International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Design and Verification of RISC-V Processor for Signal Processing Applications

Anil Darga [1], Mallikarjun Awwanna Teli[2], Shashank Otihal[3], Shashank K C[4] & M. Govinda Raju[5]

Department of Electronics and Communication Engineering,
RV College of Engineering, Mysore Road, Bengaluru,560059. anildarga.ec21@rvce.edu.in [1],
mallikarjunat.ec21@rvce.edu.in [2], shashankro.ec21@rvce.edu.in [3]
shashankakc.ec21@rvce.edu.in [4] & govindarajum@rvce.edu.in [5]

***Abstract*—Modern digital signal processing (DSP) systems require both high performance and energy efficiency, particularly for image processing. This research outlines the design and verification of a RISC-V processor using the RV32IP instruction set, which supports SIMD (Single Instruction Multiple Data) operations.It introduces support for performing parallel operations on 8, 16, and 32-bit integer data within a single instruction, significantly improving efficiency in data-parallel processing. The main objective is to test and confirm that RV32IP architecture can correctly run SIMD instructions that are useful in digital signal processing (DSP) tasks such as audio and image processing. Performance is assessed across two RISC-V instruction set architectures: the baseline scalar RV32I and the SIMD-enabled RV32IP,to highlight the benefits of parallel processing. The testing includes running programs on simulators, checking results, and comparing performance between RV32I and RV32IP architectures. The results indicate significant improvements in throughput and instruction-level parallelism.**

***Index Terms*—RISC-V, RV32I, Packed extension, DSP, SIMD operations,Instruction Set Architecture.**

I. INTRODUCTION

The increasing prevalence of real-time embedded systems in areas like surveillance, robotics, and IoT-based edge de- vices has led to a growing need for energy-efficient image processing solutions. Tasks such as edge detection, masking, and pattern matching are common in these applications but often place a heavy computational burden on traditional scalar processing units. The RISC-V architecture, known for its open-source, modular, and flexible instruction set, allows for domain-specific extensions tailored to application needs. One such variant, RV32IP, integrates SIMD (Single Instruction, Multiple Data) functionality, offering enhanced performance for parallel image processing workloads.

Single Instruction, Multiple Data (SIMD) refers to a parallel processing approach where a single operation is carried out across multiple data elements at the same time. In a SIMD-enabled processor, there are multiple processing units (or elements), each capable of performing the same task on separate pieces of data simultaneously. This method is highly effective for speeding up a wide range of tasks in signal processing applications.

Packed SIMD deals with multiple data values grouped within a single register. For example, a 32-bit packed SIMD register may contain four 4-bit values. Packed SIMD offers greater flexibility compared to vector parallelism, as it enables operations on various data types within a single instruction. This makes it especially suitable for applications that involve processing multiple types of data at once, such as multimedia tasks and image manipulation. This form of parallelism works on data vectors continuous sequences of elements that are of the same data type. For instance, a vector composed of 32-bit floating-point values might hold 32 such numbers stored in adjacent memory addresses.

The number of data elements that can be processed in parallel depends on the bit-width of the processor's regis- ters—whether general-purpose or dedicated SIMD/vector reg- isters. Some architectures use standard registers for SIMD instructions, while others rely on specialised SIMD hardware. In the case of RISC-V, known for its straightforward and extensible instruction set, SIMD functionality is available through the P-extension using instructions like ADD8 and ADD16.

Design and validation of a RISC-V-based processor tailored for digital signal processing (DSP) tasks involving image data. Utilising the SIMD capabilities of the RV32IP extension leads to notable gains in throughput and instruction-level efficiency when compared to the baseline scalar RV32I. Simple example to show the working of packed SIMD instruction (MUL8 rd, rs1,rs2 ; parallel 8-bit multiplication) is shown in Fig.1

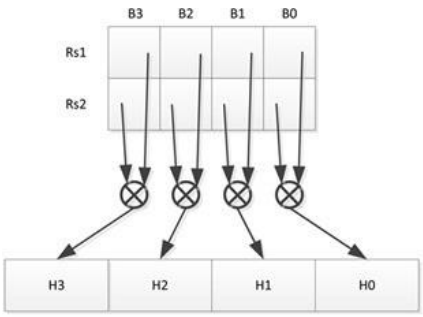


Fig. 1. Packed SIMD MUL8 instruction

II. Literature Review

Waterman et al. introduced the RISC-V instruction set architecture, laying the foundation for modern open-source processor design [1]. Asanovic´ and Patterson emphasized the case for a free, extensible ISA, which enabled rapid innovation in domains like image processing [2]. The official RISC-V "P" extension, designed for packed SIMD operations, offers a lightweight way to accelerate multimedia and DSP workloads [3]. Kothari et al. analyzed the RISC-V Vector Extension (RVV), demonstrating scalability and parallelism for compute-heavy tasks [4]. Hennessy and Patterson provided architectural insights into how SIMD improves data-level parallelism, making it suitable for image kernels [5].

Halder and Sinha implemented the Sobel edge detection filter on FPGA using SIMD, showing measurable acceleration over scalar designs [6]. Wu et al. designed a RISC-V processor optimized for image filtering, incorporating SIMD-style vector instructions for better efficiency [7]. Martin and Brossier developed a low-cost RISC-V SoC for edge image processing, targeting constrained devices like drones and surveillance systems [8]. Lee and Yoo proposed a CMOS-based image feature extraction accelerator optimized for low power embedded systems [9]. Chang et al. benchmarked multiple RISC-V cores on edge AI workloads, including image tasks, and reported improved latency with SIMD support [10].

Gonzalez and Gonzalez proposed a SIMD processor architecture with virtual memory that informed many future parallel designs [11]. Cheung et al. discussed compiling pipelined hardware from behavioral image processing specifications on FPGAs [12]. Jain introduced fundamental algorithms for digital image processing, including edge detection and convolution, foundational for hardware mapping [13]. Gonzalez and Woods extended these algorithms into performance-aware image analysis pipelines [14]. Mangard et al. addressed hardware/software design trade-offs in secure and efficient embedded computation, relevant to trusted edge vision systems [15].

Wei and Zhang proposed a lightweight RISC-V SIMD core specifically for machine vision applications, demonstrating improved edge detection speeds [16]. Tumeo and Villa explored FPGA acceleration for pattern-matching kernels in image/DNA analysis, offering insights into parallelism in specialized domains [17]. Henkel described the SoC design gap and the need for domain-specific accelerators, reinforcing the motivation for RV32IP integration in image pipelines [18]. Waterman's dissertation further examined instruction mix and performance bottlenecks in baseline RISC-V cores [19]. RISC-V International's official SIMD spec (v0.9.7) served as the reference model for implementing and evaluating RV32IP image tasks [20].

Gautier et al. introduced vectorised RISC-V intrinsics to optimise OpenCV kernels, showing significant performance gains on embedded platforms using the RISC-V Vector Extension [21]. Gupta and Perica`s studied the implementation of Winograd convolution on RVV cores, highlighting trade-offs in vector length and register usage for convolutional neural networks [22]. Wang et al. developed SPEED, a scalable vector processor based on RISC-V, which achieved efficient multi-precision ng notable acceleration in image filtering and transformation tasks [24]. Li et al. further expanded SIMD compatibility through the SIMDe framework, automating the translation of ARM NEON code to RISC-V vector intrinsics with measurable improvements in XNNPACK libraries [25].

III. RV32I Base Instruction Set

The RV32I instruction set represents the 32-bit base integer architecture of RISC-V. It offers a streamlined, efficient, and adaptable framework suitable for both general-purpose computing and embedded system applications. RV32I serves as the foundational architecture for executing programs, handling data operations, and managing control structures. RV32I includes:

- Load/store instructions (e.g., LW, SW) for moving data between memory and registers.
- Arithmetic/logical operations (e.g., ADD, SUB, AND, OR, XOR).
- Shift instructions (SLL, SRL, SRA).
- Control flow instructions (BEQ, JAL, JALR) for manag- ing loops and conditional operations in.
- Immediate operations (ADDI, ORI, etc.) for fast constant- based computations.

RV32I serves as the scalar processing baseline. This approach ensures portability and simplicity, but lacks the instruction-level parallelism found in SIMD-enhanced archi- tectures. The performance metrics and throughput results derived from RV32I help establish a reference point for comparing with extended architectures like RV32IP.

IV. RV32IP base instructin set

The RV32IP is a Packed SIMD extension to the standard RV32I RISC-V architecture, designed to enhance performance by enabling parallel operations on subword data within 32-bit registers. It is particularly suitable for image processing tasks, where similar operations are applied across multiple pixels. RV32IP includes:

- Packed arithmetic and logical instructions (e.g., PADD8, PSUB16, PAND8) to operate on multiple smaller-width data (e.g., 4×8-bit or 2×16-bit) in a single instruction, useful for simultaneous p.
- Packed shift and comparison operations (CMPEQ8, CM- PEQ16) that allow for efficient gradient calculations and conditional processing.
- Packed load and store instructions (LB8,LH16,SB8,SH16) optimized for reading/writing grouped pixel values, efficient sub-word memory access in data-parallel applications.
- Support for parallel saturating operations to prevent over- flow during image enhancement or convolution.

RV32IP allows you to process multiple pixels in parallel, increasing throughput and energy efficiency. Compared to scalar RV32I execution, RV32IP significantly reduces the instruction count and improves performance in similar operations.

A. RISC-V Packed-SIMD Instructions

Some of the RISC-V P-extension instructions which are required in signal processing applications are:

Instruction	Description
ADD8	Adds 4 pairs of 8-bit integers in 32-bit registers.
ADD16	Adds 2 pairs of 16-bit integers in 32-bit registers.
AVE	Averages two values (e.g., pixels).
BITREV	Reverses all bits in a register.
BITREVI	Bit reversal using a bitmask.
CMPEQ8	Compares 8-bit values; returns result mask.
CMPEQ16	Compares 16-bit values for equality.
CRAS16	Subtracts upper, adds lower 16-bit halves.
KABS8	Absolute value of 4 packed 8-bit integers.
KABS16	Absolute value of 2 packed 16-bit integers.
KADD8	Saturating add for packed 8-bit values.
KADD16	Saturating add for packed 16-bit values.

TABLE I
DESCRIPTIONS OF SELECTED RV32IP SIMD INSTRUCTIONS

V. DESIGN METHODOLOGY

We have designed a set of assembly-level test programs that target specific architectural features to categorise instruction groups and hardware components typically described in the RISC-V ISA manual. These tests function as synthetic benchmarks, aimed at achieving full utilisation of the functional units under test. With the integration of the RISC-V P extension, the test framework has been extended to focus on packed SIMD operations—including parallel addition, subtraction, multiplication, and division across multiple subword elements within a 32-bit register. As part of the evaluation methodology, instructions supported by both RV32I and RV32IP are executed using dedicated testbenches. Performance is then compared in terms of execution time, allowing a clear analysis of the efficiency gains provided by SIMD-enhanced processing over the baseline scalar architecture.

- Develop a five-stage pipelined datapath for instruction execution based upon RV32I.
- Resolve any hazards by installing stalls or data forward- ing.
- Develope test cases to verify the instructions supported by the CPU.
- Execute selected instructions to test on the core by implementing the RISC-V tool.

VI. 5-STAGE PIPELINED RV32I ARCHITECTURE

A 5-stage pipelined processor enhances instruction throughput by dividing execution into five stages. However, pipeline hazards: data and control hazards can disrupt seamless execution. Data hazards occur when instructions depend on previous results. Control hazards, mainly caused by branches, affect instruction flow and require effective handling strategies. since we are executing only arithematic instructions there will be no control hazards in the processor.

To mitigate data hazards- techniques such as instruction reordering, forwarding, and pipeline stalling are considered within the simulation framework. a complete 5stage pipelined RV32I processor architecture is shown in Fig.2.

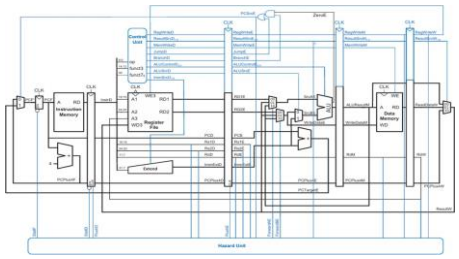


Fig. 2. 5-Stage Pipelined RV32I Processor Architecture

VII. RV32I PROCESSOR WITH PACKED EXTENSION ARCHITECTURE

The RV32IP architecture builds upon the baseline RV32I processor by incorporating the Packed SIMD (Single Instruction, Multiple Data) extension. This enhancement allows the execution of parallel operations on multiple 8-bit, 16-bit, or 32-bit data elements within a single 32-bit register, significantly improving performance for data-parallel workloads common in digital signal processing, multimedia, and embedded applications. the architectural design of RV32I with Packed (P) extension is shown in Fig. 3.

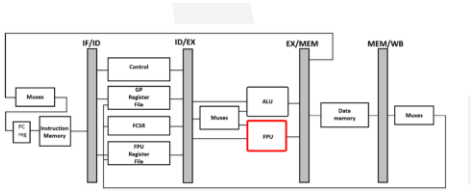


Fig. 3. RV32I with P extension

VIII. VERIFICATION ENVIRONMENT

The verification environment Makefile serves as a pivotal component for managing the verification process, as depicted in Fig. 3 It encapsulates various tasks essential for verifying the functionality and correctness of the design, compilation, simulation, and analysis stages. The Makefile manages the verification process by integrating several essential software tools required for compiling tests and running them on the verilog design. These tools include:

- Icarus Verilog (Iverilog) for Compilation Icarus Verilog (iverilog) is used to compile and simulate Verilog HDL designs. It processes the Verilog source code, linking all modules, and generates an executable simulation file. This compiled design can then be executed to verify the functionality of the RV32IP processor.

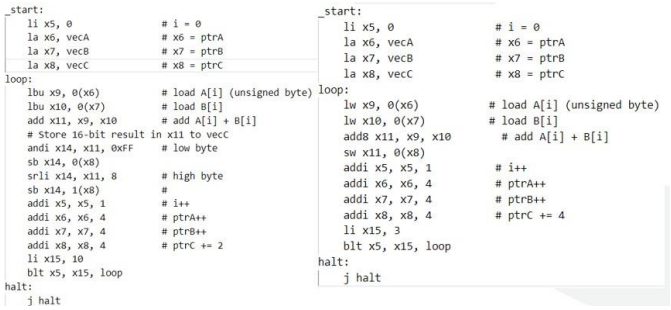
- GTKWave for Waveform Analysis GTKWave is an open- source waveform viewer used to visualise signal transi- tions within the processor. By analysing the waveform, one can verify correct instruction execution, identify hazards, and debug unexpected behaviour in different pipeline stages.

A. Performance Comparision Between RV32I and RV32IP

- RV32I is a scalar instruction set, processing one data element at a time, while RV32IP supports SIMD (Single Instruction, Multiple Data), allowing parallel processing of multiple data elements.
- In signal processing tasks, RV32IP significantly outper- forms RV32I by reducing the number of instructions and execution time.
- Clock cycles and memory accesses are substantially lower in RV32IP.
- This concludes that RV32IP is better suited for compute- intensive, data-parallel applications such as edge detec- tion, audio filtering, and machine learning on embedded platforms.
- RV32IP reduces the instruction count and improves throughput, which is shown in results.

IX. Resuls and Discussions

The verilog codes for arithematic operations such as addi- tion and multiplication are shown in Fig.4 and Fig.5 respec- tively.



```
_start:
    li x5, 0              # i = 0
    la x6, vecA           # x6 = ptrA
    la x7, vecB           # x7 = ptrB
    la x8, vecC           # x8 = ptrC
loop:
    lbu x9, 0(x6)         # load A[i] (unsigned byte)
    lbu x10, 0(x7)        # load B[i]
    add x11, x9, x10      # add A[i] + B[i]
    # Store 16-bit result in x11 to vecC
    andi x14, x11, 0xFF   # low byte
    sb x14, 0(x8)
    srli x14, x11, 8      # high byte
    sb x14, 1(x8)         #
    addi x5, x5, 1        # i++
    addi x6, x6, 4        # ptrA++
    addi x7, x7, 4        # ptrB++
    addi x8, x8, 4        # ptrC += 2
    li x15, 10
    blt x5, x15, loop
halt:
    j halt
```

```
_start:
    li x5, 0              # i = 0
    la x6, vecA           # x6 = ptrA
    la x7, vecB           # x7 = ptrB
    la x8, vecC           # x8 = ptrC
loop:
    lw x9, 0(x6)          # load A[i] (unsigned byte)
    lw x10, 0(x7)         # load B[i]
    add8 x11, x9, x10       # add A[i] + B[i]
    sw x11, 0(x8)
    addi x5, x5, 1        # i++
    addi x6, x6, 4        # ptrA++
    addi x7, x7, 4        # ptrB++
    addi x8, x8, 4        # ptrC += 4
    li x15, 3
    blt x5, x15, loop
halt:
    j halt
```

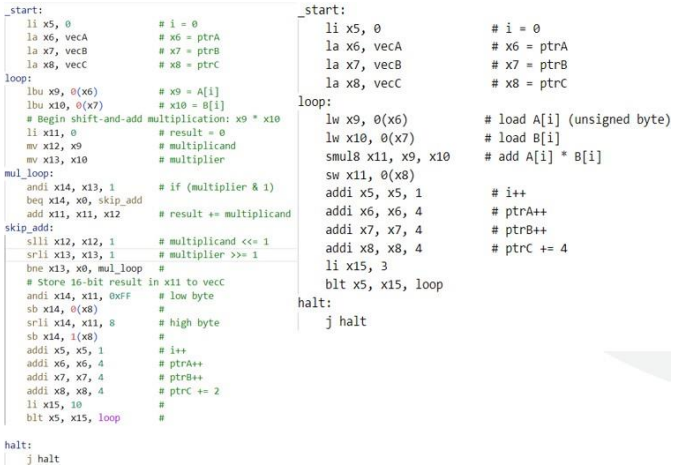
Fig. 4. Addition in RV32I (left) and RV32IP (right)

Processor Type	Execution Time	Cycles
RV32I	1.6 micro Sec	160
RV32IP	0.4 micro sec	40

TABLE II
EXECUTION TIME OF ADDITION

Processor Type	Execution Time	Cycles
RV32I	3.9 micro Sec	390
RV32IP	0.4 micro sec	40

TABLE III
EXECUTION TIME OF MULTIPLICATION



```
_start:
    li x5, 0              # i = 0
    la x6, vecA           # x6 = ptrA
    la x7, vecB           # x7 = ptrB
    la x8, vecC           # x8 = ptrC
loop:
    lbu x9, 0(x6)         # x9 = A[i]
    lbu x10, 0(x7)        # x10 = B[i]
    # Begin shift-and-add multiplication: x9 * x10
    li x11, 0             # result = 0
    mv x12, x9            # multiplicand
    mv x13, x10           # multiplier
mul_loop:
    andi x14, x13, 1      # if (multiplier & 1)
    beq x14, x0, skip_add
    add x11, x11, x12     # result += multiplicand
skip_add:
    slli x12, x12, 1      # multiplicand <<= 1
    srli x13, x13, 1      # multiplier >>= 1
    bne x13, x0, mul_loop #
    # Store 16-bit result in x11 to vecC
    andi x14, x11, 0xFF   # low byte
    sb x14, 0(x8)         #
    srli x14, x11, 8      # high byte
    sb x14, 1(x8)         #
    addi x5, x5, 1        # i++
    addi x6, x6, 4        # ptrA++
    addi x7, x7, 4        # ptrB++
    addi x8, x8, 4        # ptrC += 2
    li x15, 10            #
    blt x5, x15, loop     #

halt:
    j halt
```

```
_start:
    li x5, 0              # i = 0
    la x6, vecA           # x6 = ptrA
    la x7, vecB           # x7 = ptrB
    la x8, vecC           # x8 = ptrC
loop:
    lw x9, 0(x6)          # load A[i] (unsigned byte)
    lw x10, 0(x7)         # load B[i]
    smul8 x11, x9, x10    # add A[i] * B[i]
    sw x11, 0(x8)
    addi x5, x5, 1        # i++
    addi x6, x6, 4        # ptrA++
    addi x7, x7, 4        # ptrB++
    addi x8, x8, 4        # ptrC += 4
    li x15, 3
    blt x5, x15, loop
halt:
    j halt
```

Fig. 5. Multiplication in RV32I (left) and RV32IP (right)

The executed arithmatic operations shows the execution time of RV32IP is less compared to RV32I. This concludes that RV32IP reduces the instruction count and improves throughput and RV32IP is better suited for computentensive, data-parallel applications such as audio filtering, and machine learning on embedded platforms.

Similar to other ISAs, RISC-V includes a packed SIMD extension known as the P-extension, which supports parallel operations on 8-bit, 16-bit, and 32-bit integers within a 32-bit register. Although still in the draft stage (version 0.9.5 at the time of writing) and not yet officially ratified, the P-extension is being explored due to its potential performance advantages. Current hardware implementations are limited, and software support is still evolving, but the extension holds promise for data-parallel tasks and is gaining interest in the RISC-V community.

X. Conclusion

The design and assessment of the RISC-V P extension define a packed Single Instruction, Multiple Data (SIMD) instruction set specifically developed for RISC-V processors. This extension brings in capabilities for handling 8-bit, 16-bit, and 32-bit integer data types, effectively enhancing the RISC- V instruction set architecture. With this upgrade, developers are enabled to build more efficient and powerful data-parallel applications. The importance of the P extension is highlighted by its ability to accelerate a broad range of workloads, including graphics rendering, audio computation, and scientific data processing.

The comparative analysis between the baseline RV32I and the SIMD-enhanced RV32IP clearly demonstrates the perfor- mance benefits of parallel processing. Through software sim- ulation and result validation, the study confirms that RV32IP offers noticeable improvements in throughput and instruction-level parallelism. These findings reinforce the effectiveness of SIMD integration in the RISC-V architecture for acceler- ating data-parallel workloads, especially in signal processing domains.

ACKNOWLEDGMENT

We, Anil Darga, Mallikarjun Awwanna Teli, Shashank Otihal and Shashank K C would like to thank our guide Dr. M Govinda Raju for valuable insights and continous support throughout this research. we also acknoweledge the Department of Electronics and Communication Engineering, RV College of Engineering, for providing the necessary resources and infrastructure.

REFERENCES

[1] A. Waterman and K. Asanovic´, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-157, 2017.

[2] RISC-V International, "RISC-V 'P' Extension: Packed SIMD/DSP Ex- tension, Version 0.9.7," 2021.

[3] S. Kothari, M. Schaffner, and L. Benini, "Design and Evaluation of SmallFloat SIMD Extensions to the RISC-V ISA," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019, pp. 654-657.

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quanti- tative Approach*, 5th ed., Morgan Kaufmann, 2011.

[5] N. Nausheen, A. Seal, P. Khanna, and S. Halder, "A FPGA Based Imple- mentation of Sobel Edge Detection," *Microprocessors and Microsystems*, vol. 56, pp. 84–91, 2018.

[6] J. Schiel and A. Bainbridge-Smith, "Efficient Edge Detection on Low- Cost FPGAs," *arXiv preprint arXiv:1512.00504*, 2015.

[7] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: A 1 GHz+ Scalable and Energy-Efficient RISC-V Vector Processor with Multi-Precision Floating Point Support in 22 nm FD-SOI," *arXiv preprint arXiv:1906.00478*, 2019.

[8] D. Rossi et al., "Vega: A 10-Core SoC for IoT End-Nodes with DNN Acceleration and Cognitive Wake-Up From MRAM-Based State- Retentive Sleep Mode," *arXiv preprint arXiv:2110.09101*, 2021.

[9] J. K. L. Lee, M. Jamieson, N. Brown, and R. Jesus, "Test-driving RISC- V Vector Hardware for HPC," *arXiv preprint arXiv:2304.10319*, 2023.

[10] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "Design and Evaluation of SmallFloat SIMD Extensions to the RISC-V ISA," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019, pp. 654-657.

[11] S. Abed, "Implementation of an Edge Detection Algorithm Using FPGA Reconfigurable Hardware," *Journal of Engineering Research*, vol. 8, no. 1, pp. 179–197, 2020.

[12] A. Bettaieb, N. Filali, T. Filali, and H. B. Aissia, "GPU Acceleration of Edge Detection Algorithm Based on Local Variance and Integral Image: Application to Air Bubbles Boundaries Extraction," *Computer Optics*, vol. 43, no. 3, pp. 446–454, 2019.

[13] B. Ja¨hne, *Digital Image Processing: Concepts, Algorithms, and Scientific Applications*, Springer, 1991.

[14] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.

[15] D. Marr and E. Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London. Series B, Biological Sciences*, vol. 207, no. 1167, pp. 187–217, 1980.

[16] RISC-V International, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20190608-Priv-MSU-Ratified," 2019.

[17] S. Halder and A. Sinha, "High Performance and Energy Efficient Sobel Edge Detection," *Microprocessors and Microsystems*, vol. 56, pp. 84–91, 2018.

[18] M. Schaffner, F. Zaruba, and L. Benini, "ARA: A 1 GHz+ Scalable and Energy-Efficient RISC-V Vector Processor," in *Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[19] RISC-V International, "RISC-V 'P' Extension: Packed SIMD/DSP Ex- tension, Version 0.9.7," 2021.

[20] M. Zgheib, O. Potin, P. Rigaud, and D. Dutertre, "Extending a RISC- V Core with an AES Hardware Accelerator to Meet IoT Constraints," in *Proc. IEEE Latin American Symposium on Circuits and Systems (LASCAS)*, Arequipa, Peru, 2021, pp. 1-4.

[21] "RISC-V Packed SIMD Extension," Accessed: Oct. 16, 2022. [Online]. Available: https://github.com/riscv/riscv-p-spec.

[22] L. Gautier, R. Mangeol, F. Pe´trot, and T. Moreau, "Accelerating OpenCV Kernels with RISC-V Vector Intrinsics," Proc. of the International Workshop on RISC-V Research Activities, 2023.

[23] S. R. Gupta and M. Perica`s, "Challenges and Opportunities in the Co-design of Convolutions and RISC-V Vector Processors," arXiv preprint, arXiv:2311.05284, Nov. 2023. [Online]. Available: https://arxiv.org/abs/2311.05284.

[24] C. Wang, C. Fang, X. Wu, et al., "SPEED: A Scalable RISC-V Vector Processor Enabling Efficient Multi-Precision DNN Inference," arXiv preprint, arXiv:2401.16872, Jan. 2024. [Online]. Available: https://arxiv.org/abs/2401.16872.

[25] D. Volokitin, E. P. Vasiliev, E. A. Kozinov, et al., "Im- proved Vectorization of OpenCV Algorithms for RISC-V CPUs," arXiv preprint, arXiv:2311.12808, Sept. 2023. [Online]. Available: https://arxiv.org/abs/2311.12808,