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

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

Authors

***Abstract*—Hardware accelerators for artificial intelligence (AI) 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 fused multiply-add (FMA) unit supporting mixed/multiple precision is proposed. A wide range of conventional precision formats (such as half and single, formats) as well as emerging precision 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 floating-point numbers based on the needs of an application. The proposed FMA can be configured to perform normal FMA operation supporting multiple precision, or alternatively perform mixed precision in ASIC. The proposed FMA 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, the proposed design utilizes sharing of resources to parallelize several operations based on the available hardware and required precision. Mixed precision tends to accumulate the lower precision dot products into higher precision to avoid overflow/underflow. The proposed design 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 for normal/mixed precision FMA operation. 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 mixed precision computation compared to other designs found in the technical literature. Extensive simulation results for a comparative analysis are provided.**

***Index Terms*— Application Specific Integrated Circuits (ASIC), Machine Learning (ML), Deep Learning (DL), Floating-Point (FP), Fused-Multiply-add (FMA), Configurable, multiple precision, Mixed-Precision, Neural Networks, Accelerator.**

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 and speech recognition [1]-[3]. 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 at reduced overhead (in terms of figures of merit such as area, power and delay) and better accuracy has increased dramatically over the last few years [4]-[5].

Since the most common operation in DL algorithms is multiply-accumulation (MAC), they have been extensively studied [6]-[7]. 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 [8]-[9]. Also, the conventional IEEE-based Single precision (SP) and Double precision (DP) formats provide a sufficient dynamic range; hence, FMAs supporting these formats are commercially available, as such in NVIDIA Tesla and GTX series [10], Intel Knights Mill [11] and Haswell series [12], AMD Piledriver and Bulldozer series. Accuracy plays an important role as figure of merit in different applications including ML's complex models [18]. To address accuracy, floating-point formats due to their wide dynamic range deployed [19]-[20]. However, except for specific calculations, most of the times this wide range of conventional FP formats like single precision (SP) and half precision (HP) is not required and impose significant overhead like area and memory footprint [21]-[22]. For this reason emerging formats like DLFloat [25], BFloat16 [26], TesnsorFloat32 (TF32) [27] or different 8-bit FP numbers (such as E5M2 and E4M3 [28]) which manipulates the bit width of exponent and mantissa proposed to provide a reasonable dynamic range and accuracy for different applications. But the limited dynamic range in some cases like 8-bit format might make them susceptible to overflow/underflow [24].

Therefor mixed-precision formats proposed to address these shortcomings [33]-[37]. Different arrangements have been either investigated in the technical literature to assess the tradeoff between hardware metrics and final accuracy of DL during training; for example, in [29] E5M2 and E4M3 are utilized for inference and training. The results show a negligible degradation compared to higher precision FP such as in SP format. Similarly, other schemes including HP + SP, E5M2 + DLFloat, HP + SP, DP + SP have been investigated, and their efficiency has been evaluated in [30]-[32].

In this paper, we propose a high performance configurable multiple/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 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. 8-bit FP numbers which are widely used in accelerators for AI are considered too;

depending on the selected configuration, the largest number of dot products utilizing the provided hardware can be performed and accumulated simultaneously, so reducing the number of rounding steps and limiting the possible rounding errors compared to recent works. An innovative method to accumulate the dot products and aligned addend is also proposed for normal/mixed precision FMA operation and is explained in more detail in section VI. These characteristics are employed for implementation as per different conventional formats (HP and SP) and emerging formats (DLFloat, BFloat16, TF32, E5M2, E4M3) for the FP-FMA unit. 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. The FMA unit processes the input bit stream as per the chosen configuration and extracts the exponent and mantissa accordingly. In the normal precision mode, the FMA can process four independent dot product accumulation for 8-bit FP numbers (E5M2/E4M3), two independent dot product accumulation for 16-bit FP numbers (HP/DLFloat/BFloat16), and finally one independent dot product accumulation for 32-bit FP numbers (TF32/SP). Similarly for mixed precision mode, a set of lower precision dot products (at the same precision) can be accumulated in a higher precision. In summary, the technical contributions of this paper are as follows:

- A configurable FMA design is proposed; it can process FP numbers with arbitrary bit width for the exponent and mantissa. It supports the IEEE 754 standard FP formats (such as HP, and SP) and alternative formats (such as E5M2, E4M3, DLFloat, BFloat16, TF32).
- The proposed design can support various combinations of precision which provides better flexibility when using it for applications with different accuracy requirements, so a wider dynamic range at a smaller bit width.
- For normal operations, several FMA operations can be performed in parallel to improve performance according to required precision and the available resources to be shared.
- The proposed design performs mixed precision FP operations by accumulating lower precision dot products in higher precision at the same time; this is dependent on the precision and available hardware to reduce rounding errors.
- A unique and efficient unit for segmented addition of the aligned addend and dot product is proposed for both normal and mixed precision.
- The effect of subnormal numbers in mixed precision and their characteristics has been investigated and accounted for a more accurate alignment/addition.

The rest of this paper is organized as follows; section II provides the 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 in detail the designs of the different units of the proposed FMA. Section VII evaluates the hardware metrics and compares the proposed design with other designs found in the technical literature. Finally, Section VIII concludes this paper.

TABLE I
STANDARD AND ALTERNATIVE SUPPORTED FLOATING-POINT FORMATS

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

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 [38]. All FP formats originated from the IEEE 754 standard [39] 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:

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

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. 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. For this reason, many formats (such as Brain floating-point (BFloat16), Tensor floating-point 32 (TF32), Deep Learning floating-point (DLFloat16), E5M2 and E4E3) have been introduced. An overview of different formats is represented in Table. 1. 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. 1, TF32 and BFloat16 can 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)*

The fused multiply-add (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 [40]. 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

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

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 [16]. Different from conventional FP multiply-accumulate (MAC) units that use a separate FP multiplier and adder to implement the multiply-accumulation process, FMA units integrate these operations in a single design. The main steps of a FMA unit are as follows:

- Initially, the mantissa of B and C are multiplied using a fast multiplier and the result is generated in carry-save format. While performing multiplication, a shifter aligns the mantissa of A based on the exponent difference between A and B×C.
- The aligned addend and the multiplication result are added using a carry-save adder (CSA) and then, the carry-save format is converted to a normal representation using a conventional adder (for example, it can be a carry propagate or a carry-select adder). At the same time, the leading zero anticipator (LZA) predicts the count of leading zeros based on the two vectors of the carry-save format. Also, if the sum is negative, the result is inverted.
- The normalization takes place based on the zero-count generated by LZA and then rounding is applied. Also, the exponent value should be adjusted if required.

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 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*

As presented previously, mixed precision is a technique to improve the accuracy of DL algorithms while achieving improvements in hardware-related metrics and reducing memory bandwidth. Based on the application, the proper combination of formats for different critical calculations in DLs must be selected [15]. A mixed precision operation accumulates the lower precision products in a higher precision format. [31] has shown that training using BFloat16 + SP outperforms HP + SP because BFloat16 has a wider dynamic range compared to HP. Similarly, many investigations have been pursued to evaluate the efficiency of mixed precision in DLs [22],[29],[30],[31],[41]. The proposed design supports both normal 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 resources with a number in higher precision. This usually increases the accuracy by decreasing the number of rounding steps; for many dot-products however, 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.

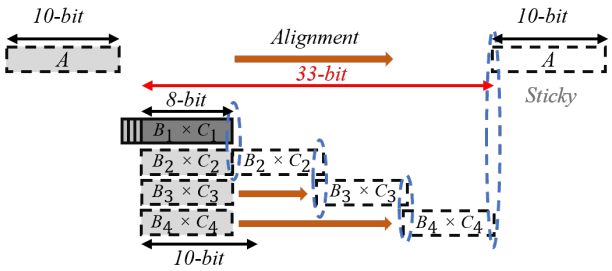


Fig. 1. Overview of multiplication alignment for mixed precision format (E4M3 and DLFloat).

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

Fig. 1 shows the alignment of multiplications in E4M3 format and its accumulation in DLFloat format. The largest multiplication is anchored ($B_1 \times C_1$) while the remaining multiplications and addend (A) can be aligned to the right. Different scenarios may occur. When the exponent difference between anchored multiplication and addend/multiplications are shown in Fig. 2, the result is given by the remaining 10 bits starting from the MSB of the anchored multiplication. As the aligned parts may overlap, then it may affect the final result from the LSB. Moreover, it may affect rounding as well; hence, the aligned parts which are outside of the final result must be considered in the calculations to preserve accuracy. By truncating this part, the design may be less complicated, but also have a substantial loss in accuracy, so not meeting the objective of using FMAs to reduce the accuracy loss in DL applications.

III. PROPOSED ASIC DESIGN

The proposed design has 4 pipeline stages; the signals A, B and C determine the addend, multiplier, and multiplicand respectively. Similarly, when the MixMode signal is high, the design is in mixed precision format, i.e. the addend is a single number in higher precision and the multiplication/multiplier consists of several pairs of numbers in lower precision; else, the design computes with a unique precision (normal FMA) that is set during configuration. As explained later, the input numbers are divided into four 8-bit segments for 8-bit precision FP numbers and for 16-bit and 32-bit numbers the input numbers

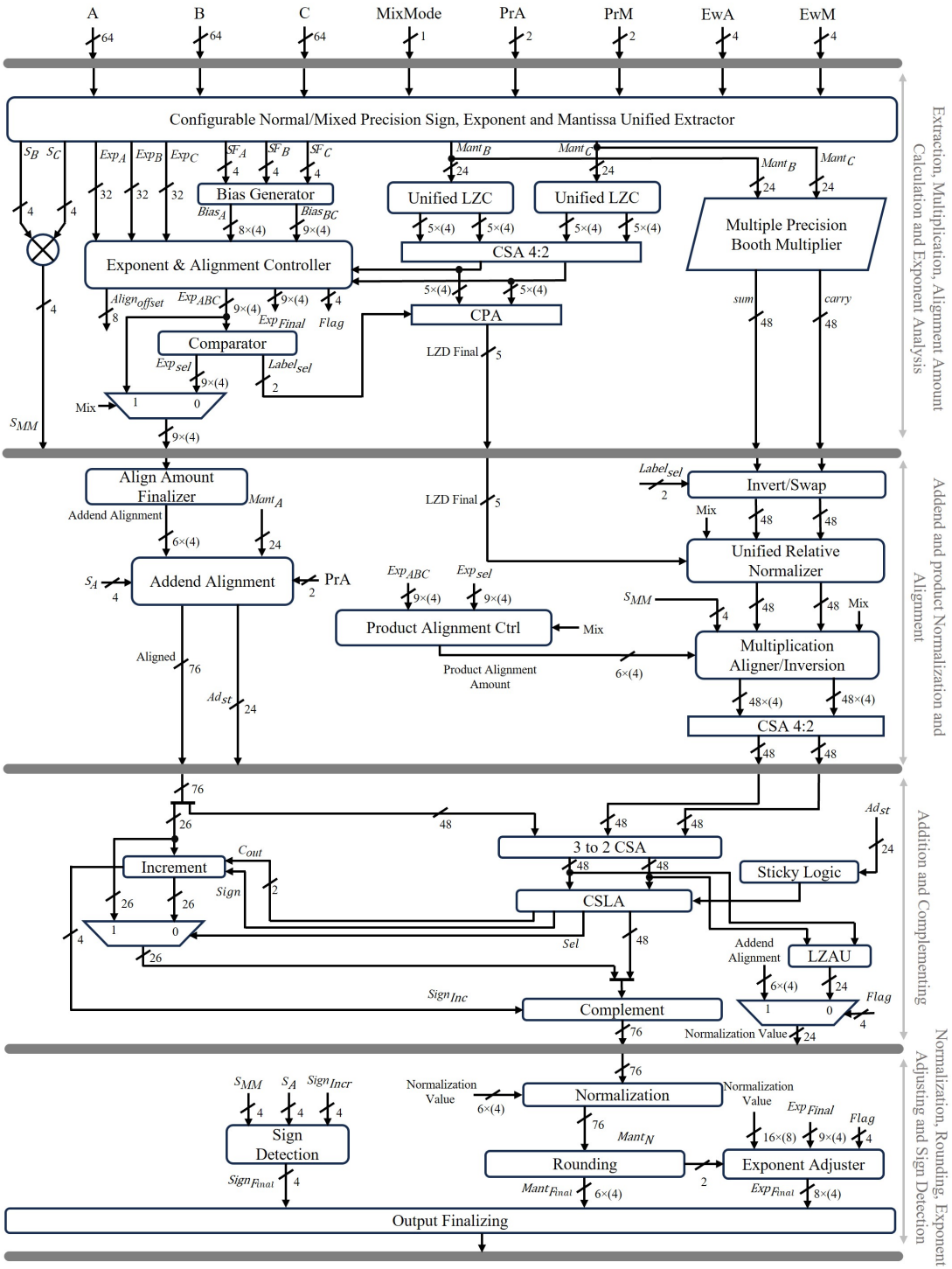


Fig. 2. Proposed fully pipelined FMA Design.

are segmented into 2 and 1 segment(s) respectively. The PrA and PrM signals determine the segmentation of the addend and multiplier/multiplicand based on precision; also, EwA and EwM determine the bit width of the exponent and consequently the bit width of the mantissa for each segment. The data path of each pipeline stage in the proposed design is given in Fig. 2 and it performs 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 set the segmentation of the input number and using EwA/EwM, the exponents of all unified numbers are extracted, and the remaining parts are the mantissa. Then, a set of unified LZD units check the number of leading zeros for each multiplicand and multiplier and generate an offset value if any of the numbers are subnormal for the mixed precision case otherwise it stays idle for a normal FMA operation. As explained in a later section, the output of each LZD unit is in CSA format and another level of CSA unit is utilized to add them to reduce the latency. Using these offsets, the difference

between the exponent(s) of the dot product(s) and addend(s) is calculated by the Exponent and Alignment Controller unit. Next, the final sign of the dot product(s) is generated and according to the mixed precision or normal FMA operation, the final LZD value for the next steps is generated. For mixed precision, the CPA is activated, while for normal FMA operation, it remains inactive. Also, a multiple precision radix-2 Booth multiplier performs the multiplication of each pair of mantissas. The Flag signal shows if the generated difference among the exponents is larger than align offset (required to align the addend). It is necessary to select the normalization amount in the next steps. Unlike other works that bypass the effect of LZD when calculating the alignment amount of the addend, we considered this condition. Because in mixed precision the subnormal numbers are normal numbers in higher precision and for this reason we need to normalize the products in the next step. However, when we normalize the product, its exponent is revised as well, which should be addressed and applied in addend alignment amount calculation too.

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. Depending on the mixed/normal FMA operation, the alignment amount is generated. For normal FMA operation, the difference generated in the previous step (Exp_{ABC}) is used as alignment amount; for mixed precision operations, all differences from the previous step are compared using a comparator to generate the largest difference value for aligning the addend. The Product Alignment Ctrl unit is responsible for generating the relative normalization amount for normalizing the products in mixed operations; this is equal to the LZD of the largest difference as found from the comparator. The product normalization and alignment are only activated for mixed operations, for normal operations it is bypassed. Initially, the products are relatively normalized (all at the same time) according to the potential leading zeros of the largest product as per the amount calculated from the previous stage (LZD Final). Then the normalized products are aligned according to the largest product by the amount calculated using the Product Alignment Ctrl unit and inverted based on the multiplication sign (S_{MM}). At the end, all aligned values are added using a CSA network to be ready for the next stage.

3- *Addition and Complementing:* the aligned products and addend are added using carry-save adders and to proceed for rounding and normalization, the carry-save format must be converted back to the conventional format. Then, a CSLA (carry-select adder) in addition to the incrementor is utilized to compute the final accumulated value. The sign of the final vector after incrementing is determined by the most significant bit (MSB) and an inversion could be performed based on that. The unified LZA unit is responsible to generate the leading zeros of the addition in case the MSB of the addend aligned in a way that it passed the MSB of multiplication, otherwise the addend alignment amount is selected as the leading zero for the normalization step.

4- *Normalization, Rounding and Exponent Adjusting:* The accumulated result at the fourth stage is required to be checked for overflow and normalization. Then, 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 after processing the output. The proposed FMA supports 1 dot-product term for 32-bit FP formats and similarly, 2 and 4 dot-product terms for 16- and 8-bits FP numbers. This increases the complexity of the product alignment in stage 2, however due to the smaller dynamic range of 8-bit FP formats, their 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. Next, each unit is described in more detail.

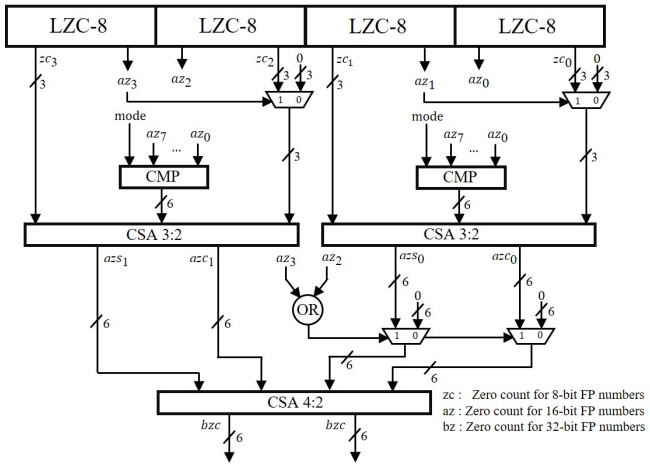


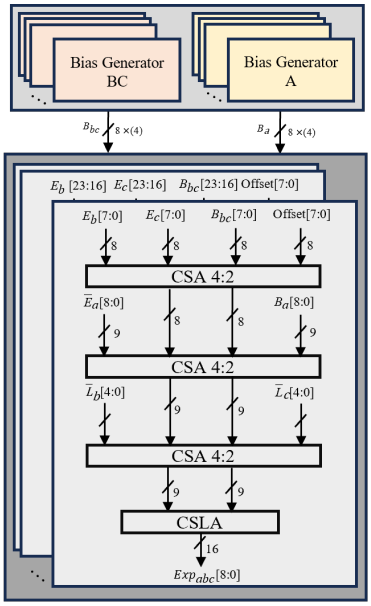
Fig. 3. Overview of the unified LZC design.

IV. Extraction, Alignment, Multiplication and, Exponent Analysis

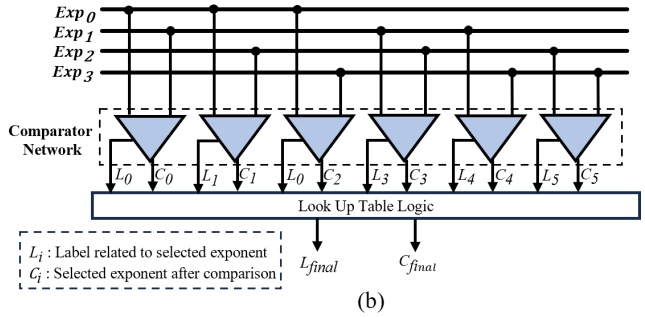
A. Unified Sign/Exponent/Mantissa Extractor

This unit is responsible for extracting different sections of FP numbers depending on configured precision and exponent width (EwA and EwM). Based on the mixed or normal FMA the exponent and mantissa are placed in proper format to be aligned in the next step. For mixed precision calculations, sub-normal 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 sub-normal exponent) value in a higher precision. Hence, the multiplication results must be normalized. 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 the data is in carry-save format.) A further solution is to use a pair of leading zero counters (LZC) for counting the leading zeros of the multiplier and the multiplicand prior to multiplication. The addition corresponds to the normalization amount after multiplication. To normalize the multiplication, the related exponent of the multiplication may require to be updated.

A unified LZC design working based on configured precision (PrM) is proposed to support different FP precisions and is shown in Fig. 3; this design consists of 4 LZC units in which each LZC supports an 8-bit number and generates 2 signals, i.e.



(a)



(b)

Fig. 4. Exponent Processing and Product Alignment Amount Calculation: (a) exponent and alignment controller; (b) enumeration comparison method.

one signal showing the zero count (zc); the other signal is activated when the input number is all zeros (az). When the input number is in 16-bit or higher precision formats, the leading zero count of the used segments must be added; however, if the zero count of upper segment is not zero, then the zero count of the lower segment(s) should be ignored. Else, a compensator unit is used to compensate the upper segment's zero count by generating an offset. 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 multiplication, a design with the lowest delay is needed. Therefore, carry-save adders are used to add the zero counts. Also, multiplexers are used for selecting the inputs of the adders based on the specified precision.

B. Exponent & Alignment Controller, Comparator and Product alignment Ctrl:

These steps are 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 mixed/multi precision constraints, we have partitioned each calculation unit into 9-bit parts as shown in Fig. 4 (a). Based on precision (PrM), the exponents of the multiplier/multiplicand (E_b/E_c) may have a bit width in a range of 4 to 8. Then for an 8-bit floating unit, 4 parallel exponent processors (EP) are required; for the 16- and 32-bit precisions, this number is 4, 2 and 1 respectively. As in mixed precision the higher precision's exponent may range from 5 bits to 8 bits, then all 4 exponent units operate in the 9-bit mode for all different configurations in both mixed/normal FMA operations. The leading zero counts are converted to a negative number in 2's complement format; hence, only an inversion of the LZD'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 one of 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. 4 (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.

To acquire this information in each processor, the input exponents of the multiplier/multiplicand, their integrated bias values (based on the exponent bit width and the subnormal flag related to each of the multiplier/multiplicand, their accumulated bias value is generated by the bias generator in 2's complement format) and the offset value (to address the number of bits of the addend placed on the left side of the largest dot product) are added. In the second stage of the CSA, the exponent of the adder and its bias value from the bias generator unit are considered; at the same time, the LZD unit generates the LZD values. It is possible to process the LZD value in the first CSA stage, but this increases the critical path delay because it needs to be generated by the LZD units first and then used in the calculations. Therefore, other calculations that already have available input data, are performed first (while the LZD unit is generating the LZD value). Then in 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 vacant (0 value LSB) places). In the third stage, a 9-bit carry-select adder is utilized to reduce the critical path delay as much as possible. The resulting unified exponents (Exp_{abc}) are generated and used for normal FMA operation in the addend alignment, however, this process is slightly different for mixed precision.

Since mixed precision operations only have one addend, then the largest amount between Exp_{abc} values must be selected, so, a fast comparator is required. For comparison purposes, 4 numbers (the exponents), a 2-level circuit of normal comparators is required; this however increases the critical path delay. The so-called EC method solves this problem and reduces the required levels to 1 while increasing the hardware overhead. An EC (Enumeration Comparison) comparator (Fig. 4(b)) is used to compare 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 label. Then, the final comparison result, and its label are generated. 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 operation. By knowing the selected

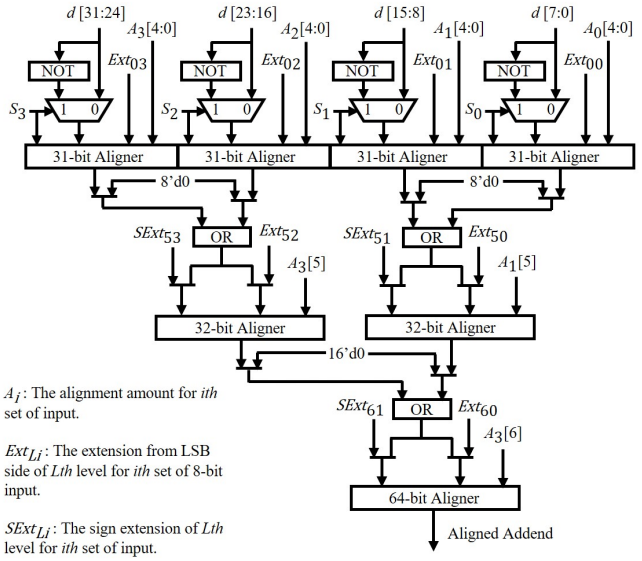


Fig. 5. The addend aligner unit.

exponent (Exp_{sel}) and Exp_{abc} the amount of product alignment is generated for the mixed precision FMA too. This data is required in the third pipeline stage; then a CSLA adder is employed to calculate the difference between Exp_{sel} and Exp_{abc}. This is the amount of alignment for products related to the largest product in mixed precision; however, as mentioned previously, this pipeline stage is bypassed for normal FMA operation. For details about multiplication unit please refer to supplemental material.

V. Addend and Product Normalization/Alignment

In FMA design, the addend is always located on the left of the product; by considering the exponent of the addend, the product and the offset (related to the number of bits between the MSB of the product and the addend), the addend could be aligned based on the product. Conventional barrel shifters are used and modified to support a unified alignment of the addends in the mixed/normal precision operations. For normal operation if each mantissa of the multiplier/multiplicand is M_p bits, then $3M_p+4$ bits are sufficient as bit width for the guard and round bits between the addend and the product and the guard and round bits from the sticky bits.

For mixed precision, consider the addition of all products with the addend; the worst-case scenario must be considered for the bit width for alignment. For 8-bit FP numbers, there are 4 products that must be added with the addend. The addition of 4 numbers may generate a 2-bit overflow which is required to be reserved (in addition to the guard and round bits). Also, to preserve the accuracy, the bit width of the alignment for mixed precision must be kept equal to the bit width of the higher precision alignment in the normal mode (in addition to the 2 reserved bits), because the products are moved to a higher precision (i.e. if a product is subnormal in lower precision, it is normalized because it is a normal number in higher precision, so to be converted from the lower to the higher precision). As hardware is available, a bit width equal to $3M_p+6$ bits is required for keeping the highest accuracy; more details are provided in section VI.

The unified addend alignment unit applies the alignment process in 7 levels (Fig. 5). The first 5 levels generate the alignment amount for 8-bit FP numbers while the 5th and 6th levels are for 16- and 32-bit FP numbers. This unit supports signed alignment; as per the sign of the addend or the partial addend for normal or mixed precision configurations, the number 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 precision is more than 8-bit. So, each pair of alignments after the 5th level must be extended with 8-bit of zeros from the MSB (for the one in the right-hand side) and extended the same from the LSB (for the one in the left-hand side): then the logic OR operation must be performed because there is an 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 and 32-bit of zeroes for the 16-bit and 32-bit precisions. For normal FMA operation, 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 and finally, A_i is identical for all possible i values. For example, if the precision is 16-bit, 2 different normal FMA operations must be performed in parallel. Hence, $A_{0\ to\ 1}$ has the same value for the first addend, while $A_{2\ to\ 3}$ is for the second addend and has the same value also in this case. For mixed precision format, there is only one addend to be aligned; therefore, for all possible precisions of addend (ranging from 16-bit to 32-bit), all A_i values are the same.

In addition to the alignment of the addend, a further step must be considered for mixed precision operations. Dot products are required to be normalized and then aligned according to the largest dot product. As numbers in lower precisions are still normal numbers in higher precisions and the least value of the exponent of the multiplication in lower precisions is still larger than the least exponent value (for sub-normal numbers) in higher precisions, so during mixed precision, there is no need to normalize the numbers separately, i.e. they are normalized relative to the largest dot product. The independent normalization of the dot products and then their alignment based on the largest dot product require additional and unnecessary left/right alignments that incur in more dynamic power. The leading zero counts of the dot products achieved in the first pipeline stage in the exponent processing unit (to adjust the exponent after normalization of each product) can be used for relative normalization. Therefore, the related leading zero count of the largest dot product (so, the dot product with the largest exponent) is the relative normalization amount and eliminates the need to independently normalize each number, hence simplifying the design.

VI. Accumulation, Normalization and Rounding

A. Addition and Complementor

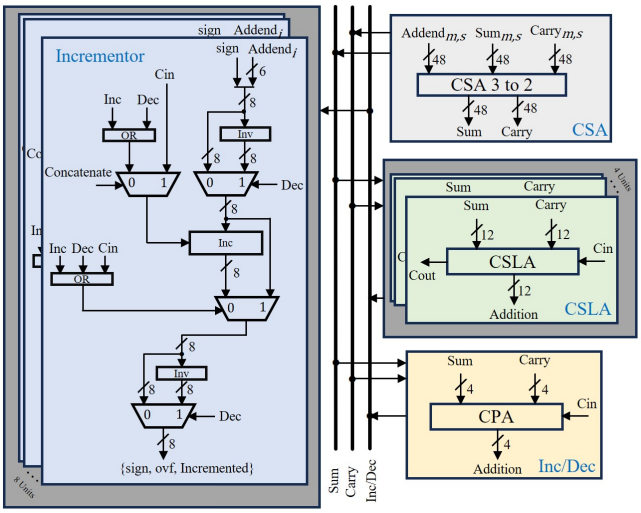


Fig. 6. Overview of Addition of addend and dot products.

The addition of the addends and dot products is slightly different for mixed and normal precision modes due to overflow. We propose a method for an efficient and fast addition supporting the requirements for both mixed and normal FMA operations. An overview of the addition unit is shown in Fig. 6. For both configurations (normal or mixed operation), a CSA adder is required for the carry save format addition of a part of the aligned addend ($Adnd_m$) that has an overlap with the dot product from the first pipeline stage, or the accumulation of the dot products from the second pipeline stage. Since each of the addend and dot products related to each normal FMA operation are organized based on the configured precision in the second pipeline stage, a number of carry-select adder units (CSLA) are required to perform the addition based on the dot product precision. For performing each addition of addend and dot products, 4 normal FMA operations are performed when the precision of a FP number is 8-bit, each of these operations uses a single CSLA independently and the Cout logic value shows if an overflow occurred. For 16-bit precision, 2 pairs of CSLAs are required, each pair utilizes two 12-bit CSLAs that operate together; the Cout of the lower CSLA is connected to the Cin of the upper CSLA. Similarly, 2 groups of 4 CSLA units operate independently for 32-bit precision. the Cout value of upper CSLA determines if an overflow happened. It then performs the increment of the MSB part of the aligned addend that does not have any overlap with the related dot product ($Adnd_i$). After incrementing, the final sign determines if complementing is also required. Like a normal FMA, this process uses the round and guard bits among $Adnd_i$ and $Adnd_m$ for computing the addition in normal FMA, while using 3 more extra bits for mixed operation due to potential overflows. The incrementor design is elaborated in more detail later in this section.

For the mixed precision format, the process is slightly different. To achieve the desired final accuracy, the design becomes slightly complicated; as per section II.C, when considering the propagation effect of the aligned dot products in mixed operations for few cases (such as for the HP + E5M2 mode). Unlike normal FMA operation, the addition of the aligned dot products with different signs can lead to positive or negative results (for normal FMA, this is always positive). Also, for 8-bit precision FP numbers, the addition of 4 dot products may lead to a 2 bits overflow in the worst-case scenario. Then, 2 guard and round bits are not sufficient, and 1 more bit in addition to the guard and round bits, are required to guarantee the correct addition of the addend and the aligned dot products. Therefore, a 4-bit carry propagate adder (CPA) is utilized to generate the increment or decrement signal for the $Adnd_i$ part in which these 4 bits are assigned as the round, guard and the 1 bits for overflow and one bit for the sign. The CPA determines the MSB part for adding the addend $Adnd_m$ and the dot products in the overflown parts; 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, when incrementing $Adnd_i$ a decrementing process may occur too as per the sign of the dot products; this is unlike during normal FMA operation in which it is possible to control the sign difference of the dot product and addend by simply inverting the addend. In this case, the design of the incrementor (consisting of only half adders) is not capable of performing the decrement operation. The half adders must be replaced with full adders to perform the decrement process; however, this worsens the critical path and overhead. Consider both increment and decrement operations. Let A be a binary number to be decremented by 1. If the A bits are flipped (1's complement) then the resulted 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. The complexity added to the design is no more than the time that half adders are replaced by full adders; however, the critical path delay is better than for full adders.

For both normal and mixed precision operations, the 4 incrementor units may operate either independently, or concatenated. When 2 or more incrementors are concatenated, the LSB incrementor computes 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 determine the potential alignment and complementing of the final addition result.

B. Rounding and Normalization

The normalization amount is generated using a unified LZA unit. The design of the LZA unit is based on [44]. The LZA unit is divided into four 12-bit LZA units and the design is very similar to the unified LZD unit explained earlier. By considering the precision and mixed/normal operation, each LZA segment is added and result is the final LZA value, however this value might have 1 bit off (which has been addressed in rounding). When the addend alignment amount is larger than the alignment offset (the LSB of addend is inserted with 2 bits as round and guard in the MSB of the product) the Flag signal is activated; then, the LZAU result is valid for

normalization. Otherwise, the exponent difference (Exp_{ABC}) is utilized for normalization.

The normalization unit design is very similar to the design of the addend alignment in Fig. 5. After normalization, a one-bit deviation (due to the use of LZA) may happen; then, it may be required to renormalize it. A 3-bit number (including guard, round and sticky bits) is generated to round to the nearest of the number [25]. Based on the working precision (the addend's precision this step takes place). Since an overflow may happen a post rounding step is required, and the final exponent is adjusted based on the aforementioned steps.

VII. Evaluation

All designs are implemented at register transfer level (RTL) using Verilog HDL and then simulated using Cadence tools. A 65 nm technology file is utilized to synthesize the design using Cadence Genus Synthesis Solution with 1.2v and 25°C. To generate the power and area when applying the timing constraints, the flip-flops are separately inserted in the inputs and outputs of each stage. Also, we have implemented normal operating FMA units for 8-, 16- and 32-bit FP numbers to compare with the proposed mixed/multiple precision designs. E4M3, HP and SP precisions have been selected as comparative designs for each conventional bit width of FP numbers.

Initially, each pipeline stage has been synthesized independently to assess the designs under different performance metrics. Simulation has shown that the lowest critical path delay is approximately 1.2 ns, hence establishing the timing constraint of the synthesis tool to this value. The first and second pipeline stages could be merged; however, we are using a barrel shifter to extract different formats due to the network of multiplexers, a Booth multiplier and the exponent and alignment processors (as accumulating all possible dot products in mixed precision); hence, the critical path delay increases to approximately 2ns. Hence, these units are split among two pipeline stages (Stage 1 and 2) and the number of cycles increases by 1. However, 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 an FMA designed with 3 cycles.

Table. IV shows the synthesis results for each pipeline stage; the first 2 stages account for the largest portion of the hardware. Input processing uses a large network of multiplexers for data extraction and the Booth multiplier is also rather complex; the comparator and the LZD unit contribute to the increase in the overhead. Therefore, not surprisingly Stage 2 incurs in the largest area and power; also note that the pipeline of the proposed design is highly homogeneous, i.e. every stage has a delay of 1.2 ns.

Table IV compares the results with conventional FMA units under 4 different precision formats. These results show that the delay of multiple precision (MP) and mixed precision (Mixed) FMAs slightly increases compared to the SP FMA; however, the mixed precision FMA unit can support a variety of precision formats compared to normal SP, hence the support of multiple precisions comes at an increase in hardware and power. This table shows the results for the energy per operation in each conventional FMA and the proposed mixed/normal FMA. For each precision, this value is equal to the product of the number of cycles, the critical path and the power dissipation divided by the number of parallel processes than can be performed for each precision. This value is one for each of the conventional FMAs while it is 4, 2 and 1 for 8, 16 and 32-bit precisions in the proposed mixed precision design.

For power results, we used Cadence Joules. For each case using the proper test vectors and considering different situations, we are recording the toggle activity of synthesized design using Cadence genus. The generated file is called a VCD file; then by proving the constraints used for synthesis and the mapped file for Cadence joules we can estimate power more accurately using this tool based on the VCD file. The area and delay are the ones resulted during synthesis.

Table V compares the proposed work with other similar designs; in this comparison, the different functions in each design can support normal and mixed precision FMA (if available) are considered. The timing analysis is then performed; the critical path delay is reported and then, it is converted to the FO4 value depending on the technology node (more details are provided in Table. V). Also, the number of cycles is reported; the reported area is based on the results from the synthesis tool and converted to the related number of NAND2 gates in the 65 nm technology node. The other designs are converted depending on their employed node for fair comparison, such that the throughput is then evaluated. The largest number of FP operations that can be performed in 1 second is reported (GFLOPS); for each design, the smallest precision is selected (i.e. the 8-bit FP precision for the proposed design); also, the number of operations per cycle and the total power are compared.

By considering the older technology node utilized in this

Table III
Comparison Between Proposed FMA and Conventional FMA Units.

Design	Area (μm^2)	Power (mw)	Delay (ns)	#Cycles	Energy/operation (fJ)		
					E4M3	HP	SP
E4M3	3496.68	6.08	0.60	3	10.94	N/A	N/A
HP	8300.16	11.46	0.75	3	N/A	25.79	N/A
SP	20921.40	22.97	1.00	3	N/A	N/A	68.91
SP-MP	35877.60	29.68	1.20	3	20.68	41.36	82.73
SP-Mixed	39482.28	33.80	1.35	4	45.63	97.26	182.52

TABLE IV
COMPARISON BETWEEN PROPOSED FMA AND SIMILAR WORKS.

Node	Supported Functions	Timing Analysis			Area Evaluation		Throughput		Power (mw)
		Delay(ns)	FO4	#Cycles	Area(μm^2)	Area(NAND2)	GFLOPS	op/Cycle	
[45]-180 nm	1 DP, 2 SP FMA/MUL/ADD	3.40	34.3	3	708590	58081	0.58	1.00	N/A
[46]-130 nm	1 DP, 2 SP FMA/MUL/ADD	3.43	52.7	3	286766	56228	0.58	1.00	35.20
[47]-130 nm	1 DP, 2 SP FMA/MUL/ADD 1 DP FMA	3.24	49.8	8	149000	29215	0.62	1.00	17.80
[48]-65 nm	1 QP, 2 DP, 4 SP FMA/MUL/ADD 2 DP, 4 SP FMA/MUL	3.41	110	3	672046	420028	1.16	1.00	381.00
[36]-90 nm	1 QP, 2 DP, 4 SP, 8 HP FMA/MUL/ADD 1 DP, 2 SP, 4 HP Mixed FMA	2.00	44.5	3-4	794790	180634	4.00	1.00	177.00
[49]-28 nm	1 SP Mixed FMA	3.50	146	3	14000	291167	0.28	1.00	N/A
[37]-28nm	1 DP MUL, 5 SP, 20 HP	0.69	28.8	4	13150	27396	27.94	0.50	29.30
[34]-28 nm	1 SP, 2 HP, FMA/MUL/ADD 1 TF/BF/HP or 1 HP/BF Mixed FMA	0.45	18.8	3-4	13032	27150	4.44	1.00	59.30
Proposed 65 nm	1 SP/TF, 2 DL/BF/HP, 4 E5M2/E4M3 *1 SP, 2 HP, 4 E5M2 Mixed FMA*	**1.35**	**43.54**	**4**	**39482.28**	**27418.25**	**2.96**	**1.00**	**33.8**

work and comparing the hardware metrics, it is possible to point out few interesting and probably worst-case features for the proposed design. Only [34] and [36] have comparable flexibility as the proposed FMA in precision functions (note that the largest precision [34] it can support is SP and the alternative supported precisions are very few including TF, BFloat). While these designs have lower feature sizes, most of the time throughputs are lower than 2.96 GFLOPs. Moreover, the proposed FMA requires the highest number of cycles for normal/mixed precision operation (4 vs 3/4); this is a positive feature that is exploited in the pipeline to have the same delay in each stage; the throughput and power of the proposed design are relatively better than all other works at the same precision when possible; for example, [34],[37] provide better power dissipation characteristics but using a lower precision only, or with no support for mixed precision operation (as stated previously, the largest precision supported by [34],[49] is SP and [37] doesn't support mixed precision). For comparison purposes at the same technology node (65 nm), the proposed design shows better values for all metrics compared with [47].

VIII. CONCLUSION

We have proposed a new design for normal/mixed precision FMA; this design offers new features in its configurable operation due to the utilization of a shifter in the input processing. The proposed design consists of a pipeline that requires 4/5 cycles for normal/mixed precision operation at one operation per cycle. This FMA design not only extracts the conventional FP precisions (such as HP and SP), but also alternative precision formats (such as DLFloat, BFloat, TF, E5M2, E4M3) as often used in DL applications. The main operational feature of the proposed design is that it considers alternative FP precisions which are necessary for DL applications; the proposed design is flexible and supports 8- 16- or 32-bit formats, by configuring the exponent and mantissa bit width. This has been achieved by setting the input data's format, such that it extracts hardware segments from the input data and then extracts the different parts of a FP; the use of mixed precision prevents any overflow/underflow when lower precisions are utilized, thus increasing the GFLOPs as well.

Moreover, the proposed FMA accumulates dot products in lower precisions at the same time, so it improves the final accuracy because only one rounding is required, so different from other designs (that accumulate 2 dot products at the same time). This method presents challenges in its hardware because it needs more efficient addition schemes in the next steps; a new scheme utilizing a segmented addition and incrementing has been proposed to solve this issue.

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 proposed design is rather flexible, Also the flexibility in configuring the bit width of the exponent and the mantissa allows the proposed design to reach the dynamic range of the QP format, and therefore it can be utilized in these applications too.

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 et al., "Image Segmentation Using Deep Learning: A Survey," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3523-3542, Jul. 2022.

[3] Menghani, Gaurav. "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] 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.

[5] I. Abiodun et al., "State-of-the-art in Artificial Neural Network applications: A Survey," *Heliyon*, vol. 4, no. 11, pp. 1-41, Nov. 2018.

[6] 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.

[7] 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.

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

[9] 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.

[10] N. Whitehead, and A. Fit-Florea. "Floating-point and IEEE-754 compliance for NVIDIA GPUs." *Nvidia Whitepaper*, 2017.

[11] T. Trader. (Jun. 2017). *How Knights Mill Gets Its Deep Learning Flops*. [Online]. Available: https://www.hpcwire.com/2017/06/22/knights-millgets-deep-learning-flops/

[12] N. Kurd *et al*., "Haswell: A Family of IA 22 nm Processors," in *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 49-58, Jan. 2015.

[13] Q. Wang, X. Zhang, Y. Zhang, and Q. Yi. "AUGEM: automatically generate high performance dense linear algebra kernels on x86 CPUs." In *Proceedings of the international conference on high performance computing, networking, storage and analysis*, No. 25, pp. 1-12, 2013.

[14] M. Lupon *et al*. "Speculative hardware/software co-designed floating-point multiply-add fusion." In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, pp. 623-638. 2014.

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

[16] 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.

[17] 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.

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

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

[20] 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.

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

[22] Wang, Naigang et al., "Training deep neural networks with 8-bit floating-point numbers." *Advances in neural information processing systems* 31 (2018).

[23] . Park, S. Lee and D. Jeon, "A Neural Network Training Processor With 8-Bit Shared Exponent Bias Floating-point and Multiple-Way Fused Multiply-Add Trees," in *IEEE Journal of Solid-State Circuits*, vol. 57, no. 3, pp. 965-977, March 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. 2019.

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

[26] N. Burgess et al."Bfloat16 Processing for Neural Networks," *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, Kyoto, Japan, 2019, pp. 88-91.

[27] M. Fasi, N.J. Higham, M. Mikaitis, and S. Pranesh. Numerical behavior of nvidia tensor cores. PeerJ ComputerScience, 7(e330):1–19, 2021.

[28] Micikevicius, Paulius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha et al. "FP8 formats for deep learning." *arXiv preprint arXiv:2209.05433* (2022).

[29] X. Sun, *et al*. "Hybrid 8-bit floating-point (HFP8) training and inference for deep neural networks." *Advances in neural information processing systems* 32 (2019).

[30] P. P. Micikevicius, *et al*. Mixed precision training. In ICLR '18: International Conference on Learning Representations, 2018.

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

[32] N. Wang, *et al*. Advances in Neural Information Processing Systems 31, pages 7675–7684. Curran Associates, Inc., 2018.

[33] 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. 2023.

[34] 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.

[35] 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.

[36] 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

[37] 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.

[38] J. M. Muller et al., *Handbook of floating-point arithmetic*. Basel, Switzerland:: Birkhäuser, 2018.

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

[40] 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.

[41] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng. Image classification at supercomputer scale. arXiv preprint arXiv:1811.06992, 2018.

[42] K. Manolopoulos, D. Reisis and V. A. Chouliaras, "An efficient multiple precision floating-point multiplier," *2011 18th IEEE International Conference on Electronics, Circuits, and Systems*, Beirut, Lebanon, 2011, pp. 153-156.

[43] J. Sohn and E. E. Swartzlander, "A fused floating-point four-term dot product unit," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 3, pp. 370–378, Mar. 2016.

[44] 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.

[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] 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.

[47] 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.

[48] K. Manolopoulos, D. Reisis, and V. A. Chouliaras, "An efficient multiple precision floating-point multiply-add fused unit," *Microelectron. J.*, vol. 49, pp. 10–18, Mar. 2016.

[49] 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