HUB Meets Posit: Arithmetic Units Implementation

Raul Murillo, Javier Hormigo, Alberto A. Del Barrio, *Senior Member, IEEE*, and Guillermo Botella, *Senior Member, IEEE*

***Abstract*—The posit^{TM} format was introduced in 2017 as an alternative to replacing the widespread IEEE 754. Posit arithmetic provides reproducible results across platforms and possesses tapered accuracy, among other improvements. Nevertheless, despite the advantages provided by such a format, their functional units are not as competitive as the IEEE 754 ones yet. The HUB approach was presented in 2016 to reduce the hardware cost of floating-point units. In this brief, we present HUB posit, a new format to mitigate the hardware overhead of posit units. Results show that it is possible to reach up to 15% and 12% in terms of area-delay product for adders and multipliers, respectively, while maintaining a similar level of accuracy. In addition, synthesis results show that HUB posit units are able to reach higher frequencies than conventional ones.**

***Index Terms*—Posit arithmetic, HUB format, adder, multiplier.**

I. Introduction

DURING the last years, the community has been researching on next generation arithmetic [1] to solve and/or mitigate the inherent problems of the standard IEEE 754 arithmetic to represent real numbers. Among these efforts, we may find high-precision anchored numbers from ARM, half-precision arithmetic, bfloat16, and others. Nonetheless, one of the most promising contributions is the posit arithmetic, introduced in 2017 by Gustafson and Yonemoto [2]. Posit numbers (or posits) have shown promising results in a wide variety of areas, namely: signal processing [3], neural networks [4], [5], or linear algebra [6], [7], [8].

The posit format offers compelling advantages over the IEEE 754 Floating-Point (FP) format. For instance, under the same bitwidth, posits provide a better trade-off than floats between dynamic range and decimal accuracy; they do not support overflow or underflow, which reduces the complexity of exception handling; they can be safely reproduced by different machines, as they possess a unique rounding (to nearest) mode and include only two special cases: single 0 and Not a Real (NaR); or they do not waste patterns in denormalized numbers, among other interesting features [9].

Although posit formats use similar elements to the FP ones, their arithmetic units are still more expensive [6], [10], [11], [12], [13], [14]. Posits are relatively new and their hardware implementations may not have been completely researched yet. Many modern techniques applied to FP units could be potentially applied to posit units to reduce their hardware cost. Recently, the Half-Unit Biased (HUB) approach was presented as a way to mitigate the implementation cost of arithmetic units working with real numbers under rounding to nearest [15]. Thanks to using an implicit Least Significant Bit (iLSB), rounding-to-nearest is performed simply by truncation and two's complement by bit-wise inversion.

Until now, previous works have only focused on classic formats, and the HUB approach has been successfully used to improve the implementation of the basic FP arithmetic operations [15], [16] and fixed-point architectures [17], [18]. However, this technique could be applied to almost any real-number representation, provided that it uses rounding-to-nearest. This is the case for posit format. Thus, in this brief, we study the modification of posit format using the HUB approach.

To the best of our knowledge, this is the first work to propose the implementation of HUB posit units. In this brief, we show how adapting a posit arithmetic unit to use HUB posits improves the implementation results. This HUB posit design is based on the state-of-the-art posit implementations [1], [19], which use two's complement representation of the f ractions. To sum up, the main contributions of this brief are:

- The first definition of a HUB posit format.
- The design of a HUB posit adder and multiplier.
- Experimental error analysis of the new format.
- The ASIC implementation results of the proposed units.

The rest of this brief is organized as follows. Section II reviews the fundamentals of the posit and HUB formats and introduces the concept of HUB posit numbers. Section III covers architectures for implementing HUB posit operations, such as addition and multiplication. Section IV provides an experimental error analysis to evaluate the viability of using HUB formats instead of classic ones and compares the results of ASIC implementations of a basic adder and multiplier. Finally, Section V presents the conclusions of this brief.

Manuscript received 11 July 2023; accepted 8 August 2023. Date of publication 22 August 2023; date of current version 8 January 2024. This work was supported in part by MCIN/AEI/10.13039/501100011033 under Grant PID2019-105396RB-I00 and Grant PID2021-123041OB-I00; in part by the "ERDF A way of making Europe"; in part by the 2020 Leonardo Grant for Researchers and Cultural Creators, from BBVA Foundation under Grant PR2003_20/01; in part by the Comunidad de Madrid under Grant S2018/TCS-4423; in part by the Fondo Europeo de Desarrollo Regional under Grant UMA20-FEDERJA-059; and in part by the Andalucía under Project P18-FR-3130. This brief was recommended by Associate Editor Y. Xia. *(Corresponding author: Raul Murillo.)*

Raul Murillo is with the Department of Computer Architecture and Automation, Faculty of Physics, Complutense University of Madrid, 28040 Madrid, Spain (e-mail: ramuri01@ucm.es).

Javier Hormigo is with the Department of Computer Architecture, Universidad de Málaga, 29071 Málaga, Spain (e-mail: fjhormigo@uma.es).

Alberto A. Del Barrio and Guillermo Botella are with the Department of Computer Architecture and Automation, Faculty of Computer Science, Complutense University of Madrid, 28040 Madrid, Spain (e-mail: abarriog@ucm.es; gbotella@ucm.es).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2023.3307488.

Digital Object Identifier 10.1109/TCSII.2023.3307488

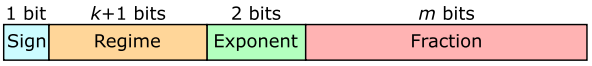


Fig. 1. Generic N-bit posit binary encoding.

II. HUB Posit Definition

A. Background

In posit arithmetic [2], only two special cases are considered: zero and NaR, which are represented as `0...0` and `10...0`, respectively. The rest of the bit patterns are used to represent different real values, which are composed of four fields as shown in Fig. 1: a sign bit (s), several bits that encode the regime value (r), up to two bits for the exponent (e), and the remaining bits for the normalized fraction (f). The regime is a sequence of k identical bits finished with a negated bit that encodes an extra scaling factor r given by (1),

$$r = \begin{cases} -k & \text{if} \quad R_0 = 0 \\ k-1 & \text{if} \quad R_0 = 1 \end{cases}. \tag{1}$$

As this field does not have a fixed length, it may cause the exponent to be encoded with less than two bits, even with no bits, if the regime is wide enough. The same occurs with the fraction, which must be normalized with respect to the size of the fraction field (2^m). The variable length of the regime allows posit arithmetic to have more fraction bits for values close to ± 1 (which increases the accuracy within that range), or to have fewer fraction bits for the sake of more exponent bits for values with large or small magnitudes (increasing the range of representable values). Thus, the real value p of a generic N-bit posit (namely, Posit$N$) is inferred from the bit fields s, r, e, and f as indicated by (2).

$$p = (1 - 3s + f) \times 2^{(1-2s)\times(4r+e+s)}. \tag{2}$$

The main differences with the IEEE 754 floating-point format are the existence of the regime field, the use of an unbiased exponent, and the value of the fraction hidden bit. Usually, in floating-point arithmetic, the hidden bit is considered to be 1. However, in the case of posits, it is considered to be 1 if the number is positive, or -2 if the number is negative [1], [6].

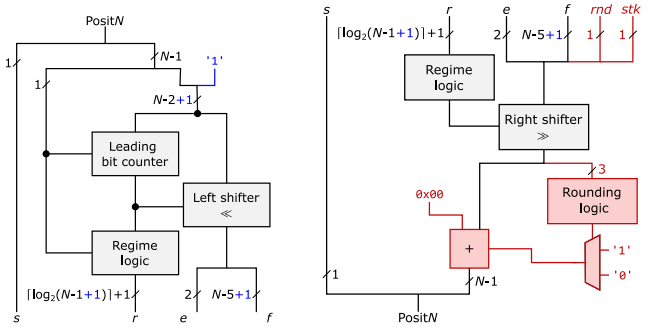
B. Half-Unit Biased Posit Numbers

In [15], HUB FP is defined as an FP number whose mantissa is a HUB fixed-point number and its exponent is a conventional one. That means the mantissa has an iLSB set to one, and consequently, each HUB FP number corresponds to the middle point of two consecutive FP ones. That allows performing rounding-to-nearest by an actual truncation and two's complement by bit inversion. This simplification at the logic level produces remarkable savings in the hardware implementation of FP arithmetic units [15], [16].

Posits only use round-to-nearest, making them a great candidate for using the HUB approach. However, unlike HUB FP, which only applies the HUB approach to the mantissa (the only field rounded in FP), posits require the HUB approach to be applied to the entire posit number since posits might also require rounding the exponent. Considering this, we define a HUB posit as a posit number with an iLSB set to one, except for the zero value. Hence, an N-bit HUB posit is like a conventional $(N+1)$-bit posit but has the LSB implicit and is always set to one. Consequently, the iLSB is part of either the regime, if there is no negated regime bit in the posit number; the exponent, if such a field has less than two bits; or the fraction, if the posit number has the maximum number of exponent bits. The introduction of the iLSB produces that, in the HUB version, the numbers exactly represented by the conventional posit format are shifted to the midpoint between those and the next posit number (in logarithmic scale when affecting the exponent). Consequently, round-to-nearest is obtained by truncating either the fraction or the exponent when coding a HUB posit number.

TABLE I
Conventional and HUB Values for Posit6

Binary	Conventional				HUB			
	r	e	f	value	r	e	f	value
000011	−3	2	0	0.0009765625	−3	3	0	0.001953125
001011	−1	1	0.5	0.1875	−1	1	0.75	0.21875
010000	0	0	0	1.0	0	0	0.25	1.25
011111	4	0	0	65536	5	0	0	1048576
100001	−4	0	0	−65536	−4	2	0	−16384
110000	0	0	0	−1.0	0	0	0.25	−0.875
110101	0	2	0.5	−0.1875	0	2	0.75	−0.15625
111101	2	2	0	−0.0009765625	2	3	0	−0.00048828125



(a) Decoder (b) Encoder

Fig. 2. Architecture of a HUB posit decoder and decoder modules.

Table I shows several examples comparing the values represented for conventional and HUB 6-bit posit numbers. The **r**, **e**, and **f** columns represent the corresponding values in (2) extracted from the bit representation, including the iLSB for the HUB version. Notice how the HUB format always results in values greater than the corresponding posit in both positive and negative cases, since all values are biased in the same direction.

III. HUB Posit Architectures

Posit arithmetic units first decode the operands to obtain the exponent and the fraction. This is performed by detecting the size of the regime and shifting accordingly the exponent and fraction bits to the left. In this way, the size of the exponent and fraction are expanded to their regular size. Fig. 2(a) shows the hardware involved in this operation. In contrast to the FP implementation, where the inclusion of the iLSB can be delayed to just right before the fixed-point operation [20], for posit numbers the iLSB should be included at this stage to append it to the right position (either the fraction, the exponent or the regime). Fig. 2(a) illustrates this in blue.

After the inclusion of the iLSB, the obtained value is a regular number but with one bit more. Note that when using a power of 2 as the bitwidth of the posit, the length of the regime is not increased; just the fraction field is increased. Thus, the corresponding number can be operated with the regular posit circuits but considering an extra bit. However, depending on the specific operation, some simplifications are made to these circuits to improve the implementation results, as shown below.

The second main difference happens in the encoding/packing. In this step, the regime is generated based on the scaling factor of the result, and the exponent and fraction bits are shifted to the right accordingly. In the regular implementation, the discarded bits are used for rounding the fraction (or the exponent if all the fraction bits are discarded). This rounding requires some logic and an addition. However, no additional operation is required after performing the shifting in the HUB implementation since the values are rounded by truncation. Fig. 2(b) illustrates the logic removed for the HUB circuit in red.

Besides modifying the decoder and encoder logic, depending on the implemented operation, some other modifications may be applied to the datapath to simplify the circuit or deal with corner cases. The following subsections provide a detailed explanation of additional modifications on the datapath of the multiplier and adder.

A. Multipliers

In order to implement the multiplication, once the posit operands are decoded, the mantissas are built by appending the implicit leading bits (1 or -2) to the fraction bits. Then, both mantissas are multiplied and the exponents are added. Regarding the HUB version, on the one hand, it requires a fixed-point multiplier one bit wider due to the iLSB. On the other hand, the sticky bit computation and the guard bit are not required, since the rounding is eliminated in the encoding step. As shown in Section IV, that means that HUB slightly increases the area for low frequencies, but that growth is compensated when frequency increases, thanks to the shorter datapath.

B. Adders

In order to implement the addition, the mantissas are added after aligning to the right the one with the smallest exponent. Three additional bits, including a sticky bit, are kept from the shifted mantissa for the later normalization and rounding steps to perform the proper rounding to nearest. Regarding the HUB version, these additional bits are not required since no rounding is performed. However, one bit more is needed in the fixed-point adder due to the included iLSB. Moreover, there are a couple of corner cases that need special attention in the HUB version:

- The alignment of a negative number for more bits than the size of the mantissa (i.e., the smallest operand is smaller than the weight of the LSB of the other operand) produces a wrong effective subtraction of one ULP to the bigger mantissa. The sign extension and the limited number of bits of the adder produce that. In the conventional implementation, this situation is spontaneously solved by the rounding. However, it is an issue in the HUB version since the rounding is eliminated. To solve this, a logic right-shifting is used instead of the arithmetic one when this situation is detected.
- When a catastrophic cancellation occurs after right-shifting the smallest number's mantissa by one bit for alignment, the discarded LSB is required for normalization. In the conventional case, this bit is kept in the rounding bit as a side effect. For the HUB version, this bit must be appended to the result of the addition before normalizing it.

IV. Implementation Results and Analysis

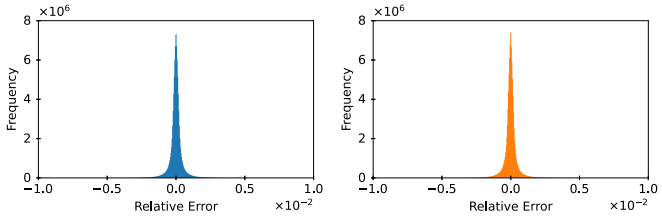
In this section, we evaluate the feasibility of using the HUB posit units for real-number computation. First, we conduct an experimental error analysis to verify that using HUB does not compromise accuracy. Second, we present the main hardware implementation results of the HUB posit circuits and compare them with the conventional ones.

A. Error Analysis

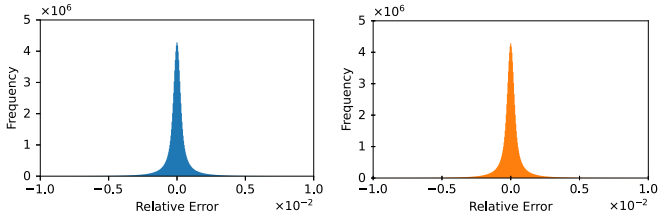
An empirical error study by Monte Carlo simulation is provided to demonstrate that the HUB posit could be used instead of the conventional one in real numbers-specific applications while guaranteeing the same level of precision. To test the arithmetic operations, we utilized two randomly uniform Posit24 numbers within the range $[-2^{20}, -2^{-20}] \cup [2^{-20}, 2^{20}]$, which is an acceptable range for data from real applications and provides variability in the width of the regime. In such a data range, neither overflow nor underflow will occur in the products. They were converted into conventional 16-bit posit and HUB format, using round-to-nearest, so both formats have an initial error with the same probability. The exact result of the operation is computed using twice the number of bits and compared with the obtained result in both conventional and HUB 16-bit posit formats by calculating the relative error. The Universal library [7] was used to emulate the computation of such formats via software.

This experiment was repeated 10^8 times for each tested operation. In the case of addition, the results which produced zero and subtractive cancellations, which were less than 0.01%, were excluded. The high relative errors produced by these cases would hide the behavior of the error in the cases of interest.

Fig. 3 shows the histograms of the computed rounding error for addition and multiplication, whereas Table II shows the main statistical parameters corresponding to these errors, including SNR. Although the rounding error values corresponding to both approaches are always different (see Section II), the probability distributions of these errors are quite similar, as shown by the histogram and the statistical parameters. For both operations, statistical parameters are almost the same, but the mean because the rounding implemented for the HUB version may produce some bias. Nonetheless, the value is still very low and enough for many applications. Unbiased rounding could also be implemented [21], but its study is out of scope.



(a) Conventional posit addition (b) HUB posit addition



(c) Conventional posit multiplication (d) HUB posit multiplication

Fig. 3. Histograms of the rounding error for arithmetic operations.

TABLE II
STATISTICAL PARAMETERS OF ROUNDING ERROR DISTRIBUTION

Format	min	mean	max	σ	SNR_{dB}
Addition					
Posit	−0.9922	4.921e−07	0.9542	2.475e−03	56.02
HUB	−0.9655	−1.697e−05	0.9532	2.465e−03	56.00
Multiplication					
Posit	−0.1158	9.290e−08	0.1163	1.951e−03	25.71
HUB	−0.1298	−4.074e−06	0.1302	1.954e−03	25.81

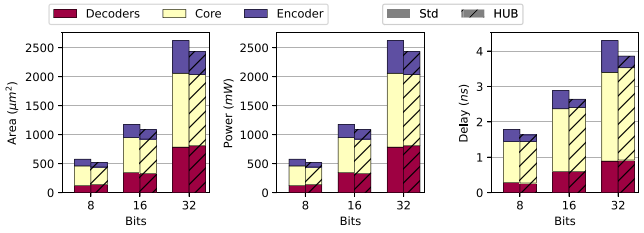


Fig. 4. Synthesis results for N-bit posit adders.

The outcomes from both theoretical and experimental investigations demonstrate that HUB formats can serve as an alternative to the traditional posit representation for handling real-number computations. It is important to note that utilizing the HUB format for solving real-number calculations may not produce identical outputs compared to those obtained using standard posits. Yet, the precision of the calculations would remain unchanged.

B. Synthesis Results and Comparison

To get a detailed evaluation of the hardware cost of the application of HUB format in posit arithmetic, the circuits presented in Section III for HUB posit numbers, along with the posit units presented in [19], have been modeled in VHDL and synthesized using the Synopsys Design Compiler with a 45nm standard cell library by TSMC and no timing constraints. Fig. 4 and Fig. 5 show the area, power, and delay of both approaches for different bitwidths. These metrics are split into components, i.e., encoder, decoder and arithmetic

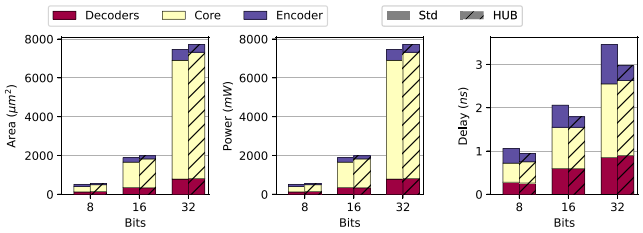
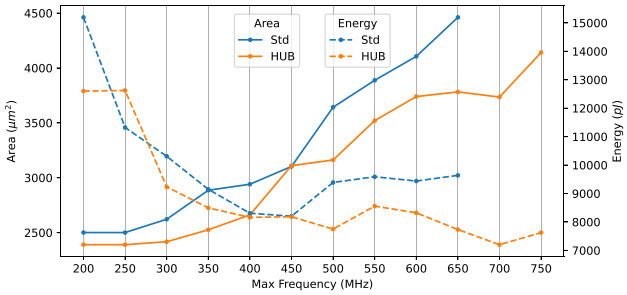
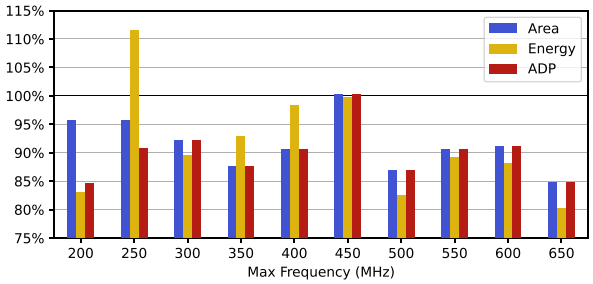


Fig. 5. Synthesis results for N-bit posit multipliers.



(a) Area and energy synthesis results



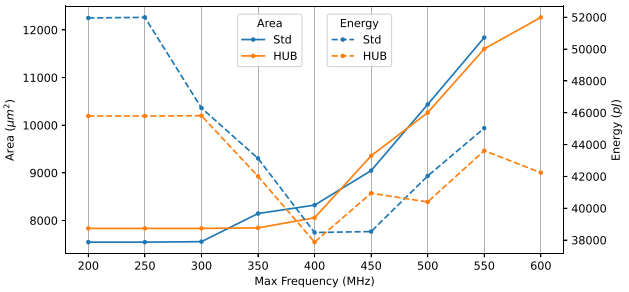
(b) Relative performance ratio of HUB posit format

Fig. 6. Comparison of Posit32 adder implementations.

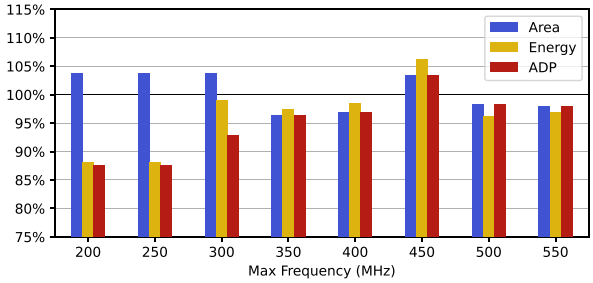
core for a detailed evaluation. The behavior is the same for all bitwidths. The iLSB from HUB has a negligible effect on the decoding and the addition core, while the encoder becomes much simpler, thanks to the rounding removal. In contrast, this simplification of the encoder does not compensate the area and power increment that HUB causes in the multiplication core. However, the delay of the HUB multipliers is significantly decreased, which will compensate the area and power increase for high frequencies.

For a more thorough evaluation, the units have been synthesized targeting different maximum clock frequencies ranging from 200 MHz to 800 MHz. The area and power-delay-product (PDP or energy) of both approaches are compared. Since similar behavior is observed for the different bitwidths, only the results for 32 bits are presented.

Fig. 6(a) shows that the HUB adders generally require significantly less area than conventional ones. A similar behavior is observed in energy consumption since most HUB results clearly outperform the conventional ones. It is also remarkable that HUB adders reach 750 MHz while the conventional ones only reach 650 MHz. This is because the HUB savings are in the rounding logic, which is in the critical path. In order to ease the comparison, Fig. 6(b) represents the normalized version of these results along with the area-delay product (ADP).



(a) Area and energy synthesis results



(b) Relative performance ratio of HUB posit format

Fig. 7. Comparison of Posit32 multiplier implementations.

As expected, the area and energy reduction are very significant, reaching up to 15% for the area and 20% for the energy with the maximum clock frequency. The average reduction for area and energy is 8.5%. Consequently, the HUB version reduces the ADP up to 15%, with a mean of 10%.

The differences in area are more subtle in the case of the multipliers. As shown in Fig. 7, they are within ±4%. Conventional posit multipliers have less area when targeting lower frequencies, whereas the area is generally a bit lower for the HUB case when targeting higher frequencies. In contrast, the energy consumption and ADP are generally lower for the HUB multipliers, especially for lower frequencies, where the reduction is by more than 12%. This is in line with the results shown in Fig. 5, which reveal a lower delay for HUB operators. On average, the energy is reduced by 3.7% and the ADP by 4.8%. Moreover, like the adders, the HUB multipliers reach 650 MHz, while the conventional ones only reach 600 MHz.

In conclusion, using the HUB posit format instead of the conventional one very likely leads to valuable improvements in the implementation of posit arithmetic units. These improvements are more important for the adders, especially targeting high frequencies, but also significant for multipliers, mainly because of the energy reduction. Furthermore, higher frequencies are easier to reach using the HUB posit format.

V. Conclusion

In this brief, we define a HUB posit format and study the implementation of the basic arithmetic units. Although the savings in posit arithmetic units by using the HUB approach are not as pronounced as the ones in FP units, they are still very valuable. That is mainly because these savings come while keeping the same level of accuracy achieved with conventional posits. Furthermore, the HUB posit designs are able to reach higher frequencies. All in all, as this brief shows, HUB posits have proved their viability, so in future work, we will consider implementing more complex operators, such as the fused multiply accumulation units.

References

[1] A. Guntoro et al., "Next generation arithmetic for edge computing," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2020, pp. 1357–1365.

[2] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Front. Innovat.*, vol. 4, no. 2, pp. 71–86, 2017.

[3] R. Murillo, A. A. Del Barrio, and G. Botella, "The effects of numerical precision in scientific applications," in *Proc. Annu. Model. Simulat. Conf. (ANNSIM)*, 2022, pp. 152–163.

[4] R. Murillo, A. A. Del Barrio, and G. Botella, "Deep PeNSieve: A deep learning framework based on the posit number system," *Digit. Signal Process. Rev. J.*, vol. 102, Jul. 2020, Art. no. 102762.

[5] M. Cococcioni, F. Rossi, E. Ruffaldi, and S. Saponara, "A lightweight posit processing unit for RISC-V processors in deep neural network applications," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1898–1908, Oct.–Dec. 2022.

[6] D. Mallasén, R. Murillo, A. A. Del Barrio, G. Botella, L. Piñuel, and M. Prieto-Matias, "PERCIVAL: Open-source posit RISC-V core with quire capability," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 3, pp. 1241–1252, Jul.–Sep. 2022.

[7] E. T. L. Omtzigt and J. Quinlan, "Universal: Reliable, reproducible, and energy-efficient numerics," in *Proc. Conf. Next Generat. Arithm. (CoNGA)*, 2022, pp. 100–116.

[8] L. Ledoux and M. Casas, "A generator of numerically-tailored and high-throughput accelerators for batched GEMMs," in *Proc. IEEE 30th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2022, pp. 1–10.

[9] R. Chaurasiya et al., "Parameterized posit arithmetic hardware generator," in *Proc. IEEE 36th Int. Conf. Comput. Design (ICCD)*, 2018, pp. 334–341.

[10] M. K. Jaiswal and H. K. So, "PACoGen: A hardware posit arithmetic core generator," *IEEE Access*, vol. 7, pp. 74586–74601, 2019.

[11] L. Sommer, L. Weber, M. Kumm, and A. Koch, "Comparison of arithmetic number formats for inference in sum-product networks on FPGAs," in *Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2020, pp. 75–83.

[12] R. Murillo, A. A. Del Barrio, and G. Botella, "Customized posit adders and multipliers using the FloPoCo core generator," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[13] H. Zhang and S.-B. Ko, "Design of power efficient posit multiplier," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 5, pp. 861–865, May 2020.

[14] L. Crespo, P. Tomás, N. Roma, and N. Neves, "Unified posit/IEEE-754 vector MAC unit for transprecision computing," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 5, pp. 2478–2482, May 2022.

[15] J. Hormigo and J. Villalba, "Measuring improvement when using HUB formats to implement floating-point systems under round-to-nearest," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2369–2377, Jun. 2016.

[16] J. Hormigo and J. Villalba, "HUB floating point for improving FPGA implementations of DSP applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 3, pp. 319–323, Mar. 2017.

[17] S. Bose, A. De, and I. Chakrabarti, "Framework for automated earthquake event detection based on denoising by adaptive filter," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 9, pp. 3070–3083, Sep. 2020.

[18] S. S. Da Silva et al., "A new chaos-based PRNG hardware architecture using the HUB fixed-point format," *IEEE Trans. Instrum. Meas.*, vol. 72, pp. 1–8, 2023.

[19] R. Murillo, D. Mallasén, A. A. Del Barrio, and G. Botella, "Comparing different decoding for posit arithmetic," in *Proc. Conf. Next Generat. Arithm. (CoNGA)*, 2022, pp. 84–99.

[20] J. Villalba, J. Hormigo, and S. González-Navarro, "Fast HUB floating-point adder for FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 6, pp. 1028–1032, Jun. 2019.

[21] J. Villalba-Moreno, J. Hormigo, and S. González-Navarro, "Unbiased rounding for HUB floating-point addition," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1359–1365, Sep. 2018.