Reduced Latency Square- Root calculation for Signal Processing using Radix-4 Hyperbolic CORDIC

Aishwarya Kumari[1] , D.P.Acharya[1]
[1] Department of Electronics and Communication Engineering, National institute of Technology, Rourkela

Abstract. Till now the CORDIC algorithm is primarily used to calculate only trigonometric operations. In the proposed work, we extend the Radix-4 CORDIC algorithm to the hyperbolic vectoring mode to implement fast computation of Square-Root. This paper illustrates the implementation of a regular VLSI architecture for Radix-4 hyperbolic vectoring CORDIC on FPGA platform. The speed can further be increased with higher version of FPGA devices. A comparison between Radix-2 and Radix-4 CORDIC algorithm based on the simulation results is also presented in the work.

Keywords: CORDIC algorithm, Radix-4, pipelined architectures**.**

1 Introduction

The ***CO***-ordinate ***R***otation ***DI***gital ***C***omputer (CORDIC) has been a prevailing computational tool for a wide range of applications. This strategy has expanded in preponderance especially in mathematical applications. Volder introduced CORDIC algorithm method for calculation of trigonometric functions as well as for complex binary techniques using CORDIC algorithm with its two different operation modes, which are vectoring and rotation mode. CORDIC based architectures have been used for inversion of matrix, Eigen-value calculations, singular value decomposition (SVD) algorithms, logarithmic functions, multiplication of complex numbers, orthogonal transformations, etc.

Conventionally, CORDIC algorithm is implemented using Radix-2 microrotations. Later, various algorithms and architectures have been developed for increasing throughput of the algorithm through the pipelined implementation. The CORDIC algorithm has become very much popular majorly because of its efficiency for cost efficient implementation of various applications. Its main attraction is its ability to provide simple architecture because it only uses shift, add, and subtract operations. It is also clear that it is really going to get much better shape in the future. In the present scenario, CORDIC is finding its great use in embedded system processors. The CORDIC algorithm is described suitable for the use in a special purpose computer where most of the computations involve basic trigonometric functions[2]. After completing 50 years of its invention, The basic evolutions made in the CORDIC algorithm and their architectures along with their effectiveness and applications in the coming times are illustrated in [1]. Various authors presented the restrictions of the

numerical values of functional arguments, which are given to the CORDIC units with an emphasis on the binary as well as the fixed point implementations[4]. Villalba discussed the Radix-4 CORDIC algorithm for circular vectoring mode[5]. In paper [5], authors presented that the proposed Radix-4 circular CORDIC algorithm in vectoring mode has a similar recurrence as the Radix-4 division algorithm and some dedicated studies are presented concerning the vectoring mode. In [6], the parallel Radix-4 architecture is implemented to show the latency and the hardware improvements to reduce the area. Radix-4 architecture for rotation mode is designed in [7] where, it can be seen that the total iterations' count in Radix-4 is half as compared to Radix-2. Hence, we can see that most of the work in the area of CORDIC is limited to rotational mode only specially for calculating trigonometric functions. Therefore, Radix-4 CORDIC can be used for hyperbolic vectoring mode too. The time required for the computation of square root can be reduced; which can be very useful for the works of signal processing.

2 CORDIC Algorithms

The Radix-2 and Radix-4 CORDIC algorithm are presented here in brief. The CORDIC algorithm can be altered to compute various hyperbolic functions. Hence, it is reformulated to a generalized form, suitable to execute rotations in circular, hyperbolic and linear coordinate systems. For this, a variable 'p' is added extra, which takes different values for different co-ordinate systems. The value of 'p' can be p =1,0 or,-1 and

$$\beta_{\mathrm{m}} = tan^{-1}(2^{-\mathrm{m}}) , (2^{-\mathrm{m}})\ or, tanh^{-1}(2^{-\mathrm{m}});$$

where, the generalised CORDIC algorithm is working respectively in circular, linear or hyperbolic coordinate systems. The generalized CORDIC is formulated as follows[1]:

$$\begin{aligned} x_{m+1} &= x_m - p\sigma_m 2^{-m} m \\ y_{m+1} &= y_m + \sigma_m 2^{-m} x_m \\ z_{m+1} &= z_m - \sigma_m \beta_{\mathrm{m}} \end{aligned} \tag{1}$$

Where,

$$\sigma_m = \begin{cases} sign(z_m); \text{for rotation mode} \\ -sign(y_m); \text{for vectoring mode} \end{cases}$$

In Radix-2 CORDIC algorithm, to have n bits output precision n clock cycles are required. Hence, latency is more. The latency of computation is a major drawback of CORDIC algorithm. In various signal-processing applications, fast computation of Square-Root is required. So, attempts are made to reduce latency of computation for calculation of Square-Root.

In the following section, a vectoring mode Radix-4 hyperbolic CORDIC algorithm is developed, which is used in the calculation of square root. To ensure the convergence,

the values of w_m are taken as, $w_m = 4^m y_m$.

The equations for Radix-4 CORDIC are as follows [5]:

$$\begin{aligned} x_{m+1} &= x_m + \sigma_m 4^{-2m} w_m \,, \\ w_{m+1} &= 4(w_m + \sigma_m x_m) \,, \\ z_{m+1} &= z_m - \beta_{\mathrm{m}}(\sigma_m) \,, \end{aligned} \tag{2}$$

$$\text{Here, } \beta_{\mathrm{m}}(\sigma_m) = \tanh^{-1}(\sigma_m 4^{-m}) \,.$$

Here, σ_i takes values $\{-2, -1, 0, 1, 2\}$. The scale factor here is

$$\mathrm{K} = \prod_m (1 + \sigma_m^2 4^{-2m})^{1/2} \quad .$$

The scaling factor 'K' varies with iterations, as it varies with the σ_m values. Its value ranges from 1.0 to 2.62. The Radix-4 CORDIC algorithm has two problems: viz complexity of selection of σ_m and variable nature of scale factor.
By selecting four different comparison points, the value of σ_m is calculated. In case of circular coordinate CORDIC, the four different comparison points are taken as,

$$\mathrm{P}_m(\pm 1) = \begin{cases} \pm \dfrac{x_0}{2}; if\ m = 0 \\ \pm \dfrac{x_1}{2}; if\ m \geq 1 \end{cases} \tag{3}$$

$$\mathrm{P}_m(\pm 2) = \begin{cases} \pm \dfrac{3x_i}{2}; if\ m \leq 0 \\ \pm \dfrac{3x_2}{2}; if\ m \geq 2 \end{cases} \tag{4}$$

Since we are using hyperbolic coordinate CORDIC, therefore iteration $m = 0$ is invalid here. The iteration count will start from $m = 1$. The calculated values for σ_m are

$$\sigma_m = \begin{cases} +2; if\ w_m > \mathrm{P}_m(2) \\ +1; if\ \mathrm{P}_m(1) < w_m \leq \mathrm{P}_m(2) \\ 0; if\ \mathrm{P}_m(-1) < w_m \leq \mathrm{P}_m(1) \\ -1; if\ \mathrm{P}_m(-2) < w_m \leq \mathrm{P}_m(-1) \\ -2; if\ w_m \leq \mathrm{P}_m(-2) \end{cases} \tag{5}$$

Due to most popular Area-Delay-Accuracy trade off [1], reducing latency in case of Radix-4 will increase the area by a smaller amount and will decrease the accuracy too.

3 Proposed Architecture for Radix-4 hyperbolic CORDIC algorithm

In the following section, we present the pipelined architecture of Radix-4 hyperbolic CORDIC algorithm.

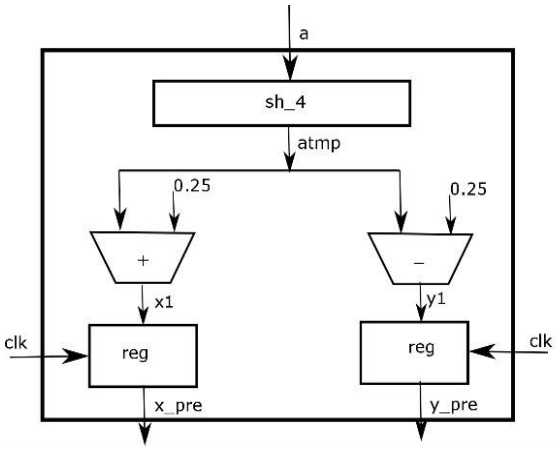

Fig. 1. Architecture of Pre-processing block in unscaled CORDIC block

The proposed architecture consists of two parallel operations:

i) Unscaled CORDIC architecture
ii) Scale factor computation architecture

The architecture of pre-processing block and first iteration is shown in fig 1 and fig 2 respectively. This architecture uses 33-bits precision. The number of iterations are three; which is half of number of iterations required in Radix-2 CORDIC.

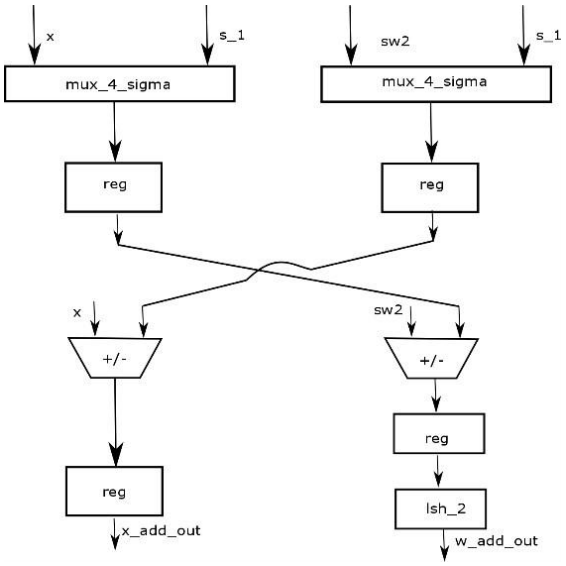

Fig. 2. Architecture of first iteration in unscaled CORDIC block

To calculate the value for σ, combinational block consisting of comparators and multiplexers are used. The output of this block is used in both parallel operations. The computation for scale factor is carried out parallely with CORDIC iterations. After

calculating scale factor for each operation, these values are stored in the LUT, which provides the final scale factor ‘K’. The final output of unscaled CORDIC block is divided by this value of scale factor to get the square root of given input.

4 Results of Implementation in FPGA

The CORDIC architecture presented here consists of three stages and a word length of 33 bits. Out of these 33 bits, 9 bits are used as integer points and remaining as fractional points. The MSB is taken as sign bit. The FPGA used here has following specifications mentioned in table 1:

TABLE 1. FPGA DEVICE & SIMULATION ENVIRONMENT

FPGA	Xilinx Virtex7
Device	XC7VX690T
Package	FFG1157
Synthesis tool	XST(VHDL/Verilog)
Speed	-3
Preffered language	Verilog
Simulator	Isim

To calculate the square root of a number ‘a’, in cordic initial values of x and y are taken as

$$x = a + 0.25 \text{ and } y = a - 0.25 \ .$$

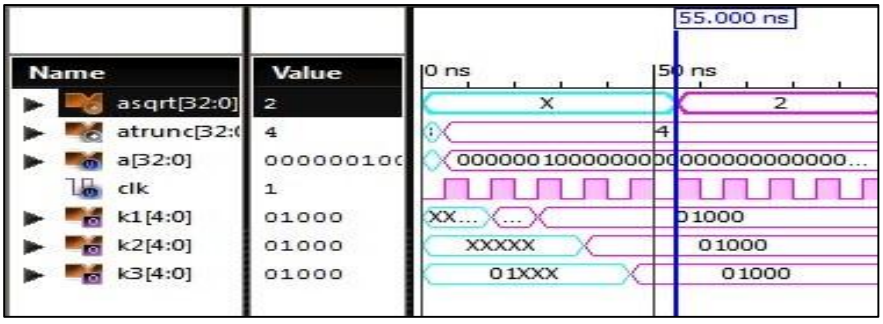

Fig. 3. Simulation result of Square-Root calculation using Radix-4 CORDIC

The input here is taken as ‘a’ and output is ‘asqrt’. Here ‘atrun’ is showing the integral value of ‘a’ without its fractional part. In ‘asqrt’ too the fractional part is truncated, which means only upper 9 bits are taken as output. Pipelined architecture uses structure similar to that of parallel implementation of CORDIC. The only difference is pipelining registers are inserted after every iterations**.** The scale factor for each iteration is truncated to five bits in which first two bits represents integral part and remaining three represents fractional part.

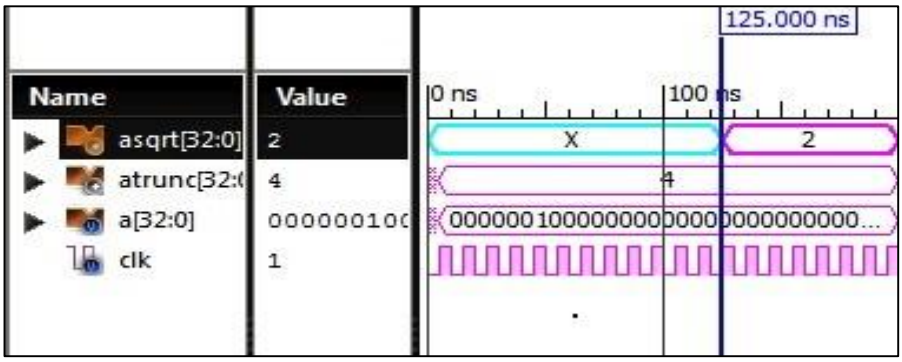

Fig. 4. Simulation result of Square-Root calculation using Radix-2 CORDIC

5 Conclusion

In the proposed work, a normally scaled Radix-4 hyperbolic CORDIC architecture is presented. The calculated latency of operation here is five clock cycles as shown in fig 3; which is approximately half of the Radix-2 CORDIC architecture shown in fig 4. But, it involves comparatively more hardware than Radix-2 CORDIC because it parallelly compute the scale factor. This study reveals that the speed optimized Radix-4 CORDIC architecture designed can be suitable for applications in real time.

References

1. Pramod K. Meher. : 50 Years of CORDIC: Algorithms, Architectures, and Applications. In: IEEE Transactions on Circuits and Systems, September 2009.
2. Volder J. E.: The CORDIC trigonometric computing technique. In: IRE Trans. Electron. Computers, vol. EC-8, pp. 330–334, Sept. 1959.
3. Waltherm S.: A unified algorithm for elementary functions. In: in Proc. 38th Spring Joint Computer Conf., Atlantic City, NJ, 1971, pp.379–385.
4. X. Hu.: Expanding the Range of Convergence of the CORDIC Algorithm. In: IEEE Transactions on Computers, January 1991.
5. Villalba J.*:* Radix-4 Vectoring CORDIC Algorithm and Architectures. In IEEE Transactions on Application Specific Systems, Architectures and Processors, 1996.
6. Lakshmi B.: VLSI architecture for parallel Radix-4 CORDIC. In: ELSEVIER transactions on Microprocessors and Microsystems 37 (2013) 79–86.
7. Antelo E.: High performance rotation architectures based on Radix-4 cordic algorithm. In: IEEE Transactions on Computers,Vol. 46, August 1997.