JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES
www.journalimcms.org

ISSN (Online) : 2454 -7190 Vol.-21, No.-07, July (2026) pp 245-255 ISSN (Print) 0973-8975

DESIGN AND EVALUATION OF AN ITERATIVE APPROXIMATE FLOATING-POINT MULTIPLIER FOR IMAGE PROCESSING

D. V. N. Bharathi[1], Bala Sindhuri Kandula[2], Gangula Manikanta[3], B. Jagadeesh Babu[4], Subhashini Tata[5], Kosaraju Swathi[6]

[1, 2] Department of E.C.E, Center of Excellence in VLSI Design, S.R.K.R Engineering College(A)), India

[3]Department of E.C.E, D.N.R College of Engineering and Technology (A), India

[4]Department of E.C.E, Aditya University, Surampalem, India.

[5]Department of IoT (E.C.E), SR Gudlavalleru Engineering College, India.

[6]Department of E.C.E, Shri Vishnu Engineering College for Women, India.

Email: [1]bapu.bharathi@gmail.com, [2]k.b.sindhuri@gmail.com, [3]manikantagangula29@gmail.com, [4]jagadishec410@gmail.com, [5]subhashinitata19@gmail.com, [6]kswathiece@svecw.edu.in

Corresponding Author: **Bala Sindhuri Kandula**

https://doi.org/10.26782/jmcms.2026.07.00015

(Received: April 10, 2026; Revised: July 05, 2026; Accepted: July 16, 2026

Abstract

Multiplier efficiency is very significant in image processing and multimedia applications, which involve numerous floating-point multiplications. Although IEEE-754 multipliers are more precise, they also require more power, hardware, and time, which are not suitable for low-power and real-time systems. Simple control logic, partial-product encoding, and error correction are methods used in approximate floating-point multipliers to enhance efficiency. They consume less hardware and power, and still provide acceptable image quality compared to their more exact counterparts, the multipliers. Therefore, they can be used in image processing applications that require error tolerance.

Keywords: Iterative approximate floating-point multiplier (IAFPM), IEEE-754 floating-point arithmetic, image multiplication, partial product encoding, error-tolerant image processing, low-power VLSI design, multimedia image processing applications

I. Introduction

Approximate floating-point multipliers improve speed and reduce power by tolerating small computational errors. Instead of fully implementing IEEE-754 operations, they simplify mantissa multiplication while keeping sign and exponent calculations accurate to avoid major errors.

This reduces hardware cost and power while still producing good results for applications like image processing. Towhidy et al. [I] proposed an iterative approximate multiplier using partial-product encoding and steering logic to reduce power and hardware usage. They showed that exact precision is not always necessary for useful results. In many real-world applications, small numerical errors do not visibly affect output quality, especially in image and multimedia processing [II]. Thus, approximate computing is considered an efficient alternative to exact computation, offering simpler hardware and lower power consumption [III]. Tadigadapa and Jagadamba [IV] highlighted that reducing complexity in partial-product generation improves power efficiency. Edavoor et al. [V] introduced a design using hybrid Radix-4 and Radix-8 Booth encoding for image tasks, while Chakraborty et al. [VI] found no noticeable loss in image quality with approximate multipliers. Nesam et al. [VII] also showed that key image features are preserved even with reduced precision. A number of studies have examined ways to reduce energy consumption directly at the circuit level. Mondal et al. [VIII] pointed out that multipliers with about 8-bit precision can already provide meaningful energy savings, while the loss in accuracy stays within acceptable limits. In later work, clock-gated approximate multipliers were proposed, in which sections of the circuit are shut down when not actively involved in computation [IX]. Because of this limitation, approximate floating-point multipliers are commonly chosen for systems with tight energy constraints, such as wireless sensor networks and embedded platforms [XI].To improve efficiency in data-intensive workloads, Cheng et al. [XII] proposed logarithm-based approximate floating-point multipliers for neural network applications. The basic ideas behind such designs were supported by earlier work on radix-K approximate multipliers based on two-dimensional pseudo-Booth encoding [XII]. Further developments introduced configurable approximate floating-point multipliers, in which accuracy and energy consumption can be adjusted at runtime based on application needs [XIV]. Because of these advantages, approximate floating-point multipliers are now commonly used in neural networks, Internet-of-Things (IoT) devices, and other energy-limited applications, where saving power is often more important than achieving fully exact numerical results [XVII].

The key contribution of this work is the design of an iterative approximate floating-point multiplier for image processing. This design, in contrast to the more traditional IEEE-754 multipliers, which consume more power and hardware, adds controlled approximation in the mantissa multiplication to save power and area. It relies on partial-product encoding, simple control logic, and simple error correction so as to reduce unnecessary computations and still produce acceptable quality output. It provides trade-offs in power, area, delay, and image quality, which are better than the exact multipliers, and thus can be used in efficient image processing applications.

The paper is organized as follows: Section II analyzes image multiplication performance using exact and approximate floating-point multipliers. Section III explains the methodology and key algorithms. Section IV presents the flow diagram and overall system process. Section V describes the design of the iterative approximate multiplier. Section VI discusses results and performance, including comparison with the exact RCA architecture. Section VII concludes the paper with key findings.

II. Image multiplication performance

A. Exact Floating Point Multiplication

In floating-point multiplication, the IEEE-754 standard is implemented to operate the sign, exponent, and mantissa to obtain precise results [I], [II]. In precise multiplication, the sign of the output is determined by the XOR of the input signs, exponents are added with bias correction, and mantissas are multiplied with full partial-product generation [I], [III], [IV]. Multiplication should also be followed by normalization and rounding to ensure that the output fits within the IEEE-754 format, which consumes more time and resources [V], [VI]. Precise multipliers are based on accurate adders, compressors, and carry propagation logic, reducing power consumption and silicon area [III], [VII]. It also results in a longer critical path, thus less ideal in low-power and high-speed systems [VIII], [IX]. Nevertheless, precise multipliers are still employed as a guide for measuring rough designs [I], [X], [XI]. Although they do a great job in both image and signal processing, such high precision is usually not needed, as small errors are often tolerated [II], [XII], [XIII].

B. Approximate floating-point multiplication

Approximate multiplication with floating points makes hardware simpler, less power-consuming, and faster by compromising accuracy [I]. The main source of errors is in mantissa multiplication, but the sign and exponent do not change, which restricts the overall error [II]. These multipliers conserve a lot of power and space and deliver near-exact results [IV]. They retain high visual quality (measured by PSNR and SSIM) in image processing despite slight errors [V]. Therefore, they can be applied to low-power systems and fast systems [VI]. Delay and resource consumption are minimized by the use of 4:2 compressors as well as error-tolerant adders [VII]. These tiny inaccuracies are tolerable in many applications such as image and signal processing where high-quality accuracy is not a mandate [VIII].

III. METHODOLOGY (Algorithmic Steps with Calculations)

The mathematical computations have been highlighted with the main ones to demonstrate how the system functions and to demonstrate the difference in performance between the precise and approximate multipliers.

Algorithm 1: Overall Image Multiplication Methodology [II], [VI]

1. Read two input images and from memory, each of size $M \times N$
2. For each pixel location (i, j)

.

$$I_1(i,j) = [R_1(i,j), G_1(i,j), B_1(i,j)]$$
$$I_2(i,j) = [R_2(i,j), G_2(i,j), B_2(i,j)]$$

Convert each pixel component from integer to IEEE-754 single-precision floating-point format:

$$FP(R_k) = (-1)^S \times 1.M \times 2^{(E-127)}$$

where $k \in \{1,2\}$

3. Perform floating-point multiplication for corresponding RGB components:
4. Normalise the multiplication results according to IEEE-754 rules.

$$R_{out} = FP(R_1) \times FP(R_2)$$
$$G_{out} = FP(G_1) \times FP(G_2)$$
$$B_{out} = FP(B_1) \times FP(B_2)$$

5. Convert the floating-point outputs back to integer values:

$$I_{out}(i,j) = \text{int}(FP_{out})$$

6. Store the reconstructed pixel in the output image.
7. Repeat steps 2–7 for all pixels.

$$I_{out}(i,j)$$
$$(i,j) \in M \times N$$

Algorithm 2: Exact Floating-Point Multiplier [1], [10]

1. Accept two IEEE-754 single-precision floating-point inputs and. A
2. Decompose inputs into sign, exponent, and mantissa:

$A = (-1)^{S_A} \times 1.M_A \times 2^{(E_A - 127)}$

$B = (-1)^{S_B} \times 1.M_B \times 2^{(E_B - 127)}$

3. Compute output sign:

$S_P = S_A \oplus S_B$

4. Compute exponent:

$E_P = E_A + E_B - 127$

5. Perform full-precision mantissa multiplication:

$M_P = M_A \times M_B$

6. Normalize the mantissa:

○ If $M_P \geq 2$, right-shift mantissa and increment exponent. ○
Else retain mantissa as it is.

7. Combine sign, exponent, and mantissa to form exact output:

$P = (-1)^{S_P} \times 1.M_P \times 2^{E_P}$

Algorithm 3: Iterative Approximate Floating -Point Multiplier [1], [2], [13]

1. Accept two IEEE-754 floating-point inputs A and B.
2. Extract sign, exponent, and mantissa fields.
3. Compute output sign:

$S_P = S_A \oplus S_B$

4. Compute the exponent approximately:

$E_P = E_A + E_B - 127$

5. Encode mantissas using Partial-Product-Based (PB) Radix-8 encoding:

$M \in \{1,1.5,2\}$

6. Generate approximate partial products:

$PP \approx M_A \times M_B$

7. Activate steering logic only when $\times 1.5$ operation is detected to reduce hardware switching. Compute the correction term C^(i) and update the accumulated mantissa according to P^(i+1)=P^(i)+C^(i), where P^(i) represents the approximate mantissa before correction.
8. Since only one correction stage is employed, the updated mantissa P^(i+1)is used for final normalization.
9. Perform error correction using lower mantissa bits:

$$error = f(M_A^{low}, M_B^{low})$$

10. Normalise approximate mantissa and adjust exponent.
11. Generate final approximate floating-point output:

$$P_{approx} = (-1)^{S_P} \times 1.M_{approx} \times 2^{E_P}$$

Theoretical Analysis of Iterative Correction

The theoretical analysis of the proposed iterative correction mechanism is based on the iterative mantissa update equation, residual error formulation, finite convergence with a single correction stage (Nmaxf_0=1)(N_{\max}=1)(Nmax=1), and the corresponding cumulative approximation error bound. The detailed mathematical expressions are presented in Section V.

Algorithm 4: Image Output Generation [VI], [XI]

1. Convert floating-point results to integer pixel values:

$$Pixel = \min(\max(FP_{out}, 0), 255)$$

2. Assign results to RGB channels:

$$I_{out}(i,j) = [R_{out}, G_{out}, B_{out}]$$

3. Generate synchronization signals (HSYNC, VSYNC) for display timing.
4. Output the processed image frame.
5. Terminate the process after all pixels are processed [VI], [XI]

IV. Flow Diagram of Image Multiplication Implementation

Fig. 1 shows the general scheme of the image multiplication system. Two input images (Image A and Image B) are initially processed in MATLAB and transformed to floating-point and HEX. Multiplication is done pixel-by-pixel with an iterative (approximately floating-point) multiplier, which minimizes hardware requirements and power use. The output is in HEX format and then converted to an image after calculating. The conclusion of the final result is that approximate floating-point multiplication can preserve almost the same visual quality, but with higher efficiency, so that it can be used in image processing.

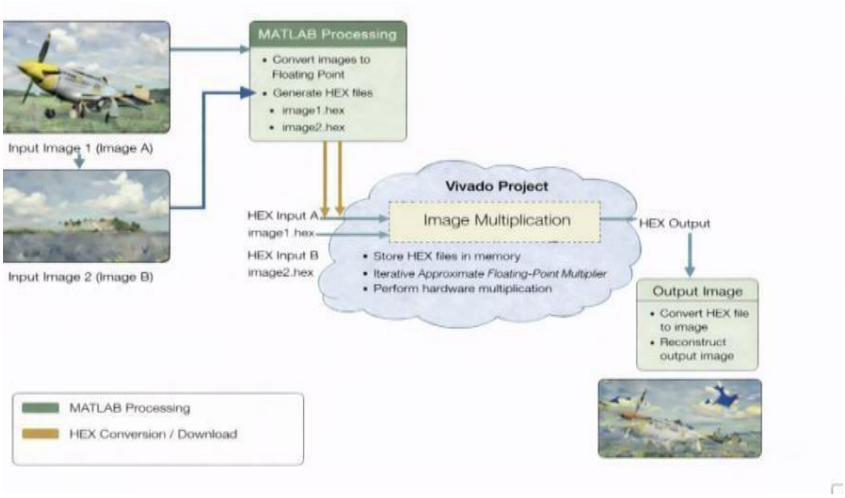


Fig. 1. Flow Diagram of Image Multiplication Implementation

II. Iterative Approximate Floating Point Multiplier

The approximate floating-point multiplier algorithm is given in Algorithm5 [I]. The design is intended to be less complex in terms of hardware and less power-consuming, yet accurate enough to handle practical applications, such as image processing and multimedia systems.

Algorithm5

Step 1: Input Floating-Point Numbers
Two floating-point numbers A and B are given as inputs.
Each floating-point number has three parts:
Sign
Exponent
Mantissa
These three parts are processed separately inside the system.
Step 2: Sign Calculation
The sign bits of A and B are combined using an XOR
gate. If both signs are the same → output is positive.
If signs are different → output is negative
This part is exact, meaning no approximation is used here.
Step 3: Exponent Calculation
The exponents of A and B are added together using an adder.
Bias is handled according to IEEE-754 rules.
This part is also exact to avoid large numerical errors.
Step 4: Mantissa Processing (Main Approximation Area)
The mantissa multiplication is the most complex and hardware-heavy operation. So, approximation techniques are applied only here.
The mantissas of A and B are encoded using Pseudo-Booth encoding.
Partial products are generated instead of full multiplication.

A steering circuit selects only the important partial products.

This reduces the number of operations and saves hardware resources.

Step 5: Approximation and Error Control

To make the design faster and smaller:

Less important partial products are ignored or simplified.

This introduces a small error, but it is acceptable for applications like image multiplication.

Step 6: Correction Unit

Since approximation introduces residual mantissa error, the proposed architecture employs a single correction stage to improve numerical accuracy. The accumulated mantissa is updated using $P^{(i+1)}=P^{(i)}+C^{(i)}$, while the residual error is reduced according to $E^{(i+1)}=E^{(i)}-C^{(i)}$. As only one correction stage is implemented, the iterative process converges after a single iteration ($N_{max}=1$). Since the proposed architecture contains only one hardware correction stage, the maximum number of correction iterations is fixed at $N_{max}=1$. Therefore, the correction process always terminates after a single update, guaranteeing finite convergence with deterministic execution time. The residual approximation error after each correction stage is expressed as $E^{(i+1)}=E^{(i)}-C^{(i)}$, where $E^{(i)}$ denotes the residual error after the i^{th} iteration and $C^{(i)}$ is the applied correction term. Assuming $|C^{(i)}|\leq|E^{(i)}|$, the residual error satisfies $|E^{(i+1)}|\leq|E^{(i)}|$. Since only one correction stage is implemented, $|E_{final}|\leq|E_0|$, demonstrating that the cumulative approximation error remains bounded while maintaining low hardware complexity. Thus, the residual approximation error decreases monotonically after the correction stage and remains bounded by the initial approximation error, ensuring stable numerical behavior of the proposed multiplier.

Step 7: Final Normalisation and Output

Finally:

The mantissa is normalised.

The exponent is adjusted if needed.

The sign, exponent, and mantissa are combined to form the final output.

The output is a floating-point number close to the exact result, but achieved with less power and area as an approximation is used.

VI. Results and Discussion

The iterative approximate multipliers were synthesized using the Cadence Genus Synthesis Solution targeted to the GPDK045 45 nm standard cell library under fast operating conditions. All iterative approximate multipliers were synthesized under identical pipeline depths and uniform optimization constraints. Table 1 is a comparison of power, area, and delay between the actual and approximate floating-point multipliers using Cadence results. The approximate multiplier has great gains in area and delay, being smaller and quicker than the exact RCA-based iterative floating-point multiplier design. It also uses less power since it is less complex. Even though it is associated with some minor errors, they are not significant in applications such as image processing. Generally, the rough multiplier proves to be more resource efficient and faster, hence a viable option in low-power and real-time applications.

Table1: Cadence Results: Performance Comparison of Exact and Approximate RCA-Based Iterative Floating-Point Multipliers

Metric	Approximate	Exact
Power(w)	6.8379*10^-4	1.17484*10^-3
Area(μm^2)	3534.581	5588.612
Delay (ns)	1.455	1.999
PDP(pJ)	0.995	2.35
ADP(μm^2.ns)	5141.8	11171.6

C. Image Multiplication Results

This section compares image multiplication with various floating-point multiplier designs. Images (a) and (b) are input and processed by (c) an approximate multiplier without RCA, (d) an approximate multiplier with RCA, and (e) the proposed approximate multiplier. At the pixel level, the output of precise multipliers with and without RCA is the same, and there is no error. The MSE is equal to zero, so SNR and PSNR are infinite, which means that there is no noise or distortion, and the system is functioning properly. Comparison of multipliers has yielded better results.

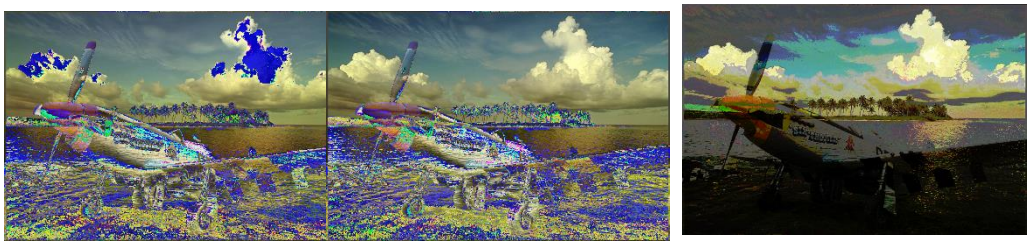
(a) Image 1

(b)Image 2

Table 2: Performance Comparison of Approximate RCA-Based Iterative Floating-Point Multipliers for Image Multiplication

Metric	Approximate multiplier using Exact RCA vs Approximate RCA	Approximate multiplier using Exact (with RCA) vs. Exact (without RCA)
Total Pixels	1,179,648	1,179,648
Bit Errors	1,126,699	0
Bit Error Rate (BER)	0.955115	0
Error Rate (ER)	0.955115	0
Pass Rate (PR)	0.044885	1.00
Mean Squared Error (MSE)	3354.59	0
Signal-to-noise Ratio(SNR) (dB)	7.13	∞
Peak signal-to-noise ratio(PSNR)(dB)	12.87	∞

Structural Similarity Index (SSIM)	0.701740	1.00
Error Distance (ED)	28,348,499.00	0
Mean Error Distance (MED)	25.16	NaN
Average Relative Error Distance(ARED)	2.18132728	0.00000000
Normalized Mean Error Distance (NMED)	0.09424047	0.00000000
Worst Case Error (WCE)	255.00	0.00
Maximum ULP Error	1132396544	0
Error Probability	0.95511458	0.00000000



(c) (d) (e)

Where (c) Approximate multiplier without RCA using Exact Adder, (d)Approximate multiplier with RCA using Exact Adder, (e) Approximate multiplier using Approximate Adder

VII. Conclusion

The current paper is aimed at the development and testing of an iterative approximate floating-point multiplier in image processing with the goals of minimizing power and cost without compromising on the useful output quality. Image processing can withstand small errors; hence, approximation is feasible. The design employs partial-product encoding, logic steering, and a simple correction unit to minimize complexity with little effect on accuracy. Cadence tool results indicate lower power, area, and delay than precise multipliers. Generally, the suggested multiplier is a good tradeoff between accuracy and efficiency, and can be used in low-power, real-time, and error-tolerant image processing. It reduces hardware requirements significantly and is efficient in embedded systems. This method also enhances faster computation as it is less complex. It shows that a little loss in accuracy does not affect visual perception in images. This approach can be generalized to other signal processing and multimedia applications. This work is limited to synthesis; post-layout validation will be addressed in future work.

Conflict of Interest :

There is no conflict of interest regarding this paper.

References

I. Towhidy, A., R. Omidi, and K. Mohammadi. "On the Design of Iterative Approximate Floating-Point Multipliers." *IEEE Transactions on Computers*, vol. 72, no. 6, June 2023, pp. 1623–1635. 10.1109/TC.2022.3216465

II. Edavoor, P. J., 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. 2024, article 100546, 2024. 10.1016/j.prime.2024.100546

III. Tadigadapa, P., and P. Jagadamba. "Design of Power Efficient Approximate Multiplier." *Procedia Computer Science*, vol. 260, 2025, pp. 784–795. 10.1016/j.procs.2025.03.259

IV. Mondal, S., M. R., and R. S. "Approximate 8-Bit Multipliers and Their Physical Design Implementation." *e-Prime – Advances in Electrical Engineering, Electronics and Energy*, vol. 6, no. 3, 2023, article 100300. 10.1016/j.prime.2023.100300

V. Chowdam, V. S., S. B. Potladurty, and P. R. Karipireddy. "Design and Evaluation of Clock-Gating-Based Approximate Multiplier for Error-Tolerant Applications." *Memories – Materials, Devices, Circuits and Systems*, vol. 9, no. 2, 2025. 10.1016/j.memori.2025.100123

VI. Chakraborty, A., V. P. A. Kumar, A. K. Vruddhula, J. N. K., and S. Balamurugan. "Energy Efficient Approximate Multiplier for Image Processing Applications." *Results in Engineering*, vol. 25, 2024, article 103798. 10.1016/j.rineng.2024.103798

VII. Nesam, J. J. J., S. S. Ganesh, and S. Ramachandran. "Effect of Bit-Size Reduced Half-Precision Floating-Point Format on Image Pixel Characterization for AI Applications." *Results in Engineering*, vol. 2024, 2024, article 103179. 10.1016/j.rineng.2024.103179

VIII. Rott, O., and E. Jarlebring. "An Iterative Method for the Multipliers of Periodic Delay-Differential Equations and the Analysis of a PDE Milling Model." *WIAS Preprint*, no. 1470, 2009. 10.20347/WIAS.PREPRINT.1470.

IX. Yu, S., F. Xia, R. Shafik, D. Balsamo, and A. Yakovlev. "Approximate Digital-in Analog-out Multiplier with Asymmetric Nonvolatility and Low Energy Consumption." *Integration, the VLSI Journal*, vol. 93, 2023. 10.1016/j.vlsi.2023.05.009

X. Brogi, F., S. Bna, G. Boga, G. Amati, T. E. Ongaro, and M. Cerminara. "On Floating Point Precision in Computational Fluid Dynamics Using OpenFOAM." *Future Generation Computer Systems*, vol. 152, 2024, pp. 1–16. 10.1016/j.future.2023.10.006

XI. Kumar J., C. R., D. V. Kumar, and M. A. Majid. "High-Performance, Energy-Efficient, and Memory-Efficient FIR Filter Architecture Utilizing 8×8 Approximate Multipliers for Wireless Sensor Network in the Internet of Things." *Memories – Materials, Devices, Circuits and Systems*, vol. 3, 2022, article 100017. 10.1016/j.memori.2022.100017

XII. Cheng, T.-Y., Y. Masuda, J. Chen, J. Yu, and M. Hashimoto. "Logarithm-Approximate Floating-Point Multiplier Is Applicable to Power-Efficient Neural Network Training." *Integration, the VLSI Journal*, vol. 74, 2020, pp. 19–31. 10.1016/j.vlsi.2020.05.002

XIII. Towhidy, A., R. Omidi, and K. Mohammadi. "On the Design of Radix-K Approximate Multiplier Using 2D Pseudo-Booth Encoding." *AEÜ – International Journal of Electronics and Communications*, vol. 142, 2021, article 153988. 10.1016/j.aeue.2021.153988

XIV. Imani, M., R. Garcia, S. Gupta, and T. Rosing. "RMAC: Runtime Configurable Floating-Point Multiplier for Approximate Computing." *CSE Department, University of California, San Diego*, 2022. 10.1145/3218603.3213621

XV. Cheng, T.-Y., J. Yu, and M. Hashimoto. "Minimising Power for Neural Network Training with Logarithm-Approximate Floating-Point Multiplier." *Department of Information Systems Engineering, Osaka University*, 2020. 10.1109/PATMOS.2019.8862162.

XVI. Yan, Meng, et al. "kNN-CAM: A k-Nearest Neighbors-Based Configurable Approximate Floating-Point Multiplier." *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED)*, 2019. IEEE. 10.1109/ISQED.2019.8697584.

XVII. Saadat, H., H. Bokhari, and S. Parameswaran. "Minimally Biased Multipliers for Approximate Integer and Floating-Point Multiplication." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 1, 2021, pp. 210–223. IEEE, 10.1109/TCAD.2018.2857262.