Customizing the CVA6 RISC-V Core to Integrate Posit and Quire Instructions

David Mallasén
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain
0000-0002-0166-834X

Raul Murillo
Facultad de Ciencias Físicas
Universidad Complutense de Madrid
28040 Madrid, Spain
0000-0003-0204-0797

Alberto A. Del Barrio
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain
0000-0002-6769-1200

Guillermo Botella
Facultad de Informática
Instituto de Tecnología del Conocimiento
Universidad Complutense de Madrid
28040 Madrid, Spain
0000-0002-0848-2636

Luis Piñuel
Facultad de Ciencias Físicas
Universidad Complutense de Madrid
28040 Madrid, Spain
0000-0002-3049-828X

Manuel Prieto-Matias
Facultad de Informática
Instituto de Tecnología del Conocimiento
Universidad Complutense de Madrid
28040 Madrid, Spain
0000-0003-0687-3737

***Abstract*—The posit representation for real numbers, aka Unum-v3, is an alternative to substitute the IEEE 754 standard and thus mitigate the inherent problems to the construction of floating-point numbers. Nonetheless, posits are not standard yet, and previously there was no approach, neither academically nor industrially, which implemented a fully compliant core for deploying this novel format. Recently, the open-source PERCIVAL posit RISC-V core was presented as the first work that fully integrates posit arithmetic and quire capabilities into hardware. In addition, Xposit, a RISC-V extension for posit operations allows for the compilation of C programs with inline assembly posit and quire instructions. As a study platform, PERCIVAL is based on the CVA6 core and has support for both posit and IEEE 754 formats, further permitting the comparison of these representations. This paper details the microarchitecture of the Posit Arithmetic Unit with quire added to this core. It also describes how to perform the necessary additions and modifications to the CVA6 core to add support for the Xposit RISC-V custom extension. Furthermore, FPGA synthesis results highlight the cost of including support for both posits with quire and IEEE 754 formats. This is done by breaking down the area resources needed for every arithmetic configuration.**

***Index Terms*—Arithmetic, Posit, IEEE 754, Floating-Point, RISC-V, CPU**

I. Introduction

Real numbers, as a superset of integer numbers, present unique challenges when dealing with them in a CPU. The IEEE 754 floating-point standard [1] is the most widespread representation that tackles this problem. However, it is not the only option. Posit arithmetic [2] was introduced as an alternative to represent and operate with real numbers on a computer. This novel arithmetic tries to solve some of the inherent problems of floats, such as rounding and reproducibility issues, signed zero, or numerous Not a Number (NaN) representations.

Nevertheless, posits are not yet fully studied and, until recently, there was no approach, neither academically nor industrially, that implements posits and quire into a CPU. The open-source posit PERCIVAL [3] core[1] is presented as a solution to the performance barriers found by previous works [4]–[6]. PERCIVAL is based on the application-level CVA6 [7] RISC-V core. It has full support for 32-bit posits with a 512-bit quire accumulation register. Furthermore, it also includes CVA6's Floating-Point Unit (FPU), thus providing a solution in which both arithmetics can coexist. Therefore, PERCIVAL is a prime platform in which to compare posits and IEEE 754 floats.

In this work, we detail the microarchitecture of the Posit Arithmetic Unit (PAU) in PERCIVAL, as well as the design choices and adaptations done when including this PAU into the CVA6. To have a fully native posit implementation, in addition to the PAU, PERCIVAL also includes a posit register bank in parallel with the floating-point and general-purpose registers. A Field-Programmable Gate Array (FPGA) synthesis evaluation of the different arithmetic configurations that can occur in PERCIVAL is also given. Highlighting the resource cost of the PAU, and especially the large quire register.

The rest of the paper is organized as follows: Section II introduces some background knowledge about the RISC-V Instruction Set Architecture (ISA), the CVA6 core, and posit arithmetic. Then, Section III presents some related work regarding previous attempts at implementing posits in hardware. The PAU design is described in Section IV before introducing the detailed modifications to the CVA6 core in Section V. Sec-

This work was supported by grant PID2021-123041OB-I00 funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF A way of making Europe", by a 2020 Leonardo Grant for Researchers and Cultural Creators, from BBVA Foundation, whose id is PR2003_20/01, and by the CM under grant S2018/TCS-4423.

[1]https://github.com/artecs-group/PERCIVAL

978-1-6654-5950-1/22/$31.00 ©2022 IEEE

tion VI provides an FPGA synthesis evaluation of PERCIVAL. Finally, Section VII concludes this paper.

II. Background

A. RISC-V

The RISC-V ISA [8] is a booming open-source architecture that started its development at UC Berkeley in 2010. It is a free and open standard that has attracted the worldwide interest of both academia and industry. RISC-V follows a modular approach that emanates from the ideas of Reduced Instruction Set Computers (RISCs). It is comprised of a base integer ISA that can be complemented with a set of standard and non-standard optional extensions. The two main base integer ISAs (RV32I and RV64I) establish the user address space as 32-bit or 64-bit. Then, each of the extensions adds some specialized functionality to the base, which is formed by only 40 instructions. This allows to fine-tune the needs of each platform, thus spanning all the range from microcontrollers to data centers.

The most common RISC-V standard extensions are the following:

- Integer multiply and divide (M);
- Atomic memory operations (A);
- Single-precision IEEE 754 floating-point (F);
- Double-precision IEEE 754 floating-point (D).

These general-purpose standard extensions (IMAFD), together with the instruction-fetch fence (Zifencei), and the control and status register (Zicsr), are abbreviated as G. This allows to summarize the set of extensions needed for broad-range general computing. All extensions have 32-bit fixed-length instructions. However, the C standard extension includes 16-bit instructions.

RISC-V defines many other standard extensions. Nonetheless, it also provides a way to integrate customized non-standard extensions. This allows a great amount of flexibility to developers, which can simply add their own extension to the ecosystem. These non-standard extensions occupy the opcode spaces that are left unused for them. In this work, we will use the Xposit custom extension that adds posit arithmetic functionality following the modifications to the F extension proposed by the inventor of posits [9]. This custom extension was presented together with the PERCIVAL core in [3].

B. CVA6

The CVA6 [7] core is one of the most prominent open-source application-class RISC-V core [10]. It originated in the PULP platform with the name Ariane, and since then it has been transferred to the OpenHW Group. CVA6 has a bright future, as it is experiencing an ongoing industrial-grade pre-silicon verification. CVA6 is written in SystemVerilog and is licensed under an open-source Solderpad Hardware License.

The CVA6 is a 6-stage, in-order, single-issue core that implements the RV64GC RISC-V extensions. It is an application-class core because it implements three privilege levels and can run a Linux operating system. The CVA6 SDK [11] contains a Buildroot [12] image to cross-compile an embedded Linux system.

Its execution phase contains an integer Arithmetic Logic Unit (ALU), a multiply/divide unit compliant with the RISC-V M extension, and an IEEE 754 FPU [13] that implements the RISC-V F and D extensions. Its FPU, called FPnew, claims to be IEEE 754-2008 compliant, except for some issues in the division and square root operations.

C. Posit Arithmetic

The posit number format, according to its latest specification [14], defines a posit configuration from its total bit-width n. One of the main benefits of posit arithmetic is that it does not have a variety of special cases that have to be checked. Posits have only two special cases. The value zero is represented as $0\cdots0$, and the Not-a-Real (NaR) is represented as $10\cdots0$. The rest of the bit patterns are composed of the four fields shown in Figure 1.

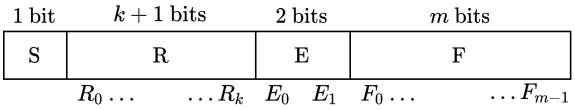

Fig. 1. Posit format with sign, regime, exponent, and fraction fields.

These four bit-fields are:

- The sign bit S, the value of which is $s = 0$ if the value is positive or $s = 1$ if the value is negative.
- The variable-length regime field R, which consists of a series of k bits equal to R_0 and terminated either by $1 - R_0$ or the end of the posit. This field represents a long-range scaling factor r given by:

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

- The exponent field E, consisting of at most 2 bits. This field encodes an integer unbiased value e. Since the regime field is variable-length, one or both of the exponent bits may be located after the least significant bit of the posit. In this case, those bits will have the value 0.
- The variable-length fraction field F, which is formed by the m remaining bits. Its value f will be given by dividing the unsigned integer F by 2^m and therefore $0 \leq f < 1$.

From these fields, we can calculate the real value p of a generic posit as:

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

This is the most efficient decodification of posits, as shown by [15], [16]. The most notable differences in this value representation between posit arithmetic and the IEEE 754 floating-point standard are the existence of the variable-length regime, the use of an unbiased exponent, and the value of the hidden bits [15]. In floating-point arithmetic, the hidden bit is fixed to 0 or 1. However, in posit arithmetic it is kept as 1

if the number is positive, or changed to -2 if the number is negative.

The variable-length regime and fraction fields allow for more flexibility in the trade-off between accuracy and dynamic range that can be achieved by a posit. If the regime field occupies more bits, it represents larger numbers at the cost of lower accuracy. On the other hand, when the regime field consists of fewer bits, posits have higher accuracy in the neighborhoods of ± 1.

Posit arithmetic also includes fused operations using the quire, a $16n$-bit fixed-point 2's complement register. This special accumulation register allows for the execution of up to $2^{31}-1$ Multiply-Accumulate (MAC) operations without intermediate rounding or accuracy loss. These operations are very common when computing dot products, matrix multiplications, or other more complex algorithms. The additional accuracy that can be achieved using the quire can allow the execution of these algorithms with narrower posit configurations [17], [18], thus avoiding the limits that can occur in memory bandwidth.

Currently, one of the main drawbacks of posit arithmetic is its higher area cost. For an accurate comparison between posits and floats, the FPU must be IEEE 754 compliant instead of being limited to normal floats only. Authors in [19] state that posit hardware is slightly more expensive than floating-point hardware that does not take into account subnormal numbers. Moreover, adding a wide quire accumulator register further increases the area cost of implementing these fused operations.

III. Related Work

Since the introduction of posits in 2017, there have been several attempts at developing native hardware for this arithmetic. PACoGen is a complete arithmetic unit with support for the four basic math operations that was presented in [20]. It aims to be a fully flexible unit for any combination of bit-width and number of exponent bits, although it has limitations for 0-bit exponents.

A VHDL generator written in C++ was used in [21] to develop posit adder/subtractor and multiplier fully-parameterized units, including 0-bit exponents, and in [22] to develop a posit logarithm-approximate multiplier for deep learning. Similarly, fused MAC units presented in [23] are used in this work.

Some previous works have also tackled the task of including some posit or quire functionality to a RISC-V core. CLARINET [24] includes a quire register into a RV64GC 5-stage in-order core. Arithmetic operations are executed in IEEE 754 format, as the only posit functionalities added to the core are fused MAC with quire, fused divide and accumulate with quire, and conversion instructions. The floating-point values have to be converted to posit when using the quire.

PERC [25] includes a PAU into the Rocket Chip core to replace the single- and double-precision floating-point units. Nonetheless, quire support is not included, as it is not part of the F and D RISC-V extensions for IEEE-754 floating-point numbers that authors reuse. More recently, PERI [26] also introduced a PAU, but in the SHAKTI C-class 5-stage in-order RV32IMAFC core. This proposal also reuses the F extension instructions and does not include quire support.

In [27] authors follow a different approach. Posits are used as a representation format when storing real numbers in memory using a lower bit-width. Then, the computations are performed using an IEEE 754 FPU, converting back and forth the values when accessing the memory. They also leverage the CVA6 core, and include a light posit processing unit to convert between 8- or 16-bit posits and 32-bit IEEE floats.

The research community has also conducted works aimed at studying the impact of posits in trending technologies such as Artificial Intelligence (AI) [4]–[6]. All of these works emulate posits. Thus, they are limited to small neural networks since their computing capability is limited. However, they have given insight into how posits can improve the execution of deep learning.

IV. Posit Arithmetic Unit

The PAU consists of the posit arithmetic and conversion modules, and a top-level module that orchestrates the execution of the instructions (Fig. 2). This top-level module uses a synchronous handshake interface to transfer the data and read the control signals. The input valid signal announces that the input operands are valid. The ready output signal indicates that the unit can receive a new instruction on the following clock cycle.

The conversion instructions only span 1 clock cycle. However, all arithmetic instructions are multi-cycle. Posit multiplication, approximate division and square root, and quire rounding require an extra clock cycle. Moreover, posit addition and subtraction, and quire fused MAC operations will require 2 extra clock cycles. This is controlled by a latency counter and a small finite-state machine that outputs the ready and valid signals.

Each input instruction contains the operation it must execute, the input operands, and a tag with its transaction identifier. This tag allows to uniquely identify each entry on the scoreboard. When signaling the output of each instruction, besides setting the output data, the PAU must also tag this result with its corresponding transaction identifier.

As can be seen in the PAU diagram of Fig. 2, fused operations make use of the quire accumulator register. In our design, this 512-bit register is allocated internally in the PAU and cannot be accessed directly by the programmer as proposed in [9]. This compromise does not limit substantially the computational capabilities of our core and keeps its hardware cost in check.

V. CVA6 Modifications

The CVA6 core has served as the base platform in which to integrate the posit arithmetic and quire capabilities. The main objective has been to maintain the compatibility between the existing operations and the new functionality. Thus, some modules have been added and others extended to include our new instructions (Fig. 3). This section presents the modifications to the microarchitecture of the CVA6, the changes to its datapath, and the code integration.

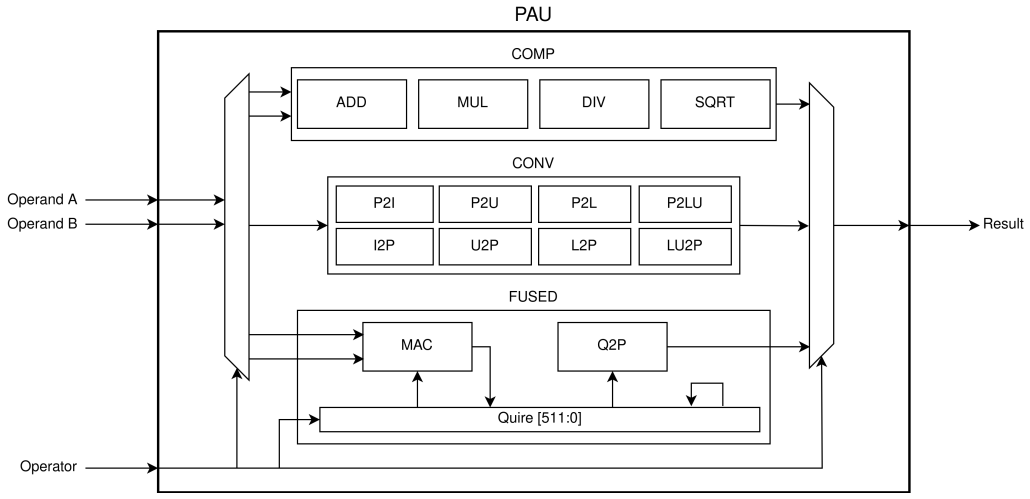

Fig. 2. Internal structure of the Posit Arithmetic Unit (PAU).

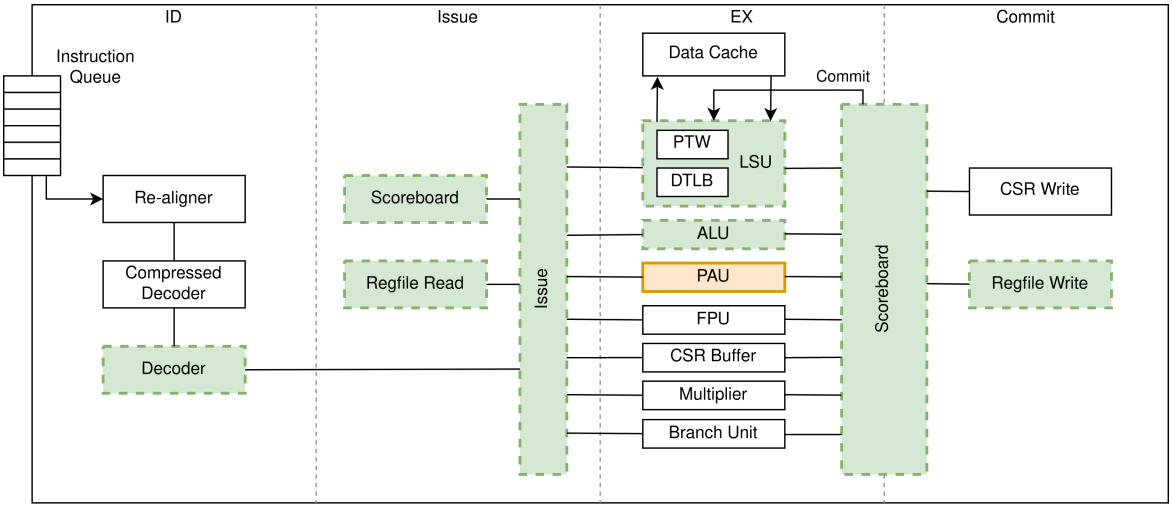

Fig. 3. Summary of the modifications to the CVA6 datapath. Extended modules are highlighted in green.

A. *Microarchitecture Modifications*

The first module that was extended was the decoder. To recognize the new posit and quire instructions, it must know their opcodes. Since the new instructions leverage the Custom-0 opcode space, this is the first comparison performed in the decoder. Then, the different types of instructions are distinguished.

The R-type instructions have the value of `funct3=000`. Each of these instructions will then be selected according to its `funct7` field. The I and R-type instructions corresponding to the loads and stores each use a different `funct3` pattern. The rest of their fields are allocated to register numbers or immediate values.

The CVA6 core contains a scoreboard structure that acts as a Re-Order Buffer (ROB) by tracking: the issued instructions, what functional unit will execute them, and to what register they will write back. It allows the dynamic scheduling of instructions and out-of-order write-back of the functional units. This scoreboard has been adapted to include the posit registers and instructions. Therefore, the datapath can anticipate whether the input data for posit operations will come from a register or will be forwarded directly as the result of a previous operation.

The integer ALU has been extended to also perform posit comparisons. Posit numbers behave exactly like two's complement integers regarding their comparison, so the integer hardware can be reused for this purpose. This is one of the implementation advantages of posit arithmetic. Therefore, we only had to sign-extend the input operands (since the CVA6 has a 64-bit datapath), include the posit operations that the ALU must recognize, and add a multiplexer to perform the minimum and maximum operations.

Furthermore, we added the Posit Arithmetic Unit (PAU) described in Section IV in parallel to the ALU and the FPU. It is located in the execution phase of the core pipeline, connecting the issue module with the scoreboard.

A new posit register file has been added to the CVA6 in parallel with the general-purpose registers and the floating-point registers. This new register file contains a set of 32 registers for posit32 values. Its interconnections with the datapath include read and write control signals as well as output data redirection to the input of the PAU.

Memory loads and stores of posit values are also supported natively. We have extended the functionality of the integer and floating-point load/store unit to include the new posit register file. Currently, this only includes 32-bit posit words, but it could be customized for any other size from a single byte up to 64-bits.

Finally, the datapath has been expanded to include posit control signals and additional interconnections in the issue, execution, and commit stages.

B. Code integration

The PAU is integrated into the datapath of the CVA6 in parallel to the IEEE 754 FPU. The required input and output ports are connected to the top-level module of the PAU, written in SystemVerilog as the rest of the core. However, a VHDL generator written in C++ was used to generate the posit execution units. Consequently, they must be assimilated with the rest of the code.

The QuestaSim simulation tool, the Vivado FPGA synthesis tool, and the Synopsys DC Application-Specific Integrated Circuit (ASIC) synthesis tool allow for the integration of multi-language code. Specifying the appropriate flags when building the simulation project, or when generating the FPGA or ASIC sources, permits the seamless combination of the SystemVerilog and VHDL code. The entity instantiation acts as the interface between both codes, and the same signals with the analogous types of each language are connected together.

Both the CVA6 core and our modifications are released under an open-source hardware license: the Solderpad Hardware License. This is a wraparound to the well-known Apache 2.0 license that extends its coverage to more forms of Intellectual Property (IP). In fact, its preamble states that the licensee can treat the work as if the Solderpad Hardware License were the Apache 2.0 license.

VI. FPGA SYNTHESIS EVALUATION

The PERCIVAL core allows for different configurations of FPU and PAU with quire. The complete version of the core supports 32- and 64-bit floating-point numbers and 32-bit posits with a 512-bit quire. However, any of these arithmetics can be removed depending on the needs of the final platform.

Synthesis results of PERCIVAL with all possible combinations of FPU and PAU with quire are presented in Table I. These results provide some insight into the total hardware cost of a posit and quire enabled CPU. Furthermore, we can compare the resource usage of posits in contrast to IEEE floats. Synthesis was executed on Vivado v.2020.2 for a Genesys II (Xilinx Kintex-7) FPGA with a target frequency of 50MHz. The critical path of the core does not traverse the PAU, as it is multi-cycle, so our modifications do not affect the original frequency of the CVA6 processor. The FPGA is comprised of 50950 logic slices, which have 4 6-input Lookup Tables (LUTs) and 8 Flip-flops (FFs) each.

As can be seen from the table, the standalone CVA6 without any real arithmetic support requires 28950 LUTs and 19579 FFs. This is only 14.2% and 4.8% of the LUTs and FFs available in the Genesys II FPGA, respectively.

Simply supporting 32-bit floating-point operations takes up 22% and 10% more resources respectively, which is a significant increase. However, this is even more significant with double-precision numbers or 32-bit posits with quire. Including doubles implies an increase of 40.7% and 20.5% resources, and for posit32 this ascends to 54.4% and 20.7%. Adding everything up, PERCIVAL with the F, D, and Xposit extensions requires 57129 LUTs and 27996 FFs. This corresponds to 28% and 6.8% of the resources available in our target FPGA, respectively.

All in all, when comparing the PAU with quire with the FPU, the PAU requires significantly more resources. Other previous works reported an increase of around 30-35% more resources when using posit arithmetic when compared to 32-bit IEEE floats [28]. In PERCIVAL, which also includes the 512-bit quire register, this amounts to tripling the resource usage of single-precision floats.

Although these results may not seem very satisfactory, the addition of a large accumulator to remove the computation error in fused MAC operations provides compelling accuracy benefits as shown in [3], [23], [24]. Furthermore, contrary to IEEE floats, the hardware implementation of posit arithmetic is not yet deeply studied. This is the case of the decoding and encoding stages of posit numbers. Most previous works work with posits in sign-magnitude form, whereas recent studies have shown that a 2's complement approach is more efficient [15].

VII. CONCLUSION

In the past years, new emerging floating-point representations have provided alternatives to the widespread IEEE 754 format. In particular, posit arithmetic has been shown to have compelling benefits in areas such as machine learning.

The PERCIVAL posit core was recently presented to advance the native integration of posit arithmetic and quire in hardware. This work has described the implementation details of PERCIVAL, an application-level RISC-V core based on the CVA6. The microarchitecture of the PAU provides insights into its use, and the modifications to integrate this PAU into the CVA6 core highlight the versatility of RISC-V. Furthermore, the hardware cost of synthesizing PERCIVAL into an FPGA has been shown.

We believe that RISC-V and its open-source ecosystem provide an excellent platform in which to study new emerging floating-point arithmetics. Its versatility and robustness allow for a thorough study of these promising alternatives to the IEEE 754 standard. Moreover, the adoption of RISC-V by the industry guarantees that the gap between academic research and real-world applications is closer than ever.

TABLE I
FPGA SYNTHESIS RESULTS OF PERCIVAL WITH DIFFERENT CONFIGURATIONS OF FPU, MARKED AS F AND D FOR 32- AND 64-BIT NUMBERS RESPECTIVELY, AND 32-BIT PAU WITH QUIRE.

	PAU				No PAU			
	F	D	FD	-	F	D	FD	-
Total core (LUT, FF)	(50318, 25727)	(55900, 27652)	(57129, 27996)	(44693, 23636)	(35402, 21618)	(40740, 23599)	(41260, 23945)	(28950, 19579)
FPU area (LUT, FF)	(3726, 1008)	(6352, 1905)	(7612, 2245)	-	(4046, 973)	(6626, 1905)	(8163, 2244)	-
PAU area (LUT, FF)	(11796, 2979)	(11810, 2979)	(11803, 2979)	(11879, 2985)	-	-	-	-

REFERENCES

[1] IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul. 2019.

[2] 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, Apr. 2017. [Online]. Available: https://superfri.org/index.php/superfri/article/view/137

[3] D. Mallasén, R. Murillo, A. A. Del Barrio, G. Botella, L. Piñuel, and M. Prieto, "PERCIVAL: Open-Source Posit RISC-V Core with Quire Capability," *arXiv:2111.15286 [cs]*, Nov. 2021. [Online]. Available: http://arxiv.org/abs/2111.15286

[4] 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, Jul. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S105120042030107X

[5] G. Raposo, P. Tomás, and N. Roma, "Positnn: Training Deep Neural Networks with Mixed Low-Precision Posit," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2021, pp. 7908–7912.

[6] H. F. Langroudi, V. Karia, Z. Carmichael, A. Zyarah, T. Pandit, J. L. Gustafson, and D. Kudithipudi, "Alps: Adaptive Quantization of Deep Neural Networks with GeneraLized PositS," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Nashville, TN, USA: IEEE, Jun. 2021, pp. 3094–3103. [Online]. Available: https://ieeexplore.ieee.org/document/9522706/

[7] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8777130/

[8] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V instruction set manual, volume I: User-level ISA, version 2.0," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[9] J. L. Gustafson, "RISC-V Proposed Extension for 32-bit Posits," Jun. 2018. [Online]. Available: https://posithub.org/docs/RISC-V/RISC-V.htm

[10] A. Dörflinger, M. Albers, B. Kleinbeck, Y. Guan, H. Michalik, R. Klink, C. Blochwitz, A. Nechi, and M. Berekovic, "A comparative survey of open-source application-class RISC-V processor implementations," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, ser. CF '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 12–20. [Online]. Available: https://doi.org/10.1145/3457388.3458657

[11] "CVA6 SDK," OpenHW Group, Apr. 2022. [Online]. Available: https://github.com/openhwgroup/cva6-sdk

[12] "Buildroot." [Online]. Available: https://buildroot.org/

[13] S. Mach, F. Schuiki, F. Zaruba, and L. Benini, "FPnew: An Open-Source Multiformat Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, Apr. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9311229/

[14] Posit Working Group, "Standard for Posit Arithmetic (2022)," Feb. 2022. [Online]. Available: https://posithub.org/docs/posit_standard-2.pdf

[15] R. Murillo, D. Mallasén, A. A. Del Barrio, and G. Botella, "Comparing Different Decodings for Posit Arithmetic," in *Conference on Next Generation Arithmetic (CoNGA)*, 2022.

[16] 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)*. Barcelona, Spain: IEEE, Sep. 2019, pp. 106–113. [Online]. Available: https://ieeexplore.ieee.org/document/8892116/

[17] 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)*, Oct. 2018, pp. 334–341.

[18] M. Klöwer, P. D. Düben, and T. N. Palmer, "Posits as an alternative to floats for weather and climate models," in *Proceedings of the Conference for Next Generation Arithmetic 2019*. Singapore Singapore: ACM, Mar. 2019, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/3316279.3316281

[19] 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)*. Grenoble, France: IEEE, Mar. 2020, pp. 1357–1365. [Online]. Available: https://ieeexplore.ieee.org/document/9116196/

[20] M. K. Jaiswal and H. K.-H. So, "PACoGen: A Hardware Posit Arithmetic Core Generator," *IEEE Access*, vol. 7, pp. 74 586–74 601, 2019.

[21] 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)*, Oct. 2020, pp. 1–5.

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

[23] R. Murillo, D. Mallasén, A. A. Del Barrio, and G. Botella, "Energy-Efficient MAC Units for Fused Posit Arithmetic," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, Oct. 2021, pp. 138–145.

[24] N. Sharma, R. Jain, M. Mohan, S. Patkar, R. Leupers, N. Rishiyur, and F. Merchant, "CLARINET: A RISC-V Based Framework for Posit Arithmetic Empiricism," *arXiv:2006.00364 [cs]*, Oct. 2021. [Online]. Available: http://arxiv.org/abs/2006.00364

[25] M. V. Arunkumar, S. G. Bhairathi, and H. G. Hayatnagarkar, "PERC: Posit Enhanced Rocket Chip," in *4th Workshop on Computer Architecture Research with RISC-V (CARRV'20)*, 2020, p. 8.

[26] S. Tiwari, N. Gala, C. Rebeiro, and V. Kamakoti, "PERI: A Configurable Posit Enabled RISC-V Core," *ACM Transactions on Architecture and Code Optimization*, vol. 18, no. 3, pp. 1–26, Jun. 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3446210

[27] M. Cococcioni, F. Rossi, E. Ruffaldi, and S. Saponara, "A Lightweight Posit Processing Unit for RISC-V Processors in Deep Neural Network Applications," *IEEE Transactions on Emerging Topics in Computing*, no. 01, pp. 1–1, Oct. 2021. [Online]. Available: https://www.computer.org/csdl/journal/ec/5555/01/09583876/1xSHUPCCvlu

[28] S. D. Ciocirlan, D. Loghin, L. Ramapantulu, N. Tapus, and Y. M. Teo, "The Accuracy and Efficiency of Posit Arithmetic," *arXiv:2109.08225 [cs]*, Sep. 2021. [Online]. Available: http://arxiv.org/abs/2109.08225