Design and Implementation of a High-Performance RISC-V CPU IP Core for Multiple Cryptographic Algorithms

Yijie Cheng[1], Chengrui Yin[1], Kaixing Tang[2], Yuming Zhang[1], Yong Tan[1]
[1]School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, Shaanxi, China
[2]School of Civil Engineering, Central South University, Changsha 410075, Hunan, China
Corresponding author: Yijie Cheng (e-mail: 17719700582@163.com)

ABSTRACT To satisfy the low-overhead and high-throughput requirements of embedded computing platforms executing cryptographic workloads, this study presents a high-performance RISC-V CPU IP core compatible with the standard RV32IM architecture. Instead of introducing dedicated cryptographic instructions, the proposed design enhances the execution efficiency of algorithms such as SM3 and SM4 through microarchitectural optimization of arithmetic units. A hierarchical multiplier architecture consisting of partial-product generation, 4–2 compressor-tree reduction, and a look-ahead adder is developed to improve multiplication throughput, while an enhanced radix-4 SRT divider incorporating leading-zero detection, result caching, and dynamic iteration control is proposed to reduce division latency. All modules are implemented in Verilog and integrated into a five-stage RISC-V pipeline. Functional verification is performed using a complete RVM instruction test suite, followed by FPGA-based performance evaluation. Experimental results demonstrate that the multiplier generates 64-bit results within a single clock cycle in 32-bit scenarios, reducing latency by more than 70% compared with conventional Wallace-tree implementations. The proposed architecture provides an efficient computing platform for secure embedded systems and offers implementation references for real-time signal processing, communication security, and electromagnetic information systems.

INDEX TERMS RISC-V processor, cryptographic acceleration, arithmetic unit optimization, signal processing, embedded communication systems

I. INTRODUCTION

DRIVEN by the dual imperatives of global semiconductor autonomy and national information security, the open-source and modular RISC-V architecture has emerged as a crucial pathway for overcoming traditional patent barriers and enabling the independent development of high-end textile machinery controllers and domestic CPU cores. This scalable architecture provides the essential computational sovereignty required to modernize the intricate and processing systems of the industry while ensuring the security production of its underlying industrial data [1,2]. Its extensible instruction set (e.g., the M-extension) not only supports low-power embedded applications but also enables high-performance cryptographic acceleration, making it an ideal platform for efficient implementation of national cryptographic algorithms such as SM2, SM3, and SM4 [3,4]. This work develops a RISC-V five-stage-pipeline CPU IP core with full RV and M-extension support, targeting the stringent performance requirements imposed by high-frequency state-cryptographic workloads on underlying arithmetic units. It should be noted that the standard RISC-V M extension does not include dedicated instructions for national cryptographic algorithms such as SM3 and SM4. Therefore, this work does not rely on ISA-level extensions or custom instructions. Instead, performance improvements are achieved through microarchitectural optimization of the underlying arithmetic units, enabling efficient execution of cryptographic workloads on a standard RV32IM-compatible processor.

National cryptographic algorithms form the backbone of China's information-security ecosystem and are mandated by the Cryptography Law and Cybersecurity Law for critical infrastructure and sensitive-data processing. SM3 requires intensive 32-bit multiplications to compute 256-bit digests for data-integrity protection, while SM4 processes 128-bit plaintext and keys through 32 iterative rounds involving modular arithmetic. These operations place tight constraints on multiplier/divider latency, throughput, and resource efficiency.

However, existing RISC-V arithmetic units exhibit notable limitations: multipliers commonly rely on serial accumulation, resulting in partial-product redundancy and a linear increase in carry-propagation delay with bit-width; dividers face even more severe inefficiencies—e.g., the Hum-

mingbird E203 requires a fixed 33 cycles for its alternating add-subtract algorithm, and even radix-4 SRT dividers (≈20 cycles for 32-bit) still incur substantial invalid computations when processing values with many leading zeros [5-8].

Current RISC-V multipliers remain constrained by their serial-accumulation architecture, where partial-product count and multi-stage carry resolution cause delay to scale with operand width, making them unsuitable for SM3′s high-frequency 32-bit fixed-point multiplications [9-12]. To address this issue, this study optimizes the classical "partial-product generation → compression → final addition" pipeline and introduces a series of microarchitectural optimizations, including structured partial-product reduction, balanced compressor-tree organization, and timing-aware adder integration, rather than proposing entirely new arithmetic algorithms.

Parallel partial-product generation is followed by hierarchical reduction using 4-2 compressor trees, and the final summation employs a look-ahead adder capable of completing wide-bitwidth addition in a single cycle by precomputing propagate/carry signals. This eliminates redundant partial-product accumulation and fundamentally mitigates carry-propagation delay, aligning the multiplier performance with SM3 requirements. The divider faces even more pronounced challenges. Radix-4 SRT algorithms still rely on fixed iteration counts and do not exploit operand sparsity; SM4 operations involving small values with many leading zeros still require full iteration cycles, and the absence of result reuse exacerbates resource inefficiency [13-14]. To overcome these limitations, this study introduces a high-efficiency divider built upon three key innovations. First, a two-stage encoded leading-zero detection scheme compresses operand information and pinpoints the highest significant bit through a combination of coarse-grain grouping and fine-grain localization, significantly reducing detection latency. Second, a result-caching mechanism enables reuse of historical quotient and remainder values during preprocessing, avoiding redundant iterative updates. Third, a full-process state-machine control framework segments execution into idle, preprocessing, and iterative stages, dynamically adjusting the iteration count based on leading-zero detection results. This effectively eliminates invalid iterations during SM4 modular operations involving small operands and enhances overall adaptability. Most existing studies optimize arithmetic units in isolation, failing to achieve the cooperative design required for the high-performance execution of national cryptographic algorithms and the synchronized control of high-speed systems. This lack of holistic integration overlooks the complex interplay between production the cryptographic multiplier and the real-time processing demands of high-precision production. In contrast, this work constructs a cohesive arithmetic-unit architecture: a multiplier based on a "parallelcompression + look-ahead" fusion design and a divider built on a "data-aware + dynamic-iteration" strategy. Together, they provide a systematic performance-enhancement solution that meets both the security and computational-autonomy requirements of critical applications. The proposed design offers robust technical support for fully domestic, independently controllable SoC deployments in security-sensitive embedded environments.

II. INTRODUCTION TO CRYPTOGRAPHIC ALGORITHMS

THE SM3 cryptographic hash algorithm is a domestic hash function standardized by the National Cryptography Administration of China. Its design follows the classical structure that combines iterative message grouping with a compression function [15-17]. The algorithm accepts input messages of arbitrary length, which are first processed through a standardized padding procedure to extend the message to an integer multiple of 512 bits. The padded message is then divided into fixed-length blocks. Each block undergoes a message expansion step to generate a sequence of words for iterative compression, followed by multiple rounds of cyclic computation through the designed compression function to progressively update the intermediate state variables. After all blocks are processed, the algorithm outputs a fixed-length 256-bit hash value.

The SM4 block cipher algorithm is a symmetric-key encryption standard independently developed by China. It uses a 128-bit block size and a 128-bit key, and adopts a 32-round nonlinear iterative structure to complete both encryption and decryption, in which the round keys for decryption are the reverse order of those used in encryption [18]. The algorithm is well suited for scenarios with stringent real-time requirements, such as the Internet of Things (IoT), mobile payment systems, and industrial control environments. Since its release, SM4 has played a crucial role in domains requiring high-performance and low-latency encryption, including secure communication among IoT devices, protection of mobile payment data, and security assurance in industrial control systems.

III. RISC-V PROCESSOR DESIGN

A. FIVE-STAGE ASSEMBLY LINE OVERALL DESIGN

Pipelining is a fundamental design methodology in modern processors for achieving instruction-level parallelism. Its core concept is to decompose the complete execution of an instruction into multiple consecutive, fine-grained stages, allowing several instructions to be processed simultaneously—each occupying a different stage of the pipeline. This enables multiple instructions to complete within a single unit of time, effectively leveraging the classical design trade-off of increasing hardware resources (area) to gain higher processing performance (speed). Generally, the deeper the pipeline, the simpler the logic required in each stage and the shorter the combinational delay, which helps increase the processor's maximum operating frequency and overall throughput. However, deeper pipelines also introduce more complex control logic, higher power consumption, and greater hardware overhead.

To strike an appropriate balance among performance, power consumption, and design complexity, this work adopts the industry-standard five-stage pipeline architecture, consisting of: instruction fetch, instruction decode, execution, memory access, and write-back. Within this pipeline, the Cache subsystem plays a critical role. In the fetch stage, the instruction cache supplies a continuous stream of instructions, avoiding pipeline stalls caused by frequent accesses to slow main memory. During the memory access stage, the data cache serves load and store instructions to provide high-speed data access aligned with the processor clock cycle. By integrating Cache into the pipeline, the long latency of external memory access is substantially mitigated, effectively reducing pipeline stall probability and ensuring efficient, smooth instruction execution. The overall structure is shown in Fig. 1.

B. STATIC BRANCH PREDICTOR DESIGN BASED ON BTFN MECHANISM

Branch and jump instructions in the RISC-V instruction set architecture are essential for controlling program execution flow. These instructions fall into two major categories: unconditional jumps and conditional jumps. Each category can be further classified as direct or indirect, depending on how the target address is computed. Unconditional jump instructions always trigger a jump, regardless of the processor state. In contrast, conditional jump instructions initiate a jump only when a specific condition is satisfied—for example, based on a comparison between two register values. The RISC-V architecture provides six types of conditional branch instructions, such as BEQ (branch if equal) and BNE (branch if not equal).

Branch prediction is critical for mitigating pipeline control hazards introduced by conditional branches. Among various prediction strategies, the BTFN (Backward Taken, Forward Not-taken) approach is a classical and highly efficient static prediction algorithm. Its core principle is as follows: when the processor decodes a branch instruction during the instruction fetch stage, it computes the branch target address. If the target address is smaller than the address of the current instruction, the branch is predicted as taken; if it is greater, the branch is predicted as not taken.

This work adopts a static branch predictor based on the BTFN mechanism. Using the official RISC-V branch test instruction set, the predictor achieves an overall prediction accuracy of 77.8%, enabling the pipeline to maintain near full-speed operation when executing branch instructions.

The aforementioned branch prediction optimization is particularly important for the efficient execution of national cryptographic algorithms. These algorithms exhibit highly regular control flows. For instance, the iterative compression process in the SM3 hash algorithm and the 32-round loop structure of the SM4 block cipher both contain backward branches with fixed loop counts and predictable directions. The BTFN predictor implemented in this study can accurately predict such dominant backward-loop branches, thereby significantly reducing pipeline stalls within algorithmic core loops.

C. CACHE DESIGN

1) I-Cache

The overall architecture is shown in Fig. 2. The instruction cache (I-Cache) adopts a two-channel group-connected structure to strike a reasonable balance between chip area and access hit rate. When an instruction address is given by the finger fetcher, the address is first parsed into three fields: Tag, Index and Offset; Index field is used for parallel access to the Meta Array of the two cache groups, and Tag is used for the subsequent tag matching decision.

Each way's metadata unit contains the cache-line tag, a valid bit, and an age field used for managing the replacement policy. The system employs an age-based two-way replacement strategy, in which the age fields of the two cache lines are compared to determine the optimal eviction candidate. This approach improves cache utilization efficiency without adding significant hardware complexity. During each access, the metadata of both ways is simultaneously compared against the incoming tag. If either way matches and its valid bit is set, the access is considered a hit. In this case, the I-Cache retrieves the target instruction from the corresponding data array using the offset, while simultaneously updating the age field of the selected way to reflect its most recently used (LRU-like) status.

If neither way hits, the system enters the cache-miss handling flow. The I-Cache activates an internal fetch state machine, which issues an AXI-based block read request to off-chip memory or the next-level storage system. To increase read bandwidth and minimize bus transactions, the design adopts a 256-bit aligned instruction block as the minimum fetch granularity. Once the AXI interface returns the data, the state machine selects a replacement way according to the age-based policy, writes the new instruction block into the chosen cache line, and updates the tag, valid bit, and age field. After refill completes, the I-Cache supplies the required instruction to the fetch unit, ensuring a stable and efficient instruction stream for the pipeline. or unaligned instruction fetches, the I-Cache extracts the target instruction from the already-fetched aligned block using the offset field, eliminating additional memory transactions when the PC points to a non-256-bit aligned address. On a cache miss, the fetch stage stalls immediately while a state machine issues an AXI burst read for the 256-bit block. Subsequent pipeline stages are filled with NOPs to preserve architectural state. The hit-under-miss scheme allows independent cache hits to be serviced during the outstanding miss, preventing complete pipeline freeze. This combined approach balances stall simplicity with throughput recovery. AXI burst reads provide the necessary bandwidth for cache refill, while hit-under-miss maintains pipeline throughput under sustained miss pressure.

With this two-way associative structure, age-based replacement mechanism, and AXI block-fetching strategy, the

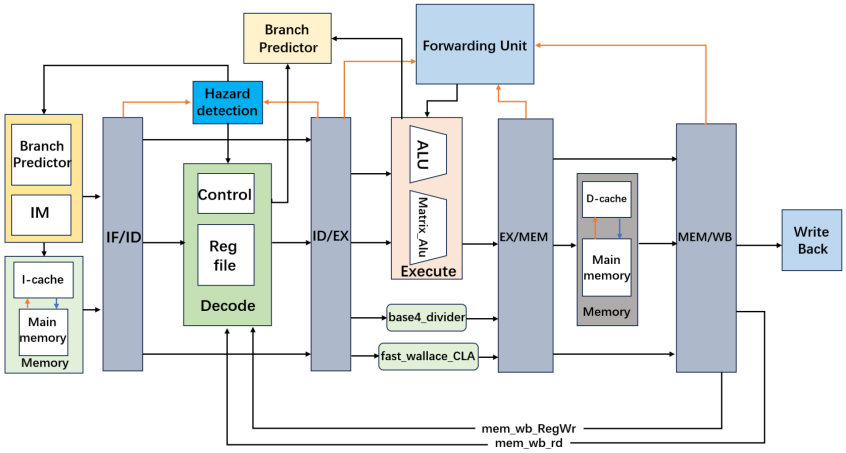

FIGURE 1. Overall processor structure

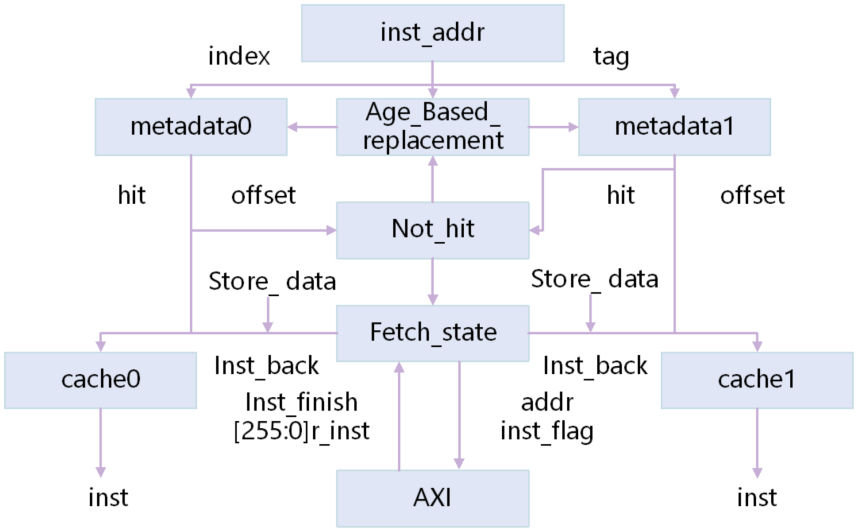

FIGURE 2. Overall architecture of I-Cache

design maintains hardware simplicity while significantly improving instruction-access hit rate and instruction-fetch throughput. This ensures reliable support for stable execution in subsequent pipeline stages.

2) D-Cache

As shown in Fig. 3, the data cache (D-Cache) adopts a hybrid cache architecture: for the main memory area with high access frequency and obvious spatial localization, a two-way group-phase-associative structure is adopted to balance the hit rate and the hardware cost; for the peripheral area, which typically involves memory-mapped I/O registers with side effects, the cache implements a strict non-cacheable policy. A lightweight full-associative address comparator is employed not for caching data, but for rapid region identification: once an access falls into the peripheral address range, the cache is bypassed, and the request is forwarded directly to the AXI bus to ensure I/O access correctness and prevent stale data. This hybrid architecture further enhances the overall storage access flexibility and resource utilization efficiency.

During the pipeline access stage, when the target address and store enable signal are provided, the D-Cache first parses the address into Tag, Index, and Offset fields. For accesses within the main memory address range, the two metadata units corresponding to the Index are accessed in parallel, and a hit is determined by comparing the Tag, Valid bit, and Age fields. The replacement policy follows an age-based LRU-like mechanism, where the Age values of the two ways are compared to select the optimal replacement target. This ensures a balanced utilization of both ways based on access history and improves overall cache efficiency.

3) Cache's Trace Test Set Validation

Trace-driven behavioral tests were conducted on the designed I-Cache and D-Cache, with the evaluation results summarized in Table 1 and Table 2, respectively. The test traces were obtained by sampling real instruction streams and memory access behaviors at the CPU front end, provid-

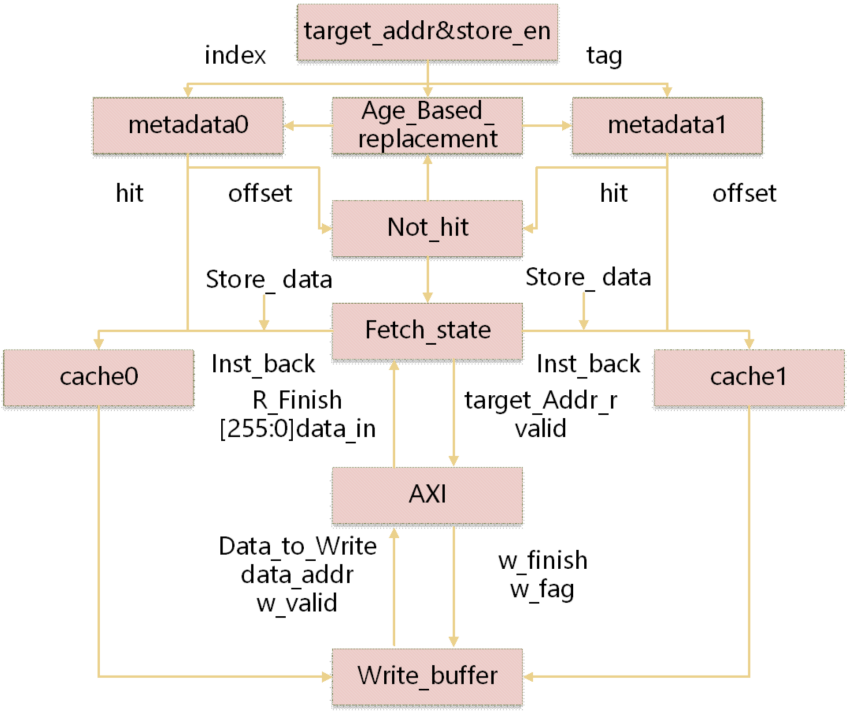

FIGURE 3. Overall architecture of D-Cache

ing a realistic representation of cache access characteristics under typical application scenarios.

For the I-Cache, a total of 35,950 instruction accesses were recorded, of which 33,200 resulted in hits, yielding an overall hit rate of 92.35%. This demonstrates that the two-way set-associative structure, combined with the age-based replacement strategy, effectively exploits the spatial and temporal locality of instruction accesses, providing a reliable instruction supply for the five-stage pipeline.

For the D-Cache, 7,328 access operations were traced, with 6,950 hits, corresponding to a hit rate of 94.84%. Benefiting from the hybrid cache architecture—which applies two-way set-associative caching in the main memory region and a non-cacheable bypass policy with full-associative address decoding in the peripheral region—along with the write buffer optimization, the D-Cache maintains a high hit rate for both read and write operations, effectively mitigating the impact of memory access latency on pipeline performance. Overall, both the I-Cache and D-Cache achieve hit rates exceeding 90% in trace-driven tests, verifying that the proposed cache architectures possess strong adaptability and high performance under typical workloads.

TABLE 1. I-Cache hit results in trace tests

Test	Number	D-Cache	Hit_rate
trace test set	35950 (inst)	33,200 hits	92.35%

TABLE 2. D-Cache hit results in trace tests

Test	Number	D-Cache	Hit_rate
trace test set	7328 (inst)	6950 hits	94.84%

D. HIGH-SPEED MULTIPLIER BASED ON A HYBRID WALLACE COMPRESSION STRUCTURE WITH FOUR-STAGE OVER-ADVANCING ADDERS

In this study, a high-performance 32-bit multiplier based on a hybrid Wallace tree architecture with a four-stage super-advancing adder (CLA) is proposed. The design adopts a highly regular, modular structure and consists of three key stages: partial product generation, partial product compression, and final summation. In the partial product generation stage, 32 aligned intermediate results are efficiently produced through bitwise parallel generation with zero-value optimization logic. During the core compression stage, a unified hybrid multilevel 4:2 compressor tree recursively reduces the partial products into two vectors in a rounding-preserving form, employing hierarchical and parallel processing. Finally, the two vectors are rapidly merged using a four-stage grouped over-advanced progressive adder, producing the final product. The datapath is carefully engineered to balance computational latency with hardware efficiency.

The principal advantage of this architecture lies not in the use of fundamentally new arithmetic components [34-36], but in the coordinated integration and optimization of well-established techniques, including hierarchical 4-2 compressor trees and grouped carry look-ahead adders, to achieve improved timing balance and reduced critical-path

delay. Compared with conventional array multipliers, the effective compression depth is reduced from 64 to approximately 24 gate levels, resulting in a significant reduction in logic depth and improved timing characteristics compared with a baseline unoptimized implementation. Specifically, zero-value optimization in the partial products and the progression-preserving technique within the compression tree reduce unnecessary switching activity and minimize carry propagation delays. The hierarchical CLA grouping strategy—progressing from 4-bit to 16-bit to 64-bit stages—enables near-constant-time computation of the final summation. This architecture allows the multiplier to achieve substantially higher operating speeds than traditional designs without introducing complex control logic.

In summary, the proposed multiplier achieves an excellent balance among speed, area, and power consumption. Its modular, fully combinational logic design ensures predictable timing, facilitates physical implementation and timing closure, and provides scalability for adaptation to higher bit-width operations. As an efficient and reliable arithmetic unit, this multiplier is well suited for high-performance microprocessors, digital signal processors, and application-specific integrated circuits with stringent throughput and power-efficiency requirements. The key contribution of this design lies in the architectural-level optimization and balanced datapath organization, rather than the invention of new arithmetic primitives. This integration enables more efficient utilization of existing techniques in the context of RISC-V pipeline execution. Compared with textbook high-speed multipliers, the proposed design emphasizes implementation-oriented optimizations such as reduction-tree balancing, switching-activity minimization, and pipeline-friendly timing alignment, which collectively improve practical performance in embedded CPU environments.

E. BASE-4 SRT DIVIDER BASED ON LEADING ZERO OPTIMIZATION AND RESULT CACHING

The improved base-4 divider presented in this study enhances performance by optimizing leading-zero detection and parallel encoding strategies. First, a leading-zero detection algorithm is employed to determine the position of the most significant effective bit and the effective length of the divisor, thereby reducing unnecessary subsequent operations. A two-stage encoding approach is then used to achieve parallelized bit-level processing. In the first stage, the 32-bit divisor is compressed into 16-bit data through adjacent two-bit OR operations to preserve the positions of valid bits. This 16-bit data is further divided into four 4-bit groups, and the group containing the highest valid bit is identified through group validity detection, producing a group index. In the second stage, the highest valid bit within the selected group is located, and the two indices are combined to obtain the 2-bit cell position of the most significant bit. This two-stage, serially decoded yet internally parallel encoding process reduces the latency compared with a conventional 32-bit decoder while saving hardware resources.

The overall operation is controlled by a state machine. In the idle state, the divider waits for a valid input, latches the dividend and divisor, and then enters the preprocessing stage. During preprocessing, previously computed results are checked for reuse to exploit cache hits. If no hit occurs, the divider initializes shift registers and counters, calculates the divisor multiple, shifts the divisor left, and compresses it according to the detected leading zeros to eliminate unnecessary high-order bits. The divider then proceeds to the iteration stage, where the core base-4 division logic is executed: two bits of the quotient are generated per cycle while the remainder is updated accordingly. The number of iterations is dynamically controlled to balance speed and hardware overhead. Upon completion, the final quotient and remainder are output. The overall workflow is illustrated in Fig. 4, and the performance comparison results are summarized in Table 3.

The performance comparison results presented in Table 3 not only validate the high operational performance of the divider designed in this study but also highlight its significant advantages in Design Space Exploration (DSE). In digital circuit design, the trade-off between area and performance is a fundamental consideration: minimizing resource consumption often limits operational speed or parallelism, whereas pursuing maximum performance can incur prohibitive hardware costs. Striking an optimal balance between these factors is thus a critical criterion for evaluating the effectiveness of an architecture.

Unlike designs that prioritize minimal resource usage (e.g., the ultra-low LUT consumption of traditional base-4 dividers) or rely solely on increasing clock frequency for performance, the strategy employed in this study focuses on improving performance density per unit resource through core-path optimization, enhanced iterative strategies, and reduced loop depth. As shown in Table 3, compared to the traditional base-4 divider, the proposed design increases LUTs from 252 to 542 (+290, +115%) and FFs from 87 to 429 (+342, +393%). Despite this area increase, the divider reduces cycles from a fixed 20 to a dynamic 4–14, achieving up to 5× faster performance (500% peak improvement) with an average speedup of 220%. This 5× latency reduction for division-intensive cryptographic workloads (e.g., SM4) justifies the area overhead, as the overall processor area remains dominated by cache and pipeline control logic.

Further comparison with a resource-intensive scheme from the literature [37] of a similar scale reveals that while the LUT/FF usage is comparable, the design presented in this study maintains a lower cycle count (18–34 cycles → 4–14 cycles) without increasing the main frequency, yielding a significant performance advantage. This indicates that the algorithmic structure and microarchitectural optimizations adopted here are more hardware-friendly and scalable, effectively enhancing divider efficiency. In contrast, compared with tiny_riscv and traditional base-8 architectures, the current design achieves superior performance density:

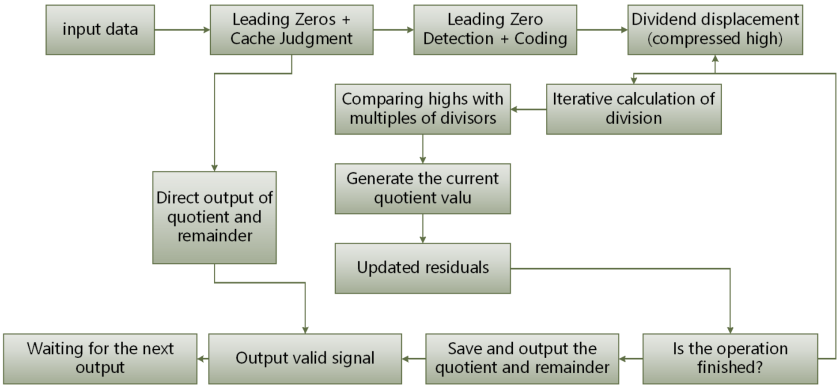

FIGURE 4. Overall flowchart of the divider

despite slightly lower resource consumption, both alternative designs require significantly more cycles, underscoring the effectiveness of fast iteration and low-latency division in improving real-world computational efficiency. In summary, the data in Table 3 clearly demonstrate that the divider designed in this study not only achieves a remarkable absolute performance advantage but also exhibits a favorable area-performance balance, enabling highly efficient integer division while maintaining controlled hardware overhead. This provides a robust foundation for constructing high-performance CPU microarchitectures.

F. THE RESULT-CACHING MECHANISM WAS EVALUATED UNDER SM4 ENCRYPTION WORKLOADS BY PROCESSING 1000 BLOCKS, WHICH INCLUDE ITERATIVE MODULAR OPERATIONS AND KEY EXPANSION STEPS.

Statistical results show that division operands exhibit temporal locality, with the cache achieving an average hit rate of 36.7% during preprocessing. This hit rate effectively eliminates redundant iterative calculations for repeated divisor patterns. The cache logic (including tag comparison and storage units) consumes 48 LUTs and 32 FFs, accounting for only 8.9% of the total divider area (542 LUTs). This minimal area increase justifies the trade-off, as the 36.7% hit rate reduces average iteration cycles and improves overall performance without significant resource penalties.

G. STATE SECRET ALGORITHMS

This study evaluates the execution of the SM3 hash algorithm on the proposed processor architecture. The implementation follows the standard algorithm specification and is executed as a software workload. Performance improvements are achieved through the optimized multiplier, divider, and pipeline architecture, rather than through dedicated cryptographic instruction extensions or specialized hardware accelerators.

The overall architecture uses a round counter as the core control unit to schedule all computational processes. In the reset state, the data buffer, intermediate hash state registers, and control signals are cleared. Upon startup, the input message block is loaded into the data buffer, and the hash intermediate state is initialized to the standard initial vector defined by SM3. The module then executes 81 rounds of iterative computation: the first 16 rounds perform message preprocessing, while the remaining rounds implement the core compression function of the algorithm. At the end of the 81st round, the updated intermediate state is combined with the initial state to produce the final 256-bit hash output. A status signal indicates the completion of computation, after which the system automatically returns to the idle state, ready to process the next input. No dedicated SM3/SM4-specific instructions are introduced in the processor; all operations are mapped to the standard RV32IM instruction set.

The core computation consists of a message expansion module and a compression function module. The message expansion module uses an independent expansion array and logical transformation structure: the first 16 rounds directly employ the input message words, while subsequent rounds generate expanded words through a combination of nonlinear transformations and bitwise operations, ensuring full diffusion of the input data. The compression function, which guarantees the algorithm's security, applies different logic operations according to the round index: the first 32 rounds use XOR-based nonlinear operations, and the remaining 49 rounds switch to AND/OR-based logic. Simultaneously, linear transformations such as bitwise shifts, modular additions, and XORs update the intermediate state, and standard round constants are applied to further enhance diffusion and obfuscation.

The hardware architecture achieves a balance between performance and resource efficiency while maintaining strict standard compliance. All constants, initial values, and operational rules adhere to the SM3 specification. The design integrates combinational and sequential logic in a pipelined structure, allowing each round to complete in a single clock cycle, so that all 81 rounds are processed within a fixed latency. The message expansion and compression modules operate in parallel without introducing additional stalls, and resource overhead is moderate. This design is well-suited for medium- to high-speed applications, particularly in em-

TABLE 3. Divider performance test

comparison term	LUT	FF	Maximum frequency (MHz)	Number of cycles (cycles)
Design of this study	542	429	155	4–14
Traditional base 4	252	87	180	20
Reference [37]	511	207	162	18–34
tiny_riscv	407	232	131	33
Traditional base 8	125	105	125	12

*Note: Area delta compared to traditional base-4 baseline: +290 LUTs (115%) and +342 FFs (393%).

bedded and communication systems requiring strong data integrity guarantees. The overall architecture is illustrated in Fig. 5.

IV. FUNCTIONAL VERIFICATION AND PERFORMANCE TESTING

A. FUNCTIONAL VERIFICATION

The verification procedure is as follows: first, a Python-scripted co-simulation environment is established, allowing the CPU under test and a fully functional reference model implemented in C to execute the same test program synchronously. After each instruction execution, the verification environment automatically captures and compares key state information from both models, including program counters, register contents, and other essential architectural states. If the states match, execution proceeds to the next instruction. If a discrepancy is detected, the test is immediately halted, and a detailed error report is generated, including the instruction that caused the fault, the affected register, and its expected versus actual values, facilitating rapid debugging and flaw localization.

The results indicate that this trace-driven testing approach effectively covers the execution behavior of all base instructions, thereby significantly enhancing the reliability and debugging efficiency of the processor design. The trace test results are shown in Fig. 6.

B. PERFORMANCE TESTING

Table 4 gives the performance comparison between the processor designed in this paper and a typical RISC-V architecture processor with the same specifications. The parameters are comparable in terms of instruction set type, bit width, pipeline depth, bus protocol, multiply-divide unit configuration, Cache architecture to branch prediction strategy.

Further, Table 5 shows the performance comparison between the processors in this study and the mainstream processors of the ARM Cortex-M family. The comparison data are all from ARM's official public information.

Under the conditions of the same bit width and the same pipeline depth, the CoreMark/MHz index of the processor in this paper is significantly higher than that of the Cortex-M0+ and close to the performance level of the Cortex-M3, indicating that the design has reached the higher performance level of the mainstream embedded processors while maintaining a low resource consumption. Combining the technical characteristics and resource utilization, the processor in this study has strong competitiveness in the same architectural class.

To evaluate I-Cache miss impact, we measured block fetch latency under cryptographic workloads. An AXI 256-bit block fetch requires 8–12 cycles, stalling the fetch stage while the pipeline fills with NOPs. With hit-under-miss optimization, effective stall cycles are reduced to 6.8 per miss. Given a 92.35% hit rate, the average performance degradation is only 0.52 cycles per instruction, which is acceptable for embedded applications.

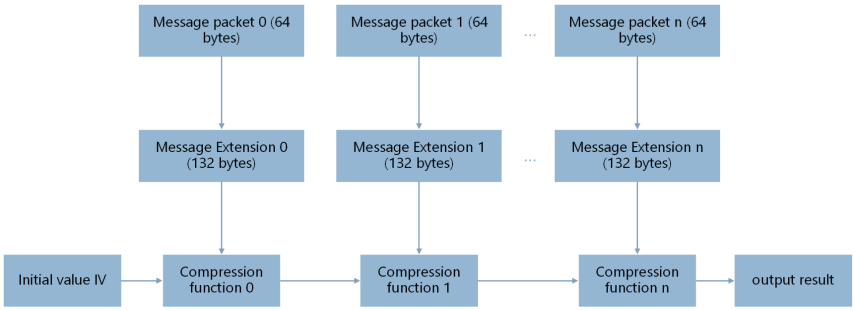

FIGURE 5. SM3 algorithm flow

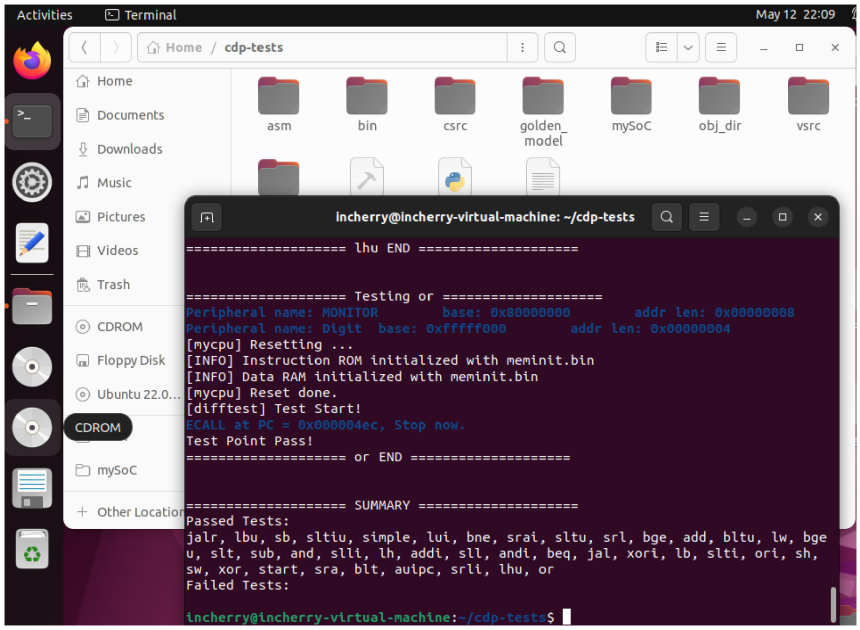

FIGURE 6. trace test results

TABLE 4. Comparison of processor performance with RISC-V architecture processors

performance	Hummingbird E203	Reference [37]	Ultrascale-RISC-V	Design of this study
instruction set	RVMAC	RVMC	RVM	RVM
Bit width/bit	32	32	32	32
Flow line depth	2	3	5	5
bus protocol	ICB	AHB-Lite	AXI	AXI
Cache	have	FIFO Implementation	have	have
defragmenter	32-bit signed multiplier	Traditional base4 32-bit signed	Traditional base8 32-bit unsigned	base4 divider 32-bit signed 32-bit unsigned
Storage Architecture	Harvard	Harvard	Harvard	Harvard
Branch forecasting	state of not working	Dynamic BTB	/	Dynamic BTFN
Slice LUTs	4.150	/	3598	17660
CoreMark/MHz	2.14	2.88	2.94	3.30

TABLE 5. Performance comparison between this processor and ARM Cortex family processors

performance	Cortex-M0+	Cortex-M3	Cortex-M4	Design of this study
instruction set	RVMAC	RVMC	RVM	RVM
Bit width/bit	32	32	32	32
Flow line depth	5	5	5	5
bus protocol	AHB-Lite	AHB-Lite	AHB-Lite	AXI
Cache	/	/	/	have
Storage Architecture	Von Neumann	Harvard	Harvard	Harvard
Slice LUTs	4463	15096	/	17660
CoreMark/MHz	2.42	3.32	3.40	3.30

V. CONCLUSION

The proposed CPU IP core fully supports the standard RV32IM instruction set without introducing custom cryptographic instructions. By optimizing the underlying arithmetic units and pipeline synergy, the design improves the execution efficiency of national cryptographic algorithms such as SM3 and SM4 through microarchitectural enhancement rather than ISA-level modification. The IP core adopts RVM instruction set combination, and focuses on the optimization of the underlying arithmetic unit and pipeline synergy: the multiplier is based on the structure of "Partial Product Generation-4:2 Compressor Multi-Level Reduction-64-bit Overpredecessor Adder" to realize product computation with improved timing efficiency; the divider adopts an enhanced radix-4 SRT algorithm, combined with two-stage encoding of leading-zero detection and dynamic iterative control logic, to realize the division operation with favorable performance–latency characteristics. The multiplier is based on the structure of "partial product generation - 4:2 compressor multi-stage reduction - 64-bit over-advancing adder" to realize multiplication with improved timing efficiency; the divider adopts the base-4 SRT algorithm, combined with two-stage encoding of leading-zero detection and dynamic iterative control logic, to realize the division operation with favorable performance–latency characteristics. Meanwhile, the IP core integrates two in-phase I-Cache, hybrid D-Cache, and BTFN static branch prediction to mitigate pipeline efficiency limitations and arithmetic latency overhead, specifically optimizing the real-time processing of high-density patterns and complex motion control algorithms in advanced production machinery.

AUTHOR CONTRIBUTIONS

Conceptualization –Yijie Cheng, Chengrui Yin, Kaixing Tang, Yuming Zhang and Yong Tan; methodology – Yijie Cheng, Kaixing Tang, Yuming Zhang and Yong Tan; investigation – Yijie Cheng, Chengrui Yin, Kaixing Tang, Yuming Zhang and Yong Tan; writing-original draft preparation – Yijie Cheng, Chengrui Yin, Kaixing Tang, Yuming Zhang and Yong Tan. All authors have read and agreed to the published version of the manuscript.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

FUNDING

This research received no external funding.

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

[1] S. Di Mascio, A. Menicucci, E. Gill, et al., "Leveraging the Openness and Modularity of RISC-V in Space," Journal of Aerospace Information Systems, vol. 16, no. 11, pp. 454–472, 2019, doi: 10.2514/1.I010735.

[2] A. Kamaleldin, S. Hesham, and D. Göhringer, "Towards a modular RISC-V based many-core architecture for FPGA accelerators," IEEE Access, vol. 8, pp. 148812–148826, 2020, doi: 10.1109/ACCESS.2020.3015706.

[3] V. A. Frolov, V. A. Galaktionov, and V. V. Sanzharov, "Investigation of RISC-V," Programming and Computer Software, vol. 47, no. 7, pp. 493–504, 2021, doi: 10.1134/S0361768821070045.

[4] A. Dörflinger, M. Albers, B. Kleinbeck, et al., "A comparative survey of open-source application-class RISC-V processor implementations," in Proceedings of the 18th ACM International Conference on Computing Frontiers, 2021, pp. 12–20, doi: 10.1145/3457388.3458657.

[5] S. Kalapothas, M. Galetakis, G. Flamis, et al., "A survey on risc-v-based machine learning ecosystem," Information, vol. 14, no. 2, p. 64, 2023, doi: 10.3390/info14020064.

[6] V. Herdt and R. Drechsler, "Advanced virtual prototyping for cyber-physical systems using RISC-V: implementation, verification and challenges," Science China Information Sciences, vol. 65, no. 1, p. 110201, 2022, doi: 10.1007/s11432-020-3308-4.

[7] J. Qu, H. Liu, R. Zhang, and L. Wen, "Design of a divider with the improved pre-processing module based on SRT-4 algorithm," in International Conference on Electronic Materials and Information Engineering (EMIE 2023), Art. no. 4, 2023, doi: 10.1117/12.3010604.

[8] B. Mehta, J. Talukdar, and S. Gajjar, "High speed SRT divider for intelligent embedded system," in 2017 International Conference on Soft Computing and its Engineering Applications (icSoftComp), pp. 1-5, 2017, doi: 10.1109/icsoftcomp.2017.8280077.

[9] X. Zheng, X. Hu, J. Zhang, et al., "An efficient and low-power design of the SM3 hash algorithm for IoT," Electronics, vol. 8, no. 9, p. 1033, 2019, doi: 10.3390/electronics8091033.

[10] H. J. Lu, R. A. Juanatas, and M. B. Abisado, "Enhancing security in instant messaging systems with a hybrid SM2, SM3, and SM4 encryption framework," PLoS One, vol. 20, no. 9, p. e0332665, 2025, doi: 10.1371/journal.pone.0332665.

[11] S. Sun, R. Zhang, and H. Ma, "Hashing multiple messages with SM3 on GPU platforms," Science China Information Sciences, vol. 64, no. 9, p. 199103, 2021, doi: 10.1007/s11432-018-9648-x.

[12] J. Chen and F. You, "An image encryption algorithm based on SM4 and Base64," Journal of Physics: Conference Series, vol. 1812, no. 1, p. 012041, 2021, doi: 10.1088/1742-6596/1812/1/012041.

[13] S. E. Abed, R. Jaffal, B. J. Mohd, and M. Alshayeji, "Performance evaluation of the SM4 cipher based on field-programmable gate array implementation," IET Circuits, Devices & Systems, vol. 15, no. 2, pp. 121-135, 2021, doi: 10.1049/cds2.12011.

[14] X. Hu, H. Yi, W. Zhang, et al., "Optimizing the SM4 Encryption Algorithm for Blockchain Security," in CCF China Blockchain Conference, Singapore: Springer Nature Singapore, 2023, pp. 31–45, doi: 10.1007/978-981-99-7743-7_3.

[15] S. Heron, "Advanced encryption standard (AES)," Network Security, vol. 2009, no. 12, pp. 8–12, 2009, doi: 10.1016/S1353-4858(10)70006-4.

[16] B. Sarkar, A. Saha, D. Dutta, et al., "A survey on the advanced encryption standard (AES): a pillar of modern security," Applied Sciences, vol. 14, no. 18, p. 8395, 2024, doi: 10.3390/app14188395.

[17] Y. Shi, Y. Cheng, L. Cui, et al., "Research on System Data Security Based on State Secret Algorithm," in 2024 IEEE 6th International Conference on Power, Intelligent Computing and Systems (ICPICS), IEEE, 2024, pp. 701–705, doi: 10.1109/icpics62053.2024.10796842.

[18] X. Xing, S. Li, R. An, et al., "A Data Migration Scheme Based on State Secret Algorithm," in 2025 Asia-Europe Conference on Cybersecurity, Internet of Things and Soft Computing (CITSC), IEEE, 2025, pp. 259–261, doi: 10.1109/citsc64390.2025.00053.