International Journal of Computer Applications (0975 – 8887)
Volume 56– No.3, October 2012

Implementation of High Speed Fixed Point CORDIC Techniques

Sukhpreet Kaur
Assistant Professor
ECED, Chitkara University, Baddi, HP

Kulbir Singh
Associate Professor
ECED, Thapar University, Patiala, Punjab

ABSTRACT
Implementation of Original CORDIC, Control CORDIC and Angle Recoding CORDIC of 16-bit, 24-bit and 32-bit fixed point number have been done in this paper. VHDL is used to modelled these architectures. Original CORDIC, Control CORDIC and Angle Recoding CORDIC are synthesized and targeted for Xilinx Virtex 5 FPGA and the results calculated for 16-bit and 24-bit has shown satisfactory improvement in speed.

General Terms
Original CORDIC, Control CORDIC and Angle Recoding CORDIC.

Keywords
CORDIC algorithm, VHDL, Fixed Point representation.

1. INTRODUCTION
The CORDIC (Coordinate Rotation Digital Computer) algorithm was introduced by Jack E. Volder [1] in 1959 and further generalized by Walther [2] in 1971. It is a class of shift and add algorithm for rotating vector in a plane, which is usually used for the calculation of trigonometric functions, multiplication, division and conversion between binary and mixed radix number systems of DSP applications, such as Fourier Transform. A fast and energy-efficient CORDIC for the calculation of elementary function is always needed in electronics systems i.e. DSP processors, image processing and arithmetic units in microprocessors. On VLSI implementation level, the area also becomes quite important as more area means more system cost. The three parameters i.e. power, speed and area are always traded off. For DSP processors area and speed are the main ones. CORDIC algorithm is very attractive for hardware implementation because it uses only elementary shift-and-add operations to perform the vector rotation. It only needs the use of 2 shifter and 3 adder modules, so its power dissipation is very less and it is also very compact. Therefore, it is frequently used in an array of processing elements on VLSI chips.

However, the major disadvantage of the CORDIC algorithm is its slow computational speed. For iterative CORDIC structure, the speed and performance of CORDIC operation is limited by the large iteration number, N which is generally equal to the internal word length, W. One solution to such a problem is to reduce the iteration count directly. Other solution is to use bit parallel CORDIC algorithm or to use bit parallel unrolled CORDIC. One other method is to use Control CORDIC algorithm instead of Original CORDIC, in which, delay is improved by eliminating the divergent rotations [3]. For applications that require forward rotation only, the Angle Recoding CORDIC can be used [4]. In Angle Recoding CORDIC, iteration count is reduced by skipping over some rotation angles. The maximum number of iterations required by this method is N/2, with an average value of approximately N/3 iterations. The associated CORDIC method that makes use of the Angle Recoding method is termed as Adaptive CORDIC [5]. The CORDIC algorithm can be applied in two different modes (e.g., rotation mode and vectoring mode) and three types (e.g., linear, circular and hyperbolic mode) [6].

In this paper, Original CORDIC, Control CORDIC and Angle Recoding CORDIC of 16-bit, 24-bit and 32-bit fixed point number have been synthesized using Xilinx 13.1. A comparison is made for 16-bit and 24-bit with the reference paper [5]. To attain a generic design, VHDL hardware description language was used to synthesize the entire CORDIC units as it presents a tremendous productivity improvement for circuit designers and descriptions of large circuits can be written in a relatively compact and concise form. Section 2 reviews the Original CORDIC algorithm. Section 3 presents a review of Control CORDIC. Section 4 presents a review of Angle Recoding CORDIC which seeks to reduce the high latency by reducing the number of iterations. Section 5 describes 16-bit, 24-bit and 32-bit fixed point number representation. Section 6 presents the results for the Original CORDIC, the Control CORDIC and the Angle Recoding CORDIC for 16 bit, 24 bit and 32 bit. Section 7 concludes the paper.

2. CORDIC ALGORITHM
The Volder's algorithm was derived from the general equations of vector rotation. CORDIC is highly useful algorithm for the calculation of elementary function. The main disadvantage of the CORDIC algorithm is the long latency. A lot of work has been done by researcher to the latency of the CORDIC algorithm. One way to improve the latency is to reduce the number of iteration count. Ercegovac and Lang [7] developed the Online CORDIC. Online CORDIC is suitable for applications where input bits became available serially. The Online CORDIC method replaces variable shifters by more area-efficient delays. Online CORDIC method could also compensate for the value of K online. Duprat and Miller [8] used fast adders that are based upon the use of redundant arithmetic to reduce the cycle time of a CORDIC iteration.

The CORDIC algorithm is used to evaluate real time calculation of the exponential and logarithmic functions using the iterative rotation of the input vector. The rotation of a given vector is realized by means of a sequence of rotations with fixed angles which results in overall rotation through a given angle or result in a final angular argument of zero. CORDIC hardware is very simple, consisting only of 2 shifters and 3 adders, it is able to evaluate a wide variety of elementary functions, and consequently it finds use in many different engineering applications. The underlying method of computing the rotation of a vector in a Cartesian coordinate system and evaluating the length and angle of a vector was developed by Volder [2].

The resulting vector $(x_n \quad y_n)$, of the rotation of a vector $(x_0 \quad y_0)$, by an angle θ in Cartesian coordinates can be computed by the following matrix operation:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_o \end{bmatrix} \qquad (1)$$

Using the identity

$$\cos\theta = \frac{1}{\sqrt{1+\tan\theta^2}} \qquad (2)$$

and factoring out $\cos\theta$ equation (1) can be modified to

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \frac{1}{\sqrt{1+\tan\theta^2}} \begin{bmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_o \end{bmatrix} \qquad (3)$$

In the CORDIC method, the rotation by an angle θ is implemented as several micro rotations by a given step angle α_i. Any angle θ can be represented to a certain accuracy by a set of n step angle α_i. Specifying a direction of rotation or sign σ_i the sum of the step angles α_i approximates a given angle θ as follows

$$\theta = \sum_{i=0}^{n-1} \sigma_i \alpha_i \quad , \sigma_i \epsilon \{-1,1\} \qquad (4)$$

The sign of the difference between the angle θ and the partial sum of step angles $\theta = \sum_{i=0}^{i-1} \sigma_i \alpha_i$ controls the sign of the step angle σ_i. To simplify the computation of the matrix product (3), the step angles α_i are chosen such that $\tan\alpha_i$ represents a series of powers of 2:

$$\tan\alpha_i = 2^{-1} \ , i = 0,1 \ldots\ldots n-1$$

An auxiliary variable z_i is introduced that contains the accumulated partial sum of step angles and can be used to control the sign of the step angles.
The CORDIC method can be employed in two different modes, known as the “rotation” mode and the “vectoring” mode. In the rotation mode, the co-ordinate components of a vector and an angle of rotation are given and the co-ordinate components of the original vector, after rotation through a given angle, are computed. In the vectoring mode, the co-ordinate components of a vector are given and the magnitude and angular argument of the original vector are computed.

In case of rotation mode:
Inputs: x_0 , y_0 , anglez_o
Iteration equations:

$$x_{i+1} = x_i - y_i \sigma_i 2^{-i} \qquad (5)$$
$$y_{i+1} = y_i + x_i 2^{-i} \qquad (6)$$
$$z_{i+1} = z_i - \sigma_i \tan^{-1} 2^{-i} \qquad (7)$$

where $i = 0,1,2 \ldots\ldots n-1$,

$$\sigma_i = \begin{cases} -1 \ if \ z_i < 0 \\ +1 \ if \ z_i \geq 0 \end{cases} \qquad (8)$$

Outputs:

$$x_n = K_n(x_0 \cos z_0 - y_0 \sin z_0)$$
$$y_n = K_n(y_0 \cos z_0 + x_0 \sin z_0)$$
$$z_n = 0$$

where $K_n = \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}}$ (9)

is a scale factor, that represents the increase in magnitude of the vector during the rotation process since the rotation is not a pure rotation but a rotation-extension. When the number of iterations/micro-rotations is fixed the scale factor is a constant approaching the value of 1.647 as the number of iterations goes to infinity.

2.1 Sine and Cosine Computation using the CORDIC Method

The rotation mode of the CORDIC algorithm could be used to compute sine and cosine of an angle θ. The computation of $\sin\theta$ and $\cos\theta$ is based on the rotation of an initial vector of unit length, that is aligned with the abscissa $(x_0 = 1, y_0 = 0)$.

Input values for n iterations:

$$x_0 = 1$$
$$y_0 = 0$$
$$z_0 = \theta$$

Outputs after n micro-rotations:

$$x_n = K_n(x_0 \cos\theta - y_0 \sin\theta)$$
$$= K_n \sin\theta \qquad (9)$$
$$y_n = K_n(y_0 \cos\theta + x_0 \sin\theta)$$
$$= K_n \sin\theta \qquad (10)$$
$$z_n = 0 \qquad (11)$$

The magnitude of the initial vector increases by a factor K_n during the micro-rotations that constitute the rotation mode and an operation of division is required at the end of the rotation process in order to obtain the value of $\sin\theta$ and $\cos\theta$. One simple way to avoid the operation of division is to compensate the scale factor by setting the initial value $x_n = 1/K_n$, since the scale factor is a constant for a given number of iterations. Fig 1 shows the block diagram of CORDIC algorithm.

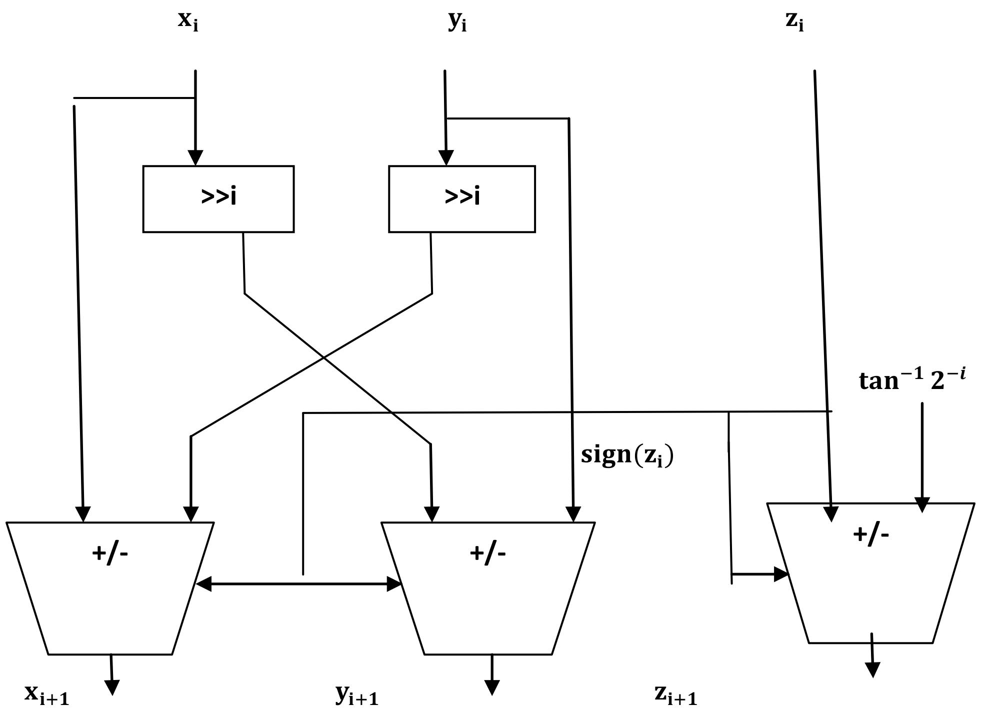


Figure 1: Block Diagram of CORDIC algorithm

2.2 Original CORDIC- Rotation

In the CORDIC algorithm, for the calculation of sine and cosine value of any angle, there is a set predefined angle constants which are used to calculate of any desired angle. These angle constants are added or subtracted depending upon equation (8) to calculate the value of desired angle. These angle constant can be stored in a table. For example: To calculate the sine or cosine value of 30 degrees, all the angle constants that are stored in a table are either added or subtracted from the previous calculated value to approach to an angle of 30 degree. Now if rotation is to be accomplished in 16 numbers of iterations, then 16 predefined angle constants are desired that are arranged in a sequence as shown below:

$$Q = \{45°, 26.565°, 14.036°, 7.125°, 3.576°, 1.7876°, 0.8938°, 0.4469°, 0.2238°, 0.1119, 0.05595°, 0.02798°, 0.01399°, 0.00699°, 0.00349°, 0.00175°\}$$

In the Original CORDIC method, all the above 16 angles are added or subtracted in sequence to approach to 30 degrees, as shown in equation (12). $30° \approx (45° - 26.564° + 14.036° - 7.125° + 3.576° +1.7876° - 0.8938° + 0.4469° - 0.2238° - 0.1119° + 0.05595° + 0.02798° - 0.01399° +0.00699° - 0.00349° - 0.00175°)$ (12)

In the equation (12), as a whole algorithm converges but there are certain rotation that leads to divergent pseudorotation. For example $i = 2 \ to \ i = 3$ is a divergent pseudo-rotation.

3. CONTROL CORDIC ALGORITHM

The Control CORDIC method [4] used to a technique to reduce the number of iterations for calculating the desired rotation angle without affecting the desired precision but Control CORDIC make use of ROM space for storing different scaling factors. The technique was based upon the observation that in the Original CORDIC algorithm, the iteration variable z_i does not always converge monotonically to 0 – some of the iterations may actually result in divergent micro-rotations, which do nothing to improve the convergence towards the target vector. As shown above in section 2.2, these divergent micro-rotations (seen at i = 3) in the Original CORDIC algorithm.

In the Original CORDIC, the angle trajectory looks very similar to the under-damped response of a second order control system, with overshoot occurring, that is already seen in section 2.2. The Control CORDIC method modifies the angle trajectory so that it now resembles a critically damped system, with no overshoot, resulting in faster convergence. Now, divergent rotations can only occur when there is an overshoot, so by eliminating completely eliminating the overshoot, divergent micro-rotations are eliminated. But some overshoots are also leads to convergent micro-rotation, that are also eliminated.

For positive angles the rotation direction (σ_i) is restricted to $\{-1,0\}$ as given below:

$$\sigma_i = \left| \begin{array}{l} +1 \ when \ z \geq \alpha_i \\ 0 \ otherwise \end{array} \right.$$

Similarly for negative angles the rotation direction is restricted to $\{-1,0\}$. This simple angle selection function thus prevents any overshoot of the target position by the moving vector.

The advantage of this method is that its angle selection function is quite simple, and is easy to implement with only a minimal effect on the cycle time, thus allowing its use in dynamic situations where the angle of rotation can take any value. However in return for the reduction in iteration count, this method requires the use of a ROM to store the different scaling factors.

3.1 Control CORDIC- Rotation

As seen in the section 3.2, for calculating the angle 30 degree, Original CORDIC take 16 number of iterations. Now in case of Original CORDIC, it again uses the predefined angle constant as shown below, but out of these angle constants, the angle constant which leads to the divergent micro-rotation are eliminated. Predefined angle constants are shown below:

$$Q = \{45°, 26.565°, 14.036°, 7.125°, 3.576°, 1.7876°, 0.8938°, 0.4469°, 0.2238°, 0.1119, 0.05595°, 0.02798°, 0.01399°, 0.00699°, 0.00349°, 0.00175°\}$$

In the Control CORDIC method, the rotation through 30 degrees is carried out by the following sequence of angular steps or pseudo-rotations, that add up to approximately 30 degrees, as shown in (13)

$$30° \approx (0 + 26.564° + 0 + 0 + 0 + 1.7876° + 0.8938° + 0.4469° + 0.2238° + 0 + 0.05595° + 0 + 0.01399° + 0.00699° + 0.00349° + 0.00175°) \quad (13)$$

As we seen in (13), in case of Control CORDIC, divergent rotations are completely eliminated by eliminating overshoot.

4. ANGLE RECODING CORDIC

The Angle Recoding (AR) technique is suitable for applications that use CORDIC algorithm in only forward rotation mode (also known as vector rotation mode) [4]. In Angle Recoding CORDIC, there is large number of reduction in the iteration count as compared to Original CORDIC and Control CORDIC as well angle precision is not affected by the reduction in iteration count. In this case, the desired rotation angle is calculated as a linear combination of very few rotation angles. Each of these elementary rotation angles takes one CORDIC iteration to compute. The fewer the number of elementary rotation angles, the fewer the number of iterations are required. Figure 2 shows the block diagram of Angle Recoding CORDIC.

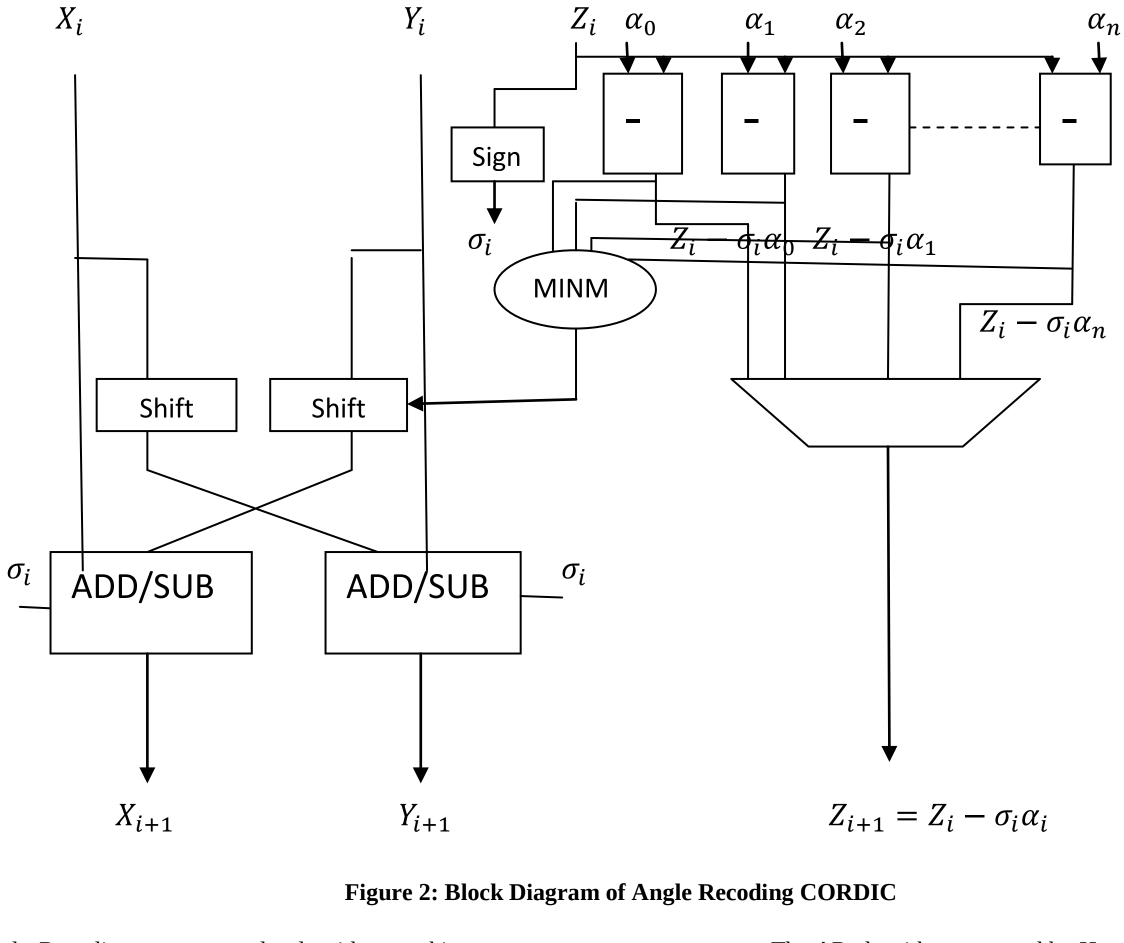


Figure 2: Block Diagram of Angle Recoding CORDIC

Angle Recoding uses a greedy algorithm to skip over some rotation angles, and can reduce the number of iterations required. The maximum number of iterations required by this method is N/2, with an average value of approximately N/3 iterations. It is studied that this algorithm is able to reduce the total number of required elementary rotation angles by at least 50% without affecting the computational accuracy.

The AR algorithm proposed by Hu and Naganathan [4] is software based algorithm due to which it used only for very application. But this Angle Recoding Algorithm must be modified, so it can be implemented in hardware also. By implementing AR algorithm in hardware, it can be used to find any desired rotation angle. This is done by replacing the serial testing of the angle constants in the original algorithm by a parallel test to be carried out in hardware in case of Adaptive CORDIC [5]. It was done by completely eliminate the angle selection step from every iteration altogether. Instead of using the current residual angle to determine the angle constant for that iteration, it would be much more efficient if all the angle constants could be identified in a single step, using only the initial rotation angle as input.

In case of Angle Recoding, instead of choosing all the angle constants stored in ROM, we make use of a technique in which residual angle z_i is compared with all the angle constants α_i stored in ROM and the angle constant with minimum difference $(z_i - \sigma_i \alpha_i)$ is chosen. The corresponding angle constant with minimum difference is chosen and the index i related to that angle constant is used for the shift operations. After shifting operation, the process of calculating the new X and Y coordinates of the vector is done. The extra logic needed for the calculation of angle constant with minimum difference will increase the cycle time. Due which the reduction in iteration count will not affect the latency of CORDIC algorithm and even we get more latency for large value of N.

4.1 Angle Recoding CORDIC- Rotation

Consider the rotation of a vector from the x-axis through an angle of, e.g. 30 degrees. Assuming that the rotation is to be accomplished for 16 number of bits $(N = 0,1,2,3 \ldots\ldots\ldots\ \ldots\ldots\ldots 14,15)$, the set Q of predetermined angle constants that are used is as follows:

$$Q = \{45°, 26.565°, 14.036°, 7.125°, 3.576°, 1.7876°, 0.8938°, 0.4469°, 0.2238°, 0.1119, 0.05595°, 0.02798°, 0.01399°, 0.00699°, 0.00349°, 0.00175°\}$$

In AR CORDIC, the rotation through 30 degrees is carried out by the following sequence of angular steps or pseudo-rotations, that add up to approximately 30 degrees, as shown on below:

$$30 \approx (26.565 + 3.576 - 0.1119 - 0.02798 - 0.00175) \qquad (14)$$

So we see in (14) that in case of AR CORDIC, we require only 5 iterations instead of 16 iterations for 16 bit data. So, number of iteration count is reduced but cycle time may be either remain same or may be increase for large value of N.

5. Fixed Point Number Representation

In real life, we deal with real numbers -- numbers with fractional part. Most modern computer have native (hardware) support for floating point numbers. However, the use of floating point is not necessarily the only way to represent fractional numbers. This section describes the fixed point representation of real numbers. The use of fixed point data type is used widely in digital signal processing (DSP) and game applications, where performance is sometimes more important than precision. Fixed point arithmetic is much faster than floating point arithmetic. In case of fixed point representation, every word has the same number of digits and the binary point is always fixed at the same position.

5.1 Signed Two's Complement Fixed Point

In case of N-bit signed two's complement fixed point number, the most significant bit starts from the left. The most significant bit represents the sign bit i.e. if it is 0, number is positive and if it is 1, then number is negative. The remaining N-1 bits are divided into m bit integer part and n bit fractional part. The binary point is always fixed at the same position.

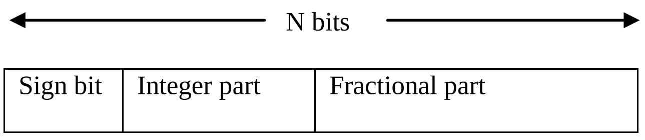


Figure 3: Signed Fixed point representation

Total bits, N = sign bit + Integer bits + Fractional bit
= 1 + m + n

In this paper, 3 types of signed fixed point number representation: 16-bit, 24-bit and 32-bit has been used. Number of bits used for integer part and fractional part is shown in the Table 1.

Table 1 Signed fixed point representation

Total number of bits	16-bit	24-bit	32-bit
Sign bit	1 bit	1 bit	1 bit
Number of integer bits	8 bits	8 bits	8 bits
Number of fractional bits	7 bits	15 bits	23 bits

6. RESULTS

VLSI implementation of Original CORDIC, Control CORDIC and Angle Recoding CORDIC for 16-bit, 24-bit and 32-bit have been successfully done on Xilinx Virtex 5.

In the figure 4 (a) – (c) the simulation result of the 16 bit, 24 bit and 32 bit signed fixed point Original CORDIC or sine and cosine computation on Modelsim 6.3f are shown.

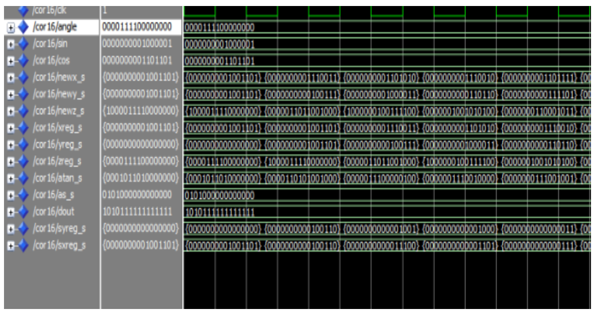


Figure 4 (a)

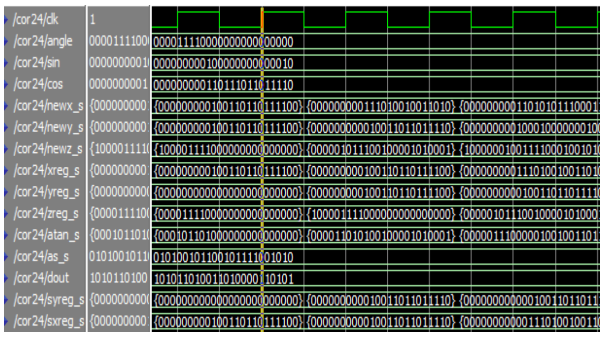
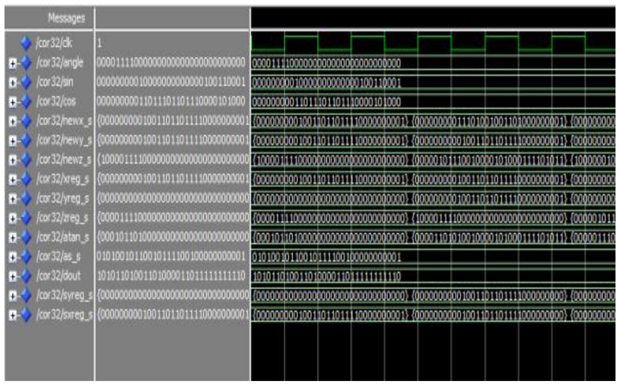


Figure 4 (b)



```
Messages
/cor32/clk       1
/cor32/angle     00001111000000000000000000000000
/cor32/sin       00000000010000000000000100110001
/cor32/cos       00000000011011101101110000101000
/cor32/newx_s    {000000000100110110111100000000001
/cor32/newy_s    {000000000100110110111100000000001
/cor32/newz_s    {10000111100000000000000000000000
/cor32/xreg_s    {000000000100110110111100000000001
/cor32/yreg_s    {00000000000000000000000000000000
/cor32/zreg_s    {00001111000000000000000000000000
/cor32/atan_s    {00010110100000000000000000000000
/cor32/as_s      01010010110010111100100000000001
/cor32/dout      10101101001101000011011111111110
/cor32/syreg_s   {00000000000000000000000000000000
/cor32/sxreg_s   {000000000100110110111100000000001
```

Figure 4 (c)

Figure 4 (a)-(c): Simulation results for generation of sine and cosine using Original CORDIC for different numbers of bits (a) 16 bit (b) 24 bit & (c) 32 bit

Synthesis Results on FPGA of Original CORDIC, Control CORDIC and Angle Recoding CORDIC

Delay of 16-bit Original CORDIC, Control CORDIC and Angle Recoding CORDIC is 1.90ns. Delay for 24-bit is 2.22ns and for 32-bit is 2.41ns. We see that the delay is increased as we increase the number of bits. Number of slice registers for various CORDIC and for different number of bits is also calculated. All these results are shown in Table 2.

Table 2 Synthesis Report of Original CORDIC, Control CORDIC and Angle Recoding CORDIC on Virtex 5

	16-bit	**24-bit**	**32-bit**
Minimum period	1.88ns	2.22ns	2.41ns
Maximum Frequency	525.91 MHz	448.69 MHz	414.81 MHz
No. of slice registers in Original CORDIC	4602/51840 (8%)	8148/51840 (15%)	14212/51840 (27%)
No. of slice registers in Control CORDIC	5942/51840 (11%)	12346/51840 (23%)	19608/51840 (37%)
No. of slice registers in AR CORDIC	7652/51840 (14%)	19004/51840 (37%)	25012/51840 (49%)

Table 3 shows the comparison of delay for 16-bit and 24-bit Original CORDIC, Control CORDIC, Angle Recoding CORDIC with the reference paper [5]. Table 3 also shows the percentage improvement in delay. Table 3 shows that for 16-bit CORDIC algorithm the improvement in speed is 2.5% and for 24-bit improvement in speed is 12.5%.

Table 3 Comparison of delay and percentage improvement for 16-bit and 24-bit Original CORDIC, Control CORDIC, Angle Recoding CORDIC

		16-bit	**24-bit**
DELAY	Original Paper	1.88ns	2.22ns
	Reference Paper [5]	1.93ns	2.55ns
Percentage Improvement in Speed		2.5%	12.95%

7. CONCLUSION

The implementation of Original CORDIC, Control CORDIC and Angle Recoding based on 16 bit, 24 bit and 32 bit is developed. Delay of Original CORDIC, Control CORDIC and Angle Recoding CORDIC remain same for same number of bits. The area required has been measured in terms of slice register. These are basically synthesized to get high speed in term of frequency. Delay comes out to be 1.88ns, 2.22ns and 2.41ns in case of 16-bit, 24-bit and 32-bit fixed point CORDIC respectively. Frequency is 525.91MHz, 448.69MHz and 414.81MHz in case of 16-bit, 24-bit and 32-bit fixed point CORDIC respectively. Delay remains same for Original CORDIC, Control CORDIC and Angle Recoding CORDIC for same numbers of bits. But the iteration count is nearly half in case of Angle Recoding CORDIC as compared to Original and Control CORDIC. It has found that 3 adders/subtractors,

3 registers and 2 shifters are required and no multiplier is used in CORDIC algorithm. Also Control CORDIC shows no overshoot as compared to Original CORDIC. Number of iteration is reduced to half in case of Angle Recoding CORDIC, but delay remains same due to increase in cycle time needed for angle selection process. Percentage improvement in speed is 2.5% for 16-bit fixed point CORDIC and 12.95%.

8. REFERENCES

[1] J. Volder, "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, vol. EC-8, no. 3, pp. 330-334, 1959.

[2] J. Walther, "A Unified Algorithm for Elementary Functions," Proceedings of Spring Joint Computer Conference, vol. 38, pp. 379-385, 1971.

[3] S. Wang and E. Swartzlander, "Critically Damped CORDIC Algorithm," Proceedings of the 37th Midwest Symposium on Circuits and Systems, vol. 1, pp. 236-239, 1994.

[4] Y. Hu and S. Naganathan, "An Angle Recoding Method for CORDIC Algorithm Implementation," IEEE Transactions on Computers, vol. 42, no. 1, pp. 99-102, 1993.

[5] T. Rodrigues and E. Swartzlander, "Adaptive CORDIC: Using Parallel Angle Recoding to Accelerate Rotations," IEEE Transactions on Computers, vol. 59,

no. 4, pp. 522-531, 2010.

[6] R. Andraka, “A survey of CORDIC algorithm for FPGA based computers,” International Symposium on Field Programmable Gate Arrays, no. 2, pp. 191-200, 1998.

[7] M. Ercegovac and L. Tomas, “Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD,” IEEE Transactions on Computers, vol. 39, pp. 725-740, 1990.

[8] J. Duprat and J. Muller, “The CORDIC Algorithm: New Results for Fast VLSI Implementation,” IEEE Transactions on Computers, vol. 42, no. 2, pp. 168-178, 1993.