THE IMPLEMENTATION OF A PIPELINED FLOATING-POINT CORDIC COPROCESSOR ON NIOS II SOFT PROCESSOR

[1]MUHAMMAD NASIR IBRAHIM, [2]CHEN KEAN TACK, [3]MARIANI IDROAS, [4]ZURAIMI YAHYA

[1,2]Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
[3]Faculty of Petroleum and Renewable Energy Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
[4]Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
E-mail: mnasir@fke.utm.my , ktchen2@live.utm.my , mariani@petroleum.utm.my, zuraimi@fke.utm.my

Abstract- This paper discusses the implementation of a pipelined floating-point Coordinate Rotation Digital Computer (CORDIC) coprocessor using Field Programmable Gate Array (FPGA) to accelerate the computation speed in solving elementary functions on NIOS II soft processor. Examples of the elementary functions are trigonometry and hyperbolic functions, exponential, natural logarithm, square root as well as multiplication and division. In order to enhance its functionality, an argument reduction algorithm was introduced to expand the convergence limit for the inputs. The design was developed using a pipelined architecture with 29 stages. In this project, the proposed CORDIC coprocessor was designed as a custom hardware component with Avalon Memory Mapped (Avalon-MM) interface. A dedicated NIOS II system was developed using hardware/software co-design methodology to allow the hardware execution on NIOS II software. Thus, the floating-point data are represented in the 32-bit single precision floating-point format that are compliant with IEEE-754 standard. The design was modelled using System Verilog HDL coding style. The verification was done by comparing the results from the CORDIC hardware and C software using math library. The performance analysis was done to obtain the speedup achieved by the proposed hardware from the corresponding software functions. Finally, the proposed coprocessor was run on Altera DE0 board with a clock frequency of 50MHz. The results achieved precision up to six decimal places, with more than 100 times of speedup against software execution time for most of the elementary functions. However, smaller speedup was achieved for the square-root, multiplication and division operations.

Keywords- CORDIC Coprocessor, Elementary Functions, Argument Reduction Algorithm, NIOS II, Avalon-MM Interface

I. INTRODUCTION

The performance of the soft processor in computing floating-point elementary functions plays a vital role in several Digital Signal Processing (DSP) and numeric applications. Hence, there is a need to design a hardware core to accelerate the execution speed for the computation of elementary functions on the soft processor. Thus, one way to compute elementary function is by using CORDIC algorithm, which has been utilized for applications in various fields in last five decade [1].

The CORDIC algorithm was first described in 1959 by Volder [2] for the computation of trigonometry functions, multiplication and division. It was then modified in 1971 by Walther [3] to be a unified and generalized algorithm to solve wider range of mathematical functions such as logarithm, exponential, square root, multiplication, division, trigonometry and hyperbolic functions. Basically, the CORDIC algorithm is an iterative algorithm that involves the rotation of two dimensional XY-planes in either circular, linear or hyperbolic coordinate systems based on the function to be evaluated. Meanwhile, the beauty of this algorithm lies on the fact that it can realize the solution for several elementary functions by simply executing some simple shift-add operations.

However, the basic CORDIC algorithm suffers from many limitations such as the limited convergence range, scaling problem and slow speed. To overcome limited convergence range issue, the division-free argument reduction algorithm was developed by Hahn [4], where he expanded the convergence domain into entire coordinate space. Thus, this paper describes the proposed design that was developed in three phases, which are a preprocessing with argument reduction algorithm, an iterative CORDIC calculation and a post-processing with scaling and normalization. In addition, to achieve 23-bit precision and to improve maximum allowable clock frequency, the pipelined architecture with 29 stages was also introduced.

II. BACKGROUND

In basic CORDIC algorithm, the unified equations were defined by Walther [3] and are as follows.

$$X_{i+1} = X_i - m d_i Y_i 2^{-S_{m,i}} \quad (1)$$

$$Y_{i+1} = Y_i + d_i X_i 2^{-S_{m,i}} \quad (2)$$

$$Z_{i+1} = Z_i - d_i \alpha_{m,i} \quad (3)$$

where i = the iteration index

d_i = the rotation direction

$$= \begin{cases} \text{sign}(Z_i) & \text{for rotation mode} \\ -\text{sign}(Y_i) & \text{for vectoring mode} \end{cases}$$

m = decision factor of the coordinate system

$$= \begin{cases} 0 & \text{for linear mode} \\ 1 & \text{for circular mode} \\ -1 & \text{for hyperbolic mode} \end{cases}$$

$\alpha_{m,i}$ = the elementary rotation angle

$$= \frac{1}{\sqrt{m}} \tan^{-1}(\sqrt{m} 2^{-S_{m,i}}) \quad (4)$$

$S_{m,i}$ = the integer shift sequence

$$= \begin{cases} 0,1,2,3,\ldots & \text{for } m = 0, m = 1 \\ 1,2,3,4,4,5,\ldots & \text{for } m = -1 \end{cases}$$

Notes: Some iterations in hyperbolic mode (m = -1), where $i \in \{4, 13, \ldots, k, 3k+1\}$ must be repeated twice to fulfil the convergence criteria [3].

Basically, this algorithm can be operated in two different modes, i.e. the rotation and vectoring modes with different rotation direction equation respectively. Therefore, during iteration process, it drives the Z value towards zero in rotation mode while it drives the Y value towards zero in vectoring mode. Meanwhile, the computable functions are different when it operated in different coordinate system and modes as summarized in Table 1.

Table 1: The computable functions of CORDIC algorithm

Coordinate System, Rotation angles, Convergence domain	Rotation mode ($Z_n \rightarrow 0$)	Vectoring mode ($Y_n \rightarrow 0$)
Circular (m = 1) $\alpha_{1,i} = \tan^{-1} 2^{-s_{1,i}}$ $C_1 = 1.74$	$X_n = K_1 (X_0 \cos(Z_0) - Y_0 \sin(Z_0))$ $Y_n = K_1 (Y_0 \cos(Z_0) + X_0 \sin(Z_0))$ where $K_1 = 1.646760258121 \ldots$	$X_n = K_1 \sqrt{X_0^2 + Y_0^2}$ $Z_n = Z_0 + \tan^{-1}\left(\frac{Y_0}{X_0}\right)$
Linear (m = 0) $\alpha_{0,i} = 2^{-s_{0,i}}$ $C_0 = 2.0$	$X_n = X_0$ $Y_n = Y_0 + X_0 Z_0$	$X_n = X_0$ $Z_n = Z_0 + \frac{Y_0}{X_0}$
Hyperbolic (m = -1) $\alpha_{-1,i} = \tanh^{-1} 2^{-s_{-1,i}}$ $C_{-1} = 1.113$	$X_n = K_{-1} (X_0 \cosh(Z_0) + Y_0 \sinh(Z_0))$ $Y_n = K_{-1} (Y_0 \cosh(Z_0) + X_0 \sinh(Z_0))$ where $K_{-1} = 0.8281593609602 \ldots$ Additional function: $e^w = \cosh(w) + \sinh(w)$	$X_n = K_{-1} \sqrt{X_0^2 - Y_0^2}$ $Z_n = Z_0 + \tanh^{-1}\left(\frac{Y_0}{X_0}\right)$ Additional functions: $\ln w = 2 \tanh^{-1}\left(\frac{w-1}{w+1}\right)$ $\sqrt{w} = \sqrt{(w+0.25)^2 - (w-0.25)^2}$

In order to yield the precision of n bits, the number of iteration should be at least equal to n. In the proposed design, since there is 23-bit of mantissa for single precision floating-point format, therefore at least 23 iterations are required to fulfill the precision requirement. Meanwhile, after the final iteration, the result of X and Y must be compensated for the scaling factor K_m where its values depends on the coordinate system as shown in Table 1.

However, the maximum convergence domain C_m defines the maximum allowable angle for the rotation of the input vector without producing incorrect result [4]. Thus, it should exceed a specific value given by the input data in order to guarantee the convergence as shown in equation (5) [4]. Therefore, the values of C_m for different coordinate system are also tabulated in Table 1.

$$C_m = \alpha_{m,N-1} + \sum_{i=0}^{N-1} \alpha_{m,i}$$

$$\geq \begin{cases} \left| \frac{1}{\sqrt{m}} \tan^{-1}\left(\sqrt{m} \frac{Y_0}{X_0}\right) \right| & \text{for vectoring mode} \\ |Z_0| & \text{for rotation mode} \end{cases} \quad (5)$$

Previously, there are several researches [5] [6] [7] were carried out for the implementation of floating-point CORDIC coprocessor. However, most of the researchers implemented the CORDIC coprocessor on FPGA but did not fully tested on NIOS II soft processor. There was only a few researchers [8] [9] performed the test on NIOS II using the CORDIC co-processor, but it was only supported for limited solving capability and did not consider the limitations of the basic CORDIC algorithm as discussed in the earlier section.

III. METHODOLOGY

The floating-point CORDIC coprocessor was developed first and then the proper Avalon-MM signals were introduced to the input and output to build a custom hardware component that can interface with NIOS II soft processor. Basically, its architecture was developed in three phases, i.e. preprocessing phase with argument reduction algorithm, CORDIC calculation phase and post-processing phase with scaling and normalization. Thus, the overall data path unit (DU) of the proposed design was partitioned into 29 stages and its control signals were controlled by one control unit (CU). Finally, a NIOS II based system was developed using Quartus qsys tool to allow the execution of custom hardware component on NIOS II soft processor using C program to undergo the performance test. All the hardware designs were coded in System Verilog HDL, meanwhile the software testing part was coded in C program.

A. Pipelined Floating-point CORDIC coprocessor

Basically, it was developed in three modules, i.e. pre processor, pipelined CORDIC module and postprocessor, which performed different operations as described in Fig. 1.

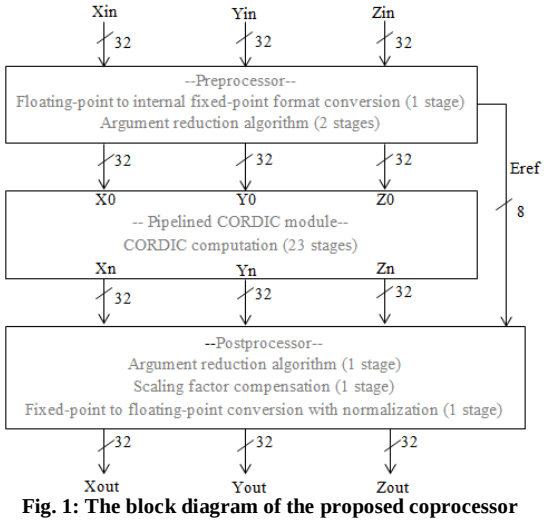


Fig. 1: The block diagram of the proposed coprocessor

(a) Pre processor (3 stages) transformed the input data of X, Y and Z from 32-bits single precision floating-point IEEE754 format into a self-defined 32-bit internal fixed-point format. In addition, it also employed with the division-free argument reduction algorithm developed by Hahn [4] to

expand the convergence range of the input data into the entire coordinate space.

(b) Pipelined CORDIC module (23 stages) used all the internal fixed-point data retrieved from the output of the pre processing part to execute the CORDIC iterations in order to evaluate certain computable elementary functions from Table 1.

(c) Postprocessor (3 stages) transformed the outputs back to 32-bits single precision floating-point format and performed scaling by multiplication and normalization process to ensure the output was compliant with IEEE754 standard. However, some CORDIC functions still required back transformation of argument reduction algorithm in this block before it was converted back to floating-point format.

Hence, in overall, the proposed coprocessor required a total of 29 pipeline stages to perform the CORDIC computation and its architecture was scheduled as shown in Fig. 2.

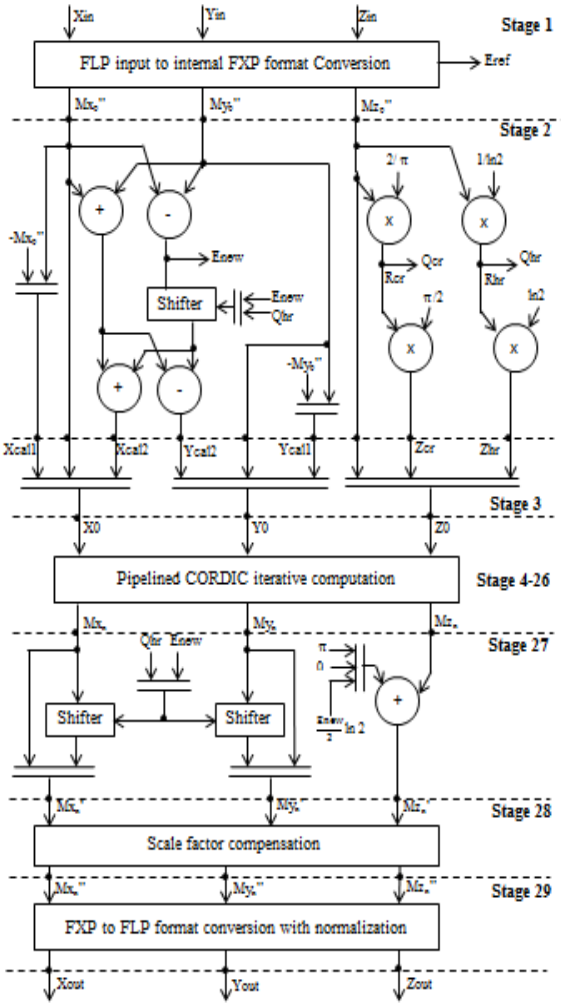


Fig. 2: The schedule and the architecture of the proposed coprocessor with argument reduction algorithm

By RTL design approach, the coprocessor design was partitioned into data path unit and controller. Thus, the overall functional block diagram is shown in Fig.3.

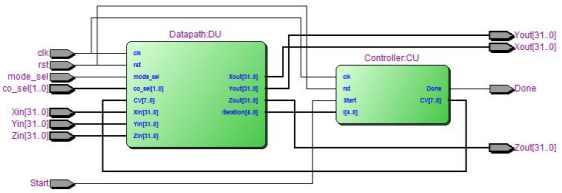


Fig. 3: The overall functional block diagram of the proposed coprocessor

A. Floating-point to Internal Fixed-point Format Transformation

Assume that the floating-point input data are represented by the following equations:

$$X_0 = Sx_0 2^{Ex_0} Mx_0 \quad (6)$$

$$Y_0 = Sy_0 2^{Ey_0} My_0 \quad (7)$$

$$Z_0 = Sz_0 2^{Ez_0} Mz_0 \quad (8)$$

where S = sign, E=exponent, and M=mantissa

Then, the reference exponent, Eref and its corresponding values for each coordinate system and mode are evaluated by referring to Table 2.

Table 2: Evaluation of Reference Exponent

Coordinate System	Rotation mode	Vectoring mode
Circular (m=1)	$Eref = \max(Ex_0, Ey_0)$	$Eref = \max(Ex_0, Ey_0)$
Linear (m=0)	$Eref = \max(Ey_0, Ex_0 + Ez_0)$	$Eref = \max(Ez_0, Ey_0 - Ex_0)$
Hyperbolic (m=-1)	$Eref = \max(Ex_0, Ey_0)$	$Eref = \max(Ex_0, Ey_0)$

Therefore, the input 32-bit single precision floating-point data of X_0, Y_0 and Z_0 were converted into respective internal fixed-point format by the pre processor in 32-bit with suitable numbers of overflow and guard bits prior to the CORDIC calculation phase, as defined in Fig. 4. This format can be treated as 2's complement binary format to perform any shift-add operations for the CORDIC algorithm.

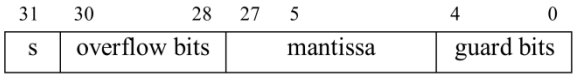


Fig. 4: Internal fixed-point format for X, Y and Z

Thus, the fixed-point formats of X_0, Y_0 and Z_0 were expressed by Mx_0", My_0" and Mz_0" as shown below:

$$Mx_0" = Sx_0\, Mx_0 2^{Ex_0 - Eref} \quad (9)$$

$$My_0" = Sy_0\, My_0 2^{Ey_0 - Eref} \quad (10)$$

$$Mz_0" = Z_0 = Sz_0 2^{Ez_0} Mz_0 \quad (11)$$

B. Argument Reduction Algorithm

A division-free argument reduction algorithm developed by Hahn [4] was employed in the CORDIC coprocessor to expand the convergence range of the input data. Basically, this algorithm depends on the coordinate system of the CORDIC algorithm because the maximum convergence domains for each coordinate system are different by referring to Table 1.

Thus, the algorithm is derived based on the mode and the coordinate system of the CORDIC algorithm and its architecture is modified as shown in Fig. 2 [4].

B. NIOS II System Development

The NIOS II system was implemented on Altera DE0 board and operated in 50MHz. Thus, this system needs to be developed in both hardware and software parts based on the hardware/software co-design methodology [10], so that the NIOS II system can run the hardware on the NIOS II software (for the purpose of verification and performance analysis in this paper) as described in the following subsections.

A. Hardware development

Firstly, a custom hardware component was built for the coprocessor and it was integrated into NIOS II system. Then, the Single-on-chip (SoC) design was developed using Qsys tool and the hardware was run on NIOS II software as described in the following subsections.

(a) Building the custom hardware component

To integrate the hardware coprocessor into NIOS II system, a custom hardware component was built using Qsys tool that shall consist of all the required Avalon-MM interface signals as shown in Fig. 5. The functional block diagram of the custom component is shown in Fig. 6.

Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
rst	reset	reset_n	1	input
write	CORDIC	write	1	input
read	CORDIC	read	1	input
writedata	CORDIC	writedata	32	input
address	CORDIC	address	4	input
readdata	CORDIC	readdata	32	output
chipselect	CORDIC	chipselect	1	input

Fig. 5: The Avalon-MM interface signals in the custom component



Fig. 6: The functional block diagram of the custom component

(b) System-on-chip (SoC) design

To run the hardware on NIOS II software, the Qsys tool was used to develop a SoC design, which consists of a NIOS II processor, SDRAM memory unit, custom hardware component, interval timer, system ID peripheral and so forth. Meanwhile, the block diagram of the completed NIOS II system is shown in Fig. 7.

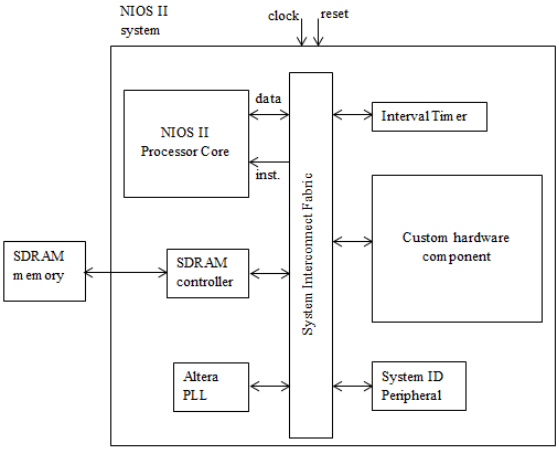


Fig. 7: The block diagram of the NIOS II system

B. Software development

The main purpose of the NIOS II software was to verify the design by obtaining the results calculated by the hardware coprocessor, and the results were compared with the results computed by the C software built-in math library. Apart from that, the NIOS II software was also used to compare the performance achieved by the proposed hardware coprocessor design with the C software execution. Thus, the timestamp timer function was used to capture the time taken for specified executed operations on the NIOS II soft processor. Therefore, by knowing the time taken for the hardware and software execution, the speedup achieved for the hardware execution can be computed based on the following equation.

$$\text{Speedup} = \frac{\text{Time taken to execute software function}}{\text{Time taken to execute hardware function}}$$

IV. RESULT AND ANALYSIS

The proposed hardware coprocessor was integrated into NIOSII soft processor and executed in NIOS II software to verify the computed results and its performance was analyzed against software execution. The hardware/software co-design was run on Altera DE0 board with the clock frequency of 50MHz.

A. Verification of the computed results

By using the proposed NIOS system, we can display the results calculated by the hardware coprocessor and also by the C software built-in math library on the console window. Thus, the output results obtained from the hardware and software execution were collected and tabulated in Table 3.

Table 3: The output results obtained by hardware and software execution

Input value(s)	Software Results	Hardware Results
Test for cos(z)		
z = 0.785398	0.707107	0.707107

z = 1.745329	-0.173648	-0.173648
z = 3.839724	-0.766045	-0.766044
z = 5.759587	0.866026	0.866026
Test for sin(z)		
z = 0.785398	0.707107	0.707107
z = 1.745329	0.984808	0.984808
z = 3.839724	-0.642787	-0.642787
z = 5.759587	-0.500000	-0.500000
Test for arctan(y)		
y = 0.562578	0.512449	0.512449
y = -13.679456	-1.497824	-1.497823
y = 278.945678	1.567211	1.567211
y = -7867.8948	-1.570669	-1.570669
Test for cosh(z)		
z = 0.457824	1.106645	1.106645
z = 0.976593	1.515993	1.734465
z = 1.741932	2.941771	2.941772
z = 2.678949	7.319206	7.319205
Test for sinh(z)		
z = 0.457824	0.473986	0.473986
z = 0.976593	1.139401	1.139401
z = 1.741932	2.766590	2.766590
z = 2.678949	7.250569	7.250570
Test for exp(z)		
z = 0.045893	1.046962	1.046962
z = 0.457824	1.580631	1.580631
z = 0.976593	2.655394	2.655394
z = 1.741932	5.708361	5.708361
Test for arctanh(y)		
y = 0.056982	0.057044	0.057044
y = -0.487651	-0.532974	-0.532974
y = 0.784569	1.057146	1.057146
y = 0.965892	2.027085	2.027085
Test for logn(w)		
w = 0.546783	-0.603703	-0.603703
w = 5.786576	1.755541	1.755541
w = 38.678400	3.655281	3.655281
w = 6968.4673	8.849151	8.849150
Test for sqrt(w)		
w = 0.546783	0.739448	0.739448
w = 5.786576	2.405530	2.405530
w = 38.678400	6.219196	6.219196
w = 6968.4673	83.477348	83.477325
Test for x*z		
x=0.546789 z=-4.687342	-2.562987	-2.562987
x=10.546789 z=1.687342	17.796041	17.796038
x=-769.6753 z=4.891341	-3764.744385	-3764.744385
x=7.658902 z=4569.3424	34996.144531	34996.148438
Test for y/x		
x=2.658902 y=-6.965892	-2.619838	-2.619838
x=0.084134 y=0.453263	5.387394	35.387395
x=-34.675342 y=543.6753	-15.679017	-15.679020
x=2351.45 y=451.7896	0.192132	0.192132

From Table 3, almost the same results were obtained from the hardware and software executions, thus the hardware design was verified. Also, the results achieved the precision up to six decimal places.

B. Performance comparison between hardware and software

By using the timestamp timer function, we can obtain the time taken to execute the hardware and software calculation for the elementary functions. Thus, Table 4 shows the time taken in microseconds (us) to perform the computations by C software functions and hardware coprocessor and also the corresponding speedup achieved for each function.

Table 4: The execution time for hardware and software and speedup achieved

Functions to be analyzed	The average execution time for software (us)	The average execution time for hardware (us)	Speedup achieved
cos(z)	16621.6	35.9	463
sin(z)	16125.9	35.9	449
arctan(y)	22392.4	37.4	599
cosh(z)	15600.2	36.8	424
sinh(z)	23128.3	36.6	632
exp(z)	12626.6	37.1	340
arctanh(y)	21625.9	37.6	575
logn(w)	19904.4	186.6	107
sqrt(w)	499.7	184.9	2.7
x*z	425.0	37.0	11.5
y/x	130.7	37.0	3.5

Based on Table 4, the proposed CORDIC coprocessor can significantly accelerated the computation for most of the elementary functions when running on NIOS II soft processor. However, there was only a small speedup achieved for the square root, multiplication and division operations since the average execution time for software for these three functions were relatively faster than the other computable functions of CORDIC algorithm.

CONCLUSION

In a nutshell, the pipelined floating-point CORDIC

coprocessor on the NIOS II soft processor was successfully implemented. The proposed design produced high accuracy results and achieved more than 100 times of speedup when executing most of the elementary functions, but small speedup for square-root, multiplication and division operations. For future work, the coprocessor can be further developed based on the issues of scale factor compensation, large number of iterations and limited convergence domain. Apart from that, the coprocessor can also be modified for the use of specific applications such as matrix inversion, eigenvalue computation and so forth.

ACKNOWLEDGEMENT

The authors would like to acknowledge Ministry of Education (Malaysia) and Universiti Teknologi Malaysia for the support and a research grant under Vot 4F502

REFERENCES

[1] P. K. Meher, J. Valls, J. Tso-Bing,K. Sridharan, and K. Maharatna, "50 Years of CORDIC: Algorithms, Architectures, and Applications," Circuits and Systems I: Regular Papers, IEEE Transactions on, vol.56, no.9, pp.1893-1907, Sept. 2009.

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

[3] J. S. Walther, "A unified algorithm for elementary functions," in AFIPS Spring Joint Computer Conference, vol. 38, pp 379-85, 1971.

[4] H. Hahn, D. Timmermann, B. J. Hosticka, and B. Rix, "A unified and division-free CORDIC argument reduction method with unlimited convergence domain including inverse hyperbolic functions," IEEE Transactions on Computers, vol.43, no.11, pp.1339-1344, Nov 1994.

[5] J. Zhou, Y. Dou, Y. Lei, and Y. Dong, "Hybrid-mode floating-point FPGA CORDIC Co-processor," 4th International Workshop, ARC 2008, London, UK, pp. 256-261, March 26-28, 2008.

[6] A. Madian and M. Aljarhi, "A Multi Cordic Architecture on FPGA Platform," International Journal of Electrical, Electronic Science and Engineering, vol. 7, no. 12, pp. 1264-1272, 2013.

[7] T. Vladimirova, D. Earney, S. Keller and Prof. Sir M. Sweeting, "Floating-Point Mathematical Co-Processor for a Single-Chip On-Board Computer", Proceedings of the 6th Military and Aerospace Programmable Logic Devices International Conference 2003 (MAPLD'03), Paper D5, 2003.

[8] T.H Tan and M. N. Marsono, ``Hardware Design Space Exploration of Cordic Algorithm For Run-Time Reconfigurable Platform", in the Proceedings of the First International Conference on Green Computing, Technology and Innovation (ICGCTI 2013), Kuala Lumpur, Malaysia, pp. 1 -7, March 2013.

[9] M. Sazali and M. Ilyas Sobirin, "Hardware implementation of coordinate rotation digital computer in field programmable gate array". Masters thesis, Universiti Teknologi Malaysia, Faculty of Electrical Engineering, 2012.

[10] X. Zhihui, L. Sikun, C. Jihua,and W. Dawei, "A platform-based SoC hardware/software co-design environment," Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on, vol.2, no., pp.443-448 Vol.2, 26-28 May 2004.

★★★