PPA Evaluation of Hardware Accelerated AES Algorithm Using RISC-V ISE

Hércules Leonel, Willian Nunes, Fernando Gehm Moraes
School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
hercules.mapl@gmail.com, willian.nunes@edu.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—RISC-V open architecture and flexibility make it a choice for embedded devices, where efficient encryption is essential for ensuring data security with minimal resource overhead. The objective of this work is to conduct a power-performance-area (PPA) evaluation of the Zkne Instruction Set Extension (ISE) for the Advanced Encryption Standard (AES) algorithm on a RISC-V processor. This evaluation uses synthesis and simulation metrics to demonstrate efficiency improvements, including area and energy costs, clock cycles, and performance monitoring registers. To analyze the impact of Zkne ISE on encryption and decryption performance, standard and extended implementations of AES are assessed on the RISC-V, using an optimized AES algorithm with T-tables, together with the TinyCrypt reference software. Results demonstrate that Zkne significantly reduces execution time and energy per operation, validating its suitability for secure and efficient cryptographic processing in resource-constrained environments.**

***Index Terms*—RISC-V, Cryptography, AES, PPA, Processor, Evaluation**

I. Introduction

The Advanced Encryption Standard (AES) [1] is a widely adopted encryption method to ensure data confidentiality and integrity. It functions as a block cipher, processing data in fixed-size blocks of 128 bits through matrix operations. The encryption key can be 128, 192, or 256 bits long, selected based on the desired security level.

RISC-V [2] is an open-standard Instruction Set Architecture (ISA) that has gained considerable interest in academic and industrial domains due to its flexibility, scalability, and open-source nature. Its modular design, supported by an open-source development model, encourages innovation and customization. This design enables adaptation to a wide range of application requirements, from compact embedded systems to high-performance computing platforms.

This work presents a PPA (power-performance-area) evaluation of the $Zkne$ ISE (Instruction Set Extension) for AES on RISC-V, using synthesis and simulation metrics such as area and energy costs, clock cycles, and memory usage to demonstrate efficiency improvements. Given the open architecture and flexibility of the RISC-V ISA, it is a suitable choice for embedded devices where efficient encryption is crucial for ensuring data security while reducing resource overhead [3].

To assess the impact of the $Zkne$ ISE [4] on encryption and decryption performance, both standard and extended AES implementations are evaluated on the RS5 RISC-V processor [5]. The evaluation incorporates optimized AES algorithms, including using T-tables executed online in hardware, eliminating the need for pre-stored lookup tables. This approach retains the performance benefits of T-tables while reducing memory access overhead and code size and improving security against timing-based side-channel attacks [6].

Using the Tinycrypt library [7] as a reference software baseline, this work comprehensively compares computational efficiency and resource utilization. By employing these techniques, this study aims to demonstrate the advantages of hardware acceleration via $Zkne$, highlighting its potential to enable secure and efficient cryptographic operations in resource-constrained environments.

II. Background Knowledge

A. RS5 Processor Core

RS5 is a 32-bit integer processor core with a modular RISC-V RV32IMCA (M: multiply, C: compresses, A: Atomic ISEs) architecture, incorporating several optional extension modules. These include $Zicntr$ and $Zihpm$ for performance counters, $Zmmul$ for hardware-driven implementations of multiplication and division, and $Xosvm$ for the Memory Management Unit (MMU). The primary focus of this work is the $Zkne$ extension, which provides an accelerated AES instruction set as part of the RISC-V Scalar Cryptography extensions.

The core operates in a four-stage pipelined architecture, as illustrated in Figure 1, comprising instruction fetch (IF), instruction decode (ID), execution unit (XU), and retire unit (WB). For interrupt handling, RS5 implements a Platform-Level Interrupt Controller (PLIC) based on the proposal by SiFive [8]. The PLIC is configurable and supports priority levels ranging from 1 to 7 for each interrupt. Additionally, the RS5 incorporates a 64-bit real-time clock (RTC), which functions as a cycle counter and is routed to the time counter in the $Zicntr$ extension.

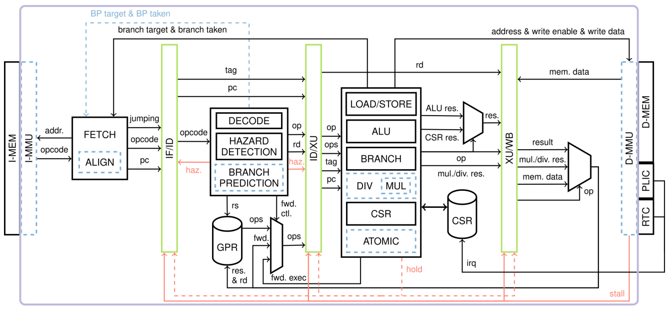

Fig. 1. RS5 Organization. Source [5]. The RS5 processor is publicly available at https://github.com/gaph-pucrs/RS5.

B. *Zkne ISE Extension*

Hardware acceleration of AES is implemented through two new instructions defined in the *Zkne* extension, ratified by RISC-V International [9]: `aes32esmi` and `aes32esi`. The `aes32esmi` instruction can be interpreted as computing T-Table entries online in hardware, then XOR-ing the current entry with previous T-Table entries for the same output column. This provides the performance benefits of a T-Table AES implementation without storing LUTs in memory, leading to gains in performance and code size, and energy efficiency gains via the reduction in overall instructions executed and memory accesses [10]. Finally, `aes32esi` performs only a single SBOX lookup and XOR, used in computing round keys and the last encryption round, with no `MixColumns`.

The *Zkne* instructions are implemented in the AES Unit module, shown in Figure 2. The AES unit is entirely combinational, executing either the `aes32esmi` and `aes32esi` instructions in a single clock cycle. Both instructions are of an extended R type, which expects as inputs 2 registers to be read from the register file, plus a 2-bit immediate Byte Select (*BS*). AND gates are used to gate inputs, minimizing switching activity in the AES unit when it is not being used.

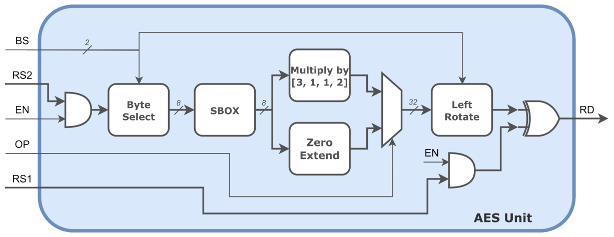

Fig. 2. AES Unit - Source: [11]

The AES functional unit is integrated in the execution stage of the pipeline. The benefits of implementing hardware acceleration as ISEs over loosely coupled accelerators include: (i) minor changes in the instruction decoding and writeback logic; (ii) resource sharing with existing processor elements, namely the register file and load/store unit [12].

III. Methodology

This section details the methodology adopted to evaluate the power, performance, and area (PPA) characteristics of the AES algorithm implemented on the RS5 RISC-V processor, with and without the $Zkne$ ISE. Section III-A presents the set of CAD tools, the cryptographic library, and the compiler to generate the binary file for simulation. Section III-B details the synthesis and verification flow.

A. *Experimental Setup*

To support the evaluation proposed in this paper, several hardware and software tools were used:

- **ModelSim**: initial RTL simulation and functional verification for both $Zkne$ and non-$Zkne$ implementations.
- **45nm PDK**: 45nm Process Design Kit (PDK) including standard cell libraries, design rules, and other essential parameters.
- **Genus**: generates the Verilog netlist from the initial RTL description. It provides early estimates for PPA (Power, Performance, and Area) evaluation.
- **Xcelium**: used to validate the netlist generated by Genus. It incorporates Standard Delay Format (SDF) information to identify timing-related issues during post-synthesis or post-layout simulation.
- **Innovus**: used for placement, routing, and physical optimization. It generates a GDSII file and reports for the final layout. Innovus also performs Design Rule Checking (DRC) and other design validations.
- **TinyCrypt** [7]: A lightweight cryptographic library written in C, used as the reference software model for the AES implementation with the $Zkne$ extension. It was executed on a baseline RV32I model to serve as a comparative foundation for evaluating the efficiency of the $Zkne$ hardware-accelerated implementation.
- **KLayout**: GDSII viewer.

The simulation process uses a C application for testing purposes, loaded into the processor's memory through an ELF binary file obtained using the *riscv64-elf* GCC compiler [13] with compilation flags. Those flags set during compilation allowed us to have two easily customizable testing environments for $Zkne$ and the non-$Zkne$ model implementation.

B. *Synthesis and Verification*

Two distinct synthesis flows were conducted for the evaluation: one incorporating the $Zkne$ extension and one without it. In the logic synthesis phase, Process, Voltage, and Temperature (PVT) operating conditions were specified using Multi-Mode Multi-Corner (MMMC) scripts. These scripts imposed constraints targeting two distinct cell libraries: one corresponding to a slow corner scenario at 0.9V and 125°C, and the other representing a fast corner at 1.1V and 0°C. Each library was evaluated under two clock frequencies: 100 and 250 MHz. Table I presents the adopted frequencies and corners.

TABLE I
Setup in MMMC with Process Corners and Frequencies.

Analysis View	Process Corner	Frequency
AV1	Slow corner	250 MHz
AV2	Fast corner	250 MHz
AV3	Slow corner	100 MHz
AV4	Fast corner	100 MHz

Within the synthesis script, several configuration variables were set. The `syn_global_effort` variable was assigned a value of `high`, enabling more aggressive optimization strategies to improve the overall quality of results. Physical Layout Estimation (PLE) was also employed, and the `use_scan_seqs_for_non_dft` variable was set to `false` as the design does not include scan chains.

After the initial synthesis, *Xcelium* was used to simulate the newly generated netlist and SDF files, generating a Value Change Dump (VCD). A few timing violations were detected, which were traced to inconsistencies within the PDK after thorough testing and analysis. The issue was resolved by turning off clock-gating optimization during synthesis.

As with logic synthesis, the initial step in the physical synthesis flow involves configuring all necessary file paths and environment variables. A key addition in this stage is identifying critical elements defined in the LEF file provided by the PDK, such as metal layers, fill cells, and power nets. Following the design initialization on Innovus, we set the initial die area and the power ring, followed by the initial placement of the design and I/O assignment.

An important step was the generation of updated SDF files used in the final annotated simulation run with *Xcelium*. Several performance monitoring registers from the $Zicntr$ instruction set were read for evaluation purposes for this simulation. These were read using a slightly modified C library for CSR access routines, publicly available on *Github* [14]. The C application on the testbench was modified to use the `mcountinhibit` register to enable accurate readings, which temporarily halts performance counter increments. This allowed precise measurement of relevant metrics at strategically chosen points - such as before and after AES operations. Performance counters related to instruction types and cycle counts were important for the analysis. For reference, these measurements were done before and after each AES operation, allowing us to get data about its performance. As $Zkne$ instructions are entirely combinational, needing only one clock cycle for execution, only the *KeySchedule* and overall performance of the AES algorithm were deemed useful to measure.

IV. Evaluation

This section evaluates both $Zkne$ and reference implementations of the AES algorithm based on results from physical synthesis and post-layout annotated simulations. The analysis focuses on performance, power, area (PPA) characteristics, and algorithm execution-level metrics.

As described in Section III-B, four distinct Analysis Views (AV1–AV4) were defined and synthesized using MMMC methodology. These views allow performance assessment under different PVT conditions and operating frequencies.

A. Power Evaluation

Given that the primary architectural difference between $Zkne$ and reference implementations lies in including dedicated AES instruction modules, the overall static power consumption remains comparable across both configurations. However, the $Zkne$ implementation exhibits marginally higher internal power and switching activity, primarily due to the additional logic required for AES-specific operations.

As shown in Table II, AV1 and AV2 report approximately 2.5× higher total power consumption than AV3 and AV4. This is expected, as AV1 and AV2 operate at 250 MHz, while AV3 and AV4 operate at 100 MHz. Furthermore, fast-operating corner cells exhibit roughly 1.67× higher power consumption than their slow-operating corner counterparts.

While further power consumption reductions could be achieved through techniques such as clock gating, experimental results indicated that such optimizations introduced compatibility issues with the PDK used in this work.

B. Performance Evaluation

Post-synthesis simulation with SDF back-annotation yielded the results summarized in Tables III and V, while slack times obtained from timing analysis are presented in Table IV. It is important to note that the `mtime` CSR in RISC-V records real-time counter (RTC) ticks and increments with every clock cycle. Since it is not inhibited by the `mcountinhibit` CSR, it is not suitable for precisely benchmarking isolated instructions but remains valuable for measuring total execution time in clock cycles.

Table III highlights the substantial performance cost associated with the *MixColumns* operation, which requires over 700 clock cycles due to its underlying GF(2^8) matrix multiplication. While less computationally intensive, other AES operations also contribute significantly to the total execution time.

TABLE II
Power (mW) comparison between $Zkne$ and non-$Zkne$ implementations.

Config	AV	Total Power	Internal Power	Switching	Leakage
$Zkne$	AV1	4.3917	3.1805	1.2052	0.005989
	AV2	7.3422	5.2726	2.0473	0.022345
	AV3	1.7628	1.2732	0.4836	0.005989
	AV4	2.9543	2.1106	0.8213	0.022345
non-$Zkne$	AV1	4.3704	3.1760	1.1885	0.005876
	AV2	7.3094	5.2678	2.0196	0.021995
	AV3	1.7537	1.2714	0.4765	0.005876
	AV4	2.9402	2.1087	0.8095	0.021995

TABLE III
Execution statistics for AES submodules in non-$Zkne$.

Metric	SUBBYTES	SHIFTROWS	MIXCOLUMNS	ADDROUNDKEY
nop_counter	47	28	185	0
logic_counter	0	2	166	32
addsub_counter	37	23	86	3
shift_counter	0	0	64	24
branch_counter	16	7	7	0
jump_counter	0	4	68	0
load_counter	32	22	102	8
store_counter	17	22	46	4
luislt_counter	1	0	0	2
cycle_counter	151	109	**725**	74
time_counter	11266	11300	12272	11495

TABLE IV
Slack Times for Different Analysis Views.

Config	Analysis View	Slack Time (ns)
non-$Zkne$	AV1	0.179
	AV2	2.133
	AV3	5.179
	AV4	7.133
$Zkne$	AV1	0.145
	AV2	2.129
	AV3	5.145
	AV4	7.129

From Table V and Figure 3, it is evident that the hardware-accelerated $Zkne$ implementation significantly improves performance. The number of clock cycles required to complete AES encryption and key scheduling is reduced by 9.9× and 5.5×, respectively. Additionally, instruction-level statistics show a noticeable decrease in arithmetic, logic, shift, and load operations, indicating more efficient execution due to hardware specialization.

TABLE V
EXECUTION STATISTICS FOR AES AND KEYSCHEDULE.

Config	Counter	AES	KeySchedule
non-$Zkne$	nop_counter	2785	123
	logic_counter	2153	342
	addsub_counter	2057	538
	shift_counter	1084	242
	branch_counter	510	76
	jump_counter	683	3
	load_counter	1563	71
	store_counter	816	45
	luislt_counter	18	2
	time_counter	99910	11879
$Zkne$	nop_counter	**264**	**40**
	logic_counter	**54**	**50**
	addsub_counter	**167**	**39**
	shift_counter	**20**	**20**
	branch_counter	**127**	**10**
	jump_counter	**18**	**2**
	load_counter	**173**	**14**
	store_counter	**141**	**44**
	luislt_counter	**10**	**1**
	time_counter	**28418**	**9287**

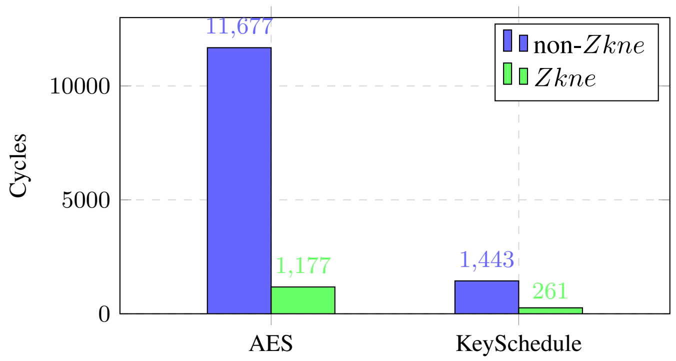

Fig. 3. Cycle count comparison for AES and KeySchedule with and without $Zkne$.

C. Area Evaluation

Table VI presents the area results. The integration of the AES unit in the $Zkne$ design introduces a marginal increase in total area, with the $Zkne$ configuration occupying 49,843.76 μm^2 compared to 49,021.60 μm^2 in the reference baseline, a change of approximately 1.68%. The additional logic required for hardware-accelerated AES operations is concentrated within the *aes_unit* module, which contributes 725.04 μm^2 and includes 381 standard cell instances. This module is absent in the non-$Zkne$ implementation.

Across the rest of the processor pipeline, only minor differences are observed. The *execute* stage in $Zkne$ contains 6114 instances and occupies 13,917.35 μm^2, compared to 5578 instances and 13,027.12 μm^2 in the reference design. This increase aligns with the integration of AES-specific datapaths and control logic. Other components such as the *CSRBank*, *RegBank*, and *decoder* show negligible variation between configurations, indicating that the $Zkne$ extension does not introduce significant overhead to the general control or register infrastructure.

These results confirm that while $Zkne$ introduces additional area due to the AES functional unit, its impact remains minimal. This trade-off is favorable when considering the substantial performance benefits described in earlier sections.

TABLE VI
CELL INSTANCE COUNT AND AREA FOR $Zkne$ AND NON-$Zkne$.

Config	Module	Inst. Count	Total Area
$Zkne$	RS5	17959	**49843.764**
	CSRBank	5332	15634.872
	RegBank	3550	11675.880
	decoder	1253	3510.972
	execute	6114	13917.348
	aes_unit	381	725.040
	fetch	1416	4265.082
	retire	197	544.464
non-$Zkne$	RS5	17494	**49021.596**
	CSRBank	5372	15671.466
	RegBank	3600	11774.376
	decoder	1229	3417.948
	execute	5578	13027.122
	fetch	1423	4264.056
	retire	207	571.824

D. Energy Evaluation

Despite the slight variations observed in internal and switching power between the $Zkne$ and non-$Zkne$ implementations (as reported in Table II), the overall power consumption remains virtually equivalent across configurations. This observation is particularly relevant when considering system-level efficiency. Given that the $Zkne$ implementation reduces the AES encryption execution time by approximately 10× (Figure 3), the energy required per encryption task is reduced.

From a system perspective, where energy is the product of power and time, the nearly identical power figures imply that the accelerated $Zkne$ configuration performs the encryption task with up to 10× less energy. This substantial improvement highlights the effectiveness of instruction set extensions for cryptographic acceleration, especially in energy-constrained embedded systems.

V. CONCLUSION

This paper evaluated $Zkne$ and non-$Zkne$ AES implementations on the RS5 processor using physical synthesis and post-layout simulation results across multiple process corners and operating frequencies. The results confirm the effectiveness of the $Zkne$ hardware acceleration approach. Although the $Zkne$ configuration introduced a marginal area increase of approximately 1.68% and negligible static power differences, it achieved significant performance gains. AES encryption and key scheduling execution times were reduced by 9.9× and 5.5×, respectively, and instruction count decreased.

Notably, given the comparable power consumption between configurations, the reduced execution time directly translates to lower energy per encryption task—up to 10× less. This outcome reinforces the suitability of $Zkne$ for energy-constrained embedded systems, highlighting a favorable performance–energy–area trade-off. Overall, the findings validate the $Zkne$ extension as an efficient and practical solution for accelerating AES workloads in RISC-V-based processors.

Future work includes the integration of additional cryptographic accelerators, such as SHA-based extensions, within the RS5 processor. Security analysis against side-channel attacks, particularly those exploiting power and timing variations, may be investigated to strengthen the proposed hardware extensions for secure embedded applications.

Acknowledgments

This work was financed in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), 305621/2024-6; Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), 23/2551-0002200-1.

References

[1] NIST, "Advanced Encryption Standard (AES)," National Institute of Standards and Technology, Tech. Rep., 2001, 38p., https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.

[2] A. Waterman, "Design of the RISC-V Instruction Set Architecture," EECS Department, University of California, Berkeley, Tech. Rep., 2016, 117p., http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.html.

[3] B. Marshall, D. Page, and T. H. Pham, "Implementing the Draft RISC-V Scalar Cryptography Extensions," in *HASP*, 2020, pp. 1–9, https://doi.org/10.1145/3458903.3458904.

[4] RISC-V Foundation, "RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions, Document Version v1.0.1," 2022, https://github.com/riscv/riscv-crypto/releases/tag/v1.0.1-scalar, February 2024.

[5] W. A. Nunes, A. E. Dal Zotto, C. d. S. Borges, and F. G. Moraes, "RS5: An Integrated Hardware and Software Ecosystem for RISC-V Embedded Systems," in *IEEE Latin American Symposium on Circuits and Systems (LASCAS)*, 2024, pp. 1–5, https://doi.org/10.1109/LASCAS60203.2024.10506171.

[6] A. Sajadi, N. Zidaric, T. Stefanov, and N. Mentens, "A systematic comparison of side-channel countermeasures for risc-v-based socs," in *NorCAS*, 2024, pp. 1–7, https://doi.org/10.1109/NorCAS64408.2024.10752477.

[7] Intel, "TinyCrypt Cryptographic Library," 2017, https://github.com/intel/tinycrypt, February 2024.

[8] SiFive, Inc, *SiFive Interrupt Cookbook, Version 1.2*, 2020, https://www.starfivetech.com/uploads/sifive-interrupt-cookbook-v1p2.pdf.

[9] "RISC-V Ratified Specifications," https://riscv.org/specifications/ratified/, accessed: April 28, 2025.

[10] H. Cheng, J. Großschädl, B. Marshall, D. Page, and T. Pham, "Risc-v instruction set extensions for lightweight symmetric cryptography," in *NIST Lightweight Cryptography Workshop 2022*, 2022. [Online]. Available: https://csrc.nist.gov/csrc/media/Events/2022/lightweight-cryptography-workshop-2022/documents/papers/risc-v-instruction-set-extensions-for-lightweight-symmetric-cryptography.pdf

[11] C. Gewehr and F. G. Moraes, "Improving the Efficiency of Cryptography Algorithms on Resource-Constrained Embedded Systems via RISC-V Instruction Set Extensions," in *SBCCI*, 2023, pp. 1–6, https://doi.org/10.1109/SBCCI60457.2023.10261964.

[12] C. Gewehr, N. Moura, L. Luza, E. Bernardon, N. Calazans, R. Garibotti, and F. G. Moraes, "Hardware Acceleration of Authenticated Encryption with Associated Data via RISC-V Instruction Set Extensions in Low Power Embedded Systems," in *LASCAS*, 2024, pp. 1–5, https://ieeexplore.ieee.org/document/10506132.

[13] GNU Compiler Collection (GCC), *GCC RISC-V Options*, 2024, accessed: April 28, 2025. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html

[14] F. EmbedDev, "Risc-V CSR Access Routines," 2020, https://github.com/five-embeddev/riscv-csr-access.