Microarchitecture Design and Benchmarking of Custom SHA-3 Instruction for RISC-V

Alperen Bolat, Sakir Sezer, Kieran McLaughlin, Henry Hui
Queen's University Belfast, Belfast, United Kingdom

***Abstract*—Integrating cryptographic accelerators into modern CPU architectures presents unique microarchitectural challenges, particularly when extending instruction sets with complex and multistage operations. Hardware-assisted cryptographic instructions, such as Intel's AES-NI and ARM's custom instructions for encryption workloads, have demonstrated substantial performance improvements. However, efficient SHA-3 acceleration remains an open problem due to its distinct permutation-based structure and memory access patterns. Existing solutions primarily rely on standalone coprocessors or software optimizations, often avoiding the complexities of direct microarchitectural integration. This study investigates the architectural challenges of embedding a SHA-3 permutation operation as a custom instruction within a general-purpose processor, focusing on pipelined simultaneous execution, storage utilization, and hardware cost. In this paper, we investigated and prototyped a SHA-3 custom instruction for the RISC-V CPU architecture. Using cycle-accurate GEM5 simulations and FPGA prototyping, our results demonstrate performance improvements of up to 8.02× for RISC-V optimized SHA-3 software workloads and up to 46.31× for Keccak-specific software workloads, with only a 15.09% increase in registers and a 11.51% increase in LUT utilization. These findings provide critical insights into the feasibility and impact of SHA-3 acceleration at the microarchitectural level, highlighting practical design considerations for future cryptographic instruction set extensions.**

I. Introduction and Related Work

SHA-3 (Secure Hash Algorithm 3), standardized by NIST, is a cryptographic hash function and known for its robust resistance to cryptographic attacks and its adaptability across diverse applications, including authentication, data integrity, and blockchain safety. Unlike previous SHA family members, SHA-3 is built on the Keccak permutation, which employs a sponge construction to absorb input and produce secure hash outputs efficiently. This design underpins SHA-3's growing adoption in security-critical domains for the future.

To accelerate cryptographic workloads, many modern CPU architectures, such as Intel x86 and ARM, have introduced dedicated instruction set extensions like AES-NI [1] and ARM Cryptographic Extensions [2] that significantly speed up algorithms such as AES and SHA-2. These hardware-assisted instructions have become standard in industry, demonstrating substantial gains in both performance and energy efficiency.

However, despite SHA-3's increasing adoption in security applications, its integration at the instruction set level remains largely unexplored. Unlike SHA-2, which benefits from well-established instruction-level acceleration, SHA-3 presents unique computational characteristics due to its permutation-based Keccak structure, which consists of extensive bitwise operations, nonlinear transformations, and complex memory access patterns. While numerous studies have explored SHA-3 acceleration via dedicated hardware implementations on ASICs [3], [4] or FPGAs [5], [6] . These approaches operate as dedicated accelerators rather than integrated or embedded within a general-purpose processor. The lack of systematic investigation into SHA-3's microarchitectural integration and integration of dedicated instruction leaves an open research question: how can SHA-3 be efficiently executed as a dedicated instruction within a CPU pipeline, similar to existing industrial solutions like cryptographic instruction extensions proposed for AES and SHA-2?

Prior research has examined alternative SHA-3 acceleration methods, including coprocessors [7], SoC level integrations [8], and modifications on existing standard vector instructions [9], as well as optimizations tailored to general-purpose CPU execution pipelines improving compatibility of integration [10] to reduce communication overhead between CPU and accelerator [11]. Besides these, some studies have also explored the performance leverage potential of SHA-3 when they closely integrated existing SIMD and Vector instruction in the CPU [12], [13]. However, studies haven't comprehensively analyzed the feasibility, microarchitectural challenges, and trade-offs of embedding dedicated SHA-3 capable custom instruction directly within a CPU's datapath. Also, key considerations such as pipeline integration, memory efficiency, and hardware resource utilization remain largely unexplored.

In this work, we propose a novel instruction-level acceleration approach for SHA-3 by directly embedding custom SHA-3 (Keccak) capable instruction into a RISC-V processor. We design and implement a SHA-3-specific instruction, analyze its execution within the CPU pipeline, and evaluate its impact on performance and area overhead. Our contributions are as follows:

- First microarchitectural study of SHA-3 instruction integration directly into CPU datapath: We analyze the feasibility and challenges of embedding SHA-3 as a dedicated instruction within a general-purpose processor.
- Custom SHA-3 instruction design with cycle-accurate CPU analysis and hardware prototype: We design and implement a SHA-3-specific instruction within a RISC-V-based processor on GEM5 and validate on FPGA hardware.
- Comprehensive performance evaluation: Using GEM5 simulations and FPGA prototype, we demonstrate sig-

nificant speedups—up to 8.02× for RISC-V-optimized SHA-3 and 46.31× for Keccak workloads—with minimal hardware overhead (15.09% increase in flip-flops and 11.51% increase in LUT utilization).

II. SHA-3 Computational Structure and Keccak-f Permutation Acceleration

Depending on its operation types, the SHA-3 algorithm consists of two primary components:

- *Data Processing*
- *Keccak-f Permutation*

Data processing involves preparing the input message for cryptographic hashing and consists of the following steps:

- *Padding:* A predefined bit sequence is appended to the input message to ensure it is compatible with block size requirements.
- *Message Splitting:* The padded message is divided into fixed-sized blocks for sequential processing.
- *Absorbing Phase:* Each block is sequentially incorporated into a structured internal state matrix, represented as a 5×5 grid of 64-bit words.

These steps primarily involve data movement operations—memory accesses, buffer handling, and state updates—rather than computationally intensive arithmetic or logic operations. Since modern CPU architectures are already optimized for general-purpose data movement and memory operations, accelerating these steps at the instruction level provides limited performance gains.

Keccak-f permutation, in contrast, constitutes the computational core of SHA-3. It is responsible for the cryptographic transformation of the state matrix and consists of 24 iterative rounds, each composed of the following five fundamental operations:

- θ (Theta) – Ensures diffusion by computing parity across column-wise slices of the existing state on calculation.
- ρ (Rho) – Applies cyclic bitwise rotations to introduce non-linearity on state.
- π (Pi) – Permutes state bits by rearranging word positions on matrix.
- χ (Chi) – Applies a non-linear Boolean function based on bitwise operations.
- ι (Iota) – Injects round-dependent constants to strengthen resistance against linear and differential cryptanalysis.

These operations rely heavily on bitwise XORs, ANDs, shifts, and modular rotations, making them computationally expensive when executed sequentially as software on general-purpose CPUs. Unlike the data processing phase, which is primarily constrained by memory bandwidth and register usage, the Keccak-f permutation needs a significant instruction execution overhead due to its reliance on intensive arithmetic and logical calculations.

A. Keccak-f Permutation as Combinational Logic

A key architectural property of the Keccak-f permutation is that it partially supports implementation as pure combinational logic.

Since each round of Keccak consists of well-defined bitwise transformations and calculations, dedicated hardware implementations can leverage fully parallelized combinational circuits to compute multiple rounds concurrently, significantly reducing execution latency. This inherent parallelism contrasts sharply with sequential software execution on general-purpose CPUs, where each logical operation turns into multiple instruction cycles, causing high computational overhead.

While dedicated or co-processor type accelerator designs can exploit this characteristic to achieve ultra-low latency hashing,

modern CPUs lack dedicated hardware primitives for Keccak-f. Instead, they execute the permutation in a highly serialized behavior, requiring multiple instructions for each XOR, shift, and bitwise permutation. This inefficiency motivates the use of custom instruction-level acceleration, wherein dedicated instructions can replace the behavior of combinational logic while remaining within the constraints of a CPU's pipeline.

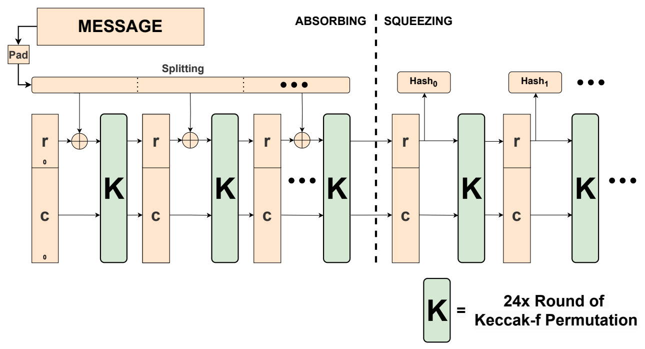

Fig. 1: SHA-3 Algorithm Overview

Figure 1 illustrates the SHA-3 algorithm, highlighting the flow and relation between data processing and Keccak-f permutation steps. The parameters, including the rate (r) and capacity (c), are adjusted based on output length requirements and input data sizes.

B. Motivation for Custom Instruction Acceleration

In this study, instead of attempting to optimize the entire algorithm, we focus on reducing the Keccak-f permutation's computational complexity, which dominates the total execution time of SHA-3 algorithm. By designing custom instruction that efficiently map the arithmetic and logical transformations involved in θ, ρ, π, χ, and ι, we aim to:

- *Reduce instruction count:* Minimize the number of general-purpose arithmetic and logic operations required to execute Keccak-f rounds.
- *Improve pipeline efficiency:* Enable a tightly coupled execution unit for SHA-3 to process permutation rounds efficiently with fewer cycles and stalls.
- *Leverage combinational logic properties:* Utilize custom instructions that align with the parallel nature of Keccak-f, enabling reduced execution latency.

This study does not address general-purpose data manipulation tasks, such as padding, message splitting, or absorbing/squeezing phases, as they are inherently constrained by CPU memory architectures rather than computational efficiency. Instead, we focus on investigating accelerating the Keccak-f permutation as a dedicated custom instruction, where the majority of SHA-3's computational effort is concentrated.

III. Architectural Design

The proposed `shatr` instruction is specifically designed to execute iterations of the Keccak-f permutation in a combinational manner. Each iteration comprises five sequential operations: Theta (θ), Rho (ρ), Pi (π), Chi (χ), and Iota (ι). Collectively, these operations implement the cryptographic hashing computations necessary for SHA-3. Given that SHA-3 requires 24 rounds of the Keccak-f permutation per input block, executing the `shatr` instruction 24 times completes the full permutation process.

Implementations of the Keccak-f permutation iteration can be realized using combinational logic [3]–[6], [14]–[16]. According to the standard specification of SHA-3, each input block processed by the Keccak-f permutation consists of 200 bytes of data. Throughout the 24 rounds of the Keccak-f permutation, this data is iteratively updated by performing combinational operations corresponding to each round.

To leverage the computational efficiency of the Keccak-f permutation and reduce reliance on standard arithmetic and logical instructions within the CPU datapath, we integrated a dedicated Keccak-f Execution Unit into the CPU pipeline. This execution unit is implemented as a combinational logic-based hardware accelerator specifically optimized for performing Keccak-f permutations. During each execution of the `shatr` instruction, this unit recursively operates on a 200-byte data chunk. Thus, after executing a single `shatr` instruction, one round of Keccak-f permutation is completed. Figure 2 outlines the high-level micro-architecture of the dedicated Keccak-f Execution Unit within a standard CPU pipeline.

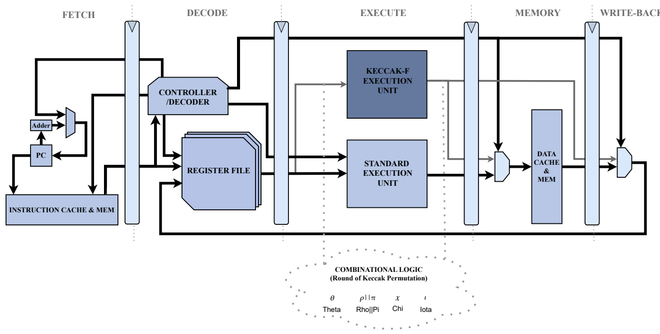

Fig. 2: Micro-Architectural Model

By embedding this dedicated Keccak-f Execution Unit into the CPU execution stage, the processor can efficiently perform calculations required by Keccak-f rounds without resorting to multiple standard arithmetic and logical instructions. Consequently, this integration significantly reduces the total number of executed instructions necessary for SHA-3 computations, thereby enhancing overall computational efficiency.

Although standard CPU registers can be utilized to feed data into the Keccak-f Execution Unit, their limited quantity can lead to additional cycles due to frequent data swaps between memory and registers. To address this limitation and improve pipeline parallelism, we implemented dedicated internal registers (flip-flops) totalling 200 bytes within the execution unit itself. These internal buffers function similarly to customized vector registers specifically tailored for buffering intermediate data during iterative Keccak-f permutations. By employing these dedicated internal registers, redundant access to standard CPU registers is minimized or eliminated altogether and with vectorisation, parallel read/write capabilities are achieved in the CPU pipeline.

The proposed `shatr` instruction does not inherently accelerate data movement nor reduce memory fetch requirements directly. The introduced internal registers are managed exclusively through standard CPU instructions without necessitating additional custom instructions or specialized memory access mechanisms associated with `shatr` itself. However, by allocating dedicated vector type registers within the CPU pipeline, we effectively mitigate resource constraints typically encountered during Keccak-f computations. This approach indirectly reduces unnecessary register swaps and additional operations required for accessing intermediate data from memory or standard CPU registers.

Figure 3 illustrates the optimized execution flow of SHA-3 enabled by the `shatr` instruction, emphasizing the reduction in communication overhead achieved through its integration. The figure also addresses the division of execution responsibilities, showing where the Keccak-f Execution Unit, operating with the `shatr` instruction, performs its specialized combinational tasks and where standard execution occurs during the complete SHA-3 computation process.

IV. Implementation Details

A. Custom Instruction Integration

The integration of the `shatr` instruction required targeted modifications to the RISC-V toolchain, including updates to the assembler, linker, compiler, and simulator, to ensure seamless compatibility with the RISC-V framework. The instruction was assigned to an unused opcode slot, carefully chosen to avoid conflicts with existing instructions while preserving backwards compatibility with the RISC-V Instruction Set Architecture (ISA). This ensures that the addition of `shatr` does not disrupt standard workloads or require significant changes to the existing software ecosystem.

Once the opcode was assigned, the RISC-V compiler was updated to recognize and generate the `shatr` instruction during standard compilation. The assembler and linker were modified to support seamless processing of `shatr` in binary generation, and the simulator was enhanced to accurately model its behavior within the CPU pipeline.

These modifications ensure that `shatr` integrates seamlessly alongside standard RISC-V instructions while adhering

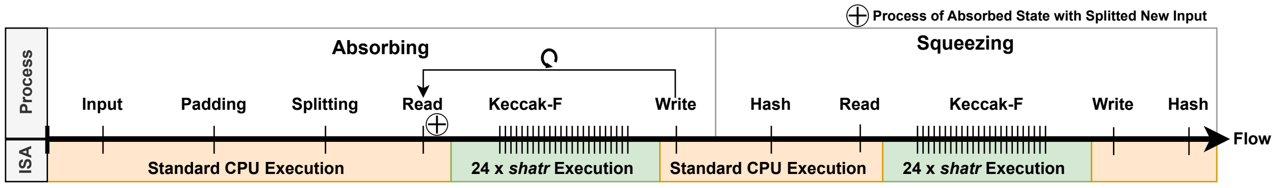

Fig. 3: Execution Flow of SHA-3

to principles of modularity and backward compatibility. Existing RISC-V applications remain unaffected, while a high-performance path is provided for cryptographic workloads. By embedding this custom instruction into the standard toolchain, the proposed architecture achieves scalability, maintainability, and ease of adoption for both legacy and new software ecosystems. This approach allows developers to leverage the enhanced computational capabilities of `shatr` without disrupting existing workflows or requiring extensive reengineering efforts.

B. Implementation Details

To ensure robust evaluation, we used hardware-friendly, C-based implementations of SHA-3 source code aligned with real-world cryptographic workloads and compatible with RISC-V architectures. To evaluate the proposed approach, we selected two software distributions as benchmarks for SHA-3: (i) the official SHA-3 software from the RISC-V organization [17] and (ii) a standalone SHA-3 implementation published by the Keccak developers [18]. Both benchmarks were tested using NIST-compliant SHA-3 test vectors [19], covering a range of input sizes, including both short and long messages.

Specifically, we selected two widely recognized distributions: the official implementation provided by the Keccak developers and a RISC-V-optimized variant. These implementations were chosen to establish a reliable baseline and reflect realistic cryptographic processing scenarios. Also, by utilizing RISC-V-based distributions alongside those provided by the Keccak developers, we aimed to ensure that our evaluation avoided reliance on non-RISC-V-optimized software accelerations, which could exaggerate the performance impact of our proposed acceleration. This selection ensures a fair and realistic assessment of the `shatr` instruction's impact within RISC-V-specific environments.

The compiling evaluation utilized the latest RISC-V toolchain, configured for the 64-bit instruction set. Customized vector-based registers are exclusively employed to connect the proposed `shatr` instruction with the combinational logic of the Keccak-f Execution Unit on GEM5. No additional registers, features or optimizations were applied into ISA or CPU, as the objective was to isolate and measure the native performance impact of the custom instruction on these SHA-3 benchmarks.

In both software distributions, we integrated the `shatr` instruction by mapping SHA-3's Keccak permutation round operations to `shatr` through minimal assembly-level modifications of the source code. Only the permutation rounds were altered, leaving the rest of the codebase unchanged. This ensured that any observed performance differences could be attributed solely to the custom instruction, without interference from unrelated optimizations.

Performance results were obtained by comparing `shatr`-enabled executables against their standard counterparts using a custom cycle-accurate RISC-V CPU model implemented in GEM5. Two configurations were evaluated: one with support for proposed custom `shatr` instruction and another using a standard RISC-V CPU model without any modification. Memory was configured as 8GB DDR3 with 1600 MT/s and 8 banks.

To further evaluate the hardware utilization introduced by the proposed approach, `shatr` was configured to work with the RISC-V CVA6 core [20] and assessed using an FPGA-based implementation. This analysis provided insights into the impact of the custom instruction on area and resource utilization.

C. Validation of Experimental Environment

The SHA-3 custom instruction was validated for compatibility and correctness within the RISC-V CPU using the GEM5 simulator. The stability of `shatr` integration was confirmed through RISC-V unit tests [21] to ensure ISA compliance. Additionally, benchmarks such as Dhrystone, Linux workloads, and MiBench [22] demonstrated stable performance under computational load. For functional accuracy, hash outputs from the custom instruction were compared against NIST test vectors for all SHA-3 variants, confirming bit-level precision. These results verified robust and efficient operation of the custom instruction without errors or performance degradation.

V. Experimental Results

A. Performance Benchmarking

We benchmarked the custom `shatr` instruction using two state-of-the-art SHA-3 software distributions: (i) the RISC-V optimized SHA-3 software [17] and (ii) the Keccak Team's SHA-3 software implementation [18] on GEM5. Performance was measured across all SHA-3 output configurations, using both short and long input vectors.

Figure 4 compares the average instruction counts per SHA-3 round, demonstrating that the integration of the custom instruction significantly reduces the number of arithmetic logic unit (ALU) instructions, as well as memory reads and writes. It

TABLE I: Total Execution Cycles (x10 Million) for RISC-V and Keccak-based Software Distributions of SHA-3 with and without Custom Instruction Across SHA-3 Sizes and Input Lengths

Benchmark		**SHA-224**		**SHA-256**		**SHA-384**		**SHA-512**	
		Short	**Long**	**Short**	**Long**	**Short**	**Long**	**Short**	**Long**
RISC-V Based Distribution [17]	**Custom SHA-3 Instruction**	9.296	436.726	8.689	446.975	6.331	181.221	4.250	147.784
	Without Custom Instruction	72.275	3307.817	66.993	3434.183	51.942	1562.557	36.095	1200.756
Keccak-Based Distribution [18]	**Custom SHA-3 Instruction**	9.228	440.837	8.694	483.317	6.384	192.422	4.248	157.557
	Without Custom Instruction	417.015	19104.412	394.303	21313.550	297.836	9080.924	207.631	7846.874
Total SHA-3 Rounds During Execution of Dataset		**15,768**	**724,056**	**14,904**	**810,480**	**11,448**	**343,584**	**7,992**	**258,480**

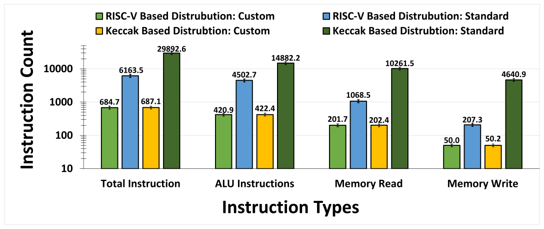

Fig. 4: Comparison of Total Executed Instruction Counts Averages Per SHA-3 Round: Custom Instruction Enabled vs. Standard Execution for Software Benchmarks

shows reduced ALU and memory operations per SHA-3 round for both benchmarks. This reduction is particularly pronounced in the Keccak-based software distribution, where the custom `shatr` instruction substantially minimizes memory request related operations. The implementation of the RISC-V Custom SHA-3 Instruction achieves a notable reduction in ALU-based computational instructions, while maintaining reduced memory access. These results show a significant decrease in memory and bit-wise (arithmetic) instructions per SHA-3 round, following the integration of the `shatr` instruction.

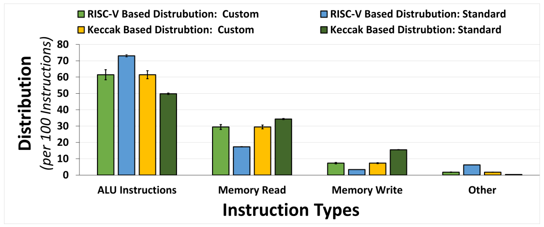

Fig. 5: Instruction Type Breakdown by Percentage: Custom Instruction vs. Standard Execution for Software Benchmarks

Figure 5 presents the distribution of instruction types per 100 instructions, alongside the trend of benchmark changes with the integration of the custom instruction. It reveals that the RISC-V-based benchmark predominantly consists of ALU instructions, while the Keccak-based software primarily utilizes instructions related to memory operations. For both applications, the integration of the custom instruction results in approximately 60% ALU instructions and a more balanced distribution between the RISC-V-optimized benchmark and the standalone Keccak software. This finding suggests that the RISC-V-optimized software distribution is more compatible with the memory system, whereas the standalone Keccak-based distribution exhibits a higher utilization of memory-related instructions.

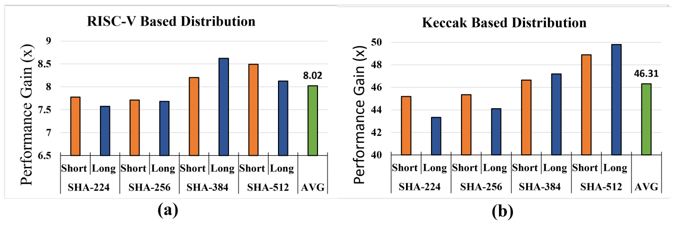

Fig. 6: Normalized Performance Gain in Execution Time: Comparison of Custom Instruction versus Standard (without SHA-3 instruction) for (a) RISC-V Based Software Distribution and (b) Keccak Based Software Distribution

Table I details the total execution cycles for both the RISC-V and Keccak-based software distributions, with and without the custom instruction, across various SHA-3 sizes and input lengths. The data indicates a substantial reduction in execution cycles, particularly for longer input vectors, where the reduction is most pronounced. This demonstrates that the hardware optimization provided by the `shatr` instruction effectively reduces computational overhead, as reflected by the marked decrease in total cycle counts. Consistent with Figure 4, the Keccak-based SHA-3 distribution is less efficient on RISC-V hardware compared to the RISC-V-optimized distribution, requiring more execution time for SHA-3 processing due to its memory-dominant load characteristics.

The normalized performance gain in execution time is illustrated in Figure 6, which demonstrates a clear improvement when the custom instruction is utilized. Across all SHA-3 sizes, the custom instruction consistently outperforms the standard execution model for both RISC-V and Keccak-based software distributions. While the performance gains are most pronounced for long input vectors with Keccak based SHA-512 and RISC-V based SHA-384, the integration of SHA-3 round instructions led to average improvements of up to 46.31x for Keccak-based software and 8.02x for RISC-V-based software across all workloads. These results confirm that, whether using RISC-V-optimized or standalone SHA-3 software, the incorporation of dedicated SHA-3 round instruction significantly offloads general-purpose SHA-3 execution.

B. Hardware Cost Results

To validate the efficiency and feasibility of the custom `shatr` instruction, we implemented and evaluated its hardware realization on an FPGA platform. The design was synthesized and deployed on the Genesys-2 FPGA board, featuring a Xilinx Kintex-7 XC7K325T device. The FPGA implementation assessed area utilization and timing performance. The custom instruction was integrated into a modified RISC-V soft-core processor, implemented in Verilog and synthesized using the Xilinx Vivado toolchain.

TABLE II: Comparison of FPGA Usage

Resource	**CPU+`shatr`**	**CPU Only**	**Change**(%)
Total LUTs	81,339	71,976	+11.51%
Logic LUTs	79,700	70,382	+11.69%
LUTRAMs	1,264	1,224	+3.16%
SRLs	375	370	+1.33%
FFs	54,223	46,041	+15.09%
RAMB36	50	50	0.00%
RAMB18	2	2	0.00%
DSP48 Blocks	27	27	0.00%
CPU Clock Rate	50 Mhz	50 Mhz	0.00%

Table II summarizes the FPGA resource usage for both the standard RISC-V soft-core and the modified version featuring the `shatr` instruction. The custom instruction introduces minimal overhead, with only a slight increase in LUTs and Flip-Flops, highlighting its efficient hardware integration without a significant hardware cost increase. Also, timing analysis shows that the custom instruction supports the official 50 MHz clock rate of the CVA6 [23] on the Genesys-2 board, with no added critical path latency.

VI. Conclusion

In this paper, we proposed a new microarchitecture for a SHA-3 custom instruction for RISC-V that can easily be tailored for any CPU architecture. Further, we presented detailed benchmark results of the proposed microarchitecture based on cycle-accurate GEM-5 simulation. In order to validate our design in terms of additional hardware and speed penalty, we developed an FPGA prototype. The experimental results clearly demonstrates that the proposed `shatr` instruction is able to achieve substantial performance improvements, with speedups ranging from 8.02x to 46.31x when compared with highly optimized software only implementation. The overall increase of hardware cost for implementing the SHA-3 custom instruction within the RISC-V data path requires additional 15.09% register and 11.51% LUTs, while maintaining the original critical path latency of the RISC-V core.

Acknowledgment

This work is funded by the UK Government through the New Deal for Northern Ireland. The funding is delivered on behalf of the Northern Ireland Office and the Department for Science, Innovation and Technology by Innovate UK. Additionally, this project is funded as part of the Grant for R&D by Invest NI (Grant: RD08201502).

References

[1] K. Akdemir, M. Dixon, W. Feghali, P. Fay, V. Gopal, J. Guilford, E. Ozturk, G. Wolrich, and R. Zohar, "Breakthrough aes performance with intel aes new instructions," *White paper, June*, vol. 12, p. 217, 2010.

[2] ARM, "Armv8-a cryptographic extension," developer.arm.com/documentation/100801/0401.

[3] F. K. Gürkaynak, K. Gaj, B. Muheim, E. Homsirikamol, C. Keller, M. Rogawski, H. Kaeslin, and J.-P. Kaps, "Lessons learned from designing a 65nm asic for evaluating third round sha-3 candidates," in *Third SHA-3 Candidate Conference*, 2012, pp. 1–22.

[4] P. Nannipieri, M. Bertolucci, L. Baldanzi, L. Crocetti, S. Di Matteo, F. Falaschi, L. Fanucci, and S. Saponara, "Sha2 and sha-3 accelerator design in a 7 nm technology within the european processor initiative," *Microprocessors and Microsystems*, vol. 87, p. 103444, 2021.

[5] K. Kobayashi, J. Ikegami, M. Knežević, E. X. Guo, S. Matsuo, S. Huang, L. Nazhandali, Ü. Kocabaş, J. Fan, A. Satoh *et al.*, "Prototyping platform for performance evaluation of sha-3 candidates," in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2010, pp. 60–63.

[6] E. Homsirikamol, M. Rogawski, and K. Gaj, "Comparing hardware performance of round 3 sha-3 candidates using multiple hardware architectures in xilinx and altera fpgas," in *Ecrypt II Hash Workshop*, vol. 2011, 2011, pp. 1–15.

[7] I. L. Azevedo, A. S. Nery, and A. d. C. Sena, "A sha-3 co-processor for iot applications," in *2020 Workshop on Communication Networks and Power Systems (WCNPS)*. IEEE, 2020, pp. 1–5.

[8] J. Rao, T. Ao, S. Xu, K. Dai, and X. Zou, "Design exploration of sha-3 asip for iot on a 32-bit risc-v processor," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 11, pp. 2698–2705, 2018.

[9] H. Li, N. Mentens, and S. Picek, "Maximizing the potential of custom risc-v vector extensions for speeding up sha-3 hash functions," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[10] A. Sideris and M. Dasygenis, "Enhancing the hardware pipelining optimization technique of the sha-3 via fpga," *Computation*, vol. 11, no. 8, p. 152, 2023.

[11] A. Bolat, F. Siddiqui, S. Sezer, K. Tasdemir, and R. Khan, "Investigation of communication overhead of soc lookaside accelerators," in *2023 IEEE 36th International System-on-Chip Conference (SOCC)*. IEEE, 2023, pp. 1–6.

[12] R. Cabral and J. López, "Implementation of the sha-3 family using avx512 instructions," in *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*. SBC, 2018, pp. 361–368.

[13] H. Rawat and P. Schaumont, "Vector instruction set extensions for efficient computation of keccak," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1778–1789, 2017.

[14] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, "Fair and comprehensive performance evaluation of 14 second round sha-3 asic implementations," in *The Second SHA-3 Candidate Conference*. Citeseer, 2010.

[15] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane, "Fpga implementations of the round two sha-3 candidates," in *2010 International Conference on Field Programmable Logic and Applications*. IEEE, 2010, pp. 400–407.

[16] L. Ioannou, H. E. Michail, and A. G. Voyiatzis, "High performance pipelined fpga implementation of the sha-3 hash algorithm," in *2015 4th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2015, pp. 68–71.

[17] RISC-V Foundation, "Risc-v cryptography extensions standardisation work," https://github.com/riscv/riscv-crypto.

[18] Keccak Team, "Keccakcodepackage," https://keccak.team/software.html.

[19] NIST, "Cryptographic algorithm validation program: Secure hashing," csrc.nist.gov/projects/cryptographic-algorithm-validation-program.

[20] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[21] RISC-V, "Risc-v tests," github.com/riscv-software-src/riscv-tests.

[22] M. R. Guthaus and R. et al., "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. of the fourth annual IEEE international workshop on workload characterization*. IEEE, 2001.

[23] Openhwgroup, "Cva6 v5.3.0," github.com/openhwgroup/cva6.