International Journal of Advanced Science and Technology
Vol. 28, No. 8s, (2019), pp. 176-183

Processing Discrete Cosine Transform using Coordinate Rotation Digital Computer CoProcessor on Field Programmable Gate Array

*Muhammad Nasir Ibrahim[1], Mariani Idroas[2], Chen Kean Tack[3]
[1]*School of Electrical Engineering*
Faculty of Engineering, Universiti Teknologi Malaysia
Johor Bahru, Johor, Malaysia
[2]*School of Energy Engineering*
Faculty of Engineering, Universiti Teknologi Malaysia
Johor Bahru, Johor, Malaysia
[3]*School of Electrical Engineering*
Faculty of Engineering, Universiti Teknologi Malaysia
Johor Bahru, Johor, Malaysia

Abstract

With the rapid growth of digital signal processing (DSP) applications, there is a high demand for efficient implementation of complex arithmetic operations. In last five decades, a Coordinate Rotation Digital Computer (CORDIC) algorithm has been widely adopted to formulate and implement a variety of DSP algorithms for reconfigurable computing. In this paper, a CORDIC coprocessor was implemented on Field Programmable Gate Array (FPGA), to accelerate the performance of several arithmetic computations such as multiplication and division, as well as 11 elementary transcendental functions. As CORDIC algorithm suffers from limitations for its convergence domain and speed, the unified argument reduction algorithm and the hybrid angle method were adopted. The coprocessor was integrated into NIOS II soft processor to develop a NIOS II-based embedded System-on-Chip (SoC), designed on Altera DE0 board running at 50MHz of clock frequency. The experimental results showed the performance improvement of approximately 553 times was achieved while executing one dimensional Discrete Cosine Transform (DCT) algorithm using the developed coprocessor.

Keywords—*cordic; fpga; coprocessor; algorithm; DSP*

I. INTRODUCTION

Due to rapid advancement of VLSI technology, current research has been directed to the high demanding real-time compute-intensive applications such as digital signal processing (DSP), graphical processing, communication and so forth. Meanwhile, the performance and efficiency of computers for these applications have been continuously developed in order to meet high demands in terms of speed and accuracy [1].

Traditionally, there are two widely used technologies to perform the computations for high demanding tasks, which are Application Specific Integrated Circuit (ASIC) and microprocessors [2][4]. Since the ASIC was designed merely to perform specific computations, it can achieve very high efficiency for that particular task [2]. However, its circuit cannot be altered after fabrication and it is not reconfigurable. Meanwhile, the microprocessor-based system provides a more flexible way to perform these tasks by a set of software instructions [2]. However, its hardware circuit is also not reconfigurable and the software solutions that optimized for the microprocessor are compute-intensive, which result in the failure to meet the performance requirements for these tasks [3]. Therefore, reconfigurable computing is emerging as another essential solution to attain higher performance than software solution while retaining a higher level of flexibility than hardware solution [2].

Reconfigurable computing requires reconfigurable hardware such as Field Programmable Gate Array (FPGA) [4] to perform a set of computations. In early stage, due to the limited FPGA density, the implementation for accelerating high demanding tasks that involves complex floating-point operations is not recommended for reconfigurable computing [2]. However, with the rapid increase in FPGA density, reconfigurable computing now becomes an efficient solution. Moreover, current technology of FPGA is extendable to realize a System-on-Chip (SoC) based embedded system by integrating the hardware partition (FPGA-based hardware) and software partition (C software of soft processor) using hardware/software co-design approach [5]. Thus, this system was adopted to increase the flexibility of high demanding real-time implementation and analysis.

In modern DSP and graphical applications, most of the algorithms are complex, which involves the computations of elementary transcendental functions such as trigonometric, hyperbolic, square-root, exponential and logarithmic. Tthere are four principal methods that can be used to implement the elementary functions, which are table lookup method, polynomial approximation, rational approximation and quadratic convergence method [6]. However, these methods are not suitable for high demanding reconfigurable computing since they require expensive hardware organization to realize the complex algorithms.

Therefore, another approach known as CORDIC algorithm is used. The CORDIC algorithm provides high efficiency and low cost solutions to compute elementary functions for reconfigurable computing [6][7][8], which has been utilized for applications in various fields in last five decade such as linear transformations [9], digital filters [9], matrix computations [9], neural networks [10], biomedical signal processing [11], kinematic computations [12], three-dimensional graphics [13] and others. This algorithm basically is an iterative shift-and-add algorithm that involves the two dimensional vector rotations either in circular, linear or hyperbolic coordinate systems, to realize the solution for several elementary functions by using the same hardware [7]. Historically, this algorithm was first described by Volder[14] in 1959 to compute only the trigonometric functions and then reformulated by Walther [15] in 1971 to be a unified algorithm by merely varying a few parameters to solve more elementary functions such as natural logarithms, exponentials, square-roots, trigonometric and hyperbolic functions as well as elementary operations such as multiplication and division.

Meanwhile, the CORDIC algorithm was commonly developed in fixed-point format to achieve high speed computation in the past. However, for practical applications

especially DSP tasks that require elementary transcendental functions, the floating-point format is preferred since it has a dynamic range and accuracy. Therefore, several works has been done to implement CORDIC design in floating-point format that comfort with IEEE-754 standard [16]. However, these implementations represent the input and output data as floating-point format but perform all internal computations in block floating-point format (internal fixed point format for mantissa) in order to achieve higher speed by reducing the overhead incurred for floating-point operations [8].

Although the CORDIC algorithm has a powerful solving capability but it suffers from many limitations. First, the execution speed is slow due to its iterative nature [17]. As the number of iterations increases, the execution speed decreases and the latency increases. However, due to the linear-rate convergence property, the number of iterations should be at least equal to N to yield the precision of N bit for the output results [7]. Thus, the execution speed is constrained by the precision requirement [7]. Second, the range of the convergence domain is limited that affects the accuracy of the results [17]. In this case, the input value must be set within the restricted range to obtain the correct results. Third, the requirement of scale factor compensation [17] incurs additional overhead during CORDIC computation that degrades the system performance. Therefore, these limitations have led to further investigations to improve the CORDIC algorithm based on the requirements of the applications.

Besides, there are several researches have been carried out to implement the floating-point CORDIC coprocessor using FPGA. However, many researchers [18][19][20] did not integrate their CORDIC coprocessor into NIOS II soft processor to develop a SoC-based embedded system for real-time analysis [21]. Anyway, there are still a few researchers [22][23] have reported this kinds of works but whether it merely support for limited solving capability or it neglect the limitations of the CORDIC algorithm. Therefore, this research was carried out to fill the gaps of the previous research works.

II. SOFTWARE DEVELOPMENT

The block diagram of the completed NIOS II-basedembedded SoC for the coprocessor is illustrated in Figure 1.

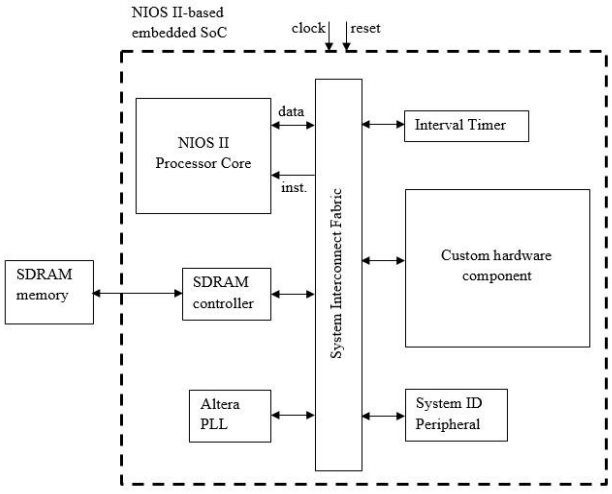

Figure 1. The block diagram of the completed NIOS II-based embedded SoC

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 NIOS II Eclipse software to start the software development.

To develop the software of NIOS II soft processor, the NIOS II Eclipse software is employed. In this research, the main function of the NIOS II software is to perform test, verification and performance analysis for the hardware coprocessor. Thus, the computable functions that are verified by the NIOS software include cosine, sine, arc-tangent, arc-cosine, arc-sine, hyperbolic cosine, hyperbolic sine, exponential, hyperbolic arc-tangent, natural logarithm, square root, multiplication and division. In addition, the 8-point one-dimensional DCT algorithm is also developed for both hardware and software function to analyze the performance of this algorithm with and without the presence of the developed coprocessor.

First of all, the subroutines of hardware execution for all the computable functions are coded in C language. Then, the functionality of these computable functions is verified by comparing the results calculated by hardware execution subroutine with the results calculated by the C software built-in math library. Therefore, the design is verified if the results for hardware and software execution are almost the same.

Secondly, the performance analysis can be done by comparing the performance achieved by the hardware execution with the C software execution and then analyze it. The time stamp timer function is used to capture the time taken for the specified executed operations on the NIOS II soft processor. By knowing the time taken for the hardware and software execution, the speedup achieved for the hardware execution from software execution can be determined based on the following equation.

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

Thirdly, the NIOS II software is used to execute the 8-point one- dimensional Discrete Cosine Transform (DCT) computations and then compare the performance of hardware with software execution to determine the speedup achieved. The 8-point one-dimensional DCT algorithm is shown below:

$$C(u) = \frac{1}{2}\alpha(u)\sum_{x=0}^{7} f(x)\cos\frac{(2x+1)u\pi}{16} \qquad (1)$$

where $\alpha(u) = \frac{1}{\sqrt{2}} \quad if\ u = 0$

$\alpha(u) = 1 \quad if\ u \neq 0$

Therefore, the subroutines for DCT computation by hardware and software are coded in C based on the equation (1). For software subroutine, the equation (1) is converted into pseudo-code as shown in Figure 2 and then written in C program.

```
Set f as input parameters with 8 arrays
Set C as output parameters with 8 arrays
Initial x and u to 0

For (x=0; x<8; x++) C[0] = C[0] + 0.5*(         )*f[x];
For (u=1; u<8; u++) begin
            Initial C[u] to 0
            For (x=0; x<8; x++) C[u] = C[u] + 0.5*f[x]*cos(π*u/8*(x+0.5))
end
```

Figure 2. The pseudo-code for DCT software subroutine

Meanwhile, for hardware subroutine, the 8-point DCT computation is calculated separately with the introduction of the proposed coprocessor. Thus, the equations for 8-

point output can be rewritten in CORDIC-liked form as shown in the following Equation (2):

Segment 1:

$$C(0) = [f(0) + f(7) + f(3) + f(4)] \cos(\pi/4) + [f(1) + f(6) + f(2) + f(5)] \sin(\pi/4)$$
$$C(4) = [f(0) + f(7) + f(3) + f(4)] \sin(\pi/4) - [f(1) + f(6) + f(2) + f(5)] \cos(\pi/4)$$

Segment 2:

$$C(2) = [f(0) + f(7) - f(3) - f(4)] \cos(\pi/8) + [f(1) + f(6) - f(2) - f(5)] \sin(\pi/8)$$
$$C(6) = [f(0) + f(7) - f(3) - f(4)] \sin(\pi/8) - [f(1) + f(6) - f(2) - f(5)] \cos(\pi/8)$$

Segments 3 and 4:

$$C(1) = [f(0) - f(7)] \cos(\pi/16) + [f(2) - f(5)] \cos(5\pi/16) - [f(4) - f(3)] \sin(\pi/16) - [f(6) - f(1)] \sin(5\pi/16)$$
$$C(7) = [f(0) - f(7)] \sin(\pi/16) + [f(6) - f(1)] \sin(5\pi/16) + [f(4) - f(3)] \cos(\pi/16) - [f(2) - f(5)] \cos(5\pi/16)$$

Segments 5 and 6:

$$C(5) = [f(0) - f(7)] \cos(5\pi/16) + [f(6) - f(1)] \sin(9\pi/16) - [f(4) - f(3)] \sin(5\pi/16) - [f(2) - f(5)] \cos(9\pi/16)$$
$$C(3) = [f(0) - f(7)] \sin(5\pi/16) - [f(2) - f(5)] \sin(9\pi/16) + [f(4) - f(3)] \cos(5\pi/16) - [f(6) - f(1)] \cos(9\pi/16)$$

The equation (2) has been divided into six segments with different input values that can be directly solved by CORDIC algorithm in circular rotation mode. Thus, the structural diagram that shows how to obtain the DCT computation output for each point is shown in Figure 3.

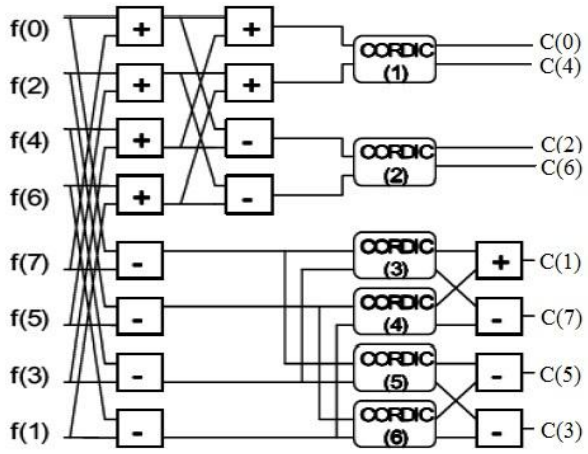

Figure 3. The structural diagram for 8-point DCT computation

Hence, the output for C(0) and C(4) can be obtained at the same time by using one instance of CORDIC coprocessor. Similarly, the output for C(2) and C(6) can be obtained by using another instance of CORDIC coprocessor. Meanwhile, the output for C(1), C(7), C(5) and C(3) can be obtained from four instance of CORDIC coprocessor.

III. Results and Discussions

To 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 is shown in Figure 4. The output results after DCT operation for hardware and software execution are the same. Thus, the software execution takes 1302198.9 us to

complete the computation process meanwhile hardware execution only takes 2354.2 us. It shows that the hardware coprocessor successfully accelerate 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 coprocessor can solve it efficiently. Furthermore, there are other CORDIC-liked algorithms that can also be solved by the coprocessor such as Fast Fourier Transform (FFT) and Singular Value Decomposition (SVD), which is left for future works.

```
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
```

Figure 4. NIOS II console output for 8-point DCT computation

IV. CONCLUSIONS

Based on the simulation and the results obtained from the developed NIOS II based embedded SoC, the coprocessor is working fine on the NIOS II soft processor to solve 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 achieves more than 100 times of speedup for most of the analyzed functions if comparing software and hardware executions. Although there is a slightly small speedup for square-root, multiplication and

division operations, but it is enough for most of the applications. Due to the reconfigurable architecture of the developed coprocessor, it is applicable for other applications such as DSP and image processing to perform the special purposes algorithm that can be derived in CORDIC-liked form.

Meanwhile, the coprocessor also reduces its limitations in terms of speed, convergence domain and scaling factor compensation. Thus, the unified argument reduction algorithm successfully expands the input range of trigonometry functions to entire coordinate space meanwhile slightly higher range for hyperbolic functions. For multiplication and division operations, the allowable input range is also unlimited. Besides that, the execution speed has been improved by hybrid angle method and pipelining architecture. However, the scaling factor compensation is done after the completion of CORDIC computation rather than before the start of CORDIC computation to avoid the redundant error after scaling.

For DCT computation, the developed coprocessor provides the CORDIC engine to compute the DCT algorithm more efficiently since the functions can be derived into CORDIC-liked form. Hence, the coprocessor successfully performs DCT computation with approximately 553 times faster than software computation. Thus, it shows that the coprocessor is suitable for DCT computation in more advances for the image compression application.

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

Acknowledgment

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

References

[1] Pongyupinpanich, S. *Optimal Design of Fixed-Point and Floating-Point Arithmetic Units for Scientific Applications*. PhD Thesis. Technische Universität Darmstadt; March 2012.
[2] Compton, K.and Hauck, S. *Reconfigurable Computing: A Survey of Systems and Software*. ASM Computing Surveys, June 2002, 34 (2): 171-210.
[3] Agarwal, A. *Trigonometric Function Generator Implementation on FPGA*. Master Thesis. Deemed University; June 2006.
[4] Todman, T.J. *et. al*. Reconfigurable computing: architectures and design methods.*IEE Proceedings – Computers and Digital Techniques*, March 2005, 152(2): 193-207.
[5] Wolf, W. H. Hardware-Software Co-Design of Embedded Systems. *Proceedings of the IEEE*, July 1994, 82(7): 967-989.
[6] Rodrigues, T. K. Adaptive CORDIC: Using Parallel Angle Recoding to Accelerate CORDIC Rotations. PhD Thesis. The University of Texas at Austin; December 2007.
[7] Meher, P. K. 50 Years of CORDIC: Algorithms, Architectures, and Applications.*IEEE Transactions on Circuits and Systems*, September 2009, 56(9): 1893-1906.
[8] Costello, J. P. Behavioural Synthesis of Low Power Floating Point CORDIC Processors. Master Thesis. Faculty of the Royal Military College of Canada; April 2000.
[9] Hen Hu, Y. CORDIC-Based VLSI Architectures for Digital Signal Processing. *IEEE Signal Processing Magazine*, July 1992, 9(3): 16-35.
[10] McLenegan, T. The CORDIC Algorithm: An Area-Efficient Technique for FPGA-based Artificial Neural Networks, Master Thesis. California Polytechnic State University; November 2006.

[11] Banerjee, A., Dhar, A. S. and Banerjee, S. FPGA realization of a CORDIC based FFT processor for biomedical signal processing. *Microprocessors and Microsystems*, May 2001, 25(3): 131-142.
[12] George Lee, C. S. A Maximum Pipelined CORDIC Architecture for Inverse Kinematic Position Computation. *IEEE Journal of Robotics and Automation*, October 1987, 3(5):445-458.
[13] Euh, J., Chittamuru, J. and Burleson, W. CORDIC Vector Interpolator for Power-aware 3D Computer Graphics. *IEEE Workshop on Signal Processing Systems*,October 16-18, 2002, 240-245.
[14] Volder, J. E.The CORDIC Trigonometric Computing Technique.*IRE Transactions on Electronic Computers*, September 1959, 8(3): 330-334.
[15] Walther, J. S.A unified algorithm for elementary functions.*AFIPS '71 (Spring) Proceedings ofSpring Joint Computer Conference*,May 18-20,1971, 38: 379-385.
[16] The Institute of Electrical and Electronic Engineers (IEEE). *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Standard-754, 1985.
[17] Maharatna, K., Troya, A., Banerjee, S. and Grass, E. Virtually scaling-free adaptive CORDIC rotator. *IEE Proceedings – Computers and Digital Techniques*, November 2004, 151(6): 448-456.
[18] Zhou, J., Dou, Y., Lei, Y. and Dong, Y. Hybrid-Mode Floating-Point FPGA CORDIC Co-processor. *4th International Workshop, ARC 2008, London, UK*, March 26-28, 2008, 256-261.
[19] Vladimirova, T., Earney, D., Keller, S. and Prof. Sir. Sweeting, M. Floating-Point Mathematical Co-Processor for a Single Chip On-board Computer. *Proceedings of the 6th Military and Aerospace Programmable Logic Devices International Conference (MAPLD)*, 2003, Paper D5.
[20] Madian, A.andAljarhi, M. A Multi Cordic Architecture on FPGA Platform. *International Journal of Electrical, Electronic Science and Engineering*, 2013, 7(12): 1264-1272.
[21] 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.
[22] Sobirin, M. I. *Hardware Implementation of Coordinate Rotation Digital Computer in Field Programmable Gate Array*. Master Thesis. UniversitiTeknologi Malaysia; January 2012.
[23] Hon, T. T. and Marsono, M. N. Hardware Design Space Exploration of Cordic Algorithm for Run-Time Reconfigurable Platform. *Proceedings of the First International Conference on Green Computing, Technology and Innovation (ICGCTI)*, Kuala Lumpur, Malaysia, March 2013, 1-7.