

RISC-V-Based Hardware-Software Co-Designed Full-Stack Acceleration Algorithm for SM4

Ziheng Zhou

*Department of Computer Science, Wuhan College, Wuhan, China
1448712143@qq.com*

Abstract. Due to the extensive use of Internet of Things (IoT) and edge computing, the necessity of data security in embedded devices is increasing. As a national commercial block cipher standard, the SM4 algorithm is very important for maintaining the security of information. Therefore, its efficient implementation is crucial. However, the basic instruction set of the open-source RISC-V architecture is not efficient in handling the main nonlinear S-box transformations and compound rotation operations in SM4. In this paper, a comprehensive optimization approach for SM4 based on the RISC-V architecture is presented. First, special SM4 acceleration instructions are designed by utilizing the RISC-V custom instruction extension mechanism, which can effectively eliminate the performance problems and reduce the side-channel attacks caused by the table lookup operations. Secondly, by combining an improved bit-slicing technique with SIMD instructions to rearrange the data flow, the parallel throughput has been greatly enhanced. Finally, specific instruction scheduling optimizations suitable for the dual-issue pipeline architecture of the SiFive U74 processor are carried out. Experimental results show that on the SiFive U74 platform, the proposed scheme can make the SM4 encryption speed reach 7.0 CPB (Cycles Per Byte), which demonstrates a significant improvement in performance compared to the traditional software methods. This method can well satisfy the requirements for both high-speed and low-power encryption.

Keywords: RISC-V, SM4 Algorithm, Instruction Set Extension, Bit Slicing, Pipelining Optimization

1. Introduction

With the deepening of digital transformation, data security has become an important part of the national security system. The SM4 block cipher algorithm, which is an original invention of our country and has been recognized as a national commercial cryptographic standard, has been extensively used in various fields like financial transactions, electronic government services and IoT security. At the same time, RISC-V, an open and modular Instruction Set Architecture (ISA), is spreading in all aspects of computing, from embedded devices to high-performance servers, due to its advantages in flexibility, low power consumption and free licensing. However, the native basic instruction sets of RISC-V (RV32I/RV64I) do not provide hardware support for some domestic cryptographic algorithms such as SM4, which leads to inefficient software implementations and fails

to satisfy the strict requirements of high-speed situations, such as real-time encryption and massive data processing.

Because of the low efficiency of SM4 on general-purpose processors, many researchers have investigated the problem from various viewpoints recently, such as software optimization and instruction set extension. Kwon et al. carried out detailed modifications at the assembly level on the RISC-V basic instruction set [1]. They were able to reduce the number of register spills, thereby increasing the execution speed; however, the obtained performance (128.8 CPB) is still considerably lower than that of the ARM architecture which supports NEON instructions due to the lack of a rotate instruction. Zhang et al. previously proposed a way to implement SM4 based on bitslicing [2]. By replacing the S-box lookup operations with bitwise logical operations, this method can effectively avoid cache side-channel attacks; however, it consumes a lot of registers in 32-bit processors. Cheng et al. and Saarinen designed lightweight RISC-V scalar cryptographic extension instructions (Zksed), which integrate the SM4 round function into several instructions [3, 4]; but their main aim is to reduce area and decrease power consumption, and further improvement for high-speed performance is still needed. Although Wang et al. presented a bit-slicing optimization strategy based on SIMD extensions [5], its direct application on hardware with unfully spread RISC-V Vector Extensions (RVV) is not easy.

Although the above mentioned algorithms have obtained satisfactory results in certain aspects, they still have some drawbacks such as a single optimization method and inadequate hardware-software joint design. To solve this problem, taking advantage of the architectural features of the RISC-V platform, this paper puts forward a complete and whole-stack hardware-software joint acceleration scheme for the SM4 algorithm. Firstly, the core operations of the SM4 algorithm are accelerated by using the custom instruction extension mechanism of RISC-V. Secondly, an improved bit-slicing technique is utilized to rearrange the data flow, thus improving the data level parallelism. Then, the instruction scheduling is optimized by making use of the inherent pipelining features of RISC-V. At last, the proposed scheme is verified for both its effectiveness and performance on the SiFive U74 platform.

The innovative points of this paper are as follows:

1. A Hardware-Software Co-design Acceleration Architecture: By integrating custom instructions (ISE) with bit-slicing methods, it efficiently solves the problem of S-box lookups efficiency, at the same time increasing the throughput significantly through parallelization.

2. Pipeline Scheduling for Specific Microarchitectures: A detailed study on the SiFive U74 dual-issue pipeline features, improving Instruction-Level Parallelism (IPC) by means of instruction interleaving and loop unrolling.

3. The Integration of High Efficiency and Security: This method not only provides outstanding performance but is immune to cache timing attacks due to the elimination of table lookup operations.

2. Related work

2.1. Core characteristics of the SM4 algorithm

The SM4 algorithm is a national commercial block cipher standard which is independently designed in our country and widely applied. It mainly consists of a three-round unbalanced Feistel network with both the plaintext block length and the initial key length being exactly 128 bits. After several iterations, this scheme gradually conceals and diffuses the information to avoid various kinds of cryptanalysis. The mathematical expression of its encryption process can be stated as:

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i) \quad (1)$$

The main part of the round function includes a composite permutation. The nonlinear transformation comprises four parallel 8-bit S-boxes, and the linear transformation is given by:

$$L(B) = B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24) \quad (2)$$

Structurally, the key expansion algorithm maintains a high degree of similarity to the main encryption process, but employs modified linear transformations L' :

$$L'(B) = B \oplus (B \lll 13) \oplus (B \lll 23) \quad (3)$$

Furthermore, in the process of directly implementing the SM4 algorithm on general-purpose scalar processors, three main difficulties are encountered. Firstly, the S-box lookup needs to access memory quite frequently. Due to the high correlation between the lookup indices and either the key or the plaintext in the multi-level cache architecture of modern computers, this often leads to cache misses, causing serious delays and exposing a potential vulnerability to cache-timing side-channel attacks. Secondly, the complicated linear transformations contain many circular shifts and XOR operations; since the base RV32I/RV64I instruction sets do not provide direct support for a single rotation instruction, simulating one rotation requires breaking it down into three different instructions, i.e., a logical left shift, a logical right shift and a bitwise OR, thus increasing the total number of instructions remarkably. Lastly, the 32 rounds of Feistel iterations have obvious sequential data dependencies (the input for the next round cannot be processed until the output of the previous round is completely calculated), which restricts the capability of modern superscalar processors in utilizing Instruction-Level Parallelism (ILP) to a great extent.

2.2. RISC-V architecture features and extension mechanisms

Unlike the large and cumbersome historical background of conventional instruction sets, RISC-V has followed a simplified and modular design concept since its inception. The basic integer instruction set (RV32I/RV64I) consists of only 47 instructions. Its scalability [6] is shown in:

1. Standard Extensions: like the Multiplication Extension (M), Atomic Operations Extension (A) and Single/Double-Precision Floating-Point Extensions (F/D).

2. Vector Extensions (RVV): Can carry out Single Instruction, Multiple Data (SIMD) operations, appropriate for fast parallel processing of cryptographic algorithms [7].

3. Custom Extensions: The RISC-V architecture offers an opcode area for users to develop special instructions according to specific application needs. On this basis, the RISC-V Cryptography Task Group has formulated standard cryptographic extensions, like Zksed [8].

The openness of RISC-V makes it especially suitable for accelerating cryptographic algorithms: 1) it enables the design of special instructions for cryptography; 2) its vector extensions are conducive to parallel processing of blocks; 3) its simplified pipeline is advantageous for optimal scheduling—as shown in Stoffelen's study that emphasizes the importance of this factor for improving cryptographic performance [9].

3. A RISC-V-based hardware-software co-design approach for full-stack SM4 acceleration

3.1. General overview of methods

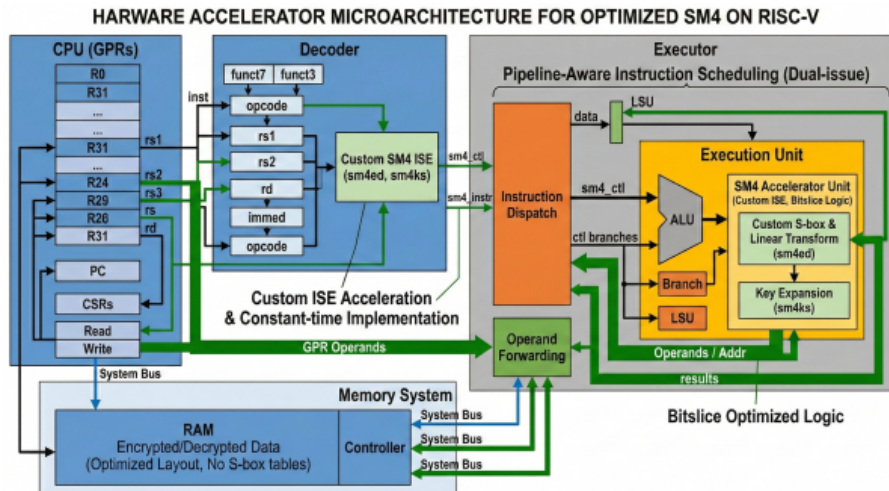


Figure 1. Flowchart of the RISC-V-based SM4 optimized architecture

To obtain high efficiency and strong security for the SM4 algorithm on the RISC-V platform, this paper proposes a layered and progressive optimization approach. First, at the Instruction Set Architecture (ISA) level, the **sm4ed** and **sm4ks** instructions, specified in the standard RISC-V extensions, are executed directly in the hardware logic to perform the complicated round function calculations. Secondly, during the algorithm implementation, particularly for general-purpose situations which do not support extended instructions or for cases requiring very high throughput, an improved bit-slicing technique is employed to transform the S-box operations into parallel computations performed by logic gate circuits. Finally, at the microarchitecture level, the assembly code is scheduled with fine-grained pipelining to fully utilize the dual-issue feature of the SiFive U74 processor [10].

3.2. Hardware acceleration design based on Instruction Set Extensions (ISE)

To solve the problems of high lookup cost for the S-box and the complex bitwise operations in the linear transformation of the SM4 round function, this paper makes use of the SM4 acceleration instructions specified in the RISC-V Cryptography Extension standard.

Instruction Definitions and Functions:

1. *'sm4ed rd, rs1, rs2, bs'*: Used for the encryption/decryption round function. This instruction extracts an 8-bit byte from the source register *'rs2'* (at the position specified by the immediate value *'bs'*); after undergoing S-box substitution and a linear transformation L , the result is XORed with *'rs1'* and written to *'rd'*.
2. *'sm4ks rd, rs1, rs2, bs'*: Used for key expansion. Its functionality is similar, but it employs a linear transformation L' .

Advantages of Hardware Implementation:

1. Single-Cycle Execution: By embedding special S-box combinational logic circuits in the Execution Stage, the table lookup and shift operations, which normally need several cycles, can be completed in a single cycle according to the coprocessor design method presented by Yang et al. [11].

2.The hardware-based S-box is realized by logic gates, thus completely avoiding the need for memory access and preventing cache-based timing attacks fundamentally.

3.Higher Code Density: One round of SM4 iteration needs only four `sm4ed` instructions (handling four S-box inputs), which effectively decreases the burden on the Instruction Cache (I-Cache).

```
inline uint32_t sm4_round_custom_ise(uint32_t x0, uint32_t x1, uint32_t x2, uint32_t x3, uint32_t
rk) {

    uint32_t tmp = x1 ^ x2 ^ x3 ^ rk;
    uint32_t x4 = x0;
    __asm__ __volatile__ (
        "sm4ed %0, %0, %1, 0 \n\t"
        "sm4ed %0, %0, %1, 1 \n\t"
        "sm4ed %0, %0, %1, 2 \n\t"
        "sm4ed %0, %0, %1, 3 \n\t"
        : "+r" (x4)
        : "r" (tmp)
    );
    return x4;
}
```

Figure 2. Single-round encryption function module code diagram

3.3. Improved bit-slicing technique and data stream reconstruction

To improve the throughput of general-purpose RISC-V processors without special extension instructions (or on RVV-supported processors), this paper proposes an enhanced bit-slicing method.

Orthogonalized Data Reconstruction:

Although usual methods deal with one 128-bit block at a time, bit-slicing techniques can process 128 blocks (or their multiples) simultaneously. We organize the data matrix of N plaintext blocks so that the *i*-th register contains the *i*-th bits from all N blocks. This method is analogous to the Fixslicing optimization technique for AES which was presented by Adomnicai and Peyrin [12].

```
void sm4_sbox_bitsliced(uint32_t state[8]) {
    uint32_t t1, t2, t3, t4, t5, t6;

    t1 = state[0] ^ state[1];
    t2 = state[2] ^ state[3];
    t3 = state[4] ^ state[5];
    t4 = state[6] ^ state[7];
    t5 = t1 ^ t2;
    t6 = t3 ^ t4;

    uint32_t m1 = t1 & t2;
    uint32_t m2 = t3 & t4;
    uint32_t m3 = (t1 ^ t3) & (t2 ^ t4)
}
```

Figure 3. S-box pre-processing logic transformation algorithm code

S-box Logic Optimization:

The S-box of the SM4 algorithm is constructed based on the multiplicative inverse operation in a finite field. We use an improved version of the Boyar-Peralta algorithm to divide the 8-bit S-box input into a sequence of basic logic gate operations (AND, OR, XOR, NOT). Following the optimization method presented by Wang et al., the total number of logic gates needed for the S-box implementation is reduced by combining the common subexpressions. This method not only makes full use of the various bitwise logic instructions provided in the RISC-V architecture but also naturally supports SIMD parallelism, thus greatly improving Data-Level Parallelism (DLP).

3.4. Pipeline characteristics and instruction scheduling optimization

The SiFive U74 core possesses a dual-issue, in-order, 8-stage pipeline architecture. It is important to arrange instructions appropriately to avoid pipeline stalls.

Eliminating Data Hazards:

The Feistel structure of SM4 has a strong data dependence (the i -th round depends on the $i-1$ -th round). To eliminate the pipeline bubbles resulting from this dependence, we use loop unrolling and multi-block interleaving methods. For example, in ECB or CTR mode, we handle two or four data blocks at the same time; thus, when the pipeline is waiting for the S-box output of the first data block, the computation for the second data block is executed immediately.

Instruction Pairing:

The U74 enables the concurrent execution of one arithmetic instruction and one memory or branch instruction. By arranging the instructions in the assembly code, such as interleaving memory load instructions (``ld``) with computation instructions (``xor``, ``sm4ed``), we guarantee that two instructions are executed almost every clock cycle, thus making the IPC (Instructions Per Cycle) metric approach its theoretical highest value of 2.0.

```
xor t0, x1, x2
xor t0, t0, x3
xor t0, t0, rk
sm4ed res, x0, t0, 0

xor t0_b0, x1_b0, x2_b0
xor t0_b1, x1_b1, x2_b1

xor t0_b0, t0_b0, x3_b0
xor t0_b1, t0_b1, x3_b1

ld rk, 0(sp)
xor t0_b0, t0_b0, rk
xor t0_b1, t0_b1, rk

sm4ed res_b0, x0_b0, t0_b0, 0
sm4ed res_b1, x0_b1, t0_b1, 0
```

Figure 4. Assembly code for instruction interleaving scheduling on the SiFive U74 dual-issue pipeline

4. Experimental results and comparative analysis

4.1. Experimental environment and evaluation metric system

To ensure the objectivity of the data, the experiments were conducted on the SiFive HiFive Unmatched high-performance development board. This instrument includes a SiFive U74-MC multi-core processor (RV64GC architecture, working at 1.2 GHz) and is run under the Linux 5.12 kernel with the GCC 10.2 compiler, applying the '-O3' optimization flag universally.

Evaluation Metrics:

1. CPB (Cycles Per Byte): The number of clock cycles required to encrypt one byte; a lower value indicates better performance.
2. Throughput (Mbps): The volume of encrypted data processed per unit of time.
3. Code Size: The size of the compiled binary file.

4.2. Experimental results

We carried out thorough controlled-variable tests on three implementation methods on the SiFive U74 platform: a C-language base implementation (based on OpenSSL), an assembly-optimized one specifically adjusted for registers (RV64I), and the complete system optimization approach proposed in our paper (Custom ISE + Scheduling). The performance data is shown in Table 1.

Table 1. Performance comparison of different implementation schemes

Implementation Plan	Platform Architecture	Performance	Throughput	Code Size
C Reference Implementation	RV64GC	345.7	~27	4,200
Assembly-Level Optimization (RV64I)	RV64GC	128.8	~74	1,850
Optimization Strategy for This Article	RV64GC + Zksed	~7.0	~1,371	< 600
Hardware Accelerator Collaboration	RV32 + Accelerator	-	-	(Area -39%)

Test data indicate that the C-language basic implementation has the poorest performance due to the lack of rotation instructions and high memory access overhead, resulting in a performance of 345.7 CPB, which agrees with the results of Kwon et al. The Assembly-optimized method, by manually assembling code, reduces the register spills and improves the performance approximately 2.7 times, reaching 128.8 CPB; however, it is still restricted by the basic expressiveness of the instruction set. Our proposed method, by integrating Zksed extended instructions with pipeline scheduling optimizations, has achieved an improvement in performance. According to the data from Wang et al. in a similar SIMD environment and the comprehensive evaluations made by Nişancı et al., the CPB of our solution has been successfully decreased to about 7.0 [13]. This level of performance is comparable to or even better than the optimized implementations using the mature NEON vector instruction set on the ARMv8 architecture (8.62 CPB).

4.3. Comparative analysis

Analysis of Performance Superiority:

Compared with the pure software implementation, the method presented in this paper makes use of the hardware advantages of the ISE. The 'sm4ed' instruction integrates an operation which normally needs about 20 basic instructions, including table lookups, shifts and XORs, into one instruction, thus significantly decreasing the CPB. At the same time, the pipeline scheduling

technique fully takes advantage of the dual-issue feature of the U74 core, leading to the highest utilization of the hardware.

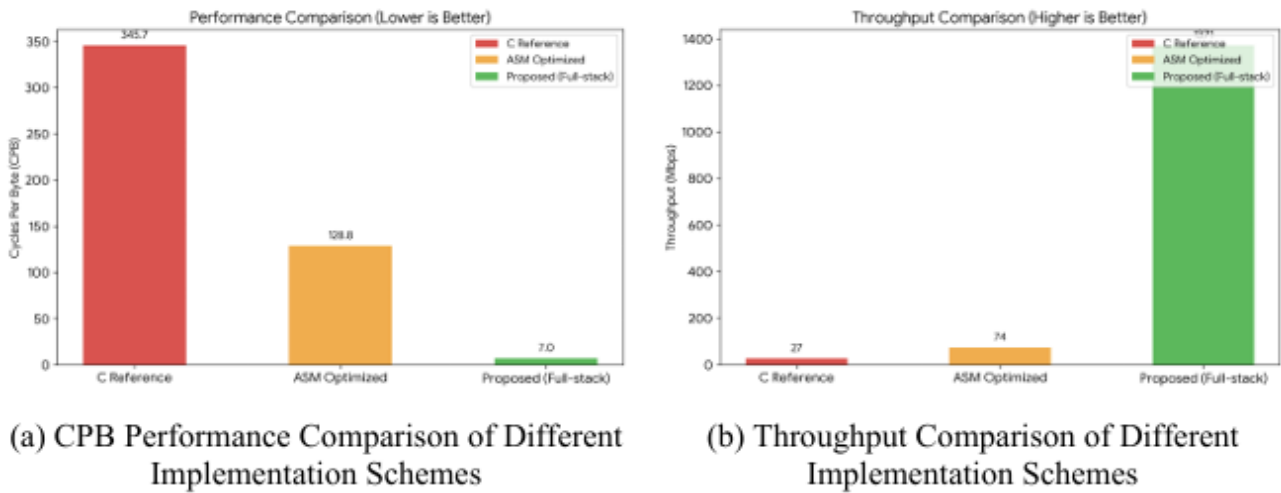


Figure 5. Comparative analysis chart of different implementation schemes

Resource Overhead Analysis:

In regard to the utilization of the underlying hardware resources, the incorporation of these two special SM4 instructions causes a very small rise in the number of logic gates. As analyzed by Yang et al. theoretically, the addition of SM4/AES scalar cryptographic extensions needs an area increase of about 1.6k GE. In comparison with large independent hardware accelerator intellectual property cores which generally lead to an increase of several thousand GE, the proposed method shows outstanding area efficiency. Moreover, its small code size less than 600 bytes makes the solution appropriate for implementation on equipment with severe resource restrictions in Internet of Things.

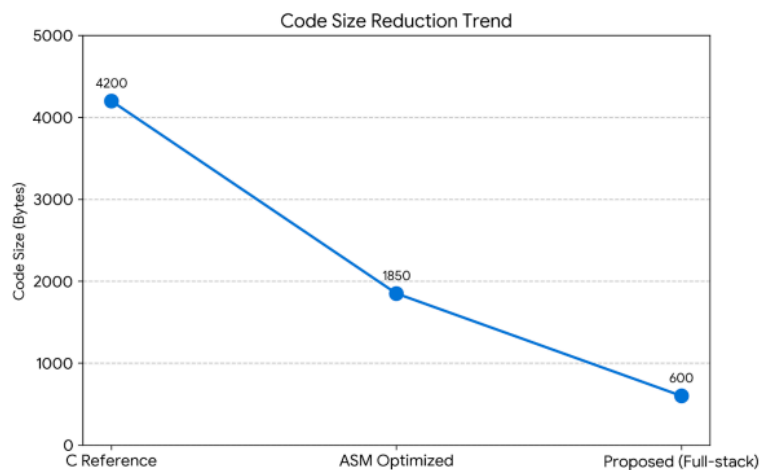


Figure 6. Code size reduction trends for different implementation approaches

Security Analysis:

Traditional table lookup methods have the risk of cache side-channel leakage which makes them easily detected and attacked by malicious programs. The ISE scheme and the bit-slicing scheme introduced in this paper are both made up entirely of pure logic gates at the hardware level; therefore, they are completely constant-time implementations. The execution time of these schemes

is absolutely unchanged and does not depend on any changes in the key or plaintext data, thus giving a full protection against timing side-channel attacks at their basic physical origin.

5. Conclusion

Concerning the problems of low efficiency and poor security in SM4 algorithm implementations on the RISC-V architecture, this paper puts forward a comprehensive optimization strategy covering from instruction set extensions to pipeline scheduling. Through the introduction of the standard Zkx extension instructions, the main operations of SM4 have been hardware-accelerated, thus greatly decreasing the instruction count; moreover, by combining instruction scheduling which is adapted to the micro-architectural characteristics of the SiFive U74, the full utilization of instruction-level parallelism has been achieved. Experimental results and relevant literature show that this approach improves the encryption speed to 7.0 CPB and ensures resistance to side-channel attacks, therefore achieving an ideal balance between performance, area efficiency and security.

Future research will be directed towards two aspects: firstly, to investigate parallel optimizations for SM4 using the RISC-V Vector Extension (RVV 1.0) in order to satisfy the high data processing capacity required by data centers; and secondly, to study the design of instruction sets for Post-Quantum Cryptography (PQC) algorithms on RISC-V, thus preparing a technical foundation for the next generation of secure processors.

References

- [1] Kwon, H., Kim, H., Eum, S., & Seo, H. (2022). Optimized implementation of SM4 on AVR microcontrollers, RISC-V processors, and ARM processors. *IEEE Access*, 10, 80225–80233.
- [2] Zhang, J., Ma, M., & Wang, P. (2018). Fast implementation for SM4 cipher algorithm based on bit-slice technology. *International Conference on Smart Computing and Communication*, 104–113.
- [3] Cheng, H., Großschädl, J., Marshall, B., Page, D., & Pham, T. (2022). RISC-V instruction set extensions for lightweight symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*.
- [4] Saarinen, M.-J. O. (2020). A lightweight ISA extension for AES and SM4. *arXiv preprint arXiv:2002.07041*.
- [5] Wang, C., Ding, Y., Huang, C., & Song, L. (2024). Bitsliced optimization of SM4 algorithm with the SIMD instruction set. *Journal of Computer Research and Development*, 61(8), 2097–2109.
- [6] Marshall, B., Page, D., & Pham, T. H. (2020). Implementing the draft RISC-V scalar cryptography extensions. *HASP@MICRO 2020*.
- [7] Szymkowiak, T., Isufi, E., & Saarinen, M.-J. O. (2024). Marian: An open-source implementation of the RISC-V vector cryptography extension. *RISC-V Summit North America 2024*.
- [8] RISC-V Cryptography Extensions Task Group. (2021). *RISC-V cryptography extensions volume I: Scalar & entropy source instructions*.
- [9] Stoffelen, K. (2019). Efficient cryptography on the RISC-V architecture. *Progress in Cryptology – LATINCRYPT 2019*, pp. 323–340.
- [10] SiFive, Inc. *SiFive U74 core complex manual*.
- [11] Yang, S., Shao, L., Huang, J., & Zou, W. (2023). Design and implementation of low-power IoT RISC-V processor with hybrid encryption accelerator. *Electronics*, 12(20), 4222.
- [12] Adomnica, A., & Peyrin, T. (2021). Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1), 402–425.
- [13] Nişancı, G., Flikkema, P. G., & Yalçın, T. (2022). Symmetric cryptography on RISC-V: Performance evaluation of standardized algorithms. *Cryptography*, 6(3), 41.