Floating-Point Division Operator based on CORDIC Algorithm

Pongyupinpanich Surapong[1] and **Faizal Arya Samman**[2], Non-members

ABSTRACT

Design and evaluation of a CORDIC (COordinate Rotation DIgital Computer) algorithm for a floating-point division operation is presented in this paper. In general, division operation based on CORDIC algorithm has a limitation in term of the range of inputs that can be processed by the CORDIC machine to give proper convergence and precise division operation result. A hardware architecture of CORDIC algorithm capable of processing broader input ranges is implemented and presented in this paper by using a pre-processing and a post-processing stage. The performance as well as the calculation error statistics over exhaustive sets of input tests are evaluated. The results show that the CORDIC algorithm can be well-convergence and gives precise division operation results with broader input ranges. The proposed hardware architecture is modeled in VHDL and synthesized on a CMOS standard-cell technology and a FPGA device, resulting 1 GFlops on the CMOS and 210.812 MFlops on the FPGA device.

Keywords: Floating-Point Operators, Accelerator Processor, Product-of-Sum, Sum-of-Product, 32-bit IEEE Standard Single-Precision.

1. INTRODUCTION

In modern digital computer architecture, floating-point arithmetic units have been important components to improve the performance of the digital computer. Arithmetic components such as adder/subtractor, multiplier and divider units are generally basic operators in a scientific computations beside other trigonometric functions such as sine, cosine, logarithm, exponent, etc. Compared to fixed point arithmetic units, the floating-point arithmetic units provides better accuracy and precision and it covers larger data ranges, which is suitable for scientific computations in engineering application areas. Compared to other arithmetic operator, designing a divider unit requires a special attention because of the complexity to implement the operation, especially when using floating-point data operands.

Compared to other arithmetic operator, designing a divider unit requires a special attention because of the complexity to implement the operation, especially when using floating-point data operands. The special attention opens a challenging issue for many scientists and researchers to introduce new efficient algorithms and methods to design the divider unit. [1]

Division operations are often used in many scientific computations of image and signal processing algorithms. Image convolution and Gaussian Filtering are example computations [2] that require divider operator as presented in Equ. (1) and Equ. (2), respectively. Equ. (1) is the equation of pixel output of a 3×3 convolution masks window, where w_i is the weight of the j adjacent input image pixel within the window.

$$P = \frac{\sum_{i=0}^{9} p_i w_i}{\sum_{i=0}^{9} w_i} \tag{1}$$

Another important equation with division operation is presented by the 2D Gaussian distribution as shown in Equ. (2), where x and y is the two dimensional image signal and σ is the standard deviation of the distribution. The Gaussian filtering is used to "blur" images and remove detail and noise [2].

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

In adaptive digital signal processing applications for instance, division operations are used in a normalized adaptive least-mean-square (LMS) algorithm presented in Equ. (5) to update the parameters of an adaptive filter. The filter output signal, Equ. (3), is compared to the desired system model output signal (d), resulting in an error signal, Equ. (4). This error signal is then used to drive the adaptive filter parameters, in such a way that finally the adaptive filter parameters (w_j) will be equal (almost equal in practice) to system model parameters. The signal divisor in this case $(\gamma + x(k-j)^2)$ is used to improve the stability of the adaptive parameter identification algorithm.

Manuscript received on July 2, 2012.

[1] The author is with Ramkhamhaeng University, Faculty of Engineering, Department of Computer Engineering, Ramkhamhaeng Road, Hua Mark, Bangkapi, Bangkok 10240, Thailand , E-mail: surapong@riees.org

[2] The author is with Universitas Hasanuddin at Makassar, Faculty of Engineering, Department of Electrical Engineering, Jl. Perintis Kemerdekaan Km. 10, Tamalanrea, Makassar 90245, Indonesia. , E-mail: faizalas@unhas.ac.id

$$y(k) = \sum_{j=1}^{N_{tap}} w_j(k)x(k-j), \quad \forall j \in \Psi \quad (3)$$

$$e(k) = d(k) - y(k) \quad (4)$$

$$w_j(k+1) = w_j(k) + \frac{\beta\, e(k)x(k-j)}{\gamma + x(k-j)^2}, \quad \forall j \in \Psi \quad (5)$$

2. STATE-OF-THE-ARTS OF DIVISION METHODS

In this section, we will present brief descriptions on the state-of-the-art of the methodologies or algorithms to implement the binary division operation. The methodologies are described as follows.

1. **Adder-Cell-based Method:** The design of division operator using the adder-cell-based method will always result in a very compact divider architecture. This method is classified as non iterative technique, where the divider unit consists of half-adder and full-adder cells as well as other logic gate units and supporting modules [3]. A binary divider that uses carry-save adder units is presented for example in [4].

2. **Digit Recurrence Algorithm:** In modern floating point arithmetic units the most common algorithm employed to division function is a digit recurrence algorithm [5] [6] [7]. The algorithm performs both operations based on shifting and subtraction as the fundamental operators. A combined floating-point square-root and division operation can also be implemented by using a subtractive SRT (Sweeney, Robertson and Tocher) algorithm [8], which can be classified as a digit recurrence algorithm. The subtractive SRT algorithm can be extended by using Radix-8 IDS (Interleaved Digit Set) algorithm to improve the performance of the traditional digit recurrence algorithm. Another variant of the digit-recurrence method is svoboda algorithm. A new Svoboda-Tung Division algorithm is for instance proposed in [9].

3. **Taylor's Series Expansion Algorithm:** A Taylor's Series Expansion Algorithm [10] for example can be used to calculate division operation using a sequential series of a harmonic equation. However, the Taylor's Series Expansion algorithm is rarely used and perform slow computation to calculate the division operations.

4. **Goldschmidt's Algorithm:** The basic idea behind the Goldschmidt's Algorithm is the iterative parallel multiplication of the dividend and divisor by updated factors in such as a way that the final divisor will be driven to one. Thus, the final dividend gives the quotient (the division result). Oberman et al. for example [11] proposes a floating-point divider and square root for AMD-K7 by using Goldschmidt's algorithm. The Goldschmidt's algorithm has been broadly used on many commercial microprocessors and is also known as division by multiplicative normalization or division by convergence [12]. The disadvantage of the Goldschmidt's algorithm in term of the area overhead is the need for two independent parallel multiplication. As we know, a multiplier requires large number of logic area, especially when it is implemented in floating-point arithmetic.

5. **Newton-Raphson Algorithm:** The Newton-Raphson division algorithm is almost similar with the Goldschmidt's algorithm. In the Newton-Raphson method however, the iterative refinement is applied only to the reciprocal value of the divisor, which will be convergent after several iterations [13]. The division operation of the Newton-Raphson method can be divided into three steps, i.e. the initial estimation of the divisor's reciprocal, the iterative refinement of the divisor's reciprocal and the multiplication step between the divided and the final convergent divisor's reciprocal. The work in [14] has presented for example a decimal floating-point divider using newton-raphson iteration, where an accurate piece-wise linear approximation is used to obtain an initial estimate of a divisor's reciprocal.

6. **CORDIC Algorithm:** Beside the aforementioned method to implement the division operation, there is also another powerful algorithm to implement the divider unit called CORDIC (COrdinate Rotation DIgital Computer) algorithm [15]. Like digit recurrence method, CORDIC is also classified into iterative method. The main powerful characteristic of the CORDIC algorithm is the capability to implement several trigonometric function [16], [15], phase and magnitude functions [17], and hyperbolic functions [18] as well as linear operational function such as multiplication and division functions.

By using CORDIC algorithms, we can also easily implement all the function in a single CORDIC hardware architecture [19]. Some basic standard and non-standard operators such as sum-of-product and product-of-sum [20], which can be used to accelerate floating-point operations, can also be implemented by using CORDIC algorithm. The work in [21] presents for example a flexible FPGA implementation of a parameterizable floating-point library allowing to compute the sine, cosine or arctangent functions. The CORDIC algorithm can also even be used to solve problematic operation in a fuzzy logic controller circuit [22]. Moreover, the CORDIC algorithm can be used to implement a unified frequency analysis or transformations functions such as DFT (Discrete-Fourier Transform), DHT (Discrete Hartley Transform), DCT (Discrete Cosine Transform) and DST (Discrete Sine Transform) [23].

There are two main issues, in which CORDIC algorithm is preferable to design of the floating-point division operator i.e.,

1. **The benefit of CORDIC Algorithm:** The CORDIC algorithm provides advantages in the performability of fundamental function for scientific and

engineering, the low algorithmic complexity, and the simplicity for VLSI implementation. The VLSI implementation is simple because it uses only shift and add operations. The area cost of the CORDIC algorithm is certainly lower than the Newton-Raphson Algorithm, Goldschmidt's Algorithm and Taylor's Series Expansion Algorithm, which require multiply and add operations. Compared to the Digit-Recurrence algorithm, CORDIC algorithm is slightly simpler.
2. **Design alternative:** From the previous works, there is few research dealing with design and investigation of a divider based on CORDIC algorithm. Therefore, this paper proposes the algorithm, design, and architecture of the floating-point divider based on CORDIC algorithm and the analysis and investigation on its error characteristics.

The CORDIC algorithm is however lack in computational latency and convergence range. These disadvantage can be alleviated by the techniques proposed in [15]. The convergence of the CORDIC algorithm can be accelerated by duplicate and triplicate the micro (angle) rotation on each stage of the CORDIC iterative algorithm.

The rest of the paper is organized as follows. Section 3shows the architecture of a CORDIC algorithm especially used for division operations. The main reasons why we propose a modification of CORDIC algorithm are also presented in this Section. Section 4. describes the performance analysis of the CORDIC divider by giving a wide range of inputs. The convergence and calculation errors are presented in this section. Synthesis results of the CORDIC divider core modeled in VHDL by using 130-nm CMOS standard-cell technology and by using an FPGA device are presented in this section.

3. CORDIC-BASED DIVIDER ARCHITECTURE

There are two basic reasons why a new CORDIC algorithm is proposed in this paper.
1. The CORDIC algorithm gives correct convergence when the expected division results are located in the following ranges: $-0.9647 \leq Q = \frac{Y}{X} \leq 2.9647$ (based on our Matlab simulation results). The values outside the range will tend to saturate at unexpected division operation results. Please see the experimental result shown in Fig. 1.
2. We cannot identify, whether the division results are in the aforementioned range or not, unless the division has been made.

Now let us see the divergence problem as presented in Fig. 1, which is obtained from our simple experimental. Two curves are presented in the figure, i.e. the ideal (MATLAB) output and the division results using CORDIC algorithm. The figure presents the θ-axis and the Y/X-ordinat, where $Y = sin\ \theta$ and $X = cos\ \theta$, θ in radian unit. The simulation results present that the division operations cannot move to a

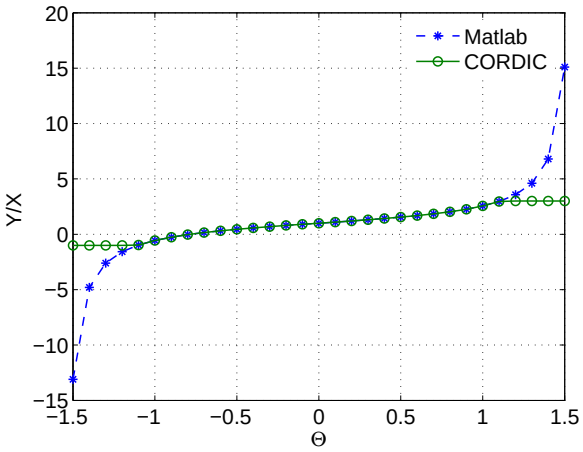

Fig.1:: The CORDIC divergence problem.

convergence value, when the division results are outside the following ranges, i.e. $-0.9647 \leq \frac{Y}{X} \leq 2.9647$.

Based on the above mentioned facts, we propose a solution to improve the range of inputs that can guarantee the precision and the convergence of the CORDIC algorithm to perform a floating-point division operation. Firstly, we will introduce the basic of the CORDIC algorithm as described in the following.

3.1 Basic Radix-2 CORDIC Division Algorithm

The basic radix-2 CORDIC iteration algorithm can be written as follows [19].

$$\begin{aligned} X_{i+1} &= X_i - m \cdot \mu_i \cdot Y_i \cdot \delta_{m,i} \\ Y_{i+1} &= Y_i + \mu_i \cdot X_i \cdot \delta_{m,i} \\ Z_{i+1} &= Z_i - \mu_i \cdot \delta_{m,i} \end{aligned} \tag{6}$$

The CORDIC implementation for Divider function can be performed by configuring the CORDIC algorithm into linear mode of operation (linear modus), i.e. by setting $m = 0$ in the Equ. (6) and $\delta_{m,i} = 2^{-i}$. Hence, the CORDIC iterative equation for the Division operator is shown in Equ. (7).

$$\begin{aligned} X_{i+1} &= X_i \\ Y_{i+1} &= Y_i + \mu_i \cdot X_i \cdot 2^{-i} \\ Z_{i+1} &= Z_i - \mu_i \cdot 2^{-i} \end{aligned} \tag{7}$$

The set of μ_i values is $\mu_i = \{-1, +1\}$, which depends on the value of Y_i as shown in Equ. (8).

$$\mu_i = \begin{cases} +1 & : Y_i < 0 \\ -1 & : Y_i \geq 0 \end{cases} \tag{8}$$

3.2 The Proposed CORDIC Division Algorithm

Based on our experience, the domain of well-convergence of the CORDIC inversion function is

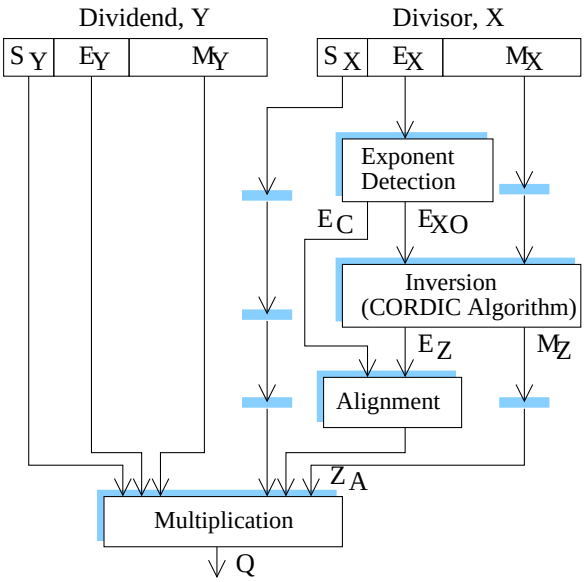

Fig.2:: The floating-point divider architecture.

shown in Equ. (9). If the domain is written in the IEEE binary floating-point standard, the domain can be described in Equ. (10) as well as Equ. (11) and Equ. (12) present the exponent and mantissa faction.

$$0.5 \leq X < 1.5 \tag{9}$$

$$1.0 \times 2^{126} \leq X < 1.5 \times 2^{127} \tag{10}$$

$$1.0 \leq M_X \leq 1.5 \tag{11}$$

$$126 \leq E_X \leq 127 \tag{12}$$

The complete division operation $Q = \frac{Y}{X}$ and its hardware architecture are proposed in this paper, where Q is the division operation result, Y is the dividend and X is the divisor, The hardware architecture is classified into four main stages as presented in Fig. 2. The description of the floating-point hardware architecture is described as follows.

1. *Divisor's Exponent Detection:* In this first stage, a credit exponent E_C and e new exponent E_{X0} for X are computed by using Alg. 1.

2. *Divisor Inversion:* In this second stage the inverse value of the new input divisor $X_{new} = (-1)^S_X \times M_X \times 2^{E_{X0}}$ is computed by using CORDIC algorithm presented in Alg. 2 to obtain the variable Z.

3. *Alignment:* In this third stage, Z_A is computed from the Z variable (computed from Alg. 2) whose exponent is aligned by using the credit exponent E_C (computed from Alg. 1). By using a formal floating point equation, then we have $Z_A = (-1)^{S_Z} \times 1.M_Z \times 2^{E_Z - E_C}$, where $S_Z = S_X$.

4. *Multiplication:* Finally we will have the complete division result as $Q = Y \times Z_A$.

Example: If we have decimal numbers $X_d = 180$ and $Y_d = 500$ then the CORDIC divider should give

Algorithm 1 $[E_C, E_{X0}]$=DivisorDetect(X), where $X \geq 1$

1: Yo=1, Zo=0, Xo=X, S_0=-1 {Initialization}
2: E_X = Exponent of the Input Divisor X
3: **if** $E_X < 126$ **then**
4: $E_C = 126 - E_X$
5: $E_{X0} = 126$
6: **else if** $E_X > 127$ **then**
7: $E_C = E_X - 127$
8: $E_{X0} = 127$
9: **else**
10: $E_C = 0$
11: $E_{X0} = E_X$
12: **end if**
13: return E_C, E_{X0}

Algorithm 2 Z=Inverting(X, I), where $X \geq 1$

1: Y_0=1, Z_0=0, X_0=X_{new}, S_0=-1 {Initialization}
2: **for** $i = 0$ to $I - 1$ **do**
3: $X_{i+1} = X_i$
4: $Y_{i+1} = Y_i + (X_i \times S_i \times 2^{-i})$
5: $Z_{i+1} = Z_i - (S_i \times 2^{-i})$
6: **if** Y < 0 **then**
7: S_{i+1}=+1
8: **else**
9: S_{i+1}=-1
10: **end if**
11: **end for**
12: return Z

$Q_d = \frac{Y_d}{X_d} = 2.77778$. The floating point formats of the input signals are $X_{fp} = 1.40625 \times 2^{134}$, where $S_X = 0$, $M_X = 1.40625$ and $E_X = 134$, meanwhile $Y_d = 1.38889 \times 2^{128}$, where $S_Y = 0$, $M_Y = 1.38889$ and $E_Y = 128$. The step-by-step computations made by using our proposed algorithm is as follows:

1. *Divisor's Exponent Detection:* By using Alg. 1, the credit exponent is obtained as $E_C = E_X - 127 = 134 - 127 = 7$.

2. *Divisor Inversion:* By using Alg. 2, inversion result gives $Z = \frac{1}{X_{new}} = \frac{1}{1.40625 \times 2^{127}} = 1.42222 \times 2^{126}$.

3. *Alignment:* In this third stage, we have $Z_A = 1.42222 \times 2^{E_Z + E_C} = 1.42222 \times 2^{126-7} = 1.42222 \times 2^{119}$.

4. *Multiplication:* Finally the division result is $Q = Y \times Z_A = 1.9531250 \times 1.42222 \times 2^{135+119-127} = 2.77778 \times 2^{127}$ or according to the normalized IEEE binary floating-point standard, $Q = 1.38889 \times 2^{128}$ as expected.

4. PERFORMANCE ANALYSIS

The CORDIC Division function has been evaluated by feeding input signals ranging for low magnitude until high magnitude. Fig. 3 shows how the Divider Hardware unit gives output signals Q by inverting input signals X ($Q = \frac{1}{X}$, where Dividend $Y1$ is set 1). The input signals are incremented linearly from $X = 0.1$ until $X = 33$. The figure shows also the comparison of the Divider Hardware and Matlab Output (Real calculated output) as presented in the upper diagram of Fig. 3. The absolute calculation errors are shown in the bottom diagram of Fig. 3.

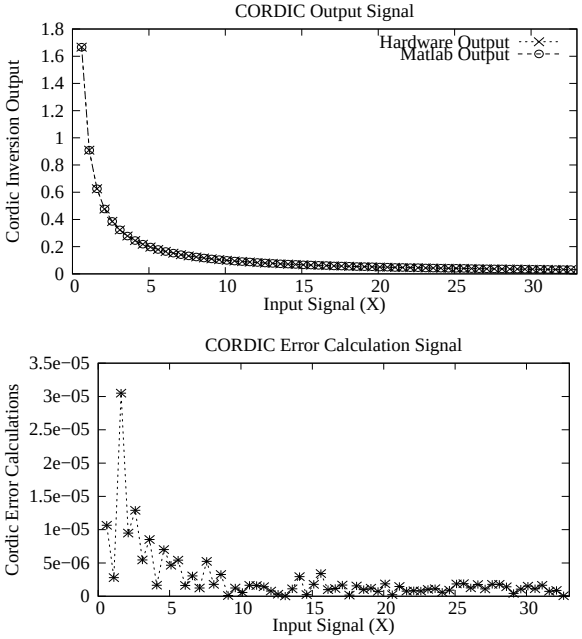

Fig.3:: Measurements of the CORDIC Division with unit Dividend ($Q = \frac{1}{X}$) and the calculation errors by testing of the 33 sets of inputs X.

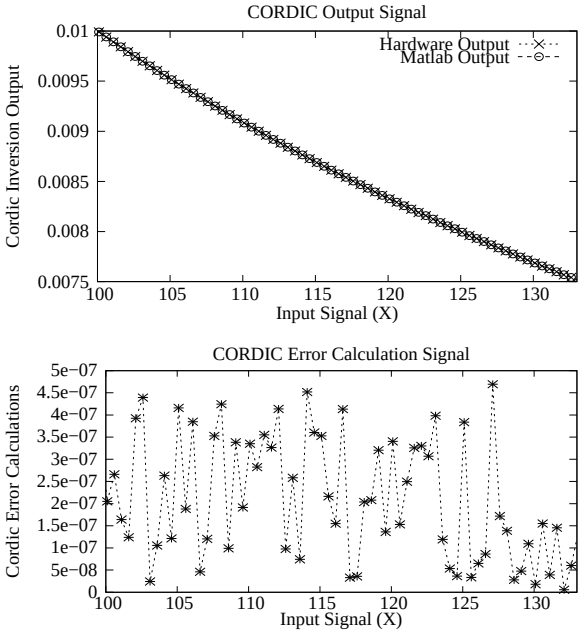

Fig.4:: Measurements of the CORDIC Division with unit Dividend ($Q = \frac{1}{X}$) and the calculation errors with another testing of the 133 sets of input X.

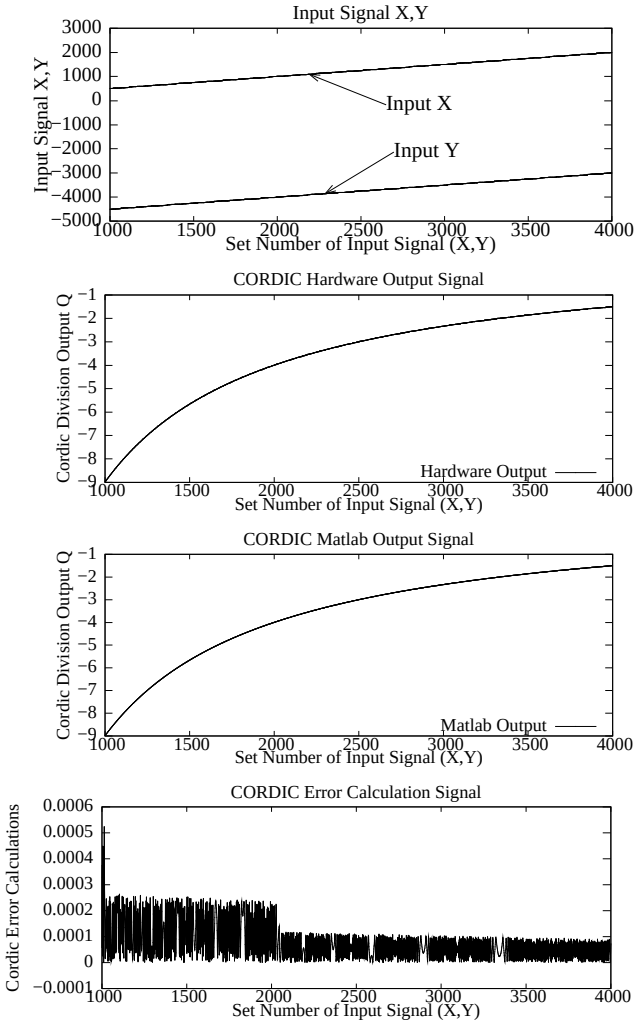

Fig.5:: Evaluation of the CORDIC Division operation ($Q = \frac{Y}{X}$) and the calculation errors by incrementing both input operands.

Fig. 4 presents also another simulation result with different input ranges, i.e. from $X = 100$ until $X = 133$. The simulation results shown in Fig. 3 and Fig. 4 are presented in this case to show the evaluation result of the inversion step in the CORDIC-based division operation. The evaluation of the division operations itself will be presented in the next figure.

Fig. 5 presents a simulations result of division operations, i.e. $Q = \frac{Y}{X}$, where the input signal X as the divisor and input signal Y as the dividend are shown in the first and second lines of the diagrams shown in the figure. Both input signals are incremented to evaluate the CORDIC division operations in different input ranges. The third and the fourth lines of the diagrams are the calculation results of the floating-point-based CORDIC Divider implemented using VHDL model and the ideal Matlab output, respectively. The diagram in the bottom line is the absolute calculation errors between the CORDIC Hardware and the Matlab output.

Another simulation result is presented in Fig. 6

Table 1:: Statistical error of CORDIC-based and traditional inverse calculation algorithm varied, where X and Y are varied from 0.1 to 500.

Iter.	Max.	Min.	Abs. Mean	Std. Deviation
8	3.9030	-15.6220	3.6760×10^{-2}	0.2999
16	0.0152	-0.0610	1.5841×10^{-4}	1.4705×10^{-3}
32	1.0×10^{-4}	-4.4680×10^{-5}	6.1798×10^{-7}	3.3635×10^{-6}
64	1.0×10^{-4}	-4.4680×10^{-5}	6.1798×10^{-7}	3.3635×10^{-6}
128	1.0694×10^{-6}	-8.0123×10^{-7}	1.1888×10^{-7}	1.9968×10^{-7}

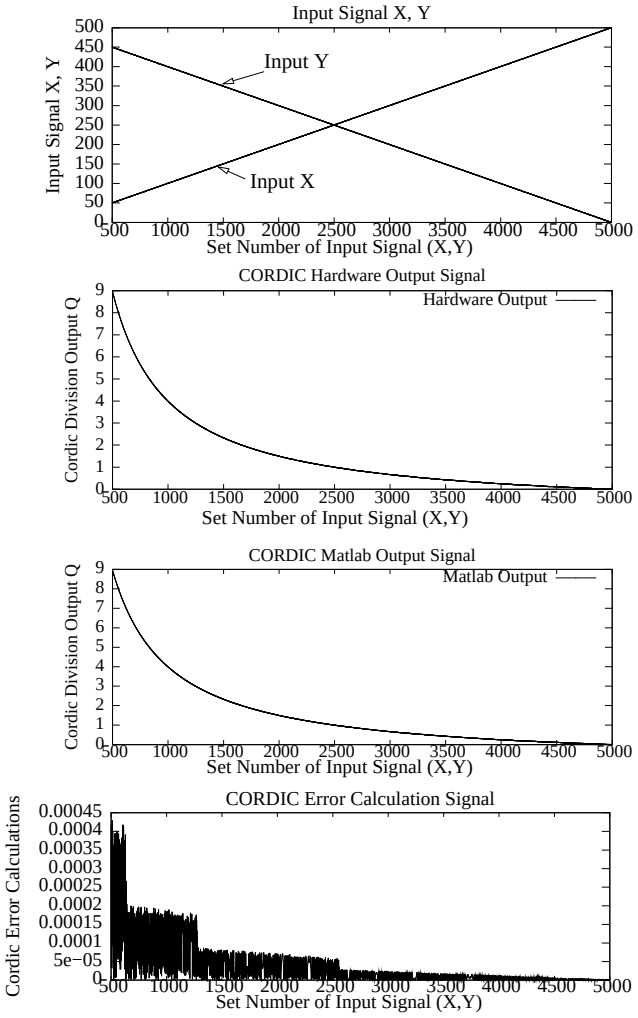

Fig.6:: Evaluation of the CORDIC Division operation ($Q = \frac{Y}{X}$) and the calculation errors by incrementing the divisor X and decrementing the dividend Y.

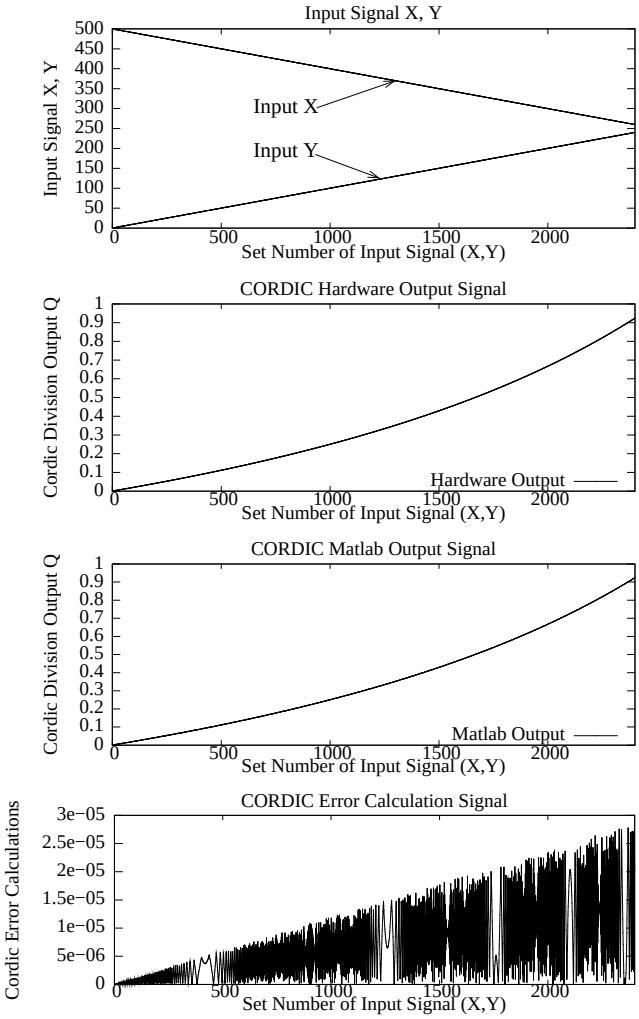

Fig.7:: Evaluation of the CORDIC Division operation ($Q = \frac{Y}{X}$) and the calculation errors by decrementing the divisor X and incrementing the dividend Y.

and Fig. 7. In the simulation result as presented in Fig. 6, the input divisor X is ramp up, while the input dividend Y is decremented. About 4500 sets of input pairs are presented in the figure. As shown in the figure, the division results of both the CORDIC hardware and the matlab simulation tend to decrease exponentially, and the absolute calculation errors tends also to decrease.

Fig. 7 shows another simulation, where the input divisor X is decremented, while the input dividend Y is incremented. As shown in the figure, the division result of the CORDIC hardware tends to increase exponentially, which in accordance with the matlab simulation result. However, in this test case scenario, the calculation errors tend to increase.

Table 1 shows the statistical analysis results of the CORDIC hardware over the computational errors compared to MATLAB simulation results. At each number of iteration, the maximum, the minimum and the absolute average errors as well as the

standard-deviation of the errors are evaluated over 5000 sets of input samples. It seems that the calculation errors decrease as the number of iterations is increased.

5. SILICON-AND FPGA-BASED SYNTHESIZED RESULT AND COMPARISON

The synthesis result of the floating-point CORDIC Divider by using a 130-nm CMOS standard-cell technology library from Faraday Technology Corporation is presented in Table 2. The synthesis result is made by using target frequency of 500 MHz, resulting in a slack-time of about 1.92 ns. The performance can be still improved by using a newest and faster CMOS standard-cell technology.

The logic cell area of components in the CORDIC Hardware Divider is presented in Table 3. The total area of the CORDIC divider is 40612 μm^2. Most of logic area is occupied by the multiplier and alignment units. The rest 5% logic-cell area is occupied by combinatorial blocks. Since two parallel floating-point operations are performed by CORDIC core, the CORDIC has about $2 \times 500\ MHz = 1\ GFlops$ (Giga floating-point operation per second).

Table 4 shows the synthesis result of the floating-point CORDIC Divider on an FPGA device from Xilinx Corporation. By using the Virtex-2 FPGA device, the maximum data frequency of the CORDIC core is slower than the synthesis result on CMOS standard-cell technology. The maximum performance of the CORDIC on the Virtex-2 FPGA is about $2 \times 105.406\ MHz = 210.812\ MFlops$.

A brief survey of several floating-point division implementations in published articles is summarized in Table 5. It is difficult to compare the designs and architectures where different design methodologies were applied, such as full custom and standard cell silicon technology approach. However, the iterative architectural comparison based on the number of basic floating-point operators and computational latency can be used for performance and efficiency evaluation in top-level design. The proposed floating-point division based on CORDIC method is compared with Goldschmit's [11] and Taylor-Series Expansion [10] methods. The computational latency includes the time to form the initial approximation and the appropriate number of iterations. The latency of the proposed method is higher than Taylor-Series Expansion method, but lower than Goldschmit's method. When the number of basic floating-point operators are considered, the proposed CORDIC-based method applies #1FPMUL and #2FPADD/SUB, where as Taylor-Series Expansion method and Goldschmit's method consume #2FPMUL, #1FPADD/SUB, and #2FPMUL, respectively.

Table 2:: Synthesis results using 130-nm CMOS standard-cell technology library with target frequency of 500 MHz.

Measurements	Synth. result
Total logic cell area	0.0406 mm^2
Slack time (critical path)	1.92 ns
Switching power (1.32V)	2.178 mW
Internal power (1.32V)	5.994 mW

Table 3:: Logic cell area of the components.

Component	Cell area (μm^2)	Percent. (%).
Top Module	40612	100.0
Inversion Module	14264	35.0
Multiplier+Alignment	24222	59.6

Table 4:: Synthesis result using Virtex-2 FPGA device (2vp30ff896-7) from Xilinx Corporation.

	Utilization	% of Total
Number of slice	1254 of 13696	9%
Number of MULT18X18 blocks	4 of 136	2%
Minimum Delay	9.487 ns	
Maximum Frequency	105.406 MHz)	

Table 5:: Floating-Point Division comparison of published literatures in single-precision.

Methodology	Latency	Basic FP Operator	Speed	CMOS Technology size
Goldschmit's [11]	16/13	#2FPMUL	2.3 GHz	65 nm
Taylor-Series Expansion [10]	12/5	#2FPMUL+#1FPADD/SUB	500 MHz	65 nm
The proposed CORDIC-based	14/8	#1FPMUL+#2FPADD/SUB	500 MHz	130 nm

FPMUL : Floating-Point MULtiplier, FPFPADD/SUB : Floating-Point ADDer/SUBtractor

6. CONCLUSIONS AND FUTURE WORKS

A CORDIC core implementing a floating-point divider operation has been presented in this paper. The new algorithm is proposed to solve the limited input domains of the input ranges that can be solved with well convergence by the traditional CORDIC algorithms to implement the division operations. The CORDIC operation is basically divided into four stages, i.e. divider exponent detection, divisor inversion, inversion result alignment and multiplication. The proposed algorithm shows that the CORDIC can increase the input ranges that can guarantee the convergence of the CORDIC algorithm. In future work, the core will be integrated within a streaming processor [24]. Due to the reconfigurability of the CORDIC core, the implementations of many trigonometric and logarithmic functions, including the division operation on top of a single programmable/reconfigurable CORDIC core will be helpful to reduce area of the processor arithmetic units.

References

[1] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Design.* Oxford University Press, New York, Oxford, 2000.

[2] D. Rao, S. Patil, N. Babu, and V. Muthukamar, "Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based Hardware Description Languages," *International Journal of Theoretical and Applied Computer Sciences*, vol. 1, no. 1, pp. 9–34, 2006.

[3] A. Beaumont-Smith and S. Samudrala, "Method and System of a Microprocessor Subtraction-Division Floating-Point Divider," Patent US 7,127,483 B2, Oct. 24, 2006.

[4] D. J. Desmonds, "Binary Divider with Carry Save Adders," Patent US 4,320,464, March 16, 1982.

[5] S. F. Oberman and M. J. Flynn, "Division Algorithms and Implementations," *IEEE Trans. Computers*, vol. 46, no. 8, pp. 833–854, Aug. 1997.

[6] J. Ebergen, I. Sutherland, and A. Chakraborty, "New Division Algorithms by Digit Recurrence," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, vol. 2*, 2004, pp. 1849–1855.

[7] L. Montalvo, K. Parhi, and A. Guyot, "A Radix-10 Digit-Recurrence Division Unit: Algorithm and Architecture," *IEEE Trans. Computers*, vol. 56, no. 6, pp. 727–739, June 2007.

[8] I. Rust and T. Noll, "A digit-set-interleaved radix-8 division/square root kernel for double-precision floating point," in *IEEE International Symposium on System on Chip (SoC)*, 2010, pp. 150–153.

[9] L. Montalvo, K. Parhi, and A. Guyot, "New Svoboda-Tung Division," *IEEE Trans. Computers*, vol. 47, no. 9, pp. 1014–1020, Sep. 1997.

[10] T.-J. Kwon, J. Sondeen, and J. Draper, "Floating-point division and square root using a Taylor-series expansion algorithm," in *the 50th Midwest Symposium on Circuits and Systems (MWSCAS 2007)*, 2007, pp. 305–308.

[11] S. F. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor," in *The 14th IEEE Symposium on Computer Arithmetic*, 1999, pp. 106–115.

[12] M. J. Schulte, D. Tan, and C. L. Lemonds, "Floating-Point Division Algorithms for an x86 Microprocessor with a Rectangular Multiplier," in *Proc. Int'l. Conf. on Computer Design (ICCD)*, 2007, pp. 304–310.

[13] W. Gallagher and E. Swartzlander, "Fault-Tolerance Newton-Raphson and Goldschmidt Dividers using Time Shared TMR," *IEEE Trans. Computers*, vol. 49, no. 6, pp. 588–595, June 2000.

[14] L.-K. Wang and M. Schulte, "Processing Unit Having Decimal Floating-Point Divider Using Newton-Raphson Iteration," Patent US 7,467,174 B2, Dec. 16, 2008.

[15] P. Surapong, F. A. Samman, and M. Glesner, "Design and Analysis of Extension-Rotation CORDIC Algorithms based on Non-Redundant Method," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 5, no. 1, pp. 65–84, March 2012.

[16] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, "Modified Virtually Scaling-Free Adaptive CORDIC Rotator Algorithm and Architecture," *IEEE Trans. on Circuits and Sys-*

tems for Video Technology, vol. 15, no. 11, pp. 1463–1474, Nov. 2005.

[17] F. A. Sammany, P. Surapong, C. Spies, and M. Glesner, "Floating-point-based Hardware Accelerator of a Beam Phase-Magnitude Detector and Filter for a Beam Phase Control System in a Heavy-Ion Synchrotron Application," in *Proc. Int'l Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS)*, 2011.

[18] H. Hahn, D. Timmermann, B. Hosticka, and B. Rix, "A Unified and Division-Free CORDIC Argument Reduction Method with Unlimited Convergence Domain Including Inverse Hyperbolic Functions," *IEEE Trans. on computers*, vol. 43, no. 11, pp. 1339–1344, Nov. 1994.

[19] J. Zhou, Y. Dou, Y. Lei, and Y. Dong, "Hybrid-Mode Floating-Point FPGA CORDIC Co-processor," in *The 4th international workshop on Reconfigurable Computing: Architectures, Tools and Applications, (ARC'08)*, 2008, pp. 256–261.

[20] P. Surapong, F. Philipp, F. A. Samman, and M. Glesner, ""Improvement of Standard and Non-Standard Floating-Point Operators"," *ECTI Trans. Computer and Information Technology*, vol. 6, no. 1, pp. 9–22, May 2012.

[21] D. Munoz, D. Sanchez, C. Llanos, and M. Ayala-Rincon, "FPGA based floating-point library for CORDIC algorithms," in *IEEE Conf. Programmable Logic Conference*, 2010, pp. 55–60.

[22] T. Lund, M. Aguirre, and A. Torralba, ""Making use of CORDIC and distributed arithmetic to produce a field-programmable fuzzy logic controller in an FPGA"," in *The 28th IEEE Annual Conf. of the Industrial Electronic Society (IECON'02)*, 2002, pp. 3205–3208.

[23] B. Das and S. Banerjee, "Unified CORDIC-based chip to realise DFT/DHT/DCT/DST," *IEE Proceedings, Computers and Digital Techniques*, vol. 149, no. 4, pp. 121–127, Nov. 2002.

[24] F. A. Samman, P. Surapong, and M. Glesner, "Reconfigurable streaming processor core with interconnected floating-point arithmetic units for multicore adaptive signal processing systems," in *Proc. of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2011, pp. 1–6.

Pongyupinpanich Surapong was born in Prachinburi, Thailand. He received his Bachelor and Master of Engineering degree in Electrical Engineering from King Mongkut's Institute of Technology Ladkrabang (KMITL), Thailand in 1998 and 2002, respectively. He received his PhD degree in 2012 from Technische Universität Darmstadt, Germany. Currently, he is a lecturer and research fellow at Department of Computer Engineering, Faculty of Engineering, Ramkhamhaeng Universitas, in Bangkok, Thailand. His research interests include computer-aided VLSI design, hardware modeling, design optimization algorithm, circuit simulation, digital signal processing, system-on-chip, all in the context of field-programmable gate-array devices and VLSI technology.

Faizal Arya Samman was born in Makassar, Indonesia. He received his Bachelor of Engineering degree in Electrical Engineering from Universitas Gadjah Mada (UGM), Yogyakarta in 1999 and his Master of Engineering degree from Institut Teknologi Bandung (ITB) in 2002. Since 2002 he has been appointed to be a research and teaching staff at Universitas Hasanuddin in Makassar, Indonesia. He received his PhD degree in 2010 from Technische Universität Darmstadt, Germany with scholarship award (2006-2010) from Deutscher Akademischer Austausch-Dienst (DAAD, German Academic Exchange Service). From 2010 until 2012, he was a postdoctoral fellow in the research project in LOEWE-Zentrum AdRIA (Adaptronik-Research, Innovation, Application) within the research cooperation framework between Technische Universität Darmstadt and Fraunhofer Institut LBF in Darmstadt. He is now a lecturer and research fellow at Department of Electrical Engineering, Faculty of Engineering, Universitas Hasanuddin, in Makassar, Indonesia. His research interests include network on-chip (NoC) microarchitecture, NoC-based multiprocessor system-on-chip, design and implementation of analog and digital electronic circuits for control system applications on FPGA/ASIC as well as energy harvesting systems and wireless sensor networks.