Fused FP8 4-Way Dot Product With Scaling and FP32 Accumulation

David R. Lutz
Arm Ltd.
Austin (TX), US
david.lutz@arm.com

Anisha Saini
Arm Ltd.
Austin (TX), US
anisha.saini@arm.com

Mairin Kroes
Arm Ltd.
Cambridge, UK
mairin.kroes@arm.com

Thomas Elmer
Arm Ltd.
Austin (TX), US
thomas.elmer@arm.com

Harsha Valsaraju
Arm Ltd.
Austin (TX), US
harsha.valsaraju@arm.com

***Abstract*—For a variety of ML applications, generalized matrix multiply (GEMM) with DOT product is the most computationally intensive operation. This paper presents a microarchitecture exploration of fused multi-way reduced precision floating point multiply-accumulate with single rounding, resulting in power and area efficient characteristics. We propose two different microarchitectures, implementing novel design techniques for computing fused FP8 DOT4 accumulating to higher precision FP32 with scaling to adjust the dynamic range. Our first design, dot product with late accumulation, computes fused FP8 DOT4 by calculating the dot product in the first two cycles, expanding the products to fixed point format and another two cycles for the accumulation operation. This design allows the reuse of a slightly modified, FMA capable, FP32 adder. Our second design, dot product with early accumulation is implemented as a standalone FP8 datapath computing products and accumulation in first two cycles, and another two cycles for normalization and single rounding operation. This design aligns addends (products and accumulator) from an "Anchor" for efficient, arithmetically fused, N-way FP DOT product computation. Furthermore, we synthesized the two designs proposed in a 5nm technology node and compared the cost of implementation.**

I. Introduction

Systems for machine learning and deep learning training are moving from floating-point single-precision 32-bit (FP32) arithmetic to other representations with reduced precision. Some of these representations are Tensor float 32 (TF32), with 8-bit exponent and 10-bit fraction, IEEE half-precision (FP16), with 5-bit exponent and 10-bit fraction, and bfloat16 (BF16), with 8-bit exponent and 7-bit fraction [1], [2].

For inference the 8-bit fixed-point INT8 representation is a popular choice. The floating-point trained models need to be ported to the INT8 format in a conversion step called quantization [3]. This quantization conversion step can be sometimes expensive.

8-bit floating-point (FP8) is the natural progression from the 16-bit floating-point representations, reducing the computational requirements for training and providing good results for inference as well [3].

The floating-point (FP) representation encodes a real number with a sign bit S and e-bit integer exponent E, and a f-bit fraction F,

$$x = (-1)^S \times 2^{E-B} \times (1 + F) \tag{1}$$

where B is the exponent bias, $B = 2^{e-1} - 1$. Note that an implict 1, the *hidden bit*, is concatenated to the fraction as an integer bit, forming the significand. A FP number with $E = 0$ has no implicit 1 in the significand so zero and subnormal values can be represented. In addition, exponent $E = 2^{e-1} - 1$ is reserved for the representation of $\pm\infty$ and NaNs.

Recently, two encodings for FP8 binary exchange format have been proposed [1], [2], E4M3 and E5M2; the name states the number of exponent and fraction bits, i.e., E4M3 has 4 exponent bits and 3 fraction bits. The exponent bias is 7 for E4M3 and 15 for E5M2. The recommended use of FP8 encodings is E4M3 for weight and activation tensors, and E5M2 for gradient tensors [1].

The FP8 dot-product computation is a basic operation in several artificial intelligence algorithms. In general, the n-way dot product is,

$$R_{ndot} = \sum_{i=0}^{n-1} a_i \times b_i \tag{2}$$

Addition and multiplication are the basic floating-point operations. The use of discrete FP multipliers and adders for the dot-product calculation produces a dot-product unit with a large area and latency. Serial multiply-addition of four products to an accumulator would require four multipliers and four adders, incurring area and power penalties, or four consecutive multiply-additions on a single MAC or FMA, incurring delay and power penalties. Area and latency can be reduced by executing several FP operations as a single FP operation. Several fused FP operations have been proposed, for example fused multiply-add units [4], and fused three-term adder [5]. The error is reduced as well because there are no intermediate roundings. Fused operations, like FMA, often maintain accuracy that would otherwise be lost if rounding

were followed by subtractive mass cancellation of leading significant digits.

In training and inference there are cases where scaling factors need to be applied to the dot product to maintain the accuracy [1], [2]. This scaling, 2^{sf}, with sf representing the scaling factor, is usually applied to the dot product before it is added to a FP accumulator.

Fused FP is widely used in the calculation of FP dot products. In this case, other than the area and latency, it is critical for getting a result with an affordable error. The limited precision in the FP representation makes the FP addition sensitive to the operand order and the error in the final result can be significant. The dot product calculation has several additions, and implementing it with discrete multipliers and adders can produce an output with a large error caused by the loss of precision in the additions and the accumulation of rounding errors. For example, using sequential FP32 FMA to add the products $\{2^{48},\ 2^{12},\ -2^{48}\}$ in the given order with round-to-nearest-even would report zero as the sum, whereas, the sum taken with infinite precision and a single, final rounding step (RNE) would result in 2^{12}.

Several dot-product units have been proposed recently. However, most of the units are for datatypes of higher precision than our input FP8 precision.

Two units performing the n-way dot-product of BF16 inputs with accumulation to FP32 are presented in [6], [7]. All the products are first computed independently and then aligned in a single step to the maximum product exponent, truncated to an internal datapath width, and then converted to 2's complement form before being added using a large carry save adder. In both cases a loss of precision is produced because of the truncation of the aligned products; due to the BF16 format used for the input operands this loss of precision could be only avoided by using a huge internal datapath width. In addition, the alignment step is quite costly requiring several subtractions to compute the maximum exponent and the alignment shift amount for each product. Some modifications to achieve an efficient FPGA implementation are discussed in [8].

A similar unit is discussed in [9]. In this case the input precision is not indicated but the alignment step is also based on the computation of the maximum exponent and the calculation of the alignment shift for each product with respect to the maximum product exponent. The internal datapath width is not indicated either.

Two multiprecision SIMD units are presented in [10], [11]. The unit in [10] supports FP64, FP32 and FP16, whereas the unit in [11] supports FP64, FP32, TF32, FP16 and BF16. Note that the FP64 dot product is just a FMA operation. To deal with the larger precision without hitting the area several small multipliers are used, so that a large multiplication is obtained by combining the results of the smaller multipliers; Again, and as discussed previously, the precisions supported lead to a loss of accuracy and to use of complex alignment circuits.

In [12] a method for computing a correctly rounded dot product is presented. This method, that can be applied to any precision, relies on a new aligment technique to prevent catastrophic cancelations and errors introduced by multi-sticky bits. When the difference between exponents is larger than a given threshold the bits shifted out the internal datapath wide are moved back to used empty slots in the internal adder. However, the aligment method implementation can be expensive becuse it needs some additional logic to detect the over-alignment and realign the products.

A unit for exact dot product computation with accumulation for several 8-bit floating-point formats, FP8 and Posit8, is described in [13]. This unit shares the target precision and some of the design decisions with our proposals. Each product is expanded to a fixed-point representation and added together with a compression tree; finally, the dot-product is added to the accumulator. Differently to the units proposed in this paper, the accumulator is a fixed-point value; its size has been chosen to provide guard bits for the sum compression tree, and round it up to the next power of two. It has been considered around 4000 products to add, which correspond to 12 extra bits. That means that the accumulator size depends on the number of products to accumulate. The fixed-point accumulation makes the scaling very difficult. Furthermore, a separate instruction is required to convert the fixed-point accumulator to FP32. Several sources report FP32 accumulation remains important for certain GEMM calculations during mixed precision training in ML, while much of the workload may benefit from accumulation to lesser precision [1], [14], [15], [16].

In this paper we present two different microachitectures for the computation of fused FP8 4-way dot product (DOT4) with scaling and accumulation to FP32 wthout any intermediate rounding. The latency for both microarchitectures is 4 cycles.

The first design calculates the dot product in the first two cycles, expanding the products to fixed point format, and another two cycles for the FP32 accumulation operation. This design allows the reuse of a slightly modified, FMA capable, FP32 adder.

The second design is implemented as a standalone FP8 datapath and spreads the addition with the FP32 accumulator input operand across all four cycles.

In both microarchitectures, the expensive product alignment is avoided by using either the maximum possible dot-product exponent, or an *anchor* as reference to align the products.

II. FP8 FUSED 4-WAY DOT PRODUCT WITH ACCUMULATION

The two proposed fused dot-product units compute the fused FP8 4-way dot product with scaling and FP32 accumulation in four cycles. Fused means that the 4-way dot-product is not rounded before the FP32 accumulation; instead, a wide full precision dot-product is added to the accumulator. Then, the addition of the four products to an accumulator is done with only a single rounding.

Two different units for FP8 dot-product with accumulation are discussed in this paper. The main difference between them, in terms of the microarchitecture organization, being how the accumulator addition is done. In the unit in section II-A the addition to the accumulator is done in the two last

cycles, which adds flexibility to the result forwarding, and allows to reuse a well-known, FMA capable, FP32 adder after modification to increase the bit width of one input from 48 bits to 68 bits. This retains the FP adder functionality for other purposes. On the other hand, the unit described in section II-B is intended to be used as a dedicated unit, with the accumulation spread across four cycles: shift-distance calculation, alignment and addition, normalization, and finally rounding.

Inputs to these datapaths are two 32-bit operands, $op1$ and $op2$, the FP32 accumulator, acc, and a 7-bit positive scaling factor, sf. Each 32-bit operand holds four FP8 numbers, $op1[3]$, $op1[2]$, $op1[1]$, $op1[0]$ and $op2[3]$, $op2[2]$, $op2[1]$, $op2[0]$. The fused sum-of-products (SoP) of the group of four FP8 values held in each 32-bit element of $op1$ and $op2$ vectors is computed.

$$\begin{aligned} SoP &= op1[3] \times op2[3] + op1[2] \times op2[2] + \\ & \quad op1[1] \times op2[1] + op1[0] \times op2[0] \end{aligned} \tag{3}$$

The maximum non-biased FP8 exponent is 2^{15} and consequently the maximum non-biased product and SoP exponent are 2^{31} and 2^{33}, respectively. The range of each FP8 input (with non-biased exponent) is in $[1.11 \times 2^{15}, 0.01 \times 2^{-14}]$ and therefore the product is in $[1.10001000 \times 2^{31}, 0.00010000 \times 2^{-28}]$. So, this fits in the FP32 dynamic range. The SoP is scaled by 2^{-sf} before being added without intermediate rounding to the FP32 accumulator so that the final result is as in (4). In this equation, *rnd()* signifies a floating-point rounding operation.

$$res = rnd(2^{-sf} \times SoP + acc) \tag{4}$$

Two FP8 formats, with different number of exponent and significand bits, are supported, E5M2 and E4M3.

- E5M2: sign bit, 5 bits for the exponent, 2 bits for the significand; exponent bias is 15 for an excess-15 format.
- E4M3: sign bit, 4 bits for the exponent, 3 bits for the significand; exponent bias is 7 for an excess-7 format.

To support both formats, an intermediate 9-bit format with 5 bits for the exponent and 3 bits for the significand, and with an exponent bias of 15 is used. Conversion to the intermediate format is quite easy: E5M2 is converted by simply extending the significand to 3 bits by placing a 0 in the least-significant bit (LSB), and E4M3 is converted by extending the exponent to 5 bits with a 0 in the most-significant bit (MSB) and adding 8 to the exponent to change from excess-7 to excess-15.

To reduce the hardware cost, several IEEE 754 features not important for ML are not provided, including rounding modes other than Round to Nearest Even (RNE), and precise exception handling. However, subnormal inputs and results are supported.

As a final remark, both datapaths can be easily adapted to the computation of the 2-way and 8-way dot products. In general, both units are easily scalable to n-way dot product.

A. Dot product with late accumulation

As said earlier, this unit computes the fused FP8 4-way dot product with FP32 accumulation in four cycles: two cycles for the 4-way dot product calculation, and two cycles for the FP32 accumulation. It is worth pointing out that the addition to the accumulator is done in the two last cycles, which adds flexibility to the result forwarding, and allows to substantially reuse a pre-existing, FMA capable FP32 adder, after modification to increase the bit width of one input.

Figure 1 shows the unit block diagram. The conversion of the FP8 input operands to the intermediate format is not shown in the figure; it is assumed that every FP8 input is in the intermediate 9-bit format.

In the first cycle the four products are obtained and the 8-bit product significands are expanded to a 68-bit fixed-point (FX) number with the first significant bit determined by the product exponent. This way, the addition of the products is carried out in the 68-bit FX format without any loss of information.

Note the carry input to each 5-bit adder for product exponent calculation. As the sum of two excess-15 exponents produces an excess-30 exponent, the excess-31 exponent of each product is obtained by adding an additional 1.

The maximum exponent difference among SoPs is 59, and the products are moved to correct locations as fixed-point numbers depending on their exponents. This is illustrated in Figure 2. The first product is right-aligned in the 68-bit FX number, which corresponds to a product with the minimum product exponent, 2^{-28}, and the last product is left-aligned which corresponds to a product with the maximum exponent, 2^{31}. The second and third products in the figure correspond to intermediate exponents. As shown in the figure products can overlap.

Note that this expansion avoids the expensive calculation of the largest product exponent because the products are aligned to the SoP maximum exponent.

The four FX products are added together with the 4-to-2 reduction and the 69-bit adder. Before the reduction the negative products are 2's complemented: each product is sign-conditionally inverted and the 1 required to complete the 2's complement is added in the reduction tree. Then, the 69-bit adder produces the 69-bit unnormalized and 2's complement SoP significand.

As a consequence of the mapping of the products into the FX variable, the product exponents are no longer needed. Instead, the maximum SoP exponent minus the scaling factor is associated with the SoP. Note that the SoP is added to the FP32 accumulator, so the product exponent is converted to an excess-127 8-bit FP32 exponent. This conversion is done in the 8-bit subtraction by using the excess-127 maximum dot4 exponent.

In the second cycle the SoP is converted to sign-and-magnitude (2's complemented if negative) and normalized. In parallel, the maximum scaled exponent is updated by subtracting the number of leading zeros. This produces a SoP with a FP32 range exponent and a normalized 68-bit significand.

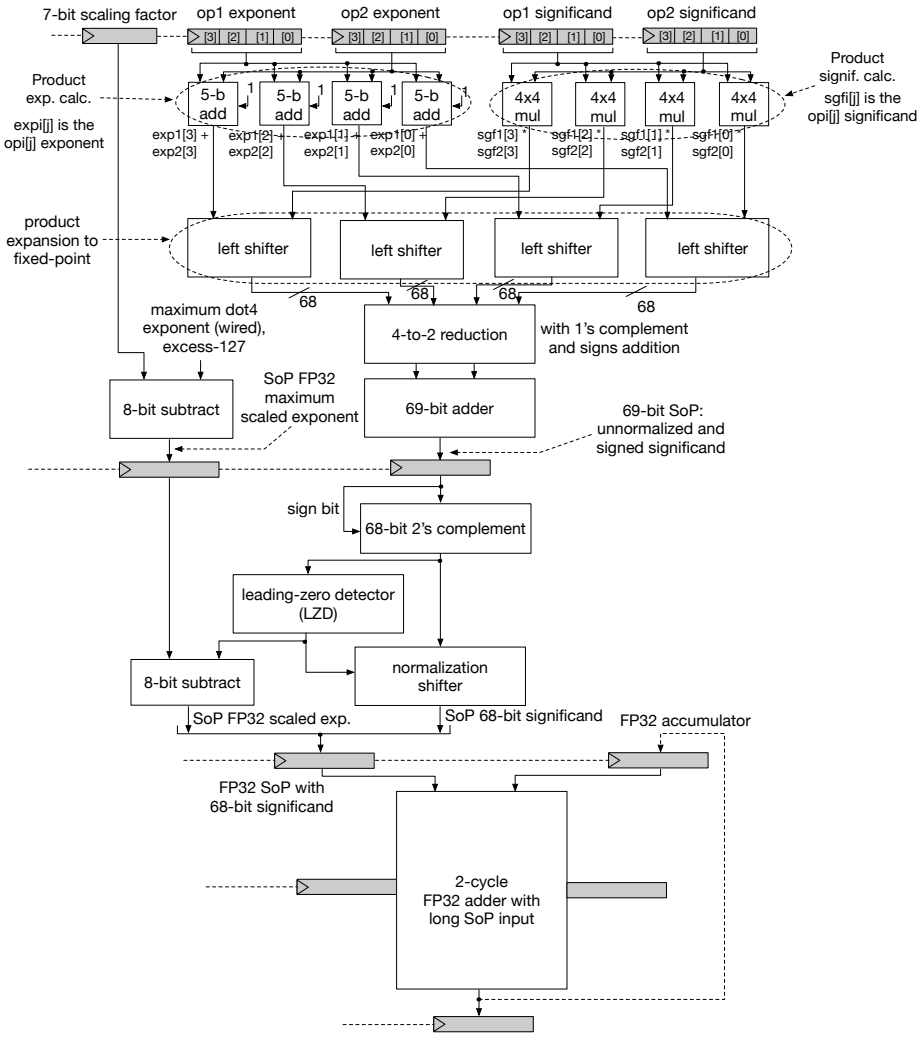

Fig. 1. Block diagram of the unit for FP8 4-way dot product with FP32 accumulation. The addition to the accumulator is done in the last two cycles

Fig. 2. 8-bit products expanded to fixed-point

Finally, the SoP is added to the accumulator in the third and fourth cycles. For this, an FMA capable FP32 adder is modified to accept the SoP with 68 bit significand as one input. FMA capable adders are not uncommon. In processors implementing FMA with separated multiplier and adder, the multiplier returns an unrounded result comprising sign, exponent and a wide significand; one of the adder inputs must have this format [17].

B. Dot product with early accumulation

This design is implemented as a dedicated unit meaning no reuse of existing FP arithmetic units. This design computes a fused FP8 4-way dot product with FP32 accumulation in four cycles: first cycle includes 4-way dot product calculation, first part-alignment of the products to the *anchor*, shift amount of the accumulator; second cycle includes final part-alignment of the products to the *anchor*, summation of products to generate full precision sum-of-products (SoP), alignment of accumulator, and summation of the SoP and accumulator without loss of precision; and the remaining two cycles are for normalization and single rounding operation. This is shown in figure 3.

In the first cycle, four products are calculated by multiplying the significands of the four pairs of input operands using a simple unsigned array scheme. An unlike-sign-multiply (usm) signal is generated by XOR'ing the signs of the multiplier and the multiplicand. Multiplier uses the usm signal to convert intermediate products, which are in carry-save form, to signed numbers and then uses a carry-propagate adder (CPA) to generate a signed product. This way four 9-bit signed products in 2's complement form are generated.

Alignment of products to perform summation to obtain the

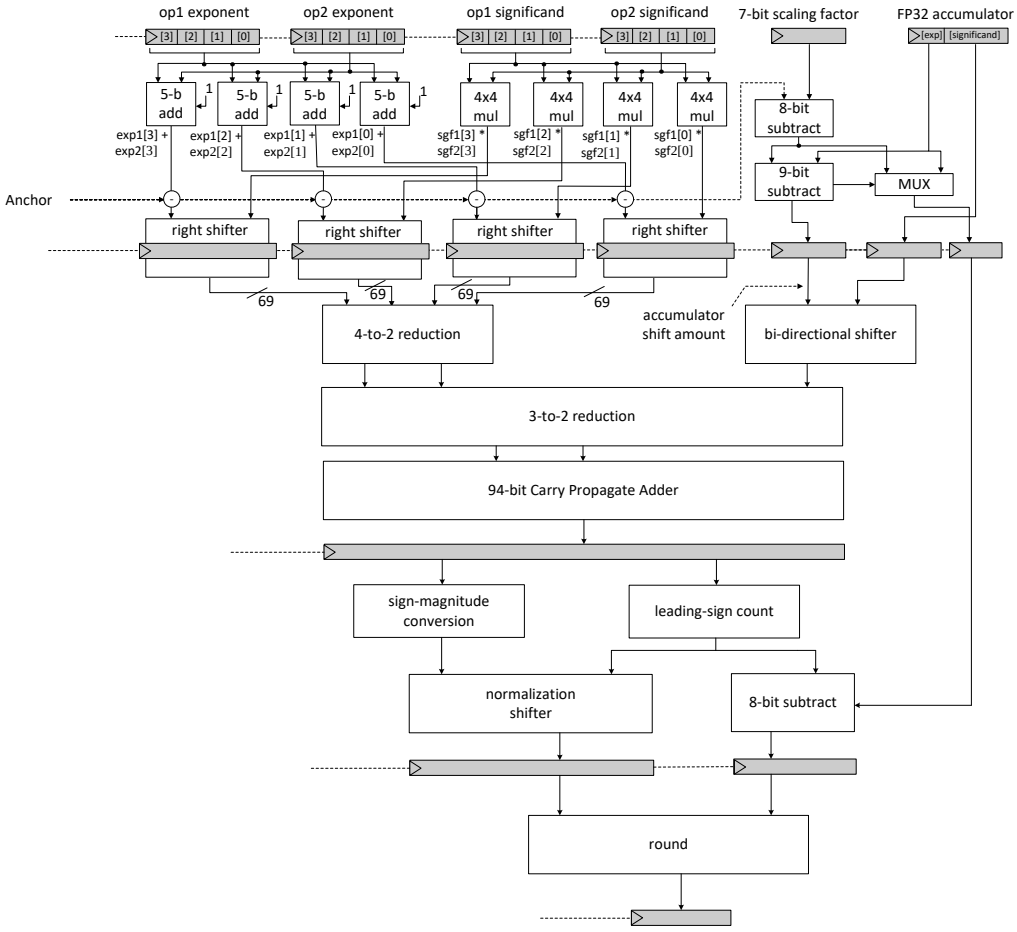

Fig. 3. Block diagram of the FP8 4-way dot product standalone unit

sum-of-products (SoP) is from a fixed and precalculated bit position called the *anchor*. Note that the use of an *anchor* enables efficient, simultaneous, ordered alignment of multiple floating-point addends. This is advantageous as calculation of shift distances from the *anchor* results in optimized subtraction circuits and area efficient circuitry. The *anchor* is optimally chosen to accommodate the range of mass cancellation and the maximum non-biased product exponent value is computed as in (5).

$$anchor = max\{P_{exp}\} + \lceil log_2(N) \rceil + 1 \tag{5}$$

Here, $max\{P_{exp}\}$ is maximum non-biased product exponent, and N is the number of addends.

For the FP8 E5M3 format $max\{P_{exp}\}$ = 31. The number of addends for 4-way DOT product is $N = 4$. Using (5), the $anchor$ is calculated as 34. The 9-bit signed products are partially aligned by right shifting by the lower 3 bits of the shift distance in this cycle. This conserves register bits required to propagate the part-aligned products to the next cycle.

The 7-bit scaling factor sf is incorporated by subtracting it from the $anchor$ forming an 8-bit scaled anchor value. The shift value for the significand of the FP32 accumulator is obtained by subtracting the FP32 accumulator exponent from the scaled anchor value. The sign of the result of this subtract operation selects between the scaled anchor value and FP32 accumulator exponent as the exponent-major. The computed shift distance for the accumulator is bounded by shift limits in order to avoid excessive shifter circuitry complexity and wider adders. After calculation of the shift distance shifting of the accumulator is delayed to the second cycle to reduce register count and area.

In the second cycle, the final part of the alignment of the products to the scaled anchor value is performed. This involves using the upper 3 bits of the shift distance to generate the 69-bit wide aligned product values. After this alignment, these product values are summed together with a Carry Save Adder (CSA) to produce the SoP. The accumulator significand is aligned in parallel with the summation of the SoP. It is shifted to the left if the exponent of the accumulator is larger than the scaled anchor value, and to the right if it is smaller. Notably, the alignment of the accumulator significand is performed with a single bi-directional shifter.

Aligning the accumulator significand to either the left or the right is preferrable over shifting multiple addends with

smaller exponents to the right from both an area and latency perspective. First, this obviates the need to determine an exponent difference between the accumulator and the SoP. Second, it can be done in parallel to the SoP formation, thus removing logic from the critical path. Third, it prevents the need to duplicate shifters since the SoP is still in redundant binary form and would require two right shifters to shift both the carry and sum values.

Finally, the SoP and accumulator values are summed together with another CSA and CPA. As previously mentioned, the SoP and accumulator significand are shifted to their respective shifting limits so that the resulting unrounded sum can be accurately rounded while avoiding excessive logic complexity. For this purpose, 94-bit adders are used to compute the unrounded fraction. Alternately, smaller adders could be used in conjunction with an incrementer for accumulator bits shifted left of the Anchor position.

The unrounded fraction is then normalized in the third cycle. Since all the summands were converted to 2's complement, and the final fraction is a signed-magnitude number, there is a negation circuit that computes the absolute value. This occurs in parallel to the leading sign counting process by means of a Leading Sign Counter (LSC) since both circuits have logarithmic logic depth.

The absolute value of the unrounded fraction is then shifted to the left by the leading sign count, subject to some constraints, and the exponent is decremented accordingly. More specifically, zero and subnormal inputs are properly detected and processed to calculate the correct fraction and exponent values.

Finally, in the fourth cycle, a single rounding operation is performed. The unrounded fraction is rounded using Round to Nearest Even (RNE) mode and the final biased exponent is calculated by subtracting the LSC value from the exponent-major. The final results (rounded fraction, biased exponent, sign) are packed in the destination floating point format.

III. Evaluation

The aforementioned units have been implemented in a 5nm technology node targeting a frequency of 3.6 GHz. The synthesis results for the units can be found in Table I. In this table FP8-DOT4-LA denotes the design with "Late Accumulation" (LA), with the area comprising the DOT4 SoP logic and modified FADD32 unit listed directly below it. FP8-DOT4-EA denotes the design with "Early Accumulation" (EA). Furthermore, the FADD32 split-path unit, and FADD16 and FADD32 units with FMA support have been added to give a perspective on the scale of the proposed 4-way dot product units. The FMA units have a widened port for one of the operands in order to accept an unrounded multiplication result.

As can be observed, FP8-DOT4-LA (1133 sq.um) is larger in overall area than FP8-DOT4-EA (674 sq.um), though a significant portion of its total area is comprised of a prospectively reused FP32 adder. In comparison, the EA unit would coexist next to any pre-existing FP32 adder if both capabilities are needed. As such, the LA unit would be more suitable in performance critical systems that already incorporate a suitable, FMA capable FP32 adder, since it introduces a lower incremental cost, whereas FP8-DOT4-EA would be more suitable to be used within dedicated accelerators as a dedicated datapath where the overall area savings are multiplicative.

TABLE I
Synthesis Results: 5nm - 3.6 GHz

Unit	**Register Count**	**Area** $[sq.\mu m]$
FP8-DOT4-LA	345	1133
$\hookrightarrow$ DOT4 SoP logic	255	572
$\hookrightarrow$ FADD32 (Reused)	90	561
FP8-DOT4-EA	434	674
FADD32 (Split-Path)	N/A	404
FADD16 for FMA	N/A	215
FADD32 for FMA	N/A	512

IV. Parallel FP8 Fused 2-Way Dot Product With Accumulation

As previously mentioned, mixed precision ML computation may benefit from reduced precision accumulation [1], [14], [15], [16]. The computational datapaths described herein are readily enhanced to provide parallel calculation of two FP8 fused 2-way dot products with FP16 accumulation.

First, the FP8-DOT4-LA datapath diagrammed 4 multipliers and associated exponent logic, and shifters are reused and partitioned into two groups; each producing two products. The products from each of those two groups are separately summed to form two SoP values. The existing scale factor calculation is shared by both prospective SoP calculations. Alternately, it may be replicated to provide separate scaling for each of the two prospective SoP.

The diagrammed logic in cycle 2 is substantially replicated to provide normalization and exponent calculation for the two respective SoP. The diagrammed 2-cycle FP adder with long SoP input is modified to provide correctly rounded FP16 output in response to a control signal. Finally, it is necessary to provide an additional FP16 adder with long SoP input to compute accumulation for the second FP8 fused 2-way dot product. Usefully, such an approach allows the provided FP16, FP32 adder functions to be repurposed for other calculations.

Considering the FP8-DOT4-EA datapath diagram, straightforward adaptations allow two, parallel, FP8 fused 2-way dot products with accumulation. First, the four diagrammed multipliers with associated exponent logic and subsequent shifters are reused and partitioned into two groups, as just described. Next, it is necessary to replicate the accumulator shift calculation, considering two FP16 input accumulators, even if they share the same scale factor.

Cycle 2 logic is substantially replicated to provide parallel 3:2 reductions of DOT2 products with respective shifted accumulator significands, and parallel carry propagate adders. Cycles 3 and 4 are also replicated to provide normalization,

TABLE II
SYNTHESIS RESULTS (DOT2/4): 5NM - 3.6 GHZ

Unit	Register Count	Area [$sq.\mu m$]
FP8-DOT2/4-LA	506	1758
↪ DOT2/4 SoP logic	407	926
↪ FADD32/16 (Reused)	99	832
FP8-DOT2/4-EA	624	975

rounding, and final exponent calculation for each of the DOT2 FP16 outputs.

Using such methods, our synthesized RTL produced the following design sizes, as listed in Table II, for the modelled LA (FP8-DOT2/4-LA) and EA (FP8-DOT2/4-EA) datapaths configured to provide both capabilities: FP8 Fused 4-Way Dot Product With FP32 Accumulation, and 2-Way Parallel FP8 Fused 2-Way Dot Product With FP16 Accumulation. Once again, the area numbers comprising the SoP logic and modified FADD32 and FADD16 units, listed collectively as FADD32/16, are broken down below.

V. QUALITATIVE COMPARISON TO PRIOR ART

A study of relevant prior art was provided in section I. In this section a more detailed, qualitative comparison between the microarchitectures described in this work, and those described in [7] and [13] will be presented.

In [7] the authors describe a 32-way BF16 dot-product unit that accumulates the products into an FP32 result. The authors present latency optimizations to the alignment of the 32 products, since they identified this latency as a limiting factor when the number of products is increased. More specifically, the alignment is split into what is referred to as a "global" alignment stage and a "local" alignment stage. By splitting the alignment into two separate stages, the local alignment of the results can be partially overlapped with the global maximum exponent calculation. The latency reductions resulting from the splitting of the alignment stages is then further compounded by speculatively calculating the maximum exponent. Despite the mentioned area overhead associated with the local alignment optimization, the authors report that the combination of latency optimizations were responsible for area savings of 12% for the targeted clock frequency.

As previously mentioned, these timing difficulties with alignment of products is avoided in the EA and LA designs presented in this work through the use of an anchor; requiring fewer comparators. Another important difference is that, while the alignment latency is mitigated by the optimizations mentioned in [7], it is still impacted by the number of products. By contrast, the anchor value we describe serves as a single reference point for alignment, and circuit delays for alignment calculation are mitigated. Finally, as previously mentioned, in [7] the SoP calculation is truncated to optimize PPA, whereas, our techniques for FP8 inputs (which are extensible to FP16 inputs), provide exact SoP calculations.

The authors of [13] present hardware for performing an exact dot product operation for a variety of 8-bit floating-point formats, including several FP8 formats and Posit8. In the paper, the authors describe three separate units. Unit (i) is capable of performing the dot product of 16 pairs of 8-bit floating operands, and accumulates the result into a 2's complement FX accumulator; the exact width of which depends on the dynamic range of the 8-bit floating point operands. Unit (ii) performs the compression of the FX accumulator to FP32, and unit (iii) the compression of an FP32 input to an 8-bit floating-point output.

There is no single unit presented in [13] that performs the same operation as the LA and EA designs. That is, unit (i) does not take an FP32 accumulator as input, and does not produce an FP32 result. However, the combination of units (i) and (ii) in the E5M2 configuration would be closest in resemblance with those designs. Regardless of the similarities in the E5M2 configuration, there are substantial differences. The most fundamental differences are that unit (i) does not support the scaling of FP8 products by means of a scaling factor, and that it uses a wider accumulator. The motivation behind using a wider accumulator is that the authors choose an accumulator width that is a power of two, and sufficiently wide to allow for multiple accumulations to be performed without causing overflows. In the E5M2 configuration the authors fixed the width of the accumulator at 128-bits.

Due to the differences in type and width of the accumulator, the numerical range at the output and overall latency will be different from the designs presented in this work. More specifically, since the LA and EA designs support FP32 inputs and a 7-bit scaling factor, their outputs cover the full FP32 range (including subnorms and underflows), whereas the output of unit (i) would be $\pm(2^{95}, 2^{-32}]$. As the accumulator appears wider than our techniques, it appears to require larger CPA and CSA circuits. As the accumulator appears wider than in our designs, it appears to require larger CPA and CSA circuits. Since various details of unit (ii) are unknown, it is not possible to make further comments on the area and latency of the combination of units (i) and (ii).

VI. CONCLUSIONS

Two different and novel micro-architectures for the calculation of the 4-way dot product with scaling and FP32 accumulation have been presented in this paper. Both microarchitectures have been pipelined into four stages for a target frequency of 3.6 GHz.

The main difference between the micro-architectures is how the FP32 accumulation is done. In the first micro-architecture the scaled dot product is obtained in the first and second cycles, and the accumulation is done in the third and fourth cycles; this allows repurposing an FMA capable FP32 adder for the accumulation with minor modifications. On the other hand, the second micro-architecture spreads the accumulation accross the four cycles, resulting in a dedicated datapath. The expensive product alignment is avoided by using the maximum possible dot-product exponent (first micro-architecture) or an *anchor* (second micro-architecture) as a reference for product

alignment. Both designs can be easily extended to the computation of other interesting dot products, such as the 2-way and 8-way dot products and, in general to any n-way dot product.

Both micro-architectures were synthesized in 5nm technology, and the results revealed different use case scenarios for each of the two. The first design can be adapted in high-performing CPU datapaths with already existing FADD32 units, since this design yields performance benefits and comes at a lower incremental area cost for such systems. The second design can be adapted in dedicated accelerators where the lower overall area cost is more beneficial.

Further implementation details were provided, in order to demonstrate how support for two parallel 2-way dot products accumulating into FP16 accumulators can be readily added while subtantially repurposing logic from the 4-way dot product datapath.

Finally, the proposed designs were compared against state-of-the-art work in a qualitative analysis, illustrating interesting differences in design considerations.

REFERENCES

[1] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, H. Wu. *FP8 Formats for Deep Learning*. arXiv preprint arXiv:2209.05433, 2022.

[2] P. Micikevicius, S. Oberman, P. Dubey, M. Cornea, P. Dubey, A. Rodriguez, I. Bratt, Grisenthwaite, N. Jouppi, C. Chou, A. Huffman, M. Schulte, R. Wittig, D. Jani, S. Deng. *OCP 8-bit Floating Point Specification (OFP8)*. Open Compute Project. Rev 1.0. 2023.

[3] M. van Baalen, A. Kuzmin, S.S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Leel, M. Nagel, J. Soriaga, T. Blankevoort. *FP8 versus INT8 for efficient deep learning inference*. arXiv preprint arXiv:2303.17951, 2023.

[4] T. Lang, J. D. Bruguera. *Floating-point fused multiply-add with reduced latency*. IEEE Transactions on Computers, Vol. 53, No8, pp. 988–1003, 2004.

[5] A. F. Tenca. *Multi-Operand Floating-Point Addition*. Proceedings of 19th IEEE international Symposium on Computer Arithmetic (ARITH-19), 2009.

[6] B. Hickmann, J. Chen, M. Rotzin, A. Yang, M. Urbanski, S. Avancha. *Intel Nervana Neural Network Processor-T (NNP-T) Fused Floating-Point Many-Term Dot Product*. Proceedings of 27th IEEE international Symposium on Computer Arithmetic (ARITH-27), 2020.

[7] H. Kaul, M. Anders, S. Mathew, S. Kim, R. Krishnamurthy. *Optimized Fused Floasting-Point Many-Term Dot-Product Hardware for Machine Learning Accelerators*. Proceedings of 26th IEEE international Symposium on Computer Arithmetic (ARITH-26), 2019.

[8] B. Pasca. *Hybrid Dot-Product Design for FP-Enabled FPGAs*. Proceedings of 26th IEEE international Symposium on Computer Arithmetic (ARITH-26), 2019.

[9] J. Sohn, E. E. Swartzlander. *A Fused Floating-Point Four-Term Dot Product Unit*. IEEE Transactions on Circuits and Systems-I, Vol. 63, NO. 3, pp 370- 378, 2016.

[10] K. Li, W. Mao, X. Xie, Q. Cheng, H. Xie, Z. Dong, H. Yu. *Multiple-Precision Floating-Point Dot Product Unit for Efficient Convolution Computation*. Proceedings of 3rd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2021.

[11] H. Tan, L. Huang, Z. Zheng, H. Guo, Q. Yang, L. Shen, G. Chen, L. Xiao, N. Xiao. *A Low Cost Floating-Point Dot-Product-Dual-Accumulate Architecture for HPC-Enabled AI*. IEEE Transactions on Copmputer-Aided Design of Integrated Circuits and Systems, accepted for publication and available in IEEE Explore, 2023.

[12] Y. Tao, G. Deyuan, F. Xiaoya, J. Nurmi. *Correctly Rounded Architectures for Floating-Point Multi-Operand Addition and Dot-Product Comoputation*. Proceedings of 24th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2013.

[13] O. Desrentes, B. Dupont de Dinechin, J. Le-Maire. *Exact Dot Product Accumulate Operators for 8-bit Floating-Point Deep Leraning*. 26th Euromicro Conference Series on Digital System Design (DSD/SEAA), 2023.

[14] P. Micikevicius. *Mixed Precision Training: Theory and Practice*. NVIDIA. 2018

[15] S. Perez, Y. Zhang, J. Briggs, C. Blake, J. Levy-Kramer, P. Balanca, C. Luschi, S. Barlow, A. Fitzgibbon. *Training and Inference of Large Language Models Using 8-bit Floating Point*. Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023), 2023.

[16] N. Wang, J. Choi, D. Brand, C. Chen, K. Gopalakrishnan. *Training Deep Neural Networks with 8-bit Floating Point Numbers*. Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18), 2018.

[17] D. R. Lutz. *Arm Floating-Point 2019: Latency, Area, Power*. Proceedings of 26th IEEE international Symposium on Computer Arithmetic (ARITH-26), 2019.