Comparison of Parallel and Pipelined CORDIC algorithm using RCA and CSA

Diego Barragán Guerrero
FEEC - UNICAMP
Campinas, São Paulo, Brazil, 13083-852
+5519 9308-9952
diego@decom.fee.unicamp.br

Luís Geraldo P. Meloni
FEEC - UNICAMP
Campinas, São Paulo, Brazil, 13083-852
+5519 9778-1523
meloni@decom.fee.unicamp.br

***Abstract*— This paper presents an implementation of the CORDIC algorithm in digital hardware using two types of algebraic adders: Ripple-Carry Adder (RCA) and Carry-Select Adder (CSA), both in parallel and pipelined architectures. Analysis of time performance and resources utilization was carried out by changing the algorithm number of iterations. These results demonstrate the efficiency in operating frequency of the pipelined architecture with respect to the parallel architecture. Also it is shown that the use of CSA reduce the timing processing without significantly increasing the slice use. The code was synthesized using FPGA development tools for the Xilinx Spartan-3E xc3s500e family.**

***Index Terms*— CORDIC, pipelined, parallel, RCA, CSA, trigonometrics functions.**

I. Introduction

In Digital Signal Processing with FPGA, trigonometric functions are used in many signal algorithms, for instance synchronization and equalization [12]. As a first approach, we can use Taylor series to approximate these functions, then the problem is to cut down into a series of multiplication and addition operations, but the program is complex and the consumption of resources is high, which is not very convenient. A more effective method to solve this problem is based on Coordinate Rotation Digital Computer (CORDIC). The CORDIC algorithm provides an iterative method for performing vector rotations by arbitrary angles using only shifts and adds [13]. CORDIC based VLSI architectures are very attractive alternatives to the architectures based on conventional multiply-and-add hardware for an extensive variety of DSP algorithms.

In this article, we present a FPGA implementation of CORDIC algorithm employing two class of adders (RCA and CSA) and two types of architectures (parallel unrolled and pipelined). By taking use of EDA Xilinx tools and hardware description language VHDL, the algorithms were implemented and verified.

II. Cordic algorithm principle

CORDIC is a versatile algorithm to compute a wide range of operations including logarithmic, hyperbolic, linear, and trigonometric functions [3]. The CORDIC algorithm provides an iterative method for performing vector rotations or a vector translation by arbitrary angles using only shifts and adds. The algorithm has two modes of operation: the rotational mode (RM) where the vector (x_i, y_i) is rotated by an angle θ to obtain a new vector (x_N, y_N), and the vectoring mode (VM) in which the algorithm computes the modulus R and phase α from the x-axis of the vector (x_0, y_0). The basic principle of the algorithm is shown in Figure 1.

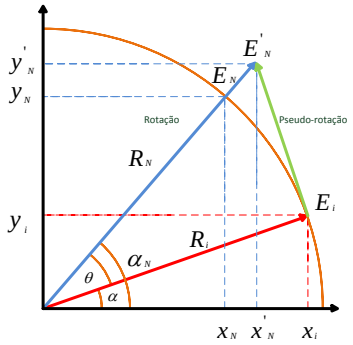

Fig. 1. CORDIC: RM and VM.

The CORDIC algorithm, executed by a finite number of N micro-rotations indexed by i = 0: N-1, was originally described for a circular coordinate system [9], then the algorithm was extended to linear and hyperbolic systems and described briefly in the following set of equations [14].

$$x_{i+1} = x_i - m\, d_i\, 2^{-i}\, y_i \tag{1}$$

$$y_{i+1} = y_i + d_i\, 2^{-i}\, x_i \tag{2}$$

$$z_{i+1} = z_i - d_i\, \alpha_i \tag{3}$$

$$\alpha_i = \begin{cases} 2^{-i}, \, Linear \\ \tan^{-1}(2^{-i}), \, Circular \end{cases} \tag{4}$$

$$d_i = \begin{cases} -sign(y_i), \, for\ VM \\ sign(z_i), \, for\ RM \end{cases} \tag{5}$$

By choosing appropriate values for the parameters *m* and α_i, we can select the different coordinate systems. When *m* = 0, 1 or -1, and the values of α_i are $\tan^{-1}\left(2^{-i}\right)$, 2^{-i}, or $\tanh^{-1}\left(2^{-i}\right)$ the algorithm operates in linear, circular, and hyperbolic coordinate systems, respectively, which provides the following result for rotation mode.

$$x_n = A_n \left[x_0 \cos z_0 - y_0 \sin z_0 \right] \tag{6}$$

$$y_n = A_n \left[y_0 \cos z_0 + x_0 \sin z_0 \right] \tag{7}$$

$$z_n = 0 \tag{8}$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}} \tag{9}$$

And for vectoring mode.

$$x_n = A_n \sqrt{x_0^2 + y_0^2} \tag{10}$$

$$y_n = 0 \tag{11}$$

$$z_n = z_0 + \tan^{-1} \left(\frac{y_0}{x_0} \right) \tag{12}$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}} \tag{13}$$

In both cases, the rotation algorithm has a gain of A_n that depends on the number of iterations.

III. Parallel, pipelined, RCA and CSA architectures

A. Parallel and pipelined architectures

CORDIC is an iterative algorithm that has the same components at each step of pseudo-rotation: three algebraic adders, two shifters, one inverter and a LUT containing the value of α_i.

Parallel architecture, showed in Figure 2, results in two significant simplifications. First the shifters are each a fixed shift, which means that they can be implemented by wiring. Second, the lookup values for the angle accumulator are distributed as constants to each adder in the angle accumulator chain. Those constants can be hardwired instead of using storage space. The need for registers is also eliminated, making the unrolled processor strictly combinatorial. However, in order to measure the operating frequency of the architecture, a register was added both the input and output of the circuit.

The parallel architecture is easily pipelined by inserting registers between every iteration step. In the case of most FPGA architectures there are already registers present in each logic cell, so the addition of the pipeline registers has no additional hardware cost [2]. A pipelined design conceptually works very similar to an assembly line in that the raw material or data inputs enter the front end, they pass through various stages of manipulation and processing, and then exist as a finished products or data outputs. The beauty of pipelined design is that new data can begin processing before the prior data has finished. Pipelines are used in nearly all very-high-performance devices [7]. Figure 3 show pipelined architecture.

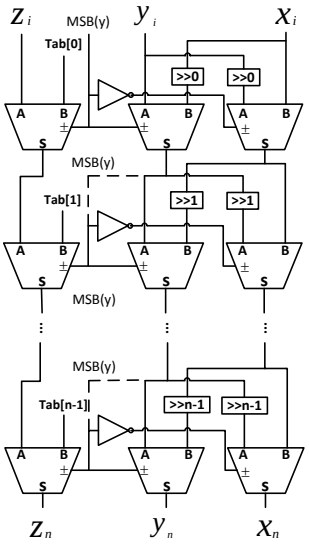

Fig. 2. Parallel CORDIC architecture.

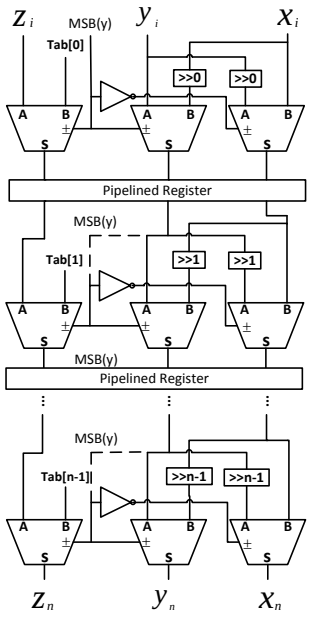

Fig. 3. Pipelined CORDIC architecture.

B. RCA and CSA architectures

Because the CORDIC algorithm needs to perform a binary sum several times throughout the process, we have implemented two types of adders in order to quantify the effect on processing time and resource consumption of the FPGA. While Ripple-Carry Adders (RCA) have the most compact design ($O(n)$ area) among all types of adders, they are the slowest types of adders ($O(n)$ time). On the other hand, Carry Look-ahead Adder (CLA) are the fastest adders ($O(\log(n))$ time), but they are the worst from the area point of view

($O(n \log(n))$ area). Carry-Select Adder (CSA) have been considered as a compromise solution between RCAs and CLAs ($O(\sqrt{n})$ time and $O(2n)$ area) because they offer a good tradeoff between the compact area of RCA and the short delay of CLA [1]. Thereby, the architectures were chosen are Ripple-Carry Adder and Carry-Select Adder [5] [11]. Figure 4 present RCA arqcitecture.

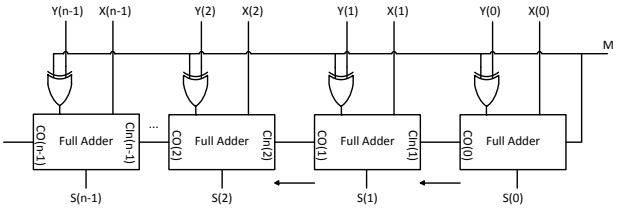

Fig. 4. Ripple Carry Adder.

The Ripple-Carry Adder is composed of a chain of full adders with length n, where n is the length of the input operands. The following boolean expressions describe the full adder.

$$p = a \oplus b \tag{14}$$

$$g = a \bullet b \tag{15}$$

Where $\oplus$ is exclusive OR and $\bullet$ represent AND operation and where a and b are the input operands and p and g are the propagate and generate signals respectively. Carry is propagated if p is high or is generated if g is high.

Thus, the sum S and carry out C_o signals can be expressed as [10]:

$$S = a \oplus b \oplus C_i = p \oplus C_i \tag{16}$$

$$C_o = g + p \bullet C_i \tag{17}$$

The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive OR gate with each full adder. A four bit adder subtractor circuit is shown in Figure 5. The mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor. Each exclusive OR gate receives input M and one of the inputs of Y. When M=0, we have $Y \oplus 0 = Y$. The full adders receive the value of Y, the input carry is 0, and the circuit performs X plus Y. When M=1, we have $Y \oplus 1 = Y'$ and $C_0 = 1$. The Y inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation X plus the 2ś complement of Y [8].

The problem of the Ripple-Carry Adder is that each adder has to wait for the arrival of its carry-input signal before the actual addition can start. The basic idea of the Carry-Select Adder is to use blocks of two Ripple-Carry Adders, one of which is fed with a constant 0 carry-in while the other is fed with a constant 1 carry-in. Therefore, both blocks can be calculated in parallel. When the actual carry-in signal for the block arrives, multiplexers are used to select the correct one of both precalculated partial sums. Also, the resulting carry-out is selected and propagated to the next carry-select block [6]. In other words, the Carry-Select Adder improves speed further with more hardware.

Thus, the sum bits s_i and group outgoing carry c_{i+1} signals can be expressed as.

$$s_m = s_m^0 \, \bar{c}_j + s_m^0 \, c_j; \quad m = j, \; j+1, ..., i \tag{18}$$

$$c_{i+1} = c_{i+1}^0 \, \bar{c}_j + c_{i+1}^1 \, c_j \tag{19}$$

The Figure 5 shows the CSA architecture.

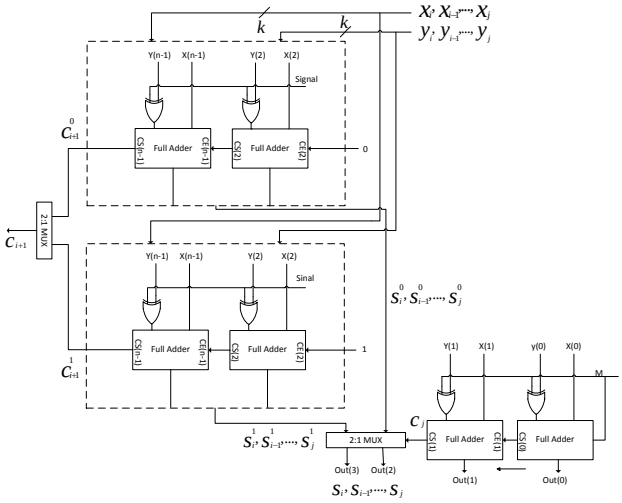

Fig. 5. Carry Select Adder.

IV. Implementation and Results

In order to verify the results, a Matlab simulation of the CORDIC algorithm was realized. All implementations were designed in VHDL using ISim simulator and ISE environment. The previous circuits have been synthesized for an Xilinx Spartan 3E device, which characteristics are presented in Table I.

TABLE I
FPGA Features used in design.

Property Name	Value
Top-Level Source Type	HDL
Family	Spartan3E
Device	xc3s500e
Package	FG320
Speed	-4
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Isim (VHDL/Verilog)
Preferred Language	VHDL
VHDL Source Analysis Standard	VHDL-93

The input vector was generated in order to cover all four quadrants and normalize to have a unit magnitude, so the fixed-point format is A(1,6) in the case of 8 bit input [15]. Since

the range of convergence of CORDIC is limited from $-\pi/2$ to $\pi/2$, every input vector is rotated by $\pm\pi/2$ (added to the input phase), moving each vector to the first and fourth quadrant in order to extend that range. In the case of CSA, block division adders was half the size of the input bits. Because final result of the algorithm is multiplying with a scaled factor K, every input is pre-divided by 1/K. The Figure 6 show the ISim output results.

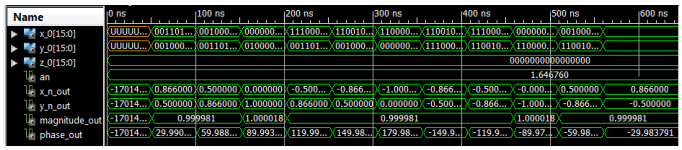

Fig. 6. Simulate Behavioral Model.

The Figure 7 show that the pipelined architecture has a smaller processing time that is approximately 12 times smaller, regardless of the adder type implemented. The use of CSA adder reduces the operation time in 13.2% using 16 iterations for the case of parallel architecture and 19.12% for pipelined architecture. However, because the CSA adder employs twice adders more than RCA, the former employs 7.7% more slices than the second adder in a parallel architecture, and 20.9% higher for the pipelined architecture with 16 iterations and 16 binary bits per word.

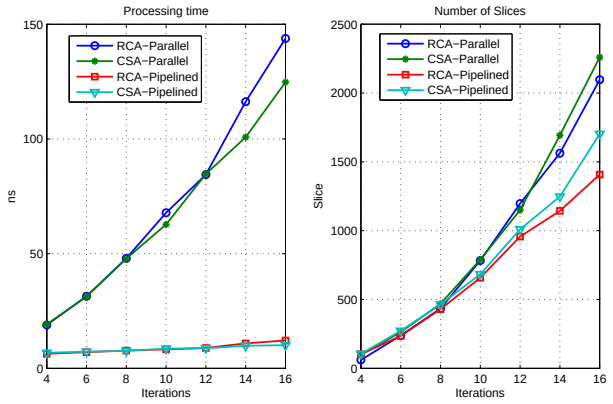

Fig. 7. Processing time and number of slices used.

V. Conclusions

The trade-off of resources/speed will determine the approach to be chose in design the CORDIC algorithm in a FPGA. Adding register layers improves timing by dividing the critical path into several paths of smaller delay. Insertion of registers between each iteration produces a faster computing results, however the outcome value is available after N clock cycles since the first data input, while the parallel architecture will prompt the result after one clock cycle. The number of slices used in both architectures does not change in the same way that the processing time, because each slice has registers that can be used to design a pipelined architecture [4]. The use of CSA improve the timing of the circuit without significantly increasing the use of FPGA slices.

References

[1] B. Amelifard, F. Fallah, and M. Pedram. Closing the gap between carry select adder and ripple carry adder: a new class of low-power high-performance adders. In *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, pages 148–152, March.

[2] Ray Andraka. A survey of cordic algorithms for fpga based computers. pages 191–200, 1998.

[3] Diego Barragán, Karlo Lenzi, and Luís Meloni. Desempenho do algoritmo paralelo cordic em implementação em fpga. *XXX Simposio Brasileiro De Telecomunicações*, 2012.

[4] Xilinx Company. Spartan-3e data sheets. Online at http://www.xilinx.com/support/documentation/spartan-3e_data_sheets.htm, March 2013.

[5] C. Ebeling. Random logic implementation. http://www.cs.washington.edu/education/courses/cse567/98au/ppt/04b-combinational/, October 2012.

[6] Norman Hendrich. Carry-select adder (8 bit). Online at http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/20-arithmetic/20-carryselect/adder_carryselect.html, Julho 2012.

[7] Steve Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Wiley-IEEE Press, 2007.

[8] M. Morris Mano. *Digital design (2. ed.)*. Prentice Hall, 1990.

[9] P.K. Meher, J. Valls, Tso-Bing Juang, K. Sridharan, and K. Maharatna. 50 years of cordic: Algorithms, architectures, and applications. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(9):1893–1907, Sept.

[10] Behrooz Parhami. *Computer arithmetic: algorithms and hardware designs*. Oxford University Press, Oxford, UK, 2000.

[11] Robert Joachim Schweers. Descripción en vhdl de arquitecturas para implementar el algoritmo cordic. http://biblioteca.universia.net/, Julho 2009.

[12] J. Valls, T. Sansaloni, A. Perez-Pascual, V. Torres, and V. Almenar. The use of cordic in software defined radios: a tutorial. *Communications Magazine, IEEE*, 44(9):46 –50, sept. 2006.

[13] Jack E. Volder. The cordic trigonometric computing technique. *Electronic Computers, IRE Transactions on*, EC-8(3):330 –334, sept. 1959.

[14] J. S. Walther. A unified algorithm for elementary functions. In *Proceedings of the May 18-20, 1971, spring joint computer conference*, AFIPS '71 (Spring), pages 379–385, New York, NY, USA, 1971. ACM.

[15] Randy Yates. Fixed-point arithmetic: An introduction. http://www.digitalsignallabs.com/fp.pdf, Julho 2009.