

TRADEOFFS IN ARITHMETIC ARCHITECTURES FOR CORDIC ALGORITHM DESIGN

Javier Oscar Giacomantone

CeTAD – Fac. de Ingeniería – UNLP

jog@info.unlp.edu.ar

Abstract

The COordinate Rotational DIgital Computer (CORDIC) is an arithmetic algorithm widely used in the computing of elementary functions and digital signal processing applications, particularly where large amounts of rotation operations are necessary. After a survey of the principal architectures for the design of a CORDIC processor, a pipeline scheme is presented to achieve performance enhancement at an architectural level. Circuit complexity constrains and performance limits for each option are provided. The design is geared to programmable devices but the architectural level design and the analysis of constrains are independent of the specific target.

Resumen

El algoritmo CORDIC (COordinate Rotational DIgital Computer) permite el cálculo de funciones elementales y de numerosos problemas en procesamiento digital de señales, en particular todos aquellos que involucren gran número de rotaciones. El siguiente trabajo presenta una revisión del algoritmo y las principales arquitecturas que permiten implementarlo, como así mismo un diseño que permite obtener alta velocidad de procesamiento. Se presenta un análisis del compromiso que existe entre una arquitectura de alto rendimiento y los recursos lógicos disponibles. Si bien el diseño propuesto está orientado a su implementación en un dispositivo programable, las arquitecturas propuestas poseen características generales independientes de la implementación final.

I. Introduction

Since 1959, when Jack Volder [1] introduced the CORDIC algorithm many papers have been published both about it's applications and it's analysis. A broad field of digital signal processing applications find a valuable kernel in the generalised CORDIC algorithm proposed by Walter in 1971 [2].

The optimisation of a CORDIC processor is obtained at three interrelated levels and with a hierarchical approach, the algorithm level, the architectural level and the circuit level. Obtaining a balance between the different aspects of the problem to be dealt with, available resources, speed of the circuit, accuracy, etc enables a better solution to be achieved for specific applications.

The following sections aim is to provide the architectural level options for the implementation of a CORDIC processor as well as identifying the hardware constrains versus circuit performance.

This paper is organised as follows: Section II reviews the CORDIC algorithm. Section III describes the general architecture options. Section IV presents a pipeline solution and finally section V gives the conclusions based on implementations in programmable devices.

II. The CORDIC Algorithm

The Coordinate Rotational Digital Computer (CORDIC) was introduced in 1959 by Jack Volder [1].

The CORDIC arithmetic technique makes it possible to perform two dimension rotations using simple hardware components. Its use was originally intended for solving navigation problems, by calculating some elementary functions [5] and as a useful kernel for many digital signal processing tasks [8], [3].

In 1971, J.Walter [2] proposed the generalization of Volder's algorithm in order to extend the number of elementary functions that could be solved.

All the evaluation procedures in CORDIC are computed as a rotation of a vector in three different coordinates systems with an iterative unified formulation.

$$R = [x^2 + my^2]^{1/2} \quad (1)$$

$$\alpha_{m,i} = \frac{1}{\sqrt{m}} \tan^{-1} [\sqrt{m} 2^{-S(m,i)}] \quad (2)$$

The parameter m defines the coordinate system as:

$m=1$	circular coordinate system
$m=-1$	hyperbolic coordinate system
$m \rightarrow 0$	linear coordinate system

R is the m norm.

$\alpha_{m,i}$ [$i = 0, \dots, n-1$] is the set of angles use to represent the desire angle of rotation using the CORDIC technique.

$S(m,i)$ is called shift sequence.

The angle rotation θ is approximately the sum of $\alpha_{m,i}$ $[i = 0, \dots, n-1]$ which are the partial step angles.

$$\theta = \sum_{i=0}^{n-1} c_i \alpha_{m,i} \quad (3)$$

$$c_i \in \{-1, 1\}$$

The set of $[c_i \quad i=0 \quad \text{to} \quad n-1]$ determine the direction of each step.

The unified CORDIC algorithm can be written as:

1) Read $x(0), y(0), z(0)$

2) For $i = 0$ to $n-1$ the CORDIC iteration equation gives the new coordinates after each step.

$$x_{i+1} = x_i - m c_i 2^{-S(m,i)} \quad (4)$$

$$y_{i+1} = y_i + c_i 2^{-S(m,i)} \quad (5)$$

3) The angle must be updated, giving

$$z_{i+1} = z_i - c_i \alpha_{m,i} \quad (6)$$

In CORDIC there are two fundamental modes of operation, rotation and vectoring modes.

$$c_i = \text{sign of } z(i) \quad \text{rotation mode}$$

$$c_i = -\text{sign of } x(i)y(i) \quad \text{vectoring mode}$$

In the rotation mode, the data that is used are the vector coordinates components and the desired angle of rotation. After n iterations the algorithm approximates the final coordinates of the rotated vector.

In the vectoring mode the coordinate components after rotation are given and the algorithm calculates the angle of rotation.

Actually the above procedure identified a pseudorotation rather than a rotation, which meant that the m norm is not constant after each step of the algorithm. In order to maintain the m norm constant, a scaling operation is needed

$$k_{m,i} = (1 + m c_i^2 2^{-2S(m,i)})^{1/2}$$

Since the rotation directions are restricted to $[-1, 1]$ the scale factors do not depend on c_i and a global scaling factor can be applied.

$$K_m = \prod_{i=0}^{n-1} (1 + m 2^{-2S(m,i)})^{1/2}$$

The shift sequence $S(m, i)$ affects the convergence and scaling factor. The adequate selection of the shift sequence, the convergence and analysis of the accuracy, the approximation error and the rounding error have been done[2][4]

The angle approximation error due to the finite set of angles $\{\alpha_{m,i}, i=0, n-1\}$ is :

$$\varepsilon = \theta - \sum_{i=0}^{n-1} c_i \alpha_{m,i}$$

It is desired that for any given rotation

$$|\varepsilon| \leq \alpha_{m,n-1}$$

In order to satisfy the above condition a partial angle in iteration i must be compensated except for the error by all the following partial angles

$$\alpha_{m,i} \leq \sum_{j=i+1}^{n-1} \alpha_{m,j} + \alpha_{m,n-1}$$

According to the above criteria the maximum angle is:

$$|\theta| \leq \sum_{i=0}^{n-1} \alpha_{m,i} + \alpha_{m,n-1} \equiv \text{max. angle}$$

The following values for the shift sequence [2] provide convergence to the algorithm:

m	S(m,i)	Max. angle
1	0,1,2,3,4,5,...,i,..	1.743287
0	1,2,3,4,5,6,...,i+1,..	1.000000
-1	1,2,3,4,4,5,...,12,13,13,14..	1.118173

The shift sequences $S(m,i)$ are different for m equal 1,0 or -1 as they must satisfy the above convergence criteria. In particular in the hyperbolic case it is necessary to repeat certain values of the sequence.

Different solutions were proposed to extend the angle range of convergence [1],[9]. Section IV presents an architecture which uses Volder's solution, which involves a simple rotation of 90 degrees before the pseudorotations steps of the algorithm begin.

III. CORDIC Architectures

There are multiple hardware structures that can be used to implement a CORDIC processor. The interaction between the three most important structures, iterative, serial iterative and unrolled (parallel implementation), as well as the basic arithmetic circuits to implement them have already been classified [7].

The iterative structures are based on the direct mapping of the equations (4), (5) and (6). The bit parallel design uses word wide data path and parallel adders/subtractors, shifters and an adequate state machine to control the operation of the system. Bit serial arithmetic can be used to achieve a more compact design, where serial

adder/subtractors and serial shift registers are required. Shift registers with initial parallel load could be used, and the more compact design facilitates a higher clock rate in FPGA architectures.

A completely different approach for the CORDIC arithmetic design is required, which is independent of the platform to be implemented. It consists of hardware structures which calculate each step of the algorithm. An unroll scheme has several advantages:

- It is not necessary a state machine
- The combinatorial approach require just simple logic to control the adder/subtractor
- The angle constant can be hardwire instead of a memory
- The shifters are fixed.
- High Performance is achieved by inserting registers between each CORDIC stage.

How many iterations can be done in a single pipeline stage defines the granularity of the pipeline from granularity N (number of stages) to 1 the fully pipeline version.

A survey of the different approaches to merge two CORDIC iterations, and the influence on the error that is introduced have been done[10].

The granularity of the pipeline is an important parameter to be considered if area resources are limited and high throughput required.

To obtain performance enhancement at an architectural level a detail analysis of the principal structures that constituted the above design must be considered. While many variations where proposed the emphasis here is on the adders: ripple carry adders, carry save adders, carry look ahead adders, whilst conditional sum adders and array look ahead adders are valid options to fit the design to the available resources with a reasonable performance.

Research efforts have been done [11],[12] to reduce the computation overhead due to the scaling factor multiplication operation. An important decision has to be taken between a very fast multiplier or a simple solution based in the same CORDIC kernel to solve the scaling factor multiplication, which although this is not an ideal solution for the multiplication, it does save considerable hardware resources in the final implementation.

A design for a programmable device can impose severe restrictions. It is important to realise that consideration must be given for the trade-off factors associated with the area consumption, accuracy and throughput must be considered.

A survey of algorithms for FPGA have been carried out [6], where the principal features of each structure has been presented.

The following section presents a particular design based on the unroll architecture. It gives a description of the different blocks and demonstrates an application of the important design strategies that are found at an architectural level.

IV. Pipeline fast Design

The following CORDIC module deals with the performance of the algorithm in both modes of operation and in the three coordinate systems.

A simple unit is used called the CORDIC element (Fig.1). The main components are three adders/subtractors, two shift registers and the necessary combinatorial hardware to control the adders and select the operation mode.

The C.E. element is the kernel of the CORDIC processor and its primary function is to perform eq.(4) and eq.(5).

It is clear from eq.(4), (5) that it can be implemented basically with adders and appropriate shifters but without using multipliers.

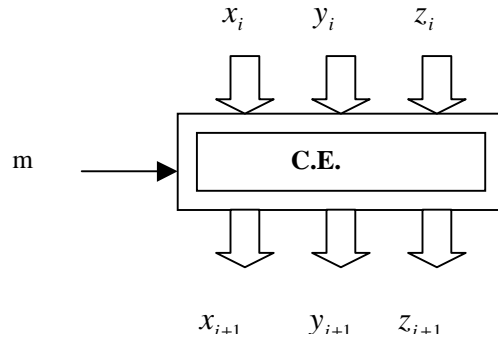


Fig.1

The basic operation that the CORDIC element performs is not more than a crossaddition [1], to sum or subtract a shifted value of x_i to y_i to obtain y_{i+1} or a shifted value of y_i to x_i to obtain x_{i+1} (Fig.2).

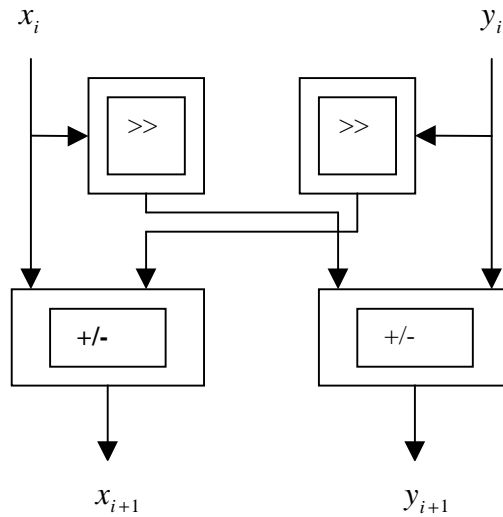


Fig.2

A parallel pipelined structure is used to implement the system consisting in an array of C.E., each of them performing a computation in parallel with the other and separated by registers to form a pipeline structure (Fig.3).

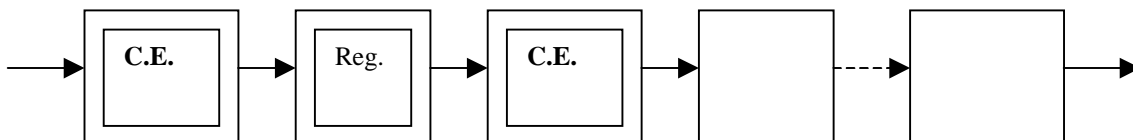


Fig.3

Any CORDIC processor should contain a proper module to perform four basic functions:

- 1) The basic iterations of eq.(4) and (5)
- 2) The scaling of the vector module
- 3) The angle update iteration
- 4) The storage of the arc tangent radix constants ATR

Fig.4 shows the whole design which basically consist of four modules, the pseudorotation (P) and the scaling factor (S) blocks are very fast parallel, pipeline structures, of which the first implements the basic iterations and the angle updating, while the second fits the vector to the correct trajectory.

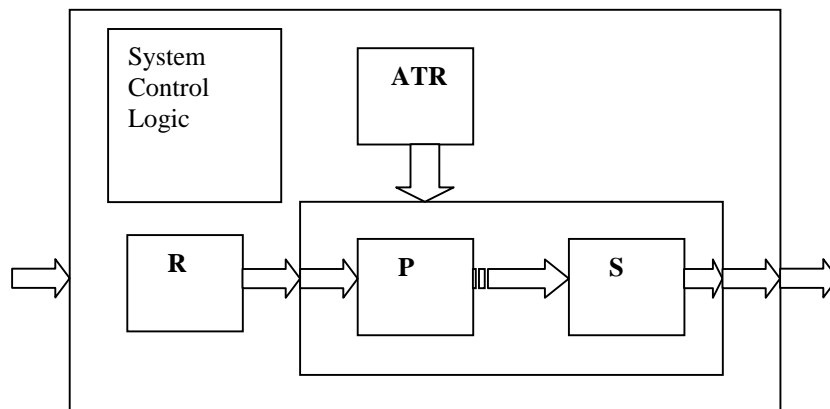


Fig.4

The ATR block only contains the angle steps for each iteration. The rotation module (R) performs a real rotation of 90 or -90 degrees, in order to extend the angle range.

The design behaviour was first simulated in C/C++ by modelling hardware accuracy constrains. Active VHDL simulator [14] was used to describe and test the above design, based on the results obtained with the C/C++ constrain simulation. The design realisation was done using XILINX Foundation Express tools [13]. The pipeline architecture was implemented in two different devices the XC4010XL-09 and XC4062XL_09, from 12 to 16 bit word length.

V Conclusions

Determining the selection of a proper architecture for a CORDIC processor is a complex task that requires the analysis of different design strategies and the evaluation of the principal parameters, speed of the implementation, area consumption, and accuracy, in order to obtain the best design for an specific application.

The main architecture proposed in this paper adopts a pipeline strategy to increase the performance at the architectural level. The separate distribution of the ATR constants to each adder permits a hardware solution instead of using a specific ROM and the shifters need not to be made programmable, which is a beneficial asset. Nevertheless, a trade-off between area and accuracy needs to be made, so that the utility of the design incorporates the considerations required of accuracy bits needed for each arithmetic task and the hardware resources available.

The granularity of the pipeline can be varied to increase the throughput but the cost is the increase of circuit complexity, the fundamental block with more influence on the system is the adder. The ripple carry adders provide the slowest solution and the conditional sum adders the fastest one but more area consuming.

Acknowledgement

The authors are grateful to Griselda Lyn for providing comments and suggestions that greatly improved this paper.

References

[1] Volder, J., "The CORDIC Trigonometric Computing Technique", IRE Trans. Electronic Computing, Vol. EC-8, Sept 1959, pp. 330-334.

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

[3] Hu, Y. H. and Sung, T. Y. "Efficient Implementation of the Chirp Z-Transform using a CORDIC processor", IEEE Trans. ASSP, Vol. 38, No. 2, Feb. 1990, pp. 352-354.

[4] Hu, Y. H., "The quantization effects of the CORDIC algorithm", IEEE Trans. on Signal Processing, Vol. 40, No. 4, April 1992, pp. 834-844.

[5] Linhardt, R. J. and Miller, H. S., "Digit-by-Digit Transcendental-Function Computation", RCA, Rev. 30 (1969), pp. 209-247.

[6] Andraka, R. "A survey of CORDIC Algorithms for FPGA Based Computers", Proc. of ACM/SIGDA Sixth International Symposium on FPGAs, Feb. 1998, Monterrey, CA, pp. 191-200.

[7] Vladimirova, T. and Tiggler, H. "FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm", Proc. of Military and Aerospace Application of Programmable Devices and Technologies Conference (MAPLD 99), Sep. 1999, Laurel, MA, A-2, pp. 28-30.

[8] J. Giacomantone, H. Villagarcia, O. Bria. "A fast CORDIC Co-Processor Architecture for Digital Signal Processing Applications". VI Congreso Argentino de Ciencias de la Computación (VI CACIC). October 2000

[9] Hu, X., R.G Harber and S. C Bass, "Expanding the range of convergence of the CORDIC algorithm", IEEE Trans on Computers, vol 40, No1, pp13-21, Jan, 1991.

[10] S.Wang and E. Swartzlander, Jr., "Merged CORDIC algorithm," Proc. IEEE ISCAS 95, pp1988-1991.

[11] Delsome, J. M, "CORDIC algorithms: theory and extensions", Proc. SPIE, Advance Algorithms and Architectures for Signal Processing IV, 1989.

[12] E.F. Deprettere, P. Dewilde, R. Udo, " Pipelined CORDIC architectures for fast VLSI filtering and array processing" , Proc. ICASSP 1984.

[13] www.xilinx.com

[14] www.aldec.com