Design of an Approximate 8-bit MAC Unit: A Step Towards RISC-V SoC Integration

Amaan Mujawar

University of Sheffield
School of Electrical and Electronic Engineering

Supervised by
Mr. Niel Powell

A report submitted in partial fulfilment of the requirements for the degree of
Masters of Engineering

7th May 2025

DECLARATION

[illegible]

Amaan Mujawar

Abstract

The rapid growth in data-intensive applications, such as signal processing, image processing, and multimedia tasks, has led to a demand for more efficient computing solutions. Many signal processing, image processing, and multimedia tasks are inherently error-tolerant and can produce results that appear indistinguishable to the human eye, even without the need for exact computations. By leveraging this error tolerance approximate computing provides an approach, offering reduced power consumption and increased processing speeds by tolerating minor inaccuracies in calculations.

The primary goal of this project is to design and implement a MAC unit that seamlessly integrates with the RISC-V processor architecture while achieving performance improvements, such as lower power consumption, faster processing speeds, and tunable accuracy levels. The project utilizes the Xilinx Vivado Design Suite for design, simulation, and synthesis, and the implementation is carried out on the Artix Nexus 7 FPGA platform. Detailed performance metrics, including power consumption, speed benchmarks, and accuracy, are measured and evaluated to ensure the validity of the design.

The design has been prepared with future integration in mind and the capability of a final integration into a RISC-V System on Chip (SoC) offering a scalable and flexible hardware solution for approximate computing in embedded systems. The results demonstrate significant power and speed improvements with acceptable accuracy losses, highlighting the potential of approximate computing techniques for future high-performance computing applications.

CONTENTS

LIST OF TABLES

List of Figures

I. Introduction

The advancement of digital technology has revolutionised the way data is processed and analysed, enabling a wide range of applications, from machine learning and computer vision to real-time multimedia and big data analytics. These data-intensive fields placed significant pressure on computing systems, pushing the boundaries of traditional processor architectures in terms of speed, energy efficiency, and scalability. Traditionally, gains in computing power have followed Moore's law [7], which predicted an exponential increase in transistor density in integrated circuits. As transistors are near physical and economic constraints, the advantages of conventional scaling have reduced, necessitating new strategies for further enhancing computational efficiency.

In response to these limitations, approximate computing has emerged as a promising paradigm that challenges the conventional notion of precise computation. Instead of striving for exact outputs, approximate computing allows for controlled inaccuracies, trading off precision for significant gains in performance, power efficiency, and silicon area reduction [1]. This trade-off is particularly acceptable in many real-world applications, such as image processing, signal filtering, and machine learning, where outputs are inherently resilient to minor errors and human perception masks small computational inaccuracies.

Given the growing need for energy-efficient computing, especially in embedded and portable systems, the integration of approximate computing techniques into hardware accelerators offers an exciting opportunity. In particular, arithmetic units such as the Multiply-Accumulate (MAC) unit, which are fundamental to numerous digital signal processing (DSP) and machine learning tasks, present an ideal candidate for approximation. The motivation for this dissertation stems from the need to explore such opportunities by designing a power and performance-optimised MAC unit for a modern Reduced Instruction Set Computing (RISC-V) System on Chip (SoC) environment.

A. Aims and Objectives

The aim of this project is to design and implement an 8-bit MAC unit that incorporates approximate computing methods, with a focus on achieving lower power consumption and faster execution time while maintaining acceptable accuracy levels for targeted applications. The MAC unit is intended for eventual integration into a RISC-V SoC, offering a scalable and efficient solution for energy-constrained computing systems.

The specific objectives of the project are as follows. First, a comprehensive literature review will be conducted to examine existing approximate computing techniques, with particular emphasis on approximate multipliers and adders that are suitable for integration into a MAC unit. The next objective is the design phase, during which a pipelined 8-bit MAC unit will be architected, incorporating selected approximation techniques such as fuzzy memoization and compressor-based multipliers, while striving to balance performance with accuracy. Following this, the implementation of the proposed design will be carried out using Verilog HDL, targeting the Xilinx Vivado Design Suite and the Artix Nexus 7 FPGA for simulation, synthesis, and hardware validation. The integration phase will involve preparing the MAC unit for incorporation into a RISC-V SoC, ensuring compatibility and modularity to support future expansion. Finally, the design will be evaluated based on key metrics such as power consumption, execution speed, and computational accuracy, with comparisons made to conventional designs or benchmarks.

B. Project outline

This dissertation is organised to lead the reader through the comprehensive design and evaluation of the approximate MAC unit. Following this introductory section:

Chapter 2 offers an extensive review of the literature on approximate computing techniques implemented, with an emphasis on hardware-level implementations and their use in arithmetic units.

Chapter 3 outlines the architectural design and methods used develop the approximate MAC unit, detailing the approximation strategies for multipliers and adders and, pipelining structure.

Chapter 4 elaborates on the Verilog implementation process and the FPGA configuration for testing purposes.

Chapter 5 details the evaluation framework, presents the results of simulation and synthesis, and examines the trade-offs between power, performance, and accuracy.

Chapter 6 encapsulates the conclusions, and suggests future research paths, such as a complete integration of the MAC unit into a RISC-V SoC.

II. Literature Review

The exponential growth in computational demands, driven by applications in machine learning, multimedia processing, and big data analytics, has strained traditional digital design paradigms. Classical computing architectures prioritise precision and exactness, which come at the cost of increased power consumption, area usage, and latency. With the diminishing benefits of Moore's Law and the rising need for energy-efficient hardware, approximate computing has emerged as a transformative approach to hardware design. Approximate computing operates on the principle that not all applications require perfect accuracy [8]. Many domains, especially those involving human perception or probabilistic outcomes, can tolerate small errors without significant degradation in performance. By trading off accuracy, approximate computing reduces hardware complexity, resulting in substantial improvements in energy efficiency and processing speed. At the heart of this model shift are arithmetic units like adders and multipliers which constitute a significant portion of computational workloads in digital systems. Optimising these units for approximate computing forms the core of this paper's contributions.

A. *Approximate Adders*

Adders are a fundamental component in digital circuits, responsible for executing arithmetic operations that often dominate computational workload. Traditional adder designs prioritise accuracy; however, approximate adders introduce intentional inaccuracies to achieve resource savings. A proposed approximation approach using Lower-Part OR-based Approximate Adders [9] aligns with similar research, introducing the concept of approximate adders as a means to trade off accuracy for reduced power consumption and area in energy-efficient VLSI systems. Ramasamy et al. proposed a carry-based approximate full adder, demonstrating that bypassing the carry propagation chain in the least significant bits (LSB) can drastically improve speed and reduce area at the cost of negligible error [10].

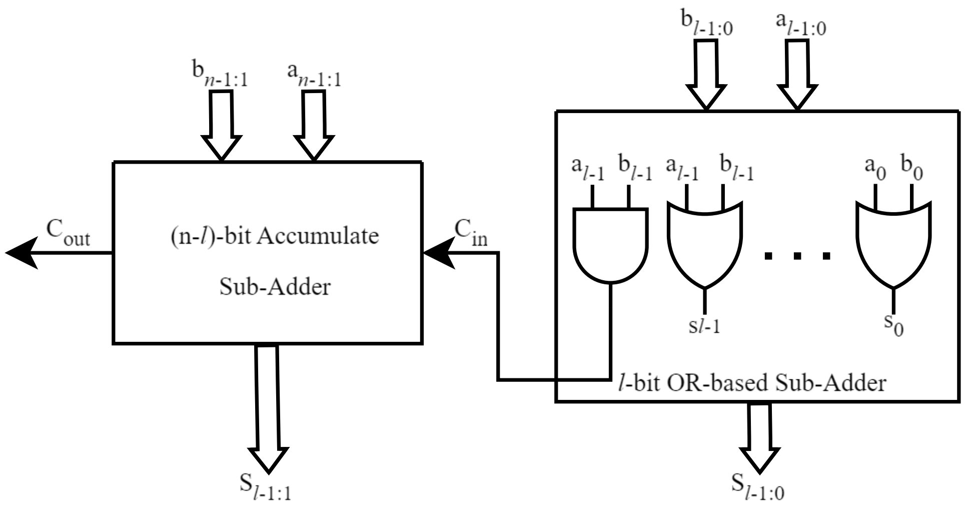

Fig. 1: Lower-Part OR Adder [1]

B. *Approximate Multipliers*

Multiplication is a computationally intensive operation, making approximate multipliers a critical focus for energy-efficient design. Approximate multipliers reduce the complexity of partial product summation, which directly impacts delay and power consumption. A novel hardware design of approximate multipliers is provided, Lower-Part OR-based Approximate Multiplier [8], integrating the concept of Wallace Tree multipliers for accurate MSBs and OR-based logic for approximate LSBs. The combination of these techniques results in a novel multiplier design that balances accuracy, speed, and resource utilisation, suitable for FPGA-based implementations.

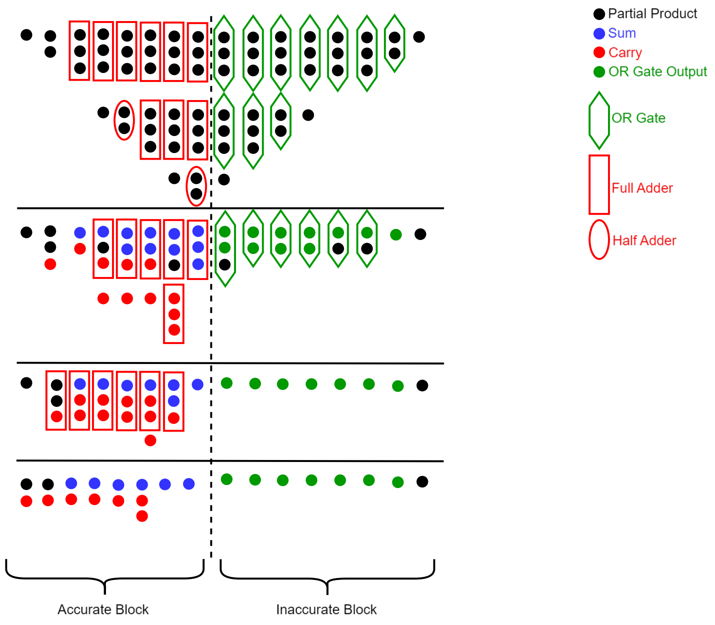

Fig. 2: Lower-Part OR Wallace Tree Multiplier [1]

C. Approximate Matrix Multipliers

Matrix multiplication is a fundamental operation in numerous computational tasks, including AI, scientific computing, and graphics processing. Despite its importance, research into approximate matrix multipliers is limited. The proposed matrix multiplier design is a significant step forward, as it combines approximate multipliers and adders in a single hardware implementation [1], by targeting an FPGA platform and demonstrating scalability across different matrix sizes and bit-widths.

The architecture of the approximate matrix multiplier is segregated into two parts, an approximate adder and an approximate multiplier. The approximate adder is segmented into two parts, an accurate and an inaccurate part. The upper (accurate) part performs the addition using a simple ribbed carry adder (RCA),, while the lower part is performed bit-by-bit using OR gates. The approximate multiplier is divided into two procedural steps, partial product generation and summation. The accurate (upper) block employs the Wallace Tree Multiplier, while the inaccurate (lower) block employs OR gates to generate the final product for the lower part [1].

Finally, the approximate adder and the approximate multiplier are brought together to perform matrix multiplications. The idea is to use these adders and multipliers in a synchronological manner to provide better control of error percentage in the approximations.

D. Compressor-Based Approximate Multipliers

Traditional Multiplier Architectures typically involve partial product generation, partial product accumulation, and final addition. Partial product generation involves producing intermediate results by multiplying bits of input operands, followed by partial product accumulation, summing the intermediate results using adders or compressors, and lastly final addition produces the output from accumulated partial products. A compressor is a combinational logic circuit used to sum multiple binary inputs and produce a small number of outputs, usually two: a sum and a carry. The most commonly used compressors are 3:2, 4:2, and 5:2 [8], [11]. In traditional designs, compressors play a critical role in the accumulation phase. However, conventional exact compressors are power-intensive and complex, especially in FPGA-based implementations due to limited logic resources and/or cascading delays and increased power consumption from logic circuits.

1) Conventional 3:2 Compressors (Full Adder)

A 3:2 compressor is equivalent to a full adder. It takes three input bits and outputs two bits: a sum, the least significant bit of the result, and a carry, the most significant bit of the result [2].

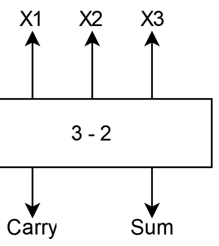

Fig. 3: Block Diagram of 3-2 Compressor [2]

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \cdot B) + (B \cdot C_{in}) + (A \cdot C_{in})$$

Figure 3 illustrates the block diagram and general structures of traditional 3-2 compressor architectures. The conventional 3-2 compressor design incorporates XOR gates, typically realised using CMOS logic, which are present in the critical path of the circuit. This placement can limit the speed of the system due to the relatively high latency associated with XOR gates. To improve performance, alternative designs have been explored that replace certain components with more efficient structures, such as approximate compressors, XOR gates replaced with multiplexers and the use of Full Adders [8]. This is the simplest compressor and serves as the building block for higher-order compressors.

2) ***Conventional 4:2 Compressors***

A 4:2 compressor takes four inputs and produces two output bits and an additional carry-in and carry-out.

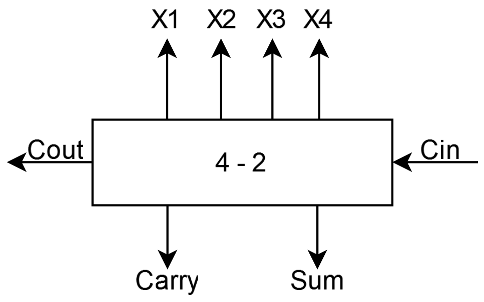

Fig. 4: Block Diagram of 4-2 Compressor [3]

$$S = A \oplus B \oplus C \oplus D \oplus C_{in}$$

$$C = (A \cdot B) + (C \cdot D) + (C_{in} \cdot (A \oplus B \oplus C \oplus D))$$

This style of compressor is advantageous as it reduces four rows of partial products to two, with a carry propagated to the next stage, and minimises delay compared to a series of 3:2 compressors. Most commonly used in high-performance multipliers to speed up partial product reduction, especially in Dadda multipliers [8].

3) ***Conventional 5:2 Compressors***

The 5:2 compressor takes five input bits and produces two output bits, along with two carry bits (one from the previous stage and one for the next stage).

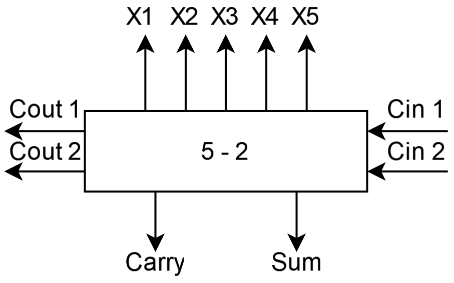

Fig. 5: Block Diagram of 5-2 Compressor [4]

$$S = A \oplus B \oplus C \oplus D \oplus E \oplus C_{in1} \oplus C_{in2}$$
$$C = \text{Majority function of inputs}$$

A 5:2 compressor is particularly efficient for reducing a large number of partial product rows in multipliers, such as 16x16 or 32x32 and used to achieve better performance, by the use of higher order compressors like 5:2 compressors.

4) Approximate m:2 Compressor

Traditional compressors focus on exact computations, which are not always necessary for error-tolerant applications such as image processing or machine learning. The Approximate m:2 Compressor is designed to aggregate multiple elements in two equal-weight outputs while minimising hardware complexity and power consumption.

The Approximate m:2 Compressor is designed into two output bits, Sum (S) and Carry (C), which represent the cumulative result of m elements with reduced precision. Probability analysis is used to determine that most partial product values are concentrated between 0 and 2, making it feasible to represent them with two outputs.

$$S = A + B + C + \cdots + m$$
$$C = (A \cdot (B + C + \cdots + m)) + (B \cdot (C + \cdots + m)) + \dots$$

The use of OR-based logic gates reduces the number of LUTs required compared to traditional compressors using XOR and AND gates, thus reducing propagation delay and power consumption [8].

E. Once-Through Multiplier Architecture

The design objectives of Content Addressable Memory, CAM2 are to minimise power consumption, area utilisation, and delay by using simple logic operations, such as OR gates, instead of more complex compressors in specific stages [8].

The CAM2 is implemented in three stages:

- Stage 1: Initial compression of partial products using carry-lagged compressors.
- Stage 2: Approximate compression of remaining partial products using OR operations.
- Stage 3: Final summation using carry chains to produce the final product.

1) Hardware Efficiency and Metrics

CAM2 achieves a power reduction of 57.90% compared to exact multipliers. The use of OR operations significantly reduces the area, leading to a 33.80% reduction in LUT usage. By simplifying logic in Stage 2 and avoiding recursive carry propagation, CAM2 reduces delay by 24.78%, with a Mean Relative Error Distance (MRED) of 5.86% and an Error Rate (ER) of 84.50% [8]. CAM2 sacrifices accuracy for greater efficiency, making it suitable for applications where minor inaccuracies are acceptable.

F. Fuzzy Memoization

Fuzzy memoization is an approximate computing technique based on instruction memoization. Here, approximate results are cached and reused instead of recomputing exact results. Both Instruction and Fuzzy memoization store input and output data for a process as an entry in the memo table, and reuse it to reduce execution time by skipping the original process [5].

This is particularly useful in applications where slight inaccuracies in the output are acceptable, such as image processing, machine learning, and signal processing.

1) Cache Initialisation

A cache (or lookup) table is created to store the results of previous computations. Each entry in the cache consists of inputs and outputs. For instance, in a MAC operation:

- Inputs: A and B
- Outputs: A * B + previous result

2) Fuzzy Matching

Search the cache for a stored input that is "similar enough" to the new input, based on a predefined similarity threshold. If a similar input is found (cache hit), reuse the cached result instead of performing the computation. If no similar input is found (cache miss), compute the result, store it in the cache, and use it for future queries [5].

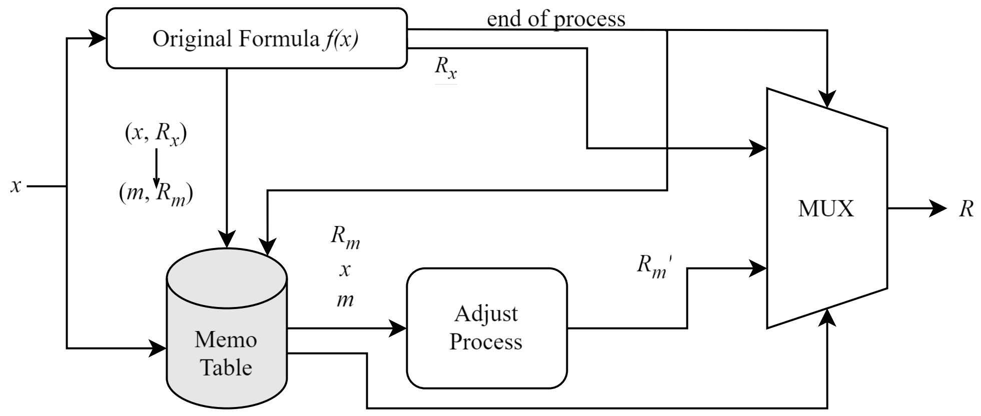

Fig. 6: Proposed Memoization System [5]

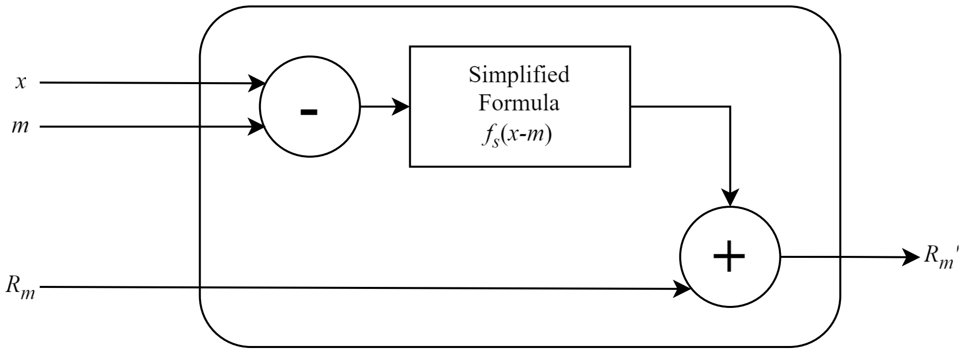

Fig. 7: Adjust Process [5]

3) Similarity Metrics

Several metrics can be used to determine the similarity between inputs, depending on the application. Hamming distance counts the number of bit positions in which the inputs differ, Euclidean distance measures the geometric distance between two vectors in a multi-dimensional space. Manhattan distance measures the sum of absolute differences between corresponding elements of two vectors, and custom thresholding is a user-defined threshold that dictates the maximum allowable threshold difference between inputs.

4) Updating the Cache

When a cache miss occurs, the cache is updated with the new input-output pair. Depending on the cache size and replacement policy, older or less relevant entries may be removed. Common cache replacement policies include:

- **Least Recently Used (LRU)**: Removes the least recently accessed entry.
- **First-In-First-Out (FIFO)**: Removes the oldest entry.
- **Random Replacement**: Randomly selects and entry to replace.

III. Architecture and Design

A. *Design Methodology*

This chapter presents the main technical developments of the Approximate 8-bit Multiply-Accumulate (MAC) unit. It begins with a discussion of the overall architecture, followed by a detailed examination of the approximate components chosen, such as the Dadda multiplier and a Lower Approximate Carry Look-Ahead (CLA) adder. The aim is to balance computational efficiency and accuracy, particularly for applications in image processing where some degree of imprecision can be tolerated. The chapter also explains the reasoning behind using pipelining and fuzzy memoization to optimise performance and efficiency, laying the foundation for the implementation phase that follows.

B. *Dadda-Based Multiplier Design*

1) *Overview*

Multiplication is a complex process and is the primary cause of time consumption during any operation. Utilizing approximate computing in multipliers can aid in large-scale operations [1]. The multiplier used is based on the Dadda tree reduction technique, chosen for its efficiency in reducing the number of reduction stages compared to Wallace trees. The Dadda multiplier is optimized for speed and area, making it an acceptable choice for hardware implementations where resource constraints are critical. The multiplier design can be split into three phases: partial product generation by bit-by-bit multiplication, partial product accumulation, and final summation of partial products.

2) *Partial Product Generation*

In the proposed design, the first stage of the multiplier generates 64 partial products using AND gates. These products form an 8x8 matrix of bits which will be reduced in the subsequent stages with the use of a reduction tree using compressors, discussed further on. This operation is repeated for all 64 combinations of the 8-bit inputs. The summation process has been performed by dividing the partial products into two blocks: a most significant bit (MSB) accurate block and a least significant bit (LSB) approximate block. The approximate LSB block employs OR gates to generate the final product. This dividing methodology has been discussed in many research papers based on numerous factors such as power consumption, delay, and accuracy [1].

3) *Reduction Tree Using Approximate Compressors*

In a multiplier, a compressor is a circuit to accumulate and reduce multiple rows of partial products into two rows for the final adder. To accumulate the partial products efficiently, exact and inexact compressors are proposed. The reduction of partial products is performed in multiple levels using a combination of half adders (HAs), full adders (FAs), and 4:2 compressors [12].

This approach to partial product reduction is a crucial step in the implementation of the Dadda-based multiplier, as it significantly affects the overall speed, area, and power consumption of the multiplier unit. In traditional Wallace multipliers, partial product bits are reduced level by level until only two rows remain for final addition. The Dadda multiplier improves upon this by minimising the number of required adders at each stage, thus optimizing the number of operations and reducing logic depth.

However, to achieve further gains in performance, this design incorporates compressor-based reduction, particularly 4:2 compressors, alongside conventional full and half adders. Compressors, such as 4:2 compressors, are well suited for dense accumulation of partial products because they can take four input bits and produce two output bits (sum and carry), thus reducing the vertical height of the partial product matrix more aggressively than full adders. This not only reduces the total number of reduction levels but also shortens the critical path delay, which is key in improving the multiplier's clock speed and throughput.

Additionally, by using compressors instead of a chain of full adders, the design achieves a more balanced load distribution across levels. This has positive implications for power consumption, which is particularly important in embedded systems.

Each stage in the reduction tree is constructed to meet the target column heights prescribed by the Dadda algorithm. Compressors are deployed in locations with higher column heights to maximise their impact, while full and half adders are used in sparser regions of the matrix to avoid unnecessary logic.

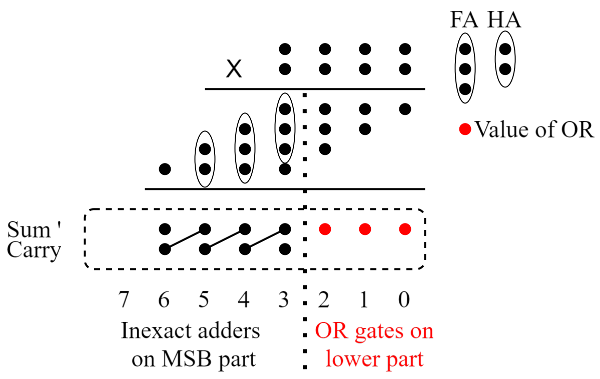

Fig. 8: Approximate Multiplication Block [1]

4) *Final Addition*

In the final stage of the Dadda multiplier, once the partial products have been reduced to two rows through successive levels of 4:2 compressors and (3,2) counters (full and half adders), a final addition is necessary to produce the complete 16-bit product. This is a critical stage in any multiplier architecture, as it determines the correctness and timing of the final result.

The two remaining rows represent intermediate results that must be added together using binary addition with full carry propagation. In the implemented design, this final addition is performed using a ripple-carry adder constructed from basic logic gates or structural Verilog. This method ensures exact summation of the bits and complete propagation of carry signals from the least significant to the most significant bits.

This stage is fully accurate, and no approximation is applied. The use of an exact binary adder guarantees that no error is introduced during the final summation, which is essential for applications requiring high numerical precision. While ripple-carry adders are not the fastest option in terms of delay (as the carry must propagate through every bit), they are hardware-efficient and easy to implement, making them a practical choice for moderate-speed designs such as this.

The design choice was to not truncate, estimate, or bypass carry propagation, which are typical features of approximate adders. Instead, the focus remains on maintaining full accuracy of the product, particularly since all preceding reduction logic (e.g., compressors) is also exact. This ensures that the final 16-bit output accurately represents the multiplication of two 8-bit operands, without approximation-induced distortion.

C. *Approximate Adder Architecture*

The approximate adder implemented in this project is designed to reduce hardware complexity, power consumption, and delay by simplifying the carry propagation logic in the lower bits of the adder. This design follows a Lower-part OR-based Carry Approximate Adder (LOCA) strategy, combined with an exact adder in the upper bits to preserve accuracy for more significant contributions.

1) *Design Structure*

In the proposed design, the adder is split into two parts, the lower part of the adder operates in an approximate manner. Specifically, the least significant bits of the operands are added using a simple bitwise OR gate. This approach eliminates the need for carry generation and propagation in the lower section, thereby significantly reducing both the logic depth and the silicon area. As these bits contribute minimally to the overall numerical value, any inaccuracies introduced at this stage have a negligible effect on the final result.

Conversely, the upper part of the adder is implemented using accurate logic, such as a Carry Lookahead Adder (CLA). In this region, full carry generation and propagation are preserved to ensure precise computation, particularly in the most significant bits where errors would have a substantial impact on the output.

This hybrid approach enables the adder to maintain low error rates while achieving noticeable hardware savings. The number of bits assigned to each part was determined experimentally to strike a balance between performance and accuracy.

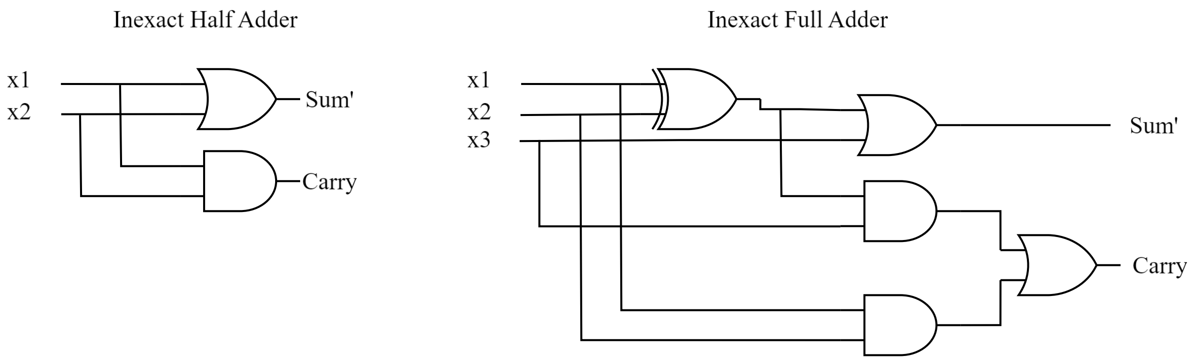

Fig. 9: Approximate Half and Full Adders [1]

2) *Functional Behaviour*

In operation, the lower OR-based segment computes each output bit as:

$$\text{Sum}[i] = A[i] \,|\, B[i] \quad \text{for } i < k$$

Where k is the boundary between the approximate and accurate sections. For bits $i \geq k$, the addition is performed using traditional full-adder logic, and carry input is properly propagated.

This architecture effectively truncates carries from the lower part, which is acceptable for error-tolerant applications. The resulting sum has a bounded and statistically controlled error profile.

D. Pipelining

Pipelining was implemented in the design of the approximate MAC unit to improve throughput and reduce critical path delays. The MAC architecture was divided into multiple pipeline stages, each responsible for a specific operation in the multiply-accumulate process.

1) *Pipeline Stages*

The data-path of the system is divided into three primary stages, each contributing to the overall computation process. The first stage, partial product generation, is managed by a compressor-based approximate multiplier. This stage is responsible for generating partial products that will later be processed in subsequent stages. The second stage involves partial product reduction and Final Summation, where the generated partial products are reduced and summed together to form the final result. The third and final stage is accumulation, which is performed using an approximate adder to accumulate the summed values.

To enhance the performance of the system, each of these stages is registered, allowing for simultaneous computation of different input sets across the pipeline. This pipelining technique significantly improves throughput when compared to a non-pipelined design, as it allows for continuous processing of multiple data sets without waiting for each stage to complete sequentially. The use of pipelining thus optimizes the overall efficiency and speed of the computation.

2) *Performance Improvement*

By introducing registers between stages, pipelining reduces the logic depth in each stage, allowing for a higher clock frequency. This design choice ensures that the MAC unit can be operated at higher speeds while maintaining functional correctness and predictable timing behaviour.

Although pipelining increases the number of registers (hence flip-flop usage), the trade-off is acceptable given the improvement in processing rate. In resource-constrained environments, the pipeline depth can be tuned based on timing and area requirements.

E. Fuzzy Memoization

Fuzzy memoization is an optimization technique used in this design to exploit redundancy in input data by reusing previously computed results when inputs are sufficiently similar. This concept is particularly effective in error-tolerant applications, such as image or signal processing, where perfect accuracy is not mandatory.

1) *Memoization Strategy*

A Lookup Table (LUT)-based cache was designed to optimise computational efficiency by storing recently computed input-output mappings. In this design, each new input to the Multiply-Accumulate (MAC) operation is compared against entries in the cache using a similarity metric, such as the Hamming distance. If the new input is found to be sufficiently close to a stored entry i.e., it falls within an acceptable similarity threshold the corresponding cached output is reused, resulting in what is known as a ”fuzzy hit.” This approach reduces the need for recalculating the operation and can significantly improve throughput, especially for operations involving recurring patterns or similar input values. On the other hand, if no suitable match is found within the cache, the operation is executed as usual, and the resulting output is subsequently stored in the table for future reuse. This design concept leverages the principles of approximate computing and cache-based optimisation, where approximate matches can substitute exact computations, yielding performance gains without a significant sacrifice in output accuracy. The effectiveness of this method depends upon the design of the similarity metric and the threshold criteria, both of which determine the trade-off between cache hit rate and computational correctness [5] [6]. By integrating this LUT-based caching strategy, the system can balance computational efficiency with resource utilisation, particularly in scenarios where exact precision is not critical but speed is.

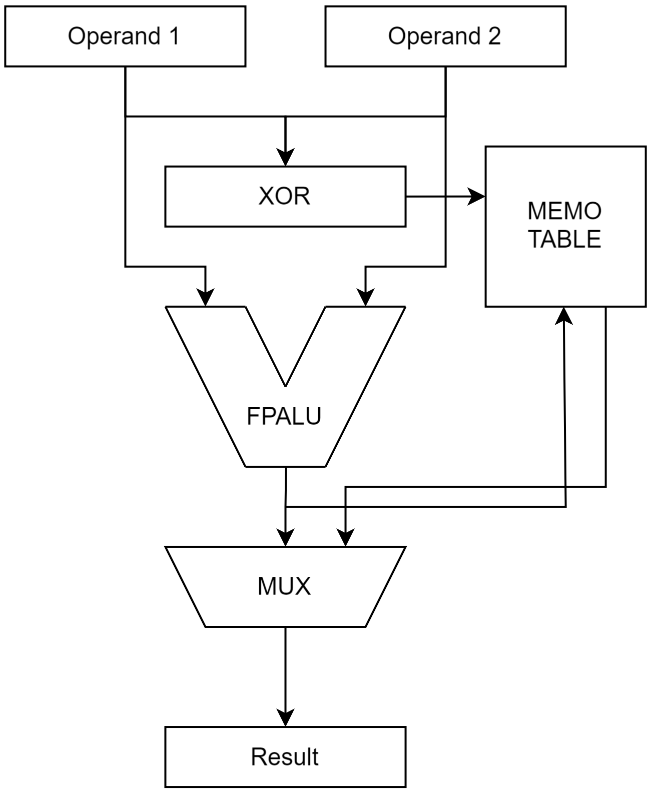

Fig. 10: Hardware classification of sequential LUT for memoization systems [6]

2) *Hardware Implementation*

The memoization unit is a key component designed to enhance computational efficiency by reusing previously computed results. It consists of several subcomponents: a small associative memory or Content Addressable Memory (CAM)-like structure for comparing recent inputs, a control logic unit to determine whether to fetch results from the cache or perform a new computation, and storage buffers for both inputs and corresponding outputs. The purpose of the associative memory is to enable fast lookups of previous inputs, facilitating the retrieval of cached results when there is a match, thus avoiding the need for redundant computations. The control logic is responsible for making the decision to either access the stored result from the cache (a ”fuzzy hit”) or initiate a fresh computation if no suitable match is found. Storage buffers are used to temporarily hold the inputs and results, ensuring that these mappings can be retrieved quickly when needed. This memoization unit is strategically placed before the multiplier-accumulator (MAC) stages in the processing pipeline to minimize unnecessary execution of the MAC stages for repeated or similar inputs. By doing so, it effectively reduces the overall computation time, especially in workloads with recurring patterns or similar input data. The integration of memoization with pipelining allows for the exploitation of both caching and parallelism, offering a substantial increase in throughput while maintaining the balance between computation time and memory usage [5]. A schematic representation of the fuzzy memoization and pipelining data-flow can be seen in Fig. 10, which illustrates how the data is processed within this system. This unit is placed before the multiplier-accumulator stages to avoid redundant execution of the MAC pipeline for repeated or similar inputs.

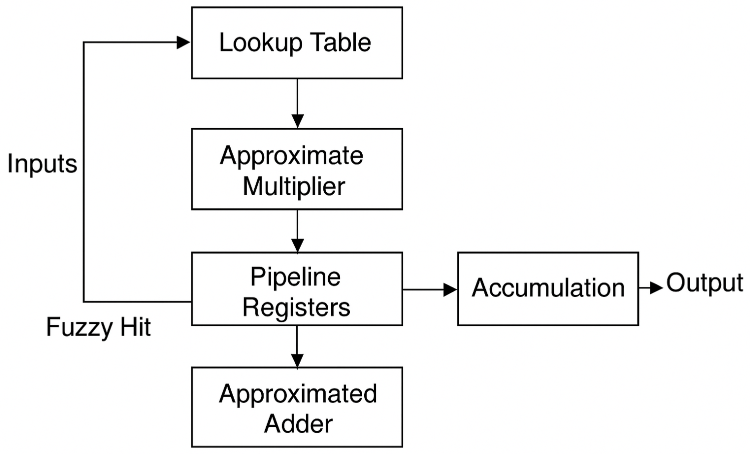

Fig. 11: Fuzzy Memoization and Pipelining Data-flow [6]

3) *Benefits and Impact*

Power Savings: By effectively avoiding redundant computations, the memoization unit reduces unnecessary switching activity, leading to lower dynamic power consumption. This reduction is particularly evident in computationally intensive workloads, where recurring operations on the same or similar inputs can be bypassed. By only computing new or mismatched inputs, the system minimizes the power expenditure associated with redundant processing, contributing to overall system power efficiency.

Latency Reduction: The use of ”fuzzy hits” significantly reduces computational latency by bypassing the full execution pipeline when a cached result is found. In scenarios where inputs match those previously encountered, the memoization unit retrieves results from memory almost instantaneously, preventing the delay associated with repeated calculations. This rapid retrieval accelerates the overall throughput of the system, especially in high-frequency or real-time processing environments.

Resource Efficiency: The reduction in computational workload afforded by fuzzy memoization leads to a more efficient utilization of hardware resources, particularly the multiplier-accumulator (MAC) stages, which are often the bottlenecks in computational pipelines. By leveraging stored results for matching inputs, the system reduces the need for excessive multiplications and additions, thereby easing the load on the processing units. This improvement not only conserves computational resources but also extends the operational lifespan of the hardware by reducing wear due to constant high-load execution. However, it is important to note that the introduction of memoization requires additional control logic and memory, which must be managed effectively to avoid negating the resource savings with excessive overhead.

IV. Implementation

A. *Implementation Overview*

This chapter provides a detailed description of the practical realisation of the design methodology outlined in Chapter 3. The primary objective of this implementation was to develop an 8-bit Dadda-based approximate Multiply-Accumulate (MAC) unit that balances computational efficiency with acceptable levels of error. The design was written in Verilog Hardware Description Language (HDL) and thoroughly simulated using Xilinx's simulation tool, ISim. For hardware synthesis and implementation, Xilinx Vivado was employed to generate a bitstream for deployment onto FPGA hardware. The implementation process involved several key phases: design entry, functional simulation, synthesis, place and route, and testing. Each stage was crucial in ensuring that the final design met the required specifications for speed, area, and power consumption while adhering to the constraints of approximate computing.

B. *Module Integration Strategy*

The integration of the Dadda-based multiplier and the approximate LowerApproxCLA adder followed a systematic, bottom-up design approach. This methodology focused on ensuring that individual modules were thoroughly tested and verified before being integrated into higher-level functional blocks. The core objective of this strategy was to strike a balance between performance (in terms of speed and throughput), area (resource utilisation), and accuracy (error tolerance). Additionally, the modular approach facilitated easier testing, debugging, and future upgrades or modifications to the design, enabling better maintainability. The modular structure also provided a clear route for isolating and resolving issues during the integration process.

Throughout the integration process, each module received rigorous testing using comprehensive test-benches. These test-benches, designed to emulate various input scenarios, allowed for the verification of both functional correctness and timing behaviour. This iterative process ensured that the design would operate correctly when the modules were eventually integrated into the complete MAC unit.

1) *Hierarchical Design*

The overall architecture of the design was structured hierarchically, which allowed for effective management of complexity by breaking down the design into smaller, more manageable functional blocks. At the lowest level, independent modules such as the approximate multiplier, the approximate adder, and their respective submodules were developed, tested, and validated in isolation before integration. This hierarchical design approach offered several advantages: it promoted reusability of components, ensured thorough testing at each level, and enabled better isolation of errors during the debugging process.

The core arithmetic operations of the unit, including the approximate multiplier and the adder, were treated as independent elements. Each of these core modules was further subdivided into smaller submodules. For instance, the Dadda multiplier was broken down into partial product generators, 4:2 compressors, and carry look-ahead logic units, which were designed separately. The 4:2 compressors, in particular, were used extensively to reduce the number of partial products by efficiently compressing them, enabling faster reduction of the product terms.

Using Verilog, these modules were implemented in a synthesisable form, meaning they could be directly mapped to hardware. Each module was accompanied by a dedicated test-bench, ensuring that its functionality was verified before moving to the next level of integration. This modular and hierarchical structure helped minimize errors and optimize the design for performance and resource usage. The approach also proved beneficial in facilitating debugging, as errors in higher-level modules could often be traced back to issues in the individual submodules.

2) *Pipelining Considerations*

To further enhance the performance of the 8-bit Dadda-based MAC unit, pipelining was incorporated at key stages of the computation process. Pipelining is a technique that allows multiple stages of computation to operate concurrently, thus improving the throughput and enabling the design to operate at higher clock frequencies. In the context of this design, pipelining was introduced after the partial product generation stage and again after the addition stage.

The decision to insert pipeline registers after the partial product generation was based on the need to ensure that the resulting partial products could be processed efficiently in subsequent stages. Similarly, pipelining was added after the addition stage to facilitate faster propagation of the sum terms. These pipeline stages were carefully chosen to balance the benefits of increased throughput with the overhead of introducing extra stages, which could potentially increase latency.

One of the main goals of this pipelining strategy was to maximise the operating frequency while maintaining a low overall latency—critical for applications requiring real-time processing, such as image filtering or signal processing tasks. Through simulations, it was determined that the added pipeline stages did not significantly affect the system's latency but allowed for a considerable increase in clock frequency, thereby improving the overall system performance.

The pipeline stages were designed to operate seamlessly with the other components of the system, ensuring that no data hazards or synchronization issues occurred. Careful attention was paid to timing analysis to guarantee that data would not be overwritten or lost during the transition between stages.

*3) **Integration Strategy***

The integration strategy also involved incorporating approximation techniques within the data-path to achieve the desired trade-off between accuracy and performance. A significant design choice was the use of approximate 4:2 compressors in the multiplier's reduction tree. Unlike traditional compressors, which ensure exact carry propagation, the approximate compressors selectively simplified the carry logic for less significant bit positions. This reduced the overall area and delay of the multiplier stage while introducing a controlled amount of error that was deemed acceptable for the target application.

Alongside the approximate compressors, an approximate adder—specifically the LowerApproxCLA adder—was employed in the addition stage. In this adder, the conventional carry-lookahead logic in the least significant bit (LSB) region was replaced by a simplified logic block. This simplification reduced the gate count and delay, contributing to faster computation. However, to preserve the accuracy of higher-order bits, the most significant bit (MSB) region retained precise carry propagation. This hybrid approach allowed for reduced resource usage and improved performance without significantly deviating from the expected results.

The architecture was specifically designed so that the flow of approximate data could be easily isolated and analysed. By doing so, it was possible to evaluate the effects of approximation on both error metrics (such as Mean Absolute Error or Mean Squared Error) and the system-level outputs. This isolation also allowed for a more controlled environment in which different approximation techniques could be tested and their impact quantified.

Ultimately, the integration strategy leveraged approximation to optimize performance in terms of speed and resource usage, while ensuring that the final results remained within acceptable error bounds. By integrating approximate components in a modular and hierarchical manner, the design achieved its goals of both high performance and low resource utilization, making it well-suited for hardware implementations where computational efficiency is critical.

V. Testing and Evaluation

A. *Introduction to testing and data processing*

This chapter presents the testing methodology, evaluation metrics, and implementation challenges encountered during the development of the proposed 8-bit Approximate MAC unit. Testing played a vital role in verifying the functionality, efficiency, and accuracy of the modules designed using approximate arithmetic principles, such as the use of approximate compressors and fuzzy memoization.

Throughout the implementation lifecycle, rigorous simulation-based testing was employed alongside hardware synthesis trials on FPGA. These efforts ensured the developed modules met performance and area efficiency requirements. To track the iterative development process and capture real-time debugging efforts, version control was employed using Git. A detailed commit history serves as a chronological log of progress, refinements and issues encountered, forming a valuable resource to retrospectively analyse implementation challenges and the effectiveness of testing strategies.

B. *Testing Methodology*

The testing strategy adopted throughout the development of the approximate 8-bit MAC units was multi-tiered, combining simulation-based verification with automated Python functional validation. This approach ensured both correctness and performance consistency across evolving stages of approximation and hardware integration.

1) *Simulation Testing in Verilog*

Initial testing was conducted through custom test-benches written in Verilog. These test-benches are tailored to the specific module under development, including the Lower-Part Approximate OR-based Carry Lookahead Adder, the Parallel Prefix Approximate Multiplier (PPAM), and the complete pipelined MAC architecture. These simulations served as the primary means to confirm correct logic behaviour, timing and synthesis compatibility.

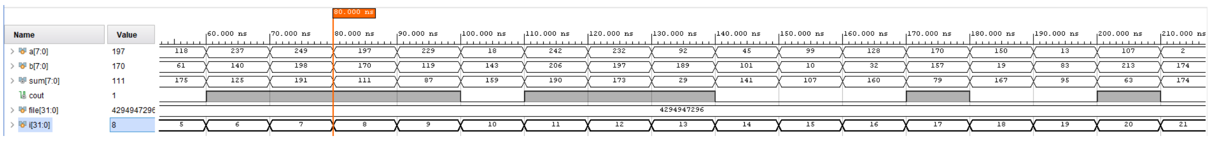

Fig. 12: Approximate Adder Simulation Waveform

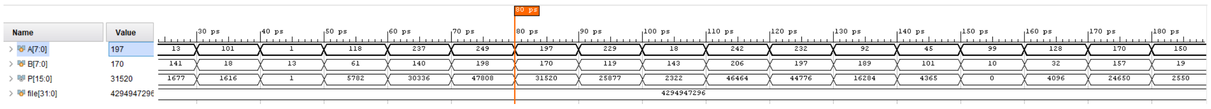

Fig. 13: Approximate Multiplier Simulation Waveform

2) *Python-Based Functional Testing*

To complement hardware-level simulations, a Python script was written to conduct automated validation. This script compared outputs from the Verilog simulation with expected values from an accurate software model. This form of golden-model testing provided a robust mechanism for checking the error introduced by approximation, as well as ensuring the correctness after implementation changes.

The test vectors generated and processed through the Python script were extensive, including random and edge-case data. This helped identify failure modes involving a case of partial integration of the PPAM, causing the results to not align with expectations. Later, a validation implementation involved more rigorous check for the CLA adder using this Python tool, verifying both functionality and error characteristics in a controlled environment.

*a) **Approximate Adder Model Testing***

TABLE I: Python unit test script results for Approximate Adder (Truncated) - Part I

Test #	a	b	Expected Sum	Expected Cout	Calculated Sum
1	00100100	10000001	10100101	0	10100101
2	00001001	01100011	01101011	0	01101011
3	00001101	10001101	10011101	0	10011101
4	01100101	00010010	01110111	0	01110111
5	00000001	00001101	00001101	0	00001101
6	01110110	00111101	10101111	0	10101111
7	11101101	10001100	01111101	1	01111101
8	11111001	11000110	10111111	1	10111111
9	11000101	10101010	01101111	1	01101111
10	11100101	01110111	01010111	1	01010111

TABLE II: Python unit test script results for Approximate Adder (Truncated) - Part II

Calculated Cout	Carry Propagation	Status	Exact Sum	Difference
0	No	Pass	10100101	0
0	No	Pass	01101100	-1
0	Propagated	Pass	10011010	3
0	No	Pass	01110111	0
0	No	Pass	00001110	-1
0	No	Pass	10110011	-4
1	Propagated	Pass	01111001	4
1	No	Pass	10111111	0
1	No	Pass	01101111	0
1	No	Pass	01011100	-5

Note: Due to space constraints, many of the values in this table have been omitted. The full table can be found in the appendix.

TABLE III: Error Difference Distribution Analysis for Approximate Adder

Difference	Frequency	Percentage
-6	4	4.00%
-5	5	5.00%
-4	7	7.00%
-3	4	4.00%
-2	14	14.00%
-1	11	11.00%
0	25	25.00%
1	1	1.00%
3	2	2.00%
4	5	5.00%
6	5	5.00%
7	8	8.00%
8	9	9.00%

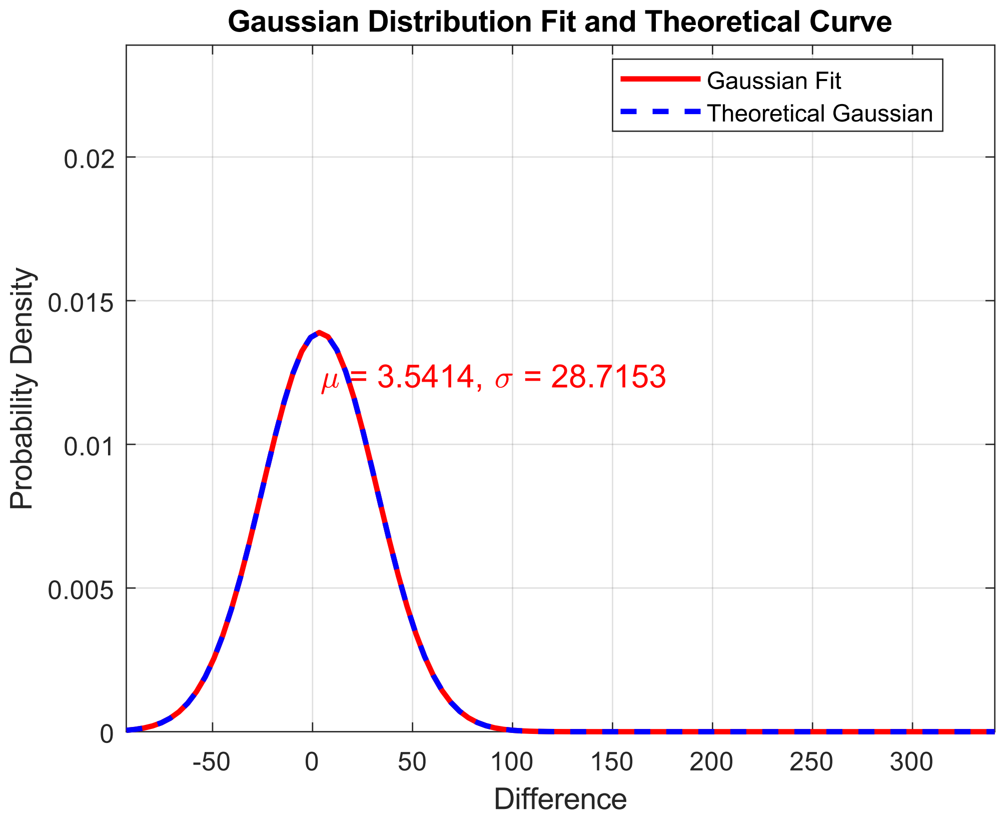

Fig. 14: Approximate Adder Gaussian Distribution Curve

b) Approximate Multiplier Model Testing

TABLE IV: Python unit test script results for Approximate Multiplier (Truncated) - Part I

Test #	A (binary)	B (binary)	Verilog Product	Calculated Product
1	00100100	10000001	0001001000100100	0001001000100100
2	00001001	01100011	0000001101101001	0000001101111011
3	00001101	10001101	0000011010001101	0000011100101001
4	01100101	00010010	0000011001010000	0000011100011010
5	00000001	00001101	0000000000000001	0000000000001101
6	01110110	00111101	0001011010010110	0001110000011110
7	11101101	10001100	0111011010000000	1000000110011100
8	11111001	11000110	1011101011000000	1100000010010110
9	11000101	10101010	0111101100100000	1000001011010010
10	11100101	01110111	0110010100010101	0110101001110011

TABLE V: Python unit test script results for Approximate Multiplier (Truncated) - Part II

Exact Product	Status	Difference
0001001000100100	Pass	0
0000001101111011	Pass	-18
0000011100101001	Pass	-156
0000011100011010	Pass	-202
0000000000001101	Pass	-12
0001110000011110	Pass	-1416
1000000110011100	Pass	-2844
1100000010010110	Pass	-1494
1000001011010010	Pass	-1970
0110101001110011	Pass	-1374

Note: Due to space constraints, many of the values in this table have been omitted. The full table can be found in the appendix.

TABLE VI: Error Difference Distribution Analysis for Approximate Multiplier

Difference	**Frequency**	**Percentage**
-3556	1	0.10%
-3542	1	0.10%
-3528	3	0.30%
-3458	1	0.10%
-3430	1	0.10%
-3416	1	0.10%
-3388	2	0.20%
-3374	1	0.10%
-3304	1	0.10%
-3290	2	0.20%
-3122	1	0.10%
-3080	1	0.10%
...continuation		
-38	2	0.20%
-36	1	0.10%
-32	3	0.30%
-28	2	0.20%
-26	2	0.20%
-24	2	0.20%
-20	1	0.10%
-18	1	0.10%
-16	3	0.30%
-14	1	0.10%
-12	1	0.10%
-8	1	0.10%
-6	1	0.10%
-4	1	0.10%
0	146	14.60%

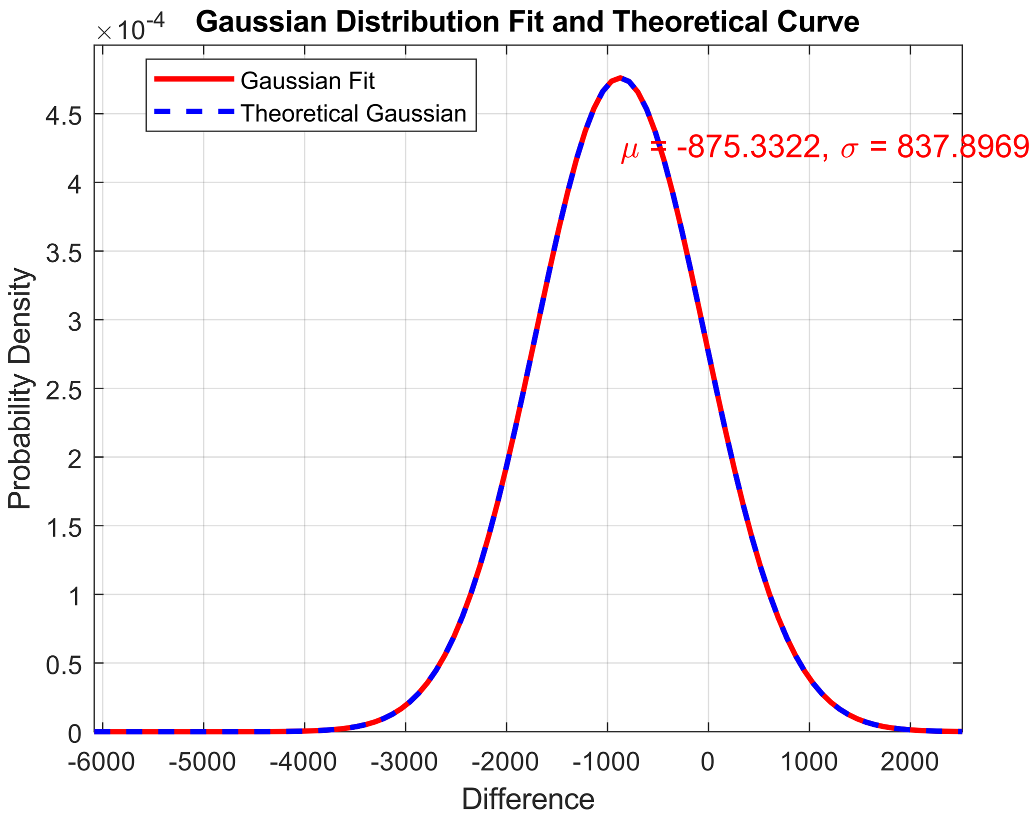

Fig. 15: Approximate Multiplier Gaussian Distribution Curve

3) *Integration Testing with Pipelined MAC*

Once both the approximate multiplier and adder components were verified, they were integrated into the pipeline MAC architecture. This integration was not without challenges. As documented, a testbench-level modification was required to accommodate larger input vector lengths, pushing the design towards a more realistic operational scale.

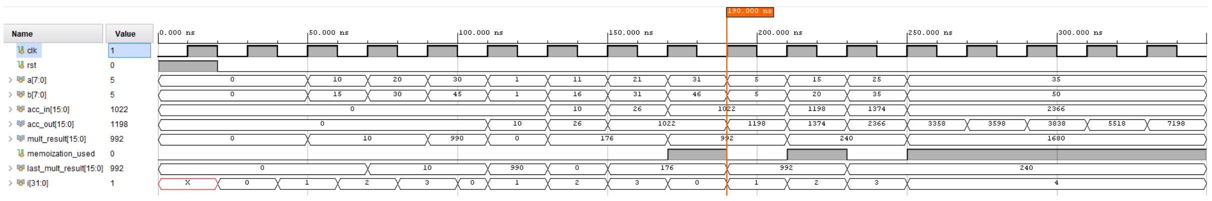

Fig. 16: Approximate MAC Integrated Simulation Waveform

C. *Implementation Challenges and Resolutions*

The development of the approximate MAC unit involved a wide range of challenges across design correctness, timing behavior, synthesis issues, and architectural calibration. These challenges were identified and documented through Git version control, enabling traceable resolution paths. Below is a comprehensive breakdown of key issues encountered during the design and implementation phases, and how they were addressed.

1) *Approximate Module Accuracy and Behavioural Bugs*

One of the earlier challenges was ensuring the approximate modules—particularly the Lower-Part Approximate OR-based Carry Lookahead Adder and the Parallel Prefix Approximate Multiplier—functioned correctly within acceptable error margins. A correction within the Carry Lookahead Adder was documented, where behavioral inconsistencies were identified during simulation testing. This required a reassessment of the internal logic structure and a modification of the OR-based carry approximation path to ensure accurate summation in the lower bits.

The PPAM, implemented as part of the approximate Dadda multiplier, also posed a unique challenge in balancing hardware reduction steps with accuracy. During initial testing, it was noted that the multiplier failed to match expected outputs, leading to an enhancement in the Python validation script that enabled clear statistical analysis of the mismatch rate.

2) *Functional Integration and Testbench Constraints*

During module integration, the testbench was updated to accommodate longer simulation cycles and larger input datasets. Previously, the testbench was limited to test lengths that had rare failure conditions, which only emerged with large-scale testing. This change enabled more robust debugging of edge cases and better evaluation of accumulated approximation error over continuous operations.

Furthermore, the challenges in combining approximate modules into a pipelined structure required new signal alignment and reset logic. Getting the approximate multiplier and adder to synchronize effectively within the pipeline's latency was particularly demanding. The solution involved fine-tuning intermediate register stages and verifying alignment across clock domains.

3) *Fuzzy Memoization Calibration*

The fuzzy memoization unit presented a unique challenge. Although initial integration was successful, it was evident that memoization required calibration to ensure that repeated input patterns would consistently trigger valid cached outputs without corrupting precision. This was particularly important as memoization errors could propagate through the pipeline. The calibration effort required experimentation. The design was left modular and parametric to allow further refinement post-evaluation.

D. *Evaluation of Accuracy and Performance*

This section presents a comprehensive evaluation of the approximate 8-bit MAC unit by analyzing both numerical accuracy and approximation quality. A Python-based testing framework was developed to process large sets of functional test vectors and extract relevant error metrics. The testing scripts captured discrepancies between approximate and exact operations for both the adder and the multiplier, enabling quantification of performance trade-offs introduced by the approximation stages.

1) *Approximate Adder: Accuracy Metrics*

The adder used in the MAC unit adopts an OR-based approximation in the lower 4 bits combined with an accurate upper half implemented using a CLA. The evaluation was conducted on a complete 8-bit input range using automated scripts.

TABLE VII: Approximate Adder: Accuracy Metrics

Metric	**Value**
Mean Absolute Error (MAE)	3.1300
Mean Relative Error (MRE)	0.0176
Worst Case Error (WCE)	8
Error Rate	0.00%

These results suggest that the adder exhibits relatively low error magnitudes despite the lower-bit approximation. The error rate of 0.00% indicates that all computed sums matched the expected binary results and carry-out. This demonstrates that while the internal value might deviate slightly from the exact sum, the overall functionality and output behaviour remained stable.

2) *Approximate Multiplier: Accuracy Metrics*

The multiplier, based on a Dadda-tree structure with approximate compression and partial product optimization, was assessed using extensive test vector comparisons between the Verilog output and the exact product values.

TABLE VIII: Approximate Multiplier: Accuracy Metrics

Metric	Value
Mean Absolute Error (MAE)	873.7460
Mean Relative Error (MRE)	0.1242
Worst Case Error (WCE)	3556
Error Rate	85.40%

These figures reflect the trade-offs inherent in the approximate multiplier design. While the MAE and MRE are within acceptable bounds for image processing or interference-tolerant applications, the high Worst Case Error (WCE) and Error Rate highlight the aggressive approximation applied to reduce logic complexity and improve speed.

3) Resource Utilization and Power Analysis

After completing synthesis and implementation of the design using Xilinx Vivado 2021, a detailed report on hardware resource utilization and power consumption was generated. This analysis provides critical insights into area and energy efficiency of the implemented approximate MAC unit.

TABLE IX: Hardware Resource Utilisation

Resource	Utilisation
Look-Up Tables (LUTs)	87
Flip-Flops (FFs)	64
BRAM	0
URAM	0
DSP Blocks	0

The design does not use any dedicated memory blocks (BRAM or URAM) or DSP slices, which is advantageous for low-resource FPGA deployments, as it allows these components to remain available for other system tasks.

The total estimated power consumption of the design, as reported post-implementation, is 30.531 mW. This figure encompasses both dynamic and static power contribution and reflects the energy-efficient nature of the approximate arithmetic units integrated into the MAC architecture.

E. Summary

This chapter provided an in-depth analysis of the testing methodology, implementation challenges, and performance evaluation of the designed approximate 8-bit Multiply-Accumulate (MAC) unit. Testing was carried out using a robust Python-based validation framework, which enabled automated, large-scale functional testing and error analysis.

The testing methodology ensured full function coverage of the MAC unit under varied input conditions, comparing the output of approximate modules with accurate, software-based references. This approach enabled the identification of function correctness, as well as quantification of approximation-induced errors.

Through the implementation challenges and resolution section, this chapter also reflected on real-world development complexities, such as simulation mismatches, corner case handling, and pipeline-related debugging. These insights were directly drawn from version control logs and served to document the iterative improvements made to the design.

The performance evaluation highlighted the contrast between the adder and multiplier in terms of approximation severity. The approximate adder introduced a very low Mean Absolute Error (MAE = 3.1300) and exhibited zero observed output mismatches (Error Rate = 0.00%), indicating it could serve as a reliable drop-in replacement in many systems. In contrast, the approximate multiplier produced a higher MAE of 873.7460 and a high error rate (85.40%), pointing to its suitability primarily in error-tolerant applications such as image filtering or AI workloads.

Overall, the approximate MAC unit demonstrates a promising trade-off between hardware efficiency and computational accuracy, validating the original design goals. The findings from this chapter lay the groundwork for assessing area, power, and delay characteristics in the final conclusion chapter, which will tie together the practical and theoretical aspects of the design.

VI. Conclusion

This project sits within a broader engineering context that emphasizes not only performance and efficiency, but also ethical, environmental, and sustainability considerations in digital system design. Through the strategic application of approximate computing techniques—such as the use of an OR-based Carry Lookahead Adder and a Dadda-based approximate multiplier—the project demonstrates a commitment to reducing power consumption and silicon area, thereby contributing to the development of more energy-efficient and sustainable hardware systems.

From an ethical standpoint, care was taken throughout the design and testing phases to uphold academic integrity, ensure reproducibility, and adhere to relevant codes of practice. The use of open-source tools and the transparency of the methodology support responsible research conduct, promote collaborative development, and respect software licensing norms. These decisions align with sustainable engineering practices that balance computational efficiency, system reliability, and resource optimisation.

The primary objective of this project—designing, implementing, and evaluating an approximate 8-bit Multiply-Accumulate (MAC) unit—was successfully achieved. The architecture integrates a Dadda-based approximate multiplier and a Lower-part OR Approximate Carry Lookahead Adder (LowerApproxCLA), both implemented in Verilog. Rigorous functional validation, supported by simulation-based testing and synthesis using Xilinx Vivado, provided a comprehensive understanding of the MAC unit's performance. Error analysis—using metrics such as Mean Absolute Error (MAE), Mean Relative Error (MRE), and Worst Case Error (WCE)—demonstrated that the approximate components achieved an acceptable trade-off between accuracy and hardware efficiency, making them well-suited to error-tolerant applications.

Beyond the core MAC functionality, architectural optimizations were introduced in the form of pipelining and fuzzy memoization. These enhancements improved system throughput and minimized redundant computations, respectively, without incurring significant area or power penalties. Their inclusion reflects the project's forward-thinking approach to architectural design and its adaptability to emerging computing paradigms.

Future Work

Looking ahead, several promising routes exist to expand and refine this work. One such direction is adaptive approximation, where future designs could incorporate dynamic adjustment of approximation levels based on workload intensity or precision requirements. This would enable the system to strike an optimal balance between computational efficiency and accuracy, providing adaptability on demand. Another intriguing possibility is the integration of error resilience through machine learning. By leveraging lightweight machine learning models, it may be possible to predict error tolerance dynamically and manage approximation parameters, enhancing the robustness and adaptability of approximate MAC units. Furthermore, the scalability of the design could be tested by extending it to higher bit-widths, such as 16-bit or 32-bit MAC units, which would validate the architecture's applicability in more complex digital signal processing tasks. Deployment on embedded platforms presents another exciting opportunity, where implementing the MAC unit on real-time embedded systems—such as ARM-based or FPGA-based Systems on Chips (SoCs)—would provide valuable insights into its practical use in applications like image filtering or edge AI. Finally, a comparative architecture analysis of various approximate adder and multiplier configurations across different FPGA families could offer a systematic approach to future optimisations. This would not only guide domain-specific improvements but also contribute to the standardisation of evaluation benchmarks for approximate computing designs.

In summary, this project contributes to the growing body of research in approximate computing by presenting a viable, low-power MAC design backed by strong empirical evaluation. It opens a pathway for continued innovation in the design of hardware accelerators that are not only performance-efficient but also mindful of real-world constraints and ethical engineering principles.

References

[1] A. Gupta and K. Suneja, "Hardware design of approximate matrix multiplier based on fpga in verilog," in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2020, pp. 496–498.

[2] C. Kommu and D. Rani, "High performance 3-2 compressors architectures for high speed multipliers," in *2018 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 2018, pp. 539–542.

[3] M. A. Rizwan, H. Waris, and M. Y. Qadri, ""”approximate multipliers based on low-power 4:2 compressors for error-tolerant applications”," in *2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2022, pp. 432–437.

[4] A. Najafi, A. Najafi, and S. Mirzakuchaki, "Low-power and high-performance 5:2 compressors," in *2014 22nd Iranian Conference on Electrical Engineering (ICEE)*, 2014, pp. 33–37.

[5] Y. Ono and K. Usami, "Approximate computing technique using memoization and simplified multiplication," in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2019, pp. 1–4.

[6] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 922–927, 2005.

[7] T. Nomani and M. Mohsin, "A novel approximate adder design methodology with single lut delay for fault-tolerant fpga-based systems," in *2019 Second International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)*, 2019, pp. 1–6.

[8] Y. Guo, X. Chen, Q. Zhou, and H. Sun, "Power-efficient and small-area approximate multiplier design with fpga-based compressors," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5.

[9] C. Padma, S. B. Potladurty, C. Nalini, T. Suguna, and C. Pallavi, "Efficient approximate adders for image processing applications," in *2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET)*, 2024, pp. 1–6.

[10] M. Ramasamy, G. Narmadha, and S. Deivasigamani, "Carry based approximate full adder for low power approximate computing," in *2019 7th International Conference on Smart Computing Communications (ICSCC)*, 2019, pp. 1–4.

[11] M. Chintagunta and V. R. Kota, "Compressor based approximate multipliers for neural network accelerators," in *2024 International Conference on Advancements in Power, Communication and Intelligent Systems (APCI)*, 2024, pp. 1–6.

[12] J. Mody, R. Lawand, R. Priyanka, S. Sivanantham, and K. Sivasankaran, "Study of approximate compressors for multiplication using fpga," in *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*, 2015, pp. 1–4.

[13] S. Ullah, S. S. Murthy, and A. Kumar, "Smapproxlib: Library of fpga-based approximate multipliers," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[14] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–6.

[15] A. M. Dalloo, A. Jaleel Humaidi, A. K. Al Mhdawi, and H. Al-Raweshidy, "Approximate computing: Concepts, architectures, challenges, applications, and future directions," *IEEE Access*, vol. 12, pp. 146 022–146 088, 2024.

[16] A. Gorantla, R. Kothapalli, and T. Spandana, "Developments of approximate computing: From algorithm level to system level," in *2022 International Conference on Computing, Communication and Power Technology (IC3P)*, 2022, pp. 52–56.

[17] S. Venkataramani, K. Roy, and A. Raghunathan, "Approximate computing," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, 2016, pp. 3–4.

[18] P. Yadav, A. Pandey, M. R. K., R. P. K.J., V. M.H., and N. K. Y.B., "Low power approximate multipliers with truncated carry propagation for lsbs," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 500–503.

[19] K. Neelima and Satyam, "High performance variable precision multiplier and accumulator unit for digital filter applications," in *2021 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, 2021, pp. 209–212.

[20] P. Thejaswini, J. Jose, and S. Nandi, "Energy efficient approximate macs," in *2021 IEEE 18th India Council International Conference (INDICON)*, 2021, pp. 1–6.