos z$ are known, $\tan z$ can be through necessary division.

TABLE 3: Choosing the signs of the rotation angles to force z to zero

i	$z_i - \alpha_i$	z_{i+1}
0	+30.0-45.0	-15
1	-15.0+26.6	11.6
2	+11.6-14.0	-2.4
3	-2.4+7.1	4.7
4	+4.7-3.6	1.1
5	+1.1-1.8	-0.7
6	-0.7+0.9	0.2
7	+0.2-0.4	-0.2
8	-0.2+0.2	0
9	+0.0-0.1	-0.1

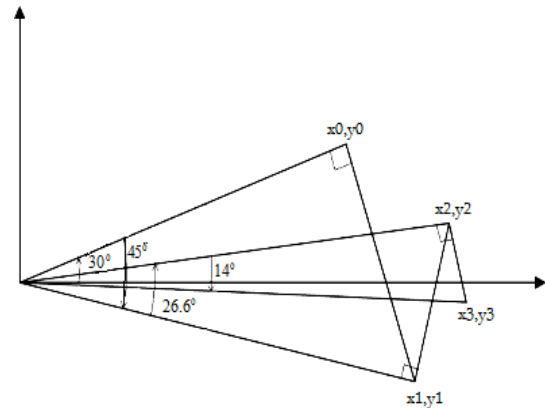


Figure 3: First three of 10 iterations leading from (x_0, y_0) to $(x_{10}, 0)$ in rotating by 30° , Rotating mode

For k bits of precision in the resulting trigonometric functions, k CORDIC iterations are needed. The reason is that for large i it can be approximated that $\tan^{-1} 2^{-i} \approx 2^{-i}$. Hence, for $i > k$, the change in the z will be less than U.L.P (Unit in the Last Place).

In the rotation mode, convergence of z to zero is possible because each angle in table 2.3 is more than half the previous angle or, equivalently, each angle is less than the sum of the entire angle following it. The domain of convergence is $-99.7 < z < 99.7$, where 99.7 is the sum of all the angles in table 2.3. Fortunately, this range includes angle from -90 to +90, or $[-\pi/2, \pi/2]$ in radians.

3.2. Vectoring Mode

In the Vectoring mode, the CORDIC rotator 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 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:

$$x_{i+1} = x_i - d_i y_i 2^{-i}$$

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

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

where $\alpha_i = \tan^{-1} 2^{-i}$,

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

After m iterations in vectoring mode $\tan \sum \alpha_i = -\frac{y}{x}$,

this means that:

$$x_m = k[x \cos(\sum \alpha_i) - y \sin(\sum \alpha_i)]$$

$$\begin{aligned} &= k(x - y \tan(\sum \alpha_i)) / [1 + \\ &\tan^2(\sum \alpha_i)]^{1/2} \\ &= k(x + \frac{y^2}{x}) / (1 + \frac{y^2}{x^2}) \end{aligned} \quad (25)$$

$$x_n = k(x^2 + y^2)^{1/2} \quad (26)$$

The CORDIC equations thus become:

$$x_m = k(x^2 + y^2)^{1/2} \quad y_m = 0 \quad (27)$$

$$z_m = z + \tan^{-1} y/x \quad (28)$$

Choose $d_i \in \{-1, 1\}$ such that $y \rightarrow 0$.

IV. Sine and cosine using CORDIC

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 result to:

$$x_n = A_n x_0 \cos z_0 \quad (29)$$

$$y_n = A_n x_0 \sin z_0 \quad (30)$$

The rotation algorithm has a gain A_n of approximately 1.647. The exact gain depends on the number of iterations and obeys the relation

$$A_n = \prod_n \sqrt{1 + 2^{-2i}} \quad (31)$$

By setting $x_0 = \frac{1}{A_n}$, the rotation produces unscaled sine and cosine of the angle argument z_0 . Very often, the sine and cosine values modulate a magnitude value. Using other techniques (e.g. a look up table) requires a pair of multipliers to obtain the modulation. The CORDIC technique perform the multiply as a part of the rotation operation and therefore eliminates the need 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 equal to the single multiplier with the same size.

V. GENERALIZED CORDIC

The basic CORDIC method can be generalized to provide the more powerful tool for function evaluation. Generalized CORDIC is defined as follows:

$$x_{i+1} = x_i - \mu d_i y_i 2^{-i} \quad (32)$$

$$y_{i+1} = y_i + d_i x_i 2^{-i} \quad (33)$$

$$z_{i+1} = z_i - d_i \alpha_i \quad (34)$$

Noting that the only difference with basic CORDIC is the introduction of the parameter μ in the equation for x and redefinition of α_i . The parameter μ can assume one of the three values: In case of

$\mu = 1$ for circular rotation (Basic CORDIC)

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

$\mu = 0$ for linear rotation $\alpha_i = 2^{-i}$

$\mu = -1$ for hyperbolic rotation $\alpha_i = \tanh^{-1} 2^{-i}$

VI. SCALING, QUANTIZATION AND ACCURACY ISSUES

Scaling is a necessary operation associated with the implementation of CORDIC algorithm. Scaling in CORDIC could be of two types: 1) constant factor scaling and 2) variable factor scaling. variable factor scaling the scale-factor changes with the rotation angle. It arises mainly because some of the iterations of conventional CORDIC are ignored (and that varies with the angle of rotation), as in the case of higher-radix CORDIC and most of the optimized CORDIC algorithms. The techniques for scaling compensation for each such algorithm have been studied extensively for minimizing the scaling overhead. In case of conventional CORDIC, after sufficiently large number of iterations, the scale-factor K converges to 1.6467605, which leads to constant factor scaling since the scale factor remains the same for all the angle of rotations. Constant factor scaling could be efficiently implemented in a dedicated scaling unit designed by canonical signed digit (CSD)-based technique and common sub-expression elimination (CSE) approach. When the sum of the output of more than one independent CORDIC operations are to be evaluated, one can perform only one scaling of the output sum in the case of constant factor scaling. In the following subsections, we briefly discuss some interesting developments on implementation of on-line scaling and realization of scaling-free CORDIC. Besides, we outline here the sources of error that may arise in a CORDIC design and their impact on implementation.

CORDIC technique is basically applied for rotation of a vector in circular, hyperbolic or linear coordinate systems, which in turn could also be used for generation of sinusoidal waveform, multiplication and division operations, and evaluation of angle of rotation, trigonometric functions, logarithms, Exponentials and square root, Table IV shows Some elementary functions and operations that can be directly implemented by CORDIC. The table also indicates whether the coordinate system is circular (CC), linear (LC), or hyperbolic (HC), and whether the CORDIC operates in rotation mode (RM) or vectoring mode (VM), the initialization of the CORDIC and the necessary pre- or post-processing step to perform the operation. The scale factors are, however, obviated in Table IV for simplicity of presentation. In this Section, we discuss how

CORDIC is used for some basic matrix problems like QR decomposition and singular-value decomposition.

Moreover, we make a brief presentation on the applications of CORDIC to signal and image processing, digital communication, robotics and 3-D graphics. The hybrid decomposition could be used for reducing the latency by ROM-based realization of coarse operation. This can also be used for reducing the hardware complexity of fine rotation phase since there is no need to find the direction of micro rotation. Several options are however possible for the implementation of these two stages. A form of hybrid CORDIC is suggested for very-high precision CORDIC rotation where the ROM size is reduced to nearly bits. The coarse rotations could be implemented as conventional CORDIC through shift-add operations of micro-rotations if the latency is tolerable.

VII. CORDIC ARCHITECTURES

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. However, the CORDIC iterations could not still be performed in parallel due to the first bottleneck. A partial parallelization has been realized in 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. 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. 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. 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 and pipelined CORDIC

architecture which we discuss briefly in the following subsections.

8.1. Bit Parallel Iterative CORDIC Architecture

The vector Rotation CORDIC structure is represented by the schematics in Figure. 3. Each branch consists of an adder-subtractor combination, a shift unit and a register for buffering the output. At the beginning of a calculation initial values are fed into the register by the multiplexer where the MSB of the stored value in the z-branch determines the operation mode for the adder-subtractor. Signals in the x and y branch pass the shift units and are then added to or subtracted from the unshifted signal in the opposite path. The z branch arithmetically combines the registers values with the values taken from a lookup table (LUT) whose address is changed accordingly to the number of iteration. For n iterations the output is mapped back to the registers before initial values are fed in again and the final sine value can be accessed at the output. A simple finite-state machine is needed to control the multiplexers, the shift distance and the addressing of the constant values.

When implemented in an FPGA the initial values for the vector coordinates as well as the constant values in the LUT can be hardwired in a word wide manner. The adder and the subtractor component are carried out separately and a multiplexer controlled by the sign of the angle accumulator distinguishes between addition and subtraction by routing the signals as required. The shift operations as implemented change the shift distance with the number of iterations but those require a high fan in and reduce the maximum speed for the application. In addition the output rate is also limited by the fact that operations are performed iteratively and therefore the maximum output rate equals $1/n$ times the clock rate.

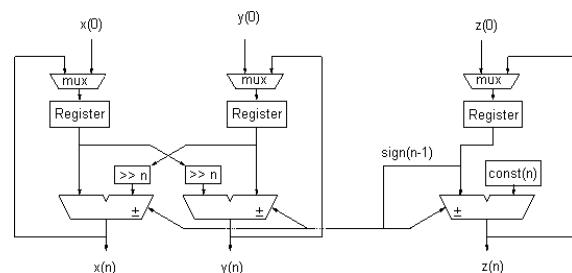


Figure 4: Iterative cordic

8.2. Parallel Unrolled CORDIC Architecture

Instead of buffering the output of one iteration and using the same resources again, one could simply cascade the iterative CORDIC, which means rebuilding the basic CORDIC structure for each iteration. Consequently, the output of one stage is the input of the next one, as shown in Figure. 4,

and in the face of separate stages two simplifications become possible.

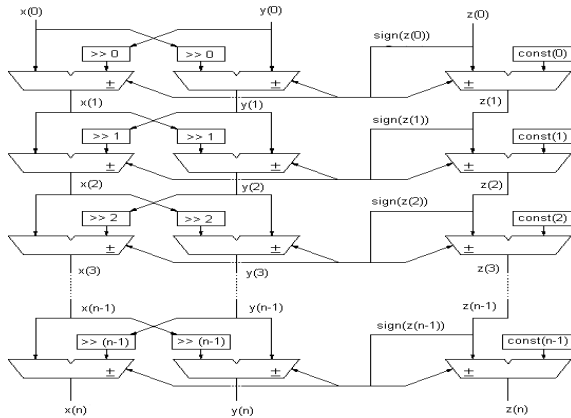


Figure 5: Unrolled CORDIC

First, the shift operations for each step can be performed by wiring the connections between stages appropriately. Second, there is no need for changing constant values and those can therefore be hardwired as well. The purely unrolled design only consists of combinatorial components and computes one sine value per clock cycle. Input values find their path through the architecture on their own and do not need to be controlled. As we know, the area in FPGAs can be measured in CLBs, each of which consist of two lookup tables as well as storage cells with additional control components. For the purely combinatorial design the CLB's function generators perform the add and shift operations and no storage cells are used. This means registers could be inserted easily without significantly increasing the area. Pipelining adds some latency, of course, but the application needs to output values at 48 kHz and the latency for 14 iterations equals 312.5 which are known to be imperceptible. However, inserting registers between stages would also reduce the maximum path delays and correspondingly a higher maximum speed can be achieved.

C. Bit Serial Iterative CORDIC Architecture

Both, the unrolled and the iterative bit-parallel designs, show disadvantages in terms of complexity and path delays going along with the large number of cross connections between single stages. To reduce this complexity one could change the design into a completely bit-serial iterative architecture. Bit-serial means only one bit is processed at a time and hence the cross connections become one bit-wide data paths. Clearly, the throughput becomes a function of \ln in spite of this the output rate can be almost as high as achieved with the unrolled design. The reason is the structural simplicity of a bit-serial design and the correspondingly high clock rate achievable. Figure. 5

shows the basic architecture of the bit serial CORDIC processor.

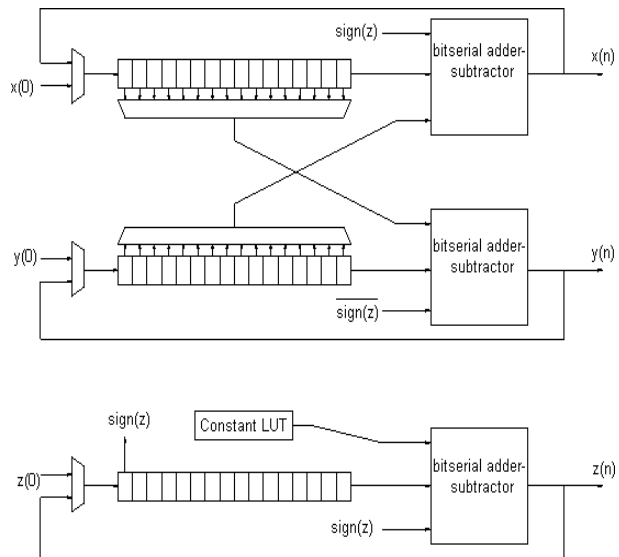


Figure 6: Bit serial CORDIC

Since the CORDIC iterations are identical, it is very much convenient to map them into pipelined architectures. The main emphasis in efficient pipelined implementation lies with the minimization of the critical path. The earliest pipelined architecture that we find was suggested in 1984. Pipelined CORDIC circuits have been used thereafter for high-throughput implementation of sinusoidal wave generation, fixed and adaptive filters, discrete orthogonal transforms and other signal processing applications.

VIII. Simulation results

Using Xilinx 12.1 tool sine and cosine signals were simulated and the results were plotted. Hardware usage and complexity were less yet giving a high throughput and accuracy.

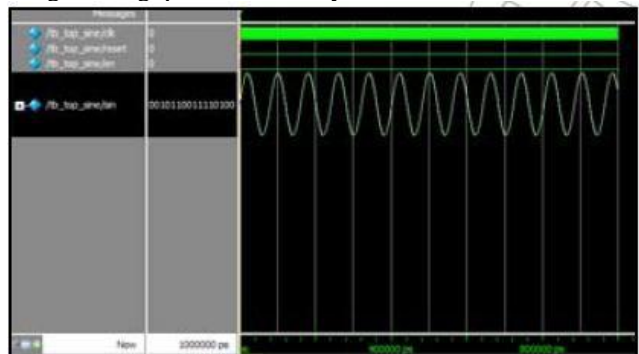


Figure 7 :simulation result

IX. Conclusion

CORDIC algorithm can be implemented by using simple hardware through repeated shift-add operations. This feature makes it attractive for a wide

variety of applications. Moreover, its applications in several diverse areas including signal processing, image processing, communication, robotics and graphics apart from general scientific and technical computations have been explored. In the last half century, several algorithms and architectures have been developed to speed up the CORDIC algorithm by reducing its iteration counts and through its pipelined implementation.

REFERENCES

- [1] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sep. 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] K. Maharatna, A. S. Dhar, and S. Banerjee, "A VLSI array architecture for realization of DFT, DHT, DCT and DST," *Signal Process.*, vol. 81, pp. 1813–1822, 2001.
- [4] C.-S. Wu, A.-Y. Wu, and C.-H. Lin, "A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol.50, no.9, pp.589–601, Sep.2003.
- [5] J. Villalba, T. Lang, and E. L. Zapata, "Parallel compensation of scale factor for the CORDIC algorithm," *J. VLSI Signal Process. Syst.*, vol.19, no. 3, pp. 227–241, Aug. 1998.
- [6] Y. H. Hu and S. Naganathan, "An angle recoding method for CORDIC algorithm implementation," *IEEE Trans. Comput.*, vol. 42, no. 1, pp.99–102, Jan. 1993.
- [7] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, "Modified virtually scaling-free adaptive CORDIC rotator algorithm and architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 11, pp. 1463–1474, Nov. 2005.
- [8] F. J. Jaime, M. A. Sanchez, J. Hormigo, J. Villalba, and E. L. Zapata, "Enhanced scaling-free CORDIC," *IEEE Trans. Circuits Syst. I, Reg.Papers*, vol. 57, no. 7, pp. 1654–1662, Jul. 2010.
- [9] 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.
- [10] S. Wang and E. E. Swartzlander, "Merged CORDIC algorithm," in IEEE International Symposium on Circuits Systems (ISCAS'95), 1995, volume 3, pp.1988–1991.