RF2P: A Lightweight RISC Processor Optimized for Rapid Migration from IEEE-754 to Posit

Hyun Woo Oh, Seongmo An, Won Sik Jeong, and Seung Eun Lee*
Department of Electronic Engineering
Seoul National University of Science and Technology, 01811, Seoul, Republic of Korea
Email: {ohhyunwoo, ahnseongmo, jeongwonsik, seung.lee}@seoultech.ac.kr

***Abstract*—This paper presents a lightweight processor and evaluation platform for migrating from IEEE-754 to posit arithmetic, with an optimized posit arithmetic unit (PAU) supporting existing floating-point instructions. The PAU features a reconfigurable divider architecture for diverse operating conditions and lightweight square root logic. The platform includes a posit-optimized compiler, divider generator, JTAG environment builder, and programmable logic controller. The experimental results demonstrate the successful execution of legacy IEEE-754 code with a small additional workload and up to 60.09 times the performance improvement through hardware acceleration. Additionally, the PAU and divider consume 11.00% and 57.87% fewer LUTs, respectively, compared to the best prior works.**

***Index Terms*—real number arithmetic, processor core, hardware acceleration, reduced instruction set computer, lightweight embedded systems**

I. Introduction

Nowadays, computation for real number arithmetic is a widespread technique in digital applications by reason of ongoing advances in semiconductor technology. As numerous applications are based on real number arithmetic, choosing the appropriate number system for applications is one of the major points in digital systems. In general, the IEEE-754 floating-point (FP) standard has been widely adopted for real number operations due to the practicality and flexibility originating from a wider dynamic range than fixed-point arithmetic since first established in 1985. However, emerging applications that require a higher precision and wider dynamic range prompted the development of novel number systems [1], [2].

Posit is a number format that has been proposed as a potential replacement for the FP [3]. Fig. 1 illustrates the main differences between the FP and posit. The key distinction of posit from IEEE-754 is the dynamic bit-width of the exponent and mantissa parts, which is facilitated by the use of a separate encoding format for the exponent field. This architecture encodes data through the run-length method for the regime field and raw least-significant bits for the exponent field, enabling posit numbers to achieve a wider dynamic range and higher precision in certain regions that are frequently used in computation [3]. Indeed, many studies adopting posit were held and reported the performance enhancements of adopting posit arithmetic in a variety of applications [4]–[6].

Hardware acceleration of posit arithmetic is one of the main topics because of the low throughput of software-only (SW-only) implementation [7]. In lightweight embedded systems where area usage and energy consumption are major concerns, optimizing the microarchitecture to minimize hardware resource usage is a major challenge in the design process [8], [9]. In the case of systems that adopted the FP acceleration, the floating-point unit (FPU) takes a major share of resource usage [10]. As a consequence, many researchers regard posit as an appropriate replacement for the FP because posit generally requires fewer resources, derived by such factors including the abandonment of the extraordinary representations such as subnormal numbers and adoption of the geometric rounding [3], [10], [11]. However, optimizing the resources of the posit arithmetic hardware still has complications originating from the decoding and encoding algorithms and general arithmetic algorithms for posit that are more complex than those for IEEE-754 due to differences in the representation of the exponent part and higher maximum bit-width of the mantissa part. This attribute enforces the entire system to have a lower operating frequency due to the increment of the critical path, slowing down the overall performance. On the other hand, the additional workload for porting legacy codes to the posit hardware is one of the impediments to the prevalent diffusion of posit arithmetic. To address this issue, preserving the former methodology for SW development is essential. For this reason, fostering a legacy-friendly environment by optimizing both hardware design and the compiler should be pursued.

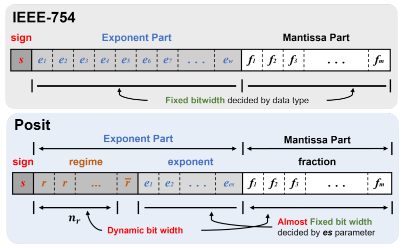

Fig. 1. Representation architecture of both IEEE-754 and posit.

In this paper, we propose a lightweight processor with an adequate platform optimized for migrating from IEEE-754 to posit. The processor architecture is built to support the posit arithmetic operation by utilizing the legacy FP extensions. To provide a wide range of resource optimization for numerous operating conditions, we designed the reconfigurable posit arithmetic unit (PAU). Next, we constructed the integrated platform for the development and evaluation based on the modern system-on-chip field-programmable gate array (SoC FPGA) environment composed of a posit-optimized GNU C compiler (GCC), programmable logic (PL) controller, and JTAG environment builder. We verified and evaluated the PAU as well as the processor through the PL implementation on a number of operating conditions and execution of the functions on the PL utilizing the evaluation platform.

II. Related Works and Motivations

The works on [12] and [13] focus on adding the posit arithmetic support running with custom instruction set extensions (ISE), providing faster execution than previous accelerators connected to the system bus. However, these works do not provide compiler support for a high-level language, which leads to a significant amount of additional workload required for utilizing posit hardware due to the writing of the assembly language. The work on [14] adds the 32-bit and 64-bit posit support on RISC-V utilizing the FP extensions. Through this, a major share of the additional workload reduces because the compiler generates the FP instruction codes from the FP variables and operators written in previous source codes. However, as this work excludes compiler modification, the variable initialization does not create the appropriate binary representation for posit. Therefore, the additional workload to fix the representation still exists. One of the key research that shares a similar aspect in fostering the legacy-friendly environment is explained in [15], which handles this issue through ISA-compatible core integration and compiler optimization. This work presents the reconfigurable PAU configured by the *es* parameter and integrates the PAU to the RISC-V core while providing compatibility for the FP extensions in RISC-V specification. Despite these solutions, applying posit to lightweight systems is still distant as the PAU design in [15] is optimized for a complex processor running on limited operating conditions. Meanwhile, research in [16] suggests a method for reducing the area usage and energy consumption by fixing the configurable parameters of the posit format specifications, i.e., es, and unit bit-width N. The key aspects of our work differentiate from previous works are as follows:

- Designing a reconfigurable PAU that parameterizes the internal architecture of the arithmetic unit, supporting optimization for a variety of lightweight operating conditions while reducing resource usage by fixing the configurable parameters on the posit specifications.
- Providing the rapid evaluation platform for the proposed processor utilizing the SoC FPGA environment, including compiler optimization and subsidiary SW running on the pre-fabricated processor (PS) block in the SoC FPGA.

III. Processor Architecture

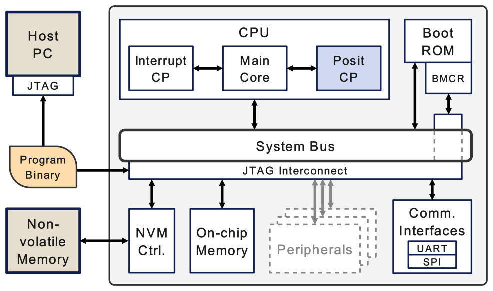

Fig. 2. The architecture of the proposed lightweight processor

Fig. 2 presents the proposed processor architecture. To design a processor providing posit arithmetic with swift evaluation and versatility while using minimal resources, the processor architecture consists of only the essential components: central processing unit (CPU), system bus with nested JTAG interconnect, boot ROM with boot mode configuration register (BMCR), on-chip memory, controller for external non-volatile memory (NVM), communication interfaces. Extendability support for additional peripheral devices, e.g., domain-specific accelerators, is provided through the bus topology.

The CPU comprises the main core, the posit coprocessor, and the interrupt coprocessor (CP) to handle the basic ISA, FP extensions, and trap conditions. The JTAG interconnect provides direct and precise access to the peripherals organizing bus topology from the host device by passing the signals for manipulating the system bus. This characteristic simplifies and improves the accuracy of the evaluation process, as the peripherals are operated and monitored cycle-accurate.

One crucial consideration for applying to the embedded application is the independent operability without any host device. For this reason, the processor supports two separate boot modes called JTAG boot and NVM boot. While the processor is in the booting state, the program in the boot ROM firstly checks the BMCR to select the mode. As the BMCR, similar to the other peripherals, is configured through JTAG interconnect, changing the boot mode can be done from the external host device. Through this, the processor can be used in both the evaluation process and distribution-level applications.

IV. Posit Arithmetic Unit Architecture

Fig. 3 shows the architecture of the posit coprocessor. The coprocessor is composed of the six pipeline stages similar to the main core: instruction fetch, decode, execute, memory 1, memory 2, and write back. The coprocessor maintains the pipeline logic separate from the main core except for the instruction fetch stage, which is shared with the main core and other coprocessors due to the in-order instruction queue. The PAU is embedded in the execute stage and operates through three stages: decoding, calculation, and encoding. The temporary store registers are inserted between each stage to reduce critical path delay. The decoding stage, which is processed by the posit decoder, extracts the two sets of sign, exponent, and mantissa parts from the two operands. These extracted sets are used as input for the calculation stage. The calculation stage contains the arithmetic logic responsible for every operation that is present in MIPS FP extensions: add/subtract (ADD), multiply (MUL), divide (DIV), integer to posit conversion (I2P), square root (SQRT), absolute/negate (ABS/NEG), posit to integer conversion (P2I), and compare (COMP). These logics begin their calculations using the

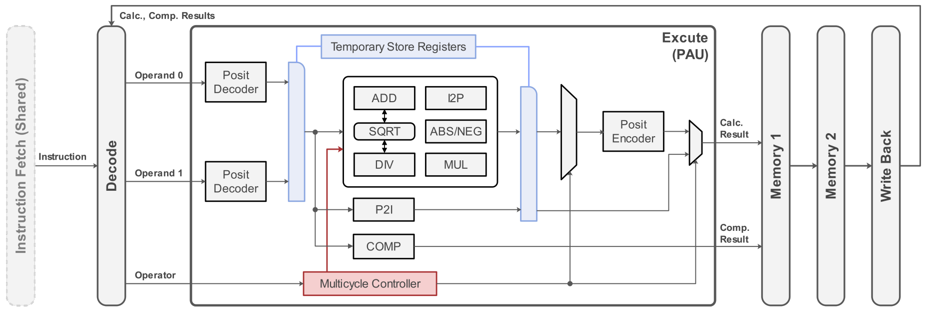

Fig. 3. The architecture of the posit coprocessor

extracted sets simultaneously and save the output results to temporary store registers, except for the I2P, which also operates simultaneously but uses the raw operand inputs. The encoding stage selects the appropriate result from the temporary store registers and compresses the selected result to posit representation by using the multiplexer (MUX) and the posit encoder. As the results of the operations are not limited to the data encoded to posit, the final calculation result is selected by another MUX, which selects the output data from the result of the P2I or the output from the encoder. Considering that the required cycles for each operation differ, the multicycle controller, not part of the stages, exists to handle operations that are processed over multiple cycles by sending the control signals for each logic and triggering the stall signals for the current stage while waiting for the operation completion.

Typically, the division and the square root are regarded as one of the most intensive operations in real number arithmetic. Reducing the critical path delay of these operations to less than that of the main core preserves the operating frequency. Therefore, we focused on two major concerns to enable flexible adoption in a wide range of systems: designing an SW-generated scalable divider architecture reflecting the operating frequency and developing a minimal square root architecture.

A. Decoder and Encoder

Algorithm 1 represent the decoding process of the regime field. The posit decoder uses concatenated 2-input MUXs to extract the exponent and mantissa from any regime and exponent field cases in a single clock cycle. The number of required MUXs in this architecture is derived by equations (1), (2), and (3). The w refers to the bit-width of any data.

$$w_{exp\_val} = \lceil \log_2 max(v_{exp}) \rceil + 1 \tag{1}$$

$$w_{m\_val} = w_{raw\_data} - 3 - es \tag{2}$$

$$N_{MUX} = (w_{raw\_data} - 1) \times (w_{exp\_val} + w_{m\_val}) \tag{3}$$

The encoding process is performed by the reversed execution of the decoding process.

As our design targets lightweight systems operating at low frequency, the maximum path delay of the concatenated MUXs does not violate the timing constraints, which means that the throughput performance is maximized without degrading energy efficiency by avoiding extra stall conditions.

Algorithm 1 Decoding the regime field

```
Input:
    $A_r$ : Bit array storing the regime field data
    $N$ : Bit-width of the data type
    $es$ : Configured parameter for exponent calculation
Output:
    $regime$ : Decoded exponent value
 1: $r \leftarrow A_r[0]$
 2: $n_r \leftarrow 1$
 3: for ($i = 1; A_r[i] == r$ && $i < N; i = i + 1$) do
 4:     $n_r \leftarrow n_r + 1$
 5: end for
 6: if $r == 0$ then
 7:     $regime \leftarrow -2^{es} \times n_r$
 8: else
 9:     $regime \leftarrow 2^{es} \times (n_r - 1)$
10: end if
```

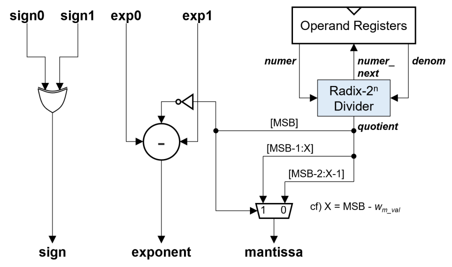

Fig. 4. The architecture of the divider logic

B. Division

Fig. 4 shows the divider logic architecture. The divider logic consists of the operand registers, scalable radix-2^n divider, subtractor, and ancillary components. The calculation flow of the division is as follows:

1) Store the mantissa values extracted by the decoder to the operand registers.
2) Calculate the division for certain clock cycles through the radix-2^n divider and the operand registers.
3) Derive the output exponent value and mantissa value using the subtractor and the MUX.

The radix-2^n division is the algorithm that parameterizes the number of subtractions performed in one clock period. Fig.

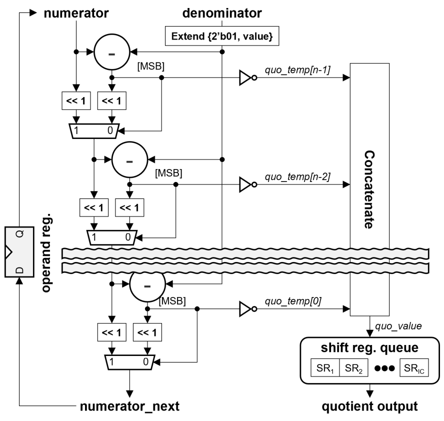

Fig. 5. The architecture of the proposed radix-2^n divider

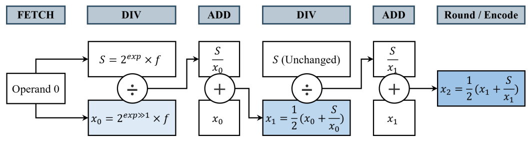

Fig. 6. The sequence to calculate square root

5 illustrates the architecture of the radix-2^n divider. In this architecture, the division is performed by iterating the divider sequence for certain times decided by the w_{m_val} constant and the n parameter. The iteration count is calculated through the following equation.

$$i = \lceil \frac{w_{m\_val} + 2}{n} \rceil \tag{4}$$

We composed the memory to temporarily store the output as the shift register (SR) queue to eliminate the path delay for searching the block entry to be stored.

According to the architecture, the maximum path delay is proportionate to the number of subtractions specified by the n parameter. Thus, the scalable radix-2^n divider provides versatility for numerous operating conditions. Furthermore, we developed the radix-2^n divider generator to provide the divider reflecting the n parameter swiftly.

C. Square Root

The square root operation is performed by repeating the Babylonian method twice times. The Babylonian method is organized by basic arithmetic operations: multiply, add, and divide, as shown in equation (5). The S refers to the input value to acquire the square root.

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right) \tag{5}$$

This attribute eliminates the necessity of extra arithmetic logic because the algorithm is executed by utilizing the maintained logic. Thus, we designed only the routing logic steered by the multicycle controller (see Fig. 6) to compose the equation (5).

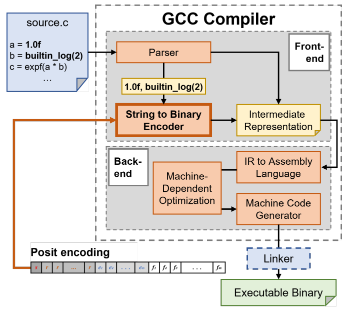

Fig. 7. Compiler operation optimized for posit arithmetic

V. Evaluation Platform

We construct the evaluation platform utilizing the SoC FPGA environment with three additional SW components: posit-optimized compiler, JTAG environment builder, and PL controller. The posit-optimized compiler, which is developed by modifying the string to the binary encoder in the GCC front-end (see Fig. 7), enables swift conversion from IEEE-754-based operation to posit-based operation. Through this compiler, the binary representation converted from the variable initialization codes replaced from the FP to the posit. The JTAG environment builder generates the driver that directly controls the peripherals linked to the system bus with nested JTAG interconnect (see Fig. 2), supporting accessibility from the PS block. The PL controller, which is executed on the Linux operating system, provides user-friendly interfaces to manipulate the designed processor.

VI. Implementation & Evaluation

A. Radix-2^n Divider

We verified and evaluated the scalable radix-2^n divider architecture through register-transfer level (RTL) simulation running on the Vivado simulator. Additionally, we conducted FPGA synthesis through Vivado 2022.2 targeting the Xilinx xc7z020 SoC FPGA for the entire divider logic with several different configurations. Each divider logic was designed for posit(N=32, es=3). Fig. 8 and 9 present the performance of each configuration. Except for the $n = 1$ configuration, the maximum frequency of the divider is approximately inversely proportional to the n parameter, and the throughput performance tends to maintain similar values as expected. The abnormality of the $n = 1$ configuration originates from the other circuits because the path delay of the radix-2^1 divider is already reduced enough. In the case of a relative analysis between area usage, i.e., look-up tables (LUTs) required for constructing the logic, and throughput performance (see Fig. 9), reducing the n parameter makes the area efficiency higher as the required LUTs per throughput decrease except for radix-2^1. These results are in line with the previous abnormality: the radix-2^n divider is relatively small to the other circuits in the entire divider logic. The overall result shows that the designed radix-2^n divider with generation SW provides flexible division logic for a variety of operating conditions.

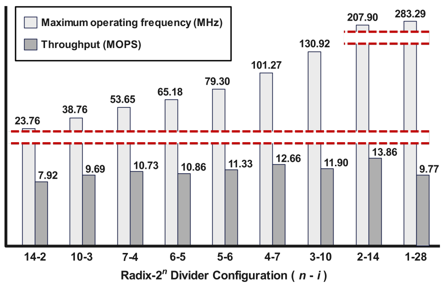

Fig. 8. Maximum operating frequency and throughput on FPGA

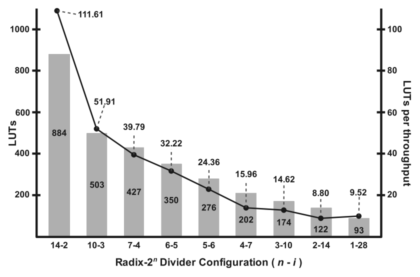

Fig. 9. Throughput performance by area usage on FPGA

To analyze the divider logic more precisely, we synthesized the logic with separated area-first (AF) and throughput-first (TF) constraints using Synopsys Design Compiler and TSMC 180nm technology. Fig. 10 presents the performance analysis for each configuration. As shown in Fig. 10 (a), the area usage varies from 4322.35 μm^2 to 69561.50 μm^2 across the different configurations. This represents a dynamic range of 16.09 and demonstrates versatility. In the case of power and energy (see Fig. 10 (b), and (c)), the result shows that the divider consumes the power of less than 15.53 mW and the energy of less than 273.94 pJ for one division operation, ensuring the aptness for lightweight systems. The maximum throughput of the divider is 56.02 mega operations per second (MOPS) (see Fig. 10 (e). The overall result indicates that the divider architecture is suitable for both lightweight systems and high-performance systems due to the variation in area usage and throughput.

Table I shows the power per throughput (PPT) and energy per throughput (EPT) of each configuration, regarded as the breakdown for each power and energy efficiency. One concern is that the most common configurations, $n = 2$ and $n = 4$, are not always the most efficient in terms of power and energy. The most energy-efficient configuration for the AF is $n = 7$, while $n = 2$ shows the second-worst performance. Though $n = 2$ or $n = 4$ configuration shows the best power efficiency with the TF, embedding those circuits in lightweight systems is not appropriate in terms of power and area. Therefore, utilizing the scalable divider architecture to adopt the adequate configuration with respect to the specifications, e.g., power, energy, and throughput, enhances the overall system efficiency.

B. Processor Architecture & Evaluation Platform

We implemented the proposed processor architecture with our evaluation platform by designing the processor written in Verilog HDL and testing the processor utilizing the platform. Operating at 100 MHz, we selected the $n = 3$ configuration for the divider to maximize performance. The DIV instruction requires three more cycles for division as the temporary registers exist in between the three stages (see Fig. 3).

We verified the proposed PAU and the processor with our evaluation platform through the following processes:

1) Generate the JTAG driver through the builder.
2) Develop the PL controller utilizing the JTAG driver.
3) Build the testing SW with the posit-optimized GCC.
4) Upload the testing SW to the main memory of the designed processor through the PL controller.
5) Check the results through the communication SW.

The testing SW executes the exponential function included in the standard math library. According to the result, the hardware acceleration with the posit coprocessor achieved up to approximately 60.09 times the performance improvement compared to the SW-only implementation.

We evaluated the throughput and area performance through comparative analysis of latencies and FPGA synthesis results of state-of-the-art works and our work. Every PAU presented in the tables is for handling 32-bit data type. The work on [13], in which DIV and SQRT operations are based on approximate

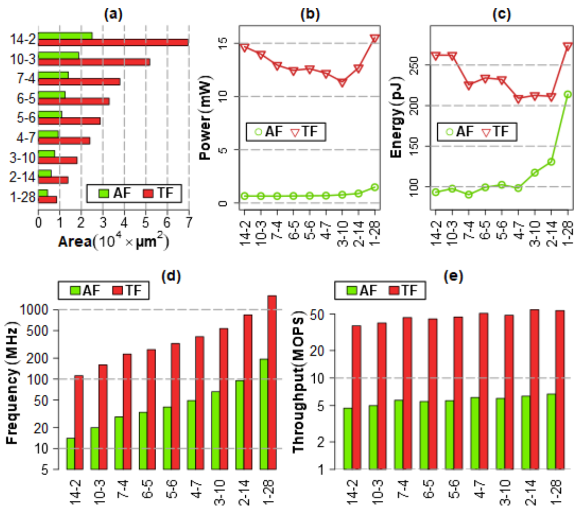

Fig. 10. ASIC synthesis results. **(a)** Area usage. **(b)** Maximum power consumption. **(c)** Energy dissipation for one operation. **(d)** Maximum operating frequency. **(e)** Throughput running on maximum frequency.

TABLE I
EFFICIENCY BREAKDOWNS ON POWER AND ENERGY

	Area-first synthesis results		Throughput-first synthesis results	
	PPT ($\mu W/MOPS$)	EPT ($pJ/MOPS$)	PPT ($\mu W/MOPS$)	EPT ($pJ/MOPS$)
14-2	13.99	19.94	39.34	7.03
10-3	13.02	19.55	34.95	6.55
7-4	11.29	**15.81**	28.21	4.92
6-5	11.91	17.93	28.09	5.28
5-6	11.95	18.12	27.10	4.99
4-7	**11.24**	16.05	23.89	4.10
3-10	12.91	19.61	23.39	4.37
2-14	14.01	20.64	**22.66**	**3.77**
1-28	22.15	32.07	28.37	5.00

TABLE II
COMPARISONS OF THROUGHPUT PERFORMANCE OF THE POSIT INSTRUCTIONS BETWEEN THE PREVIOUS WORKS

	This work	[13]	[14]	[15]
Required cycles for each instruction				
ADD/SUB	5	3	1	6
MUL	5	2	1	6
DIV	13	2[a]	32	3-31[b]
SQRT	27	2[a]	32	3-30[b]
I2P	4	1	1	3
P2I	2	1	1	3
Others	1	1	1	1
FPGA implementation environment				
Target FPGA	xc7z020-1	xc7k325t-2	xc7s75t-1	xc7a100t-1
Target clock	100 MHz	50 MHz	24.5 MHz[c]	100 MHz

[a]Based on the logarithm-approximate multiplier.
[b]Proportionate to the bit-width of the mantissa part of numerators.
[c]Based on the critical path indicated in the timing report.

TABLE III
COMPARISONS OF AREA USAGE BETWEEN THE PREVIOUS WORKS

	This work		[13]		[14]		[15]	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
PAU*	2856	289	4753	255	4046	145	3209	184
DIV	174	66	413	43	1033	145	794	184

*Only for the arithmetic logics.

logic with a maximum relative error of 11.11%, suggests the best throughput despite the target clock being relatively lower than our work, except for ADD and SUB (see Table II). Concerning only the exact arithmetic, our processor has advantages in DIV instruction as the number of required cycles is lower when the bit-width of the mantissa part is higher, compared to the best prior work, [15] (see Table II). In terms of area usage, our PAU and divider consume 11.00% and 57.87% fewer LUTs compared to the best prior works, the PAU of [15] and the divider logic of [13], respectively, demonstrating the compactness of our design (see Table III). Though evaluating the performance of different works with varying specifications, such as operating clock and ISA, is subjective, we consider our work as a promising approach to adopting posit arithmetic in lightweight systems due to the reduced area usage.

VII. CONCLUSION

This paper presented a lightweight processor and evaluation platform for rapid migration from IEEE-754 to posit. The key aspects we considered, versatility and usability, were provided by the following studies: designing the reconfigurable PAU focusing on the most intensive operations such as division and square root, designing the processor architecture supporting posit responsible for existing FP extensions, and constructing the platform to supply the swift conversion of the previous applications and fast evaluation. The experimental results showed that our work is suitable for lightweight systems due to the variable operating frequency and feasibility and, moreover, the reduced area usage compared to the previous studies. In future work, we plan to port a variety of applications to evaluate our designs in detail. Ultimately, we aim to fabricate the processor into a chip to confirm feasibility and compactness.

ACKNOWLEDGMENT

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2023-RS-2022-00156295) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation). This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (2022-0-01013, Development of DRAM PIM semiconductor technology for enhanced computing function for edge, 50%).

REFERENCES

[1] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.
[2] V. Popescu, M. Nassar, X. Wang, E. Tumer, and T. Webb, "Flexpoint: Predictive numerics for deep learning," in *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, 2018, pp. 1–4.
[3] Gustafson and Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 2, p. 71–86, jun 2017. [Online]. Available: https://doi.org/10.14529/jsfi170206
[4] Z. Carmichael, S. H. Fatemi Langroudi, C. Khazanov, J. Lillie, J. 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)*, 03 2019, pp. 1421–1426.
[5] J. Lu, C. Fang, M. Xu, J. Lin, and Z. Wang, "Evaluations on deep neural networks training using posit number system," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 174–187, 2021.
[6] N. Buoncristiani, S. Shah, D. Donofrio, and J. Shalf, "Evaluating the numerical stability of posit arithmetic," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 612–621.
[7] H. W. Oh, W. S. Jeong, and S. E. Lee, "Evaluation of posit arithmetic on machine learning based on approximate exponential functions," in *2022 19th International SoC Design Conference (ISOCC)*, 2022, pp. 358–359.
[8] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.
[9] I. Pérez and M. Figueroa, "A heterogeneous hardware accelerator for image classification in embedded systems," *Sensors*, vol. 21, no. 8, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/8/2637
[10] 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)*, 2018, pp. 334–341.
[11] A. A. Esmaeel, S. Abed, B. J. Mohd, and A. A. Fairouz, "Posit vs. floating point in implementing IIR notch filter by enhancing radix-4 modified booth multiplier," *Electronics*, vol. 11, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/1/163
[12] 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*, vol. 10, no. 4, pp. 1898–1908, 2022.
[13] D. Mallasén, R. Murillo, A. A. D. Barrio, G. Botella, L. Piñuel, and M. Prieto-Matias, "PERCIVAL: Open-source posit RISC-V core with quire capability," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1241–1252, 2022.
[14] A. M Vaidyanathan, G. Bhairathi, and H. Hayatnagarkar, "PERC: Posit enhanced rocket chip," in *4th Workshop on RISC-V for Computer Architecture Research (CARRV)*, no. 8, May 2020.
[15] S. Tiwari, N. Gala, C. Rebeiro, and V. Kamakoti, "PERI: A configurable posit enabled RISC-V core," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 3, Apr 2021. [Online]. Available: https://doi.org/10.1145/3446210
[16] V. Gohil, S. Walia, J. Mekie, and M. Awasthi, "Fixed-posit: A floating-point representation for error-resilient applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 10, pp. 3341–3345, 2021.