A PIPELINED 32-BIT SINGLE PRECISION FLOATING-POINT COORDINATE ROTATION DIGITAL COMPUTER COPROCESSOR ON FIELD PROGRAMMABLE GATE ARRAY

M.Nasir Ibrahim[1,*], Chen Kean Tack[1], Mariani Idroas[2]

[1*]*School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia*

[2]*School of Energy Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia*

[*]For correspondence; Tel. + (607) 5535260, E-mail: mdnasir@utm.my

ABSTRACT: *With the rapid growth of digital signal processing (DSP) applications, there is a high demand for efficient implementation of complex arithmetic operations. In the last five decades, the Coordinate Rotation Digital Computer (CORDIC) algorithm has been widely adopted to formulate and implement a variety of DSP algorithms for reconfigurable computing. In this research, an IEEE-754 compliant pipelined 32-bit single-precision floating-point CORDIC coprocessor for reconfigurable computing was implemented on Field Programmable Gate Array (FPGA) to accelerate the performance of several arithmetic computations, such as multiplication, division, and 11 elementary transcendental functions. As the CORDIC algorithm suffers from limitations for its convergence domain and speed, the unified argument reduction algorithm and the hybrid angle method were adopted. In this research, the developed coprocessor was integrated into a NIOS II soft processor to develop a NIOS II-based embedded System-on-Chip (SoC) for verification and performance analysis. The coprocessor was run on Altera DE0 board with a clock frequency of 50MHz. The experimental results showed the precision up to six decimal places and the speedup from software to hardware executions up to approximately 600 times. Besides, the performance improvement of approximately 553 times was achieved when executing one-dimensional Discrete Cosine Transform (DCT) algorithm using the developed coprocessor.*

Keywords: CORDIC, FPGA, coprocessor, algorithm, arithmetic

1. INTRODUCTION

A basic CORDIC algorithm is derived based on two-dimensional vector rotation in a circular coordinate system through a given angle, ϕ, as illustrated in Figure (1) and its equations are shown in equation (1) and equation (2) [1].

$$x' = x\cos\phi - y\sin\phi \quad (1)$$

$$y' = y\cos\phi + x\sin\phi \quad (2)$$

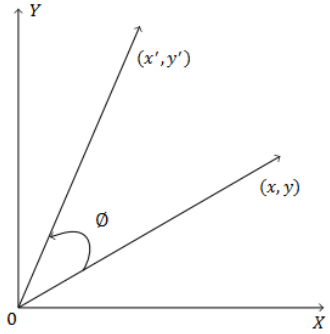

Fig (1) A vector rotation of basic CORDIC algorithm

The equations (1) and (2) can be expressed in matrix form as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3)$$

After simplification, the equation (3) becomes

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos\phi \begin{bmatrix} 1 & -\tan\phi \\ \tan\phi & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2\phi}} \begin{bmatrix} 1 & -\tan\phi \\ \tan\phi & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (4)$$

(Trigonometry identity: $\cos\phi = \frac{1}{\sqrt{1+\tan^2\phi}}$)

The rotation angle ϕ is then approximated by the sum of a series of small successively micro-rotation angles, α_i. Since the micro-rotation angle can be positive or negative (depends on the rotation direction), a rotation direction parameter, σ_i, is introduced.

$$\phi = \alpha_0 \pm \alpha_1 \pm \alpha_2 \pm \alpha_3 \pm \cdots \alpha_{n-1} \quad (5)$$

or $\phi = \sum_{i=0}^{n-1} \sigma_i \alpha_i$ where $\sigma_i \in \{-1, 1\}$.

To further simplify the equation (4), the micro-rotation angles α_i are chosen such that $\tan\alpha_i = \pm 2^{-i}$ where i = 0, 1, 2, 3….n-1. By doing so, the multiplication by the tangent term is reduced to simple shift operation, which results in faster operation for hardware implementation. Besides, the cosine term in equation (4) can also be further simplified and then represented as the scaling factor term, k_i. The k_i can be treated as a constant where a set of predetermined α_i is fixed and constant in either clockwise or anti-clockwise direction. Therefore, the simplified equation is shown in equation (6).

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = k_i 1 \begin{bmatrix} \cos\phi & -\sigma_i 2^{-1} \\ \sigma_i 2^{-1} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (6)$$

where $k_i = \frac{1}{\sqrt{1+2^{-2i}}}$ and $\sigma_i \in \{-1, 1\}$.

However, the k_i needs to be compensated from equation (6) so that the equation consists of a simple shift and add operation only. To achieve this, the vector rotation is performed first by removing the k_i term followed by the scaling compensation operation upon the results after n iteration. Since the product of k_i terms converges to a constant value as the number of iterations approach infinity, the scaling compensation operation is reduced to a single step

after n iteration. The product of the reciprocal of k_i terms, K_n was computed as shown below [2].

$$K_n = \prod_{i=0}^{n-1} \frac{1}{k_i} = \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} = 1.646760258$$

As mention earlier, the rotation angle ϕ is approximated by the sum of a series of smaller micro-rotation angle α_i with a unique sequence of rotation directions denoted by σ_i. Thus, an angle accumulator is required to accumulate the micro-rotation angles at each rotation while the corresponding micro-rotation angular values are stored inside a small lookup table or hardwired. Equation (7) shows how the rotation angles perform the accumulation, where the values of $\tan^{-1}(2^{-i})$ for each iteration are stored in a lookup table.

$$z_{i+1} = z_i - \sigma_i \tan^{-1}(2^{-i}) \quad (7)$$

The vector rotations of the CORDIC algorithm are usually performed in one of two modes, which are rotation mode and vectoring mode. In the rotation mode, the input vector is rotated by a specific angle where the angle accumulator is initialized with the desired rotation angle and then aimed to reduce the angle towards zero with a suitable choice of rotation directions for each rotation. Meanwhile, in the vectoring mode, the input vector is rotated towards the x-axis through any angle where the angle accumulator is initialized with zero and then aimed to reduce the y component towards zero with a suitable choice of rotation directions for each rotation. Therefore, the rotation direction parameter σ_i can be determined by the following equation (8).

$$\sigma_i = \{sign(z_i) \quad \text{for rotation mode} \quad (8)$$

$$\sigma_i = \{-sign(y_i) \quad \text{for vectoring mode}$$

The final equation set for the basic iterative algorithm of CORDIC in circular coordinate system is summarized below:

$$x_{i+1} = k_i (x_i - \sigma_i 2^{-i} y_i) \quad (9a)$$

$$y_{i+1} = k_i (y_i + \sigma_i 2^{-i} x_i) \quad (9b)$$

$$z_{i+1} = z_i - \sigma_i \alpha_i \quad (9c)$$

where

$$\sigma_i = \{sign(z_i) \quad \text{for rotation mode}$$

$$\sigma_i = \{-sign(y_i) \quad \text{for vectoring mode}$$

$$\alpha_i = \tan^{-1}(2^{-i})$$

After n iterations, the equations are different based on the mode of operation as shown below:

For rotation mode,

$$x_n = K_n (x_0 \cos z_0 - y_0 \sin z_0) \quad (10a)$$

$$y_n = K_n (y_0 \cos z_0 - x_0 \sin z_0) \quad (10b)$$

$$z_n = 0 \quad (10c)$$

For vectoring mode,

$$x_n = K_n \sqrt{(x_0^2 + y_0^2)} \quad (11a)$$

$$y_n = 0 \quad (11b)$$

$$z_n = z_0 + \tan^{-1}\left(\frac{y_0}{x_0}\right) \quad (11c)$$

Due to the iterative nature of the CORDIC algorithm, the iterative computations for one rotation must be done first before proceed to the next rotation [3] in order to prevent concurrent computations. Therefore, this algorithm suffers from latency and speed bottlenecks.

Pipelining is a powerful method to increase the performance or speed of a system by reducing the critical computational path delay. The basic concept for this method is to overlap the processing of several tasks so that more tasks can be accomplished in the same amount of time, and then increase the throughput of the system. The throughput referred to the shortest possible time interval between subsequent pipeline computations. Deprettere et al. [4] has first suggested a pipelined architecture for CORDIC algorithm to benefit the execution speed in array processing. In addition, the pipelined based CORDIC has been used for several signals and graphical processing applications such as digital filters [5], linear transformations [6], waveform generators [7], and 3D graphics [8].

In order to pipeline the hardware architecture of the basic CORDIC, the register needs to be inserted into the data-path for each partitioned computational stages (iterations) [9]. By pipelining, the time and area consuming barrel shifters (which used in basic CORDIC architecture) can be replaced by hardwired shifters since the shift numbers are fixed for each stage. A ROM to store micro-rotation angles can also be eliminated since the corresponding micro-rotation angles values can be hardwired for each stage. Therefore, the pipelining improves the throughput by a factor of N but increases the hardware utilization by a factor less than N, where N is the total number of iterations [10].

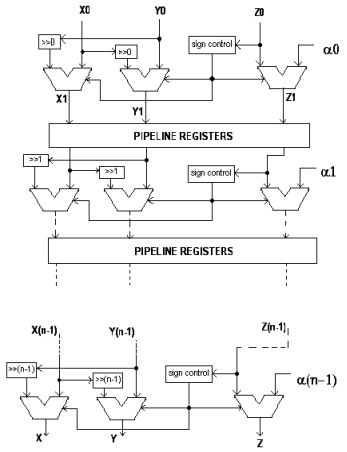

Fig (2) Pipelined CORDIC Architecture [8]

2. PIPELINED FLOATING-POINT CORDIC COPROCESSOR

In this research, the NIOS II-based embedded SoC is implemented on Altera DE0 board and operated in 50MHz. The system was developed in both hardware and software parts [4] so that it can run the hardware on the NIOS II soft processor for the purpose of verification and performance analysis [11].

First, a custom hardware component that consists of all the required Avalon-MM interface signals needs to be developed in order to interface with the NIOS II system. This custom hardware component will be created as a new component using the Qsys tool to make it executable and accessible by

the NIOS II soft processor. The functional block diagram of the custom component is shown in Figure (3).

Fig (3) The functional block diagram of the custom component

The Qsys tool of Altera Quartus II software is employed to establish an SoC design that connects all the required hardware on the Altera DE0 board through Qsys interconnect, such as a NIOS II processor, an SDRAM memory unit, a custom hardware component, an interval timer, and a system ID peripheral. The block diagram of the completed NIOS II-based embedded SoC for the design is illustrated in Figure (4).

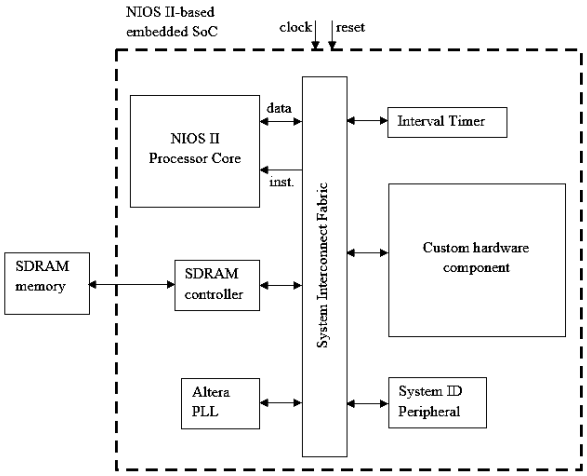

Fig (4) The block diagram of the completed NIOS II-based embedded SoC

Once the SoC design for Qsys system is completed, the corresponding HDL source file and SOPC information file can be generated by Qsys tool. Then, the SOPC information file will be used by the NIOS II Eclipse software to start the software development.

3. RESULTS AND DISCUSSIONS

The functionality of the developed coprocessor can be tested and verified using the NIOS II based embedded SoC. The result from the coprocessor is compared with the result of the NIOS II software math functions. Both results are displayed on the console window of NIOS II Eclipse software. The output results for 13 computable functions that were obtained from the hardware and software execution on the NIOS II soft processor are collected and tabulated in Table (1).

Based on the Table (1), all the output results from the hardware and software executions are almost the same and achieved the precision up to 6 decimal places. Therefore, the functionality of the developed hardware coprocessor design is fully verified.

In addition, the allowable input domain for the developed coprocessor is also tabulated in Table (2) to show the effect of the unified argument reduction algorithm to the convergence domain of the proposed design. As discussed earlier, the convergence domain is 1.74 for circular mode, 1.12 for hyperbolic mode and 1 for linear mode. By employing the unified argument reduction algorithm, the convergence domain for these functions has been successfully expanded into entire function domain except for hyperbolic function, where its domain is slightly expanded.

Table (1) The output results obtained by hardware and software execution for different functions to be evaluated

Functions to be evaluated	Input values	Software Result (using C software math library)	Hardware Result (using proposed coprocessor)
cos(z)	0.785398	0.707107	0.707107
	1.745329	-0.173648	-0.173648
	3.839724	-0.766045	-0.766044
	5.759587	0.866026	0.866026
sin(z)	0.785398	0.707107	0.707107
	1.745329	0.984808	0.984808
	3.839724	-0.642787	-0.642788
	5.759587	-0.500000	-0.500000
arctan(y)	0.562578	0.512449	0.512449
	-13.679456	-1.497824	-1.497823
	278.945678	1.567211	1.567211
	-7867.8948	-1.570669	-1.570669
arccos(m)	0.945623	0.331292	0.331292
	0.746982	0.727285	0.727285
	0.538921	1.001641	1.001641
	0.247789	1.320399	1.320399
arcsin(m)	0.945623	1.239505	1.239504
	0.469423	0.488637	0.488637
	-0.376981	-0.386534	-0.386534
	-0.759988	-0.863295	-0.863294
cosh(z)	0.457824	1.106645	1.106645

Table (2) The allowable input domain for the proposed design

Functions	Actual function domain	Domain for the proposed design
cos(z)	[-∞, +∞]	[0, 2π]
sin(z)	[-∞, +∞]	[0, 2π]
arctan(y)	[-∞, +∞]	[-∞, +∞]
acos(m)	[-1, 1]	[-1, 1]
asin(m)	[-1, 1]	[-1, 1]
cosh(z)	[-∞, +∞]	[0, 2.77]
sinh(z)	[-∞, +∞]	[0, 2.77]
exp(z)	[-∞, +∞]	[0, 2.77]
arctanh(y)	[-1, 1]	[-1, 1]
logn(w)	[0, +∞]	[0, +∞]
sqrt(w)	[0, +∞]	[0, +∞]
x*z	[-∞, +∞]	[-∞, +∞]
y/x	[-∞, +∞]	[-∞, +∞]

By using the timestamp timer function, the time elapsed for hardware and software executions can be captured. The average execution time in microseconds (us) from 10 samples and the corresponding approximate speedup are calculated and tabulated in Table (3).

Table (3) The average execution time from 10 samples for hardware and software and the approximate speedup achieved

Functions to be analyzed	The average execution time for software (us)	The average execution time for hardware (us)	Approximate speedup achieved
cos(z)	16621.6	35.9	463
sin(z)	16125.9	35.9	449
arctan(y)	22392.4	37.4	599
acos(m)	15600.2	54.6	286
asin(m)	20259.4	54.7	370
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	3
x*z	425.0	37.0	11
y/x	130.7	37.0	4

Based on Table (2), the developed coprocessor had significantly accelerated the computation for most of the functions when running on NIOS II soft processor. However, there is only a small speedup achieved for the square root, multiplication and division operations since the average execution time is relatively faster than the other computable functions of the CORDIC algorithm.

To further investigate the performance of the developed coprocessor in DSP application, the coprocessor is used to compute one dimensional 8-points DCT algorithm and its NIOS II console output.

```
8-point DCT computation

Generated 8-point input values for DCT
0.7500000000
0.7225341797
0.6621093750
0.5655029416
0.4312500060
0.2596435547
0.0527343750
-0.1856689453

After DCT for SW:
1.1519142389
0.8621989489
-0.2277814299
0.0889946744
-0.0480611660
0.0265222322
-0.0150733842
0.0066919546
The time for SW execution is 1302198.9 us

After DCT for HW:
1.1519142389
0.8621989489
-0.2277814299
0.0889946744
-0.0480611660
0.0265222322
-0.0150733842
0.0066919546
The time for SW execution is 2354.2 us

Speedup achieved (SW to HW) = 553.1
```

Fig (5) NIOS II console output for 8-point DCT computation

As shown in Figure (5), the output results after DCT operation for hardware and software executions are the same. The software execution took 1302198.9 µs to complete the computation process while the hardware execution only took 2354.2 µs. It shows that the developed hardware coprocessor has accelerated the performance of the DCT computation with the approximate speedup of 553.1 times from software execution on NIOS II soft processor. Since the DCT algorithm consists of sine and cosine functions and can be written in CORDIC-liked form, therefore the developed coprocessor can solve it efficiently. There are other CORDIC-liked algorithms that can be solved by the coprocessor such as Fast Fourier Transform (FFT) and Singular Value Decomposition (SVD), which is left for future works.

4. CONCLUSIONS

Based on the simulation and the results obtained from the NIOS II based embedded SoC, the developed coprocessor has successfully solved 13 computable functions, which include cosine, sine, arc-tangent, arc-cosine, arc-sine, hyperbolic cosine, hyperbolic sine, hyperbolic arc-tangent, exponential, logarithmic, square root, multiplication, and division. Thus, it is able to produce precise results with the precision up to 6 decimal places. Furthermore, it also achieved more than 100 times of speedup for most of the analyzed functions when comparing software and hardware executions. Although there was a slightly small speedup for square-root, multiplication and division operations, it is sufficient for most of the applications. Due to the reconfigurable architecture of the developed coprocessor, it is also applicable for other applications such as DSP and image processing.

As for the DCT computation, the developed coprocessor computed the DCT algorithm in a more efficient way since the functions can be derived into CORDIC-liked form. The coprocessor successfully performed the DCT computation with approximately 553 times faster than the software computation. This result shows that the coprocessor has the ability to perform in a more advanced application such as image compression.

It can be concluded that a pipelined 32-bit single-precision floating-point CORDIC coprocessor had been successfully designed for reconfigurable computing on FPGA and has achieved good performance for several complex arithmetic functions. In addition, the limitations of the CORDIC algorithm are also reduced. The developed coprocessor is applicable for the computation of other algorithms that can be derived into CORDIC- liked form.

5. ACKNOWLEDGEMENT

The authors would like to acknowledge the Ministry of Education and Universiti Teknologi Malaysia for the research grant (GUP 14J98).

6. REFERENCE

[1] Murthy, H.N.S. and Roopa, M. FPGA Implementation of Sine and Cosine Generators using CORDIC Algorithm. *International Journal of Innovative Technology and Exploring Engineering*, November 2012, Vol. 1, Issue 6, 16-19.

[2] Sharat, K., Uma, B.V. and Sagar, D.M. Calculation of Sine and Cosine of an Angle using the CORDIC

Algorithm. *International Journal of Innovative Technology and Research*, March 2014, Vol. 2, Issue 2, 891-895.

[3] Meher, P. K. *et. al.* 50 Years of CORDIC: Algorithms, Architectures and Applications. *IEEE Transactions on Circuits and Systems*, September 2009, 56(9): 1893-1907.

[4] Deprettere, E., Dewilde, P. and Udo, R. Pipelined CORDIC Architectures for Fast VLSI Filtering and Array Processing. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1984, 250-253.

[5] Ma, J. *et. al.* Efficient Implementations of Pipelined CORDIC Based IIR Digital Filters Using Fast Orthonormal μ-Rotations. *IEEE Transactions on Signal Processing*, September 2000, 48(9): 2712-2716.

[6] He, S. and Torkelson, M. A New Approach to Pipeline FFT Processor. *The Proceedings of 10th International Parallel Processing Symposium*, 1996, 766-770.

[7] Garcia, E., Cumplido, R. and Arias, M. Pipelined CORDIC Design on FPGA for a Digital Sine and Cosine Waves Generator. *3rd International Conference on Electrical and Electronics Engineering*, September 6-8, 2006, 154-157.

[8] Lang, T. and Antelo, E. High-Throughput CORDIC-Based Geometry Operations for 3D Computer Graphics. *IEEE Transactions on Computers*, March 2006, 54(3): 347-361.

[9] Arora, M., Chauhan, R.S. and Bagga, L. FPGA Prototyping of Hardware Implementation of CORDIC Algorithm. *International Journal of Scientific and Engineering Research*, January 2012, Vol. 3 Issue 1, 1-6.

[10] Lakshmi, B. and Dhar, A.S. CORDIC Architecture: A Survey. *VLSI Design*, 2010, vol. 2010, 1-19.

[11] M.N. Ibrahim, C.K. Tack, M. Idroas, Z. Yahya. The Implementation of a Pipelined Floating-point CORDIC Coprocessor on NIOS II Soft Processor. *International Journal of Electrical, Electronics and Data Communication*, Vol. 3, Issue 4, April 2015, 15-20.

*For correspondence; Tel. + (607) 5535260, E-mail:mdnasir@utm.my