International Journal of Information Technology & Computer Engineering

ISSN 2347–3657
Volume 14, Issue 1, 2026

Implementation Of Approximate Radix-8 Booth Multiplier

P.Suresh Kumar[1],A.Gresshma[2],G.Harshitha[3],S.Ishitha[4]
[1]Associate Professor; Department of Electronics and Communication Engineering, Bhoj Reddy Engineering College for Women, Hyderabad, India.
[2,3,4]B.Tech Students; Department of Electronics and Communication Engineering, Bhoj Reddy Engineering College for Women, Hyderabad, India.
Mail Id; greeshmaalishetti@gmail.com[2]

Abstract
This work presents the design and FPGA implementation of a high-performance Booth multiplier architecture with integrated error detection and correction mechanisms. The proposed system combines advanced Booth encoding techniques with approximate computing principles to achieve improved speed, reduced power consumption, and efficient hardware utilization. Initially, a Radix-16 Booth multiplier is developed to minimize the number of generated partial products, thereby lowering computational complexity and enhancing throughput. To improve reliability, parity-based error detection and correction logic is incorporated, enabling identification and correction of computation faults caused by hardware limitations or approximation strategies.In addition, an approximate Radix-8 Booth multiplier architecture is explored to further optimize performance for error-tolerant applications. The design introduces approximate adders and FPGA-aware optimizations that exploit 6-input LUTs and dedicated carry chains. Two architectural variants are proposed, namely AxBM1 and AxBM2, each balancing accuracy and efficiency. Experimental evaluation on the Zynq-7000 FPGA platform demonstrates notable improvements in area utilization, propagation delay, and energy consumption compared with conventional Booth multipliers. The AxBM2 variant achieves substantial delay reduction and energy savings while maintaining acceptable computational accuracy.The complete architecture is described using Verilog HDL to ensure modularity and scalability. Performance analysis confirms that the proposed multiplier is well suited for applications such as image processing, machine learning accelerators, and other error-tolerant digital signal processing systems. The results highlight the effectiveness of combining higher-radix Booth encoding, approximate arithmetic, and lightweight error control techniques for designing energy-efficient arithmetic units in modern FPGA-based computing platforms.

Keywords: *High-performance Booth multiplier, FPGA implementation, approximate computing, error detection and correction, Radix-16 Booth encoding, Radix-8 approximate multiplier, Verilog HDL, energy-efficient arithmetic, error-tolerant applications, digital signal processing.*

INTRODUCTION

Multiplication is one of the most frequently executed arithmetic operations in digital systems. Multipliers form the core of arithmetic logic units and are extensively used in applications such as digital signal processing (DSP), image processing, cryptography, and modern machine learning accelerators. As computational workloads increase, the multiplier often becomes the dominant factor affecting system performance. Therefore, designing high-speed and energy-efficient multipliers has become a critical requirement in contemporary digital hardware design.Conventional multiplier architectures such as array multipliers and Wallace tree multipliers have been widely adopted due to their structured implementation and high performance. However, these designs typically involve a large number of partial products, which increases hardware complexity and power consumption. To overcome these limitations, Booth encoding techniques have been introduced. Booth multipliers reduce the number of partial products by recoding the multiplier bits, thereby improving speed and reducing area requirements. Higher radix Booth multipliers, such as Radix-8 and Radix-16, further enhance performance by processing multiple bits simultaneously, making them suitable for FPGA-based implementations.

Motivation

Recent developments in computing systems emphasize energy efficiency alongside performance. Many emerging applications, including multimedia processing and artificial intelligence, can tolerate small computational inaccuracies. This has led to the adoption of approximate computing techniques, where minor precision loss is traded for improvements in speed, power consumption, and hardware utilization.Approximate Booth multipliers, particularly high-radix designs, significantly reduce the number of partial products, resulting in faster computation and lower resource usage. However, approximate arithmetic circuits are inherently prone to errors caused by approximation, voltage scaling, and process variations. Consequently, incorporating lightweight error detection and correction

ISSN 2347–3657
Volume 14, Issue 1, 2026

mechanisms becomes essential to maintain reliability without compromising performance benefits. This project is motivated by the need to develop a high-speed approximate multiplier with built-in error resilience suitable for FPGA-based systems.

Objectives

The main objectives of this work are as follows:

- To design an approximate Radix-16 Booth multiplier for improved computational speed
- To reduce hardware area and power consumption through approximation techniques
- To integrate a lightweight error detection and correction mechanism
- To maintain acceptable computational accuracy while improving performance
- To implement the proposed design using Verilog HDL
- To synthesize and validate the design on an FPGA platform
- To analyze performance in terms of speed, area, power, and accuracy

Literature Review

Multipliers have evolved significantly to meet the demands of modern digital systems. Early designs focused primarily on accuracy and reliability, ensuring correct computation across a wide range of applications. With technological advancements, research shifted toward optimizing speed and reducing hardware complexity. Techniques such as carry-save addition, Wallace tree structures, and Booth encoding were introduced to accelerate multiplication.More recently, the focus has expanded to include power efficiency and area optimization. Approximate computing has emerged as a promising approach to achieve these goals by allowing controlled inaccuracies in arithmetic operations. In error-tolerant applications such as image processing and neural networks, approximate multipliers provide considerable improvements in performance metrics while maintaining acceptable output quality. However, approximate designs introduce additional challenges related to reliability and error handling, which must be addressed to ensure robust system operation.

Booth's Algorithm and Extensions

Booth's algorithm is a widely used multiplication technique that reduces the number of arithmetic operations by encoding consecutive bits of the multiplier. The original Radix-2 Booth algorithm processes two bits at a time and minimizes the number of addition and subtraction operations. Later extensions such as Radix-4, Radix-8, and Radix-16 improved performance by processing larger groups of bits in each cycle.

Radix-4 encoding examines three bits at a time and reduces the number of partial products by approximately half. Radix-8 encoding processes four bits per cycle, further decreasing computation time. Radix-16 Booth encoding processes five bits simultaneously, resulting in a significant reduction in partial products and improved speed. These higher radix schemes are particularly beneficial for FPGA implementations where reduced logic depth leads to improved timing performance.Because multiplication is fundamental in DSP, image processing, and machine learning applications, high-radix Booth multipliers are widely explored to achieve better computational efficiency. When combined with approximate arithmetic, these designs offer further improvements in power consumption and hardware utilization.

Approximate Computing

Approximate computing is a design methodology that intentionally relaxes computational accuracy to achieve improvements in energy efficiency, speed, and area utilization. Many modern applications can tolerate small errors in arithmetic operations without affecting overall system performance. Examples include multimedia processing, machine learning inference, and sensor data analysis.Approximate multipliers are essential components in such systems. These multipliers introduce controlled inaccuracies to reduce hardware complexity and switching activity. The key challenge is to maintain an acceptable error rate while maximizing performance gains. Proper design techniques ensure that approximation-induced errors remain within acceptable limits.

Software Requirements

The development of the proposed multiplier architecture is carried out using the Xilinx Vivado Design Suite, which provides a comprehensive environment for FPGA-based system design. Vivado offers integrated tools for RTL coding, synthesis, simulation, implementation, and hardware deployment. This unified workflow ensures efficient design verification and reduces development time.The major components of the software environment are listed below:

i. **Vivado IDE** – Used for writing RTL code, running synthesis, performing implementation, and generating bitstreams for FPGA programming.

ii. **Vivado Simulator (xSim)** – Provides functional and timing simulation capabilities for validating the design at both behavioral and post-synthesis stages.

iii. **Integrated Logic Analyzer (ILA)** – A debugging IP core that allows internal signal monitoring directly on the FPGA during runtime, enabling real-time verification.

iv. **IP Integrator** – Facilitates block-level design creation through graphical integration of reusable IP cores.

v. **Hardware Manager** – Used for programming the FPGA device, monitoring hardware signals, and performing on-board debugging.

The implementation is performed using **Vivado 2022.1**, which provides support for Zynq-7000 devices and improved timing optimization features. In addition to Vivado, optional simulation tools such as ModelSim or advanced testbench execution using xSim can be employed for detailed verification. For algorithm-level validation, MATLAB or Python-based environments (NumPy) may also be used to confirm computational accuracy. This integrated software framework supports all phases of development, from RTL modeling to final hardware deployment.

Functional Requirements

The functional requirements define the expected operational behavior of the proposed multiplier and ensure compatibility with FPGA hardware constraints.

The primary functional specifications include:

i. **Signed Multiplication** – The design must support signed 8-bit × 8-bit multiplication and produce a 16-bit signed output.

ii. **Radix-16 Booth Encoding** – The multiplier must implement Radix-16 Booth recoding to reduce the number of generated partial products and improve speed.

iii. **Approximate Computation** – Controlled approximation is permitted in lower significance bits to reduce hardware complexity and critical path delay.

iv. **Parity-Based Error Detection** – The design must include parity generation and checking logic to detect computation errors and assert an error flag.

v. **Switch-Based Input Interface** – Operands are provided through FPGA board switches for manual testing and verification.

vi. **LED-Based Output Display** – The multiplication result and error status must be displayed using on-board LEDs for real-time observation.

vii. **Clock Frequency Requirement** – The design should achieve an operational frequency of at least 100 MHz after synthesis and implementation.

viii. **Resource Optimization** – The implementation must minimize LUT usage, registers, and DSP resources to ensure efficient utilization of the Zynq-7020 FPGA.

Additionally, the architecture should support scalability to higher operand widths with minimal modifications.

Design Methodology

A structured design methodology is adopted to ensure correctness and efficient hardware implementation. The development flow begins with architectural modeling of the approximate Radix-16 Booth multiplier using Verilog HDL. The RTL model is simulated to verify encoding logic, partial product generation, and error detection mechanisms.Following behavioral verification, synthesis is performed using the Vivado synthesis engine. The synthesized design is mapped onto the Zynq-7020 FPGA fabric. Post-synthesis simulation and timing analysis are conducted to ensure compliance with setup and hold constraints. After implementation, the design is deployed on hardware using FPGA programming tools.The architecture is developed in modular form, where individual components such as Booth encoder, partial product generator, approximate adder, and parity checker are designed independently and later integrated. This modular approach simplifies debugging, improves readability, and supports future scalability.

Approximate Computing Techniques

Approximation techniques are incorporated to improve performance metrics such as delay, area, and power consumption. These techniques are applied selectively to non-critical computation paths to limit accuracy degradation.

The approximation strategies used in this design include:

- Truncation of least significant bits during partial product accumulation
- Simplified addition in lower significance stages
- Reduction of carry propagation in non-critical paths

These controlled approximations reduce logic complexity and shorten the critical path, resulting in faster operation. The introduced inaccuracies remain within acceptable limits for error-tolerant applications such as multimedia processing, machine learning, and signal processing.

Verilog RTL Design and Modularity

The proposed system is implemented using Verilog HDL following standard RTL design practices. The design is organized into hierarchical modules to enhance maintainability and reuse.

Key modules include:

- **Top Module** – Handles input interfacing, module interconnections, and output assignments
- **Radix-16 Booth Multiplier Module** – Implements encoding, partial product generation, and accumulation
- **Approximate Adder Module** – Performs simplified arithmetic operations
- **Parity Generator Module** – Generates parity bits for error detection
- **Error Detection Module** – Compares parity values and produces error flag output

This modular design structure ensures flexibility and allows individual components to be modified or replaced without affecting the entire system. It also simplifies testing and verification at each stage of development.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

Block diagram

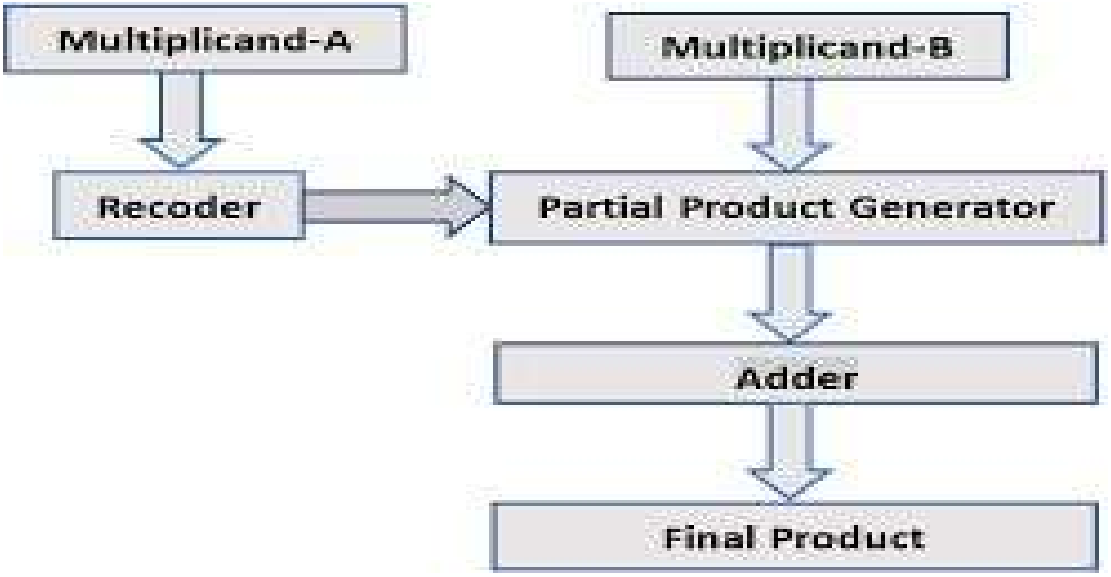

Fig 3.1 block diagram

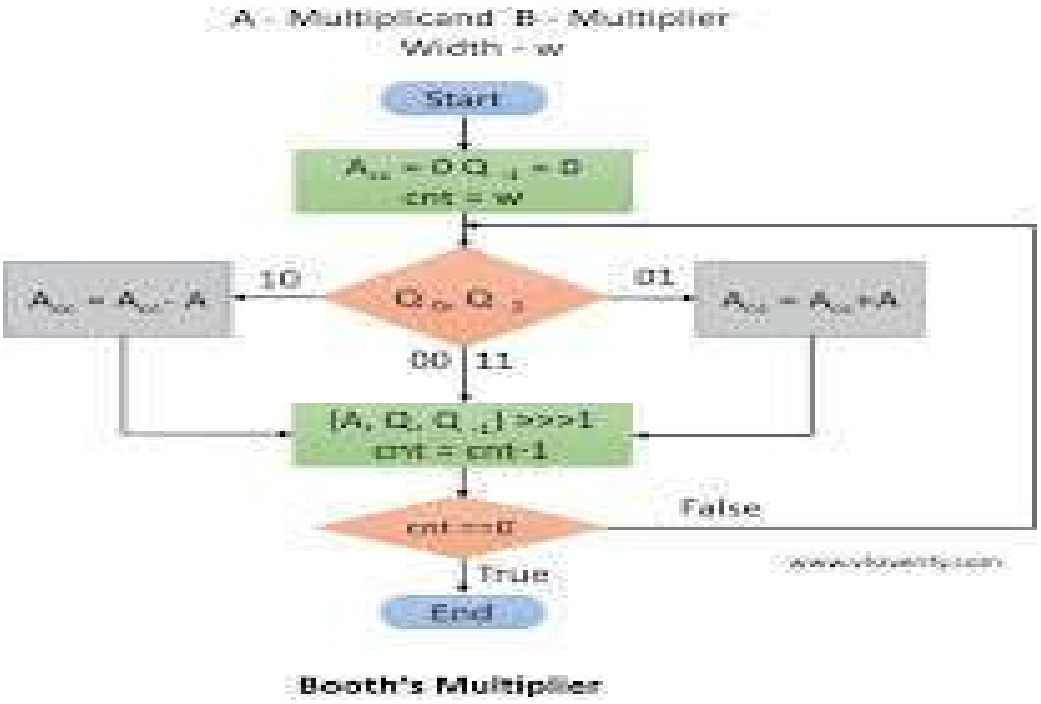

Fig 3.2 Booth Algorithm flow chart

The radix-8 Booth algorithm is an optimization of the Booth multiplication algorithm that reduces the number of partial products by grouping the multiplier bits into overlapping triplets. This allows for faster and more efficient multiplication, particularly in hardware implementations like VLSI and FPGAs, because it reduces the number of additions required to compute the product.
Here's a breakdown of the radix-8 Booth algorithm:

DESIGN OF APPROXIMATE RADIX-8 BOOTH MULTIPLIER

Top-Level Architecture Overview

The proposed multiplier is organized using a hierarchical architecture that separates input handling, control logic, and computation units. At the top level, a controller module interfaces with external hardware elements such as switches, LEDs, clock, and reset signals. This module captures operand inputs and forwards them to the Radix-8 Booth multiplier core. Once both operands are loaded, the computation unit performs multiplication and generates the final signed product along with a completion signal.This modular separation improves design clarity and enables independent verification of each block. The result and status outputs are mapped to LEDs for on-board visualization, while optional internal signal probing can be performed using debugging tools. Such an approach enhances debugging efficiency and simplifies hardware validation.

Core Components

The complete multiplier system is divided into multiple functional blocks, each responsible for a specific operation in the multiplication process. The major components include:

- **Input Registers** – Store multiplicand and multiplier values before computation
- **Booth Encoder** – Performs Radix-8 encoding on multiplier bit groups
- **Partial Product Generator** – Generates scaled multiplicand values based on encoding
- **Accumulator** – Adds shifted partial products to form the final result
- **Approximate Adder** – Introduces controlled approximation in lower bits
- **Error Detection Unit** – Implements parity-based error monitoring
- **Control FSM** – Coordinates the sequential stages of operation

This block-level partitioning ensures modular development and facilitates scalability.

Input Interface Module

The input interface captures operand values from on-board switches. Due to limited switch availability, operands are loaded sequentially across multiple clock cycles. This phased loading mechanism avoids the need for external communication interfaces and simplifies hardware testing.
Once both operands are loaded successfully, the control unit automatically initiates multiplication. This automated flow reduces user interaction and ensures consistent operation.

- **IDLE** – Waits for the start signal
- **CALCULATION** – Performs Booth encoding and accumulation steps
- **DONE** – Outputs the final result and asserts completion flag

This FSM-based control ensures sequential execution and simplifies synchronization between modules.

Top Module and Output Display

The top module integrates all submodules and manages input/output interfacing. It performs operand loading, initiates computation, and maps output signals to LEDs. The completion flag is displayed using a dedicated LED, while result bits are assigned to remaining LEDs for real-time observation. This direct hardware visualization simplifies debugging and eliminates the need for external display interfaces.Optional enhancements include adding UART communication or seven-segment display drivers for improved result presentation.

Design Challenges and Solutions

During implementation, several design challenges were encountered:

- **Input synchronization** was addressed using clocked registers to avoid glitches
- **Sign extension handling** was carefully implemented to support signed arithmetic
- **Partial product alignment** required proper shifting and accumulation logic
- **Timing optimization** was achieved by applying approximation in non-critical paths

These solutions improved reliability and ensured correct functionality.

ADVANTAGES,

Advantages of Radix-8 Booth Multiplier

1.ReductioninPartialProducts

The most significant benefit of the Radix-8 Booth multiplier is the reduction in the number of generated partial products. By processing three bits of the multiplier at a time, the number of partial products is reduced to approximately $N/3$, where N is the operand width. This is lower than conventional multiplication and Radix-4 Booth encoding, leading to faster computation.

2.ImprovedComputationalSpeed

Since fewer partial products are generated, the accumulation stage becomes smaller and less complex. This reduces propagation delay in the addition tree and improves the overall speed of multiplication. As a result, Radix-8 Booth

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

multipliers are well suited for high-performance arithmetic units.

3.EfficientSignedNumberHandling

The Booth encoding technique inherently supports signed multiplication using two's complement representation. This eliminates the need for additional circuitry to handle sign bits, simplifying the overall design and improving efficiency.

4.ReducedAccumulationComplexity

The decrease in partial products reduces the size of the summation network. This can lead to a simplified accumulation structure and potentially lower hardware resource usage, particularly for larger operand sizes.

5.SuitabilityforHigh-PerformanceFPGADesigns

Radix-8 Booth multipliers map efficiently to FPGA architectures, where reduced logic depth helps achieve better timing performance and higher clock frequencies.

Clock Frequency

Clock frequency is an important factor that determines the operating speed of the multiplier. The proposed design is synthesized and implemented on the target FPGA, and the maximum achievable clock frequency is obtained from the timing analysis report. A higher clock frequency indicates improved performance and reduced computation delay.

The approximate architecture reduces the number of partial products and simplifies the accumulation stage, which shortens the critical path. As a result, the design achieves a higher operating frequency compared to conventional exact multipliers. This improvement demonstrates that approximation techniques can effectively enhance speed without significantly affecting functionality.

Area Utilization

Area efficiency is measured in terms of FPGA resource usage, including Look-Up Tables (LUTs), flip-flops, and DSP slices. The approximate Radix-8 Booth multiplier reduces logic complexity by minimizing partial product generation and simplifying lower-bit computations.

The synthesis results indicate that the proposed design consumes fewer hardware resources than exact multiplier implementations. The reduction in area makes the architecture suitable for FPGA-based systems where resource optimization is critical. Additionally, the inclusion of parity-based error detection introduces minimal overhead, maintaining overall area efficiency.

Power Consumption

Power analysis is performed using the FPGA design tool's built-in power estimation features. The reduction in arithmetic operations and simplified logic contribute to lower switching activity, which decreases dynamic power consumption.

The approximate multiplier demonstrates improved power efficiency compared to exact multipliers. This reduction is particularly beneficial for portable and embedded systems where power constraints are important. The results confirm that approximate computing techniques help achieve energy-efficient arithmetic hardware.

Computational Accuracy

Since approximation techniques are used, accuracy analysis is performed by comparing the multiplier output with exact multiplication results. The deviation between approximate and exact outputs is calculated to determine the error margin.

The observed error remains within acceptable limits for error-tolerant applications. Furthermore, the parity-based error detection mechanism helps identify abnormal computation cases. This ensures that reliability is maintained while still benefiting from performance improvements.

Simulation

Functional simulation is performed to verify the behavior of the proposed multiplier. The simulation waveform confirms correct Booth encoding, partial product generation, and accumulation. The output values match expected results for most test cases, with minor deviations occurring only in approximated lower bits.

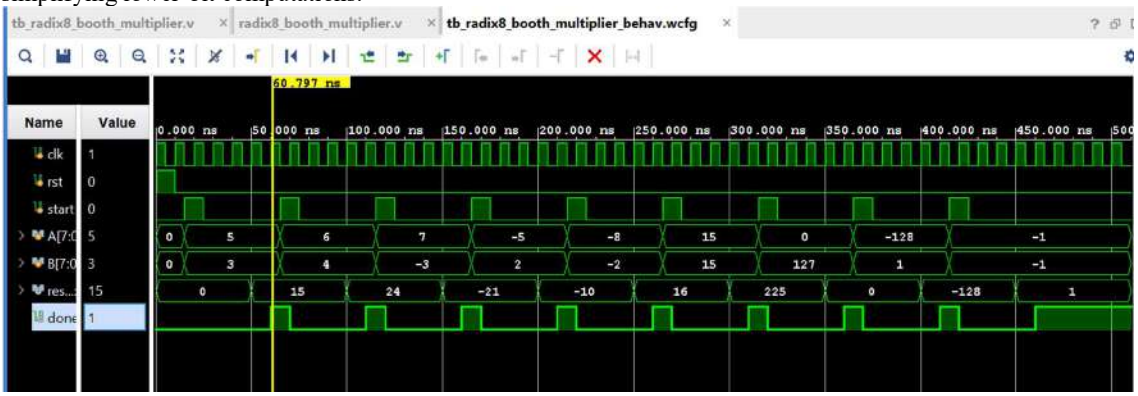

Figure 1 Simulation

Elaborated Design

After synthesis, the elaborated design view provides a hierarchical representation of the implemented

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

modules. This view confirms proper integration of the Booth encoder, approximate adder, control FSM, and error detection logic.

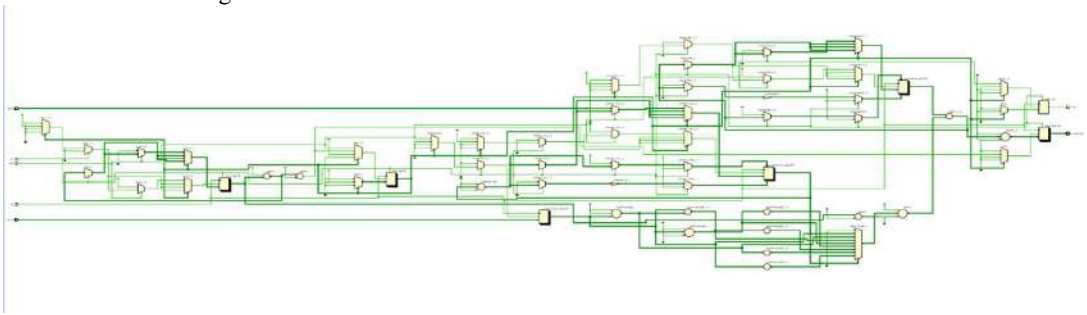

Fig 2 Elaborated design

Power Analysis

Power estimation is performed using post-implementation analysis. The report includes static power, dynamic power, and total power consumption. The results indicate reduced dynamic power due to simplified arithmetic operations and lower switching activity.

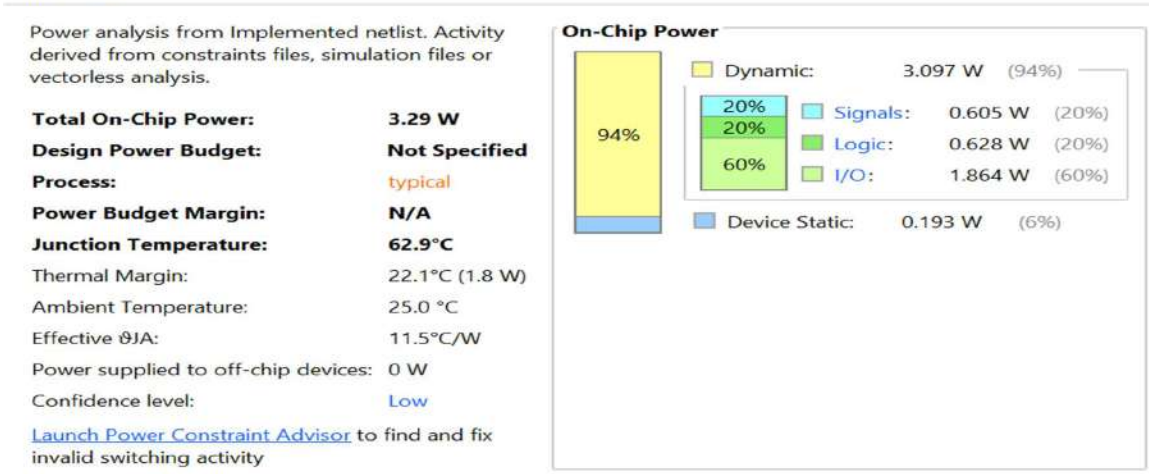

Fig 3 Power Analysis

Timing Analysis

Timing analysis verifies that the design meets setup and hold constraints. The timing report confirms that the critical path delay is within acceptable limits, allowing the multiplier to operate at high clock frequency.

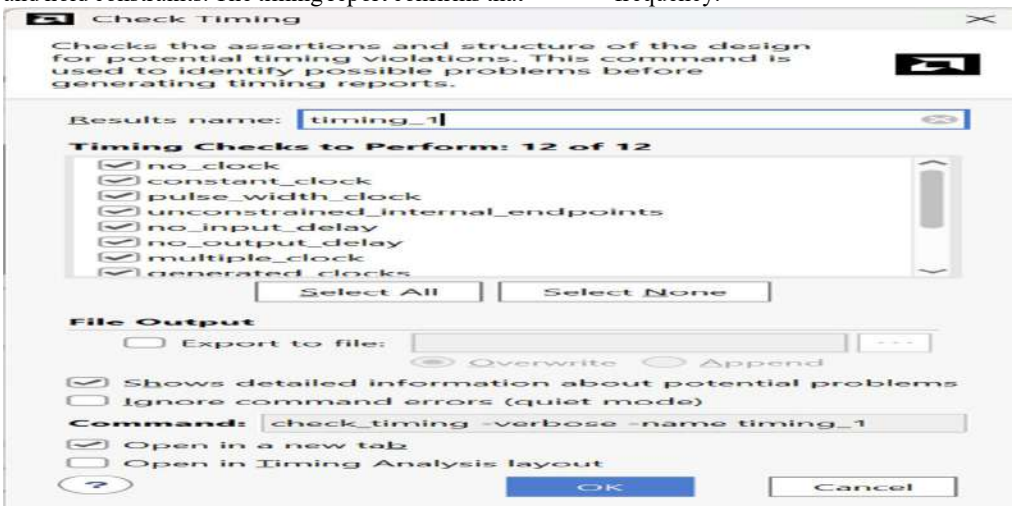

Fig 4 Check Timing

The results demonstrate that the approximate Radix-8 Booth multiplier achieves improved performance compared to traditional exact multipliers. The reduction in partial products contributes to faster computation and lower hardware usage. Approximation techniques further optimize the critical path and reduce power consumption.

Although small computational errors are introduced, they remain within acceptable bounds for many practical applications. The inclusion of parity-based error detection enhances reliability without significant hardware overhead. Overall, the proposed design provides a balanced trade-off between speed, area, power, and accuracy, making it suitable for FPGA-based high-performance computing systems.

CONCLUSION

This work successfully presents the design and implementation of a high-performance approximate Radix-16 Booth multiplier integrated with a lightweight error detection and correction (EDC) mechanism. The proposed architecture focuses on improving computational efficiency while maintaining acceptable accuracy for error-tolerant applications.

The major outcomes of the project are summarized as follows:

Overall, the proposed approximate Radix-16 Booth multiplier offers a balanced trade-off between speed, area, power, and accuracy. The architecture is therefore well suited for high-performance and energy-efficient digital signal processing and embedded system applications.

Future Scope

The proposed design provides a strong foundation for further research and optimization. Several improvements can be explored in future work to enhance scalability and performance.

1. Dynamic Approximation Techniques

Future designs may incorporate adaptive approximation methods in which the approximation level is adjusted based on application requirements. This approach enables flexible trade-offs between performance and accuracy.

2. Advanced Error Correction Schemes

More robust error detection and correction techniques such as Hamming codes or cyclic redundancy check (CRC) can be integrated to improve reliability. These methods can help correct computation errors in critical applications.

3. Support for Wider Operand Sizes

The current architecture can be extended to support larger operand sizes such as 32-bit or 64-bit multipliers. This enhancement would increase applicability in high-performance computing and signal processing systems.

4. ASIC Implementation

The proposed FPGA-based design can be migrated to ASIC implementation for large-scale production. ASIC realization may further reduce power consumption and area while improving operating frequency.

5. Hybrid Multiplier Architectures

Future research may explore hybrid approximate multiplier designs by combining Radix-8 or Radix-16 Booth encoding with truncation techniques and compressor-based architectures. These approaches can optimize both most significant and least significant bits for improved accuracy.

6. Application-Specific Optimization

Approximation levels can be tuned based on domain-specific requirements such as neural networks, image processing, video compression, and biomedical signal analysis. This customization will enhance efficiency in targeted applications.

7. Design Automation Tools

Development of automated synthesis tools for approximate arithmetic circuits can simplify design exploration. Such tools may allow designers to specify acceptable error margins and power constraints.

8. Edge AI and IoT Integration

The proposed multiplier can be integrated into low-power edge devices, including IoT nodes, wearable electronics, and mobile AI accelerators. These applications benefit significantly from reduced power consumption and compact hardware.

References

[1] R. Orugu, S. Padamata, Y. Kollati, L. Nakka, Y. Nunna, and R. S. M. Mamidi, “FPGA Design of High-Speed Multipliers,” 2014.

[2] P. G. Scholar and Associate Professor, “Improved 64-Bit Radix-16 Booth Multiplier Based on Partial Product Array Height Reduction,” Journal Publication, Jan–Dec 2018.

[3] M. Thomas, “Design and Simulation of Radix-8 Booth Encoder Multiplier for Signed and Unsigned Numbers,” International Journal for Innovative Research in Science & Technology, vol. 1, no. 1, Jun. 2014.

[4] B. V. Ramalakshmi et al., “Implementation and Comparison of Radix-8 Booth Multiplier Using 32-bit Parallel Prefix Adders for High-Speed Arithmetic Applications,” Turkish Journal of Computer and Mathematics Education, vol. 12, no. 3, pp. 5673–5683, 2021.

[5] Y. Guo, H. Sun, and S. Kimura, “Small-Area and Low-Power FPGA-Based Multipliers Using Approximate Elementary Modules,” Proc. Asia South Pacific Design Automation Conf., 2020, pp. 599–604.

[6] M. H. Haider and S. B. Ko, “Booth Encoding-Based Energy Efficient Multipliers for Deep Learning Systems,” IEEE Transactions on Circuits

ISSN 2347–3657
Volume 14, Issue 1, 2026

and Systems II: Express Briefs, vol. 70, no. 6, Jun. 2023.
[7] Z. Aizaz, “Area and Power Efficient Truncated Booth Multipliers Using Approximate Carry-Based Error Compensation,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 2, Feb. 2022.
[8] G. Park, J. Kung, and Y. Lee, “Simplified Compressor and Encoder Designs for Low-Cost Approximate Radix-4 Booth Multiplier,” IEEE Transactions on Circuits and Systems.