IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676, p-ISSN: 2320-3331
PP 22-25
www.iosrjournals.org

Low latency, area optimized, high throughput double preicision pipeleined floating point multiplier using VHDL on FPGA

Tushar S. Muratkar[1], Prof. Sudhir N.Shelke[2]
[1] *ELECTRONICS, JDCOEM, NAGPUR INDIA*
[2]*HOD EN/ETC, JDCOEM, NAGPUR INDIA*

ABSTRACT: *Floating-point numbers are widely adopted in many applications due their dynamic representation capabilities. Floating-point representation is able to retain its resolution and accuracy compared to fixed-point representations. Unfortunately, floating point operators require excessive area (or time) for conventional implementations. The creation of floating point units under a collection of area, latency, and throughput constraints is an important consideration for system designers. This paper presents the implementation of a general purpose, scalable architecture used to synthesize floating point multipliers on FPGAs. Although several implementations of floating point units targeted to FPGAs have been previously reported, most of them are customized for specific applications. Multiplication is an important fundamental function in arithmetic operations. It can be performed with the help of different multipliers using different techniques. The objective of good multiplier is to provide a physically compact high speed and low power consumption. To save significant power consumption of multiplier design, it is a good direction to reduce number of operations. The main objective of this paper is to design "Simulation of IEEE 754 standard double precision multiplier and check for the latency, area and troughput" using VHDL.*

***Keywords*:** *Floating point multiplier, VHDL, FPGA*

I. INTRODUCTION

Field-programmable gate arrays (FPGAs) have long been attractive for accelerating fixed-point applications. Early on, FPGAs could deliver tens of narrow, low latency fixed-point operations. As FPGAs matured, the amount of parallelism to be exploited grew rapidly with FPGA size. This was a boon to many application designers as it enabled them to capture more of the application. It also meant that the performance of FPGAs was growing faster than that of CPUs [7]. Every computer has a floating point processor or a dedicated accelerator that fulfils the requirements of precision using detailed floating point arithmetic. The main applications of floating points today are in the field of medical imaging, biometrics, motion capture and audio applications. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier with more accuracy. Reducing the time delay and power consumption are very essential requirements for many applications. Floating Point Numbers: The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, that is, the decimal point can float. There are also representations in which the number of digits before and after the decimal point is set, called fixed-point representations.

1.1Floating Point Arithmetic

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely used standard for floating-point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point number (including ±zero and denormals) and special values (infinities and NaNs) together with a set of floating-point operations that operate on these values. It also specifies four rounding modes and five exceptions. 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). Many languages specify that IEEE formats and arithmetic be implemented, although sometimes it is optional. For example, the C programming language, which pre-dated IEEE 754, now allows but does not require IEEE arithmetic (the C float typically is used for IEEE single-precision and double uses IEEE double-precision).

1.2Double Precision Floating Point Numbers

Thus, a total of 64 bits is needed for double-precision number representation. To achieve a bias equal to 2^{n-1}- 1 is added to the actual exponent in order to obtain the stored exponent. This equal 1023 for an 11-bit exponent of the double precision format. The addition of bias allows the use of an exponent in the range from

−1023 to +1024, corresponding to a range of 0.2047 for double precision number. The double precision format offers a range from $^{2\text{-}1023}$ to 2^{+1023}, which is equivalent to 10^{-308} to 10^{+308}.

Sign: 1-bit wide and used to denote the sign of the Number i.e. 0 indicate positive number and 1 represent negative number.

Exponent: 11-bit wide and signed exponent in excess- 1023 representation. Mantissa: 52-bit wide and fractional component.

SIGN	**EXPONENT**	**FRACTION**
1 Bit	**11 Bits**	**52 Bits**

Fig 1. Double precision Floating point format

1.3 Floating-Point Multiplication

Multiplication of two floating point normalized numbers is performed by multiplying the fractional components, adding the exponents, and an exclusive or operation of the sign fields of both of the operands The most complicated part is performing the integer-like multiplication on the fraction fields . Essentially the multiplication is done in two steps, partial product generation and partial product addition. For double precision operands (53-bit fraction fields), a total of 53 53bit partial products are generated**.** The general form of the representation of floating point is:

$$(-1)\ S * M * 2E$$

Where

S represents the sign bit, M represents the mantissa and E represents the exponent. Given two FP numbers n1 and n2, the product of both, denoted as n, can be expressed as:

$n = n1 \times n2$

$= (-1)\ S1 \cdot p1 \cdot 2^{E1} \times (-1)\ S2 \cdot p2 \cdot 2^{E2}$

$=(-1)^{S1+S2} \cdot (p1 \cdot p2) \cdot 2^{E1+E2}$

In order to perform floating-point multiplication, a simple algorithm is realized:

Add the exponents and subtract 1023.

Multiply the mantissas and determine the sign of the result.

Normalize the resulting value, if necessary.

II. LITERATURE SURVEY

A few research work have been conducted to explain the concept of Floating Point Numbers. D. Goldberg [1] explained the concept of Floating Point Numbers used to describe very small to very large numbers with a varying level of precision. They are comprised of three fields, a sign, a fraction and an exponent field. B. Parhami [2] proposed IEEE-754 standard defining several floating point number formats and the size of the fields that comprise them. This Standard defines several rounding schemes, which include round to zero, round to infinity, round to negative infinity, and round to nearest. Michael L. Overton [3] performed the multiplication of two floating point normalized numbers by multiplying the fractional components, adding the exponents, and an Exclusive OR operation of the sign fields of both of the operands. Cho, J. Hong et al. and N. Besli et al.[4][5] multiplied double precision operands (53-bit fraction fields),in which a total of 53 53-bit partial products are generated . To speed up this process, the two obvious solutions are to generate fewer partial products and to sum them faster. Sumit Vaidya et al.[6] compared the different multipliers on the basis of power, speed, delay and area to get the efficient multiplier. It can be concluded that array Multiplier requires more power consumption and gives optimum number of components required.

FPGA

FPGA stands for Field Programmable Gate Arrays. It is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip flops or more complete blocks of memories. A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field programmable", i.e. programmable in the field) so that the FPGA can perform whatever logical function is needed. FPGAs are generally slower than their application

specific integrated circuit (ASIC) counterparts, as they can't handle as complex a design, and draw more power. However, they have several advantages such as a shorter time to market, ability to re-program in the field to fix bugs, and lower non recurring engineering cost costs. Vendors can sell cheaper, less flexible versions of their FPGAs which cannot be modified after the design is committed. The development of these designs is made on regular FPGAs and then migrated into a fixed version that more resembles an ASIC. Complex programmable logic devices, or CPLDs, are another alternative. FPGA architectures include dedicated blocks such as RAM, hardwired multipliers, multiply-accumulate unit, high-speed clock management circuitry, and serial transceivers, embedded hard processor cores such as PowerPC or ARM, and soft processor cores such as NIOS or Microblaze.

III. METHODOLOGY

There are number of techniques that can be used to perform multiplication. In general, the choice is based upon factors such as latency, throughput, area, and design complexity and hence we use Array Multiplier for implementing the multiplier.

4.1 Array Multiplier

Array multiplier is an efficient layout of a combinational multiplier. Multiplication of two binary number can be obtained with one micro-operation by using a combinational circuit that forms the product bit all at once thus making it a fast way of multiplying two numbers since only delay is the time for the signals to propagate through the gates that forms the multiplication array.

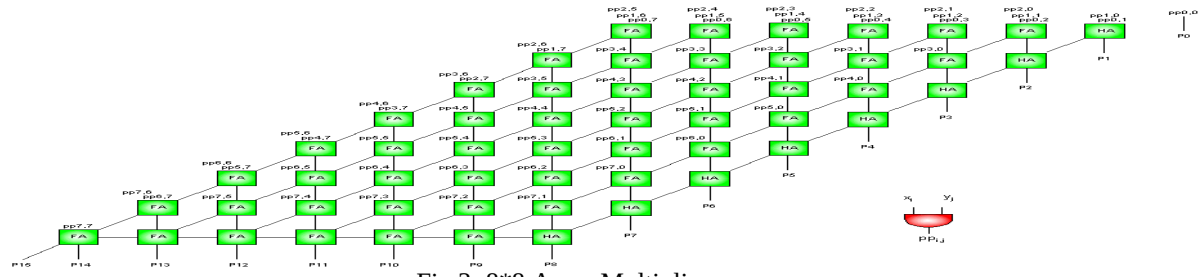

Fig 3: 8*8 Array Multiplier

4.2 Pipelined floating point multiplier module

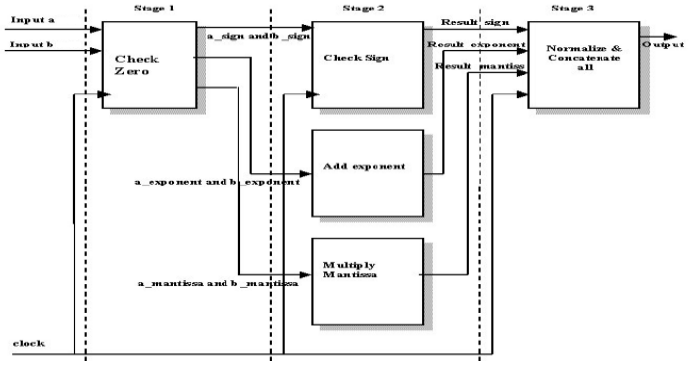

Fig 2.Pipelined Floating point multiplier

The details of each block is as given below

Check Zero Module

Here both operands are checked to determine whether they contain a zero. If one of them is zero , zero _flag is set to zero. If none of them are zero , then inputs in IEEE 754 format is unpacked and assigned to the check sign , add exponent, and multiply mantissa module.

Add Exponent Module

This module is activated if both the operands are non-zero. Two extra bits are also added to indicate the overflow and underflow conditions. The resulting sum has a double bias, so the extra bias is subtracted from the exponent sum. After this, Exp_Flag is set to 1.

Multiply Mantissa module

Here zero _flag is checked first. If zero _flag is set to zero the no calculation and normalization is performed. The Mantissa_Flag is set to zero. If both the operands are not zero then operation is done with multiplication operator. Mantissa_Flag is set to 1, indicating that the operation is executed.

Check Sign Module

It determines the sign of the two operands .The resultant sign is positive if both the operands have same sign else it is negative. XOR circuit is used

Normalize and concatenate all modules

It checks the overflow and underflow after adding the exponent .Overflow occurs if 8^{th} bit is 1, underflow occurs if 9^{th} bit is 1. If Exp_Flag, Mant_Flag, Sign_Flag are set, then normalization is carried out. Lastly all are concatenated and are normalized.

IV. CONCLUSION

Thus I had implemented the multiplier part using XILINX ISE 13.1. 1.XILINX ISE 13.1. The RTL VIEW and Simulation waveforms is also shown below .

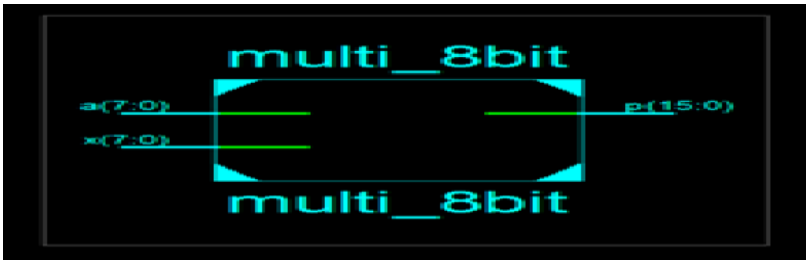

Fig 3.RTL view

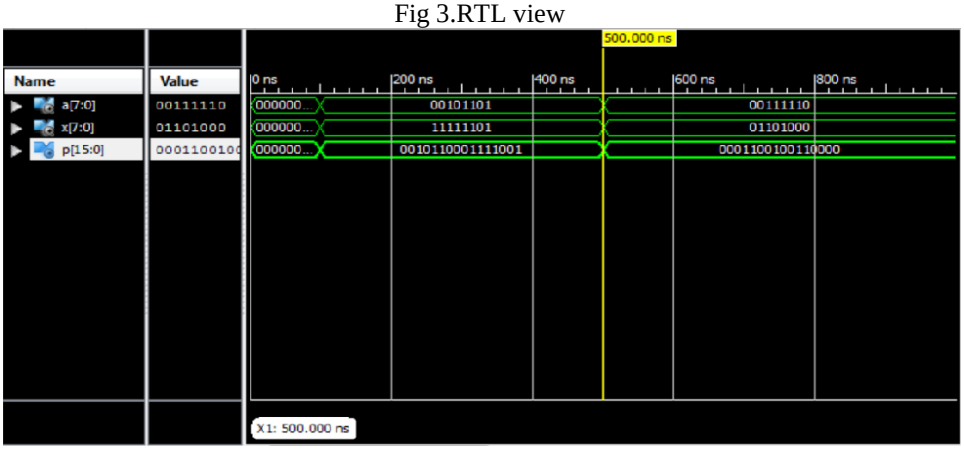

Fig 4.Simulation Waveform

REFERENCES

[1] D. Goldberg, "What every computer scientist should know about floating-point arithmetic ,*ACM Computing Surveys vol. 23-1 , pp. 5-48 ,1991.*

[2] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs",Oxford University Press, 2000.

[3] Michael L. Overton, "Numerical Computing with IEEE Floating Point Arithmetic, Published by Society for *Industrial and Applied Mathematics,2001*.

[4] Cho, J. Hong, and G Choi, "54x54-bit Radix-4 Multiplier based on Modified Booth Algorithm," *13th ACM Symp.VLSI, pp 233-236, Apr. 2003.*

[5] N. Besli, R. G. DeshMukh, "A 54*54-bit Multiplier with a new Redundant Booth's Encoding," *IEEE Conf. Electrical and Computer Engineering, vol. 2, pp 597-602, 12-15 May 2002.*

[6] Sumit Vaidya and Deepak Dandekar," Delay-Power Performance comparison of multipliers in VLSI circuit design", *International Journal of Computer Networks & Communications (IJCNC), Vol.2, No.4, pg 47-55, July 2010.*

[7] . K. D. Underwood. FPGAs vs. CPUs: Trends in peak floating-point performance. In Proceedings of the *ACM International Symposium on Field Programmable Gate Arrays, Monterrey, CA, February 2004.*

[8] P. Assady, "A New Multiplication Algo Using High-Speed Counters"*, European Journal of Scientific Research ISSN 1450-216X, Vol.26 No.3 ,pp.362-368, 2009.*

[9] Y. Wang, Y. Jiang, and E. Sha, "On Area Efficient Low Power Array Multipliers",*In the 8th IEEE International Conference on Electronics, Circuits and Systems, pp 1429– 1432,2001.*

[10] U. Kulisch, "Advanced Arithmetic for the Digital Computers", *Springer- Verlag, Vienna, 2002.*