# SiWB: A 28nm 800MHz 4.2-to-14.2Gbps/W Configurable Multi-core Architecture for White-box Block Cipher with Area-efficient Random Linear Transformation and Load-aware Inter-core Scheduling

Xiangren Chen[1], Bohan Yang[1], *Member, IEEE*, Wenping Zhu[1], Hanning Wang[1], Jinjiang Yang[1], Min Zhu[2], Aoyang Zhang[1], Leibo Liu[1], *Senior Member, IEEE*

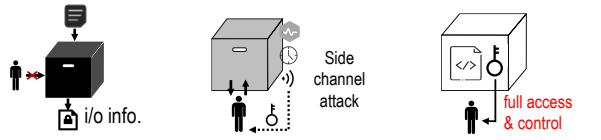
[1]Beijing National Research Center for Information Science and Technology (BNRist), School of Integrated Circuits, Tsinghua University, China. [2]Wuxi Micro Innovation Integrated Circuit Design Co., Ltd.

***Abstract*—White-box cryptography (WBC) seeks to protect secret keys even under the white-box security model that features adversaries having full control of the execution environment. Due to the ever-growing demand for content protection under security-critical scenarios, the recent progress on white-box cryptography has been nothing short of spectacular. However, the security-prioritized strategy also brings massive computation overhead to the current dedicated white-box block cipher (DWBC), seriously hindering its broad application. Compared to traditional software-only solutions, this work proposes a highly efficient domain-specific and silicon-proven processor (named SiWB) to improve the throughput (energy efficiency) by hundreds (several orders) of magnitude. First, not only a standalone crypto-core in prior work [1], we develop a complete and robust system that integrates multiple hardware cores for accelerating DWBC, user-defined high-speed interface (UPIF) for improving data transmission speed and other synergetic components. Second, to enhance throughput while maintaining energy efficiency, we devise a multi-core architecture based on inter-core load-aware scheduling, ultimately achieving 11.4Gbps peak throughput and 14.2Gbps/W energy efficiency. Third, a configurable and area-efficient datapath supporting multi-size random linear transformation (LT) and non-linear transformation (NLT) is designed to reduce resource redundancy by means of algorithm-hardware co-optimization. To our best knowledge, this the first 28nm silicon-proven hardware accelerator for white-box block ciphers, which occupies 1.8mm$^2$ area and achieves 800MHz peak frequency at 0.9V. It supports several white-box block ciphers including SPNbox-8/16/24/32, Yoroi-16/32, and WARX, delivering 389× speed-up on average versus software programs optimized with AES-NI instructions on 3.4GHz CPU. It enables white-box cryptography to be a viable solution for applications with stringent throughput requirements, such as streaming media commonly demanding several Gbps.**

***Index Terms*—white-box block cipher, multi-core architecture, random linear transformation, algorithm-hardware co-design.**

## I. Introduction

WHITE-BOX cryptography, originating from works by Chow et al. (CEJO) [2], seeks to protect secret keys even under the white-box security model where adversaries have full control of the execution environment as shown in Fig. 1. It aims to secure the block cipher against key extraction attack under white-box model. In contrast, the conventional black-box model assumes that only the external execution environment of cryptographic algorithm, such as the chosen plaintext and ciphertext pairs, is observed by the attacker. Devices on the server side usually execute under black-box model, where multiple communication responses along with secret keys are likely to be processed simultaneously. Additionally, the side-channel attack focusing on execution time, power consumption, etc. leads to the gray-box model. As the cryptanalysis advances rapidly, WBC is in fast-rising demand among many applications like digital rights management (DRM), mobile payment, IP protection [3] and so on.




| Approach | CEJO WBC | DWBC (This work) |
|---|---|---|
| Technique | LUT + encoding/ Implicit function | Space-hardness/ Incompressibility |
| Complexity | Middle | High |
| Security | Low (No secure by now) | High (Provable security) |
| Schemes | AES, DES, SM4, ARX, etc. | **SPNbox***, **Yoroi***, **WARX***, WEM, etc. |

* target ciphers with Nested Substitution Permutation Network (NSPN) structure

Fig. 1. The comparison of attacker ability among three security models.

Plenty of related works following CEJO framework are published, such as the white-box implementation of AES [2], SM4 [4] and lightweight block cipher [5]. Nevertheless, almost all of them are penetrated by key extraction or decomposition attacks up to now. Recently, another approach of white-box block cipher resorts to the variant of well-understood stand block cipher (e.g. AES) to construct the look-up table for the white-box implementation, which is referred to as dedicated white-box block cipher (DWBC) in this paper. Based on this approach, the security against key extraction and decomposition attacks in white-box mode is reported to be reduced to the well-studied key-recovery problem of standard block cipher within black-box mode [6]. Without the need to utilize external code, DWBC is more flexible to be deployed on the general execution platform. There are commonly three operation modes existing in DWBC, including *white-box mode, table-generation mode and black-box mode*. DWBC

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received April 19, 2021; revised August 16, 2021.

under white-box mode is usually executed on the embedded device equipped with software program. However, DWBC under table-generation and black-box modes still takes in a secret key and mainly perform heavier logic operations for encryption and decryption, which are usually employed on the server side (and the focus of this paper). In fact, DWBC can be regarded as an asymmetrically hard scheme [7], as its white-box and black-box modes exhibit equivalent functionality while differing in memory and computation overhead. Because the white-box implementation is generated by transforming the key-related operations of black-box implementation into large precomputed lookup tables. This transformation serves two primary objectives: first, to obscure the secret key, thereby mitigating the risk of key extraction attacks; and second, to increase the space hardness or incompressibility, enhancing resilience against code-lifting attacks [8].

For the well-known AES block cipher, although having relatively low computation overhead, extensive research has been devoted to designing specialized accelerators to accommodate the high data rate of applications such as streaming media and memory encryption [9]–[14]. Given the fundamental trade-off between security and computation efficiency in cryptographic schemes, DWBCs, operating under the strongest adversarial model, incur significantly higher computation overhead compared to conventional block ciphers. On general-purpose computing platforms like CPUs, their performance is typically constrained to only tens of Mbps, with substantial energy consumption, which underscores the need for dedicated acceleration. Although DWBCs are still undergoing rapid evolution and refinement, this work innovatively demonstrates that the performance limitation of traditional software-based solutions can be overcome through hardware-software co-optimization. Such an approach has the potential to shape the future development and practical deployment of white-box cryptography.

Previous work [1] on WBC accelerator primarily focused on single-core design and was not silicon proven. This work advances the state-of-the-art by improving the datapath and adopting a multi-core architecture with load-aware scheduling to enhance throughput while maintaining energy efficiency. As a result, the proposed design achieves Gbps-level throughput, meeting the high-speed requirements of applications such as streaming media. Furthermore, this work integrates the multi-core architecture into a System-on-Chip (SoC) and conducts fabrication and test using the TSMC 28nm HPC process. The main contributions of this paper are summarized as follows.

- We develop a complete and robust system that integrates multiple hardware cores for processing DWBCs, user-defined high-speed interface for improving data transmission speed and other synergetic components. To our best knowledge, this the first 28nm silicon-proven hardware accelerator for multiple white-box block ciphers.
- We design an out-of-order data dispatcher and an in-order data recycler to implement the load-aware inter-core scheduler. This approach is amenable to workloads of varying runtime and substantially improves the resource utilization, yielding a peak throughput of 11.4 Gbps and an energy efficiency of 14.2 Gbps/W. For the intra-core scheduling regarding inherent nested loop dependency of DWBCs, a batched-processing method is adopted to interleave data threads, thereby enhancing the parallelization of a single core while still maintaining high area efficiency.
- Bearing multi-scale random linear and diverse non-linear transformations in mind, we propose configurable and area-efficient datapaths based on insightful algorithm-hardware co-design. Starting from the LSB-first bit-serial modular multiplication algorithm, we derive a novel vectorized version that supports almost redundancy-free configuration for different bit-widths, number of modular multiplications, and irreducible polynomials. This algorithm achieves a shorter critical path compared to that of UpWB [1]. We also reduce the area footprint of the combined Sbox and constant modular multipliers without compromising performance.

The chip measurement demonstrates that the proposed hardware accelerator for DWBCs occupies 1.8mm$^2$ and achieves 800MHz peak frequency at 0.9V. It supports several DWBCs including SPNbox-8/16/24/32, Yoroi-16/32, and WARX, delivering 389× speed-up on average versus software programs optimized with AES-NI instructions on 3.4GHz CPU.

The structure of this paper is organized as follows. Section II describes the algorithmic characteristics and presents the specific design challenges. Section III presents the overall system architecture, detailing the load-aware inter-core scheduling and the batch-processing intra-core parallelization. Section IV further introduces the configurable and area-efficient datapath. Section V provides the chip test results, comparing the throughput and energy efficiency of the hardware implementation with the optimized software counterparts. Section VI concludes with a summary and outlook.

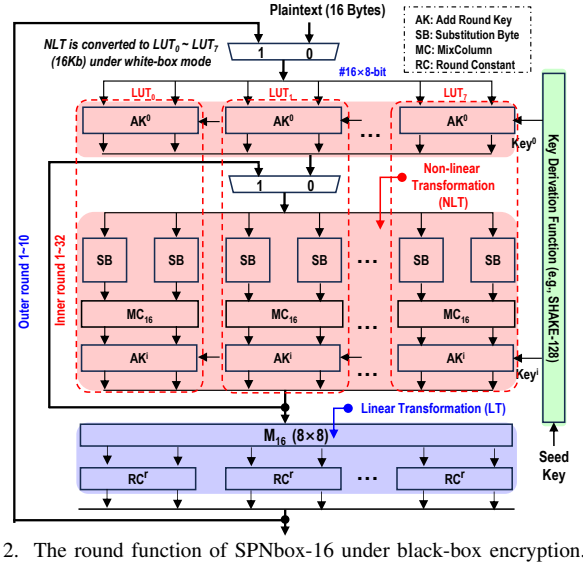



Fig. 2. The round function of SPNbox-16 under black-box encryption.

## II. Background and Motivation

### A. Algorithmic Descriptions

Since detailed specifications can be found in the literature, we focus on analyzing the data flow and operational features of target algorithms. Allowing for the trade-off between security

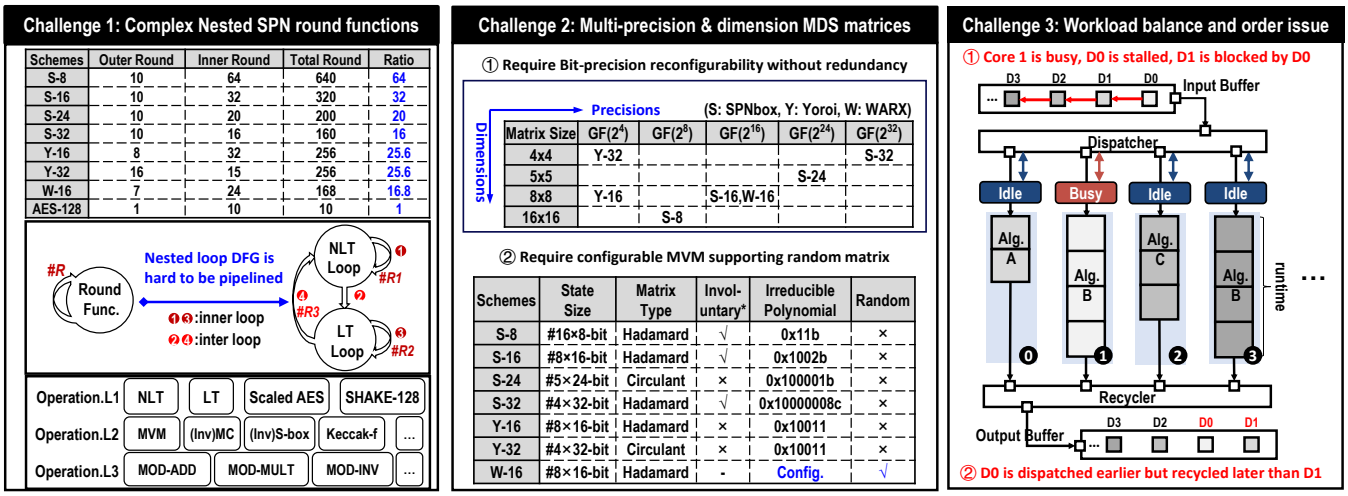



Fig. 3. Specific design challenges for supporting multiple white-box block ciphers.

and performance, one of the prevailing techniques to construct DWBC is to utilize nested SPN structure, where the S-box is developed by another small-scale variant of SPN block cipher. [15] presents the first NSPN-based white-box block cipher named SPNbox, which leverages AES variants with different sizes to build the internal S-box and applies different dimensions of MDS matrices to construct the linear layer. Subsequently, [6] puts forward an updatable white-box scheme based on the partial MDS layer, whose structure enables to refresh the white-box table without the necessity for the re-encryption of all data with the updated secret key. [16] employs addition/rotation/XOR (ARX) primitives and random MDS matrix to reduce the round counts of block cipher and improves the computation performance. Taking SPNbox-16 as an example, the round function of encryption under the black-box mode is shown in Figure 1. As can be seen, the internal SPN structure iterates 32 rounds to build the non-linear transformation and the outer round count is determined as 10. A $8 \times 8$ Hadamard MDS matrix with elements over GF($2^{16}$) is adopted to implement the linear layer, which is much larger than the MixColumn of AES.

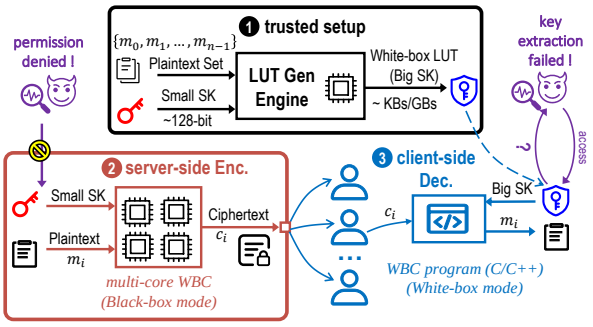



Fig. 4. The application of DWBC processor in cloud-based DRM.

## B. Demonstrative Application

Figure 4 illustrates an exemplary application of a DWBC processor for cloud-based DRM. At the trusted setup, the accelerator is used to quickly convert and hide the small secret key (SK) into the huge LUT named big SK. Then, only the server side who knows the small SK under black-box model is accelerated by the processor to enable highly concurrent applications. Note that the client side who merely knows the big SK under white-box model could use the software WBC to resist key extraction attacks, without needing hardware support. Here, we emphasize that the white-box mode is generally designed to protect software programs that are easily observed and controlled by adversaries, and it is currently less commonly applied in hardware scenarios. Therefore, this paper takes no account of hardware acceleration for the white-box mode on the client side.

## C. Design Challenges

Figure 3 outlines the possible challenges associated with designing a configurable and efficient architecture for multiple white-box block ciphers. First, white-box block ciphers feature complex nested SPN round functions, leading to massive round count, nested data flow dependency, and various underlying operations. Taking SPNbox-16 as an example, it has 10 outer rounds and 32 inner rounds, resulting in a total of 320 rounds, which is 32 times that of AES-128. From a data flow perspective, the nonlinear and linear transformations in SPNbox-16 each form inner loop dependency graph, with additional inter loop dependency between them. This nested loop structure is more complex than the single-loop dependency in traditional block ciphers, making efficient pipelining and loop unrolling more challenging, especially given the varying round rounds across different algorithms. In terms of operation breakdown, the fundamental operations of white-box block ciphers are similar to those of traditional block ciphers, but they incorporate a greater number of basic components and cryptographic primitives, including the computationally intensive key generation function SHAKE-128.

The linear layer of white-box block ciphers represents a significant difference from that of traditional block ciphers. In particular, the MDS matrices employed in NSPN-based white-box block ciphers are generally of higher dimension, which tends to degrade execution performance. Several studies

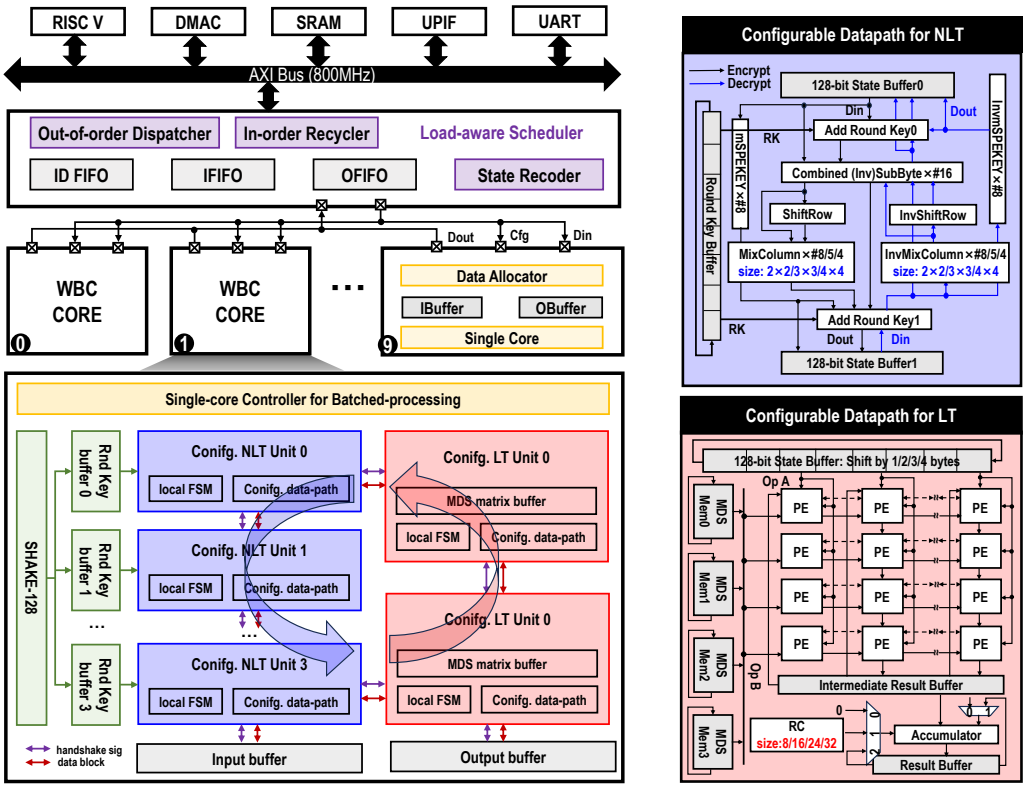



Fig. 5. The overall system architecture for white-box block ciphers.

[16]–[18] even adopt dynamic MDS matrices to enhance substantial security or reduce round counts for block cipher. A plethora of works are carried out to either construct lightweight MDS matrices [19]–[21] or optimize implementation of the prescribed linear mapping for low latency and low area footprint [22], [23], [23]. However, a configurable and efficient hardware design for dynamic MDS matrices with different sizes is rarely investigated. Indeed, to implement the linear mapping with optimal circuit depth and the smallest circuit area is a proven NP-hard [24] problem. Finding a sub-optimal implementation solution based on a variety of heuristics is still time-consuming, especially for high-dimension matrix [21]. [25], [26] propose flexible units for Mixcolumn with different irreducible polynomials and field elements, which cannot handle large-scale matrix-vector multiplication (MVM).

Due to the near-linear relationship between dynamic power consumption and frequency in CMOS chips, increasing the clock frequency of a single hardware core to improve system throughput is constrained by the power wall. As evidenced by the historical development of microprocessors, processor frequency growth has slowed dramatically since 2003, and the industry has shifted toward multi-core scaling to improve performance. Therefore, our core design goal is to achieve high energy efficiency while maximizing single-core area efficiency and flexibility. On top of that, we scale up overall system throughput using a multi-core architecture. However, similar to multi-core microprocessors, how to design an efficient scheduling strategy remains a significant challenge. As shown in Figure 3, the scheduler needs to handle workload balance to avoid resource underutilization and enable effective throughput scaling. For instance, multiple cores may be configured to execute different algorithms, each with varying execution times. If a task at the front of the queue is waiting for a busy core, it may block the tasks behind it, even if their target cores are idle. An effective solution to this issue is to adopt the out-of-order dispatch mechanism. Specifically, each core has a reservation station to buffer blocked tasks, allowing later tasks to proceed if their cores are ready. The system dynamically monitors the core status. Once operands and hardware resources become available, the buffered task is dispatched for execution without delay. Another issue is maintaining the correct order of resulting data. Since the execution time of different algorithms varies, earlier-dispatched data blocks may complete later than those dispatched afterward, possibly leading to state incoherence issues. Therefore, we also introduce an in-order retrieval strategy to ensure that data blocks are committed in the same order as they were dispatched. The proposed out-of-order dispatch and in-order retrieval will be elaborated in later sections. These techniques are architectural strategies independent of algorithm types.

## III. Overall Multi-core Architecture

Figure 5 illustrates the proposed system architecture, comprising the RISC-V processor, DMA (Direct Memory Access) controller, and other components within the SoC, along with the multi-core WBC architecture. A UPIF is utilized to facilitate high-speed communication between the SoC and external devices (e.g., FPGA), ensuring efficient plaintext-ciphertext switching. The WBC architecture consists of a top-level load-aware scheduler and ten parallel processing cores. Each core

integrates a batch-processing controller, a SHAKE-128-based round key generator, four configurable nonlinear transformation units, two configurable linear transformation units, and input-output buffers. The data path within each core adopts a round-based design paradigm to exploit the inherent byte-level parallelism of block ciphers. The configurable nonlinear transformation units incorporate tower-field-based combined (Inv)SubByte and a unified multi-size forward and inverse MixColumns, maximizing resource reuse to reduce area overhead. The configurable linear transformation units support multi-scale, multi-precision matrix-vector multiplications. This section details the load-aware inter-core scheduler and the batched-processing intra-core controller, while the next section elaborates on the NLT and LT units.

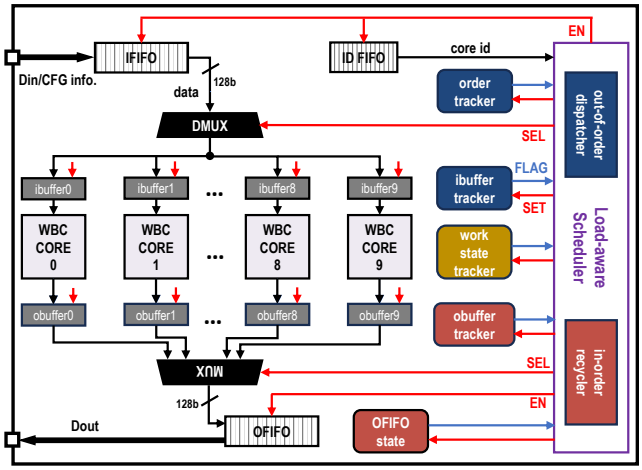



Fig. 6. The overall multi-core architecture including the load-aware scheduler, I/O FIFOs and several status trackers.

### A. Load-aware Inter-core Scheduler

To achieve a balanced workload distribution across parallel cores and ensure the efficient dispatch and retrieval of data blocks, a load-aware inter-core scheduler is designed to enhance system throughput and resource utilization, which is further comprised of an out-of-order dispatcher and in-order recycler. As shown in Figure 6, the overall multi-core architecture mainly consists of a top load-aware scheduler, several status trackers, I/O FIFOs and ten independent cores. The out-of-order dispatcher dynamically assigns data blocks to available hardware cores once both an idle core and a ready data block are detected, leveraging the status tracker for coordination. Compared to sequential dispatching, out-of-order processing mitigates pipeline stalls caused by busy cores, thereby balancing the workload and improving overall throughput. The in-order recycler, in contrast, ensures that resultant data blocks are retrieved in the order in which they are dispatched.

Figure 7 presents the concrete flow diagram of the load-aware scheduling mechanism that contains the out-of-order dispatch, intra-core batched processing, and in-order retrieval. At the dispatch phase, the data blocks from IFIFO are sorted by the id number from ID FIFO, which are further sent to the available input buffer according to the status of the ibuffer tracker. Then, the controller assigns a sequence value to the current batch of data and stores it in the order_tracker[id] for subsequent in-order data retrieval. If the number of accumulated data entries in the ibuffer reaches the expected quantity for this batch, the status of ibuffer_tracker[id] is updated to ”full” (=1). The system then further checks whether the status of work_state tracker[id] is idle (=0). If the core is idle, it initiates data loading for computation; otherwise, it enters a waiting state. During the batched-processing period, the status of work_state tracker[id] is first updated to ”ready” (=1), indicating that the core is in the warm-up phase, during which it simultaneously loads data and performs computation. Once the required number of data entries for this batch has been loaded, the work_state tracker[id] is updated to ”busy” (=2), and the status of ibuffer tracker[id] is changed to ”empty,” signifying that the ibuffer is ready to accept the next batch of data. The core then continues processing data in batches, as described in greater detail in Section B. Upon completion of the batch processing, the system enters the final in-order retrieval phase. The results are stored in the corresponding obuffer, and the obuffer tracker[id] status is updated to ”full” (=1). The system then checks whether the order tracker[id] hits the sequence value stored in the current ofifo_state. If they match, the data starts to be popped from the obuffer to the OFIFO; otherwise, the system enters a waiting state. Once the number of accumulated data entries in the OFIFO matches the expected count for this batch, the obuffer tracker[id] is updated to ”empty” (=0), and the work_state tracker[id] is updated to ”idle” (=0), indicating that the core is now available to process the next batch of data.

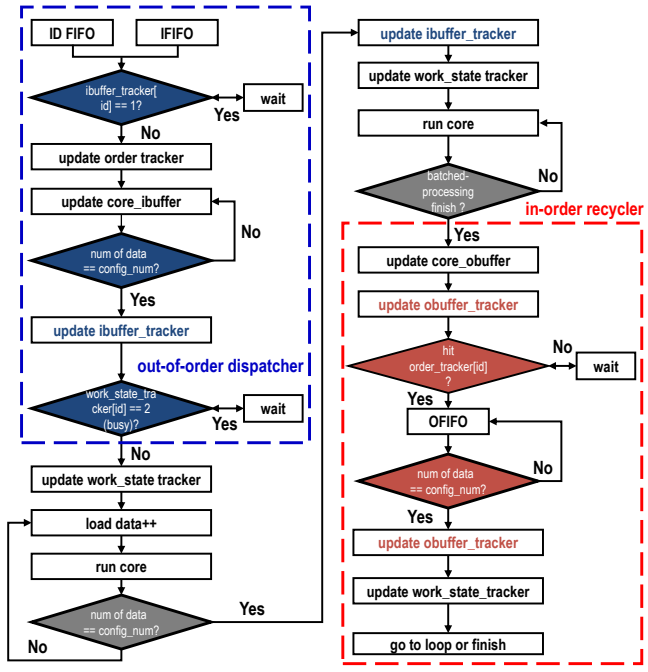



Fig. 7. The concrete load-aware scheduling mechanism for out-of-order dispatch and in-order retrieval.

The inter-core scheduling mechanism is designed to ensure that once both operands and computation resources are available, data processing begins immediately, minimizing idle

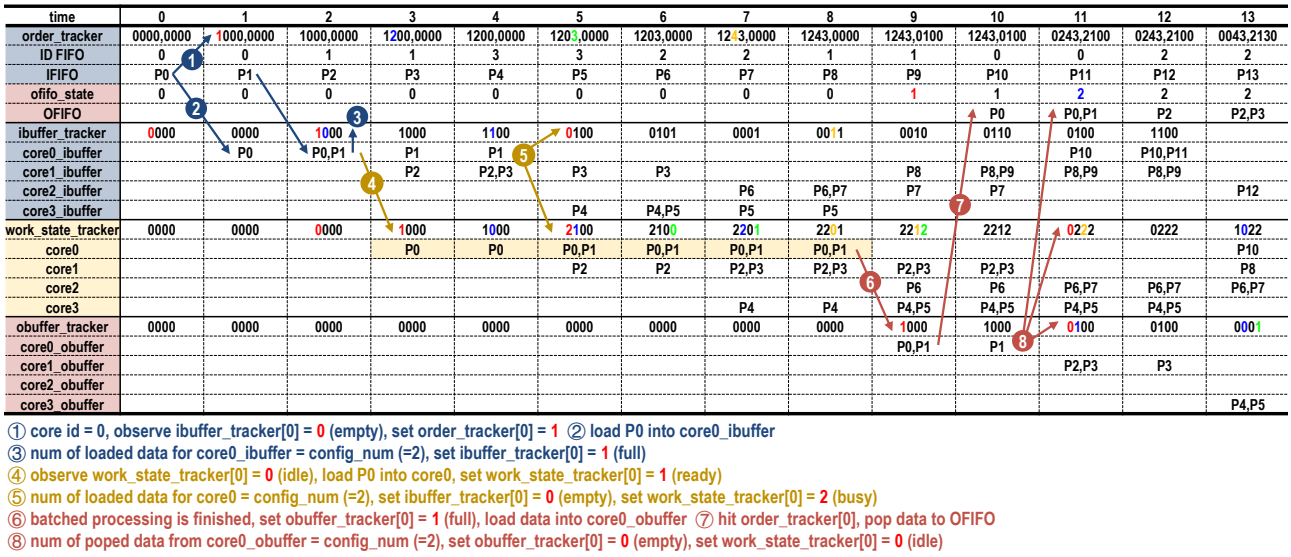



Fig. 8. An examplary timing diagram for load-aware inter-core scheduling. Assume the number of core is 4, the number of data point processed per batch is 2, the period of batched-processing is 6 cycles. The notations describe the entire life span of plaintexts P0, P1 including the dynamic dispatch, batched-processing and in-order retrieval.

time and optimizing system throughput. It also guarantees that, regardless of the varying processing times of individual cores, data is retrieved in the order of its dispatch. Given that plaintext data are independent, only spatial resource conflicts need to be addressed. To illustrate how the multi-core architecture processes multiple batches, Figure 8 presents a cycle-level timing diagram using a toy example. This simplified model assumes 4 parallel cores, with 2 data points per batch and a batched-processing period of 6 cycles. The diagram details the complete life span of the first batch (P0, P1, ID0), from allocation to core processing, result generation, and final retrieval. The process is broken down into 8 steps, in line with the scheduling mechanism shown in Figure 7, with annotations indicating the events and signal updates at each step. Additionally, the diagram displays the life span of 6 other batches, which are processed consecutively by the respective cores and retrieved in the allocated order, forming a continuous streaming processing.

### B. Workload Pattern Analysis

Through an in-depth analysis of various workload patterns, we observe that for a fixed number of cores, the achievable throughput of multi-core parallelism is mainly determined by several factors. These include the algorithm type assigned to each core, the execution time of each algorithm, the uniformity and variability of the core ID distribution across data packets (i.e., the load distribution pattern), and the output buffer (obuffer) size. These factors jointly influence the waiting time during in-order data recycle. In particular, under an out-of-order dispatch mechanism, the waiting time is strongly correlated with the load distribution and obuffer size, which can significantly affect overall throughput. It is worth noting that the core ID corresponding to each data packet is both application-dependent and known in advance. Hence, static scheduling at the software level can be applied to maintain a balanced load distribution. We present several representative patterns and, through variable-controlled experiments and detailed comparisons, demonstrate how different design strategies and configuration modes affect the throughput of the SiWB multi-core architecture. Figures 9-13 further illustrate, using a 4-core example, the influence of six distinct factors on workload patterns, which will be elaborated in the following.

| | uneven degree of load distribution | load distribution is random ? | out-of-order dispatch ? | in-order recycle ? | runtime of each core is same ? | obuffer size |
|---|---|---|---|---|---|---|
| pattern 0 | 0 | × | × | √ | √ | 0 |
| pattern 1 | 0 | × | √ | √ | √ | 0 |

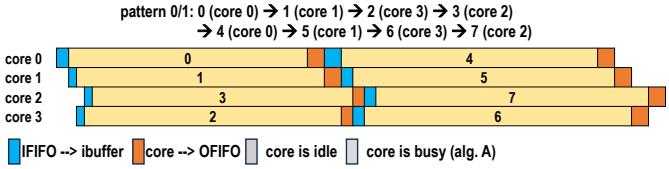



Fig. 9. The configuration and timing diagram for pattern 0 and 1.

**Insight 1.** Figure 9 illustrates the timing diagrams of Pattern 0 and Pattern 1. The only difference between them lies in whether the out-of-order dispatch mechanism is enabled, while the total latency for processing eight data packets remains identical, i.e., T(pattern0) = T(pattern1). In the table, an *uneven degree of load distribution* equal to zero indicates a perfectly balanced allocation, meaning that each newly arrived batch of four packets is distributed to four distinct cores. This balanced distribution can be achieved through static scheduling at the software level. A *fixed load distribution* implies that the subsequent batch of packets is assigned to the same cores as the previous one.

The *obuffer* serves as a local cache between each core's output and the OFIFO. When the *obuffer size* is zero, the core output directly connects to the OFIFO without any intermediate buffering. If multiple cores are configured with the same algorithm type and the workload is both balanced and fixed, the completion order naturally matches the dispatch

order. Consequently, each packet can be retrieved immediately once its corresponding core finishes execution, eliminating waiting overhead during in-order collection.

Under such simple circumstance, the penalty time for ordered recycle is zero, resulting in nearly full hardware utilization. When the number of incoming packets is sufficient and data transmission time is negligible compared with computation time, the system throughput scales almost linearly with the number of cores. The limiting factor then stems from the impact of DMUX and MUX size on the critical path during dispatch and collection.

| | uneven degree of load distribution | load distribution is fixed ? | out-of-order dispatch ? | in-order recycle ? | runtime of each core is same ? | obuffer size |
|---|---|---|---|---|---|---|
| pattern 2 | 0 | × | × | √ | √ | 0 |
| pattern 3,4 | 0 | × | √ | √ | √ | 0 |
| pattern 5 | 0 | × | √ | √ | √ | 1 |

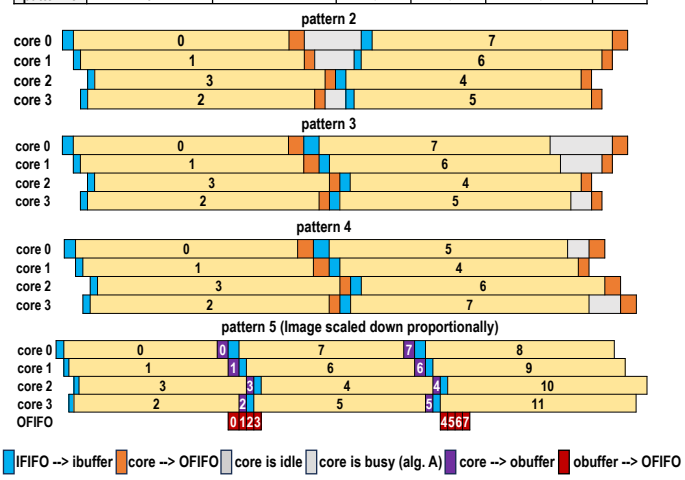



Fig. 10. The configuration and timing diagram for pattern 2-5.

**Insight 2.** Figure 11 illustrates the timing diagrams of Pattern 2 to Pattern 5. Among them, Pattern 2 and Pattern 3 exhibit identical load distributions that are both balanced and dynamic, with an *obuffer size* of zero. Dynamic load means that the load distribution is not fixed. The only distinction between the two lies in whether the out-of-order dispatch mechanism is enabled. As observed, unlike Pattern 0/1, the combination of dynamic load distribution and in-order recycle constraint introduces idle periods in certain cores. Nevertheless, the overall latency remains identical, that is, T(pattern2) = T(pattern3) >T(pattern0/1). In this scenario, the out-of-order dispatch mechanism fails to enhance throughput. Although it allows earlier operand fetching and execution compared with in-order dispatch, the absence of an *obuffer* prevents immediate result commitment. Consequently, each core must wait until preceding results are written back before committing its own output and receiving new packets for processing.

**Insight 3.** The only difference between Pattern 3 and Pattern 4 lies in the load distribution of the second batch of data packets. In both cases, the distribution remains balanced and supports out-of-order processing with *obuffer = 0*. It can be observed that the idle periods of cores in Pattern 4 are shorter. As a result, the overall latency is reduced, i.e., T(pattern4) <T(pattern3). This demonstrates that even when workloads are balanced, variations in the load distribution across different batches can lead to differences in overall throughput.

**Insight 4.** Patterns 4 and 5 share the same load distribution. The only difference is the presence of an *obuffer* component, which is crucial for exploiting the benefits of the out-of-order dispatch mechanism. In Pattern 5, the inclusion of the *obuffer* allows cores to immediately commit result data upon completion. Cores can then simultaneously receive new packets for processing. The *obuffer* effectively decouples the cores from the OFIFO, enabling computation and in-order recycle to proceed in parallel. Compared with Patterns 2, 3, and 4, idle periods in Pattern 5 are almost completely eliminated. As a result, hardware utilization approaches 100%. Under these conditions, system throughput scales nearly linearly with the number of cores.

| | uneven degree of load distribution | load distribution is fixed ? | out-of-order dispatch ? | in-order recycle ? | runtime of each core is same ? | obuffer size |
|---|---|---|---|---|---|---|
| pattern 6 | 1 | √ | × | √ | √ | 0 |
| pattern 7 | 1 | √ | √ | √ | √ | 0 |

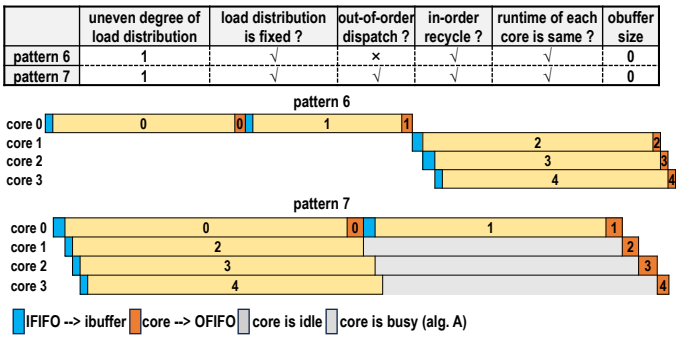



Fig. 11. The configuration and timing diagram for pattern 6 and 7.

**Insight 5.** As shown in Figure 11, Patterns 6 and 7 investigate the impact of uneven load distribution on overall throughput, assuming no *obuffer* is present. In the table, an *uneven degree of load distribution* equal to 1 indicates that, in the first batch of four dispatched packets, two consecutive packets (0 and 1) are both assigned to Core 0. Under in-order dispatch, the subsequent packets (2, 3, 4) cannot be sent until packets 0 and 1 are fully processed. In Pattern 7, which employs out-of-order dispatch, cores can receive and process packets earlier than in Pattern 6. However, the in-order recycle constraint still forces the corresponding cores to remain idle for certain periods. Overall, the total latency of the two patterns is identical, i.e., T(pattern6) = T(pattern7). This demonstrates that, in the absence of an *obuffer*, in-order and out-of-order dispatch achieve essentially the same performance under the imbalanced workload distribution.

**Insight 6.** Figure 12 illustrates the timing diagrams of Patterns 8 to 10. Pattern 8 exhibits the same load imbalance as Pattern 7 and also supports out-of-order dispatch. The key difference is that Pattern 8 has an *obuffer size* of 1. This allows the *obuffer* to store a single result. As a result, Cores 1–3 can continue receiving and processing new packets while the in-order recycle constraint is effectively mitigated. Consequently, hardware utilization approaches 100%, and overall throughput scales nearly linearly with the number of cores.

However, when the *uneven degree of load distribution* equals 2, the first three dispatched packets (0–2) are all assigned to Core 0. This creates a more severe load imbalance. Since the *obuffer* can store only one result, Core 1 must wait

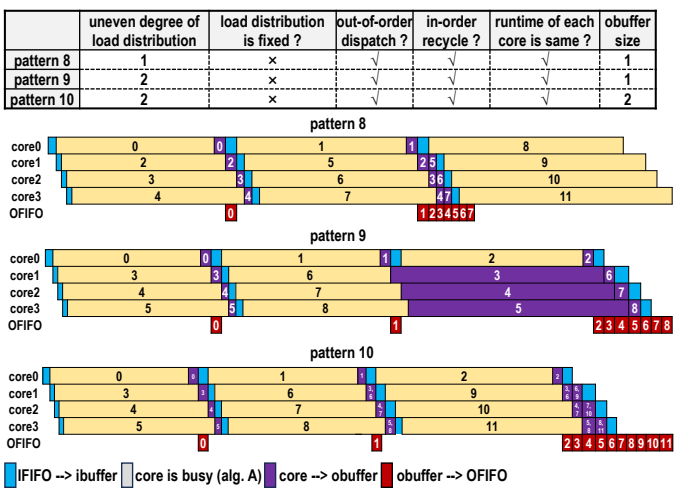



Fig. 12. The configuration and timing diagram for pattern 8-10.

until the packet 3 in the *obuffer* is committed before it can receive new packets after processing packet 6. In this situation, an *obuffer size* of 1 is insufficient to tolerate an uneven degree of 2, leading to prolonged idle periods. The solution is to increase the *obuffer size* to 2. With this configuration, the *obuffer* can store two result packets, which effectively eliminates idle time. This behavior is illustrated in Pattern 10.

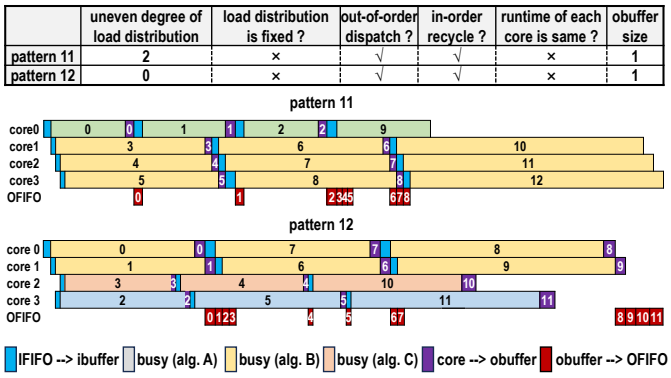



Fig. 13. The configuration and timing diagram for pattern 11 and 12.

**Insight 7.** Figure 13 illustrates the timing diagrams of Patterns 11 and 12, examining the impact of concurrently executing different algorithms. In Pattern 11, the *uneven degree of load distribution* remains 2. However, the algorithm executed by Core 0 has a shorter execution time, which mitigates the effect of load imbalance. When Core 1 completes processing packet 6, packet 3 in the *obuffer* has already been committed to the OFIFO. Therefore, packet 6 can be immediately stored in the *obuffer*. In this scenario, an *obuffer size* of 1 is sufficient to tolerate a load imbalance of degree 2, resulting in nearly 100% core utilization.

**Insight 8.** Pattern 12 illustrates a scenario in which cores are configured to more types of algorithms. For workloads with a uniform distribution, an *obuffer size* of 1 is sufficient to achieve nearly 100% hardware utilization. Even when the execution times of algorithms differ across cores, no penalty time due to in-order recycle is observed.

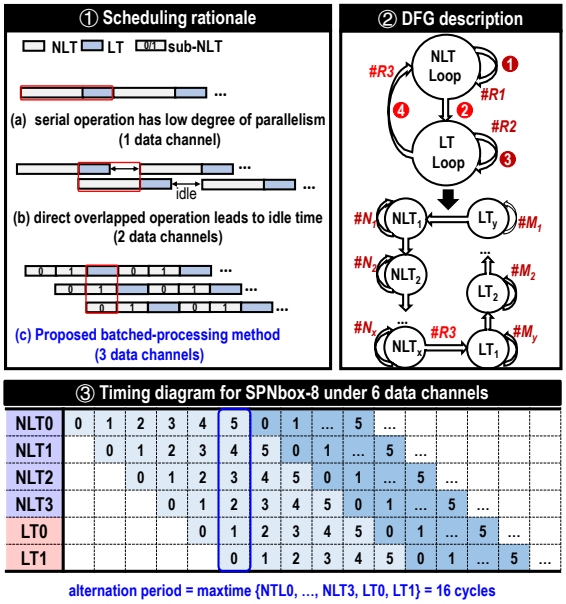



Fig. 14. Adopted batched-processing method.

**Conclusion.** Since the load distribution can be statically scheduled at the software level, the multicore performance evaluation in this work primarily corresponds to the conditions of Pattern 5. Under these conditions, the workload is both uniformly distributed and variable, and out-of-order dispatch is supported. The introduction of the *obuffer* eliminates waiting time due to in-order recycle, resulting in near-100% core hardware utilization. Consequently, the overall system throughput scales almost linearly with the number of cores, thereby justifying the net benefit of our scheduling mechanism.

### C. Batched-processing Intra-core Controller

To address the previously mentioned design challenge, we adopt the efficient batched-processing method like UpWB, which interleaves data threads to exploit parallelism and improve throughput. Specifically, we divide the single NLT loop into four sub-loops, each handling one-quarter of the total iterations, and similarly divide the LT loop into two sub-loops, each handling half of the iterations. The goal is to ensure that the number of iterations in each sub-loop is as equal as possible, or in a multiple relationship, so that the maximum parallelism is achieved, with each batch capable of interleaving 6 data points for processing.

Figure 14 demonstrates that, compared to serial computation or direct overlap methods, our batched-processing approach achieves the highest parallelism with minimal area overhead. This method can be generally described using a data flow graph, which decomposes the nested loop dependency graph into multiple sub-loops for interleaved processing. Taking the SPNbox-8 encryption as an example, Figure 14 presents the timing diagram based on the batched-processing approach. A total of 6 independent data blocks are alternately processed across 4 NLT units and 2 LT units, with the alternation period being the maximum computation time among these 6 units. Since the inner loop count for SPNbox-8 is 64, each NLT sub-loop handles 16 iterations, which exactly matches the clock

cycles for each LT sub-loop. This method minimizes the idle time, achieving near 100% hardware utilization.

## IV. Configurable and Efficient Datapath

This section presents comprehensive discussions of the configurable and efficient datapath within each core, including both NLT and LT units. Bearing diverse algorithms and parameters in mind, insightful hardware-algorithm co-design approaches are explored to develop the desired computation components. In order to achieve high area efficiency, redundant resources are minimized through extensive reuse without sacrificing performance.

### A. Vector Modular Multiplier

The modular addition and subtraction over binary Galois field are implemented by bit-wise XOR operation, whereas the modular multiplication consists of bit-wise AND operation and extra modular reduction by an irreducible polynomial. To tackle Challenge 2, it is essential to design a vector modular multiplier that allows flexible bit-width decomposition, thereby facilitating support for random MDS matrices with varying dimensions and precision requirements. Unlike plain multiplication, modular multiplication involves additional reduction operations. Therefore, decomposing a single wide-width modular multiplier into multiple narrow-width ones is not a straightforward task. Previous ECC-oriented works focus solely on optimizing single wide-width modular multipliers, while AES-oriented approaches only involve customized constant multipliers. Neither of them supports redundancy-free bit-width decomposition or configurable numbers of multipliers. UpWB [1] proposes a vectorized modular multiplier based on Montgomery reduction, which meets the design requirements discussed here. However, it requires each matrix element to be pre-multiplied by the Montgomery factor. In this work, we adopt an alternative vectorized computation unit based on bit-serial modular multiplication, which achieves a shorter critical path compared to that of UpWB.

Algorithm 1 presents the conventional least-significant-bit-first (LSB-first) bit-serial modular multiplier, which requires $m$ iterations. Each iteration consists of two steps: a multiply-accumulate operation and an operand update. In contrast to the conventional radix-2 Montgomery algorithm, the two operations in each iteration of Algorithm 1 can be executed in parallel, thus resulting in a shorter critical path. Building upon this algorithm, we derive a new vectorized modular multiplication (algorithm 2) through the following steps. **Step 1:** Considering that operands in white-box block ciphers are predominantly 128 bits wide, we further partition all operands in Algorithm 1 (except for $b(x)$) into chunk vectors with 8-bit granularity. **Step 2:** We apply $k$-level unrolling to the for-loop in Algorithm 1, such that each iteration includes $k$ instances of multiply-accumulate and operand update operations. At this time, each iteration will scan the vector Vec_B by size of 16×k-bits, and the vectors Vec_A and Vec_F by size of 16×8-bits. **Step 3:** We define a configurable update function to dynamically support both different vector sizes and bit-widths modular reduction, which are employed by MDS matrices of varying scales. This function includes five specific configurations: 16×8-bit, 8×16-bit, 5×24-bit, 4×32-bit, and 16×4-bit. Each configuration resorts to a distinct bit-width of modular reduction, which can be realized by appropriately configuring the source of the most significant bit (MSB) for each computation unit.

**Algorithm 1** Bit-serial modular multiplication over GF($2^m$)

**Input:** $a(x)$, $b(x)$, $f(x)$. $b(x) = (b_0, b_1, ..., b_{m-1})$
**Output:** $c(x) = a(x) \cdot b(x) \cdot r(x)^{-1} \bmod f(x)$, $r(x)^{-1} \cdot r(x) \bmod f(x) = 1$.

```
c(x) = (0, 0, ..., 0)
for i = 0 to m − 1 do
    c(x) = c(x) + b_i · a(x)
    a(x) = a(x) · x mod f(x)
end for
return c(x)
```

Specifically, Figure 15 illustrates the vectorized radix-2 Montgomery modular multiplier from UpWB and the vectorized radix-2 LSB-first modular multiplier proposed in this paper. Both circuits set the parameter $k$ in Algorithm 2 to 4. In the right diagram, each tPE unit contains a 4-to-1 multiplexer, which is used to select the input source for the MSB of the PE, thereby enabling the different configuration cases of the update function in Algorithm 2. The detailed descriptions of the circuit in the left diagram can be found in reference [1]. In this context, we focus on highlighting the key differences between the two circuits. First, in the circuit on the right, the critical path for each PE consists of 4-to-1MUX + AND + XOR, which is shorter than the critical path for each PE on the left, which includes 2AND + 2XOR + 2-to-1MUX. Considering that each column contains $k$ PEs, this difference in critical path delay becomes more pronounced. Furthermore, each PE in the last row of the circuit on the right exhibits a smaller area overhead compared to its counterpart in the left circuit. Taking the above advantages into account, the circuit proposed in this work (shown on the right) can achieve clearly higher area efficiency than that of UpWB.

### B. Combined (Inv)Sbox

Efficient hardware design for AES Sbox is a well-known objective [27], [28]. A direct lookup table implementation is likely to achieve the lowest latency, but it naturally incurs a higher area overhead. In this work, we adopt a round-based design approach, which instantiates 16 forward and inverse Sbox units, and therefore places greater emphasis on area efficiency. For such cases, we would like to implement the parallel Sbox units by using logic gates over composite fields, and carefully explore the reuse of operations between the forward and inverse Sbox to reduce area overhead. This is because the main operation of the AES SBox is taking the inverse of a field element, which naturally is its own inverse, and we expect that many gates of the two circuits can be shared. Numerous prior works have explored gate-level implementations of Sbox with a focus on minimizing area and latency. In 2005, Canright et al. [29] investigated the importance of the representation of the subfield, testing many different isomorphisms that led to the smallest area design. This construction is perhaps the most cited and used implementation of an area-constrained combined AES SBox.

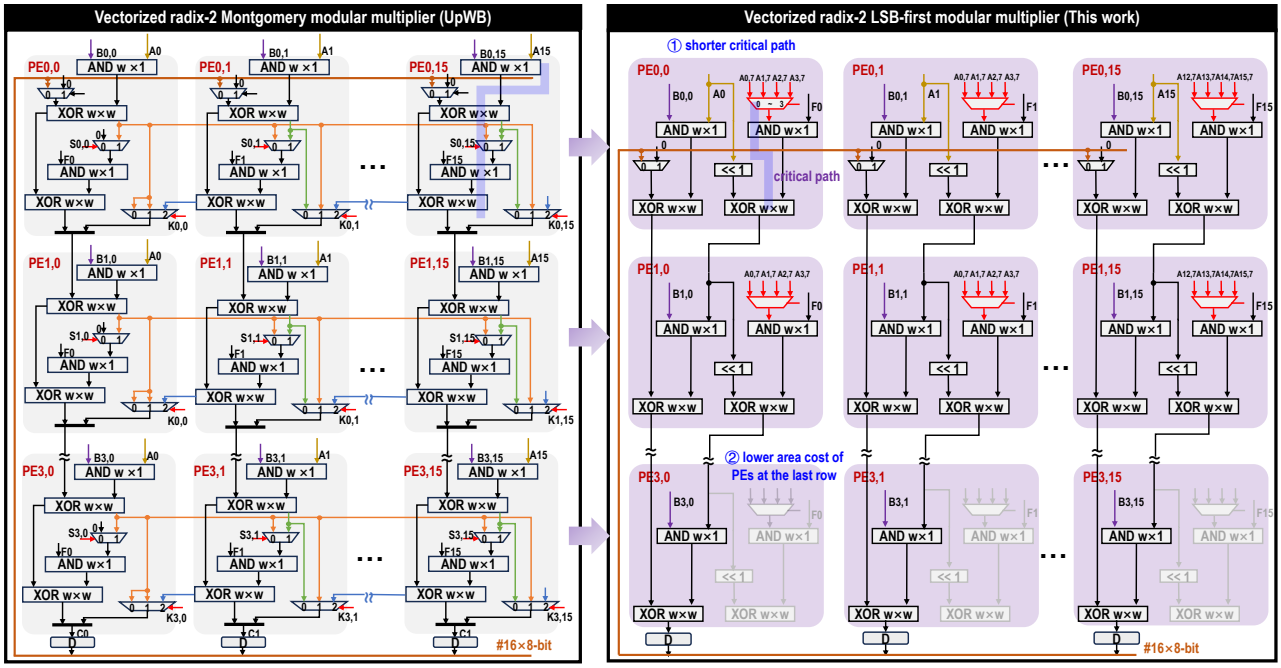



Fig. 15. Comparison of VMM between UpWB (prior work) and SiWB (this work).

Following the notation used in [29] to build the tower field representation, the irreducible polynomials, roots, and normal basis are shown in Figure 16. The derivation process for the expression of inverting a general element $A = a_0Y + a_1Y^{16}$ over $GF(2^8)$ is as follows:

$$\begin{aligned} A^{-1} &= (AA^{16})^{-1}A^{16} \\ &= ((a_0Y + a_1Y^{16})(a_1Y + a_0Y^{16}))^{-1}(a_1Y + a_0Y^{16}) \\ &= ((a_0^2 + a_1^2)Y^{17} + a_0a_1(Y^2 + Y^{32}))^{-1}(a_1Y + a_0Y^{16}) \\ &= ((a_0 + a_1)^2Y^{17} + a_0a_1(Y^2 + Y^{32}))^{-1}(a_1Y + a_0Y^{16}) \\ &= ((a_0 + a_1)^2WZ + a_0a_1)^{-1}(a_1Y + a_0Y^{16}) \end{aligned}$$

Thus, the final step can be viewed done over $GF(2^4)$, which can be organized as:

$$\begin{aligned} &t_1 = (a_0 + a_1) \quad t_2 = (WZ)t_1^2 \quad t_3 = a_0a_1 \quad t_4 = t_2 + t_3 \\ &t_5 = t_4^{-1} \quad t_6 = t_5a_1 \quad t_7 = t_5a_0 \end{aligned}$$

The inversion result is obtained as $A^{-1} = t_6Y + t_7Y^{16}$. Merging the base conversion and linear part of inversion results in the Top linear transformation. Merging the base conversion and affine transformation of Sbox yields the Bottom linear transformation. Finally, the forward Sbox circuit from prior work [29] is shown in Figure 16 (a). Building on the baseline structure proposed in that work, we adopt a more efficient implementation of the Top linear transformation to reduce area overhead. Additionally, the MULT×2 and Bottom linear transformation are merged into a single module comprising 32 NAND2 and 8 XOR4 gates, effectively reducing the circuit depth. The resulting architecture is shown in Figure 16 (b). The implementation details can be found in the Python model provided in the referenced link. The comparison among our used technique, LUT-based and famous design [29] of combined S-box is also shown in Figure 17. As can be seen, our design achieves the lowest area overhead, with latency falling between the two compared approaches, ultimately resulting in the highest area efficiency.

| Tower Field | Irreducible Poly. | Root | Filed of Coefficients | Normal Base |
|---|---|---|---|---|
| GF($2^2$) | $x^2+x+1$ | W | GF(2) | [W, $W^2$] |
| GF($2^2$)$^2$ | $x^2+x+W^2$ | Z | GF($2^2$) | [$Z^2$, $Z^8$] |
| GF((($2^2$)$^2$)$^2$ | $x^2+x+WZ$ | Y | GF(($2^2$)$^2$) | [Y, $Y^{16}$] |

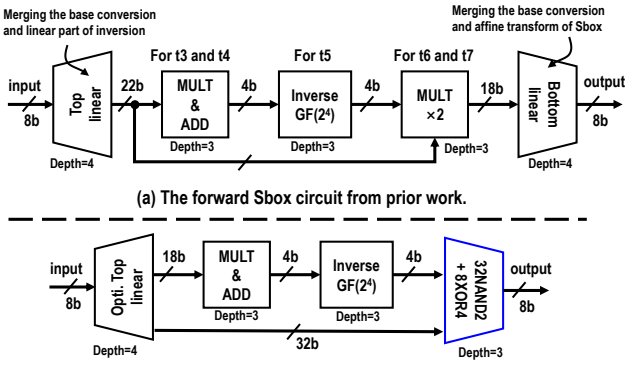



Fig. 16. The tower field based Sbox circuit using depth-decreased method.

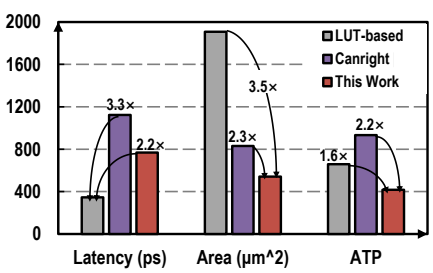



Fig. 17. Area-efficiency comparison about combined Sbox.

**Algorithm 2** Proposed vectorized radix-2 LSB-first modular multiplier

**Input:** $(A_0, A_1, ..., A_{15})$ is 8-bit chunk vector of 128-bit plaintext. $(F_0, F_1, ..., F_{15})$ is 8-bit chunk vector of irreducible polynomial. $(B_0, B_1, ..., B_{15})$ is 8-bit chunk vector from MDS matrix, where $B_{i,j}$ denotes the $i$-th bit of $j$-th chunk. The MODE signal configures the size of vector multiplier for different sizes of linear transformation without redundancy.
**Output:** $(C_0, C_1, ..., C_{15})$ is 8-bit chunk vector of 128-bit ciphertext.

**FUNCTION Update_Vec_A** $((A_0, A_1, ..., A_{15})$, MODE, $(F_0, F_1, ..., F_{15}))$
  **case** (MODE)
  **#16 × 8**: $A_0 = A_0 \cdot x \bmod F_0, A_1 = A_1 \cdot x \bmod F_1, ..., A_{15} = A_{15} \cdot x \bmod F_{15}$ //8-bit modular reduction
  **#8 × 16**: $(A_0, A_1) = (A_0, A_1) \cdot x \bmod (F_0, F_1), (A_2, A_3) = (A_2, A_3) \cdot x \bmod (F_2, F_3), ...,$
    $(A_{14}, A_{15}) = (A_{14}, A_{15}) \cdot x \bmod (F_{14}, F_{15})$ //16-bit modular reduction
  **#5 × 24**: $(A_0, A_1, A_2) = (A_0, A_1, A_2) \cdot x \bmod (F_0, F_1, F_2), (A_3, A_4, A_5) = (A_3, A_4, A_5) \cdot x \bmod (F_3, F_4, F_5), ...,$
    $(A_{13}, A_{14}, A_{15}) = (A_{13}, A_{14}, A_{15}) \cdot x \bmod (F_{13}, F_{14}, F_{15})$ //24-bit modular reduction
  **#4 × 32**: $(A_0, A_1, A_2, A_3) = (A_0, A_1, A_2, A_3) \cdot x \bmod (F_0, F_1, F_2, F_3), ...,$
    $(A_{12}, A_{13}, A_{14}, A_{15}) = (A_{12}, A_{13}, A_{14}, A_{15}) \cdot x \bmod (F_{12}, F_{13}, F_{14}, F_{15})$ //32-bit modular reduction
  **#16 × 4**: $A_0[3:0] = A_0[3:0] \cdot x \bmod F_0[3:0], A_1[3:0] = A_1[3:0] \cdot x \bmod F_1[3:0], ...,$
    $A_{15}[3:0] = A_{15}[3:0] \cdot x \bmod F_{15}[3:0]$ //4-bit modular reduction
  **endcase**
**return** $(A_0, A_1, ..., A_{15})$

$(C_0, C_1, ..., C_{15}) = (0, 0, ..., 0)$ //scan #16×k-bit of Vec_B per cycle.
**for** $i = 0$ to $\frac{m}{k} - 1$ **do**
  $(C_0, C_1, ..., C_{15}) = (C_0, C_1, ..., C_{15}) + (B_{ki,0}, B_{ki,1}, ..., B_{ki,15}) \cdot (A_0, A_1, ..., A_{15})$ //For Vec_C at row 0
  $(A_0, A_1, ..., A_{15})$ =**Update_Vec_A** $((A_0, A_1, ..., A_{15})$, MODE, $(F_0, F_1, ..., F_{15}))$
  $(C_0, C_1, ..., C_{15}) = (C_0, C_1, ..., C_{15}) + (B_{ki+1,0}, B_{ki+1,1}, ..., B_{ki+1,15}) \cdot (A_0, A_1, ..., A_{15})$ //For Vec_C at row 1
  $(A_0, A_1, ..., A_{15})$ =**Update_Vec_A** $((A_0, A_1, ..., A_{15})$, MODE, $(F_0, F_1, ..., F_{15}))$
  ...
  $(C_0, C_1, ..., C_{15}) = (C_0, C_1, ..., C_{15}) + (B_{ki+k-1,0}, B_{ki+k-1,1}, ..., B_{ki+k-1,15}) \cdot (A_0, A_1, ..., A_{15})$ //For Vec_C at row (k-1)
  $(A_0, A_1, ..., A_{15})$ =**Update_Vec_A** $((A_0, A_1, ..., A_{15})$, MODE, $(F_0, F_1, ..., F_{15}))$
**end for**
**return** $(C_0, C_1, ..., C_{15})$

### C. Constant Modular Multiplier

In SPNbox and Yoroi algorithms with different parameters, forward and inverse MixColumns units of varying scales are required. This motivates us to design a unified and compact hardware circuit. The left side of Figure 18 illustrates three matrix sizes used in multi-scale forward MixColumns. Given that smaller-scale matrices are submatrices of larger ones, a large number of reuse opportunities can be exploited following the UpWB approach to reduce area overhead. The right side of Figure 13 shows the resulting matrices obtained by applying the additive and multiplicative matrix decomposition techniques from UpWB. However, the key difference from UpWB lies in our optimized design for constant modular multiplications by 0x68 and 0xD1, which further reduces the area overhead while ensuring minimal circuit depth.

Figure 19 illustrates the three-step automated flow for generating constant modular multipliers. The first step is to input the irreducible polynomial, the finite field, and the constant. The second step uses Mastrovito's method to convert the constant into an $n \times n$ binary matrix. In the third step, unlike the backward heuristic search used in UpWB, we adopt an exhaustive search approach to construct the constant modular multiplication circuit, resulting in a solution with lower area overhead. Since the binary matrices are only of size $8 \times 8$, the runtime overhead of exhaustive search remains acceptable. Figure 20 compares the area and depth of the 0x68 and 0xD1 constant modular multipliers (CMMs) generated by the direct method, the UpWB method, and our proposed method. Our 0x68 CMM has an area cost of 14 XOR gates, and the 0xD1 CMM uses 17 XOR gates, representing 1.3× and 1.1× reductions in area compared to those in UpWB, respectively, while maintaining the same circuit depth.

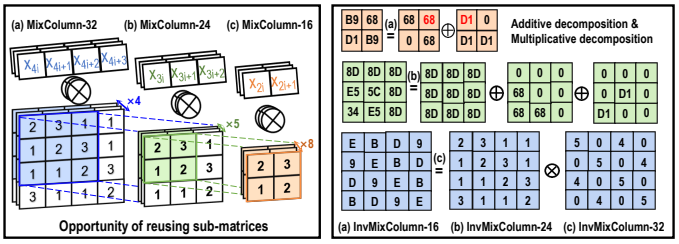



Fig. 18. Implementation schemes for Multi-scale (Inv)MixColumns.

## V. Silicon Result and Comparison

### A. Implementation Result

SiWB, the proposed DWBC processor based on multi-core architecture, is fabricated in TSMC HPC 28nm technology. Figure 21 shows the overall die photograph, which has an area of 5.8mm$^2$. The packaging type is ELQFP, and the operating voltage range is from 0.7V to 1.1V. The entire die includes the 10-core-based SiWB processor, PLL, test circuitry, IO pads, and other components within the SoC system. The standalone SiWB processor occupies 1.8 mm$^2$ and consumes 5KB of FIFO and 3.4M NAND2 equivalent gates. The SiWB processor supports vector modular multiplication with five different bit widths to perform multi-scale random linear transformations. It employs a highly area-efficient composite-filed combined S-box and uses SHAKE-128 as the round key generation function. Thanks to intensive optimizations for the NLT and LT units, the critical path of the cryptographic core is yet located within the computation units, but in the state machine controller. The chip test results at room temperature show that the frequency of SiWB processor ranges from 400 MHz

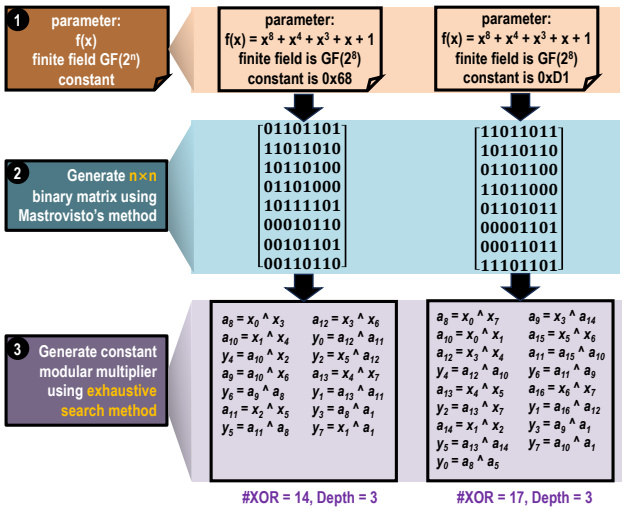



Fig. 19. CMM generator and its application to 0x68 and 0xD1.

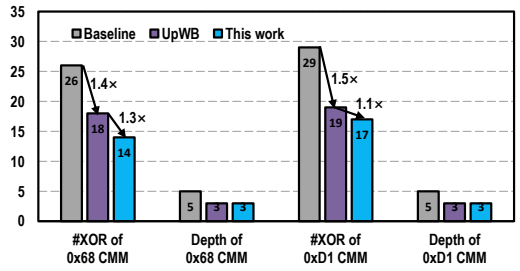
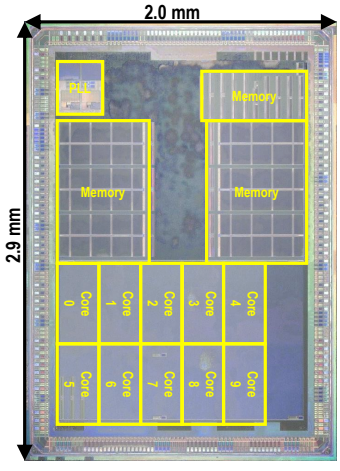



Fig. 20. Evaluations on the optimized CMMs.

at 0.7V to 1100 MHz at 1.1V, with an average operational power spanning from 190 mW at 0.7V to 418 mW at 0.9V. We also measure the system power as a reference, which includes leakage power, operational power, test circuit power, and peripheral power. To the best of our knowledge, SiWB is the first silicon-proven multi-core processor that supports multiple white-box block ciphers. In contrast, the previous UpWB work is validated solely through simulation and DC synthesis, and is limited to a single-core architecture.




| Chip Specifications | |
|---|---|
| Technology | TSMC 28nm HPC |
| Supply voltage | 0.7 ~ 1.1V |
| Package | ELQFP |
| Die size | 2.0mm × 2.9mm |
| SRAM | 204 KB |
| **Cryptographic Cores** | |
| Cores area | 1.8mm$^2$ |
| FIFO | 5KB |
| Logic gates | 3.4M (NAND2 equiv.) |
| Data width of VMM | 4/8/16/24/32-bit |
| Combined S-box | GF((2$^2$)$^2$)based |
| KDF for DWBC | SHAKE-128 |
| Max. Frequency (MHz) | 400 @ 0.7V<br>1100 @ 1.1V |
| Avg. Power (mW) | 190$^{\#1}$/350$^{\#2}$ @ 0.7V<br>418$^{\#1}$/820$^{\#2}$ @ 0.9V |

**#1:** operation power **#2:** system power (including leakage power, operation power, test circuit power, peripheral power)
*All evaluations are under room temperature 25°C

Fig. 21. Die photography and chip specifications.

The left side of Figure 22 presents the throughput results for seven algorithms in both encryption and decryption modes, executed on the 10-core DWBC processor and a 3.4 GHz CPU platform. The DWBC processor achieves an average throughput of 7.8 Gbps, with a peak throughput of 11.4 Gbps, which corresponds to the encryption throughput of SPNbox-32. The second-highest throughput is 10.8 Gbps, corresponding to the encryption throughput of WARX. The lowest throughput, at 3.5 Gbps, is observed for the decryption throughput of SPNbox-24. This can be attributed to the fact that, in terms of algorithm parameters, SPNbox-32 has the fewest round count, and its sparse linear layer is designed in a standalone manner, whereas SPNbox-24 involves more rounds and employs a dense MDS matrix for decryption. The software implementations of the relevant algorithms, as referenced in [6], [15], were optimized using AES-NI instructions. The right side of Figure 22 shows that the throughput acceleration ratio of the DWBC processor relative to the software ranges from 282× to 546×, with an average acceleration ratio of 382×. The left side of Figure 23 presents the energy efficiency of seven algorithms in both encryption and decryption modes. The DWBC processor achieves an average energy efficiency of 8.8 Gbps/W, with a peak throughput of 14.2 Gbps, corresponding to the encryption energy efficiency of SPNbox-32. The lowest energy efficiency is 4.2 Gbps/W, corresponding to the decryption energy efficiency of SPNbox-16. The right side of Figure 23 shows that the energy efficiency ratio of the DWBC processor relative to the software ranges from 8200× to 14,800×, demonstrating a significant improvement.

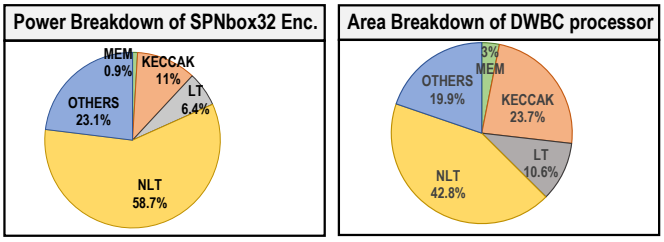



Fig. 24. Power and area breakdown for SiWB.

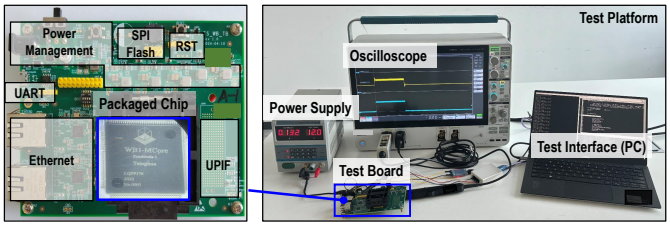



Fig. 25. Test setup and measurement environment.

Figure 24 provides a breakdown of the power consumption for SPNbox-32 encryption and the area distribution for the DWBC processor, which includes components such as NLT, LT, KECCAK, and others. Among these, the NLT part exhibits the highest power consumption and occupies the largest area. Figure 25 shows the practical setup and measurement picture

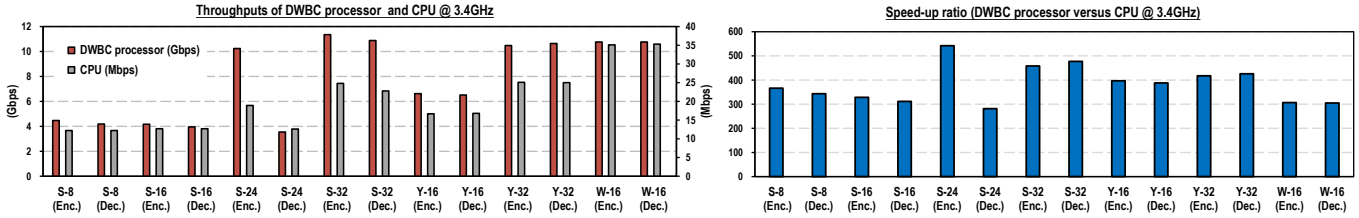



Fig. 22. Evaluations on the throughput of SiWB DWBC processor.

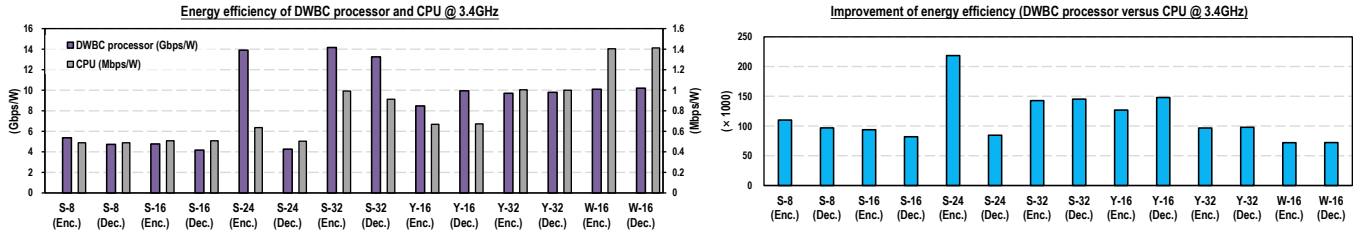



Fig. 23. Evaluations on the energy efficiency of SiWB DWBC processor.

of SiWB processor, which includes the oscilloscope and test interface. Figure 26 illustrates the variation in frequency, power consumption, throughput, and energy efficiency of SPNbox-32 as the voltage increases. It is evident that the processor operates reliably within a voltage range of 0.7V to 1.1V, with throughput varying from 5.4 Gbps to 11.4 Gbps and energy efficiency ranging from 9.1 Gbps/W to 14.2 Gbps/W.

### B. Comparisons among Silicon-proven Works

Many previous works on hardware implementations of block ciphers merely report area and performance results halting at the simulation phase. Without actual silicon validation, some works often obtain more optimistic outcomes, ignoring the substantial impact of factors like layout placement, routing and process corner. Therefore, our comparisons are mainly made with silicon-proven works. Table I presents a comparison of SiWB with related works in terms of performance, flexibility, and energy efficiency. Considering the impact of process nodes, we normalize the energy efficiency following the method within reference [30]. As the first simulation-based work for DWBCs, UpWB can be evaluated on FPGA but is limited to a single cryptographic core. It yet consider supporting LUT-gen functionality or round configuration as well. UpWB only reports the performance of the standalone accelerator and is not integrated into a complete SoC system for evaluation. Reference [25] proposes a multi-core architecture for the AES algorithm with configurable algorithm parameters. However, its flexibility in the linear layer remains insufficient, supporting only modulus configuration with obsolete throughput and energy efficiency. Reference [9] proposes a specialized two-stage pipelined architecture aimed at achieving high throughput and energy efficiency for AES encryption and decryption. Nevertheless, the flexibility remains limited and unexplored. Reference [31] introduces a reconfigurable low-power cryptographic chip based on in-memory computation, supporting classical AES, ECC point multiplication, and the Keccak hash function. However, its flexibility of the linear layer leaves much room for further improvement, and its throughput is relatively much low. The proposed multi-core processor supports LUT-gen and en/decryption for 7 DWBCs with high normalized energy-efficiency, offering 5 dimensions of flexibility and round configuration that are not completely covered in other accelerators. Compared to the single DWBC core on FPGA [1], about 30.5× improvement on throughput is achieved by our fabricated processor. SiWB adopts an efficient multi-core architecture to significantly boost throughput while maintaining energy efficiency, effectively addressing the needs of plentiful real-world scenarios.

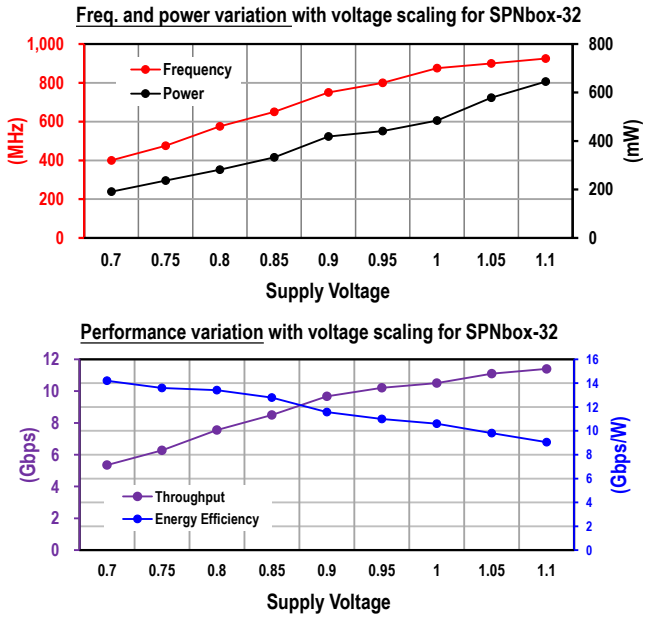



Fig. 26. The influence of voltage scaling on SiWB processor.

### C. Discussions on Security

The security of cryptographic accelerators mainly relies on the security property of the implemented algorithms and the

TABLE I
COMPARISONS BETWEEN SIWB WITH RELATED WORKS ABOUT PERFORMANCE, FLEXIBILITY AND ENERGY-EFFICIENCY

| | | TVLSI'10 [25] | JSSC'11 [9] | JSSC'17 [31] | Software | CHES'24 [1] | This Work |
|---|---|---|---|---|---|---|---|
| Platform | | ASIC | ASIC | ASIC | i7-6700 CPU | FPGA | ASIC |
| Technology | | 250nm | 45nm | 40nm | 14nm | 16nm | 28nm |
| Frequency (MHz) | | 66 | 2100 | 28.8 | 3400 | 240 | 800 |
| Voltage (V) | | 0.7 | 1.1 | 0.7 | — | — | 0.9 |
| Scheme (DWBC Support) | | Congfi. AES-128/192/256 (×) | AES-128/192/256 (×) | AES, MM(ECC), Keccak (×) | All (software) (√) | 7 DWBCs, SHAKE-128 (√) | 7 DWBCs, SHAKE-128 (√) |
| Mode | | Enc / Dec | Enc / Dec | Enc | Enc / Dec | Enc / Dec | Enc / Dec |
| LUT-gen Support | | — | — | — | √ | × | √ |
| Supported Round Count | | 10/12/14 | 10/12/14 | 10 | √ | 160 ~ 640 | **< 1024 (Config.)** |
| Supported Flexibility of LT | Square Matrix Size | × | — | × | √ | √ | √ (4/5/8/16) |
| | Modulus F(x) | √ (deg-8) | | √ | | √ | √ (deg-4/8/16/24/32 |
| | Precision | × | | √ | | √ | √ (4/8/16/24/32-bit) |
| | Random Matrix | × | | — | | √ | √ |
| | On-the-fly Config. | × | | × | | √ | √ |
| Multi-core Support | | √ (1 ~ 3) | × | × | | × | **√ (1 ~ 10)** |
| Round Config. | | √ | × | × | | × | √ |
| Kernel Area ($mm^2$) | | 6.29 | 0.15 | 0.128 | — | — | 1.8 |
| Average Power (mW) | | 259.1 | 125 | 0.28 #3 | — | — | 190 #6/350 #7@0.7V 640 #6/820 #7@0.9V |
| Throughput (Gbps) | | 0.84/0.70/0.61 | 53/44/38 | 0.005 #3 | 0.012 ~ 0.035 #5 | 0.58 | 14.8@1.1V #1 12.6@0.9V |
| Energy Efficiency (Gbps/W) | | 3.2/2.7/2.3 | 424/352/304 | 18.2 | — | — | 38.8@0.7V #2 30.1@0.9V |
| Norm. Energy Efficiency #4 (Gbps/W) | | 11.1 #3 | 102.4 #3 | 1.6#3 | — | — | 38.8 |

**#1:** Peak throughput for SPNbox-32 with 10 cores working **#2:** Peak enenerg effi. for SPNbox-32 with 10 cores working **#3**: Data for AES-128
**#4:** Norm. Energy-effi. = Energy-effi. / ($X^2$ $Y^2$ Z) where X = 28nm / used tech., Y = 0.9V / used voltage, Z = 160 / used round count, 160 is round count of SPNbox-32
**#5:** Software throughput = 8-bit × 3400MHz / CPB, CPB is cycle per byte.
**#6:** operation power **#7:** system power (including leakage power, operation power, test circuit power, peripheral power)

physical protections against side-channel attacks (SCAs). The SiWB processor focuses on enhancing throughput and energy efficiency, making it suitable to be deployed on server-side or other scenarios. Thus, the resilience against time-relevant SCAs or simple power attacks is taken into account. The execution time of all algorithms on the SiWB processor is independent of any secret data and remains constant, making it inherently resistant to certain time-dependent SCAs. Furthermore, this work implements a highly area-efficient random linear layer, enabling the MDS matrix to be randomly and dynamically updated. By randomly and uniformly distributing the hamming weight of elements, dynamic matrix tends to be more immerse towards SCAs, especially to the power related attacks [17], [32].

Additionally, our design supports configurable round count, which aids in resisting the recently discovered hybrid code-lifting attack [33]. This attack involves collaboration between white-box and black-box attackers to reverse-engineer the program, compromising the security of schemes that were originally secure under the assumption of a single white-box attacker. Although white-box block ciphers are still evolving, our work tends to exert downstream effects. We hope that this acceleration will encourage broader applications of white-box cryptography, contributing to the development of advanced cryptographic infrastructures.

## VI. CONCLUSION

For the first time, this work presents a silicon-proven white-box block cipher processor that is integrated into a complete SoC system, rather than just a standalone cryptographic accelerator. An energy-efficient and high-throughput acceleration platform is a prerequisite for widespread applications of white-box block ciphers that have much heavier computation overheads than traditional block ciphers. Although previous work UpWB attempts decent hardware acceleration, the throughput improvement in a single-core architecture remains limited and is still insufficient for real-time applications such as streaming media. Building on UpWB, this work makes remarkable optimizations and really tapes out the processor named SiWB. At the architectural level, a load-aware scheduler is designed to enable out-of-order dispatch and in-order retrieval for the multi-core architecture. This scheduling approach effectively enhances the resource utilization of the multi-core system, ultimately improving throughput while maintaining energy efficiency. Without loss of generality, the proposed two scheduling strategies are independent of algorithm types, offering potential applicability and extendability to other scenarios. At the circuit level, we employ algorithm-hardware co-design to construct a more area-efficient vector modular multiplier. We also implement a composite-field combined S-box, achieving a better balance between area and latency. Additionally, we develop cost-reduced constant modular multipliers, further lowering the overall area overhead.

# REFERENCES


[1] Xiangren Chen, Bohan Yang, Jianfeng Zhu, Jun Liu, Shuying Yin, Guang Yang, Min Zhu, Shaojun Wei, and Leibo Liu. Upwb: An uncoupled architecture design for white-box cryptography using vectorized montgomery multiplication. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):677–713, 2024.

[2] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[3] Lubos Gaspar, Viktor Fischer, Tim Güneysu, and Zouha Cherif Jouini. Two IP protection schemes for multi-fpga systems. In *2012 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2012, Cancun, Mexico, December 5-7, 2012*, pages 1–6. IEEE, 2012.

[4] Tingting Lin, Manfred von Willich, Dafu Lou, and Phil Eisen. A dca-resistant implementation of SM4 for the white-box context. In Christophe Hauser, Yonghwi Kwon, and Sebastian Banescu, editors, *Checkmate@CCS 2021, Proceedings of the Research on offensive and defensive techniques in the Context of Man At The End (MATE) Attacks, Virtual Event, Republic of Korea, 19 November 2021*, pages 21–33. ACM, 2021.

[5] Alex Charlès and Chloé Gravouil. Review of the white-box encodability of nist lightweight finalists. Cryptology ePrint Archive, Paper 2022/804, 2022. https://eprint.iacr.org/2022/804.

[6] Yuji Koike and Takanori Isobe. Yoroi: Updatable whitebox cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):587–617, 2021.

[7] Alex Biryukov and Léo Perrin. Symmetrically and asymmetrically hard cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 417–445. Springer, 2017.

[8] Estuardo Alpirez Bock, Alessandro Amadori, Chris Brzuska, and Wil Michiels. On the security goals of white-box cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):327–357, 2020.

[9] Sanu K. Mathew, Farhana Sheikh, Michael Kounavis, Shay Gueron, Amit Agarwal, Steven K. Hsu, Himanshu Kaul, Mark A. Anders, and Ram K. Krishnamurthy. 53 gbps native $\mathrm{gf}(2^4)^2$ composite-field aes-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors. *IEEE Journal of Solid-State Circuits*, 46(4):767–776, 2011.

[10] Sheng-Jung Yu, Yu-Chi Lee, Liang-Hsin Lin, and Chia-Hsiang Yang. An energy-efficient double ratchet cryptographic processor with backward secrecy for iot devices. *IEEE Journal of Solid-State Circuits*, 58(6):1810–1819, 2023.

[11] Utsav Banerjee, Andrew Wright, Chiraag Juvekar, Madeleine Waller, Arvind, and Anantha P. Chandrakasan. An energy-efficient reconfigurable dtls cryptographic engine for securing internet-of-things applications. *IEEE Journal of Solid-State Circuits*, 54(8):2339–2352, 2019.

[12] Raghavan Kumar, Sachin Taneja, Vivek De, and Sanu K. Mathew. A 4.7-to-5.3-gb/s fault-injection and side-channel attack-resistant aes-256 engine using masked isomorphic composite fields in intel 4 cmos. *IEEE Journal of Solid-State Circuits*, 60(4):1349–1358, 2025.

[13] Yiqun Zhang, Kaiyuan Yang, Mehdi Saligane, David Blaauw, and Dennis Sylvester. A compact 446 gbps/w aes accelerator for mobile soc and iot in 40nm. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pages 1–2, 2016.

[14] Sanu Mathew, Sudhir Satpathy, Vikram Suresh, Mark Anders, Himanshu Kaul, Amit Agarwal, Steven Hsu, Gregory Chen, and Ram Krishnamurthy. 340 mv–1.1 v, 289 gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt gf(2 4 ) 2 polynomials in 22 nm tri-gate cmos. *IEEE Journal of Solid-State Circuits*, 50(4):1048–1058, 2015.

[15] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[16] Jun Liu, Vincent Rijmen, Yupu Hu, Jie Chen, and Baocang Wang. WARX: efficient white-box block cipher based on ARX primitives and random MDS matrix. *Sci. China Inf. Sci.*, 65(3), 2022.

[17] Muhammad Yasir Malik and Jong-Seon No. Dynamic MDS matrices for substantial cryptographic strength. *CoRR*, abs/1108.6302, 2011.

[18] Tran Thi Luong, Nguyen Ngoc Cuong, and Hoang Duc Tho. On the calculation of input and output for dynamic mds matrices in diffusion layer of spn block ciphers. In *2017 International Conference on Information and Communications (ICIC)*, pages 281–287, 2017.

[19] Daniel Augot and Matthieu Finiasz. Direct construction of recursive MDS diffusion layers using shortened BCH codes. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2014.

[20] Siang Meng Sim, Khoongming Khoo, Frédérique E. Oggier, and Thomas Peyrin. Lightweight MDS involution matrices. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 471–493. Springer, 2015.

[21] Ayineedi Venkateswarlu, Abhishek Kesarwani, and Sumanta Sarkar. On the lower bound of cost of MDS matrices. *IACR Trans. Symmetric Cryptol.*, 2022(4):266–290, 2022.

[22] Qun Liu, Weijia Wang, Yanhong Fan, Lixuan Wu, Ling Sun, and Meiqin Wang. Towards low-latency implementation of linear layers. *IACR Trans. Symmetric Cryptol.*, 2022(1):158–182, 2022.

[23] Zejun Xiang, Xiangyong Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(2):120–145, 2020.

[24] Joan Boyar and Rene Peralta. New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Paper 2009/191, 2009. https://eprint.iacr.org/2009/191.

[25] Mao-Yin Wang, Chih-Pin Su, Chia-Lung Horng, Cheng-Wen Wu, and Chih-Tsun Huang. Single- and multi-core configurable AES architectures for flexible security. *IEEE Trans. Very Large Scale Integr. Syst.*, 18(4):541–552, 2010.

[26] Leibo Liu, Bo Wang, Chenchen Deng, Min Zhu, Shouyi Yin, and Shaojun Wei. Anole: A highly efficient dynamically reconfigurable crypto-processor for symmetric-key algorithms. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(12):3081–3094, 2018.

[27] Doaa Ashmawy. Performance evaluation, comparison and improvement of the hardware implementations of the advanced encryption standard s-box. *Electronic Thesis and Dissertation Repository*, 2020.

[28] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES sboxes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):91–125, 2019.

[29] David Canright. A very compact s-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[30] Dongyun Kam, Sangbu Yun, Jeongwon Choe, Zhengya Zhang, Namyoon Lee, and Youngjoo Lee. 2.8 A 21.9ns 15.7 gbps/mm$^2$ (128,15) BOSS FEC decoder for 5g/6g URLLC applications. In *IEEE International Solid-State Circuits Conference, ISSCC 2024, San Francisco, CA, USA, February 18-22, 2024*, pages 50–52. IEEE, 2024.

[31] Yiqun Zhang, Li Xu, Qing Dong, Jingcheng Wang, David Blaauw, and Dennis Sylvester. Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security. *IEEE Journal of Solid-State Circuits*, 53(4):995–1005, 2018.

[32] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels - . In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2016.

[33] Yosuke Todo and Takanori Isobe. Hybrid code lifting on space-hard block ciphers application to yoroi and spnbox. *IACR Trans. Symmetric Cryptol.*, 2022(3):368–402, 2022.