International Journal of Scientific & Engineering Research, Volume 4, Issue 8, August 2013
ISSN 2229-5518

A High Performance and Low Latency Fpga Implementation of Cordic Algorithm

M.A Darshan
Department of Electronics and Communication Engineering,
PES Institute of Technology, Bangalore, Karnataka-560085, INDIA
Email: darshanmakam@gmail.com

Abstract— CORDIC is generally faster than other approaches when a hardware multiplier (e.g., a microcontroller) is not available, or when the number of gates required to implement the functions it supports should be minimized (e.g., in an FPGA). On the other hand, when a hardware multiplier is available (e.g., in a DSP microprocessor), table-lookup methods and power series are generally faster than CORDIC. In recent years, the CORDIC algorithm has been used extensively for various biomedical applications, especially in FPGA implementations. Here we use Unfolded architecture for CORDIC in order to achieve low latency for rotation and various functions such as multiplication, division logarithmic exponential and trigonometric functions. The approach of this architecture provides high performance and low latency field programmable gate array implementation of rotational CORDIC algorithm. CORDIC device is highly suitable for computing many functions with precisely the same hardware, so they are ideal for applications with an emphasis on reduction of cost (e.g. by reducing gate counts in FPGAs) over speed and low clock rate can be utilized to meet low power consumption requirement.

Index Terms— CORDIC, FPGA, Pre-computation, Radix-4, Unfolded architecture, vectors.

1 INTRODUCTION

The COordinate Rotation DIgital Computer (CORDIC) algorithm was described by J. E. Volder in 1959 [1] for the computation of trigonometric functions. The CORDIC is known as an efficient method for the computation of these elementary functions such as multiplication, division, logarithmic and exponential functions in addition to the computation of two dimensional vector rotations. It's one among the best compromise between the look up table approach which require more memory, and polynomial approximation method, they are slow in obtaining the desired accuracy or precision. These transcendental functions are the core for many applications such as digital signal processing, graphics, image processing, and kinematic processing [3]-[6]. The application of CORDIC algorithm is extended to Neural Networks, satellite communication systems, wireless devices to mention a few. Research in the area of expanding the allowed ranges of the input variables for which accurate output values can be obtained is presented. It is observed that the latency and hardware of radix-2 CORDIC can be reduced by employing redundant radix-4 arithmetic and pre-computing the direction of rotations.
Thus the architecture proposed to reduce the latency which helps in implementing VLSI designs.

2 CORDIC ARCHITECTURES

The architecture for implementation of CORDIC Algorithm with iterative nature depends on some of the important factors such as speed, area of silicon, power consumption. In general, the architectures can be broadly classified as folded and unfolded. The conventional radix-2 CORDIC algorithm was implemented using word serial and pipelined architectures for non-redundant radix-2 arithmetic, additionally it is advantageous if relatively long stream of data have to be processed in the same manner. The computation time and the achievable throughput of these CORDIC architectures depends on the delay of carry propagate additions/subtractions. The two common redundant number systems employed in CORDIC arithmetic are the signed-digit (SD) [7] and the carry-save (CS) [8] number systems. In both SD and CS number systems, each number can be represented in multiple ways, and this redundancy restricts the carry propagation from each stage to its immediate more significant bit position. This architecture helps us to develop constant scale factor methods using signed digit arithmetic [9] [10] and carry save arithmetic [11]. The delay of every iteration is differentiated into two delays, the delay to predict the new directional rotation and the delay in the application of computed rotation. The total computation delay is reduced by eliminating iterative nature in the x/y-data path completely [13] and partially [14]. The latency of this CORDIC architecture is improved by radix-4 arithmetic. Hence we go with an unfolded architecture using radix-4 arithmetic.

3 CORDIC ALGORITHM

The CORDIC algorithm [1] is a well-known method to compute of two dimensional rotation of vector in linear, circular and hyperbolic coordinate system. The CORDIC iteration is completely free from multipliers and requires only shift and add operations. The CORDIC method can be implemented in two different modes, namely, rotation or forward rotation mode and vectoring or backward rotation mode. For a given initial coordinates of a vector and target angle, the rotation mode is used for general rotation of vector by the given angle and to compute elementary operations such as trigonometric functions, multiplication, exponential, and hyperbolic func-

tions depending on the coordinate system. The vectoring mode is used to determine the angular argument of the original vector, and to compute division and logarithmic functions, for the given initial and final coordinates of vector.

The iteration equations of the CORDIC algorithm at the (i+1)th step are,

$$x_{i+1} = x_i - m\sigma i\, \rho^{-Sm,i}\, y_i \qquad (1a)$$

$$y_{i+1} = \sigma_i\, \rho\text{-}Sm,i\, x_i + y_i \qquad (1b)$$

$$z_{i+1} = z_i - \sigma i\, \alpha_{m,i} \qquad (1c)$$

where, σi ∈ {−1, +1} represents the direction of rotation in each iteration, ρ represents the radix of the number system, m steers the choice of linear (m = 0), circular (m = 1), or hyperbolic (m = −1) coordinate systems, Sm,i is the nondecreasing integer shift sequence, and αm,i is the elementary rotation angle. The relation between αm,i and Sm,i is

$$\alpha_{m,i} = 1/\sqrt{m}\ \tan\text{-}1\ (\sqrt{m}\ \rho^{-Sm,i}) \qquad (2)$$

The value of σi is determined by the following equation assuming that vector is in the first or fourth quadrant:

$$\sigma_i = \begin{cases} \text{sign}(z_i) & \text{for rotation} \\ -\text{sign}(y_i), & \text{for vectoring} \end{cases} \qquad (3)$$

Where, zi and yi are the steering variables in rotation and vectoring mode respectively. For every CORDIC rotation the length of the vector alters. Hence, the resultant vector after n iteration has to be scaled by the scale factor, K

$$K = \prod_{i=0}^{n-1} k_i \qquad (4a)$$

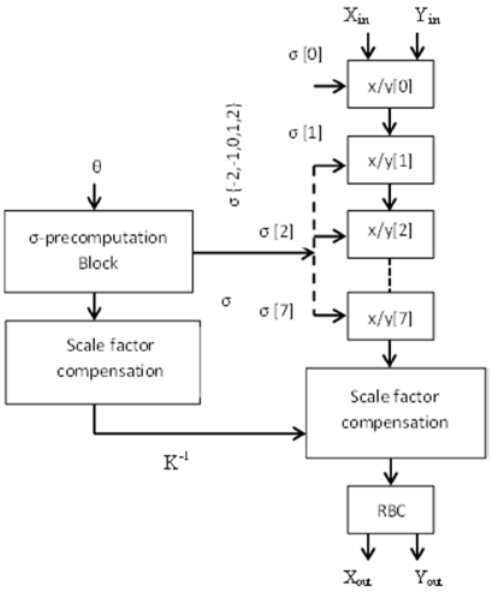

Fig. 1. Organization of the proposed CORDIC for 16-bit precision.

$$k_i = \sqrt{1+m\sigma^2_i}\,\rho^{-2Sm,i} \qquad (4b)$$

3.1 Radix-4 Cordic Algorithm

The iteration equations of the radix-4 CORDIC algorithm in rotation mode at the $(i + 1)^{th}$ step are as follows:

$$x_{i+1} = x_i - \sigma_i\ y_i\ 4^{-i} \qquad (5a)$$

$$y_{i+1} = \sigma_i\ \ x_i\ 4^{-i} + y_i \qquad (5b)$$

$$z_{i+1} = z_i - \tan^{-1}(\sigma_i\ 4^{-i}) \qquad (5c)$$

Where σi ∈ {−2, −1, 0, 1, 2} and $\tan^{-1}$ (σi4−i) is an elementary angle. This algorithm results in a variable scale factor and has to be computed in every iteration. To implement perfect rotation, final coordinates must be multiplied by the reciprocal of scalefactor, K−1

$$K^{-1} = \prod_{i>0} k_i^{-1} = \prod_{i>0}(1 + \sigma_i^2\, 4^{-2i})^{-1/2} \qquad (6)$$

4 PROPOSED ARCHITECTURE

Here we make use of unfolded architecture to implement the rotational CORDIC algorithm. In order to achieve the latency improvement over unfolded architecture we make use of radix-4 arithmetic and precomputation of direction of rotation in this architecture [9] [12]. Fig. 1 shows the architecture of the proposed CORDIC rotator. The direction of rotations σi for the given target angle θ are determined before the actual CORDIC rotation starts iteratively in the x/y-datapath. These precomputed σi values are used in the x/y-datapath to implement rotation as described in the subsequent sections.

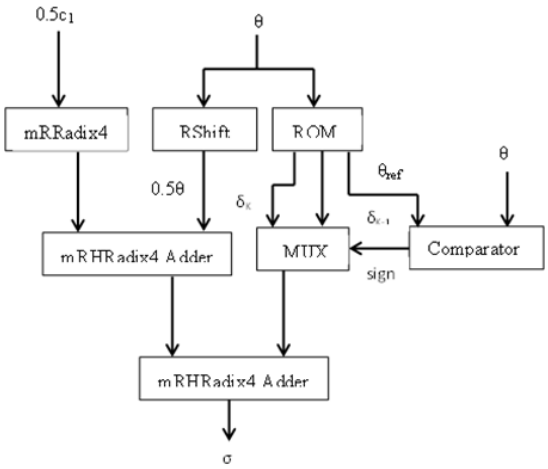

Fig 2: Realization of σ Computation

4.1 Σ-PRECOMUTATION

This block computes directions in minimally redundant radix-4 representation (mRRadix4) using the linear relation between the rotation angle θ and constructed binary representation of directions of microrotations d [19]

$$d = 0.5\theta + 0.5_{c1} + \delta \qquad (7)$$

where $c1 = 2 - \sum_{i=0}^{\infty} (2^{-i} - \arctan(2^{-i}))$, $\delta = \sum_{i=0}^{n/3} (di\varepsilon i)$ and $\varepsilon i = 2^{-i} - \arctan(2^{-i})$. Fig. 2 shows the realization of Eq. 7 using a ROM, comparator, minimally redundant hybrid radix-4 adder (mRHRadix4 Adder).

ROM: A 32×48 ROM is selected to store the 28 approximate offset values for any input angle in the range (−π/2, π/2) for 16-bit precision [12]. Each location of ROM contains a reference angle θref , and two offset values (δk ,δk−1) corresponding to that reference angle and the previous reference angle respectively. A 2-to-1 multiplexer is used to select one of the two offset values accessed from the ROM for the given input angle based on the select signal. This select signal is generated by a comparator. Since the size of ROM increases exponentially with precision, offset value δ is partitioned into δROM and δr for precision greater than 16-bit. For any input angle within the acceptable range, δROM is stored in ROM and δr can be computed employing a binary tree adder circuit.

Comparator: The comparator generates sign signal '0' (θ > θref for selecting δk) or '1' (θ < θref for selecting δk−1) with a delay of (log2 n)tmux.

mRHRadix4 Adder: The 16-bit redundant radix-4 representation of 0.5c1 is added to nonredundant radix-4 representation of 0.5θ and δROM using two maximally redundant hybrid radix-4 adders to obtain σ ∈ {−2,−1, 0, 1, 2} each with a delay of 2tFA, where tFA is one full adder delay.

4.2 X/Y-DATAPATH

The x/y-datapath (see Fig. 1) requires eight microrotation stages for 16-bit precision to compute x/y coordinates. The maximally redundant radix-4 (MRRadix4) adder/subtractor is designed to realize Eq. 5a and Eq. 5b. Each microrotation stage of the x/y-datapath requires MRRadix4 adder/subtractor, 2-to-1 multiplexer and an AND gate. The SD adder/subtractor is controlled by the sign of σi value. The accuracy of the computation of x/y-coordinates is affected by two primary sources of error, angle approximation and rounding error. The first type of error depends on how closely z variable is driven to zero using finite number of elementary angles. This introduces angle approximation error due to the quantized representation of rotation angle. The data path accuracy can be increased by performing a large number of iterations to reduce the angle approximation error ψ given by

$$\Psi = \theta - \sum_{i=0}^{n-1} \sigma i \alpha i \qquad (8)$$

such that | ψ | < αn−1 and is negligible in practical computation [2]. The second type of error is due to the limitation of the finite length of storage elements available for the implementation of the CORDIC rotator. These finite length storage elements introduce rounding error due to the truncation of variables during the computation. The numerical analysis of these errors is provided in [18]. Since the truncation of intermediate results after every n iterations introduces maximum rounding error of log2 n bits in binary arithmetic, an additional log2 n guard bits must be considered in the implementation of this algorithm [2]. Hence, we have considered 20-bit wide data bus for 16-bit precision to design x/y-datapath in the proposed architecture.

4.3. SCALE FACTOR COMPUTATION

Redundant radix-4 CORDIC algorithm results in a variable scale factor,

$$K\text{-}1 = \prod_{i=0}^{n/4} (1 + \sigma i 24\text{-}2i)\text{-}1/2 \qquad (9)$$

Where σi ∈ {−2,−1, 0, 1, 2}. We consider only the first (n/4 + 1) iterations in the computation of K−1, since ki-1 = (1 + 4−2i)−1/2 becomes unity thereafter. Thus, for this 16- bit implementation, we compute scale factor for the first five iterations. The hardware overhead for this scale factor computation in every iteration can be reduced by storing all possible scale factors in memory. Since σi 2can take any one of the three values {0, 1, 4} in each iteration i, it is required to store 35 possible scale factors in ROM for 16-bit precision. The size of ROM can be reduced by using Taylor series expansion of the scale factor [15]. The scale factor can be approximated by the first two terms of its Taylor series expansion for i ≥ [n/8 + 1] for n-bit precision. Hence for 16-bit precision, the scale factor corresponding to the first three iterations is obtained from 27 × 16 ROM and this value is used to compute the scale factor for the remaining two iterations using a combinational circuit as shown in Fig.3.The 27 possible values for the scale factor K(2)-1 corresponding to the first three iterations are computed as

$$k_i^{-1} = (1+\sigma_i^2 4^{-2i})^{-1/2} \qquad (10a)$$

The scale factors for the remaining two iterations k3−1and k4-1 are obtained by performing the shift and subtraction operations over the scale factor obtained from ROM. This is implemented by realizing the first two terms of their Taylor series expansions.

4.4. SCALE FACTOR COMPENSATION

The final x/y-coordinates are scaled by performing two parallel multiplications corresponding to K−1.x and K−1.y. This is implemented using redundant tree adder circuit after the last iteration. The number of partial products can be decreased by reducing the number of nonzero digits in the multiplier. The canonical signed digit (CSD) representation is preferred for representing scale factor as it can reduce the average number of nonzero digits to n/3 [17].

4.5. Redundant to Binary Converter (RBC)

RBC converts scaled x/y-coordinates from radix-4 signed binary digit representation into radix-2 binary. Fast converters can be designed using tree based or carry select approaches wherein both employ multiplexer cells. We have designed RBC using tree based approach, which generates all carries

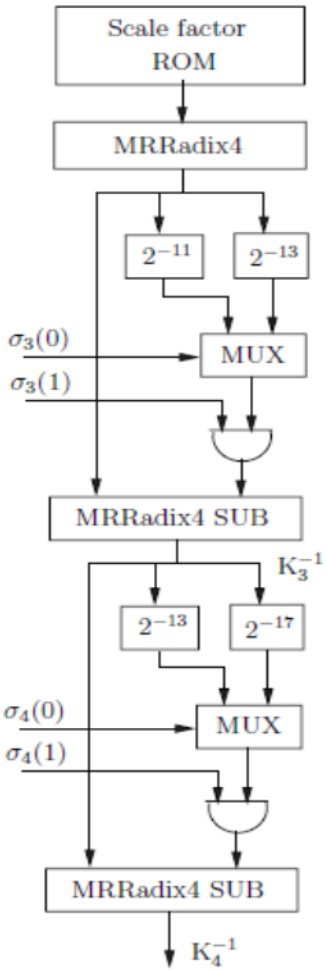

Fig 3. Scheme for scale factor computation.

after (log2 n)tmux delay. These multiplexer based converters are faster than binary ripple carry adder with a delay of ntFA or fast carry propagate adder with a delay of (log2 n)tFA [19].

5 EVALUATION

The estimated latency of the proposed unfolded rotational CORDIC architecture is compared with the various unfolded architectures available in the literature. The latency comparison and basis for deriving these metrics is summarized below. The word length of datapath is considered as n to simplify comparison.

The proposed architecture requires n/2 microrotation stages with stage delay as the sum of the delays of MRRadix4 adder/subtractor (t4−2SD add/sub), 2-to-1 multiplexer (t2−1mux) and an AND gate (tand). The delay to compute the direction of rotations is the sum of the delays of ROM, comparator, multiplexer, minimally redundant hybrid radix-4 signed digit adder (mRHRadix4 adder).

In [9], double rotation and correcting rotation architectures compute the directions iteratively using the estimated value of zi. Both these architectures require 1.5n iterations with stage delay determined by the z-path. The stage delay of z path is sum of the delays of 3-2 SD adder/subtractor, 2-to-1 multiplexer and the delay to compute the directions of rotations (tσi). The delay for the computation of the estimated value of zi and evaluation of the selection function determine tσi .

In [11], an unfolded architecture is presented to reduce latency by predicting σi's in parallel for a certain group of iterations at a time avoiding iterative nature in the z path with n stages. The x/y-path decides the stage delay and hence the overall iteration delay corresponds to the sum of the delays of 4-2 CSA adder/subtractor and 2-1 multiplexer.

In [12], the number of stages are 3n/4 with the stage delay as the sum of the delays of 4-2 CSA adder/subtractor and 2-to- 1 multiplexer for the first n/2 iterations and with an additional 2-1 multiplexer delay for the next n/4 iterations.

In [13], flat CORDIC algorithm was proposed to eliminate iterative nature completely in the x/y-datapath for reducing the total computation time. While the directions for the first n/3 iterations are precomputed using Split Decomposition Algorithm (SDA), directions for the last 2n/3 iterations are predicted from the remaining angle after (n/3) iterations. The implementation of flat CORDIC achieves low computation time and low implementation area. However, this requires complex combinational hardware blocks with poor scalability limiting the range of input angle.

In [14], architecture is proposed for 16-bit precision to reduce latency by eliminating iterative nature partially in the x/y/z-path at the cost of scalability. For the first λ iterations of n, x/y recurrences are computed iteratively using the double rotation method [9]. It is observed from the simulation results that the best trade-off is obtained with λ = 6 and λ = 8 for a 16- and 32-bit CORDIC respectively. After λ iterations, z-path is eliminated and parallelized x/y-datapath is implemented using Wallace tree. However, this architecture has poor scalability.

The latency comparison in terms of number of stages for the proposed architecture and other architectures is presented in Table I.

TABLE I
LATENCY COMPARSION

Method	Latency(Stages)
Double iteration/Correcting iteration[16]	1.5n
Low Latency CORDIC [18]	n
Branching [17]	n
PCORDIC[19]	3n/4
Flat CORDIC [20]	Combinational
Semi flat CORDIC [21]	λ + combinational
Proposed CORDIC	n/2

TABLE II
FPGA IMPLEMENTATION RESULTS

	Sparten-3E	Sparten-6
Devices	XC3S250E	XC6SLX4
Speed grade	2	4
No. of 4 input LUT's	4,508	6840
No. od occupied slices	1,389	1,033
Delay	80ns	68ns

Implementation results: The proposed architecture is implemented with the word length of 20 bits for 16-bit precision using signed digit arithmetic on a Xilinx Sparten-3E device (XC3S250E).In this implementation, we have considered four guard bits in order to reduce the quantization errors. We obtained hardware complexity for prototype as 4,508 lookup tables occupying 1,389 slices out of available 2,448 resulting in the latency of 80 ns. The latency is further reduced to 68 ns, using Xilinx Sparten-6 (XC6SLX4) with better routing resources. Table II summarizes the resource usage for both the devices. The proposed architecture is coded using the Verilog language and synthesized to the using XILINX ISE13.1i.

6. CONCLUSIONS

In this work, the architecture is to reduce the area and computation delay of rotational CORDIC by halving the number of iterations and pre-computation of directions for all rotations. The proposed architecture is fully scalable and can be extended to higher accuracy as well. It is evident from the comparison of the proposed CORDIC with other unfolded architectures available in the literature that it achieves reduction in the number of stages while eliminating the z-path completely.

REFERENCES

[1] J. E. Volder, "The CORDIC trigonometric computing tech nique," IRE Trans. Electronic Comput., vol. 8, no. 3, pp. 330–334, Sept. 1959.

[2] J. S. Walther, "A unified algorithm for elementary functions," Proc. AFIPS Spring Joint Comput. Conf., pp. 379–385, 1971.

[3] Y.H. Hu, "CORDIC-based VLSI architectures for digital signal processing," IEEE Signal Process. Mag., vol. 9, no. 3, pp. 16–35, July 1992.

[4] A.S. Dhar, and S. Banerjee, "An array architecture for fast computation of discrete Hartley transform," IEEE Trans. on Circuits Syst., vol. 38, no. 9, pp. 1095-1098, Sept. 1991.

[5] K. Maharatna, A.S. Dhar, and S. Banerjee, "A VLSI array architecture for realization of DFT, DHT, DCT and DST," Elsevier Journal of Signal Process., vol. 81, no. 9, pp. 1813–1822, Sept. 2001.

[6] Ch. Krieger, B. Hosticka, Th. Krupp, M. Hiller, and A. Kecskemethy, "A combined hardware/software approach for fast kinematic processing," Elsevier Journal of Microprocessors and Microsystems, vol. 22, no. 5, pp. 263–275, Sept. 1998.

[7] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," IRE Trans. on Electronic Comput., vol. 10, pp. 389–400, Sept. 1961.

[8] T.G. Noll, "Carry save arithmetic for high-speed digital signal processing," In the Proc. of IEEE International Symposium on Circuits and Systems-1990, New Orleans, LA, USA, vol. 2, pp. 982–986, May 1990.

[9] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," IEEE Trans. Comput., vol. 40, no. 9, pp. 989–995, Sept. 1991.

[10] J. Duprat, and J. M. Muller, "The CORDIC algorithm: new results for fast VLSI implementation," IEEE Trans. Comput. vol. 42, no. 2, pp. 168–178, Feb. 1993.

[11] D. Timmermann, H. Hahn, and B. J. Hosticka, "Low latency time CORDIC algorithms," IEEE Trans. Comput., vol. 41, no. 8, pp. 1010– 1015, Aug. 1992.

[12] M. Kuhlmann, and K. K. Parhi, "P-CORDIC: A precomputation based rotation CORDIC algorithm," EURASIP J. Appl. Signal Process., vol. 2002, no. 9, pp. 936–943, Jan. 2002.

[13] B. Gisuthan, and T. Srikanthan, "Pipelining flat CORDIC based trigonometric function generators," Elsevier Microelectronics Journal, vol. 33, no. 1-2, pp. 77–89, Jan. 2002.

[14] H. S. Kebati, J. Ph. Blonde, and F. Braun, "A new semi-flat architecture for high speed and reduced area CORDIC chip," Elsevier Microelectronics Journal, vol. 37, no.2, pp. 181–187, Feb. 2006.

[15] E. Antelo, J. Villalba, J. D. Bruguera, and E. L. Zapata, "High performance rotation architectures based on the radix-4 CORDIC algorithm", IEEE Trans. Comput., vol. 46, no. 8, pp. 855-870, Aug. 1997.

[16] B. Lakshmi, A. S. Dhar, "High speed architectural implementation of CORDIC algorithm," In Proc. of IEEE Region 10 Conference TENCON-2008, Hyderabad, India pp. 1–5, Nov. 2008.

[17] K. Hwang, "Computer Arithmetic: Principles, Architecture and Design," John Wiley & Sons, New York, 1979.

[18] Y.H. Hu, "The quantization effects of the CORDIC algorithm," IEEE Trans. on Signal Processing, vol. 40, no. 4, pp. 834-844, Apr. 1992.

[19] K. K. Parhi, "Low-energy CSMT carry generators and binary adders," Proc. of IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 7, no. 4, pp. 450–462, Dec. 1999.