Gyancity Journal of Electronics and Computer Science,
Vol.2, No.1, pp.43-48, March 2017
ISSN: 2446–2918 DOI: 10.21058/gjecs.2017.21005

VHDL Implementation of a Fast Division Algorithm

[1]Sumit Kumar, [2]Mandeep Narula, [3]Shivam Chauhan
[1,2]Jaypee Institute of Information Technology, A-10, Sector-62, Noida (U.P.), India.
[3]G. B. Pant University of Agriculture and Technology, Pantnagar, Uttarakhand, India.
[1]sumitkr.abit@gmail.com, [2] mandeep.narula@jiit.ac.in [3]shivamchauhan581@gmail.com

***Abstract*-- Today is an era of digital or smart systems. All the smart gadgets or systems deal with binary operations occurring inside in order to bring out the desired results. Several binary operations occurring inside the digital system are very much known to us and are Binary addition, Binary subtraction, followed my multiplication and division. These four are the basic ones however, there are many more advanced operations developed. Our work describes the RTL (Register Transfer Level) depiction of Goldschmidt divider. The depictions are synchronized to the working clock of the employed microprocessor. The significant operations that get a highlight in this work are that the divider is synthesizable. VHDL is the software platform for our work. This work focuses to demonstrate that synchronized applications can be actualized at the front-end level of VLSI architectural system. This project describes a basic algorithm for a division operation. Its performance and consideration of the implementation in VHDL are discussed.**

***Keywords*-- Smart system, binary operations, VHDL, VLSI**

I. INTRODUCTION

Microprocessor, which is a backbone to any computer, is being driven or rather works in synchronization with the operating clock of a computer. This clock is the active element that controls every task or job assigned to the computer. Thereby, it also consumes the highest electrical power within the computer. Designers for a decade have been trying to reduce down the clock power using various techniques like clock gating. But all these techniques are done at circuit level of the design which manifests to large investments. Therefore, the focus is to accomplish it at block level abstraction of the design. VLSI (Very Large Scale Integration) design methodology basically follows two design flows: (A.) Front–end design flow. (B.) Back-end design flow. Major concern in back–end design flow is to lower down and optimize the area, delay and power of the circuits. Here, designers deal with the transistor level schematics and its corresponding layouts defining the original geometry of the design that gets to fabrication. In front-end design flow, the major focus remains on the functionality of the individual components allocated in the processor along with its overall functionality. Now, in industry both these design methodology work hand in hand. Front-end flow does the functional analysis of the design system. We are giving input to a known system and getting an output as RTL description using at the back–end for circuit implementation. But on the other hand, if we assume that we do not know the system and try to estimate the system using the output and the input; it becomes difficult in regular cases. Therefore, the main purpose in doing this work is to have a RTL description of Goldschmidt divider which are synthesizable without even using the costly synthesis tools and are synchronized with the operating clock of the computer. Our research describes a basic algorithm for a division operation. Its performance and consideration of the implementation in VHDL are discussed. We have described three possible implementations, the maximum performance in FPGAs, e.g. propagation delays and number of necessary steps to list the correct result.

II. LITERATURE REVIEW

VHDL is a most commonly used and popular standard language in industry used to describe the hardware from the abstract form to the concrete level [7, 8], technically from the design specifications to the RTL (Register Transfer Level) descriptions. VHDL was proposed as an IEEE standard in 1986. It is a very powerful and effective language with enormous language constructs which are capable of describing very sensitive and complex behaviours. Amongst, the commonly used HDL languages, divide and multiply operators are present. Though, in general, they are not synthesizable. The use of procedures for synthesis is not frequent in this language; more common is the use of functions. However, functions can return only one value [4]. Use of procedure gives more compact solution with quotient and remainder as outputs in the case of binary divider. Thus, we have shown the implementation of the designed synthesizable multiplier and divider on to an ALU. The entire paper has been constructed in sections. Section II defines the detailing of the programme-level architecture of ALU and its peripherals. Section III states the VHDL description of the synthesizable binary multiplier & divider and the section IV concludes the paper depicting the utility of the work. The realization of a high-speed division function has always been a difficult task requiring a considerable amount of hardware and power. In the past, integrated circuit manufacturers have virtually ignored the division problem, placing a major portion of their efforts towards the stand-alone multiplier chip. Many manufacturers now have 8- and 16-bit dedicated multiplier chips that operate in the 100- to 200-ns speed range. The division operation has received only moderate interest recently by Monolithic Memories Inc. [1], with their introduction of their 57508 and 57516 multiplier/divider chips, and in the past by Advanced Micro Devices [2] with their special function integrated circuit, the AMD 9511. Both of these device types do provide a means for hardware division, but both fall short of the operating speeds achieved by standalone multiplier circuits. The MMI 57516 device requires I to 3us to produce a 16-bit result, while the AMD 951 1 requires about 50 ps. various hardware techniques for high-speed division have been advanced in the literature but not commercially produced [3]. This paper describes an efficient method for generating the quotient of two binary numbers at speeds comparable to existing multiplier chips. The new method is labelled the "best line approach." The reciprocal of the denominator is first generated and used for subsequent multiplication by the numerator using a conventional multiplier chip. The best line method closely resembles a two-term Taylor series approximation in that the reciprocal is mechanized by forming the sum of a constant with the product of another constant and variable. It will be shown that a carefully designed best line divider is statistically similar to the reciprocal lookup table technique, and provides a binary divider that demands much less board area and power.

III. SOFTWARE DESCRIPTION

There are two simulation software used:
1.) Modelsim6.1e:
Modelsim provides simulation environment by mentor graphics and has a C debugger built in. It allows simulation of specific hardware description languages like SystemC, VHDL, Verilog etc. Modelsim can be utilized autonomously, or in conjunction with Xilinx or Altera Quartus. Simulation can be done automatically utilizing scripts or by utilizing GUI.
2.) Xilinx Vivado:
Vivado Design Suite is software created by Xilinx for analysis and synthesis of HDL outlines, superseding Xilinx ISE with extra components for framework on a IC advancement and high-performance synthesis. ISE depend on Modelsim in order to simulate, but this is not the case with this software. Vivado has incorporated an embedded logic simulator and it additionally

presents abnormal state analysis, using a tool that changes C code into a logic that is programmable.

IV. GOLDSCHMIDT DIVIDER

Goldschmidt division uses an iterative process of repeatedly multiplying both the dividend and divisor by a common factor F_i, chosen such that the divisor converges to 1. This causes the dividend to converge to the sought quotient Q. In this paper, we would examine a division algorithm called Goldschmidt's Algorithm, build hardware block diagram for it, and look for a suitable reuse of the hardware without losing sync with the global clock. Goldschmidt's Division Algorithm was improved in an excellent manner and examined by. It not only visited division but used this algorithm for Square Root and Square Root reciprocal also. The basic Idea of this algorithm from the lines developed can be explained as follows. Assume Numerator (N) and Denominator (D) to satisfy the constraint 1≤ N and D< 2 (considering them as normalized significands of floating point numbers). The basic division method is performed as follows: Quotient Q = N/D. The Goldschmidt's algorithm points at finding a sequence K1, K2, K3…. Ki such that the product ri = D. K1. K2. K3. K4. K5……. Ki approaches 1 as it goes to infinity. Thus we have qi = N. K1.K2. K3 .K4.K5……….. Ki → Q. K1 is obtained from the lookup table which is an optimal reciprocal table with p-bits in and p+2 bits out that uses D as input and obtains a p+2 bit approximation K1 to 1/D. The input bits and their accuracy considerations were met previously so we would rather go ahead with the basic idea needed to develop Hardware architecture and look more into it.

V. ALGORITHM USED

Given

Dividend:

N−1=8610=01010110.0000000000002N−1=8610=01010110.0000000000002

Divisor:

D−1=710=00000111.0000000000002D−1=710=00000111.0000000000002

Correct result:

Q=N−1D−1=12.285714285714285714285714285714285714285714Q=N−1D−1=12.285714285714285714285714285714285714285714

Initial reciprocal is the inverse of divisor which is calculated by shifting bits around the fraction point:

F−1=1D−1≈0.054687510=00000000.0000111000002F−1=1D−1≈0.054687510=00000000.0000111000002

In this case we have 8 bits for integers and 12 bits for fraction digits.

Algorithm steps

Iteration #1:

N0=F−1×N−1=4.70312510=00000100.1011010000002N0=F−1×N−1=4.70312510=00000100.1011010000002D0=F−1×D−1=0.382812510=00000000.0110001000002D0=F−1×D−1=0.382812510=00000000.0110001000002

F0=2−D0=1.617187510=00000001.1001111000002F0=2−D0=1.617187510=00000001.1001111000002

Iteration #2:

N1=F0×N0=7.60571289062510=00000111.1001101100012N1=F0×N0=7.60571289062510=00000111.1001101100012

D1=F0×D0=0.61889648437510=00000000.1001111001112D1=F0×D0=0.61889648437510=00000000.1001111001112
F1=2−D1=1.38110351562510=00000001.0110000110012F1=2−D1=1.38110351562510=00000001.0110000110012

Iteration #3:
N2=F1×N1=10.50415039062510=00001010.1000000100012N2=F1×N1=10.50415039062510=00001010.1000000100012
D2=F1×D1=0.85473632812510=00000000.1101101011012D2=F1×D1=0.85473632812510=00000000.1101101011012
F2=2−D2=1.14526367187510=00000001.0010010100112F2=2−D2=1.14526367187510=00000001.0010010100112

Iteration #4:
N3=F2×N2=12.02978515625100=00001100.0000011110102N3=F2×N2=12.0297851562510=00001100.0000011110102
D3=F2×D2=0.97875976562510=00000000.1111101010012D3=F2×D2=0.97875976562510=00000000.1111101010012
F3=2−D3=1.02124023437510=00000001.0000010101112F3=2−D3=1.02124023437510=00000001.0000010101112

Iteration #5:
N4=F3×N3=12.2851562510=00001100.0100100100002N4=F3×N3=12.2851562510=00001100.0100100100002
D4=F3×D3=0.9995117187510=00000000.1111111111102D4=F3×D3=0.9995117187510=00000000.1111111111102
F4=2−D4=1.0004882812510=00000001.0000000000102F4=2−D4=1.0004882812510=00000001.0000000000102

The result after 5 iterations is:
N5=12.2851562510=00001100.0100100100002N5=12.2851562510=00001100.0100100100002
which deviates from the correct result by
100%−|N5Q|=0.0045%

VI. IMPLEMENTATION RESULTS

RTL netlist of the Goldschmidt algorithm of fast division has been shown in fig 2. In this work Xilinx Vivado is used for the implementation of Goldschmidt algorithm of fast division where MODELSIM has been used for its simulation.

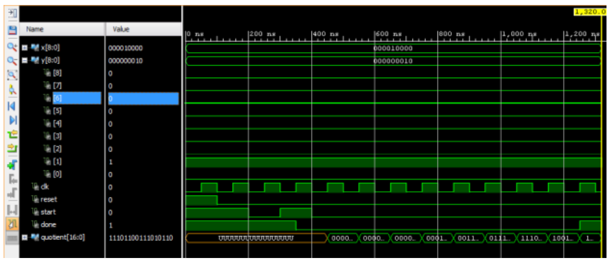

Fig1: Implementation of Goldschmidt Divider

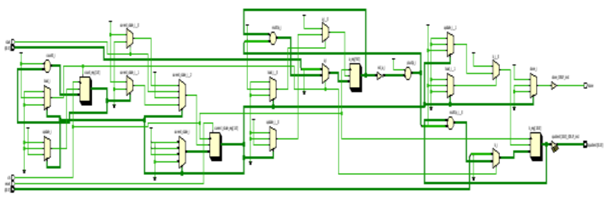

Fig2: RTL netlist of the above Binary Divider

Resource	Estimation	Available	utilization
FF	25	35200	0.07
LUT	37	17600	0.21
I/O	33	100	33.00
BUFG	1	32	3.12
DSP48	2	80	2.50

Fig3: Post-implementation summary

VII. CONCLUSION

Goldschmidt algorithm of fast division has been successfully implemented using Xilinx Vivado and the work summary has been successfully shown in the above diagrams.

VIII. FUTURE WORK

Goldschmidt's algorithm lead to the reduction in the total hardware used even if pipelining was used. Using partial pipelining and feedback, the result with the same accuracy level can be obtained. The topicality of this subject is growing with recent advances in various hardware technologies, as the structure density of the new devices grows and the price per element decreases and with rising demands for minimization of the current consumption. The synthesized model is capable of performing arithmetic operations of unsigned bits as well.

IX. REFERENCES

[1] Purushottam D. Chidgupkar and Mangesh T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", Global J. of Engng. Educ., Vol.8, No.2 © 2004 UICEE Published in Australia.

[2] Himanshu Thapliyal and Hamid R. Arabnia, "A Time-Area- Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics", Department of Computer Science, The University of Georgia, 415 Graduate Studies Research Center Athens, Georgia 30602-7404, U.S.A.

[3] E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, The University ofSouthwestern Louisiana Lafayette, LA 70504.

[4] Harpreet Singh Dhillon and Abhijit Mitra, "A Reduced- Bit Multiplication Algorithm for Digital Arithmetics", International Journal of Computational and Mathematical Sciences 2;2 © www.waset.org Spring 2008.

[5] Shamim Akhter, "VHDL Implementation of Fast NXN Multiplier Based on Vedic Mathematics", Jaypee Institute of Information Technology University, Noida, 201307 UP, INDIA, 2007 IEEE.

[6] Charles E. Stroud, "A Designer's Guide to Built-In Self-Test", University of North Carolina at Charlotte, ©2002 Kluwer Academic Publishers New York, Boston, Dordrecht, London, Moscow.

[7] Douglas Densmore, "Built-In-Self Test (BIST) Implementations An overview of design tradeoffs", University of Michigan EECS 579 – Digital Systems Testing by Professor John P. Hayes 12/7/01.

[8] Shripad Kulkarni, “Discrete Fourier Transform (DFT) by using Vedic Mathematics”, report, vedicmathsindia.blogspot.com, 2007.

[9] Jagadguru Swami Sri Bharati Krishna Tirthji Maharaja,“Vedic Mathematics”, Motilal Banarsidas, Varanasi, India, 1986.

[10] Himanshu Thapliyal, Saurabh Kotiyal and M. B Srinivas, “Design and Analysis of A Novel Parallel Square and Cube Architecture Based On Ancient Indian Vedic Mathematics”, Centre for VLSI and Embedded System Technologies, International Institute of Information Technology, Hyderabad, 500019, India, 2005 IEEE.