International Journal of Electronic and Electrical Engineering.
ISSN 0974-2174 Volume 8, Number 1 (2015), pp. 27-32

http://www.irphouse.com

# Development of an FPGA based high speed single precision floating point multiplier

**Sneha S.jumle and M. V. Vyawhare**

*RTMNU, India*
*snehajumle99@gmail.com*

**Abstract**

Floating point arithmetic is widely used in many areas.Multipliers play an important role in today's digital signal processing and various other applications. A system's performance is generally determined by the performance of the multiplier, because the multiplier is generally the slowest element in the system. The way floating point operations are executed depends on the data format of the operands. IEEE standards specify a set of floating point formats single precision and double precision. This paper presents an efficient FPGA implementation of Single precision floating point multiplier using VHDL. The aim of project is design and simulation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format, the proposed multiplier doesn't implement rounding and presents **the significant multiplication result as is (48 bits).**

**Keywords** – Significant multiplier, VHDL simulation.

## *I.* INTRODUCTION

Floating point numbers are one possible way of representing real numbers in binary format, the IEEE 754[1] standard presents two different floating point formats, Binary interchange format and Decimal interchange format.

Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range.

My project focuses only on single precision normalized binary interchange format. It consists of a one bit sign (S), an eight bit exponent (E), and twenty three bit fraction (M or Mantissa). An extra bit is added to the fraction to form what is called the significand.

The aim of project is design and simulation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format, the proposed multiplier

doesn't implement rounding and presents the significant multiplication result as is (48 bits).

Multiplying two numbers in floating point format is done by adding the exponent of the two numbers then subtracting the bias from their result, and multiplying the significant of the two numbers, and calculating the sign by XORing the sign of the two numbers.The way floating point operations are executed depends onthe data format of the operands. IEEE standards specify aset of floating point data formats, single precision and double precision. The Single precision consists of 32 bits and the Double precision consists of 64 bits.

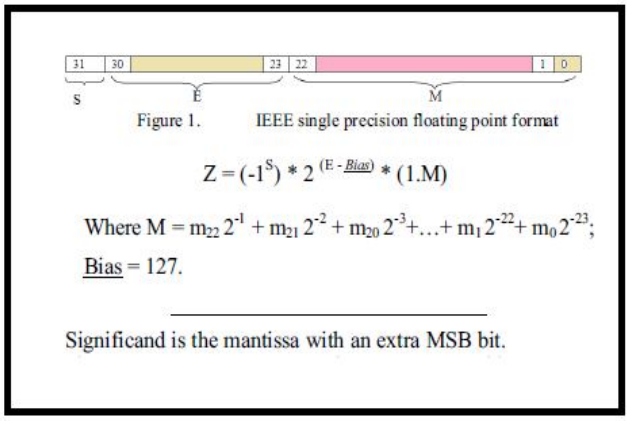



**Figure 1**

I. shows the IEEE single and double precision data formats

**Floating point multiplication algorithm**

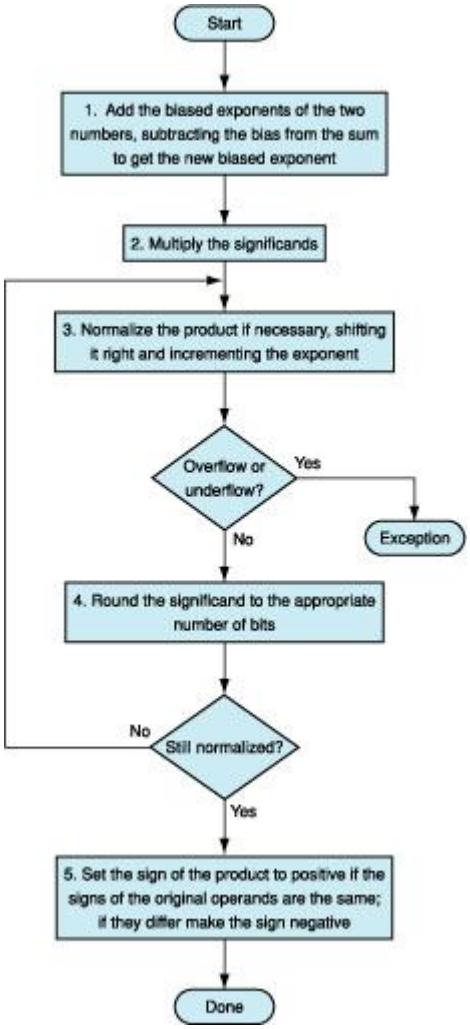
1. Add exponents
2. Multiply fractions
3. If product is 0, adjust for proper 0
4. Normalize product fraction
5. Check for exponent overflow or underflow
6. Round product fraction

- Exponents 00000000 and 11111111 reserved
- Smallest value

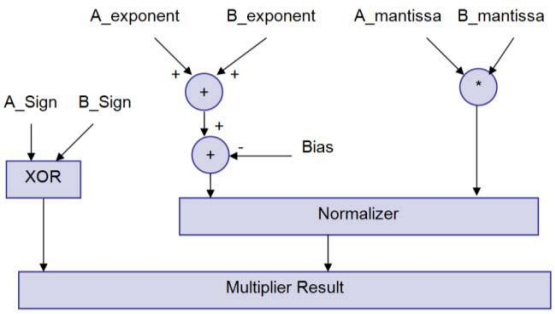
– Exponent: 00000001
  Þ actual exponent = 1 – 127 = –126
– Fraction: 000…00 Þ significand = 1.0
– $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$

- Largest value

– exponent: 11111110
  Þ actual exponent = 254 – 127 = +127
– Fraction: 111…11 Þ significand ≈ 2.0
  $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$




## II. PROPOSED METHODOLOGY



## III. ROUNDING

| e = -1 | e = 0 | e = 1 |
|---|---|---|
| 1.00 X 2^(-1) = 1/2 | 1.00 X 2^0 = 1 | 1.00 X 2^1 = 2 |
| 1.01 X 2^(-1) = 5/8 | 1.01 X 2^0 = 5/4 | 1.01 X 2^1 = 5/2 |
| 1.10 X 2^(-1) = 3/4 | 1.10 X 2^0 = 3/2 | 1.10 X 2^1= 3 |
| 1.11 X 2^(-1) = 7/8 | 1.11 X 2^0 = 7/4 | 1.11 X 2^1 = 7/2 |

## IV. OVERFLOW/UNDERFLOWDETECTION.

Overflow/underflow means that the result‟s exponent is too large/small to be represented in the exponent field. The exponent of the result must be 8 bits in size, and must be between 1 and 254 otherwise the value is not a normalized one .An overflow may occur while adding the two exponents or during normalization. Overflow due to exponent addition maybe compensated during subtraction of the bias; resulting in a normal output value (normal operation). An underflow may occur while subtracting the bias to form the intermediate exponent. If the intermediate exponent < 0 then it□s an underflow that can never be compensated; if the intermediate exponent = 0 then it□s an underflow that may be compensated during normalization by adding 1 to it .When an overflow occurs an overflow flag signal goes high and the result turns to ±Infinity (sign determined according to the sign of the floating point multiplier inputs). When an underflow occurs an underflow flag signal goes high and the result turns to ±Zero (sign determined according to the sign of the floating point multiplier inputs). Denormalized numbers are signaled to zero with the appropriate sign calculated from the inputs and an underflow flag is raised.

## V. Simulation results of proposed floating point multiplier

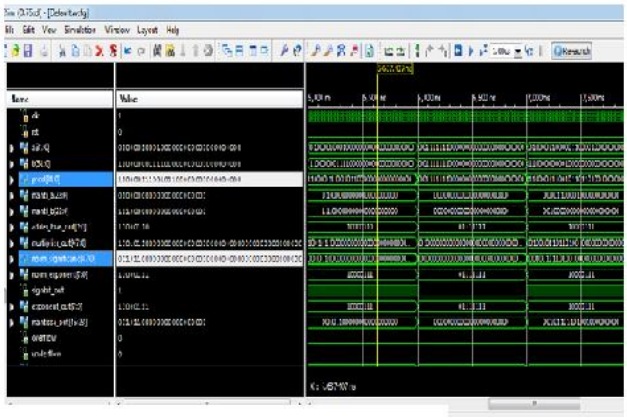
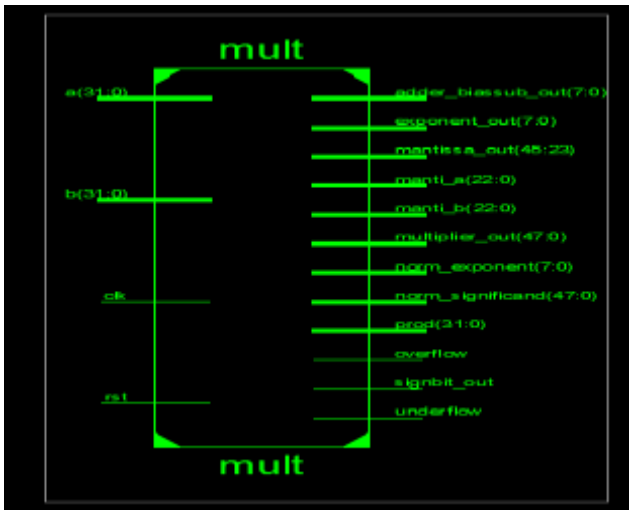


**Figure 9: Simulation result of Proposed floating point multiplier Figure 10: RTL Schematic of proposed Xilinx floating point multiplier**




## VI. CONCLUSION

THIS PAPER PRESENTS DESIGN AND SIMULATION OF A FLOATING POINT MULTIPLIER THAT SUPPORTS THE IEEE 754-2008 BINARY INTERCHANGE FORMAT, THE PROPOSED MULTIPLIER DOESN□T IMPLEMENT ROUNDING AND PRESENTS THE SIGNIFICAND MULTIPLICATION RESULT AS IS (48 BITS), THIS GIVES BETTER PRECISION IF THE WHOLE 48 BITS ARE UTILIZED IN ANOTHER UNIT; I.E.WITH A FLOATING POINT ADDER TO FORM A MAC UNIT. BUT THE FLOATING POINT FMULTIPLIER CORE GENERATED BY XILINX CORE GENERATOR DOES NOT INDICATES THE ENTIRE 48 BITS OF MANTISSA DUE TO ROUNDING AND IS NOT BENEFICIAL IN CASE OF DSP APPLICATION OF LARGE DYNAMIC RANGE ESPECIALLY WHEN USING IT IN ANOTHER HIGH PRECISION FLOATING POINT UNITS LIKE MULTIPLY AND ACCUMULATE (MAC) UNIT.

## VII. REFERENCES

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.

[2] B. Fagin and C. Renard, “Field Programmable Gate Arrays and FloatingPoint Arithmetic,” *IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365–367, 1994.*

[3] N. Shirazi, A. Walters, and P. Athanas, “Quantitative Analysis ofFloating Point Arithmetic on FPGA Based Custom Computing Machines,” *Proceedings of the IEEE Symposium on FPGAs for CustomComputing Machines (FCCM’95), pp.155–162, 1995*.

[4] L. Louca, T. A. Cook, and W. H. Johnson, “Implementation of IEEESingle Precision Floating Point Addition and Multiplication on FPGAs,”*Proceedings of 83 the IEEE Symposium on FPGAs for CustomComputing Machines (FCCM’96), pp. 107–116, 1996.*

[5] A. Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs",

*Proc. of IEEE ICASSP, 2001, vol. 2, pp.897-900.*

[6] B. Lee and N. Burgess, "Parameterisable Floating-point Operations on FPGA," *Conference Record of the Thirty-Sixth Asilomar Conference onSignals, Systems, and Computers, 2002*

[7] Mohamed Al-Ashrafy, Ashraf Salem and Wagdy Anis" An Efficient Implementation of Floating PointMultiplier" *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*