Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves Generator

Esteban O. Garcia, Rene Cumplido, Miguel Arias
Department of Computer Science, INAOE. Puebla, Mexico
Phone +52 222 2663100 Fax +52 222 2663152 E-mail:{eomargr,rcumplido,ariasm}@inaoep.mx

Abstract — **Sine and Cosine waves have been used in countless applications; in recent research on Software Defined Radio (SDR), digital modalities of sine and cosine waves have received special attention. SDR involves highly reconfigurable resources and uses digital generated waves for modulation and demodulation of signals. *Coordinate Rotation Digital Computer* (CORDIC) is a well known algorithm used to approximate iteratively some transcendental functions. In this work, a pipelined CORDIC architecture is used for designing a flexible and scalable digital sine and cosine waves generator. An FPGA-Based architecture is presented and the design has been implemented on a Xilinx Spartan 3 device. Synthesis and implementation results are shown and discussed.**

Keywords — **CORDIC, FPGA, SDR.**

I. Introduction

There are plenty of applications which require digital wave generators. Wireless and mobile systems are among the fastest growing application areas; in particular, Software Defined Radio (SDR) is currently a focus of research and development. An SDR system allows performing many functions based on a single hardware platform, thus highly reconfigurable resources for signal processing are needed, mainly for modulation and demodulation of digital signals. Fields of development are increasing every day with applications such as cell phones or military communications.

There are several ways to generate digital sine and cosine waves, all of them based on trigonometric functions. The use of previously calculated tables is one of the choices, but it requires excessive memory usage when good quantization level is needed. CORDIC algorithm on the other hand, offers an excellent alternative, and its best characteristic is flexibility. It allows having as much quantization accuracy as permitted by word length. So it is useful to generate low frequencies without losing definition.

CORDIC algorithm, also known as Volder algorithm, had its first application in pocket calculators but nowadays is used to design coprocessors, build Discrete Fourier and Cosine transforms, among other mathematical processes [1-6]. For its electronic implementation, CORDIC offers the favorable condition of not requiring either multiplication or division blocks, instead of that, it works only with adder-subtractor and shifter.

Reconfigurable computing techniques combine software flexibility with hardware performance. System designers may use FPGA technologies in different ways. In this work, FPGA is used for quick prototyping and testing. CORDIC characteristics and FPGA flexibility are used to implement CORDIC in pipeline, so iterative cycles in CORDIC are avoided.

Present work is divided as follows: Section II introduces CORDIC algorithm; Section III presents architecture and details of the pipeline and angles sequencer; Section IV is dedicated to results and at Section V conclusions are discussed.

II. Cordic algorithm

There exist two modalities of CORDIC algorithm, vectoring and rotation mode. In vectoring mode, coordinates (x_0,y_0) are rotated until y_0 converges to zero. In rotation mode, initial vector (x_0,y_0) starts aligned with the x axis and is rotated by an angle of θ_i every cycle, so after n iterations, θ_n is the obtained angle. For this work, rotation mode is used to approximate sine and cosine functions.

The way CORDIC works is related to trigonometric functions properties. The main idea consists in taking a unary vector and applying successive rotations, called microrotations, until the desired angle is reached. The rotating vector is unary, so after n iterations it will contain $\sin\theta_n$ and $\cos\theta_n$ in its second and first components respectively. In the following paragraphs it will be shown that calculus can be simplified if the starting vector is approximated to a constant value K.

Starting vector is defined as $v_0=(x_0,y_0)$. In order to make the vector rotation, a linear transform, which can be described by a matrix which is multiplied by a vector, is used. After n iterations, the vector (x_n,y_n) is

$$x_n = x_0 \cos\theta - y_0 \sin\theta \quad (1)$$
$$y_n = y_0 \cos\theta + x_0 \sin\theta \quad (2)$$

In each iteration i, the vector performs a microrotation by θ_i, so the new vector is calculated with a similar function:

$$x_{i+1} = x_i \cos\theta_{i+1} - y_i \sin\theta_{i+1} \quad (3)$$
$$y_{i+1} = y_i \cos\theta_{i+1} + x_i \sin\theta_{i+1} \quad (4)$$

When the term cos θ_{i+1} is factorized, components in the vector are described by

$$x_{i+1} = \cos\theta_{i+1} (x_i - y_i \tan \theta_{i+1}) \quad (5)$$
$$y_{i+1} = \cos\theta_{i+1} (y_i + x_i \tan \theta_{i+1}) \quad (6)$$

tan θ_{i+1} is restricted to $\pm 2^{-i}$, so multiplication is converted in an arithmetic right shift. It is also useful to use the identity cos θ_{i+1} = cos (arctan 2^{-i}) to define the next variables

$$K_i = \cos (\arctan 2^{-i}) = 1/\sqrt{(1+2^{-2i})} \quad (7)$$
$$d_i = \pm 1 \quad (8)$$

Cosine is an even function, therefore cos (α) = cos (-α). So (5) and (6) can be transformed into

$$x_{i+1} = K_i (x_i - y_i d_i 2^{-i}) \quad (9)$$
$$y_{i+1} = K_i (y_i + x_i d_i 2^{-i}) \quad (10)$$

Multiplication by K_i is avoided by considering it as a gain factor for all iterations. If n iterations are performed, then K is defined as the multiplication of every K_i .

$$K = \prod K_i = \prod 1/\sqrt{(1+2^{-2i})} \quad (11)$$

K_i is retired from (9) and (10), then it is considered at the starting vector, which must be initialized as $v_0 = (K, 0)$. At the end of n rotations, the length of vector will be 1, so its components will contain cosine and sine values of the desired angle (Fig. 1). As the vector is initialized with constant K, the vector components for each iteration are simplified to

$$x_{i+1} = x_i - y_i d_i 2^{-i} \quad (12)$$
$$y_{i+1} = y_i + x_i d_i 2^{-i} \quad (13)$$

On each iteration it is necessary to decide whether $d_i=1$ or $d_i=-1$. In order to make that decision, the difference between the desired angle and the current angle is used. So a new variable known as accumulator is defined as

$$z_{i+1} = z_i - d_i \arctan 2^{-i} \quad (14)$$

The value of z_0 is the angle for which sine and cosine are to be calculated. To know whether d_i should be positive or negative, the following rule is used

$$di = \begin{cases} -1 & \sin z_i < 0 \\ +1 & \sin z_i \geq 0 \end{cases} \quad (15)$$

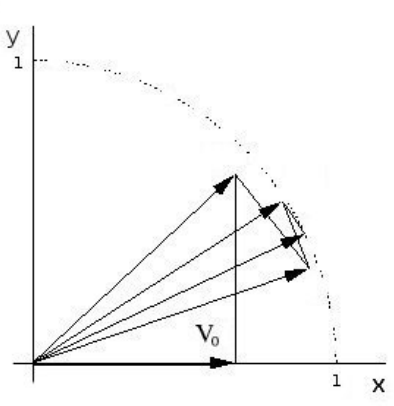

Fig. 1. Microrotations with K compensation.

The sum of the rotating angles gives the desired angle

$$\theta_n = \sum d_i \arctan 2^{-i} \quad (16)$$

Because the first tangent value is 2^0=1, it is possible to rotate angles only in the range [-π/2,π/2], which is the convergence range. Arctan 2^{-i} can be calculated a priori to avoid the implementation of a arctan function on the FPGA.

III. ARCHITECTURE

As it has been mentioned, a CORDIC implementation does not need to have either sine or cosine values on memory, instead, it approximates the results on each step of the iterative process. The proposed architecture may be extended for a specific resolution without a considerable increase in the number of components used.

Iterative CORDIC implementations take more than a clock cycle for each output value. It is useful when no successive calculi are needed; such is the case of calculators. In this work, a generator needs to calculate successive values of sine and cosine. Therefore the proposed digital wave generator has a pipelined CORDIC module as nucleus. It is composed of three adders-subtractors and two arithmetic shifters (Fig. 2). Each adder-subtractor is associated to one of the variables of vector component and accumulator angle, called x, y, z. Each module has three inputs and outputs of this type, a clock signal and an input to manage *cosine inversion*, which is used to decide whether or not to correct the quadrant in cosine function. This is necessary due to the convergence range of CORDIC explained en section II. Details about *cosine inversion* will be given when angle sequencer is explained. In an iterative implementation of CORDIC, a step control module is necessary to adapt the shift amount to the corresponding iteration. The first iteration there is only one move shift, for the second there will be two moves and so on.

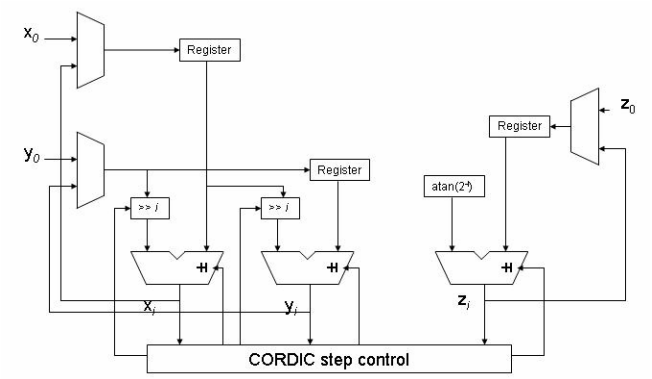

Fig. 2. Iterative CORDIC

In order to reduce the number of clock cycles, the proposed pipelined architecture uses a set of stages, each of them based on iterative CORDIC module, but shifts are fixed instead of having a step control to change shift amount.

A. Pipeline

CORDIC pipelined implementation uses the basic CORDIC module described. If an iterative implementation of CORDIC were used, the generator would take several clock cycles to build a single output sample of the wave. However, using pipeline converts iterations into pipeline phases. In this way an output is obtained at every clock cycle, after pipeline stages propagation. Each pipeline stage takes exactly one clock cycle to complete. In Fig. 3, registers are implicit between each stage.

One of the most recurrent problems for a CORDIC implementation is overflow. It is mentioned in some previous works [1],[6], but not solved. The risk is present when angles $\pi/2$ or $-\pi/2$ are reached. This is due to the fact that the difference in binary representation between these two angles is one bit. When approximation is being made, an angle could cross from a positive right angle to a negative one. Overflow is solved by adding an overflow control which checks for the signs of the operands involved in additions or substraction and the result of the operation. When an overflow is produced, the result keeps its last sign; this method does not affect the final result. For each stage, the value arctan 2^{-i} is taken from memory. In the overflow control the sign of z_i is used to determine d_i, so for each stage d_i is applied as input to adders-subtractors to decide if a sum or a subtraction is performed.

B. Angle Sequencer

CORDIC is a module which calculates sine and cosine given an angle, so it is necessary to give it a sequence of angles in the range of CORDIC convergence. It is not a simple sequence of angles, but it must be able to change frequency of the signal created by CORDIC. The module created to accomplish this function has been called *angle sequencer*. Given than CORDIC only converges within the first and fourth quadrant, it was needed a sequencer to generate angles going from *0* to $\pi/2$, then from $\pi/2$ to $-\pi/2$ and finally from $\pi/2$ to *0*. To solve this, a sawtooth wave generator was build. It starts at zero and increases its output value until 2^n, where *n* is the angle depth in bits. As 2's complement logic is used, once 2^n is reached the signal goes from the top positive value to the minimum negative value, then it increases again and repeats the cycle.

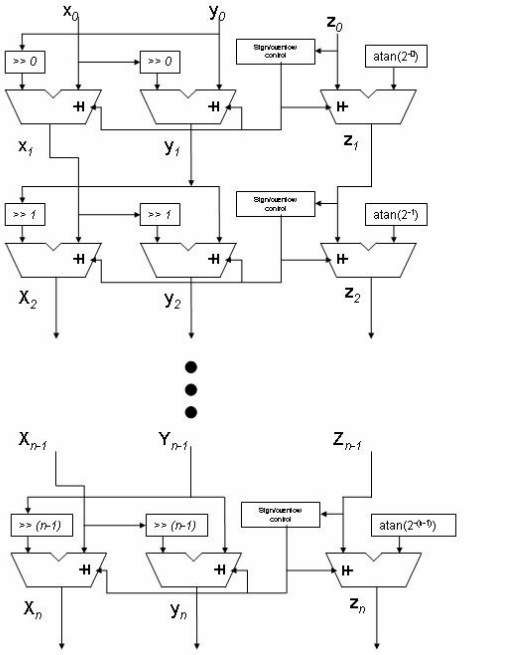

Fig. 3. Pipelined CORDIC

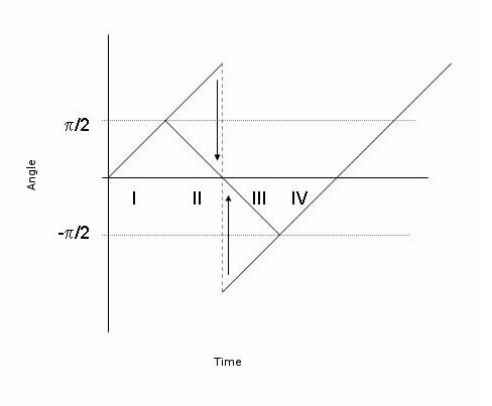

Fig. 4. Angles Sequencer signal

Sawtooth wave is not useful by itself, it needs to be transformed into a triangle wave to feed the CORDIC module. Triangle must represent the travel made on first and fourth quadrants. Sawtooth wave is transformed into triangle

wave by looking at two most significant bits because they indicate the current quadrant. When the two most significant bits are *01* or *10*, then the sawtooth signal must be transformed. That is done by mean of inverting every single bit, without changing the sign bit (Fig. 4). This scheme had been previously explored by Norbert Lindlbauer [5] in a sound synthesizer, but we have reduced one bit from the sawtooth wave and applied quadrant correction on the fly. The speed of the sawtooth wave determines the frequency of the sine and cosine waves. In fact, sine and cosine waves will have the same frequency than the triangle wave.

Let us denote with t the time in seconds which takes a cycle in the reference clock. CORDIC algorithm is able to perform a maximum of $1/t$ Hz. As stated by Nyquist, the higher frequency for the output is $(2t)^{-1}$ Hz. To establish a relation between the increase value in sawtooth wave and output frequency, the following formula is used, having n as depth in bits. Let w be the increase value for the sawtooth wave; f_r the reference clock frequency and f_o the frequency of output waves. w is expressed in terms of f_r and f_o in the following way:

$$w = (2^{n+1} f_r) / f_o \quad (17)$$

IV. Results

All modules were written in VHDL and tested as components, which were simulated on ModelSim. All critical cases, such as high and low frequencies responses, were tested. This was accomplished by testing the whole angle sequencer's dynamic range. The full design was targeted to a Xilinx Spartan 3 xc3s200-5ft256, using Xilinx ISE 7.1 to synthesize and implement the architecture. Speed and area optimization were tested. Tables 1 and 2 show the comparative results.

Output values were tested using a *black box* component in Xilinx System Generator to include VHDL design. JTAG library was built in order to perform hardware co-simulation and plot sine and cosine waves using Simulink. When synthesis was optimized for speed, the maximum frequency of operation was estimated in 154.9MHz.

V. Conclusion

It has been presented a pipelined CORDIC-based architecture for sine and cosine waves generator targeted to support modulation and demodulation in SDR. Compared with other techniques, CORDIC has shown to have benefits when applied to SDR. The main one is that CORDIC makes possible creating high accuracy waves, even for low frequencies. In this work CORDIC has been implemented in pipeline to avoid iterative cycles, which means that a sample output is presented on each clock cycle.

TABLE I
SPEED OPTIMIZATION RESULTS

Parameter	Used	%
Number of Slices	1104	57
Number of Flip Flops	615	16
Numer of 4 input LUTs	1748	45
Number of bonded IOBs	43	24
Number of GCLKs	1	12
Maximum Frequency		154.69MHz

TABLE II
AREA OPTIMIZATION RESULTS

Parameter	Used	%
Number of Slices	1075	55
Number of Flip Flops	570	14
Numer of 4 input LUTs	1737	44
Number of bonded IOBs	43	24
Number of GCLKs	1	12
Maximum Frequency		124.67MHz

Overflow and quadrant correction are CORDIC inherent issues which had not been detailed in other related work. A solution for them was presented and the implementation results on a FPGA Xilinx Spartan 3 were presented and discussed.

Aknowledgment

This work has been partially supported by CONACYT, under grant number 201562.

References

[1] 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, NY, USA, 1998. ACM Press.

[2] Javier Valls, Martin Kuhlmann, and Keshar K. Parhi. "Evaluation of CORDIC algorithms for FPGA design". *Journal of VLSI Signal Processing.* vol. 32, no. 3, pp. 207-222, 2002.

[3] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer-Verlag, New York, Inc., Secaucus, NJ, USA, pp. 70-75, 200.

[4] Eckhard Grass, Bodhisatya Sarker, and Koushik Maharatna. "A dual-mode synchronous/asynchronous CORDIC processor". *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems*, vol. 0, no. 76, 2002.

[5] Bar-Giora Goldberg. *Digital Frequency Synthesis Digital Frequency Synthesis Demystified*. LLH Technical Publishing, 1999.

[6] Norbert Lindlbauer. *Applications of FPGA's to Musical Gesture Communication and Processing*. Master's thesis, CNMAT, University of California, Berkeley, 1999.

[7] Jeffrey H. Reed. *Software Radio: A modern approach to Radio Engineering*. Prentice-Hall, New Jersey, 2002.