Scalable, Reconfigurable RISC-V Vector ALU for Energy-Efficient Embedded AI and Graphics

1st Shaik Maaz Ahmed
Department of ECE
NIT Andhra Pradesh
Tadepalligudem, India
621248@student.nitandhra.ac.in

2nd Dr. Pulikishore Kumar
Department of ECE
NIT Andhra Pradesh
Tadepalligudem, India
pulikishorek@nitandhra.ac.in

3rd Dr. Kiran Kumar Gurrala
Department of ECE
NIT Andhra Pradesh
Tadepalligudem, India
kirankumargurrala@nitandhra.ac.in

***Abstract*—The increasing demand for energy-efficient computing in embedded AI and graphics applications has heightened the need for scalable, reconfigurable vector architectures. This work presents a RISC-V-based Vector Arithmetic Logic Unit (VALU) which addresses these demands while being power efficient. The unit adheres to the RISC-V Vector ISA, supporting 32-bit floating-point and integer operations. Its architecture integrates a 4-lane pipelined datapath and an optimized dot product unit to efficiently handle vector and matrix computations. The RTL design, implemented in Verilog HDL, was functionally validated using the Xilinx Vivado simulation tool. Synthesized with a 14nm process design kit (PDK) in Synopsys Design Compiler, the 4-lane configuration occupies 0.038 mm², consumes 182 uW, and operates at 437 MHz with a datapath delay of 2.29 ns. It achieves a peak throughput of 1.746 GFLOPS, a bandwidth of 111.8 Gbps, and an energy efficiency of 104.27 pJ per operation (EPC). Scaling to 8-lane and 16-lane configurations further boosts performance, with the 16-lane variant reaching 6.168 GFLOPS and 406.3 Gbps. When compared to existing solutions, the design demonstrates substantial improvements, reducing area by up to 66.7% and power consumption by 94.1%, while maintaining competitive performance. These metrics underscore its viability for energy-constrained embedded AI and graphics workloads.**

***Index Terms*—RISC-V, Vector ALU, Energy Efficient Computing, Embedded AI Acceleration, SIMD Architecture.**

I. Introduction

A. Background & Motivation

Traditional general-purpose processors, designed for instruction-level parallelism (ILP), struggle to manage modern workloads requiring high data-level parallelism (DLP), such as neural networks or real-time data processing. While SIMD extensions [6] aim to tackle these challenges by enhancing DLP support, their rigid vector lengths and bandwidth limitations hinder adaptability. GPUs excel in parallel throughput but are often impractical for embedded or battery-powered systems due to high power demands. Conversely, domain-specific architectures achieve efficiency for targeted tasks but typically lack the versatility to accommodate evolving algorithms.

Growing requirements for edge AI [5], digital signal processing [1], and autonomous systems necessitate hardware that balances performance with energy efficiency. Vector processing architectures address this gap by providing scalable parallelism and reduced power consumption. Their configurable vector lengths and efficient resource utilization enable dynamic adaptation to varying computational demands, positioning them as a robust choice for embedded environments where energy constraints and algorithmic diversity coexist.

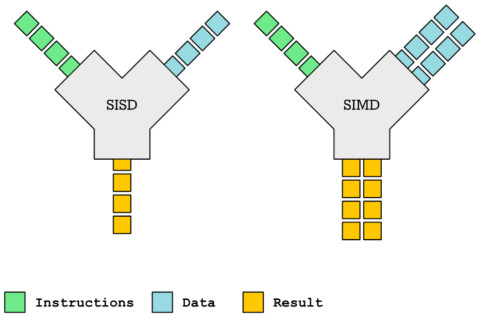

Fig. 1. SIMD Data Model [7]

B. SIMD

SIMD architectures [11] are extensively used in high-performance computing to leverage data-level parallelism (DLP), enabling simultaneous execution of operations across multiple data elements. These units [6] serve as core components in modern processors, excelling in tasks like image processing, machine learning, and scientific simulations that demand parallel processing of large datasets. However, traditional SIMD designs [10] struggle with inflexible vector lengths, poor scalability, and inefficiency when managing irregular or dynamic workloads.

In embedded and edge computing contexts [5] where power, area, and performance trade-offs are critical—these shortcomings are amplified. Vector architectures address these challenges by enabling variable-length vectors and scalable execution lanes, enhancing adaptability. The proposed Vector ALU, compliant with the RISC-V Vector Extension (RVV) [12], extends SIMD principles with configurable vector processing, low-power operation, and optimized support for floating-point

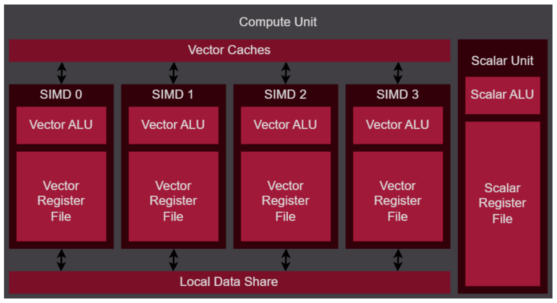

Fig. 2. SIMD Compute Unit of Ryzen AI Engine [4]

and integer computations [7]. This approach ensures compatibility with modern embedded systems prioritizing energy efficiency without sacrificing computational versatility.

C. RISC V Vector ISA

The RISC-V Vector Extension (RVV) offers a flexible vector processing model tailored for data-parallel workloads, optimizing performance without rigid architectural constraints. Unlike fixed-width SIMD designs, RVV uses vector registers with a hardware-defined maximum length (VLEN) [12], enabling scalability across systems—from embedded devices to high-performance computing. Runtime configuration via the vsetvli instruction dynamically sets the active Vector Length (VL), adapting to application demands. Key parameters include SEW (element bit-width), LMUL (register grouping multiplier), and ELEN (hardware's maximum supported element size) [12], which collectively optimize resource usage. AVL (application-specified workload size) further guides vector operations. This adaptability supports diverse applications—neural networks, graphics processing, scientific simulations—while ensuring software remains portable across RVV implementations. By balancing configurability and efficiency, RVV eliminates reliance on fixed data widths, enabling tailored vectorization for varying computational needs.

The rest of the paper is organized as follows: Section II presents the related work in the field of RISC-V vector architectures. Section III outlines the methodology and design approach for the proposed Vector ALU. Section IV discusses the results and performance analysis of the designed architecture. Finally, the paper is concluded in Section V, followed by acknowledgments in Section VI.

II. Related Work

Recent studies have explored enhancements to RISC-V vector architectures aimed at improving data-level parallelism in embedded systems. " RISC-V2 " [2] introduces a vector processor with register remapping and dynamic register file allocation, reporting an average 2.1× throughput gain. The addition of a hardware reduction tree further enhanced neural network benchmarks, although performance was constrained by irregular memory access patterns. "Vector Processing Unit: A RISC-V based SIMD Coprocessor for Embedded Processing" [4] supports varying vector lengths and execution lanes, enabling design flexibility. While configurations with larger VLEN and more lanes showed better CPI, the design lacked full V-extension support—particularly floating-point operations—and exhibited underutilization during strip-mining loops in certain cases.

ZeroVex [5], another effort, targets scalability with a 256-bit VLEN and 8 lanes, achieving speedups up to 235× over scalar baselines. The design scaled linearly with additional lanes and reported 30× better energy efficiency. However, performance on matrix-heavy workloads was less significant due to an increased proportion of scalar instructions and loop overhead. Overall, while these designs offer configurable and scalable approaches, they each face limitations in instruction coverage, memory access handling, or workload-specific performance—highlighting the need for further architectural refinement in vector processing for embedded use cases.

This work presents a Reconfigurable RISC-V-based Vector ALU (VALU), a power-efficient architecture targeting embedded AI and graphics workloads. Departing from rigid designs like ZeroVex and RISC-V2, the VALU features a parameterized lane structure paired with a dedicated dot-product pipeline to accelerate both floating-point and integer vector operations. Early evaluations highlight gains in area efficiency and power savings, particularly in low-lane configurations, where it outperforms existing vector cores in silicon footprint and energy use. Scalability to higher lane counts preserves these efficiency advantages, demonstrating adaptability across workloads.

The modular design enables dynamic hardware reconfiguration, allowing runtime trade-offs between performance, power, and resource utilization. ASIC synthesis results for matrix operations reveal marked improvements in GFLOPS/mm² (area efficiency) and GFLOPS/W (energy efficiency) compared to scalar and fixed-lane SIMD alternatives. Planned enhancements include integration into a full RISC-V SoC with lane virtualization (sharing hardware across multiple vector threads) and intelligent instruction scheduling. These features position the VALU as a tunable solution for compute-heavy yet power-constrained embedded systems, bridging flexibility with efficiency in a way fixed-core architectures cannot.

III. Methodology

A. Vector ALU Architecture

The proposed Vector Arithmetic Logic Unit (Vector ALU) is a reconfigurable parallel processing unit designed to efficiently handle vectorized computations in RISC-V-based embedded systems. The architecture consists of N parallel ALUs, where N is a configurable parameter, enabling scalability to match varying workload demands. The core components of the design include:

- Parallel ALUs (N Lanes): Each ALU independently processes a subset of vector elements, executing arithmetic and logical operations in parallel.

- Dot Product Unit: A dedicated functional unit computes the 64-bit dot product of input vectors, enhancing performance for machine learning and DSP applications.
- Control Signals: Inputs such as mode selection (4-bit) and carry-in (cin) enable flexible operation modes across execution lanes.
- Data Inputs & Outputs: The ALU accepts two $N \times 32$-bit input vectors (A and B), generating $N \times 64$-bit results alongside an N-bit carry-out signal for chained operations.
- Clock & Reset: Standard clock and reset signals manage synchronization and initialization.

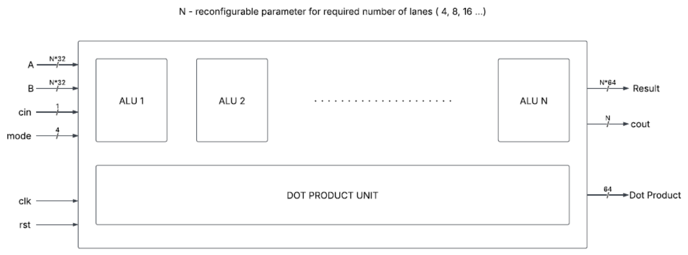

Fig. 3. Proposed Vector ALU block diagram

This architecture balances parallelism and efficiency, allowing dynamic adjustment of ALU lanes to optimize power, area, and performance for various embedded applications.

The ALU supports a rich set of operations essential for compute-intensive embedded AI and graphics workloads. As shown in Table I, it implements both floating-point (FP) and integer arithmetic—specifically, FP multiply/add and integer multiply/add—enabling efficient execution of vector and matrix operations commonly found in deep learning and signal processing. In addition, the ALU includes bitwise logic (AND, OR, XOR) and various shift operations (logical and arithmetic), which are crucial for low-level data manipulation, control flow, and performance optimization. This carefully selected instruction set ensures compatibility with the RISC-V vector ISA and provides the flexibility required for scalable parallel computing across a wide range of embedded applications.

TABLE I
VECTOR ALU OPCODE FUNCTIONALITY

Mode	Operation
4'b0000	FP MULTIPLY
4'b0001	FP ADDITION
4'b0010	BITWISE AND
4'b0011	BITWISE OR
4'b0100	BITWISE XOR
4'b0101	LEFT SHIFT (LSL)
4'b0110	RIGHT SHIFT (LSR)
4'b0111	ARITH SHIFT (ASR)
4'b1000	INTEGER MULT
4'b1001	INTEGER ADD

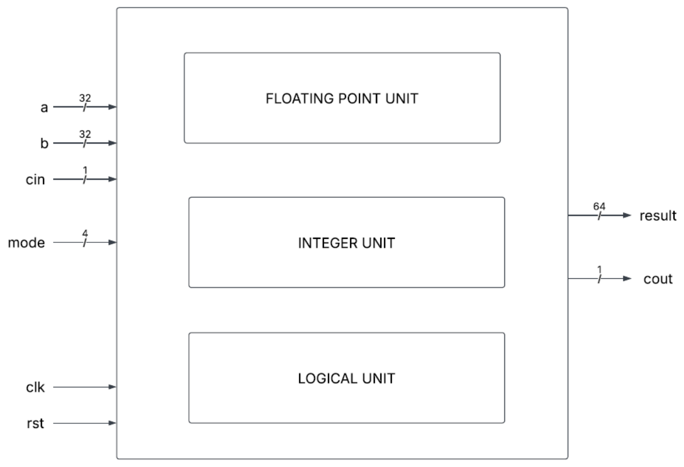

Fig. 4. Proposed ALU block diagram

B. RISC V Vector ISA parameters

The proposed Vector ALU follows the RISC-V Vector Extension (RVV) model, enabling scalable SIMD-style parallel computations. It is designed with a reconfigurable number of lanes ($N = 4, 8, 16, \dots$), optimizing execution efficiency for vectorized workloads.

- Vector Length (VL = N): Defines the number of elements processed per vector operation, dynamically adjustable based on workload demands. 4-lane (128 bits) , 8-lane (256 bits) and so on.
- Vector Register Length (VLEN = N $\times$ 32-bit): Determines the total storage capacity for vector operands, impacting execution throughput.
- Element Width (ELEN = 32-bit): Specifies the bit-width of individual vector elements, supporting integer and floating-point computations.
- Striping Factor (N lanes): Distributes vector elements across multiple ALU lanes, optimizing parallel execution efficiency.

By adhering to RVV principles, this scalable Vector ALU efficiently handles AI, Graphics, and embedded workloads while maintaining power efficiency.

C. Custom RISCV Instruction Format

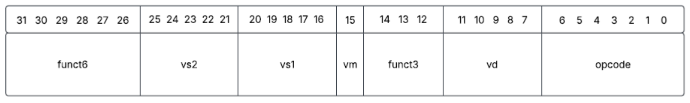

Fig. 5. Proposed Instruction Format

To efficiently support vectorized arithmetic operations in RISC-V-based architectures, we propose a custom R-type vector instruction format. This format aligns with the RISC-V Vector Extension (RVV) and is designed to enable parallel execution while maintaining encoding efficiency.

The proposed instruction consists of 32 bits, structured as follows:

- opcode (7 bits): Identifies the instruction as part of the vector extension.
- vd (5 bits): Destination vector register.
- funct6 (6 bits): Encodes the operation mode (e.g., vector addition, multiplication, floating-point operations). Since only 4 bits are needed, 2 bits are reserved for future use.
- vs1 (5 bits) & vs2 (5 bits): Specify source vector registers.
- vm (1 bit): Mask enable bit for selective execution.
- funct3 (3 bits): Specifies additional control functionalities.

This format ensures encoding efficiency while enabling dynamic vector length execution. The inclusion of reserved bits allows for future architectural extensions.

D. Implementation

The RTL code for the ALU, Dot Product Unit, and VALU was developed using Verilog HDL, incorporating the design methods outlined earlier. The VALU architecture was synthesized at 14nm using Synopsys Design Compiler (DC) [16], with an emphasis on optimizing the balance between power, area, and delay. To ensure power-efficient operation, the design incorporated clock gating directly in the architecture, disabling inactive computational elements, while operand gating minimized unnecessary switching activity. These techniques effectively reduced dynamic power consumption without compromising throughput. The modular nature of the RTL implementation allowed scalability across various lane configurations, supporting single-lane, 4-lane, 8-lane, and 16-lane setups.

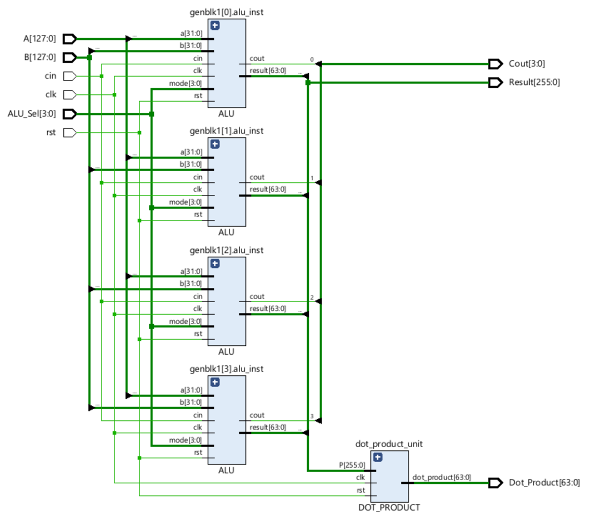

Fig. 6. Schematic of 4-lane Vector ALU in Xilinx Vivado

A Simple Vector x Vector FP multiplication and 4×4 matrix multiplication simulations were also conducted to validate the design's vectorized execution flow, ensuring proper data movement. This step verified the correct implementation of parallel computation strategies, ensuring that the VALU meets the requirements for compute intensive workloads.

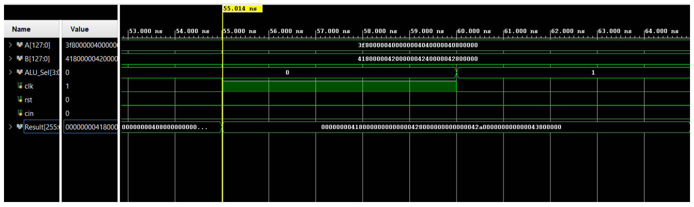

Fig. 7. Vector FP Multiplication Simulation in Xilinx Vivado

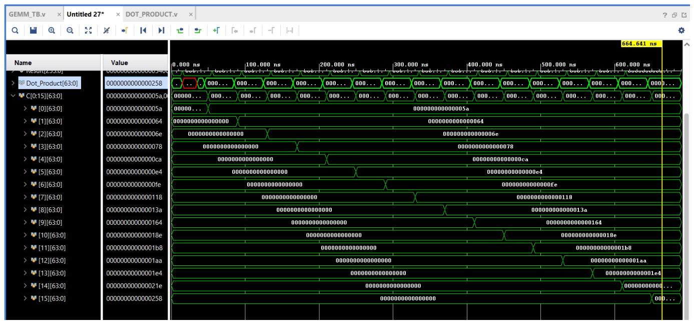

Fig. 8. 4x4 Matrix Multiplication Simulation in Xilinx Vivado

IV. Results & Discussion

Figure 6 illustrates the schematic of the 4-lane Vector ALU, which aligns with the proposed architectural block diagram. In this configuration, the output width is 256 bits, scaling up to 512 bits and 1024 bits in the 8-lane and 16-lane variants, respectively. Figure 7 demonstrates the functional simulation of a 4×4 matrix multiplication, wherein the Vector ALU performs parallel row–column vector multiplications. For a 4×4 matrix, this results in 16 vector-by-vector operations.

Table II presents a comparative analysis of area and power consumption for the proposed design against prior works [2] and [5], across different SIMD configurations (Scalar, 4-Lane, 8-Lane, and 16-Lane). The proposed design consistently demonstrates superior efficiency. For instance, in the scalar configuration, it achieves an area of only 0.08 mm², which is a 66.7% and 46.7% reduction compared to [2] and [5], respectively. Similarly, the power consumption is significantly lower at 0.77 mW, which is 94.1% and 92.8% less than [2] and [5], respectively. In the 4-Lane configuration, the proposed design reduces area and power to 0.39 mm² and 3.55 mW, outperforming [2] (0.61 mm², 43.9 mW) and [5] (0.65 mm², 41.59 mW) by considerable margins.

The trend continues in the 8-Lane and 16-Lane configurations, where the proposed design achieves competitive area footprints (0.90 mm² and 1.96 mm²) and remarkable power savings (7.81 mW and 17.0 mW, respectively). Notably, in the 16-Lane case, the power consumption is reduced by 86.3% compared to [2].

Table III summarizes the performance of the proposed Vector ALU across 4, 8, and 16-lane configurations using 14nm technology. As lane width scales, area and power increase proportionally—from 38k μm^2 and 182 μW in the 4-lane

TABLE II
DESIGN COMPARISON AND IMPROVEMENTS (SCALED UP TO 45NM)

Design	Metric	Scalar	4-Lane	8-Lane	16-Lane
[2]	Area (mm²)	0.24	0.61	0.97	1.67
	Power (mW)	13.1	43.9	75.3	124
[5]	Area (mm²)	0.15	0.65	1.13	–
	Power (mW)	10.68	41.59	71.22	–
Proposed	Area (mm²)	**0.08**	**0.39**	**0.90**	**1.96**
	Power (mW)	**0.77**	**3.55**	**7.81**	**17.0**

TABLE III
VECTOR ALU PERFORMANCE ACROSS SCALABLE CONFIGURATIONS (14NM)

Parameter	**4-Lane**	**8-Lane**	**16-Lane**
Area (μm^2)	38037	87485	190185
Power (μW)	182	400	874
Datapath Delay (ns)	2.29	2.40	2.52
Clock Frequency (MHz)	437	417	397
Throughput (Gbps)	111.8	213.3	406.3
Peak Performance (GFLOPS)	1.746	3.258	6.168

to 190k μm^2 and 874 μW in the 16-lane variant. Datapath delay increases marginally (2.29 ns to 2.52 ns), but the design sustains high-frequency operation and throughput, achieving up to 6.168 GFLOPS and 406.3 Gbps.

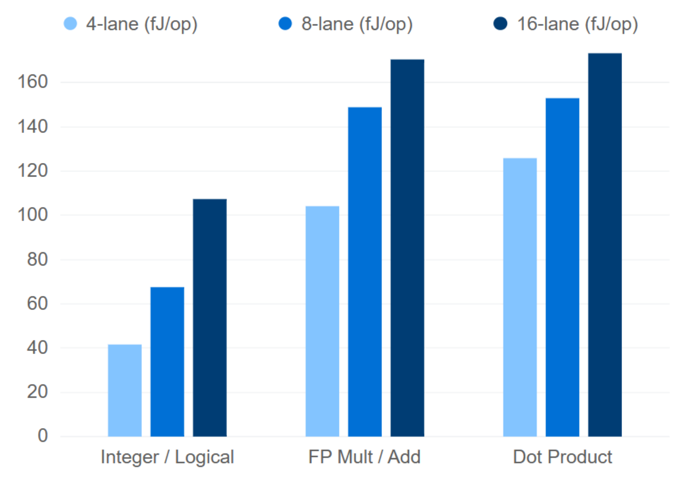

Fig. 9. EPC Comparison Across Operations and Lane Configurations

Figure 8 illustrates the Energy Per Computation (EPC) for various operation types—integer/logical, floating-point multiply/add, and dot product—across 4-, 8-, and 16-lane VALU configurations. The 4-lane architecture achieves the most energy-efficient operation, registering just 41.6 fJ/op for integer/logical operations. In contrast, the EPC for the same operation increases to 67.5 fJ/op and 107.3 fJ/op in the 8-lane and 16-lane variants, respectively. A similar trend is observed for floating-point operations, where EPC rises from 104.1 fJ/op in the 4-lane case to 148.8 fJ/op and 170.4 fJ/op in higher-lane implementations. For the dot product operation, which involves both arithmetic and accumulation, the EPC ranges from 125.8 fJ/op (4-lane) to 152.9 fJ/op (8-lane) and 173.2 fJ/op (16-lane). The observed increase in energy per operation with lane scaling is primarily attributed to interconnect complexity and switching overhead. Nonetheless, the design demonstrates effective energy scaling, retaining competitive EPC figures across all workloads, which underscores its suitability for energy-constrained embedded AI and graphics applications.

V. Conclusion

This work presents a scalable, reconfigurable Vector ALU tailored for energy-efficient embedded AI and graphics applications. Designed around the RISC-V Vector Extension and implemented in 14nm technology, the proposed architecture delivers high performance with minimal power and area overhead. It demonstrates excellent scalability and energy efficiency across 4, 8, and 16-lane configurations. Compared to prior works [2] and [5], it consistently achieves significant reductions in area (up to 66.7%) and power (up to 94.1%), while sustaining high performance (up to 6.168 GFLOPS and 406.3 Gbps). These results validate its suitability for energy-constrained embedded AI and graphics applications.

In summary, the proposed VALU offers a flexible and power-aware vector processing solution, making it a compelling candidate for future RISC-V-based embedded AI and graphics processing systems.

VI. Acknowledgments

This work was supported by the C2S program, providing access to Synopsys Design Compiler tools via CDAC Bangalore and MeitY, which were instrumental for design synthesis and optimization. The author also thanks the faculty and staff of NIT Andhra Pradesh for their guidance and support.

References

[1] M. Gautschi et al., ”Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2700-2713, Oct. 2017, doi: 10.1109/TVLSI.2017.2654506.

[2] K. Patsidis, C. Nicopoulos, G. C. Sirakoulis and G. Dimitrakopoulos, ”RISC-V2: A Scalable RISC-V Vector Processor,” 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9181071.

[3] M. Johns and T. J. Kazmierski, ”A Minimal RISC-V Vector Processor for Embedded Systems,” 2020 Forum for Specification and Design Languages (FDL), Kiel, Germany, 2020, pp. 1-4, doi: 10.1109/FDL50818.2020.9232940.

[4] M. Ali, M. von Ameln and D. Goehringer, ”Vector Processing Unit: A RISC-V based SIMD Co-processor for Embedded Processing,” 2021 24th Euromicro Conference on Digital System Design (DSD), Palermo, Italy, 2021, pp. 30-34, doi: 10.1109/DSD53832.2021.00014.

[5] T. Zhao and Z. Ye, ”ZeroVex: A Scalable and High-performance RISC-V Vector Processor Core for Embedded Systems,” 2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP), Hong Kong, Hong Kong, 2024, pp. 32-33, doi: 10.1109/ASAP61560.2024.00018.

[6] Z. Wang et al., ”GRS: A General RISC-V SIMD Vector Acceleration Processor for Artificial Intelligence Applications,” 2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Taipei, Taiwan, 2024, pp. 702-706, doi: 10.1109/APCCAS62602.2024.10808289

[7] . Chen et al., ”A Multiple Precision Floating-Point Arithmetic Unit Based on the RISC-V Instruction Set,” 2024 4th International Conference on Electronic Information Engineering and Computer (EIECT), Shenzhen, China, 2024, pp. 573-578, doi: 10.1109/EIECT64462.2024.10867213.

[8] G. Zeng and M. Huang, ”Design and Demonstration of a SIMD System Based on the Bit-serial PE Array in FPGA,” 2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE), Xiamen, China, 2019, pp. 1801-1804, doi: 10.1109/EITCE47263.2019.9095060.

[9] Z. Ebrahimi and A. Kumar, ”GREEN: An Approximate SIMD/MIMD CGRA for Energy-Efficient Processing at the Edge,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 10, pp. 2874-2887, Oct. 2024, doi: 10.1109/TCAD.2024.3383349.

[10] Y. Wu and J. McAllister, ”Architectural Synthesis of Multi-SIMD Dataflow Accelerators for FPGA,” in IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 1, pp. 43-55, 1 Jan. 2018, doi: 10.1109/TPDS.2017.2746081.

[11] AMD ROCm Documentation – HIP Hardware Implementation: https://rocm.docs.amd.com/projects/HIP/en/docs-6.2.4/understand/hardware_implementation.html

[12] RISC-V Vector Extension (VEXT-bcn-v1) Presentation: https://riscv.org/wp-content/uploads/2024/12/15.20-15.55-18.05.06.VEXT-bcn-v1.pdf

[13] Introduction to the RISC-V Vector Extension: https://eupilot.eu/wp-content/uploads/2023/03/Roger-Ferrer-ACM-2022_RISC-V-VectorExtension_v2.pdf

[14] Semidynamics Vector Unit Technology: https://semidynamics.com/en/technology/vector-unit

[15] Crash course introduction to parallelism: SIMD Parallelism: https://johnnysswlab.com/crash-course-introduction-to-parallelism-simd-parallelism/

[16] S. Gayathri and T. C. Taranath, ”RTL synthesis of case study using design compiler,” 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India, 2017, pp. 1-7, doi: 10.1109/ICEECCOT.2017.8284603.