Regular Article

SHA-RV: A RISC-V Accelerator for SHA-224/256 with Cycle-Reduced ISA Extensions for Blockchain Applications

Pham Tuan Dat[1], Luu Van Tuan[1], Tran Thi Diem[2], Dao Hoang Nam[3], Le Vu Trung Duong[5], Nguyen Van Tinh[1], Pham Thi Mai[1], Pham Xuan Nghia[1]

[1] Le Quy Don Technical University, Hanoi, Vietnam
[2] University of Information Technology, Vietnam National University, Ho Chi Minh City, Vietnam
[3] High-Tech Telecommunication Center, Ha Noi, Vietnam
[4] Foreign Trade University
[5] Nara Institute of Science and Technology, Nara, Japan

Correspondence: Nguyen Van Tinh, take@lqdtu.edu.vn
Communication: received 01 November 2025, revised 04 December 2025, accepted 17 December 2025
Online publication: 25 December 2025, Digital Object Identifier: 10.21553/rev-jec.423

Abstract– **The Secure Hash Algorithm SHA-256 and SHA-224 are widely used for software integrity, digital signatures, and blockchain across embedded and edge platforms. Prior RISC-V accelerators still struggle to achieve low cycle counts and high system throughput on long message streams. This paper proposes a hardware-efficient RISC-V accelerator with low-latency SHA instruction extensions, named SHA-RV, to reduce cycles and improve end-to-end performance. SHA-RV integrates three optimizations: a high-bandwidth BufferSet for continuous data supply, a four-stage pipelined SHA core, a system-level double-buffering pipeline, and an FSM-orchestrated BufferSet mapping. Implemented on a Xilinx ZCU102 system on a chip, SHA-RV operates at up to 300 MHz and uses 3,146 flip-flops, 5,175 lookup tables, and 15 block RAMs. On 64-byte blocks, SHA-RV completes a block in 257 cycles, improving over related RISC-V designs by between 9.7 and 134.9 times, while reducing logic resources versus the ISOCC 2024 design by 89.4 percent in flip-flops and 85.2 percent in lookup tables. At the system level, SHA-RV achieves a throughput of 599 megabits per second and an energy efficiency of 798.7 megabits per second per watt under a real-time dynamic power under a measured average PS+PL dynamic power of 0.75 W (INA226-based on-board measurement), outperforming representative CPUs by between 61 and 454 times in energy efficiency. These results show lower latency and superior hardware efficiency relative to prior work.**

Keywords– **RISC-V, SHA-224/256, custom ISA extensions, pipelined hardware accelerator, blockchain applications.**

1 Introduction

RISC-V, introduced by the Berkeley research group in the late 2010s, is an open-source CPU architecture designed to promote flexibility and innovation [1–3]. Its open Instruction Set Architecture (ISA) allows developers to design custom processors without licensing fees, facilitating advancements in specialized hardware. With high compatibility across various applications and superior energy efficiency, RISC-V has become an ideal choice for resource-constrained devices [4–6]. Specifically, this architecture accelerates cryptographic algorithms on IoT devices, supporting the efficient implementation of security standards such as AES, SM4, and SHA-256 [7, 8]. Although RISC-V offers numerous advantages, its ISA is continuously evolving to meet new demands in fields like artificial intelligence and cloud computing.

The Secure Hash Algorithm (SHA-256), standardized by the National Institute of Standards and Technology (NIST) [9–12], is a pillar of modern digital security. Its application has expanded to cost-sensitive systems such as IoT devices, edge servers, and personal computers, where it is used to ensure the data integrity of software updates, digital signatures, and blockchain transactions [13–15]. These platforms require SHA-256 processing solutions that are both energy-efficient to preserve battery life and high-performance to meet increasingly stringent security requirements [16–18]. The flexible integration capabilities of RISC-V allow for the design of specialized SHA-256 accelerators, enhancing performance and reducing hardware complexity [19].

Many previous studies have developed SHA-256 hardware platforms with high speed and low power for IoT security domains. The platform proposed in [20] has significantly improved speed compared to software. However, its latency is still very high because of the lack of dedicated hardware with ISA extensions. Other works in [7, 8] have proposed RISC-V instruction extensions for cryptographic functions to achieve latency reduction. However, their latency and hardware resources are still high because of integrating many dedicated cores to support multiple algorithms. Overall, current SHA-256 implementations on the RISC-V platform are still not able to provide high power efficiency to adapt to SHA applications [21, 22].

To address these challenges, this paper proposes SHA-RV, a RISC-V architecture for accelerating SHA-224/256 with significantly reduced cycle counts. The main contributions are (1) High-Bandwidth Buffer-

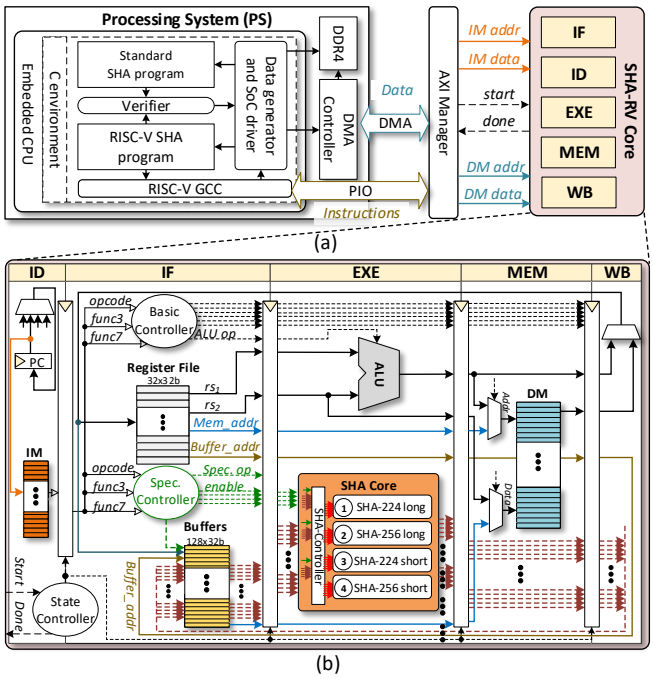

Figure 1. System overview and SHA-RV core datapath. (a) SoC integration with PS and coprocessor interfaces. (b) Internal pipeline with BufferSet and four-stage SHA core.

Set for Continuous Processing, (2) Four-Stage Pipelined SHA-RV Core, (3) System-Level Double-Buffering Pipeline, and (4) Efficient FSM for BufferSet mapping. The design is implemented on the ZCU102 FPGA and demonstrates lower latency and better hardware efficiency than prior work.

The remainder of the paper is organized as follows: Section II presents the SHA-RV architecture, Section III reports implementation and evaluation results, and Section IV concludes the paper.

2 Proposed SHA-RV

2.1 System Architecture Overview

The overall architecture of SHA-RV on an SoC platform, as illustrated in Figure 1(a), consists of two main components: the Processing System (PS) and the SHA-RV Core. The PS, typically controlled by an embedded CPU, executes the SHA application, RISC-V host program, verifier, data generator, and SoC driver. When a hash operation is required, the application prepares the input data and dispatches it to the SHA-RV module via a standard bus interface such as AXI. To optimize data movement, bulk payloads are transferred using Direct Memory Access (DMA), while control/status transactions are handled through peripheral I/O (PIO).

The SHA-RV Core is implemented as an independent co-processor. It exchanges control/status registers and computation data with the PS via AXI manager/supervisor interconnect. As shown in Figure 1(b), SHA-RV employs five pipeline stages: instruction fetch (IF), instruction decode (ID), execute (EXE), memory access (MEM), and write-back (WB).

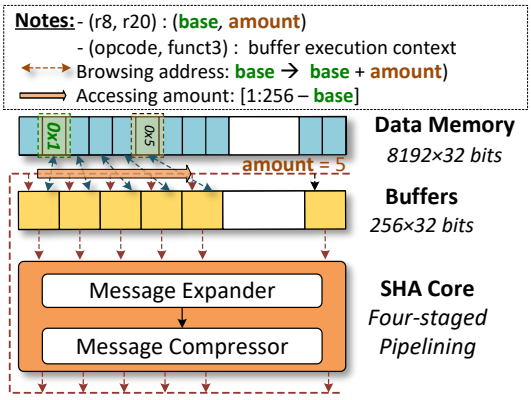

Figure 2. Buffer management and bulk transfers between DMEM and the 256×32-bit BufferSet.

Buffer Accessing Custom Instructions

funct7	rs2	rs1	funct3	rd	opcode
31 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0

opcode	funct3	Function
0101011	000	Latch base and amount
	001	Enable buffer write flag
	010	Enable buffer read flag

SHA Custom Instructions

funct7	rs2	rs1	funct3	rd	opcode
31 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0

opcode	func3	Function
00 010 11	000	SHA-224 short
00 010 11	001	SHA-256 short
00 010 11	010	SHA-224 long
00 010 11	011	SHA-256 long

Figure 3. Custom instructions for BufferSet access and SHA-224/256 invocation on RISC-V.

Specifically, the EXE stage integrates an internal BufferSet and a four-stage pipelined SHA Core to accelerate SHA-224/256 under custom instructions. Under PS control, the SHA-RV Core achieves high efficiency through high-bandwidth DMA transfers on a low-power SoC platform. The detailed architecture and operation of the BufferSet and SHA Core are described in the following sections.

2.2 Internal BufferSet for Continuous Processing

The use of the general-purpose ALU in a conventional RISC-V core, following standard resource-usage rules, requires many instructions just to marshal the data needed for SHA's primary processing. As a result, SHA-256/224 performance on a baseline RISC-V core is low and fails to meet current security/latency targets. Therefore, the proposed SHA-RV improves throughput by using an SHA Core that integrates specialized components optimized for SHA-224/256. The SHA core requires sustained data availability across the compression loops. To ensure high computational efficiency, a 256×32-bit (4096 bits total) FF-based BufferSet is placed in front of the SHU, holding critical data: the K constants (64×32-bit), the previous digest H (8×32-bit), and the message block M (16×32-bit). Unlike traditional BRAM, this BufferSet is designed for true parallel access: each 32-bit lane has its own dedicated read/write path. This architecture lets the SHU load or store the

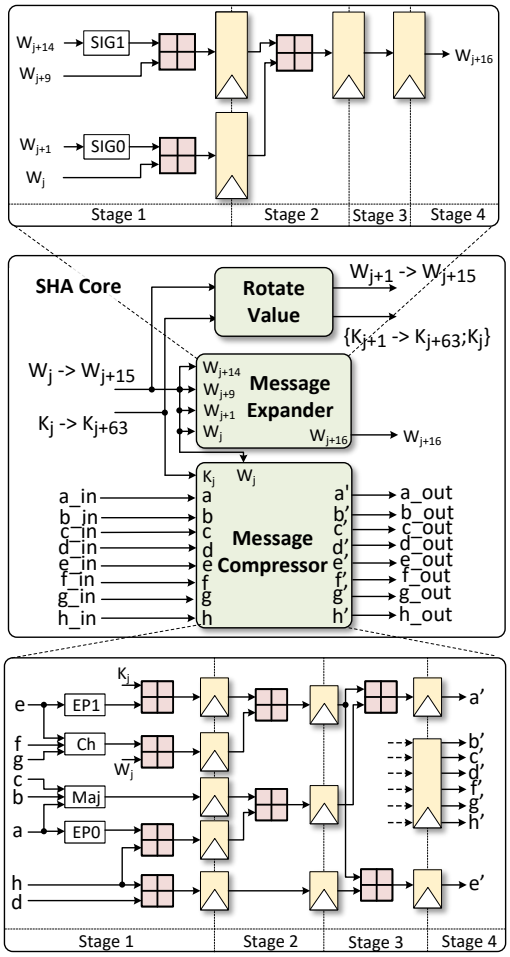

Figure 4. Four-stage SHA Core hardware architecture.

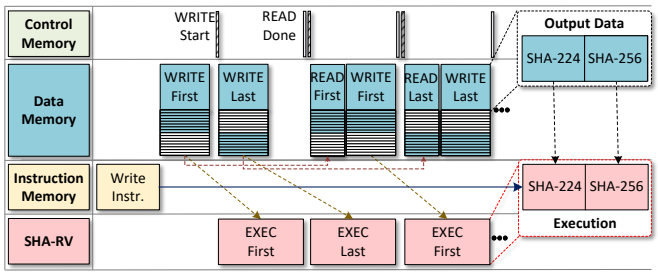

Figure 5. System-level pipelining with double-buffering scheduling to overlap WRITE/EXEC/READ.

entire 4096-bit window in a single cycle, bypassing shared-bus limits and providing the extreme parallelism required for continuous, high-speed hashing.

Figure 2 shows the detailed communication mechanism between the high-bandwidth BufferSet and data memory. In this scheme, a burst of words is transferred between data memory (DMEM) and the buffer under the control of registers `r8` and `r20`. Register `r8` holds the DMEM base address for read/write operations, while `r20` specifies the number of 32-bit words to transfer. This process is executed by the buffer-access instructions in Figure 3: one instruction configures the address and length, and another sets a start flag to trigger the transfer. Once the BufferSet is filled, the SHA core immediately performs the message-schedule expansion and compression rounds. To initiate hashing, the SHA-RV uses a set of **custom SHA instructions** that call the SHA core in four modes: SHA-224 short, SHA-256 short, SHA-224 long, and SHA-256 long. These instructions enable the core to select the corresponding configuration and number of rounds automatically, ensuring optimal throughput for both short and long message blocks. Using the high-bandwidth BufferSet in combination with these custom instructions eliminates redundant data movement and instruction overhead present in the standard RISC-V ISA, resulting in significantly higher hashing efficiency.

2.3 Four-Stage pipelined SHA Core

As shown in Figure 4, the proposed SHA-RV core employs a four-stage pipelined SHA-224/256 architecture to improve throughput while maintaining hardware efficiency. The design consists of three main modules: the Message Expander (ME), the Message Compressor (MC), and the Value Rotator (VR). The ME module expands 16 input message words into 64 message words, which are subsequently processed in 64 main hashing rounds by the MC module. After each computation, the VR rotates the active message words and the 64 constant K values before storing them back into the internal buffer for the next iteration.

To ensure continuous operation, the internal BufferSet stores at least four input cases that are concurrently fetched by the pipelined SHA core. This configuration allows efficient utilization of all four pipeline stages, minimizing idle cycles during message scheduling and compression. The buffer capacity supports up to 11 input cases for long-message processing and up to 8 cases for short messages, ensuring optimal system throughput across different message lengths.

The four pipeline stages are equally divided between the ME and MC modules, with registers inserted between stages to balance the critical path. Each stage contains no more than one adder, keeping the delay comparable to the baseline RISC-V ALU. This pipelined architecture enables the SHA core to complete one round per cycle after the pipeline is fully filled. The overall latency of the four-stage pipeline can be expressed as

$$L_{\text{SHA}} = 64 + (N_{\text{in}} - 1), \tag{1}$$

where 64 represents the 64 hashing rounds for SHA-224/256, and N_{in} denotes the number of input cases concurrently processed in the pipeline. With $N_{\text{in}} = 4$, the system achieves the highest steady-state throughput while maintaining balanced timing and resource utilization.

2.4 System-Level Pipeline with Double Buffering Mechanism

Transferring large blocks of input and output data to the SHA-RV core through the AXI interface introduces considerable latency. In a conventional SoC design, the waiting time for data reads and writes may exceed the computation time of the SHA-224/256 core, creating a system bottleneck. In such a case, simply

Algorithm 1 System-Level Pipelined Execution with Double Buffering

Initialize: Split data memory into two regions (`First`, `Last`)
Load SHA-RV configuration and custom SHA-224/256 instruction
while there are message blocks to process **do**
  if First buffer is free **then**
    WRITE new input data to `First`
    EXEC computation on `Last`
    READ output from `Last`
  else
    WRITE new input data to `Last`
    EXEC computation on `First`
    READ output from `First`
  end if
end while

Table I
BufferSet data map for SHA-224/256

Range (32-bit)	Content	Note
Long–message mode (digest stored after all blocks)		
0–7	IV	H0..H7
8–71	K[0..63]	64 round constants
72–79	digest	final digest
80–95	message block 0	W[0..15] source
96–111	message block 1	continue 16 words steps
⋮	⋮	until end of DMEM copy
Short–message mode (digest after each block)		
0–7	IV	H0..H7
8–71	K[0..63]	64 round constants
72–87	message block 0	W[0..15] source
88–95	digest 0	digest of block 0
96–111	message block 1	next block
112–119	digest 1	digest of block 1
⋮	⋮	until end of DMEM copy

accelerating the SHA-RV core does not significantly improve the overall system performance. To overcome this limitation, a **system-level pipeline** combined with a **ping-pong (double-buffering) memory mechanism** is proposed to hide data-transfer latency and maximize throughput.

As shown in Figure 5 and Algorithm 1, the data memory is divided into two equal regions, referred to as **First** and **Last**. Initially, configuration instructions and custom SHA commands are issued from the instruction memory to the SHA-RV core. Input data for multiple message blocks is then written into the **First** half of the data memory (WRITE First). When this region is filled, the SHA-RV core begins execution (EXEC First). During this execution phase, new data is written to the **Last** half (WRITE Last). Once the first execution is completed, the system performs overlapped operations: while the SHA-RV core processes data in the **Last** half (EXEC Last), the processing system reads output from the **First** half (READ First) and writes new input into that same region (WRITE First). This alternating process continues seamlessly, ensuring

Algorithm 2 Cycle-accurate FSM control for SHA-256 with BufferSet

state ← *IDLE*; *cnt* ← 0;
k_iv_loaded ← 0; *blk_idx* ← 0
while true **do**
  if *state* = *IDLE* **then**
    if *start_sha* = 1 **then**
      state ← *PREP*
    end if
  else if *state* = *PREP* **then**
    ▷ one-time IV and K staging
    if *k_iv_loaded* = 0 **then**
      DMEM→BufferSet:
      IV→[0..7], K[0..63]→[8..71];
      k_iv_loaded ← 1
    end if
    state ← *LOADMSG*
  else if *state* = *LOADMSG* **then**
    ▷ stage next message block
    DMEM→BufferSet:
    copy 16 words of block *blk_idx*
    long: to $[80 + 16 \cdot blk\_idx..95 + 16 \cdot blk\_idx]$; short: per Table I
    cnt ← 0; *state* ← *EXEC*
  else if *state* = *EXEC* **then**
    ▷ rounds and schedule with 4-stage core
    enable BufferSet reads for $K[i]$ and $W[i]$
    if *cnt* = 16 **then**
      advance to schedule expansion
    end if
    if *cnt* = 65 **then**
      enter finalization window
    end if
    if *cnt* = 82 **then**
      state ← *FINAL*
    else
      cnt ← *cnt* + 1
    end if
  else if *state* = *FINAL* **then**
    ▷ write digest to mapped slots
    long: digest→[72..79];
    short: per-block slots
    state ← *DONE*
  else if *state* = *DONE* **then**
    done_sha ← 1
    if *next_block* = 1 **then**
      blk_idx ← *blk_idx* + 1; *done_sha* ← 0;
      state ← *LOADMSG*
    else if *start_sha* = 0 **then**
      done_sha ← 0; *state* ← *IDLE*
    end if
  end if
end while

continuous computation without idle transfer periods. The ping-pong buffering strategy enables concurrent execution and data movement, fully utilizing both the SHA-RV core and the memory bus. For optimal performance, the transfer time T_{mem} must satisfy $T_{mem} < T_{exe}$, ensuring that data movement is completed before the

Table II
FSM specification for SHA-256 control (states, transitions, and counter windows)

State	Code	Transition condition	Counter window	Primary operations
IDLE	000	start_sha = 1	–	wait for command
PREP	001	k_iv_loaded set	–	stage IV and K to BufferSet
LOADMSG	010	message copied	–	stage next 16-word block
EXEC	011	cnt = 82	0–15, 16–64, 65–81	rounds and schedule from BufferSet
FINAL	100	digest written	–	store digest to mapped slots
DONE	101	next_block or stop	–	signal done; iterate or return

next pipeline iteration. Under this condition, the total system throughput approaches the raw hardware speed of the SHA-RV core.

Overall, this system-level pipelining mechanism effectively eliminates transfer-induced stalls and ensures continuous SHA-224/256 processing. As a result, the SoC can maintain near-peak throughput even when processing long message streams, achieving efficient data reuse and balanced utilization between computation and communication.

2.5 FSM-Orchestrated BufferSet Mapping

The control bottleneck that arises when a baseline RISC-V host must micromanage SHA-224/256 rounds with generic loads, stores, and branches. In such a flow, the four-stage pipeline SHA core frequently stalls because message words and K constants are not staged at cycle-accurate boundaries, and the digest writeback interleaves with data movement. To keep the pipeline fully utilized, the read–write session between data memory (DMEM) and the internal *BufferSet* must be regulated so operands arrive exactly when needed, while the host is relieved from per-round sequencing.

The proposed idea is a lightweight FSM-driven control paired with a fixed BufferSet map and a single custom SHA instruction. The host performs one bulk move of data from DMEM into the *BufferSet* following Table I (long- and short-message layouts). It then issues the custom instruction to start the SHA session. From that point, the FSM in Algorithm 2 (summarized in Table II) proceeds through *PREP*, *LOADMSG*, *EXEC*, *FINAL*, and *DONE*: it ensures IV and K are present once, stages each 16-word message block into its slice, opens read windows so the four-stage core consumes $W[i]$ and $K[i]$ at cycles 0–15 and 16–63, and deposits the digest in the mapped locations (global for long-message mode, per-block for short-message mode). The controller thus maintains continuous issue to the pipeline without host-inserted bubbles.

This scheme minimizes instruction overhead on the RISC-V host, sustains back-to-back block processing, and aligns operand timing with the four-stage SHA core. The next section presents the implementation on ZCU102 and reports utilization, frequency, latency per block, and system-level throughput and energy-efficiency measurements.

3 Evaluation Results

3.1 Implementation Results on ZCU102 FPGA SoC

To evaluate the functionality and efficiency, SHA-RV was implemented on the Zynq UltraScale+ MPSoC ZCU102 FPGA SoC platform, as illustrated in Figure 1. The processing system is managed by an ARM Cortex-A53 CPU running Debian GNU/Linux 11, installed via PetaLinux 2022.2. The programmable logic hosts the SHA-RV IP, synthesized using Vivado Design Suite 2022.2.

SHA-RV was verified by executing all four supported hashing configurations, defined by the set $\mathcal{H}$ = {SHA-224 short, SHA-256 short, SHA-224 long, SHA-256 long}. Real-time SoC verification was conducted with 100,000 randomly generated message blocks per mode, and all tests achieved 100% correctness at the maximum operating frequency of 300 MHz. We additionally investigated timing closure at 300 MHz. The critical path is located inside the SHA-256 datapath, mainly along the logic path from variable/register a to the EP0 computation. To enable high-frequency operation, the SHA core is partitioned into four pipeline stages, where each stage is constrained to include at most one adder and only lightweight bitwise transformations, thereby reducing the combinational depth per stage and facilitating timing closure at 300 MHz. For completeness, we also report the resource utilization of the whole SHA-RV SoC system on ZCU102, which consumes 28,732 LUTs, 14,856 FFs, and 16 BRAM blocks, including the RISC-V host integration and system management logic. The synthesis and implementation results show that SHA-RV occupies 5,175 LUTs, 3,146 flip-flops (FFs), and 15 Block RAM tiles (36 KB each). Power estimation for the SHA-RV core was obtained from the Vivado post-implementation power report, resulting in 0.137 W dynamic power under the reported operating conditions. System-level power was measured on the ZCU102 board using the on-board INA226 power sensors, reflecting the total dynamic power of PS + PL during execution. We ran 1,000,000 SHA-256 hash cases and repeated the measurement 10 times, reporting the average value ($\approx$ 0.75 W).

Overall, SHA-RV demonstrates stable high-frequency operation and complete functional correctness on a real-time SoC system. The combination of its high throughput, low dynamic power, and efficient FPGA resource utilization makes it an excellent candidate for

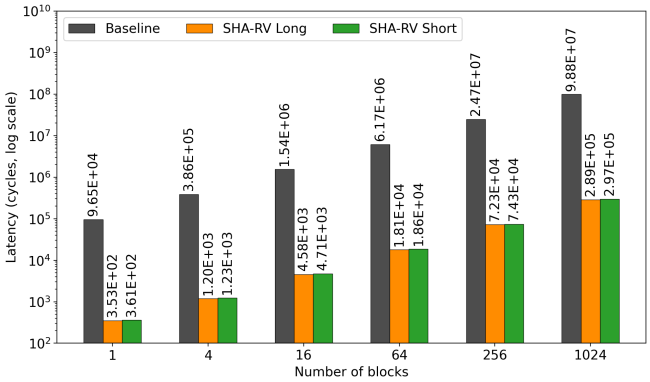

Figure 6. Total cycle counts of SHA-RV vs. baseline RISC-V on SHA-224/256 (short/long) across different numbers of blocks; effect of BufferSet mapping and the four-stage core.

integration into low-power, secure SoC applications.

3.2 Cycle count benchmarking on SHA-224/256 (short/long) and comparison with baseline RISC-V

Figure 6 reports end-to-end cycle counts for SHA-RV versus a baseline RISC-V software path on SHA-224/256 across 1, 4, 16, 64, 256, and 1024 blocks. The baseline executes a fixed 96,466 cycles per 64-byte block, while SHA-RV amortizes a one-time IV + K staging (72 cycles) and then runs block processing with low transfer overheads. For N=1, SHA-RV requires 353 cycles (long) or 361 cycles (short). For N>1, the total cycles are

$$T_{\text{long}}(N) = 282N + 72, \qquad T_{\text{short}}(N) = 290N + 72,$$

where the $+24N$ (long) or $+32N$ (short) terms account for data movement, and the $258N$ term covers four-stage SHA core execution.

The improvement over the baseline scales with N. On *long-message* mode, SHA-RV reduces cycles by **273**× at N=1 (96,466 vs. 353 cycles) and grows to **342**× at N=1024 (98.78×10^6 vs. 288,840 cycles), with a representative **341**× at N=256 (24.70×10^6 vs. 72,264 cycles). On *short-message* mode, SHA-RV delivers **267**× speedup at N=1 (96,466 vs. 361 cycles), rising to **333**× at N=1024 (98.78×10^6 vs. 297,032 cycles), and about **332**× at N=256 (24.70×10^6 vs. 74,312 cycles). These results confirm that the custom BufferSet mapping plus the four-stage core keeps hashing throughput saturated while the per-block overhead remains negligible.

Overall, SHA-RV achieves from **273**× **to 342**× fewer cycles than baseline RISC-V for long messages and from **267**× **to 333**× for short messages, depending on the batch size N. This establishes a consistent cycle-level advantage over the baseline across both operating modes.

3.3 Performance Comparison with Baseline RISC-V Implementation

To assess the hardware efficiency and computational performance, the proposed SHA-RV was benchmarked against recent RISC-V-based SHA-256 accelerators, as summarized in Table III. The comparison includes representative works from ATC2023 [7], ISOCC2024 [8], HP3C2022 [19], and LCIoT2025 [20].

Table III
Comparison with related RISC-V works on SHA-256 with a 64-byte message block

Reference	Cycles	FFs	LUTs	BRAMs
ATC2023 [7]	2,495	8,347	2,968	16
ISOCC2024 [8]	2,495	29,644	34,898	16
HP3C2022 [19]	8,278	–	–	–
LCIoT2025 [20]	34,667	1,345	2,434	–
SHA-RV (This Work)*	**257**	**3,146**	**5,175**	**15**

* The FF/LUT/BRAM results report only the SHA-RV core (RISC-V core with SHA extensions) implemented in PL, excluding the full SoC integration overhead. The whole SoC resource utilization on ZCU102 is additionally reported in Section 3.1.

As shown in Table III, SHA-RV achieves a remarkable reduction in computation latency, requiring only **257 cycles** to process one 512-bit message block, while the previous designs require 2,495 [7], 8,278 [19], and 34,667 [20] cycles, respectively. This corresponds to an improvement of approximately **9.7**×, **32.2**×, and **134.9**× compared with ATC2023, HP3C2022, and LCIoT2025. Even when compared to the high-performance ISOCC2024 implementation [8], SHA-RV still achieves a **9.7**× reduction in cycles.

In terms of resource utilization, SHA-RV demonstrates an excellent balance between performance and hardware cost. The design uses only **5,175 LUTs**, **3,146 FFs**, and **15 BRAMs**, which is significantly smaller than the ISOCC2024 [8] implementation that consumes 34,898 LUTs and 29,644 FFs. This corresponds to a reduction of approximately **85.2%** in LUTs and **89.4%** in FFs while maintaining superior processing speed.

Overall, SHA-RV delivers outstanding hashing performance with a minimal hardware footprint, validating the efficiency of its four-stage pipelined SHA core and high-bandwidth buffer mechanism. The architecture demonstrates the capability to outperform prior RISC-V implementations by an order of magnitude in throughput while significantly reducing hardware complexity, making it highly suitable for embedded and IoT-oriented secure computing systems.

3.4 Comparison with powerful CPUs in throughput and energy efficiency

Figure 7 summarizes the throughput and energy–efficiency of SHA-RV against contemporary CPUs on SHA-256 with a 512-bit block.

As the throughput chart in Figure 7(a), SHA-RV reaches **599 Mbps**, outperforming all baseline CPUs: **17.4**× over Core i7-12700H (34.5 Mbps), **15.7**× over i5-12600K (38.18 Mbps), **13.6**× over Ryzen 7 7800X3D (44.22 Mbps), **13.8**× over Ryzen 7 8845HS (43.40 Mbps), **5.1**× over Core i9-10940X (117.8 Mbps), and **33.7**× over Cortex-A53 (17.8 Mbps).

The energy–efficiency chart in Figure 7(b) further accentuates this gap: using the measured average full-system (PS+PL) dynamic power of

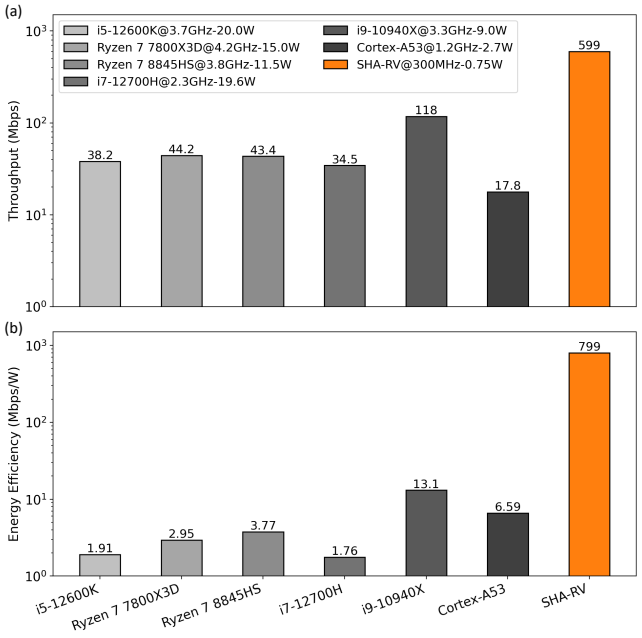

Figure 7. Throughput and energy–efficiency comparison on SHA-256 with a 512-bit block: SHA-RV vs. contemporary CPUs. (a) Throughput (Mbps). (b) Energy–efficiency (Mbps/W).

0.75 W obtained from INA226 sensors for SHA-RV, the design delivers **798.7 Mbps/W**, which is **454**× higher than i7-12700H (1.76 Mbps/W), **418**× higher than i5-12600K (1.91 Mbps/W), **271**× higher than 7800X3D (2.95 Mbps/W), **212**× higher than 8845HS (3.77 Mbps/W), **61**× higher than i9-10940X (13.09 Mbps/W), and **121**× higher than Cortex-A53 (6.59 Mbps/W).

Overall, SHA-RV achieves CPU-class or better throughput at a fraction of the power, yielding two orders of magnitude improvement in *Mbps/W* for real-time, energy-constrained deployments.

4 Conclusion

This paper presented SHA-RV, a hardware-efficient RISC-V accelerator with low-latency instruction extensions for SHA-256 and SHA-224. By combining a high-bandwidth BufferSet, a four-stage pipelined SHA core, a system-level double-buffering pipeline, and an FSM-orchestrated BufferSet mapping, the design achieves 257 cycles per 64-byte block and runs at up to 300 MHz on ZCU102 using 3,146 flip-flops, 5,175 lookup tables, and 15 block RAMs. Compared with recent RISC-V implementations, SHA-RV reduces cycle counts by between 9.7 and 134.9 times and substantially lowers logic resources. In system measurements, the design reaches 599 Mbps throughput and 798.7 Mbps/W energy efficiency with a 0.75 W measured average PS+PL dynamic power (INA226-based on-board measurement), providing large margins over contemporary CPUs. Future work will extend the instruction set and buffering strategy to multi-block chaining and additional hash variants, and will refine rail-level power measurements for complete system energy reporting.

Acknowledgment

This research was funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.01-2025.50

References

[1] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for RISC-V," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, 2014.

[2] T. N. Van, P. H. Pham, V. T. D. Le, H. L. Pham, T. H. Vu, and T. D. Tran, "AES-RV: Hardware-efficient RISC-V accelerator with low-latency AES instruction extension for IoT security," *IEICE Electronics Express*, vol. 22, no. 16, pp. 20 250 329–20 250 329, 2025.

[3] M. Ao, X. Zhou, X. Kong, S. Gou, S. Chen, X. Dong, Y. Zhu, Q. Sun, Z. Zhang, J. Zhang *et al.*, "A RISC-V 32-bit microprocessor based on two-dimensional semiconductors," *Nature*, pp. 1–8, 2025.

[4] N. H. Nguyen, D. H. A. Le, V. T. D. Le, V. T. Nguyen, T. H. Vu, H. L. Pham, and Y. Nakashima, "LI-RV: A Fast and Efficient RISC-V based Coprocessor for Lightweight Cryptography," in *Proceedings of the 2024 21st International SoC Design Conference (ISOCC)*, 2024, pp. 1–2.

[5] A. Waterman and K. Asanović, "The RISC-V instruction set manual, volume I: User-level ISA," in *Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley*, 2014.

[6] J. Park, K. Han, E. Choi, J.-J. Lee, K. Lee, W. Lee, and M. Pedram, "Designing low-power RISC-V multicore processors with a shared lightweight floating point unit for IoT endnodes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[7] V. T. D. Le, D. H. A. Le, T. H. Y. Tran, T. H. Vu, and H. L. Pham, "RVCP: High-Efficiency RISC-V Co-Processor for Security Applications in IoT and Server Systems," in *Proceedings of the 2023 International Conference on Advanced Technologies for Communications (ATC)*, 2023, pp. 1–6.

[8] D. H. A. Le, V. T. D. Le, V. A. Ho, V. T. Nguyen, H. L. Pham, V. D. Tran, T. H. Vu, and Y. Nakashima, "High-Efficiency RISC-V-Based Cryptographic Coprocessor for Security Applications," in *Proceedings of the 2024 21st International SoC Design Conference*, 2024, pp. 103–104.

[9] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," U.S. Department of Commerce, Tech. Rep., August 2015.

[10] F. Kahri, H. Mestiri, B. Bouallegue, and M. Machhout, "Efficient FPGA hardware implementation of secure hash function SHA-256/Blake-256," in *Proceedings of the 2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15)*. IEEE, 2015, pp. 1–5.

[11] T. Li, C. Cheng, R. Wang, C. Wang, X. Zou, and D. Yu, "A High Performance Hardware Implementation of SHA-256 Algorithm," in *Proceedings of the 2024 IEEE 4th International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, vol. 4, 2024, pp. 477–481.

[12] M. Kammoun, M. Elleuchi, M. Abid, and M. S. BenSaleh, "FPGA-based implementation of the SHA-256 hash algorithm," in *Proceedings of the 2020 IEEE international conference on design & test of integrated micro & nano-systems (DTS)*. IEEE, 2020, pp. 1–6.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[14] A. A. Ahmed and O. O. Alabi, "Secure and scalable blockchain-based federated learning for cryptocurrency fraud detection: A systematic review," *IEEE Access*, vol. 12, pp. 102 219–102 241, 2024.

[15] U. Q. Bajra, E. Rogova, and S. Avdiaj, "Cryptocurrency

blockchain and its carbon footprint: Anticipating future challenges," *Technology in Society*, vol. 77, p. 102571, 2024.
[16] R. Martino and A. Cilardo, "Designing a SHA-256 processor for blockchain-based IoT applications," *Internet of Things*, vol. 11, p. 100254, 2020.
[17] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge Computing Security: State of the Art and Challenges," in *Proceedings of the 2020 IEEE international conference on design & test of integrated micro & nano-systems (DTS)*, vol. 107, no. 8, 2019, pp. 1608–1631.
[18] V. T. D. Le, H. L. Pham, T. H. Tran, T. S. Duong, and Y. Nakashima, "Efficient and high-speed cgra accelerator for cryptographic applications," in *2023 Eleventh International Symposium on Computing and Networking (CANDAR)*. IEEE, 2023, pp. 189–195.
[19] J. Wu, X. Zheng, S. Zeng, H. Gao, and X. Xiong, "High-Performance Cryptographic SoC Virtual Prototyping Platform Based on RISC-V VP," in *Proceedings of the 6th International Conference on High Performance Compilation, Computing and Communications (HP3C)*, 2022, pp. 84–90.
[20] F. Kreff, L. R. Prade, R. M. de Figueiredo, and J. Schmith, "A RISC-V Approach to Energy-Efficient Cryptographic Processing in IoT Application," in *Proceedings of the 2025 IEEE Latin Conference on IoT (LCIoT)*, 2025, pp. 298–301.
[21] V. T. D. Le, H. L. Pham, T. H. Tran, and Y. Nakashima, "Flexible and Energy-efficient Crypto-Processor for Arbitrary Input Length Processing in Blockchain-based IoT Applications," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 107, no. 3, pp. 319–330, 2024.
[22] V. T. D. Le, P. H. Luan, T. H. Tran, and Y. Nakashima, "CSIP: A Compact Scrypt IP design with single PBKDF2 core for Blockchain mining," in *Proceedings of the 2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2022, pp. 1–6.

Pham Tuan Dat received the Engineering degree in Electrical and Electronic Engineering from Le Quy Don Technical University, Ha Noi, Viet Nam, in 2021. Author has been active in the field of electronics and information engineering. His research interests include electronic systems, communication technologies, and information processing.

Luu Van Tuan received the B.Sc. degree in electrical and electronic engineering and the M.S. degree in Radar and navigation engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2012 and 2018, respectively. He also is currently working toward the Ph.D. degree in electronic engineering at Le Quy Don Technical University, Hanoi, Vietnam. His research interests include hardware security, embedded systems, and VLSI architecture for digital processing.

Tran Thi Diem received B.E. and M.E. degrees in Physical Electronics Engineering from the University of Science, VNU-HCM, Vietnam, in 2006 and 2009, respectively. She received a Ph.D. degree in Information Science from the Nara Institute of Science and Technology, Japan. Since 2006, she has been a lecturer in the Department of Computer Engineering, University of Information Technology, VNU-HCM, Vietnam. Her research interests include computer architecture, signal processing, and artificial neural networks.

Dao Hoang Nam received the B.Eng. and M.Eng. degrees from Le Quy Don Technical University, Hanoi, Vietnam, in 2010 and 2017, respectively. He earned the D.Eng. degree in Telecommunication Engineering from King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand, in 2022. He is currently a researcher at the High-Tech Telecommunication Center in Hanoi, Vietnam. His research interests include antennas for wireless communications and microwave applications in agriculture.

Le Vu Trung Duong received the Bachelor of Engineering degree in IC and hardware design from Vietnam National University Ho Chi Minh City (VNU-HCM)—University of Information Technology (UIT) in 2020, and the Master's degree in information science from the Nara Institute of Science and Technology (NAIST), Japan, in 2022. He completed his Ph.D. degree in 2024 at NAIST, where he now serves as an Assistant Professor at the Computing Architecture Laboratory. His research interests include computing architecture, reconfigurable processors, and accelerator design for quantum emulation, AI and cryptography.

Nguyen Van Tinh received the Ph.D. degree in computer science from Division of Information Science, Nara Institute of Science and Technology, Nara, Japan, in 2022. Since 2023, he has been a lecturer in the Electrical Engineering Department of Le Quy Don Technical University, Ha Noi, Viet Nam. His research interests include machine learning, bigdata, blockchain, and hardware security.

Pham Thi Mai holds a Ph.D. in English Linguistics and is currently a lecturer at Foreign Trade University, Vietnam. Her research focuses on English for Specific Purposes (ESP), academic writing, and scientific communication in engineering and technology. She has extensive experience in interdisciplinary research involving language use in technical contexts.

Pham Xuan Nghia is currently a lecturer at the Faculty of Radio-Electronics Engineering, Le Qui Don Technical University, Hanoi, Vietnam. He earned his bachelor's and master's degrees in Information Engineering from the Le Qui Don Technical University and holds a Ph.D. in Electronics Engineering from the Russian Federation. His research interests include information theory, signal processing and wireless communications.