International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Design and Analysis of Double Precision Floating Point Division Operator Based on CORDIC Algorithm

Chetan Dudhagave[1], Hari Krishna Moorthy[2]

[1]M.Tech Student (SP & VLSI) Department of Electronics and Communication Engineering, Jain University, Karnataka, India

[2]Assistant Professor, Department of Electronics and Communication Engineering, Jain University, Karnataka, India

Abstract: ***Floating point arithmetic units provides better accuracy, precision and it covers larger data ranges compared to fixed point. Designing of floating point division operator is complex compared to other data operands. 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. This paper involves the design of Double precision floating point division operator using CORDIC algorithm. The new architecture of CORDIC Algorithm is proposed in this project which overcomes the limitation in terms of range of inputs and is capable of processing broader input ranges. The performance is evaluated for large input tests. The results show that the proposed system gives precise division operation results with broader input ranges. The proposed hardware architecture is modeled in VERILOG and synthesized on Virtex-4FPGA device (xc4vsx25). The design has achieved maximum frequency of 211.879MHz.***

Keywords: Floating-Point operators, CORDIC Algorithm, 64-bit IEEE Standard Double-Precision

1. Introduction

In modern digital computer architecture, performance of digital computer is vastly improved due to floating point arithmetic units. Floating-point arithmetic units is preferred over fixed point arithmetic units as it provides better accuracy and precision and it covers larger data ranges, which is suitable for scientific computations in engineering application areas. Adder, substractor, multiplier and divider units are considered basic operators in scientific computations along with other trigonometric functions such as sine, cosine, logarithm, exponent, etc. Floating-point division is hard to implement due to the complexity of the algorithms, compared to other floating point arithmetic operators. So many scientists and researchers have taken keen interest to introduce efficient algorithm for designing floating point division operator.

Floating point division algorithms are classified in three categories: digit recurrence algorithms, lookup table and multiplier based algorithms, and iterative algorithms. Digital recurrence is the most popular class of algorithms due to its simplicity. SRT algorithm named by Freiman [1] is most popular digit recurrence division algorithm is the. Many variations of the original SRT division algorithm were thoroughly studied by Ercegovac and Lang [2]. Oberman and Flynn [3] discussed about overall system performance floating point operations including division. Design of floating-point division and square root in several microprocessor architectures were discussed by Leeser [4]. The earliest implementation of single precision floating-point divider on a Xilinx XC4010 [5, 6] used a variation of the radix-2 SRT algorithm. The first IEEE double-precision floating-point divider implemented on a Xilinx XCV1000 FPGA [7] also used the simple radix-2 digit-recurrence algorithm. This is a non-pipelined design so it is very small in area but very low in throughput with a very high latency of 60 clock cycles for double-precision division. Based on this sequential design, a pipelined design [8] was presented, which unrolls the iterations of the digit recurrence computations and optimizes the operations within each unrolled iteration. The maximum frequency of this pipelined design for double-precision floating-point divide is over 100MHz on a Xilinx XC2V6000 FPGA while consumes a modest 8% of the chip area. Another work to improve the throughput of the division via a simple digit-recurrence algorithm is based on a scalable pipeline insertion method [9]. This is also a heavily pipelined design so it has the maximum frequency of over 200MHz for a single-precision floating point divider on a Xilinx Virtex-II XC2V1000 FPGA.

The divider is bit serial and exhibits long latency. All the above digit recurrence algorithms are radix 2 based. Floating-point dividers based on a high radix SRT division algorithm for radix 2, 4, and were compared [10]. Three implementations - a low cost iterative implementation, a low latency array implementation, and a high throughput pipelined implementation were studied. Another high radix SRT divider (radix 4) of both pipelined and non-pipelined structure [11] were implemented on Xilinx Virtex-II XC2V1000 and XC2V6000 FPGAs. The single-precision pipelined implementation takes 15 clock cycles with a clock rate of $9{:}6ns$ while the double-precision pipelined implementation takes 29 clock cycles with a clock rate of $12{:}2ns$. The double-precision divide [12] supporting most features of the IEEE Std 754 is also based on digit recurrence algorithms. Extending this work, more recent work [13] support both 64-bit double-precision divide and 128-bit quad-precision divide on a Xilinx Virtex-II Pro FPGA.

Lookup table based algorithms are another class of division algorithm. A popular approach to carry out a division is to

calculate the inverse of the denominator based on a lookup table, followed by multiplying that inverse by the numerator. However, the size of the table to do the inversion increases exponentially when the size of divider increases.

So this inverse lookup table method is only feasible for small floating-point dividers. One work [14] built custom floating-point formats divider - 16 bits (1-6-9) and 18 bits (1-7-10) for 2-D FFT and FIR filter applications. Another work [15] built the floating-point divide with 8 bit output accuracy for a video processing application in a Xilinx Virtex E FPGA.As an extension of [14], the authors coded the inverse to reduce the required bit width of the following multiplication.

The third approach for division is iterative algorithms. One work using iterative algorithms to implement a floating-point divider in FPGAs was presented by Roesler and Nelson [16]. Based on the Newton-Rapson algorithm and using repeated multiplication to approximate the result, this algorithm requires only multipliers. However, a small lookup table can be added to compute the result of the first few iterations to reduce the total number of iterations. Unlike the lookup table based algorithms, larger floating-point format

Along with these methods to implement the division operation, there is also another powerful algorithm to implement the divider unit called CORDIC (COrdinate Rotation DIgital Computer) algorithm [17]. The main powerful characteristic of the CORDIC algorithm is the capability to implement several trigonometric function [18] and hyperbolic functions [19] as well as linear operational function such as multiplication and division functions.

The paper is organized as follows. Section 2 shows basic architecture of CORDIC division algorithm. In section 3 we propose Double Precision Floating point architecture based on CORDIC algorithm. The convergence and calculation errors are presented in section 4. Synthesis results of the CORDIC divider core modeled in Verilog by using FPGA device are presented in this section.Finally section 5 concludes the work.

2. Basic Radix-2 CORDIC Division Algorithm

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

$$X_{i+1} = X_i - m.\mu_i.Y_i.\delta_{m,i}$$
$$Y_{i+1} = Y_i + \mu_i.X_i.\delta_{m,i}$$
$$Z_{i+1} = Z_i - \mu_i.\delta_{m,i} \quad (1)$$

If CORDIC has to implement division function then CORDIC Algorithm has to be configured in linear mode, i.e. by assigning m=0 and $\delta_{m,i} = 2^{-i}$ in Equ. 1. Thus, the CORDIC iterative equation for Division operator is shown below.

$$X_{i+1} = X_i$$
$$Y_{i+1} = Y_i + \mu_i.X_i.2^{-i}$$
$$Z_{i+1} = Z_i - \mu_i.2^{-i} \quad (2)$$

The value of μ_i can be either +1 or -1, depending on the value of Y_i as shown in Eqn. below

$$\mu_i = \begin{cases} +1 & Y_i < 0 \\ -1 & Y_i \geq 0 \end{cases} \quad (3)$$

3. Proposed System

In this brief, floating point division operator using CORDIC Algorithm is proposed which can overcome limitation in term of the range of inputs that has been processed by the CORDIC machine to give proper convergence.

The complete division operation Q = Y/Xand its hardware architecture is proposed, where Qis the division operation result, *Y* is the dividend and *X* is the divisor.

There are two basic reasons why a new algorithm is proposed:

1. The CORDIC algorithm gives correct convergence when the expected division results are located in the following ranges:$-2 \leq Q = Y/X \leq 2$The values outside the range will tend to saturate at unexpected division operation results.
2. We cannot identify, whether the division results are in the aforementioned range or not, unless the division has been made.
3. The domain of well convergence of the CORDIC inversion function is shown in Equ. (4). If the domain is written in the IEEE binary Double precision floating-point standard, the domain can be described in Equ. (5) as well as Equ. (6) and Equ. (7) Present the exponent and mantissa faction.

$$0.5 \leq X < 1.5 \quad (4)$$
$$1 * 2^{1022} \leq X < 1.5 * 2^{1023} \quad (5)$$
$$1 \leq Mx \leq 1.5 \quad (6)$$
$$1022 \leq Ex \leq 1023 \quad (7)$$

The hardware architecture is classified into four main stages as presented in Figure. 1.

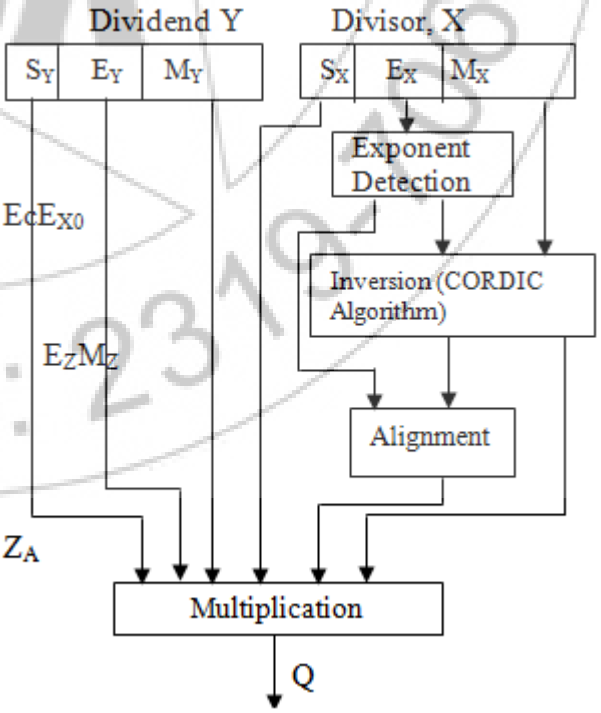

Figure 1: Double Precision Floating Point Division Architecture

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 using Algorithm 1.
2. Divisor Inversion: In this second stage the inverse value of the new input divisor Xnew = $(-1)S_X$ *M_X*2^{Exo} is computed by using CORDIC algorithm presented in Algorithm 2 to obtain the variable Z.
3. Alignment: In this third stage, Z_A is computed from the Z variable whose exponent is aligned by using the credit exponent E_C. By using a formal floating point equation, then we have $Z_A = (-1)^{Sz}$ *1.M_Z *2^{Ez-Ec} , where $S_Z = S_X$.
4. Multiplication: Finally we will have the complete division result as $Q = Y*Z_A$.

Algorithm 1[EC, EX0] =Divisor Detect(X), where X ≥1

1. Yo=1, Zo=0, Xo=X, S0=-1 {Initialization}
2. E_X= Exponent of the Input Divisor X
3. if E_X<1022 then
4. E_C= 1022 - E_X
5. EX0 = 1022
6. else if EX >1023 then
7. EC = EX - 1023
8. EX0 = 1023
9. else
10. EC = 0
11. EX0 = EX
12. end if
13. return EC,EX0

Algorithm 2Z=Inverting(X, I), where X ≥ 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. $Y_{i+1} = Y_i + (X_i * Si * 2^{-i})$
5. $Z_{i+1} = Z_i - (S_i * 2^{-i})$
6. if Y_i<0 then
7. S_{i+1}=+1
8. else
9. S_{i+1}=-1
10. end if
11. end for
12. return Z

4. Results

The design is modeled Verilog and synthesized on Virtex-4 FPGA devices (xc4vsx25). Table I shows the statistical analysis results of the CORDI Chardware over the computational errors compared to actual results computed by MATLAB. 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 100 sets of input samples. It seems that the calculation errors decrease as the number of iterations is increased.

Table 1: Statistical error calculation

Iter.	Max.	Min.	Abs. Mean	Std. Deviation
8	3.3320	-10.3750	2.570E-3	0.2780
16	0.0096	-0.0440	9.8750E-5	7.1450E-4
32	0.8E-5	-3.560E-6	6.3650E-8	3.1850E-7
64	0.65E-6	-8.780E-7	9.8670E-9	2.5760E-8

The schematic of area utilization design summary shown in the below represents the how much of area used regarding of area allotted. In this project the Number of Slice Flip Flops available is 20,480 used only 1,854 and 9% of area utilized. Similarly the other parameters related to area as shown in Table 2 and the timing analysis is shown in table 3.

Table 2: Utilization Summary

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	1,854	20,480	9%
Number of 4 input LUTs	3,077	20,480	15%
Number of occupied Slices	2,086	10,240	20%
Number of Slices containing only related logic	2,086	2,086	100%
Number of Slices containing unrelated logic	0	2,086	0%
Total Number of 4 input LUTs	3,148	20,480	15%
Number used as logic	3,076		
Number used as a route-thru	71		
Number of bonded IOB'S	195	320	60%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fan out of Non-Clock Nets	3.75		

Table 3: Timing Analysis

Minimum Period	4.72ns
Maximum Frequency	211.879MHz

5. Conclusion

In this brief, Double precision floating-point divider operation based on CORDIC algorithm is presented. The proposed design overcomes the limitation of traditional CORDIC algorithm to give good convergence for large input ranges. The proposed design is carried out in four stages i.e. divisor exponent detection in first stage, then divisor inversion in second stage, followed by alignment in third stage and multiplication in last stage. The design has been synthesized on Virtex-4 FPGA device (XC4VSX25). The design has achieved maximum frequency of 211.879MHz. In future work, the CORDIC core reconfigured for implementation of many trigonometric and logarithmic functions which help to reduce area of the processor arithmetic units.

References

[1] C. V. Freiman. Statistical analysis of certain binary division algorithms. In IRE Proc., volume 49, pages 91–103, 1961.
[2] M. D. Ercegovac and T. Lang. Division and Square Root: Digit Recurrence Algorithms and Implementations. Kluwer Academic Publishers, Mass., 1994.

[3] S. F. Oberman and M. J. Flynn. Design issues in division and other floating-point operations. In IEEE Transactions on Computers, pages 154–161, 1997.
[4] P. Soderquist and M. Leeser. Division and square root: choosing the right implementation. IEEE Micro, 17(4):56–66, July/August 1997.
[5] M. E. Louie and M. D. Ercegovac. Mapping division algorithms to field programmable gate arrays. In 1992 Conference Record of the 26th Asilomar Conference on Signals, Systems and Computers, pages 371–375, 1992.
[6] M. E. Louie and M. D. Ercegovac. On digit-recurrence division implementations for field programmalbe gate arrays. In Proc. 11th Symposium on Computer Arithmetic, pages 202–209, 1993.
[7] S. Paschalakis and P. Lee. Double precision floating-point arithmetic on FPGAs. In IEEE International Conference on Field-Programmable Technology (FPT), pages 352–358, December 2003.
[8] A. J. Thakkar and A. Ejnioui. Pipelining of double precision floating point division and square root operations. In ACM Proceedings of the 44th Annual Southeast Regional Conference, pages 488–493, March 2006.
[9] I. Ortiz and M. Jimenez. Scalable pipeline insertion in floating-point division and square root units. In Proceedings of the 2004 47th Midwest Symposium on Circuits and Systems (MWSCAS'04), volume 2, pages II–225–II–228, July 2004.
[10] X.Wang and B. E. Nelson. Tradeoffs of designing floating-point division and square root on virtex FPGAs. In 11th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 195–203, April 2003.
[11] B. Lee and N. Burgess. Parameterisable floating-point operations on FPGA. In 36th Asilomar Conference on Signals, Systems and Computers, volume 2, pages 1064– 1068. IEEE Signal Processing Society, November 2002.
[12] G. Govindu, R. Scrofano, and V. K. Prasanna. A library of parameterizable floating point cores for FPGAs and their application to scientific computing. In The 2005International Conference on Engineering of Reconfigurable Systems and Algorithms(ERSA 2005), June 2005.
[13] P. C. Diniz and G. Govindu. Design of Field-Programmable Dual-precision Floating-Point Arithmetic Units. In Proceedings of the 16th international conference on field-programmable logic and applications (FPL'06), pages 733–736, August 2006.
[14] N. Shirazi, A. Walters, and P. Athanas. Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), pages 155–162, April 1995.
[15] J. Dido, N. Geraudie, et al. A flexible floating-point format for optimizing data-paths and operators in FPGA based DSPs. In ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), pages 50–55, February 2002.
[16] E. Roesler and B. E. Nelson. Novel optimizations for hardware floating-point units in a modern FPGA architecture. In 12th International Conference on Field-Programmable Logic and Applications (FPL'02), pages 637–646, Sept. 2002.
[17] P. Surapong, F. A. Samman, and M. Glesner. Design and Analysis of Extension-Rotation CORDIC Algorithms based on Non-Redundant Method. In International Journal of Signal Processing, Image Processing and Pattern Recognition, vol. 5, no. 1, pp. 65–84, March 2012.
[18] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya. Modified Virtually Scaling-Free Adaptive CORDIC Rotator Algorithm and Architecture. In IEEE Trans. on Circuits and Systems for Video Technology, vol. 15, no. 11, pp. 1463–1474, Nov. 2005.
[19] 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. InIEEE Trans. on computers, vol. 43, no. 11, pp. 1339–1344, Nov.1994.

Author Proile

Chetan Dudhagave received Bachelor of Engineering from Rajarambapu Institute Technology, Rajaramnagar, Shivaji University, Kolhapur in 2012 and now pursuing Master of Technology in School of Engineering and Technology, Jain University, Bangalore.His area of interest includes VLSI and Signal Processing.

Hari Krishna Moorthy obtained Bachelor of Engineering from G.V.I.T.K.G.F, Bangalore University in 2001 and Master of Engineering from Sathyabama University, Chennai in the year 2007.Assistant Professor in the Department of Electronics and Communication Engineering, School of Engineering and Technology,Jain University, Bangalore-Karnataka, India