FPGA Implementation of Optimized Floating Point Multiplier Using Minimal Error Logarithmic Approach

Sheik Jameer Basha, Rakurthi Tharun Sai Eswar, Talluri Sai Vidya, Sannapu Aravind Kumar, Shaik Afreen, and B. V. V. Satyanarayana(✉)

ECE Department, Vishnu Institute of Technology, Bhimavaram 534202, India
satyanarayana.b@vishnu.edu.in

Abstract. Floating numbers are represented using IEEE standard 754. IEEE 754 defines formats for representing single-precision and double-precision floating-point numbers, along with rules for performing basic arithmetic operations on these numbers. The standard specifies the following components for both 3single-precision (32-bit) and double-precision (64-bit) floating-point numbers. This project introduces an efficient approach to IEEE 754 floating-point multiplication by implementing it in the logarithmic domain using Logarithmic Number System (LNS). It overcomes the limitations of the traditional floating pointing multipliers. By utilizing logarithmic and antilogarithmic converters, the logarithmic multiplier allows multiplication through addition, enabling support for higher accuracy levels. Floating point multipliers play a vital role in high-power computing applications like image and signal processing. This project approach presents a providing solution to the challenges posed by floating-point multiplication, offering improved performance, reduced delay, and low power consumption for demanding computational tasks in various applications. The multiplier is implemented using Verilog HDL, targeted on Spartan-3E.

Keywords: IEEE Standard 754 · Floating Point Multiplier · Log and Antilog Converters · Minimal Error Approach

1 Introduction

1.1 VLSI in Modern Tech

The field of Very Large Scale Integration (VLSI) holds paramount significance in contemporary technological landscapes due to its capability to accommodate millions to billions of transistors onto a single chip [1]. This miniaturization paves the way for the development of compact yet potent devices, including smart phones, IoT sensors, and high-performance computing systems. The integration of numerous transistors facilitates the implementation of intricate functionalities within a solitary device, fostering the creation of robust processors, sophisticated memory systems, and versatile chips [2, 3]. By

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2025
Published by Springer Nature Switzerland AG 2025. All Rights Reserved
P. Pareek et al. (Eds.): IC4S 2024, LNICST 598, pp. 171–188, 2025.
https://doi.org/10.1007/978-3-031-77078-4_16

amalgamating multiple functions onto a solitary chip, VLSI substantially diminishes the overall cost associated with manufacturing, assembling, and maintaining electronic devices, thus fostering innovation across diverse sectors such as artificial intelligence, machine learning, telecommunications, automotive technology, healthcare, and beyond [4, 5].

Moreover, VLSI technology has spearheaded notable advancements in energy efficiency, incorporating features for power management aimed at regulating and optimizing power consumption. These features encompass sleep modes, power domains, and adaptive voltage scaling, enabling intelligent power management tailored to workload and system requisites [6]. Furthermore, VLSI facilitates the integration of energy harvesting mechanisms into electronic devices, leveraging ambient energy sources like solar, thermal, or kinetic energy to enhance operational efficiency [7]. Beyond its efficiency, VLSI offers compactness, reduced power consumption compared to discrete components, accelerated data processing speeds, and heightened computational capabilities, thereby enabling efficient signal processing, data transmission, and reception for wireless networks, cellular devices, and the Internet of Things (IoT) [8].

1.2 Multiplier in Computing Systems

Multipliers serve as indispensable components in digital signal processing (DSP) and a myriad of computational tasks, primarily owing to their efficacy in executing complex arithmetic operations, notably multiplication. Foundational to various mathematical and computational algorithms, multipliers expedite tasks involving repetitive multiplication, such as matrix operations and polynomial evaluations. Particularly in DSP applications encompassing audio processing, image processing, and communication systems, multipliers play a pivotal role in implementing filters, convolution operations, and other mathematical transformations critical for processing digital signals. Moreover, algorithms like Fast Fourier Transform (FFT) and finite impulse response (FIR) filters heavily lean on efficient multiplication implementations, underscoring multipliers' contributions to algorithmic efficiency and speed, especially in dedicated hardware environments offering superior power efficiency compared to software-based alternatives. [9, 10] In domains like electrical engineering and physics, where complex numbers are pervasive, multipliers are indispensable for efficiently managing complex arithmetic, while in linear algebra, where matrix multiplication forms the backbone of solving linear equations and executing transformations, multipliers are pivotal for ensuring efficient computation in diverse applications such as graphics rendering and scientific simulations [11].

Furthermore, in the realm of VLSI (Very Large Scale Integration), multipliers stand as elemental building blocks executing multiplication operations swiftly and resource-efficiently, critical for applications demanding real-time or high-throughput performance, such as DSP. VLSI-optimized multipliers are engineered to balance compactness with high performance, minimizing chip area occupation while maximizing computational efficiency [12]. This emphasis on power efficiency is particularly crucial in battery-operated devices and other power-sensitive applications, where energy-efficient multipliers significantly contribute to overall system energy savings. In essence, multipliers represent foundational constituents in digital systems, shaping the landscape of

computational efficiency, power optimization, and resource utilization across a spectrum of applications reliant on efficient and high-speed multiplication operations [13].

1.3 Standard Representation of Floating Point Number

The intricacies of floating-point representation, encompassing sign bits, mantissas, and exponents, play a vital role in facilitating numerical precision, accuracy, and consistency across diverse computational applications [14]. The IEEE 754 format serves as a benchmark for standardizing these representations, ensuring reliable and efficient handling of real numbers in various computing environments and the standard is shown in the Fig. 1.

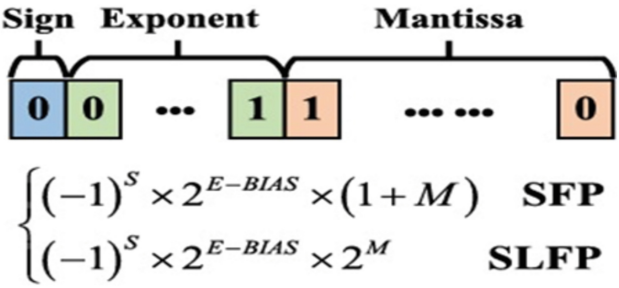


Fig.1. IEEE Representation of Floating Point Number

The proposed multiplier directly aligns with the overarching goals of advancing computational efficiency and accuracy in intelligent systems. Cognitive computing involves the emulation of human thought processes through advanced algorithms and data analytics, while cyber-physical systems integrate computational elements with physical processes to enable real-time decision-making and control. Our study addresses both aspects by proposing a novel approach to floating-point multiplication on FPGA platforms, which are pivotal in the implementation of cyber-physical systems due to their high performance and reconfigurability. By employing a minimal error logarithmic method, we not only enhance computational precision but also optimize resource utilization, thereby contributing to the development of more efficient and reliable cognitive computing and cyber-physical systems. This paper explores the intricate relationship between computational methodologies and the broader theme of cognitive computing and cyber-physical systems, underscoring the significance of our research in advancing these fields.

2 Traditional Floating Point Multiplier

Traditional multipliers constitute essential digital circuits designed for binary multiplication, employing algorithms like the array multiplier and Booth′s algorithm. The former utilizes an array of AND gates and adders to generate and accumulate partial products, while the latter optimizes multiplication by considering bit sequences, thereby reducing computational load. Design Trade-offs between speed and chip area influence design decisions, with more complex algorithms providing faster multiplication at the

expense of increased hardware resources [15, 16]. Multipliers exploit parallelism to enhance performance, breaking down the process into multiple stages for simultaneous execution. Bit-serial multipliers, operating on one bit at a time, offer advantages in simplicity and efficiency, particularly in area-constrained applications [17]. They often incorporate pipelining techniques to improve throughput by enabling concurrent operation of different multiplier stages. Traditional multipliers find widespread application in digital signal processors, microprocessors, and various embedded systems where binary multiplication is integral [18].

The research and development in multiplier design focus on enhancing efficiency and performance through novel architectures and algorithms. Integration of approximate computing techniques within multipliers aims to leverage error resilience for faster and more power-efficient results in certain applications [19]. Advancements in semiconductor technology, such as smaller process nodes and three-dimensional integration, offer opportunities to increase transistor density and reduce power consumption in traditional multiplier designs [20, 21]. These ongoing innovations ensure the continued relevance

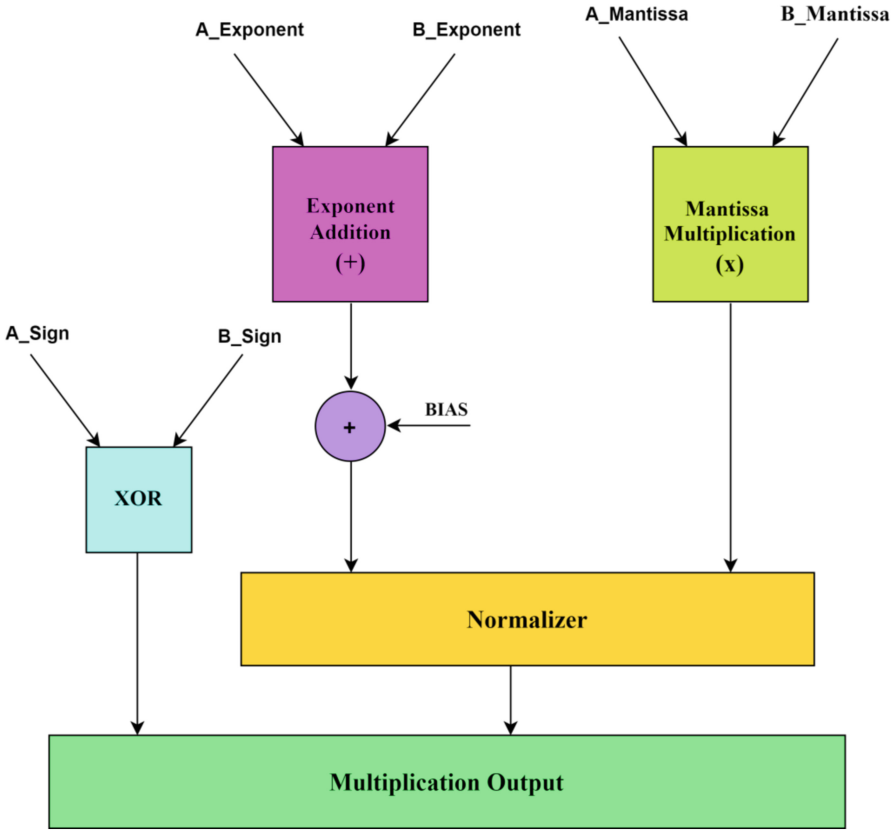


Fig.2. Block diagram of traditional floating point multiplier

and effectiveness of traditional multipliers in the dynamic landscape of digital electronics [22]. The block diagram of traditional floating point multiplier is shown Fig. 2.

3 Proposed Floating-Point Multiplier with Minimal Error Logarithmic Approach

The logarithmic floating-point multiplier introduces a novel approach to address limitations inherent in traditional floating-point multipliers, particularly concerning exponent and mantissa field sizes. By leveraging logarithmic representation, this proposed multiplier aims to optimize resource utilization, efficiently encoding and processing floating-point numbers with larger exponents and mantissas within constrained hardware resources. Specifically, in Ultra Scale + devices, such as LUTs, this approach demonstrates promise in enhancing computational efficiency while accommodating complex arithmetic operations necessary in various computational domains. The block diagram of proposed floating Point Multiplier with minimal error is shown Fig. 3.

In the IEEE 754 floating-point representation, the distribution of the sign bit, exponent, and mantissa plays a critical role in ensuring accurate representation and arithmetic operations for floating-point numbers. The sign bit, isolated from the mantissa, serves as a sentinel for determining the positivity or negativity of the number, preserving its overall numerical value. Meanwhile, the mantissa encapsulates significant digits and the fractional part, influencing precision and accuracy in arithmetic computations. This distinct segregation of components allows for a standardized representation of floating-point numbers, facilitating compatibility and consistency across different computational platforms, whether in single or double precision formats.

Sign Bit: The sign bit is a single bit that indicates the sign of the floating - point number, determining whether the number is positive or negative. In most of the floating-point representations, including IEEE 754, the sign bit is typically the leftmost bit

Mantissa: The Mantissa represents the significant digits of the floating - point number, including the fractional part. It determines the precision or accuracy of the number. The sign bit isn't part of the mantissa.

Exponent: The exponent represents the scale or magnitude of the floating - point number. It indicates how many positions the decimal point should be shifted to obtain the actual value.

This Verilog project presents the development of a floating-point multiplier, comprising modules for logarithmic computation, exponent addition, and antilogarithmic conversion. The primary module, 'floating point multiplier,' takes two 8-bit binary numbers (a and b), transforms them into logarithmic format, combines their exponents, multiplies their mantissas, and merges the outcomes to yield a 12-bit product. Specialized files, including Log calculator and Antilog calculator, facilitate the conversion between binary, logarithmic, and antilogarithmic representations. Additionally, a floating Addition module handles exponent addition for accurate logarithmic multiplication results. This comprehensive approach leverages logarithmic arithmetic to enhance precision and efficiency, underscoring the significance of logarithmic representations in advancing digital system computational capabilities.

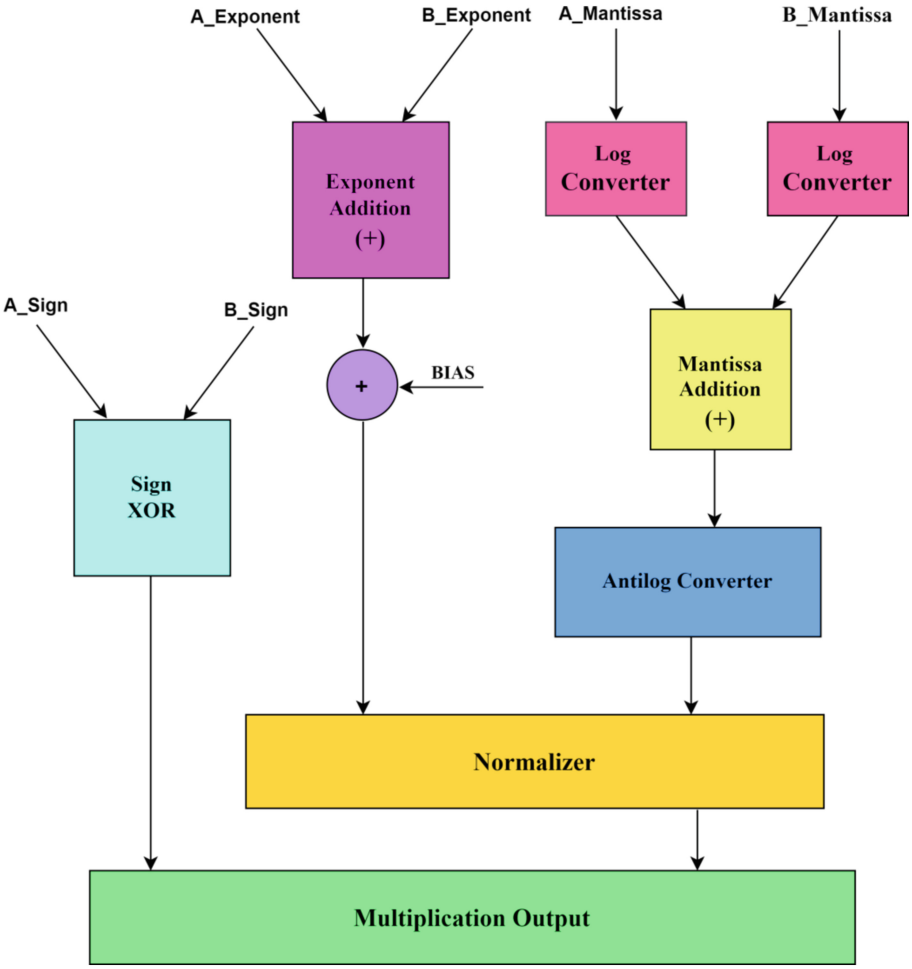


Fig.3. Block diagram of Proposed floating Point Multiplier with minimal error

Minimal error logarithmic approach stems from the need to reconcile the efficiency of logarithmic representations with the demand for utmost precision. To ensure minimal error in floating-point multiplication output, it is recommended that the logarithm output consists of at least nine significant digits. This precision allows the anti-log converter to produce an output with minimal or no error. By ensuring a sufficient number of significant digits in the logarithm output, the anti-log converter can accurately reverse the logarithmic transformation, thus minimizing errors in the multiplication process.

4 Results and Discussion

4.1 Traditional Floating Point Multiplier

Figure 4 and Fig. 5 shows the schematic symbol and RTL schematic of Traditional Floating Point Multiplier respectively.

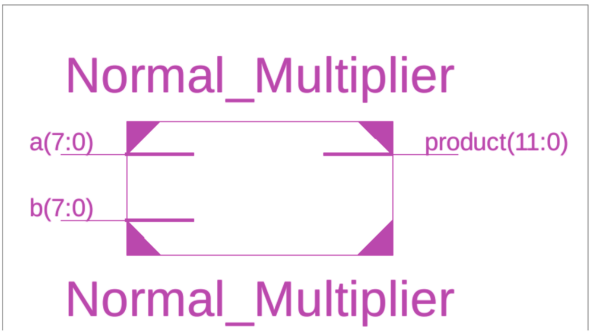


Fig.4. Schematic symbol of traditional floating point multiplier

Traditional floating point multiplier consists of the normal and combinational multipliers in RTL schematic view.

Normal Multiplier: This is a circuit that multiplies two numbers using a series of adders and shifters. The multiplicand is shifted to the left by one position for each bit of the multiplier (the number doing the multiplying) that is a 1. The shifted multiplicand is then added to the product, which is initially 0. This process is repeated for each bit of the multiplier.

Combinational Multiplier: This is a circuit that multiplies two numbers using a lookup table contains the pre-computed products of all possible pairs of input bits. The multiplier and multiplicand are used as indices into the lookup table, and the output of the lookup table is the product.

The diagram shows the simplified version of these two types of multipliers. The normal multiplier is on the left, and the combinational multiplier is on the right. The text at the top of the diagram indicates that the normal multiplier is part of a larger circuit called a “Madden result exponent1”. The text at the bottom of the diagram indicates that the combinational multiplier is part of a larger circuit called a“Mmult_result_mantissa_mult00011”.

The Fig. 6 shows the logic diagram of a 4-bit multiplier circuit. This uses an LPM_MUX21 component to multiplex the multiplicand, and LPM_XOR2 component to generate the partial products. The circuit also includes an adder tree to sum the partial products. The output of the multiplier is a 4-bit product.

COMMULTIPLIER1: This is the main module representing the entire 4-bit multiplier circuit. It might contain all the other blocks mentioned and handle their interconnections.

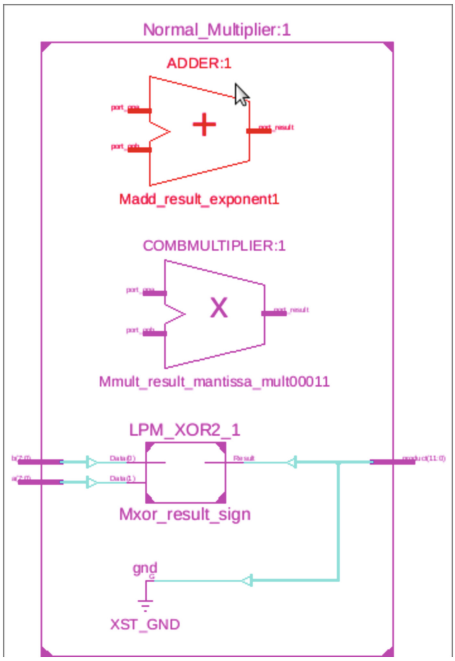


Fig.5. RTL schematic of traditional floating point multiplier

LPM_XOR2_1: This refers to an Intel "Logic Primitive Macros" component, specifically an XOR gate with two inputs. In a multiplier, it probably generates partial products based on individual multiplicand and multiplier bits.

Or2: This simply suggests a 2-input OR gate used for some logical operation within the circuit.

Result and0001 Imp result_and00011: These labels might represent intermediate results or specific stages of the calculation, possibly related to AND operations and their outcomes.

Mxor result sign: This could be a block of handling the sign bit of the final product calculated from the multiplied bits.

XST GND: This indicates the ground connection for the circuit, essential for proper power supply.

The Fig. 7 representing the simulation results for a normal multiplier circuit, providing insights into its performance over a specific time range. The text on the image explicitly mentions "Simulation Results of Normal Multiplier" and includes numerical values associated with the simulation. To understand the context, it′s crucial to note that a multiplier circuit is designed to multiply two binary numbers, generating the product as its output. The intricacies of this circuit′s design and implementation can vary, influenced by factors such as data width and the chosen multiplier algorithm. The data width represents the number of bits processed simultaneously, with examples including 4-bit,

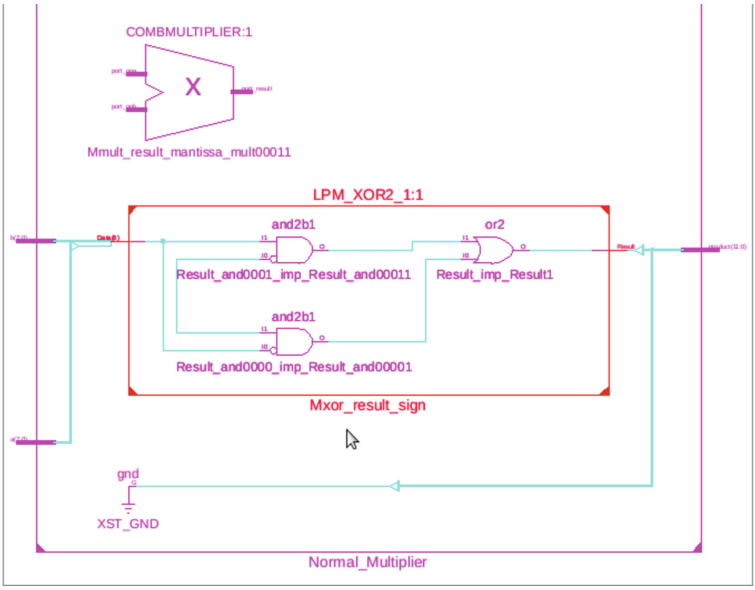


Fig.6. Logic diagram of traditional floating point multiplier

8-bit, or 16-bit multipliers. This parameter significantly impacts the circuit′s capabilities and the range of numbers it can efficiently handle. Equally significant is the multiplier algorithm employed in the circuit. Various methods exist for performing multiplication, with booth encoding and array multiplier being two examples. The selection of a specific algorithm can influence the efficiency, speed, and resource utilization of the multiplier circuit. Finally, the provided information delves into the simulation results of a normal multiplier circuit, emphasizing the importance of considering factors like data width and multiplier algorithm in the design and implementation of such circuits for binary number multiplication. Technological Schematic of traditional floating point multiplier is shown in Fig. 8.

The Fig. 9 shows the design summary of the normal multiplier project, developed using Xilinx ISE software, encompasses various crucial aspects. The project file, named "Project.xise," reports no parser errors. The implemented module, labeled "Normal_Multiplier," is currently in the synthesized state on the specific FPGA chip xc3s500e-5fg320. The project employs Xilinx ISE software version 14.7, with an overall design goal defined as "Balanced" and a design strategy chosen as "Xilinx Default" within the System Settings environment. In terms of device utilization, the FPGA chip shows minimal logic resource usage, with 19 out of 4656 available slices (0% usage) and 36 out of 9312 available 4-input LUTs (0% usage). This comprehensive summary provides valuable insights into the project′s files, status, module implementation, FPGA chip details, and resource utilization.

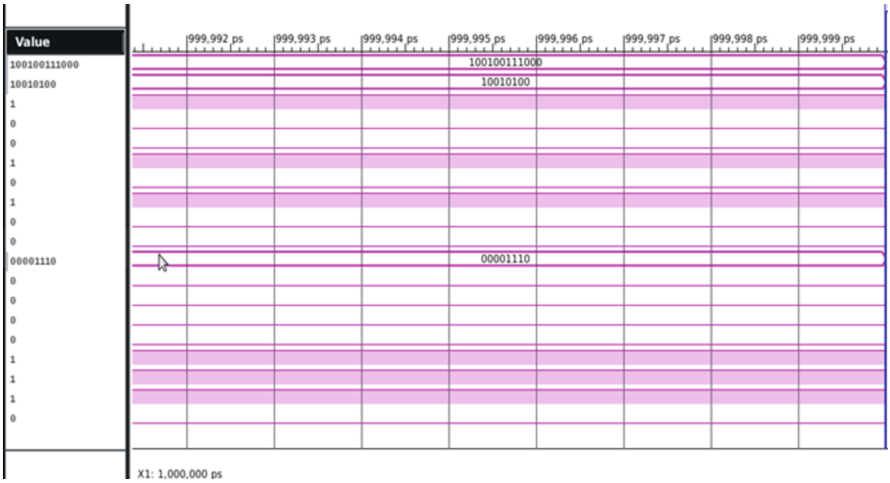


Fig.7. Simulation waveforms of traditional floating point multiplier

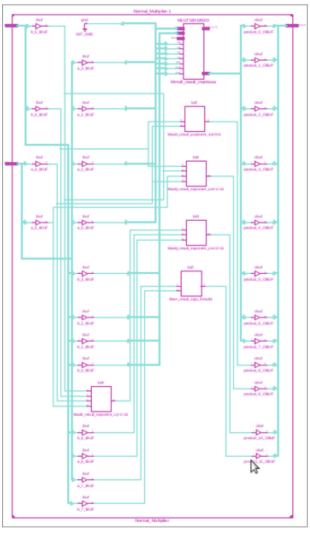


Fig.8. Technological Schematic of traditional floating point multiplier

4.2 Proposed Floating Point Multiplier with Minimal Error

The Fig. 10 shows that The floating-point multiplier design involves specific inputs and outputs, contributing to its functionality. The inputs, denoted as a(7:0) and b(7:0), are 8-bit representations of the first and second floating-point numbers, respectively. Both inputs likely follow a bit distribution where bits 0 to 7 represent the significand (mantissa), and bit 7 indicates the sign. The outputs, represented by product(11:0), constitute a 12-bit result of the multiplication operation. Here, bit 11 denotes the sign, and bits 0 to

Normal_Multiplier Project Status			
Project File:	Project.xise	**Parser Errors:**	No Errors
Module Name:	Normal_Multiplier	**Implementation State:**	Synthesized
Target Device:	xc3s500e-5fg320	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	18 Warnings (18 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	19	4656	0%
Number of 4 input LUTs	36	9312	0%
Number of bonded IOBs	101	232	43%
Number of MULT18X18SIOs	3	20	15%

Fig.9. Design Summary of traditional floating point multiplier

10 encapsulate the significand. The internal structure includes a crucial block named Floating_point_multiplier, responsible for executing the floating-point multiplication operation. This block is expected to handle distinct processes for managing the signs and significands of the input floating-point numbers, ensuring accurate and comprehensive multiplication results. Figure 10 represents the schematic symbol of proposed floating point multiplier.

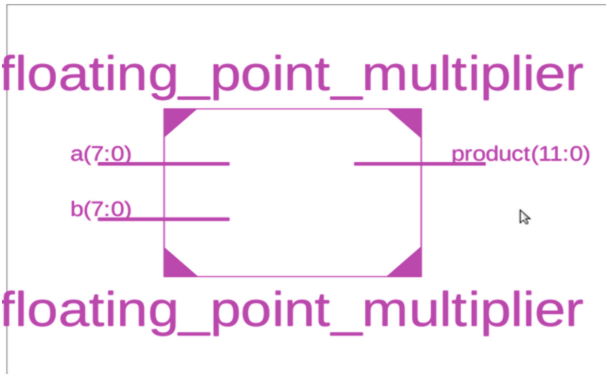


Fig.10. Schematic symbol of proposed floating point multiplier with minimal error

The Fig. 11 shows the floating-point multiplier circuit comprises several key components contributing to its intricate functionality. Multiple multiplier blocks, such as "Mmut ream11" and "Mxor product," are apparent, likely responsible for executing partial product generation for different bits of the input numbers. These blocks play a crucial role in the multiplication process. Additionally, there is an adder tree, consisting of interconnected adder blocks arranged in a tree-like structure. This configuration suggests its role in accumulating the partial products generated by the multipliers, contributing to the final output. Control logic is also integral to the design, with logic gates like XOR gates and multiplexers (denoted by "21:1" label) managing control signals and selecting specific data paths within the circuit. Furthermore, the presence of various labeled

blocks such as “LPM_XOR2 111,” “and 21:1,” “or2,” and “Mxor result sign” indicates the involvement of other blocks, although their specific functions remain unclear without additional context. Together, these components work in tandem to execute the floating-point multiplication operation with precision.

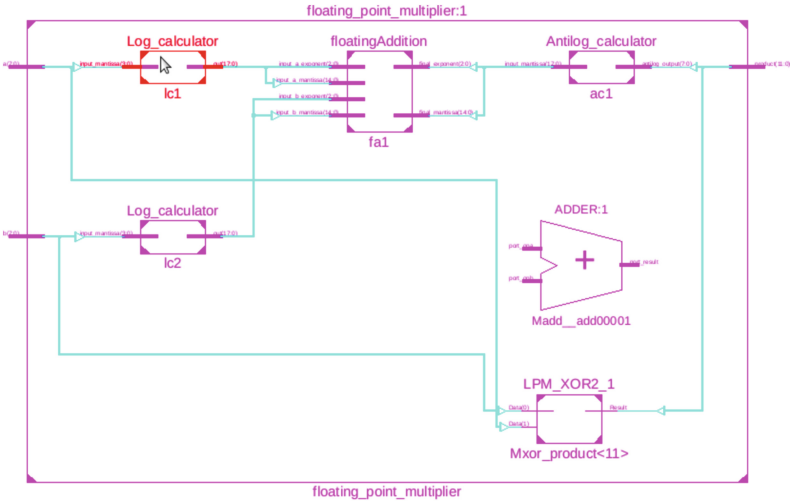


Fig.11. RTL schematic of proposed floating point multiplier with minimal error

Figure 12 and Fig. 13 represent logic diagrams of log calculator and Floating point Addition and Technological schematic of proposed floating point multiplier with minimal error is shown in Fig. 14.

The simulation result of the proposed floating point multiplier with minimal error is shown in Fig. 15 provides insight into its dynamics through various components. The leftmost column showcases two input values, a and b, presented in hexadecimal format (e.g., 0x3F800000 and 0x3F000000), likely encoding floating-point numbers adhering to standards like IEEE 754. The simulation time, ranging from 1999.992 ps to 2000.001 ps, is depicted in the top row. Internal signals are displayed in several columns, although their specific names and functions remain unclear due to the limited image resolution. Potential signal names include “product,” representing intermediate or final multiplication results, and “a_reg” and “b_reg,” which could signify registers storing input values during simulation. Additionally, signals like “exp_out” and “sig_out” may suggest the extraction of the exponent and significand from the input values. Despite the lack of clarity in signal names, these internal components contribute to the intricate processes within the multiplier circuit during simulation. Simulation waveforms of proposed floating point multiplier.

As represent in design summary in Fig. 16, the design uses 3 out of 4656 slices, 5 out of 9312 4-input LUTs, 28 out of 232 bonded IOBs, and 1 out of 20 MULT18X18SIOS.The project status shows that there are no errors, the implementation state is synthesized, and the routing results are successful. The device utilization is 0% for slices and 4-input LUTs, 12% for bonded IOBs, and 5% for MULT18X18SIOS.

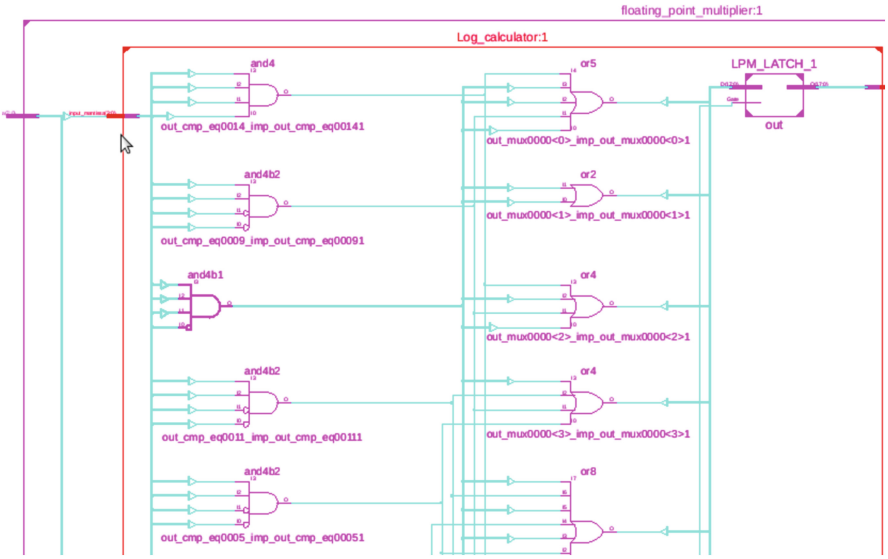


Fig.12. Logic diagram of Log Calculator

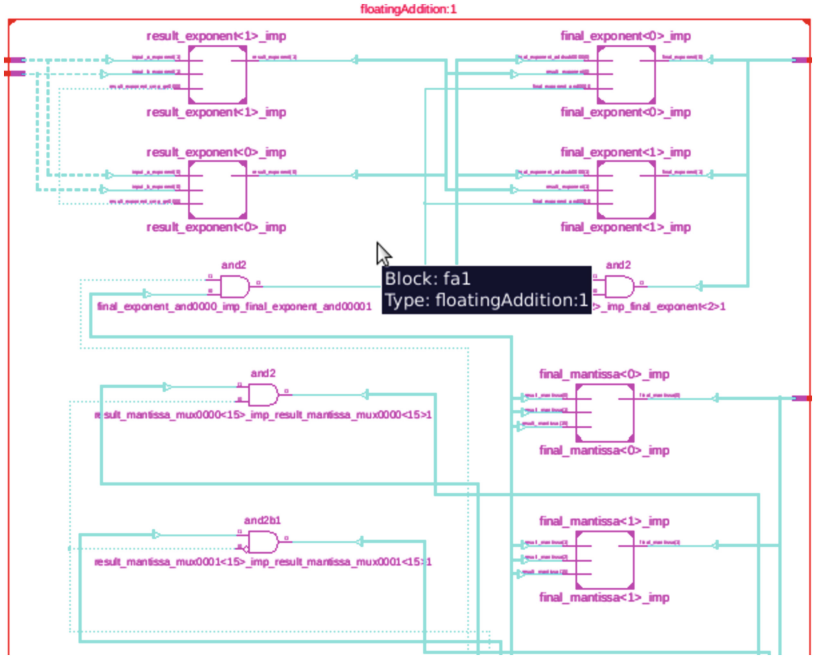


Fig.13. Logic diagram Floating point Addition

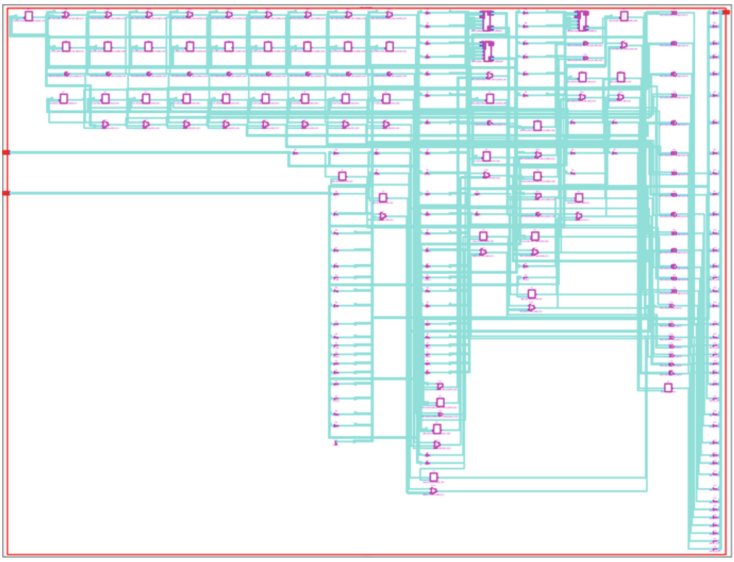
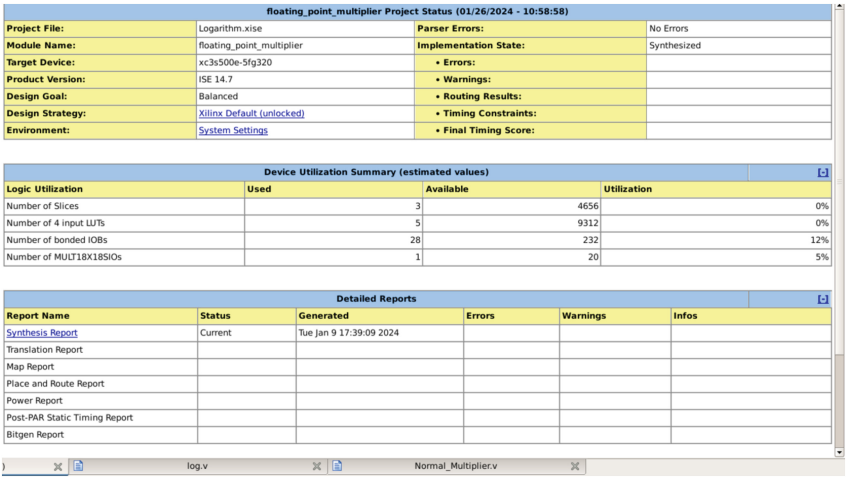


Fig.14. Technological schematic of proposed floating point multiplier with minimal error

Fig.15. Simulation waveforms of proposed floating point multiplier with minimal error



floating_point_multiplier Project Status (01/26/2024 - 10:58:58)			
Project File:	Logarithm.xise	**Parser Errors:**	No Errors
Module Name:	floating_point_multiplier	**Implementation State:**	Synthesized
Target Device:	xc3s500e-5fg320	**• Errors:**	
Product Version:	ISE 14.7	**• Warnings:**	
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3	4656	0%
Number of 4 input LUTs	5	9312	0%
Number of bonded IOBs	28	232	12%
Number of MULT18X18SIOs	1	20	5%

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Tue Jan 9 17:39:09 2024			
Translation Report					
Map Report					
Place and Route Report					
Power Report					
Post-PAR Static Timing Report					
Bitgen Report					

Fig.16. Design Summary of proposed floating point multiplier with minimal error

5 Conclusion

In this paper implemented a floating-point multiplier using Verilog, comprising modules for log conversion, log addition, antilog conversion, and exponent addition. The design leverages fixed-point representations for logarithmic and antilogarithmic calculations, providing a flexible and efficient approach to floating-point multiplication. The Log calculator file converts 4-bit mantissas to their corresponding 18-bit logarithmic representations, and the Antilog calculator file performs the reverse operation. These modules serve as lookup tables, enabling quick and accurate conversion between linear and logarithmic scales. The floating point multiplier file combines log conversion, log addition, antilog conversion, and exponent addition to achieve the multiplication of two floating-point numbers. The provided test bench validates the functionality of the floating-point multiplier by applying stimulus to the inputs and monitoring the resulting product output. Simulation results can be analyzed to ensure correct operation across various input scenarios. Overall, this project showcases the successful implementation of a floating-point multiplier in Verilog, demonstrating its effectiveness in handling logarithmic conversions and providing a foundation for further exploration and integration into complex digital systems.

Appendix

Example 1: An example for the traditional Floating point Multiplier is given below. For instance, in single-precision IEEE 754 format:

Consider two numbers:

A = 5.25B = 2.5

A = 1.01001 x 2^2B = 1.01 x 2^1.

1. Multiplication Result: Result = (11000000101001000000000000000000) x (110000000 01000000000000000000000).

2. Sign bit (S): In this case, both A and B are positive, so the sign bit (S) is 0.
3. Exponent (E): Add the biased exponent to get the actual exponent.
- (129 + 128 = 257) (In binary: 100000001).
Now, the IEEE 754 representation is:
(-1)^0 x 1.01001 x 2^130.
So, the final representation of the multiplication result in IEEE 754 format is:1.01001 x 2^130.

Example 2: An example for the proposed floating-point multiplier is given below. For simplicity, choose a = -0.5 and b = 0.875.
a = 8′b10010100.
b = 8′b00001110;
1. Logarithmic Conversion:
- Logarithmic conversion of (a) results in 18′b010110010101110000.
- Logarithmic conversion of (b) results in 18′b010100101101110000.
2. Logarithmic Addition:
- Logarithmic addition is performed, resulting in a mantissa of 18′b010101111100101100 and an exponent of 3.
3. Antilogarithmic Conversion:
- The antilogarithmic conversion results in an output of 8′b00000010.
4. Exponent Addition:
- The exponent addition results in an exponent of 3.
5. Sign Bit Handling:
- Since both a is positive and b is negative, the sign bit for the final result is 1.
6. Final Result:
- The final product is 8′b10110010 with the determined sign bit.
Therefore, for the inputs a = -0.5 and b = 0.875, the output product is -0.4375.

References

1. Xiong, B., Fan, S., He, X., Xu, T., Chang, Y.: Small logarithmic floating-point multiplier based on FPGA and Its application on mobilenet. IEEE Trans. Circ. Syst. II: Express Briefs **69**(12), 5119–5123 (2022)
2. Prakash, M.D., Krsihna, B.V., Satyanarayana, B.V.V., Vignesh, N.A., Panigrahy, A.K., Ahmadsaidulu, S.: A study of an ultrasensitive label free silicon nanowire FET biosensor for cardiac troponin i detection. SILICON **14**, 5683–5690 (2022)
3. Meriga, C., Ponnuri, R.T., Satyanarayana, B.V.V., Gudivada, A.A.K., Panigrahy, A.K., Prakash, M.D.: A novel teeth junction less gate all around FET for improving electrical characteristics. SILICON **14**, 1979–1984 (2022)
4. Satyanarayana, B.V.V., Prakash, M D.: Design analysis of GOS-HEFET on lower Sub-threshold Swing SOI. Analog Integr. Circ. Sig. Process. **109**, 683–694 (2021)
5. Saadat, H., Bokhari, H., Parameswaran, S.: Minimally biased multipliers for approximate integer and floating-point multiplication. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2623–2635 (2018)
6. Prabakaran, B.S., et al.: DeMAS: an efficient design methodology for building approximate adders for FPGA-based systems. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 917–920. IEEE, (2018)

7. Hashemi, S., Bahar, R.I., Reda, S.: DRUM: a dynamic range unbiased multiplier for approximate applications. In: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 418–425. IEEE (2015)
8. Narayanamoorthy, S., Moghaddam, H.A., Liu, Z., Park, T., Kim, N.S.: Energy-efficient approximate multiplication for digital signal processing and classification applications. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **23**(6), 1180–1184 (2014)
9. Pandit, A.A., Reddy, C.A., Narayan, G.: Design and simulation of 16 × 16 bit iter-ative logarithmic multiplier for accurate results. In: 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 985–990. IEEE (2018)
10. Kanhe, A., Das, S.K., Singh, A.K.: Design and implementation of floating point multiplier based on vedic multiplication technique. In: 2012 International Conference on Communication, Information & Computing Technology (ICCICT), pp. 1–4. IEEE (2012)
11. Sunesh, N.V., Sathishkumar, P.: Design and implementation of fast floating point multiplier unit. In: 2015 International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA), pp. 1–5. IEEE (2015)
12. Fagin, B., Renard, C.: Field programmable gate arrays and floating point arithmetic. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **2**(3), 365–367 (1994)
13. Kong, M.Y., Langlois, J.P., Al-Khalili, D.: Efficient FPGA implementation of com-plex multipliers using the logarithmic number system. In: 2008 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3154–3157. IEEE (2008)
14. Cody, W.J.: IEEE standards 754 and 854 for floating-point arithmetic. IEEE Mag. MICRO 84–100. IEEE (1984)
15. Govindu, G., Zhuo, L., Choi, S. Prasanna, V.: Analysis of high-performance float-ing-point arithmetic on FPGAs. In: 18th International Parallel and Distributed Processing Symposium 2004 Proceedings (IPDPS '04), pp. 149–156. IEEE (2004)
16. Ykuntam, Y.D., Penumutchi, B., Gubbala, S.: Design of speed and area efficient non linear carry select adder (NLCSLA) architecture using XOR less ad-der module. In: Chakravarthy, V., Bhateja, V., Flores Fuentes, W., Anguera, J., Vasavi, K.P. (eds.) Advances in Signal Processing, Embedded Systems and IoT . Lecture Notes in Electrical Engineering, vol. 992, pp. 81–91. Springer, Singapore (2023)
17. Rao, M.V., Kumar, P.R., Balaji, T.: A high performance dual stage face detection algorithm implementation using FPGA chip and DSP processor. J. Inf. Syst. Telecommun. (JIST) **4**(40), 241 (2022)
18. Arvapally, S., Khan, M.A., Vemula, P.C., Sonnaila, P. and Chary, U.G.: FPGA imple-mentation of industry automated bottle counter. In: 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), vol. 2, pp. 461–465. IEEE (2017). https://doi.org/10.1007/978-981-19-8865-3_7
19. Prasad, D.D., Satyanarayana, B.V.V., Shaik, A.R., Indirapriyadarsini, K., Reddy, K.S.R., Hemalatha, M.: Full swing logic based full adder for low power applications. In: Pareek, P., Gupta, N., Reis, M.J.C.S. (eds.) Cognitive Computing and Cyber Physical Systems. IC4S 2023. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 537, pp 19–36. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-48891-7_2
20. Kumar, T.S., Rao, I.R.S.N., Vinod, Y.S., Harika, P., Satyanarayana, B.V.V., Pravin, A.: Area efficient and ultra low power full adder design based on GDI technique for computing systems. In: Pareek, P., Gupta, N., Reis, M.J.C.S. (eds.) Cognitive Computing and Cyber Physical Systems. IC4S 2023. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 537, pp. 63–75. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-48891-7_5

21. Satyanarayana, B.V.V., Lakshmi, B.K.S., Kumar, G.P., Srinivas, K.: LUT-based area-optimized accurate multiplier design for signal processing applications. In: Pareek, P., Gupta, N., Reis, M.J.C.S. (eds.) Cognitive Computing and Cyber Physical Systems. IC4S 2023. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 536, pp. 309–317. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-48888-7_26
22. Parvathi, M.: High-accurate, area-efficient approximate multiplier for error-toler-ant applications. In: Pant, M., Kumar Sharma, T., Arya, R., Sahana, B., Zolfagharinia, H. (eds.) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol. 1154, pp. 91–102. Springer, Singapore (2020)