A Configurable Floating-Point Fused Multiply-Add Design with Mixed Precision for AI Accelerators

Farzad Niknia, *Student Member, IEEE*, Ziheng Wang, *Student Member, IEEE*, Shanshan Liu, *Senior Member, IEEE*, Pedro Reviriego, *Senior Member*, *IEEE*, Zhen Gao, *Senior Member, IEEE*, Paolo Montuschi, *Fellow*, *IEEE*, Fabrizio Lombardi, *Life Fellow*, *IEEE*

***Abstract*—Hardware accelerators for deep learning in artificial intelligence applications must often meet stringent constraints for accuracy and throughput. In addition to architecture/algorithm improvements, high performance computational techniques such as mixed precision are also required. In this paper, a floating-point (FP) fused multiply-add (FMA) unit supporting mixed/multiple precision is proposed. A wide range of conventional FP formats (such as half and single) as well as emerging formats (including E4M3, E5M2, DLFloat, BFLoat16 and TF32) are supported in the proposed design. In addition to all these formats, the proposed design is flexible in manipulating the exponent and mantissa lengths for 8, 16 and 32-bit FP numbers based on the needs of an application. The proposed FMA can be configured to support either multiple normal FMA operations, or alternatively mixed precision in ASIC. It is fully pipelined and in each cycle, the input bit streams are processed based on the provided configuration, so independent of the previous cycles. For normal FMA operations, the proposed design utilizes sharing of resources to parallelize multiple operations based on the available hardware and required precision. For mixed precision the FMA accumulates the lower precision dot products into higher precision to avoid overflow/underflow. It improves computational accuracy by adding all possible dot products at the same time while decreasing the number of rounding operations to prevent rounding errors. An innovative method to accumulate the dot products and the aligned addend is also proposed. By considering tradeoffs between reusing the available hardware and removing unnecessary complex units, a more efficient and flexible design is attained in terms of hardware metrics and supported different precision computation compared to other designs found in the technical literature. Extensive simulation results for comparative analysis are provided.**

***Index Terms*—Artificial Intelligence (AI), Deep Learning (DL), Floating-Point (FP), Fused-Multiply-add (FMA), Multiple Precision, Mixed-Precision, Neural Networks, Arithmetic.**

I. Introduction

Machine learning (ML) and deep learning (DL) have been successfully utilized in many commercial applications; currently, DL is one of the dominant methodologies for training ML models in computer vision, natural language processing, communication systems, and speech recognition [1]-[4]. One of the most important benefits of DL is its ability to process massive amounts of data at high speeds. The complexity of DL models has rapidly grown, so the demand for designing high performance arithmetic accelerators with minimized overhead —measured in terms of figures of merit such as area, power and delay— while maintaining better accuracy has increased dramatically over the last few years [5]-[6].

Since the most common operation in DL algorithms is the multiply-accumulation (MAC), they have been extensively studied [7]-[9]. The so-called fused multiply-add units (FMA) have been utilized in general purpose CPUs, GPUs and ASIC accelerators due to their efficiency in integrating both multiplication and addition in a single design [10]-[11]. Also, the conventional IEEE-based Single precision (SP) and Double precision (DP) floating-point (FP) formats provide a sufficient dynamic range; hence, FMAs supporting these formats are commercially available, such as in NVIDIA Hooper and Ampere series [12], Intel 4th generation Xeon series [13] and AMD's 5th generation Zen processors [14]. They are often used for achieving high computation accuracy, which is an important figure of merit in different applications including DL's complex models [15]-[17]. However, except for specific calculations, the wide range of conventional FP formats like SP and DP is often not required and impose significant overheads in area and memory footprint [18]-[19]. Hence emerging formats like DLFloat [20], BFloat16 [21], TesnsorFloat32 (TF32) [22] or different 8-bit FP numbers (such as E5M2 and E4M3 [23]) are proposed recently. These formats manipulate the bit width of the exponent and mantissa provide a reasonable dynamic range and then accuracy for different applications; however, the limited dynamic range of certain formats such as 8-bit or half precision (HP) FP formats may increase the chance of overflow/underflow [24].

Manuscript received December 3, 2024, revised March 27, 2025 and accepted May 8, 2025. This research was partially supported by the Spanish Agencia Estatal de Investigación under Grants FUN4DATE (PID2022-136684OB-C22) and SMARTY (PCI2024-153434), by TUCAN6-CM (TEC-2024/COM-460), funded by CM (ORDEN 5696/2024), by the European Commission through the Chips Act Joint Undertaking project SMARTY (Grant 101140087), and by NSFC under Grant 62474030. Corresponding author: Shanshan Liu (email: ssliu@uestc.edu.cn); co-corresponding author: Zhen Gao.

Farzad Niknia, Ziheng Wang, and Fabrizio Lombardi are with Department of Electrical and Computer Engineering, Northeastern University Boston, MA 02215, USA.

Shanshan Liu is with School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China.

Pedro Reviriego is with Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad Politécnica de Madrid, Madrid 28040, Spain.

Zhen Gao is with the School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China.

Paolo Montuschi is with Dipartimento di Automatica e Informatica, Politecnico di Torino, Torina 10129, Italy.

Mixed-precision formats have been proposed to address these shortcomings [25]-[27]. Different arrangements have been investigated in the technical literatures to assess the tradeoffs between hardware metrics and model performance of DL. For example, in [29] E5M2 and E4M3 are utilized for inference and training; the results show a negligible degradation in accuracy compared to higher precision FP formats such as SP. Similarly, other schemes including HP + SP, E5M2 + Deep Learning FP (DLFloat16), Brain FP (BFloat16) + SP, SP + DP have been investigated, and their efficiency has been evaluated in [30], [31].

In this paper, we propose a high performance configurable multiple[1]/mixed-precision FMA unit. Unlike similar works found in the literature, the proposed FMA unit supports more combinations of FP formats for both normal and mixed FMA operations and provides a better flexibility in setting dynamic range at a reduced bit width. Also, the design can be configured to operate with arbitrary bit widths for the exponent and mantissa, ranging from 4-bit to 8-bit for the exponent. Therefore, the proposed design covers a variety of formats including both conventional and emerging formats; depending on the selected configuration, the largest number of dot products (utilizing the available hardware) is performed and then accumulated simultaneously. It reduces the number of rounding steps and limiting the possible rounding errors compared to recent work. An innovative method to accumulate the dot products and aligned addend is also proposed. Moreover, the proposed FMA is fully pipelined and in each cycle, the input bit streams can be processed based on the selected configuration only, so independent from the previous cycles; it processes the input bit stream as per the chosen configuration and extracts the exponent and mantissa accordingly.

In the multiple-precision mode, the proposed FMA can process four, two and one independent normal FMA operation in 8-bit (E5M2/E4M3), 16-bit (HP/DLFloat16/BFloat16) and 19/32-bit (Tensor FP(TF32)/SP) FP formats respectively. Similarly, for mixed-precision mode, a set of lower precision dot products (at the same precision) can be accumulated in a higher precision. For example, 4 independent dot products in the 8-bit FP format can be accumulated with a 16/19/32-bit FP format number; also, accumulation of 2 independent dot products in 16-bit FP format with a 19/32-bit FP format number is supported. Then, for multiple precision operations, our proposed FMA unit parallelizes the maximum number of normal FMA operations based on available resources. Similarly, in mixed-precision mode the maximum number of dot products according to the availability of resources are accumulated in a higher precision at the same time.

In summary, the technical contributions of this paper are as follows:

- A configurable FMA design is proposed; it processes FP numbers with flexible bit width for the exponent and mantissa. It can work in either multiple-precision or mixed-precision mode and support both conventional (such as HP and SP) and emerging FP formats (such as E5M2, E4M3, DLFloat, BFloat16, TF32).
- In multiple-precision mode, the proposed FMA unit parallelizes the largest possible number of multiple normal FMA operations based on the available hardware, so the design supports four, two and one independent FMA operations for all types of 8-bit, 16-bit and 19/32-bit FP formats respectively.
- In mixed-precision mode, the proposed FMA support various combinations of precision to provide better flexibility when used for applications with different accuracy requirements. It accumulates the maximum number of dot products in a higher precision depending on the available resources. Specifically, the supported combinations are 1 (16-bit) + 4 (8-bit), 1 (19/32-bit) + 4 (8-bit), 1 (19/32-bit) + 2 (16-bit). This approach can be applied to all types of format having exponent bit widths ranging from 4 to 8.
- A unique and efficient unit for segmented addition of the aligned addend and dot products is proposed for both multiple/mixed-precision modes. The effects of subnormal numbers on mixed precision and their characteristics have been investigated and considered for a more accurate alignment/addition process.

The rest of this paper is organized as follows; section II provides review and background information about the basic concepts and principles of this work. Then section III gives a general overview regarding the proposed design. Sections IV, V and VI present and elaborate more on the designs of different units of the proposed FMA. Section VII evaluates the hardware metrics and compares the proposed design with other designs found in technical literature. Finally, Section VIII concludes this paper.

II. Review

A. IEEE 754 Standard and Alternative Formats

FP format numbers are extensively utilized in computers to represent real numbers, and it has been proven to be accurate enough for majority of numerical needs [32]. All FP formats originated from the IEEE 754 standard [33] consisting of three different subfields: sign (S), exponent (E) and mantissa (M). E and M determine the range and precision respectively. The value of a FP is given by Eqn. 1:

$$\text{real value} = (-1)^{S} \times 2^{E-Bias} \times (h.M) \quad (1)$$

The value of h is the logic OR of bits in E and is referred to as the hidden bit. If it is 1 then the number is said to be normal, otherwise it is considered subnormal, and the Bias value is replaced with Bias - 1. The well-known FP formats (less/equal 64 bits) provided by the IEEE 754-2008 standard are HP, SP, DP [33]. Over the past decade most applications like DL have introduced other emerging formats due to the tradeoff between the required dynamic range, hardware metrics and accuracy. Formats introduced previously such as BFloat16, TF32, DLFloat16 and 8-bit E5M2 and E4E3 have been proposed in [18]-[23]. An overview of these formats is represented in Table

[1]In this paper, the "multiple precision" term proposes the same context as normal FMA operation. Specifically, in multiple-precision mode, the FMA can support several normal FMA operations in parallel based on the primary configuration.

TABLE I
STANDARD AND ALTERNATIVE SUPPORTED FLOATING-POINT FORMATS

Format	Length (bits)	#bits (S, E, M)	Bias	Maximum Positive	Minimum Positive
DP	64	1,11,52	1023	1.80e+308	4.94e-324
SP	32	1,8,23	127	3.40e+38	1.40e-45
TF32	19	1,8,10	127	3.40e+38	1.15e-41
BFloat16	16	1,8,7	127	3.39e+38	9.18e-41
DLFloat16	16	1,6,9	31	4.29e+9	1.82e-12
HP	16	1,5,10	15	6.55e+4	5.97e-8
E5M2	8	1,5,2	15	5.73e+4	1.52e-5
E4M3	8	1,4,3	7	2.40e+2	1.95e-3

I. The HP format has a narrow range which increases the chance of overflow/underflow. A potential solution to this problem is to use a format with a wider range, such as the SP format; however, it comes with a computational overhead of at least twice compared to HP. Another alternative is to manipulate the bit width of E and M in the HP format to make a wider range with the bit width (in this case, 16 bits). Based on Table I, TF32 and BFloat16 can almost reach the range of SP and similarly E5M2 can reach the range of HP. DLFloat16 provides a good balance for the dynamic range and precision between HP and BFloat16. Also, other formats (such as E4M3) can be utilized in combination with others in mixed precision formats as per the application.

B. Fused Multiply-add (FMA) Unit

The FMA unit has been introduced in the IBM RS/6000 processors to increase performance and accuracy of FP calculations by making the multiply-accumulation operation indivisible [34]. This feature has made FMA suitable for general-purpose processors to calculate A + (B × C) (where A, B and C are numerical data). Although the main purpose of this unit is to calculate the accumulation of products, an independent FP addition or multiplication can also be performed by setting B = 1 or C = 1 for the FP addition, and by setting A = 0 for the multiplication [35]. Different from conventional FP multiply-accumulate (MAC) units that use a separate FP multiplier and adder to perform the same calculation, FMA units integrate these operations in a single design.

The overview of a conventional 3-stage FMA design is depicted in Fig. 1; it mainly includes three stages as follows:

1) Extraction, Multiplication, Alignment Calculation and Exponent Analysis: As shown in the top part of Fig. 1, in this stage the mantissas of B and C are multiplied using a fast multiplier (e.g., Booth algorithm), producing a result in carry save add (CSA) format. During multiplication, the bias generator computes the bias values for the exponents of the product (B×C) and addend (A), considering the subnormal flag (SF or hidden bit) of each input. The temporary exponent (largest among product and addend) is selected, then the exponent difference (by including the bias values also) aligns with the addend's mantissa. For simplicity, the addend is always positioned to the left of the product, ensuring alignment uses only a right shifter. The product alignment process is illustrated in Fig. 2(a).

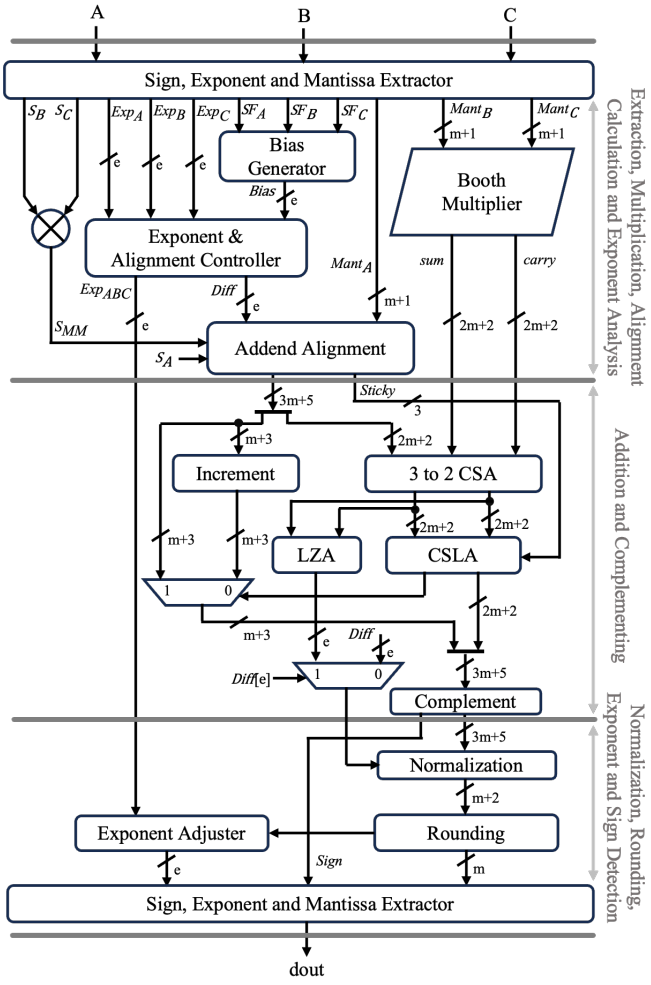


Fig. 1. Conventional 3-stage FMA design (the grey text on the right refers to the main stages of the calculation).

2) Addition and Complementing: As shown in the middle part of Fig. 1, the aligned addend and multiplication result are added using a CSA adder and converted to normal representation usually with a carry-select adder (CSLA). A leading zero anticipator (LZA) predicts leading zeros using the CSA adder output, assisting normalization if needed. This improves the critical path delay by performing it concurrent to the CSLA operation. For efficiency, only the overlapping parts of the aligned addend and product are processed by the CSLA and LZA, while non-overlapping parts are managed by an incrementor in case CSLA overflow.

3) Normalization, Routing, Exponent and Sign Detection: Normalization is based on the LZA zero count when the addend is smaller than the product (fully aligned to the right of the product's most significant bit (MSB)); otherwise, the addend's alignment amount is used. Finally, the rounding is applied followed by exponent adjustment if needed. This process is shown in the bottom part of Fig. 1.

In general, the advantages of FMA over MAC are as follows: 1) The FMA unit uses a single rounding at the end of the multiply-accumulation process, while the conventional MAC unit uses two rounding units (so, separate for the multiplier and

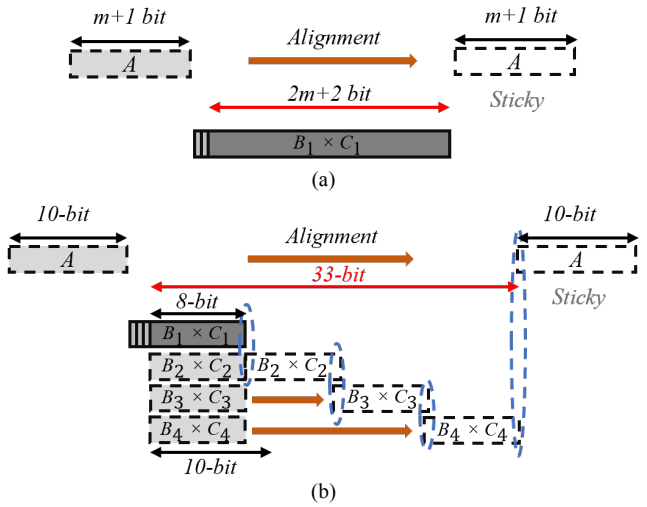


Fig. 2. Overview of the product alignment process: (a) for conventional 3-stage FMA; (b) for mixed-precision FMA in 1 (16-bit) + 4 (8-bit) mode (E4M3 and DLFloat16 in this figure).

the adder). Thus, the overall delay and the rounding error is decreased. 2) The integration of arithmetic units like addition and multiplication leads to a reduction in the overall overhead.

C. *Basic Principles of Mixed Precision for FMAs*

The increasing demand for higher accuracy in DL algorithms has led to complex models with large network sizes. Therefore, reducing computational precision is crucial for improving efficiency in metrics like memory bandwidth, energy, and speed. However better model accuracy is also important, which requires the use of higher precision formats. Mixed precision mitigates this issue by reducing overhead while maintaining nearly the same accuracy as higher precisions. Many mixed-precision approaches have been introduced in recent years [29]-[31][36]. In some methods like Bit Fusion [37], the bit-width is adjusted dynamically at runtime while providing the mixture of lower and higher bit widths for better flexibility and preventing accuracy loss. Similarly, Mix-GEMM [38] enables mixed-precision computations supporting quantization down to 2-bit for dot products to provide better energy efficiency. These methods demonstrate the effectiveness of mixed precision in training complex DL models. However, due to the limited dynamic range of integers, further research is required to explore mixed precision also in these formats.

For instance, ANT [39] introduces a novel numerical format that combines FP and integer representations to optimize quantization schemes. This approach enhances mixed-precision computation by dynamically adjusting precision while improving both efficiency and accuracy. A complementary approach, RaPiD [40], focuses on ultra-low precisions, supporting DLFloat16 and 8-bit FP down to 2-bit fixed point. This architecture demonstrates the viability of extreme precision scaling for reducing energy consumption while maintaining computational robustness; it can be extended and investigated more with other FP format combinations as well. Additionally, Bucket Getter [41] method explores block FP (BFP) formats, which enhance energy efficiency by dynamically adjusting accumulator architectures. Despite

TABLE II
THE ALIGNMENT RANGE FOR MULTIPLICATIONS IN CASE OF DIFFERENT BIT WIDTHS FOR EXPONENT

#bits Exponent	3	4	5	6	7	8
$E_{Maximum}$	6	14	30	62	126	254
$E_{Minimum}$	-4	-12	-28	-60	-124	-252
Alignment Range	10	26	58	122	250	506

significant improvement, BFP format is used when the distribution of all numbers is in a specific range due to the sharing of exponent between different numbers. Some works like [48][49] are focusing purely on FP formats and evaluate mixed precision by considering different conventional and emerged FP formats. They are considering a few combinations of formats and focusing on performing only 2 dot products simultaneously in mixed precision.

In this paper, our goal is to make the FMA design configurable in case of exponent bit width and cover the maximum number of possible formats. Also, we aim to increase the number of dot products in each mixed-precision operation. The proposed design supports both multiple and mixed-precision operations; instead of calculating the accumulation of two lower precision dot products in higher precision and parallelizing several similar calculations to improve hardware utilization, the proposed design accumulates the largest possible number of dot products according to the required precision and the available hardware. This usually increases the accuracy by decreasing the number of rounding steps; however, for many dot-products, it makes the alignment of multiplications slightly more complicated. The selection of the appropriate size of the dot product aligners decreases this complexity, as explained in more detail next.

Table II shows the alignment range of dot product vectors for the supported exponent bit widths based on the difference between the largest and the least possible exponents. When both numbers are normal with the largest exponent value, then after decreasing the bias, their product has the largest possible exponent; also, when both numbers are subnormal with the least exponent values, then after decreasing the bias, the product will also have the least possible exponent. Usually, it is not possible to multiply a pair of numbers when both exponents have these extreme values, because the result will be out of bound; however, for mixed precision, the accumulation is in higher precision, so the resulting value is likely still in range.

Fig. 2(b) shows the alignment of products in the E4M3 format and their accumulation in the DLFloat16 format. The largest product is anchored ($B_1 \times C_1$) while the remaining products are required to be normalized based on it. Since addend (A) is placed on the left of the anchored product, then it may be aligned to the right according to the exponent difference. Hence, different scenarios may occur; if the addend is larger than the anchored product and it is not required to be aligned to the right, then it is not an issue for the addend representing the final mantissa. However, when the addend is required to be aligned to the right of the anchored product, the final accuracy can be affected. As shown in Fig. 2(b), in the worst-case scenario after normalization, all non-anchored products may

have an overlap. A simple way is to choose the anchored product as the final mantissa and truncate the rest. However, this causes an accuracy loss; hence, the normalized products and aligned addend must be considered in the calculations to preserve accuracy and all are required to be accumulated for possible overflows. By using truncation, the design is most likely to be less complicated, but it also incurs a substantial loss in accuracy.

Therefore, in the proposed design to prevent the loss of accuracy when accumulating dot products in the mixed precision mode, instead of truncation, all normalization and alignment amounts of dot products and their accumulation are considered. This scheme comes with some modifications to the conventional FMA design and is explained in more details in sections IV, V and V respectively. Section IV focuses on calculating the normalization amount; then section V explains the alignment amount calculation as well as applying the sign, normalization and alignment to each of the dot products. Finally, section VI elaborates in more detail on the modifications for accumulating the addend and the dot products flow to address different scenarios such as potential overflow and sign changes.

III. Proposed ASIC Design

This section initially presents an overview of the proposed FMA unit; its modifications to a normal FMA unit and the characteristics in supporting multiple/mixed operations are elaborated. Then, each computing stage of the proposed design and related logic are explained.

A. Overview

The proposed design is shown in Fig. 3, which is derived from the conventional FMA (Fig. 1) with modifications to support multiple/mixed operations. In the multiple-precision mode, several normal FMA operations are performed in parallel, while in the mixed-precision mode, computations with different formats can be involved. Next, the components required for supporting each mode and their difference from the conventional design are elaborated.

Multiple-precision mode: When performing multiple normal FMA operations in this mode, the required blocks are similar to the conventional FMA but with some modifications for implementing segmentations; these blocks are marked in purple in Fig. 3. In this mode, few normal FMA operations are performed in parallel. The data flow is shared among the normal FMA operations if the target is a FP format with a small number of bits, while each FMA operation is completely independent; so, for instance, if the highest supported FP format is the SP format (with 32-bit), the unit can perform up to four 8-bit FMA operations independently. Therefore, the design needs to be segmented into a few sections. This arrangement requires additional control circuits and modifications in each unit compared to the conventional FMA. Overall, multiple precision operation is like a conventional FMA design; so, the four conventional 8-bit FMA units are effectively stacked together but other similar units (e.g. Booth multipliers) are only concatenated for higher precision FP format operations.

Mixed-precision mode: To support mixed-precision operations, the blue blocks marked in Fig. 3 are additionally added or significantly modified over the conventional design. They are required due to the following reasons:

i) The proposed design supports the accumulation of largest number of products based on the selected configuration. In this case the largest product must be anchored, and other products should be normalized based on it prior to the accumulation process (as per Fig. 2 (b)). Therefore, some additional hardware is required to handle the normalization, alignment, the decision of their amount, as well as the control process.
ii) Among the mixed-precision operations, products in lower precision may be subnormal numbers and thus, they need to be reformatted (e.g., to handle the hidden bit). Therefore, some leading zero count (LZC) blocks, adders, comparator logics are required to detect the subnormal cases and handle the reformat process.
iii) For accumulating the products together, the addend is aligned based on the exponent difference with the anchored product. Such addition in the mixed-precision mode is different from the conventional FMA, because the sign of the accumulated products is unknown. To track and determine the final sign, we propose a novel method by introducing blocks such as the Increment and Sign Detection blocks. Finally, an Output Finalizing block is also required to integrate all processes and generate the final result.

In this section we only review the overall design; the design principle and circuits of these blocks are elaborated in Sections IV to VI in detail. Overall, by employing the propose design, all supported combinations of formats for multiple/mixed precisions are summarized next.

- In multiple-precision mode, our design supports 4, 2 and 1 FMA operation in 8-bit, 16-bit and 32/19-bit FP precisions.
- In mixed-precision mode, the design supports accumulation of one 32-bit number with two (four) 16-bit (8-bit) FP format numbers; similarly, it also supports accumulation of one 16-bit number with four 8-bit FP format number.

B. Workflow

As shown in Fig. 3, the proposed design has 4 stages (with pipeline schemes to achieve better performance); it includes the second stage additionally compared to the 3-stage conventional FMA. Again, the signals A, B and C determine the addend, multiplier, and multiplicand respectively. When the MixMode signal is high, the design is in mixed-precision mode; the addend is a single number in higher precision while the multiplicand/multiplier consists of several pairs of numbers in lower precision. Else, it is in the multiple-precision mode, and the design computes with a unique precision (normal FMA operation) that is set during configuration.

In this design the primary inputs (addend, multiplier and multiplicand) are divided into four 8-bit segments for 8-bit precisions. Therefore, for a 16-bit format, each pair of segments represents a number; similarly, for 32-bit input number, all 4

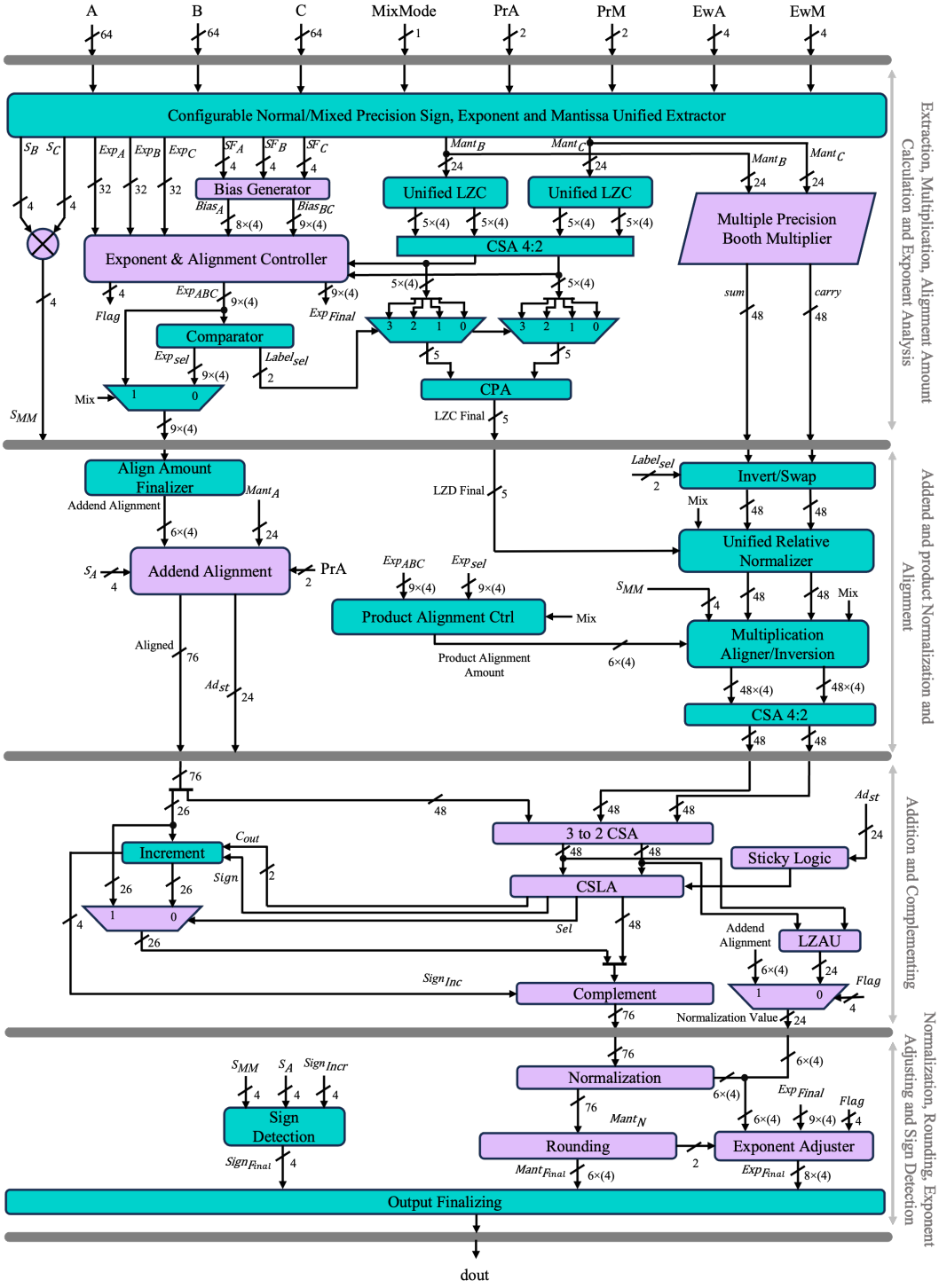


Fig. 3. Proposed fully pipelined FMA Design (the grey text on the right refers to the main stages of the calculation; the purple blocks are the same/similar to that in the conventional design of Fig. 1 and the blue blocks are required in this proposed design to support mixed operation).

segments are required to be concatenated to process a number. The data path of each stage in the proposed design is as follows:

1) Extraction, Multiplication, Alignment Amount Calculation and Exponent Analysis: This stage extracts the sign, exponent and mantissa subfields of the addend, multiplier and multiplicand using a unified extraction unit. The signals PrA/PrM configure the precision of the FMA, while the signals EwA/EwM set the exponents bit width for the input operands. Then, in mixed-precision mode, a set of unified LZC units are activated to check the number of leading zeros for each multiplicand and multiplier and generate a non-zero value if any of the numbers are subnormal. This value is used for normalizing the subnormal products (if required). In multiple-precision mode, the exponent difference of the addend and product (considering the bias values too) is sufficient for the alignment of the addend (Fig. 2 (a)). For example, in the 8-bit configuration 4 differences are calculated independently because there are 4 independent normal FMA operations. Also,

the exponent difference between addend and all products is required to be considered independently and then compared to detect the anchored product. Then the selected difference is used for the addend alignment (Fig. 2 (b)). This difference is calculated by the Exponent & Alignment Controller block. However, in the mixed-precision mode, this process is slightly different; consider SP+E4M3 as an example, 4 exponent differences are performed. The addend's exponent requires attention among all of them because in mixed precision, all products are accumulated with a single addend in a higher precision. The LZC amounts must be also considered; the differences must be compared, and the largest one is selected as representative of the anchored product. Next, the final sign of the dot product(s) is generated and according to the mixed or multiple precision operation, the final LZC value is passed to the next stage for normalizing the anchored product as well as the normalization of the non-anchored products. The Flag signal shows if the difference generated among the exponents is larger than the align offset (the offset when the addend is located to the left side of the dot product). Unlike existing designs (e.g., [48], [49]) that ignore the effect of LZC when calculating the alignment amount of the addend, this condition is met in the proposed design. Also, a unified multiple-precision radix-2 Booth multiplier performs the multiplication of each pair of mantissas.

2) Addend and Product Normalization and Alignment: In this step, the addend(s) alignment is performed using a unified shifter according to the alignment amount generated in the previous step (Exp_{ABC}). Initially, in mixed-precision model, the products are normalized (all at the same time) according to the leading zeros of the largest product (LZC Final). The Product Alignment Ctrl block is responsible for generating the amount to normalize the products according to the anchored product for mixed operation; this amount is equal to the difference between the exponent difference value of each product and the anchored product. By aligning the products according to the anchored product, all products will have a unique exponent value equal to the anchored product. The product normalization and alignment are only activated for mixed-precision operations, for multiple-precision operations it is bypassed. After alignment the bit inversion (based on the multiplication sign (S_{MM})) is applied too. Aligning all products increases the complexity in stage 2; however, due to the smaller dynamic range of 8-bit FP formats, the alignments are less complex than for other formats. By using this characteristic of lower precision numbers, aligners with different sizes are utilized for each number. Finally, all aligned values are added to be ready for the next stage. In summary, except the addend alignment, which is common between mixed and multiple-precision modes, the remaining operations in this stage are for the mixed mode only.

3) Addition and Complementing: The aligned products and addend are added using CSAs; to proceed for rounding and normalization, the carry-save format must be converted back to the conventional format. Then, a CSLA in addition to the incrementor is utilized to compute the final accumulated value. As explained in section VI, in mixed-precision mode the accumulation of overlapped parts of the aligned addend and products (like in Fig. 2 (b)) can be either positive or negative. Therefore, instead of only the increment unit, both the increment/decrement processes for the nonoverlapped part of the aligned addend are required; However, using only an incrementor and some modifications to its design, we address both the increment/decrement processes. For multiple-precision operations, the same design is applicable but with segmentation to support lower FP format configurations. The sign of the final vector after incrementing is determined by the MSB and an inversion could be performed based on that. The unified LZA unit (designed in [43]) is responsible to predict the leading zeros of the CSLA addition for the case in which the MSB of the addend is aligned, such that it is passed as the MSB of anchored product for final normalization; otherwise, the addend alignment amount is selected as the leading zero for the normalization stage.

4) Normalization, Rounding, Exponent Adjustment and Sign Detection: The normalization and rounding may lead to an overflow which potentially affects the final exponent value. By considering this issue, the exponent is adjusted after normalization/rounding and the final result is ready. All the process in the last stage is exactly similar to the conventional FMA in Fig. 1. For multiple-precision mode there might be few independent normal FMA operations according to configured precision. Therefore, a segmentation might be required. However, for mixed-precision, since the result is a single number (in higher precision) all the segments are required to be concatenated.

IV. Extraction, Multiplication, Alignment Amount Calculation and Exponent Analysis

This section goes into the circuit details of the main blocks for processing the first stage (extraction, multiplication, alignment amount calculation and exponent analysis) of the proposed design. Examples are also provided to illustrate the design principles for better understanding.

A. Sign/Exponent/Mantissa Unified Extractor

This extractor is a simple logic responsible for extracting different sections of FP numbers depending on the configured precision and the exponent width (*EwA* and *EwM*). As per the mode of FMA, the exponent and mantissa are placed in a proper format to be aligned in the next stage. For mixed-precision calculations, subnormal numbers and their multiplications in lower precisions are still normal numbers in higher precision because the least exponent value in a lower precision is still more than the least exponent (the subnormal exponent) value in a higher precision. Hence, the products must be normalized first. An intuitive solution consists of performing the multiplication and then normalizing it. This is a challenging task because the bit width is doubled after multiplication and as the data is in a carry-save format, two sequences are required to be normalized (sum and carry of the CSA format). A further solution is to use a pair of LZC units for counting the leading zeros of the multiplier and the multiplicand prior to multiplication. The addition of the LZC values of the multiplicand and the multiplier generates the normalization

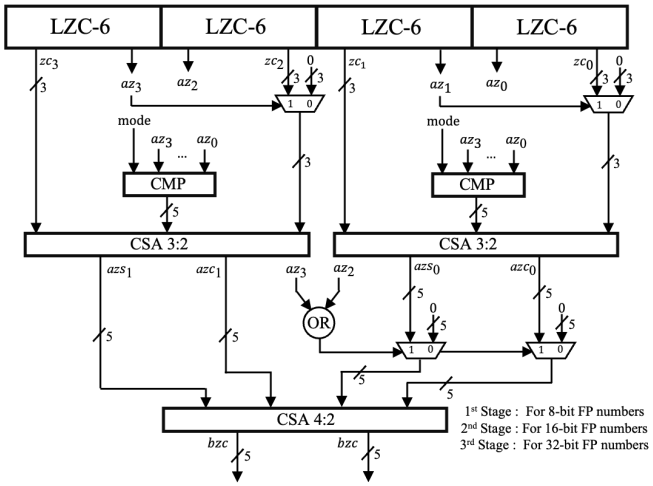


Fig. 4. Overview of the unified LZC design.

amount after multiplication; normalizing the multiplication affects its exponent and it must be updated.

B. *Unified Leading LZC*

A unified LZC design based on the configured precision of multiplier/multiplicand (*PrM*) is proposed as Fig. 4 to support different FP precisions; this unified design consists of 4 LZC units with each supporting a 6-bit number and generating 2 signals, i.e., signal *zc* shows the zero count and signal *az* is activated when the input number is all zeros. When the *az* signal is set, a unit is used to compensate the upper segment's zero count by generating an offset to be added with the next segment's *zc* value. For example, if the configured precision is a 16-bit format then each of the 2 pairs should be added independently. If the first segment has *az* set and second segment has a *zc* equal to 3, then the final LZC is equal to 6 + 3 = 9. However, if the zero count of upper segment is not zero, then the zero count of the lower segment(s) should be ignored. If the zero counts of all related segments are zero, then the generated offset is zero. As the result of this unit is directly used in the first pipelined stage to adjust the exponent of each product, a design with the lowest delay is needed. Therefore, CASs are used to add zero counts. Also, multiplexers are used for selecting the inputs of the adders based on the specified precision configuration.

C. *Exponent & Alignment Controller*

This block is responsible for combining the extracted unified exponents, their relevant bias values and the leading zero counts (for mixed operations) from the two unified LZC units. To meet the multiple/mixed-precision constraints, we have partitioned exponent processors (EP) unit into four segments (Fig. 5 (a)). Based on the precision and the exponents bit widths (*EwA* and *EwM*), the exponents of the multiplier/multiplicand (E_b/E_c) may have a bit width in a range of 4 to 8. Then in the multiple-precision mode, for an 8-, 16- and 32-bit format, 4, 2 and 1 parallel EPs are required respectively. While in mixed precision, since the higher precision's exponent may range from 5 to 8 bits, then all 4 exponent units operate in 9-bit width to support all different configurations. Hence, the exponent difference of an incoming higher precision format and the

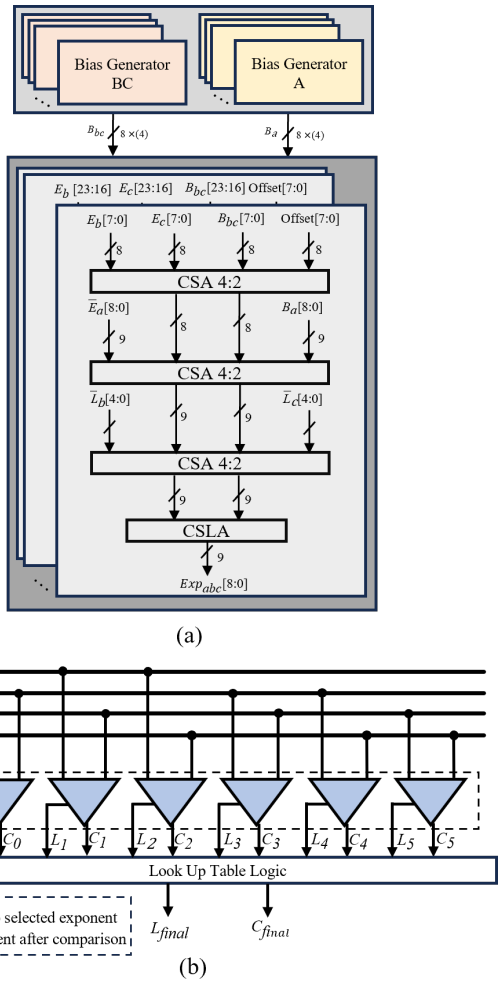


Fig. 5. Exponent Processing and Product Alignment Amount Calculation: (a) exponent and alignment controller; (b) enumeration comparison method.

proposed 8-bit format are required to be calculated separately using 4 different EPs. Also, LZC values are converted to a negative number in 2's complement format; hence, only an inversion of the LZC's sum and carry sequences is performed and the plus one of the 2's complement can be computed by inserting 1 in the LSB of the carry sequence of the CSAs. For example, when adding the sum and the carry of the previous CSA stage with the new data in the current stage (so, from the stage 2 CSA in Fig. 5 (a)), the carry sequence must left shifted by 1 bit and a zero inserted in the LSB; this zero can be replaced by a 1 to address "the plus one" requirement of the 2's complement format in CSA-based additions.

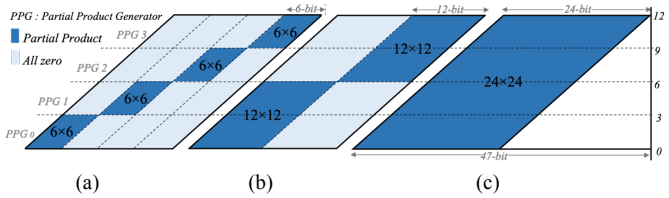
Each EP receives as inputs the exponents of the multiplier/multiplicand, their integrated bias values and their LZC (only in mixed-precision mode), as well as the alignment offset value of the addend. The addend exponent is identical for all four EPs in mixed-precision mode, while the other inputs depend on the multiplication configured format. For 8-bit formats, all other inputs to the EPs are independent and relate to each of the primary input pairs; however, in the 16-bit formats, only a pair of EPs are required, and each EP receives the inputs from the related multiplier/multiplicand. For multiple precision, all inputs of EPs including the addend's exponent depend on the selected format; for example, in the case of 8-bit

formats, each EP's inputs are related to one set of inputs (addend/multiplier/multiplicand). For the 16-bit format, it is similar but only a pair of EPs are necessary; for 32/19-bit formats only one EP is needed. For all processes, based on the exponent bit width and the subnormal flag (related to each of the multiplier/multiplicand values), the accumulated bias value is generated by the bias generator in 2's complement format. In the second stage CSA, the exponent of the addend and its bias value (from the bias generator unit) are considered; at the same time, the unified LZC units generate the zero count values. It is possible to process the LZC value in the first CSA stage, but this increases the critical path delay because it needs to be generated by the LZC units first and then used in the calculations (this step is applicable only to mixed-precision formats). Therefore, other parts of the calculations for already available data are performed first while the LZC unit is generating the LZC values. Then in the third stage, its effect is considered; the input leading zero counts (L_c, L_b) are first inverted and then the plus 1 operation is considered in a similar manner (i.e., by inserting a 1 in the LSB of the carry word when the number is still in CSA format). In the third stage, a 9-bit CSLA is utilized to reduce the critical path delay as much as possible. For mixed precision, the LZC value is also used to normalize the subnormal anchored product; also, other dot products are normalized with respect to the anchored dot product. The resulting unified exponents (Exp_{abc}) are generated and used for normal FMA operation in the addend alignment; its selection is illustrated in next subsection.

D. Comparator

Since mixed precision operations only have one addend, then the largest values among the Exp_{abc} must be selected, so, a fast comparator is required. For comparison of 4 numbers (the exponent differences), a 2-level circuit of normal comparators is required; however, this increases the critical path delay. The so-called Enumeration Comparison (EC) method solves this problem and reduces the required levels to 1 while increasing the hardware overhead. An EC comparator (Fig. 5(b)) is used to compare the 4 numbers in a stage. All 4 numbers are compared one by one and then a look-up table generates the final comparison result and its related label. The comparison result is used for alignment of the addend in mixed precision, while the exponent values prior to comparison are used for the addend alignment in normal FMA operations. By knowing the selected exponent's (Exp_{sel}) label, the anchored product is determined, and the amount of dot product alignments (as per the exponent difference of each product and the anchored product) is also generated for mixed-precision operations. This data is required in the second pipeline stage; then the product alignment ctrl is employed to calculate the difference between Exp_{sel} and Exp_{abc}. This is the amount of alignment for the products which is equivalent to normalizing the products according to the largest product value in mixed precision. The invert/swap unit inverts the negative products while anchoring the largest value.

E. Unified Multiple Precision Booth Multiplier



(a) (b) (c)

Fig. 6. Overview of multiplication for different precisions: (a) 8-bit precision; (b) 16-bit precision; (c) 32-bit precision.

The most critical and complex operation in the first pipelined stage of the proposed FMA is multiplication; a radix-4 Booth based unified multiplication unit is utilized to support different FP formats by sharing available resources. The mantissa bit length may differ for FP formats with same bit width (e.g., BFloat16 and HP); therefore, all combinations of possible bit width for the mantissas of 8-, 16-, and 19/32-bit FP formats require a complex controller to categorize them and then generate the related partial product (as per the Booth's algorithm). Hence, the input mantissa is four 6-bit, two 12-bit or one 24-bit block(s) for each of the 8-bit, 16-bit or 19/32-bit FP formats to perform multiplication. Although this format slightly increases the bit width, it simplifies the control for the multiplication of a pair of mantissas with different bit width.

A divide-and-conquer method is used for multiple-precision multiplication; In this method lower precision independent multipliers are employed to partition a multiplication in higher precision into small pieces. For example, to perform a 24×24 multiplication (in the 32-bit SP format) sixteen 6×6 multiplication units are required; for 8-bit formats, four 6×6 multipliers are required because only 4 parallel multiplications are supported at once. Therefore, most of the hardware units are idle; this scheme also increases the critical path delay to accumulate the partial products (PPs) using the CSA adders. Hence, a 24×24 Booth multiplier is utilized; the proposed multiplier uses 4 partial product generators (PPG) working in parallel. Each PPG is responsible for generating three partial products by allocating the proper position for each PP; the design of all units including the zero padding of the PPs, the Booth recoding units and the PP extenders (from the MSB and the LSB side as per the generated sign by the recoding unit) are the same, except for the allocation of the proper position of the PP as dependent on the precision. An overview of this multiplication process is shown in Fig. 6.

The PPGs operate independently and have the same design except the allocation unit; for example, PPG0 allocates the entire bit width for 24-bit precision while allocating 6- and 12-bit from the MSB for the related precision. Similarly, PPG3 performs from the LSB side by allocating bits 0 to 5 for 8-bit FP numbers, bits 0 to 11 for 16-bit FP numbers and bits 0 to 23 for 32-bit format (i.e., the entire partial product); hence, a control unit is responsible for activating/deactivating each segment of the PPG according to *PrM* and the sequence of PPG shown in Fig. 6. Using this method, all hardware units are utilized for each multiplication including recoding and extenders (that are shared); only simple hardware is added to set the utilized bit positions for each PPG. In Fig. 6, the utilized bit positions for each PPG are shown in dark blue, while the

unused bits are marked in pale blue. For 32-bit precision, all bits are utilized by the PPGs (Fig. 6 (c)). This multiplier operates independently depending on whether the operation is mixed precision or not; it only considers the precision of the multiplier and multiplicand.

V. Addend Alignment and Product Normalization and Alignment

In this section, the design principle and circuit details of the main blocks for processing the second stage (addend alignment and product normalization and alignment) of the proposed design, are explained in detail.

A. Addend Alignment

As explained previously, in the FMA design the addend is primarily located on the left side of the product; then, it is aligned to the right if required (Fig. 2); so, the exponent difference between the added and the product, bias values and the LZC values (for mixed precision only) as well as the alignment offset (related to the number of bits between the MSBs of the product and the addend because the addend is located to the left of the product) must be considered in the addend alignment. In this case, depending on the previous calculation, the addend may be required to be aligned to the right. The addend alignment amount (from multiplexer at first stage) is evaluated by Align Amount Finalizer unit. If an align amount is negative, then it is converted to zero and so on.

The Addend Alignment block (shown in Fig. 7) is employed for both multiple/mixed precision operations. Conventional barrel shifters are used and modified to support a unified alignment of the addends. For normal FMA operations in the multiple-precision mode, if the concatenation of the hidden bit and mantissa of each multiplier/multiplicand has M_p bits, then $3M_p+4$ bits are sufficient considering the guard and round bits between the addend and the product in addition to the guard and round bits from the sticky bits side.

As mentioned later in subsection B, four dot products must be accumulated with the addend. This addition of four numbers may generate a 2-bit overflow which is required to be reserved (in addition to the guard and round bits). Therefore, to preserve the accuracy, the final bit width of the addend alignment for mixed precision must be kept equal to the bit width of the configured higher precision in addition to the 2 reserved bits; then, a bit width equal to $3M_p+6$ bits is required for keeping the highest accuracy. For normal FMA operation, a segmentation in the Addend Alignment unit is utilized to address the different format configurations.

As shown in Fig. 7, the Addend Alignment block applies to the alignment process in 7 levels. The first 5 levels generate the alignment amount for 8-bit FP numbers while the 6th and 7th levels are for 16- and 32-bit FP numbers. This unit supports signed alignment for both mixed/multiple precision. In these scenarios, the addend is inverted and then, the sign may be extended accordingly. Each of the 31-bit aligners operates independently, but the results of each pair of aligners must be concatenated to be used in the next levels if the configured

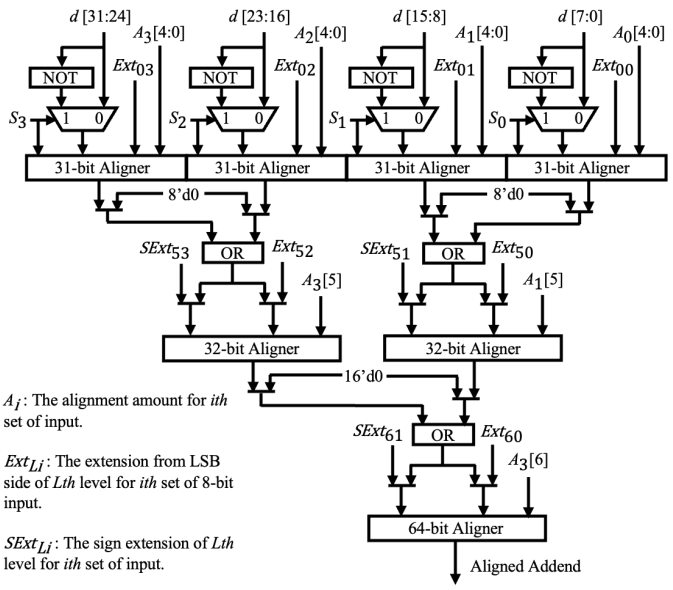


Fig. 7. The addend alignment block.

precision is more than 8-bit. Each of the input of the two neighboring 31-bit aligner has an 8-bit difference; so, the value of the MSB in the left aligner is 2^8 times the MSB in the right aligner. Therefore, each pair of alignments after the 5th level must be extended with 8-bit of zeros from the MSB for the aligner at right-hand side, and extended by the same amount from the LSB for the aligner at the left-hand side for 16-bit formats; then the logic OR operation must be performed by considering the 8-bit difference between the MSB of the primary input of the right and left-hand aligners.

The same method is utilized for the remaining aligning levels but with an extension of 16-bit of zeroes for the 32/19-bit precisions. For normal FMA operations, when the precision is 8-bit, then each alignment amount (A_i) is set independently, while for 16-bit precision, each set $A_{i\ to\ i+1}$ has a distinct value for i = 0, 2. Similarly for 32-bit precision, $A_{i\ to\ i+3}$ for i = 0 has a distinct value. For example, if the precision is 16-bit, 2 different normal FMA operations must be performed in parallel. Hence, $A_{0\ to\ 1}$ have the same value for the first addend, while $A_{2\ to\ 3}$ are for the second addend and have the same value. For mixed-precision format, there is only one addend to be aligned; therefore, all possible configurations have the same A_i values.

B. Dot Product Alignment

In addition to the alignment of the addend, a further step for aligning the dot products must be considered for mixed-precision operations. To do so, first, the largest dot product is selected based on $Lable_{sel}$ and placed in the MSB segment using Invert/Swap unit. Recalling from section IV the booth multiplier is segmented which means the final product's result is segmented as well (e.g., the 48-bit output consists of four 12-bit segments when the multiplier's configuration is set to 8-bit). Then the other dot products are required to be normalized and aligned according to the largest (anchored) dot product. Since numbers in lower precisions are still normal numbers in higher precisions, there is no need to normalize the numbers

separately, i.e., they are normalized relative to the largest dot product. However, the independent normalization of the dot products and their alignment requiring additional left/right alignments incur in more dynamic power. To address this issue, the LZC of the anchored dot product found in the first pipeline stage can be used for the normalization of other dot products using the Unified Relative Normalizer block (shown in Fig. 3); as a result, this process eliminates the need to independently normalize each number, hence simplifying the design.

The amount of dot product aligners is calculated by the Product Alignment Ctrl unit as per the difference between Exp_{sel} and Exp_{abc}. In worst case, four dot products are generated, therefore three aligners are needed as one of the products is anchored. Moreover, two of these aligners are for 8-bit pairs product and they do not need to be long as per Table II; while the third aligner is used for 16-bit numbers which might be a little bit longer than the other two. All these alignments and the addend are added using the CSA to save the critical path delay. Finally, this addition is used in third stage of the FMA calculation, which is explained in the next section.

VI. Accumulation, Normalization and Rounding

This section goes into the design principle and circuit details of the main blocks for processing the last two stages of the proposed FMA design.

A. Addition and Complementor

Due to the likely occurrence of an overflow, the addition of the addend and dot products is slightly different for multiple/mixed precision modes. We propose an efficient and fast method for supporting the different requirements for addition in both modes. An overview of the Addition block is shown in Fig. 8, which includes four submodules: CSA, CSLA, carry propagated adder (CPA), and Incrementor.

For both mixed and multiple configurations, the CSA is utilized to add the overlapping parts of the aligned addend and the dot products from the second pipeline stage (i.e., the accumulation of the dot products for mixed mode, or just the dot products without alignment for normal operation). Since each addition of the addend and dot products in normal FMA operation are organized based on precision in the second pipeline stage, several CSLAs are used to perform the addition (due to segmentation). When the FP format is 8-bit, each addition uses a single independent CSLA and the Cout value shows if an overflow has occurred. For 16-bit precision, 2 pairs of concatenated 12-bit CSLAs are required and the Cout of the lower CSLA is connected to the Cin of the upper CSLA in each pair. Similarly, for 32-bit precision all 4 CSLAs operate together and the Cout value of the upper CSLA determines if an overflow has occurred.

Unlike normal FMA operations in the multiple-precision mode, in mixed precision, the addition of the aligned dot products with different signs can lead to positive or negative results (for normal FMA operation, it is always positive). For example, for 8-bit precision FP numbers, the addition of the addend and 4 dot products may lead to a 3-bits overflow in the worst-case scenario. Then an additional bit is required for the

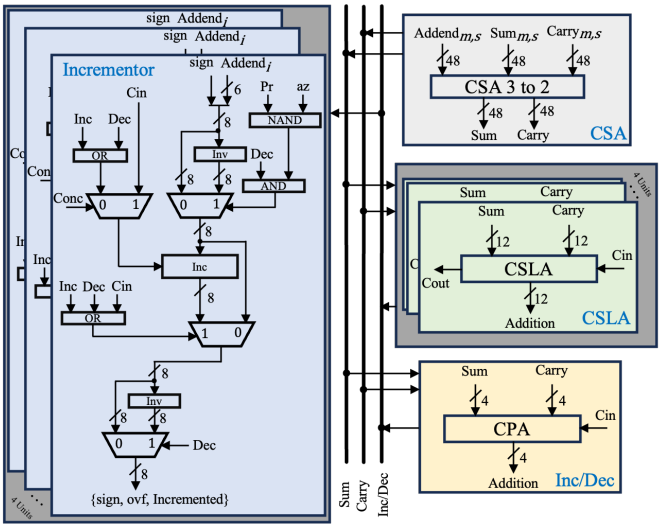


Fig. 8. Overview of the addition of the addend and dot products.

overflow when adding the addend and dot products; also, a sign bit is necessary to determine the final sign of the accumulation. Therefore, a 4-bit CPA is utilized to generate the increment or decrement signal for the nonoverlapped part of the addend. The overlapped parts of the addend with the dot products are represented as $Adnd_m$ while the nonoverlapped parts are represented as $Adnd_i$. The CPA determines the MSB part for adding the $Adnd_m$ and the dot products to detect the potential overflow and final sign; specifically, the MSB bit gives the final sign, and the second bit from the MSB identifies whether an overflow has occurred (when the number is positive) as utilized by the Incrementor unit. Therefore, incrementing or decrementing of $Adnd_i$ may be required for mixed-precision operations, but not for normal FMA operation in which the sign difference can be controlled by simply inverting the addend.

When designing the incrementor, full adders are intuitively required to perform the increment/decrement process; however, this tends to result in a worse critical path and overhead. An alternative scheme using only half adders is proposed in this paper. Let A be a binary number to be decreased by 1; if the A is flipped (1's complement) then the new value is -A-1. The increment of this value by 1 and applying the 1's complement on it (-A) generates A-1 which is equivalent to decrementing but just by utilizing the incrementing hardware. In this case, the complexity of the design is no more than that of full adders, but its critical path is shorter. For both multiple/mixed precision operations, four incrementors are required and they may operate either independently or concatenated. When two or more incrementors are concatenated, the LSB incrementor performs the increment and decrement operation based on the sign and overflow results from the CSLAs; however, the upper incrementors perform the incrementing process based on the overflow signal from the lower incrementor. The sign and overflow value of the MSB incrementor determines the potential need for normalization and complementing of the final addition result. In an incrementor, the *conc* signal determines if the segmented incrementor is going to be concatenated to other segments; the *pr* and *az* signals stand for the propagate and all zero conditions. If the design is in a concatenated mode and all

bits in that segment are zero (az = 1) in the most significant segments, then a 1 can be propagated all the way to the LSB of that segment. These signals guarantee the scenario for the decrementing process.

B. Normalization and Rounding

After accumulation, the result should be normalized. The normalization amount is generated using four 12-bit unified LZAs (which is designed as per [43]). By considering the precision and mixed/normal operation, each LZA segment is added, and the result is the final LZA value; however, this value could be 1-bit off which has been addressed in rounding. When the addend alignment amount is larger than the alignment offset (the LSB of the addend is inserted with 2 bits as round and guard in the MSB of the product), the Flag signal is activated; then, the LZA result is valid for normalization. Otherwise, the exponent difference (Exp_{ABC}) is utilized for normalization. The workflow of normalization unit is very similar to the addend alignment in Fig. 7.

After normalization, a 1-bit deviation may occur due to the use of LZAs; then, the result may be required to renormalized. After the normalization is completed, a 3-bit number (including guard, round and sticky bits) is generated from the sticky part of the final normalized value. Then rounding to the nearest method [20] is executed. Based on a specified precision of the addend, this step takes place; since an overflow may happen after rounding, a post rounding step is required to renormalize the final rounded value. Please note that, the final exponent is adjusted based on the above steps including normalization and rounding.

VII. Evaluation

In this section, some hardware metrics are evaluated and compared for different designs including conventional MAC, conventional and the state-of-the-art FMA designs and the proposed design when supporting various FP formats.

A. Simulation Setting

All designs are implemented at register transfer level (RTL) using Verilog HDL and then simulated using Modelsim. A 65 nm technology file is utilized to synthesize the design using Cadence Genus Synthesis Solution with 1.2v and 25°C. The area and delay results are obtained during synthesis, while for obtaining more accurate power results, Cadence Joules has been employed[2].

For evaluating different FMA units, conventional bit width of FP numbers including 8-, 16- and 32-bit FP numbers are considered, and E4M3, HP and SP precisions are selected. Simulation has shown that the lowest critical path delay is approximately 1.2 ns, hence the timing constraint of the synthesis tool is established as per this value. In the proposed design, the first and second pipeline stages could be merged; however, the exponent and alignment processor and the comparator and addend alignment unit increase the critical path delay to approximately 2ns. So, these units are split among two pipeline stages (stage 1 and 2) and the number of cycles increases by 1. As the critical path delay is decreased, the throughput is improved because the total delay of 4 cycles for the mixed/normal FMA operation is less than for a similar design with 3 cycles.

B. Evaluation Results

Table III compares the hardware results of the proposed FMA with conventional 3-stage FMA unit (Fig. 1) and the conventional MAC design [9] under different format configurations. In addition to the area, power, delay and the number of clock cycles, the energy per operation is also calculated and compared. This is equal to the product of the number of cycles, the critical path delay and the power dissipation divided by the number of parallel processes that can be performed; the number of parallel processes is one for each of the conventional FMAs while it is 4, 2 and 1 for 8, 16 and 32-bit FP formats in the proposed mixed precision design.

The results given in Table III confirm the benefits of FMA over MAC in every metric. For the proposed FMA, its overhead increases compared to the FMA-SP as baseline; this is because in the proposed design an added complexity is introduced for designing control units for segmentation to handle the multiple/mixed-precision operations.

C. Comparison

Table IV compares the proposed FMA with the state-of-the-art designs that can support different precision operations; the

[2]Compared to Genus that generates random test vectors to estimate the power dissipation, in Joules the test vectors can be defined more accurately by the designer. For each format using the correct test vectors and by considering different scenarios, we record the toggle activity of the synthesized design with the testbench stimulated by Modelsim and generate the VCD file; then by providing the constraints file (.sdc) used for synthesis, the mapped Verilog design (after synthesis) and the liberty file (.lib) with the targeted temperature and voltage using Joules, the power is evaluated more accurately.

TABLE III
COMPARISON BETWEEN PROPOSED FMA SUPPORTING MULTIPLE/MIXED OPERATIONS WITH CONVENTIONAL FMAS AND MAC DESIGNS

Design-Format		Area (μm²)	Power (mw)	Delay (ns)	#Cycles	Energy/operation (fJ)				
						E4M3	BFloat16	HP	TF32	SP
Conventional MAC	MAC-E4M3	4945.32	9.91	0.60	8	47.56	N/A	N/A	N/A	N/A
	MAC-HP	12128.04	19.15	0.75	8	N/A	N/A	114.90	N/A	N/A
	MAC-SP	26669.88	26.45	1.00	8	N/A	N/A	N/A	N/A	211.60
Conventional FMA	FMA/E4M3	3496.68	6.08	0.60	3	10.94	N/A	N/A	N/A	N/A
	FMA-BFloat16	6455.88	8.72	0.78	3	N/A	20.40	N/A	N/A	N/A
	FMA-HP	8300.16	11.46	0.75	3	N/A	N/A	25.79	N/A	N/A
	FMA-TF32	8849.88	12.26	0.78	3	N/A	N/A	N/A	28.69	N/A
	FMA-SP	20921.40	22.97	1.00	3	N/A	N/A	N/A	N/A	68.91
Proposed FMA	**FMA-SP-Multiple**	**35877.60**	**29.68**	**1.20**	**4**	**35.62**	**71.23**	**71.23**	**142.46**	**142.46**
	FMA-SP-Mixed	**39482.28**	**33.80**	**1.35**	**4**	**45.63**	**97.26**	**97.26**	**182.52**	**182.52**

TABLE IV
COMPARISON BETWEEN PROPOSED FMA AND SIMILAR WORKS

Node	Supported Functions	Timing Analysis			Area Evaluation		Throughput		Power (mw)	GFLOPS / W	GFLOPS / mm²
		Delay (ns)	FO4	# Cycle	Area (mm^2)	Area (NAND2)	GFLOPS	MIX			
[44]-130nm	1 DP, 2 SP FMA/MUL/ADD	3.43	52.7	3	0.287	56274.51	1.17	×	35.20	33.24	4.08
[45]-180nm	1 DP, 2 SP FMA/MUL/ADD	3.40	34.3	3	0.709	58114.75	1.18	×	N/A	N/A	1.66
[46]-130nm	1 DP, 2 SP FMA/MUL/ADD	3.24	49.8	8	0.149	29215.69	1.23	×	17.80	69.10	8.26
[47]-65nm	1 QP, 2 DP, 4 SP FMA/MUL/ADD	3.41	110.0	3	0.672	466666.67	2.35	×	381.00	6.17	3.50
[48]-90nm	1 DP, 2 SP, 4 HP FMA/MUL/ADD 1 DP + 2 SP, 2(1 SP + 2 HP) Mix	1.50	33.3	3-4	0.191	43409.09	5.33	✓	47.30	112.68	27.90
[49]-28nm	0.5 DP, 2.5 SP, 10 HP MUL/ADD	0.69	28.8	4	0.013	28260.87	14.49	×	29.30	494.53	1114.62
[50]-28nm	1 SP, 2 HP, FMA/MUL/ADD 1 SP + 1 TF32/1 BF/1 HP, 1 SP + 2 HP/2 BF Mix	0.45	18.8	3-4	0.013	28260.87	8.88	✓	59.30	149.75	683.08
[51]-28nm	1 SP, 3 HP, 9 INT8 FMA/MUL/ADD	1.03	42.9	6	0.007	15217.39	5.83	×	9.75	597.94	832.86
[52]-28nm	1 DP, 2 SP/TF32/BF16, 4 HP FMA/MUL/ADD 1 DP + 1 SP, 2(SP + HP) Mix	0.64	26.7	3	0.022	47826.09	12.50	✓	72.30	172.89	568.18
Proposed-65nm	**1 SP/TF32, 2 DL/BF/HP FMA/MUL/ADD 4 E5M2/E4M3 FMA/MUL/ADD 1 SP + 2 HP/2 DL/2 BF Mix 1 TF32 + 2 HP/2 DL/2 BF Mix 1 SP/TF32/HP/DL/BF + 4 E4M3/E5M2 Mix**	**1.35**	**43.5**	**4**	**0.039**	**27083.83**	**5.93**	✓	**33.80**	**175.44**	**138.21**

The functions ending with (without) "Mix" represent mixed (multiple) precision configurations.
1 FO4 = 24(ps), 1 NAND2 = 0.46(μm^2) @28nm; 1 FO4 = 31(ps), 1 NAND2 = 1.44(μm^2) @65nm; 1 FO4 = 45(ps), 1 NAND2 = 4.4(μm^2) @90nm;
1 FO4 = 65(ps), 1 NAND2 = 5.1(μm^2) @130nm; 1 FO4 = 99(ps), 1 NAND2 = 12.2(μm^2) @180nm.

results for the proposed design used in this comparison are the same as that of "FMA-SP-Mixed" in Table III, because it is the version supporting all configuration of formats for both multiple/mixed-precision operations. To perform timing analysis; the critical path delay is reported and it is also converted to the FO4 value depending on the technology node for achieving a fair comparison; also, the number of cycles is reported. Similar, the synthesized area results are converted to the equivalent area of NAND2 gates depending on the employed node for fair comparison. The throughput is evaluated in terms of GFLOPS, which is the largest number of FP operations that can be performed per second. For a fair comparison in GFLOPs, the smallest supported precision is configured in each design; so, in the proposed design, the product of four 8-bit (which is the smallest supported format) inputs are accumulated at a higher precision, and these four multiplications/additions operate in parallel. The same approach is applicable to other designs if they support mixed-precision operations; else, the number of FP operations for parallel normal FMA operations are considered. Other metrics that can provide more comprehensive comparison, such as GFLOPS per power and per area, are also considered.

As given in Table IV, among the designs only [48], [50] and [52] have comparable flexibility as the proposed FMA for supporting both multiple/mixed precisions, but they only support limited combinations or lower precisions. In addition, none of these designs can aggregate more than 2 dot products in mixed precision and accumulate them all at a higher precision number in a single FMA operation. While some of these designs (such as [50] and [52]) are with a lower technology node, all of them have a lower throughput per power compared to the proposed design.

For the other designs reported in Table IV, some may be better in throughput, but they do not support mixed precision and their supported formats are also very limited and simple (low precision FP formats or INT formats). Considering the FO4 and NAND2 results that involves the impact of technology node, the proposed design is also superior to most of these designs, when providing the most powerful functions. Overall, the results prove that compared to all existing designs, the proposed design is a very good candidate for supporting flexible and powerful FMA calculation with efficient hardware.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a new configurable ASIC-based fused multiply-add (FMA) design; it can support both multiple/mixed-precision operations for 8- 16- or 32-bit floating-point (FP) operations by configuring the exponent and mantissa bit widths. Specifically, it supports not only conventional FP precisions (such as HP and SP), but also alternative emerging precision formats such as DLFloat, BFloat, TF, E5M2, E4M3 as often used in deep leaning (DL) applications. The unique feature of mixed-precision for FMA calculation can prevent overflow/underflow when lower precisions are utilized, so leveraging an accuracy loss while trying to decrease complexity when migrating from complex higher precision to lower precision units. This also leads to an increase in performance (as measured in GFLOPs).

The proposed design consists of a pipeline that requires 4 stages/cycles for computation; it accumulates all possible dot products at the same time while only requires one rounding is required after accumulation for improving accuracy. To address the challenges in the hardware implementation faced by existing designs (which require more adder circuits); a new scheme utilizing a segmented addition and incrementing has been proposed. Simulation has shown that compared with other

designs found in the technical literature with the same mixed-precision capabilities, the proposed FMA is best in terms of power and other overhead metrics; moreover, the flexibility in configuring the bit width of the exponent and the mantissa allows the proposed design to be utilized in different applications.

Future work can explore the efficiency of the proposed design for supporting more FP formats, especially for the recently introduced ultra-low precision such as FP4. Note that such formats can also be supported by our current design, but without an improvement on the circuit, the complexity of control units can significantly increase for supporting formats ranging from 4 to 32 bits. Therefore, different computational schemes to assess a balance in format flexibility and hardware overheads should be investigated. Moreover, as follow up to this manuscript it is of interest to evaluate accuracy related metrics for different FMA designs on specific DL architectures to have a better understanding of the entire system operation.

REFERENCES

[1] M. I. Jordan, and T. M. Mitchell. "Machine learning: Trends, perspectives, and prospects." *Science* vol.349, no. 6245, pp. 255-260, Jul. 2015.

[2] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3523-3542, 1 July 2022.

[3] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better." *ACM Computing Surveys,* vol. 55, no. 12, pp. 1-37, Mar. 2023.

[4] F. Niknia, P. Wang, A. Agarwal and Z. Wang, "Edge Caching Based on Deep Reinforcement Learning," *2023 IEEE/CIC International Conference on Communications in China (ICCC)*, Dalian, China, 2023, pp. 1-6.

[5] T. Nordström and B. Svensson. "Using and Designing Massively Parallel Computers for Artificial Neural Networks." *Journal of parallel and distributed computing,* vol. 14, no. 3, pp. 260-285, Mar. 1992.

[6] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," Heliyon, vol. 4, no. 11, Nov. 2018, Art. no. e00938.

[7] J. Garland, and D. Gregg. "Low complexity multiply-accumulate units for convolutional neural networks with weight-sharing." *ACM Transactions on Architecture and Code Optimization,* vol. 15, no. 3, pp. 1-24, Sept. 2018.

[8] S. Yu, H. Jiang, S. Huang, X. Peng and A. Lu, "Compute-in-Memory Chips for Deep Learning: Recent Trends and Prospects," in *IEEE Circuits and Systems Magazine*, vol. 21, no. 3, pp. 31-56, Aug. 2021.

[9] F. Niknia, Z. Wang, S. Liu, P. Reviriego, A. Louri and F. Lombardi, "ASIC Design of Nanoscale Artificial Neural Networks for Inference/Training by Floating-Point Arithmetic," in *IEEE Transactions on Nanotechnology*, vol. 23, pp. 208-216, Feb. 2024.

[10] E. Quinnell, E. E. Swartzlander and C. Lemonds, "Floating-Point Fused Multiply-Add Architectures," *Conference Record of the 41st Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, 2007, pp. 331-337.

[11] J. D. Bruguera and T. Lang, "Floating-point fused multiply-add: reduced latency for floating-point addition," *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, Cape Cod, MA, USA, 2005, pp. 42-51.

[12] A. Michael, *et al*. "Nvidia hopper architecture in-depth." *NVIDIA Technical Blog*, 2022. Accessed: Jan. 26, 2025. [Online]. Available: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth.

[13] Intel, "Intel® 64 and IA-32 Architectures Optimization Reference Manual: Volume 1, 2023. Accessed: Jan. 26, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/671488/intel-64-and-ia-32-architectures-optimization-reference-manual-volume-1.html.

[14] AMD, "5TH Gen AMD EPYC Processor Architecture, First Edition, 2024. Accessed Jan. 26, 2025. [Online]. Available: amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/5th-gen-amd-epyc-processor-architecture-white-paper.pdf.

[15] N. Brunie, F. de Dinechin and B. de Dinechin, "A mixed-precision fused multiply and add," *Conference Record of the 45th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, 2011, pp. 165-169.

[16] J. O. Ríos, A. Armejach, E. Petit, G. Henry and M. Casas, "Dynamically Adapting Floating-Point Precision to Accelerate Deep Neural Network Training," *20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Pasadena, CA, USA, 2021, pp. 980-987.

[17] F. Niknia, Z. Wang, S. Liu, A. Louri and F. Lombardi, "Nanoscale Accelerators for Artificial Neural Networks," in *IEEE Nanotechnology Magazine*, vol. 16, no. 6, pp. 14-21, Dec. 2022.

[18] S. Gupta *et al*., "Deep learning with limited numerical precision." In *International conference on machine learning (PMLR)*, 2015, pp. 1737-1746.

[19] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in Proc. Adv. Neural Inf. Process. Syst., 2018, pp. 7675–7684.

[20] A. Agrawal *et al*., "DLFloat: A 16-b Floating-point Format Designed for Deep Learning Training and Inference," *26th Symposium on Computer Arithmetic (ARITH)*, Kyoto, Japan, 2019, pp. 92-95.

[21] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, and D. Mansell, "Bfloat16 processing for neural networks," *26th Symposium on Computer Arithmetic (ARITH)*, Kyoto, Japan, 2019, pp. 92-95.

[22] M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh, "Numerical behavior of NVIDIA tensor cores," PeerJ Comput. Sci., vol. 7, 2021, Art. no. e330.

[23] P. Micikevicius et al., "FP8 formats for deep learning," arXiv preprint arXiv:2209.05433, 2022.

[24] Z. Carmichael, *et al*. "Performance-efficiency trade-off of low-precision numerical formats in deep neural networks." In *Proceedings of the conference for next generation arithmetic,* 2019, pp. 1-9.

[25] J. Zhang, L. Huang, H. Tan, L. Yang, Z. Zheng, and Q. Yang. "Low-Cost Multiple-Precision Multiplication Unit Design For Deep Learning." In *Proceedings of the Great Lakes Symposium on VLSI,* 2023, pp. 9-14.

[26] H. Zhang and S. -B. Ko, "Variable-Precision Approximate Floating-Point Multiplier for Efficient Deep Learning Computation," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 5, pp. 2503-2507, May 2022.

[27] H. Zhang, H. J. Lee, and S.-B. Ko, "Efficient fixed/floating-point merged mixed-precision multiply-accumulate unit for deep learning processors," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[28] A. A. Wahba and H. A. H. Fahmy, "Area Efficient and Fast Combined Binary/Decimal Floating-point Fused Multiply-add Unit," in *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 226-239, 1 Feb. 2017.

[29] X. Sun et al., "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in Proc. 33rd Int. Conf. Neural Inf. Process. Syst., 2019, Art. no. 441.

[30] P. Micikevicius et al., "Mixed precision training," arXiv:1710.03740, 2017.

[31] D. Kalamkar, *et al*. "A study of bfloat16 for deep learning training," arXiv preprint arXiv:1905.12322, 2019.

[32] J.-M. Müller et al., Handbook of Floating-Point Arithmetic, 2nd ed. Boston, MA, USA: Birkhäuser, 2018.

[33] IS Committee. (2008). 754–2008 IEEE Standard for Floating-point Arithmetic. IEEE Computer Society Std, 2008.

[34] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," IBM J. Res.Develop., vol. 34, no. 1, pp. 59–70, Jan. 1990.

[35] T. Lang and J. D. Bruguera, "Floating-point multiply-add-fused with reduced latency," in *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 988-1003, Aug. 2004.

[36] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul. "Mixed precision training with 8-bit floating point (2019)." *arXiv preprint arXiv:1905.12334* (1905).

[37] H. Sharma *et al*., "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," *2018 ACM/IEEE*

45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 2018, pp. 764-775.

[38] E. Reggiani *et al*., "Mix-GEMM: An efficient HW-SW Architecture for Mixed-Precision Quantized Deep Neural Networks Inference on Edge Devices," *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Montreal, QC, Canada, 2023, pp. 1085-1098.

[39] C. Guo *et al*., "ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization," *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Chicago, IL, USA, 2022, pp. 1414-1433.

[40] S. Venkataramani *et al*., "RaPiD: AI Accelerator for Ultra-low Precision Training and Inference," *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, 2021, pp. 153-166.

[41] Y. Lo, and R. Liu. "Bucket Getter: A Bucket-based Processing Engine for Low-bit Block Floating Point (BFP) DNNs." In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1002-1015. 2023.

[42] J. Lee, *et al.* "Resource-efficient deep learning: A survey on model-, arithmetic-, and implementation-level techniques." *arXiv preprint arXiv:2112.15131*, 2021.

[43] M. S. Schmookler and K. J. Nowka, "Leading zero anticipation and detection-a comparison of methods," in *Proc. 15th IEEE Symp. Comput. Arithmetic*, 2001, pp. 7–12.

[44] K. Manolopoulos, D. Reisis, and V. A. Chouliaras, "An efficient dualmode floating-point multiply-add fused unit," in *Proc. 17th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2010, pp. 5–8.

[45] L. Huang, L. Shen, K. Dai, and Z. Wang, "A new architecture for multiple-precision floating-point multiply-add fused unit design," in *Proc. 18th IEEE Symp. Comput. Arithmetic (ARITH)*, Jun. 2007, pp. 69–76.

[46] V. Arunachalam, A. N. J. Raj, N. Hampannavar, and C. B. Bidul, "Efficient dual-precision floating-point fused-multiply-add architecture," *Microprocessors Microsystems*, vol. 57, pp. 23–31, Mar. 2018.

[47] K. Manolopoulos, D. Reisis, and V. A. Chouliaras, "An efficient dualmode floating-point multiply-add fused unit," in *Proc. 17th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2010, pp. 5–8.

[48] H. Zhang, D. Chen and S. -B. Ko, "Efficient Multiple-Precision Floating-Point Fused Multiply-Add with Mixed-Precision Support," in *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1035-1048, 1 July 2019.

[49] W. Mao *et al*., "A Configurable Floating-Point Multiple-Precision Processing Element for HPC and AI Converged Computing," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 2, pp. 213-226, Feb. 2022.

[50] H. Tan, G. Tong, L. Huang, L. Xiao and N. Xiao, "Multiple-Mode-Supporting Floating-Point FMA Unit for Deep Learning Processors," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 2, pp. 253-266, Feb. 2023.

[51] H. Liu, X. Lu, X. Yu, K. Li, K. Yang, H. Xia, S. Li, and T. Deng. "A 3-D Multi-Precision Scalable Systolic FMA Architecture." *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 1, pp. 265-276, Jan. 2025.

[52] H. Tan, J. Zhang, X. He, L. Huang, Y. Wang, and L. Xiao. "A low-cost floating-point FMA unit supporting package operations for HPC-AI applications." *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 71, no. 7, Jul. 2024.