WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Comprehensive Study of Various IEEE-754 Floating Point Multiplier Designs

Charan N[1], Dr Sapna PJ[2]

[1]Department of Electronics and Communication Engineering, Dayananda Sagar College Of Engineering, Bangalore
ncharanprasad10@gmail.com
[2]Associate Professor, Department of Electronics and Communication Engineering, Dayananda Sagar College Of Engineering, Bangalore
Sapna-ece@dayanandasagar.edu

ABSTRACT—

In this paper, various algorithms to design a floating point multiplier along with their architecture and implementation details are studied, which are essential for designing digital systems and an important block in digital signal processing. IEEE 754 Floating-point is a way of representing and performing arithmetic operations on real numbers in computer architecture. It's a numerical data type that allows us to handle values with fractional parts and a wide range of magnitudes. This number representation is essential in computing because they enable us to work with real-world values that are not whole numbers. Many scientific and engineering calculations require precise representation of decimal numbers with varying levels of precision, such as half, single and double. Floating-point multiplication is the arithmetic operation most frequently utilized and is a very important component of many engineering applications such as signal processing, video processing, image processing, etc.

Keywords – IEEE, Floating Point Arithmetic, Multiplier

I. INTRODUCTION

In digital computing, representing real numbers—those containing fractional components or requiring extremely large or small magnitudes—poses a significant challenge. To address this, the IEEE (Institute of Electrical and Electronics Engineers) developed the IEEE 754 standard, a widely adopted and rigorously defined system for representing floating-point numbers in binary form. Introduced in 1985 and refined through later revisions, this standard forms the foundation for most modern computational platforms, allowing consistency, accuracy, and portability in numerical operations across a variety of hardware and software systems. The primary motivation behind the IEEE 754 standard is to mimic the scientific notation used in mathematics, but in a binary format suitable for electronic processing. A floating-point number, in this system, is represented using three major components: a sign bit, an exponent, and a mantissa (or significand). The sign bit determines the polarity of the number, the exponent encodes the scale or magnitude using a bias to handle both positive and negative exponents, and the mantissa stores the significant digits of the number in binary, excluding the implicit leading bit in normalized forms. There are a total of 32 bits in case of Single precision and 64 bits for Double precision and the format to represent these numbers is shown in Fig 1.

The significance of the IEEE 754 floating-point standard lies not only in its structured and efficient representation of real numbers but also in the consistency it brings to mathematical computations across various computing systems. It ensures that numerical results remain predictable, repeatable, and robust even when performed on different hardware or with different compilers. This universality has made IEEE 754 essential in high-performance computing, real-time systems, embedded design, and virtually every field that relies on digital computation.

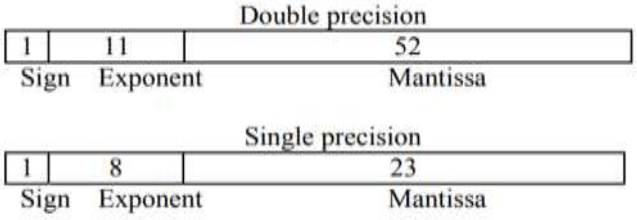

Fig. 1 Floating point formats in the proposed model

Floating-point multiplication is one of the most frequently employed arithmetic operations in an FPU (Floating Point Unit) and serves as a critical component in numerous engineering domains, including image processing, video processing, and signal processing. Due to its ability to represent both extremely large and very small numerical values, the floating-point format offers a significantly wider dynamic range compared to fixed-point representation. This versatility makes it indispensable in various scientific and computational applications. Consequently, there is a growing demand for the design of floating-point multiplier units that are optimized for reduced propagation delay, lower hardware resource utilization, and improved energy efficiency.

II. REVIEW OF EXISTING SYSTEMS

Fig. 2 showcases the floating point multiplication procedure in three main steps. Floating-point multiplication is the process of computing the product of two floating-point numbers while maintaining compliance with the IEEE 754 standard. In single-precision (32-bit), each floating-point number is represented using three fields: a 1-bit sign, an 8-bit biased exponent, and a 23-bit fraction, also known as the significand or mantissa, with an implicit leading '1' assumed for normalized numbers. Multiplying two numbers in floating point format is done by:

1- adding the exponent of the two numbers and then subtracting the bias from their result,

2- multiplying the significand of the two numbers, and

3- calculating the sign by XORing the sign of the two numbers.

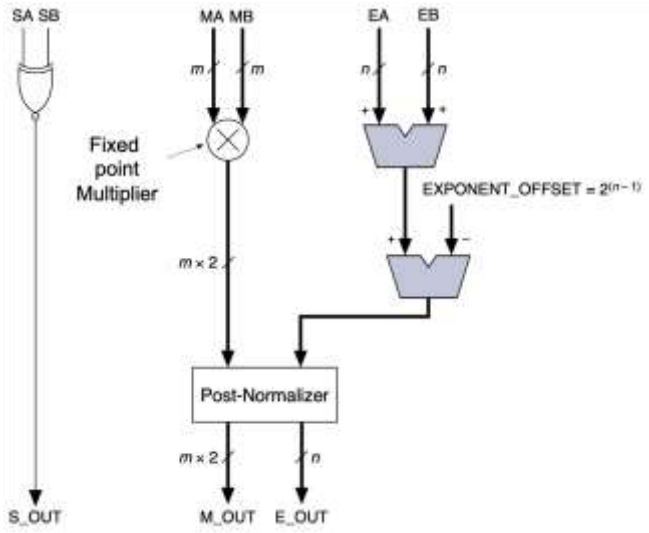

Fig. 2: Floating Point Multiplication Process

There are various other algorithms used to approach this significand multiplication process:

1. Booth multiplication algorithm
2. Array multiplication
3. Wallace tree multipliers
4. Dadda multipliers
5. Vedic multiplication (Urdhva Tiryak Sutra)

A) Booth Multiplication Algorithm

The Booth multiplication algorithm is an efficient technique for performing signed binary multiplication and is widely used in hardware implementations of floating-point multipliers. Its primary advantage lies in reducing the number of partial products, which in turn lowers the computational complexity and latency of the multiplier circuit. The algorithm achieves this reduction by encoding the multiplier in such a way that sequences of consecutive 1s are handled using fewer arithmetic operations.

In its basic form, known as Radix-2 Booth encoding, the algorithm examines two bits of the multiplier at a time—the current bit and the previous bit—to determine whether an addition, subtraction, or no operation is required. The encoding rules are as follows:

Put multiplicand in BR and multiplier in QR and then the algorithm works as per the following conditions :

1. If Qn and Qn+1 are same i.e. 00 or 11 perform arithmetic shift by 1 bit.

2. If Qn Qn+1 = 01 do A= A + BR and perform arithmetic shift by 1 bit.

3. If Qn Qn+1 = 10 do A= A – BR and perform arithmetic shift by 1 bit.

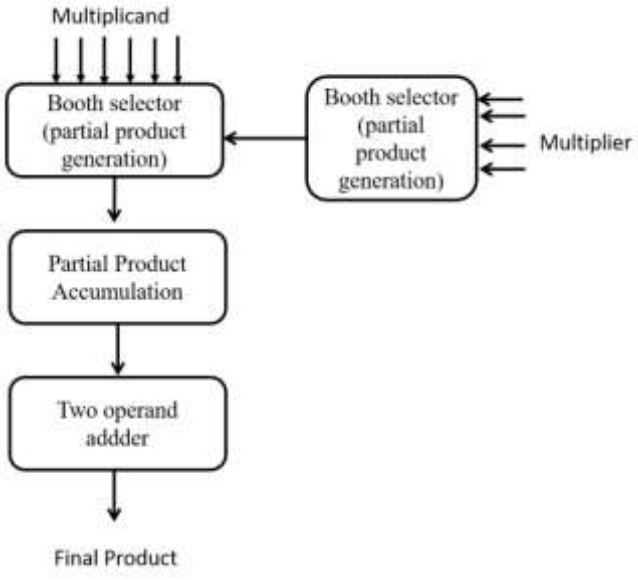

Fig. 3: Basic Booth Algorithm Block

Booth multiplication is smaller, faster multiplication algorithm through encoding the signed numbers to 2's complement, which is also a standard technique used in chip design, and provides significant improvements by reducing the number of partial product to half over "long multiplication" techniques. Modified Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation.

B) Array Multiplier

Array multipliers are derived from the concept of parallelogram-style multiplication, so named due to the geometric resemblance of their structure to a parallelogram. In this architecture, a grid of parallel adders is arranged such that each stage receives partial product inputs and propagates carry outputs to the subsequent row. All partial products are generated concurrently, allowing for efficient computation. The structure of an array multiplier can be conceptually divided into vertical and horizontal components, both of which introduce delays—primarily due to full adder operations and logic gate propagation. Owing to their highly regular and compact layout, array multipliers are straightforward to implement in hardware, making them suitable for VLSI design where simplicity and uniformity are advantageous.

Because of its small and regular structure, the array multiplier is easier and faster to design compared to more complex designs like the Wallace tree multiplier. Its simplicity also makes it suitable for pipelined architectures. However, this design also has drawbacks, such as higher worst-case delay and slower overall speed, which may limit its use in high-performance systems.

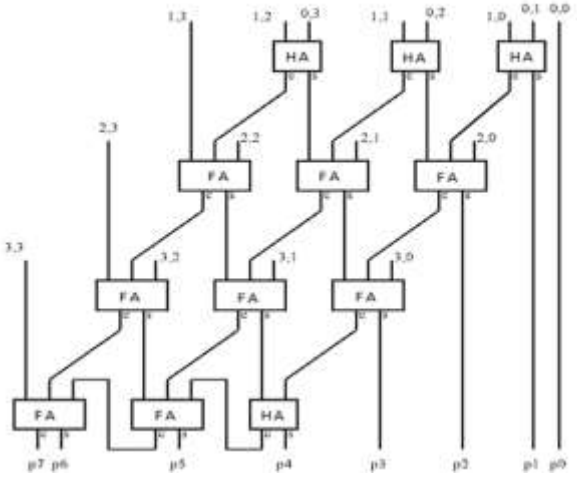

Fig 4: Array multiplier

C) Wallace Tree Multiplier

The Wallace tree multiplier is a high-speed hardware architecture used for multiplying binary numbers by efficiently reducing the number of partial products. It operates in three main stages: partial product generation, reduction, and final addition. In the first stage, partial products are generated using basic AND gates. The key innovation lies in the second stage, where a tree-like structure composed of carry-save adders (CSAs) is used to compress these partial products into two final rows—one for the sum and one for the carry. These rows are then added using a fast adder, such as a carry-lookahead or ripple-carry adder, to produce the final result.

The Wallace tree method significantly reduces the critical path delay by minimizing the number of sequential addition stages, allowing multiple additions to occur in parallel. This makes it much faster than traditional array multipliers, especially for operands with a large number of bits. However, this improvement in speed comes at the cost of increased design complexity and irregular layout, which can complicate routing and increase power consumption in VLSI implementations. Despite these challenges, the Wallace tree multiplier is widely used in performance-critical applications such as digital signal processing and high-speed arithmetic units, where low latency is more important than layout simplicity.

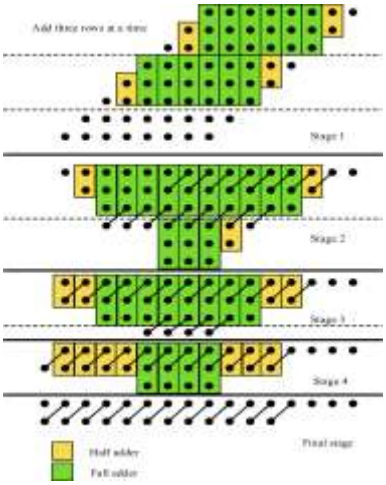

Fig. 5: Simple 8*8 implementation of Wallace Algorithm

D) Dadda multiplier

The Dadda multiplier is an optimized hardware multiplication technique that focuses on reducing the number of required components during the partial product reduction stage, while maintaining high computational speed. It is conceptually similar to the Wallace tree multiplier but differs in the strategy it employs for compressing partial products. The algorithm was introduced by Luigi Dadda in 1965, and it aims to minimize the number of adders used, thus reducing the hardware complexity compared to Wallace tree multipliers. Like other parallel multipliers, the Dadda algorithm proceeds in three primary steps:

1. Partial product generation:All partial products are generated simultaneously using a set of AND gates, one for each bit-pair between the multiplicand and multiplier.

2. Reduction of partial products: Instead of immediately reducing the partial products at every level—as in the Wallace tree—the Dadda approach delays reduction as long as possible. It uses a threshold-based reduction scheme, where the maximum allowed number of bits in each column is determined in advance for each stage. This ensures that each stage uses the fewest possible full adders and half adders, which helps optimize area without significantly compromising speed. The reduction continues stage-by-stage until only two rows of partial products remain.

3. Final addition: The last two rows are added using a fast adder, such as a carry-lookahead or carry-select adder, to produce the final product.

The primary advantage of the Dadda algorithm lies in its balanced trade-off between speed and hardware efficiency. While its performance in terms of delay is comparable to the Wallace tree approach, it generally consumes less area because of its more conservative use of adders during the reduction phase. This makes the Dadda multiplier especially attractive for VLSI implementations where area efficiency is a concern. However, due to the structured but less regular interconnect compared to simpler architectures like array multipliers, Dadda multipliers can be more difficult to route and verify, especially as operand widths increase. Despite this, they remain a popular choice for medium- to high-performance multiplication in processors and digital signal processing units

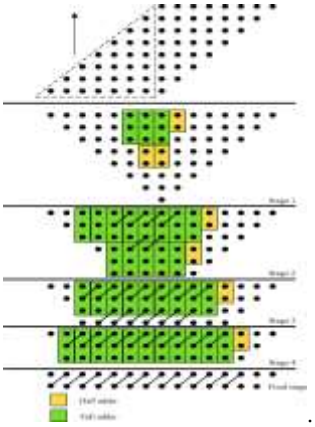

Fig. 6: Simple 8*8 Dadda algorithm implementation

E) Vedic Multiplier

The speed of a multiplier during the accumulation of partial products is often limited by the carry propagation in adders. This limitation can be addressed using techniques from Vedic mathematics, which enable single-step summation and help reduce delay. The core concept of a Vedic multiplier is rooted in Vedic mathematics, an ancient computational system. This methodology is based on sixteen mathematical sutras that cover various branches of mathematics, including calculus, trigonometry, conic sections, and geometry. Today, Vedic multipliers are widely applied in communication systems and digital signal processing.

Among the sixteen sutras, the most commonly employed in multiplier design are the Urdhva Tiryakbhyam and Nikhilam sutras. The Urdhva Tiryakbhyam approach, in particular, is based on the principles of vertical and crosswise multiplication. This allows for faster multiplication and makes it well-suited for applications requiring high-speed arithmetic operations.

A basic illustration of a 4×4 Vedic multiplier is typically used to demonstrate this method. In the context of arithmetic circuit design, having a fast and efficient multiplier is essential. Vedic multipliers offer several advantages:

1. They enhance the speed of multiplication.

2. They improve the overall efficiency of the multiplier.

3. They reduce time and path delays.

4. They occupy less hardware area.

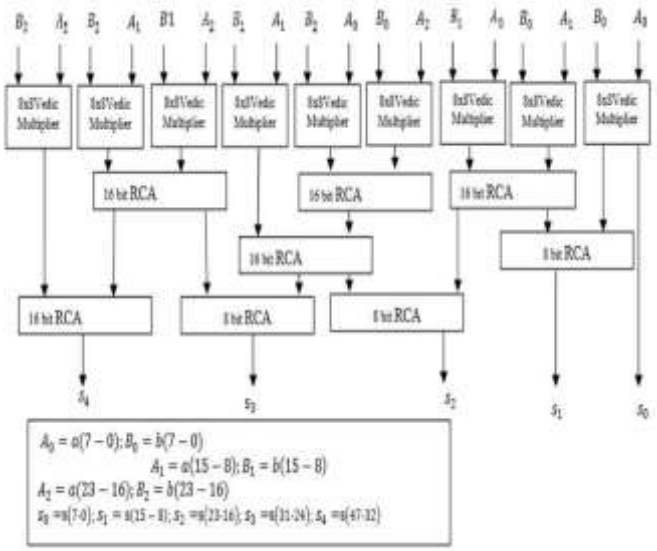

Fig.7: 24 bit Vedic Multiplier

III. CONCLUSION

Floating-point multiplication is a fundamental operation in modern computing systems, particularly in domains such as signal processing, scientific computing, and embedded applications. This review has presented a comparative analysis of various multiplier architectures, including array, Wallace tree, Dadda, Booth, Vedic, and other algorithmic approaches. Each design offers distinct trade-offs in terms of speed, area, power consumption, and implementation complexity. While traditional methods like array multipliers provide simplicity and regularity, advanced techniques such as Wallace and Dadda multipliers significantly enhance performance by reducing partial product accumulation delay. Algorithmic innovations like Booth encoding and Vedic multiplication offer further optimization tailored to specific applications.

The choice of multiplier architecture depends heavily on design requirements such as latency constraints, area limitations, and energy efficiency. As the demand for high-performance and low-power computing continues to grow, future research will likely focus on hybrid and adaptive multiplier designs that combine the strengths of multiple approaches. This ongoing development will ensure that floating-point multiplication remains efficient and scalable across a wide range of computing platforms.

References

1. K.Hwang, Computer Arithmetic: Principles, Architecture and Design. New York: John Wiley & Sons, 1979.
2. Boldo, Sylvie & Jeannerod, Claude-Pierre & Melquiond, Guillaume & Muller, Jean-Michel. (2023). “Floating-point arithmetic”. Acta Numerica. 32. 203-290. 10.1017/S0962492922000101.
3. Singh, Prateek & Bhole, Kalyani. (2015). “Optimized floating point arithmetic unit”. 10.1109/INDICON.2014.7030552.
4. Addanki Puma Ramesh, A. V. N. Tilak, A.M.Prasad “An FPGA Based High Speed IEEE-754 Double Precision Floating Point Multiplier Using Verilog” 2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), pp. 1-5, 7-9 Jan. 2017
5. Ushasree G, R Dhanabal, Sarat Kumar Sahoo “Implementation of a High Speed Single Precision Floating Point Unit using Verilog” International Journal of Computer Applications National conference on VSLI and Embedded systems, pp.32-36, 2019
6. M Al-Ashrafy, A Salem and W Anis, “An efficient implementation of floating point multiplier”, in Proc. IEEE Electronics, Communications and Photonics Conference (SIECPC), Saudi International, April 24-262011.
7. M Al-Ashrafy, A Salem and W Anis, “An efficient implementation of floating point multiplier”, in Proc. IEEE Electronics, Communications and Photonics Conference (SIECPC), Saudi International, April 24-262011.
8. Surendra Singh Rajpoot, Nidhi Maheshwari, D S Yadav, “ Design and implementation of efficient 32-bit floating point multiplier "International Journal Of Engineering And Computer Science ISSN:2319-7242 Volume 2 Issue 6 June 2013 Page No.2098-2101.
9. H.D.Tiwari, G.Gankhuyag, C.M. Kim,, and Y.B.Cho,"Multiplier design based on ancient Indian Vedic Mathematics,"Proc. IEEE ISOCC’08, Vol.2, pp.65-68, 2008..
10. K. D. Rao, P. V. Muralikrishna and C. Gangadhar, "FPGA Implementation of 32 Bit Complex Floating Point Multiplier Using Vedic Real Multipliers with Minimum Path Delay," 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), Gorakhpur, India, 2018, pp. 1-6, doi: 10.1109/UPCON.2018.8597031
11. B. Yeshwanth, V. Venkatesh and R. Akhil, "High-Speed Single Precision Floating Point Multiplier using CORDIC Algorithm," 2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Msyuru, India, 2018, pp. 135-141, doi: 10.1109/ICEECCOT43722.2018.9001506.
12. Buddhe, Vinod, Prasanna Palsodakar. “Design and verification of Dadda algorithm based Binary Floating Point Multiplier”, 2014 International Conference on Communication and Signal Processing, 2014.
13. K. D. Rao, P. V. Muralikrishna and C. Gangadhar, "FPGA Implementation of 32 Bit Complex Floating Point Multiplier Using Vedic Real Multipliers with Minimum Path Delay," 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), Gorakhpur, India, 2018, pp. 1-6, doi: 10.1109/UPCON.2018.8597031.
14. Prof. Jyoti Goudar1, Prof. A H Birasal, “Design of Floating Point Multiplier using Modified Wallace & Dadda Algorithms”, International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS) Volume VII, Issue III, March 2018 | ISSN 2278-2540