Energy-Efficient MAC Units for Fused Posit Arithmetic

Raul Murillo, David Mallasén, Alberto A. Del Barrio and Guillermo Botella
Department of Computer Architecture and Automation
Complutense University of Madrid, Madrid, Spain
Email: {ramuri01, dmallase, abarriog, gbotella}@ucm.es

2021 IEEE 39th International Conference on Computer Design (ICCD) | 978-1-6654-3219-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCD53106.2021.00032

***Abstract*—Posit arithmetic is an alternative format to the standard IEEE 754 for floating-point numbers that claims to provide compelling advantages over floats, including higher accuracy, larger dynamic range, or bitwise compatibility across systems. The interest in the design of arithmetic units for this novel format has increased in the last few years. However, while multiple designs for posit adder and multiplier have been developed recently in the literature, fused units for posit arithmetic are still in the early stages of research. Moreover, due to the large size of accumulators needed in fused operations, the few fused posit units proposed so far still require many hardware resources. In order to contribute to the development of the posit number format, and facilitate its use in applications such as deep learning, this paper presents several designs of energy-efficient posit multiply-accumulate (MAC) units with support for standard quire format. Concretely, the proposed designs are capable of computing fused dot products of large vectors without accuracy drop, while consuming less energy than previous implementations. Experiments show that, compared to previous implementations, the proposed designs consume up to 75.49%, 88.45% and 83.43% less energy and are 73.18%, 87.36% and 83.00% faster for 8, 16 and 32 bitwidths, with an additional area of only 4.97%, 7.44% and 4.24%, respectively.**

***Index Terms*—Posit arithmetic, Computer arithmetic, MAC, Energy efficiency.**

I. Introduction

The IEEE 754 standard for floating-point arithmetic [1] has been for decades the de facto implementation for the vast majority of scientific computing and real number-based applications. However, over the years diverse problems have been encountered with the standard floating-point format, such as rounding and reproducibility issues, signed zero or excess of NaN representations [2].

Recently, several computer arithmetic encodings and formats, including high-precision anchored numbers from ARM, half-precision arithmetic, bfloat16, etc. have been considered as an alternative to IEEE 754-2008 compliant arithmetic [3]. But probably, one of the most promising contributions is the posit™ arithmetic [4]. Several research efforts have investigated the application of posit arithmetic and its benefits in a wide variety of areas. Its non-uniform representation makes it suitable in deep learning applications [5]–[7], and the higher accuracy of this format in certain regions can accelerate simulations with negligible model degradation [8].

One of the most interesting properties of posit arithmetic is that it includes support for *fused arithmetic*, i.e., the computation of expressions with two or more operations that are exactly evaluated before rounding to a representable value. This kind of arithmetic provides high error reduction in computations such as multiply–accumulate (MAC) or dot product operations. In addition, the lack of intermediate rounding in fused arithmetic can speed-up calculations with a large number of operands [9]. For this reason, MAC units are widely used in deep learning applications to perform and accelerate convolutions and matrix multiplications [10], [11]. The IEEE 754 standard did not include support for fused multiply–add (FMA) operations until it was revised in 2008 [1], specifying that the operation $a + (b \times c)$ must be performed with one single rounding. On the other hand, in the posit literature, some designs of standard posit units such as adders, multipliers, or even dividers have been proposed so far [12]–[15], but it is hard to find fused posit functional units in literature. The posit standard introduces a large size accumulator, so-called *quire*, that allows to perform more fused expressions than FMA, such as fused dot products. This does not only increase the overall accuracy of the operations, but also makes them associative, which enables better compiler optimizations. However, fused operations usually require more hardware resources than standard ones. The use of large-size accumulators for fused arithmetic is a design challenge, especially in its adoption in conjunction with the novel posit format.

This paper proposes the design of posit MAC units for fused arithmetic. The functionality of the fused MAC operation is compared to that of the standard addition and multiplication operations for the case of matrix multiplication. In such case, the use of fused operations is shown to dramatically reduce the computation error. To improve the performance and energy efficiency of the fused operators, several pipeline schemes and large adder designs are compared. The different implementations of the posit MAC units provide efficient solutions for environments with different design goals, such as limited area, or high throughput. The main contributions of this paper are:

- A design of posit MAC unit for fused arithmetic is proposed. The design includes support for quire accumulation, and is compliant with the posit standard draft [16].
- The use of fused posit arithmetic is experimentally shown to reduce precision error when multiplying large matrices by more than two orders of magnitude when comparing

978-1-6654-3219-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICCD53106.2021.00032

with standard operators.

- As a practical use case, the utilization of fused posit arithmetic is evaluated for performing inference on deep neural networks with low-precison formats, reducing the model size by a factor of 4 with a minimal accuracy drop.
- The proposed algorithm is integrated in the open-source FloPoCo framework, allowing to generate MAC units for any possible posit and quire format configurations.
- Different pipeline and adder designs are analyzed when implementing the proposed units, including Brent–Kung and Kogge–Stone adders.
- When compared with cutting-edge posit implementations, the proposed approaches reduce the average energy and latency by up to 88.45% and 87.36%, respectively, with an average area increment of just 7.44%.

The rest of the paper is organized as follows: Section II reviews preliminary concepts about posit arithmetic as well as fused operations in this numerical format. Previous implementations of fused posit units are summarized in Section III. The foundations of the fused MAC operation, as well as the conversions between posit and quire formats, are discussed in Section IV. In Section V, several evaluations of the proposed operator are presented, including accuracy in practical applications, hardware resource of different implementations, and comparison with the state-of-the-art. Finally, Section VI concludes this paper.

II. Background

A. Posit Arithmetic

Posit arithmetic was proposed in 2017 by John Gustafson as a direct replacement for IEEE 754 floats [4]. A posit number format is defined as a tuple $\langle n, es\rangle$, where n is the total bitwidth and es is the maximum number of bits reserved for the exponent field. The posit format just considers two special cases, zero and Not-a-Real (NaR). These exceptions are encoded with all the bits equal to 0 except the leftmost, which is `1` for NaR exception and `0` for zero posit value. Note that this simplifies hardware design with respect to the IEEE 754 standard, which includes signed representations of zero and infinity, as well as multiple bit patters for representing NaN exceptions. For all other cases, posit numbers are encoded with four fields, as shown in Fig. 1:

- A sign bit (s);
- The regime field, of variable-length, composed of equal bits r, and which encodes a scaling factor (k);
- The exponent, of at most es bits (it can be absent), which encodes an integer unbiased value (e);
- The fraction field, composed of the remaining rightmost bits (it can be absent too), and encodes a normalized fraction (f).

Thus, the numerical value X of a generic Posit$\langle n, es\rangle$ is expressed by (1), where $\widehat{e}$ is the exponent value e, bitwise inverted if $s = 1$. The value encoded by the regime field is given by (2), where l is the length of the sequence of equal bits r.

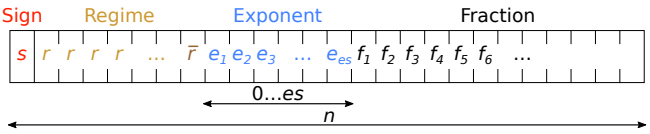


Fig. 1. Layout of an Posit$\langle n, es\rangle$ number.

$$X = 2^{(k \ll es)+\widehat{e}} \times (1 - 3s + f), \tag{1}$$

$$k = \begin{cases} -l & \text{if } r = s \\ l - 1 & \text{if } r \neq s \end{cases} \tag{2}$$

The main differences with floating-point format are the utilization of an unbiased exponent, if such exponent field exists, the fraction hidden bit can be 1 or -2 (for positive or negative posit numbers, respectively), and the existence of the regime field. This new field consists of a sequence of bits with the same value (r) finished with the negation of such value ($\bar{r}$), as shown in Fig. 1. For instance, suppose the Posit$\langle 8, 2\rangle$ with binary encoding `11101010`. The sign bit indicates it is a negative number. The regime field, `110`, encodes the scaling factor $k = -2$. Next, the two exponent bits must be inverted, since $s = 1$, resulting in the exponent value $e = 1$. Finally, the fraction bits are `10`, which correspond to the decimal value 0.5. Thus, substituting these fields in (1) results in the value given by (3):

$$2^{(-2 \ll 2)+1} \times (1 - 3 + 0.5) = -0.01171875 \tag{3}$$

The variable-length regime field may cause the exponent to be encoded with less than es bits, or even no bits if regime is wide enough. The same occurs with the fraction. It is noteworthy that, while the new regime field provides important scaling capabilities that improve the dynamic range of posits, detecting the resulting varying-sized fields adds a hardware overhead.

Posit arithmetic offers compelling advantages over the IEEE 754 floating-point format. Under the same bitwidth, posits provide a better trade-off than floats between dynamic range and decimal accuracy. What is more, it has been shown that an n-bit floating-point adder/multiplier could be safely replaced by an m-bit posit adder/multiplier where $m < n$ [12]. As already mentioned, posit arithmetic includes just two special cases (single 0 and $\pm\infty$), but it also uses a single rounding scheme (round to nearest, ties to even), solving the reproducibility issue mentioned for IEEE 754 floats while simplifying the hardware designs. However, posit arithmetic is still in an early stage of development, and the lack of hardware support for this novel format hinders its use in computer-intensive applications.

B. Fused Operations and Quire Register

Apart from the aforementioned properties of the posit format, the standard [16] introduces the so-called *fused operations*, which allow to perform computations with more than two operands (such as the dot product) without intermediate

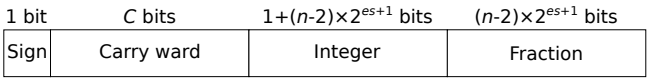


Fig. 2. Quire format encoding for Posit$\langle n, es\rangle$.

rounding, thus reducing the error of such computations. To achieve this, posit arithmetic introduces the concept of *quire*, a fixed-point accumulator large enough to allow for the exact accumulation of posit products. This concept is similar to the Kulisch accumulator for floating-point format [17]. However, this is not included in the IEEE 754 standard. The only similarity to posit fused operations in the IEEE floats is the fused multiply-add operation, which allows to perform a single product and accumulation without rounding, but was not included in the standard until the 2008 revision [1].

According to the posit standard, fused expressions must be distinct from non-fused expressions in source code, and the quire accumulator must be accessible to the programmer. A posit compliant system needs to support rounding from quire to posit and conversion of posit to quire in the matching posit precision. The quire format corresponding to a Posit$\langle n, es\rangle$ configuration is depicted in Fig. 2. The largest exponent for such posit configuration is given by (4).

$$maxE = (n-2) \times 2^{es} \tag{4}$$

To correctly represent a posit in fixed-point format, $2 \times maxE$ bits are needed for the fractional part and $2 \times maxE + 1$ bits for the integer part, so a total of $4 \times maxE + 1$ bits are required to hold the result of multiplying two posit numbers in fixed-point format, plus a sign bit. Adding extra C bits allows to perform up to 2^C sums of products with no overflow risk. A common rule of thumb is to take C carry bits so that the total length of the quire ($qSize$) is a power of two. Table I summarizes the quire sizes and relevant parameters for different posit formats.

TABLE I
QUIRE BITWIDTH PARAMETERS FOR DIFFERENT POSIT FORMATS

Posit		Quire		
n	es	C	$maxE$	$qSize$
8	0	6	6	32
8	1	14	12	64
8	2	30	24	128
16	1	14	28	128
16	2	30	56	256
32	2	30	120	512
64	2	30	248	1024

III. RELATED WORK

Since the appearance of posits in 2017, the interest on hardware implementations for this arithmetic format has increased rapidly, and several approaches have been proposed so far. While many designs of basic posit functional units (such as addition, multiplication and division) are proposed in literature [12]–[15], just a few fused arithmetic operators have been presented so far.

Some works propose FMA units in a similar manner as for the floating-point format [5], [18], [19]: three posits are taken as input operands, two of them are multiplied and added to the third one, and the result is rounded into a posit number. Obviously, this approach introduces multiple intermediate roundings and a higher error in operations such as vector dot product. For large computations, this kind of operators introduce a large overhead derived from the rounding performed in every step of the computation.

On the other hand, [20], [21] present arithmetic units to perform exact sums of products in posit format with a quire accumulator as the input/output of the exact operation. Authors from [20] propose some optimizations to the operations with the quire, such as handling the NaR exception in an extra bit for faster exception checking, or to segment the quire to reduce the long carry propagation delay. The unit presented in [21] is integrated into a RISC-V core for posit arithmetic. While both of these implementations include the quire-to-posit conversion, only the latter includes the posit-to-quire conversion and is compliant with the standard, while [20] considers the quire format as the output of the exact posit multiplier.

IV. POSIT MAC UNIT DESIGN

The architecture of the proposed posit fused MAC unit is shown in Fig. 3. For the sake of clarification, some signals are omitted in the datapath diagram. The design does not depend on the total bitwidth or the number of exponent bits.

In addition to the core of the fused operation, special operations are required by the posit standard to convert from posit to quire format and vice versa. These units can be independent from the fused operations, but are explained in this section as well.

A. Exact Accumulation of Products

As in general MAC operations, three operands are taken as input. In this case, only two of them are in posit format, while the remaining is the quire accumulator, which holds an initial value for the accumulation. This allows to perform even dot products in a fused manner, with a single rounding. The first step in the operation is to decode the fields of the input posits. Then, the product of the two posits is computed by multiplying both fractions (frac) and adding the scaling factors (sf), $(k \ll es) + \widehat{e}$ from (1). In order to transform the intermediate result into quire format, the product is shifted by a number of bits given by the addition of scaling factors. Finally, both quires are added (the one received as input and the one converted from the previous product), so that the result of the operation is a new quire in which successive products can be accumulated in the same manner.

As one may guess, and given the bitwidths shown in Table I, the addition of quires is the module with the highest resource consumption. For example, in a Posit$\langle 32, 2\rangle$ format, the 512-bit quires to be added are much larger than the 29-bit fractions

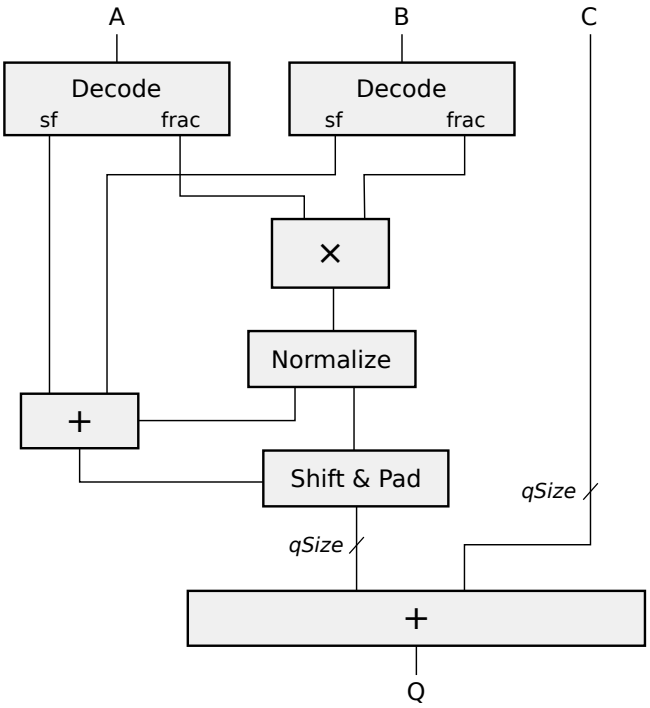


Fig. 3. Architecture of a posit quire multiply-and-accumulate operation.

(including hidden bits) to be multiplied. For this reason, the quire adder is a critical component for the efficiency of the whole fused MAC unit. Previous works have addressed this issue, either by detecting the smallest amount of bits that are necessary to add (only for 32-bit posits) [22], or by segmenting the quire into smaller words (at the cost of a larger number of cycles for correct carry propagation) [20]. In this work, several variants of quire adders and segmented units will be evaluated in Section V.

B. Posit to Quire Conversion

The quire accumulator encodes numbers in fixed-point format, as shown in Fig 2. Therefore, converting a posit value into quire format is straightforward. The posit fraction (including the hidden bits) must be shifted to the correct position according to its scaling factor. As the quire is prepared to hold any possible result of the multiplication of two posits, converting a single posit value to quire encoding would require to pad the shifted fraction with 0's on the right and extending the sign bit on the left.

C. Quire to Posit Conversion

Converting an accumulated result back into posit format requires quite more resources than the previous process. First, it is necessary to detect any possible overflow by checking if any of the carry ward bits are different from the sign bit. Next, the scaling factor of the corresponding posit is obtained by counting the leading 0's (leading 1's in the case of negative values) in the integer part. Then, the fraction of the posit can be directly extracted from the quire. Finally, this fraction must be correctly rounded according to the remaining bits from the quire.

While this process seems to be computationally expensive, it must be noted that it only needs to be done once, at the end of the fused operation. In contrast, standard operations perform rounding and encoding of the results after each single calculation, which can increase the latency of calculations that require a large number of operands [9].

V. Evaluation

The design for posit MAC units presented in this paper has been implemented using FloPoCo, an open-source C++ framework that generates arithmetic datapaths in synthesizable VHDL from the operator specifications via a command-line interface [23]. This tool allows operators to be automatically generated with the specified parameters and maximum frequency. In this case, FloPoCo is capable of generating posit MAC units for different $\langle n, es \rangle$ and $qSize$ with the same base design. In order to verify that the proposed architectures are correct, exhaustive tests were generated with the Universal software library[1] for 8, 10 and 12-bit posits, and corner case tests for 16 and 32-bit posits, with different exponent and quire sizes. All these tests were successful.

In all of the following experiments and the several posit MAC units, standard cell synthesis is performed using Synopsys Design Compiler with a 45-nm library by TSMC with typical case parameters.

A. Area Cost and Computation Error

When compared with standard operations, using fused MAC operations provides clear benefits in terms of computation accuracy, but it comes with higher area and power costs when implemented in hardware. As these operators can perform a large number of multiplications and additions without rounding, General Matrix Multiplication (GeMM) is a good benchmark to evaluate the accuracy of standard against fused operators. In this case, large matrices with uniformly distributed random values in $[-2, 2]$ are generated, and GeMM is performed with fused dot product and with intermediate rounding under multiple formats. The matrices obtained for each configuration are compared with the result for double-precision floating-point numbers, using the Mean Squared Error (MSE) as metric. For the sake of completeness, the MSE obtained using standard floating-point formats is also included.

Fig. 4 shows area synthesis results for both standard and fused posit units, while Table II compares the error obtained in GeMM with each type of operation. In this case, the number of exponent and carry bits are fixed to 2 and 30, respectively, and combinational arithmetic units are considered at synthesis evaluation, but similar results are obtained with other configurations.

As depicted in Fig. 4, an exact accumulator consumes quite more resources than standard operators: up to 58% extra area in the case of 32-bit posits. However, using fused

[1] github.com/stillwater-sc/universal.

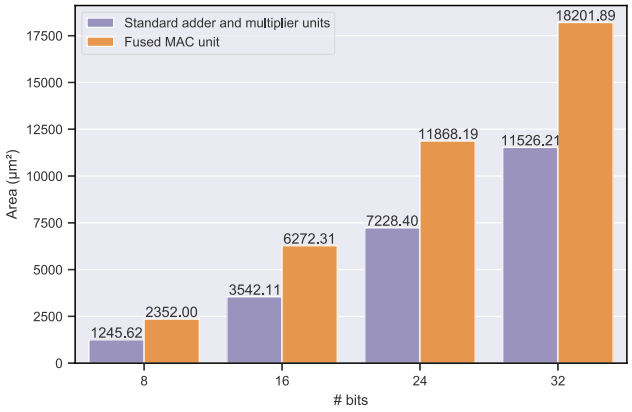


Fig. 4. Area comparison between standard and fused Posit$\langle n, 2\rangle$ units.

TABLE II
GEMM ERROR COMPARISON BETWEEN FLOATING-POINT, STANDARD POSIT AND FUSED POSIT ARITHMETIC

Format	Matrix size	8 bits	16 bits	24 bits	32 bits
IEEE 754	128×128	–	4.49e-04	–	1.00e-11
	256×256	–	8.69e-04	–	3.78e-11
	512×512	–	1.75e-03	–	1.50e-10
Standard posit operations	128×128	2.41e+01	4.62e-04	7.03e-09	1.08e-13
	256×256	1.03e+02	2.18e-03	3.34e-08	5.07e-13
	512×512	3.91e+02	9.52e-03	1.44e-07	2.23e-12
Fused posit operations	128×128	8.25e-01	1.22e-05	1.96e-10	1.99e-15
	256×256	1.67e+00	2.59e-05	4.01e-10	4.29e-15
	512×512	3.50e+00	5.40e-05	8.31e-10	8.93e-15

operations with an exact accumulator for GeMM drastically reduces the error of the calculations, up to three orders of magnitude in the case of the largest matrices. Another advantage of fused operations with regard to standard operators is that intermediate rounding avoidance reduces the latency of large computations. What is more, thanks to the fact that the accumulation of products is a fixed-point addition, such operators offer the possibility to exploit parallelism while maintaining associativity in operations such as dot products, thus eliminating reproducibility issues. With regard to floating-point computations, the results shown in Table II confirms the fact that such format presents lower decimal accuracy than posit arithmetic. It must be noted that in the case of 16-bit formats, half-precision achieves better figures than standalone posit operations, but this is due to the use of a larger datatype, such as 32-bit floats, to accumulate the intermediate dot products on it. This is necessary for half-precision floats in order to avoid overflows or enormous calculation errors due to the limited dynamic range of this format. In any case, employing fused posit operations outperforms both options in several orders of magnitude.

B. Fused Operations in Deep Learning

In this section, the effects of fused posit arithmetic are tested in the deep learning scenario. Due to its multiple properties, such as the higher precision for small values, posit arithmetic seems to be a suitable format for deep learning tasks [4]–[6]. The use of fused arithmetic in this number format can also mitigate the accuracy drop when performing quantization of DNNs. This technique consists on converting the parameters of a trained DNN into smaller data types (typically from 32-bit to 8-bit length). To evaluate this, different DNNs architectures and datasets of reference for image classification are considered. LeNet-5 is a well-known convolutional network that provides nice performance on greyscale-image datasets such as MNIST and Fashion-MNIST. For RGB-images datasets like SVHN or CIFAR-10, deeper networks as CifarNet are recommended. Such DNN models are trained under Posit$\langle 32, 2\rangle$ format using the Deep PeNSieve framework, which allows to generate DNN models and perform both inference and training entirely using posits [7]. The trained models are quantized into Posit$\langle 8, 0\rangle$ format and inference results are compared when standard and fused operations are used to perform GeMM.

Table III shows inference results for the multiple models and formats. Single-precision floating-point accuracy is also shown, since this format is usually considered as a reference in deep learning. The results are consistent with those obtained previously. 8-bit posits have quite less precision than the 32-bit formats, and some accuracy drop is introduced when applying quantization to the DNNs. However, using fused dot product to perform GeMM mitigates this problem, reducing the accuracy drop from 25.43% in the most complex model (CifarNet with CIFAR-10 dataset) to just 0.44%. Recall that, in the case of Posit$\langle 8, 0\rangle$, the quire accumulator has a total length of 32 bits, and the size of the in-memory model is reduced by a factor of 4 with respect to the original 32-bit model.

TABLE III
ACCURACY RESULTS (%) FOR DNN INFERENCE

Format	MNIST	Fashion-MNIST	SVHN	CIFAR-10
Float 32	99.17	89.34	89.32	68.06
Posit$\langle 32, 2\rangle$	99.09	89.90	89.51	69.32
Posit$\langle 8, 0\rangle$	98.77	88.52	81.31	43.89
Fused Posit$\langle 8, 0\rangle$	99.07	89.92	89.13	68.88

C. Pipeline and Design Variations

As mentioned in Section IV, the addition of quires is the most resource-intensive part within the fused MAC operation. The quire accumulator can be quite large even for low bitwidth posits. Such a large adder is the main component that restricts the maximum frequency achievable by these arithmetic units due to the long carry propagation. Therefore, this module deserves special attention when designing fused operators.

This work compares different designs for the final quire adder depicted in the MAC architecture of Fig. 3. In particular, the design of the ripple-carry adder is replaced by

Brent–Kung [24] and Kogge–Stone [25] adder designs[2]. Such designs provide better performance with a higher area cost. In order to improve the throughput of the posit MAC unit in practical applications, different pipeline approaches can be introduced to the posit MAC design. In this case, a 2-stage pipeline architecture separating quire addition from the rest of the operation, and a 3-stage pipeline architecture that also separates posit multiplication from posit-to-quire shifting at different stages are compared.

Unconstrained synthesis results for Posit$\langle 16, 2\rangle$ are graphically shown in Fig. 5, although similar ratios are obtained with other bitwidths. By using a Brent–Kung adder for quire addition instead of a ripple-carry adder, the area is slightly increased ($\sim 5\%$), which is expectable, since the area of this adder is $O(n)$ [26]. However, the power and datapath delay are reduced, resulting in energy savings of more than 26% on average. A larger delay reduction is achieved by using the Kogge–Stone design. In this case, a delay reduction of 63.88% is achieved with the combinational design and about 87.15% in the pipelined cases, resulting in energy savings of 59.62% and 86.12%, respectively. On the other hand, and in contrast to the previous design, there is a considerable area ($\sim 38.84\%$) and power increment ($\sim 9.21\%$). This is also expected, since the area of Kogge–Stone adder is $O(n\log(n))$ [26].

It is noteworthy that the critical path of the pipelined units is the same for both cases. As hypothesized, the final quire adder is the component with the largest delay, even in the case of the Kogge–Stone design. For this reason, the 2-stage pipeline is the most energy-efficient design option.

D. Comparison with State-of-the-art Work

Deep Positron is a DNN kernel accelerator that works with posit arithmetic [5]. The architecture includes an exact MAC posit unit[3] for performing DNN inference with 8 bits or less with comparable accuracy to 32-bit floating-point. This exact MAC unit makes use of the quire, but both input and output are in posit format, so quire-to-posit conversion is incorporated in the unit. This implementation is not compliant with the standard draft, since the quire is not accessible to the programmer, it is just for internal accumulation. A different open-source design of exact accumulator with quire is SmallPositHDL[4]. In contrast with the previous one, this unit presents similar inputs/outputs as the one proposed in this work. Finally, authors in [20] presented MArTo, a library for generating parameterized posit-compliant units, including exact accumulation of products with quire[5].

The posit MAC units presented in those works are pipelined into three stages, generally comprised of 1) posit multiplication, 2) quire alignment, and 3) quire summation. To make a relatively fair comparison, the proposed 3-stages pipelined designs are selected. Also, the same number of exponent ($es = 2$) and carry ($C = 30$) bits are selected for all the designs, so the total bitwidth of the quire is the same in each case.

Table IV lists the detailed comparison of standard cell synthesis between the different designs proposed in the literature. As can be seen, Deep Positron does not provide competitive arithmetic units, especially for large bitwidths, when compared with previous and proposed designs. The quire-to-posit converter included in these MAC units dramatically increases the hardware requirements, so keeping those modules separately seems to be a better design choice, as well as being compliant with the standard. The rest of the works follow that criteria, and they get results that are more similar to each other.

TABLE IV
COMPARISON WITH STATE-OF-THE-ART POSIT MAC UNITS

Design	n	Area (μm^2)	Power (mW)	Max Delay (ns)	Energy (pJ)
Deep Positron [5]	8	5349.15	4.46	4.38	19.51
	16	16147.42	12.18	11.70	142.48
	32	58937.59	40.07	42.01	1683.30
SmallPositHDL	8	4590.87	4.73	3.02	14.30
	16	11263.02	12.37	5.99	74.11
	32	28366.06	34.79	11.94	415.39
MArTo [20]	8	**4234.78**	4.78	3.16	15.10
	16	11011.83	11.88	6.09	72.36
	32	29045.79	33.28	6.00	199.69
Proposed with ripple-carry adder	8	4346.26	4.47	3.01	13.45
	16	**10257.78**	10.91	5.99	65.37
	32	**26204.34**	31.68	11.93	377.91
Proposed with Brent-Kung adder	8	4525.95	**4.18**	2.66	11.11
	16	10710.77	**10.26**	4.57	46.90
	32	27267.44	**30.51**	7.15	218.15
Proposed with Kogge-Stone adder	8	5585.53	4.84	**0.81**	**3.92**
	16	13391.11	11.74	**0.77**	**9.04**
	32	33417.22	33.88	**1.02**	**34.56**

In practically all cases, each of the proposed designs obtains the best results in each of the synthesis sections. The units using a ripple-carry adder provide less area (except compared with 8-bit unit from MArTo), power and delay than previous implementations, resulting in a more energy-efficient design. There is an exception for the 32-bit unit from MArTo, which has approximately the same delay as the 16-bit one. The explanation for this is found in the pipeline of the unit. In the 32-bit case, the 512-bit quire addition is split into two phases, using a 256-bit adder in each. This way, the datapath delay is the same as for the 16-bit unit, which also uses a 256-bit adder to add the quires. Such optimization is not taken in the proposed designs but could be considered for future works. As can be seen, among the implementations from previous papers, SmallPositHDL offers the lowest energy consumption for the 8-bit operator, while MArTo provides the best results for the 16 and 32-bit cases. Thus, for each bitwidth, these operators are considered as the baseline for comparison.

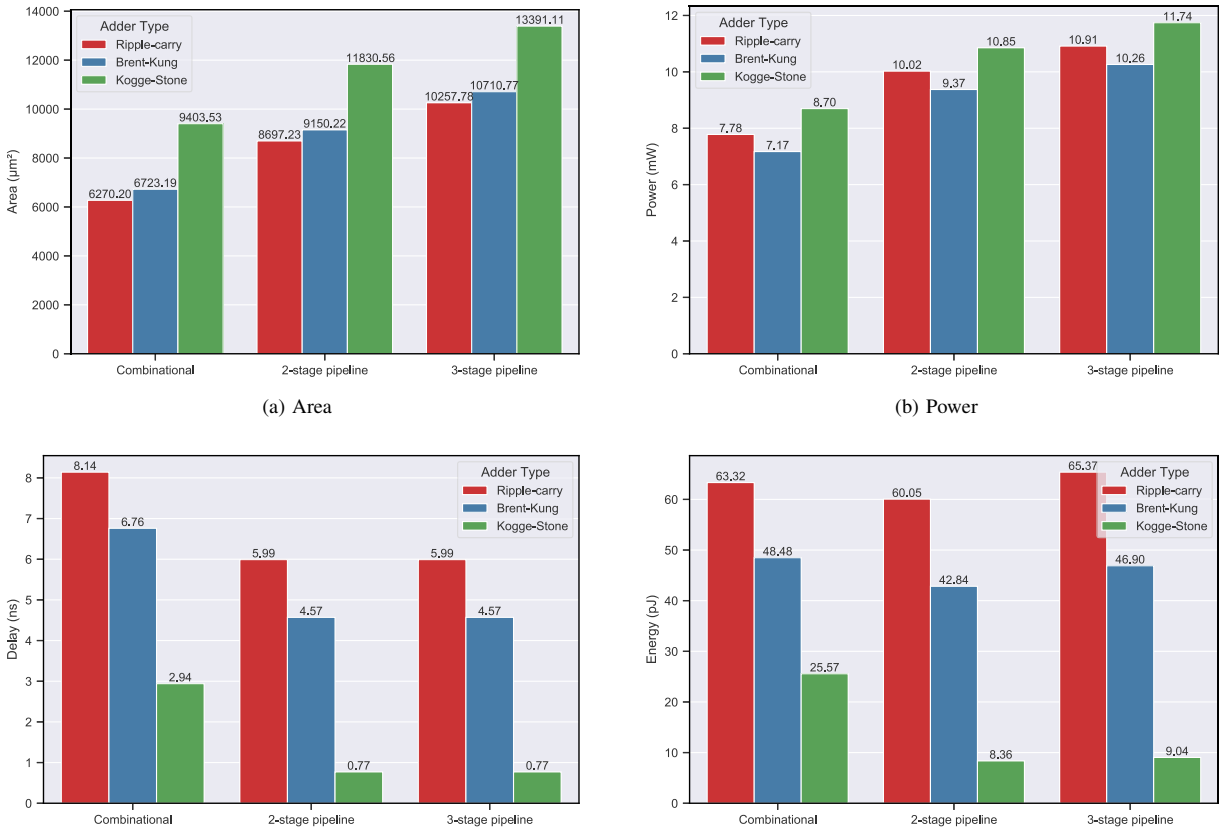
Regarding the proposed implementations, and as the previous experiments show, using a Kogge-Stone adder for quire addition results in the fastest and most energy-efficient fused MAC units. In this case, the proposed design reduces energy consumption by 72.60%, 87.50% and 82.70% for the 8, 16 and

[2]Designs obtained from github.com/albertodbg/vhdl-arithmetic.

[3]Source code from github.com/craymichael/Low-Precision-EMACs/tree/b6fe0d1.

[4]Source code from github.com/starbrilliance/SmallPositHDL/tree/c862e91.

[5]Source code from gitlab.inria.fr/lforget/marto/tree/0bc08ce8.



(a) Area

(b) Power

(c) Total delay

(d) Energy

Fig. 5. Implementation results for different designs of Posit$\langle 16, 2\rangle$ MAC units with 256-bit quire.

32-bit units, respectively, when compared with previous works. On the other hand, this energy reduction is accompanied by an increase in area of 21.66%, 21.61% and 15.05%, respectively, when comparing with the same alternatives. Recall that Table IV compares 3-stage MAC units, but the results shown in Fig. 5 reveal that the 2-stage design is more efficient than the 3-stage one in terms of area, power and energy. In such case, the energy reduction would be slightly similar (75.49%, 88.45% and 83.43%, respectively) while the area of the units would be just 4.97%, 7.44% and 4.24% more than that of the state-of-the-art operators. In addition, the latency of the operators is reduced by 73.18%, 87.36% and 83.00%, respectively.

Finally, performing the quire addition with a Brent-Kung adder is an intermediate compromise between the other two solutions. This design choice requires quite less power, delay and energy than the implementations of previous papers, while still taking a similar or even smaller area.

VI. Conclusions

While the posit format has demonstrated to be a promising alternative to the IEEE754 floating-point standard in a variety of areas such as deep learning and physical simulations, fused arithmetic units for this format are still under development. In this paper, a fused posit multiply-accumulate (MAC) unit design is proposed to contribute to the development of the posit number format, as well as facilitate its use in computationally intensive applications. The experimental results show that using fused posit MAC instead of standard operators for large matrix multiplication reduces the computational error derived from intermediate rounding in more than two orders of magnitude. In addition, as a practical use case, the performance of posit fused operations is evaluated in the context of deep learning, providing similar accuracy as the original models when performing 8-bit quantization. The parameterized design is implemented under different posit configurations, with 2 and 3-stage pipeline, and using multiple adder designs, such as Brent–Kung and Kogge–Stone. In comparison with other posit hardware solutions, the proposed implementation achieves energy and latency reduction up to 88.45% and 87.36%, respectively, with an area increment of only 7.44%.

Acknowledgments

This work was supported in part by a 2020 Leonardo Grant for Researchers and Cultural Creators, from BBVA Foundation, whose id is PR2003_20/01, in part by the EU(FEDER) and the Spanish MINECO under grant RTI2018-093684-B-I00, and in part by the CM under grant S2018/TCS-4423.

REFERENCES

[1] IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008 (Revision of IEEE 754-1985)*, vol. 2008, pp. 1–70, 2008.

[2] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[3] A. Guntoro, C. De La Parra, F. Merchant, F. De Dinechin, J. L. Gustafson, M. Langhammer, R. Leupers, and S. Nambiar, "Next Generation Arithmetic for Edge Computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1357–1365.

[4] J. Gustafson and I. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[5] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, "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.

[6] R. Murillo Montero, A. A. Del Barrio, and G. Botella, "Template-Based Posit Multiplication for Training and Inferring in Neural Networks," *arXiv e-prints*, jul 2019.

[7] R. Murillo, A. A. Del Barrio, and G. Botella, "Deep PeNSieve: A deep learning framework based on the posit number system," *Digital Signal Processing*, vol. 102, p. 102762, 2020.

[8] M. Klöwer, P. D. Düben, and T. N. Palmer, "Number Formats, Error Mitigation, and Scope for 16-Bit Arithmetics in Weather and Climate Modeling Analyzed With a Shallow Water Model," *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 10, 2020.

[9] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[10] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[11] V. Camus, C. Enz, and M. Verhelst, "Survey of Precision-Scalable Multiply-Accumulate Units for Neural-Network Processing," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 57–61.

[12] R. Chaurasiya, J. Gustafson, R. Shrestha, J. Neudorfer, S. Nambiar, K. Niyogi, F. Merchant, and R. Leupers, "Parameterized Posit Arithmetic Hardware Generator," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 334–341.

[13] M. K. Jaiswal and H. K. So, "PACoGen: A hardware posit arithmetic core generator," *IEEE Access*, vol. 7, pp. 74 586–74 601, 2019.

[14] R. Murillo, A. A. Del Barrio, and G. Botella, "Customized posit adders and multipliers using the FloPoCo core generator," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[15] R. Murillo, A. A. Del Barrio, G. Botella, M. S. Kim, H. Kim, and N. Bagherzadeh, "PLAM: a Posit Logarithm-Approximate Multiplier," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–7, 2021.

[16] Posit Working Group, "Posit Standard Documentation," 2018. [Online]. Available: https://posithub.org/docs/posit_standard.pdf

[17] U. Kulisch, *Computer Arithmetic and Validity: Theory, Implementation, and Applications*. De Gruyter, 2013.

[18] H. Zhang, J. He, and S.-B. Ko, "Efficient Posit Multiply-Accumulate Unit Generator for Deep Learning Applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2019-May. IEEE, 2019, pp. 1–5.

[19] N. Neves, P. Tomas, and N. Roma, "Dynamic Fused Multiply-Accumulate Posit Unit with Variable Exponent Size for Low-Precision DSP Applications," *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, 2020.

[20] Y. Uguen, L. Forget, and F. de Dinechin, "Evaluating the Hardware Cost of the Posit Number System," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 106–113.

[21] R. Jain, N. Sharma, F. Merchant, S. Patkar, and R. Leupers, "CLARINET: A RISC-V Based Framework for Posit Arithmetic Empiricism," *arXiv e-prints*, vol. 1, no. 1, pp. 1–18, 2020.

[22] J. Chen, Z. Al-Ars, and H. P. Hofstee, "A matrix-multiply unit for posits in reconfigurable logic leveraging (Open)CAPI," in *Proceedings of the Conference for Next Generation Arithmetic - CoNGA '18*. ACM Press, 2018, pp. 1–5.

[23] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[24] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[25] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

[26] A. A. Del Barrio, R. Hermida, and S. O. Memik, "Exploring the energy efficiency of Multispeculative Adders," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 2013, pp. 309–315.