FPGA Implementation of the CORDIC Algorithm for Fingerprints Recognition Systems

Nihel Neji, Anis Boudabous, Wajdi Kharrat, Nouri Masmoudi

University of Sfax, Electronics and Information Technology Laboratory,
National School of Engineering, BP W 3038, Sfax, TUNISIA

ABSTRACT
In this paper, we propose a low-cost sequential architecture for the implementation of CORDIC algorithm in two computation modes. It suited for serial operation that performs conversion between polar and rectangular coordinate systems, essentially sin/cos, sinh/cosh and arctan computation. The design targets real time application of fingerprint recognition. We present a VHDL description of CORDIC algorithm. To reduce iteration delay, we used some combinatory blocks. Fixed point arithmetic was considered. To valid our conception and its CORDIC accuracy, we present relative error calculated in convergence range for some trigonometric and hyperbolic functions. All measurements show an enhancement compared with our previous work. The architecture was implemented and tested. The contribution of the paper includes the CORDIC design flow.

Keywords
CORDIC algorithm, fingerprint, VHDL, hardware, FPGA

1. INTRODUCTION
Fingerprint recognition systems are the focus of research and development. They allow new types of services universally available to consumers and for industrial applications. This paper is based on a project which aims to develop a fingerprint recognition system. The most difficult to implement functional blocks is Fast Fourier Transform (FFT) processor. A Coordinate Rotation Digital Computer offers an elegant way of its implementation[1]. It can be applied to FPGA applications, in which the rotation angles are usually known, the twiddle factor in FFT and kernel components in other sinusoidal transforms [2],[3]. The CORDIC scheme has been applied to the FFT processor design and found to result in significant hardware reduction in the implementation of twiddle-factor multiplications.

In this work, we exploit the FPGA circuit capacity to design a reconfigurable architecture for computation of elementary functions such as sine, cosines, exponential and arctangent using this algorithm. We focus on polynomial approximations with fixed coefficients and powers of x to search errors over a bounded interval. Then, we deal with CORDIC evaluation to calculate outputs in fixed-point-format. The obtained average of error is close to the error of polynomial approximations. This makes our method an attractive solution for signal processing applications. The remaining paper is organized as follows. Section 2 represents the previous work which proposed different types of CORDIC architectures. The CORDIC algorithm is described in Section 3. Section 4 presents the proposed architecture for rotation and vectoring mode derived from the algorithm specification. Finally, in section 5 the results of the implementation are reported and the performance comparison of proposed architecture with the other architectures available in the literature is explained. The conclusion is drawn in section 6.

2. RELATED WORK
Large numbers of architectures have been proposed in the literature for CORDIC algorithm, which vary from bit-serial implementations to word parallel pipelined architectures. The choice depends on the requirements for computing throughput and constraints that hold for area usage, latency and power dissipation. Traditionally [4], [5], implementations of the CORDIC algorithm have been carried out on word serial architectures using conventional non-redundant arithmetic with radix-2 micro-rotations and fixed point internal format.
Lang and Ercegovac [6] have proposed redundant arithmetic to the implementation of conventional radix-2 CORDIC [3], [4]. However this resulted in increasing the iteration delay and additional cost due to variable scale factor. Double rotation and correcting rotation methods [7] were proposed to implement constant scale factor CORDIC which resulted in 50% increase in number of iterations. This increase in latency is reduced by proposing branching algorithm [8], which requires additional CORDIC module to perform rotations in both directions, if the direction cannot be determined using intermediate results. The main disadvantage of branching method is the necessity of performing two conventional CORDIC iterations in parallel, which consumes more silicon area than the conventional methods. However, this method gives a faster implementation than [7]. Low latency CORDIC algorithm is proposed in [9] to achieve latency reduction by 25% compared to the method in [7].
In contrast to these methods, new algorithms are proposed in [10] and [11], which avoids the determination of direction of rotation using intermediate results of steering variable. However, there is an area cost for registers because of pipelining at the full adder level and n initial register rows for performing skew of input data. This redundant radix-2 CORDIC algorithm has been extended to radix-4 to halve the number of iterations [12]. However, the computation time per iteration increases, since it takes more time to decide amongst the five micro-rotation direction values and to select an appropriate one out of five elementary angles. Both redundant and higher radix based CORDIC algorithms are still iterative in nature and greatly restrict the speed of implementation of the algorithm. The delay of every iteration can be decomposed into two different delays, the delay to predict, the new rotation direction and the delay involved in the application of computed rotation. Improvements have been especially made in the reduction of delay to predict the new micro-rotation direction.

3. OVERVIEW OF ITERATIVE CORDIC ALGORITHM

The CORDIC computing technique was developed by J. E. Volder in the late 1959's [4] for the computation of trigonometric functions, multiplication and division operations. Walther, in 1971, has generalized this algorithm to implement hyperbolic, logarithm and exponential functions. This algorithm is iterative with an ability to decimate elementary operations with simple shift and addition operations. The number of iterations is determined by the word length of the inputs.

3.1. CORDIC modes

The CORDIC method can be employed in two different modes, namely, the rotation mode and the vectoring mode.

In the rotation mode, the coordinate components of a vector and an angle of rotation are given, and the coordinate components of the original vector, after rotation through a given angle, are computed.

In the vectoring mode, the coordinate components of a vector are given, and the magnitude and angular argument of the original vector are computed.

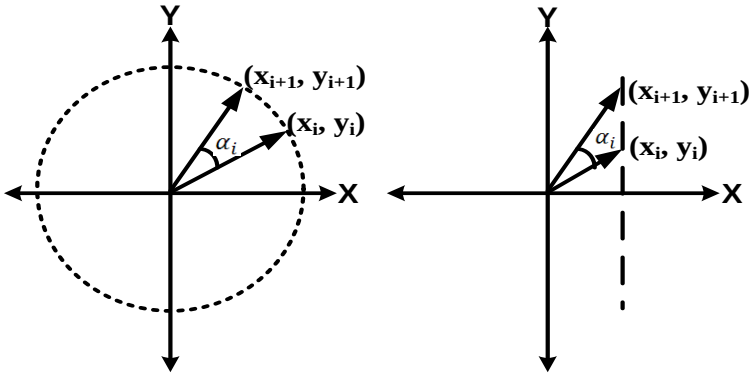

Figure 1. Graphical representation of circular and linear CORDIC

The CORDIC algorithm performs the rotation of a vector in both modes as a sequence of micro-rotations by elementary angles [4] recalled from ROM. The number of micro-rotations for a given precision is decided by radix being used for the implementation of CORDIC algorithm. The CORDIC's graphical representation is shown in Figure 2.

Here, the circular CORDIC architecture computes trigonometric function and magnitude of a vector whereas the linear mode of CORDIC architecture computes linear functions such as multiplication and division in different mode of operation i.e rotational and vectoring mode respectively.

3.2. Generalized CORDIC

The generalized iteration equations of the CORDIC algorithm [5] at the $(i + 1)^{th}$ step are as follows:

$$x_{i+1} = x_i - m\ \sigma_i y_i \rho^{-S_{m,i}} \quad (1)$$

$$y_{i+1} = y_i + \sigma_i x_i \rho^{-S_{m,i}} \quad (2)$$

$$z_{i+1} = z_i - \sigma_i \alpha_{m,i} \quad (3)$$

Where σ_i represents the choice of 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.

$S_{m,i}$ is the nondecreasing integer shift sequence, and $\alpha_{m,i}$ the rotation angle.

The latter directly depends on $S_{m,i}$ according to

$$\alpha_{m,i} = \frac{1}{\sqrt{m}} tan^{-1}(\sqrt{m}\ \rho^{-S_{m,i}}) \quad (4)$$

The value of σ_i is determined by the following equation:

$$\sigma_i = \begin{cases} sign(z_i) & \text{for rotation} \\ -sign(x_i).sign(y_i) & \text{for vectoring} \end{cases} \quad (5)$$

where z is a steering variable in rotation mode, x and y are steering variables in vectoring mode. The required micro-rotations are not perfect rotations as they increase the length of the vector.

In order to maintain a constant vector length, the obtained results have to be scaled by the scale factor K as given by

$$k_i = \sqrt{1 + m\ \sigma_i{}^2 \rho^{-2S_{m,i}}} \quad (6)$$

$$K = \prod_{i=1}^{n} k_i \quad (7)$$

where ki denotes the vector amplification factor for the ith iteration, and K is the resultant vector amplification factor after n iterations.

3.3. Outputs of the CORDIC algorithm

Table 1. Outputs of the CORDIC algorithm

Coordinate	Rotation	Vectoring
Circular (m=1)	$X_n = \frac{1}{K_1}(X * \cos Z - Y * \sin Z)$ $Y_n = \frac{1}{K_1}(Y * \cos Z + X * \sin Z)$ $\lvert Z\rvert \leq 1.7433$	$X_n = \frac{1}{K_1}(X^2 + Y^2)^{\frac{1}{2}}$ $Z_n = Z + \tan^{-1}(Y/X)$ $\lvert\tan^{-1}(Y/X)\rvert \leq 1.7433$
Linear (m=0)	$X_n = X$ $Y_n = Y + X * Z$ $\lvert Z\rvert \leq 1$	$X_n = X$ $Y_n = Z + Y/X$ $\lvert Y/X\rvert \leq 1$
Hyperbolic (m= -1)	$X_n = \frac{1}{K_{-1}}(X * \cosh Z + Y * \sinh Z)$ $Y_n = \frac{1}{K_{-1}}(Y * \cosh Z + X * \sinh Z)$ $\lvert Z\rvert \leq 1.1182$	$X_n = \frac{1}{K_{-1}}(X^2 - Y^2)^{\frac{1}{2}}$ $Y_n = Z + \tanh^{-1}(Y/X)$ $\lvert\tanh(Y/X)\rvert \leq 1.1182$

In order to better understand how CORDIC processor works, we explain the simplest form of the CORDIC algorithm with

4. CORDIC DESIGN

As the CORDIC is an iterative method, it requires many clock cycles to achieve the required accuracy. For a given precision, the increase of radix reduces the number of micro-rotations compared to radix-2.

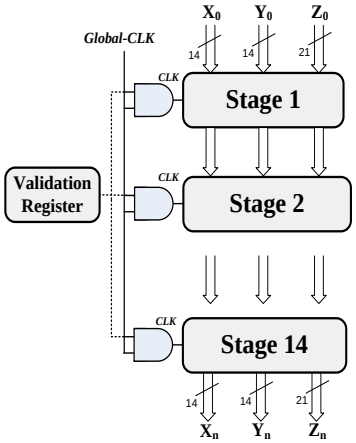

Figure 2. CORDIC schema

The CORDIC module performs 14 iterations for 14 bits precision using radix-2 number representation (Figure 3),

$K_1 = \prod_{i=0}^{14} \cos(\theta_i)$ (0, 1, 2, 3, 4 …) and $K_{-1} = \prod_{i=1}^{14} \cosh(\theta_i)$ $(1, 2, 3, 4, 4 \ldots)$[13] .

with the constraint that the (i+1)th iteration may begin only after the ith rotation has been completed.

4.1. Sinus/cosines and exponential function implementation

For sinus/cosines functions, we use m=1 and $\alpha_{m,i} = \tan^{-1}2^{-i}(i = 0,1, \ldots, 14)$ in the rotation mode.

If we affect K_1 to X_0 and 0 to Y_0 , we get cos(θ) and sin(θ) values in X_n and Y_n.

Using m = -1 and $\alpha_{m,i} = \tanh^{-1} 2^{-i}$ $(i = 1,2,3,4,4, \ldots, 13, 13,14)$, the same algorithm can calculate exponential function (cosh(θ) and sinh(θ)) affecting K_{-1} to X_0 and 0 to Y_0. Some iteration is repeated to ensure algorithm convergence. For the implementation,
we use [14]:

$$e^z = e^{Z1+pln2} = 2^p . e^{Z1} \quad (8)$$

where z = Z1 + pln2

p an integer equal to Fix(z/ln2).

For the implementation, a state machine generates signals initialization and loading of the register for each block (Figure 3).

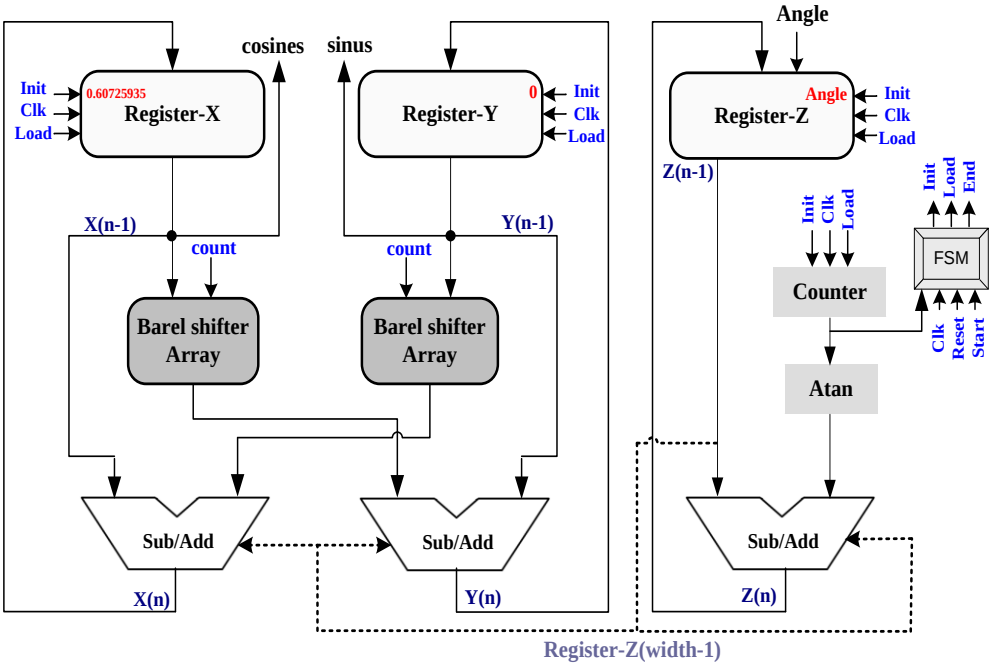

Figure 3. CORDIC iterative structure SIN/COS function

4.2. Arctangent function implementation

To obtain this function, we use the vectoring mode and circular coordinates as described in Table 1.

The implementation (Figure 4) was done using the same architecture as for the first design. But, the adder/subtracter is commanded by signed numbers of register-Y.

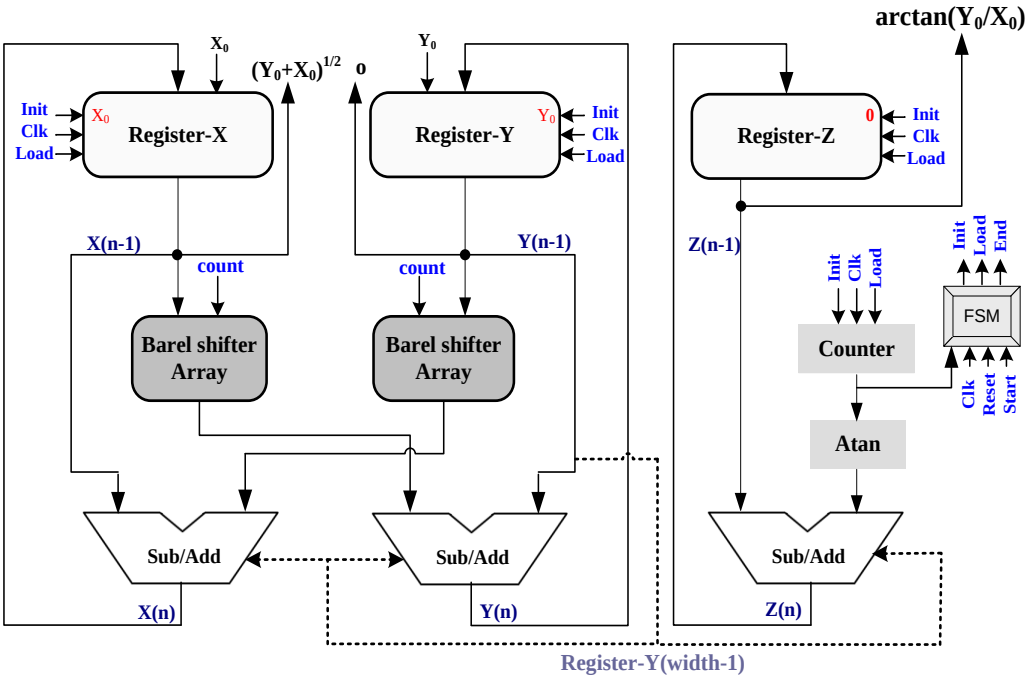

Figure 4. CORDIC iterative structure for ATAN function

5. RESULTS OF FPGA IMPLEMENTATION

The concept was implemented in VHDL with ModelSim SE 6.0 simulator from Mentor Graphics, verified and synthesized with Quartus II version 8.0 (32 bits) of ALTERA.

We use Stratix III : EP3SL150F1152C3 component. The implementation results are given in Table 2.

Table 2. The synthesis results of chosen functions

Function	Sinus & Cosines	Arctangent	Exponential
Combinational ALUT	153 /113600(<1%)	216(<1%)	178(<1%)
Logic registers	61/113600(<1%)	90(<1%)	71(<1%)
Pins	53/733(7%)	76(9%)	68(9%)
Latency	78 ns	108 ns	168 ns

Max frequency is 250 Mhz and we obtain 1% area occupation of FPGA. Latency is different for such function because it's not the same mode and not the same number of iteration. It depends on the clock frequency.

6. PRECISION WITH CORDIC METHOD AND ERROR ANALYSIS

In this section, we will conduct simulations to show the effectiveness of the proposed architecture. To analyze the error performance, we define the error as the distance between the ideal rotated point and the feasible rotated point divided by the ideal rotated point. The error is thus determined by:

$$\varepsilon = \left| \frac{\text{fonction}_{pc} - \text{fonction}_{Cordic}}{\text{fonction}_{pc}} \right| \quad (9)$$

In the design flow, one important step is the fixed-point simulation on which we assist to determine the required word-length. If the word-length is over-determined, we will suffer from higher cost and slower computational speed.
So, we will explore the 14-point format for the data and we will fix the scaling factors. The following relative error curves present the CORDIC precision after the extraction of the values from ModelSim simulation, which are generated from the test bench.

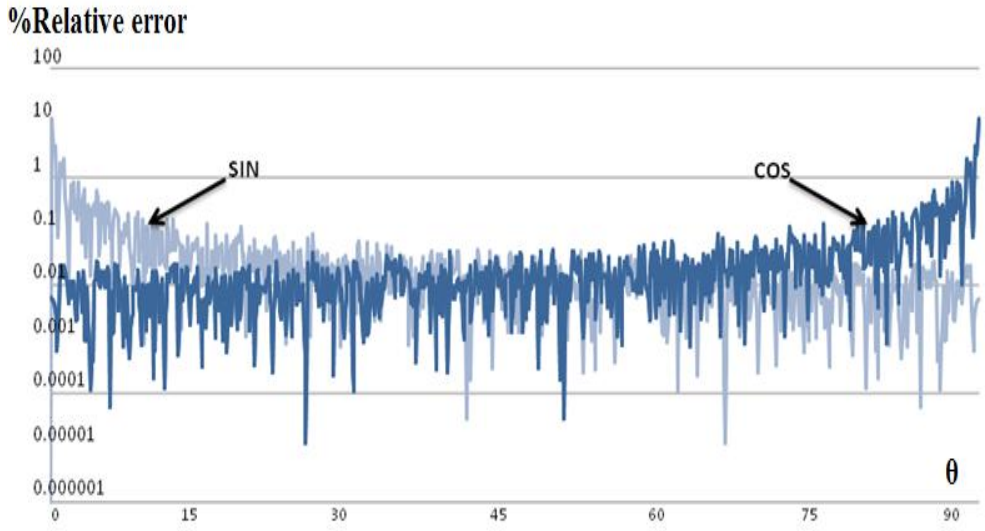

Figure 5. Relative error of CORDIC sinus/cosines functions

For $\theta \in [0°, 90°]$, the error for sinus /cosines ranges within 0.001 % to 1 % and the mean relative error is 0.013 %.

We notice that we have peaks in $K * \frac{\pi}{2}$ where K is an integer.

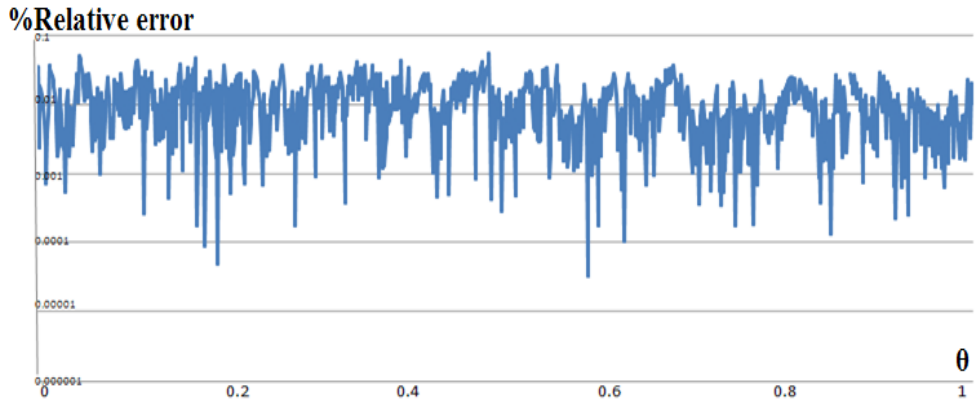

Figure 6. Relative error of CORDIC exponential function

Affecting the scale factor K_{-1} in the input (X_0) and zero in the input (Y_0) and using the rotation mode, we can calculate the cosh and sinh functions with hyperbolic coordinate. We can get the exponential value:

$$e^x = \cosh(x) + \sinh(x) \quad (10)$$

The mean error for exponential is about 0.005 %, an acceptable error in the specified convergence range.

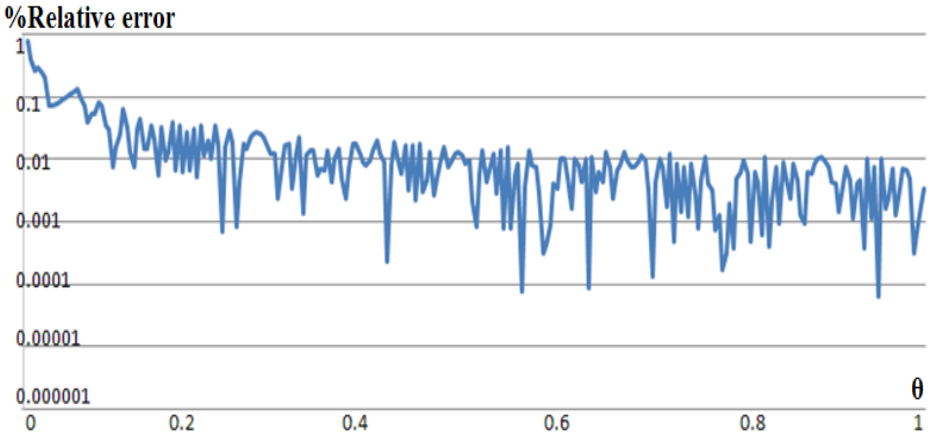

Figure 7. Relative error of CORDIC arctangent function

For the arctangent function, the mean error does not exceed 0.01%.

7. CONCLUSION

This paper proposes CORDIC architecture as an approach to implement some operators in a fingerprint recognification application. The CORDIC architecture leads to fast and small operators up to 14 bits of precision.

The principal drawbacks of this algorithm are the requirement of a scale factor and the slow rate of convergence. The convergence range can be extended over the entire coordinate space by repeating certain iteration steps and by exploiting the symmetry of the coordinate axes. To cover the whole coordinate space, we compute the angle on the interval [0, 90°]. The result of CORDIC rotations for any angle between 90° and 360° can be extrapolated for the result of a rotation corresponding to [0, 90°].Our basic CORDIC processor has been designed in VHDL implementation. The implemented architecture is dedicated to the computation of trigonometric, exponential and arctangent functions with internal wordlength of 14 bits. Nevertheless, it can be adapted to all functions by reprogramming the FPGA.

The module uses radix-2 number representation, this leads to small circuits by replacing the costly multiplications by a small number of additions. The obtained operators provide very small average error with reasonable maximum error what's makes our algorithm suitable for many applications.

8. REFERENCES

[1] Ray and Andraka, " A Survey of CORDIC Algorithms for FPGA based Computers", Andraka Consulting Group, Inc, North Kingstown, RI02852, 2011.

[2] Gualberto Aguilar, Gabriel Sánchez, Karina Toscano, Mariko Nakano-Miyatake, Héctor Pérez-Meana. "Automatic Fingerprint Recognition System Using Fast Fourier Transform and Gabor Filters", Cient'fica Vol. 12 Nœm. 1 pp. 9-16, ESIME-IPN. ISSN 1665-0654 , 2008

[3] P.K. Meher, J. Valls, T.B. Juan, K. Sridharan and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, vol. 56, no. 10, pp. 9, Sept. 2009

[4] J. E. Volder, "The CORDIC trigonometric computing technique", IRE Trans. Electronic Computers, vol. 8, no. 3, pp. 330{334, Sept. 1959

[5] J. S. Walther, "A unified algorithm for elementary functions", Proc. AFIPS spring Joint Comput. Conf., pp. 379{385, 1971

[6] M. D. Ercegovac, and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD", IEEE Trans. Comput., vol. 39, no. 6, pp. 725{740, June 1990

[7] 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

[8] 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

[9] 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

[10] H. Dawid, and H. Meyr, "The differential CORDIC algorithm:constant scale factor redundant implementation without correcting iterations", IEEE Trans. Comput. vol. 45, no. 3, pp.307-318, Mar. 1996

[11] Pongyupinpanich Surapong, Faizal Arya Samman and Manfred Glesner, "Design and Analysis of Extension-Rotation CORDIC Algorithms based on Non-Redundant Method", International Journal of Signal Processing, Image Processing and Pattern Recognition Vol. 5, No. 1, March, 2012

[12] 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

[13] Stefan Lachowicz and Hans-Jorg Pfleiderer, "Fast evaluation of the square root and other nonlinear functions in FPGA", 4th IEEE international symposium on electronic design, 2008

[14] Anis BOUDABOUS, Fahmi GHOZZI, M. Wajdi KHARRAT, Nouri MASMOUDI, "Function Generator Based CORDIC Algorithm", Third international conference on systems, signals & devices, vol. 4, March 2005