Tradeoff of FPGA Design of Floating-point Transcendental Functions

Daniel M. Muñoz[1], Diego F. Sánchez[1], Carlos H. Llanos[1], Mauricio Ayala-Rincón[2]
[1]Departamento de Engenharia Mecânica and [2]Instituto de Ciências Exatas

Universidade de Brasília
Brasília, D.F., Brasil, 70910-900
Email: {damuz, difesanchez, llanos, ayala}@unb.br

***Abstract*—Several scientific applications need a high precision computation of transcendental functions. This paper presents a hardware implementation of a parameterizable floating-point library for computing sine, cosine and arctangent functions using both CORDIC algorithm and Taylor series expansion for different bit-width representations. The results include the accuracy as a design criterion of the proposed hardware architectures; therefore, a tradeoff analysis between the cost in area and the number of iterations against the error associated is done in order to choose a suitable format for computing transcendental functions. The proposed architectures were validated using the Matlab results as a statistical estimator in order to compute the Mean Square Error (MSE). Synthesis and simulation results demonstrate the correctness and effectiveness of the implemented hardware transcendental functions.**

I. Introduction

Transcendental functions are important operations for several applications such as multimedia, image processing, Digital Signal Processing (DSP) and manipulators kinematics, in which, a large number of sine, cosine and arctangent operations are performed [1], [2], [3]. Floating-point arithmetic is an essential component of high-performance computer systems due to its dynamic range for representing real numbers and capabilities to retain its resolution and accuracy compared with fixed-point arithmetics [4], [5].

Common criteria for DSP hardware design are the area cost, latency and power consumption. High accurate operations require a large area and/or more latency than less accuracy operations; therefore, to include accuracy as a further design criterion of arithmetic and transcendental functions in hardware, has a relevant importance.

Modern Field Programmable Gates Arrays (FPGAs) can be used to embedded directly in hardware more complex algorithms, exploring the inherent parallelism in order to obtain an expressive speed-up [6]. There are two important aspects that must be considered when implementing floating-point operations: (a) the tradeoff between the need of reasonable accuracy and the cost of logic area and (b) the choice of a suitable format such that dynamic range is large enough to guarantee that saturation will not occur for a general-purpose application.

There are several previous works covering FPGA implementations of floating-point transcendental functions. However, these works have not received enough attention on the cost in area associated with the precision level, as well as, their respective error analysis.

This paper presents an FPGA implementation of a parameterizable library for computing floating-point sine, cosine and arctangent functions using the CORDIC algorithm and the Taylor series expansion. The implemented cores were described in VHDL and are based on the IEEE-754 standard. This paper provides a tradeoff analysis between the cost in area (bit-width representation) against the error associated, in order to analyze behavioral aspects of the proposed architectures.

Section 2 describes the CORDIC and the Taylor series expansion algorithms. Section 3 presents related works on FPGAs implementation of transcendental functions. Section 4 describes the hardware implementations and before conclude, section 5 presents the synthesis and simulations results, as well as, an error analysis of the proposed hardware implementations.

II. Background

The IEEE-754 format [7] is a floating-point number representation characterized by three components: a sign S, a biased exponent E with Ew bit-width and a mantissa M with Mw bit-width as shown in Fig. 1. A zero sign bit denotes a positive number and a one sign bit denotes a negative number. A constant ($bias$) is added to the exponent in order to make the exponents range nonnegative and the mantissa represents the magnitude of the number.

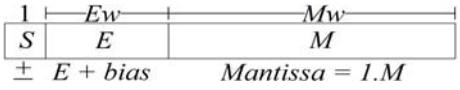

Fig. 1. IEEE-754 format

This standard allows the user to work not only with the 32-bit single precision and 64-bit double precision, but also with a suitable precision according to the application. This is suitable for supporting variable precision floating-point operations [2].

A. The CORDIC algorithm

The CORDIC (COordinate Rotation Digital Computer) algorithm is an shift-addition based iterative method to compute

rotation of a two-dimensional vector and to compute length and angle of a vector. The algorithm has two modes of operation: rotation and vectoring mode. In the rotation mode the inputs are the vector (X_i, Y_i) and the input angle θ to be rotated. The output is the rotated vector (X_{i+1}, Y_{i+1}). The rotations are based over small precomputed elementary angles (θ = atan(2^{-i})) stored in a ROM. In the vectoring mode the input is the vector (X_i, Y_i) and the outputs are the magnitude R and the angle θ of the vector. The CORDIC iterative equations are:

$$X_{i+1} = X_i - \sigma_i 2^{-i} Y_i \tag{1a}$$

$$Y_{i+1} = Y_i + \sigma_i 2^{-i} X_i \tag{1b}$$

$$Z_{i+1} = Z_i + \sigma_i \theta_i \tag{1c}$$

In the CORDIC algorithm, the rotational mode is used for computing the sin and cos functions and the vectoring mode is employed for computing the $atan$ function. The algorithm is performed in two stages.

1) Vector initialization:
   a) For computing *sin* and *cos* are used the initial values of $X_0 = (1/P)$, $Y_0 = 0$, and $Z_0 = \theta$, where $P = 1.64676$
   b) For computing $atan$ are used the initial values $X_0 = X, Y_0 = Y, Z_0 = \pi/2$. In this case is calculated the function $atan(X_0/Y_0)$.
2) Successive iterations (micro-rotations):
   a) Identify the sign of the operation (σ_i = 1)
      i) Do σ_i =$\sigma_i(Z_i)$ in the case of *sin* and *cos* functions and $\sigma_i = \sigma_i(-Y_i)$ in case of *atan* function.
      ii) Execute the equations (1).
   b) After *N* iterations (*N* micro-rotations) the output variables will store the following values:
      i) $[X_N, Y_N, Z_N] = [cos(\theta), sin(\theta), 0]$ in the case of sin and cos functions.
      ii) $[X_N, Y_N, Z_N] = [P\sqrt{X_0^2 + Y_0^2}, 0, atan(\frac{X_0}{Y_0})]$ in case of the $atan$ function.

Notice that the addition/subtraction in equations (1) can be performed in a parallel approach.

1) Taylor Series Expansion: The Taylor series expansion for sine, cosine and arctangent functions, around 0, are given respectively as:

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots \tag{2}$$

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots \tag{3}$$

$$atan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots \tag{4}$$

The factors $1/n!$ and $1/n$ are precomputed and stored in a ROM avoiding additional operations. Before the first iteration, the x^2 term is computed and stored. In each iteration one floating-point addition of the accumulated approximation with the i^{th} term is necessary. Additionally, for computing the i^{th} term two floating-point multiplications are needed: (1) for computing either $x^2 \cdot x^{2i-1}$ or $x^2 \cdot x^{2(i-1)}$ for *sin/atan* or *cos*, respectively, and (2) for computing either $1/(2i+1)! \cdot x^{2i+1}$ for *sin*, $1/(2i+1) \cdot x^{2i+1}$ for *atan* and $1/(2i)! \cdot x^{2i}$ for *cos*.

III. Related Works

Most previous FPGA implementations for computing sin, cos and $atan$ functions are based on fixed-point arithmetic. [8] explores the FPGA implementation of the CORDIC algorithm for computing a wide range of functions using both iterative and unrolled architectures. In [9] and [10] a variable precision method of fixed-point CORDIC processor was implemented in an FPGA, whereas, the CORDIC and multipartite LUT-based architectures for $atan$ function are compared in [11].

FPGA implementations of floating-point analytic functions using a LUT-based method and the CORDIC algorithm are presented in [12]. In [13] an FPGA implementation of double-precision FP-CORDIC processor is presented. In [14] a HOTBM method for computing trigonometric functions in FPGAs is presented, achieving a reduced area and high performance without sacrificing accuracy.

In summary, previous work proposes different architectures for implementing fixed and floating-point transcendental functions on FPGAs. However, most previous works do not consider the accuracy as design criterion for implementing floating-point transcendental functions in hardware. In this work, the CORDIC algorithm and the Taylor expansion are implemented on FPGAs, for computing in an integrate approach, the sin, cos and $atan$ functions, using a floating-point arithmetic based on the IEEE-754 standard. This work provides an error analysis associated to the bit-width representation and the area cost, as well as, an error analysis associated to the number of iterations. This results are useful for choosing a suitable format for representing real numbers according to general-purpose applications.

IV. Hardware Implementations

The architectures have been described in VHDL using the Xilinx ISE 9.1 development tool. All the floating-point cores are based on the IEEE-754 standard and are parameterizable by bit-width and number of micro-rotations for the CORDIC algorithm and number of powers for the Taylor series.

Fig. 2 shows the implemented iterative CORDIC algorithm. It consist of a micro-rotation unit and a Finite State Machine (FSM) unit. The *sin/cos atan* input selects the desired function. The micro-rotation unit is based on three floating-point add/sub units (*FPadd*), a search in the ROM for recovering the precomputed angle θ_i and two (E-bits) fixed-point additions for computing $2^{-i} \cdot Y_i$ and $2^{-i} \cdot X_i$ that can be performed by subtracting the exponent bit-word (E) to the current iteration (n). Notice that the dominant operations are the three *FPadd*. In principle, one can measure the running time complexity as the number of micro-rotations times the computational cost of these additions, that is $O(NCostadd)$.

The proposed architecture use the same hardware resources for computing the *sin*/*cos* or *atan* functions, representing an improvement in terms of area cost and running time, taking into account the parallelism of the add/sub operations.

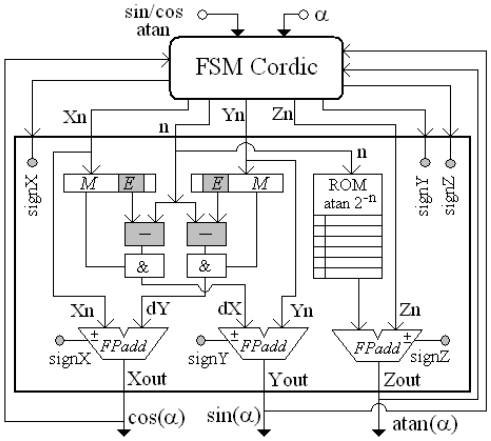

Fig. 2. Floating-point iterative CORDIC

Fig. 3 shows the architecture for the Taylor series expansion. This approach was achieved using only one $FPmultiplier$, a $FPadd/sub$, a FSM and three ROMs for storing the precomputed $1/n!$ and $1/n$ factors.

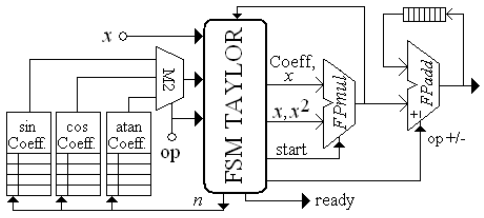

Fig. 3. Floating-point iterative Taylor series expansion

The *op* signal chooses between a *sin*, *cos* or *atan* functions. The FSM synchronizes the *FPmul*, *FPadd* units and alternates the *op*+/- signal. At the first iteration, the factor x^2 is computed, stored in a register and feeding back to the FSM. Afterwards, the precomputed coefficients are selected in order to compute the current term either $x^{2n-1}/(2n-1)!$ for *sin*, $x^{2n-1}/(2n-1)$ for *atan* and $x^{2n}/2n!$ for *cos*. The *add*/*sub* unit computes and accumulates both the addition or subtraction operations. After N iterations, the *ready* signal indicates a valid output.

V. RESULTS

The implemented cores were simulated using the ModelSim 6.3g simulator tool. A simulation environment, developed in Matlab, creates one hundred test vectors for different bit-widths. The binary floating-point results were contrasted with the Matlab results as statistical estimator (double precision), in order to calculate the Mean Square Error (MSE) of the implemented architectures.

A. Synthesis Results

Table 1 shows the synthesis results for the Xilinx Virtex5 family using the ISE 9.1 development tool. The CORDIC architecture is more expensive in slices and LUTs in comparison with the Taylor architecture; however, the CORDIC architecture does not consume embedded multipliers. The higher area cost of the FP-CORDIC core is due to the three $FPadd$ units for implementing the micro-rotation, in comparison with only one $FPadd$ unit of the Taylor series expansion. The floating-point add/sub unit is over 6.7 times more expensive than the floating-point multiplier unit [15]. Notice that the performance decrease with large bit-width representations.

TABLE I
SYNTHESIS RESULTS. VIRTEX5 XC5VLX330

Bit width (Exp, Man)	Implemented Core	Slices 207360	LUTs 207360	MUL 192	Freq MHz
24 (6,17)	FPCORDIC	240	2182	0	96.5
	FPTaylor	285	1074	1	98.7
28 (7,20)	FPCORDIC	286	2456	0	89.3
	FPTaylor	329	1285	2	98.5
32 (8,23)	FPCORDIC	320	2832	0	86.1
	FPTaylor	373	1436	2	95.7
43 (11,31)	FPCORDIC	421	4113	0	80
	FPTaylor	494	1848	4	73.5
64 (11,52)	FPCORDIC	600	11718	0	67.1
	FPTaylor	725	4297	15	56.3

B. Simulation Results

Tables 2 and 3 present the MSE, latency in clock cycles and throughput in mega operations per second (Moper/s) for the CORDIC algorithm and the Taylor series expansion respectively, using a simple precision format. The input arguments for sin and cos were $[-\pi/2$ to $\pi/2]$ and $[-100$ to $100]$. Notice that $FPTaylor$ core needs few iterations for achieving the same error than the $FPCORDIC$ core for the sin and cos functions. However, the $FPCORDIC$ core achieves errors around E^{-17} for 26 micro-rotations, whereas, the smaller error achieved by the $FPTaylor$ core is over E^{-15}, for five powers (x^{11} polynomial degree). More than five powers of the series expansion increase the error due to the Runge's phenomenon, when using polynomial interpolation with polynomials of high degree.

Each micro-rotation of the $FPCORDIC$ is performed in 3 clock cycles, whereas each iteration of the $FPTaylor$ core requires 4 clock cycles.

TABLE II
MSE AND LATENCY FPCORDIC ARCHITECTURE

uRotations	14	17	20	23	26
MSE sin(x)	6.68E-10	1.03E-11	1.97E-13	2E-15	4.48E-17
MSE cos(x)	5.74E-10	1.16E-11	1.99E-13	2.60E-15	4.27E-17
MSE atan(x)	3.58E-06	6.26E-08	1.24E-09	1.76E-11	2.46E-13
Latency	46	55	64	73	82
Throughput	1.87	1.56	1.34	1.18	1.05

In the case of the $atan$ function, the $FPCORDIC$ core achieves better results, obtaning errors over E^{-13} for 26

TABLE III
MSE AND LATENCY FPTAYLOR ARCHITECTURE

Powers	2	3	4	5	6
MSE sin(x)	1.59E-06	1.56E-09	7.01E-13	3.13E-15	2.13E-14
MSE cos(x)	3.52E-05	5.58E-08	3.62E-11	1.15E-14	5.91E-15
MSE atan(x)	0.00137	0.00136	0.00136	0.00136	0.00136
Latency	11	15	19	23	27
Throughput	8.7	6.38	5.04	4.16	3.54

micro-rotations. Fig. 4 shows a comparison between the $FPTaylor$ core and the Matlab results in case of the $atan$ function. It can be observed a satisfactory approximation for input values between [-0.8 to 0.8], achieving errors over E^{-19}. However, input values between [0.8 to 1] produce a poor approximation of the the $FPTaylor$ core, achieving errors around E^{-3}. Input values higher than 1 produce a divergence of the MSE, given that the powers of the series expansion for $atan$ function grows faster than their respective coefficients.

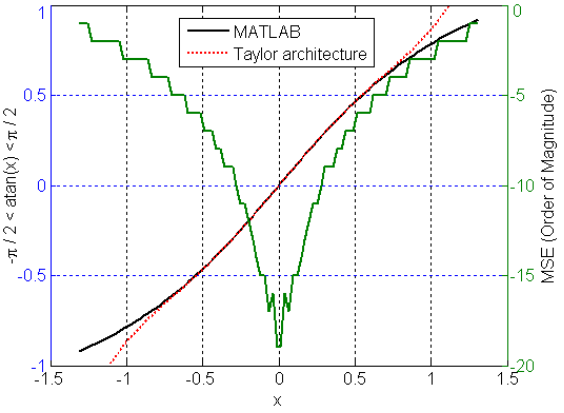

Fig. 4. MSE of the $FPTaylor$ core for $atan(x)$

Table 4 shows the MSE results of the implemented architectures for different bit-width representations. The best results are achieved using the double precision format; however, the cost in area is increased.

TABLE IV
MSE USING 20 MICRO-ROTATIONS / 5 NONZERO POWERS

Core	24(6,17)	28(7,20)	32(8,23)	43(11,31)	64(11,52)
CORDIC sin/	2.32E-10	8.88E-11	1.50E-10	1.33E-13	1.07E-13
CORDIC atan	2.58E-07	5.60E-09	1.24E-09	9.02E-10	1.10E-09
Taylor sin/cos	6.31E-09	9.43E-09	8.80E-09	6.26E-09	1.94E-16
Taylor atan	0.073	0.015	0.0014	0.015	0.015

VI. CONCLUSIONS

This work describes an FPGA implementation of a parameterizable library for computing floating-point sine, cosine and arctangent functions, using both, the CORDIC algorithm and the Taylor series expansion. The $FPCORDIC$ core uses three floating-point add/sub units and a LUT, whereas the $FPTaylor$ core needs one floating-point multipliers, one floating-point add/sub and a search in ROM.

The accuracy of the implemented cores has been included as a design criterion in order to solve the tradeoff among the cost in area, latency and the error associated. The Taylor architecture has a smaller area cost and higher throughput than the CORDIC architecture; however, the smaller errors were achieved by accomplishing the CORDIC core. Although the Taylor architecture provides a satisfactory approximation for computing sine and cosine functions, is not suitable for computing the arctangent function.

As future works we pretend to extend the accuracy as design criterion in others floating-point hardware operators, as well as, the application of the transcendental functions described in this paper to direct and inverse kinematics in robotic manipulators. Also, the implementation of an FPGA floating-point $atan2(X/Y)$ function can be easily accomplished by using the $FPCORDIC$ core.

REFERENCES

[1] G. Cappuccino, G. Cocorullo, P. Corsonello, and G. Schirinzi, "Design and demonstration of a real time processor for one-bit coded SAR signals," *IEE proc. Radar, sonar and navigation*, vol. 143, no. 4, pp. 261–267, 1996.

[2] X. Wang, "Variable precision floating-point divide and square root for efficient FPGA implementation of image and signal processing algorithms," Ph.D. dissertation, Northeastern University, 2007.

[3] Y. Kung, K. Tseng, C. Chen, H. Sze, and A. Wang, "FPGA-implementation of inverse kinematics and servo controller for robot manipulator," in *Proc. IEEE Int. on Robotics and Biomimetics*, Dec. 2006, pp. 1163–1168.

[4] M. Ibne, S. Islam, and M. Sulaiman, "Pipeline floating point ALU design using VHDL," in *IEEE Proc. Int. Semiconductor Electronics*, 2002, pp. 204–208.

[5] S. Hauck and A. Dehon, *Reconfigurable Computing*. United States: Elsevier Inc., 2008.

[6] M. Beauchamp, S. Hauck, K. Underwood, and K. Hemmert, "Embedded floating-point units in FPGAs," in *Proc. ACM Int. Symp. Field Programmable Gate Arrays*, 2006, pp. 12–20.

[7] *IEEE standard for binary floating-point arithmetic*, ANSI/IEEE Std 754-1985, Aug. 1985. [Online]. Available: http://www.altera.com/

[8] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *Proc. ACM Inter. Symp. on Field Programmable Gate Arrays*, Feb. 1998, pp. 191–200.

[9] E. Saez, J. Villaba, J. Hormigo, F. Quiles, J. Benavides, and E. Zapata, "FPGA implementation of a variable precision CORDIC processor," in *Proc. IEEE Conf. Design Circuits and Integrated Syst. Computers*, 1998, pp. 604–609.

[10] J. Valls, M. Kuhlmann, and K. Parhi, "Evaluation of cordic algorithms for FPGA design," *Journal of VLSI Signal Processing*, vol. 32, no. 3, pp. 207–222, 2002.

[11] R. Gutierrez and J. Valls, "Low-power FPGA-implementation of atan(y/x) using LUT methods for communication applications," *Journal of Signal Processing Systems*, vol. 56, no. 1, pp. 25–33, 2009.

[12] F. Ortiz, J. Humphrey, J. Durbano, and D. Prather, "A study on the design of floating-point functions in FPGAs," *Field Programmable Logic and Applications*, vol. 2778, p. 11311135, 2003.

[13] J. Zhou, J. Dou, Y. Lei, and J. Xu, "Dynamic configurable floating-point FFT pipelines and hybrid-mode CORDIC on FPGA," in *Proc. IEEE Int. Conf. Embedded Circuits and Systems*, 2008, pp. 616–620.

[14] J. Detrey and F. Dinechin, "Floating-point trigonometric functions for FPGAs," in *Proc. IEEE Int. Conf. Field Programmable Logic and Applications*, 2007, pp. 29–34.

[15] D. Sánchez, D. Munoz, C. Llanos, and M. Ayala-Rincón, "Parameterizable floating-point library for arithmetic operations in FPGAs," in *in Press Proc. ACM Inter. Symp. on Int. Circuits and Systems Design*, 2009.