Available online at www.sciencedirect.com
ScienceDirect
Procedia Technology 24 (2016) 1120 – 1126
Procedia Technology

International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015)

FPGA Implementation of Area-Efficient IEEE 754 Complex Divider

Anila Ann Varghese[a*], Pradeep C.[b], Madhuri Elsa Eapen[c], R. Radhakrishnan[d]

[a, b, c] *Dept. of Electronics and Communication, SAINTGITS College of Engineering, Kottayam, India, 686532*
[d]*Dept. of Electronics and Communication, Vidhya Mandhir Institute of Technology, Erode, India, 638052*

Abstract

Division algorithms are less often used unlike other arithmetic operations. But it cannot be avoided in some systems to achieve some functionality. The division of complex numbers has got applications in fields like telecommunication, microwave systems, signal processing, GPS etc. This work proposes an area-efficient method for complex divider implementation on FPGA. The operands are represented in single precision floating point (IEEE754) format. A novel method called module reuse technique is used for reducing the device utilization on FPGA. The proposed design is analyzed using the simulation and implementation results on Xilinx Artix-7 and Virtex-5 FPGA families.

Keywords: Complex Division; FPGA; IEEE754; Module Reuse

1. Introduction

In comparison with arithmetic operations like addition, subtraction and multiplication, division arithmetic is hardly used. This is because of the inherent complexity of a divider module. The size of the divider module is more and it takes more time for completion unlike other arithmetic operations. Because of this, most of the architectures avoid the use of a divider module. But some systems require a divider module in order to achieve the required functionality. Therefore, the design of a divider has to be given enough importance to meet the performance requirements of the system.

* Corresponding Author. Tel.: +91-9820962406
*E-mail address:*anilaann7@gmail.com

doi:10.1016/j.protcy.2016.05.245

A complex divider is a module which takes two complex numbers as its numerator and denominator inputs and produces another complex number at its quotient output. A complex number is a number which consists of a real part and an imaginary part. Complex number division has got many applications in fields like signal processing, telecommunication, control theory, microwave systems etc. A complex divider that can be used in FPGA based systems which uses single precision floating point representation is proposed in this paper. FPGAs helps in solving different issues associated with reliability and process availability due to its reconfiguration capability. A module reuse technique is used to reduce the total size of the divider at the cost of increased quotient computation time and extra control signals.

This paper is organized as in five sections. Section II discusses the previous works related to division algorithms and complex number arithmetic. The proposed single precision floating point divider design and the method of module reuse is discussed in section III. The simulation results and the implementation details are given in section IV. Section V concludes the work along with the scope of the future work.

2. Previous Works

[1] proposed an algorithm which yields the complex quotient of $a+ib$ divided by $c+id$, which avoids arithmetic overflow or underflow. It was modified to make it more robust in the presence of underflows [2]. The algorithm works for virtually all problems in which the numerator, denominator, and quotient are representable as normalized floating point numbers returning an almost accurate answer.

SRT division is a division algorithm which is simple. This widely implemented algorithm uses digit recurrence method. Subtraction is used as the fundamental operator and it retires a fixed number of quotient bits in each iteration. In [3], the effects of divider architectures and circuit families on performance and area are analyzed for radix-2 and radix-4 SRT dividers. [4] verified the correctness of SRT division circuit similar to one in the Intel Pentium Processor.

The technique used for high radix complex division proposed by [5] based on operand prescaling and digit recurrence, which made the selection of quotient digits simple and led to a simple hardware implementation, and allowed correct rounding of complex quotient. [6] described the original version of SRT division as a dynamical system.

A fixed-point implementation of a robust complex valued divider architecture is presented in [7]. The technique proposed in [5] had later been implemented on FPGAs with different radices [8] and [9]. Based on the same algorithm, [10] presented a radix-16 combined complex divider/square root module. The implementation and analysis of interval radix-2 SRT division in double precision is presented in [11]. A slightly modified radix-4 SRT division considering the reliability and performance metrics objectives is presented in [12].

3. Proposed Design

3.1. Complex Divider using Look Up Table Approach

The complex division module proposed for floating point contains different modules as shown in the Fig.1. The modules are the multiplier module, normalization module, exception handler, exponent calculator, quotient selection look up table and final quotient computation module.

i) Multiplication module and Denominator calculator

This module is used to multiply the complex conjugate of the denominator with both numerator and denominator. The module produces three outputs which are numerator for the real part, numerator for the imaginary part, denominator common for both parts, all of which are real numbers. For two complex numbers with $y=a+ib$ and $z=c+id$,

$$\frac{a+ib}{c+id} = \frac{(a+ib)(c-id)}{(c+id)(c-id)} = \frac{xreal}{D} + i\,\frac{ximag}{D} \tag{1}$$

where $D = c^2 + d^2$ and $xreal$ and $ximag$ are calculated using the Golub's method of multiplication as described below. Golub's multiplier uses a more efficient but indirect approach. For two complex numbers, y=a+ib and z=c+id,

$$x = y \times z = (t2 - t3) + i(t1 - t2 - t3) \tag{2}$$
$$xreal = t2 - t3$$
$$ximag = t1 - t2 - t3$$
$$t1 = (a + b) \times (c + d), t2 = a \times c, t3 = b \times d$$

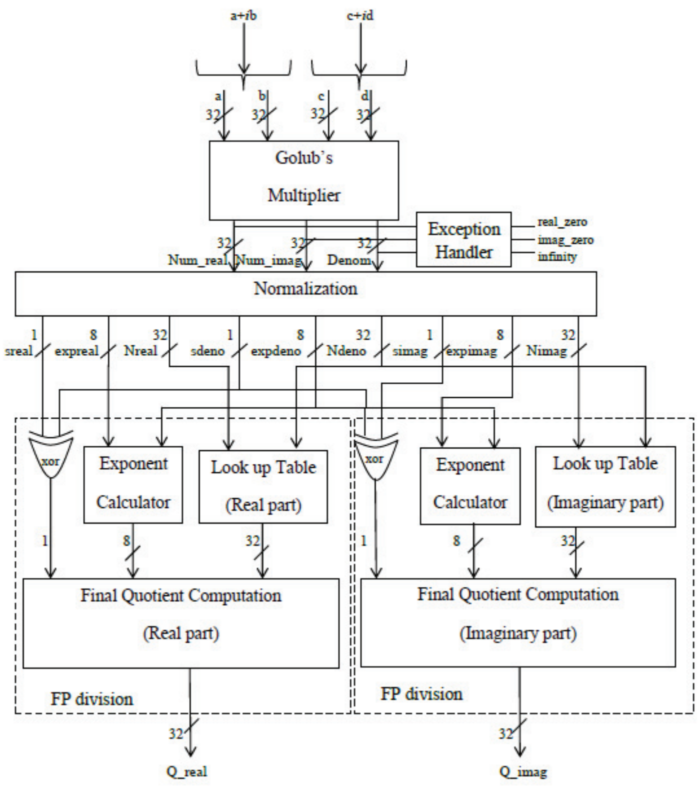


Fig. 1. Proposed Single Precision Floating Point complex Divider.

This approach requires three real multiplications and five additions, instead of four real multiplications and two additions required in the case of direct multiplication. This is more area efficient because multiplication requires more area than addition. A floating point adder and a multiplier are used for the respective addition and multiplication operations.

ii)Normalization module

The normalization of the $xreal, ximag$ and D to $Nreal, Nimag$ and $Ndeno$ is done such that $1 \leq Nreal, Nimag\ Ndeno \leq 2$. After normalization, the actual division operation is done in real and imaginary modules separately.

iii) Exception Handler

This module handles the exceptions which the floating point division unit cannot handle. Such situation arises when the numerator of the real part, numerator of the imaginary part or the denominator is 0. The signals $real\_zero$, $imag\_zero$ and infinity will turn high respectively for 0 values of num_real, num_imag, and $denom$.

iv) Quotient Computation

The quotient computation module has an xor gate, exponent calculator and a quotient selection look up table. The xor gate directly calculates the sign bit of the quotient taking the sign bits of numerator and denominator as its inputs. The exponent calculator calculates the exponent for the quotient based on the equation

$$e_{quo} = e_{num} - e_{deno} + 127 \tag{3}$$

The output of this module will be given to the final quotient computation module. The bits of normalized dividend and the divisor is given as the inputs to the quotient selection look up table. There is a trade-off between the size of the look up table and the accuracy of the final quotient.

v) Final Quotient Computation Module

This module takes the outputs of the first quotient computation module. The calculated exponent value is multiplied with the output of the look up table. The sign bit is simply the bit, computed using the xor gate, which takes the sign bits of the dividend and the divisor as its input.

3.2. Complex Divider using Look Up Table Approach.

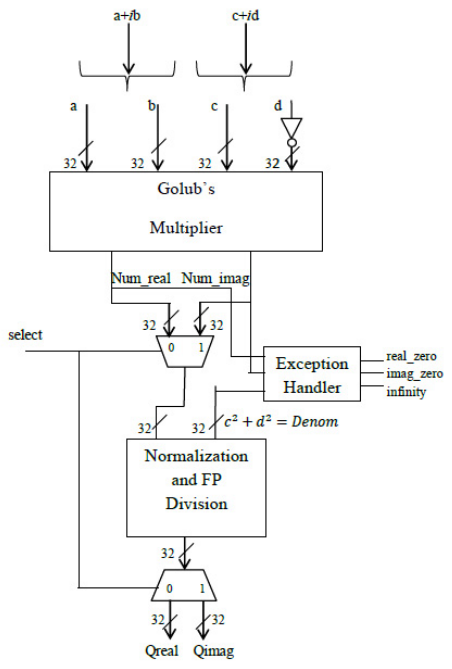


Fig. 2. Modified Complex divider with Module Reuse.

The floating point division units in the real part and the imaginary part of the complex division architecture are redundant modules. The concept of time redundancy is used to eliminate the use of one of the floating point division units in order to reduce the overall area of the complex division module. This is as shown in Fig. 2. But this reduction in area comes at the cost of increase in the total time required to compute the final result. Also, registers are required to hold the real part of the quotient while the computation of the imaginary part of the quotient goes on, and vice versa.

Since floating point division unit is the one which takes the major portion of the total area, the reduction in the total area is considerable. The area taken by the multiplexers and demultiplexers required to switch between the division operation on real and imaginary parts is very small in comparison with the area of one floating point division unit.

4. FPGA Implementation and Simulation Results

4.1. FPGA Implementation

To understand the effectiveness of the proposed complex division architecture, the design is implemented on Xilinx Artix-7 and Virtex-5 FPGA families. The FPGA implementations are done using ISE version 14.5 and synthesized for Artix-7 xc7a100t-2csg324 and Virtex-5 xc5vlx110t-2ff1136 devices.

Through this analysis, the performance and implementation metrics of the complex divider for two different FPGAs can be observed with Verilog as the design entry. The design style used is the structural modelling. Golub’s multiplier, denominator calculator, floating point divider, normalization unit, exception handler multiplexers and demultiplexers are implemented individually. These units are port mapped at the higher level. Among these, Golub’s multiplier, denominator calculator, floating point divider modules have floating point adders and binary adders hierarchically called inside them.

Table 1 shows the comparison of the FPGA implementation result on the families Artix-7 and Virtex-5. Artix-7 and Virtex-5 are high end devices. In Artix-7 device xc7a100t-2csg324, out of 63400 slice LUTs, 6600 LUTs (10%)is used in both the original architecture and 6119 LUTs (9%) is used in the modified architecture. The reduction in area in the modified architecture is 7.28%. The Virtex-5 device xc5vlx110t-2ff1136 has 69120 slice LUTs. Out of these, 6962 LUTs(10%)is used for the original architecture whereas the architecture with module reuse uses 6195 (8%). The area reduction is 11.01%. Fig. 3 shows the comparison of device utilization in different FPGA families.

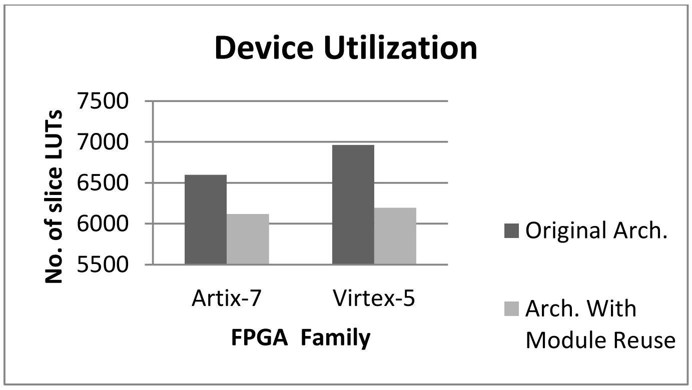


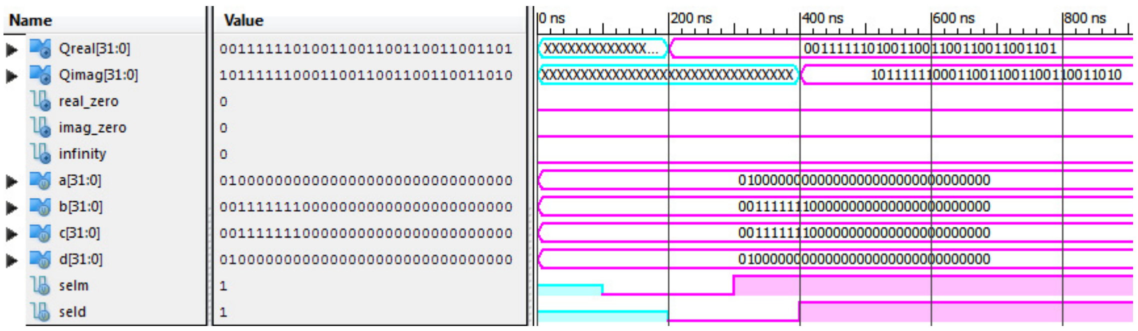
Fig. 3. Comparison of Device Utilization in different FPGA families.

Table 1. Comparison of FPGA implementation result on different families

Device Utilization Summary	Artix-7 XC7A100T-2CSG324		Virtex-5 XC5VLX110T-2FF1136	
	Original Architecture	Architecture with Module Reuse	Original Architecture	Architecture with Module Reuse
Total Slice LUTs	63400		69120	
No. of Slice LUTs	6600 (10%)	6119 (9%)	6962 (10%)	6195(8%)
Reduction in Area with Module Reuse	7.28%		11.01%	

4.2. Simulation Results

Behavioral simulation is done prior to FPGA implementation to check the functionality of the circuit. After the different phases of implementation, namely, translate, map, and place & route, post route simulation is done to observe the exact performance the architecture. The post-route simulation is the closest emulation to actually downloading a design to a device. It is done in order to check if the design meets the actual timing requirements as expected or not. Fig. 4 shows the post-route simulation of the floating point complex divider without fault detection. To perform floating point division of $2 + 1i$ divided by $1 + 2i, a, b, c$ and d are given the single precision floating point representation of 2,1,1 and 2 as the inputs, respectively. The select lines of mux and demux are switched from low to high once $Qreal$ is obtained. The computation of $Qimag$ is done thereafter. After the completion of the operation, $Qreal$ and $Qimag$ show the floating point representation of 0.5 and -0.5, respectively. Thus for the operation $2 + 1i$ divided by $1 + 2i$, the quotient is obtained as $0.5 - 0.5i$. The signals $real\_zero, imag\_zero$ and $infinity$ remain low since numerator of real part, numerator of imaginary part and denominator are not 0s.



Name	Value
Qreal[31:0]	00111111010011001100110011001101
Qimag[31:0]	10111111000110011001100110011010
real_zero	0
imag_zero	0
infinity	0
a[31:0]	01000000000000000000000000000000
b[31:0]	00111111100000000000000000000000
c[31:0]	00111111100000000000000000000000
d[31:0]	01000000000000000000000000000000
selm	1
seld	1

Fig. 4. Post-route simulation for 2+1i divided by 1+2i.

5. Conclusion

An IEEE754 complex divider is implemented on Artix-7 and Virtex-5 FPGA families using the look up table approach. The architecture is made area-efficient by a novel technique called module reuse. A comparison of the original architecture and the modified architecture using module reuse is done on both families. Virtex-5 board has slightly high device utilization compared to the Artix-7 family in both cases. The results show that there is a significant reduction in the device utilization at the cost of increased computational time when the module reuse technique is used. Reducing the computational time without further increase in area can be considered as the future scope of this work. Also, a fault-tolerant complex divider can also be designed with slight modifications in the original architecture.

References

[1] R. L. Smith, “Algorithm 116: Complex division,” ACM Transactions on Communications, vol. 5, no. 8, 1962, pp. 435.
[2] G. W. Stewart, “A note on complex division,” ACM Transactions on Mathematical Software, vol. 11, no. 3, 1985, pp. 238-24.
[3] D. L. Harris, S. F. Oberman, and M. A. Horowitz, “SRT division architectures and implementations,” in Proc. of IEEE Symosium. on Computer Arithmetic, 1997, pp. 18-25.
[4] E. M. Clarke, S. M. German, and X. Zhao, “Verifying SRT algorithm using Theorem Proving Techniques”, Formal Methods in System Design, Kluwer Academics Publishers, 1999, pp. 7-44.
[5] M. D. Ercegovac and J. M. Muller, “Complex division with prescaling of operands,” in Proc. IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2003, pp. 304-314.
[6] McCann, Mark, and Pippenger, Nicholas, “SRT Division Algorithms as Dynamical Systems”, Siam Journals on Compuer. Society for Industrial and Applied Mathematics, Vol. 34, No. 6, 2005, pp. 1279–1301.
[7] F. Edman and V. Oewall, “Fixed-point implementation of a robust complex valued divider architecture,” in Proceedings of European Conf. Circuit Theory and Design, Aug. 2005.
[8] D. Wang, M. D. Ercegovac, and N. Zheng, “A radix-8 complex divider for FPGA implementation,” in Proceedings of IEEE International Conference on Field Programmable Logic and Applications, 2009, pp. 236-241.
[9] P. Dormiani, M. D. Ercegovac, and J. M. Muller, “Design and implementation of a radix-4 complex division unit with prescaling,” in Proceedings of IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009, pp. 83-90.
[10]D. Wang and M. D. Ercegovac, “A radix-16 combined complex division/square root unit with operand prescaling,” IEEE Transactions of Computer,Vol. 61, No. 9, 2012, pp. 1243-1255.
[11]M. R. Patel, T. V. Shah, D. H. Shah, “Implementation and Analysis of Interval SRT Radix-2 Division Algorithm”, International Journal of Electronics and Computer Science Engineering, Vol. 1, No. 3, 2012, pp. 971-976.
[12]M. M. Kermani, N. Manoharan, and R. Azarderakhsh, “Reliable Radix-4 Complex Division for Fault-Sensitive Applications”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, pp. 1-12