Bu-Ali Sina University
Department of Computer Engineering

Muplitlier
“Impacts of approximate computing in logic design for energy efficiency”

Digital Logic Design
Final project
Dr.Abdoli
Fall 2024

Students: Reza Namvaran, Mahdi Sadeghi

Student No: 40212358042, 40212358026

Introduction:

This project report focuses on designing multipliers using approximate computing techniques. As the demand for faster and more efficient computing grows, approximate computing offers a way to reduce power consumption and increase speed by allowing for some loss of accuracy. In this report, we will discuss the design process, key considerations, and performance evaluations of the approximate multipliers we developed. Our goal is to demonstrate how these multipliers can effectively balance efficiency and accuracy for various applications.

In this project we used VHDL as our hardware description language alongside GHDL & GTKwave for simulating and testing our designs.

Development process is available on GitHub:

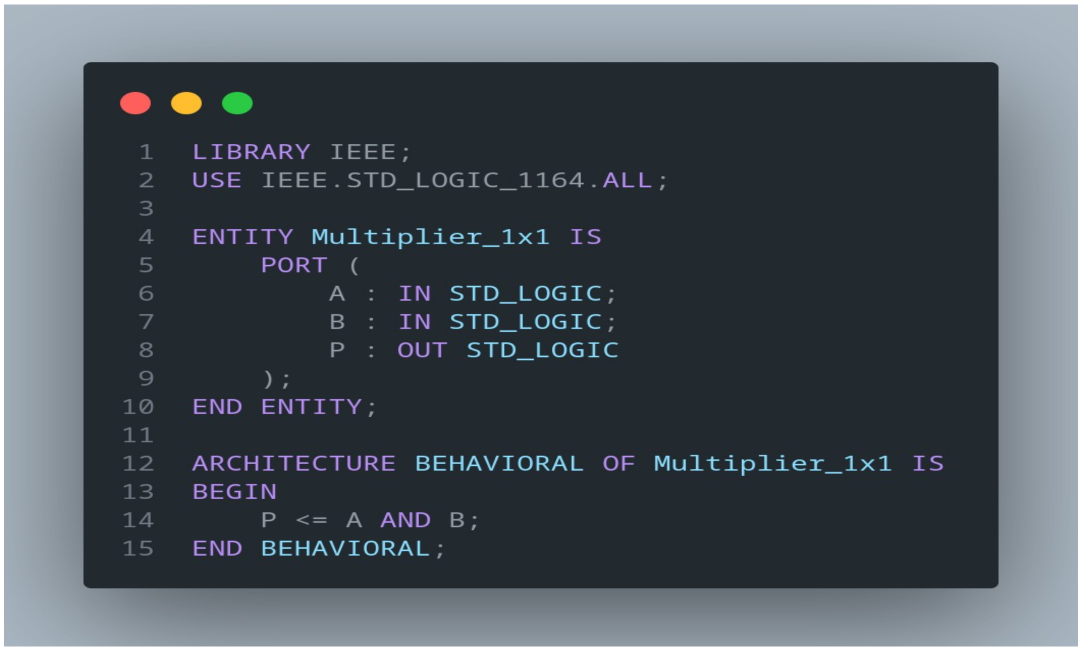
-- https://github.com/Reza-namvaran/Muplitlier

Description of each module in each step:

Step 1:

. 1x1 Multiplier:

The Multiplier_1x1 entity is designed to multiply two single-bit binary inputs, A and B, producing a single-bit output P.

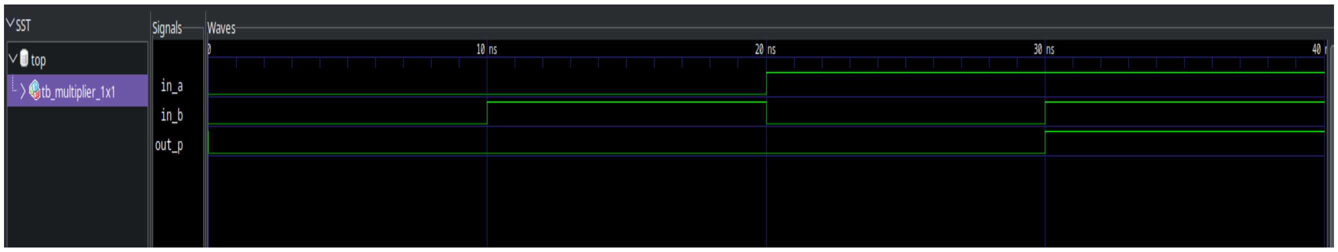


```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Multiplier_1x1 IS
    PORT (
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        P : OUT STD_LOGIC
    );
END ENTITY;

ARCHITECTURE BEHAVIORAL OF Multiplier_1x1 IS
BEGIN
    P <= A AND B;
END BEHAVIORAL;
```

. Waveform results:

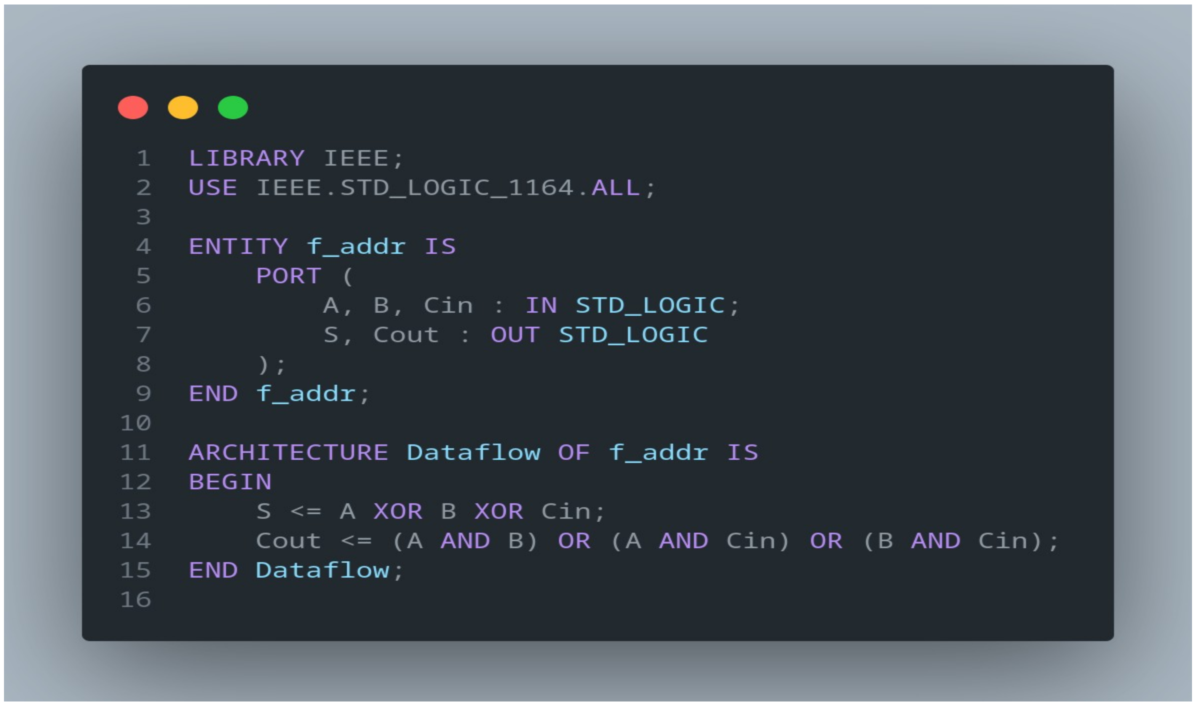


Step 2:

. Full Adder:

The f_addr entity implements a full adder that takes three single-bit binary inputs: A, B, and Cin (carry-in). It produces two outputs: S (sum) and Cout (carry-out).

Full adders used in 4x4 multipliers to calculate sum of the partial products.



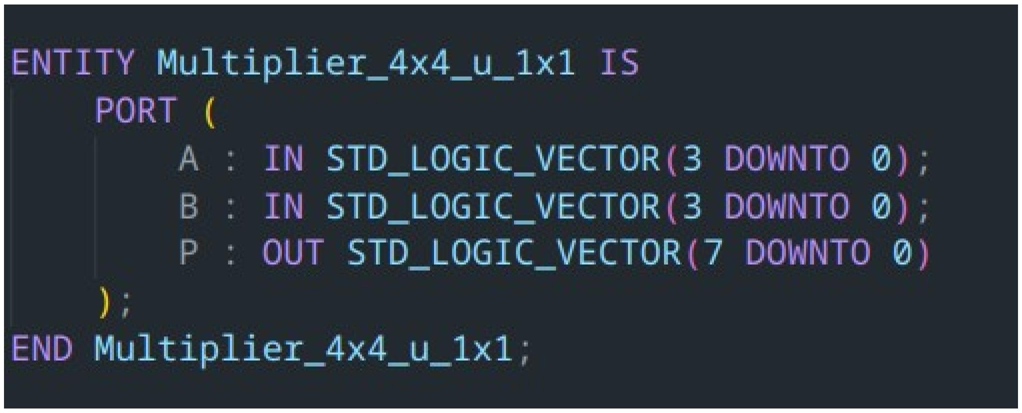
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY f_addr IS
    PORT (
        A, B, Cin : IN STD_LOGIC;
        S, Cout : OUT STD_LOGIC
    );
END f_addr;

ARCHITECTURE Dataflow OF f_addr IS
BEGIN
    S <= A XOR B XOR Cin;
    Cout <= (A AND B) OR (A AND Cin) OR (B AND Cin);
END Dataflow;
```

. 4x4 multiplier using 1x1 unit:

The Multiplier_4x4_u_1x1 entity is designed to multiply two 4-bit binary inputs, A and B, producing an 8-bit output P. This design utilizes smaller components, specifically a 1x1 multiplier and a full adder, to achieve the multiplication.



```vhdl
ENTITY Multiplier_4x4_u_1x1 IS
    PORT (
        A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        P : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END Multiplier_4x4_u_1x1;
```

Partial Product handling:

A 2D array PP of type pp_matrix is defined to hold the partial products generated by multiplying each bit of A with each bit of B. The Gen_PP_Rows and Gen_PP_Cols loops instantiate the Multiplier_1x1 component to fill this matrix.

Addition stages:

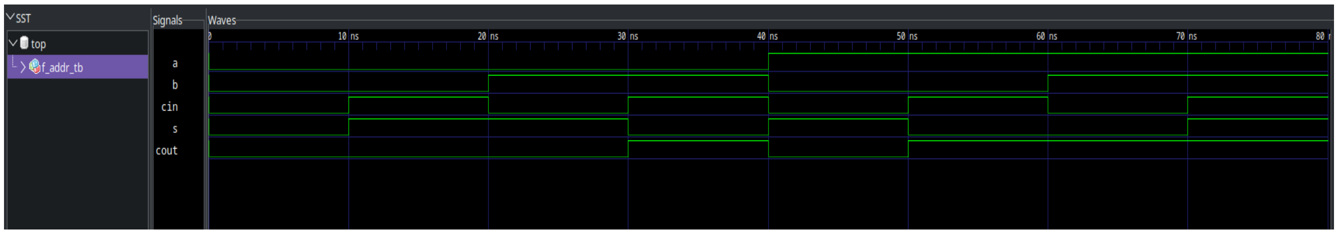
-- The design uses multiple instances of the f_addr component to sum the partial products. The first set of full adders (FA1) combines the first row of partial products, producing intermediate sums and carry signals.

-- The second set of full adders (FA2) processes the next row of partial products, incorporating the carry from the previous stage.

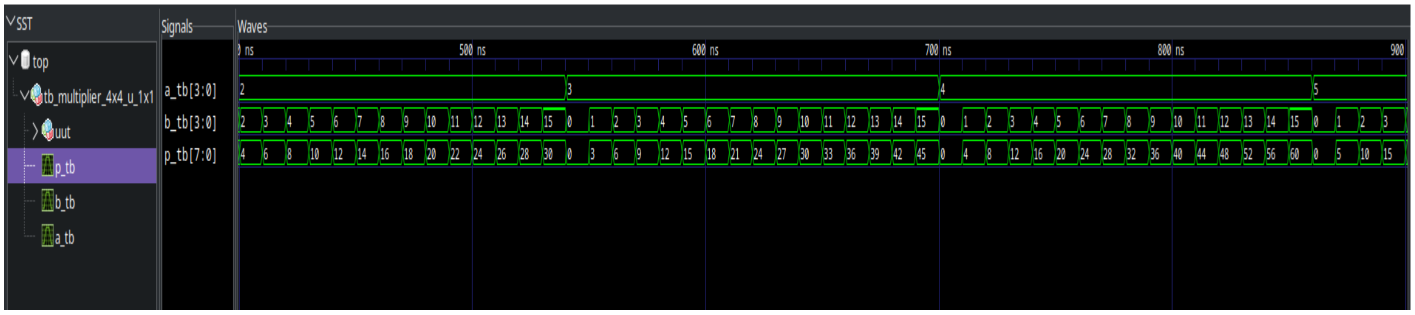
-- The third set of full adders (FA3) completes the addition of the remaining partial products and carries, producing the final output P.

. Waveform results:

. f_addr:



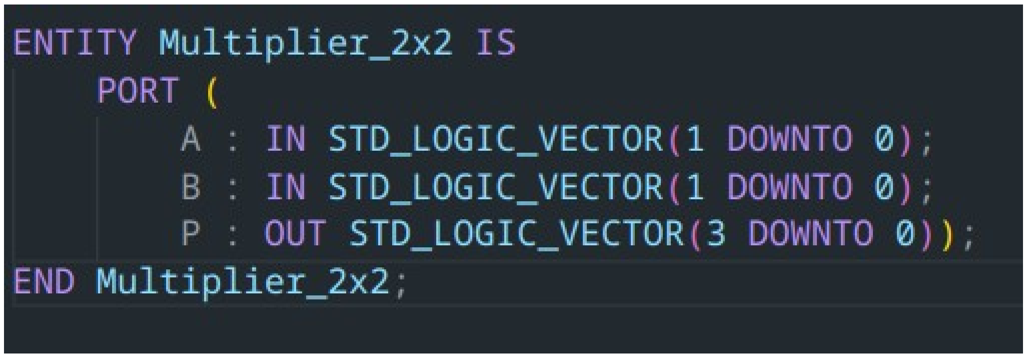
. Multiplier_4x4_u_1x1:



Step 3:

. Multiplier 2x2:

The Multiplier_2x2 entity is designed to multiply two 2-bit binary inputs, A and B, producing a 4-bit output P.



```
ENTITY Multiplier_2x2 IS
    PORT (
        A : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        P : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END Multiplier_2x2;
```

. Partial Products:

The circuit calculates four partial products based on the inputs:
-- p0 = A(0) AND B(0)
-- p1 = A(0) AND B(1)
-- p2 = A(1) AND B(0)
-- p3 = A(1) AND B(1)

These partial products represent the contributions of each bit of the inputs to the final product.

. Addition stages:

Two instances of the f_addr component (full adder) are used to sum the partial products:

-- The first full adder (FA1) adds p1 and p2, producing a sum (sum1) and a carry (carry1).

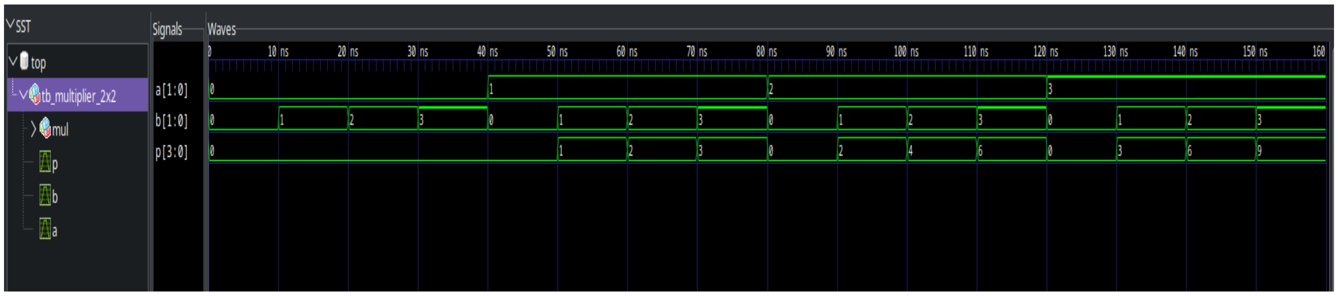
-- The second full adder (FA2) adds the carry from the first adder and p3, producing another sum (sum2) and a carry (carry2).

. Final Product:

The output P is assigned as follows:

P(0) = p0 (the least significant bit of the product)
P(1) = sum1 (the result of the first addition)
P(2) = sum2 (the result of the second addition)
P(3) = carry2 (the most significant bit of the product)

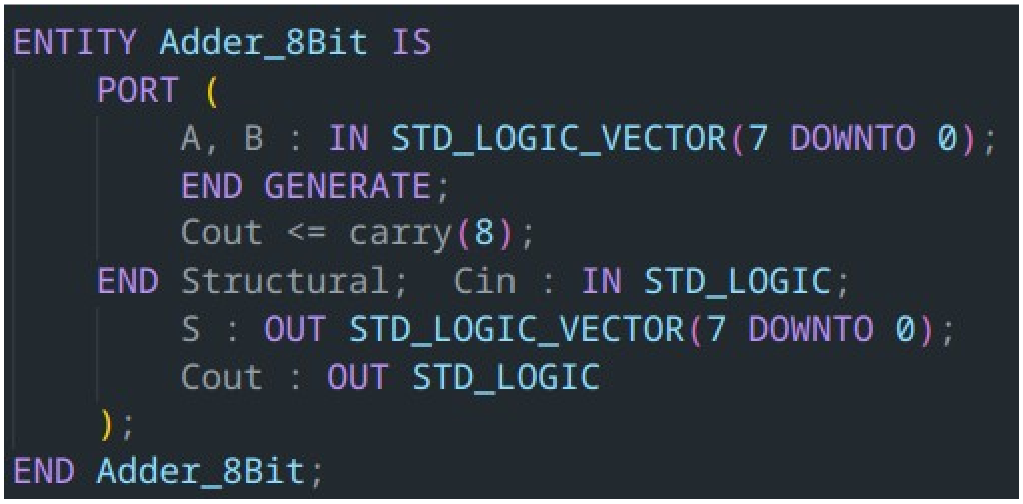
. Waveform results:



Step 4:

. 8 bit adder:

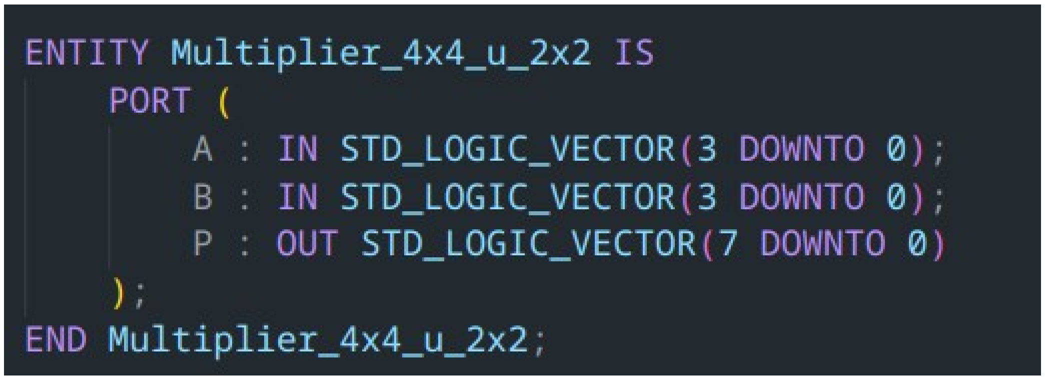
This module adds two 8-bit numbers together, along with a carry-in bit, to produce an 8-bit sum and a carry-out bit. It chains full adders to calculate the result of 4x4 multiplication.



```
ENTITY Adder_8Bit IS
    PORT (
        A, B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        END GENERATE;
        Cout <= carry(8);
    END Structural;  Cin : IN STD_LOGIC;
        S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Cout : OUT STD_LOGIC
    );
END Adder_8Bit;
```

. 4x4 Multiplier module:

The Multiplier_4x4_u_2x2 module implements a 4-bit by 4-bit multiplication using four instances of a 2-bit by 2-bit multiplier. It takes two 4-bit inputs, A and B, and produces an 8-bit output P, which is the product of the two inputs.



```
ENTITY Multiplier_4x4_u_2x2 IS
    PORT (
        A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        P : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END Multiplier_4x4_u_2x2;
```

. Partial Products handling:

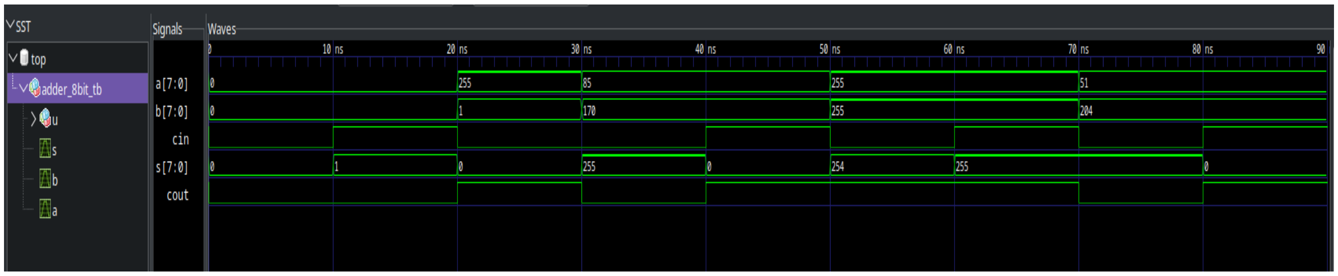
Partial Products: Each 2x2 multiplier generates a 2-bit partial product. The outputs of these multipliers are then aligned to form 8-bit partial products (P0_8bit, P1_8bit, P2_8bit, P3_8bit) by shifting them appropriately.

. Addition Stage:

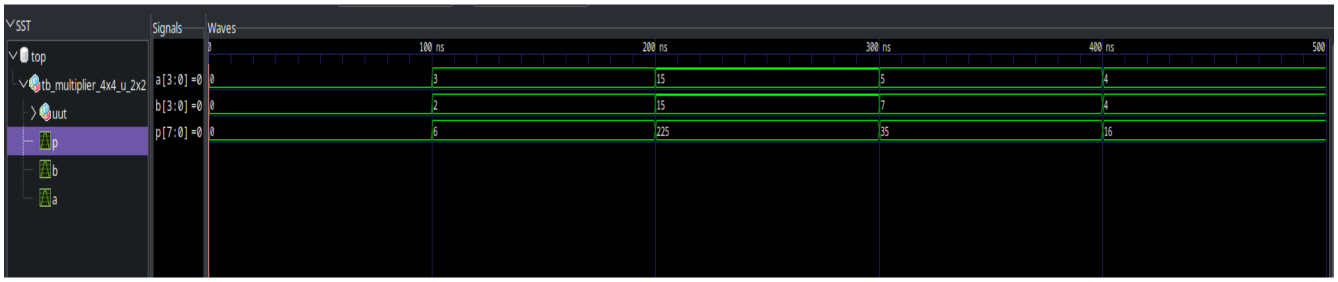
The module uses the previously defined adder_8bit to sum the partial products. Three instances of the adder are used to combine the partial products into the final product P.

. Waveform results:

. 8 bit adder:



. 4x4 Multiplier using 2x2:



Step 5:

Approximation Techniques for Power-Efficient Digital Circuits

Abstract

The growing demand for energy-efficient computing has driven the development of approximation techniques aimed at reducing power consumption in digital circuits. This report explores five approximation methods: Voltage Over-Scaling (VOS), Logic-Level Approximation (Inexact Multipliers), Truncation-Based Approximation, Inaccurate Adder-Based Multipliers, and Hybrid Approximation with Correction Mechanisms. Each technique is analyzed in terms of power savings, error rates, advantages, disadvantages, and application suitability. A comparative analysis highlights the trade-offs involved, guiding the selection of an appropriate approximation technique for different computational needs.

1. Introduction

Power consumption in digital circuits is a critical design consideration, particularly for mobile, embedded, and low-power applications. Approximate computing techniques allow for controlled accuracy losses to achieve significant power savings. These methods are increasingly relevant as computational loads rise, particularly in artificial intelligence, image processing, and signal processing applications. This report evaluates various approximation techniques and their trade-offs, helping to identify the most suitable method for different applications.

2. Approximation Techniques

2.1 Voltage Over-Scaling (VOS)

Voltage Over-Scaling (VOS) is an approximation technique that reduces the supply voltage below the nominal level required for error-free operation. This reduction leads to significant power savings by decreasing dynamic power consumption, which is proportional to the square of the supply voltage. However, this technique also introduces timing violations, as lower voltage levels slow down transistor switching speeds, leading to computational errors due to setup and hold time violations.

In VOS, the reduced voltage results in longer transistor switching times, potentially causing some logic transitions to miss timing windows. Designers may incorporate error detection circuits to manage errors adaptively. Applications such as image processing and AI acceleration often tolerate minor errors, making VOS a viable energy-saving strategy in such domains.

How It Works:

1. Lower the supply voltage below nominal levels.

2. Transistor switching slows, increasing propagation delays.
3. Some computations miss timing constraints, resulting in incorrect outputs.
4. Can be compensated using error-tolerant algorithms or adaptive voltage scaling.

Advantages: Significant power savings without requiring hardware modifications. Suitable for low-power applications such as IoT and battery-powered devices. Reduces thermal dissipation, enhancing system longevity.

Disadvantages: High and unpredictable error rates, making it unsuitable for critical applications. Can cause significant signal integrity issues in high-speed circuits. Difficult to implement in highly optimized or real-time systems. Voltage Over-Scaling (VOS) is an approximation technique that reduces the supply voltage below the nominal level required for error-free operation. This reduction leads to significant power savings by decreasing dynamic power consumption, which is proportional to the square of the supply voltage. However, this technique also introduces timing violations, as lower voltage levels slow down transistor switching speeds, leading to computational errors due to setup and hold time violations.

Statistical Analysis:

- Average power savings: 30% - 50%
- Mean error rate: 20.86% - 38.07%
- Maximum observed error: 100%
- Suitable for: Battery-powered and energy-efficient computing devices.

2.2 Logic-Level Approximation (Inexact Multipliers)

Logic-level approximation modifies the internal structure of multipliers by simplifying logic gates, reducing transistor count, and optimizing computational pathways to obtain an approximate output with reduced power consumption.

Techniques include using approximate adders and reducing the carry propagation path, which lowers switching activity and dynamic power usage. By selectively removing logic gates that contribute minimally to accuracy, this approach ensures a balance between efficiency and correctness. These methods are highly effective in DSP applications, image compression, and AI inference tasks.

How It Works:

1. Reduce complexity in the internal logic of multipliers by eliminating or simplifying gates.
2. Approximate adders and partial product optimizations decrease the propagation delay.
3. The resulting inexact computations trade accuracy for efficiency, suitable for DSP and AI workloads.

Advantages: Good balance between power efficiency and computational accuracy. Reduces circuit complexity and transistor count. Highly effective in DSP, AI, and multimedia applications where perfect accuracy is not required.

Disadvantages: Fixed error patterns can lead to predictable inaccuracies. Not suitable for cryptographic or financial applications requiring high precision. May require fine-tuning based on specific workloads to ensure optimal performance. Logic-level approximation modifies the internal structure of multipliers by simplifying logic gates, reducing transistor count, and optimizing computational pathways to obtain an approximate output with reduced power consumption.

Statistical Analysis:

- Power savings: 31.8% - 45.4%
- Mean error rate: 1.39% - 3.32%
- Maximum observed error: 22.22%
- Applications: Digital signal processing, AI accelerators, multimedia processing.

2.3 Truncation-Based Approximation

Truncation-based approximation reduces power consumption by discarding less significant bits (LSB) or most significant bits (MSB), depending on the application's error tolerance requirements.

Truncation at the LSB level results in predictable and minor errors, while MSB truncation can introduce significant deviations. Hardware designers frequently use this approach in digital filters and neural networks, where slight precision losses are acceptable for power efficiency gains.

How It Works:

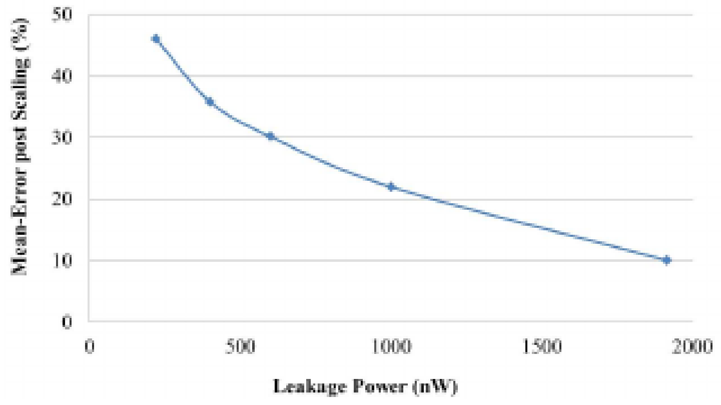
1. Selectively discard LSBs or MSBs based on application needs.
2. LSB truncation results in minor precision loss, useful in approximate AI computations.
3. MSB truncation has significant impact, making it suitable only for highly error-tolerant tasks.

Advantages: Simple and easy to implement with minimal design modifications. Provides moderate power savings with controlled accuracy loss. Well-suited for applications tolerating small precision reductions, such as digital filters and AI models.

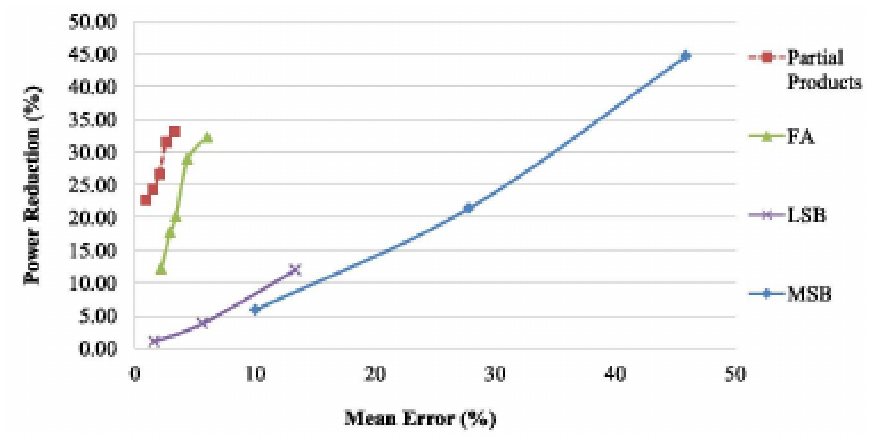
Disadvantages: Limited accuracy control, making it unsuitable for mission-critical applications. Significant truncation can lead to high errors in numerical computations. Not ideal for applications requiring high dynamic range processing. Truncation-based approximation reduces power consumption by discarding less significant bits (LSB) or most significant bits (MSB), depending on the application's error tolerance requirements.

Statistical Analysis:

- Power savings: Varies with bit-width and configuration.
- Mean error rate: Up to 3.32%
- Maximum error: 44.44%
- Best suited for: Low-power embedded systems, digital filters, neural networks.



Accuracy vs. power tradeoff for truncation for an 8-bit multiplier. Both adder and partial-product based approaches offer better tradeoffs.



Accuracy vs. power tradeoff for truncation. Both adder and partial-product based approaches offer better tradeoffs.

2.4 Inaccurate Adder-Based Multipliers

This method introduces approximation in the adder network used for summing partial products instead of modifying the entire multiplier architecture. It leverages simplified addition logic, such as carry-cutting adders or approximate compressors, to reduce complexity and energy consumption.

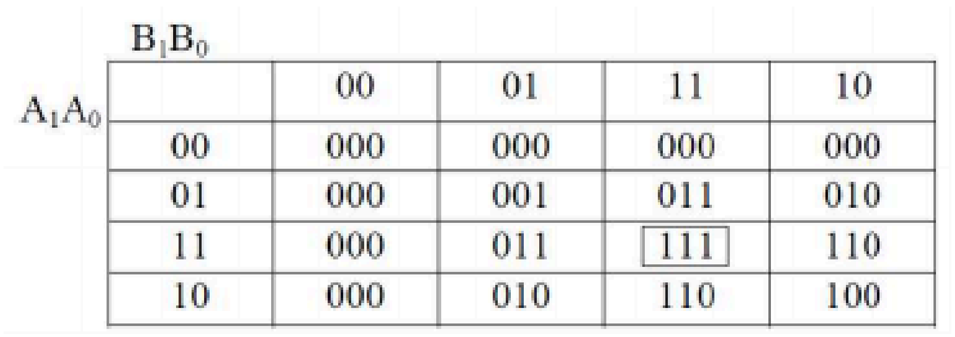
By allowing controlled inaccuracies in sum propagation, designers achieve significant power savings while keeping error margins acceptable. The technique is effective in applications where arithmetic precision can be relaxed, such as deep learning accelerators and signal processing.

How It Works:

1. Approximate adders reduce propagation delays in summing partial products.
2. Reduces energy-hungry switching activity within adder networks.
3. Suitable for applications that tolerate small numerical errors, such as deep learning inference.

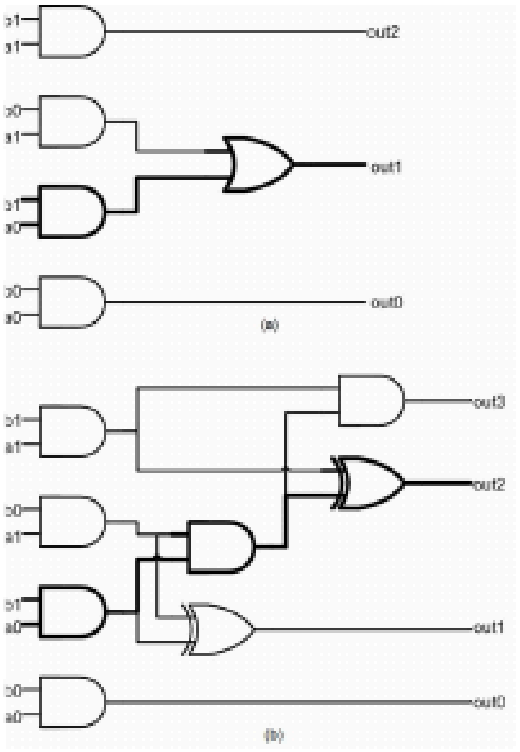
Advantages: Achieves significant power savings with moderate computational accuracy. Can be applied selectively to reduce computational overhead. Useful in AI accelerators and DSP applications where approximations are tolerable.

Disadvantages: Higher mean error compared to logic-level approximations. Error propagation is less controllable, which may impact performance. May require additional design considerations to mitigate accuracy loss in critical computations. This method introduces approximation in the adder network used for summing partial products instead of modifying the entire multiplier architecture. It leverages simplified addition logic, such as carry-cutting adders or approximate compressors, to reduce complexity and energy consumption.



B_1B_0 / A_1A_0	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	$\boxed{111}$	110
10	000	010	110	100

Figure 1: Karnaugh-Map for the inaccurate 2x2 multiplier



The accurate (b) and inaccurate (a) 2x2 multipliers, with the critical paths highlighted

Statistical Analysis:

- Power savings: 20% - 38%
- Mean error rate: 6.01% - 10.07%
- Maximum error: 62.22%
- Common applications: AI-based arithmetic circuits, DSP, and edge computing.

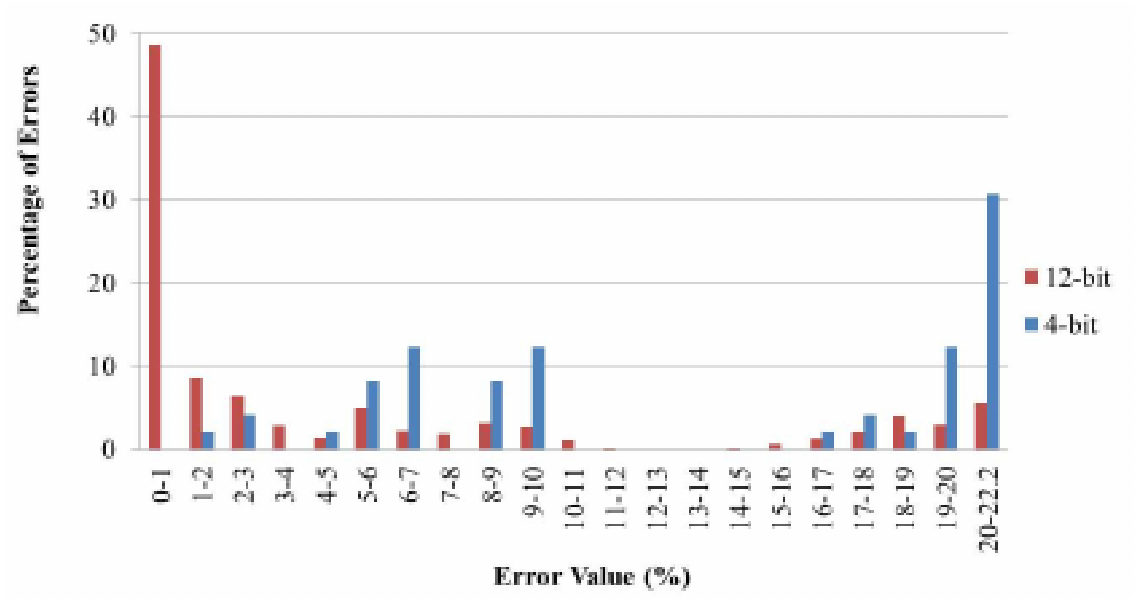
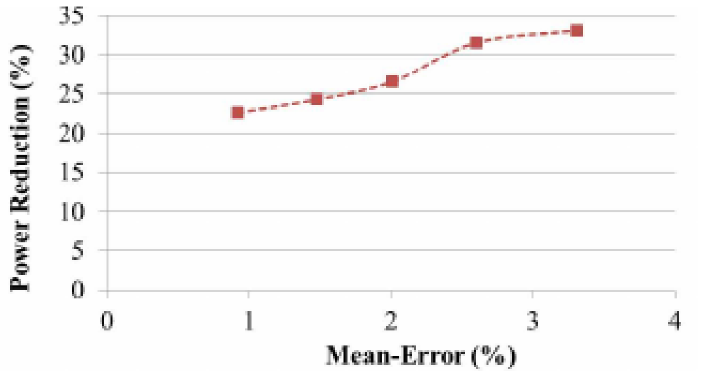
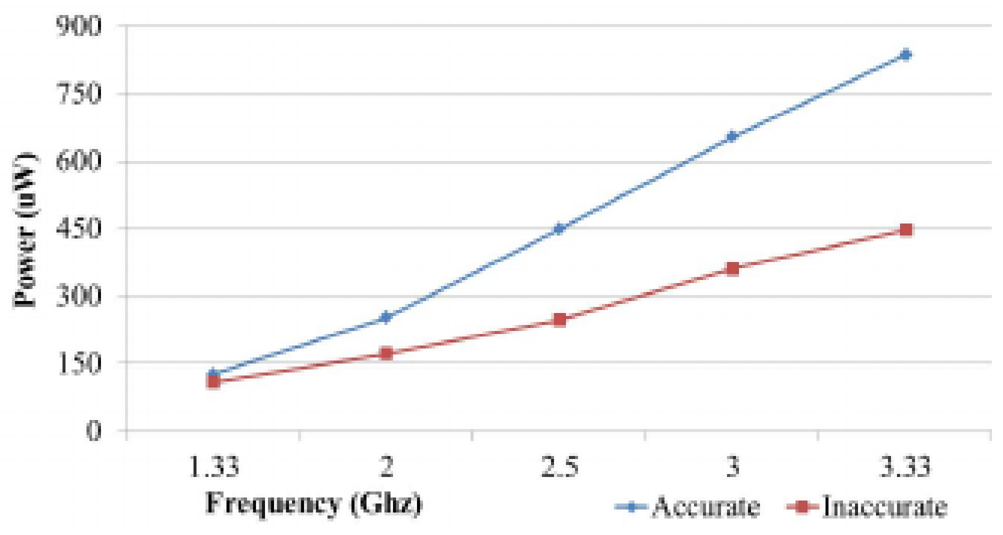


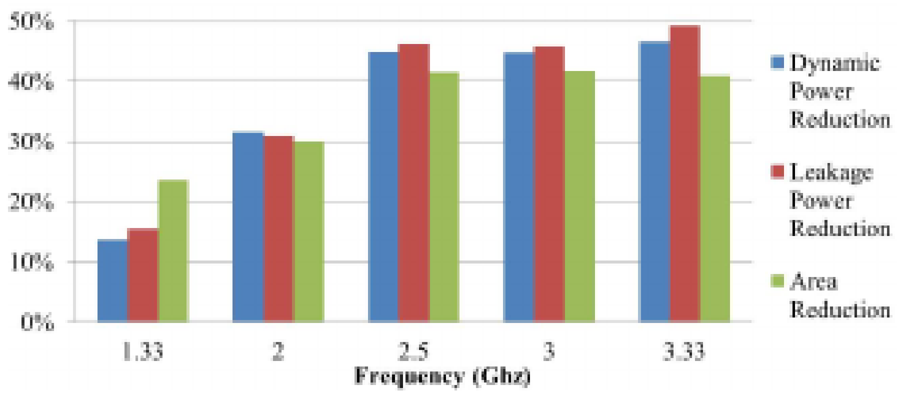
Figure 4: Error percentage distribution for 4-bit and 12-bit inaccurate multipliers



Accuracy vs. Power Tradeoff for the 4-bit inaccurate multiplier. Increasing mean-error leads to an increase in power savings.



Dynamic power vs. frequency for 4-bit accurate and inaccurate multipliers



Dynamic power, leakage power and area savings for a 4-bit inaccurate multiplier

2.5 Hybrid Approximation with Correction Mechanisms

This approach dynamically switches between accurate and approximate modes, incorporating error detection and correction units to adjust accuracy based on computational needs.

Hybrid approximation employs runtime monitoring mechanisms to determine when precision is essential and activates correction circuits accordingly. Correction methods include redundancy-based error detection and adaptive scaling strategies. These systems are ideal for real-time processing where workloads fluctuate, such as adaptive video encoding and edge computing devices.

How It Works:

1. System monitors workload and switches between accurate and approximate operations.
2. Error detection mechanisms identify critical operations that require correction.
3. Adaptive correction algorithms restore accuracy selectively.

Advantages: Best trade-off between power efficiency and computational accuracy. Suitable for adaptive systems where precision requirements change dynamically. Enables real-time adjustments for energy-efficient computing without significant loss in performance.

Disadvantages: Additional hardware overhead due to error detection and correction circuits. Increased implementation complexity compared to simpler approximation methods. May not be ideal for ultra-low-power applications due to correction overhead. This approach dynamically switches between accurate and approximate modes, incorporating error detection and correction units to adjust accuracy based on computational needs.

Statistical Analysis:

- Power savings: Up to 33.22%
- Mean error rate (Partial Correction): 1.49% - 2.29%
- Correction overhead: 4.6% - 10.5%
- Application areas: Adaptive computing, high-performance low-power systems.

3. Comparative Analysis

Approximation Method	Power Savings	Mean Error	Max Error	Best Use Cases
Voltage Over-Scaling	30% - 50%	20.86% - 38.07%	100%	Low-power applications with error tolerance
Logic-Level Approximation	31.8% - 45.4%	1.39% - 3.32%	22.22%	DSP, machine learning, image processing
Truncation-Based	Varies	Up to 3.32%	44.44%	Power-efficient circuits with controlled errors
Adder-Based	20% - 38%	6.01% - 10.07%	62.22%	Arithmetic circuits needing power reduction
Hybrid with Correction	Up to 33.22%	1.49% - 2.29%	Varies	Adaptive systems with critical operations

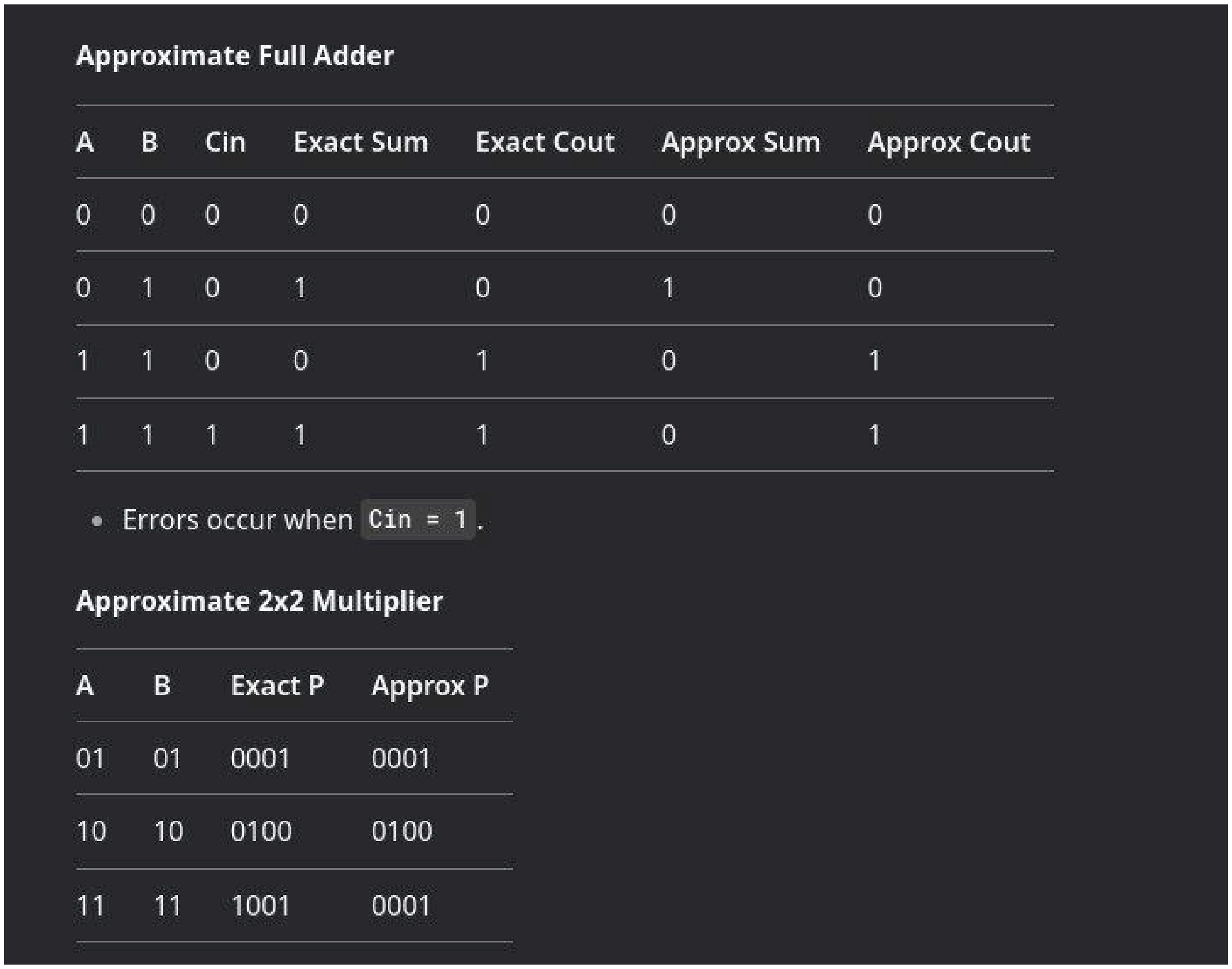
4. Approximate Code:

. Approximate Full Adder (Logic-Level Approximation):

This design simplifies the logic by ignoring the carry-in (Cin) for the sum calculation and using a simplified carry-out logic.

. Approximate 2x2 Multiplier (Inaccurate Adder-Based Multipliers):

This design uses an approximate adder network to sum the partial products, reducing complexity and power consumption.



Approximate Full Adder

A	B	Cin	Exact Sum	Exact Cout	Approx Sum	Approx Cout
0	0	0	0	0	0	0
0	1	0	1	0	1	0
1	1	0	0	1	0	1
1	1	1	1	1	0	1

- Errors occur when `Cin = 1`.

Approximate 2x2 Multiplier

A	B	Exact P	Approx P
01	01	0001	0001
10	10	0100	0100
11	11	1001	0001

5. Impacts on Image processing:

-- **Compression**: Approximate multipliers can enhance image compression algorithms by speeding up calculations without significantly impacting visual quality.

-- **Denoising**: For real-time applications, approximate computing allows faster noise reduction algorithms while maintaining acceptable quality levels.

-- **Deep Learning** Neural networks for image classification can benefit from approximate multipliers, reducing inference time and energy consumption.

-- **Edge Detection**: In applications where real-time processing is critical (e.g., robotics), approximate computing can speed up edge detection with minimal quality compromise.

-- **Image Filtering(Gaussian smoothing/sharpening):** Tasks like smoothing or sharpening can use approximate multipliers to execute faster while still providing visually plausible results.

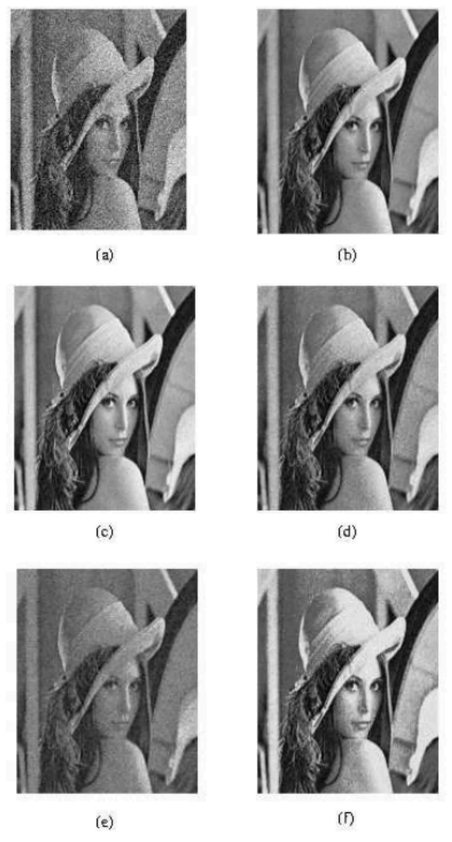


Figure 11: Image sharpening (a) original blurred image; (b) enhanced using accurate multiplier; (c) by inaccurate multiplier, power reduction 41.5%, SNR : $20.365dB$**; (d) voltage over-scaling for** 30% **power reduction, SNR :** $9.16dB$**; (e) voltage over-scaling for** 50% **power reduction, SNR :** $2.64dB$**; (f) by introducing errors via the adders, SNR :** $7.3dB$

6. Conclusion

Each approximation method provides different trade-offs between power efficiency and computational accuracy. These techniques are critical in designing next-generation low-power computing systems. Understanding the trade-offs allows system designers to select the appropriate technique based on computational constraints and energy efficiency requirements. Future research will likely refine these methodologies further, incorporating AI-driven optimization techniques to maximize power savings while maintaining system reliability.