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

Area and Delay Optimized 4×4 Dadda Multiplier Design in 45nm CMOS Technology

Sathya Prabha Goli[1], K Naresh[2]
[1]PG Scholar, Sree dattha Institute of Engineering and Science, sathyaprabhamtech@gmail.com
[2]Assistant professor, Sree Dattha Institute of Engineering and Science, balu.482@gmail.com

Abstract- *This project presents a detailed model of a 4-bit multiplier emphasizing low power consumption and high-speed operation, achieved using the Dadda algorithm. The primary building block of this design is an optimized full adder, featuring low power dissipation and minimal propagation delay. To realize these characteristics, the full and half adder blocks are carefully designed using pass-transistor logic and implemented with 45nm CMOS process technology, significantly reducing both power consumption and delay. The application of the Dadda algorithm further enhances performance by minimizing propagation delay within the multiplier architecture.*
The model has been developed using Vivado, ModelSim, DSCH, and Microwind tools, utilizing advanced 90nm technology for implementation. The proposed multiplier operates at a frequency of 3.83 GHz, demonstrating its high-speed performance. It also exhibits an average dynamic power consumption of 184.3 μW at a supply voltage of 1V. This efficient design results from the combination of innovative algorithmic techniques and advanced circuit design methodologies, ensuring both low power operation and high-speed performance.
Keywords: *Dadda multiplier, 4-bit multiplier, low power, high speed, pass-transistor logic, CMOS 45nm, Vivado, ModelSim, Microwind, dynamic power, propagation delay, digital circuit design*

INTRODUCTION

Multiplication is one of the fundamental arithmetic operations essential to various electronic systems and digital signal processing applications. In modern digital systems—ranging from image processing and communications to embedded computing—high-performance multipliers play a crucial role in ensuring system efficiency and computational speed.

To achieve optimal performance, low-latency and power-efficient multiplier architectures are highly preferred. Such designs enable maximum throughput with minimal response time, which is critical in time-sensitive and resource-constrained environments.

The core building blocks of multiplier circuits are typically full adders and half adders. The performance of these blocks directly influences the overall speed and power consumption of the multiplier. Various design techniques have been proposed to enhance the efficiency of these adders. These include pass transistor logic (PTL) for reduced transistor count and power savings, complementary metal-oxide-semiconductor (CMOS) logic for robust switching and improved noise margins, split-percentage data-driven logic (SP-DDL) to minimize dynamic power, and advanced CMOS process technologies that support finer feature sizes and lower voltage operation[1].

In addition to optimizing adder designs, the use of advanced multiplication algorithms contributes significantly to improving performance metrics such as the power-delay product (PDP). Widely adopted algorithms include the Dadda tree algorithm, known for reducing the number of partial product rows more efficiently than the Wallace tree; the Wallace tree algorithm, popular for its fast parallel reduction of partial products; Booth's algorithm, which reduces the number of partial products through encoding; and Vedic multiplication, which is based on ancient Indian mathematics and offers advantages in recursive and parallel computation[2].

Among recent innovations, the reduced-sp-D3Lsum adder combined with the Dadda algorithm has demonstrated improved performance in terms of speed and power consumption. Although these designs show promising results under high-frequency operation, there remains significant potential for further reduction in power consumption, particularly when scaling to larger systems where the multiplier constitutes a major component of the datapath or processing core.

In the proposed work, a novel multiplier architecture has been developed using full adders and half adders as the foundational blocks. To minimize power dissipation, the design employs pass transistor logic in conjunction with CMOS process technology. Additionally, the Dadda multiplication algorithm

is utilized to efficiently compress and reduce partial products, thereby minimizing propagation delay and enhancing overall computation speed.

This architecture is designed to meet the dual objectives of low power consumption and high-speed operation, making it well-suited for integration into complex digital systems where power efficiency and rapid response are critical[3].

Multipliers play a crucial role as core components in many high-performance digital systems such as finite impulse response (FIR) filters, microprocessors, and optical signal processors. The efficiency of these systems often hinges on the speed and accuracy of their multiplier units.

One of the foundational techniques in multiplication is arithmetic based on positional notation, which can become increasingly complex in digital implementations, especially when dealing with large bit-width operands or signed numbers. In unsigned multiplication, standard algorithms can be directly applied, as the interpretation of bits is straightforward.

However, signed multiplication introduces additional challenges due to the representation of negative numbers—typically in two's complement form. If the same multiplication method used for unsigned numbers is applied to signed numbers without proper consideration of their format, it leads to incorrect results. The sign bit must be carefully handled, and algorithms must account for its effect on the magnitude of the product. Therefore, specialized techniques such as Booth's algorithm or sign extension methods are commonly used to correctly process signed numbers and ensure accurate results in signed multiplication[4].

Multiplication is a fundamental arithmetic operation that essentially represents the process of adding a number to itself repeatedly, as defined by another number. In formal terms, to compute a product, a number known as the multiplicand is multiplied by another number called the multiplier, indicating how many times the multiplicand is to be added.

At an elementary level, multiplication is often taught using partial products, where the multiplicand is multiplied individually by each digit of the multiplier, starting from the least significant digit (LSD) and moving leftwards. Each resulting partial product is then shifted appropriately to align with the corresponding place value. These intermediate results are finally summed together to obtain the final product[5].

This method, commonly used in base-10 (decimal) arithmetic, is equally applicable in other number systems such as binary (base-2), which is extensively used in digital electronics and computer architecture. Binary multiplication follows the same logical flow, with simplifications due to the binary digits being only 0 and 1.

Figure 1 illustrates the data flow of this basic multiplication approach. Each operation is represented using symbolic notation—such as black dots denoting individual bit-level multiplication steps in binary implementations.

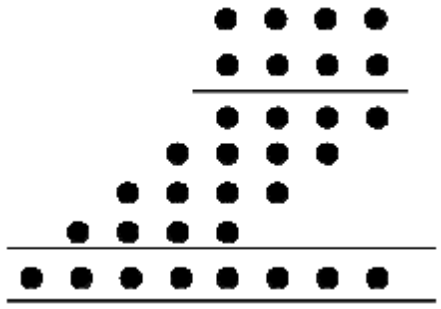


Figure 1: basic Multiplication

In binary multiplication, it is commonly assumed that the most significant bit (MSB) represents the sign bit, particularly when using two's complement representation for signed numbers. In digital systems—especially those optimized for power efficiency and parallel computation—the multiplication process is frequently streamlined through efficient hardware implementations.

The fundamental operations involved in binary multiplication are bitwise addition and left-shift operations, which correspond to partial product generation and alignment. When two binary numbers are multiplied, the procedure involves the iterative addition of shifted versions of the multiplicand, guided by the individual bits of the multiplier.

From this approach, a generalized multiplication algorithm can be derived for both signed and unsigned numbers. In the initial stage, the sign of the final product can be predicted by analyzing the MSBs of the

operands. For instance, when using two's complement representation, the sign of the result is determined by the XOR of the operand signs. After the multiplication is completed, the MSB of the resulting product acts as an indicator of the sign (positive or negative), thereby enabling accurate signed multiplication with appropriate logic.

The most straightforward method of binary multiplication involves performing the operation manually on two binary inputs. This process typically follows three main steps:

1. Partial Product Generation
2. Partial Product Reduction
3. Final Addition

To illustrate, consider the multiplication of two binary numbers: 1101_2 (which represents -3 in two's complement) and 0101_2 (which represents 5 in decimal). The step-by-step multiplication for this example is illustrated in Figure 1.2. In this setup, the multiplicand is often referred to as the generator, while the multiplier is called the accelerator. Each intermediate result, obtained by multiplying the generator by a single bit of the accelerator, is termed a partial product. These partial products are aligned according to their bit position and summed to obtain the final product, which is referred to as the standard result[6]. In educational contexts, intermediate values—often italicized—highlight the carry-forward or partial significance across bit positions during manual calculation.

When translated into hardware, this multiplication methodology is executed through a structured set of operations:

- Partial Product (PP) Generation
- Partial Product Reduction
- Final Summation

These stages are realized using arithmetic logic circuits that manage bit-wise additions and the propagation of carry values. The sum and carry bits are fundamental to constructing high-speed and efficient multipliers within Arithmetic Logic Units (ALUs), Digital Signal Processors (DSPs), and microprocessors.

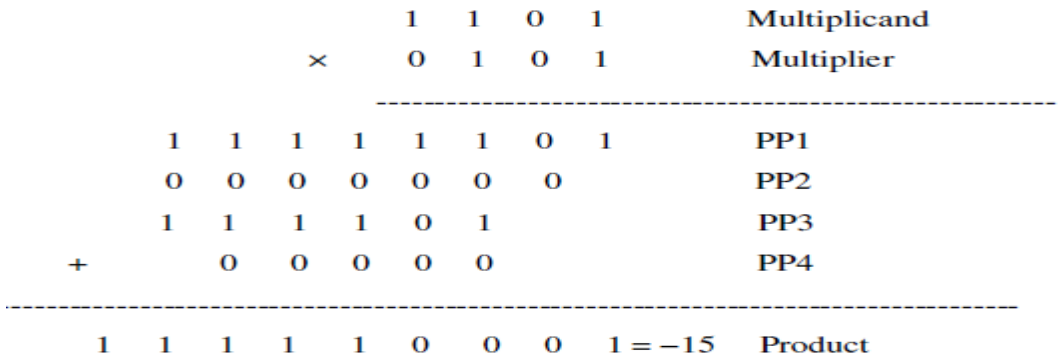


Fig 2. Multiplication process

The core principle of binary multiplication involves handling one bit of the multiplier at a time, proceeding sequentially from the least significant bit (LSB) to the most significant bit (MSB). For each bit of the multiplier, the multiplicand is either added—if the bit is 1—or ignored—if the bit is 0. Each resulting partial product is then shifted one position to the left relative to the previous partial product, aligning it according to the bit's positional weight.

To derive the final product, all partial products are added column-wise. Each column in this summation includes two components:

- Sum bits, representing the direct bit-wise addition at that position
- Carry bits, which are produced during the addition and propagated to the next higher column

This addition process is realized through the use of full adders and half adders, which compute the sum and carry at each bit position. The complexity of the circuit increases with both the number of partial products and the bit-width of the operands.

For the multiplication of an n-bit multiplicand and an m-bit multiplier, the resulting product is (n + m) bits long. This requires generating m partial products, each appropriately shifted. To improve performance, particularly in hardware implementations, these partial products are efficiently combined using reduction structures such as Dadda trees or Wallace trees. These architectures reduce the number of sequential addition stages, thereby lowering the overall propagation delay[7].

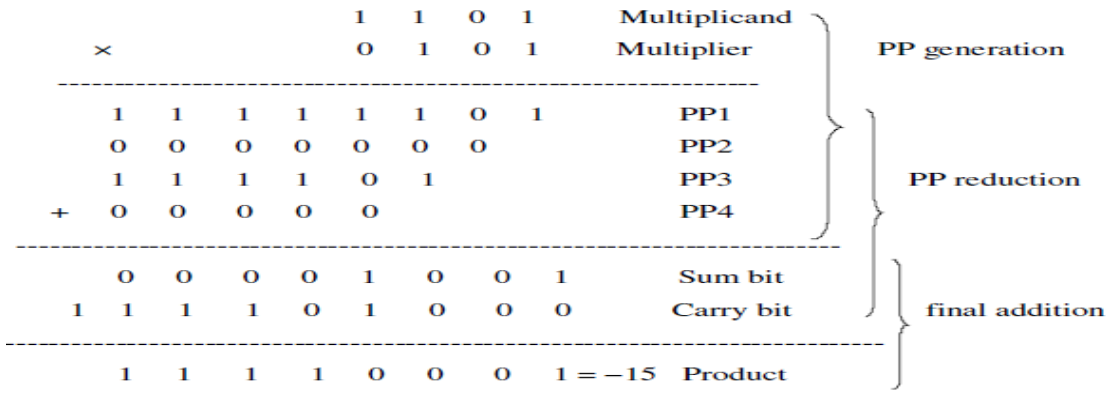


Fig 3: Multiplication showing sum and carry

Multipliers serve as core computational units in high-performance digital systems, such as Finite Impulse Response (FIR) filters, digital signal processors, and embedded control units. Their design significantly affects overall system speed and power consumption, especially in arithmetic-intensive applications like image processing, communications, and artificial intelligence.

Fundamentals of Binary Multiplication

Binary multiplication is essentially a sequence of bit-level operations. For unsigned numbers, each bit of the multiplier determines whether the multiplicand is added to the result, with the final product obtained by summing appropriately shifted partial products. In contrast, signed multiplication, typically using two's complement representation, introduces added complexity. Special handling is required for the sign bit to ensure accurate results. Techniques such as sign extension and Booth's encoding are commonly used for handling signed operands.

Three-Stage Multiplication Model

The multiplication operation is classically divided into three main stages:

Partial Product Generation:

Each bit of the multiplier yields a partial product, which is a shifted version of the multiplicand. For n × m multiplication, there are m partial products.

Partial Product Reduction:

The multiple partial products are combined using structured reduction trees such as **Dadda** or **Wallace trees**, which minimize the number of sequential addition stages and improve speed.

Final Addition:

The reduced partial products are summed using ripple carry adders, carry-lookahead adders, or carry-save adders to produce the final product.

This structured approach supports efficient hardware realization and offers scalability for higher bit-width operations.

Handling Signed Multiplication

In two's complement arithmetic, the most significant bit (MSB) indicates the sign. Signed multiplication must consider MSBs of both operands and apply sign correction logic accordingly. The sign of the result is determined using XOR of the input signs. Proper handling of sign extension ensures correctness of the final output.

Approximate Multiplier Design

Modern AI and multimedia applications can tolerate minimal computational errors, enabling the use of **approximate multipliers** for better performance and energy savings. These architectures reduce hardware complexity and switching activity by allowing small inaccuracies in exchange for substantial reductions in power and area[8].

One such design, the **Dynamic Range-Unbiased Multiplier (DRUM)**, introduces statistical error neutrality, where errors tend to cancel out across operations. This results in:

- **Power savings** of 54% to 80%
- **Gaussian error distribution** centered around zero
- **Configurable accuracy vs. power trade-off**

The DRUM architecture focuses computation on the most significant bits, maintaining acceptable accuracy while optimizing for power-critical applications.

VLSI Architecture and Optimization

VLSI implementation of multipliers must address:

- **Latency minimization**: Using Dadda/Wallace trees
- **Power efficiency**: Employing PTL (Pass Transistor Logic) or SPST (Spurious Power Suppression Technique)
- **Area reduction**: Using multiplexed adders and shared logic
- **Error resilience**: Especially in approximate computing

For instance, Modified Booth Encoding (MBE) and SPST can significantly reduce unnecessary transitions in partial product accumulation, further decreasing dynamic power.

The **array multiplier**, known for its regular layout, is often used as a baseline architecture. In such designs, (N–1) adders are typically required for N-bit multiplication. To improve this baseline, recent studies propose using modified full adders built using multiplexers, yielding:

- **35.45% power reduction**
- **40.75% area savings**
- **15.65% delay improvement**

These advancements make them suitable for real-time and embedded systems with stringent performance constraints[9-10].

PROPOSED SYSTEM

Modified 4×4 Multiplier Design Using 12T Full Adder and Dadda Algorithm

In the proposed system, a modified 4×4 multiplier circuit has been developed with the objective of achieving low power consumption and minimal propagation delay, specifically optimized for VLSI applications. This design makes use of an enhanced 12-transistor (12T) full adder as the primary computational element. The architecture of this full adder integrates pass-transistor logic with standard CMOS process technology to create a hybrid logic structure. This approach significantly contributes to reducing the power-delay product (PDP) and improving overall circuit performance. The full and half adders used in this multiplier exhibit improved switching characteristics, shallower logic depth, and reduced dynamic power dissipation when compared to traditional designs. These features make the proposed system particularly effective for arithmetic circuits in environments where speed and energy efficiency are of paramount importance.

A critical aspect of multiplier design lies in the efficient handling of partial product summation. In a 4×4 multiplication scenario, sixteen partial products are generated through logical AND operations between bits of the multiplicand and the multiplier. These partial products initially form a matrix with a maximum height of four. To minimize the latency introduced during their summation, the Dadda tree algorithm is employed. This algorithm, in contrast to a conventional ripple-carry-based adder which accumulates delay due to sequential carry propagation, utilizes a layered tree structure for summing partial products in a parallel manner. The Dadda method strategically reduces the number of rows in the partial product matrix at each stage by deploying full adders and half adders to compress columns of bits, eventually bringing the matrix down to just two rows that can be summed using a standard carry-propagate adder. This organized reduction process significantly lowers the propagation delay and enhances overall throughput.

The implementation of the Dadda algorithm in this design proceeds in two major stages. In the first stage, the matrix height is reduced from four to three. This is done by identifying those columns with more than three bits and applying full adders to compress them, directing their sum and carry outputs into subsequent lower and upper columns respectively. In the second stage, the matrix is further compressed from three rows to two, again using a combination of full and half adders to ensure that no column contains more than two bits. The remaining two rows are then added using a ripple carry adder. Although ripple carry adders are inherently slower due to their sequential nature, in this architecture, their use is restricted solely to the final summation stage, thereby keeping their delay impact minimal.

The proposed multiplier architecture brings several advantages through this approach. The use of the Dadda algorithm, in combination with the 12T hybrid full adder, ensures minimal propagation delay. The hybrid adder, with its reduced transistor count and fewer switching events, effectively minimizes dynamic power consumption. Furthermore, the design's modularity and reliance on the structured reduction offered by the Dadda algorithm make it highly scalable for implementation in larger bit-width multipliers. In addition, the compact transistor-level design contributes to reduced silicon area and lower capacitive loading, further enhancing power efficiency. Overall, the combination of algorithmic efficiency and optimized circuit design provides a balanced solution for high-speed, low-power multiplication in modern VLSI systems.

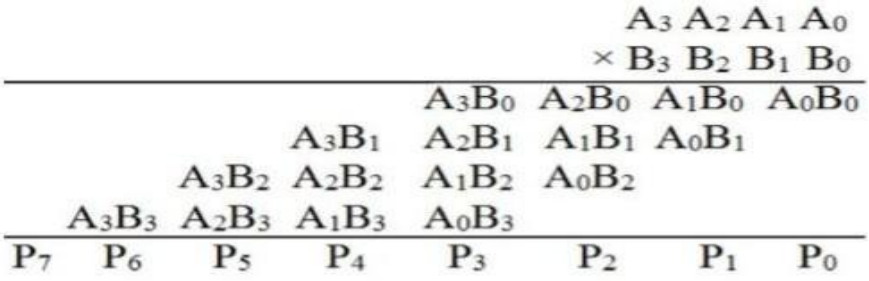


Figure 4: 4 × 4 Multiplications.

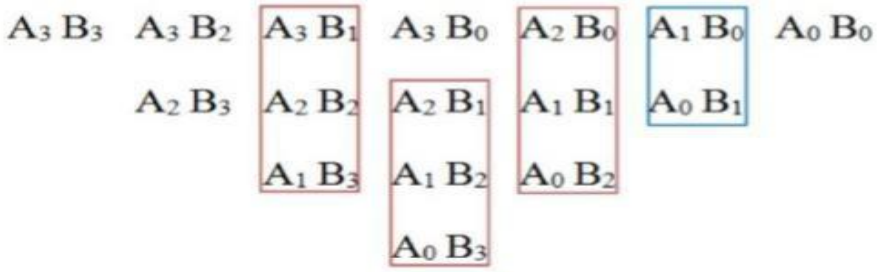


Figure 5: First Dadda Stage

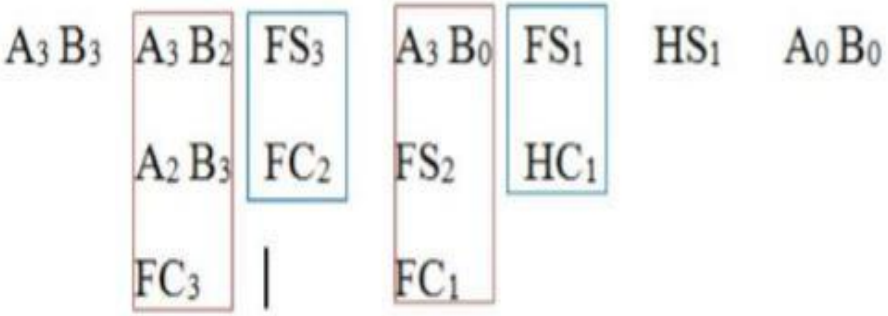


Figure 6: Second Dadda Stage

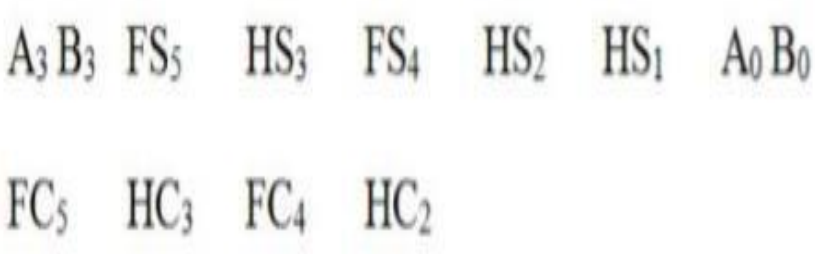


Figure 7: Last Dadda Stage

B. Algorithm Implementation

In the design of a 4×4 multiplier, the multiplication process begins with the generation of a partial product matrix formed by the bitwise AND operation between each bit of the 4-bit multiplicand and the 4-bit multiplier. This operation results in 16 partial products, arranged in a matrix that initially has a maximum height of four. To perform high-speed multiplication and minimize delay, the Dadda tree reduction technique is employed, which systematically reduces the height of this matrix in multiple stages before the final summation is carried out.

The first stage of the Dadda algorithm targets a matrix height of four. At this level, several adders are introduced to begin the reduction process. Specifically, a full adder is deployed in the fourth column, which originally contains four bits, effectively compressing its height. Additionally, a half adder is utilized in the second column, which holds three bits, thereby helping maintain the target height constraint. Furthermore, two more full adders are inserted in the third and fifth columns where necessary, ensuring that no column exceeds a height of three. This stage is executed in full parallelism, allowing all adders

to operate independently without dependencies between columns. This parallel operation is key to the Dadda approach, as it minimizes sequential delays and sets the foundation for further reductions.

Following the initial compression, the second stage commences with the matrix now having a maximum height of three. At this point, full and half adders are again strategically placed to reduce the matrix further. The goal is to bring each column's height down to two, thereby making the data structure suitable for the final summation stage. The adders used in this stage are chosen based on the number of bits present in each column and are positioned to maintain a uniform reduction pattern across the matrix.

In the third and final stage, the matrix reaches a height of two, making it ready for the final addition phase. A ripple carry adder is then used to sum the two remaining rows of partial products. While ripple carry adders inherently involve sequential carry propagation and are thus slower compared to more complex adders, their usage here is limited to the final addition stage, where the number of bits to process is minimal. This allows the design to retain high-speed performance without the need for more complex final adders.

The effectiveness of the Dadda reduction process is partly due to the structured and minimal use of logic elements. The number of half adders (HA) and full adders (FA) required at each stage can be approximated based on the current height (H) of the matrix. These estimations help in allocating resources efficiently and maintaining the desired compression pattern throughout the reduction stages. The total number of logic components is kept minimal while achieving the target height constraints for each stage, thus ensuring optimal hardware utilization.

The overall flow and architecture of the proposed multiplier, including the Dadda reduction stages, are illustrated in the block diagram presented in Figure 5. This diagram captures the sequence of operations from partial product generation through successive reductions to final summation, reflecting the modular and hierarchical nature of the multiplier's architecture.

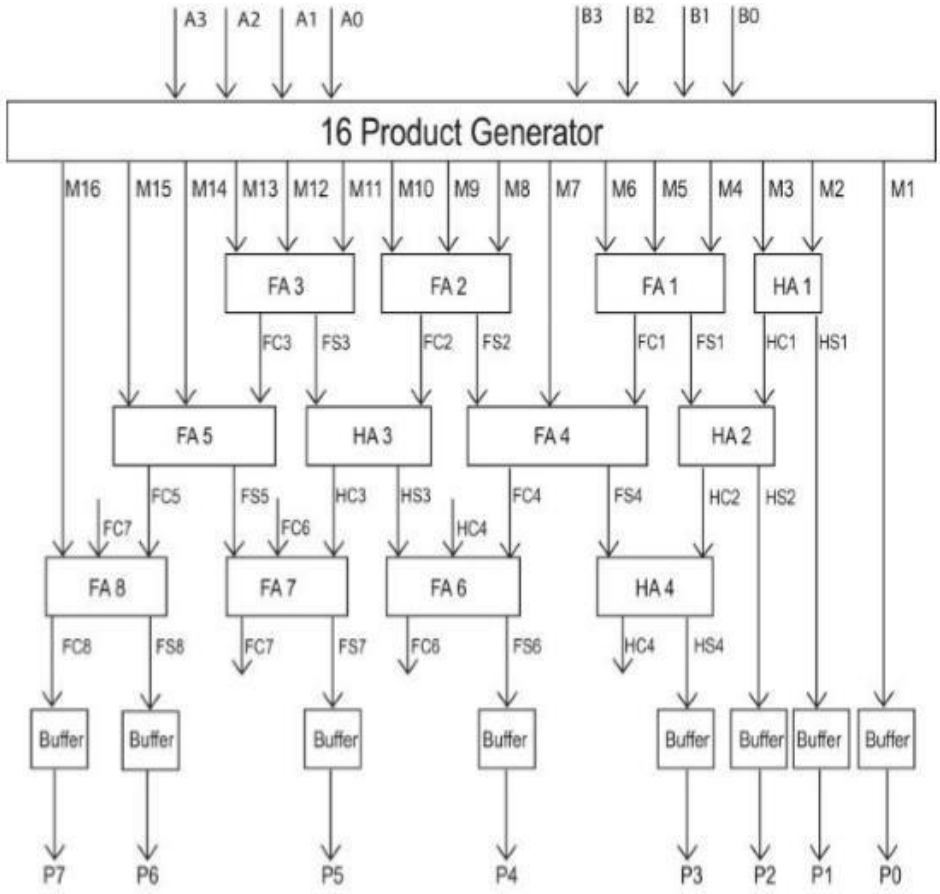


Figure 5: Block Diagram of the Multiplier

Working Stages of the Proposed Model

The operation of the proposed 4×4 Dadda-based multiplier follows a methodically structured sequence designed to optimize both computation speed and power efficiency. The process begins with the generation of partial products, which serves as the foundation for subsequent reduction and summation

stages.

Initially, the 4-bit multiplicand and 4-bit multiplier undergo bitwise AND operations to produce a total of sixteen partial products. These binary products are arranged into a 4×4 matrix, representing all possible pairwise combinations of the input bits. This matrix acts as the primary input to the reduction stages.

The first Dadda reduction stage focuses on compressing the height of this matrix from four rows to three. To achieve this, full adders and half adders are carefully allocated to columns containing more than two bits. Specifically, three full adders and one half adder are used in this stage, strategically placed to reduce the height without introducing sequential dependencies. This parallel arrangement enables high-speed processing, as each adder operates independently within its respective column.

Following the initial compression, the second Dadda reduction stage further condenses the matrix from a height of three to two. This stage employs two full adders and two half adders to streamline the remaining bits. The goal is to prepare the matrix for the final summation while preserving computational integrity and minimizing critical path delay. By the end of this stage, the partial products are reduced to just two rows, making the matrix suitable for final addition.

The final summation is carried out using a ripple carry adder (RCA), which adds the two remaining rows of bits to produce the final product. Although ripple carry adders are not the fastest among available adder architectures, their low power consumption and structural simplicity make them particularly well-suited for small-bit-width multipliers such as the 4×4 configuration in this design.

To ensure the accuracy and reliability of the output, a buffer stage is integrated at the final stage of the circuit. This output buffering serves to stabilize voltage levels, filter transient fluctuations, and maintain signal integrity before the product is delivered to subsequent stages of the system or to external components.

Together, these stages comprise a compact and efficient multiplier architecture that leverages the advantages of the Dadda reduction technique, optimized adder components, and power-aware design principles suitable for VLSI applications.

PROPOSED DADDA MULTIPLIER

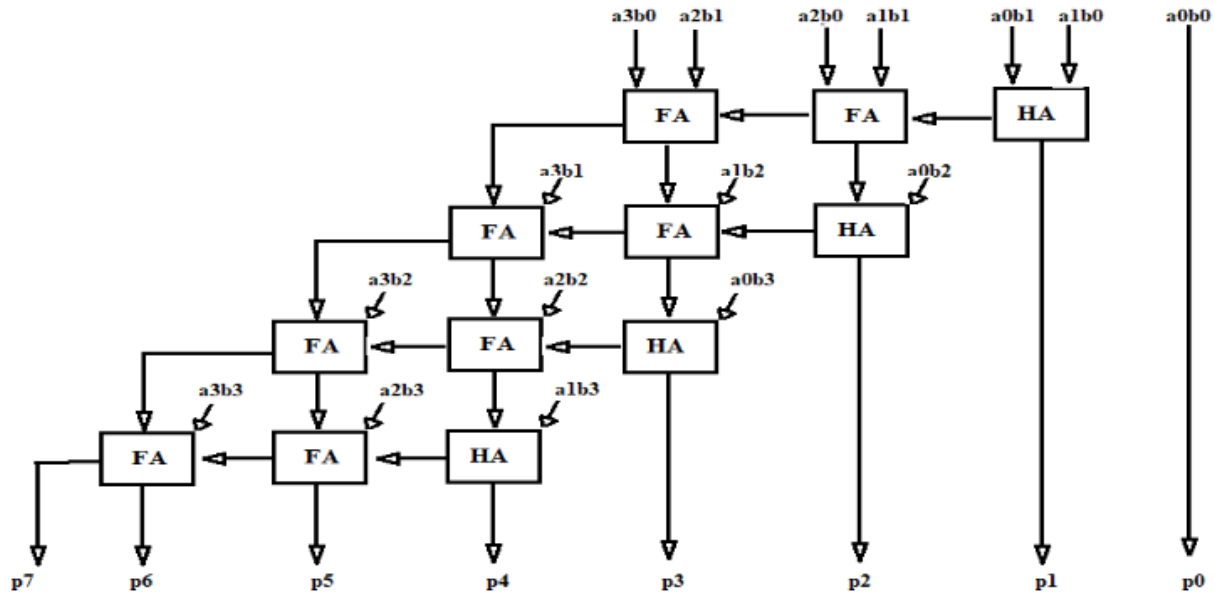


Figure 6: Proposed Multiplier Diagram

A. AND GATE

In the proposed 4×4 multiplier architecture, AND gates are integral to the generation of partial products. Each bit of the 4-bit multiplicand is logically combined with each bit of the 4-bit multiplier using AND operations, resulting in a total of 16 partial products. Accordingly, the design incorporates 16 AND gates to carry out these bitwise multiplications.

These gates are implemented using an optimized low-power schematic that ensures minimal energy consumption while maintaining high-speed operation. Positioned as the first computational layer in the multiplier, their performance directly impacts the overall speed and efficiency of the system.

At an operating frequency of 5 GHz, each AND gate exhibits an average dynamic power consumption of 7.65 μW and a propagation delay of just 0.05 nanoseconds, enabling rapid partial product generation. To evaluate their performance across varying conditions, the Power Delay Product (PDP)—a key figure of merit that balances power and speed—was analyzed and graphically represented. The results confirm the suitability of the design for low-power, high-speed arithmetic applications.

This carefully engineered AND gate configuration forms a reliable and efficient foundation for the partial product generation stage, significantly contributing to the overall performance of the multiplier.

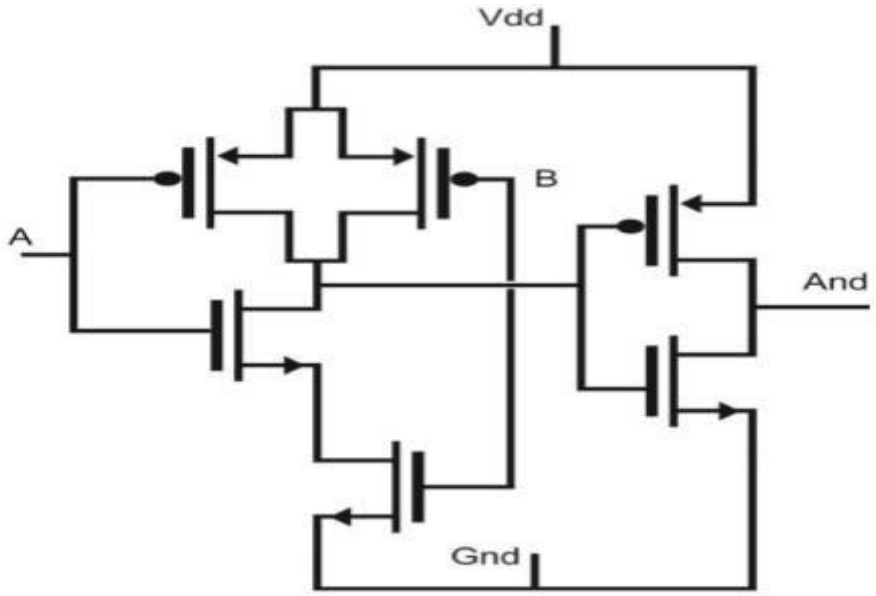


Figure 7: Two input AND gate.

B. Full Adder

The Full Adder serves as a pivotal computational element in the architecture of the proposed 4×4 multiplier. In this design, the Full Adder has been meticulously optimized to minimize propagation delay and dynamic power dissipation—two fundamental factors that determine the efficiency of digital multipliers. Enhancing the Full Adder's operational characteristics directly improves the overall performance of the multiplier, particularly in terms of power efficiency, computational speed, and silicon area utilization.

A key innovation lies in the restructuring of the XOR module within the Full Adder. The new XOR configuration employs only four transistors, resulting in a highly compact and energy-efficient design. Departing from conventional layouts, where the PMOS transistor source is typically tied to a constant voltage supply, the proposed design connects the PMOS source to the first input signal. Simultaneously, the NMOS source is linked to the inverted version of this input, while the second input is applied to the gate terminals of both transistors. This complementary switching strategy enables accurate XOR functionality with substantially reduced power dissipation.

This architectural refinement leads to a significant reduction in transistor count, a lower power footprint, and enhanced switching speed—all of which contribute to minimizing the critical path delay of the overall multiplier. By incorporating this modified Full Adder into the Dadda tree reduction stages of the multiplier, the proposed system achieves superior computational efficiency and a lower power-delay product (PDP), making it highly suitable for power-constrained, high-performance VLSI arithmetic applications.

$$XOR = A'B + AB'$$

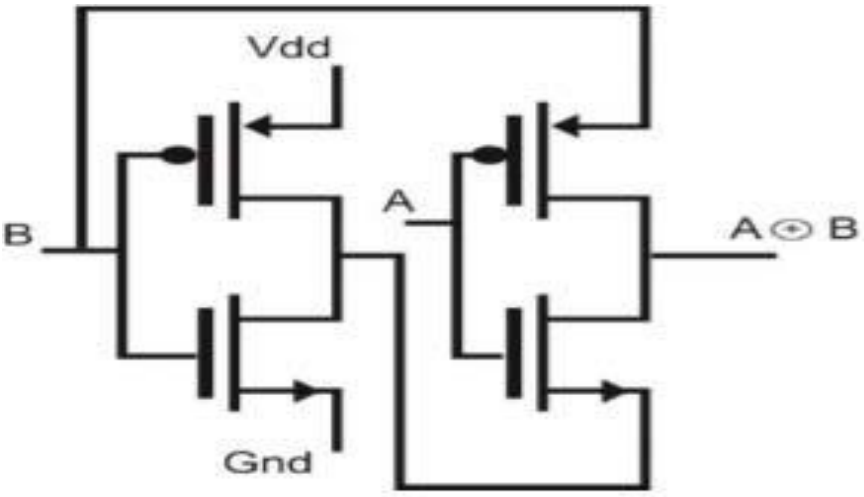


Figure 8: Proposed XOR Module

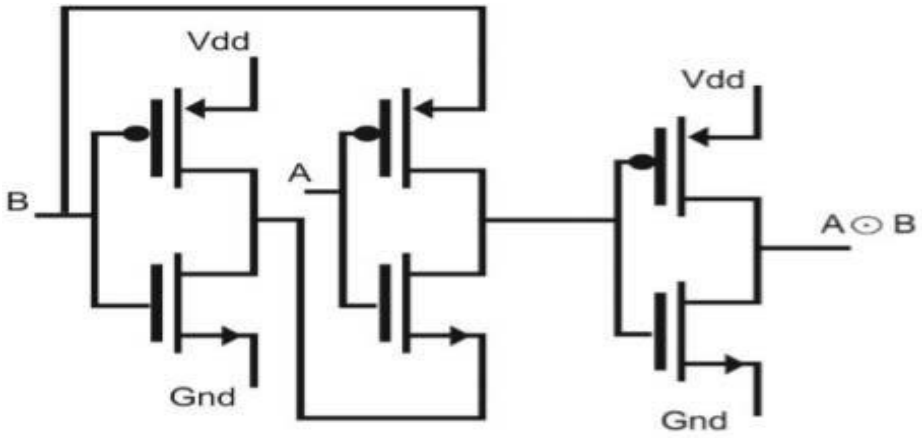


Figure 9: XNOR Module

C. XNOR Module

The XNOR module serves as a complementary logic element to the XOR module and holds considerable significance in arithmetic and control logic design. Within the proposed 4×4 multiplier architecture, the XNOR function is efficiently derived from the existing XOR circuit. Rather than constructing an independent logic block, the XNOR output is generated by appending a single inverter to the output of the optimized XOR module previously designed.

This design choice offers notable advantages in terms of circuit compactness and efficiency. By leveraging the same underlying XOR architecture, the implementation avoids redundant logic, thereby minimizing additional power consumption and silicon area overhead. Furthermore, the XNOR module inherits the high-speed and low-power characteristics of the XOR logic, ensuring consistent performance across the multiplier architecture.

The inclusion of the XNOR module is particularly valuable in operations involving bit-level comparisons, parity checks, and inversion control within the multiplier's internal stages or in subsequent processing blocks. Its shared logic foundation with the XOR circuit also facilitates simplified routing and reduced transistor complexity, contributing to an overall streamlined and area-efficient VLSI layout.

$$XNOR = AB + A'B'$$

D. Full Adder – Sum and Carry Generation

In the proposed full adder architecture, the logic design has been meticulously optimized to minimize both propagation delay and dynamic power consumption. A key enhancement involves the strategic inversion of the XOR output, which facilitates the construction of a secondary XOR stage essential for generating the sum output. Specifically, this stage integrates the outputs of the primary XOR and XNOR modules to compute the final sum bit with improved switching characteristics and a reduced transistor footprint.

In this implementation, the source terminal of the PMOS transistor is connected to the output of the XOR module, while the NMOS source is linked to the output of the XNOR module. This arrangement ensures that the logic conditions required for accurate sum generation are met with minimal complexity and improved speed.

For carry output generation, the design employs a pass-transistor logic configuration comprising two transistors. These are selectively driven by inputs B and C, enabling the circuit to route the carry signal through the most time-efficient path based on the input conditions. This approach significantly reduces signal propagation delay compared to traditional CMOS-based full adder designs, where redundant logic paths often degrade performance.

The schematic diagram of the proposed full adder is illustrated in Fig. 10, detailing the compact and efficient arrangement of transistors. Furthermore, the behavioral performance of the full adder across various operating frequencies is presented in Table 2. At an operating frequency of 3 GHz, the full adder achieves an average dynamic power consumption of 17.5 μW and a propagation delay of merely 0.02 nanoseconds. These metrics underscore the circuit's suitability for integration into high-speed, low-power arithmetic blocks in modern VLSI systems.

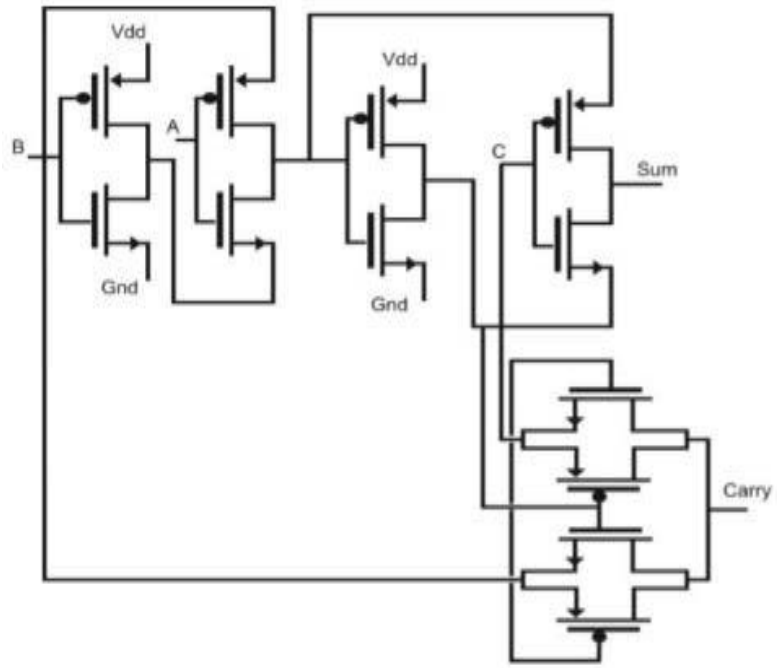


Figure 10: Proposed Full Adder

E. Half Adder

The half adder serves as a fundamental logic unit within the proposed 4×4 multiplier architecture, playing a critical role in the efficient accumulation of partial products during the Dadda reduction process. In this design, a total of four half adders are employed at specific positions where only two bits need to be summed. This selective usage minimizes circuit complexity and transistor count, thus achieving reductions in both area and power consumption while maintaining computational accuracy.

The schematic of the proposed half adder, as illustrated in Fig. 10, reveals a streamlined transistor arrangement designed to support low-power operation and shallow logic depth. This configuration enhances switching efficiency and contributes to the overall speed of the multiplier.

Comprehensive performance analysis of the half adder under varying operating frequencies demonstrated its effectiveness in high-speed arithmetic operations. At a frequency of 3 GHz, the half adder exhibited minimal dynamic power consumption and an extremely low propagation delay, validating its suitability for energy-efficient VLSI implementations. Additionally, the variation in the power-delay product (PDP) across different frequencies, as depicted in Fig. 13, confirms the robustness and consistency of the design. These characteristics establish the half adder as a key component in achieving the proposed multiplier's low-power, high-performance goals.

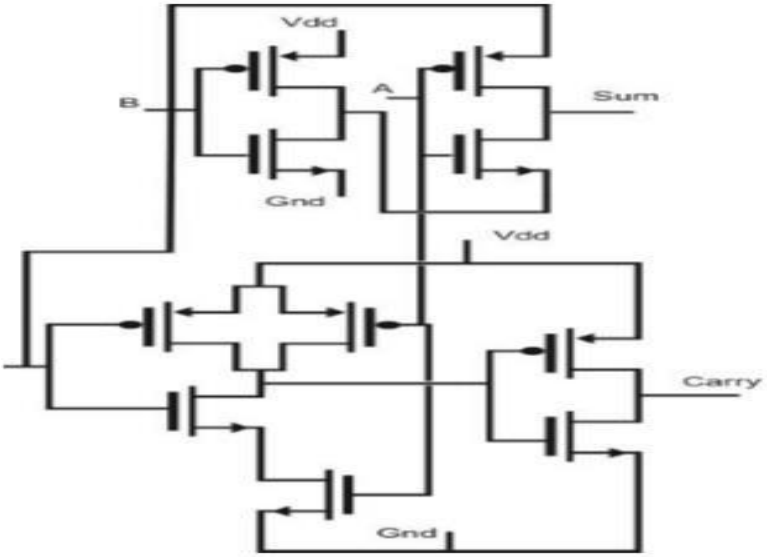


Figure 11: Half Adder

f. Buffer

In digital circuit design, the buffer serves as a critical intermediary component, particularly within high-speed arithmetic units such as multipliers. It functions primarily as a unity-gain amplifier—preserving the voltage level of the input signal while significantly enhancing its current-driving capability. This characteristic is essential in preventing signal degradation, especially when the circuit must interface with stages that present low input impedance or capacitive loading. Within the proposed 4×4 multiplier architecture, a voltage buffer is strategically placed at the final output stage to ensure signal integrity. It enables the smooth and reliable delivery of output signals by mitigating voltage drops and preserving signal strength across interconnects and downstream logic. This is particularly vital when the multiplier is integrated into larger digital systems where the output may drive complex modules or external interfaces. Current buffers, in contrast, are typically employed where current fidelity must be preserved while interfacing with high-impedance loads. While not explicitly used in this design, understanding their role provides a broader perspective on signal interfacing in analog-mixed signal systems. The voltage buffer in this multiplier operates effectively at a frequency of 3.84 GHz, ensuring minimal dynamic power overhead and negligible delay contribution. Its integration contributes to robust output stability and enhances the overall performance of the design. By incorporating a buffer stage, the proposed multiplier maintains not only computational accuracy but also superior electrical performance, making it well-suited for deployment in high-speed, low-power VLSI environments.

SIMULATION RESULTS

Figures 12 to 24 collectively illustrate the simulated and synthesized performance, layout architecture, and key component behaviors of the proposed 4×4 Dadda-based multiplier design. Figure 12 showcases the transistor-level implementation of the two-input AND gate, emphasizing its compactness and low power dissipation. Figure 13 presents the Full Adder schematic, highlighting the efficient transistor arrangement used to optimize switching speed and area. Figure 14 depicts the Half Adder design, illustrating its minimalistic structure tailored for summing two-bit values within the Dadda tree. Figure 15 outlines the propagation function that governs timing behavior across logic stages. Figures 16 and 17 detail the XNOR and XOR logic modules, respectively, each derived with a reduced transistor count to minimize power while maintaining high-speed operation. Figure 18 illustrates the overall functional layout of the multiplier, integrating all key logic units. Figure 19 provides the physical layout of the multiplier design, as implemented in the Microwind tool, showing how various modules are placed and interconnected in silicon. Figure 20 gives the RTL schematic view generated by synthesis tools, offering a structural overview of logic gates and connections. Figure 21 presents the timing analysis results, confirming the operational frequency and setup/hold parameters. Figure 22 contains simulation waveforms verifying correct multiplication outputs under test conditions. Figure 23 captures RTL component usage statistics, quantifying logic elements consumed during synthesis. Finally, Figure 24 displays waveform outputs obtained in Microwind, along with power estimation results, which affirm

the low dynamic power consumption of the design. Together, these figures offer a complete visual and analytical validation of the multiplier's performance in both simulation and physical implementation environments.

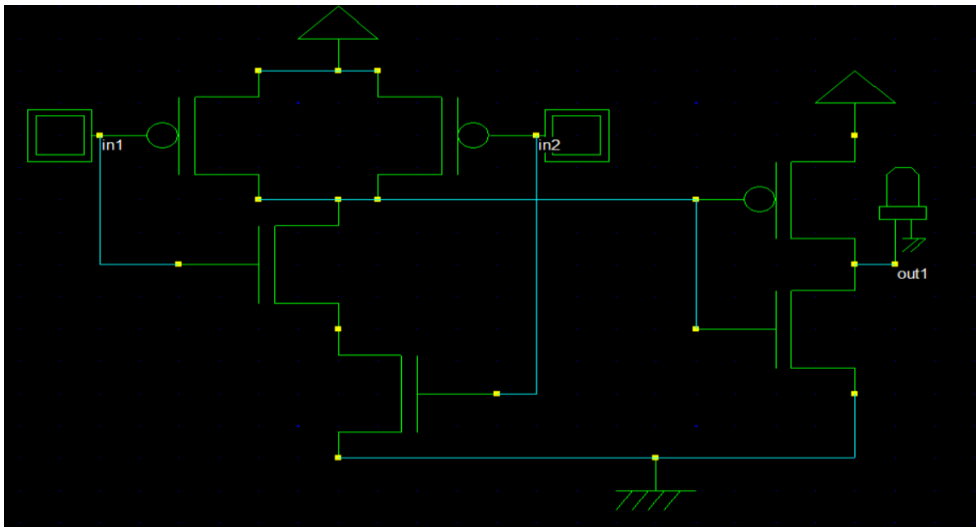


Figure 12: AND Gate

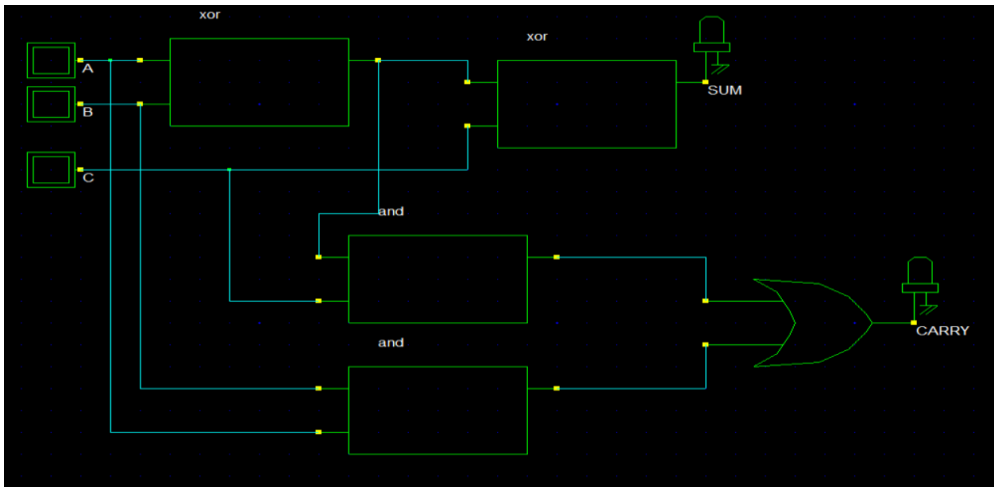


Figure 13: FULL ADDER

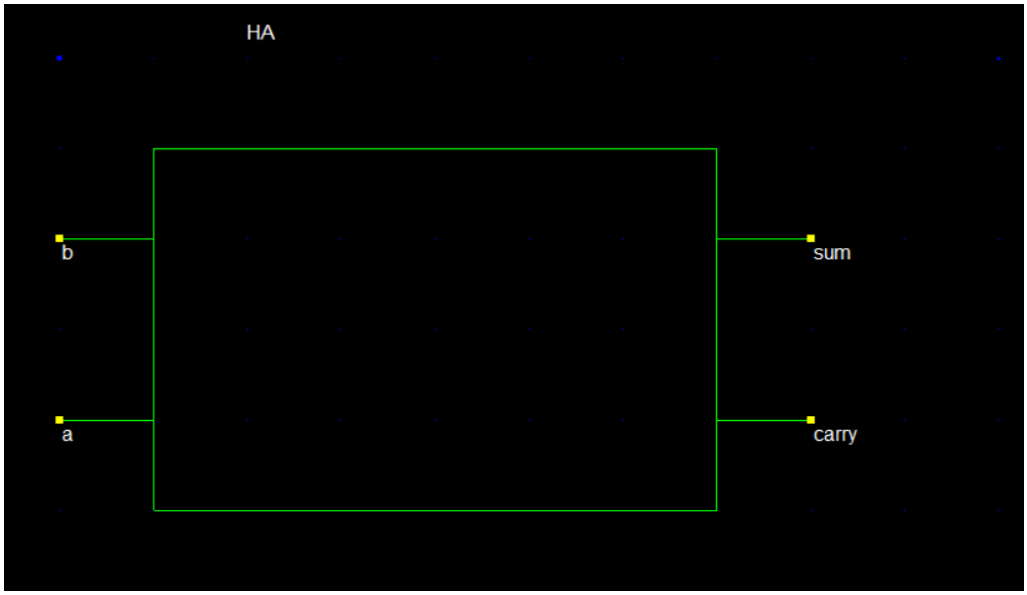


Figure 14: Half Adder

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

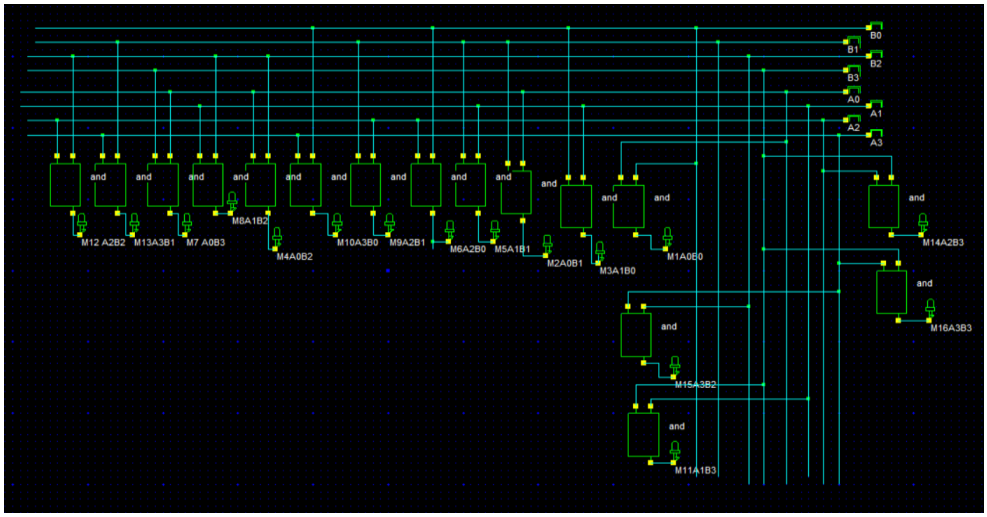


Figure 15: Propagation Function

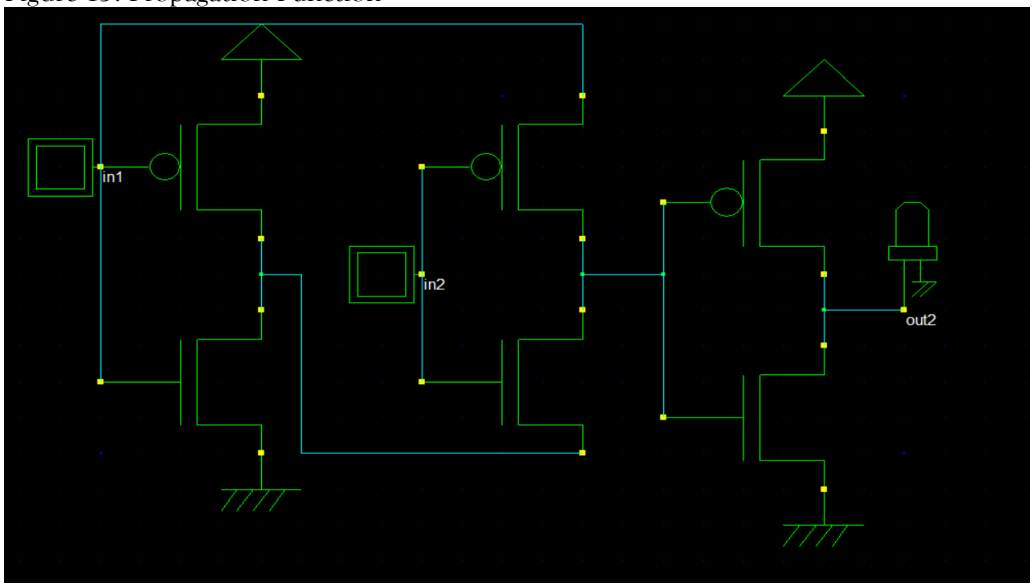


Figure 16: XNOR

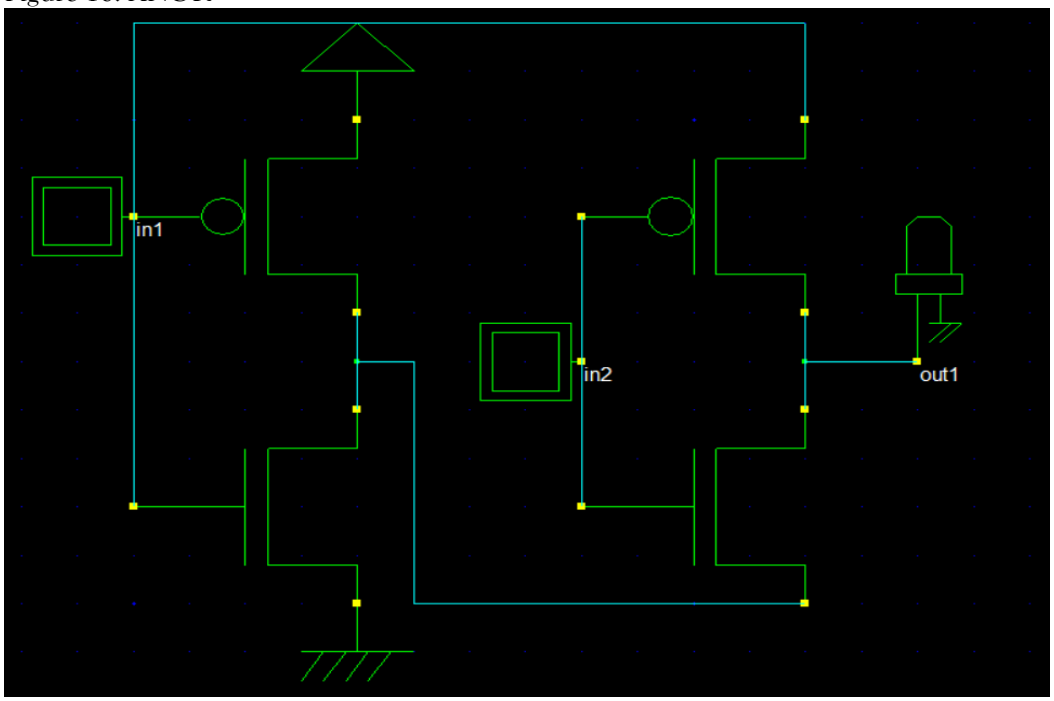


Figure 17: XOR

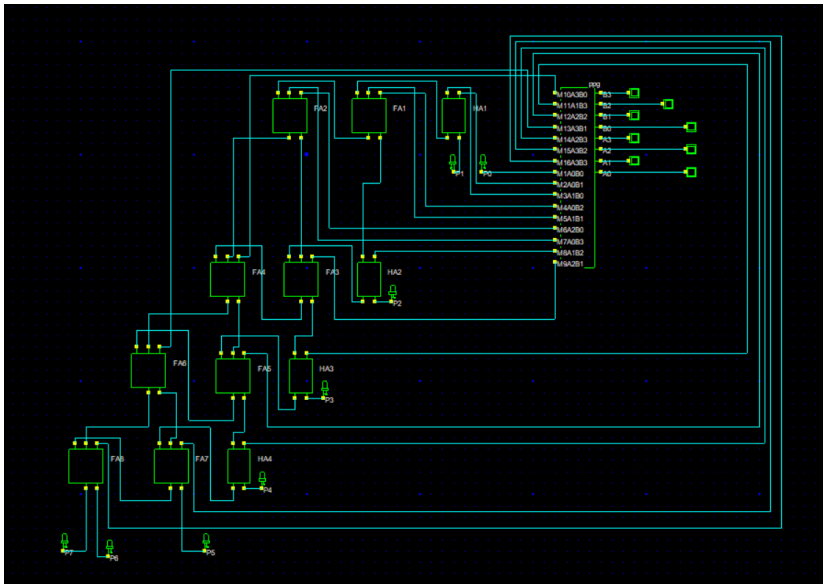


Figure 18: Multiplier

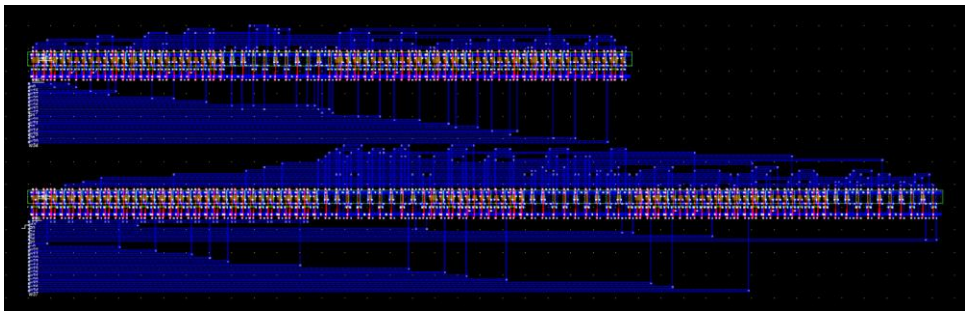


Figure 19: Multiplier Layout Diagram

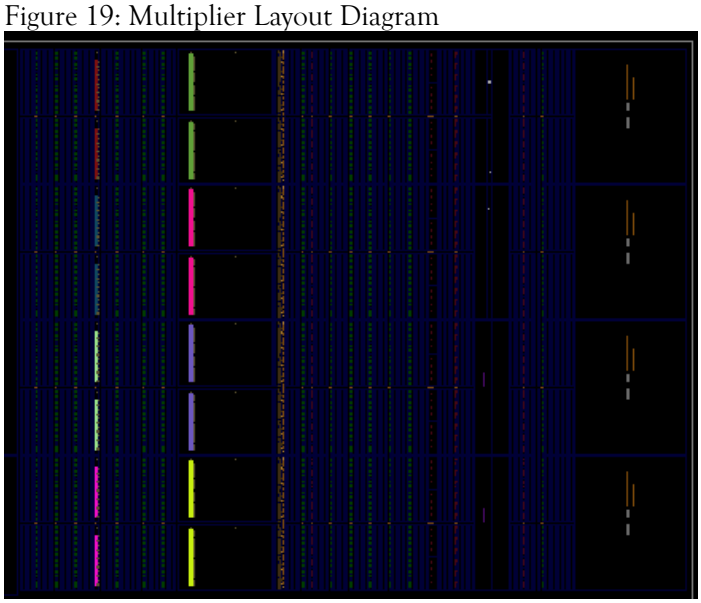
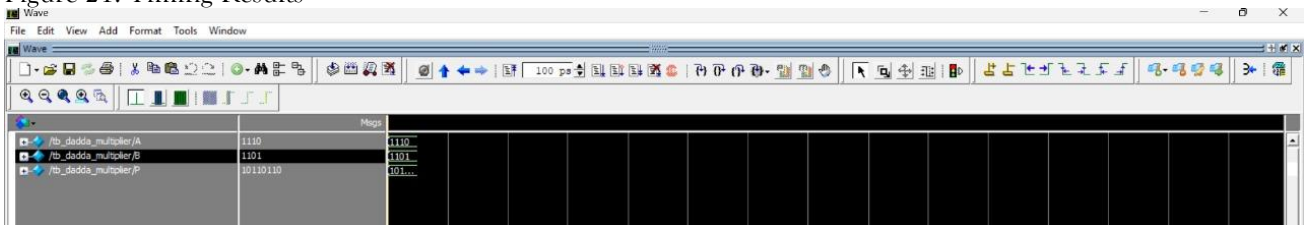


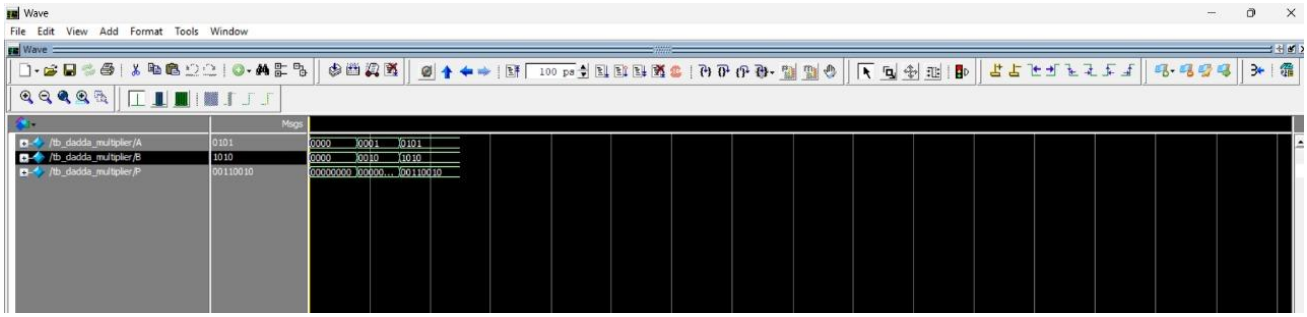
Figure 20: RTL Schematic

```
----------------------------------------------------------------------------
Starting Synthesize : Time (s): cpu = 00:00:06 ; elapsed = 00:00:09 .
```

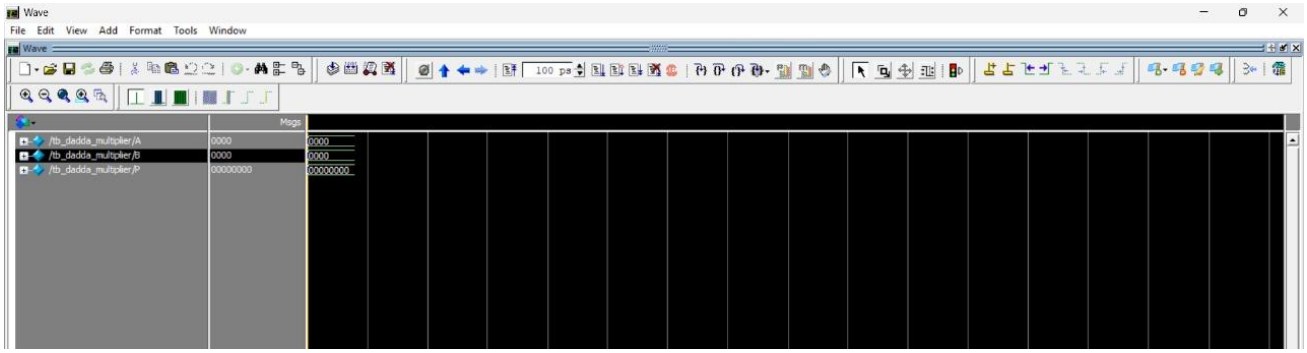
Figure 21: Timing Results



(a)



(b)



(c)

Figure 22: Waveforms-Multiplier

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

Figure 23: RTL Component Usage

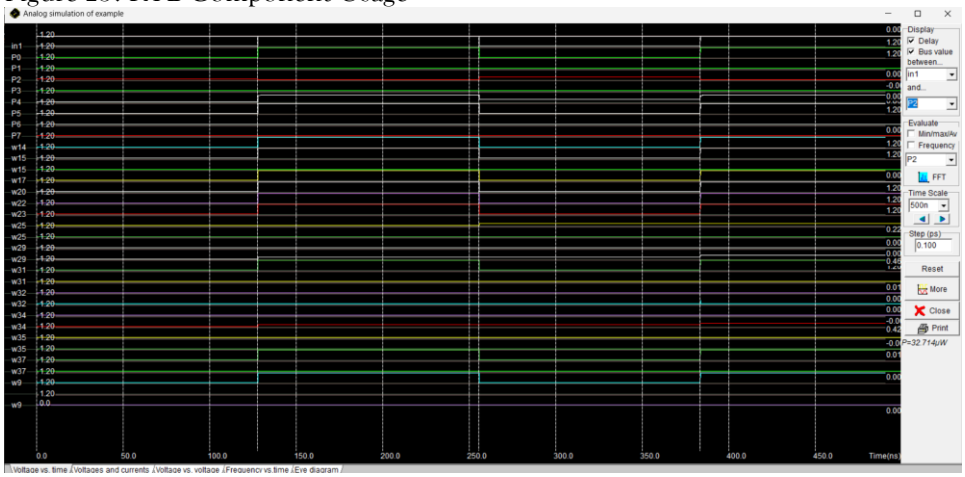


Figure 24: Waveforms-Multiplier in Microwind and Power results

CONCLUSION

The proposed 4-bit multiplier model presents a highly efficient design that achieves an optimal balance between low power consumption and high-speed performance. Utilizing the Dadda reduction algorithm, the architecture effectively minimizes propagation delay, making it well-suited for high-speed arithmetic applications. The fundamental building blocks—optimized full and half adders—are realized using pass-transistor logic combined with CMOS process technology, resulting in reduced power dissipation and minimal delay. Implemented using 90-nm technology within the Cadence Virtuoso environment, the design operates reliably at a frequency of 3.83 GHz and exhibits an average dynamic power consumption of just 184.3 µW at a supply voltage of 1V. These results affirm the design's effectiveness in addressing the critical demands of modern low-power, high-performance VLSI systems.

REFERENCES

[1]. B. Ramkumar and H.M. Kittur, "Low Power and Area Efficient Carry Select Adder," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, no. 2, pp. 371–375, Feb. 2012.
[2]. M. Alioto and G. Palumbo, "Analysis and Comparison on Full Adder Block in Submicron Technology," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 10, no. 6, pp. 806–823, Dec. 2002.
[3]. S. Radhakrishnan and R. P. Sharma, "Design and Analysis of Low Power High-Speed 1-Bit Full Adder Cell," IEEE Conference on Emerging Devices and Smart Systems (ICEDSS), pp. 32–36, 2019.
[4]. N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed., Pearson Education, 2011.
[5]. C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
[6]. L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349–356, Mar. 1965.
[7]. A. M. Shams and M. A. Bayoumi, "A Novel High-Performance CMOS 1-Bit Full-Adder Cell," IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 47, no. 5, pp. 478–481, May 2000.
[8]. D. Radhakrishnan, "Low-voltage low-power CMOS full adder," IEE Proceedings - Circuits, Devices and Systems, vol. 148, no. 1, pp. 19–24, Feb. 2001.
[9]. A. Nagaria and K. S. Rana, "Delay and Area Efficient Carry Select Adder," International Journal of VLSI Design & Communication Systems (VLSICS), vol. 2, no. 5, pp. 47–59, Oct. 2011.
[10]. J. M. Rabaey, A. Chandrakasan, and B. Nikolic, Digital Integrated Circuits: A Design Perspective, 2nd ed., Pearson Education, 2003.