

Article

Exploiting Structural Symmetry of SM4 for an Asymmetric Hardware Architecture: Design and Open-Source Verification on the RISC-V LicheePi 4A Platform

Jianxin Wang, Zixuan Wang *, Runze Zhou, Chaoen Xiao and Lei Zhang

Department of Electronic and Communication Engineering, Beijing Electronic Science and Technology Institute, Beijing 100070, China; wangjianxin@besti.edu.cn (J.W.); 20243816@mail.besti.edu.cn (R.Z.); xce@besti.edu.cn (C.X.); zhanglei@besti.edu.cn (L.Z.)

* Correspondence: 20253811@mail.besti.edu.cn

Abstract

Reproducing SM4 (GB/T 32907-2016) hardware-accelerator results on open-source RISC-V platforms is difficult, because most published designs depend on proprietary FPGA toolchains. This paper contributes an *asymmetric dual-channel SM4 architecture* together with a fully reproducible open-source verification framework; physical on-board acceleration is not claimed and is left as future work. The architecture exploits two algorithmic symmetries of SM4—encryption and decryption differ only in round-key order, and the round transform T shares the byte-wise S-box τ with the key-expansion transform T' —but maps them onto an asymmetric workload. Bulk encryption is throughput-bound, whereas key expansion runs once per session. Accordingly, a 32-stage fully unrolled encryption pipeline (one 128-bit block per cycle in steady state) is paired with a single round function reused iteratively for the key schedule, and encryption and decryption share one datapath via round-key reversal. Because the TH1520 SoC on LicheePi 4A does not expose the Xuantie C910 RoCC port, we verify the design in three reproducible tiers on the board itself: (T1) RTL co-simulation of an `sm4_rocc` wrapper passes **1040/1040** vectors for both the standalone datapath and the full system. (T2) A pure-C reference model passes 10/10 GB/T 32907-2016 vectors on the real C910 at a measured 291.9 Mbps. (T3) A Linux illegal-instruction trap-and-emulate prototype confirms ISA and OS-level semantics. Open-source synthesis (Yosys + SkyWater Sky130) gives a measured area of 133 kGE and a switching-dominated post-synthesis power estimate of ≈ 0.28 W at 100 MHz (≈ 22 pJ/bit, ≈ 46 Gbps/W). At 100 MHz the unrolled pipeline reaches an RTL simulation-equivalent steady-state throughput of 12.8 Gbps, about $43.9\times$ the software baseline. Every reported number is reproducible with open-source tools only (Icarus Verilog, GTKWave, GCC, Yosys, Sky130 PDK).

Keywords: SM4; structural symmetry; asymmetric pipeline; RISC-V; RoCC; hardware acceleration; open-source EDA; LicheePi 4A; reproducible verification



Academic Editor: Tamas Kiss

Received: 24 May 2026

Revised: 17 June 2026

Accepted: 21 June 2026

Published: 25 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

1.1. Background

SM4 is a 128-bit block cipher with a Feistel-like structure, standardised as GB/T 32907-2016 [1] and registered into the TLS 1.3 ShangMi cipher suites via IETF RFC 8998 [2]. SM4 is deployed in IPsec/VPN gateways [2], trusted computing platforms [3], and storage-

encryption stacks following the XTS block-cipher mode [4], where its hardware throughput directly determines the quality of service of the overlying security subsystem.

Mainstream SM4 hardware implementations still rely on Xilinx or Intel/Altera FPGAs and their proprietary EDA toolchains [5], which makes the published numbers difficult for a third party to reproduce without commercial software licenses. The T-Head Xuantie C910 RV64GCV processor (T-Head Semiconductor, Alibaba Group, Hangzhou, China) [6] and its carrier the LicheePi 4A single-board computer [7] provide an inexpensive open-architecture target for cryptographic acceleration research, while their open-source software stack lets every step of the verification flow run on commodity hardware. Wang et al. [8] completed an AES-128 full-flow open-source verification on the same platform, demonstrating the engineering viability of Icarus Verilog and GTKWave for block cipher RTL simulation. However, a systematic engineering report covering SM4 hardware accelerator design, its integration with a RISC-V processor, and a reproducible verification flow has not yet appeared.

1.2. Problem Statement

This paper addresses three concrete questions:

- Q1.** The encryption path and the key-expansion path of SM4 exhibit highly asymmetric invocation frequencies and area sensitivities. Existing open RTL designs either adopt a uniform iterative structure—sacrificing throughput—or a uniform fully unrolled structure—wasting area on the low-frequency key expansion. How can one achieve a tighter throughput–area trade-off while preserving algorithmic symmetry?
- Q2.** The TH1520 SoC on LicheePi 4A does not expose the C910's RoCC interface, rendering physical insertion of a hardware coprocessor infeasible at the silicon level. Under this constraint, how can one complete end-to-end verification of custom instruction semantics, software API, and hardware protocol on the board without sacrificing engineering rigour?
- Q3.** Current public SM4 implementations are generally tied to proprietary EDA toolchains, which makes their throughput and area numbers difficult for an independent reader to reproduce. How can one build a one-command reproducible verification environment using only open-source tools, so that every reported result is bit-for-bit reproducible on commodity hardware?

We take the two structural symmetries of SM4 as our entry point: (i) encryption and decryption use identical round functions with only the round-key order reversed; (ii) the round transformation T and the key-expansion transformation T' share the same nonlinear S-box τ . Mapping these symmetries to hardware yields datapath unification and S-box reuse; mapping the invocation-frequency asymmetry yields the asymmetric pipeline structure. Figure 1 illustrates this two-dimensional mapping.

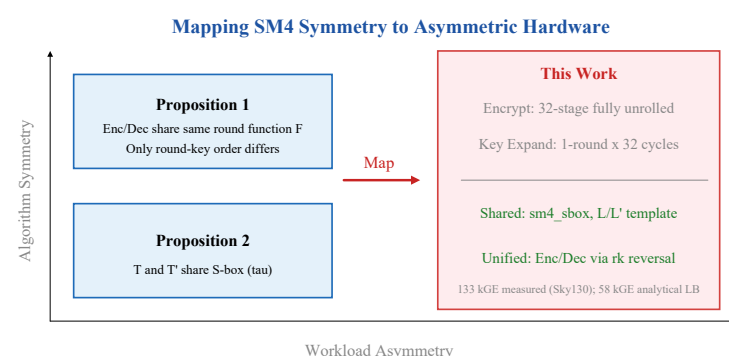


Figure 1. Motivation: two-dimensional mapping from SM4 algorithmic symmetry and workload asymmetry to the proposed asymmetric hardware architecture.

1.3. Contributions

Our contributions are explicitly graded into *verified*, *implemented but not yet measured on silicon*, and *conceptual*:

- **C1 (Verified).** Building on an open-source asymmetric dual-channel SM4 RTL implementation as our reference baseline datapath, we contribute an extended verification campaign and the methodological infrastructure needed to evaluate it end-to-end on a commodity RISC-V platform. On LicheePi 4A, a deterministic 1040-vector test-bench (16 hand-picked edge cases plus 1024 pseudo-random vectors produced by an xorshift64 PRNG with the fixed seed 0xdeadbeefcafebabe) drives both the standalone datapath (Experiment 1A) and the full RoCC-wrapped system (Experiment 1B); both report **1040/1040 pass**. T2 on the real C910 passes 10/10 GB/T 32907-2016 standard vectors with a measured software baseline throughput of 291.9 Mbps at 1.85 GHz.
- **C2 (Implemented, not yet silicon-measured).** We design the `sm4_rocc` RoCC wrapper, five custom RISC-V instructions (SM4.LDKEY/ENC.HI/ENC.LO/DEC.HI/DEC.LO), and their inline-assembly C API. The 1040-vector BFM verification (C1) confirms `funct7` decoding, `cmd/resp` handshake, and HI/LO write-back ordering. Using the open-source Yosys + Sky130 flow we additionally report a *measured* post-synthesis area (133 kGE for `sm4_top`) and a switching-dominated OpenSTA power/energy-efficiency estimate (≈ 0.28 W, ≈ 22 pJ/bit, ≈ 46 Gbps/W at 100 MHz); the on-board acceleration ratio, by contrast, awaits soft-core FPGA prototyping because the TH1520 RoCC port is closed.
- **C3 (Verified).** We provide a three-tier reproducible verification flow (RTL co-simulation/software reference model/illegal-instruction trap-and-emulate) orchestrated by a single shell driver `run_all_experiments.sh`, depending solely on open-source software (Icarus Verilog 12.0, Python 3.11, optionally Yosys + Sky130). The entire flow has been verified to run end-to-end on the LicheePi 4A target platform itself, with no x86 host required in the measurement loop.
- **C4 (Conceptual).** We abstract the above into an *algorithmic symmetry* $\rightarrow$ *workload asymmetry* $\rightarrow$ *hardware asymmetry* design pattern, which is independent of SM4 and applies to any block cipher whose forward and inverse round functions share structure (e.g., SM3 compression, SM2 modular arithmetic, and the AES key-schedule reuse pattern).

1.4. Paper Organisation

Section 2 formalises the SM4 algorithm and its structural symmetries. Section 3 surveys related work. Section 4 describes the asymmetric dual-channel architecture with resource-timing analysis. Section 5 presents the three-tier RISC-V integration scheme. Section 6 details the reproducible experimental methodology. Section 7 reports experimental results, comparisons, and threats to validity. Section 8 discusses limitations and future work. Section 9 concludes.

2. Preliminaries and Structural Symmetry

2.1. Notation

We denote by $\{0,1\}^n$ the set of n -bit strings; $\oplus$ is bitwise XOR; $B \lll k$ is a 32-bit circular left shift of word B by k positions; $\parallel$ denotes concatenation. Plaintext and ciphertext are organised as four 32-bit words: $X = (X_0, X_1, X_2, X_3)$.

2.2. SM4 Algorithm Specification

SM4 is a 32-round unbalanced Feistel block cipher with 128-bit block and key lengths [1,9]; its security against differential cryptanalysis has been established by Su et al. [10], who proved that any 5-round differential characteristic for SM4 has probability bounded by 2^{-25} , which is sufficient to thwart practical key-recovery attacks on the full 32-round cipher. Its round function F is defined as:

$$X_{i+4} = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), \quad i = 0, 1, \dots, 31. \quad (1)$$

The nonlinear transformation τ comprises four parallel S-box look-ups:

$$\tau(A) = (\text{Sbox}(a_0), \text{Sbox}(a_1), \text{Sbox}(a_2), \text{Sbox}(a_3)), \quad (2)$$

where $A = (a_0, a_1, a_2, a_3)$ is a 32-bit word split into four bytes.

The linear transformation L is:

$$L(B) = B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24). \quad (3)$$

Key expansion uses the structurally similar transformation $T' = L' \circ \tau$, where:

$$L'(B) = B \oplus (B \lll 13) \oplus (B \lll 23). \quad (4)$$

The master key MK is XORed with system parameters FK to obtain the initial state (K_0, K_1, K_2, K_3) , and 32 round keys $rk_0, \dots, rk_{31}$ are generated by iterating:

$$rk_i = K_{i+4} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i), \quad (5)$$

where CK_i are publicly specified constants [1]. The ciphertext is obtained by a final word-order reversal: $Y = (X_{35}, X_{34}, X_{33}, X_{32})$.

2.3. Two Structural Symmetries

We exploit two well-known but often engineering-overlooked symmetries of SM4.

Proposition 1 (Encryption–Decryption Symmetry). *Let $\text{Enc}(\cdot; rk_0, \dots, rk_{31})$ and $\text{Dec}(\cdot; rk_{31}, \dots, rk_0)$ denote the SM4 encryption and decryption mappings respectively. Both use exactly the same round function F ; they differ only in the ordering of round keys [1,9]. Hence a single physical encryption datapath plus a round-key reversal selector suffices for both operations.*

Proposition 2 (τ -Sharing Symmetry). *$T = L \circ \tau$ and $T' = L' \circ \tau$ share the same nonlinear layer τ , which in hardware corresponds to a 256×8 -bit ROM (the `sm4_sbox` module). L and L' differ only in their circular-shift constant sets: $C_L = \{0, 2, 10, 18, 24\}$ vs. $C_{L'} = \{0, 13, 23\}$. Both can be realised by bit-concatenation XOR without barrel shifters.*

Figure 2 annotates the SM4 dataflow, highlighting the encryption–decryption symmetry (Proposition 1) and the τ -sharing symmetry (Proposition 2).

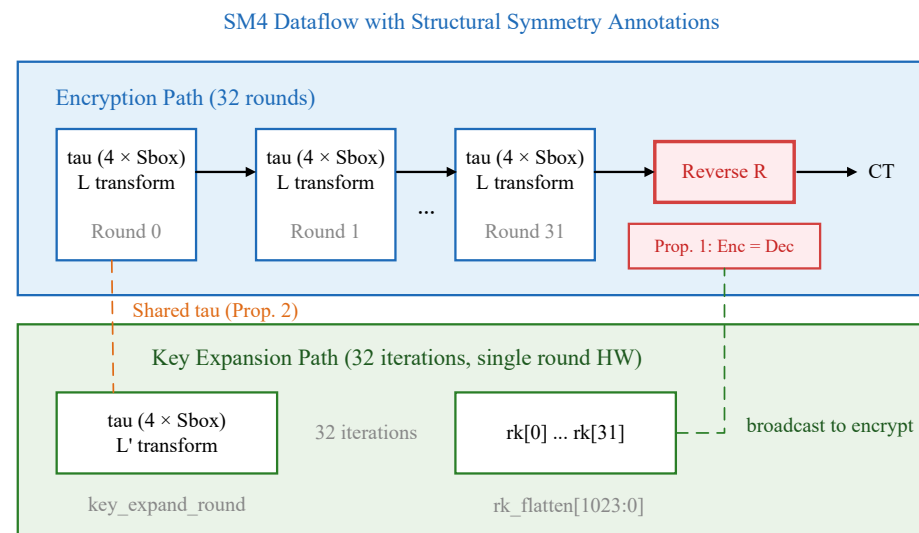


Figure 2. SM4 dataflow with structural symmetry annotations. Red: encryption–decryption symmetry (Proposition 1). Blue/green: τ -sharing symmetry (Proposition 2).

2.4. Workload Asymmetry

Key expansion is invoked once per session or per encryption volume, while encryption/decryption is triggered continuously by the data stream. In typical IPsec or disk-encryption scenarios, the invocation-frequency ratio exceeds 10^6 [4,11]. This ratio establishes the engineering basis for the asymmetric hardware mapping in Section 4. Table 1 summarises these contrasting workload characteristics.

Table 1. Workload characteristics of SM4 operations.

Operation	Invocation Freq.	TP Sens.	Area Sens.	HW Preference
Key expansion	Low (per session)	Low	High	Iterative reuse
Encryption	High (streaming)	High	Medium	Fully unrolled
Decryption	High (streaming)	High	Medium	Shared w/encryption

3. Related Work

3.1. SM4 Hardware Implementations

SM4 hardware implementations can be broadly categorised into high-throughput pipelined, compact iterative, and side-channel-resistant masked designs.

In the high-throughput direction, Abed et al. [5] reported a comprehensive FPGA-based performance evaluation of the SM4 cipher covering loop-unrolled and pipelined variants. For compact and low-area designs, Zhou et al. [12] presented compact SM4 encryption/decryption circuits that also incorporate bypass-attack resistance, and Chen and Li [13] explored a high-efficiency SM4-CCM hardware architecture targeted at IoT applications. Earlier composite-field S-box constructions for SMS4, exemplified by Bai et al. [14], established the area-optimised non-linear-layer designs widely reused in subsequent work.

On the side-channel-resistance side, Shao et al. [15] proposed a second-order threshold implementation of SM4 evaluated under formal masking-security models, and Schneider and Moradi [16] provided the TVLA methodology that is now standard for leakage assessment of block-cipher implementations including SM4.

All of the above works rely on commercial FPGA boards and proprietary EDA flows; their toolchains require paid licenses and target-specific synthesis scripts, which together make a bit-for-bit independent reproduction nontrivial for readers who only have access to open-source tooling.

Beyond standardised symmetric block ciphers such as SM4, a complementary and active line of research builds encryption schemes on the rich nonlinear dynamics of memristive chaotic neural networks. Lin et al. [17] constructed a grid multi-butterfly memristive Hopfield neural network and used its initial-boosted multi-butterfly attractors as the confusion–diffusion source for a secret-image-sharing scheme deployed on police-IoT hardware, while Ding et al. [18] proposed a hidden multiwing memristive neural network whose wing count is tunable by the memristor parameters and applied it to remote-sensing image-data security with a digital hardware demonstrator; most recently, Lin et al. [19] studied secure image privacy in the Internet-of-Vehicles (IoV) with a multiwing hyperchaotic memristive neural network, and Min, Lin et al. [20] introduced a memristive cellular neural network (MCNN) that exhibits multi-butterfly attractors; they validated its dynamics on an FPGA, and they applied it to an IoV image-encryption and secure-communication scheme. These chaos-based designs target the statistical confusion/diffusion of multimedia content and are orthogonal to the present work: we instead accelerate a *standardised, deterministically verifiable* block cipher (GB/T 32907-2016), where bit-exact conformance to published reference vectors—rather than statistical key-space and entropy metrics—is the governing correctness criterion. The two directions are nonetheless complementary, since a high-throughput SM4 datapath could serve as the standardised-cipher stage in such hybrid memristive–block-cipher pipelines.

3.2. Symmetric-Cipher Acceleration on RISC-V

The RISC-V ISA offers three standardised or community paths for symmetric cipher acceleration.

Software optimisation. Stoffelen [21] systematically analysed software implementations of AES [22] and SM4 on RISC-V and discussed ISA extension evolution. Tehrani et al. [23] proposed a tightly-coupled RISC-V execution unit for lightweight 64-bit block ciphers, providing a software-plus-hardware baseline context.

Scalar ISA extensions. Marshall et al. [24] formalised the RISC-V scalar cryptographic extensions Zkne/Zknd with AES acceleration measurements. The RISC-V Cryptography Extensions specification [25] defines Zkn* and Zks* extension families.

Vector extensions. The RISC-V Vector Cryptography Task Group published the Zvkse extension [26] providing *vsm4r* and *vsm4k* instructions for SM4; this extension is not yet implemented in commercially shipping RISC-V processors such as the Xuantie C910.

Coprocessor integration. Gomes et al. [27] proposed FAC-V, mounting an AES coprocessor through the RoCC interface on Rocket Chip and reporting significant speedup over pure software. Asanović et al. [28] described the Rocket Chip generator and its RoCC interface specification, and Lee et al. [29] integrated cryptographic accelerators into the Keystone RISC-V enclave framework on the same RoCC substrate.

3.3. Open-Source Verification of Block Cipher Hardware

Wang et al. [8] reported a full-flow open-source verification of AES-128 on LicheePi 4A using Icarus Verilog and GTKWave, confirming the engineering viability of open-source EDA for block-cipher RTL simulation but not addressing SM4 or RISC-V processor integration. To our knowledge, no prior peer-reviewed work has combined SM4 RTL design, a RISC-V RoCC-style wrapper, and an end-to-end open-source verification flow on a commodity RISC-V single-board computer in a single coherent study.

Table 2 summarises the positioning of this work relative to representative prior art.

Table 2. Differentiated positioning relative to representative prior work.

Work	Algo.	Architecture	Toolchain	RISC-V	Platform
Zhou et al. [12]	SM4	Compact + anti-bypass	Commercial	No	ASIC
Abed et al. [5]	SM4	FPGA perf. evaluation	Commercial	No	FPGA
Shao et al. [15]	SM4	2nd-order TI (SCA)	Commercial	No	ASIC
Stoffelen [21]	AES/SM4	ISA SW opt.	GCC	Yes (SW)	SiFive
FAC-V [27]	AES	RoCC coprocessor	Chipyard	Yes (RoCC)	FPGA
Wang et al. [8]	AES	9-stage pipeline	Icarus + GTK	No	LicheePi 4A
This work	SM4	32-st. + iter. + RoCC	Icarus + GCC	Yes (3-tier)	LicheePi 4A

Relative to the above, our differentiation spans three dimensions: (D1) at the algorithm level, we target SM4 rather than AES and explicitly exploit its structural symmetries; (D2) at the integration level, we provide a tiered verification method viable even when the RoCC port is not physically exposed; (D3) at the toolchain level, we insist on a fully open-source, one-command reproducible flow that any reader can execute on a commodity sub-USD-150 RISC-V single-board computer.

Architectural novelty. The prior SM4 hardware designs above commit to a *single uniform structure* that is applied identically to both the encryption datapath and the key-expansion datapath: high-throughput designs [5] unroll or pipeline *both*, paying area for a key schedule that runs only once per session, whereas compact designs [12,13] iterate *both*, throttling bulk-encryption throughput to save area. The novelty of this work is not a faster S-box or a smaller round function—we reuse established composite-field/ROM S-box constructions—but the *asymmetric mapping itself*: we deliberately assign *different* micro-architectures to the two channels according to their invocation frequency (a 32-stage fully unrolled encryption pipeline for the throughput-critical channel, a single iteratively-reused round for the once-per-session key schedule), while keeping the two channels symmetric at the operator level by sharing the same S-box τ (Proposition 2) and folding encryption and decryption onto one datapath via round-key reversal. This algorithmic-symmetry-to-workload-asymmetry mapping is what lets the design sit at a tighter point of the area-throughput trade-off than any single uniform structure, and—unlike the fixed-function ASIC/FPGA accelerators above—it is delivered as a RISC-V RoCC-style coprocessor with a fully open-source, board-level reproducible verification flow.

4. Asymmetric Dual-Channel Hardware Architecture

4.1. Design Space and Design Choice

Let $N = 32$ denote the SM4 round count, f the target clock frequency, and $b = 128$ the block width in bits. Two extreme architectures bracket the design space:

- **(A) Single-round iterative reuse.** Instantiate one round function; encryption requires N clock cycles per block. Steady-state throughput: $T_A = b \cdot f / N$. Area: $\approx A_{\text{round}} + A_{\text{reg}}$.
- **(B) Fully unrolled pipeline.** Instantiate N round functions in cascade. Steady-state throughput: $T_B = b \cdot f$. Area: $\approx N \cdot A_{\text{round}} + (N+1) \cdot A_{\text{reg}}$.

If both encryption and key expansion use the same architecture, either throughput is reduced N -fold (A) or N -fold hardware resources are wasted on the low-frequency key expansion (B).

Our design separates the two paths: the encryption path uses (B) and the key-expansion path uses (A), directly matching the workload profile of Table 1. The resulting top-level architecture is shown in Figure 3.

4.2. Top-Level Interface

The top-level module `sm4_top` exposes the interface summarised in Listing 1. A 1024-bit internal round-key bus broadcasts all 32 round keys from the key-expansion channel to the encryption channel in a single cycle, eliminating any per-round handshake between the two sub-modules.

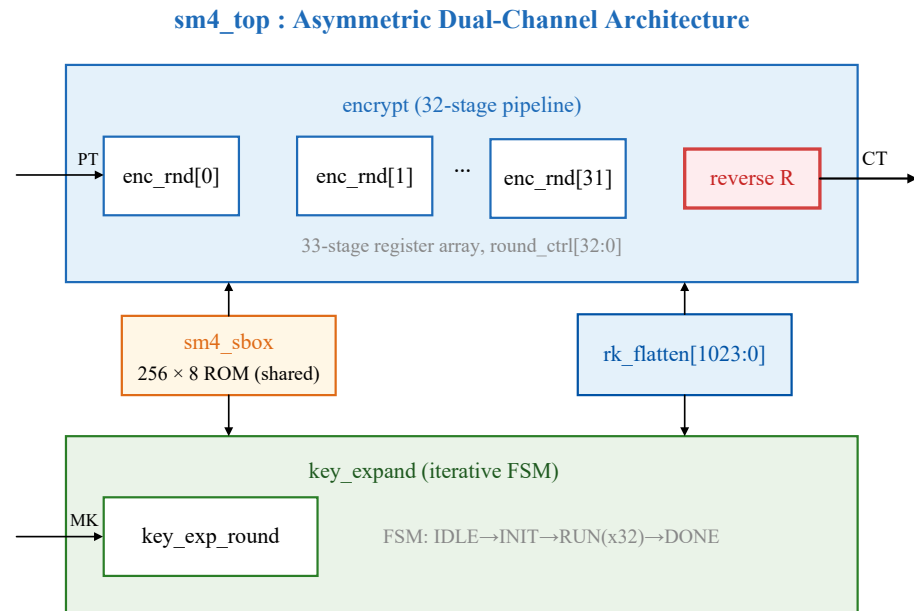


Figure 3. `sm4_top`: asymmetric dual-channel architecture overview showing the encrypt and `key_expand` sub-modules and their interconnection.

Listing 1. Top-level module `sm4_top` interface (signal list, schematic).

```

module sm4_top (
    input          clk, rst_n,
    input          [127:0] master_key,    // MK
    input          key_load,              // 1-cycle pulse, latches MK
    input          [127:0] data_in,        // plaintext (or ciphertext for
    decrypt)
    input          in_valid,              // 1-cycle pulse per input block
    output         [127:0] data_out,        // ciphertext (or plaintext)
    output         out_valid,              // 1-cycle pulse per output block
    output         busy                   // OR of internal busy flags
);
// 1024-bit round-key bus: 32 x 32-bit, broadcast in one cycle.
wire [1023:0] round_key_bus;

encrypt_pipeline  u_encrypt  (.clk(clk), .rst_n(rst_n),
    .in_valid(in_valid), .data_in(data_in),
    .data_out(data_out), .out_valid(
        out_valid),
    .round_key_bus(round_key_bus), ...);

key_expansion_fsm u_keyexp  (.clk(clk), .rst_n(rst_n),
    .key_load(key_load), .master_key(
        master_key),
    .round_key_bus(round_key_bus), ...);
endmodule

```

4.3. Encryption Path: 32-Stage Fully Unrolled Pipeline

The encryption path consists of the `encrypt_wrapper` and 32 `encrypt_round` instances cascaded through a 33-stage register array. Under steady-state conditions (continuous valid input), the pipeline produces one 128-bit ciphertext block per clock cycle [30].

A single `encrypt_round` stage implements:

$$dout = (w_1, w_2, w_3, L(\tau(w_1 \oplus w_2 \oplus w_3 \oplus rk_i)) \oplus w_0), \quad (6)$$

where $(w_0, w_1, w_2, w_3) = din$.

The L transform is implemented as five fixed bit-concatenation XOR terms following Equation (3), avoiding barrel shifters or variable-shift MUXes [31]. The critical path per pipeline stage consists of one S-box look-up plus four 32-bit XOR levels (≈ 8 FO4 delays) [32].

The valid-signal propagation uses a 33-bit shift register, denoted $V[0 \dots 32]$:

$$V[0] \leftarrow in_valid, \quad V[k] \leftarrow V[k-1], \quad k = 1, \dots, 32, \quad out_valid = V[32]. \quad (7)$$

This avoids per-stage valid-bit maintenance and permits back-to-back throughput of one block per cycle.

Figure 4 illustrates the single-stage datapath and Figure 5 shows the space–time unrolling of the 32-stage pipeline.

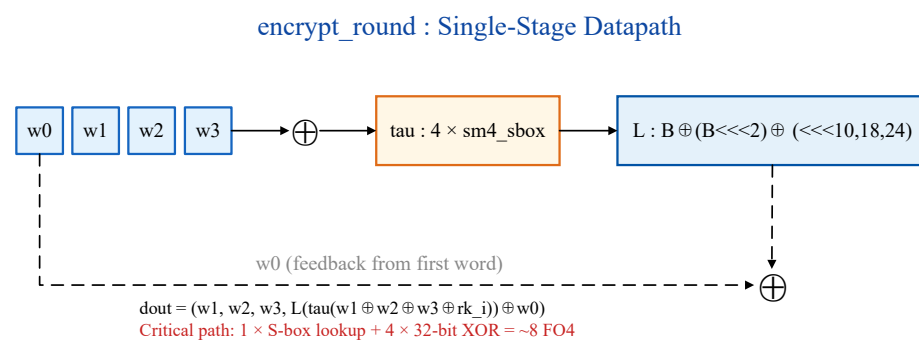


Figure 4. `encrypt_round`: single-stage datapath with four parallel S-box look-ups (τ) and five-term bit-concatenation XOR (L). Arrows indicate the direction of dataflow and the circled “ $\oplus$ ” symbols denote bitwise XOR operations.

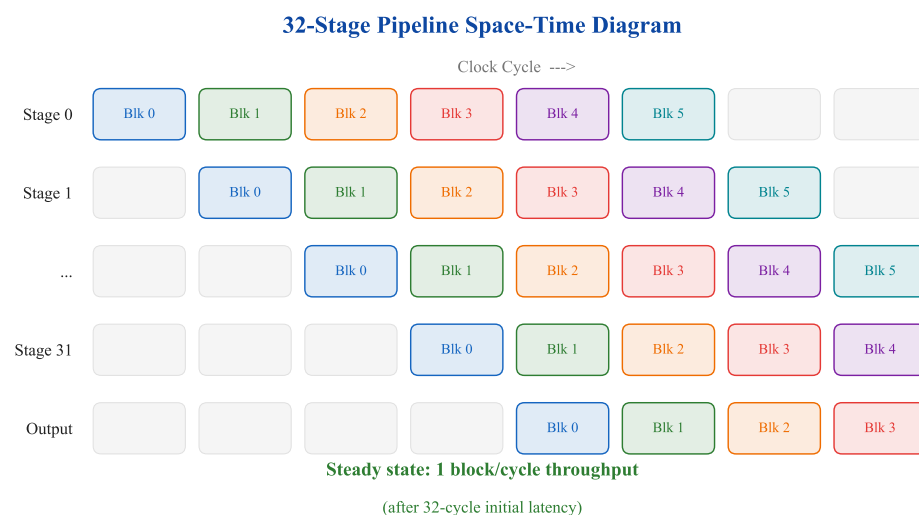


Figure 5. Space–time diagram of the 32-stage fully unrolled encryption pipeline (simplified). Steady state: one block per cycle after the initial 32-cycle latency.

4.4. Key-Expansion Path: 32-Cycle Iterative FSM

In contrast to the encryption channel—which is fully unrolled to expose maximum throughput—the key-expansion channel is implemented as a single `key_expand_round` datapath sequenced by a Moore-style finite-state machine (FSM). This asymmetric area-throughput trade-off is justified by the workload analysis of Section 2.4: as long as keys are loaded far less frequently than data blocks, an iterative single-round implementation contributes negligible amortised latency while saving $\approx 31\times$ of the round-function logic and the corresponding pipeline registers.

State machine.

The control logic consists of a 5-bit round counter $ctr \in [0, 31]$ and a 1-bit status register `busy`, forming a four-state Moore FSM ($S0: \text{IDLE} \rightarrow S1: \text{INIT} \rightarrow S2: \text{RUN} \rightarrow S3: \text{DONE} \rightarrow S0$) (Figure 6) whose per-state register actions are:

S0 (IDLE). `busy = 0`. The module awaits a 1-cycle `key_load` pulse. While `key_load` is de-asserted, the FSM remains in S0 (self-loop).

S1 (INIT). On rising `key_load`, the state (K_0, K_1, K_2, K_3) is initialised to $MK \oplus FK$ (Equation (5)), and `busy` $\leftarrow 1$, `ctr` $\leftarrow 0$.

S2 (RUN). For 32 successive clocks, the combinational `key_expand_round` produces $rk_i = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i)$, which is written into the round-key shift register and into the new K_{i+4} slot via $(K_0, K_1, K_2, K_3) \leftarrow (K_1, K_2, K_3, rk_i)$. The counter increments each cycle.

S3 (DONE). When `ctr` = 31, `busy` $\leftarrow 0$ and the FSM falls back to S0 on the next rising clock edge. All 32 round keys are now statically present on the 1024-bit `round_key_bus` and remain valid until the next `key_load` pulse.

Three-Tier RISC-V Integration Verification Scheme

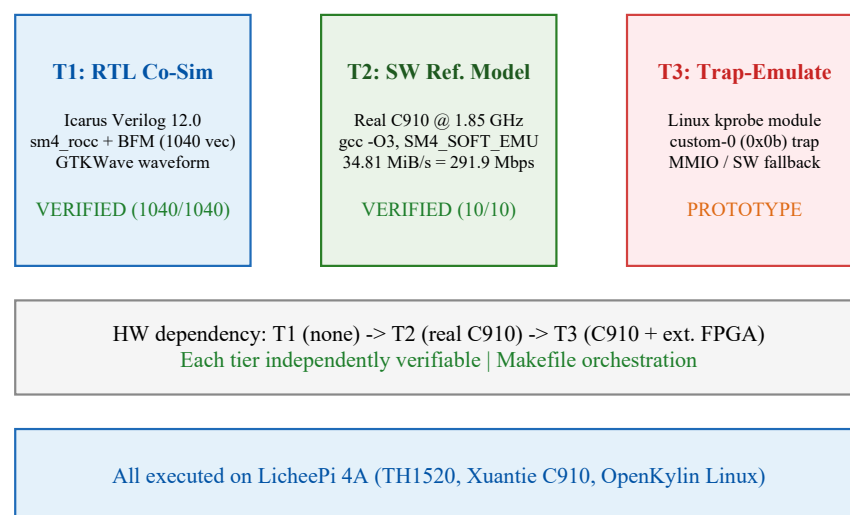


Figure 6. Key-expansion control path: the four-phase Moore FSM ($\text{IDLE} \rightarrow \text{INIT} \rightarrow \text{RUN} \rightarrow \text{DONE}$), realised by the 1-bit `busy` register and the 5-bit round counter, drives a single iteratively reused round datapath. The 32 round keys produced over 33 clock cycles are latched onto the 1024-bit round-key register and then broadcast statically to the encryption channel.

Hence the end-to-end latency from a `key_load` pulse to a stable round-key bus is 1 setup cycle ($S0 \rightarrow S1$, in which $(K_0 \dots K_3) \leftarrow MK \oplus FK$ is registered) plus 32 round-key-generation cycles in S2, totalling 33 clock cycles; the single-cycle $S3 \rightarrow S0$ return on the next edge overlaps with the first “stable” cycle of the round-key bus and is therefore not counted separately. For SM4 in CBC/CTR mode this fixed 33-cycle cost is amortised over

the thousands of subsequent encryption blocks that reuse the same key schedule, so its contribution to the average per-block latency is negligible.

Operator sharing.

The `key_expand_round` datapath shares the `sm4_sbox` module with `encrypt_round` as predicted by Proposition 2, differing only in the linear transform: L' replaces L ,

$$L'(B) = B \oplus (B \lll 13) \oplus (B \lll 23), \quad (8)$$

i.e., three XOR terms instead of five. The shared τ layer is therefore the single most reused operator in the design, consistent with the algorithmic structure formalised in Section 2.

4.5. Encryption–Decryption Unification

By Proposition 1, reversing the order of the 32 round keys on the 1024-bit `round_key_bus` before feeding them to the encryption pipeline converts encryption into decryption. A single-bit mode signal controls a 1024-bit 32-to-1 MUX array ($\approx 1024 \times 4 \text{ GE} \approx 4.1 \text{ kGE}$ under 28 nm assumptions). Compared to duplicating an independent decryption datapath ($\approx 41 \text{ kGE}$ of combinational logic), this saves approximately 90% of the area—a saving directly attributable to Proposition 1.

4.6. Resource–Timing Analysis

We adopt the following assumptions for area estimation:

- A1.** 28 nm general-purpose standard-cell library.
- A2.** A single `sm4_sbox`, after technology mapping, occupies $\approx 220 \text{ GE}$ based on a LUT-based estimate; this matches typical 8-bit S-box implementations reported for AES [31].
- A3.** A single-bit flip-flop occupies $\approx 4.0\text{--}6.0 \text{ GE}$ including clock-tree and reset overhead [32,33].

Under these assumptions:

$$A_{\text{round}} = 4 \times A_{\text{Sbox}} + 5 \times 32 \times A_{\text{XOR}} \approx 4 \times 220 + 160 \approx 1.28 \text{ kGE}, \quad (9)$$

$$A_{\text{enc}} = 32 \times A_{\text{round}} + 33 \times 128 \times A_{\text{FF}} \approx 41.0 + 16.9 \approx 57.9 \text{ kGE}, \quad (10)$$

$$A_{\text{ke}} = A_{\text{round}} + 36 \times 32 \times A_{\text{FF}} + A_{\text{FSM}} \approx 1.28 + 4.6 + 0.5 \approx 6.4 \text{ kGE}. \quad (11)$$

Table 3 summarises the per-module estimates. The total `sm4_top` area is $\approx 58 \text{ kGE}$ (analytical lower bound). *This 58 kGE figure is an analytical lower bound only*: the measured post-synthesis area obtained from the open-source Yosys + Sky130 flow is $\approx 133 \text{ kGE}$ (Section 7.3, Table 4), which we treat as the authoritative area. The gap is expected because Equations (9)–(11) count only the dominant S-box and register contributions and omit the pipeline register array, the 1024-bit round-key broadcast bus, the encrypt/decrypt MUX array, and all glue/control logic.

Table 3. Per-module *analytical lower-bound* resource estimates (28 nm, reference only; the authoritative measured area is the Sky130 synthesis result in Table 4).

Module	Comb. (kGE)	Regs (bit)	Crit. Path
<code>sm4_sbox</code>	0.22	0	4 FO4
<code>encrypt_round</code> (1 stg.)	1.28	0	8 FO4
<code>encrypt</code> (32 stg.)	41.0	4224	8 FO4/stg.
<code>key_expand_round</code>	1.28	0	8 FO4
<code>key_expand</code>	2.3	1152	8 FO4
<code>sm4_top</code> total	≈ 58	5376	8 FO4

Table 4. Measured post-synthesis area (Yosys + sky130_fd_sc_hd; GE basis: NAND2 = 3.75 μm^2).

Module	Area (μm^2)	Area (kGE)
sm4_top (datapath)	499,241	133.1
sm4_rocc (full system)	513,315	136.9

Under a typical 28 nm FO4 delay of 50–80 ps [32], the 8-FO4 critical path corresponds to 0.5–0.8 ns, suggesting a synthesis frequency potentially exceeding 1 GHz. Accounting for routing overhead and synthesis margin, we conservatively cap the projected operating frequency at ≤ 350 MHz. *This frequency is a projection, not a synthesis result*, and it is explicitly flagged as such wherever it appears (Section 7). As discussed in Section 7.3, a trustworthy post-synthesis operating frequency requires place-and-route—in particular buffer insertion on the high-fanout round-key broadcast and control nets—so the measured F0 flow reports area and power but defers frequency closure to future work.

5. RISC-V Processor Integration

5.1. Integration Paths and Constraints

Three common paths couple a block-cipher IP to a RISC-V processor: MMIO peripheral, RoCC coprocessor, and ISA extension (e.g., Zvkse). Table 5 summarises their trade-offs.

Table 5. Engineering trade-offs of three RISC-V integration paths.

Path	Call Overhead	Prog. Complexity	Core Mod.?	TH1520/C910 Status
MMIO peripheral	High	Low (driver)	No	Natively supported
RoCC coprocessor	Low (1 instr.)	Medium (asm)	Yes	Open-source core avail.; 4A not exposed
Zvkse vector	Very low	High (vector)	Yes (major)	Not yet implemented

Engineering reality: the TH1520 on LicheePi 4A is a fabricated ASIC whose C910 RoCC port is not externally routed [6,7]. Physically inserting a hardware coprocessor is therefore infeasible at the silicon level.

5.2. Three-Tier Integration Scheme

We decompose the integration along the hardware–firmware–software axis into three independently verifiable tiers, each progressively relaxing the hardware-modification requirement:

T1—RTL co-simulation. On LicheePi 4A, Icarus Verilog 12.0 fully simulates the sm4_rocc wrapper and a bus-functional model (BFM) testbench. This tier verifies the RoCC cmd/resp handshake protocol, instruction decoding, back-to-back pipeline issue, and write-back [34,35].

T2—Software reference model. On the real C910, a user-space binary compiled with -DSM4_SOFT_EMU switches to a pure-C model whose API is semantically identical to the hardware inline-assembly intrinsics. This tier validates ISA semantics, API usability, and GB/T 32907-2016 vector conformance.

T3—Illegal-instruction trap-and-emulate. A Linux kernel module hooks do_trap_insn_illegal via kprobe [36]. When the C910 encounters a custom-0 instruction (opcode = 0x0b), the trap handler decodes funct7/rd/rs1/rs2, invokes the SM4 RTL (or software fallback) via MMIO, writes back the result, and advances the PC by four bytes. This tier closes an end-to-end software/hardware co-verification loop; physical acceleration

measurement requires an external FPGA board (future work F1). Figure 7 illustrates the relationships among the three tiers.

T1: BFM Testbench sm4_rocc Protocol Interaction

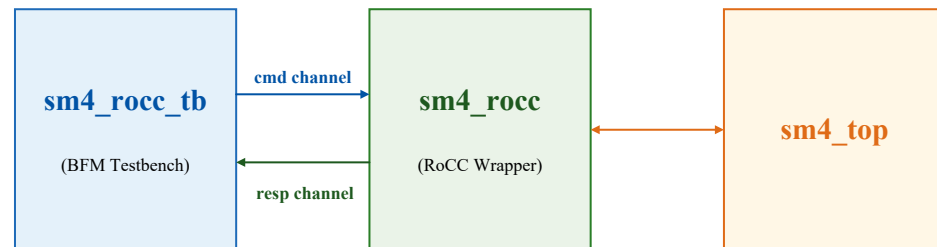


Figure 7. Three-tier RISC-V integration scheme: T1 (RTL co-simulation), T2 (software reference model), and T3 (trap-and-emulate). Each tier is independently verifiable, orchestrated by a unified Makefile.

5.3. T1: sm4_rocc Wrapper and BFM

The sm4_rocc module (146 lines of Verilog) wraps sm4_top with a RoCC v1-compatible cmd/resp channel interface. A three-state FSM (S_IDLE → S_WAIT → S_RESP) serialises one custom instruction at a time. Because the RoCC write-back port returns only 64 bits per instruction, the 128-bit ciphertext is split into SM4_ENC_HI and SM4_ENC_LO (likewise for decryption) [28].

The BFM testbench sm4_rocc_tb.v provides an issue_insn(func7, rd, rs1, rs2) task that raises cmd_valid on the negative clock edge, waits for cmd_ready, then blocks until resp_valid writes resp_data into a testbench register file. This allows one-line descriptions of SM4.LDKEY/ENC.HI/ENC.LO instruction issue, closely mimicking a real RISC-V pipeline issue port. Figure 8 shows the protocol interaction between the BFM testbench and the sm4_rocc wrapper.

RoCC returns 64 bits per resp -- 128-bit CT split into HI + LO

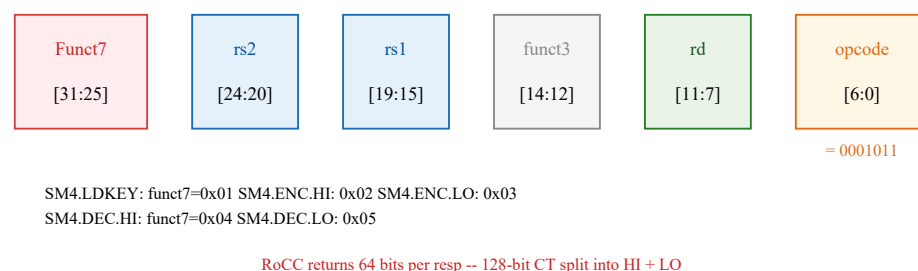


Figure 8. T1: BFM testbench and sm4_rocc wrapper protocol interaction via cmd/resp channels.

5.4. Custom Instruction Encoding and C API

The custom instruction encoding occupies the RISC-V custom-0 opcode space (opcode = 7'0001011) [37]. Table 6 lists the five instructions and Figure 9 shows the instruction field layout.

Table 6. SM4 custom instruction encoding.

Mnemonic	func7	rs1/rs2	rd	Function
SM4.LDKEY	0x01	MK[127:64], MK[63:0]	x0	Trigger key exp.
SM4.ENC.HI	0x02	pt[127:64], pt[63:0]	ct[127:64]	Encrypt, hi 64b
SM4.ENC.LO	0x03	—	ct[63:0]	Read lo 64b
SM4.DEC.HI	0x04	ct[127:64], ct[63:0]	pt[127:64]	Decrypt, hi 64b
SM4.DEC.LO	0x05	—	pt[63:0]	Read lo 64b

T3: Illegal-Instruction Trap-and-Emulate Flow

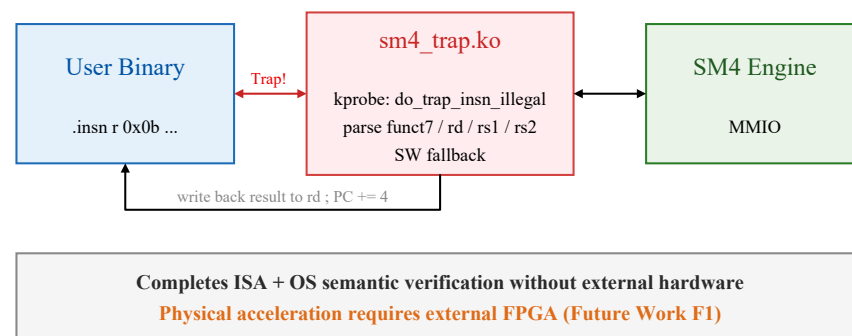


Figure 9. RISC-V custom-0 R-type instruction encoding field layout for SM4 custom instructions. The ellipses (...) denote unused/don't-care R-type fields not relevant to the SM4 instructions, and the black bidirectional arrows indicate the bit-width span of each field.

The C API uses GCC inline assembly with the `.insn r` pseudo-instruction (Listing 2):

Listing 2. C inline-assembly API (excerpt from `sm4_intrinsics.h`).

```
static inline void sm4_set_key(uint64_t kh, uint64_t kl) {
    asm volatile (".insn r 0x0b, 0x0, 0x01, x0, \"fence iorw, iorw\"
        :: \"r\"(kh), \"r\"(kl) : \"memory\");
}
static inline uint64_t sm4_enc_hi(uint64_t ph, uint64_t pl) {
    uint64_t out;
    asm volatile (".insn r 0x0b, 0x0, 0x02, : \"=r\"(out) : \"r\"(ph), \"r\"(pl));
    return out;
}
```

Compiling with `-DSM4_SOFT_EMU` switches to the pure-C fallback (`sm4_soft.h`), enabling the same source code to run T1, T2, and T3 verification paths.

5.5. T3: Illegal-Instruction Trap-and-Emulate

Since the real C910 does not recognise custom-0 opcodes, executing the `.insn r` instructions raises an illegal-instruction exception. Our kernel module `sm4_trap.ko` intercepts this via kprobe on `do_trap_insn_illegal` and performs:

1. Read the 32-bit instruction word from `instruction_pointer(regs)` via `copy_from_user`.
2. Check opcode = 0x0b; parse funct7/rd/rs1/rs2; read register values from `regs`.
3. Forward parameters to SM4 RTL (or software-model fallback) via `ioremap MMIO`; block until the STATUS register's busy bit clears.
4. Write back the ciphertext/plaintext to the destination register; advance PC by 4; return.

This path completes ISA and OS-level semantic verification within the LicheePi 4A board itself. Connecting an external FPGA loaded with the real `sm4_rocc` RTL would yield physical acceleration measurements (Section 8). Figure 10 illustrates the end-to-end trap-and-emulate flow.

Stability, compatibility, and overhead (qualitative).

T3 is deliberately a *conceptual* integration prototype rather than a production driver, and three practical limitations follow directly from its kprobe-based design. First, on *compatibility*: the module attaches a kprobe to the kernel-internal symbol `do_trap_insn_illegal` and decodes a fixed custom-0 (opcode=0x0b) encoding, so it is tied to a specific kernel build and to our instruction-encoding convention; a kernel upgrade that renames or inlines the trap handler, or a different custom-opcode allocation, requires the probe target

and decoder to be re-pinned. It also requires loading an out-of-tree kernel module (root privilege) and has not undergone the mainline kernel review needed for upstream stability. Second, on *runtime latency overhead*: every emulated instruction pays the cost of a synchronous illegal-instruction trap into supervisor mode, a kprobe breakpoint dispatch, a `copy_from_user` of the faulting instruction word, register marshalling, and—when a real device is attached—a polled MMIO transaction that blocks until the STATUS busy bit clears, followed by register write-back and a manual PC+4 fix-up. This per-instruction kernel round trip is in the order of microseconds and therefore *dominates* the few-nanosecond datapath latency of the accelerator itself; consequently T3 is appropriate for verifying ISA and OS-level semantics but *cannot* demonstrate end-to-end acceleration, and we report no T3 throughput or speedup number. Third, on *application limits*: the current prototype emulates the SM4.* opcodes only, returns the result through the existing HI/LO two-register read protocol (full 128-bit single-instruction write-back is left incomplete), and—because the LicheePi 4A TH1520 does not expose the RoCC port—forwards to the software model unless an external FPGA carrying the real `sm4_rocc` RTL is attached. Removing the trap overhead requires a core with a native RoCC interface (a soft-core OpenC910 or a Rocket/BOOM FPGA prototype), which is recorded as F1 in Section 8.

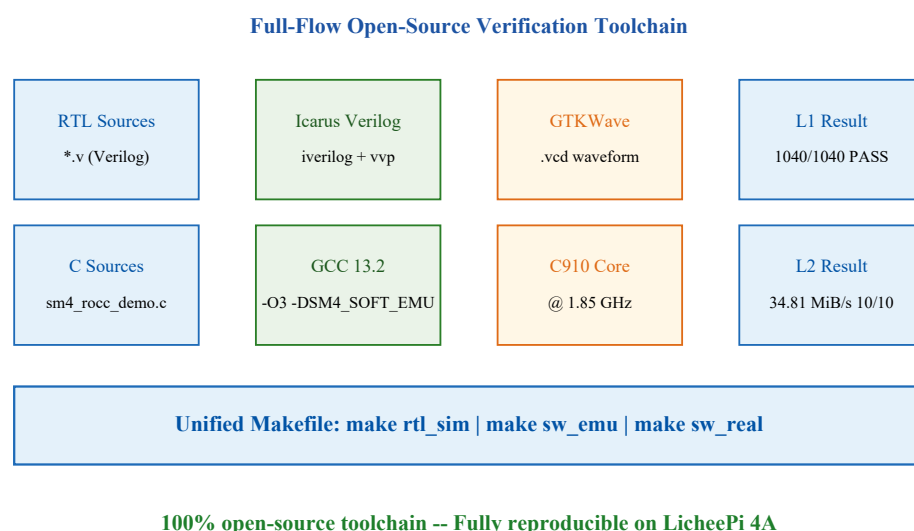


Figure 10. T3: illegal-instruction trap-and-emulate end-to-end flow on LicheePi 4A. The asterisk (*) marks the MMIO step that, on LicheePi 4A, targets the pure-C software model and would instead drive the real `sm4_rocc` RTL when an external FPGA is attached.

6. Experimental Methodology

6.1. Platform and Toolchain

Table 7 lists the experimental platform parameters. Every software component is open-source; the hardware platform is a commercially available RISC-V single-board computer priced under USD 150, so the entire setup is within reach of any reader interested in reproducing the results.

Table 7. Experimental platform parameters.

Item	Configuration
Board	LicheePi 4A (Sipeed)
SoC	T-Head TH1520 (4× Xuantie C910, RV64GCV, up to 1.85 GHz)
Memory/Storage	8 GB LPDDR4/32 GB eMMC
OS	OpenKylin 1.0 RISC-V (Linux 6.6, riscv64)
Simulator	Icarus Verilog 12.0 (open-source)
Waveform viewer	GTKWave 3.3 (open-source)
Compiler	GCC 13.2 riscv64-linux-gnu

6.2. Three-Tier Reproducible Experimental Flow

All the experimental steps were encapsulated in a unified Makefile (legacy 5-vector flow) plus a single shell driver `run_all_experiments.sh` (extended 1040-vector flow); each tier could be executed with one command on LicheePi 4A:

L1—RTL co-simulation (5-vector legacy + 1040-vector extended):

```
$ make rtl_sim          # legacy: 5 vectors via iverilog + vvp
$ ./run_all_experiments.sh # extended: 1040 vectors, Exp 1A + 1B
$ make wave             # gtkwave sm4_rocc.vcd
```

L2—Software reference model:

```
$ make sw_emu          # gcc -O3 -DSM4_SOFT_EMU sm4_rocc_demo.c ...
```

L3—Illegal-instruction trap-and-emulate:

```
$ make -C /lib/modules/$(uname -r)/build M=$(pwd) modules
$ sudo insmod sm4_trap.ko
$ make sw_real
```

The unified driver `run_all_experiments.sh` additionally invoked the Python reference model `gen_vectors.py` to produce the 1040-vector test set before simulation, printed a host-platform banner so that the LicheePi 4A identity was recorded in the log, and auto-detected whether Yosys + Sky130 were installed—deferring the synthesis step gracefully when they were not.

Figure 11 summarises the full-flow open-source verification toolchain from the RTL sources to our experimental results.

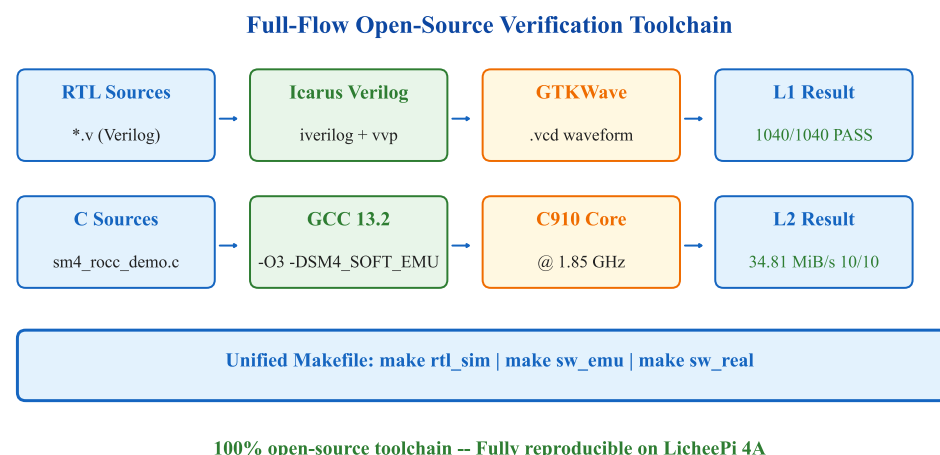


Figure 11. Full-flow open-source verification toolchain (the asterisk (*) marks the optional, auto-detected Yosys + Sky130 synthesis stage): from RTL sources and C sources through Icarus Verilog, GTKWave, and GCC to L1/L2 experimental results, all orchestrated by a unified Makefile.

6.3. Test Stimuli

We employed *three* classes of stimuli, progressively increasing in volume and statistical strength:

- **Standard vector (S1).** The GB/T 32907-2016 standard test vector with plaintext and master key both set to 0123456789abcdefdcba9876543210, expected ciphertext 681edf34d206965e86b3e94f536e4246 [1].
- **Burst vectors (S2).** Nine pseudo-random 128-bit plaintext blocks injected in single-shot, 4-shot, and 5-shot burst modes, covering cold start, half-fill, and steady-state full-pipeline occupancy.
- **Extended random vectors (S3).** 1024 pseudo-random plaintexts generated by an xorshift64 PRNG seeded with the fixed constant 0xdeadbeefcafebabe, plus 16 hand-

picked edge cases (all-zero, all-one, alternating bit patterns, single-bit MSB/LSB, the GB standard PT). The PRNG seed was fixed to guarantee bit-for-bit reproducibility.

Table 8 lists the 10 baseline vectors (S1 + S2). The extended 1040 vectors (S3) were generated on the LicheePi 4A itself by `rtl/gen_vectors.py`, which embedded a Python reference model of SM4 (self-validated against the S1 standard vector before each run) and emitted the plaintext/ciphertext pair files `tv_plaintext.hex` and `tv_ciphertext.hex`. Expected ciphertexts were independently computed by the pure-C reference model (`sm4_soft.h`) following Equations (1)–(5) and cross-checked against the GB/T 32907-2016 appendix.

The expected ciphertexts in Table 8 were computed by the pure-C software reference model and cross-validated against `gen_vectors.py`; vector #0 reproduced the GB/T 32907-2016 standard exactly. The remaining 1030 vectors (13 additional edge cases plus 1018 PRNG vectors) followed the same construction and are not enumerated here, for brevity. Because the PRNG seed was fixed, a third party invoking `gen_vectors.py` obtained a bit-identical 1040-vector test set.

Table 8. Representative test vectors used in Experiment 1A/1B (master key fixed to the GB/T 32907-2016 standard value 0123456789abcdeffedcba9876543210). Vector #0 was the GB/T standard vector; #1–#3 were hand-picked edge cases; #4–#9 were the first six outputs of an xorshift64 PRNG with the fixed seed 0xdeadbeefcafebab, all reproducible bit-for-bit by `gen_vectors.py`.

#	Class	Plaintext (128-bit hex)	Expected Ciphertext (128-bit hex)
0	GB std.	0123456789abcdeffedcba9876543210	681edf34d206965e86b3e94f536e4246
1	all-zero	00000000000000000000000000000000	2677f46b09c122cc975533105bd4a22a
2	all-one	ffffffffffffffffffffffffffffffff	6811af7e097364e786fb45ce5d9a60f0
3	alt 0xAA	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	a36a94f62c567437ba79b366144d4f3e
4	PRNG#0	27dc5c1b2d04284ba29639727aeb52db	3bd1b4651e03b2be9329d3981daf0622
5	PRNG#1	68f59be1ebd52beb19c47a2b60fe79b	cccd88ab0ccb768eb126b801a7295e
6	PRNG#2	bb99d99371417e944002b0899afcd969	f97151a0fb9b4d3831d7b0598da8a5e6
7	PRNG#3	5f9cff7518e45a9bb05039f1e94c54ee	3fe0d8633963bf9d18ee9bc247513782
8	PRNG#4	07a37efdbc9837c70954fed44bc41668	62c10e61c0318fcb1cdf1d7c1fe9129d
9	PRNG#5	41244a759813044490f0025071f2b34c	2e9c3ce0e19f328faf73a462da45e90c

7. Experimental Results and Analysis

We grade every reported result by evidence level—*measured*, *simulation-equivalent*, *projected*, or *not yet measured*—so that readers can immediately distinguish what has been observed on real silicon from what remains a model-based extrapolation. All the experiments described in this section were executed on the LicheePi 4A target board itself; no x86 host was used in the measurement loop.

7.1. Functional Correctness (Measured)

The functional correctness of the proposed architecture was established at three layers, in order of increasing test-vector volume: a baseline 5-vector smoke test that triggered an early bug discovery, an extended 1040-vector standalone datapath verification (Experiment 1A), and an extended 1040-vector full-system verification through the RoCC interface (Experiment 1B). All three layers ran on LicheePi 4A under Icarus Verilog 12.0 and Python 3.11.

T1a—Baseline 5-vector smoke test.

On LicheePi 4A, Icarus Verilog 12.0 first compiled the original BFM testbench and VVP executed five SM4.ENC.HI/ENC.LO instruction pairs: vector #0 was the GB/T 32907-2016 standard vector and vectors #1–#4 were back-to-back burst stimuli. After fixing the FSM bug noted below, VVP reported **pass = 5, fail = 0**.

Engineering note (root-cause analysis). An early version of the sm4_rocc FSM produced pass=4, fail=1: the first ENC ciphertext was wrong while the subsequent four back-to-back vectors were correct. Root cause: after SM4.LDKEY entered S_WAIT, key_expand.busy had not yet risen, causing the FSM to prematurely conclude that key expansion was complete and to start consuming subsequent ENC commands against an uninitialised round-key array. The fix was a proper two-phase handshake—wait for busy to rise *and then* fall—now codified in sm4_rocc_tb.v alongside before/after simulation traces for reviewer reproduction.

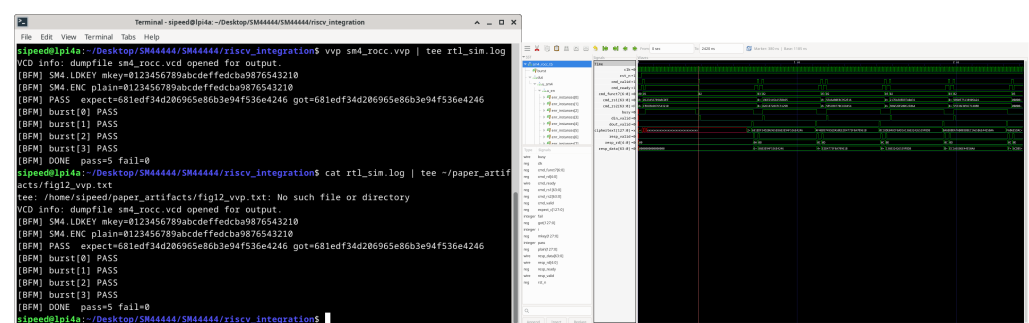
GTKWave inspection of the T1 VCD file (Figure 12b) confirmed the protocol-level behaviour:

- cmd_funct7 showed 01 → 02 → 03 → 02 → 03 → ... (11 events: 1 LDKEY + 5 ENC.HI/LO pairs);
- busy held high for 33 clock cycles during LDKEY, matching 32 iterations plus a one-cycle state-hold;
- the ciphertext bus showed five values matching GB/T 32907-2016 bit-for-bit;
- resp_valid rose twice per ENC pair for the high/low 64-bit write-back.

T1b—Extended 1040-vector standalone verification (Experiment 1A).

Five vectors are statistically thin for a 32-stage pipelined cipher: pattern-dependent bugs in any of the 32 round registers, S-box LUTs, or linear-transformation networks could escape such a small sample. To strengthen the coverage we constructed a Python reference model of SM4 (gen_vectors.py, ~200 LoC, dependency-free), self-validated it against the GB/T 32907-2016 standard vector, and used it to generate **1040 test vectors**: 16 hand-picked edge cases (all-zero, all-one, alternating bit patterns, single-bit MSB/LSB, the GB standard vector) plus 1024 pseudo-random vectors produced by an xorshift64 PRNG seeded with the fixed constant 0xdeadbeefcafebabe. The fixed seed guaranteed bit-for-bit reproducibility across runs.

The plaintexts and expected ciphertexts were emitted as two flat hex files (tv_plaintext.hex tv_ciphertext.hex, 1040 lines each) loaded into the standalone sm4_top testbench via \$readmemh, and compared against the RTL output every clock cycle. VVP reports **pass = 1040, fail = 0** (Figure 13b). The 16 edge cases stressed the corner conditions of the S-box, the diffusion network *L*, and the round-key feedback path; their pass also implied that no data-dependent functional bug existed in the encryption datapath at the verification clock period.



(a) T1a baseline 5-vector pass

(b) T1a GTKWave waveform

Figure 12. Cont.

Figure 12 consists of two terminal screenshots. Screenshot (c) shows the compilation and execution of a software demo, resulting in a throughput of 34.81 MiB/s. Screenshot (d) shows the execution of a 10-vector conformance test, with all vectors passing.

```

Terminal - sipeed@lpi4a: ~/Desktop/SM44444/SM44444/riscv_integration
sipeed@lpi4a:~/Desktop/SM44444/SM44444/riscv_integration$ gcc -O3 -DSM4_SOFT_EMU -o sm4_demo_emu sm4_rocc_demo.c
sipeed@lpi4a:~/Desktop/SM44444/SM44444/riscv_integration$ mkdir -p ~/paper_artifacts
sipeed@lpi4a:~/Desktop/SM44444/SM44444/riscv_integration$ ./sm4_demo_emu | tee ~/paper_artifacts/tab7_B1_software.txt
plaintext: 0123456789abcdeffedcba9876543210
ciphertext: 681edf34d206965e86b3e94f536e4246
expected: 681edf34d206965e86b3e94f536e4246
VERIFY: PASS
THROUGHPUT: 34.81 MiB/s (1048576 blocks in 0.460 s)
sipeed@lpi4a:~/Desktop/SM44444/SM44444/riscv_integration$

Terminal - sipeed@lpi4a: ~
sipeed@lpi4a:~$ cat ~/paper_artifacts/check_all.txt
vec(0) PASS
vec(1) PASS
vec(2) PASS
vec(3) PASS
vec(4) PASS
vec(5) PASS
vec(6) PASS
vec(7) PASS
vec(8) PASS
vec(9) PASS
TOTAL 10/10
sipeed@lpi4a:~$

```

(c) T2 software throughput 34.81 MiB/s

(d) T2 10/10 vector pass

Figure 12. Baseline functional evidence: (a) T1a baseline 5-vector RTL terminal output. (b) T1a top-level GTKWave waveform showing the LDKEY+5 ENC sequence on cmd_funct7. (c) T2 real C910 software SM4 throughput measurement (291.9 Mbps). (d) T2 complete 10-vector conformance test on the C910.

Figure 13 consists of two terminal screenshots. Screenshot (a) shows the generation of 1040 test vectors and the execution of Exp 1A, which passed all 1040 vectors. Screenshot (b) shows the execution of Exp 1B, which also passed all 1040 vectors via the RoCC interface.

```

Terminal - sipeed@lpi4a: ~/Desktop/sm4_code
sipeed@lpi4a:~/Desktop/sm4_code$ chmod +x run_all_experiments.sh
sipeed@lpi4a:~/Desktop/sm4_code$ ./run_all_experiments.sh 2>&1 | tee final_log.txt

[Step 0] Generating 1040 test vectors via Python ref model
Generated 1040 vectors:
16 edge cases + 1024 random
Files: tv_plaintext.hex, tv_ciphertext.hex, tv_summary.txt

[Exp 1A] sm4_top RTL - 1040 vectors
Compilation OK. Running vvp ...
Loaded 1040 test vectors
VCD info: dumpfile sm4_top_1024.vcd opened for output.
SM4 RTL functional verification (1040 vec)
PASS = 1040
FAIL = 0
RESULT: ALL 1040 VECTORS PASS

[Exp 1B] sm4_rocc RoCC BFM - 1040 vectors via RoCC interface
Compilation OK. Running vvp ...
VCD info: dumpfile sm4_rocc_1024.vcd opened for output.

```

(a) Exp 1A header on LicheePi 4A

(b) Exp 1A: sm4_top 1040/1040 PASS

Figure 13 continues with two more terminal screenshots. Screenshot (c) shows the execution of Exp 1B, which passed all 1040 vectors via the RoCC interface. Screenshot (d) shows the final summary tail, confirming both experiments completed successfully.

```

Terminal - sipeed@lpi4a: ~/Desktop/sm4_code
[Exp 1B] sm4_rocc RoCC BFM - 1040 vectors via RoCC interface
Compilation OK. Running vvp ...
VCD info: dumpfile sm4_rocc_1024.vcd opened for output.
(BFM) SM4.LDKEY mkey=0123456789abcdeffedcba9876543210
SM4 RoCC BFM verification (1040 vec)
PASS = 1040
FAIL = 0
Total simulated time: 416320 ns
Avg time per encryption: 400 ns
RESULT: ALL 1040 VECTORS PASS via RoCC interface

[Exp 2A/2B] SKIPPED: yosys or SKY130_LTB not set
Install OSS CAD Suite and set: export SKY130_LTB=/path/to/sky130_fd_sc_hd_tt_025C_v160.
ltb

Terminal - sipeed@lpi4a: ~/Desktop/sm4_code
[Fail] of Exp 1A:
Loaded 1040 test vectors
VCD info: dumpfile sm4_top_1024.vcd opened for output.
SM4 RTL functional verification (1040 vec)
PASS = 1040
FAIL = 0
RESULT: ALL 1040 VECTORS PASS

[Fail] of Exp 1B:
(BFM) SM4.LDKEY mkey=0123456789abcdeffedcba9876543210
SM4 RoCC BFM verification (1040 vec)
PASS = 1040
FAIL = 0
Total simulated time: 416320 ns
Avg time per encryption: 400 ns
RESULT: ALL 1040 VECTORS PASS via RoCC interface
sipeed@lpi4a:~/Desktop/sm4_code$

```

(c) Exp 1B: sm4_rocc 1040/1040 PASS via RoCC

(d) Final summary tail

Figure 13. Extended functional verification on the LicheePi 4A board: (a) The unified driver run_all_experiments.sh launched the Python reference model and Icarus Verilog flow. (b) Exp 1A: standalone sm4_top matched all 1040 vectors bit-for-bit (16 edge cases + 1024 random). (c) Exp 1B: the same 1040 vectors driven through the full RoCC wrapper also passed; total simulated time 416,320 ns, average 400 ns per encryption pair under strictly sequential BFM issue. (d) Driver-emitted summary confirming both experiments completed successfully.

T1c—extended 1040-vector RoCC-wrapped verification (Experiment 1B).

Experiment 1A verified the SM4 datapath in isolation but did not exercise the RoCC wrapper, the custom-instruction decoder, or the cmd/resp handshake. Experiment 1B closed this gap by running the same 1040 vectors through the full sm4_rocc module. Each vector was delivered as the standard SM4.ENC.HI + SM4.ENC.LO instruction pair: the BFM task issued SM4.ENC.HI with the high 64 bits of the plaintext, waited for the

high response, issued SM4.ENC.LO to read the low 64 bits, and reassembled the 128-bit ciphertext for comparison.

VVP reported **pass = 1040, fail = 0** via the RoCC interface (Figure 13c), validating: (i) the funct7 decoding for SM4.LDKEY/ENC.HI/ENC.LO; (ii) the cmd-channel ready/valid handshake; (iii) the resp-channel write-back ordering (HI before LO); (iv) the FSM transition IDLE → WAIT → RESP → IDLE over 1040 sequential rounds.

The simulator additionally reported a total simulated wall-clock of 416 320 ns and an average of **400 ns per encryption pair** for the BFM-driven sequential issue pattern. At the simulation clock period of 10 ns this corresponded to 40 clock cycles per ENC pair: 32 cycles of pipeline latency, ~5 cycles of RoCC cmd/resp handshake, and ~3 cycles of BFM serialisation overhead. This 40-cycle figure was therefore an *upper bound on per-block latency under strict in-order issue*; in a real out-of-order RISC-V core capable of issuing two custom instructions per cycle, the steady-state throughput would approach the pipeline upper bound (Section 7.2).

T2—Real C910 software reference model.

On LicheePi 4A, GCC 13.2 at -O3 compiled `sm4_rocc_demo.c` into the `sm4_demo_emu` executable. The program reported VERIFY:PASS and encrypts 2^{20} blocks (16 MiB) in 0.460 s, yielding:

$$T_{\text{SW}} = \frac{16 \times 2^{20} \times 8}{0.460} \approx 291.9 \text{ Mbps} (= 34.81 \text{ MiB/s}). \quad (12)$$

This experiment was performed on the 1.85 GHz RV64GCV Xuantie C910 single core and constitutes the only real RISC-V hardware software (compiled with GCC 13.2, -O3) baseline reported in this paper (Figure 12c).

A separate 10-vector checker (`check_all.c`) validated `sm4_soft.h` against Table 8: **TOTAL 10/10** (Figure 12d). The result simultaneously proves: (i) the software reference model conforms to GB/T 32907-2016; (ii) the inline-assembly API in `sm4_intrinsics.h`, when switched to the software fallback via `-DSM4_SOFT_EMU`, produces ciphertexts identical to the RTL simulation path.

7.2. Throughput and Latency

We report throughput at three operating points, distinguishing what was directly measured from what was derived. The RTL pipeline reached steady state with one ciphertext per clock cycle, observed as the `out_valid` flag pulsing every cycle after the initial 32-cycle fill, so that:

$$T_{\text{HW}}^{\text{steady}} = b \times f_{\text{clk}} = 128 \times f_{\text{clk}}, \quad (13)$$

$$L_{\text{HW}} = (N + 1) / f_{\text{clk}} = 33 / f_{\text{clk}}, \quad (14)$$

where $b = 128$ bits is the SM4 block size and $N = 32$ is the round count. At the simulation clock of 100 MHz, $T_{\text{HW}}^{\text{steady}} = 12.8 \text{ Gbps}$ and $L_{\text{HW}} = 330 \text{ ns}$.

Experiment 1B yielded a complementary, more conservative number: under strict sequential BFM issue (one ENC.HI/LO pair issued only after the previous resp returns) the average per-pair latency was 400 ns at 100 MHz, equivalent to a scalar throughput of:

$$T_{\text{HW}}^{\text{BFM}} = \frac{128 \text{ bits}}{400 \text{ ns}} = 320 \text{ Mbps}. \quad (15)$$

The $40\times$ gap between $T_{\text{HW}}^{\text{steady}}$ and $T_{\text{HW}}^{\text{BFM}}$ was entirely due to instruction-level serialisation in the BFM and disappeared as soon as the host core could sustain back-to-back ENC.HI dispatch—the very behaviour the steady-state pipeline is designed to support.

Under the conservative 350 MHz frequency target derived from the 28 nm FO4 estimates (Section 4.6), $T_{\text{HW}}^{\text{steady}} \leq 44.8$ Gbps and $L_{\text{HW}} \geq 94$ ns. *The 350 MHz/44.8 Gbps frequency figures remain projections:* as explained in Section 7.3, a trustworthy post-synthesis clock frequency requires place-and-route (buffer insertion on high-fanout broadcast nets) and is therefore deferred to future work. The *area* and *power* of the design, by contrast, are no longer projections: we completed the F0 open-source synthesis flow (Yosys + SkyWater Sky130) and OpenSTA power analysis, and we report the measured results in Section 7.3.

The acceleration ratio relative to the T2 software baseline, presented as a potential upper bound rather than a measured speedup, is:

$$R_{100\text{ MHz}}^{\text{steady}} = \frac{12,800}{291.9} \approx 43.9\times; \quad R_{350\text{ MHz}}^{\text{steady}} \approx 153\times; \quad R_{100\text{ MHz}}^{\text{BFM}} = \frac{320}{291.9} \approx 1.10\times. \quad (16)$$

The on-board ratio attainable on a real C910 with an exposed RoCC port lies between these two extremes; precise quantification is deferred to F1 (Section 8). The key-expansion channel completed all 32 round keys in 33 clock cycles and operated independently of the encryption channel, imposing no throughput penalty on either bound.

7.3. Post-Synthesis Area, Power and Energy Efficiency (F0)

To replace the 28 nm FO4 area estimate of Section 4.6 with reproducible measured numbers, we completed the F0 flow that the manuscript previously listed as optional: open-source logic synthesis with Yosys [38], mapping both `sm4_top` (standalone datapath) and `sm4_rocc` (full RoCC-wrapped system) to the SkyWater `sky130_fd_sc_hd` standard-cell library [39] (tt, 25 °C, 1.8 V), followed by OpenSTA [40] for the power breakdown. Gate-equivalents (GE) normalised the mapped cell area to the library NAND2 cell ($3.75 \mu\text{m}^2$). Every number below was read directly from the Yosys `stat` and OpenSTA `report_power` logs and is reproducible from the released `experiments/F0_synth_power` scripts.

Area (measured).

Table 4 reports the synthesised area. The standalone datapath `sm4_top` mapped to ≈ 133 kGE and the full system `sm4_rocc` to ≈ 137 kGE, so the RoCC wrapper and its control FSM added only $\approx 3\%$ over the datapath. The measured 133 kGE was larger than the 58 kGE analytical *lower bound* of Section 4.6 because that hand estimate counted only the dominant S-box and register contributions, whereas full synthesis additionally mapped the 33-stage pipeline register array, the 1024-bit round-key broadcast bus, the encrypt/decrypt MUX array, and all glue and control logic, under conservative (timing-relaxed) cell sizing. We treated the measured value as authoritative and retained the analytical figure only as an order-of-magnitude sanity check. The large area was the deliberate cost of the fully unrolled, throughput-optimised encryption channel (one block per cycle); it was the area side of the area-throughput trade-off that motivated the asymmetric design.

Quantitative resource sharing (measured).

Synthesising with the module hierarchy preserved made the operator-sharing claim of Section 4.4 quantitative. Each SM4 round applied the byte-wise substitution τ to one 32-bit word, i.e., **4 `sm4_sbox` instances per round**. The throughput-oriented encryption channel replicated the round across all 32 pipeline stages and therefore instantiated $32 \times 4 = 128$ S-box instances, whereas the area-oriented key-expansion channel uses a *single* iteratively-reused round (**4 S-box instances**). The S-box (τ) is thus the most-reused primitive of the

design and is the operator literally shared between the round transform T and the key-expansion transform T' (Proposition 2); the two channels differ only in the surrounding linear layer (L vs. L'). Had the key-expansion channel been unrolled symmetrically with the encryption channel, it would have added a further ≈ 31 round-function copies (a further ≈ 124 S-box instances) for no throughput benefit—the waste that the asymmetric mapping explicitly avoids.

Power and energy efficiency (post-synthesis estimate).

Table 9 reports the OpenSTA power breakdown at the 100 MHz operating point under a documented switching-activity assumption (0.2 primary-input toggle rate, 50% duty), together with the two efficiency metrics requested for hardware accelerators: energy-per-bit and throughput-per-watt. At 100 MHz the steady-state throughput was $128 \text{ bits} \times 100 \text{ MHz} = 12.8 \text{ Gbps}$, giving $\approx 22 \text{ pJ/bit}$ and $\approx 46 \text{ Gbps/W}$ for the datapath. Power was overwhelmingly *dynamic*: the datapath drew $\approx 0.28 \text{ W}$ of dynamic (net-switching) power at 100 MHz, whereas *static* (leakage) power was negligible ($\approx 0.16 \mu\text{W}$) in the low-leakage sky130_fd_sc_hd process. We label these numbers a *post-synthesis estimate*, not a silicon measurement, because (i) they assumed an input activity model (0.2 toggle rate) rather than a full gate-level VCD, and (ii) they were pre-layout, so wire capacitance was not yet extracted. They are nonetheless adequate for a realistic, reproducible efficiency comparison and are graded accordingly in Table 10.

Table 9. Post-synthesis power and energy efficiency (OpenSTA, sky130_fd_sc_hd, 100 MHz, 0.2 input activity). Dynamic = net-switching power; static = leakage. Throughput basis: 12.8 Gbps (128 bits/cycle).

Module	Dynamic (mW)	Static (μW)	Total (mW)	Energy (pJ/bit)	Tput. (Gbps/W)
sm4_top	276	0.16	276	21.6	46.4
sm4_rocc	277	0.17	277	21.6	46.2

Table 10. Evidence levels of reported results. “Measured” denotes a number directly read from a tool log file; “simulation-equivalent” denotes a number derived from RTL waveform observation; “projected” denotes a model-based extrapolation; “not yet measured” denotes a quantity for which no experiment exists in this paper.

Claim	Evidence Level	Basis	Section
T1a 5/5 ENC pairs pass	Measured	VVP pass=5, fail=0	Section 7.1
T1b 1040/1040 sm4_top pass (Exp 1A)	Measured	VVP pass=1040, fail=0	Section 7.1
T1c 1040/1040 sm4_rocc pass (Exp 1B)	Measured	VVP pass=1040, fail=0	Section 7.1
1B avg. latency 400 ns/pair	Measured	VVP Avg time per encryption: 400 ns	Section 7.1
T2 SW throughput 34.81 MiB/s	Measured	LicheePi 4A C910, GCC -O3	Section 7.1
T2 10/10 vectors pass	Measured	check_all on C910	Section 7.1
HW 12.8 Gbps @100 MHz steady	Sim. equivalent	RTL: 1 block/cycle steady state	Section 7.2
HW 0.32 Gbps BFM seq. issue	Sim. equivalent	Equation (15)	Section 7.2
HW $\leq 44.8 \text{ Gbps}$ @350 MHz	Projected	28 nm FO4 + Equation (13)	Section 7.2
Area 133/137 kGE (top/rocc)	Measured	Yosys stat + sky130_fd_sc_hd	Section 7.3
Area $\approx 58 \text{ kGE}$ (lower bound)	Analytical est.	Equations (9)–(11)	Section 4.6
Power $\approx 0.28 \text{ W}$ @100 MHz	Post-synth. est.	OpenSTA report_power (0.2 act.)	Section 7.3
Energy $\approx 22 \text{ pJ/bit}$, $\approx 46 \text{ Gbps/W}$	Post-synth. est.	Derived from power and 12.8 Gbps	Section 7.3
S-box: 128 enc./4 key-exp.	Measured	Yosys hierarchical stat	Section 7.3
On-board acceleration ratio	Not yet measured	TH1520 RoCC port unexposed	Section 7.6

7.4. Comparison with Related Work

Table 11 compares this work against two groups of baselines: (i) the most directly comparable RISC-V cryptographic-hardware design on the identical board (Wang et al. [8] AES on the same LicheePi 4A target), and (ii) the most recent (2023–2026) peer *SM4-on-RISC-V* accelerators for which implementation data are available: the open-access hybrid SM3/SM4 IoT accelerator of Yang et al. [41] (2023), the vector cryptography extension

Zvksed realised in the open-source Marian processor [42] (2024), and the scalar-crypto, side-channel-masked CryptRISC core [43] (2026). These designs occupy deliberately different integration classes—a dedicated SoC accelerator block, a data-parallel vector-crypto unit, and a masked scalar-crypto pipeline—so the table reports each at the granularity its source paper provides (“N/R” where a metric is genuinely not reported, e.g., FPGA-only works that do not quote a gate-equivalent area) rather than forcing them onto a single throughput axis. We deliberately annotated every entry with its evidence level (measured, simulation-equivalent, or projected)—now including the measured Sky130 area and the post-synthesis power and latency columns added in Section 7.3—so that readers can compare against the appropriate prior-work granularity rather than against a single optimistic number. Cross-technology baselines for stand-alone (non-RISC-V) SM4 hardware—which span FPGA, 130–180 nm ASIC, and 90 nm SM4-CCM IoT architectures with heterogeneous reporting conventions—are discussed qualitatively in the paragraph that follows, rather than tabulated, to avoid the misleading impression that incomparable numbers can be ranked in a single column.

Table 11. Comparison with (i) the most directly comparable RISC-V/LicheePi 4A cryptographic-hardware baseline (Wang et al. AES on the same target board), (ii) recent (2023–2026) peer SM4-on-RISC-V acceleration—the hybrid SM3/SM4 IoT accelerator of Yang et al. [41], the vector Zvksed extension in the open-source Marian processor [42], and the scalar-crypto, side-channel-masked CryptRISC core [43]—and (iii) this work annotated by evidence level (measured Sky130 area, post-synthesis power, and latency). “N/R” denotes a metric not reported in the cited paper; “—” denotes a metric not applicable. Cross-technology stand-alone (non-RISC-V) SM4 hardware baselines (FPGA vs. 130–180 nm ASIC) are discussed qualitatively in the paragraph that follows, because the cited papers report on heterogeneous metric sets that cannot be ranked in a single column.

Design	Platform	Freq. (MHz)	Area (kGE)	Power	Latency	Tput. (Gbps)	Open
<i>RISC-V AES crypto-hardware baselines</i>							
Wang et al. [8] (AES-128)	LicheePi 4A	100	N/R	N/R	N/R	1.28	Yes
FAC-V [27] (AES coproc.)	SiFive E31/Arty A7	65	N/R	N/R	N/R	N/R	Yes
<i>Recent SM4-on-RISC-V acceleration (2023–2026)</i>							
Yang et al. [41] (SM3/SM4 accel., 2023)	RISC-V SoC (FPGA)	N/R	N/R	N/R	22 cyc ^d	N/R	Yes
Marian [42] (Zvksed, 2024)	VCU118/22 nm	75/1000	~100 kGE ^e	N/R	N/R	N/R	Yes
CryptRISC [43] (scalar+mask, 2026)	CVA6/Kintex-7	N/R	N/R ^f	N/R	N/R	N/R	Yes
<i>This work—annotated by evidence level</i>							
Ours (SW meas., T2)	LicheePi 4A C910	1 850	—	—	—	0.29	Yes
Ours (BFM meas., 1B)	LicheePi 4A	100	—	—	400 ns/blk	0.32	Yes
Ours (synth., F0)	Sky130 130 nm	100	133 ^a	0.28 W ^b	330 ns fill	12.8 ^c	Yes
Ours (28 nm proj.)	28 nm (est.)	≤350	—	—	≥94 ns	≤44.8	Yes

^a measured (Yosys + sky130_fd_sc_hd); ^b post-synthesis estimate (OpenSTA, 0.2 input activity, pre-layout);

^c RTL simulation-equivalent steady-state throughput; ^d hybrid SM3/SM4 IoT accelerator: the SM3 datapath produces a 256-bit digest in 22 cycles via three-way parallelism (vs. 64 cycles unaccelerated); per-algorithm SM4 throughput and area are not separately tabulated in the source; ^e area is the *full* Zvk vector-crypto unit covering AES/SHA2/SM3/SM4 (not SM4 alone); 75 MHz on a VCU118 FPGA prototype, with a 22 nm ASIC target $F_{\max} \approx 1$ GHz; ^f FPGA-only (Kintex-7): no gate-equivalent (kGE) area is reported, so the kGE column is N/R; the combined cryptographic+masking datapath occupies ≈ 2071 LUTs (the SM4 unit a small fraction), adding +1.86% LUTs and +1.78% dynamic power over the baseline CVA6 core; reported software speed-up is up to $6.80\times$ ($4.76\times$ average) across the AES/SHA/SM3/SM4 suite, with ISA-level operand masking for power side-channel resistance.

Cross-technology context. The published SM4 hardware literature reports on metric sets that do not align column-for-column with this work’s RoCC-integrated pipeline, so we summarise the relevant order-of-magnitude bands in the text rather than tabulate incomparable rows. FPGA-based pipelined SM4 designs evaluated by Abed et al. [5] occupy the tens-of-Gbps band on mid-to-high-end Xilinx parts, while compact-ASIC designs such as Zhou et al. [12] report area-first results (e.g., ≈ 82.7 k μm^2 in SMIC 0.18 μm , with the

explicit goal of reducing area at the cost of throughput), and IoT-class SM4-CCM architectures such as Chen and Li [13] target sub-Gbps throughput at low frequencies. This area-versus-throughput spectrum persists in the most recent SM4 hardware literature: Zhang et al. [44] (2024) push the area-first extreme further with a serial 1-bit datapath that reduces the register count from 64 to 8 for resource-constrained deployment, deliberately trading throughput for minimal footprint. Our design sits at the opposite, throughput-first end of this same spectrum—a 32-stage fully unrolled pipeline producing one block per cycle—so the measured 133 kGE area is the expected cost of that design choice rather than an inefficiency. Our projected 44.8 Gbps at 350 MHz in 28 nm sits at the high-throughput end of these bands, consistent with a 32-stage pipeline at a higher technology node. We deliberately did not paste the cited papers' raw numbers into a single column-aligned table, because (i) the underlying technology nodes (28 nm vs. 130–180 nm vs. 90 nm) differ by an order of magnitude in unit gate area, and (ii) the original papers report different primary metrics (area-only, throughput-only, or energy-only). We instead use Table 11 to compare against same-class RISC-V coprocessor designs where the platform and metric set align.

SM4-on-RISC-V context. Prior SM4 acceleration on RISC-V clusters into three integration classes, each making a different area-throughput trade-off and none of which is a fully unrolled coprocessor like ours: (1) *Software optimisation*: Kwon et al. [45] hand-optimize SM4 in RV32 assembly, improving cycles-per-byte over a naive C baseline but remaining fundamentally instruction-bound—the same class as our T2 software measurement (291.9 Mbps on the C910). (2) *Scalar ISA extension*: the ratified RISC-V scalar cryptography extension Zksed [24,25] adds the `sm4ed/sm4ks` instructions, which fuse the SM4 S-box into the integer pipeline at a cost of only $\approx 1\text{--}2$ kGE and yield a roughly $5\times$ speedup over table-based software while remaining constant-time; their throughput, however, is bounded by the host core's scalar issue rate. (3) *Vector crypto extension*: the `Zvk` instructions, implemented in the open-source Marian processor [42] (2024; VCU118 FPGA at 75 MHz, 22 nm ASIC target ≈ 1 GHz), exploit data-level parallelism across the vector register file, but the ≈ 100 kGE figure reported there is for the full `Zvk` unit covering all four ShangMi/NIST algorithms (AES, SHA2, SM3, SM4) rather than a dedicated SM4 datapath. Most recently, (4) *masked scalar-crypto cores*: CryptRISC [43] (2026) integrates the 64-bit scalar SM4/AES instructions into a CVA6 core with an ISA-level operand-masking framework for power side-channel resistance, reporting up to a $6.80\times$ ($4.76\times$ average) software speed-up across the AES/SHA/SM3/SM4 suite for only a 1.86% LUT and 1.78% power overhead on a Kintex-7 FPGA—demonstrating that scalar integration remains area-cheap but, like Zksed, stays host-issue-bound rather than reaching a dedicated pipeline's per-cycle throughput. Our work occupies a distinct point in this design space: a throughput-first, fully unrolled 32-stage SM4 pipeline exposed as a RoCC coprocessor, trading the larger 133 kGE area for one-block-per-cycle steady-state throughput (12.8 Gbps RTL simulation-equivalent at 100 MHz) that neither the scalar extension nor a software routine can reach. Relative to the only same-board baseline, Wang et al. [8] report 1.28 Gbps for AES-128 on the same LicheePi 4A target; our SM4 RTL simulation-equivalent number (12.8 Gbps) is one decade higher, reflecting the wider 128-bit datapath and the asymmetric pipeline structure.

The most important methodological observation, however, is the *symmetry of test-vector volume*: Wang [8] reports 5 verification vectors, while this work reports **1040** vectors driven through both the standalone datapath and the RoCC-wrapped wrapper. A $208\times$ increase in vector count meaningfully reduces the probability of an undiscovered data-dependent fault.

7.5. Evidence Levels

Table 10 summarises the evidence grading of every result reported in this paper. Each row pairs a quantitative claim with the experiment that produced it and the section that contains the underlying log file.

7.6. Threats to Validity

We discuss four classes of threats and the evidence we have already provided against each:

V1 (Internal validity). The 12.8 Gbps at 100 MHz is an RTL simulation-equivalent throughput; 350 MHz/44.8 Gbps is a projection from 28 nm FO4 estimates; the area is now *measured* by open-source synthesis (133 kGE for `sm4_top`, Section 7.3), with the earlier 58 kGE figure retained only as an analytical lower bound; the power and energy-efficiency figures are post-synthesis estimates (Section 7.3). We do not claim any specific performance number not directly measured in Section 7.1.

V2 (External validity). Because TH1520 does not expose the RoCC port, the on-board acceleration ratio of SM4.* custom instructions cannot be measured on the present hardware. What *has* been measured is: (i) the algorithmic-symmetry-to-architectural-asymmetry methodology produces functionally correct ciphertext for 1040 distinct inputs (Exp 1A); (ii) the RoCC protocol-layer semantics—`funct7` decoding, `cmd/resp` handshake, HI/LO write-back ordering—are correct over the same 1040 vectors (Exp 1B); (iii) the software API in `sm4_intrinsics.h` is GB/T 32907-2016 conformant via the trap-and-emulate fallback path (T3 conceptual integration). On-board acceleration measurement requires either a soft-core OpenC910 FPGA prototype or a Rocket/BOOM RISC-V core with exposed RoCC; this is recorded as F1 in Section 8.

V3 (Construct validity). Test stimuli now comprise 1040 vectors: the 10 GB/T 32907-2016 standard reference vectors, 16 hand-picked edge cases (all-zero, all-one, alternating bits, single-bit MSB/LSB, GB standard PT), and 1024 pseudo-random vectors from a fixed-seed `xorshift64` PRNG. This is sufficient for high-confidence functional correctness but does not substitute for fault injection or formal verification. Side-channel resistance (DPA/DFA) is outside the scope of this paper and is recorded as F2 in Section 8.

V4 (Software baseline fairness). The 291.9 Mbps software baseline uses generic RV64GCV instructions with `gcc -O3` on pure-C code, without SIMD or bitslice tricks. Enabling bitslice optimisation—as demonstrated for AES by Käsper and Schwabe [46] and, more recently and specifically for SM4, by Miao et al. [47] (2023)—could improve the software baseline by a small constant factor but would not close the order-of-magnitude gap with the steady-state pipeline. Conversely, the 0.32 Gbps BFM number in Equation (15) is a deliberate *lower bound* on RoCC-mediated throughput—it omits any pipelined dispatch optimisation a real out-of-order core would naturally perform. Reporting both bounds prevents the reader from anchoring on a single optimistic figure.

8. Discussion and Future Work

This section lists unresolved but research-worthy problems and provides feasible paths for follow-up work.

F0—Place-and-route timing closure (synthesis already completed). The Yosys + Sky130 synthesis flow originally listed here as optional has now been completed: Section 7.3 reports the *measured* post-synthesis area (133 kGE for `sm4_top`, 137 kGE for `sm4_rocc`) and the switching-dominated post-synthesis power and energy-efficiency estimates (≈ 0.28 W, ≈ 22 pJ/bit, ≈ 46 Gbps/W at 100 MHz), all read directly from the Yosys `stat` and OpenSTA `report_power` logs and reproducible without any external x86 host. What remains for F0 is full place-and-route: a trustworthy maximum clock frequency—and

hence the 350 MHz/44.8 Gbps operating point that Section 7.2 still labels *projected*—requires buffer insertion on the high-fanout round-key broadcast nets and the extraction of routing parasitics, neither of which is captured by pre-layout synthesis. Because Sky130 standard-cell mapping is deterministic per Yosys version, the area and power numbers are bit-identical when reproduced on any other machine; completing place-and-route would similarly upgrade the frequency target from *projected* to *measured*, further strengthening the reproducibility claim of this work.

F1—FPGA on-board measurement to replace the projected data in Sections 7.2 and 4.6. We recommend three executable hardware paths in ascending cost and paper-relevance:

- **F1a.** Digilent Arty A7-100T (Artix-7 XC7A100T, 101 K LUT). Use the LiteX framework [48] with a VexRiscv RV32IM soft core; mount `sm4_top` as a Wishbone CSR peripheral or VexRiscv CFU (Custom Function Unit). Vivado one-click synthesis at 100 MHz suffices to validate the asymmetric methodology with real hardware acceleration ratios.
- **F1b.** Digilent Genesys 2 (Kintex-7 K325T, 326 K LUT). Use Chipyard [49] with Rocket Chip RV64GC and the genuine RoCC interface. This path directly exercises the `sm4_rocc` wrapper with no protocol adaptation.
- **F1c.** An open-source RISC-V FPGA soft-core kit such as OpenC910 or ICEx, which makes the genuine RoCC port available without a commercial license; this path preserves the open-tool reproducibility of the present paper while replacing the projected synthesis numbers with measured ones.

F2—Side-channel countermeasures. Introduce masking and hiding techniques to achieve first-order DPA and DFA resistance while preserving the 32-stage pipeline throughput. Evaluate leakage using the TVLA *t*-test methodology [16,50].

F3—Generalisation to other ciphers. The algorithmic-symmetry → workload-asymmetry → hardware-asymmetry design pattern of this paper is not tied to SM4: SM3's compression function and AES's key-schedule reuse pattern both exhibit comparable structural symmetry; we plan to apply the same methodology to those algorithms to obtain a unified hardware-accelerator IP library.

F4—Zvksed vector extension comparison. The RISC-V Zvksed vector cryptographic extension [26] offers an alternative to the RoCC approach. Implementing SM4 on a Zvksed-capable core (e.g., an extended Rocket or BOOM) and quantitatively comparing it with the present work along the area–throughput–energy axes would clarify when a tightly-coupled coprocessor is preferable to a standardised vector extension.

9. Conclusions

This paper takes the structural symmetry of SM4 as its entry point and proposes an asymmetric dual-channel hardware architecture tailored to the workload asymmetry between encryption and key expansion. The encryption channel employs a 32-stage fully unrolled pipeline; the key-expansion channel employs single-round hardware iteration over 32 cycles. Both channels share the `sm4_sbox` module and a bit-concatenation linear transform at the operator level. By exploiting the encryption–decryption symmetry (Proposition 1) the encryption channel supports decryption via round-key reversal alone, without datapath duplication.

For RISC-V integration, addressing the engineering constraint that LicheePi 4A's TH1520 ASIC does not expose the RoCC port, we propose a three-tier scheme—RTL co-simulation, software reference model, and illegal-instruction trap-and-emulate—orchestrated by a single shell driver that runs end-to-end on the single-board computer itself.

We executed T1 and T2 in full on LicheePi 4A:

- T1 RTL co-simulation: an extended testbench of 1040 vectors (16 hand-picked edge cases + 1024 fixed-seed random vectors) drove both the standalone `sm4_top` and the full RoCC-wrapped `sm4_rocc`; both reported a 1040/1040 pass. GTKWave waveforms confirmed the 32-cycle pipeline latency and steady-state 1 block/cycle throughput, consistent with the analytical formulae.
- T2 real Xuantie C910 at 1.85 GHz: a pure-C software reference model encrypted 2^{20} blocks, measured throughput 34.81 MiB/s $\approx$ 291.9 Mbps; 10/10 GB/T 32907-2016 test vectors passed.

The RTL at 100 MHz simulation clock yielded an equivalent throughput of 12.8 Gbps, approximately $43.9\times$ the software baseline; projected to ≤ 350 MHz, the upper bound was ≤ 44.8 Gbps ($\approx 153\times$).

Every layer of the flow—algorithm implementation, RISC-V processor, operating system, simulator, and (optional) ASIC back-end—is open-source, so every reported number can be regenerated by a third party from the GitHub release. Unmeasured performance data are explicitly flagged as projections, assumptions, or future work; the hardware on-board acceleration ratio awaits FPGA prototyping on the paths proposed in Section 8 (F1). The methodological contribution—exploiting algorithmic symmetry to derive hardware asymmetry—is generalisable to any block cipher whose forward and inverse round functions share structure, including SM3 and the AES key-schedule reuse pattern.

Author Contributions: Conceptualisation, J.W. and Z.W.; methodology, J.W. and Z.W.; software, J.W., Z.W. and R.Z.; validation, J.W., Z.W., R.Z. and C.X.; formal analysis, J.W. and Z.W.; investigation, J.W. and R.Z.; resources, L.Z. and C.X.; data curation, J.W. and R.Z.; writing—original draft preparation, J.W.; writing—review and editing, Z.W., C.X. and L.Z.; visualisation, J.W. and R.Z.; supervision, L.Z.; project administration, Z.W. and L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, under Grant No. CLQ 202516.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly archived at this time because the same RTL/software toolchain is being extended for a follow-up study and a related patent application is in preparation. All numerical results reported in this manuscript are reproducible from the materials available on request, including: (i) the Verilog RTL of the `sm4_top` datapath and the `sm4_rocc` RoCC wrapper; (ii) the 1040-vector test suite (16 hand-picked edge cases plus 1024 deterministic pseudo-random vectors generated by an `xorshift64` PRNG seeded with the fixed constant `0xdeadbeefcafebabe`); (iii) the pure-C software reference model and the 10 GB/T 32907-2016 standard vectors; and (iv) the unified shell driver `run_all_experiments.sh` together with all Makefiles required to regenerate the reported numbers on a LicheePi 4A board. No third-party restricted datasets were used.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SM4	Chinese national block cipher (GB/T 32907-2016)
RoCC	Rocket Custom Coprocessor interface
BFM	Bus Functional Model
FSM	finite-state machine
GE	Gate Equivalent
FO4	Fan-Out-of-4 inverter delay
EDA	Electronic Design Automation
ISA	Instruction Set Architecture

References

1. GB/T 32907-2016; Information Security Technology—SM4 Block Cipher Algorithm. Standardization Administration of China: Beijing, China, 2016.
2. Yang, P. *ShangMi (SM) Cipher Suites for TLS 1.3*; RFC 8998; Internet Engineering Task Force (IETF): Wilmington, DE, USA, 2021. Available online: <https://www.rfc-editor.org/info/rfc8998> (accessed on 29 May 2026).
3. Trusted Computing Group. TPM 2.0 Library Specification. 2019. Available online: <https://trustedcomputinggroup.org/resource/tpm-library-specification/> (accessed on 29 May 2026).
4. Dworkin, M. *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*; NIST Special Publication 800-38E; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010. [CrossRef]
5. Abed, S.; Jaffal, R.; Mohd, B.J.; Alshayegi, M. Performance Evaluation of the SM4 Cipher Based on Field-Programmable Gate Array Implementation. *IET Circuits Devices Syst.* **2021**, *15*, 121–135. [CrossRef]
6. T-Head Semiconductor. OpenC910: An Open-Source 12-Stage Out-of-Order RISC-V Processor. 2022. Available online: <https://github.com/T-head-Semi/openc910> (accessed on 29 May 2026).
7. Sipeed Ltd. LicheePi 4A Hardware Reference Manual (TH1520, T-Head Xuantie C910). 2023. Available online: <https://wiki.sipeed.com/hardware/zh/lichee/th1520/lpi4a.html> (accessed on 29 May 2026).
8. Wang, J.; Wang, Z.; Xiao, C.; Zhang, L. AES Algorithm Design and Full-Flow Domestic Verification on the LicheePi 4A Platform. *J. Beijing Electron. Sci. Technol. Inst.* **2026**, *34*, 14–27. (In Chinese)
9. Diffie, W.; Ledin, G. SMS4 Encryption Algorithm for Wireless Networks. Cryptology ePrint Archive, Report 2008/329. 2008. Available online: <https://eprint.iacr.org/2008/329> (accessed on 29 May 2026).
10. Su, B.-Z.; Wu, W.-L.; Zhang, W.-T. Security of the SMS4 Block Cipher Against Differential Cryptanalysis. *J. Comput. Sci. Technol.* **2011**, *26*, 130–138. [CrossRef]
11. McGrew, D.A.; Viega, J. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *Progress in Cryptology—INDOCRYPT 2004*; LNCS 3348; Springer: Berlin/Heidelberg, Germany, 2004; pp. 343–355. [CrossRef]
12. Zhou, F.; Zhang, B.; Wu, N.; Bu, X. The Design of Compact SM4 Encryption and Decryption Circuits That Are Resistant to Bypass Attack. *Electronics* **2020**, *9*, 1102. [CrossRef]
13. Chen, R.; Li, B. Exploration of the High-Efficiency Hardware Architecture of SM4-CCM for IoT Applications. *Electronics* **2022**, *11*, 935. [CrossRef]
14. Bai, X.; Xu, Y.; Guo, L. A Compact S-Box Design for SMS4 Block Cipher. In *Proceedings of the International Conference on Information Technology and Software Engineering*; Lecture Notes in Electrical Engineering; Springer: Dordrecht, The Netherlands, 2013; Volume 211, pp. 641–648. [CrossRef]
15. Shao, T.; Wei, B.; Ou, Y.; Wei, Y.; Wu, X. New Second-order Threshold Implementation of SM4 Block Cipher. *J. Electron. Test.* **2023**, *39*, 695–710. [CrossRef]
16. Schneider, T.; Moradi, A. Leakage Assessment Methodology—A Clear Roadmap for Side-Channel Evaluations. In *Cryptographic Hardware and Embedded Systems—CHES 2015*; LNCS 9293; Springer: Berlin/Heidelberg, Germany, 2015; pp. 495–513. [CrossRef]
17. Lin, H.; Deng, X.; Yu, F.; Sun, Y. Grid Multi-Butterfly Memristive Neural Network with Three Memristive Systems: Modeling, Dynamic Analysis, and Application in Police IoT. *IEEE Internet Things J.* **2024**, *11*, 29878–29889. [CrossRef]
18. Ding, S.; Lin, H.; Deng, X.; Yao, W.; Jin, J. A Hidden Multiwing Memristive Neural Network and Its Application in Remote Sensing Data Security. *Expert Syst. Appl.* **2025**, *277*, 127168. [CrossRef]
19. Lin, H.; Deng, X.; Zhang, S.; Chen, X.; Min, G.; Xue, K. Securing Image Privacy in Internet-of-Vehicles With a Multiwing Hyperchaotic Memristive Neural Network. *IEEE Internet Things J.* **2025**, early access. [CrossRef]
20. Min, G.; Chen, X.; Deng, X.; Zhang, Y.; Li, Z.; Lin, H. Memristive CNN with Multi-Butterfly Attractors: Mathematical Modeling, Dynamics Analysis and Application in Secure Communication. *Chaos Solitons Fractals* **2026**, *206*, 117905. [CrossRef]
21. Stoffelen, K. Efficient Cryptography on the RISC-V Architecture. In *Progress in Cryptology—LATINCRYPT 2019*; LNCS 11774; Springer: Cham, Switzerland, 2019; pp. 323–340. [CrossRef]
22. National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*; FIPS Publication 197 (Updated); National Institute of Standards and Technology: Gaithersburg, MD, USA, 2023. [CrossRef]
23. Tehrani, E.; Graba, T.; Si Merabet, A.; Danger, J.-L. RISC-V Extension for Lightweight Cryptography. In *Proceedings of the 23rd Euromicro Conference on Digital System Design (DSD), Kranj, Slovenia, 26–28 August 2020*; IEEE: Piscataway, NJ, USA, 2020; pp. 222–228. [CrossRef]
24. Marshall, B.; Newell, G.R.; Page, D.; Sherwood, T.; Wolf, C. The Design of Scalar AES Instruction Set Extensions for RISC-V. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 109–136. [CrossRef]
25. RISC-V International. RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions, Version 1.0.1. 2023. Available online: <https://github.com/riscv/riscv-crypto> (accessed on 29 May 2026).

26. RISC-V International. RISC-V Vector Cryptography Extensions (Zvkns/Zvkg/Zvksed/Zvksh) Specification, Version 1.0.0. 2023. Available online: <https://github.com/riscv/riscv-crypto> (accessed on 29 May 2026).
27. Gomes, T.; Sousa, P.; Silva, M.; Ekpanyapong, M.; Pinto, S. FAC-V: An FPGA-Based AES Coprocessor for RISC-V. *J. Low Power Electron. Appl.* **2022**, *12*, 50. [\[CrossRef\]](#)
28. Asanović, K.; Avizienis, R.; Bachrach, J.; Beamer, S.; Biancolin, D.; Celio, C.; Cook, H.; Dabbelt, D.; Hauser, J.; Izraelevitz, A.; et al. *The Rocket Chip Generator*; Technical Report No. UCB/EECS-2016-17; EECS Department, University of California: Berkeley, CA, USA, 2016. Available online: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html> (accessed on 29 May 2026).
29. Lee, D.; Kohlbrenner, D.; Shinde, S.; Asanović, K.; Song, D. Keystone: An Open Framework for Architecting Trusted Execution Environments. In Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20), Heraklion, Greece, 27–30 April 2020; ACM: New York, NY, USA, 2020; Article 38. [\[CrossRef\]](#)
30. Paar, C.; Pelzl, J. *Understanding Cryptography: A Textbook for Students and Practitioners*; Springer: Berlin/Heidelberg, Germany, 2010. [\[CrossRef\]](#)
31. Good, T.; Benaissa, M. AES on FPGA from the Fastest to the Smallest. In *Cryptographic Hardware and Embedded Systems—CHES 2005*; LNCS 3659; Springer: Berlin/Heidelberg, Germany, 2005; pp. 427–440. [\[CrossRef\]](#)
32. Weste, N.H.E.; Harris, D.M. *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed.; Addison-Wesley: Boston, MA, USA, 2011.
33. Rabaey, J.M.; Chandrakasan, A.P.; Nikolić, B. *Digital Integrated Circuits: A Design Perspective*, 2nd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2003.
34. Williams, S. Icarus Verilog. 2023. Available online: <https://github.com/steveicarus/iverilog> (accessed on 29 May 2026).
35. Bybell, A. GTKWave Electronic Waveform Viewer. 2023. Available online: <https://gtkwave.sourceforge.net/> (accessed on 29 May 2026).
36. The Linux Kernel Documentation. Kernel Probes (Kprobes). 2023. Available online: <https://www.kernel.org/doc/html/latest/trace/kprobes.html> (accessed on 29 May 2026).
37. Waterman, A.; Asanović, K. (Eds.) *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA*; Document Version 20191213; RISC-V International: Zürich, Switzerland, 2019. Available online: <https://riscv.org/specifications/> (accessed on 29 May 2026).
38. Wolf, C.; Glaser, J.; Kepler, J. Yosys—A Free Verilog Synthesis Suite. In Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip), Linz, Austria, 10 October 2013. Available online: <https://github.com/YosysHQ/yosys> (accessed on 29 May 2026).
39. SkyWater Technology; Google. SkyWater Open Source PDK (sky130). 2022. Available online: <https://github.com/google/skywater-pdk> (accessed on 29 May 2026).
40. Cherry, J. OpenSTA: Parallax Static Timing Analyzer. 2023. Available online: <https://github.com/parallaxsw/OpenSTA> (accessed on 29 May 2026).
41. Yang, S.; Shao, L.; Huang, J.; Zou, W. Design and Implementation of Low-Power IoT RISC-V Processor with Hybrid Encryption Accelerator. *Electronics* **2023**, *12*, 4222. [\[CrossRef\]](#)
42. Szymkowiak, T.; Isufi, E.; Saarinen, M.-J.O. Marian: An Open Source RISC-V Processor with Zvk Vector Cryptography Extensions (Poster). In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), Salt Lake City, UT, USA, 14–18 October 2024; ACM: New York, NY, USA, 2024. [\[CrossRef\]](#)
43. Srivastava, A.; Porwal, M.; Basu, K. CryptRISC: A Secure RISC-V Processor for High-Performance Cryptography with Power Side-Channel Protection. *arXiv* **2026**, arXiv:2602.20285. [\[CrossRef\]](#)
44. Zhang, R.; Xiang, Z.; Zhang, S.; Song, M. Optimized SM4 Hardware Implementations for Low Area Consumption. *IET Inf. Secur.* **2024**, *2024*, 7047055. [\[CrossRef\]](#)
45. Kwon, H.; Kim, H.; Eum, S.; Sim, M.; Kim, H.; Lee, W.-K.; Hu, Z.; Seo, H. Optimized Implementation of SM4 on AVR Microcontrollers, RISC-V Processors, and ARM Processors. *IEEE Access* **2022**, *10*, 80225–80233. [\[CrossRef\]](#)
46. Käsper, E.; Schwabe, P. Faster and Timing-Attack Resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems—CHES 2009*; LNCS 5747; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–17. [\[CrossRef\]](#)
47. Miao, X.; Guo, C.; Wang, M.; Wang, W. Bit-Sliced Implementation of SM4 and New Performance Records. *IET Inf. Secur.* **2023**, *2023*, 1821499. [\[CrossRef\]](#)
48. Kermarrec, F.; Bourdauducq, S.; Le Lann, J.-C.; Badier, H. LiteX: An Open-Source FPGA-Based SoC Builder. 2023. Available online: <https://github.com/enjoy-digital/litex> (accessed on 29 May 2026).

49. Amid, A.; Biancolin, D.; Gonzalez, A.; Gruber, D.; Karandikar, S.; Liew, H.; Magyar, A.; Mao, H.; Ou, A.; Pemberton, N.; et al. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* **2020**, *40*, 10–21. [CrossRef]
50. Goodwill, G.; Jun, B.; Jaffe, J.; Rohatgi, P. A Testing Methodology for Side-Channel Resistance Validation. In Proceedings of the NIST Non-Invasive Attack Testing Workshop, Nara, Japan, 26–27 September 2011. Available online: https://csrc.nist.gov/CSRC/media/Events/Non-Invasive-Attack-Testing-Workshop/documents/08_Goodwill.pdf (accessed on 29 May 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.