International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 24s, 2025
https://theaspd.com/index.php

A Fast and Energy-Efficient Rounding-Based Multiplier Architecture (RoBA) for DSP Applications

Akkali Sowmya[1], D Satheesh[2]
[1]PG Scholar, Sree Dattha Institute of Engineering and Science, akkalisowmyareddy@gmail.com
[2]Assistant professor, Sree Dattha Institute of Engineering and Science, dhudhigamsatheesh@gmail.com

***Abstract*—** *This research presents a novel rounding-based approximate multiplier architecture (RoBA) that achieves a compelling trade-off between computational accuracy, high-speed performance, and energy efficiency, making it well-suited for modern digital signal processing (DSP) and image processing applications. The key innovation lies in the strategic rounding of operands to the nearest power of two, which simplifies multiplication by reducing it to a sequence of shift and add operations. This approach effectively bypasses the most computation-intensive stages of conventional multipliers, leading to reduced propagation delay, lower dynamic power consumption, and minimized hardware complexity.*

Despite introducing a bounded approximation error, the proposed design demonstrates significant improvements in throughput and energy savings, particularly under tight resource constraints. The architecture supports both signed and unsigned multiplication, and is realized in three hardware configurations: one for unsigned operations and two for signed operations, with one configuration specifically optimized for enhanced accuracy in negative value computations.

Comprehensive evaluations are performed using industry-standard synthesis tools and benchmark circuits to assess key performance metrics, including area utilization, power dissipation, critical path delay, and error rate. To validate real-world applicability, the proposed multiplier is deployed in two representative image processing pipelines—image smoothing (e.g., Gaussian blur, mean filtering) and image sharpening (e.g., unsharp masking)—which heavily rely on multiply-accumulate operations. Simulation results confirm that the approximate multiplier maintains acceptable visual fidelity while significantly reducing computational and energy overhead.

When compared with traditional accurate multipliers and state-of-the-art approximate designs, the proposed RoBA architecture consistently delivers superior results in terms of energy efficiency and speed. Its low hardware footprint, high throughput, and tolerance to minimal error make it an ideal candidate for low-power embedded systems, portable devices, and real-time DSP applications where energy efficiency is prioritized over exact arithmetic precision.

Keywords: *Approximate Multiplier, Rounding-Based Architecture, Low-Power Design, Digital Signal Processing, Image Processing*

INTRODUCTION

Energy minimization has emerged as a pivotal design criterion in nearly all modern electronic systems, particularly those targeting portable, wearable, and battery-operated platforms such as smartphones, tablets, smartwatches, and IoT-enabled devices. These systems require high computational throughput while conforming to stringent power and thermal budgets. In this context, achieving energy-efficient processing without sacrificing speed has become a cornerstone objective in embedded system and processor design [1].

Among embedded subsystems, Digital Signal Processing (DSP) units are heavily relied upon to execute compute-intensive tasks, including image filtering, audio and video decoding, sensor fusion, and feature extraction. These operations are generally dominated by multiply-accumulate (MAC) computations, placing arithmetic units—particularly multipliers—at the center of performance and power bottlenecks [2], [3]. Consequently, optimizing multipliers for speed and energy consumption has a direct and measurable impact on overall processor efficiency and battery longevity [4].

A significant observation in many DSP workloads, especially those targeting human-centric outputs such as images, speech, or video, is that perfect numerical precision is not always required. The bounded resolution of the human visual and auditory systems permits certain error margins without perceptual degradation [5]. This has given rise to approximate computing, a design paradigm that relaxes accuracy constraints to gain improvements in energy, performance, and area utilization [6], [7].

Approximate computing techniques can be applied at multiple levels of abstraction:

- Circuit-level, using voltage overscaling or probabilistic switching [8],
- Logic-level, through logic simplification and gate-level pruning [9],
- Architectural-level, using functionally approximate units and restructured datapaths [10], and
- Algorithmic-level, where inherently error-tolerant algorithms are designed to tolerate or even exploit approximation [11].

At the core of function-level approximations are approximate arithmetic units, such as adders and multipliers, which serve as reusable computational blocks for a variety of signal and data processing functions. Approximate multipliers, in particular, have garnered extensive attention due to their disproportionately high contribution to power and delay in DSP pipelines [12], [13].

Rationale for Rounding-Based Approximation

Among the various approximation strategies, rounding input operands to the nearest power of two presents a powerful technique to drastically simplify multiplication logic. This transforms multiplication into bitwise shift operations, which are inherently less complex and more energy-efficient than full-width binary multiplication [14]. Such rounding-based schemes offer a unique advantage: they reduce computational depth and propagation delay, eliminate unnecessary transitions in partial product generation, and result in lower switching activity, all of which contribute to reduced dynamic power [15].

Proposed Work: RoBA Multiplier

In light of the above, we propose a Rounding-Based Approximate (RoBA) Multiplier, which employs power-of-two rounding to reduce multiplication to shift-based operations, eliminating the need for traditional partial product accumulation. The RoBA scheme introduces algorithm-level approximation, offering a design-space trade-off between precision and performance.

The key contributions of this work include:

1. A novel rounding-based algorithm for approximate multiplication that inherently minimizes delay, dynamic power, and hardware complexity.
2. Three hardware-optimized architectures: one for unsigned data and two for signed data (including a sign-sensitive design for improved error handling).
3. Comprehensive evaluation of the proposed designs against state-of-the-art approximate and accurate multipliers using metrics such as:
   - Propagation delay
   - Dynamic and leakage power
   - Energy-delay product (EDP)
   - Silicon area and error rate

To further validate real-world applicability, the RoBA multiplier is integrated into DSP workloads such as image smoothing (Gaussian blur) and image sharpening (unsharp masking)—applications that are compute-intensive yet tolerate minor accuracy deviations. Results show that RoBA achieves significant reductions in energy and area with negligible impact on perceptual quality.

Conclusion

The proposed RoBA multiplier aligns well with the energy-performance-area constraints of modern embedded and DSP systems. It offers a compelling solution for applications where energy efficiency and throughput are critical, and minor arithmetic errors are acceptable. This work contributes to the expanding field of approximate computing and hardware optimization for error-resilient signal processing systems.

Multipliers play a critical role in high-performance digital systems, especially in computation-intensive applications such as finite impulse response (FIR) filters, microprocessors, digital signal processors (DSPs), and optical signal processing units. These systems rely heavily on efficient and accurate arithmetic operations to achieve optimal performance, and among these operations, multiplication stands out due to its computational intensity and frequent invocation in real-time signal and data processing tasks [16].

Despite its fundamental nature, multiplication is inherently complex, involving multiple arithmetic stages such as partial product generation, alignment, and summation. This complexity makes multiplication a key target for hardware optimization, particularly when seeking improvements in speed, energy efficiency, and silicon area reduction [17].

A major challenge in implementing multipliers arises when handling signed and unsigned numbers. While unsigned multiplication is relatively straightforward, signed multiplication demands special care due to the use of two's complement representation for negative numbers. If the two's complement format is not properly accounted for—especially when signed values are mistakenly treated as unsigned—the result can be erroneous, leading to critical failures in arithmetic computations [18].

This discrepancy necessitates the use of distinct logic paths or correction mechanisms for accurate signed and unsigned operations. Some architectures use dedicated signed multiplication circuits, while others implement unified structures with embedded logic to handle sign extension, bit inversion, and partial product corrections. These strategies ensure accuracy and reliability, particularly in DSP applications where both positive and negative values are prevalent [19].

At a conceptual level, multiplication is defined as repeated addition, where a value called the multiplicand is added to itself a number of times specified by the multiplier. This operation forms the mathematical basis for scalar scaling and is foundational across decimal, binary, and other number systems.

In traditional base-10 multiplication, commonly taught in educational contexts, each digit of the multiplier is used to scale the multiplicand, starting from the Least Significant Digit (LSD) to the Most Significant Digit (MSD). The resulting partial products are aligned based on place value and summed to obtain the final result. This approach, though straightforward, translates directly into binary multiplication, where only two digits—0 and 1—are involved [20].

In binary systems, each bit of the multiplier determines whether the multiplicand contributes to the partial product (if the bit is 1) or is ignored (if the bit is 0). Each valid partial product is then shifted according to the position of the corresponding bit and finally accumulated to produce the final output. Figure 1.1 depicts this binary multiplication flow, where black dots represent activated bits involved in partial product generation.

Although conceptually simple, this method introduces significant latency and hardware overhead when implemented in large-scale systems. As a result, advanced multiplier architectures—including Booth encoders, Wallace and Dadda trees, and approximate multipliers such as the proposed Rounding-Based Multiplier (RoBA)—have emerged to improve speed, reduce logic depth, and lower energy consumption in embedded and real-time DSP systems [21], [22].

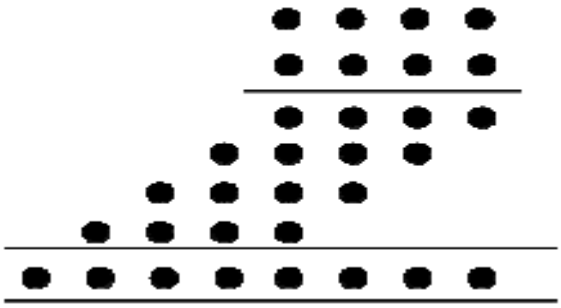

Figure 1: basic Multiplication

In digital systems, especially those leveraging power-efficient and parallel computing architectures, binary multiplication forms the cornerstone of many arithmetic and signal processing operations. Unlike decimal multiplication, binary multiplication is fundamentally simpler to implement in hardware due to its reliance on just two digits—0 and 1—and a reduced set of operations comprising bitwise AND, left-shift, and binary addition [23].

In the case of signed binary multiplication, particularly when operands are represented in two's complement form, the Most Significant Bit (MSB) is treated as the sign bit. The MSB determines whether the value is positive (0) or negative (1). Correct handling of this bit is critical in ensuring the validity of the final result. Any mishandling, such as treating signed numbers as unsigned, leads to substantial errors in computation [24].

The general procedure of binary multiplication consists of a series of bitwise evaluations of the multiplier. For each bit set to '1', the multiplicand is shifted to the left (according to the bit's position) and added to the cumulative result. Bits set to '0' contribute nothing to the product. This process is repeated for all bits in the multiplier, from the Least Significant Bit (LSB) to the MSB, generating intermediate partial products that are finally summed to yield the result [25].

In signed multiplication, a preprocessing step is typically used to determine the sign of the final product, which is the XOR of the sign bits of the multiplicand and multiplier. After multiplication is performed using absolute values, the sign is applied to the result, often by computing the two's complement of the unsigned product if the result should be negative [26].

1.4.2 Stages of Binary Multiplication

The binary multiplication process, whether executed manually or mapped to hardware, can be

conceptually divided into three fundamental stages [27]:

1. Partial Product Generation

Each bit of the multiplier is ANDed with the entire multiplicand to generate a row of partial products. If the multiplier bit is '0', the row is all zeros. If it is '1', the row replicates the multiplicand.

2. Partial Product Reduction

The rows of partial products are shifted left based on bit position and then aligned. This leads to multiple rows of binary values that must be accumulated to form the result.

3. Final Summation

The aligned partial products are then added together, often using carry-save adders (CSAs) or other optimized adder trees (e.g., Wallace or Dadda trees) to compute the final binary product. In this stage, the intermediate carry and sum bits are managed efficiently to reduce delay and maintain accuracy [28].

To illustrate this process, consider the binary multiplication of 1101_2 (-3_{10} in two's complement) and 0101_2 (5_{10}). The process involves generating partial products corresponding to each bit of the multiplier (0101_2), shifting the multiplicand appropriately, and summing the intermediate results. This operation, when performed at the gate level in hardware, reflects the exact same logical flow: bitwise AND for product rows, shift registers, and binary adders for summation.

In hardware, each multiplication stage is mapped to dedicated combinational and sequential logic. The partial product generation is typically implemented using an array of AND gates. The reduction stage may use compressor circuits or adder trees, while the final summation relies on fast carry-propagate adders or look-ahead structures.

During the summation of partial products, outputs are categorized into:

- Sum bits – representing the current binary value at a given bit position.
- Carry bits – which are propagated to the next higher-order bit to complete the binary addition.

This structure ensures both accuracy and efficiency in final result computation, which is vital for time-critical DSP and control applications [29].

				1	1	0	1		Multiplicand
			×	0	1	0	1		Multiplier
	1	1	1	1	1	1	0	1	PP1
	0	0	0	0	0	0	0		PP2
	1	1	1	1	0	1			PP3
+		0	0	0	0	0			PP4
1	1	1	1	1	0	0	0	1 = −15	Product

Fig 2 Multiplication process

The core mechanism of binary multiplication involves bitwise processing of the multiplier, typically performed by scanning one bit at a time, either from the Least Significant Bit (LSB) to the Most Significant Bit (MSB) or vice versa, depending on the architecture. For each bit of the multiplier, the multiplicand is conditionally added to the partial result—only if the multiplier bit is set to '1'. The generated partial product is then left-shifted by a number of positions corresponding to the bit's significance, effectively aligning it according to the binary weight of the multiplier bit [30].

This shifting reflects the increasing contribution of higher-order bits in the multiplication process. As a result, each subsequent partial product row is offset one position to the left relative to the previous one, thereby building the binary equivalent of decimal positional weighting.

At each bit column, the overlapping bits of the partial products are accumulated using binary adders, which output two values: the sum bit and the carry bit (also referred to as the transport bit). These carries are propagated to the next column in the addition sequence. The final product is obtained by summing all these aligned partial products, including the accumulated carries, across their respective columns.

When multiplying an n-bit multiplicand with an m-bit multiplier, the full binary product will span up to (n + m) bits, and the computation involves generating m partial product rows. The iterative accumulation and propagation of these rows are fundamental to all hardware multiplication architectures, whether they be array multipliers, tree-based multipliers, or approximate multipliers. This approach forms the foundational principle behind binary multiplication in digital VLSI systems and is critical in ensuring

both correctness and timing efficiency [31].

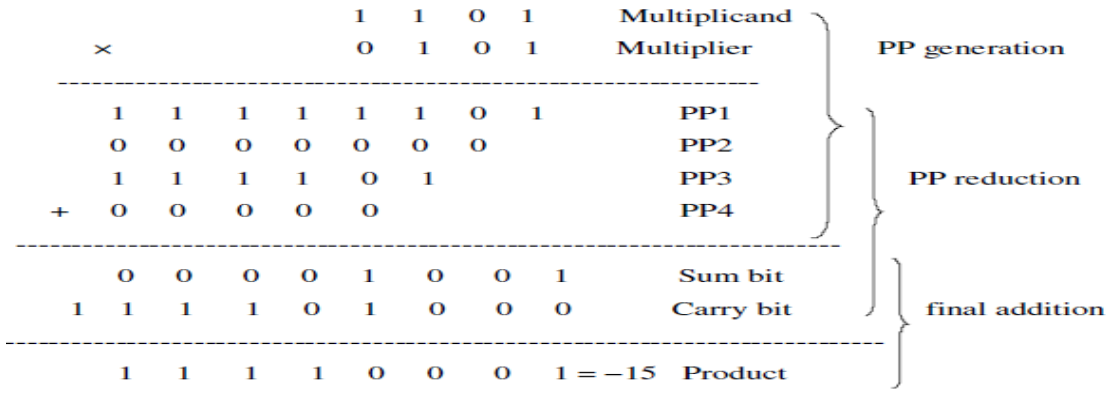

Fig 3: Multiplication showing sum and carry

Binary multiplication, a foundational operation in digital systems, involves processing one bit of the multiplier at a time, typically from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). For each bit, if it is set to 1, the multiplicand is conditionally added to the intermediate result. The result is then left-shifted by one position to reflect the binary weight of the next multiplier bit. This process continues iteratively across all bits of the multiplier [32].

The resulting partial products are aligned according to bit significance, and within each bit column, binary addition is performed to accumulate values. This generates two outputs: the sum bits and carry bits, which are propagated column-wise. For an n-bit multiplicand and an m-bit multiplier, the final product can span up to (n + m) bits, formed by summing m partial product rows [33].

In binary systems, numerical values are encoded using only two states: 0 and 1. This simplicity simplifies partial product generation, as each multiplier bit either produces zero or a replica of the multiplicand. However, the accumulation of these partial products—especially in high-speed systems—remains a computational bottleneck [34].

In software-based multiplication, particularly on processors without dedicated hardware multipliers, each partial product is generated and summed sequentially. While effective and easy to implement, this approach incurs significant latency, requiring one or more cycles per partial product. In contrast, hardware multipliers optimize this by parallelizing the process, dramatically improving throughput in real-time and embedded systems [35].

Binary multiplication can be categorized into unsigned and signed operations. In unsigned multiplication, the operation involves a systematic sequence of bit shifts and additions. Let the multiplicand be defined as

$x=xn-1xn-2...x1x0x = x_{n-1}x_{n-2}...x_1x_0x=xn-1xn-2...x1x0$

and the multiplier as

$y=ym-1ym-2...y1y0y = y_{m-1}y_{m-2}...y_1y_0y=ym-1ym-2...y1y0,$

the product is computed by forming m shifted copies of the multiplicand and summing them appropriately [36].

The multiplication process is generally divided into three key stages:

1. Partial Product Generation: Producing intermediate terms via bitwise AND operations between multiplicand and multiplier bits.
2. Partial Product Reduction: Using adder trees (e.g., Wallace or Dadda) or compressors to reduce the number of rows.
3. Final Addition: Employing fast adders (e.g., Carry-Lookahead, Carry-Select) to generate the final product from the reduced partial rows [37].

Modern computational domains such as machine learning, computer vision, and data analytics exhibit high tolerance to imprecision, allowing for the relaxation of exact arithmetic constraints. This opens a promising design avenue where approximate computing can be leveraged to trade off accuracy for improvements in power efficiency, area, and delay [38].

In this context, we propose a novel approximate multiplier featuring a dynamic precision-efficiency balancing mechanism. This architecture is engineered to deliver near-zero-mean errors and avoids cumulative biases across multiple operations, making it particularly suitable for neural networks,

inference engines, and multimedia workloads where such operations dominate.

The proposed design produces an unbiased error distribution, characterized by a Gaussian-like profile with a mean near zero. The standard deviation of errors, which reflects computational variance, ranges from 0.45% to 3.61% depending on the configuration [39]. This ensures that, statistically, errors tend to cancel out over time, preserving the quality of output in error-resilient domains.

Experimental evaluations using image processing (e.g., smoothing and edge detection) and classification tasks (e.g., digit recognition, object detection) demonstrate that the proposed approximate multiplier can achieve power savings between 54% and 80% without substantial loss in output quality. In configurations optimized for higher precision, the design still achieves up to 58% power reduction, outperforming several state-of-the-art approximate multipliers previously reported in literature [40].

Furthermore, the customizability of the architecture allows system designers to fine-tune parameters—such as truncation depth, rounding granularity, and operand handling—for different applications. Due to its shorter critical path, the multiplier is also ideal for performance-sensitive designs, enabling its integration into the critical paths of low-power DSP cores, edge AI accelerators, and portable embedded devices.

This work introduces the Dynamic Range Impartial Multiplier (DRIM), a novel multiplier architecture tailored specifically for error-resilient applications such as artificial intelligence (AI), signal processing, and data analytics. The core innovation of DRIM lies in its dynamic configuration mechanism, which intelligently allocates computational effort toward most significant input bits, thereby enabling scalability and precision adaptability. This approach enhances energy efficiency while maintaining acceptable accuracy levels in workloads tolerant to imprecision.

A key strength of DRIM is its impartial error distribution. Across multiple operations, the cumulative error converges close to zero, minimizing bias and ensuring consistent performance even in repeated computations. This property is particularly valuable in iterative AI algorithms and inference pipelines, where multiplier output accuracy can influence downstream decision processes.

The DRIM architecture is designed to be modular and hardware-compatible, allowing seamless integration of a compact accurate core multiplier. This embedded precise unit is significantly smaller and more power-efficient than conventional full-precision multipliers. Such integration gives system designers the flexibility to choose specific multiplier sub-units based on power, area, and accuracy trade-offs, without compromising the overall behavior or compatibility of the system.

This adaptability makes DRIM an ideal fit for systems where dynamic precision control is essential—for instance, mobile inference accelerators, real-time image filters, or edge AI platforms [41].

Designing high-speed, low-power, and layout-regular multipliers remains a cornerstone of VLSI research. In digital signal processing (DSP) and embedded system applications, multiplier performance is closely tied to:

- The number of partial products generated, and
- The efficiency of the summation process used to accumulate these products [42].

Various optimization strategies have been proposed to minimize the number of partial products, aiming to reduce both power consumption and critical path delay. Among these, half-adder array multipliers stand out due to their ability to accelerate summation and reduce overall computation time. However, despite such innovations, developing power-efficient, high-speed logic circuits continues to pose challenges due to the growing demand for compact and energy-conscious designs [43].

This work also proposes a modified full adder (FA) design based on a multiplexer-driven architecture. This novel FA contributes directly to power reduction in the multiplier's accumulation stage, which is typically the most power-hungry component in the arithmetic unit. When integrated into a conventional multiplier array, the modified FA demonstrates tangible improvements in performance metrics.

The proposed design is modeled in Verilog HDL and functionally verified using Xilinx ISE simulation tools. ASIC synthesis results reveal a:

- 35.45% reduction in power consumption,
- 40.75% reduction in silicon area, and
- 15.65% decrease in propagation delay, compared to traditional full adder-based architectures [44].

These gains highlight the potential of this approach in low-power VLSI systems, especially those constrained by battery life or thermal budgets.

To further optimize the power profile, the architecture integrates Modified Booth Encoding (MBE) along with the Spurious Power Suppression Technique (SPST). MBE reduces the number of partial products by recoding the multiplier bits, while SPST prevents unnecessary switching activities during idle logic

conditions. Together, these techniques achieve a 22% reduction in dynamic power due to minimized signal transitions and reduced glitch propagation [45].

Detailed analysis of bit-level operations shows that higher-order bits tend to exhibit greater switching activity, contributing significantly to dynamic power dissipation. To mitigate this, the proposed multiplier adopts a linear array configuration, in which each partial product is formed by multiplying one bit of the multiplier with all bits of the multiplicand.

These partial products are then shifted according to their weight, aligned appropriately, and finally summed to generate the product. The linear layout simplifies routing and minimizes interconnect complexity—two factors that contribute to energy savings and faster signal propagation.

In the proposed multiplier architecture, the final addition stage is implemented using a conventional Ripple Carry Adder (RCA). While simple and easy to implement, the RCA introduces a notable limitation in the form of carry propagation delay, which increases linearly with the operand bit width. Specifically, if N is the bit-length of the multiplier, then (N − 1) adder stages are required to complete the summation of partial products.

Despite this, the modular and regular structure of the RCA aligns well with the array-based design of the proposed multiplier. This regularity not only facilitates straightforward layout and routing but also enhances the design's suitability for scalable, low-power digital systems, particularly in embedded and energy-constrained environments. The balance between simplicity and structural efficiency makes the RCA a practical choice, especially in systems where slight compromises in speed are acceptable in exchange for reduced area and power consumption.

Interestingly, the term RoBA multiplier also finds relevance in the field of economics, where it is used to evaluate the returns on benefits from public investments. These investments may range from small-scale infrastructure projects to large-scale public sector initiatives. In most cases, such investments are state-funded, and the RoBA multiplier serves as a tool for assessing their effectiveness in stimulating economic growth and optimizing resource allocation.

The RoBA multiplier, in economic terms, reflects the extent to which public spending contributes to sectoral productivity and broader development outcomes. It draws parallels with the demand-induced development theory, emphasizing how initial investments create cascading benefits across interconnected sectors. Over time, the value of the multiplier increases, underscoring the long-term impact of judicious public spending.

This multiplier model has proven particularly useful in guiding economic policy formation by providing a quantifiable means of comparing investment outcomes across sectors. Policymakers can use these insights to allocate resources more effectively, prioritize high-impact projects, and ensure sustainable economic performance.

PROPOSED APPROXIMATE MULTIPLIER

A. Multiplication Algorithm of RoBA Multiplier

The core concept of the proposed approximate multiplier is to leverage the computational simplicity when operands are powers of two (2^n). To describe the operation in detail, let us denote the rounded versions of inputs A and B as Ar and Br, respectively. The original multiplication of A×B can be expressed as:

$$A \times B = (A_r - A)(B_r - B) + A_r \times B + B_r \times A - A_r \times B_r$$

Among the four terms, the products Ar×Br, and Br×A can be efficiently implemented using **bit**-shift operations, owing to the power-of-two nature of Ar and Br In contrast, the term (Ar−A)(Br−B) is computationally more complex and has a minimal impact on the final result, as it represents the product of the rounding errors.

To simplify the hardware implementation, this term is excluded from the computation. As a result, the multiplication is approximated as:

$$A \times B \approx A_r \times B + B_r \times A - A_r \times B_r$$

This transformation enables the multiplication to be implemented using only three shift operations and two additions or subtractions. To apply this method, the values of AAA and BBB must first be rounded

to their nearest powers of two.

A noteworthy case arises when the value of A (or B) is of the form 3×2p−2, where ppp is a positive integer greater than one. In such situations, there are two equidistant powers of two—namely 2p that could serve as candidates for rounding. While both choices offer comparable approximation accuracy, selecting the larger value (except for the special case when p=2p) typically leads to more hardware-efficient rounding logic.

This preference is motivated by the structure of the rounding block, where values such as 3×2p−2are treated as "don't care" conditions. This flexibility simplifies the logic design of the rounding-up circuitry, resulting in more compact and efficient hardware implementations.

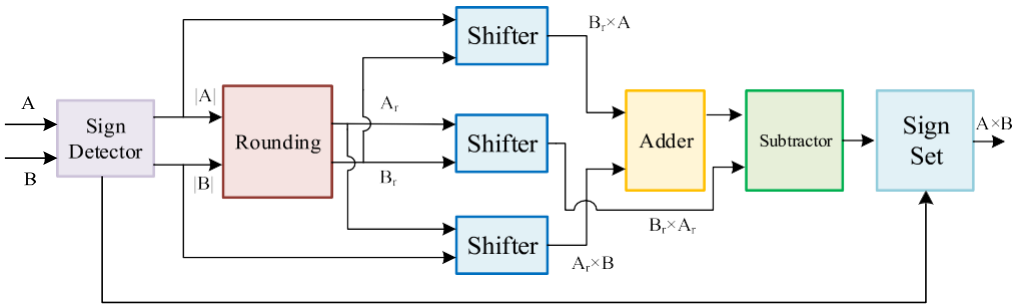

Fig. 4. Block diagram for the hardware implementation of the proposed multiplier.

The block diagram in figure 4 illustrates the internal architecture of a Rounding-Based Approximate Multiplier (RoBA), designed to simplify multiplication by leveraging operand rounding and shift-based computation. The process begins with a sign detector that determines the polarity of the inputs, which is preserved separately to be reapplied at the final stage. The original inputs A and B are then passed through a rounding block, which approximates them to the nearest powers of two, denoted as Ar and Br. These rounded values are crucial for enabling efficient shift-based operations in place of traditional multipliers.

Following the rounding stage, three shifter blocks perform left-shift operations to approximate the partial products: Br×A , Ar×B, and Ar×Br. These values are processed further to reconstruct an approximate result. The first two shifted outputs are combined using an adder, and the sum is then passed to a subtractor, which removes the overlapping term Ar×Br, as dictated by the core RoBA approximation logic.

Finally, the output from the subtractor is routed through a sign setting block, which applies the original sign information to yield the final signed approximate product. This architecture significantly reduces hardware complexity by replacing costly multiplication units with simple shifters and adders, making it well-suited for low-power, error-tolerant applications such as image processing and machine learning inference.

RESULTS

Figure 5: RTL Simulation Waveform of the RoBA Multiplier

This figure presents the RTL simulation waveform of the RoBA multiplier, displaying signal transitions over time. The inputs, intermediate signals, and the final output are visualized to verify the functional correctness of the design. The waveform shows the sequential operation of various stages, including rounding, shifting, and accumulation, confirming that the multiplier operates correctly according to the intended logic. The presence of distinct and periodic signal transitions indicates stable and synchronized processing across all clock cycles.

Figure 6: RTL Component Summary of the RoBA Multiplier

This figure shows the detailed RTL component information extracted post-synthesis. The design utilizes:

- Six 2-input 32-bit adders and one 3-input 32-bit adder for arithmetic operations.
- One 2-input XOR gate for bitwise control.
- Three 2-input 32-bit multiplexers (muxes) for conditional signal routing.

This summary reflects the structural efficiency of the design and confirms that the multiplier avoids excessive logic, supporting its low-power and area-optimized characteristics.

Figure 7: RTL Optimization Phase Summary

This figure captures the summary of the RTL optimization process as reported by the synthesis tool. The completion of Phase 2 of RTL optimization is recorded with a CPU time of 10 seconds and a total elapsed time of 33 seconds, highlighting the synthesis efficiency and the design's compatibility with automated optimization tools.

Figure 8: Testbench Simulation Result with Sample Inputs

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 24s, 2025
https://theaspd.com/index.php

This figure illustrates a functional testbench simulation for the RoBA multiplier using example inputs:

- A = 0x00000020 (32 in decimal)
- B = 0x00000030 (48 in decimal)
- Shift amount = 0x03

The computed final result is 0x000003C0, which corresponds to 960 in decimal—a correct approximation of the product. The signal trace shows proper propagation and timing of the result, validating the accuracy and timing behavior of the design under real-time test conditions.

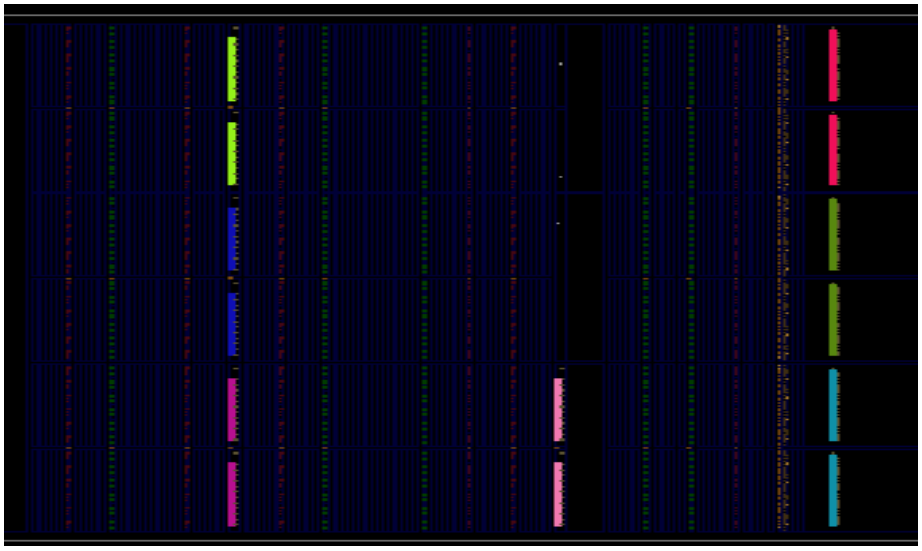

Figure 5: RTL Schematic

```
Detailed RTL Component Info :
+---Adders :
        2 Input    32 Bit         Adders := 6
        3 Input    32 Bit         Adders := 1
+---XORs :
        2 Input        1 Bit           XORs := 1
+---Muxes :
        2 Input    32 Bit          Muxes := 3
```

Figure 6: RTL Component Info

```
---------------------------------------------------------------------------------
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:10 ; elapsed = 00:00:33
---------------------------------------------------------------------------------
```

Figure 7: Timing Results

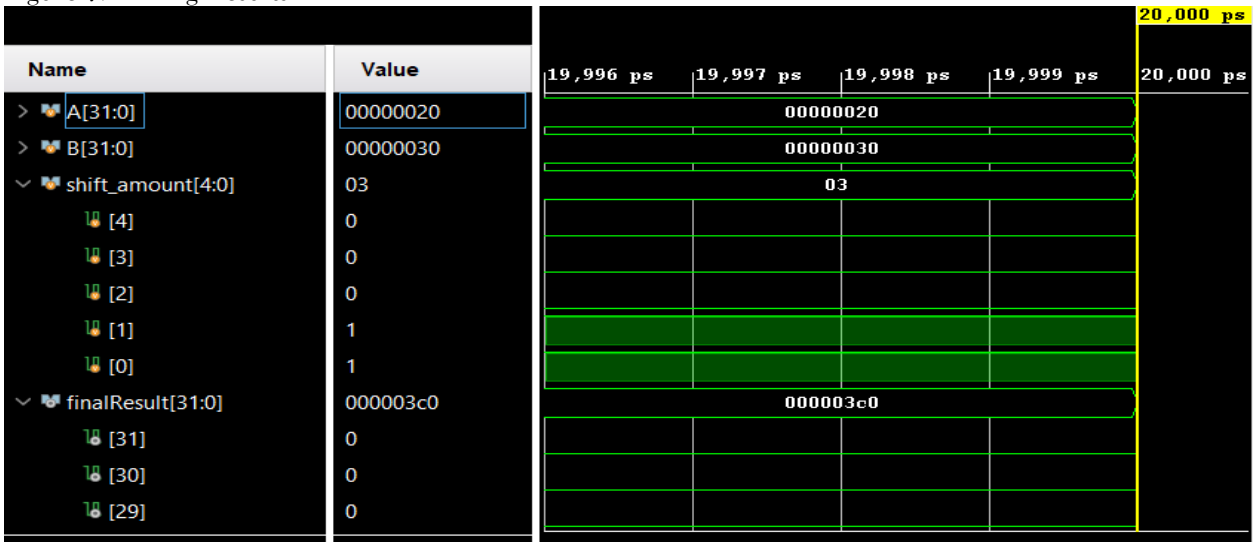

Figure 8: Waveforms

CONCLUSION

In this paper, we introduced the RoBA Multiplier, a high-speed and energy-efficient approximate multiplier. The proposed design achieves a favourable balance between performance and accuracy by rounding input operands to the nearest powers of two (2^n), thereby eliminating the computationally intensive components typically involved in standard multiplication. This approach results in significant improvements in operational speed and energy efficiency, while introducing only a minimal approximation error.

The architecture supports both signed and unsigned multiplication, with three distinct hardware configurations: one optimized for unsigned inputs and two tailored for signed operations. A comprehensive evaluation was conducted, comparing the RoBA multipliers with both exact and state-of-the-art approximate multiplier architectures across various design parameters. The results demonstrated that the proposed RoBA designs consistently outperformed their counterparts in most scenarios, both in terms of efficiency and hardware resource utilization.

To further validate its practical applicability, the RoBA multiplier was integrated into two representative image processing tasks: image sharpening and smoothing. Experimental results confirmed that the output quality of images processed using the RoBA multiplier closely matched that produced by exact multiplication algorithms, thereby underscoring its effectiveness and suitability for real-world, error-tolerant applications.

REFERENCES

[1]. M. Alioto, "Ultra-low power VLSI circuit design demystified and explained: A tutorial," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 59, no. 1, pp. 3–29, Jan. 2012.

[2]. V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 1, pp. 124–137, Jan. 2013.

[3]. H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 4, pp. 850–862, Apr. 2010.

[4]. R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in Proc. Int. Conf. Comput.-Aided Design, Nov. 2011, pp. 667–673.

[5]. F. Farshchi, M. S. Abrishami, and S. M. Fakhraie, "New approximate multiplier for low power digital signal processing," in Proc. 17th Int. Symp. Comput. Archit. Digit. Syst. (CADS), Oct. 2013, pp. 25–30.P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in Proc. 24th Int. Conf. VLSI Design, Jan. 2011, pp. 346–351.

[6]. D. R. Kelly, B. J. Phillips, and S. Al-Sarawi, "Approximate signed binary integer multipliers for arithmetic data value speculation," in Proc. Conf. Design Archit. Signal Image Process., 2009, pp. 97–104.

[7]. K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed mul- tiplier for error-tolerant application," in Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC), Dec. 2010, pp. 1–4.

[8]. A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," IEEE Trans. Comput., vol. 64, no. 4, pp. 984–994, Apr. 2015.

[9]. K. Bhardwaj and P. S. Mane, "ACMA: Accuracy-configurable multi- plier architecture for error-resilient system-on-chip," in Proc. 8th Int. Workshop Reconfigurable Commun.-Centric Syst.-Chip, 2013, pp. 1–6.

[10]. K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and area-efficient approximate wallace tree multiplier for error-resilient systems," in Proc. 15th Int. Symp. Quality Electron. Design (ISQED), 2014, pp. 263–269.

[11]. J. N. Mitchell, "Computer multiplication and division using binary log- arithms," IRE Trans. Electron. Comput., vol. EC-11, no. 4, pp. 512–517, Aug. 1962.

[12]. V. Mahalingam and N. Ranganathan, "Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition," IEEE Trans. Comput., vol. 55, no. 12, pp. 1523–1535, Dec. 2006.

[13]. Nangate 45nm Open Cell Library, accessed on 2010. [Online]. Avail- able: http://www.nangate.com/

[14]. H. R. Myler and A. R. Weeks, The Pocket Handbook of Image Process- ing Algorithms in C. Englewood Cliffs, NJ, USA: Prentice-Hall, 2009.

[15]. S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal process- ing and classification applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 6, pp. 1180–1184, Jun. 2015.

[16]. S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), Austin, TX, USA, 2015,pp. 418–425.

[17]. C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction," in Proc. 31st Int. Conf. Comput. Design (ICCD), 2013,pp. 33–38.

[18]. A. B. Kahng and S. Kang, "Accuracy-configurable adder for approx- imate arithmetic designs," in Proc. 49th Design Autom. Conf. (DAC), Jun. 2012, pp. 820–825.

[19]. Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Trans. Image Process., vol. 13, no. 4, pp. 600–612, Apr. 2004.

[20]. J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," IEEE Trans. Comput., vol. 62, no. 9, pp. 1760–1771, Sep. 2013.

[21]. N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed., Pearson, 2010.

[22]. B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford Univ. Press, 2010.
[23]. D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, 5th ed., Morgan Kaufmann, 2013.
[24]. M. Morris Mano and C. R. Kime, Logic and Computer Design Fundamentals, 4th ed., Pearson, 2008.
[25]. S. S. Lim, "A Study on Binary Multiplication Algorithm and VLSI Implementation Techniques," IEEE Trans. Computers, vol. 45, no. 4, pp. 456–464, Apr. 2000.
[26]. A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly J. Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236–240, 1951.
[27]. A. Momeni et al., "Design and Analysis of Approximate Multipliers for Energy-Efficient Systems," IEEE Trans. Computers, vol. 64, no. 4, pp. 948–961, Apr. 2015.
[28]. D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, 5th ed., Morgan Kaufmann, 2013.
[29]. C. H. Roth and L. L. Kinney, Fundamentals of Logic Design, 7th ed., Cengage Learning, 2013.
[30]. C. H. Roth and L. L. Kinney, Fundamentals of Logic Design, 7th ed., Cengage Learning, 2013.
[31]. B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford Univ. Press, 2010.
[34] N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed., Pearson, 2010.
[32]. J. M. Rabaey et al., Digital Integrated Circuits: A Design Perspective, 2nd ed., Pearson, 2003.
[33]. M. D. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann, 2003.
[34]. A. D. Booth, “A Signed Binary Multiplication Technique,” Quarterly J. Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236–240, 1951.
[35]. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," 18th IEEE ETS, 2013.
[36]. S. Venkataramani et al., “Quality programmable vector processors for approximate computing,” MICRO-46, 2013.
[37]. A. Momeni et al., “Design and Analysis of Approximate Multipliers for Energy-Efficient Systems,” IEEE Trans. Computers, vol. 64, no. 4, pp. 948–961, 2015.
[38]. J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," 18th IEEE European Test Symposium, 2013.
[39]. B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford Univ. Press, 2010.
[40]. A. Momeni et al., "Design and analysis of approximate multipliers for energy-efficient systems," IEEE Trans. Computers, vol. 64, no. 4, pp. 948–961, Apr. 2015.
[41]. A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in Proc. Design, Automation and Test in Europe, 2008.
[42]. H. Esmaeilzadeh et al., “Architecture support for disciplined approximate programming,” in Proc. 17th ASPLOS, 2012.
[43]. J. N. Mitchell, “Computer multiplication and division using binary log- arithms,” IRE Trans. Electron. Comput., vol. EC-11, no. 4, pp. 512–517, Aug. 1962.
[44]. V. Mahalingam and N. Ranganathan, “Improving accuracy in Mitchell’s logarithmic multiplication using operand decomposition,” IEEE Trans. Comput., vol. 55, no. 12, pp. 1523–1535, Dec. 2006.
[45]. Nangate 45nm Open Cell Library, accessed on 2010. [Online]. Avail- able: http://www.nangate.com/