Design and Synthesis of Single Precision Floating Point Division based on Newton-Raphson Algorithm on FPGA

Naginder Singh[1, a] and Trailokya Nath Sasamal[2]

[1]*School of VLSI Design and Embedded System, National Institute of Technology, Kurukshetra, Haryana, India*
[2]*Department of Electronics and Communication Engineering, National Institute of Technology, Kurukshetra. Haryana, India*

Abstract. This paper describes a single precision floating point division based on Newton-Raphson computational division algorithm. The Newton-Raphson computational algorithm is implemented using 32-bit floating point multiplier and subtractor. The salient feature of this proposed design is that the module for computing mantissa in 32-floating point multiplier is designed using a 24-bit Vedic multiplication (Urdhva-triyakbhyam-sutra) technique. 32-bit floating point multiplier, designed using Vedic multiplication technique, yields a higher computational speed, hence, is efficiently used in floating point divider. Another important feature is the efficient use of device utilization parameters and reduced power consumption. An advantage of the Newton-Raphson algorithm is the higher versatility and precision. For representing 32-bit floating point numbers, IEEE 754 standard format is used. ISim simulator is used for simulation. The proposed floating point divider is designed using Verilog Hardware Description Language (HDL) and is verified on Xilinx Spartan 6 SP605 Evaluation Platform FPGA.

1 Introduction

The role of a reconfigurable processor in embedded system design has increased greatly from the past decades. Due to the advancement of field programmable gate array (FPGA), we have reached a point where the architecture of processors can be modified instantaneously. Reconfigurable computing processing provides very versatile high-speed computing. The enhanced feature of Spartan-6 voluntarily reduces the cost per logic cell designed. Newton-Raphson computational algorithm requires mathematical operations such as, multiplication and subtraction. Here, mathematical operations used to find the reciprocal of the denominator (D) and multiply that reciprocal by the numerator (N) to find the final quotient (Q). Newton-Raphson computational algorithm initializes with an approximation close to the final value of the quotient (Q) and produces twice as many digits of the final quotient after each iteration. An iterative process which is based on complex operation used for division in many signal-processing algorithms, where not only precision to be maintained, but also the precision is to be maintained for very large data intervals and should be high for better operation. This can be achieved by the design and implementation of floating point division by using Newton-Raphson algorithms. Several different algorithms described in literature [1-3]. The Newton-Raphson algorithm computes three multiplicative inverse at the same time to provide high throughput [4]. To implement and design 32-bit floating point division based on Newton-Raphson computational algorithm, 32-bit floating point multiplier and 32-bit floating point subtraction modules are used [5,6]. For efficient implementation of the floating point multiplier Vedic multiplication is used for calculating mantissa part [7, 8]. The format for representing 32-bit and 64-bit floating point numbers are provided by the IEEE 754 standard [9, 10]. IEEE 754 uses a fixed number of bits for representing the 32-bit floating point number. The representation format divides into three parts, i.e., sign (b), exponent (e) and the mantissa (s). Table 1 shows the structure for IEEE 754 formats and describes the single and double precision. In IEEE 754 Single precision format the mantissa is represented by 23 bits, exponent is represented by 8-bits and MSB corresponds to sign bit. The Sign of the floating point number depends on the sign bit or MSB. The number is positive when the MSB bit is 0 and negative when the MSB bit is 1.

Table 1. IEEE 754 Standard Format for single (32-bit) and double precision (64-bit).

	Sign (s)	Exponent (e)	Mantissa (m)
32-bit	1-bit	8-bit	23-bit
64-bit	1-bit	11-bit	52-bit

The formulation of the paper is as follows. Section 2 explains the architecture of the floating point multiplier using Vedic multiplication. Section 3 presents the description of 32-bit floating point subtractor. Section 4 describes the Newton-Raphson computational algorithm. Sections 5 presents the Simulation Results of floating point division using Newton-Raphson algorithm. The Conclusion and References are presented in the final section.

2 Floating point multiplier

In Figure 1 shows the complete architecture of proposed 32-bit floating point multiplier. This multiplier module is designed using a Vedic multiplication technique, where mantissa calculation is done using a 24x24 bit Vedic multiplier. The main purpose of using Vedic multiplier is to improve the overall performance of the 32-bit floating point multiplier. IEEE 754 format presents a fixed number of bits for representing the sign, exponent and mantissa. The inputs given to the floating point multiplier

[a] Naginder Singh: naginder.singh0110@gmail.com

are A[31-0] and B[31-0] as per IEEE 754 format. 32-bit Floating point multiplication unit is divided into three parts - sign unit, exponent unit and mantissa unit.

2.1. Mantissa unit

In mantissa unit, for calculation of mantissa a 24x24 bit Vedic multiplier is used efficiently for higher throughput and computation. The lower bits of inputs, m1 (A[22-0]) and m2 (B[22-0]) are given to the 24-bit Vedic multiplier which produces 24-bit normalized output and should have leading one as their MSB.

2.2. Exponent unit

In Exponent unit, the exponent calculation is done by using ripple carry adder. The exponent is computed by providing inputs e1 (A[30-23]) and e2 (B[30 – 23]) to the 8-bit ripple carry adder unit and result is biased to 127. The overflow and underflow cases are carefully handled.

2.3. Sign unit

In sign unit, the sign bit is computed by xoring the 31st bit of inputs, s1 (A[31]) and s2 (B[31]) of floating point inputs. The output of xor gate represents the sign of the floating point multiplier. The Vedic multiplication technique is efficiently used for high computational speed and throughput.

The complete architecture of proposed 32-bit floating point multiplier is shown in the Figure 1. This proposed multiplier module designed using Vedic multiplier is used in the Newton-Raphson computational algorithm for performing the iteration process.

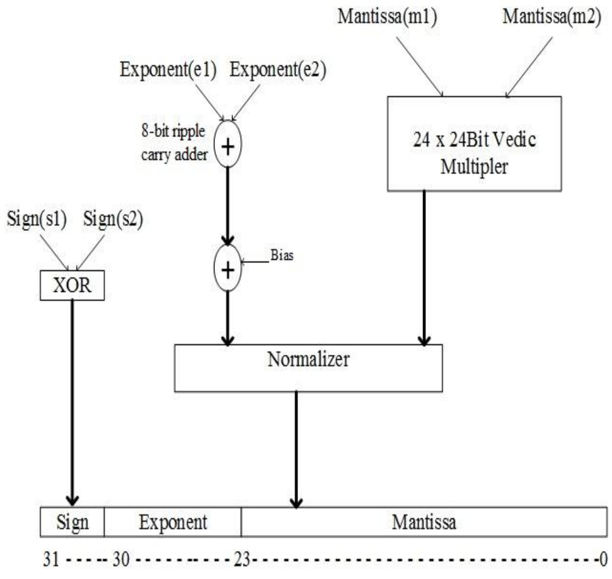


Figure 1. The proposed architecture for 32-bit Floating point multiplier.

3 Floating point subtractor

In the subtractor module, X[31-0] and Y[31-0] are given as inputs to the floating point subtractor. The sign (s), exponent (e) and mantissa (m) are represented in IEEE 754 format. 32-bit floating point subtraction operation is done in a stepwise manner as explained further. First of all, the floating point numbers are unpacked. After unpacking, the sign, exponent and mantissa are identified for performing the subtraction operation. Next, the exponent is equalized for performing alignment and normalization of mantissa part. If neither of the operands are infinity, then the relation between e1 (X[30-23]) and e2 (Y[30-23]) is determined by comparing the mantissa m1 and m2. The mantissa is shifted right until the exponent becomes equal, i.e. e1 (X[30-23]) = e2 (Y[30-23]). After alignment and normalization, the mantissa m1 (X[23-0]) and m2 (Y[23-0]) are subtracted. After that, the mantissa values are rounded off. Finally, the sign, exponent and mantissa parts are concatenated. The 32-bit floating point subtraction module is used in the Newton-Raphson computational algorithm for performing the iteration process. Figure 2 shows the complete architecture of 32-bit floating point subtractor. This 32-bit floating point subtractor module is used for calculating the iterative process in Newton-Raphson algorithm.

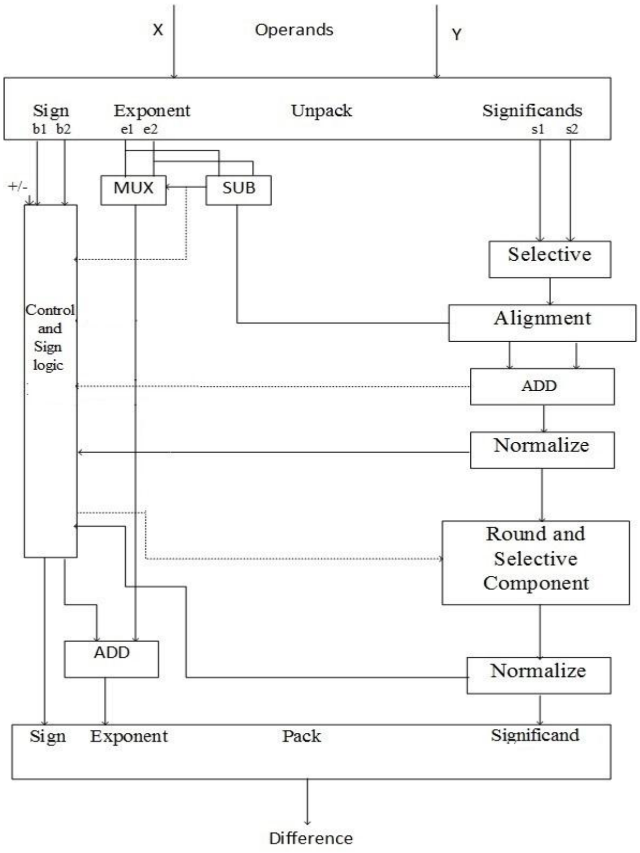


Figure 2. The architecture of 32-bit Floating point subtractor.

4 Newton-Raphson division algorithm

Newton-Raphson computational division algorithm computes the multiplicative inverse, which is calculated by the iterative process. Then the calculated multiplicative inverse is multiplied to the dividend to compute the final quotient (Q). In this proposed design, Newton-Raphson computational division algorithm designed by using a 32-floating point multiplier module and subtractor module. In this division algorithm minimal of maximal relative error can be achieved by scaling the divisor (D) in the interval (0.5, 1). Scaling of the divisor is done by shifting operation. To produce a precise result more iterations are required. For this purpose, fast division algorithms are developed. The Newton-Raphson algorithm converges much faster for computing one iteration. Thus, several multiplication and subtraction operations needed to perform for the continual iteration process. Figure 3 shows the flowchart of floating point division using the Newton-Raphson computational algorithm in which two multiplier and a subtraction module are used to produce one iteration. Newton Raphson computational algorithm produced optimized result by computing three iterations in one cycle. More iterations are performed to refine the multiplicative inverse.

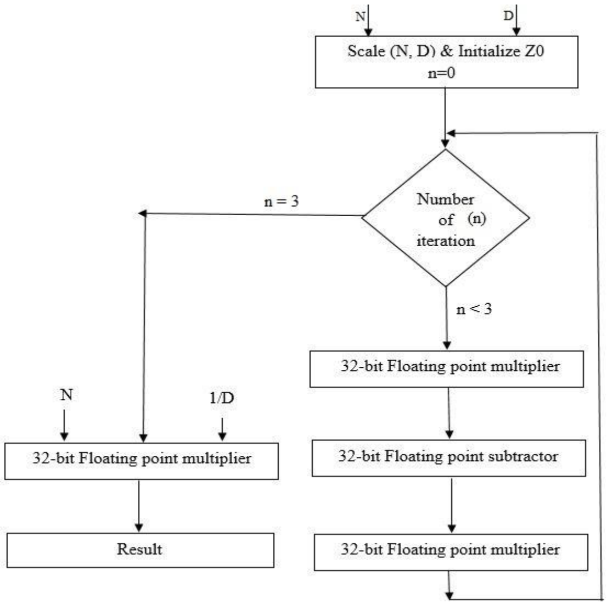

N
D
Scale (N, D) & Initialize Z0
n=0
n = 3
Number of (n) iteration
n < 3
32-bit Floating point multiplier
N
1/D
32-bit Floating point multiplier
32-bit Floating point subtractor
Result
32-bit Floating point multiplier

Figure 3. A Flowchart of the 32-bit floating point division using Newton-Raphson method.

The calculation of multiplicative inverse of the divisor is performed by equation (1), where Z_i is multiplicative inverse at iteration i, as given below.

$$Z_{i+1} = Z_i + Z_i\,(1 - DZ_i) = Z_i\,(2 - DZ_i) \qquad (1)$$

Newton-Raphson division algorithm uses a complex initialization for computing continual iterations. To minimize the maximum of the relative error in the interval (0.5, 1), Z_i should be initialized. The initialization is represented by Z_0 and is initialized as follows.

$$Z_0 = (48/17) - (32/17)\,D \qquad (2)$$

5 Simulation Results

The simulation was performed on ISim. Figure 4 shows the simulation results of the proposed 32-bit floating point division of two numbers using the Newton-Raphson method. Table 3 shows the device utilization of the Xilinx Spartan 6 SP605 Evaluation Platform FPGA. Table 4 shows the Xilinx Power Estimator (XPE) -14.3 device summary report for the proposed 32-bit floating point division carried out for the Spartan-6 SP605 Evaluation Platform FPGA. In Figure 4, N represents the numerator, D represents the denominator, Z0 is the initial value, the Z1 is the first iteration result, Z2 is the second iteration result, Z3 is the final iteration result and Q represents the quotient. The two inputs N and D are given to the divider in IEEE 754 standard format as shown in Table 2.

Table 2. Sample inputs and its output for simulation

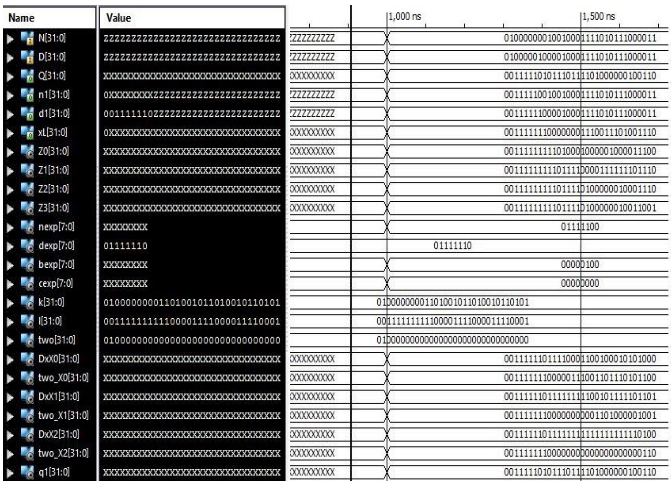
	Decimal	Sign	Exponent	Mantissa
N	3.14	0	10000000	10010001111010111000011
D	8.56	0	10000010	00010001111010111000011
Q	0.366822	0	01111101	01110111101000000100110

Table 3. Device Utilization of the Xilinx Spartan 6 SP605 Evaluation Platform.

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	408	54,576	1%
Number of Slice LUTs	10,019	27,288	36%
Number of occupied Slices	3,878	6,822	56%
Number of bonded IOBs	192	296	64%

Table 4. Xilinx Power Estimator (XPE) -14.3 device summary report of Spartan-6 SP605 Evaluation Platform FPGA.

Specifications	Values	
Junction Temperature	25.5 ºC	
Total On-Chip Power	0.036 W	
Thermal Margin	59.5 ºC	4.0 W
Effective ΘJA	14.3 ºC/W	



```
Name          Value                              |1,000 ns                 |1,500 ns
N[31:0]       ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ   01000000010010001111010111000011
D[31:0]       ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ   01000001000010001111010111000011
Q[31:0]       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111110101110111101000000100110
n1[31:0]      0XXXXXXXXZZZZZZZZZZZZZZZZZZZZZZZ   00111110010010001111010111000011
d1[31:0]      001111110ZZZZZZZZZZZZZZZZZZZZZZZ   00111111000010001111010111000011
xL[31:0]      0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111100000001110011101001110
Z0[31:0]      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111111010001000001000011100
Z1[31:0]      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111111011110000111111101110
Z2[31:0]      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111111011110100000010001110
Z3[31:0]      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111111011110100000010011001
nexp[7:0]     XXXXXXXX                           01111100
dexp[7:0]     01111110                           01111110
bexp[7:0]     XXXXXXXX                           00000100
cexp[7:0]     XXXXXXXX                           00000000
k[31:0]       01000000001101001011010010110101   01000000001101001011010010110101
l[31:0]       00111111111100001111000011110001   00111111111100001111000011110001
two[31:0]     01000000000000000000000000000000   01000000000000000000000000000000
DxX0[31:0]    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111011110001100100010101000
two_X0[31:0]  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111100000111001101110101100
DxX1[31:0]    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111011111111100101111101101
two_X1[31:0]  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111100000000001101000001001
DxX2[31:0]    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111011111111111111111110100
two_X2[31:0]  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111111100000000000000000000110
q1[31:0]      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   00111110101110111101000000100110
```

Figure 4. Simulation result of Floating point division.

6 Conclusion

The Single precision floating point division using Newton–Raphson computational algorithm is designed and synthesized on FPGA. This computational technique provides high computation speed and throughput by computing one multiplicative inverse in one iteration. The multiplier performance is increased by Vedic technique and hence improved the efficiency of the floating point divider. The device utilisation parameters were optimized thereby the power consumption is reduced. This presented design is useful in high computation demanding applications.

References

1. J. M. Muller, Avoiding Double Roundings in Scaled Newton-Raphson Division, *Proceedings of Asilomar Conference on Signals, Systems and Computers*, 396-399 (2013)
2. I. Kong, E. E. Swartzlander, A Rounding Method to Reduce the Required Multiplier Precision for Goldschmidt Division, *IEEE Transactions on Computers* **59**, 1703-1708 (2010)
3. A. R. Garca, L. P. Escalante, R. P. Michel, O. L. Gandara, J. Cortez, Fast Fixed-Point Divider Based on Newton-Raphson Method and Piecewise Polynomial Approximation, *Proceedings International Conference on Reconfigurable Computing and FPGA*, 1-6 (2013)
4. Peter Malik, High throughput floating-point dividers implemented in FPGA, *IEEE conference*, (2015)
5. A. Rathor, L. Bandil, Design Of 32 Bit Floating Point Addition And Subtraction Units Based On IEEE 754 Standard, *International Journal of Engineering Research & Technology* **2**, 2278-0181 (2013)
6. Jagadguru Swami Sri Bharati Krisna Tirthaji Maharaja, Vedic Mathematics Sixteen Simple Mathematical Formulae from the Veda, (1965)
7. A. Kanhe, S. K. Das, A. K. Singh, Design And Implementation Of Low Power Multiplier Using Vedic Multiplication Technique, *International Journal of Computer Science and Communication* **3**, 131-132 (2012)
8. A. Ashrafy, M. Salem, A. Anis, An efficient implementation of floating point multiplier, *Electronics Communications and Photonics Conference*, (2011)
9. B. Hickmann, A. Krioukov, M. Schulte, M. Erle, A Parallel IEEE 754 Decimal Floating Point Multiplier, *25th International Conference on Computer Design*, (2007)
10. IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, (2008)