Comparison and Optimization of Pipelined and Parallel Implemenations of CORDIC Algorithm

Saqib Khan, Gayathri Koya, and Kaustubh Shinde

Abstract

This project aims at implementing a sine and cosine generator using the CORDIC algorithm. Different implementations of CORDIC algorithm namely Serial, Parallel and Pipelined are explored, the Parallel and Pipelined architecture are then implemented using Verilog. The design accepts an angle between [-pi pi] and outputs a 16-bit sine and cosine value, 2s complement representation of numbers is used. Timing, Area and Power optimized implementations of the code are synthesized using Design vision. Finally, the two architectures are compared in terms Area, Power and Speed.

Index Terms

VLSI, CORDIC, RTL, Verilog, Pipelined, Parallel.

I. INTRODUCTION

THis document provides a comparison between various Co-ordinate Rotation Digital Computer (CORDIC) architectures that can be used for the implementation of trigonometric functions such as sine and cosine using signed $2^{'}$s complement representation.

In this document we provide an overview of the CORDIC algorithm, focusing on the real-world applications for this design and its simplicity compared to other more complex algorithms. We then provide a low-level description of each of the three major CORDIC architectures, namely Serial, Parallel and Pipeline. We then provide a Verilog HDL implementation of CORDIC algorithm based on parallel and pipeline design.

By taking use of the digital design tools like Synopsys and Cadence, the implemented algorithms were verified, and the results of power, area and timing analysis were also obtained and presented in this paper. At last we provide our view on the best CORDIC architecture and further optimizations that could help improve the overall performance.

A. CORDIC Algorithm

CORDIC is a multipurpose algorithm that can perform a wide range of operations such as linear, trigonometric, and logarithmic functions. The CORDIC algorithm provides an iterative method of vector rotation using just shift and add logic.

The CORDIC algorithm is based on two modes of operation, namely ROTATIONAL and VECTOR. In rotation mode, the initial co-ordinates (x_o, y_o) and the angle of rotation (θ) is given, and the CORDIC algorithm computes the new coordinates (x_n, y_n) by rotating (x_o, y_o) by an angle of (θ). In vector mode, the coordinates (x_o, y_o) are provided and is rotated until y_o converges to zero.

The main idea behind CORDIC algorithm is to take a vector (x_o, y_o) and apply small successive rotations called micro-rotations until the desired angle is reached. The resulting vector (x_n, y_n) can be defined as:

$$x_n = x_o cos(\theta) - y_o sin(\theta)$$

$$y_n = x_o cos(\theta) + x_o sin(\theta)$$

With each iteration "i", the vector performs a micro-rotation by (θ_i), therefore the resulting vector can be defined as:

$$x_{i+1} = x_i cos(\theta_{i+1}) - y_i sin(\theta_{i+1})$$

$$y_{i+1} = y_i cos(\theta_{i+1}) + x_i sin(\theta_{i+1})$$

Factoring out $cos(\theta_{i+1})$ results in:

$$x_{i+1} = cos(\theta_{i+1})(x_i - y_i tan((\theta_{i+1}))$$

$$y_{i+1} = cos(\theta_{i+1})(y_i - x_i tan((\theta_{i+1}))$$

We can restrict $tan(\theta_{i+1})$ to 2^{-i}, thereby allowing the use of arithmetic shift right to easily carry out multiplication. Since $cos(\theta_{i+1}) = cos(tan^{-1}(2^{-i}))$

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

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

where $K_i = \pi * cos(tan^{-1}(2^{-i})) = \pi/\sqrt{1+2^{-2i}}$ is also known as the gain factor for each micro-rotation and $d_i = 1$. This allows us to efficiently calculate the sine and cosine values using the rotation mode of CORDIC by starting with an initial vector (x_o, y_o) and rotation angle θ, and ending with a new vector (x_n, y_n).

II. CORDIC ARCHITECTURES

CORDIC algorithm can be implemented using different architectures for mapping into hardware. These can be classified into three different types serial, parallel and pipeline. Each architecture has its own pros and cons which make it suitable only for certain applications.

A. Serial (Sequential/Iterative) Architecture

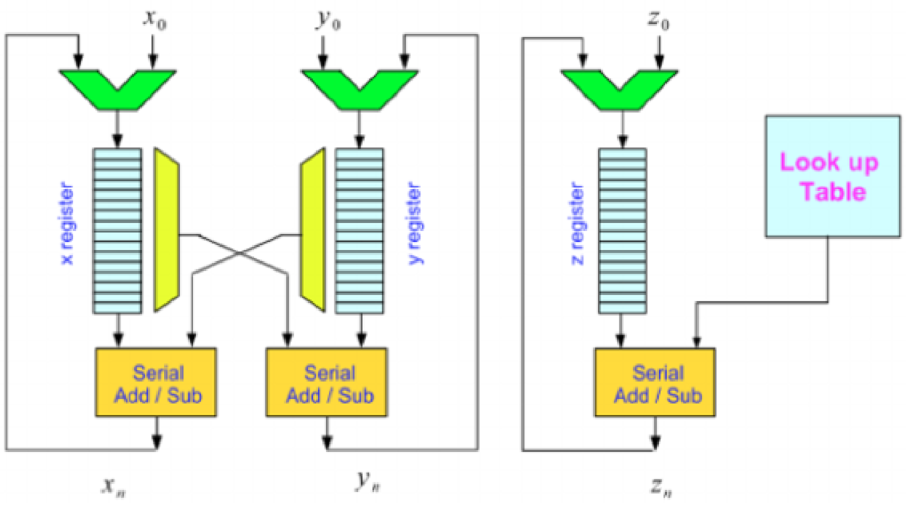

Fig. 1. Serial Architecture

Serial implementation minimizes the hardware requirements by utilizing a single CORDIC block repeatedly. Figure 1 provides a block diagram for CORDIC's serial implementation.

A serial CORDIC unit consists of a Logic block and a Control block. The Logic block contains adder, shifter, look-up-table (LUT), and a sign-control logic for generating decision bits. A Control block acts as a state machine keeping track of all the iterations performed and to select the degree of shift for each iteration.

Once the data is loaded into the CORDIC block, it is passed on to the logic block. The control unit is notified which selects the degree of rotation from LUT and provides it to the logic block. After the shift is performed, the output register is fed back into the logic block and the control unit is notified. The control unit moves on to the next state and selects the next degree of rotation from LUT. When all iterations get completed, the control unit provides no further rotation angle and the output registers are held stable.

A serial CORDIC algorithm is hardware efficient algorithm that occupies less area and components, thereby making it attractive for portable and low-cost devices such as calculators, pagers etc. However, each iteration takes one clock cycle and so N bit width takes N clock cycles to produce new output. This slows down computation and hence, is not suitable for high speed implementations.

B. Parallel (Cascaded) Architecture

Parallel implementation uses multiple instances of serial structures. This allows to perform shift and add operations in parallel, which significantly cuts down the processing time. Figure 2 shows the block diagram for a parallel architecture for CORDIC.

The parallel architecture makes two significant simplifications to the design. First, the shift registers perform a fixed shift, which means that they can be hardwired, thereby reducing the hardware requirement and processing time for computation of the rotation angles.

Second, the LUT values can be hardcoded to each adders, which helps prevent the use of storage registers making it completely combinational and significantly cuts down the processing time.

Since the parallel implementation performs shift and add operations in parallel, a vector with N bit output has a latency of one clock cycle. Each block in the parallel structure has its own latency and plays a role in determining the clock frequency. This would lead to parallel CORDIC having a lesser operating frequency than its serial counterpart. However, when dealing with a sequence of inputs, the parallel structure would prove more efficient since it has a much greater throughput compared to serial structure.

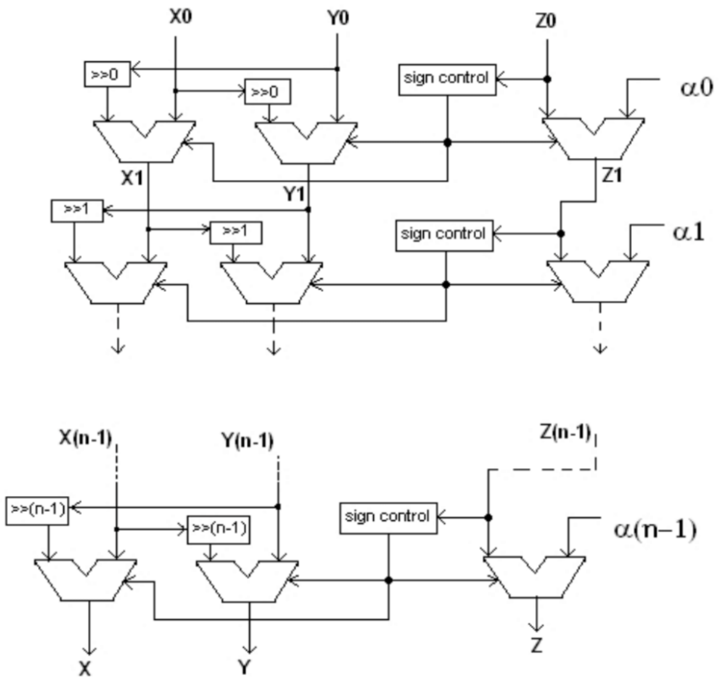

Fig. 2. Parallel Architecture

C. Pipelined Architecture

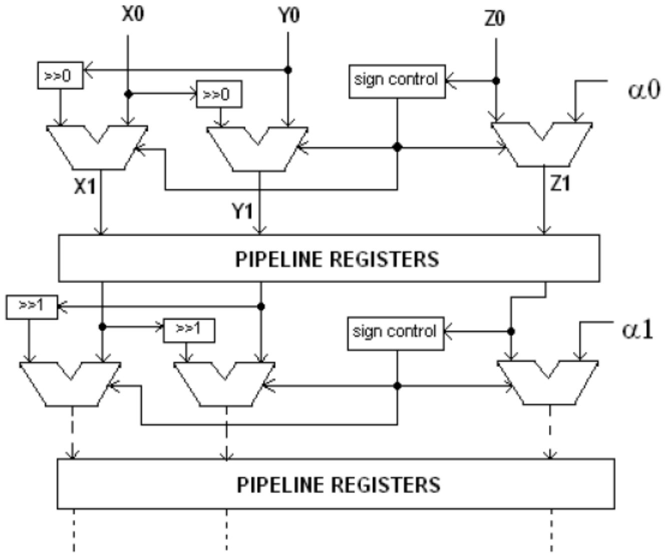

Fig. 3. Pipelined Architecture

A parallel implementation is pipelined by inserting a register between each iteration block. These registers hold the computed value for the next iteration while the previous iteration block is made available to take a new value for computation. Therefore, it takes a N bit CORDIC using pipeline structure, N clock cycle to produce the first result, after which an output is generated every clock cycle. This allows CORDIC to operate at a much higher frequency compared to serial and parallel architecture.

Due to the high operating frequency of pipeline structure, it is widely used in very-high performance devices, such as satellite communication and other communication systems.

The major disadvantage of pipeline structure is the increase in area introduced by the pipeline registers.

III. Implementation of Optimization Procedure

Tools in the Design Vision software were used to optimize the parallel and pipelined algorithms as a part of this project. Once the optimization was done, optimized results were compared and analyzed.

The three main parameters taken into consideration for optimization were:

1. Area
2. Speed
3. Power Consumption

A. Optimization Setup

The following steps were taken in order to setup for the optimization procedure:

After Design Vision was loaded, the verilog codes for parallel and pipelined algorithm were read into the software to ensure synthesizibility. Once it was made sure that the codes were synthesizable, environmental constraints were set to the design. The library used was the standard gsm45nm library and the wire load was set as NONE. Once this was done, a clock was specified. Initially, the clock period was kept as 2000ns with 0ns rise time and 1000ns fall time. This changed later depending upon the optimization type.

Finally, the design was compiled after setting the above parameters. The area, timing and power reports were obtained both the parallel and pipelined methods after compilation from the following optimization methods:

B. Optimization Methods

1) No Optimization: In this method, no constraints were made on area or power and the clock was kept as 2000ns with 0ns rise time and 1000ns fall time. The area, timing and power reports were obtained after compilation.

2) Area Optimization: In this method, the area obtained from the previous method was noted and an area less than that was kept as a constraint for the 'maximum area' parameter. The clock was kept as 2000ns with 0ns rise time and 1000ns fall time. The area, timing and power reports were obtained after compilation.

3) Timing Optimization: In this method, constraints from the previous method were reset and a clock period smaller than the delay from Area Optimization was initiate taking care that the slack was not negative. The 'maximum area' parameter was cleared and the clock was kept at 2ns with 0ns rise time and 1ns fall time, giving a slack of 0.00 ns. The area, timing and power reports were obtained after compilation.

4) Power Optimization: In this method, the lowest power obtained from the above 3 methods was noted and a power level lower than that was kept as a constraint for 'max total power' parameter. There were no area or timing constraints and the clock was set back to 2000ns with 0ns rise time and 1000ns fall time. The area, timing and power reports were obtained after compilation.

TABLE I
Optimization Methods

1	No Optimization
2	Area Optimization
3	Timing Optimization
4	Power Optimization

IV. Results

A. Algorithm Correctness

Before testing for optimization, we made sure that both the pipelined and the parallel algorithms were working. This was done by comparing the theoretical values of sines and cosines to the numbers obtained during simulation. Also, another way of checking both the codes were correct was by checking their sine and cosine values and making sure they matched each other for the same angles.

Figure 4 and 5 show the sine and cosine values obtained using the parallel algorithm where Xout = Cosine and Yout = Sine.

Figure 6 and 7 show the sine and cosine values obtained using the pipelined algorithm where Xout = Cosine and Yout = Sine.

B. Optimization Results

Each algorithm, parallel and pipelined were optimized separately for area, speed and power.

Here is a summary of the area, timing and power optimization results for the parallel and pipelined algorithms.

1) Parallel Algorithm: The optimization results for the parallel algorithm are summarized in tables II-V.

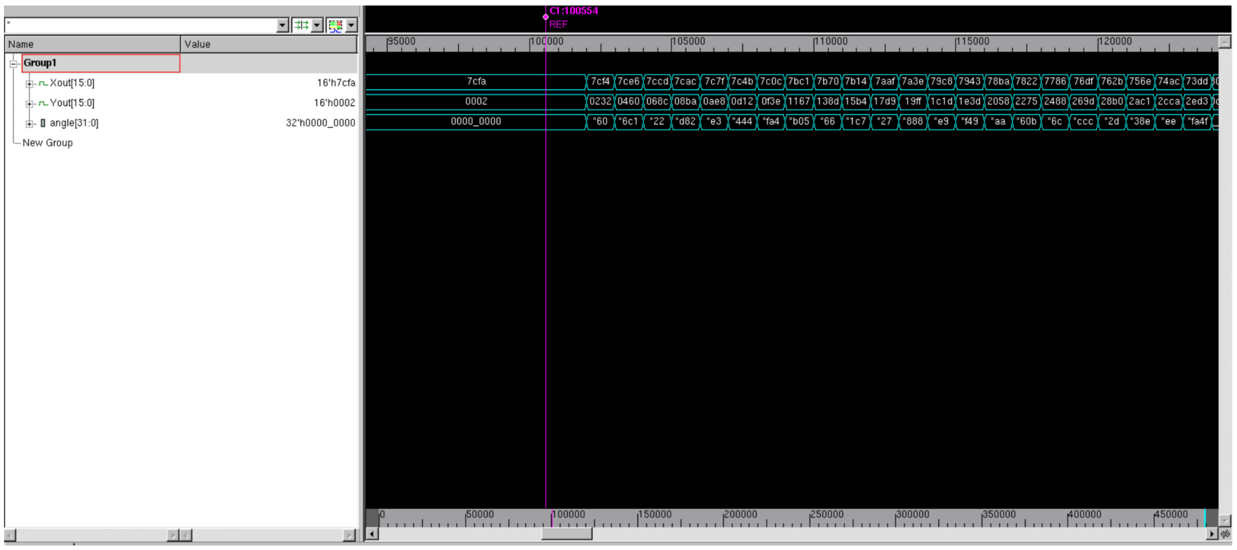

Fig. 4. Correctness of Parallel Algorithm (a)

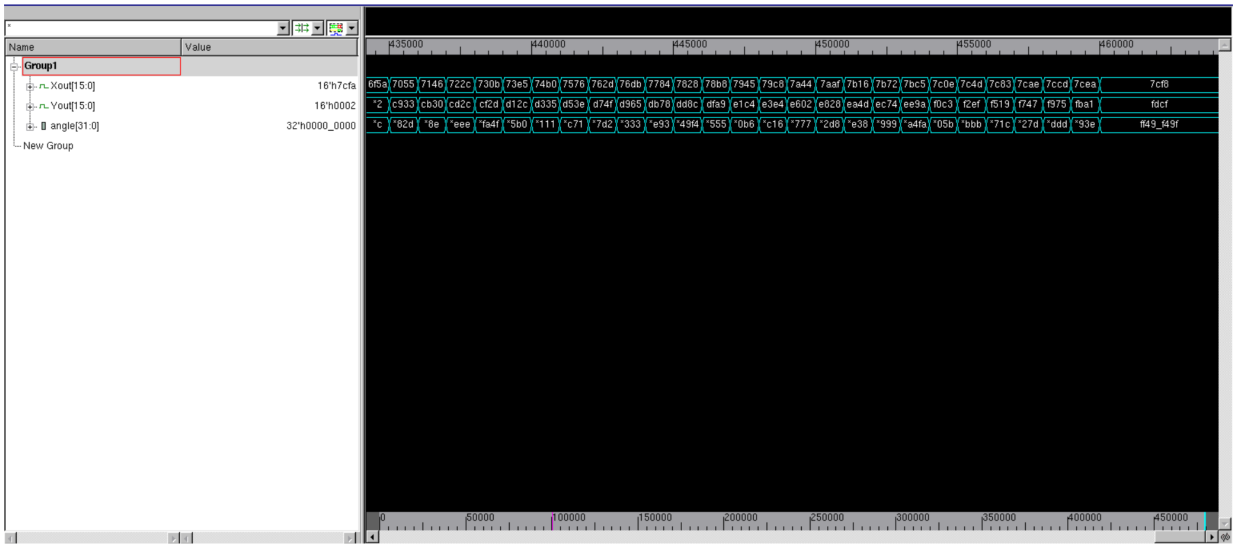

Fig. 5. Correctness of Parallel Algorithm (b)

TABLE II
PARALLEL: NO OPTIMIZATION

Area (uM)	20147
Arrival Time (nS)	8
Power (mW)	0.106

TABLE III
PARALLEL: AREA OPTIMIZATION

Area (uM)	19922
Arrival Time (nS)	8
Power (mW)	0.1063

2) Pipelined Algorithm: The optimization results for the pipelined algorithm are summarized in tables VI-IX.

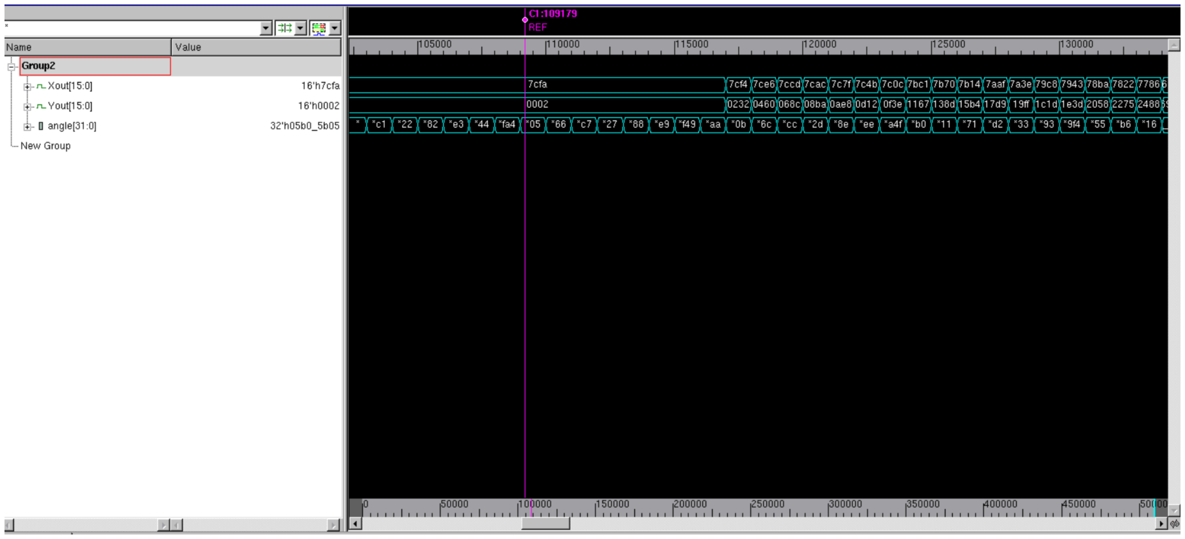

Fig. 6. Correctness of Pipelined Algorithm (a)

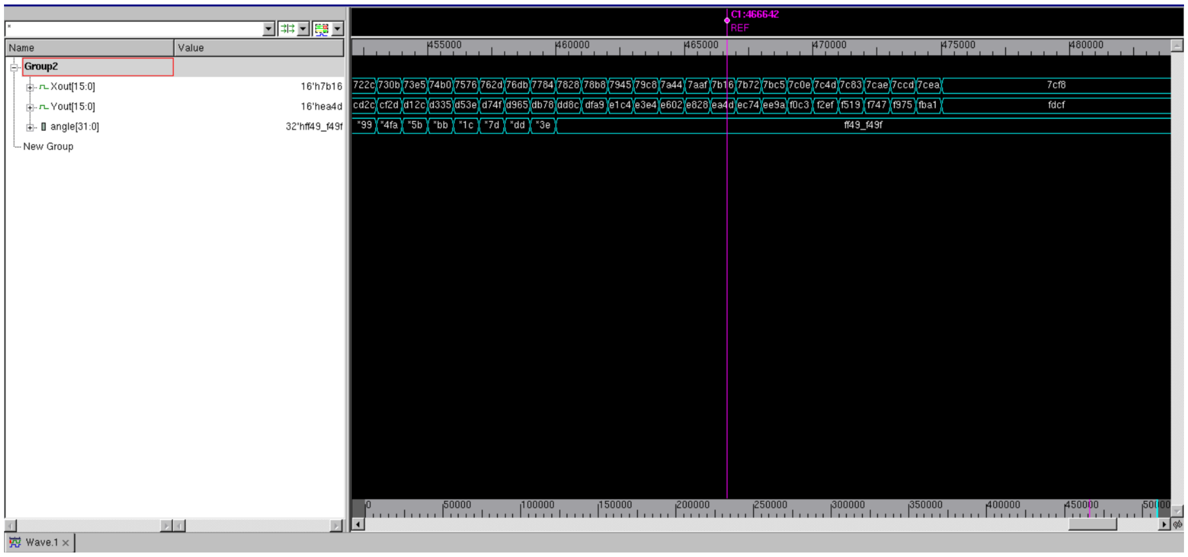

Fig. 7. Correctness of Pipelined Algorithm (b)

TABLE IV
PARALLEL: TIMING OPTIMIZATION

Area (uM)	20147
Arrival Time (nS)	8
Power (mW)	11.83

TABLE V
PARALLEL: POWER OPTIMIZATION

Area (uM)	20138
Arrival Time (nS)	8
Power (mW)	0.1057

C. Comparison

After all the optimization methods are implemented, the parallel and the pipelined algorithms can be compared to each other in terms of best area, speed and power characteristics. This comparision is shown in Table X.

TABLE VI
PIPELINED: NO OPTIMIZATION

Area (uM)	27523
Arrival Time (nS)	2.02
Power (mW)	0.1656

TABLE VII
PIPELINED: AREA OPTIMIZATION

Area (uM)	27139
Arrival Time (nS)	1.98
Power (mW)	0.1631

TABLE VIII
PIPELINED: TIMING OPTIMIZATION

Area (uM)	27264
Arrival Time (nS)	1.94
Power (mW)	20.42

TABLE IX
PIPELINED: POWER OPTIMIZATION

Area (uM)	27225
Arrival Time (nS)	2
Power (mW)	0.1629

TABLE X
PARALLEL VS PIPELINED ALGORITHM

Parameter	Parallel	Pipelined
Area (uM)	19922	27139
Arrival Time (nS)	8	1.94
Total Power (mW)	0.1057	0.1629

V. CONCLUSION

The parallel and pipelined architectures of CORDIC are implemented and optimized for Area, Power and Speed. From the results it can be concluded that the parallel algorithm has better area and power performance. The frequency of the parallel clock is determined by the combined latency of the combinational logic. Hence speed of the parallel implementation can be improved by optimizing the combinational logic. The pipelined algorithm provides better speed performance but has reduced performance in terms of area and power, due to the pipelined registers. The clock frequency of pipelined depends on the maximum latency of each block. Hence, depending upon the need of the application, either parallel or pipelined algorithm can be implemented.

REFERENCES

[1] J. E. Volder., *The CORDIC Trigonometric Computing Technique*, IRE Trans. on Electronic Computers, 1959, pp. 330-334.

[2] C. Ramanamma., *Bit Serial Iterative Cordic Implementation for the Calculation of Trigonometric Functions*, IJSERT, ISSN 2319-8885 Vol.03, Issue.01, January-2014, Pages:0073-0080

[3] Shah, Ankit, Oza, Saharsh, Thokala, Tarun and Gujjar, Pratik., *Hardware Architecture for High-Radix Adaptive CORDIC Algorithm*, 10.13140/RG.2.2.18776.49923.

[4] Deprettere E., Dewilde P., and Udo R., *Pipelined CORDIC Architecture for Fast VLSI Filtering and Array Processing*, Proc. ICASSP'84, 1984, pp. 41.A.6.1- 41.A.6.4.

[5] Hu Y. H., *Pipelined CORDIC architecture for the implementation of rotational based algorithm*, in Proceedings of the International Symposium on VLSI Technology, Systems and Applications, p. 259, May 1985.

[6] Ray Andraka., *A survey of CORDIC algorithms for FPGA based computers*, Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays ,pages 192-200, New York.

[7] A. A. J. de Lange and E. F. Deprettere., *Design and implementation of a floating-point quasi-systolic general purpose CORDIC rotator for high-rate parallel data and signal processing*, in Proceedings of the 10th IEEE Symposium on Computer Arithmetic, pp. 272281, June 1991.

[8] A. Avizienis., *Signed-digit number representation for fast parallel arithmetic*, IRE Transactions on Electronic Computers, vol. 10, pp. 389400, 1961.