ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 3, September 2014

Multiplication of floating point numbers using VHDL

Sumi M.S., Sobin Daniel
M.Tech - VLSI and Embedded Systems, Asst Prof. Dept of ECE
Indira Gandhi Institute of Engineering and Technology, Ernakulam
Kerala, India

Abstract*— Floating Point Numbers are used in the field of medical imaging, biometrics, motion capture and audio applications, including broadcast, conferencing, musical instruments and professional audio. For this IEEE 754 standard is used .The floating point standard defines not only normalized values but it also considers special cases. It also specifies four rounding modes. Rounding modes implemented in this multiplier are round to infinity, round to zero and round to nearest. Exceptions used in this floating point are inexact, invalid, overflow, underflow, infinity operations, etc.

***Index Terms*— Floating point multiplication, Normalized, Denormalized, Rounding, VHDL simulation.**

I. INTRODUCTION

Every computer has a floating point processor or a dedicated accelerator that fulfills the requirements of precision using detailed floating point arithmetic. Motion Capturing, biometrics, and audio applications and also for broadcasting, conferencing musical instruments and professional audio are fields that use floating point multilplication. They can be of great importance because the performances of computers that use multiplication are measured in terms of the number of floating point operations they can perform per second. [1]

Decimal numbers are also known as Floating Points because a single number can be represented with one or more significant digits depending on the position of the decimal point.

II. FLOATING POINT ARITHMETIC

IEEE 754 specifies four formats for representing floating-point values: single-precision (32-bit), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits).

Fig. 1 shows the IEEE 754 single precision binary format representation; it mainly consists of a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). An extra bit is added to the fraction to form and is called the significand part. In this if the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significand then the number is said to be a normalized number. The normalized floating point numbers have the

2. Placing the decimal point in the result
3. Adding the exponents; i.e. (E1 + E2 – *Bias*)
4. Obtaining the sign; i.e. s1 xor s2
5. Normalizing the result; i.e. obtaining 1 at the MSB of the results' significand
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence

$Z = (-1S) * 2^{(E - Bias)} * (1.M)$

Where M = m22 2-1 + m21 2-2 + m20 2-3+…+ m1 2-22+ m0 2-23;

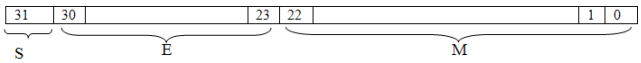

Fig1:- IEEE single precision floating point format

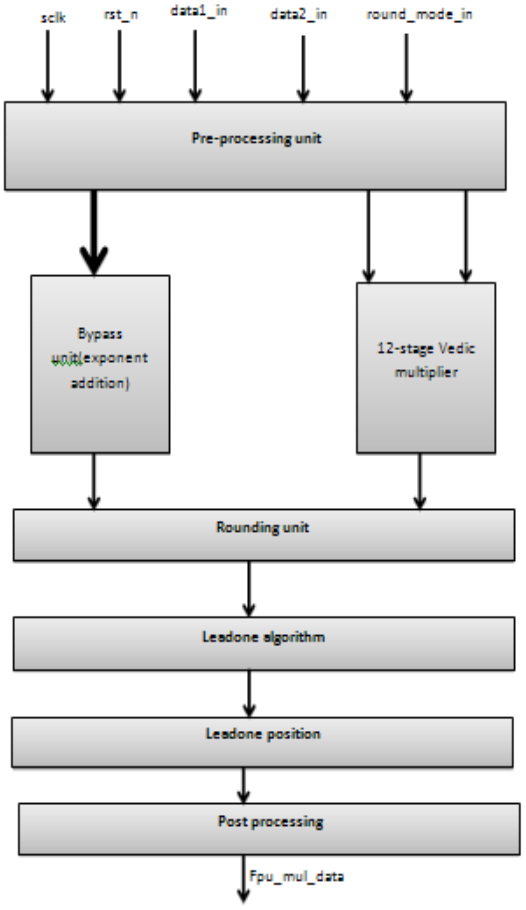

Fig 2:-Floating Point Multiplier architecture

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 3, September 2014

After multiplying two floating point numbers, the result obtained is of 48bits which is not of IEEE standard. So truncation of the result obtained is needed. So we use rounding methods to concatenate the mantissa part. The result obtained at the exponent is 127+127 bias value, in order to make it to actual IEEE standard we subtract one bias value (-127) from the result. The significand part is obtained by Xoring the inputs.

III.RANGES OF FLOATING-POINT NUMBERS

The range of positive floating point numbers can be divided into normalized and denormalized numbers. Since the sign of floating point numbers is given by a special leading bit, the range for negative numbers is given by the negation of the above values. There are mainly five different numerical ranges that single-precision floating-point numbers that are not able to represent:

- Zero
- Positive numbers less than 2^{-149} (positive underflow)
- Positive numbers greater than $(2-2^{-23}) < 2^{127}$ (positive overflow)
- Negative numbers less than $-(2-2^{-23}) < 2^{127}$ (negative overflow)
- Negative numbers greater than -2^{-149} (negative underflow)

The designers must decide whether to deliver the low-order word of the product or the entire product in floating-point multiplication, where the exact product can be rounded to the precision of the operands or to the next higher precision.

A. Special values

IEEE reserves exponent field values of all 0s and all 1s to denote special values in the floating-point scheme.

1. Zero

Zero is not directly representable in the straight format, due to the assumption of a leading 1 .Zero is a special value denoted with an exponent field of zero and a fraction field of zero. In this -0 and +0 are distinct values, even though they both compare as equal.

2. Denormalized

If the exponent have all 0s, but the fraction is non-zero, then the value is a denormalized number which does not have an assumed leading 1 before the binary point. Zero can also be considered as special type of denormalized number.

3. Infinity

The values +infinity and -infinity are denoted with an exponent of all 1s and a fraction of all 0s. The sign bit distinguishes between negative infinity and positive infinity.

4. Underflow

Two events cause the underflow exception to be signaled, tininess and loss of accuracy. Tininess is detected after or before rounding when a result lies between ±2. . Loss of accuracy is detected when the result is simply inexact or only when a de normalization loss occurs. Subnormal numbers also called Denormalized which causes underflow to occur. Detected.

5. Overflow

The overflow exception is signaled whenever the result exceeds the maximum value. There is a limit of number range that can be represented. It is not signaled when one of the operands is infinity, because infinity arithmetic is always considered as exact value. Division by zero also does not affect this exception condition.

6. Not a number

The value NaN (Not a Number) is used to represent a value that does not represent a real number. NaN's are represented such that it has exponent with all 1s and fraction with a non zero. There are two categories of NaN: QNaN (Quiet NaN) and SNaN (Signalling NaN). They have the following format, where s is the sign bit:

QNaN = s 11111111 10000000000000000000000

SNaN = s 11111111 00000000000000000000001

A QNaN is a NaN with the most significant bit as QNaN's propagate freely through most arithmetic operations. It is used to signal an exception when floating point numbers of undefined numbers are done. QNaN's indicates indeterminate operations, while SNaN's indicates invalid operations.

7. Special operations

Operations on special numbers are well-defined by IEEE. Any operation with a NaN yields a NaN result.

IV. ROUNDING MODES

Since the result obtained after floating point number multiplication is not precise, rounding is necessary. In order to increase the precision of the result and to enable rounding modes. In this round-to-nearest-even rounding mode, three bits were added internally and temporally to the actual fraction value: guard, round, and hidden bit. While guard and round bits are used as normal storage, the hidden bit is turned as whenever ít is shifted out of range. [9].As an example we take a 5-bits binary number: 1.1101. If we left-shift the number four positions, the number will be 0.0001, no rounding is possible and the result will not be accurate. Now we need to add three extra bits. After left-shifting the number four positions, the number will be 0.0001 101. If we round it back to 5-bits it will yield: 0.0010, therefore giving a more accurate result.

The problem with multiplication result must fit into the same n bits as the multiplier and the multiplicand. This, of course, often leads to loss of precision. By introducing rounding modes this can be reduced.. When all are enabled, the multiplier supports all IEEE rounding modes: round to nearest even, round to zero, round to positive infinity, and round to negative infinity. The user can select the desired rounding mode by giving the corresponding two bit binary input during simulation. The encoding for the rounding modes is shown in Table 2.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 3, September 2014

Table 2: Rounding mode encodings

Rounding Mode	Abbr.	Encoding
Nearest even	RN	00
Zero	RZ	01
Positive infinity	RP	10
Negative (minus) infinity	RM	11

The standard specifies four rounding modes:

A. Round to nearest even

This is the standard default rounding. The value is rounded up or down to the nearest value. If the value is exactly halfway between two infinitely precise results, then it should be rounded up to the nearest infinitely precise even.

For example:

Table 3: Example of rounding to nearest even

Unrounded	Rounded
3.4	3
5.6	6
3.5	4
2.5	2

B. Round-to-zero

Basically in this mode the number will not be rounded. The excess bits will simply get truncated, e.g. 3.47 will be truncated to 3.4

C. Round-Up

The number will be rounded up towards +∞, e.g. 3.2 will be rounded to 4, while -3.2 to -3.

D. Round-down

The opposite of round-up, the number will be rounded up towards -∞, e.g. 3.2 will be rounded to 3, while 3.2 to -4.

V. SIMULATION RESULTS

This design has been implemented, simulated in Model Sim and synthesized for VHDL. The HDL code uses VHDL 2001 constructs that provide certain benefits over the VHDL 95 standard in terms of scalability and code reusability.

Simulation based verification is one of the methods for functional verification of a design. In this method, test inputs are provided using standard test benches. The test bench forms the top module that instantiates other modules. Simulation based verification ensures that the design is functionally correct when tested with a given set of inputs.

The following snapshots are taken from ModelSim after the timing simulation of the floating point multiplier core.

Consider the inputs to the floating point multiplier are :
A = 32h'412D1234
B = 32h'412D1234

The output of the multiplier should be: 32h'42EA0337 and flags outputs of this multiplier overflow = 0; qnan = 0; snan = 0; underflow = 0; zero = 0;

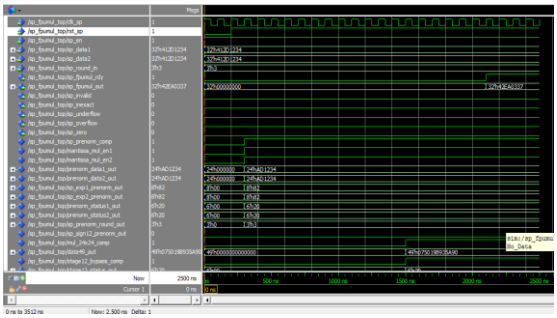

Fig 3:-Simulation results of proposed floating point multiplier

VI. CONCLUSION

Single precision floating point multiplier is designed and implemented using ModelSim in this paper. The designed multiplier is of IEEE 754 single precision floating point standard. In this implementation exceptions (like invalid, inexact, infinity, etc) are considered and is implemented. In this implementation rounding modes like round to positive infinity, round to negative infinity, round to zero and round to even are also considered. The designed is verified using fpu_test test bench. The design is also verified for overflow and underflow cases.

As a future advancement Double Precision Floating Point multiplier can be implemented which comprises of 64 bit.

REFERENCES

[1] IEEE standard for binary-floating point arithmetic, ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronic Engineers Inc., New York, August 1985.

[2] David Goldberg: What Every Computer Scientist Should Know About Floating- Point Arithmetic, 1991.

[3] Koren, Computer Arithmetic Algorithms, Second Edition, prentice Hall, 2002.

[4] An ANSI/ IEEE Standard for Radix-Independent Floating-Point Arithmetic, Technical Committee on microprocessor of IEEE computer society, October, 1987.

[5] Steve Hollasch, IEEE Standard 754 Floating Point Numbers, February 2005.

[6] BROWN, Stephen D. Fundamentals of Digital Logic with VHDL designs. Boston: McGraw-Hill, 2000.

[7] John L Hennesy & David A. Patterson Computer Architecture a Quantitative Approach Second edition; A Harcourt Publishers International Company.

[8] J. Bhasker, A VHDL Primer, Third Edition, Pearson, 1999.

[9] M. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann Publishers, 2004.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 3, September 2014

AUTHORS PROFILE

Sumi M.S, M.Tech - VLSI and Embedded Systems, Indira Gandhi Institute of Engineering and Technology, Ernakulam Kerala, India

Sobin Daniel Asst Prof. Dept of ECE Indira Gandhi Institute of Engineering and Technology, Ernakulam Kerala, India