Dynamic Fused Multiply-Accumulate Posit Unit With Variable Exponent Size For Low-Precision DSP Applications

Nuno Neves
INESC-ID
Instituto de Telecomunicações
Lisbon, Portugal
nuno.neves@inesc-id.pt

Pedro Tomás
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
pedro.tomas@inesc-id.pt

Nuno Roma
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
nuno.roma@inesc-id.pt

***Abstract*—The current processing power and energy efficiency demands have been pushing the research on low-precision floating-point formats alternative to the IEEE-754 standard. One such alternative is the Posit format, which has demonstrated to provide improved accuracy and larger dynamic ranges with lower precision arithmetic (when compared to the IEEE-754 standard). Such advantages are achieved by adopting a dynamic encoding of the value's scale factor, which combines a variable-sized regime field with a parameterized exponent field. Current hardware implementations are based on fixed exponent sizes. However, this limits the range of values that can be represented by the format for a given precision. Alternatively, this paper proposes a new posit Dynamic Fused Multiply-Accumulate (DFMA) unit with support for variable exponent sizes, capable of encoding an extended representation range, from values with high decimal precisions to very large integer numbers, within the same hardware. When compared to other posit implementations, the proposed DFMA unit shows that the overheads imposed by the improved representation range are marginal, making it suitable to a wide range of application domains.**

***Index Terms*—Fused multiply-accumulate, Posit number system, Low-precision arithmetic, DSP applications**

I. Introduction

Driven by the end of Moore's Law and Dennard scaling, the current computing paradigm has been shifting the industrial and academic research focus to domain-specific architectures [1], [2]. In particular, recent advances in Deep Neural Networks (DNNs) have provided important algorithmic breakthroughs in many domains. However, the ever-increasing availability of data, allied with the complexity of these algorithm, have pushed the computational capacity of off-the-shelf processing architectures to their limit.

As a result, renewed attention has been given to dedicated architectures [3]–[7], and classical design paradigms have been revisited to cope with the computing demands of several application domains. In particular, recent research studies realized that the adoption of alternative floating-point formats with reduced precision (i.e., number of bits) may provide straightforward computing acceleration [5]–[10]. They provide reductions in chip area for floating-point computation, allowing the released area to be used for additional computing and storage resources. As a result, less memory storage is required per operand and higher computing bandwidths can be achieved, while reaching lower power and energy consumptions. Some major computing market players, such as Intel [7], [8], Google [9], NVIDIA [10], Xilinx [6], and IBM [5], have already proposed or adopted such alternative formats in their off-the-shelf Deep Learning platforms and accelerators.

Despite their success, most of the referred solutions have been tailored for the Deep Learning domain. To that end, the posit number system [11] is gaining attention as a possible alternative (or complement) to the IEEE-754 floating-point standard. In many cases, posits offer a wider dynamic range and better accuracy. These are achieved by adopting a dynamic encoding of the value's scale factor, which combines a regime field with variable size and a small exponent field (with a size defined at design-time). Such an approach allows releasing exponent bits to extend the precision of the fraction, effectively allowing a higher accuracy, while lowering the operand width, which also reduces the memory requirements of the application. On the other hand, the posit format is particularly suited for fused operations (e.g., multiply-add/sub operations), since it does not require re-normalization of intermediate results, in turn avoiding accuracy losses. Accordingly, when combining the extended accuracy and reduced memory requirements, the posit format can be a major asset to the bioinformatics and signal processing application domains [12].

Most current posit implementations [12]–[16] strictly follow the proposed posit standard formats [11], by adopting a fixed exponent size, and/or being tailored to mirror the IEEE-754 dynamic range. This results in very small exponent sizes (e.g., 2 bits for 32-bit posits), which in turn simplifies the underlying hardware for posit arithmetic. However, the adoption of a fixed exponent size limits the dynamic range that can be represented by the posit format, not fully exploiting its capabilities. In fact, by varying the exponent size, it is possible to encode a larger dynamic range, capable of supporting (within the same hardware) both values with high decimal precisions and very large integer numbers. Such an approach allows the utilization of

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects UIDB/50021/2020 and PTDC/EEI-HAC/30485/2017.

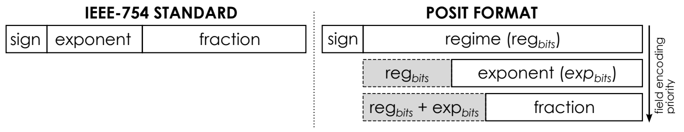

Fig. 1. IEEE-754 standard (left) and Posit (right) formats. The posit regime bit field has a dynamic width and can occupy the entire representation length after the sign bit. The remaining bits (if any) are left to represent the exponent and fraction fields (in that order).

very low-precision arithmetic hardware to perform operations with numbers with large dynamic ranges. It also opens the adoption of the posit format for scientific domains that deal with very large numbers, such as mathematics, cosmology, and cryptography, by allowing the deployment of efficient accelerators with low-footprint units.

Accordingly, this paper proposes a new posit Dynamic Fused Multiply-Accumulate (DFMA) architecture that takes advantage of the whole dynamic range that can be encoded by the posit format. The designed unit is configured at runtime by specifying the maximum exponent size for the considered set of input posit values. Not only does it support the ordinary addition, subtraction and multiplication arithmetic operations, but also fused multiply-add and multiply-accumulate operations. When compared to other posit hardware solutions, implementations of the proposed DFMA on a Field-Programmable Gate Array (FPGA) device and in 45*nm* Application-Specific Integrated Circuit (ASIC) technology show that the resource overhead and delay impacts imposed by supporting a variable exponent size are not only marginal, but they are also offset by the flexibility of providing a very large dynamic range with low-precision operators.

II. Posit Number System

The posit number system was proposed in 2017 [11], being the third iteration of *unum*, a numbering format to represent floating-point values in computer arithmetic alternative to the IEEE-754 standard. The posit format was introduced by relaxing some mathematical properties from the previous iterations and by making its utilization more hardware-friendly.

A posit is formally defined as *posit<n,es>*, where n is the total number of bits (precision) and es is the maximum exponent size. A posit's binary representation is given by:

$$\underbrace{\overbrace{s}^{sign} \quad \overbrace{r_0\ r_1 \ldots \overline{r}_{m+1}}^{regime} \quad \overbrace{e_0\ e_1 \ldots e_{es-1}}^{exponent} \quad \overbrace{f_0\ f_1\ f_2 \cdots}^{fraction}}_{posit\ (n\ bits)} \tag{1}$$

Similarly to the IEEE-754, the structure of the posit format (depicted in Fig. 1) includes a *sign* bit field, an *exponent* field, and a *fraction* (or mantissa) field. However, the posit also adds a variable-sized *regime* field (in the bit format $rrr...\overline{r}$) that encodes a signed value k. Moreover, whenever the sign bit corresponds to a negative number, it is necessary to take the 2's complement before decoding the remaining fields.

Together with the exponent field, the regime (k) represents the working range of the represented value (or scale factor). The numerical value of k is determined by the run length (m) of 1s or 0s in the regime bits, such that:

$$k = \begin{cases} m-1 & , \textit{if } r_0 = 1 \\ -m & , \textit{otherwise} \end{cases} \tag{2}$$

As a result of the variable width of the regime field, the exponent and fraction contents are unknown before decoding the regime (see Fig. 1). Depending on its width, they can be partly (or fully) left out of the binary encoding. Accordingly, the posit value (encoded by (1) and (2)) is given by:

$$(-1)^s \times 2^{exp+k2^{es}} \times 1.fraction \tag{3}$$

The posit format also provides two binary values that are reserved to represent the zero value (000...0) and Not-a-Real (100...0). The latter comprises all mathematical exceptions. In the posit format, there are no subnormal numbers [11].

To implement fused operations (such as fused multiply-accumulate), the posit format requires the utilization of a quire, based on the Kulisch accumulator [17]. It is a fixed-point 2's complement value of length $n^2/2$, with enough fraction precision to avoid cancellations. It is also dimensioned with a carry guard size of $n-1$ bits, allowing the accumulation of up to $2^{n-1}-1$ products without overflowing.

Finally, despite the precision and exponent parameters of a posit format being arbitrary, there are 4 standardized configurations ($n = 64/32/16/8$, $es = 3/2/1/0$) that correspond to the most commonly adopted precisions and dynamic ranges used in IEEE-754 floating-point arithmetic [18]. Accordingly, for the standard 64/32/16/8-bit posit precisions, the quire maintains a length of 2048/512/128/32 bits.

III. Related Work

Recent studies showed that posits are consistently capable of attaining similar accuracies to IEEE-754, with precisions (i.e., number of bits) as low as half of those used by the IEEE-754 standard [12]–[14]. Moreover, posit arithmetic operators (e.g., adders and multipliers) can be implemented with silicon area and power costs comparable to the IEEE-754 counterparts [12]. However, the accuracy and non-overflow benefits attained by the adoption of a quire for fused operations imposes a large area overhead [15]. While this overhead becomes prohibitive for precisions larger than 32-bit posits [15], it is particularly cheap for 8 and 16-bit posits, making them a promising alternative for the next generations of accelerators.

Some hardware implementations have already been proposed that seek the adoption of the posit format. Jaiswal et al. [19] proposed one of the first parameterized algorithmic computational flows for posit addition/subtraction arithmetic and modeled its architecture implementation. Forget et al. [15] introduced a template library to implement operators for custom size posits and their associated quire. Following these initial approaches, Chaurasiya et al. [12] proposed a parameterized pre-synthesis posit unit generator for adders and multipliers of any bit-width. They observed that the area and energy consumption of the operators are comparable to their IEEE-754 compliant counterparts, and that they can provide comparable accuracies to the IEEE-754 standard for

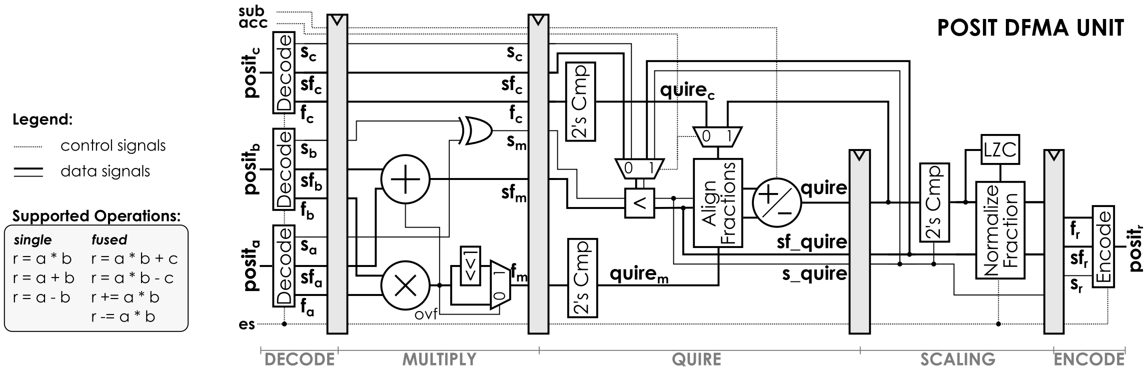

Fig. 2. Proposed Posit DFMA unit.

FIR filter implementations. More recently, Charmichael et al. [13] applied the posit format to DNNs. They proposed the Deep Positron with ≤8-bit posit precisions for the inference phase. By implementing a precision-adaptable FPGA soft-core for the exact multiply-and-accumulate (MAC) operation, they demonstrated that the 8-bit posit precision achieves an accuracy comparable to those obtained with a 32-bit IEEE-754 floating-point implementation. Zhang et al. [20] proposed the first ASIC implementation of a posit-based accelerator, by introducing a posit MAC unit generator for deep learning applications. They presented a 5-stage pipeline design capable of meeting the speed requirements of modern processors.

Despite their success, current posit hardware units adopt a fixed exponent size (defined at design time), which limits the dynamic range that can be represented by the posit format. Contrarily, the solution proposed in this paper supports the definition of the exponent size at runtime, in turn providing enough flexibility to support the entire representable dynamic range for a given posit precision, in a single compute unit.

IV. Methodology

As established by its format, a posit number is defined by its precision (number of bits, n) and its maximum exponent size (es) [11]. Most current hardware implementations [12]–[16] define and fix both parameters at design-time. Although fixing es provides simpler implementations, it also limits both the represented dynamic range of the posit and the balance between the value's dynamic range and decimal precision.

Since the dynamic range of a posit is given by the interval $\left[2^{(2-n)*2^{es}} : 2^{(n-2)*2^{es}}\right]$, a fixed es limits the dynamic range by the scale factor provided by the number of bits that encode the regime (at a proportional cost in fraction bits). However, it is easy to observe that when es is increased by 1, the dynamic range is doubled (in a log_2 scale). As an example, for an 8-bit precision posit, with $es = 0$ (standard) the dynamic range is $\left[2^{-6} : 2^{6}\right]$, and with $es = 1$ it becomes $\left[2^{-12} : 2^{12}\right]$, at the cost of 1 bit of fraction precision.

Accordingly, it is herein proposed a Dynamic Fused Multiply-Accumulate (DFMA) posit architecture with support for configurable exponent size. As a result, since a minimum of three bits of the posit representation have to be reserved for the sign and regime fields, the es parameter can be set at runtime to any value within the range $\left[0 : n-3\right]$. This provides the programmer with enough flexibility to utilize the entire representable dynamic range for a given posit precision, by specifying the exponent size configuration of the set of posit input values. Such an approach not only maintains the support for standard posit configurations but also allows arithmetic operations with very large numbers with low-precision formats.

V. Proposed DFMA Architecture

The proposed posit DFMA (see Fig. 2) comprises a fully pipelined architecture, supporting addition, subtraction, and multiplication operations, together with fused multiply-add and multiply-accumulate operations. However, contrarily to previous approaches, it introduces support for a configurable exponent size by including a set of shifters throughout the pipeline stages. The proposed architecture implements a 5-stage pipeline compute unit, with the following stages: *i) posit decode*; *ii) multiply*; *iii) quire*; *iv) scaling*; and *v) posit encode*. The following sections describe each DFMA stage in detail.

A. Posit Decoding

The *decode* stage translates the tree input posit values ($\mathbf{posit}_a$, $\mathbf{posit}_b$, and $\mathbf{posit}_c$) to their corresponding sign (**s**), scale factor (**sf**) and fraction (**f**) fields. For each input value (see Fig. 3.A), it starts by taking the 2's complement according to the sign bit (**s**). Then, the regime is decoded by counting the number of leading ones or zeroes after the sign bit. To avoid the implementation of both a leading zero and a leading one counter, the binary is first inverted according to the regime's first bit, and only a leading zero counter (LZC) is used. The zero count is then used to shift out the regime from the binary and calculate k (in 2's complement), which is again shifted by es. The binary value is also shifted by es, to split the exponent value and the fraction. These shifters (highlighted in Fig. 3.A) represent the main difference between a standard posit decoder and the proposed architecture. Finally, the shifted k value is added with the exponent to obtain **sf**, and a '1' bit is concatenated with the fraction to obtain **f** (where **f** = *1.f*).

B. Multiplier

The *multiply* stage performs the multiplication of two decoded posit values ($\mathbf{posit}_a$ and $\mathbf{posit}_b$) and propagates the third to the next stage ($\mathbf{posit}_c$). The multiplication operator (see

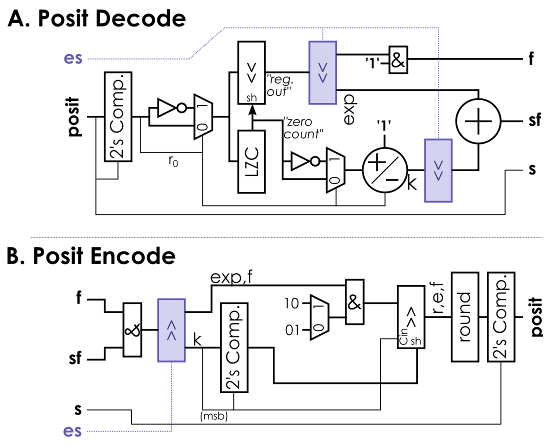

Fig. 3. Posit (A) decoding and (B) encoding modules of the proposed DFMA unit. The highlighted shifters are required to support a variable exponent size implementation.

Fig. 2) adopts a typical floating-point architecture, comprising a sign bit XOR gate, a scale factor adder, and a fraction multiplier. The multiplier is followed by an overflow protection circuit, which adjusts the scale factor accordingly.

C. *Quire Arithmetic Unit*

The quire arithmetic unit is split in two stages: *quire* and *scaling*. This design choice results from the critical path that would otherwise incur from the added logic to support a variable exponent size. In particular, with such an added support, the scale factor maximum value is in the order of 2^{2^n}. Such a value easily overflows the quire's fixed-point format if a standard translation is adopted. Instead, the scale factor is kept in a separate register, and typical floating-point fraction alignment logic for addition/subtraction is adopted in the *quire* stage (see Fig. 2). The `acc` input signal is used to choose one of the adder inputs between the third posit input (**posit**$_c$) or a registered quire value. The `sub` signal is used to choose between addition and subtraction operations.

Finally, the scale factor extraction phase is moved to a subsequent pipeline stage (*scaling*), due to the two's complement, LZC and shifting logic required to re-normalize the fraction and the scale factor for output, according to the `es` signal.

D. *Posit Encoding*

The final *encode* stage (see Fig. 3.B) translates back the **s**, **sf**, and **f** fields of the result to a posit binary format. It starts by concatenating **sf** and **f**, and by taking out the k value through a es-sized right shift (highlighted in Fig. 3.B), leaving in the exponent and fraction fields (unrounded). The k value's 2's complement is taken and the regime is shifted-in to **sf** and **f**, according to k's sign. The resulting binary value is then rounded using straightforward convergent rounding logic. Finally, the 2's complement of the value is taken according to **s**, and the sign is concatenated, resulting the output posit value.

E. *DFMA Operation and Control*

The proposed DFMA architecture provides three control signals `es`, `sub` and `acc`, to configure the exponent size, to choose between addition/subtraction operations, and to activate the quire accumulation. These control signals are propagated to each DFMA stage with the corresponding posit values, allowing the calculation, in each clock cycle, of posit arithmetic operations with different configurations. On the other hand, since the intermediate data formats used by the *multiply* and *quire* stages are independent of the exponent size (`es`), the DFMA is also capable of accumulating the results of fused operations with different posit configurations.

VI. Implementation Results

This section presents FPGA and ASIC implementation and optimization results for the proposed DFMA architecture.

In particular, the proposed DFMA was implemented on a Xilinx Virtex-7 FPGA device (xc7vx485t-2ffg1761). Synthesis and place-and-route results were obtained with the Vivado 2017.2 suite. An ASIC synthesis was also performed for a 45*nm* technology, by considering the Nangate 45*nm* PDK. Hardware resources and power estimation results were obtained with Cadence Genus 19.11. The DFMA was also functionally verified with the SoftPosit library.

To validate the proposed DFMA implementations, they were compared with state-of-the-art posit MAC units. In particular, the FPGA implementation is compared with the Deep Positron (DP) [13] and the MAC units from [15]; and the ASIC implementation is compared with the MAC units from [20]. Hence, although the proposed DFMA can be implemented with any precision, implementation results for 8, 16, and 32-bit DFMA configurations were herein considered solely to obtain a fair comparison with the state-of-the-art references.

A. *FPGA Implementation*

The obtained FPGA implementation results (see Table I) show that the proposed DFMA requires a reduced amount of hardware resources for the considered precision configurations. As it could be expected, for the 8-bit and 16-bit DFMA configurations, it was observed that the critical path is imposed by the encode stage. This is mainly due to the logic complexity of the required operations to translate the intermediate data format to posit. However, when considering the 32-bit configuration, the critical path is imposed by the scaling stage, due to the logic required to normalize a 512-bit quire. To counteract what would be a prohibitive delay, the scaling stage is split after the LZC operation. This way the critical path becomes the quire stage.

Accordingly, for the 8-bit configuration, the DFMA only requires 667 LUTs, amounting to a 0.22% utilization of the FPGA's available resources. This, combined with an attained maximum operating frequency of 200 MHz, makes the proposed DFMA particularly suited for deployment in FPGA accelerators. When considering the higher 32-bit precision, the resource requirements increase to 4134 LUTs and the operating frequency drops to 85 MHz, mainly due to the size of the quire (512 bits).

When compared to DP [13], for an 8-bit configuration, the proposed DFMA is capable of representing a dynamic range 32x larger, given by $log_2(2^{(n-2)*2^{es}}/2^{(2-n)*2^{es}})$, as it can be

TABLE I
COMPARISON OF FPGA IMPLEMENTATION RESULTS.

	MAX. EXP. SIZE (es)	DYN. RANGE (log_2 SCALE)	PIPELINE STAGES	NO. LUTS	NO. REGS.	NO. DSPS	FREQ (MHZ)	OP. EXEC. TIME (ns)
8-bit DFMA	5	384	5	667	205	0	200	25.0
8-bit DP [13]	0	12	4	400	n.a.	0	230	17.4
16-bit DFMA	13	$2.2*10^5$	5	1344	407	1	175	28.6
16-bit MAC [15]	1	56	28	1409	1763	1	311	90.0
32-bit DFMA	29	$3.2*10^{10}$	6[(1)]	4134	1580	4	85	70.6
32-bit MAC [15]	2	240	40	5068	6256	4	112	357.1

[(1)] The *scaling* stage was split to cope with the delay imposed by the normalization of a 512-bit quire.

TABLE II
COMPARISON OF ASIC IMPLEMENTATION RESULTS.

	MAX. EXP. SIZE (es)	DYN. RANGE (log_2 SCALE)	PIPELINE STAGES	ASIC TECH.	DELAY (ns)	AREA (μm^2)
8-bit DFMA	5	384	5	$45nm$	1.2	9601
8-bit DFMA	5	384	6	$45nm$	0.95	10153
8-bit MAC [20]	0	12	5	$28nm$	1.0	1800
8-bit MAC [20]	4	192	5	$28nm$	1.0	1116
16-bit DFMA	13	$2.2*10^5$	5	$45nm$	1.5	32649
16-bit DFMA	13	$2.2*10^5$	6	$45nm$	1.2	33903
16-bit MAC [20]	1	56	5	$28nm$	1.3	3850
16-bit MAC [20]	5	896	5	$28nm$	1.3	3533
32-bit DFMA	29	$3.2*10^{10}$	5	$45nm$	1.8	95861
32-bit DFMA	29	$3.2*10^{10}$	6	$45nm$	1.5	112350
32-bit MAC [20]	2	240	5	$28nm$	1.6	10550
32-bit MAC [20]	8	15360	5	$28nm$	1.6	8992

observed in Table I. This is achieved with a 1.6x resource overhead (measured with the utilization of FPGA LUTs) and a 30 MHz decrease in the maximum operating frequency.

Naturally, when comparing higher precision DFMA configurations with the corresponding MAC [15] counterparts, the observed difference between the representable dynamic range becomes larger. It is increased by 4 and 8 orders of magnitude for 16 and 32-bit precisions, respectively. Such an improvement results in lower operating frequencies (80 MHz on average), when compared to the MAC [15] implementations (see Table I). However, the observed decrease is mostly because the MAC [15] units are implemented by heavily pipelined architectures. Hence, should the proposed DFMA adopt a deeper pipeline architecture, the attained operating frequency would easily match that of the MAC [15] units.

Accordingly, the MAC [15] units 16 and 32-bit configurations require significant greater latencies of 23 and 34 clock cycles, respectively, when compared to the DFMA. As a result, the DFMA attains execution times per operation that are 3x and 5x faster than the 16 and 32-bit MAC [15] units, respectively. Moreover, such deep pipeline structures (of the MAC [15] units) impose resource overheads up to 20% higher in the number of utilized LUTs, and as high as 4.3x in the number of registers, when compared to the DFMA. Furthermore, although power consumption values are not available for the MAC [15] units, it is safe to assume that the higher amount of hardware resources would also impose increased power consumption when compared to the DFMA.

B. ASIC Synthesis

The DFMA was also implemented using a $45nm$ ASIC technology. Although implemented in a much smaller process ($28nm$), the MAC units from [20], with different exponent size and precision configurations, are used as baseline references to validate the proposed implementation (see Table II).

Similarly to the FPGA implementation, the critical path of the ASIC implementation also lies in the encode stage. Also, when compared to the reference MAC units, the major architectural difference comes from the introduction of the shifters to deal with the variable es parameter.

Accordingly, the DFMA was synthesized with timing constraints of 1.2, 1.5, and 1.8 ns for the 8-, 16-, and 32-bit precision configurations, respectively. Although these setups are close to the reference values presented by the MAC units [20], by splitting the encode stage after the two's complement module (see Fig. 3.B) the critical path becomes imposed by the quire stage. With such an optimization, it was possible to outperform the reference MAC units by further constraining the DFMA, in turn achieving minimum timing delays of 0.95, 1.2, and 1.5 ns, for the 8-, 16-, and 32-bit configurations, respectively. Such improvement is achieved at a minimal cost of 6%, 4%, 17% area increase, due to the introduction of additional registers.

C. Energy Efficiency Study

The obtained results for the FPGA and ASIC implementations showed that it is possible to adopt the proposed flexibility and the offered high representation range with marginal overheads, especially for very-low precision hardware. To further support such a conclusion, an energy efficiency study was also performed for the presented DFMA implementations. For such purpose, energy efficiency is hereafter characterized by an energy-delay product (EDP) metric ($Power * Delay^2$), which quantifies the trade-off between execution time and energy

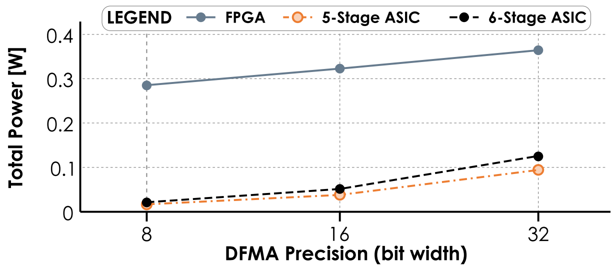

Fig. 4. Power consumption *vs.* bit-precision for the DFMA.

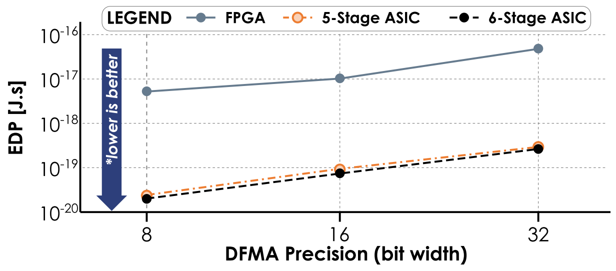

Fig. 5. Energy-delay product *vs.* bit-precision for the DFMA.

consumption. This study does not consider comparisons with the reference setups since the FPGA solutions [13], [15] do not provide power consumption values and the ASIC solution [20] is implemented in a much smaller technology process.

The obtained power estimations (see Fig. 4) show that the FPGA implementation results in power consumptions ranging from 285 mW to 364 mW. This represents a 79 mW difference between the 8-bit and 32-bit configurations. By considering the static power of about 240 mW imposed by the FPGA device, these results show that the deployment of the DFMA in FPGA accelerators would present a minimal impact.

Naturally, the ASIC implementation reduces power consumption, to 17 mW and 95 mW, for the 8-bit and 32-bit configurations, respectively, when compared with the FPGA implementation. On the other hand, when comparing the 5- and 6-stage ASIC implementations, the addition of an extra pipeline stage in the architecture to reduce the critical path (and allow to increase operating frequency) imposes a 25% power increase. However, this trade-off is compensated by an increase in performance, which, in turn, results in 1.2x energy efficiency improvements (see Fig. 5).

In conclusion, the obtained results highlight the viability of adopting the proposed flexibility and the offered high representation range with marginal overheads, especially for very-low precision hardware. In that scenario, the proposed approach becomes particularly suited for deploying energy-efficient accelerators for a wide range of application domains. This is further highlighted when comparing the energy efficiency (through the EDP metric) of the implemented DFMA configurations (see Fig. 5). For the considered setups, the 8- and 16-bit DFMAs provide an energy-efficiency that is one order of magnitude higher than that of the 32-bit configuration.

VII. Conclusions

While the posit format is still in its early stages, there is still the question of whether it will replace or complement the IEEE-754 floating-point standard. Nonetheless, its applicability has already been demonstrated for low-precision arithmetic accelerators. This paper extends such applicability by proposing a new posit DFMA architecture that takes advantage of the full dynamic range that can be encoded by the format, by supporting a runtime-configurable exponent size. When compared to other posit hardware solutions, the proposed implementation presents marginal resource overheads and operating frequency impacts, which are offset by the greater flexibility of providing a very large dynamic range. An energy efficiency study further validated the DFMA as a solution for low-precision hardware.

References

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture." *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.
[2] J. Dean *et al.*, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.
[3] N. P. Jouppi *et al.*, "A domain-specific architecture for deep neural networks," *Communications of the ACM*, vol. 61, no. 9, pp. 50–59, 2018.
[4] J. Fowers *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 1–14.
[5] E. Chung *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.
[6] E. Delaye *et al.*, "Deep learning challenges and solutions with xilinx fpgas," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 908–913.
[7] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 267–278.
[8] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in neural information processing systems*, 2017, pp. 1742–1752.
[9] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.
[10] NVIDIA, "Nvidia tesla v100 gpu architecture." *White paper. [Online]. Available: http://images.nvidia.com/content/volta-architecture/pdf/voltaarchitecture-whitepaper.pdf*, 2017.
[11] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.
[12] R. Chaurasiya *et al.*, "Parameterized posit arithmetic hardware generator," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 334–341.
[13] Z. Carmichael *et al.*, "Deep positron: A deep neural network using the posit number system," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1421–1426.
[14] F. De Dinechin *et al.*, "Posits: the good, the bad and the ugly," in *Proceedings of the Conference for Next Generation Arithmetic 2019*. ACM, 2019, p. 6.
[15] L. Forget *et al.*, "Hardware cost evaluation of the posit number system." in *Compas'2019 - Conférence d'informatique en Parallélisme, Architecture et Système*, Jun 2019, pp. 1–7.
[16] A. Podobas and S. Matsuoka, "Hardware implementation of posits and their application in fpgas," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 138–145.
[17] U. Kulisch, *Computer arithmetic and validity: theory, implementation, and applications*. Walter de Gruyter, 2013, vol. 33.
[18] P. W. Group, "Posit standard documentation," *Release 3.2*, Jun. 2018.
[19] M. K. Jaiswal and H. K.-H. So, "Architecture generator for type-3 unum posit adder/subtractor," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.
[20] H. Zhang *et al.*, "Efficient posit multiply-accumulate unit generator for deep learning applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.