Review of Computer Engineering Research
2026 Vol. 13, No. 3, pp. 53-65
ISSN(e): 2410-9142
ISSN(p): 2412-4281
DOI: 10.18488/76.v13i3.5085

An efficient reconfigurable floating point multiplier design for signal processing applications

Subodh Kumar Singhal[1]
Sujit Kumar Patel[2]
Anurag Mahajan[3+]

[1]*Jaypee University of Engineering and Technology, Guna, M.P., India.*
[1]*Email: subodh.singhal@juet.ac.in*
[2]*Thapar Institute of Engineering and Technology, Patiala, Punjab, India.*
[2]*Email: sujit.patel@thapar.edu*
[3]*Department of E&TC, Symbiosis Institute of Technology, Pune Campus, Symbiosis International (Deemed University), Pune, India.*
[3]*Email: anurag.mahajan@sitpune.edu.in*

(+ Corresponding author)

Article History
Received: 12 December 2025
Revised: 24 March 2026
Accepted: 29 June 2026
Published: 13 August 2026

Keywords
Double-precision
Floating-point
FPGA
Multiplier
Single-precision
VLSI.

ABSTRACT

Floating-point arithmetic plays a vital role in modern signal processing applications due to its capability to represent a wide dynamic range of numerical values while maintaining acceptable precision. In this paper, an efficient reconfigurable floating-point multiplier architecture is proposed, specifically designed to meet the performance requirements of signal processing tasks. The proposed design exploits the flexibility of floating-point representations during mantissa computation, enabling optimized arithmetic operations. As a result, the architecture significantly reduces computational complexity, hardware resource utilization, and overall power consumption compared to conventional floating-point multiplier designs. The proposed and existing reconfigurable floating-point multiplier designs were coded in VHDL and implemented on a Kintex-7 XC7K325TFFG900-2 field-programmable gate array (FPGA). The implementation results demonstrate that the proposed modified reconfigurable floating-point multiplier achieves a 55% reduction in look-up table (LUT) utilization, a 47% reduction in critical path delay, and a 56% reduction in power consumption compared to the best existing floating-point multiplier design. The performance metrics such as area, delay, and power show that the proposed design is better compared to existing designs. Overall, the presented reconfigurable floating-point multiplier design offers a promising solution to meet the demanding computational requirements of modern signal processing systems, making it suitable for integration into high-performance digital signal processors and other embedded systems.

Contribution/Originality: This study contributes to the existing literature by presenting an optimized, reconfigurable floating-point multiplier architecture for signal processing applications. It uses a new estimation methodology for mantissa computation and resource optimization. The paper provides the first logical analysis of performance–power trade-offs in modified floating-point multiplier architectures.

1. INTRODUCTION

In many modern scientific, image processing, and machine learning computer systems, floating-point arithmetic (FPA) is used, especially in real numbers with a broad dynamic range [1-3]. The IEEE-754 floating-point standards specify three FPA formats: double-precision (64-bit), single-precision (32-bit), and half-precision (16-bit) [4, 5]. Apart from this, various signal processing applications involve floating-point arithmetic (FPA) operations such as addition, subtraction, and multiplication. Among these, floating-point multiplication is the most complex operation in terms of hardware area, power consumption, and delay. Conventional single-precision and double-precision floating-point multipliers are typically designed independently, leading to underutilization of resources when only one precision is

required. In modern FPGA-based embedded and signal processing systems, where flexibility, low power, and compact design are critical, such fixed-precision designs are inefficient.

Recent FPGA-based reconfigurable multiplier architectures aim to enhance flexibility; however, many designs suffer from increased control overhead, limited scalability, or insufficient optimization for power and area when switching between precisions. This highlights a clear research gap in developing an efficient reconfigurable floating-point multiplier that supports multiple precisions while maintaining high speed, low power consumption, and reduced hardware complexity.

The primary objective of this research is to design an efficient FPGA-based reconfigurable floating-point multiplier capable of supporting single and double-precision operations using shared hardware resources. This study contributes to existing literature by proposing a compact and power-efficient reconfigurable FPM architecture and providing a detailed comparison with recent FPGA-based reconfigurable multiplier designs to demonstrate its advantages.

Based on this objective, the research addresses the following questions:

- How can a single architecture efficiently support both single and double precision floating-point multiplication?
- How does the proposed reconfigurable design compare with existing FPGA-based multipliers in terms of area, power, and performance?

The remainder of this paper is organized as follows. Section II presents a review of related work and recent FPGA-based reconfigurable multiplier designs. Section III provides the background of floating-point multiplication. Section IV describes the proposed methodology. Section V discusses the implementation results and comparative analysis. Finally, Sections VI and VII conclude the paper and outline future work.

2. LITERATURE REVIEW

Krishnan et al. [6] analyzed the performance of floating-point multipliers realized using array, Vedic, and Wallace tree mantissa multipliers. They used a ripple-carry adder for the addition of intermediate products in mantissa multiplication. Subsequently, Ramya and Seshasayanan [7] suggested a binary-coded decimal (BCD) floating-point multiplier and a binary floating-point multiplier with a binary-to-BCD converter. In these designs, they utilized the concept of the Urdhva-Tiryakbhyam sutra for implementing mantissa multiplication. An FPGA-based single-precision FP multiplier using Booth-2 encoding and a Wallace tree structure, reported in Li [8], serves as a basic unit for convolutional neural networks. Later, a mixed-precision floating-point accelerator supporting both inference and training for IoT applications was proposed in Junaid et al. [9]. By combining multiple FP formats, the design achieves high accuracy with substantially lower area and energy compared to conventional full-precision architectures. Consequently, an approximate floating-point multiplier based on 2-D pseudo-Booth encoding with tunable precision was proposed in Towhidy et al. [10]. They also designed simplified-steering iterative multipliers to minimize power usage in the correction path. Subsequently, a top-down design strategy for approximate floating-point FFT was proposed in Yan et al. [11], by combining a mantissa bit-width adjustment algorithm with a step-by-step multiplier approximation method.

Further, in Rahaman et al. [12], the authors have developed a floating-point multiplier based on the Vedic Urdhva-Tiryakbhyam (UT) technique to meet the growing demand for low-delay, high-speed digital signal processing applications. Consequently, a high-performance, energy-efficient double-precision floating-point multiplier was presented in Zhang et al. [13] for FPGA platforms, featuring a novel mantissa-mapping technique that fully utilizes DSP blocks with fewer pipeline stages. In Anuradha et al. [14], a detailed analysis of single-precision floating-point multipliers is presented to explore the reduction of computational delay and silicon area. In this work, they have developed an improved FPM structure that outperforms conventional designs in overall efficiency and is effectively applied in image-processing tasks. Later, in Li et al. [15], the authors have introduced a

compact multi-precision multiplier that enhances hardware resource sharing in embedded floating-point DSP units within FPGAs. By using a high-precision multiplier structure to efficiently support multiple low-precision formats, the design overcomes the resource wastage seen in traditional recursive and dual-precision multipliers.

Subsequently, in Li et al. [16], the authors have proposed a runtime-reconfigurable approximate floating-point multiplier with an error-correction module controlled by configurable clock signals, addressing glitch and metastability issues. An efficient IEEE 754 floating-point multiplier implemented in the logarithmic domain using the Logarithmic Number System (LNS) was presented in Basha et al. [17] to overcome limitations of traditional multipliers. Consequently, a low-power floating-point multiplier was proposed in Tegazzini et al. [18] that use dynamic segmentation to approximate the mantissa product through simple segment addition and a small correction lookup table for reduction of power consumption. Further, a novel angle-based representation for floating-point multiplication was developed in Gan et al. [19] that eliminates DSP usage and significantly reduces hardware resources. Later, a power-efficient Hybrid Floating Point Multiplier (HFPM) was proposed in Edavoor et al. [20] using an approximate hybrid radix-4/radix-8 Booth encoder to increase operational speed and optimize hardware. Subsequently, a highly reconfigurable multi-format floating-point multiplier was developed in Kermani and Zarandi [21] that supports various formats, including both signed and unsigned versions. Consequently, designs for IEEE 754–compliant floating-point multipliers were presented in Singh et al. [22] using FPGA implementations. The most recent development in the floating-point multipliers is summarized in Table 1.

Table 1. Summary of related work on floating-point multipliers.

Ref.	Problem addressed	Method/ technique	Key contribution	Limitations	Research gap identified
Basha, et al. [17]	Complexity of traditional FP multipliers	Logarithmic Number System (LNS)	Reduced multiplication complexity	Conversion overhead	FPGA-friendly reconfigurable FP designs lacking
Tegazzini, et al. [18]	Power reduction in FP multipliers	Dynamic segmentation with correction LUT	Low-power FP multiplier	Approximation-induced errors	Need for precise, reconfigurable low-power FPM
Gan, et al. [19]	DSP block dependency	Angle-based FP representation	Eliminates DSP usage	Limited general applicability	Reconfigurable FPGA-based FP multipliers still required
Edavoor, et al. [20]	Speed and power optimization	Hybrid radix-4/radix-8 Booth encoder	Improved speed and efficiency	Approximation impact	Exact reconfigurable multiplier architectures missing
Kermani and Zarandi [21]	Multi-format FP support	Highly reconfigurable multi-format FP multiplier	Supports signed and unsigned formats	Increased control complexity	Simplified reconfigurable FP multiplier needed
Singh, et al. [22]	IEEE 754-compliant FPGA implementation	FPGA-based FP multipliers	Standards-compliant designs	Fixed precision	Reconfigurable IEEE 754 FP multiplier not explored

From the literature review, it can be observed that a reconfigurable FPM architecture could be a better choice for the development of a digital signal processing system as compared to either single-precision or double-precision FPM designs. Therefore, in this paper, an efficient reconfigurable floating-point multiplier design is developed.

3. BRIEF BACKGROUND

In this section, the single-precision, double-precision and reconfigurable floating point multiplier designs are discussed to explore the possibility of further optimization (Area, power and delay).

3.1. Single Precision Floating Point Multiplier

The conventional single-precision floating-point multiplier architecture is shown in Figure 1, which is taken from [14]. It performs the multiplication of two floating-point numbers in three parts. The first part computes the sign bit of the output by XORing the sign bits of the numbers. The second part generates the exponent of the output by adding the exponents of the numbers and subtracting the bias value. The third part produces the mantissa output by performing mantissa multiplication along with normalization. From the architecture of Figure 1, it can be observed that the single-precision FPM delay involves the delay of the mantissa multiplier and exponent normalization unit.

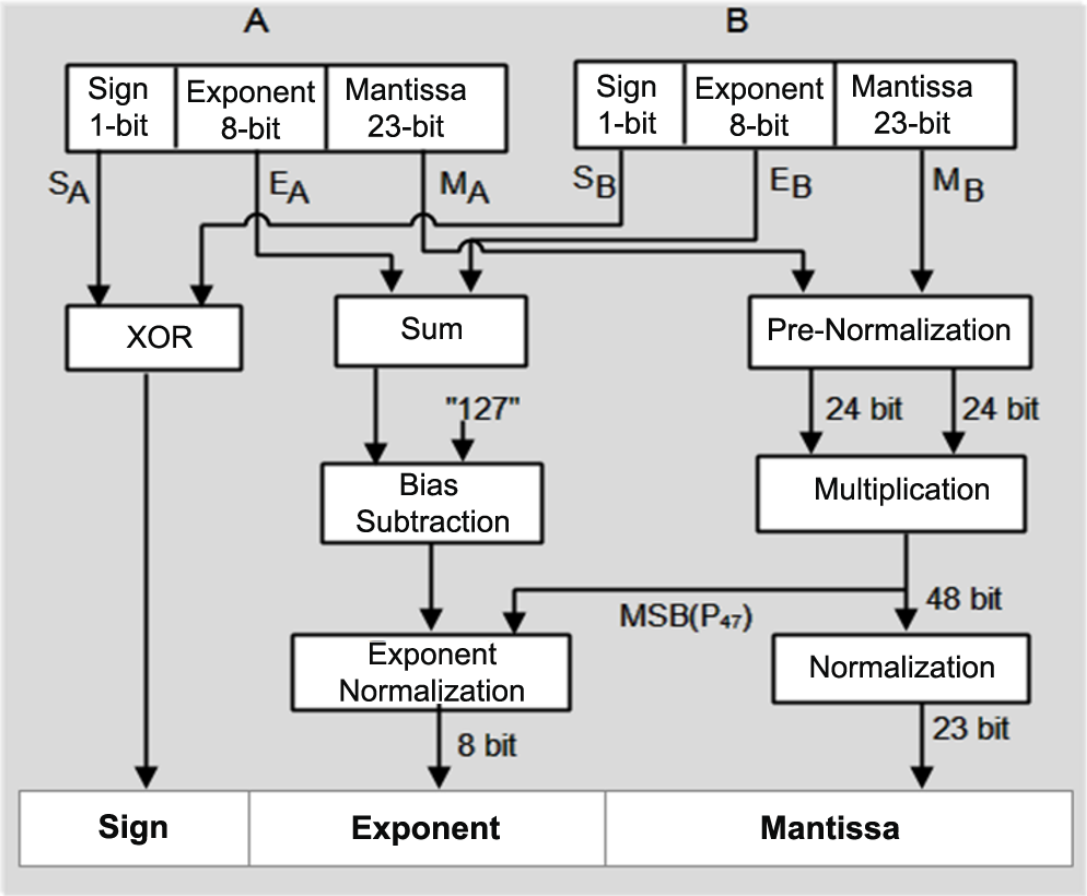

Figure 1. Conventional single precision floating-point multiplier [14].

3.2. Double Precision Floating Point Multiplier

The basic architecture of double-precision FPM is given in Figure 2, which is developed using the approach of single-precision FPM given in Anuradha et al. [14]. It consists of exponent, mantissa, and sign generation units. The operations performed in double precision FPM are the same as those of single precision FPM discussed in Subsection A. In this design, the size of the exponent and mantissa are 11 and 52, respectively. Also, the bias value is 1023 instead of 127 as in single-precision FPM.

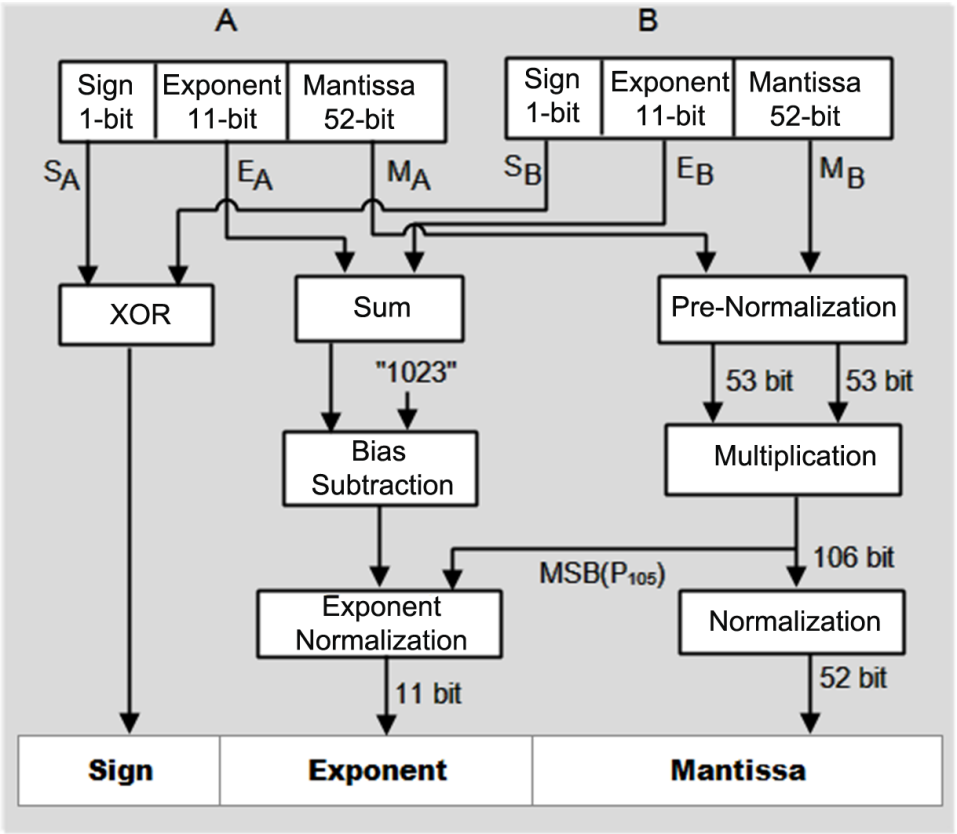

Figure 2. Block diagram of double precision floating-point multiplier [14].

Here, a large size, i.e., 53-bit mantissa multiplier, is needed, which consumes most of the area and introduces delay in the double-precision FPM. It can be observed from both the designs (single and double precision) that their architectures are almost similar, but only the size of different units varies. There are some applications, such as graphics processing units, machine learning, games, and simulations, where single-precision FPMs are used, whereas in applications like scientific computing, medical imaging, and financial modeling, double-precision FPMs are employed. Therefore, a reconfigurable (hybrid) architecture is required, capable of performing both single and double-precision floating-point multiplication. In the next subsection, a straightforward hybrid FPM design is derived.

3.3. Straight Forward Reconfigurable Floating-Point Multiplier

The straightforward reconfigurable floating-point multiplier design is developed using [14] as shown in Figure 3, which can perform two single-precision or one double-precision floating-point multiplication. It takes two 64-bit numbers as inputs, where each number represents either one double-precision number or two single-precision numbers. Therefore, this architecture either performs two single-precision FP multiplications when the select line s= '0' or performs one double-precision multiplication when s= '1'. From this architecture, it can be observed that it uses three mantissa multiplier units, two of size 24 bits and one of size 54 bits. It can also be seen that at a time, either two 24-bit multipliers are used or one 54-bit multiplier is used while performing floating-point multiplication. This means that the hardware utilization efficiency is less than 50%. Consequently, the given straightforward design is not efficient in terms of area, delay, and power. Therefore, this architecture needs to be modified so that the hardware utilization efficiency reaches nearly 100%, resulting in a more efficient architecture in terms of area, delay, and power. To achieve this, a hybrid floating-point multiplier architecture is proposed and presented in the next section.

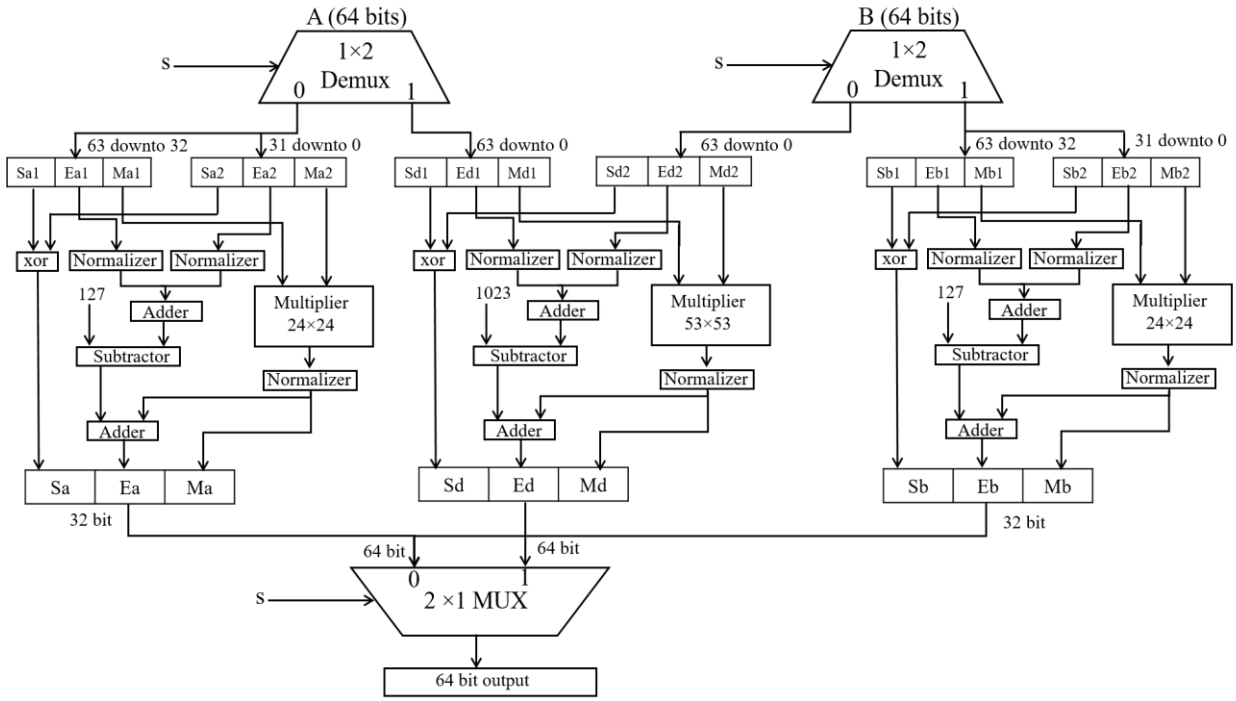

Figure 3. Straight forward reconfigurable floating point multiplier [14].

4. METHODOLOGY

The proposed reconfigurable floating-point multiplier (FPM) structure is depicted in Figure 4. It is suitable to perform the four single-precision or one double-precision FP multiplication(s) in two iterations. For this, it uses two 27-bit multipliers in place of three multipliers (two 24-bit and one 53-bit) as discussed in subsection B. The proposed architecture takes two inputs 'a' and 'b' each of 64 bits, where each input either consists of two single-precision FP numbers or one double-precision FP number. It needs two iterations to compute the double-precision floating-point multiplication. On the other hand, if it is used for single precision FP multiplication, then four single-precision point multiplications can be performed in two iterations, i.e., two single-precision FPM in each iteration.

As discussed earlier, the proposed architecture consists of two multipliers of 27 bits each. Now, the mantissa multiplication is performed through 27-bit multipliers. So, the pre-normalization of the 23-bit mantissas is done before multiplying them together. The mantissas Ma1, Ma2, and Mb1, Mb2 will be 27 bits each after normalization. As discussed earlier, the proposed model can perform two single-precision and one double-precision multiplication using the same architecture. Before multiplication, the mantissas are passed through a 2×1 multiplexer with a selection line S. If S = 0, then the multiplication takes place between the mantissas of single-precision 32-bit floating-point numbers. The first multiplication is performed between Ma1 and Ma2, and the second between Mb1 and Mb2. The outputs, each 54 bits, are passed through a 2×1 De-Mux with the same selection line as the Mux. If S=0, the output will be a mantissa of a single-precision floating-point number and further normalized. After normalization, the outputs are Ma and Mb, each 23 bits.

Simultaneously, the XOR operation is performed between the sign bits of all the inputs, one between sa1 and sa2 and another between sb1 and sb2. The outputs obtained after the XOR operation are Sa and Sb, respectively. Now, the 8-bit exponents (Ea1 and Ea2) and (Eb1 and Eb2) are added through an adder and taken in biased form, so bias subtraction of the exponents is done by subtracting 127 from both exponents.

When the selection line is kept at S = '1', the multiplication of 52-bit mantissa of a double-precision floating-point number occurs. In this model, two 27-bit multipliers are used, so the multiplication is performed by splitting the number into two parts. Before splitting, the 52-bit number is first pre-normalized to 54 bits by padding '01' at the MSB side. Now, the 54-bit numbers (m1 and m2) are split into two parts.

m1(54bit) = dph1 (27MSB bits) and dpl1 (27LSB bits).

m2(54bit) =dph2(27MSB bits) and dpl2(27LSB bits).

When itr = 0,

d11 = dpl1 × dpl2

d12 = dph1 × dpl2

When itr = 1,

d21 = dpl1 × dph2

d22 = dph1 × dph2

Now, d11 = 54 bits, d12 = 54 bits, d21 = 54 bits, d22 = 54 bits.

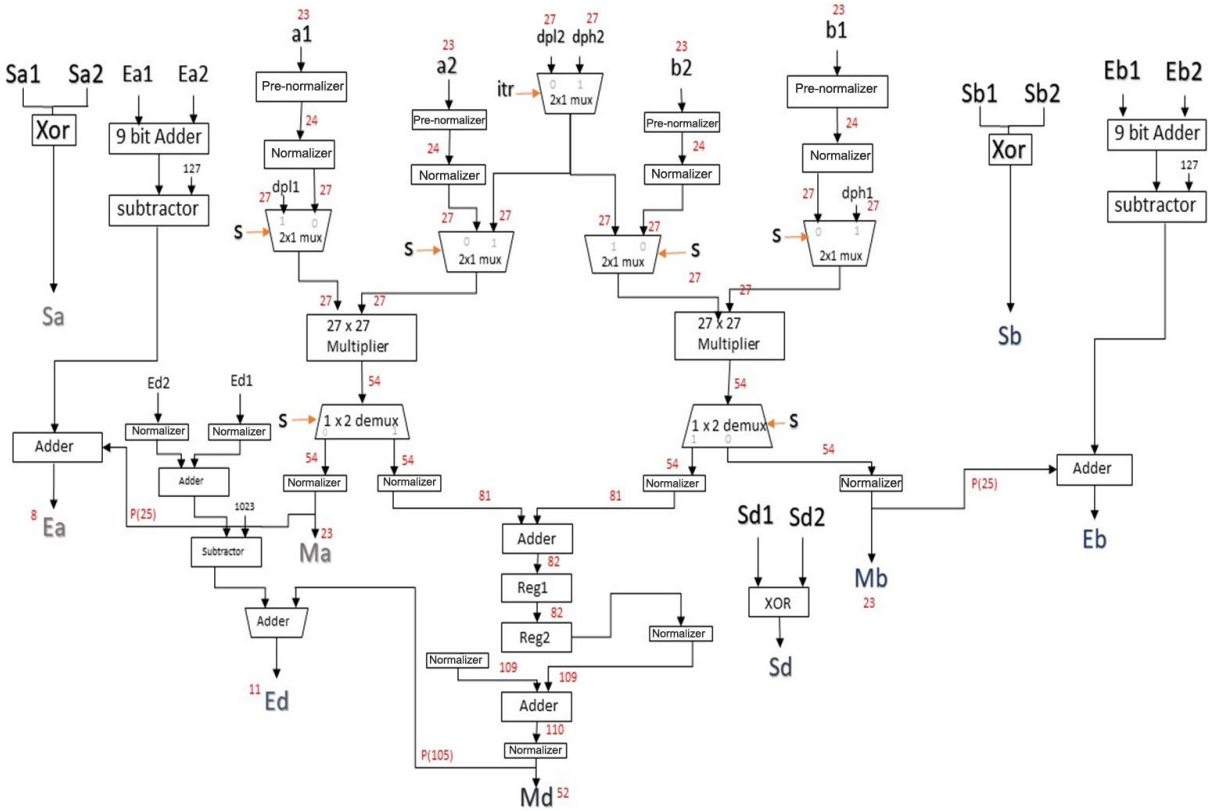

Figure 4. Proposed reconfigurable floating-point multiplier structure.

According to the algorithm, both 54-bit results will be normalized by padding with 27 zeroes. The padding is done in such a way that for d11, the zeroes are padded at the MSB side, while for d12, the zeroes are padded at the LSB side. After normalization, d11 and d12 become 81 bits. Now, they are added through an 82-bit adder (one extra bit for the carry). The 82-bit result (p1) is stored in Register-1. Similarly, for d21, the zeroes are padded at the MSB side, while for d22, the zeroes are padded at the LSB side. After normalization, d21 and d22 become 81 bits. They are then added through an 82-bit adder (one extra bit for the carry). The 82-bit result (p2) is stored in Register-2. In the further process, the 82-bit results stored in Register-1 and Register-2 are added again, but before adding, padding of 27 zeroes is done again. The padding is done such that for p1, zeroes are padded at the MSB side, while for p2, zeroes are padded at the LSB side.

Now, p1 = 82 + 27 = 109 bits

p2 = 27 + 82 = 109 bits

The result after adding p1 and p2 is p = 110 bits (including the carry bit).

Lastly, for normalizing the final mantissa, the first bit at MSB side is sent to the multiplexer for exponent normalization. After first bit, the next 5 bits are removed and 52 bits from LSB side are also removed. Hence, 52 bits mantissa (Md) is obtained.

From this design, it can be observed that it requires two 27-bit multipliers, which increases hardware complexity and does not fully utilize hardware resources. Therefore, the design is further modified to improve hardware efficiency, as discussed in the next paragraph.

Based on the above observations, the modified reconfigurable floating-point multiplier is proposed as shown in Figure 5. This modified design uses only one 27-bit multiplier to perform four single-precision or one double-precision FP multiplication in four iterations. To perform single-precision multiplication, S=0 is set, and the 23-bit mantissas of two single-precision floating-point numbers are first pre-normalized to 24 bits by padding '1' at the MSB side. Then, they are normalized to 27 bits by padding '000' at the MSB. For single-precision multiplication, the selection line of the multiplexer is kept at S=0. The SP mantissas are multiplied through a 27-bit multiplier. The 54-bit result passes through a demux with the same selection line. The 54-bit result is normalized to a final mantissa (M1) of 23 bits by removing 25 LSB bits and 5 MSB bits. The 6th MSB bit is given to the Mux for exponent normalization. In this way, four single-precision multiplication outputs are obtained in four iterations.

Similarly, to perform double-precision operations, S=1 is set. To multiply two double-precision floating-point numbers, the 52-bit mantissas are first pre-normalized to 54 bits by padding '01' at the MSB side. Since a 27-bit multiplier is used for multiplying mantissas, the 54-bit mantissas are split into 27-bit numbers. The further multiplication occurs according to an algorithm.

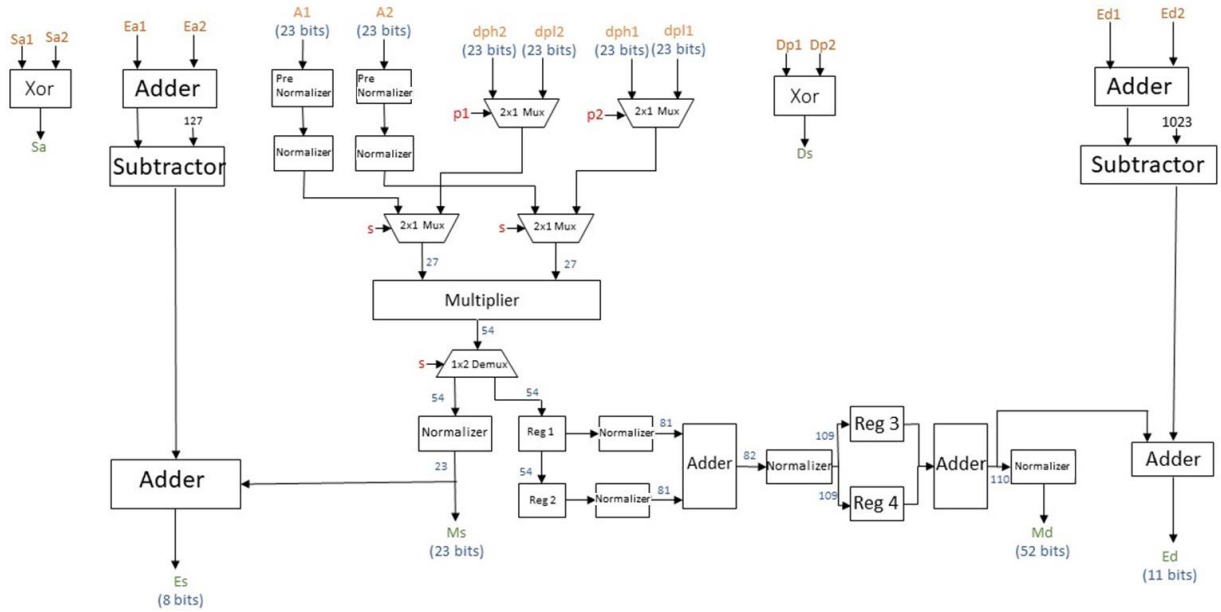

Figure 5. Proposed modified reconfigurable floating-point multiplier structure.

In the first iteration, select p1 = 0 and p2 = 0, so that multiplication takes place between dpl1 and dpl2. The 54-bit result (r1) will be stored in Register-1. Later, in the second iteration, select p1 = 1 and p2 = 0, so that the multiplication is performed in dph1 and dpl2. The 54-bit result (r2) will be stored in Register-2. According to the algorithm, both 54-bit results will be normalized by padding with 27 zeroes. The padding is done such that, for r1, the zeroes are padded at the MSB side, while for r2, the zeroes are padded at the LSB side. After normalization, r1 and r2 become 81 bits. Now, they are added through an 82-bit adder (one extra bit for the carry). The 82-bit result (z1) is stored in Register 3.

Further in iteration 3, select p1 = 0 and p2 = 1; the multiplication takes place between dpl1 and dph2. The 54-bit result (r3) will be stored in Register 1. Later, in iteration 4, select p1 = 1 and p2 = 1; the multiplication takes place

between dph1 and dph2. The 54-bit result (r4) will be stored in Register 2. According to the algorithm, both 54-bit results will be normalized by padding with 27 zeroes. The padding is done such that, for r3, the zeroes are padded at the MSB side, while for r4, the zeroes are padded at the LSB side. After normalization, r1 and r2 become 81 bits. They are then added through an 82-bit adder (one extra bit for the carry). The 82-bit result (z2) is stored in Register 4. In the further process, the results stored in Register 3 and 4 are added again, but before adding, padding of 27 zeroes is done again. The padding is such that, for z1, the zeroes are padded at the MSB side, while for z2, the zeroes are padded at the LSB side.

Now, $z1 = 82 + 27 = 109$ bits

$z2 = 27 + 82 = 109$ bits

The result after adding z1 and z2 is $z = 110$ bits (including the carry bit). Lastly, for normalizing the final mantissa, the first bit at the MSB side is sent to the multiplexer for exponent normalization. After the first bit, the next five bits are removed, and 52 bits from the LSB side are also removed. Hence, a 52-bit mantissa (M2) is obtained.

The numerical examples considered to illustrate the operation of the modified reconfigurable FPM in single and double precision modes. The proposed modified FPM design performs either four single-precision FPMs (each iteration computes one multiplication) or one double-precision FPM in four iterations. The Figure 6 and Figure 7 demonstrate the process of single precision and double precision modes of the proposed design.

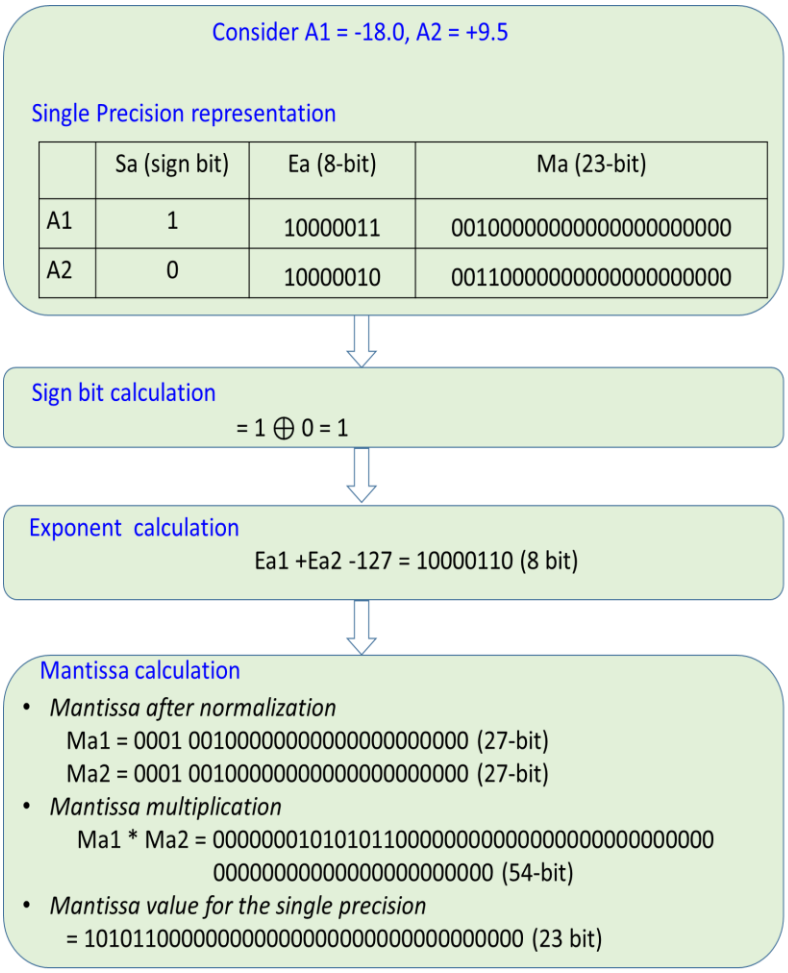

Figure 6. Numerical example to illustrate the operation of proposed modified reconfigurable floating-point multiplier in single precision mode in one iteration.

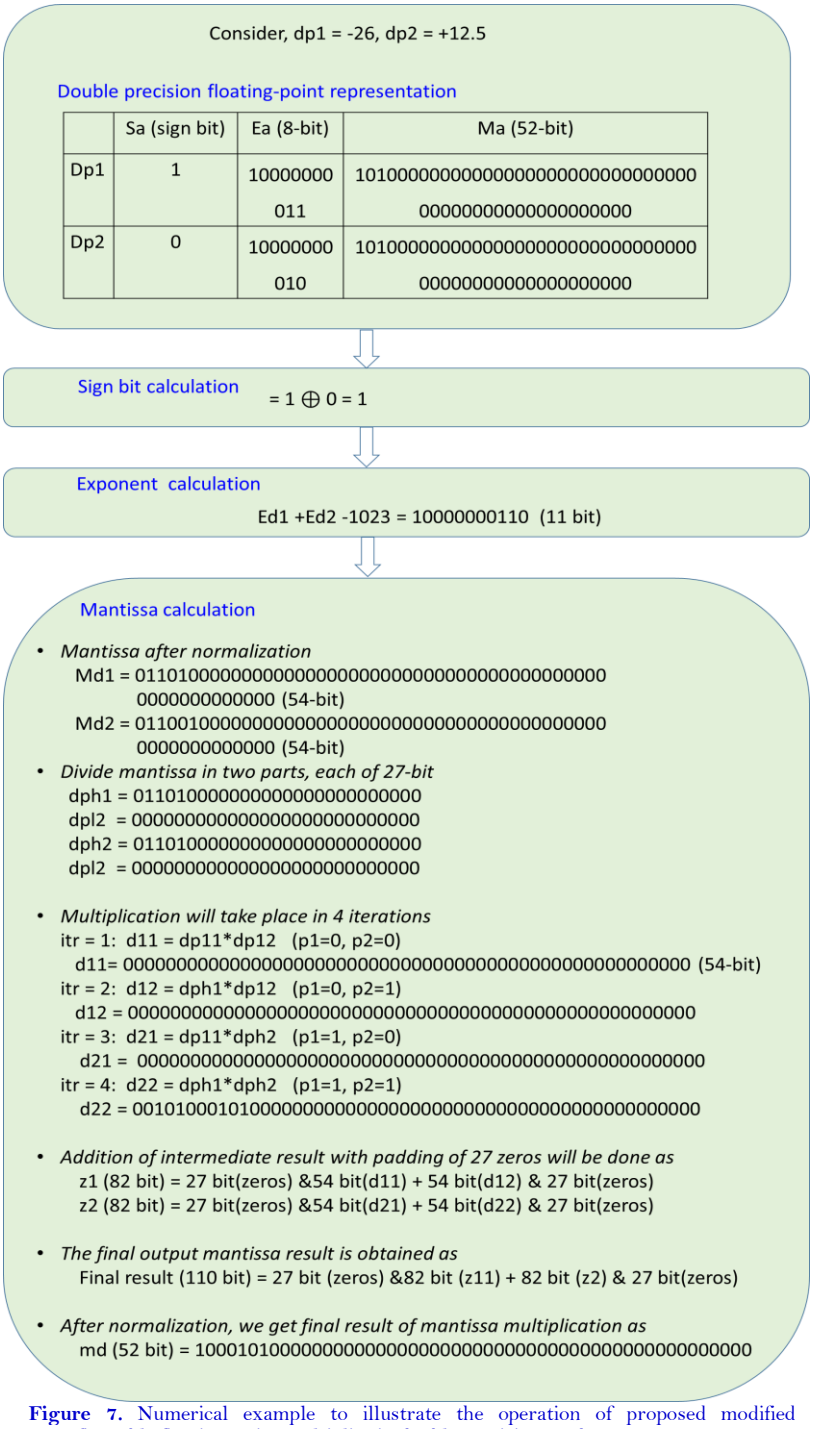

Figure 7. Numerical example to illustrate the operation of proposed modified reconfigurable floating-point multiplier in double precision mode.

5. RESULTS AND DISCUSSION

In this section, the simulation and FPGA synthesis results will be presented to show the functional correctness and performance of proposed design over the existing design.

5.1. Simulation Results

To check the validity of the proposed reconfigurable FPM and its modified version, these designs are coded in VHDL and simulated using the iSim simulator. The obtained simulation outputs are shown in Figure 8 and Figure 9.

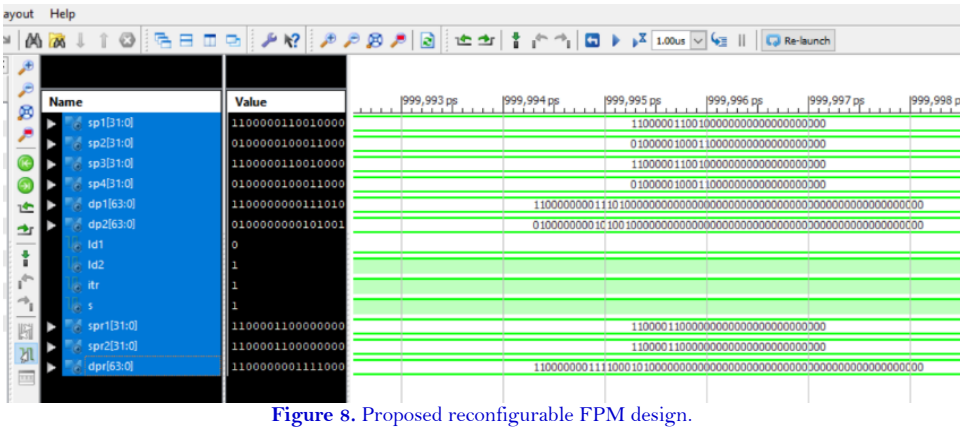

Figure 8. Proposed reconfigurable FPM design.

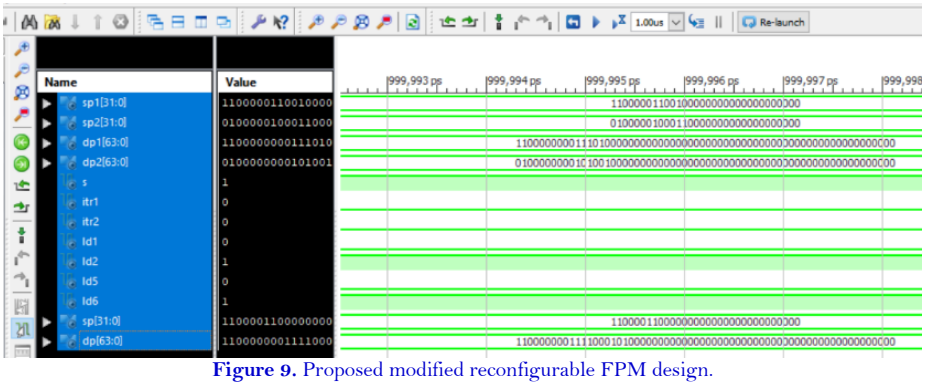

Figure 9. Proposed modified reconfigurable FPM design.

5.2. Data

For the comparative analysis, no human/real-world data is involved. For the comparison, we have extracted the reconfigurable FPM architectures from the existing design of Edavoor et al. [20] and Kermani and Zarandi [21].

5.3. Synthesis Results

All these existing architectures and the proposed FPM designs are coded in VHDL and performed the synthesis using the Vivado tool for the Xilinx Kintex-7 XC7K325TFFG900-2. The results obtained after synthesis of all the proposed and existing architectures are shown in Table 2. From Table 2, it can be observed that the proposed reconfigurable FPM design and its modified version are more efficient in terms of area (LUT and FF count), delay, and power over the existing designs of Edavoor et al. [20] and Kermani and Zarandi [21]. The implementation results demonstrate that the proposed modified reconfigurable floating-point multiplier achieves 55% reduction in look-up-table (LUT) utilization, 47% reduction in critical path delay, and 56% reduction in power consumption compared to the best existing floating-point multiplier design [21]. The performance metrics, such as area, delay and power, show that the proposed design is better compared to the existing designs. Overall, the presented reconfigurable floating point multiplier design offers a promising solution to meet the demanding computational requirements of modern signal processing systems, making it suitable for integration into high-performance digital signal processors and other embedded systems.

Table 2. VIVADO Synthesis Results for the XILINX KINTEX-7 XC7K325TFFG900-2.

Design	LUT Count	FF Count	Delay (ns)	Power (W)
Existing reconfigurable FPM [20].	3912	384	32.308	188.91
Existing reconfigurable FPM [21].	2958	243	24.32	156.12
Proposed reconfigurable FPM	2358	135	12.44	112.65
Proposed modified reconfigurable FPM	1306	322	12.866	68.293

6. CONCLUSION

Floating-point arithmetic is essential in signal processing applications for its ability to handle a wide range of values with reasonable precision. In this paper, reconfigurable floating-point multiplier designs are proposed for signal processing tasks. The synthesis results reveal that the proposed reconfigurable FPM design and its modified version are more efficient in terms of area (LUT and FF count), delay, and power compared to the straightforward design. It can also be observed that the modified version is more efficient compared to the proposed reconfigurable FPM. Thus, the modified reconfigurable FPM demonstrates superior efficiency over both the straightforward FPM and the initial reconfigurable FPM design. It could be scalable for higher precision; however, it needs to redesign the architecture of the multiplier to achieve an efficient design.

7. FUTURE RESEARCH

The proposed architecture can be further optimized at the circuit level, and the advantage of configurability can be further explored based on the demand of different applications.

Funding: This study received no specific financial support.
Institutional Review Board Statement: Not applicable.
Transparency: The authors state that the manuscript is honest, truthful, and transparent, that no key aspects of the investigation have been omitted, and that any differences from the study as planned have been clarified. This study followed all writing ethics.
Competing Interests: The authors declare that they have no competing interests.
Authors' Contributions: All authors contributed equally to the conception and design of the study. All authors have read and agreed to the published version of the manuscript.

REFERENCES

[1] R. C. Gonzalez and R. E. Woods, *Digital image processing*, 3rd ed. New Delhi, India: Pearson Education, 2009.

[2] M. Al-Ashrafy, A. Salem, and W. Anis, "An efficient implementation of floating point multiplier," in *Proceedings of the Saudi International Electronics, Communications and Photonics Conference (SIECPC). IEEE*, 2011, pp. 1–5.

[3] S.-R. Kuang, K.-Y. Wu, and K.-K. Yu, "Energy-efficient multiple-precision floating-point multiplier for embedded applications," *Journal of Signal Processing Systems*, vol. 72, no. 1, pp. 43-55, 2013. https://doi.org/10.1007/s11265-012-0695-1

[4] K. V. Gowreesrinivas and P. Samundiswary, "Comparative analysis of single precision floating point multiplication using compressor techniques," in *Proceedings of the International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). IEEE*, 2017, pp. 2428–2433.

[5] S. Gargave, Y. Agrawal, and R. Parekh, *Single-precision floating point matrix multiplier using low-power arithmetic circuits. In A. Garg, V. S. K. V. Rao, & R. K. Singh (Eds.), Advances in Power Systems and Energy Management (Lecture Notes in Electrical Engineering*. Singapore: Springer, 2018.

[6] R. V. Krishnan, S. A. Rajiv, and R. N. Deborah, "A comparative study on the performance of FPGA implementations of high-speed single-precision binary floating-point multipliers," in *Proceedings of the 2nd International Conference on Smart Systems and Inventive Technology (ICSSIT 2019). IEEE*, 2019, pp. 1041–1045.

[7] V. Ramya and R. Seshasayanan, "Low power single precision BCD floating-point Vedic multiplier," *Microprocessors and Microsystems*, vol. 72, p. 102930, 2020. https://doi.org/10.1016/j.micpro.2019.102930

[8] H. Li, "A single precision floating point multiplier for machine learning hardware acceleration," in *Proceedings of the 2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS). Piscataway, NJ: IEEE*, 2021, pp. 674–677.

[9] M. Junaid, S. Arslan, T. Lee, and H. Kim, "Optimal architecture of floating-point arithmetic for neural network training processors," *Sensors*, vol. 22, no. 3, p. 1230, 2022. https://doi.org/10.3390/s22031230

[10] A. Towhidy, R. Omidi, and K. Mohammadi, "On the design of iterative approximate floating-point multipliers," *IEEE Transactions on Computers*, vol. 72, no. 6, pp. 1623-1635, 2023. https://doi.org/10.1109/TC.2022.3216465

[11] C. Yan, X. Zhao, T. Zhang, J. Ge, C. Wang, and W. Liu, "Design of high hardware efficiency approximate floating-point FFT processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 11, pp. 4283-4294, 2023. https://doi.org/10.1109/TCSI.2023.3298882

[12] S. Rahaman, C. Krishna, K. Nunna, D. Jeethendra, and M. N. Chaithanya, "Floating point multipliers based on Vedic mathematics," in *Proceedings of the 2023 International Conference on Network, Multimedia and Information Technology (NMITCON). Piscataway, NJ: IEEE*, 2023, pp. 1–7.

[13] H. Zhang, X. Zheng, and S.-B. Ko, "High performance and energy efficient floating-point multiplier on FPGA," in *Proceedings of the 2023 IEEE 3rd International Conference on Computer Systems (ICCS). Piscataway, NJ: IEEE*, 2023, pp. 178–182.

[14] Anuradha, S. K. Patel and S. K. Singhal, "An area-delay efficient single-precision floating-point multiplier for VLSI systems," *Microprocessors and Microsystems*, vol. 98, p. 104798, 2023. https://doi.org/10.1016/j.micpro.2023.104798

[15] Y. Li, Z. Huang, G. Cai, and R. Chen, "A multi-precision floating-point multiplier structure applied to FPGA embedded DSP," in *Proceedings of the 2023 6th International Conference on Artificial Intelligence and Pattern Recognition (AIPR), Xiamen, China: ACM*, 2023, pp. 932–939.

[16] Z. Li *et al*., "Efficient approximate floating-point multiplier with runtime reconfigurable frequency and precision," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 7, pp. 3533-3537, 2024. https://doi.org/10.1109/TCSII.2024.3364717

[17] S. J. Basha, R. T. S. Eswar, T. S. Vidya, S. A. Kumar, S. Afreen, and B. V. V. Satyanarayana, *FPGA implementation of optimized floating point multiplier using minimal error logarithmic approach. In P. Pareek, S. Mishra, M. J. C. S. Reis, & N. Gupta (Eds.), Cognitive Computing and Cyber Physical Systems (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Cham, Switzerland: Springer 2025.

[18] L. Tegazzini, G. Di Meo, D. De Caro, and A. G. M. Strollo, "Design of a hardware-efficient floating-point multiplier with dynamic segmentation," in *Proceedings of the 2024 19th Conference on Ph.D Research in Microelectronics and Electronics (PRIME). Piscataway, NJ: IEEE*, 2024, pp. 1–4.

[19] B. Gan, K. Wang, G. Wang, H. Zheng, and G. Chen, "A floating-point multiplier based on angle representation method," *International Journal of Circuit Theory and Applications*, vol. 52, no. 12, pp. 6479-6487, 2024. https://doi.org/10.1002/cta.4151

[20] P. J. Edavoor, A. K. Samantaray, and A. D. Rahulkar, "Design of floating point multiplier using approximate hybrid Radix-4/Radix-8 booth encoder for image analysis," *e-Prime-Advances in Electrical Engineering, Electronics and Energy*, vol. 8, p. 100546, 2024. https://doi.org/10.1016/j.prime.2024.100546

[21] H. A. Kermani and A. A. E. Zarandi, "An efficient multi-format low-precision floating-point multiplier," *Sustainable Computing: Informatics and Systems*, vol. 41, p. 100928, 2024. https://doi.org/10.1016/j.suscom.2023.100928

[22] U. Singh, R. Mundliya, R. Maharana, and B. Jajodia, "Floating point multipliers on FPGAs," in *Proceedings of the 2025 Devices for Integrated Circuit (DevIC). Piscataway, NJ: IEEE*, 2025, pp. 702–707.