Comparative study of single precision floating point division using different computational algorithms

Naginder Singh, Kapil Parihar
Department of Electronics and Communication Engineering, M.B.M. University, Jodhpur, India

Article Info

Article history:

Received Jun 25, 2022
Revised Dec 10, 2022
Accepted Mar 20, 2023

Keywords:

Field programmable gate array
Floating point division
Goldschmidt algorithm
Newton-Raphson algorithm
Vedic-multiplication

ABSTRACT

This paper presents different computational algorithms to implement single precision floating point division on field programmable gate arrays (FPGA). Fast division computation algorithms can apply to all division cases by which an efficient result will be obtained in terms of delay time and power consumption. 24-bit Vedic multiplication (Urdhva-Triyakbhyam-sutra) technique enhances the computational speed of the mantissa module and this module is used to design a 32-bit floating point multiplier which is the crucial feature of this proposed design, which yields a higher computational speed and reduced delay time. The proposed design of floating-point divider using fast computational algorithms synthesized using Verilog hardware description language has a 32-bit floating point multiplier module unit and a 32-bit floating point subtractor module unit. Xilinx Spartan 6 SP605 evaluation platform is used to verify this proposed design on FPGA. Synthesis results provide the device utilization and propagation delay parameters for the proposed design and a comparative study is done with previous work. Input to the divider is provided in IEEE 754 32-bit formats.

Corresponding Author:

Naginder Singh
Department of Electronics and Communication Engineering, M.B.M. University
Jodhpur, Rajasthan, India
Email: naginder.singh0110@gmail.com

1. INTRODUCTION

Field programmable gate array (FPGA) provides a very versatile high computational environment and currently plays an important role due to the advancement of technology process or architectures modified instantaneously. For the proposed design the cost per logic cell is reduced voluntarily because of the enhanced feature of Spartan 6. In many digital processing systems for many arithmetic and logical units, floating-point dividers play a crucial role. To design a divider power consumption and throughput pays major factors for processing the information. This paper is based on computational algorithms to design a single precision floating point divider; in which multiplication and subtraction are the basic mathematical operations as described earlier in [1]-[3]. The arithmetic operations such as multiplication and subtraction modules are used to implement and design Goldschmidt and Newton-Raphson algorithm for the proposed 32-bit floating point division [4]. For multiplication Urdhva-Tiryagbhyam-sutra is an ancient Vedic mathematics method, this algorithm reduces the number of steps required for multiplication and can increase the speed of multiplication [5]. In this proposed design of divider, the module to compute mantissa in the multiplication unit is designed using medic mathematics (Urdhva-Tiryagbhyam-sutra). A module of 16*16 Vedic multiplier implemented by using the concept Urdhva-Tiryagbhyam-sutra and Nikhilam-sutra multiplication techniques [6]-[9]. In this paper the exponent module used in the floating-point multiplier uses a ripple carry adder to provide a low area and simple layout [10]. Binary floating point multiplier implementation and blocks described and Urdhva-Tiryagbhyam-sutra used to achieve high speed [11]. In designing of processor, arithmetic-logic unit (ALU) unit arithmetic operations are done by addition, subtraction, and multiplication Vedic algorithm plays an important role [12]. Nikhilam-sutra based multiplier is designed to get high speed as the computational speed gained by Vedic mathematics described [13]. High speed squaring circuit for binary numbers [14] is proposed based on Vedic mathematics. To perform signal processing Goldschmidt computational algorithm-based division provides the high computational speed and throughput described [15] and it's suitable for high computational demanding applications. Power consumption plays an important role and is reduced efficiently by designing a multiplier using Vedic multiplication which reduces the required look up tables and slices resulting in reduced power consumption [16]. A floating-point multiplier is implemented which is designed for efficient power consumption and device utilization is reduced [17].

In digital signal processing application encryption and decryption algorithms in cryptography; a divider is one of the most important hardware blocks which is applicable in most applications [18] and also in various mathematical computational algorithms used to design floating-point divide-add fused (DAF) architecture [19]. The floating-point DAF architecture has a dedicated unit for floating point division followed by addition or subtraction so that a combined operation is performed [20]. The delay time and better device utilization which is an important factor and have to optimize are done by a division architecture based on the Vedic mathematics algorithm Parvartya sutra [21]. The restoring division is described for sixteen-bit division and well explained its implementation of restoring division algorithm in encryption system [22]. Floating point architectures have low frequency, larger area and high latency in nature described [23]. A reversible Vedic multiplier is designed using reversible logic gates and results in less delay and less power consumption [24]. High computational speed and throughput provide by Newton-Raphson and Goldschmidt computational division algorithm methods both converge much faster within one iteration to produce the final quotient, so these computational algorithms in a single iteration generate twice as many digits to find the final quotient as they start with a close approximation to find the final quotient [25]. To represent 32-bit and 64-bit floating point numbers IEEE 754 standard format is used [26]. Booth-based multiplication is used to pre-calculate the carry and the presented design improved the speed and dissipated less power with respect to the existing multipliers [27]. A fast-floating point unit is designed and implemented for FPGAs [28]. As floating-point number is separated into three parts where a fixed number of bits is used to represent these three parts that are sign, exponent and mantissa. To describe single and double precision the structure of the IEE 754 format is shown in Table 1. For single precision IEEE 754 format, 23 bits are used to represent mantissa, exponent part is represented by 8-bits and the sign bit is represented by the most significant bit (MSB) part where the sign of the floating-point number depends on the MSB bit, floating point number is negative when MSB is 1 else the number is positive when MSB bit is 0. For division purposes in many signals processing algorithms complex iterative process is used so, not only is the precision to be maintained but it has to be maintained for very large data intervals and is achieved by using Newton-Raphson and Goldschmidt computational algorithms.

This paper presents implementation using Goldschmidt and Newton-Raphson computational algorithm. To design of 32-bit floating point division which uses arithmetic operations such as 32-bit floating point multiplication module and floating-point subtraction modules and a 24-bit Vedic multiplication used for computing mantissa in 32-floating point multiplier module in the proposed designed. The paper describes the implementation and design of Newton-Raphson and Goldschmidt computational algorithms single precision floating point divider on FPGA.

The basics of IEEE 754 floating-point representation are discussed: in section 2 and the division algorithms which are i) Newton Raphson and ii) Goldschmidt computational algorithm methods are discussed in section 3. The implementation of Vedic multiplication technique base 32-bit floating point multiplier is discussed in section 4. Floating point sub-tractor describes in section 5. The simulation results in 6 and conclusion describes is in section 7.

2. FLOATING POINT STANDARD

IEEE 754 standard representation for binary floating-point numbers [26]. The binary number represents the single and double precision formats, 32-bit represented by single precision and 64-bit represented by double precision. Single and double precision formats of IEEE 754 structure are shown in Table 1 sign, exponent and mantissa are 3 fields' that present a fixed number of bits represented in IEEE 754 format. In single precision, the mantissa part represents by 23-bits and for normalization 1-bit is added to the MSB, the exponent part represents by 8-bits, and it is biased to 127, and the exponent part is represented in excess of 127-bit format and the sign bit represents by 1 bit or MSB. The sign bit or MSB represents the sign of the floating-point number, hence the floating-point number is negative when MSB is 1 else the number is positive when the MSB bit is 0.

Table 1. IEEE 754 format representation of 32-bit and 64-bit

Precision	Sign	Exponent	Mantissa
Single precision (32 bit)	1-bit	8-bit	23-bit
Double precision (64 bit)	1-bit	11-bit	52-bit

3. DIVISION ALGORITHMS

The division operation requires a very high number of iterations and is implemented as an inverse process of the multiplication operation. When combined with shift operation the number of iterations is reduced to the number of result valid bits. The fast algorithm was developed to produce a more precise result that requires more iteration and for this purpose algorithms that were developed are Goldschmidt and Newton-Raphson division algorithms. These algorithms within a single iteration converge much faster however; several additions or subtractions and multiplications operations are needed. The Goldschmidt division computational algorithm goal to converge the denominator to 1 is achieved by continual multiplication of the numerator and denominator by the same factor Fi. The quotient produced as a result directly converges the numerator at the same time. By calculating the multiplicative inverse of denominator which is produced by multiplied numerator to produce the result and this process is based on Newton-Raphson division computational algorithm.

3.1. Goldschmidt algorithm

A complex initialization is used to compute continual iterations for the Goldschmidt computational division algorithm. To get the resultant quotient this algorithm used a common factor G_i, this common factor continually multiplying both the dividend and divisor that converge the divisor (D) to one is an iterative process used by this algorithm [25]. In this proposed floating point divider design to perform arithmetic operations, a 32-floating point multiplier module and subtractor module are used in Goldschmidt computational division algorithm. In this division algorithm, the divisor should be in the interval (0, 1) by scaling the numerator and denominator, shifting operation is used for scaling the divisor. The fast division algorithms are developed to produce a precise result for which more iteration is required. For a single iteration, the Goldschmidt algorithm converges much faster for computing the result to perform the continual iteration process thus; more multiplication and subtraction operations are needed. This computational algorithm produces an optimized result by computing iterations in one cycle. As shown in (1) where N is the numerator, Q is a quotient, D is the denominator and G_i is factoring at iteration i.

$$Q = \frac{N}{D} \frac{\prod_i^n G_i}{\prod_i^n G_i} \tag{1}$$

The G_{i+1} is the factor for the next iteration calculated using (2) and that is used to substitute in (1) for calculating the further, iterations. For the next iteration, the numerator and denominator have to be calculated and it’s done by (3).

$$G_{i+1} = 2 - D_i \tag{2}$$

$$\frac{N_{i+1}}{D_{i+1}} = \frac{N_i}{D_i} \frac{G_{i+1}}{G_{i+1}} \tag{3}$$

Floating point division is based on the Goldschmidt computational algorithm where inputs N and D are the numerator and the denominator given as input for division. To calculate one iteration Goldschmidt's computational algorithm uses one multiplier and one subtraction module. This computational algorithm scales the numerator and denominator by the common factor G_i, in four iterations this algorithm computes directly and produces the quotient in the result. Hence, the numerator converges directly to the quotient as the denominator converges to one. For more refined results, more iterations are performed. F1, F2 and F3 are the three floating point input numbers the resultant division of F1 by F2 is followed by the addition or subtraction of F3 this operation performed by the Goldschmidt computational algorithm based proposed floating-point divider is carried out as shown in the simulation results.

3.2. Newton-Raphson algorithm

Newton-Raphson division algorithm, this computational algorithm starts with the computation of the multiplicative inverse; the multiplicative inverse is calculated by an iterative process and the resultant quotient is found by l multiply dividend and the calculated multiplicative inverse. The shifting operation is used for scaling the divisor. To get more precise results more iterations are required and for this purpose, fast division algorithms are used. Newton-Raphson computational algorithm-based floating point division flowchart, using this algorithm to compute iteration converges much faster. Thus, for getting precise results number of iterations required and to perform continual iterations several multiplication and subtraction operations are required.

In this computational algorithm, an optimized result can be achieved in one cycle by computing three iterations, for which two multipliers and a subtraction module are for iteration. More iteration is performed to refine the multiplicative inverse. The Newton-Raphson division algorithm is based on calculating of reciprocal of the denominator and producing the result reciprocal of the denominator multiplied by the numerator. The formula to calculate the multiplicative inverse of the denominator) where Z_i is the multiplicative inverse at iteration i is given by (4) in which D is the denominator.

$$Z_{i+1} = Z_i + Z_i(1 - DZ_i) = Z_i(2 - DZ_i) \tag{4}$$

In Newton-Raphson division by scaling the denominator, minimization of maximal relative error can be achieved in the interval [0.5, 1]. The Newton-Raphson division uses complex initialization which requires addition, multiplication and subtraction operations and therefore this initialization has been classified as the first iteration cycle. Z_0 represents the initial iteration and its initialization is carried out by (5).

$$Z_0 = \frac{48}{17} - \frac{32}{17}D \tag{5}$$

4. FLOATING POINT MULTIPLIER

Figure 1 shows the architecture of the proposed multiplier module which used Vedic multiplication, so (24x24) Vedic multiplier is used for calculating the mantissa part which enhances the overall performance of the proposed 32-bit multiplier. The multiplication module unit is divided into three parts for two inputs, input A [31-0] and B [31-0] which are in 32-bit floating point number IEEE 754 format for multiplication, this format describes a fixed number of bits i.e.; sign (s1 and s2), exponent (e1, e2) & mantissa (m1, m2).

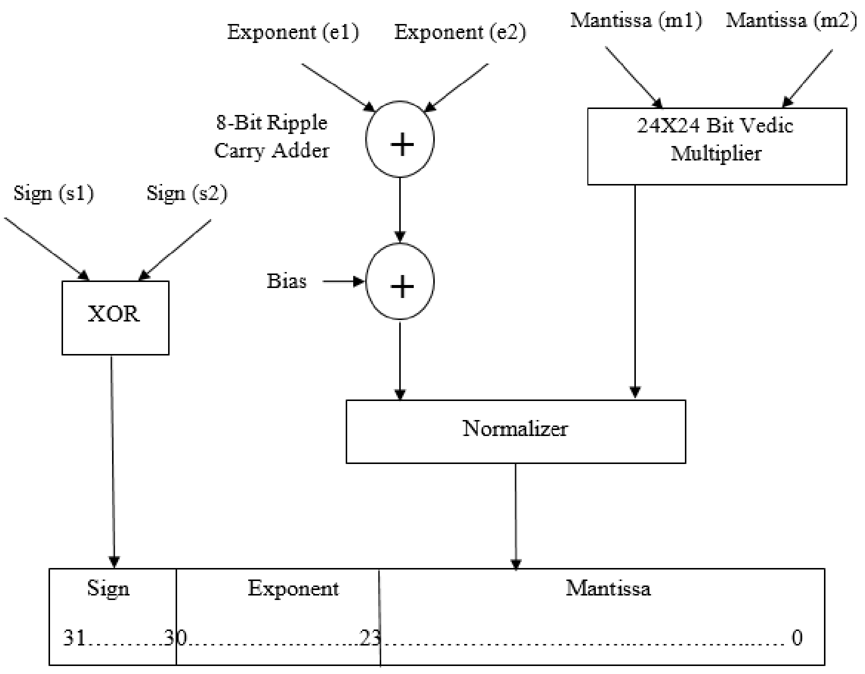


Figure 1. The proposed design for 32-bit floating point multiplier (FPM)

4.1. Mantissa unit

For getting high computation speed in this module for multiplication of the mantissa unit, lower bits of inputs m1 and m2 that is A [22-0] and B [22-0]. Calculation is done using (24x24)-bit Vedic multiplier and produces an output which is 24-bit normalized output which has leading 1 as their MSB. This multiplier module efficiently used Vedic mathematics which also provides higher throughput.

4.2. Exponent unit

A ripple carry adder is used in the exponent unit for the exponent calculation, e1 and e2 that is A [30-23]) and B [30–23]. A [30-23]) and B [30–23] are the inputs given to the 8-bit ripple carry adder unit which computes and the result is biased to 127. The result by this exponent unit the two cases overflow and underflow are carefully handled so that to get the optimized result.

4.3. Sign unit

The two inputs s1 and s2 that A [31] and (B [31] is the 31st bit of the 32-bit floating point numbers are given to the sign unit and computed by XORing s1 and s2 and the output of the exclusively-OR (XOR) gate is the sign floating point number. Both Goldschmidt and Newton-Raphson computational algorithms were designed using this multiplier module which is implemented using Vedic mathematics for getting efficient results. If 32-bit single precision IEEE-754 format is considered then a 24-bit Vedic multiplier is used to calculate the mantissa. To calculate the exponent, an 8-bit ripple carries adder is used, this implementation results in a low–area, ease of implementation and simple layout. The input to the multiplier for multiplication is provided in standard format for single precision which is in IEEE 754 format.

5. FLOATING POINT SUBTRACTOR

In the substractor module X [31-0] and Y [31-0] are the inputs given to the floating point substractor. The floating-point inputs to the substractor module are separated as a sign, exponent and mantissa and they are represented in IEEE 754 format further, the substraction operation is done in a stepwise manner. Initially, to perform the substraction operation the floating-point number is unpacked so that the sign (b1, b2), exponent (e1, e2) and mantissa (s1, s2) are identified for both the inputs. Then the alignment is done to equalize the exponent and normalization is performed in the mantissa part. By comparing the mantissa m1 and m2 the relation between inputs exponent e1 and e2 which are (X [30-23]) and (Y [30-23]) is determined if neither of the operands is infinite. After this the shifting operation is done on mantissa until the exponents e1 and e2 that is becomes equal that is X [30-23]) = Y [30-23]. Further, the mantissa m1 and m2 are substracted that is X [23-0] and (Y [23-0] after alignment and normalization process and after this resultant value of mantissa are rounded off. Finally, the concatenation is done to the sign, exponent and mantissa parts. The result of this concatenation is the single precision floating point number. The complete architecture of the 32-bit floating point substractor module is shown in Figure 2 where X and Y are the inputs given to substractor and this is used to calculate the iterative process computes in the fast computational division algorithms that are the Goldschmidt algorithm and Newton-Raphson algorithm for getting the optimized division result.

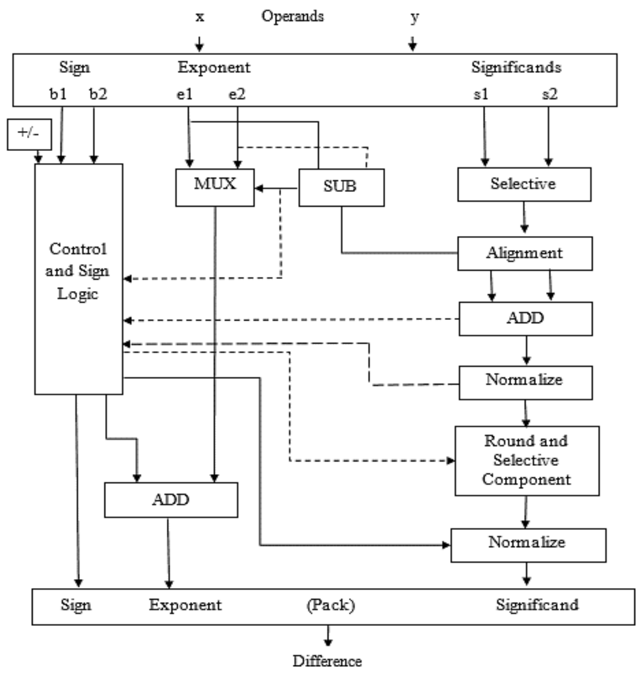


Figure 2. Architecture of floating point substractor (FPS)

6. SIMULATION RESULT

ISim simulator is used for performing simulations and the Xilinx Spartan 6 SP605 evaluation platform is used to implement and synthesize the proposed floating point divider design which is based on Vedic mathematics. The two numbers (N, D) are given as input for the division to 32-bit floating point divider. Table 2 shows the sample inputs (N, D) and the resultant output quotient (Q) of the sample inputs for simulation for 32-bit floating point division.

Table 2. The sample inputs (N, D) and the resultant output (Q) of the sample inputs

Parameter	Decimal	Sign	Exponent	Mantissa
N	3.14	0	10000000	10010001111010111000011
D	8.56	0	10000010	00010001111010111000011
Q	0.366822	0	01111101	01110111101000000100110

The device utilization parameters of the Xilinx power estimator (XPE)-14.3 are shown in Table 3 and Table 4 for the Goldschmidt and Newton-Raphson computational algorithm-based 32-bit floating point division implemented on the Xilinx Spartan 6 SP605 valuation platform FPGA. The device utilization parameters such as the number of slice registers, look-up tables (LUTs), occupied slices and bounded input and bounded output utilization are efficiently less utilized. By using Goldschmidt-based single floating-point divider with respect to Newton-Raphson based single precision floating point divider as shown in device utilization parameter Table 3 and Table 4.

Table 3. Device utilization parameters Goldschmidt computational algorithm

Logic utilization	Used	Available	Utilization
Slice Registers Nos.	408	54,576	0.7%
Slice LUTs Nos.	9,185	27,288	33%
Occupied Slices Nos.	2,913	6,822	42%
Bonded IOBs NOs.	193	296	65%

Table 4. Device utilization parameters for Newton-Raphson computational algorithm

Logic utilization	Used	Available	Utilization
Slice Registers Nos.	556	54,576	1%
Slice LUTs Nos.	11,584	27,288	42%
Occupied Slices Nos.	3,719	6,822	54%
Bonded IOBs NOs.	257	296	86%

The Table 5 compares the performance of the proposed DAF units using two computational algorithms, Newton-Raphson and Goldschmidt, in terms of power consumption and latency time. The table clearly indicates that the Goldschmidt-based DAF performs better in both aspects compared to the Newton-Raphson-based DAF. Therefore, the statement correctly concludes that the Goldschmidt-based DAF gives better performance in terms of latency and power with respect to the others.

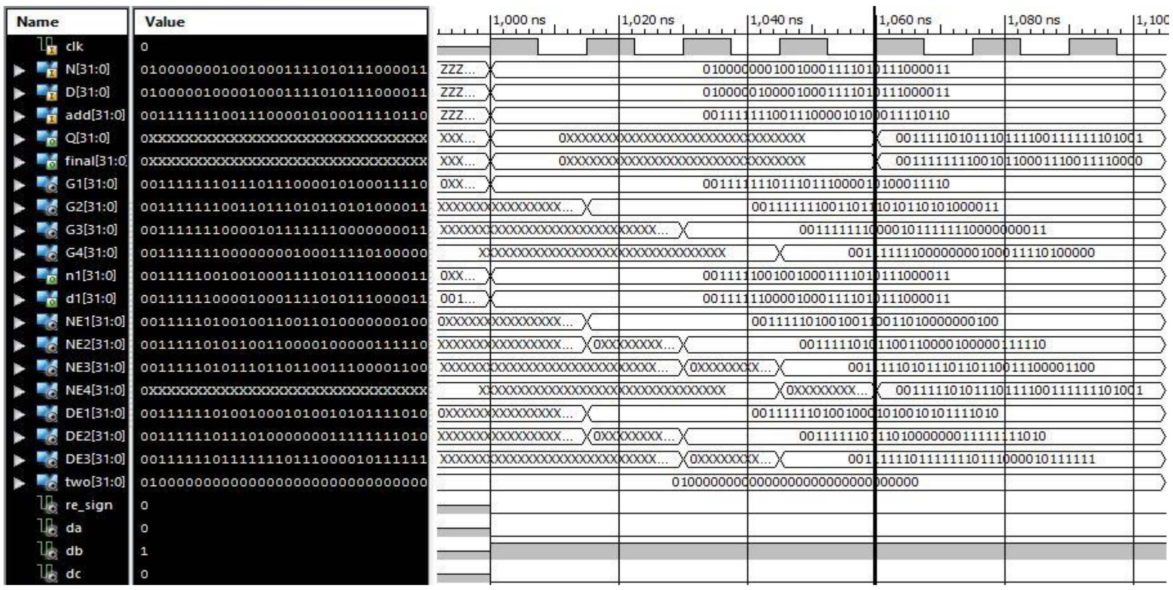
Table 5. Comparative analysis of DAF unit in terms of power and latency time

S.No.	Jeevan *et al.* [9]	Nagendra *et al.* [10]	Using DAF Newton-Raphson	Using DAF Goldschmidt based
Latency time	175.49 ns	130.8 ns	110 ns	75 ns
Power	-	0.050 W	0.057 W	0.031 W

The device summary reports of XPE-14.3 for the proposed 32-bit floating point division using Goldschmidt and Newton-Raphson computational algorithm is shown in Table 6 where Goldschmidt-based division parameters such as junction temperature, on-chip power, thermal margin and ambient temperature (ϴJA) are less as compared to the Newton-Raphson based division. The synthesis and implementation is done on Spartan-6 (SP605) evaluation platform field programmable gate array of the proposed 32-bit floating point divider. Figures 3 and 4 show the simulation results where Q is used to represent the quotient and the two numbers N and D represent the numerator and denominator given as input in IEEE-754 format to 32-bit floating-point divider given as input to the proposed 32-bit floating point divider which is designed using Goldschmidt and Newton-Raphson computational algorithms.

Table 6. Device summary report of XPE-14.3 on Spartan-6 SP605 evaluation platform FPGA of floating point divider

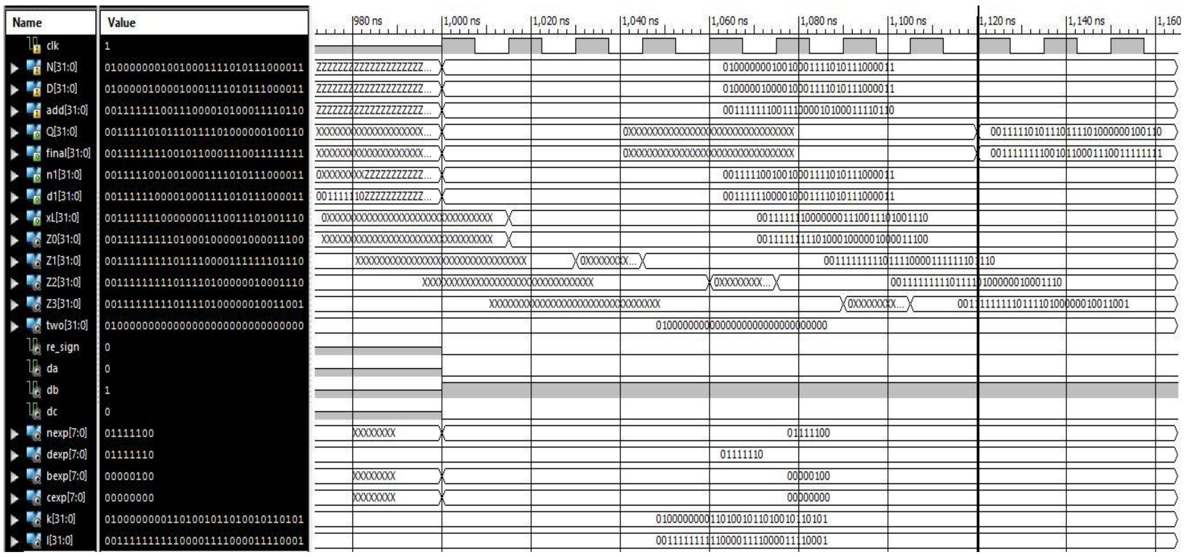
	Using Goldschmidth algorithm	Using Newton-Raphson algoroitm
Specifications	Values	Values
Junction temperature (ºC)	22.5	25.5
On-Chip power (Total)	0.031W	0.037 W
Thermal margin	58.5 ºC (3.8 W)	59.5 ºC (4.0 W)
Effective (ѲJA)	13.3 ºC/W	14.3 ºC/W



```
Name          Value
clk           0
N[31:0]       01000000010010001111010111000011
D[31:0]       01000001000010001111010111000011
add[31:0]     00111111100111000010100011110110
Q[31:0]       0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
final[31:0]   0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
G1[31:0]      00111111101110111000010100011110
G2[31:0]      00111111100110111010110101000011
G3[31:0]      00111111100001011111110000000011
G4[31:0]      00111111100000000100011110100000
n1[31:0]      00111110010010001111010111000011
d1[31:0]      00111111000010001111010111000011
NE1[31:0]     00111110100100110011010000000100
NE2[31:0]     00111110101100110000100000111110
NE3[31:0]     00111110101110110110011100001100
NE4[31:0]     0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
DE1[31:0]     00111111010010001010010101111010
DE2[31:0]     00111111011101000000011111111010
DE3[31:0]     00111111011111110111000010111111
two[31:0]     01000000000000000000000000000000
re_sign       0
da            0
db            1
dc            0
```

Figure 3. Goldschmidt computational algorithm based floating point division simulation result

In Figure 3 for Goldschmidt-based single precision division computations G1, G2, and G3 are used to represent the first, second, and third iteration results and the final iteration result is represented by G4. In Figure 4 for Newton-Raphson-based single precision division computations. The initial value represented by Z0, the Z1, and Z2 are used to represent the first and second iteration results, and the final iteration result is represented by Z3.



```
Name          Value
clk           1
N[31:0]       01000000010010001111010111000011
D[31:0]       01000001000010001111010111000011
add[31:0]     00111111100111000010100011110110
Q[31:0]       00111110101110111101000000100110
final[31:0]   00111111110010110001110011111111
n1[31:0]      00111110010010001111010111000011
d1[31:0]      00111111000010001111010111000011
xL[31:0]      00111111100000000111001110100111O
Z0[31:0]      00111111111010001000001000011100
Z1[31:0]      00111111111011110000111111101110
Z2[31:0]      00111111111011110100000010001110
Z3[31:0]      00111111111011110100000010011001
two[31:0]     01000000000000000000000000000000
re_sign       0
da            0
db            1
dc            0
nexp[7:0]     01111100
dexp[7:0]     01111110
bexp[7:0]     00000100
cexp[7:0]     00000000
k[31:0]       01000000001101001011010010110101
l[31:0]       00111111111100001111000011110001
```

Figure 4. Newton-Raphson computational algorithm based floating point division simulation result

7. CONCLUSION

Goldschmidt and Newton–Raphson computational algorithm-based single precision floating point division were synthesized and implemented using Xilinx Spartan 6 SP605 evaluation platform on FPGA and simulations of the proposed module were done on ISim simulator. For high computational arithmetic operations design presented in this paper is useful. Floating point divider performance is improved by designing multiplier module used in it using Vedic mathematics so, high computational speed and throughput are achieved. The device utilization parameters are optimized which results in an improvement in power consumption which is reduced by 26% and latency time reduced by 42.6% respectively with respect to existing DAF using Goldschmidt-based single precision floating point division.

REFERENCES

[1] A. Parker and J. O. Hamblen, "Optimal value for the Newton-Raphson division algorithm," *Information Processing Letters*, vol. 42, no. 3, pp. 141–144, May 1992, doi: 10.1016/0020-0190(92)90137-K.

[2] I. Kong and E. E. Swartzlander, "A rounding method to reduce the required multiplier precision for Goldschmidt division," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1703–1708, Dec. 2010, doi: 10.1109/TC.2010.86.

[3] A. Rodriguez-Garcia, L. Pizano-Escalante, R. Parra-Michel, O. Longoria-Gandara, and J. Cortez, "Fast fixed-point divider based on Newton-Raphson method and piecewise polynomial approximation," in *2013 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2013*, Dec. 2013, pp. 1–6, doi: 10.1109/ReConFig.2013.6732291.

[4] A. Rathor and L. Bandil, "Design Of 32 bit floating point addition and subtraction units based on IEEE 754 standard," *International Journal of Engineering Research and Technology*, vol. 2, no. 6, pp. 2708–2712, 2013.

[5] S. B. K. Tirtha and V. S. Agrawala, *Vedic Mathematics: Sixteen Simple Mathematical Formulae from the Vedas*, Motilal Banarsidass Publishing House, 1992.

[6] S. S. Mahakalkar and S. L. Haridas, "Design of high performance IEEE754 floating point multiplier using Vedic mathematics," in *Proceedings - 2014 6th International Conference on Computational Intelligence and Communication Networks, CICN 2014*, Nov. 2014, pp. 985–988, doi: 10.1109/CICN.2014.207.

[7] M. Pradhan and R. Panda, "Speed comparison of 16x16 vedic multipliers," *International Journal of Computer Applications*, vol. 21, no. 6, pp. 12-19, May 2011, doi: 10.5120/2516-3417.

[8] S. K. Panda, R. Das, and T. R. Sahoo, "VLSI implementation of vedic multiplier using Urdhva– Tiryakbhyam sutra in VHDL environment: A novelty," *IOSR Journal of VLSI and Signal Processing*, vol. 5, no. 1, pp. 17–24, 2015, doi: 10.9790/4200-05131724.

[9] B. Jeevan, S. Narender, C. V. K. Reddy, and K. Sivani, "A high speed binary floating point multiplier using Dadda algorithm," in *Proceedings - 2013 IEEE International Multi Conference on Automation, Computing, Control, Communication and Compressed Sensing, iMac4s 2013*, Mar. 2013, pp. 455–460, doi: 10.1109/iMac4s.2013.6526454.

[10] C. Nagendra, R. M. Owens, and M. J. Irwin, "Power-delay characteristics of CMOS adders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 3, pp. 377–381, Sep. 1994, doi: 10.1109/92.311649.

[11] A. Gupta, "Arithmetic Unit implementation using delay optimized vedic multiplier with BIST capability," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 1, no. 5, pp. 59–62, 2012.

[12] A. Sahu, S. K. Panda, and S. Jena, "HDL implementation and performance comparison of an optimized high speed multiplier," *IOSR Journal of VLSI and Signal Processing*, vol. 5, no. 2, pp. 10–19, 2015, doi: 10.9790/4200-05211019.

[13] M. Pradhan and R. Panda, "High speed multiplier using Nikhilam Sutra algorithm of Vedic mathematics," *International Journal of Electronics*, vol. 101, no. 3, pp. 300–307, 2014, doi: 10.1080/00207217.2013.780298.

[14] K. Sethi and R. Panda, "An improved squaring circuit for binary numbers," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 2, 2012, doi: 10.14569/ijacsa.2012.030220.

[15] P. Malik, "High throughput floating-point dividers implemented in FPGA," in *Proceedings - 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2015*, Apr. 2015, pp. 291–294, doi: 10.1109/DDECS.2015.66.

[16] A. Kanhe, S. K. Das, and A. K. Singh, "Design and implementation of low power multiplier using vedic multiplication technique," *International Journal of Computer Science and Communication (IJCSC)*, vol. 3, no. 1, pp. 131–132, 2012.

[17] M. Al-Ashrafy, A. Salem, and W. Anis, "An efficient implementation of floating point multiplier," in *Saudi International Electronics, Communications and Photonics Conference 2011, SIECPC 2011*, Apr. 2011, pp. 1–5, doi: 10.1109/SIECPC.2011.5876905.

[18] R. Bhaskar, G. Hegde, and P. R. Vaya, "An efficient hardware model for RSA encryption system using Vedic mathematics," *Procedia Engineering*, vol. 30, pp. 124–128, 2012, doi: 10.1016/j.proeng.2012.01.842.

[19] A. Amaricai, M. Vladutiu, and O. Boncalo, "Design issues and implementations for floating-point divide-add fused," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 4, pp. 295–299, Apr. 2010, doi: 10.1109/TCSII.2010.2043473.

[20] K. Pande, A. Parkhi, S. Jaykar, and A. Peshattiwar, "Design and implementation of floating point divide-add fused architecture," in *Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015*, Apr. 2015, pp. 797–800, doi: 10.1109/CSNT.2015.179.

[21] S. K. Panda and A. Sahu, "A novel vedic divider architecture with reduced delay for VLSI applications," *International Journal of Computer Applications*, vol. 120, no. 17, pp. 31–36, Jun. 2015, doi: 10.5120/21322-4350.

[22] M. Der Shieh, J. H. Chen, H. H. Wu, and W. C. Lin, "A new modular exponentiation architecture for efficient design of RSA cryptosystem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1151–1161, Sep. 2008, doi: 10.1109/TVLSI.2008.2000524.

[23] A. P. Ramesh, A. V. N. Tilak, and A. M. Prasad, "FPGA based implementation of high speed double precision floating point multiplier with tiling technique using verilog," *International Journal of Computer Applications*, vol. 58, no. 21, pp. 17–25, 2012, doi: 10.5120/9407-3814.

[24] G. Srikanth and N. S. Kumar, "Design of high speed low power reversible vedic multiplier and reversible divider," *International Journal of Engineering Research and Applications*, vol. 4, no. 9, pp. 70–74, 2014.

[25] I. Kong and E. E. Swartzlander, "A goldschmidt division method with faster than quadratic convergence," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 4, pp. 696–700, 2011, doi: 10.1109/TVLSI.2009.2036926.

[26] K. Underwood, “FPGAs vs. CPUs: Trends in peak floating-point performance,” *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*, vol. 12, pp. 171–180, 2004, doi: 10.1145/968280.968305.
[27] C. V. S. Chaitanya, C. Sundaresan, P. R. Venkateswaran, and K. Prasad, “Design of modified booth based multiplier with carry pre-computation,” *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 13, no. 3, pp. 1048–1055, Mar. 2019, doi: 10.11591/ijeecs.v13.i3.pp1048-1055.
[28] M. F. Hassan, K. F. Hussein, and B. Al-Musawi, “Design and implementation of fast floating point units for FPGAs,” *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 19, no. 3, pp. 1480-1489, Sep. 2020, doi: 10.11591/ijeecs.v19.i3.pp1480-1489.

BIOGRAPHIES OF AUTHORS

Naginder Singh completed his Master of Technology from the National Institute of Technology, Kurukshetra (Haryana) and Bachelor of Technology from College of Engineering and Technology, Bikaner. He is currently serving as an Assistant Professor in the Department of Electronics and Communication Engineering, M.B.M. University, Jodhpur, Rajasthan (India). His research interests are in FPGA-based design, embedded systems and sensors, embedded system design, and VLSI. He can be contacted at email: naginder.ece@mbm.ac.in.

Kapil Parihar received his B.Tech. degree in Electronics and Communication Engineering at Bikaner Engineering College, Bikaner and M.E. in Digital Communication at M.B.M. Engineering College, Jodhpur, Rajasthan (India). He is full-time Assistant professor at M.B.M. University, Jodhpur, India. His research lines are optical communication and photonic crystal fiber. He doing his Ph.D. in Electronics and Communication Engineering Department from M.B.M. University Jodhpur and his field interest is terahertz antennas, electromagnetic engineering and antennas/sensors for communication, radar, imaging and sensing systems, UWB wireless communication, antennas for portable devices, and antennas for base stations in wireless communications. He can be contacted at email: kapil.ece@mbm.ac.in.