POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Design and Evaluation of a Precision-Scalable Block Floating-Point Multiplier

Advisors
Prof. Mario Roberto Casu
Dr. Edward Manca

Candidate
Fabio Gianino

Academic Year 2025-2026

Summary

This thesis presents the design and implementation of a high-performance Fused Multiply-Add (FMA) unit featuring a precision-scalable multiplier based on MXINTn building blocks, specifically tailored for AI accelerator applications. By adopting a modular MXINTn-based multiplication architecture, the proposed FMA enables configurable numerical precision, allowing dynamic adaptation of computational accuracy to workload requirements. This approach supports efficient execution of mixed-precision operations commonly used in neural networks and machine learning workloads.

The design focuses on modularity, hardware reusability, and efficient data-path organization to optimize area, power consumption, and performance. Precision scalability allows the architecture to trade computational accuracy for improved throughput and energy efficiency when full precision is not required. This flexibility is particularly beneficial for AI inference and training scenarios, where different layers and models can operate at varying precision levels.

Experimental evaluation demonstrates that the proposed MXINTn-based FMA achieves improved energy efficiency and area-performance trade-offs compared to conventional fixed-precision FMA designs. Scalability analysis and architectural trade-offs are discussed, providing practical design guidelines for integrating precision-scalable FMA units into next-generation AI accelerators and adaptive computing systems.

Acknowledgements

Thanks to everybody I met during this adventure. It was a big battle, and many times it even seemed impossible to me.

Thanks to my mom and my father for supporting my desire to reach this objective, no matter the difficulty and the sacrifices required.

Thanks to my siblings, and especially to Valeria, always extremely helpful and always ready to support me immediately at my first word.

Thanks to Davide for strongly supporting me in these last steps, stopping me when I was thinking about giving up and even requesting the "rinuncia agli studi". Always close to me and supportive in my life.

Thanks to Salvatore, for making me fall in love with reading, for pushing me into life, and for always supporting me.

Thanks to Sergio, for his words that made me reflect and grow, for his presence.

Thanks to Giovanni, for his understanding, support, and example of commitment and dedication.

Thanks to my aunts, who have helped me throughout my whole life.

Thanks to Sofia for the reciprocal support, the strong help and presence in difficult moments, and for always listening.

Thanks to Kevin for believing in my dream and hoping for it like I did, for the long friendship and constant support in life, not only during this study path.

Thanks to Mattione, who for years had to listen to all my complaints and still does not hate me for it, and for his lightness and humor in life.

Thanks to Adriano for his supporting words, sometimes with existential jokes that I really enjoy.

Thanks to Michelangelo, my Hard Techno master, for sharing with me beautiful moments of freedom.

Thanks to Max and Andrea, mates of this adventure from the beginning, from the written history at Casa Robotti.

Thanks to my team of OSTÈRIA, fighting together every day during a moment when I was losing strength and hope while trying to reach this goal.

Finally, thanks to all the people I met during this path who shared words of support with me, helping me believe that it was possible to reach this objective, even when we knew each other only a little.

Contents

Chapter 1

Introduction

The rapid advancement of Artificial Intelligence (AI) and Deep Learning (DL) has driven the development of specialized hardware accelerators capable of performing complex computations with high efficiency [12]. Among these, floating-point operations, especially multiply-accumulate (MAC) operations, play a central role in neural network inference and training. A common optimization in hardware is the use of **Fused Multiply-Add (FMA)** units, which combine multiplication and addition into a single operation, allowing the rounding to be performed only once [7, 9]. Deep neural networks consist of multiple layers of interconnected neurons that progressively transform input data into higher-level feature representations, as illustrated in Figure 1.1 [19].

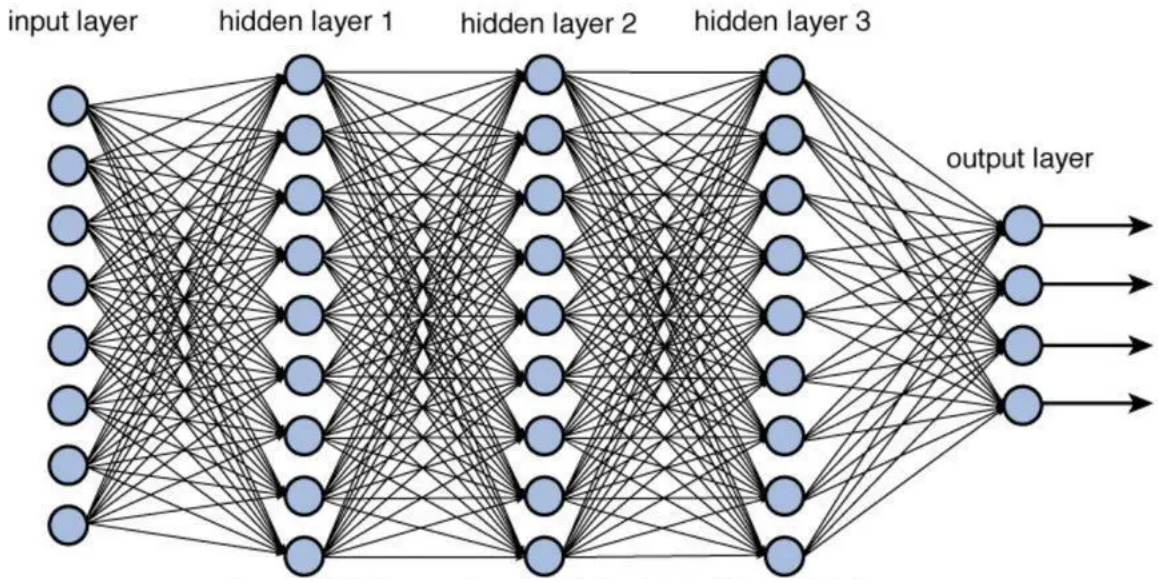

Figure 1.1: Example architecture of a deep neural network composed of an input layer, multiple hidden layers, and an output layer. Source: [19].

While conventional FMA architectures offer significant computational benefits, modern AI workloads increasingly exploit reduced-precision arithmetic to achieve higher throughput and lower energy per operation [14, 6]. Fully Connected (**FC**) layers, with large matrices of weights and activations, are particularly suited for low-precision computation [12]. Properly managed, these formats increase parallelism and hardware utilization, improving performance-per-watt with minimal impact on model accuracy.

However, numerical robustness remains essential in specific scenarios. For this reason, the proposed architecture preserves support for standard FP32 operations whenever higher precision is required, ensuring compatibility and accuracy where needed.

This thesis therefore focuses on the design and evaluation of a flexible FMA architecture capable of operating across multiple precision levels. By enabling scalable low-precision multiplications while maintaining full-precision support, the work explores the trade-off between throughput, energy efficiency, and numerical accuracy, targeting efficient acceleration of Fully Connected layers in modern AI systems.

In particular, this study exploits **MXINT block-based data operations**, where groups of values share a common exponent and are processed collectively. Using the MXINT block structure together with precision-scalable integer multipliers, the proposed architecture increases parallelism and improves hardware utilization when operating at reduced precision, while still converting the final result to standard FP32 format for further processing.

1.1 Motivation for the Study: Precision and Efficiency in AI Accelerators

The rapid growth of artificial intelligence workloads has increased the demand for hardware architectures capable of delivering **high throughput**, **energy efficiency**, and **runtime precision flexibility**. Modern computing architectures for artificial intelligence must balance the numerical robustness of high-precision arithmetic with the efficiency advantages of reduced-precision formats. Achieving this balance is the primary motivation of this work.

Several challenges drive the development of a precision-scalable floating-point FMA unit:

1. **Numerical Robustness in Accumulation:** Neural networks are dominated by multiply-accumulate operations [2]. Performing multiplication and addition separately introduces intermediate rounding errors that may accumulate across layers. Fused multiply-add (FMA) units reduce this effect by executing $A + B \times C$ with a single rounding step [7]. While FP32 ensures wide dynamic range and stability, its hardware cost motivates exploring more efficient

representations for multiplicative operands while preserving high-precision accumulation.

2. **Throughput and Energy Efficiency:** Individual FP32 multiplications are not inherently expensive. However, high-performance inference and machine learning workloads require executing many multiplications in parallel, which significantly increases area and power consumption when using full-precision units. Reduced-precision integer multipliers are smaller and more energy-efficient [5], enabling a higher degree of parallelism within the same hardware budget. By representing operands with reduced-precision integer mantissas (MXINT8, MXINT4, MXINT2) combined with a shared floating-point exponent, multiple low-precision multiplications can be executed in parallel within the same datapath, increasing computational density while preserving dynamic range through floating-point scaling.

3. **Mixed-Precision Integration:** Quantization reduces memory footprint and bandwidth but may degrade accuracy if not properly scaled and accumulated [14]. A key architectural challenge is integrating low-precision multiplicative formats with FP32 accumulation. By internally rescaling partial products into the FP32 domain before summation, the proposed architecture preserves dynamic range and limits additional precision loss.

4. **Runtime Adaptability:** Different AI workloads and network layers require different numerical precisions. Supporting both full FP32 multiplication and MXINT-based parallel modes within a unified FMA architecture enables dynamic switching between maximum numerical flexibility and maximum hardware efficiency.

Overall, this work is motivated by the need to combine floating-point accuracy and wide numerical range with integer-based efficiency. Floating-point representations enable both very large and very small values to be expressed using the same number of bits, ensuring numerical robustness across diverse workloads. The proposed precision-scalable FMA unit provides a configurable arithmetic foundation capable of adapting to the evolving precision requirements of modern AI accelerators.

1.2 Objectives of the Thesis: FMA in Fully Connected Layers and MXINT Adoption

The primary objective of this thesis is to design and evaluate a precision-scalable Floating-Point Fused Multiply-Add (FMA) unit optimized for Fully Connected (FC) layers in AI accelerators, while enabling efficient integration with MXINT-based mixed-precision computation.

More specifically, the goals of this work are summarized as follows:

1. **Analysis of Numerical Behavior in FMA Operations:** To study the numerical properties of floating-point fused multiply-add operations, including rounding mechanisms, guard/round/sticky bit handling, overflow to infinity, and invalid operations producing NaN (Not a Number) [7, 9]. The objective is to maintain consistency with the IEEE 754 standard as much as possible, even when supporting reduced-precision formats with inherently lower representational precision.

2. **Design of a High-Precision FMA Architecture:** To propose and implement an FMA unit that performs multiplication and addition with extended internal precision and a single final rounding step, ensuring that the fused operation itself complies with IEEE 754 requirements rather than treating multiplication and addition as separate rounded stages.

3. **Adoption of MXINT for Efficient Inference:** To investigate the use of MXINT representations for weights and activations in order to reduce memory footprint, bandwidth consumption, and energy usage, while leveraging high-precision FP32 accumulation to mitigate accuracy loss due to reduced-precision representations.

4. **Scaling Factor Management:** To develop and implement appropriate scaling strategies that allow MXINT block computation results to be correctly mapped into the high-precision accumulation domain, preserving dynamic range while limiting additional hardware complexity.

5. **Performance and Accuracy Evaluation:** To evaluate the proposed precision-scalable FMA architecture in terms of:

   - latency and throughput,
   - hardware resource utilization (area and timing),
   - and energy efficiency.

Through these objectives, this thesis aims to demonstrate that a carefully designed precision-scalable FMA unit, combined with MXINT-based mixed-precision computation, can effectively balance computational accuracy, numerical range, and hardware efficiency in modern AI accelerators.

1.3 Applications in AI accelerators: Neural Networks and Mixed Precision

Modern AI accelerators are specifically designed to execute the computational kernels that dominate neural network workloads [2]. Among these kernels, dense linear algebra operations (particularly matrix-vector and matrix-matrix multiplications) represent the most performance-critical components. These operations are extensively used in **Fully Connected (FC) layers**, convolutional layers (after transformation), and attention mechanisms in transformer architectures.

1.3.1 Fully Connected Layers in Neural Networks

Fully Connected layers compute a series of dot products between input activations and weight matrices [12]:

$$y_i = \sum_{j=1}^{N} w_{ij} x_j + b_i \tag{1.1}$$

Each output neuron requires multiple multiply-accumulate (MAC) operations, which can be efficiently implemented using Fused Multiply-Add (FMA) units. Given the large dimensionality of modern neural networks (especially in transformer-based models and recommendation systems) the number of FMA operations per inference pass can be extremely high.

In this context, the numerical stability of accumulation plays a crucial role. Small rounding errors introduced during intermediate multiply-add steps may accumulate over hundreds or thousands of operations. A high-precision FMA architecture with single-rounding behavior significantly improves the robustness of these dot-product computations.

1.3.2 Mixed-Precision Computation in AI Workloads

To address the increasing demand for higher throughput and lower power consumption, AI accelerators frequently adopt **mixed-precision arithmetic** [14]. Instead of relying exclusively on FP32 representations, weights and activations can be stored and processed in reduced precision formats such as INT8 or MXINT8, while accumulation is performed in higher precision (e.g., FP32 or extended precision).

Mixed precision provides several benefits:

- Reduced memory footprint and bandwidth requirements.
- Lower energy consumption per operation.
- Increased computational density and parallelism.

However, reduced-precision formats introduce challenges related to numerical accuracy. If the operand resolution is too low or the accumulation precision is insufficient, significant errors may accumulate, degrading inference performance. Therefore, it is common practice to combine low-precision operands with higher-precision accumulation, ensuring that the limited resolution of the representation does not dominate the final result.

As illustrated in Figure 1.2, neural network weights and activations often follow a distribution centered around zero. This property allows efficient quantization into low-precision integer formats using an appropriate scaling factor [20].

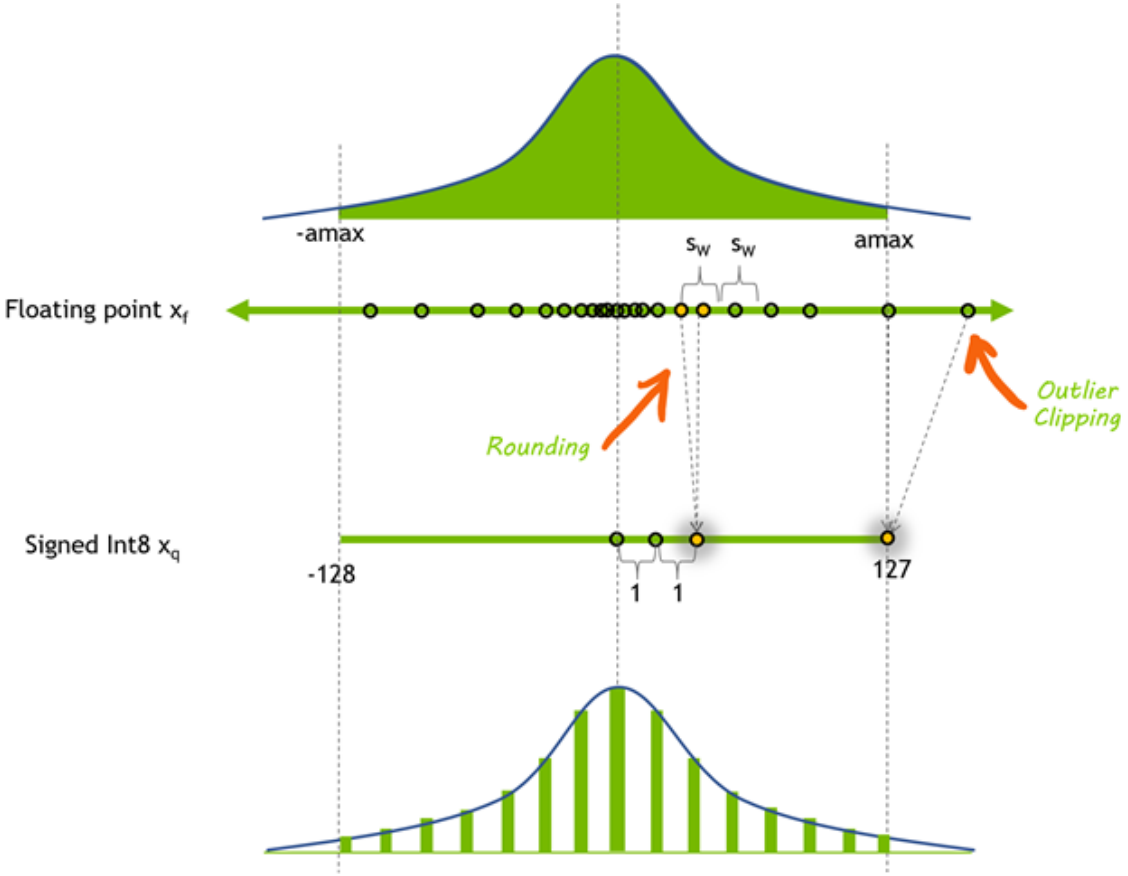

Figure 1.2: Distribution of floating-point tensor values and their mapping to signed INT8 representation through symmetric quantization. Most neural network parameters are concentrated around zero, allowing effective representation with low-precision formats. Adapted from [20].

This behavior motivates block-based integer formats such as MXINT, where groups of values share a scaling factor while the mantissas are stored as low-precision integers.

1.3.3 MXINT and Scaling Strategies

MXINT represents a structured approach to integer-based computation in AI accelerators. By associating integer values with appropriate scaling factors, real-valued neural network parameters can be approximated while maintaining efficient hardware implementation.

The correctness of MXINT-based computation depends on:

- Accurate scaling factor selection.
- Proper alignment of dynamic ranges between inputs and weights.
- High-precision accumulation to prevent overflow and rounding artifacts.

In Fully Connected layers, MXINT operands can be multiplied using integer arithmetic, while the accumulation stage benefits from extended precision—such as the high-precision FMA architecture proposed in this thesis. This hybrid approach enables efficient inference without sacrificing numerical reliability.

1.3.4 Relevance to Modern AI Accelerators

State-of-the-art AI accelerators increasingly combine specialized arithmetic units with flexible precision support [13]. Architectures designed for deep learning must simultaneously optimize:

- Throughput and parallelism,
- Energy efficiency,
- Numerical accuracy,
- Scalability to large neural models.

The integration of a high-precision FMA unit tailored for Fully Connected layers, together with MXINT8-based mixed-precision computation, addresses these design goals. It bridges the gap between algorithm-level quantization strategies and hardware-level arithmetic implementation, enabling efficient deployment of advanced neural networks in both edge and data-center environments.

1.4 Thesis structure

This thesis is organized to progressively introduce the theoretical foundations, architectural design choices, implementation details, and experimental validation of the proposed high-precision FMA unit for Fully Connected layers with MXINT support.

- **Chapter 2 – Floating-Point arithmetic, MXINT Block background, FMA units and their limitations in AI Workloads** Reviews traditional FMA implementations, IEEE 754 compliance, rounding mechanisms, and accumulation strategies. The chapter also introduces mixed-precision computation paradigms and the use of MXINT representations for efficient inference in modern AI accelerators.
- **Chapter 3 – Design of the FMA Unit and Integration of the Precision-Scalable Block Multiplier** Describes the proposed floating-point FMA architecture, including internal precision extension, multiplication and alignment strategy, accumulation scheme, rounding logic, and hardware optimizations. The integration of the FMA unit within a Fully Connected layer datapath is also detailed.
- **Chapter 4 – Implementation and Methodology** Details the hardware description flow, simulation framework, verification strategy, and validation methodology. This chapter explains how IEEE-compliant behavior is verified and how numerical comparisons with software reference models (including double-precision and fused operations) are performed.
- **Chapter 5 – Results and Performance Evaluation** Presents experimental results evaluating the proposed FMA unit in terms of throughput, latency, hardware area, power consumption, and numerical accuracy, including the performance of MXINT-based parallel multiplications with FP32 accumulation.
- **Chapter 7 – Conclusions and Future Work** Summarizes the key contributions of the thesis and outlines potential extensions, including further optimization of mixed-precision strategies, integration with larger neural network accelerators, and exploration of alternative low-precision formats.

The overall structure is designed to connect numerical theory, hardware architecture, and AI workload requirements into a coherent framework, demonstrating how high-precision FMA design and MXINT8 adoption can improve both efficiency and numerical robustness in modern AI accelerators.

Chapter 2

Floating-Point arithmetic, MXINT Block background, FMA units and their limitations in AI Workloads

2.1 Floating-Point Number Representation

Floating-point arithmetic is the standard representation for real numbers in modern computing systems, providing a wide dynamic range while maintaining reasonable precision [8]. In this work, the FP32 datapath has been designed to remain compliant with the IEEE 754 standard [7], including correct rounding semantics and handling of special values. Denormal (subnormal) numbers, however, are flushed to zero in the implemented architecture, following common industrial practice to reduce hardware complexity and latency, with negligible impact on neural network workloads.

In addition to IEEE-compliant FP32 support, next section introduces the MXINT8 reduced-precision format adopted in the proposed FMA architecture. While MXINT block representations enable improved computational density and efficiency, they are not intrinsically IEEE 754 compliant, as they rely on shared exponents and fixed-point mantissas rather than independent floating-point encodings.

2.1.1 IEEE 754 Single-Precision (FP32)

The FP32 format represents numbers using 32 bits divided into three fields as defined in the IEEE 754 standard [7]:

- **Sign (1 bit):** Determines the positivity or negativity of the number.

- **Exponent (8 bits):** Encodes the range of the number with a bias of 127.
- **Mantissa / Fraction (23 bits):** Encodes the fractional part of the significant digits.

The value of a normalized FP32 number is computed as:

$$(-1)^{\text{sign}} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$$

Significand and Exponent Bias To clarify the FP32 structure:

- **Significand (24 bits):** The full set of significant digits of the number, consisting of the 23-bit mantissa (fraction) plus the implicit leading 1 for normalized numbers. This value determines the precision of the representation.
- **Exponent Bias (127):** The exponent field uses a bias of 127, meaning the actual exponent is computed as

  $$E_{\text{unbiased}} = E_{\text{biased}} - 127$$

  This allows representation of both very large and very small numbers in a normalized way.

Special Values FP32 supports several special cases:

- **Zero:** All exponent and mantissa bits are 0. The sign bit distinguishes $+0$ and -0.
- **Infinity:** Exponent bits are all 1s, mantissa bits are 0.
- **NaN (Not a Number):** Exponent bits are all 1s, mantissa is non-zero, representing undefined or invalid operations.
- **Denormals (Subnormals):** Exponent is 0, mantissa non-zero, representing very small numbers near zero. In our design, denormals are not supported and are flushed to zero whenever produced in the datapath. This approach, commonly adopted in industry, reduces logic complexity while incurring only a negligible loss in accuracy.

Rounding FP32 operations commonly use **rounding to nearest, ties to even**, which is the default rounding mode defined in the IEEE 754 specification [7, 8]:

- The result is rounded to the nearest representable value.
- If the number is exactly halfway between two representable values, it is rounded to the one with an even least significant bit.

This reduces rounding bias in long sequences of arithmetic operations, which is particularly important in deep neural networks.

Advantages of FP32

- Wide dynamic range, able to represent very large and very small numbers.
- High numerical robustness and reproducibility.
- Supports special values (NaN, Infinity) for safe handling of exceptional cases.

Disadvantages of FP32 The complexity of floating-point operations arises from exponent alignment, normalization, and rounding stages, which significantly increase hardware cost compared to fixed-point arithmetic [9].

- High hardware cost in terms of area and power.
- Slower throughput for large-scale parallel operations due to the complexity of mantissa alignment, normalization, and rounding.

The IEEE 754 single-precision floating-point format (FP32) represents numbers using three fields: a sign bit, an exponent, and a significand. As illustrated in Figure 2.1, the format uses 1 bit for the sign, 8 bits for the exponent, and 23 bits for the mantissa, allowing a wide dynamic range of representable values [21].

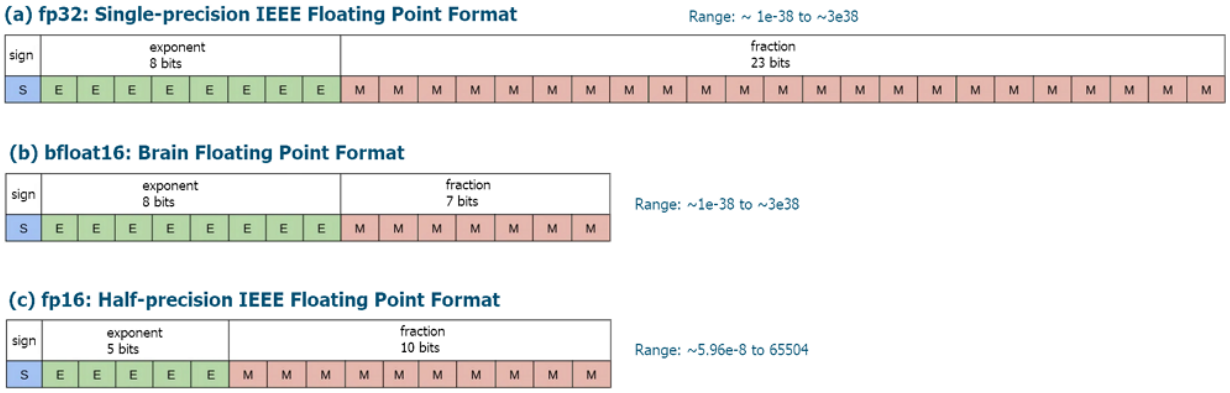

Figure 2.1: IEEE 754 single-precision (FP32) floating-point representation composed of a sign bit, an 8-bit exponent, and a 23-bit mantissa (significand). Source: [21].

2.2 MXINT Reduced-Precision Format

To improve computational throughput and hardware efficiency, the proposed architecture introduces MXINT formats, which combine a small signed integer mantissa with a shared floating-point exponent (E8M0) across a block of values. This idea builds on the concept of *block floating point* (BFP), where a group of significands share a common exponent to approximate floating-point arithmetic more efficiently in hardware [17, 16]. Block floating-point formats have recently gained attention in deep learning accelerators because they significantly reduce memory bandwidth and hardware complexity while preserving dynamic range [16, 18].

- **Mantissa (8 bits for MXINT8):** A signed integer representing the significant digits. MXINT8 uses a fixed-point format with 6 fractional bits, 1 integer-like bit (analogous to the hidden bit in unsigned FP representations), and the most significant bit as the sign. The mantissa is encoded in 2's complement.
- **Shared Exponent (8 bits, E8M0):** A single biased exponent shared across a block of MXINT operands, scaling all values in the block uniformly.

Value Representation The numerical value of an MXINT8 operand is computed as:

$$\text{value} = \text{mantissa} \times 2^{\text{shared exponent} - \text{bias}}$$

This representation combines the hardware efficiency of small integer multipliers with a floating-point-like dynamic range provided by the shared exponent (E8M0) across the block.

Special Values Since MXINT8 is an integer-based representation, it does not inherently support NaN or Infinity. Zero is represented naturally as a mantissa of 0.

Rounding Operations using MXINT8 employ **rounding to nearest, ties to even** during accumulation into the FP32 domain to preserve numerical accuracy when combining multiple low-precision products.

Advantages of MXINT8

- Very compact representation, reducing memory footprint and bandwidth requirements.
- Allows multiple parallel multiplications in the same datapath, increasing throughput.

- Shared exponent preserves a wide dynamic range across the block, combining efficiency with approximate floating-point scaling.

Disadvantages of MXINT8

- Lower per-value precision compared to FP32, which may introduce approximation error.
- Lack of native support for special values (NaN, Infinity) and subnormals.
- Requires careful management of the shared exponent to avoid overflow or underflow within a block.

The MX format organizes data into blocks where multiple elements share a common scaling factor, as illustrated in Figure 2.2. Each value is stored using a reduced-precision representation while the shared scale preserves the dynamic range of the original floating-point values [22].

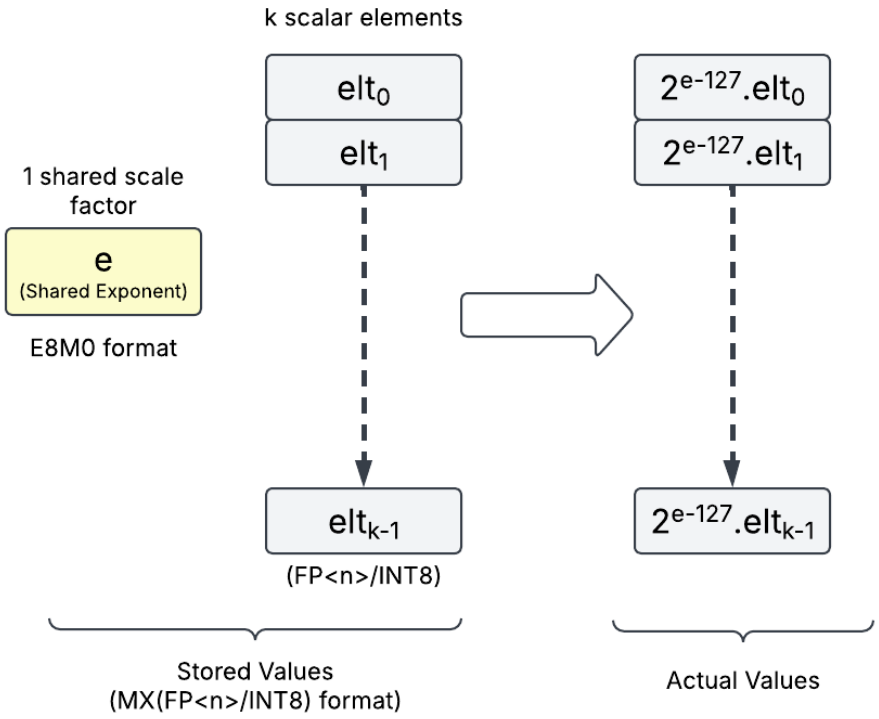

Figure 2.2: Example representation of an MX scaling block. A group of values shares a common scale factor while each element is stored using a low-precision numerical format (e.g., INT8 or FPx). Adapted from [22].

In MX formats, a tensor is partitioned into blocks of fixed size (e.g., 32 elements). Each block stores its elements using a low-precision numerical representation together with a shared scale factor. The original value can therefore be reconstructed

as the product of the stored element value and the block scale. This approach significantly reduces memory usage while maintaining sufficient dynamic range for neural network computations.

2.3 Fundamentals of Floating-Point FMA Operations

Floating-point fused multiply-add (FMA) operations are fundamental to modern numerical computing and are formally defined in the IEEE 754-2008 standard [7]. An FMA computes the expression $A + B \cdot C$ in a single, atomic operation. Unlike performing multiplication and addition separately, the fused operation performs only one final rounding step, significantly improving numerical accuracy and stability.

From a hardware perspective, an FMA unit combines three major components:

- Floating-point multiplication
- Floating-point addition (with alignment)
- Final normalization and rounding

To understand the architectural complexity of FMA units, it is necessary to analyze floating-point multiplication and addition separately.

2.3.1 Floating-Point Multiplication

The structure of floating-point multipliers and adders is extensively discussed in classical floating-point hardware literature [9, 10]. A binary floating-point number is represented as:

$$x = s \times 2^e$$

where s is the significand and e the unbiased exponent. The multiplication of two floating-point numbers follows:

$$(s_1 \times 2^{e_1}) \cdot (s_2 \times 2^{e_2}) = (s_1 \cdot s_2) \times 2^{e_1+e_2}$$

A floating-point multiplier therefore consists of three main steps:

1. **Significand multiplication:** The two p-bit significands are multiplied using integer multiplication. The product can be up to $2p$ bits wide (48 bits for FP32).
2. **Exponent computation:** Since exponents are stored with bias, the resulting exponent is computed as
$$E = E_1 + E_2 - \text{bias}$$
3. **Normalization and rounding:** The product must be normalized and rounded back to p bits.

Rounding in Multiplication Because the exact product may contain up to $2p$ bits, rounding is required. IEEE 754 rounding to nearest, ties to even is implemented using three extra bits:

- **Guard bit (g)**
- **Round bit (r)**
- **Sticky bit (s)** — logical OR of all remaining discarded bits

Guard, round, and sticky bits are commonly used in floating-point implementations to correctly implement IEEE rounding semantics [8]. The rounding decision depends on r, s, and the least significant retained bit. If rounding causes a carry-out, renormalization and exponent adjustment are required.

Overflow and Underflow Overflow occurs if the final exponent exceeds the maximum representable value after rounding. Underflow occurs when the exponent becomes too small; in high-performance hardware, denormals are often flushed to zero to reduce complexity.

2.3.2 Floating-Point Addition

Floating-point addition is more complex than multiplication because the operands may have different exponents.

Let:

$$a_1 = s_1 \times 2^{e_1}, \quad a_2 = s_2 \times 2^{e_2}$$

The addition algorithm consists of the following steps:

1. **Exponent comparison and alignment:** If $e_1 < e_2$, swap operands. Shift the smaller significand right by $d = e_1 - e_2$. During shifting, generate guard, round, and sticky bits.
2. **Sign handling:** If signs differ, subtraction is performed using two's complement.
3. **Significand addition/subtraction:** Add aligned significands.
4. **Normalization:** If carry-out occurs, shift right and increment exponent. If cancellation occurs, shift left and decrement exponent.
5. **Rounding:** Apply rounding to nearest, ties to even using GRS bits.
6. **Sign computation:** Determine the final sign depending on operand signs and cancellation.

Critical Aspects Addition requires careful handling of:

- Large exponent differences (which increase sticky-bit importance)
- Cancellation when subtracting nearly equal values
- Proper normalization before rounding

These aspects make floating-point adders more control-intensive than multipliers.

2.3.3 Fused Multiply-Add (FMA)

Fused multiply-add (FMA) units perform multiplication and addition with a single final rounding step, significantly improving numerical accuracy compared to separate operations [11]. Instead of rounding after multiplication and again after addition, FMA performs:

- Exact multiplication (full internal precision)
- Alignment with the addend
- Single final rounding

This improves numerical accuracy by eliminating one intermediate rounding step. Architecturally, an FMA requires:

- Wide internal datapaths (to hold extended product precision)
- Precise exponent comparison logic
- Efficient alignment shifters
- Robust normalization and rounding units

The IEEE 754-2008 standard formally defines the FMA operation to ensure deterministic behavior across compliant implementations [7]. In AI accelerators, FMA units form the computational core of dot products and fully connected layers, where thousands of multiply–accumulate operations are executed per cycle. Therefore, designing a numerically robust and hardware-efficient FMA architecture is critical to achieving both high throughput and high precision in neural network workloads.

Key architectural principles of floating-point FMA operations include:

- **Single Rounding Semantics:** The fused operation guarantees that only one final rounding step is performed after the complete multiply–add sequence, improving numerical accuracy compared to separate operations.

- **Extended Internal Precision:** The intermediate product is retained at full internal precision and aligned with the addend before normalization and rounding, ensuring correct handling of cancellation and exponent differences.
- **IEEE 754 Compliance:** The IEEE 754-2008 standard formally defines the fused multiply-add operation, ensuring deterministic and reproducible behavior across compliant implementations.
- **Throughput Optimization:** FMA units are deeply pipelined and designed for high reuse within dot-product structures, making them the computational core of dense linear algebra kernels in AI accelerators.
- **Precision–Efficiency Tradeoff:** Modern designs often combine reduced-precision multiplicative units with higher-precision accumulation to balance computational density, energy efficiency, and numerical robustness.

The FMA operation therefore represents the fundamental arithmetic primitive in AI accelerators. Its architectural properties directly influence both performance and numerical stability, particularly in mixed-precision datapaths where low-precision operands are accumulated in higher-precision domains.

The classical datapath of a floating-point fused multiply-add unit is shown in Figure 2.3. The architecture combines multiplication and addition into a single fused operation, avoiding intermediate rounding and improving numerical accuracy [23].

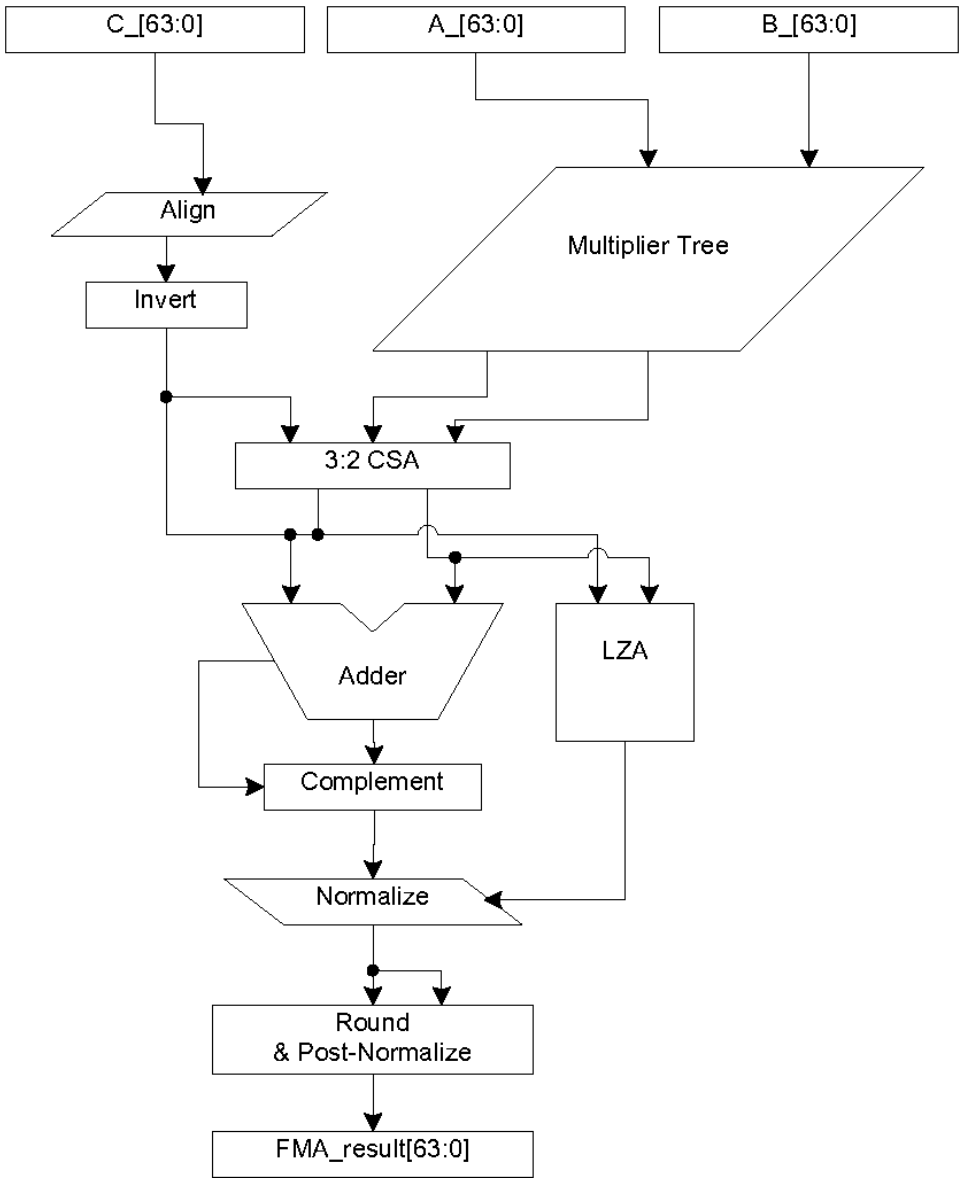

Figure 2.3: Classical floating-point fused multiply-add (FMA) architecture. The unit performs the operation $Z = (A \times B) + C$ by combining multiplication and addition into a single datapath, allowing rounding to be applied only once at the end of the computation. Adapted from [23].

The classical FMA architecture multiplies the significands of operands A and B, aligns the result with operand C, and performs the addition before a final normalization and rounding stage.

2.4 Integration of FMA in Fully Connected Layers

Dense linear algebra operations such as matrix-vector and matrix-matrix multiplication dominate the computational workload of deep neural networks [12]. Fully connected (FC) layers form the backbone of many neural network architectures, performing dense matrix-vector multiplications that dominate computation time in deep learning workloads. Integrating a high-precision floating-point FMA unit directly into these layers can significantly enhance both computational efficiency and numerical accuracy. Modern AI accelerators therefore implement large arrays of multiply–accumulate or FMA units to sustain the high throughput required by neural network inference and training [13].

2.4.1 Mapping Matrix-Vector Operations to FMA

In a fully connected layer, the output vector $\mathbf{y}$ is computed as:

$$y_i = \sum_j w_{ij} x_j + b_i$$

where w_{ij} are the weights, x_j are the input activations, and b_i is the bias. Each term $w_{ij}x_j + y_i$ is a natural candidate for a fused multiply-add operation. By performing the multiplication and accumulation in a single high-precision step, the FMA unit minimizes intermediate rounding errors, which can otherwise accumulate over many neurons and layers.

2.4.2 Benefits for Mixed-Precision Workloads

Mixed-precision computation has become a standard approach in deep learning systems to improve performance while preserving model accuracy [14]. Modern AI accelerators often employ mixed-precision computation, combining lower-precision formats (e.g., INT8 or FP16) with high-precision accumulation. Integrating FMA units allows for low-precision inputs while maintaining a high-precision internal accumulation. This approach preserves numerical stability without sacrificing throughput, making it particularly valuable for deep networks or networks sensitive to small gradient updates.

2.4.3 Hardware Considerations

Incorporating FMA units in fully connected layers requires careful hardware planning. Key considerations include:

- **Pipeline Depth:** Ensuring the FMA operations are fully pipelined to maintain high throughput across many neurons.

- **Parallelism:** Exploiting multiple FMA units in parallel to process multiple neurons simultaneously.
- **Resource Sharing:** Balancing the number of FMA units with area and energy constraints, especially in large-scale AI accelerators.
- **IEEE Compliance:** Maintaining correct rounding behavior according to IEEE-754 to prevent cumulative numerical errors.

2.4.4 Impact on Neural Network Performance

The integration of high-precision FMA units in FC layers results in:

- Reduced rounding errors and improved numerical stability.
- Higher computational density due to the fused multiply-add operation.
- Enhanced support for mixed-precision AI workloads without requiring additional conversion or scaling units.

Overall, embedding FMA directly within fully connected layers ensures that neural network computations are both accurate and efficient, laying the foundation for high-performance AI accelerators.

2.5 Limitations of Conventional FMA Units

Conventional floating-point FMA units provide high performance and improved precision over separate multiply and add operations. However, traditional floating-point units designed for general-purpose processors may not fully exploit the massive parallelism required by modern AI workloads [13]. These limitations can impact numerical accuracy, energy efficiency, and overall hardware utilization.

2.5.1 Precision Limitations

While FMA reduces rounding errors by performing multiplication and addition in a single step, the operation is still limited by the precision of the input and output formats. For example, standard FP32 FMA units cannot represent extremely large or small intermediate results without rounding, which may lead to cumulative errors in deep neural networks with many layers. In workloads with mixed precision or very high dynamic ranges, this limitation can become significant.

2.5.2 Throughput and Latency Constraints

Conventional FMA units are often optimized for general-purpose processors. As a result, they may not fully exploit parallelism in AI workloads, especially in fully connected layers or convolutional operations. Latency can also be affected if multiple FMA operations compete for the same pipeline resources, limiting overall throughput in high-performance accelerators.

2.5.3 Energy and Area Overheads

High-precision floating-point arithmetic also incurs significant energy and silicon area overhead compared to reduced-precision alternatives [15]. In large-scale AI accelerators, where thousands of FMA units may operate in parallel, this overhead can be a limiting factor for energy-efficient design.

2.5.4 Limited Flexibility for Mixed Precision

Modern AI accelerators increasingly rely on mixed-precision computation to balance performance and accuracy. Conventional FMA units, designed for single precision or double precision, often require additional hardware to support lower-precision formats (e.g., INT8 or MXINT8). This extra circuitry adds design complexity and may negate some performance advantages.

2.5.5 Scalability Challenges

Scaling conventional FMA units for large neural network layers introduces additional design challenges. As the number of parallel operations grows, ensuring consistent timing, minimizing interconnect delays, and maintaining IEEE compliance across all units becomes increasingly difficult.

Overall, while conventional FMA units offer clear advantages over separate multiply-add sequences, their limitations highlight the need for specialized architectures that address precision, efficiency, and flexibility requirements in modern AI accelerators.

2.6 Precision Challenges and Numerical Stability in AI Workloads

Artificial intelligence workloads, particularly deep neural network computations, consist primarily of large sequences of multiply–accumulate operations, typically mapped to FMA units in hardware accelerators [12, 13].

Although each FMA is individually rounded according to IEEE 754 semantics, long accumulation chains and reduced-precision operands can amplify numerical errors. For this reason, accumulation strategy and internal precision management remain critical aspects in the design of AI-oriented arithmetic units.

2.6.1 Error Propagation in Accumulations

Even with IEEE-compliant rounding, repeated floating-point operations may accumulate rounding errors, especially in long dot-product computations [8]. While each error is typically bounded by 0.5 ULP, the cumulative effect across thousands of operations can become non-negligible.

In fully connected layers, partial sums may span a wide dynamic range. When adding values of very different magnitudes, smaller contributions can be partially or completely lost due to exponent alignment shifts. This phenomenon, known as *absorption*, reduces effective precision.

Cancellation is another critical issue. When operands have opposite signs and similar magnitudes, subtraction can produce a result with many leading zeros, effectively reducing significant bits. This loss of significance increases relative error and may destabilize gradient computations during training.

2.6.2 Dynamic Range and Activation Distributions

Neural network activations and weights often follow non-uniform statistical distributions. During inference, values may cluster around zero but occasionally exhibit outliers with much larger magnitude. These dynamic range variations stress floating-point units in two ways:

- Large exponent differences during alignment cause extensive right shifts and reliance on sticky-bit logic.
- Extremely small values may underflow or become subnormal, increasing rounding sensitivity.

In reduced-precision formats, the limited exponent width exacerbates these effects, increasing the likelihood of overflow or underflow.

2.6.3 Mixed-Precision Computation

Recent work has demonstrated that mixed-precision arithmetic can maintain training accuracy while significantly improving hardware efficiency [14]. To improve efficiency, many AI accelerators employ mixed-precision arithmetic. For example, operands may be represented in reduced precision (e.g., INT8 or FP16), while accumulation occurs in higher precision (e.g., FP32). This strategy reduces memory bandwidth and multiplier cost while preserving accumulation accuracy.

However, mixed precision introduces additional considerations:

- Quantization noise in inputs
- Scaling factor management
- Format conversion overhead
- Potential mismatch between operand precision and accumulator precision

If scaling factors are poorly chosen, values may saturate or lose resolution. Frequent re-scaling also increases hardware complexity and control overhead.

2.6.4 Impact of Rounding Modes and Internal Precision

Although IEEE 754 FMA performs a single final rounding, internal datapath width strongly influences numerical stability. Insufficient guard, round, and sticky handling during alignment may introduce systematic bias. Similarly, truncation-based rounding can lead to consistent underestimation or overestimation.

For large-magnitude operands, even small rounding differences can produce measurable divergence compared to higher-precision reference models (e.g., double precision software computation). Therefore, careful management of intermediate precision is essential to minimize deviation from ideal real-number arithmetic.

2.6.5 Implications for Fully Connected Layers

Fully connected layers are particularly sensitive to precision loss because:

- They involve large dot products
- Accumulation depth may be high
- Output values propagate to subsequent nonlinear activations
- Errors may amplify across layers

In inference, moderate precision degradation may be tolerable if classification accuracy remains stable. However, during training, gradient computation is significantly more sensitive to rounding error and cancellation effects.

2.6.6 Design Considerations for Numerical Stability

To enhance numerical stability in AI workloads, hardware designers may adopt:

- Extended internal precision before final rounding
- Accurate guard, round, and sticky bit management
- Mixed-precision accumulation strategies
- Alternative numeric formats such as MXINT8 to balance dynamic range and hardware cost

Ultimately, achieving an optimal trade-off between precision and efficiency requires a holistic approach that considers arithmetic design, numeric representation, and workload characteristics. The integration of a carefully designed FMA unit within fully connected layers must therefore account not only for performance metrics but also for long-term numerical robustness across deep computational pipelines.

Chapter 3

Design of the FMA Unit and Integration of the Precision-Scalable Block Multiplier

This chapter describes the design methodology adopted for the proposed arithmetic unit. The development started from the implementation of fundamental FP32 floating-point operations, namely multiplication and addition, which were combined to build an IEEE-compliant fused multiply-add (FMA) unit.

Subsequently, a precision-scalable block multiplier was introduced to enable reduced-precision parallel computations. The outputs produced by the block multiplier are internally converted and aligned to the FP32 domain, allowing reuse of the existing accumulation, normalization, and rounding stages of the FMA unit. In this way, the original floating-point datapath is preserved while extending the architecture to support scalable precision without modifying the core accumulation structure.

3.1 FMA Architecture

The proposed architecture is centered on the design of a fully IEEE-compliant floating-point fused multiply-add (FMA) unit. The development of this unit originated from a detailed study of floating-point multiplication and addition algorithms, including normalization, alignment, and rounding mechanisms. By carefully combining these fundamental operations into a single datapath, a complete FMA implementation was realized, performing the operation $A + B \cdot C$ with a single final rounding step. This approach ensures numerical consistency with the IEEE 754 standard while providing a solid foundation for subsequent architectural extensions.

3.1.1 FP32 FMA: Starting Point

The proposed architecture is based on the realization of a fully IEEE 754–compliant floating-point fused multiply–add (FMA) unit operating in single precision (FP32). The FMA evaluates the operation

$$R = A + B \cdot C \tag{3.1}$$

while performing a single final rounding step. Unlike conventional floating-point pipelines, where multiplication and addition are executed sequentially with intermediate rounding, the fused implementation preserves intermediate precision until completion of the operation.

The design originates from an extensive study of floating-point arithmetic, following the numerical model described by Goldberg, where a floating-point operation ideally computes the exact mathematical result and subsequently rounds it to the target precision. In cascaded multiply–add implementations, the intermediate product is rounded before the addition stage, introducing an additional rounding error. The FMA avoids this limitation by merging both operations into a unified datapath.

Floating-Point Multiplication A floating-point number can be represented as

$$x = s \times 2^e, \tag{3.2}$$

where s denotes the significand and e the exponent. Floating-point multiplication therefore consists of three main steps:

1. multiplication of the significands using integer arithmetic,
2. computation of the resulting exponent

$$e = e_1 + e_2 - \text{bias}, \tag{3.3}$$

3. normalization and rounding of the product.

The product of two p-bit significands produces up to $2p$ bits and must be reduced to the target precision. Correct rounding requires retaining additional information in the form of guard, round, and sticky bits. The sticky bit records whether any discarded lower-order bits are non-zero, allowing correct handling of halfway rounding cases according to IEEE 754 rules.

Floating-Point Addition Floating-point addition requires exponent alignment prior to significand accumulation. The operand with the smaller exponent is right-shifted, generating guard, round, and sticky bits from discarded information. Depending on operand signs, the operation may correspond to either addition or subtraction.

After significand accumulation, normalization is performed to restore the leading bit position, followed by rounding according to the active IEEE rounding mode. Goldberg demonstrates that exact computation of the infinite-precision sum is unnecessary if sufficient auxiliary bits are maintained during alignment and normalization.

Fusion into a Single Operation In a conventional datapath,

$$(A \times B) + C \tag{3.4}$$

introduces two independent rounding steps. The fused multiply–add instead computes

$$R = \mathrm{round}(A + B \cdot C), \tag{3.5}$$

thereby matching the ideal arithmetic model and improving numerical accuracy. This property is particularly beneficial in accumulation-dominated workloads such as dot products and fully connected neural-network layers.

3.1.2 Double-Precision Internal Addition and Rounding

Although operands are represented in FP32 format, the internal datapath employs extended precision during intermediate computation. This choice follows the error analysis presented by Goldberg, showing that cancellation effects during addition may expose low-order bits otherwise lost under immediate rounding.

After multiplication, the intermediate product retains extended significand precision. Instead of truncating the result, the architecture performs the addition using a wider internal accumulator equivalent to double-precision resolution.

This extended representation enables:

- reduced cancellation error,
- improved accumulation accuracy,
- correct implementation of fused rounding.

Rounding is applied only after final normalization of the accumulated result. IEEE 754 rounding modes are implemented using the guard (g), round (r), and sticky (s) bits together with the least-significant retained bit p_0. If rounding produces a carry-out, a final renormalization step adjusts both significand and exponent fields.

This approach guarantees numerical equivalence with the reference IEEE algorithm while preserving the advantages of operation fusion.

3.1.3 Managing Special Cases

Full IEEE 754 compliance requires explicit handling of exceptional numerical conditions. These cases are detected prior to entering the arithmetic datapath and handled through dedicated control logic.

Zero, Underflow, and Denormals Zero is encoded using a null exponent and significand while preserving an independent sign bit. Underflow occurs when the computed exponent falls below the minimum normalized exponent representable in FP32.

In the proposed architecture, denormal (subnormal) numbers are not supported along the datapath. Whenever an operation produces a result in the subnormal range, the value is flushed to signed zero (flush-to-zero policy). This choice simplifies normalization and alignment hardware, reduces latency, and follows a common practice in high-performance AI accelerators, where the numerical impact of subnormals is typically negligible.

Consequently, any operand detected as denormal at the input, as well as any intermediate or final underflow result, is treated as zero while preserving its sign.

Infinity and Overflow Infinity is represented by a maximal exponent and zero significand. Overflow occurs when normalization or rounding produces an exponent exceeding the representable range. In this situation, the result is converted to positive or negative infinity depending on the resulting sign and rounding mode.

Typical overflow scenarios include multiplication of large operands or carry generation during post-round normalization.

NaN Generation NaN (Not-a-Number) values are encoded using a maximal exponent and a non-zero significand. NaNs are generated whenever an undefined arithmetic operation occurs, including:

- $0 \times \infty$,
- $\infty - \infty$,
- operations involving signaling NaNs,
- invalid fused multiply–add combinations.

NaN operands propagate through the computation while preserving diagnostic information when required by the IEEE standard.

3.1.4 MXINTn Block Multiplier

The MXINTn block multiplier is derived from precision-scalable computational architectures developed for systolic-array (SA) accelerators targeting Deep Neural Network (DNN) workloads. Since matrix multiplication dominates the computational cost of modern neural networks, improving the efficiency of Multiply–Accumulate (MAC) operations represents a central design objective.

Motivation and Design Context Systolic arrays efficiently execute matrix multiplications by exploiting both spatial and temporal data reuse across Processing Elements (PEs). However, recent DNN models increasingly rely on mixed-precision computation, where different layers operate at different numerical precisions to balance accuracy, energy consumption, and memory footprint.

Supporting such variability efficiently requires hardware capable of adapting its computational precision without replicating arithmetic resources. Precision-scalable multipliers address this requirement by allowing a single hardware structure to execute either one high-precision multiplication or multiple lower-precision operations in parallel.

Sub-Unit Organization The MXINTn multiplier is composed of multiple small multiplication Sub-Units (SUs), typically low-bit integer multipliers. These SUs can be combined to form higher-precision operations or, alternatively, operate concurrently when reduced precision is selected.

When operating at lower precision, the available SUs are fully utilized to increase parallelism rather than remaining partially inactive. Compared to conventional designs that merge all partial products into a single output, the MXINTn architecture distributes computation across multiple parallel results. This organization improves internal data reuse while reducing register pressure and operand bandwidth requirements inside the PE.

Operating Modes and Precision Scalability Computation within the multiplier is controlled through configurable *operating modes*, which define:

- the number of parallel outputs produced, and
- the accumulation depth associated with each output.

Selecting appropriate operating modes for each supported precision level allows the architecture to maintain high utilization of the available SUs while minimizing internal storage requirements. As a result, the same hardware efficiently supports operand precisions ranging from low-bit integer formats up to full-precision operations.

Impact on Systolic-Array Efficiency Increasing the number of SUs within each PE enables the construction of systolic arrays composed of fewer but more capable processing elements. This approach improves area efficiency and energy utilization while preserving overall computational throughput.

By balancing internal parallelism and accumulation structure, the MXINTn multiplier achieves improved performance-per-watt and better array utilization compared to fixed-structure precision-scalable multipliers.

Integration within the FMA Datapath The MXINTn block acts as a scalable multiplication engine that can be seamlessly integrated into the floating-point FMA pipeline. Its output is converted back to FP32 format, allowing the remaining stages of the datapath (including normalization, rounding, and accumulation) to remain unchanged.

Overall, the MXINTn multiplier provides a flexible and reconfigurable computation primitive well suited for mixed-precision AI workloads while enabling efficient operation in high-frequency FMA-based accelerator architectures.

3.1.5 Pipeline Design

Initial Implementation A fully combinational implementation of the datapath was initially evaluated. Although functionally correct, this approach proved impractical, as the resulting critical path prevented the design from meeting the minimum target operating frequency.

Progressive Pipeline Refinement Timing performance was significantly improved through the introduction of pipeline registers. A first refinement added one register stage after the multiplication units and a second stage after the addition logic, increasing the achievable clock frequency to approximately 700 MHz. Despite this improvement, the obtained performance remained below the design target.

To further shorten the critical path, an additional pipeline stage was inserted after the parallel multiplier inside the `MXINTn` block, before the accumulation stage. This partitioning effectively reduced combinational delay across the datapath, enabling operation at 1 GHz.

Impact of Pipelining in AI Workloads Deep neural network computations involve a large number of multiplication operations followed by dependent accumulations, particularly in Fully Connected (FC) layers implementing dot-product evaluations. For this reason, aggressive pipelining is essential to sustain high operating frequencies while maintaining acceptable latency.

As multiplication stages become sufficiently optimized through pipelining, the accumulation network naturally emerges as the dominant contributor to the critical path.

Multiplication–Accumulation Separation An alternative architectural exploration considered separating multiplication and accumulation into independent pipeline segments. While this approach simplifies timing closure for the multiplication stage, it shifts complexity toward the accumulation logic.

Floating-point accumulation introduces additional delay due to operand alignment, carry propagation, normalization, and rounding. Moreover, the parallelism generated by the `MXINTn` block requires multiple partial products to be combined within the same cycle, further increasing accumulation complexity.

Consequently, decoupling multiplication from accumulation alone is insufficient to guarantee optimal timing performance.

Design Considerations and Future Optimization These observations indicate that effective pipeline design requires a balanced distribution of logic depth across multiplier, adder, and accumulation stages. Proper register placement is therefore critical to prevent localized timing bottlenecks.

Further improvements may be achieved through optimized accumulation schemes, such as carry-save accumulation or additional intermediate pipeline stages, aimed at reducing latency while preserving high-frequency operation.

3.2 Block Diagrams and Schematics of the FMA Unit

The internal organization of the proposed FMA unit (see Fig. 3.1) is best understood through its block diagrams and schematics, which highlight the datapath structure, control signals, and pipeline partitioning. These diagrams summarize both the floating-point computation flow and the integration of the MXINTn block multiplier.

Two control signals are central to the architecture:

- `type_sel` selects between the standard FP32 multiplier and the MXINTn block multiplier. Depending on this signal, portions of the datapath are either activated, bypassed, or reconfigured.
- `CTRL_in` selects the operating precision of the MXINTn multiplier, determining its internal parallelism and accumulation mode.

3.2.1 Overall Architecture

The FMA unit is composed of the following main functional blocks:

- **Sign, Exponent and Mantissa Extractor:** Parses the FP32 operands and extracts sign bits, biased exponents, fractional fields, and reconstructed significands. In MXINTn mode, the corresponding block mantissas and shared exponents are forwarded.
- **Special-Case Detector (Inputs):** Detects zero, infinity, NaN, and denormal inputs. Subnormal operands are flushed to zero according to the selected policy.
- **Multiplication Stage:** Performs either standard FP32 significand multiplication or MXINTn block multiplication, depending on `type_sel`.
- **MXINTn Conversion and Precision Extension:** When operating in MXINTn mode, the integer product is converted into FP32 format and its significand is extended to support high-precision accumulation.
- **Exponent and Alignment Controller:** Computes the product exponent and determines the exponent difference with respect to operand A. The resulting shift amount is used for significand alignment prior to addition.
- **Product Exception Logic:** Detects overflow, underflow, zero, infinity, and NaN conditions generated during multiplication and normalization.

- **Swap Logic:** Ensures that the operand with the larger magnitude is selected as the primary operand of the addition/subtraction stage.
- **Complement and Alignment Unit:** Performs conditional two's complement operations when subtraction is required and applies right-shift alignment to the secondary operand.
- **Leading Zero Detector (LZD):** Detects leading zeros after addition in case of cancellation, providing the shift amount required for renormalization.
- **Normalization Unit:** Shifts the result to restore normalized form ($1.xxxxx$) before rounding.
- **Rounding Unit:** Implements round-to-nearest, ties-to-even using Guard (G), Round (R), and Sticky (S) bits. Rounding may trigger exponent increment and overflow.
- **Sign Selection Logic:** Determines the final sign based on operand signs, swap decisions, and complement operations.
- **Output Packing:** Reassembles sign, biased exponent, and fraction into the final FP32 result.

3.2.2 Pipeline Organization

To sustain high throughput, multiple pipeline configurations were implemented. The most optimized version includes three internal pipeline registers; simplified variants were derived by progressively removing registers, yielding 2-stage, 1-stage, and fully combinational implementations.

The 3-register pipeline is organized as follows:

- **Stage 1 – Operand Parsing and FP32 Multiplication:** Operand extraction, input special-case detection, FP32 significand or MXINTn block multiplication.
- **Stage 2 – MXINTn Processing and Exception Handling:** MXINTn block internal accumulation, product normalization, and intermediate exception detection.
- **Stage 3 – Swap, Alignment and Addition:** Operand swapping, alignment shifting, and significand addition/subtraction.
- **Stage 4 – Normalization, Rounding and Packing:** Post-add normalization, rounding, sign selection, exponent adjustment, and final FP32 packing.

3.2.3 Functional Block Diagram

Figure 3.1 illustrates the complete functional organization of the FMA unit.

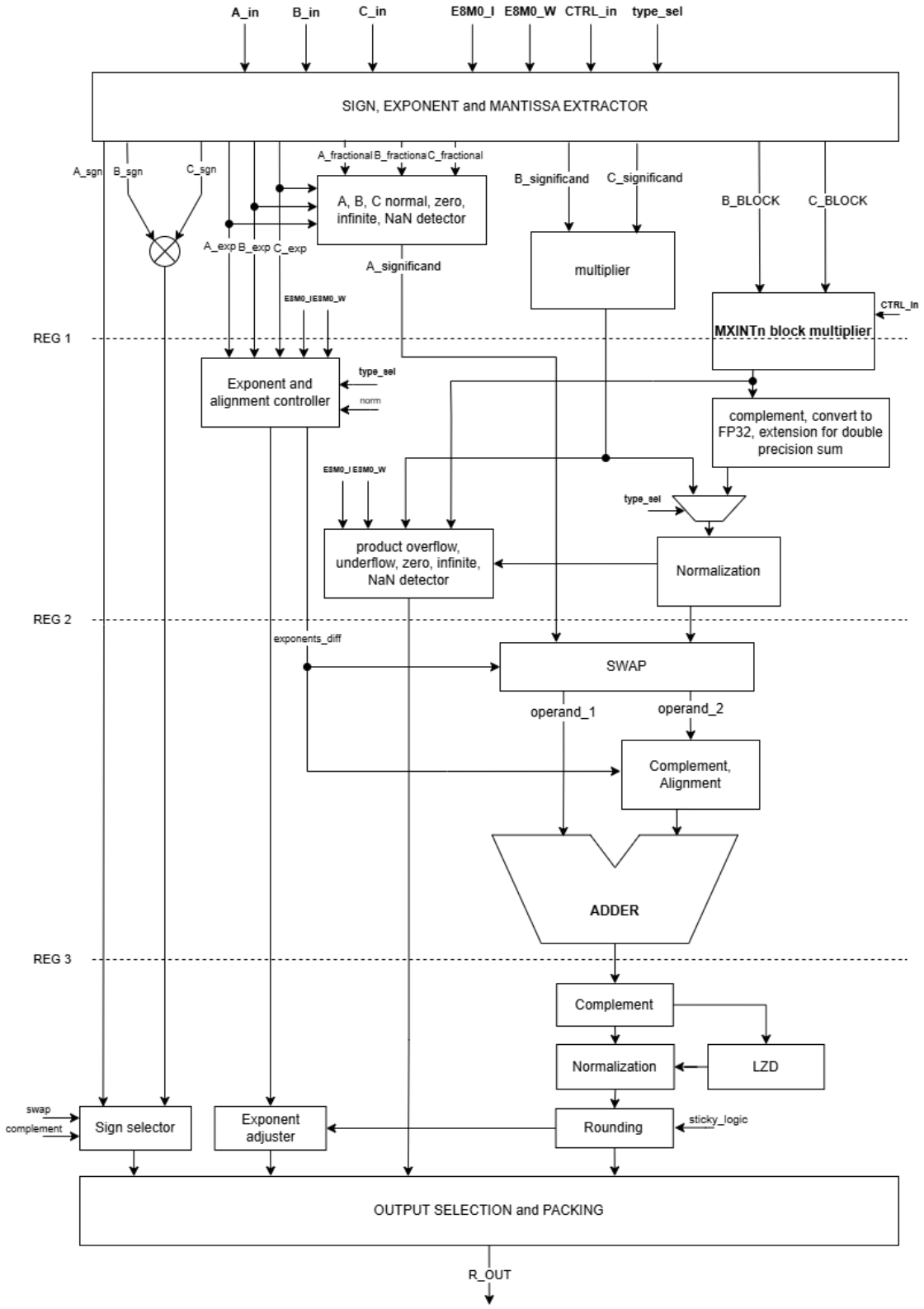

Figure 3.1: Functional block diagram of the proposed FMA unit.

The block diagrams provide a clear view of datapath partitioning, control-flow interaction, and pipeline boundaries. They serve as a reference for FPGA and ASIC implementation, ensuring that both performance targets and numerical accuracy requirements of AI workloads are satisfied.

Chapter 4

Implementation and Methodology

The implementation and methodology chapter describes the practical steps taken to realize the proposed high-precision FMA unit, as well as the tools and frameworks used for simulation, verification, and performance evaluation. The focus is on bridging the gap between the architectural design and a working hardware or software prototype, ensuring that the FMA unit can be accurately tested in realistic AI workloads.

This chapter covers the design environment, including simulation platforms, hardware description languages, and mixed-precision support. It also details the verification strategy, encompassing both functional correctness and numerical accuracy, as well as the performance evaluation approach. By providing a clear methodology, this chapter ensures that the proposed FMA architecture can be reliably implemented, tested, and analyzed for its precision, throughput, and suitability for fully connected layers in deep learning networks.

4.1 Simulation

The proposed high-precision FMA unit, including the integrated precision-scalable MXINTn block multiplier, was extensively simulated using ModelSim and Vivado. The objective of this phase was to verify numerical correctness, validate special-case handling, and ensure architectural consistency before hardware synthesis.

4.1.1 Simulation Environment

Simulation was used as a controlled validation platform prior to FPGA or ASIC deployment. All components of the architecture were described in VHDL, with selected repetitive structures and configurations automatically generated through Python scripts to improve scalability and reduce manual coding errors.

The development process started from a baseline FP32 FMA unit. To better understand internal architectural behavior, the first implementation was intentionally designed as fully combinational. Each functional block (extraction, multiplication, alignment, addition, normalization, rounding, and packing) was written as a separate, clearly structured module to simplify debugging and progressive refinement.

FP32 FMA Validation The initial validation phase focused exclusively on the standard FP32 FMA implementation. Numerical correctness was verified through a dedicated VHDL testbench with extensive signal tracing.

Golden reference results were generated using a C++ program implementing the standard `fma()` function, which is fully IEEE 754 compliant. The C++ script automatically produced both reference outputs and formatted stimulus vectors compatible with the VHDL testbench. This ensured bit-accurate comparison between hardware simulation and software reference.

Once basic arithmetic correctness was verified, full handling of special cases (zero, infinity, NaN, overflow, and underflow) was implemented. This stage significantly increased control-path complexity, since exceptional conditions can arise not only from input operands but also from intermediate normalization, rounding, and exponent adjustments.

The C++ reference environment proved essential for identifying corner cases. Numerous special scenarios were detected and corresponding corrections were introduced until the simulated hardware behavior matched the mathematical reference model.

Given the extremely large input space of FP32 operands, targeted corner-case vectors were explicitly added to the testbench to ensure proper coverage of:

- combinations of zeros, infinities, and NaNs,
- overflow and underflow boundaries,

- cancellation scenarios requiring renormalization,
- rounding edge cases (ties-to-even conditions).

Integration of the MXINTn Block Multiplier After validating the baseline FP32 FMA, the MXINTn block multiplier was integrated into the architecture.

The block multiplier was first verified independently to ensure correct functionality across all supported precision modes. Once validated in isolation, it was deployed in parallel with the existing FP32 multiplier and controlled via the `type_sel` signal.

To simplify integration, a conversion stage was introduced to transform the MXINTn output into a representation coherent with the internal FP32 datapath. This approach allowed reuse of the existing alignment, accumulation, normalization, and rounding logic with minimal structural modifications. Only additional control logic and specific exception handling were required to support the MXINTn operating modes.

Since the MXINTn block had already been validated separately, the integration phase focused primarily on:

- correct mode switching between FP32 and MXINTn,
- proper exponent reconstruction and scaling,
- interaction with normalization and rounding stages,
- preservation of IEEE-compliant behavior in FP32 mode.

4.1.2 Summary of Simulation Activities

The simulation campaign can be summarized as follows:

- **Behavioral Simulation:** VHDL testbenches exercised the FMA unit across a wide range of input combinations, comparing results against the C++ `fma()` reference implementation.
- **Bit-Accurate Numerical Validation:** Software double-precision and standard-library FMA results were used as golden references to ensure correct rounding and compliance in FP32 mode.
- **Corner-Case Analysis:** Extensive verification of overflow, underflow, cancellation, NaN propagation, infinity arithmetic, and flush-to-zero behavior for subnormals.
- **Mixed-Precision Verification:** Validation of MXINTn integration, including correct scaling, exponent handling, and interaction with the floating-point accumulation pipeline.

Overall, the simulation phase ensured that both the standard FP32 FMA and the extended mixed-precision architecture operate correctly, providing a reliable foundation for subsequent synthesis and hardware evaluation.

4.2 Synthesis

After functional validation through simulation, the design flow proceeds to the synthesis phase. During synthesis, the RTL description written in VHDL is translated into an optimized network of logic gates and registers selected from a predefined technology cell library. This step transforms the behavioral hardware description into an implementable digital circuit while respecting timing, area, and power constraints.

The synthesis process was automated using a `TCL` script, which allowed systematic exploration of different architectural configurations. In particular, the script was adapted to synthesize multiple versions of the FMA unit, ranging from a fully combinational implementation to progressively deeper pipelined variants.

To ensure synthesizability and proper timing analysis, the FMA unit was encapsulated within a top-level entity. In this wrapper module, all input signals are first captured in input registers and the final result is stored in an output register. This structure provides a well-defined synchronous interface for the synthesis tool, simplifies timing constraint specification, and allows accurate evaluation of the critical path between sequential elements.

Once the design successfully passed synthesizability checks, the following architectural configurations were evaluated:

- **Fully Combinational Implementation:**

  The first configuration implemented the entire FMA datapath without internal pipeline registers. As expected, the majority of the logic area was concentrated inside the MXINTn block multiplier and the floating-point accumulation network. Due to the extremely long critical path, the design was unable to meet the minimum timing target of 500 MHz, resulting in significant negative slack. This configuration therefore proved impractical for realistic deployment.

- **Single Pipeline Register (After Multiplication):**

  Introducing a pipeline register immediately after the multiplication stage partially reduced the critical path. However, the multiplier and the subsequent floating-point accumulation logic still formed a dominant timing bottleneck. Although some improvement in slack was observed, the design remained far from achieving the desired operating frequency.

- **Two Pipeline Registers:**

  A second register stage was introduced after the large floating-point adder used for extended-precision accumulation. While this modification reduced the depth of the arithmetic path, the improvement in maximum clock frequency was limited. The accumulation stage remained a major contributor to the critical path due to operand alignment, carry propagation, and normalization operations.

- **Three Pipeline Registers:**

  The most effective configuration was obtained by introducing a third pipeline register inside the MXINT_n block multiplier, effectively splitting the parallel multiplication network into two stages. This significantly reduced the logic depth of the most critical computation block. With this configuration, the synthesis tool reported achievable clock frequencies exceeding 1 GHz (approximately 1.1 GHz in the best case).

  Although some mixed-precision configurations still required further verification to guarantee complete result consistency, this architecture demonstrated promising performance considering the complexity of the datapath and the large number of operations executed in parallel.

4.2.1 Synthesis and Optimization

During synthesis, several optimization strategies were applied in order to balance performance and hardware cost.

- **Synthesis Constraints:**

  Timing constraints were defined to evaluate the design across a range of target clock frequencies between 500 MHz and 1.5 GHz. Area and power considerations were also monitored to guide architectural decisions.

- **Resource Optimization:**

  Arithmetic operators such as multipliers, adders, and shifters were mapped to the most efficient hardware resources available in the target technology. In particular, synthesis optimizations attempted to minimize logic duplication while preserving pipeline throughput.

- **Mixed-Precision Support:**

  Additional hardware paths were introduced to support both FP32 operations and MXINTn block multiplication modes. Conversion, alignment, and rounding logic ensure that results remain compatible with the internal floating-point representation.

4.2.2 Post-Synthesis Verification

Following synthesis, the generated gate-level netlist was verified to ensure that the optimized implementation preserves the functional behavior of the original RTL description.

- **Functional Gate-Level Simulation:**

Post-synthesis simulations were performed using the same test vectors employed during behavioral verification. This step confirms that the synthesized netlist produces results identical to the RTL model.

- **Static Timing Analysis:**

  Static timing analysis identifies the worst-case propagation paths and determines whether the design satisfies the specified clock constraints. Particular attention was given to the multiplier, alignment network, and floating-point accumulation logic.

- **Power and Area Estimation:**

  Early estimates of logic utilization, register count, and power consumption were extracted from synthesis reports to evaluate the efficiency of the architecture.

This synthesis methodology provides a clear view of the trade-offs between architectural complexity, achievable clock frequency, and hardware cost, enabling informed decisions in the design of high-performance arithmetic units.

4.2.3 Evaluated Design Parameters

The architectural exploration focused primarily on three key metrics:

- **Area:**

  Measured in terms of combinational cells and sequential cells. Due to the large amount of arithmetic logic present in the design, the additional registers introduced for pipelining have only a moderate impact on the overall silicon area.

- **Power Consumption:**

  Power estimates include switching power, internal dynamic power, and leakage power. The MXINTn block multiplier dominates the power profile due to its extensive parallel logic. However, this cost is justified by the large number of operations executed in parallel within neural network workloads.

- **Timing Performance:**

  Timing reports provide detailed information about signal arrival times and the worst-case critical path. The most relevant metric is the slack value, defined as the difference between the required arrival time and the actual propagation delay. Experiments were conducted across target clock frequencies ranging from 500 MHz to 1.5 GHz.

4.3 Verification and Testing Strategies

Comprehensive verification and testing procedures are essential to ensure that the high-precision FMA unit operates correctly and maintains numerical accuracy across all supported precision modes.

4.3.1 Unit-Level Verification

Each major functional block of the architecture was validated independently before integration into the complete pipeline.

- **MXINTn Block Multiplier:**

  The block multiplier was verified using a wide range of input vectors, including extreme values and corner cases such as zeros and small magnitudes. This ensured correct generation of partial products and proper accumulation across the scalable precision configurations.

- **Multiplier Output Conversion:**

  Additional tests verified the correctness of the conversion logic that transforms the MXINTn multiplication result into a format compatible with the FP32 datapath used for accumulation.

- **FP32 FMA Core:**

  The floating-point FMA unit was tested using extensive input sets covering standard arithmetic operations and all IEEE special cases, including NaN propagation, infinities, overflow, underflow, and flush-to-zero behavior for denormals.

- **Mixed-Precision Operation:**

  Dedicated test sequences validated the correct interaction between floating-point and MXINTn modes. Different configurations were evaluated, including FP32-only operation and mixed-precision execution with MXINT8, MXINT4, MXINT2, and other scalable precision formats.

4.3.2 Pipeline Verification

Beyond functional correctness, the entire pipeline was verified to ensure proper data propagation and timing consistency across all stages.

- **Stage-to-Stage Propagation:**

  Simulation waveforms were analyzed to verify that operands and control signals propagate correctly through each pipeline stage, from operand parsing to final rounding and packing.

- **Latency and Throughput Analysis:**

  Pipeline depth and register placement were validated to confirm that the architecture sustains the expected throughput while maintaining deterministic latency.

- **Stress and Boundary Testing:**

  Extreme operand combinations and rapid switching between precision modes were applied to identify potential timing or data-consistency issues under demanding operating conditions.

Together, these verification procedures ensure that the proposed FMA architecture operates reliably across all supported operating modes and numerical conditions.

Chapter 5

Results and Performance Evaluation

This chapter presents the results obtained from the design and implementation of the high-precision Fused Multiply–Add (FMA) unit intended for fully connected layers in deep learning workloads. The proposed architecture integrates both a standard IEEE FP32 multiplier and the MXINTn precision-scalable block multiplier, allowing the unit to support different numerical formats while maintaining high-precision accumulation.

The evaluation focuses on the main hardware metrics that characterize digital arithmetic units, namely area, power consumption, and timing performance. These metrics are analyzed in order to assess the impact of architectural choices such as pipeline depth, register placement, and the integration of the MXINTn multiplier within the FMA datapath.

All results are obtained through synthesis in a Synopsys design environment. The synthesized architectures are evaluated under different clock frequency constraints to observe the behavior of the design under varying performance requirements. The main performance indicators considered in this analysis include:

- Area versus frequency
- Power versus frequency
- Critical path and timing slack
- Power density
- Area versus power trade-off

The purpose of this evaluation is to analyze how different pipeline configurations affect the overall efficiency of the FMA unit and to identify the most balanced design point in terms of performance, hardware cost, and energy efficiency.

5.1 Area, Power and Performance of FMA Units

The performance of the proposed FMA architecture is evaluated by synthesizing several versions of the design with different pipeline configurations. Registers are inserted at different points of the datapath in order to reduce the critical path and allow the circuit to operate at higher clock frequencies.

Each architecture variant is synthesized under a set of frequency constraints ranging from 100 MHz up to 1.5 GHz. This range allows the evaluation of the design both in low-power scenarios and in high-performance configurations.

The results obtained from synthesis reports are used to analyze the relationship between timing constraints, silicon area, and power consumption.

5.1.1 Area vs Frequency

The silicon area, measured in μm^2, is a fundamental metric to evaluate the hardware cost of the proposed architecture. The synthesis results show that the achievable frequency strongly depends on the level of pipelining introduced in the design.

The fully combinational implementation of the FMA unit exhibits the longest critical path, which limits the maximum achievable clock frequency. In the analyzed design, the combinational implementation is able to meet timing constraints only up to approximately 400 MHz. Beyond this frequency, the propagation delay of the arithmetic datapath causes violations of the timing requirements.

Introducing pipeline registers reduces the length of the combinational paths by splitting the computation across multiple clock cycles. In particular, the configuration with three pipeline registers, corresponding to a four-stage pipeline, achieves frequencies up to approximately 1.2 GHz while maintaining correct timing behavior.

An additional observation is that deeper pipelining may also slightly reduce the synthesized area. This occurs because timing-driven synthesis can apply more efficient optimizations when combinational paths are shorter.

The relationship between area and target frequency is illustrated in Figure 5.1.

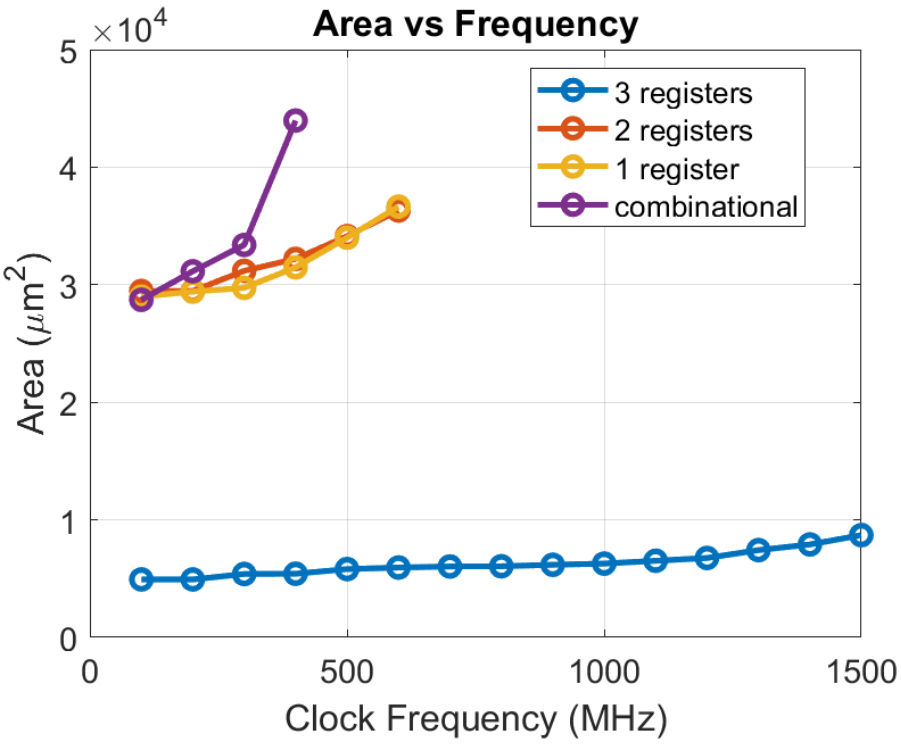

Figure 5.1: Area of the synthesized FMA architectures as a function of the target clock frequency. Deeper pipeline configurations allow the design to satisfy higher timing constraints while maintaining efficient area utilization.

5.1.2 Power vs Frequency

Power consumption is evaluated as the total power reported by the synthesis tool, measured in milliwatts (mW). The results show that power consumption increases approximately linearly with the operating frequency.

This behavior is consistent with the theoretical dynamic power model of CMOS circuits:

$$P_{dynamic} = \alpha C V^2 f$$

where α represents the switching activity factor, C is the load capacitance, V is the supply voltage, and f is the operating frequency.

Since the supply voltage and switching activity remain approximately constant across the analyzed experiments, the dominant factor influencing power consumption is the clock frequency.

The level of pipelining has a smaller direct impact on power consumption. However, deeper pipelines introduce additional registers, which increase switching activity and clock network load, resulting in a moderate increase in power consumption.

From a system-level perspective, the operating frequency must be carefully selected depending on the target application. High-frequency operation maximizes computational throughput, while lower frequencies reduce power consumption and improve energy efficiency.

The Power vs Frequency relationship is shown in Figure 5.2.

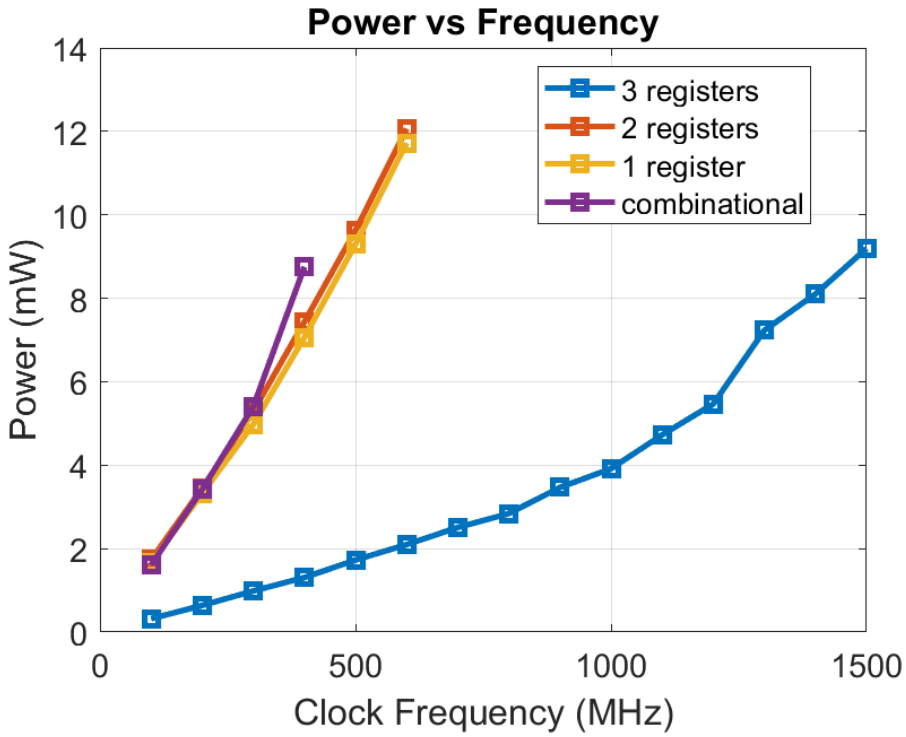

Figure 5.2: Power consumption of the FMA unit under different frequency constraints. The results show an approximately linear increase in power with the operating frequency.

5.1.3 Critical Path vs Clock Period

Timing performance is evaluated by analyzing the relationship between the critical path delay and the imposed clock period constraints. In digital design, this relationship is commonly expressed through the timing slack, defined as the difference between the required arrival time and the actual signal arrival time.

$$Slack = Required\ Time - Arrival\ Time$$

A positive slack indicates that the timing constraints are satisfied, while a negative slack indicates that the design cannot operate correctly at the specified clock frequency.

As shown in Figure 5.3, architectures with longer combinational paths fail to meet timing requirements when the clock period becomes too short. Conversely,

pipelined implementations significantly reduce the critical path delay, allowing the design to satisfy tighter timing constraints.

Among the analyzed configurations, the version with three pipeline registers provides the best timing behavior, maintaining positive slack across a wider range of clock periods.

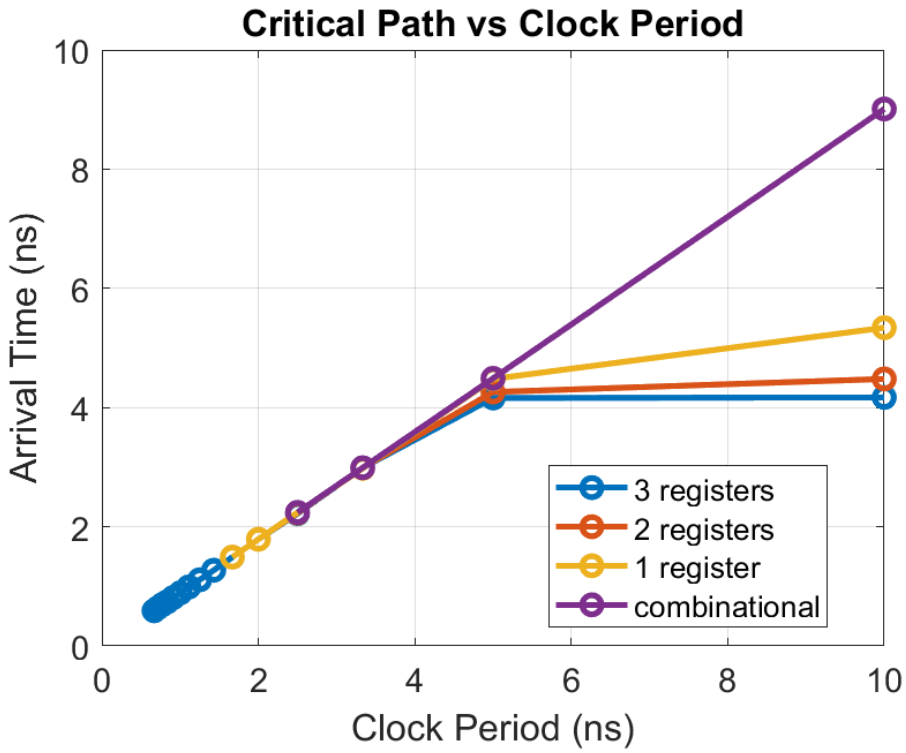

Figure 5.3: Critical path delay compared to the imposed clock period for different pipeline configurations. Deeper pipelining significantly reduces the critical path, enabling operation at higher clock frequencies.

5.1.4 Power Density vs Frequency

Power density is defined as the ratio between total power consumption and silicon area:

$$Power\ Density = \frac{Power}{Area}$$

This metric is particularly relevant in modern integrated circuits, as high power density may lead to thermal management challenges and reliability issues.

The analysis shows that both power and area increase slightly with deeper pipelining due to the introduction of additional registers and clock distribution

elements. As the operating frequency increases, dynamic power consumption also increases, resulting in higher power density values.

Therefore, while higher frequencies improve computational throughput, they also increase thermal stress and energy consumption. Designers must balance performance requirements with thermal and energy constraints when selecting the optimal operating point.

The Power Density vs Frequency relationship is illustrated in Figure 5.4.

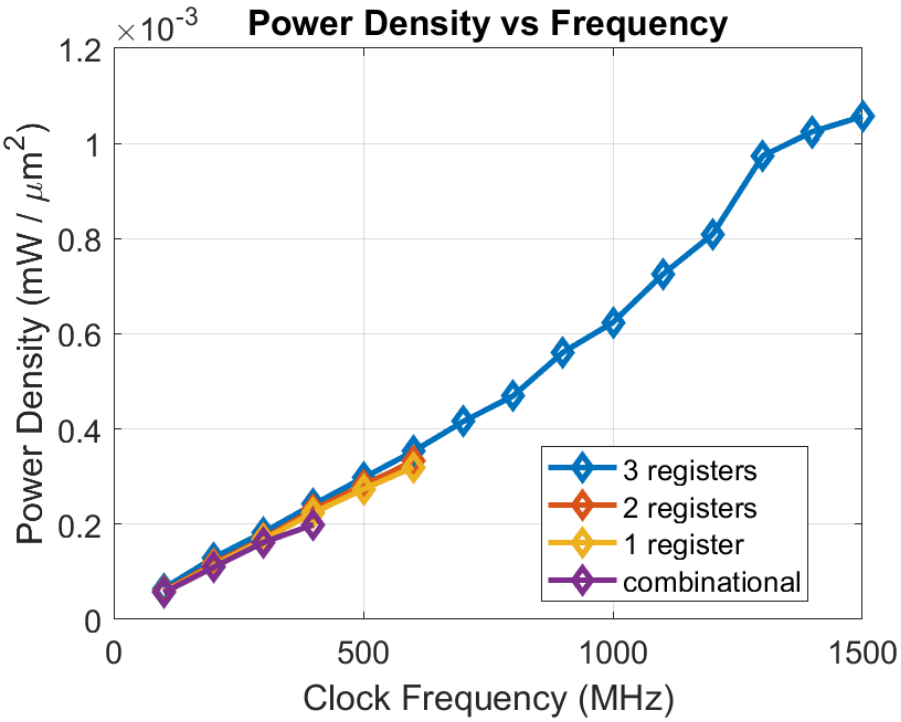

Figure 5.4: Power density of the synthesized FMA architectures as a function of the operating frequency. Higher frequencies increase dynamic power consumption, resulting in increased power density.

5.1.5 Area vs Power

The Area vs Power graph provides a useful visualization of the trade-offs between hardware cost and energy consumption. This type of analysis is commonly used to identify the most balanced architecture among several design alternatives.

As shown in Figure 5.5, pipelined architectures generally provide a better balance between power and area compared to the fully combinational implementation. Although the insertion of registers slightly increases the area, the improved timing behavior allows the unit to operate more efficiently at higher clock frequencies.

Such trade-offs are particularly relevant in large-scale AI accelerators, where thousands of arithmetic units may be instantiated on the same chip.

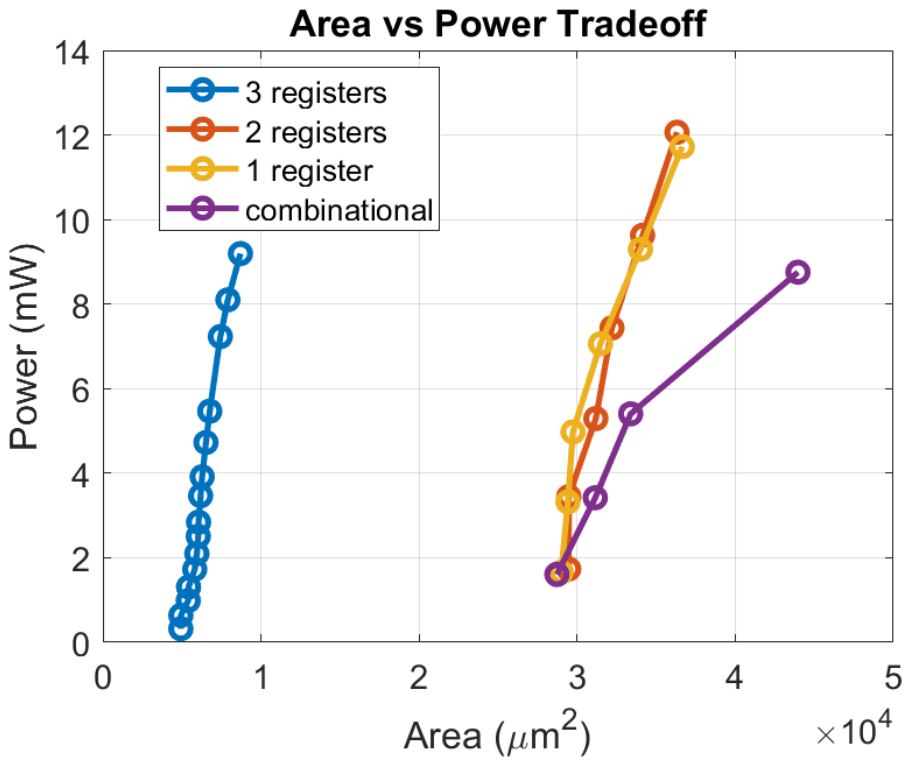

Figure 5.5: Area versus power trade-off for the evaluated FMA architectures. This graph highlights the balance between hardware cost and energy consumption for the different pipeline configurations.

Overall, the results highlight the importance of pipeline design in high-performance arithmetic units. By carefully selecting the number and placement of pipeline registers, it is possible to significantly improve the maximum operating frequency of the FMA unit while maintaining reasonable area and power overhead.

Chapter 6

Discussion and Design Trade-offs

This chapter provides a preliminary discussion of the design trade-offs encountered during the development of high-precision FMA units and MXINT8 adoption in fully connected layers. It examines the balance between computational accuracy, hardware complexity, and resource utilization, highlighting the compromises required to achieve both efficiency and precision in deep learning accelerators.

Key considerations include:

- **Hardware Complexity vs. Precision:** Increasing internal precision improves numerical accuracy but may require additional logic and deeper pipelines.
- **Throughput vs. Rounding Accuracy:** High-precision accumulation can slightly affect latency, necessitating careful pipeline design to maintain throughput.
- **Mixed-Precision Integration:** Supporting MXINTn and floating-point operations simultaneously demands flexible data paths and scaling factor management.
- **Energy and Resource Efficiency:** Optimizations must ensure that accuracy gains do not disproportionately increase power consumption or FPGA/ASIC area.

The discussion frames the design choices in terms of practical AI accelerator deployment, preparing for the detailed performance evaluation presented in the following chapters.

6.1 Hardware Efficiency vs. Computational Complexity

This section analyzes the trade-off between achieving high hardware efficiency and managing the computational complexity of the FMA units integrated in fully connected layers.

Key points include:

- **Pipeline Depth vs. Latency:** Deeper pipelines allow higher throughput but increase latency and resource usage.
- **Extended-Precision Operations:** Using double-precision internally improves accuracy but consumes more logic and power.
- **Mixed-Precision Support:** Supporting MXINTn alongside floating-point operations adds flexibility but requires additional control logic and scaling mechanisms.
- **Memory and Dataflow Optimization:** Efficient buffering and operand alignment are crucial to minimize stalls while keeping hardware resource usage low.

Balancing these factors ensures that the FMA units can deliver high computational efficiency while maintaining acceptable hardware complexity for deployment in AI accelerators.

6.2 Limitations of the Proposed FMA Architecture

Despite the high accuracy and efficiency achieved by the proposed FMA unit, several limitations remain that impact its deployment in real-world AI accelerators:

- **Resource Utilization:** The use of extended-precision arithmetic and mixed-precision support increases the area and power consumption compared to conventional FP32 FMA units.
- **Latency for Small Workloads:** While pipelining enhances throughput for large matrix operations, smaller workloads may suffer from higher relative latency.
- **Control Complexity:** Managing scaling factors for MXINTn alongside floating-point operations introduces additional control logic and timing constraints.
- **Scalability Constraints:** Integration into very large fully connected layers may require careful memory and interconnect planning to avoid bandwidth bottlenecks.
- **FPGA vs. ASIC Differences:** Some optimizations may not translate efficiently between FPGA prototyping and ASIC implementation, particularly regarding routing and power.

These limitations highlight the trade-offs involved in designing a high-precision, mixed-precision FMA unit and serve as guidance for further hardware and algorithmic improvements.

Chapter 7

Conclusions and Future Work

This chapter summarizes the main outcomes of this work and outlines possible directions for future research. The thesis focused on the design and evaluation of a high-precision Floating-Point Fused Multiply-Add (FMA) unit capable of supporting both standard FP32 operations and MXINTn mixed-precision formats. The objective was to improve the efficiency of arithmetic operations commonly used in fully connected layers of neural networks while maintaining numerical robustness.

The experimental results demonstrate that extending the internal precision of the FMA datapath effectively reduces numerical errors associated with intermediate rounding. At the same time, the integration of MXINTn block-based computation enables efficient low-precision multiplication, allowing multiple operations to be executed in parallel while preserving the dynamic range through shared scaling factors.

Overall, the proposed architecture shows that combining high-precision accumulation with scalable low-precision multiplicative formats can provide an effective balance between numerical accuracy, hardware efficiency, and computational throughput.

The main improvements observed in this work can be summarized as follows:

- **Improved Numerical Accuracy:** The fused multiply-add implementation performs multiplication and addition in a single operation with one final rounding stage. This reduces the accumulation of rounding errors that would otherwise occur when the two operations are executed separately.
- **Performance Improvements:** Pipeline optimizations allow the architecture to operate at high clock frequencies while maintaining consistent throughput. The introduction of multiple pipeline stages significantly reduces the

critical path delay, enabling efficient execution of large numbers of multiply-accumulate operations.

- **Efficient Resource Utilization:** The architecture supports both FP32 and MXINTn computation within the same configurable unit. This approach allows hardware resources to be reused across different precision modes while adding only limited overhead for higher-precision operations.
- **Robust Low-Precision Computation:** The adoption of MXINTn formats provides a structured alternative to conventional quantization approaches. By representing operands as low-precision integers combined with a shared scaling factor, efficient integer arithmetic can be used while preserving the dynamic range required for neural network computations.

In summary, the proposed design methodology demonstrates that high-precision FMA units combined with MXINTn block-based computation provide a practical solution for improving the efficiency of arithmetic operations in modern AI accelerators.

7.1 Implications for AI Accelerators: Fully Connected Layers and Mixed-Precision Performance

The results of this work have several implications for the design of future AI accelerators, particularly in the context of fully connected layers and mixed-precision computation.

Fully connected layers involve large numbers of dot-product operations, each composed of many multiply-accumulate steps. Because of the large number of sequential accumulations, these layers are particularly sensitive to numerical errors. A high-precision FMA architecture improves the reliability of these computations by minimizing intermediate rounding effects and preserving numerical stability throughout the accumulation process.

The integration of MXINTn representations further enhances computational efficiency. By storing weights and activations using low-precision integer formats combined with shared scaling factors, memory usage and data movement can be significantly reduced. At the same time, maintaining high-precision accumulation prevents the loss of accuracy that is often associated with aggressive quantization techniques.

Several architectural advantages arise from this approach:

- **Improved Numerical Stability:** High-precision accumulation reduces rounding errors in large dot-product computations, improving the reliability of neural network inference.
- **Reduced Memory Footprint:** MXINTn representations allow weights and activations to be stored in lower precision, reducing memory bandwidth requirements and improving overall system efficiency.
- **Higher Computational Density:** Low-precision integer multipliers occupy significantly less hardware area than full FP32 units, enabling higher levels of parallelism.
- **Efficient Hardware Integration:** Because the final result is converted to standard FP32 format, the proposed architecture can be integrated into existing accelerator pipelines with minimal changes.
- **Scalability to Large Neural Networks:** The combination of scalable precision and high-precision accumulation allows the architecture to support increasingly large neural network models while maintaining stable numerical behavior.

These characteristics make the proposed architecture well suited for next-generation AI accelerators that must balance performance, energy efficiency, and numerical reliability.

7.2 Future Improvements and Integration Opportunities

While the proposed FMA architecture provides strong numerical accuracy and efficient support for MXINTn computation, several directions remain for future research and optimization.

One potential improvement involves the introduction of adaptive precision control mechanisms. Dynamically adjusting the precision of intermediate computations based on workload requirements could allow hardware resources to be used more efficiently while maintaining the desired level of numerical accuracy.

Another important area concerns memory architecture. Optimizing buffer organization, memory hierarchies, and data access patterns could significantly reduce latency and energy consumption when processing large neural network layers.

Further architectural improvements may also be explored in the datapath itself. Increasing the degree of parallelism within the MXINTn block multiplier or introducing deeper pipeline stages could further improve throughput for high-performance workloads.

Additional opportunities exist at the system level. Integrating the proposed arithmetic unit with machine learning frameworks and hardware-aware compilation tools would allow software stacks to automatically exploit the mixed-precision capabilities of the architecture.

Finally, joint optimization between algorithms and hardware could provide additional benefits. By co-designing neural network training and inference techniques together with the arithmetic architecture, it may be possible to further improve both computational efficiency and model accuracy.

These research directions highlight the potential for continued development of precision-scalable arithmetic units, supporting the growing computational demands of modern artificial intelligence systems.

List of Figures

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville. "Deep Learning." MIT Press, 2016.

[2] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." Proceedings of the IEEE, 2017.

[3] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." ISCA, 2017.

[4] Paulius Micikevicius et al. "Mixed Precision Training." International Conference on Learning Representations (ICLR), 2018.

[5] Patrick Judd et al. "Stripes: Bit-Serial Deep Neural Network Computing." MICRO, 2016.

[6] Kuan Wang et al. "HAQ: Hardware-Aware Automated Quantization." CVPR, 2019.

[7] IEEE Computer Society. "IEEE Standard for Floating-Point Arithmetic (IEEE 754-2008)." IEEE, 2008.

[8] Goldberg, David. "What Every Computer Scientist Should Know About Floating-Point Arithmetic." ACM Computing Surveys, vol. 23, no. 1, 1991, pp. 5–48.

[9] Muller, Jean-Michel, et al. "Handbook of Floating-Point Arithmetic." Birkhäuser, 2010.

[10] Koren, Israel. "Computer Arithmetic Algorithms." A K Peters/CRC Press, 2002.

[11] Brisebarre, Nicolas, and Jean-Michel Muller. "Correctly Rounded Floating-Point Arithmetic." In Handbook of Floating-Point Arithmetic, Birkhäuser, 2010.

[12] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep Learning." MIT Press, 2016.

[13] Jouppi, Norman P., et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), ACM, 2017.

[14] Micikevicius, Paulius, et al. "Mixed Precision Training." International Conference on Learning Representations (ICLR), 2018.

[15] Horowitz, Mark. "1.1 Computing's Energy Problem (and What We Can Do About It)." IEEE International Solid-State Circuits Conference (ISSCC), 2014.
[16] Soloveychik, Igor, et al. "Training Deep Neural Networks with Block Floating Point Numbers." arXiv preprint arXiv:1804.01526, 2018.
[17] Wilkinson, James H. "Block Floating Point Arithmetic." Numerische Mathematik, 1960.
[18] Wang, Shibo, et al. "BFloat16: The Secret to High Performance on Cloud TPUs." Google AI Blog / related IEEE publications, 2019.
[19] Towards Data Science. "Training Deep Neural Networks." Medium, 2017.
[20] Neta Zmora, Hao Wu, Jay Rodge. "Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with TensorRT." NVIDIA Developer Blog, 2021.
[21] MetaQuotes Ltd. "From Basic to Intermediate: Floating Point." MQL5 Articles, 2024.
[22] FPX Research. "OCP MX Scaling Formats." Substack Article, 2024.
[23] Eric Quinnell, Earl E. Swartzlander Jr., Carl Lemonds. "Floating-Point Fused Multiply-Add Architectures." Proceedings of the Asilomar Conference on Signals, Systems and Computers, 2007.