SINGLE PRECISION FLOATING POINT DIVISION

[1]NAJIB GHATTE, [2]SHILPA PATIL, [3]DEEPAK BHOIR

[1,2,3]Fr. Conceicao Rodrigues College of Engineering, Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai: 400 050, India

Abstract- Binary Division is one of the most crucial and silicon-intensive and of immense importance in the field of hardware implementation. A Divider is one of the key hardware blocks in most of applications such as digital signal processing, encryption and decryption algorithms in cryptography and in other logical computations. Being sequential type of operation, it is more prominent in terms of computational complexity and latency. This paper deals with the novel division algorithm for single precision floating point division Verilog Code is written and implemented on Virtex-5 FPGA series. Power dissipation has been reduced. Moreover, significant improvement has been observed in terms of area-utilisation and latency bounds.

Keywords- Single precision, Binary Division, Long Division, Vedic, Virtex, FPGA, IEEE-754.

I. INTRODUCTION

The term floating point implicates that there is no fixed number of digits before and after the decimal point; i.e. the decimal point can float. Floating-point representations are slower and less accurate than fixed-point representations, but can handle a larger range of numbers. Because mathematics with floating-point numbers requires a great deal of computing power, many microprocessors come with a chip, called a floating point unit (FPU), specialized for performing floating-point arithmetic. FPUs are also called math coprocessors and numeric co-processors. Floating-point representation has a complex encoding scheme with three basic components: mantissa, exponent and sign. Usage of binary numeration and powers of 2 resulted in floating point numbers being represented as single precision (32-bit) and double precision (64-bit) floating-point numbers. Both single and double precision numbers as illustrated in Fig. 1 are defined by the IEEE 754 standard.

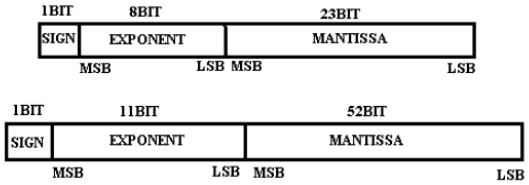


Fig. 1 IEEE-754 Floating-point Representation Standards

For a single precision format, 8-bits are reserved for exponent thereby having a bias value of +127 and 23 bits are reserved for mantissa.

When sign bit is 1, it indicates negative number and when it is 0, it argues as a positive number.

The similar explanation is extended for double precision format where exponents are biased to +1023. Binary division operation is of much significance in the area of hardware implementation of signal processing, high quality graphics rendering, filter applications, etc. It is the most important goal of a designer to enhance the performance of the ALU thereby reducing its design complexity to have better figure of merit. Many algorithms were proposed and implemented to reduce the latency of binary division. In algorithmic and structural levels, a lot of division techniques had been developed to reduce the latency of the divider circuitry; which reduces the iteration aiming to reduction of latency but the principle behind division was same in all cases.

Vedic math is known to more optimised and efficient than algorithms based on conventional logic. The sutras defined can be used in digital design to improve the performance of ALUs based on conventional logic. Nikhilam Navatashcaramam Dashatah Sutra and the Paravartya Sutra deals with the division. These sutras find its limitations when number of bits are increased as this paper deals with IEEE-754 floating-point representation.

II. VARIOUS DIVISION ALGORITHMS

Researchers have proposed many algorithms and procedural architectures to carry out division in order to reduce the computational time and thus enhancing the performance.

A. Restoring Division

Restoring division operates on fixed-point fractional numbers and depends on the following assumptions: (a) $D < N$ and (b) $0 < N$, $D < 1$.The quotient digits q are formed from the digit set {0, 1}. The basic algorithm for binary (radix 2) restoring division is:

To compute a/b, put a in register A, b in register B and 0 in register P.

1. Shift the register pair (P, A) one bit left.
2. Subtract the content of Register B from register P.
3. If the result of step 2 is negative, set the A_0 to 0, otherwise to 1.

4. If the result of step 2 is negative, restore the old value of P by adding the contents of register B back into P.

After repeating the algorithm, n times, register A will have quotient and register P with reminder.

B. Non-Restoring Division

Non-restoring division uses the digit set {−1, 1} for the quotient digits instead of {0, 1}. The basic algorithm for binary (radix 2) non-restoring division is:

If P is negative,
1a. Shift (P,A) one bit left.
2a. Add the content of register B to P.
else
1b. Shift (P,A) one bit left.
2b. Subtract the contents of register B from P.
3. If P is negative, set the low-order bit of A to 0, otherwise set it to 1.

Repeat the above procedure n times. After n cycles, register A will have the quotient and if P is positive, it is the remainder, otherwise it has to be restored (add B to it) to get the remainder.

C. SRT Division

Named for its creators (Sweeney, Robertson, and Tocher), SRT division is a popular method for division in many microprocessor implementations. SRT division is similar to non-restoring division, but it uses a lookup table based on the dividend and the divisor to determine each quotient digit. The basic algorithm for binary (radix 2) non-restoring division is:

1. Load a and b into A and B registers (Figure A.2)
2. If B has k leading zero, shift B and (P,A) left k bits
3. For l=0, n-1,
   a) If top 3 bits of P are equal, set qi=0 and shift (P,A) one bit left.
   b) If top 3 bits of P are not equal and P is negative, set q_i=-1 (write as $\bar{1}$) shift (P,A) one bit left, add B.
   c) Otherwise, set q_i=1, shift (P,A) one bit left, sub B.
4. If final remainder is negative, correct the remainder by adding B; correct the quotient by subtracting 1 from q0.
5. Shift remainder k bits right.

D. Vedic Architecture: Nikhilam Sutra

Vedic Sutra, Nikhilam can be further extended to carry out Binary Division as an alternative to conventional algorithm. Assume that, A and B are dividend and divisor respectively. Dividend is n-bits wide. The flowchart diagram can be executed as follows:

a) Initialize the incrementer with '0'.
b) Determine the complement B with respect to 2n assume the complemented result is equal to B'.
c) Add B' with A. If the carry is '1', then feed the result to the adder.
d) Increment the content of the incrementer by one.
e) Repeat step-3 until the result is less than B.
f) The final result of the incrementer is the quotient and result from the adder is the remainder.

III. PROPOSED DIVISION ALGORITHM

This paper deals with the efficient algorithm which incorporates the positive attributes of the division architectures mentioned. The division algorithm find its limitations with the wide increase in the number of bits as this paper deals with the floating point representation where mantissa of dividend and divider are each 23-bit wide.

The basic algorithm for the proposed design as follows:

First and foremost, the sign-bit of result is obtained by performing logical XOR of sign bits of dividend and divisor. The Exponent is evaluated by subtracting 8-bit exponent of divisor from 8-bit dividend and then adding the bias value (+127). Lastly, 23-bit mantissa of the result is computed using division logic for which stepwise procedure enlisted below.

1. Pad leading zeroes before the dividend making its width, double the width of mantissa.
2. Pad a leading zero before a divisor and trailing zeroes after the divisor making its width, double the width of mantissa.
3. Perform subtraction of dividend and divisor.
4. If the difference is negative Put 0 in the quotient else 1.
5. Right Shift divisor
6. Left Shift quotient

Fig. 2 shows the example based on proposed algorithm where a decimal number 12 is divided by decimal number 3. Architectural block diagram of the proposed design is shown in Fig. 3.

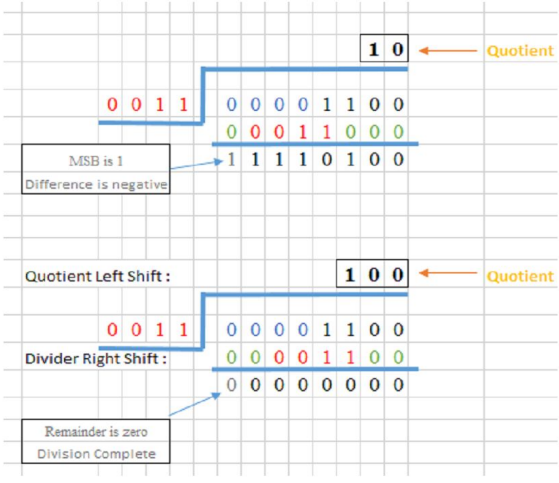


Fig. 2 Floating point Vedic Divider Illustration 12 divided by 3

IV. DESIGN IMPLEMENTATION

Verilog HDL code for Division of IEEE-754 Single Precision Numbers is being developed and then is simulated using ModelSim SE Plus 6.5. Verilog HDL code was break down into modules which deals with the division of 23-bit dividend and 23-bit divisor. code is also written for exponent addition and normalisation process. Top module connects all of them as shown in Fig. 4. Various sets of inputs are fed to the top modular block to get the results. The further part of the document deals with simulation and synthesis results.

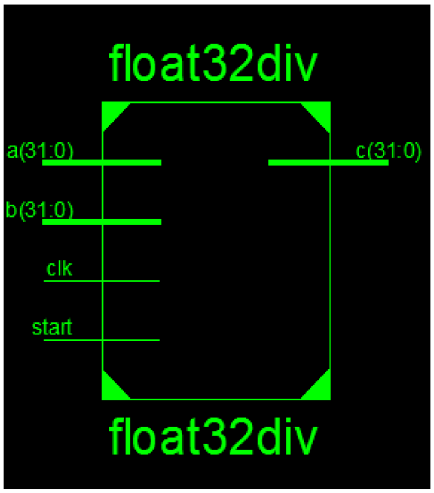


Fig. 4 Single Precision Vedic Divider: Block Diagram

A. ModelSim Simulation

Consider division of two decimal numbers, 73.68 and 73.68 fed as A = 73.68 (0 10000101 00100110101110000101001) and B = 73.68 (0 10000101 00100110101110000101001) were fed to the algorithm to get the desired output as C = 1 (0 01111111 00000000000000000000000) as shown in Fig. 5.

B. Xilinx ISE Synthesis

Verilog HDL Code for division of IEEE-754 Single Precision (32-bit) numbers are then synthesized for device XC5VLX30 having package as FF324 of Virtex™-5 FPGA family. From the datasheet cited in, this device has following attributes manifests in Table I.

TABLE I
XILINX VIRTEX™-5 XC5VLX30 ATTRIBUTES

Device	CLB Array (One CLB = Four Slices × 2			Total Slices	Max. User I/O
	Rows	Column	Total		
xc5vlx30	80	30	4800	19,200	220

Table II shows the Device Utilisation Summary of the Verilog HDL code, so written, it is been observed that number of device parameters used are very less. Hence, an optimum Device Utilisation is obtained.

From the timing report obtained, it is found that the maximum combinational path delay is 5.405 ns. Maximum combinational path delay is only for paths that start at an input to the design and go to an output of the design without being clocked along the way. Also, it is estimated that it can operate with the maximum frequency of 326.163 MHz.

V. PERFORMANCE ANALYSIS: VEDIC VERSUS CONVENTIONAL

It can be easily deduced from the Table III which shows the performance analysis of the Vedic divider with that of conventional dividers that Vedic dividers are more optimistic.

TABLE II
FLOATING POINT VEDIC DIVISION (SINGLE PRECISION):

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	120	19,200	1%
Number used as Flip Flops	120		
Number of Slice LUTs	159	19,200	1%
Number used as logic	159	19,200	1%
Number using O6 output only	155		
Number using O5 and O6	4		
Number of occupied Slices	54	4,800	1%
Number of LUT Flip Flop pairs used	182		
Number with an unused Flip Flop	62	182	34%
Number with an unused LUT	23	182	12%
Number of fully used LUT-FF pairs	97	182	53%
Number of unique control sets	2		
Number of slice register sites lost to control set restrictions	4	19,200	1%
Number of bonded IOBs	98	220	44%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	2.37		

TABLE III
PERFORMANCE ANALYSIS:
PROPOSED WORK (VEDIC) VERSUS CONVENTIONAL

Parameters	This Work	[6]	[9]	
Number of bits	23	9	M	
Target Device	Virtex 5 XC5VLX30 -3 FF324	Spartan 3 XC3S100E -5 VQ100	Spartan 3E XC3S500E	
Number of Slices	54	103	Restoring	Non - Restoring
			229	447
Number of 4-input LUTs	159	176	---	---
Number of IOBs	98	38	---	---
Maximum Frequency (MHz)	326.163	---	12.63	53.77

Number of slice LUTs used in the design are pretty less, 159 whereas other algorithms based dividers are normally hooked to 176. Also one can visualize that this proposed divider is just consuming 54 number of slices compared to other algorithms which utilize minimum of 103 to 447 number of slices. Thus, there is near about 12% improvement in the utilisation of

the resources, thereby enhancing the packaging density. This proposed design can be operated on the processor clocked with 326.16 MHz, Thus, computational speed is more as compared to other design which operates with maximum frequency of 53.77 MHz. The maximum combinational delay incurred is just a meagre value of 5.405 nanoseconds. Power consumption of proposed design is drastically reduced to 2 mW, whereas traditional ALUs are consuming more power. Since, proposed design consumes less power, so bulky heat-sinks or extensive blowers can be eliminated. This makes the installation of processor simple and can be equipped with portable or mobile devices.

CONCLUSION

The importance and usefulness of floating point format nowadays does not allow any discussion. Any computer or electronic device, which operates with real numbers, implements this type of representation and operation. Division is one of the most important arithmetic operation. Various architectures were proposed which include recursive iterations to increase the computational performance of the ALU. The proposed design uses ancient Math technique which pad zeroes for dividend. By means of logical shift, the estimated results were obtained. Synthesis results obtained provides better results in terms of computational speed and area utilization thereby reducing the size of silicon dye used. Also reduces the size and cost of the device. Moreover, less time delay incurred leads to fast computation and speedy calculations. Power dissipation has been reduced, which can eliminate the need of extensive cooling mechanism comprising thermal gels and blowers. Thus, proposed divider is more efficient than traditional ALUs and serve as better optimisation technique as per today's need and wants.

REFERENCES

[1] Alex N. D. Zamfirescu, "Floating Point Type for Synthesis", CA USA, 2000.

[2] Prabir Saha, Arindam Banerjee, Partha Bhattacharya, Anup Dandapat, "Vedic Divider: Novel Architechture (ASIC) for High Speed VLSI Applications" in International Symposium on Electronic System Design, pp. 67-71, Oct. 2011.

[3] Deepa and Sanal's (2005) Vedic Mathematics [Online]. Available: http://www.sanalnair.org/articles/vedmath/intro.htm

[4] Diganta Sengupta, Mahamuda Sultana, Atal Chaudhuri, "A New Paradigm in Fast BCD Division using Ancient Indian Vedicmathematics Sutras" pp. 11-19, 2013.

[5] Sherif Galal, Dung Pham, "Division Algorithms and Hardware Implementations" in EE 213A: Advance DSP Circuit Design, 2000.

[6] Sukhmeet Kaur, Suman, Manpreet Singh Manna, Rajeev Agarwal, "VHDL Implementation of Non Restoring Division Algorithm using High Speed Adder/Subtractor" in International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering" Vol. 2, Issue 7, pp. 3317-3324, July 2013.

[7] R. Tamil Chelvan and S. Roobini Priya, "Implementation of Fixed and Floating Point Division using Dhvanjanka Sutra" in International Journal of VLSI and Embedded Systems, pp. 234-237, Apr. 2013.

[8] Virtex-5 Family Overview datasheet, Xilinx 2009.

[9] Bojan Jovanovic, Milun Jevtic, "FPGA Implementation of Throughput Increasing Techniques of the Binary Dividers" in International Scientific Conference, Gabrovo, Nov. 2010.

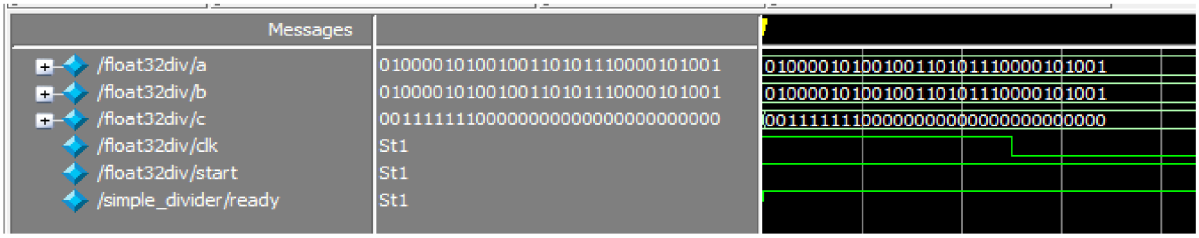
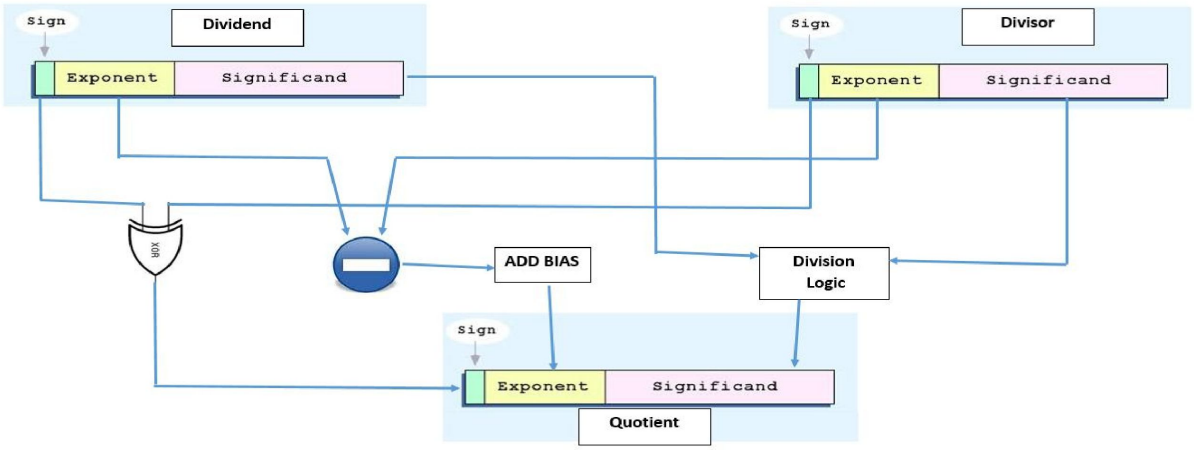


Fig. 5 Floating Point Vedic Division (Single Precision): Timing Diagram 73.68 ÷ 73.68 = 1



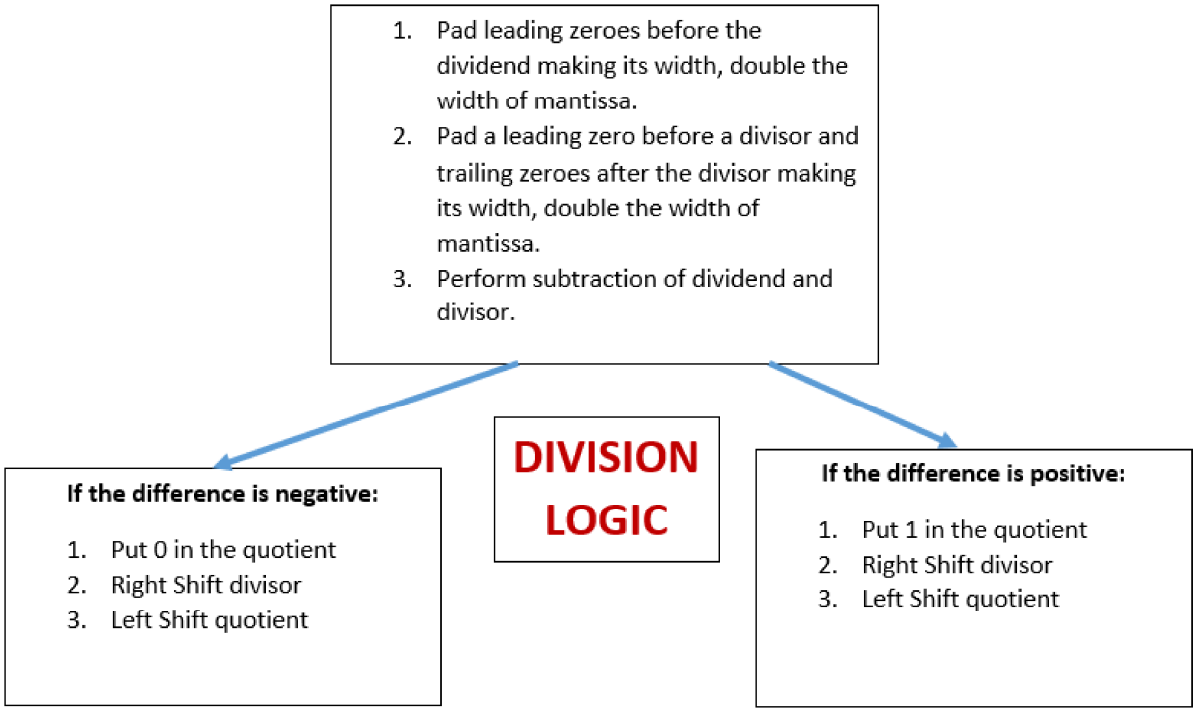


Fig. 3 Floating Point Vedic Division: Architectural Block Diagram

★★★