Aspect-Oriented Fault Injection for the Non-Intrusive Resilience Evaluation of Cryptographic Hardware Models

Hassen Mestiri
Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj, Saudi Arabia
h.mestiri@psau.edu.sa (corresponding author)

Mariem Bouraoui
Electrical Engineering Department, College of Engineering, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia
m.bouraoui@psau.edu.sa

Mohsen Machhout
Electronics and Micro-Electronics Laboratory, Faculty of Sciences of Monastir, University of Monastir, Tunisia
mohsen.machhout@fsm.rnu.tn

Received: 19 January 2026 | Revised: 11 February 2026, 14 February 2026, and 27 February 2026 | Accepted: 28 February 2026

 DOI: https://doi.org/10.48084/etasr.17646

ABSTRACT

Evaluating the resilience of sophisticated cryptographic hardware against fault injection attacks necessitates high-speed simulation environments. At the Electronic System Level (ESL), SystemC provides a valuable framework for constructing fast functional models. However, traditional fault injection techniques require direct and intrusive code modifications to these models, which complicates the security assessment process. This paper proposes a novel methodology that circumvents this limitation by leveraging Aspect-Oriented Programming (AOP). A non-intrusive fault injection and detection environment is introduced, where faults are woven into SystemC cryptographic models using AspectC++, eliminating the need for source code alterations and offering a practical alternative to complex physical cryptanalysis. This approach is validated through a thorough case study on a SystemC model of the LED lightweight algorithm, focusing on two critical aspects: the efficacy of AOP for accurate fault detection and its overhead regarding simulation performance and executable size. The results demonstrate that the proposed method effectively evaluates a design's efficiency against fault attacks. It is also confirmed that AOP integration imposes a negligible impact on simulation time, preserving the speed advantages of ESL simulation while enabling seamless security analysis.

Keywords-Aspect-Oriented Programming (AOP); LED lightweight algorithm; fault injection attacks; hardware security; data transmission security

I. INTRODUCTION

Electronic cryptographic modules are integral to embedded systems, where they protect sensitive data. These modules typically store a secret cryptographic key, which is used together with an encryption algorithm. Although such algorithms are mathematically robust and designed to resist cryptanalytic attacks, their hardware implementations can remain exposed to various physical threats. Attackers often target these devices with the aim of extracting secret keys, accessing protected information, or disrupting normal operation. Among the most effective physical attacks are fault injection attacks, which deliberately introduce errors into a device's operation to reveal confidential key material [1].

As cryptographic embedded systems grow more complex, they challenge the design and verification capabilities of engineering teams. To address this, SystemC [2, 3] is a widely adopted standard for modeling and verifying complex hardware systems, which has proven particularly suitable for simulating fault injection in hardware platforms and System-on-Chip (SoC) designs. Several techniques have been developed to model security fault attacks on cryptographic algorithms in simulation [4-7]. However, a common limitation of these

approaches is that they usually require manual modification of the SystemC source code to introduce and monitor faults. AOP offers a promising alternative to maintain the integrity of the original cryptographic implementation during testing [8], as it enables the clean separation of distinct system concerns into modular units. In the context of cryptography, three primary concerns can be isolated: (i) the cryptographic system itself, (ii) the process of injecting faults, and (iii) the mechanisms for detecting and analyzing those faults. These separate concern modules are later integrated through a weaving process, which occurs during compilation rather than during code development.

This study presents a novel method for simulating fault injection attacks on cryptographic hardware models at the ESL. Although existing AOP-based verification has focused on functional validation or assertion checking, this work innovates by applying the AOP paradigm specifically to the non-intrusive, system-level assessment of fault attack resilience. SystemC is employed for modeling, and AspectC++ is used to create an injection and detection environment. This approach allows for the evaluation of cryptographic designs against sophisticated fault models without any alteration to the core implementation, bridging a critical gap between ESL design and security verification. This work supports the development of resilient digital infrastructure, a key aspect of Sustainable Development Goal (SDG) 9, by providing a methodology to evaluate and enhance the security of cryptographic systems used in critical applications such as healthcare devices and industrial control systems, where fault injection attacks could compromise sensitive data or system availability. The main contributions of this work can be summarized as:

- Methodological Advancement: Introduces a verification paradigm decoupling security analysis from functional implementation, enabling security as a separate, weavable concern rather than intrusive modification.
- Spatio-Temporal Weaving Framework: Defines and validates the first weaving rules synchronizing fault injection with cryptographic rounds in SystemC, achieving Register-Transfer Level (RTL) temporal precision at ESL.
- Modular Security Architecture: Structures fault injection, control, and analysis as independently verifiable aspects, enabling systematic design-space exploration and reusable countermeasure prototyping.
- Empirical Validation: Demonstrates that non-intrusive AOP-based fault injection achieves identical detection accuracy to intrusive methods while preserving simulation efficiency and design integrity.

Despite extensive use of SystemC for hardware modeling and AOP for modularizing concerns, no existing approach unifies these paradigms for non-intrusive cryptographic fault analysis. Current methods either require intrusive RTL modifications or apply AOP only to functional verification. This study addresses how to conduct comprehensive fault campaigns without source code modification while maintaining simulation efficiency. This work presents the first security-oriented AOP-SystemC framework for cryptographic fault analysis, with its novelty lying in: (i) spatio-temporal weaving rules synchronizing fault injection with cryptographic rounds, (ii) modular separation of security concerns into independent aspects, and (iii) demonstrating non-intrusive fault campaigns, achieving identical detection accuracy while preserving design integrity and simulation efficiency.

II. BACKGROUND

A. Aspect-Oriented Programming

AOP modularizes crosscutting concerns (logging, security, synchronization) into aspects, isolating them from primary functionality to reduce duplication and improve maintainability [8-9]. Core constructs include: aspects encapsulating concerns; advice defining code executed before, after, or around join points; and pointcuts specifying where advice applies. Aspects weave into applications at compile-time or runtime, centralizing crosscutting logic while minimizing code entanglement. This promotes modularity, reusability, and isolated testing, making AOP ideal for managing complexity in large-scale systems.

Figure 1 illustrates this mechanism. To integrate new functionality as an aspect into an existing system, pointcuts first identify the relevant join points within the primary code. An aspect weaver then injects the specified advice at those locations, either during compilation or runtime. This mechanism enables the modification of system behavior without altering the original source code directly.

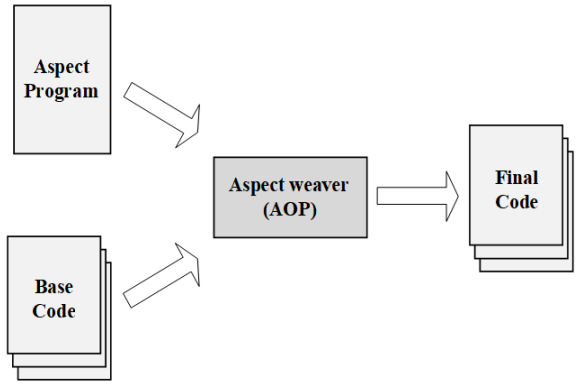

Fig. 1. Weaving the aspect module into the base code.

B. LED Algorithm

Each stage comprises a sequence of four identical rounds followed by a subkey addition operation, designated as addRoundKey (ARK). The AddRoundKey(*state*, SK_i) operation performs a bitwise XOR between the state array and corresponding nibbles of subkey SK_i while maintaining positional alignment [10]. The encryption process utilizes two operations: addRoundKey(*state*, *SKi*) and step (*state*). The iteration count (*s*) varies with key size: 8 steps for 64-bit keys and 12 steps for larger keys up to 128 bits.

III. RELATED WORK

A. AOP in Software Development

In software development, AOP is a well-established paradigm for system testing, where tools such as AspectJ and AspectC++ act as weavers to non-intrusively inject monitoring and diagnostic code into programs [11]. This enables automated fault detection and reporting for issues such as memory leaks and algorithmic errors [12]. This method has been effectively applied to embedded systems for evaluating memory usage, performance, robustness, and test coverage [13]. The primary advantage of AOP is its ability to modularize cross-cutting concerns, including logging, security, and fault tolerance, thereby enhancing code maintainability and reusability. For instance, it has been used to implement memory encryption for security hardening [6] and to structure fault-tolerant mechanisms in distributed systems [14]. Although AOP is a powerful tool for testing and improving system flexibility, its specific application for designing detailed fault-tolerant or security-critical systems remains an area with significant potential for further exploration [11-14].

B. AOP for Analyzing Hardware/Software Systems

AOP has shown promise for the analysis of complex hybrid SoC architectures blending hardware and software [15-17]. Frameworks using AspectC++ create unified specifications that describe hardware traits and software properties in modular aspects, which are woven together to form complete SoC designs—particularly useful for hybrid FPGA platforms [15-17]. Building on this, LARA introduces specialized languages and toolchains, enabling developers to augment code with instrumentation for monitoring, logging, and debugging. Through an AOP layer, LARA separates and manages hardware/software co-design concerns efficiently, streamlining optimization for FPGA-based systems [16]. Thus, AOP helps untangle intertwined hardware and software concerns, enabling cleaner development and easier maintenance. However, a gap remains: the use of AOP to specifically model or handle encrypted data pathway specialization within heterogeneous systems remains unexplored.

C. AOP for SystemC Modeling

In hardware design, AOP is used mainly for verification rather than synthesis [12]. Aspects to model timing and concurrency in SystemC have been employed, enabling synthesizable SoC descriptions [12]. While AOP facilitates extracting performance metrics, such models are typically constrained to simulation due to synthesis limitations. AOP is ill-suited for direct circuit design, as expressing low-level constructs through aspects yields models that are less efficient than VHDL or Verilog [15, 18]. The modeling of cryptographic systems presents additional challenges that require spatiotemporal precision. Spatially, aspects must interface accurately with diverse SoC elements (buses, channels, ports, registers, software threads). Temporally, fault injection and monitoring logic must align precisely with system events. SystemC's event-driven kernel provides basic synchronization, but cryptographic applications demand fine-grained coordination, fault insertion, error detection, and countermeasure activation that must tightly couple with the algorithm's operational steps [15, 18].

D. Comparative Analysis with Existing Verification Approaches

To contextualize the contributions of the proposed method, Table I presents a systematic comparison against representative existing approaches for cryptographic hardware verification. The comparison considers six key dimensions: abstraction level, intrusiveness (whether source code modification is required), scalability to complex designs, support for fault injection campaigns, verification flexibility, and reported runtime overhead. As shown in Table I, existing AOP-based approaches for SystemC [12, 15, 18] have targeted functional verification, assertion checking, or performance monitoring exclusively, without addressing fault injection attacks. In contrast, traditional fault injection methods [3-5] require intrusive RTL modifications and suffer from poor scalability. The proposed method combines the non-intrusiveness of AOP with comprehensive fault injection capabilities at ESL, achieving the first unified framework that supports systematic security evaluation without compromising design integrity or simulation efficiency.

TABLE I. COMPARISON OF CRYPTOGRAPHIC HARDWARE VERIFICATION APPROACHES

Approach	Abstraction level	Intrusiveness	Scalability	Fault injection support	Verification flexibility	Runtime overhead
RTL Fault Injection [3-5]	RTL	High	Low	Yes	Low	High
HDL Assertion-Based [7]	RTL	Medium	Medium	No	Medium	Medium
Standard SystemC [4, 6]	ESL	High	Medium	Yes	Low	Low
AOP Performance Monitoring [12,15]	ESL/Software	None	High	No	High	Low
AOP Functional Verification [18]	ESL	None	High	No	High	Low
This work (SystemC-AOP Security)	ESL	None	High	Yes	High	Low

IV. LED SYSTEMC-AOP ENVIRONMENT

A. RTL-Level Security Verification Constraints

This work addresses the challenge of verifying fault detection mechanisms in LED cryptographic hardware. Although testing at the RTL offers precision, it imposes two major constraints. First, it requires intrusive modifications to the Hardware Description Language (HDL) source code to insert fault injection logic, which tangles verification and functional code and risks corrupting the design's integrity. Second, RTL simulation is significantly slow, making exhaustive, statistically significant fault campaigns, which are time-consuming and often force a compromise between verification rigor and project deadlines. To overcome these limitations, this study proposes a paradigm shift in the ESL, introducing a dedicated verification environment by employing AOP within a high-level modeling framework.

B. Proposed LED SystemC-AOP Environment

The proposed method employs AspectC++ as the AOP language and SystemC as the modeling language at the ESL level for hardware design. AspectC++ and SystemC are used at the system level to provide an LED AOP environment for evaluating resilience and detection efficacy against fault attacks. The introduced LED SystemC-AOP environment is meant to achieve the following objectives:

- Simulate the LED security fault attack at ESL.
- Evaluate the robustness of fault detection mechanisms in the LED algorithm at the system level using AspectC++ and SystemC integration.
- Regulate the fault injection and detection process: timing and location of faults.
- Inject and identify faults in real-time processes.

Figure 2 provides an overview of the proposed method for assessing the resilience of the LED model against fault attacks, illustrating the four-stage workflow of the LED SystemC-AOP environment. The initial Modeling Phase integrates both the core LED SystemC implementation and specialized aspect modules. These modules consist of three main components: LED Fault Injector (LFI) for precise spatiotemporal error introduction, LED Fault Controller (LFC) managing module synchronization, and LED Fault Analysis (LFA) generating comprehensive security reports. The LFI component targets specific execution points and memory locations for controlled fault insertion, while the LFC orchestrates timing synchronization across all SystemC/AOP components. The LFA module produces detailed reports, including detected faults, error classification, injection parameters, and their functional consequences on the cryptographic implementation.

The second stage involves the Aspect LED Weaving Phase, which executes two significant operations: aspect code integration and model compilation. Using AspectC++ specifications from the initial phase, the three aspect modules (LFI, LFC, LFA) are automatically incorporated into the LED SystemC architecture through non-invasive weaving techniques that preserve the original SystemC implementation. This integration is followed by compilation of the augmented source code, producing a functional executable that combines cryptographic operations with fault analysis capabilities while maintaining architectural integrity.

The Aspect LED Simulation Phase is the third critical stage in the LED SystemC-AOP environment, comprising two parallel verification components: the woven executable system and a pristine reference implementation of LED. As depicted in Figure 2, this phase performs differential analysis by systematically comparing outputs from both modules, the executable aspect (containing potential fault injections) against the certified fault-free reference model. Any discrepancies detected during this comparative execution directly indicate errors introduced either through fault injection or aspect integration processes, enabling a precise fault behavior description.

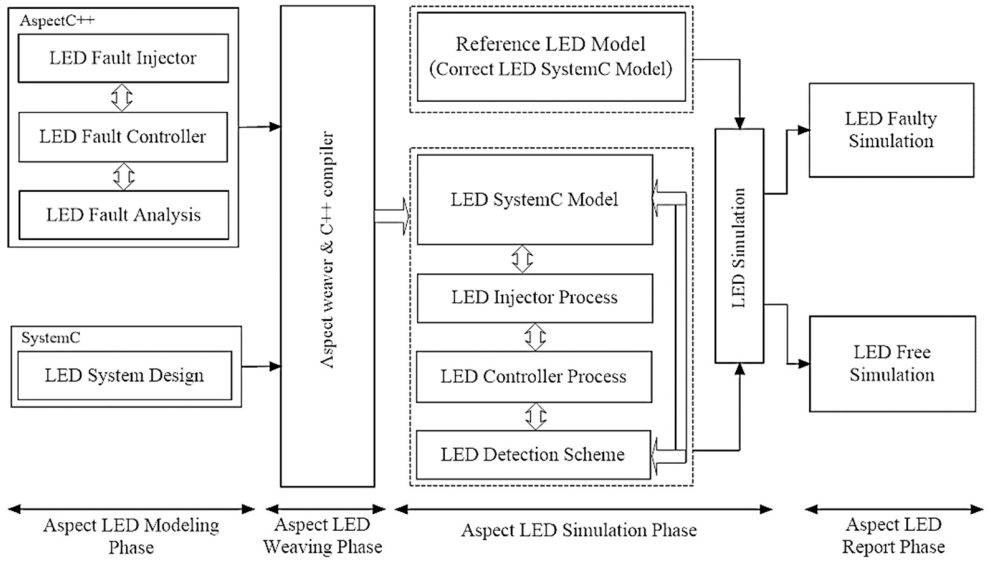

Fig. 2. General view of the LED security analysis environment.

The final phase of the LED SystemC-AOP environment involves the Aspect LED Report Phase, which systematically documents the operational impact of injected faults on cryptographic functionality and the precise classification of all faults into four distinct categories: Detected errors, Undetected errors, Silent faults, and False positives. This analytical phase generates significant security metrics that evaluate the effectiveness of the protection mechanisms implemented. These metrics provide insights into the overall security posture of the cryptographic system and help in identifying areas for improvement. The Aspect LED Report Phase is crucial in ensuring that the system meets its security requirements and can withstand potential attacks.

Figure 3 presents the flow diagram of the fault injection and attack process within the LED SystemC-AOP environment. The process begins by defining the AOP aspect and its pointcuts to target specific locations in the LED model. Based on the selected fault type (permanent or transient), corresponding single-bit or multi-bit fault attacks are conducted against the model.

The spatio-temporal weaving mechanism enables non-intrusive fault injection by mapping AspectC++ join points to structural SystemC elements (module entry points, round executions, register updates) and synchronizing with the simulation kernel's event scheduler. Fault injection is triggered when the simulation time reaches thresholds or specific events occur, targeting the exact moment the fourth round is completed. The weaver instruments the executable at these precise points with conditional execution based on runtime parameters, enabling fine-grained fault control without modifying the source code.

In summary, this methodology establishes a modular, non-intrusive workflow for security verification. By separating fault injection, control, and analysis into distinct AOP modules woven into a SystemC model, precise, reusable, and efficient evaluation of cryptographic resilience can be achieved in the ESL, overcoming the intrusiveness and speed limitations of traditional RTL approaches.

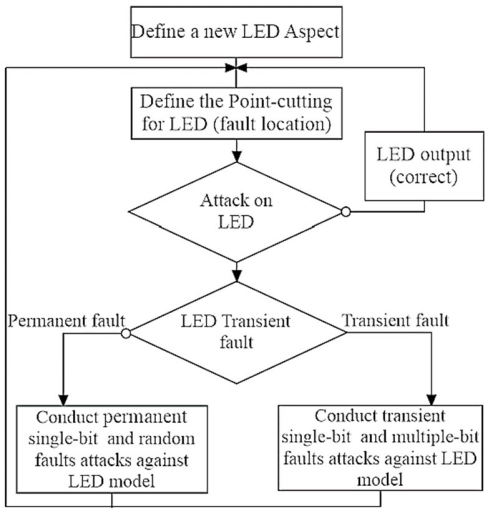

Fig. 3. Flowchart of aspect LFI.

V. SYSTEMC-AOP LED ENVIRONMENT VALIDATION

A. Impact of AOP on Fault Detection Capabilities

The fault detection efficacy of the LED SystemC-AOP environment was validated through controlled simulations. A dual-track evaluation compared a standard SystemC implementation with the proposed SystemC-AOP approach, with additional reference to established detection schemes [6]. The detection scheme from [6] was applied to the LED algorithm. Using integrated SystemC and AspectC++ kernels, detection accuracy, latency, and coverage were assessed across all configurations. The conventional method requires direct code modification for fault injection and detection, whereas the SystemC-AOP approach enables non-intrusive analysis through aspect weaving while preserving the original codebase. Both support transient and persistent fault analysis, enabling extensive resilience evaluation [6].

1) Impact of AOP on Single-Fault Detection Capabilities

Experimental validation assessed single-bit fault detection through 1,000,000 controlled transient injections targeting each byte position, arithmetic operation, and round transformation in the LED architecture. As displayed in Table II, both standard SystemC and SystemC-AOP environments demonstrated identical detection rates. This equivalence empirically validates that the proposed AOP approach maintains full fault detection capabilities of traditional implementations while offering superior modularity and non-invasive verification advantages. The identical detection rates are fundamentally expected, as the core detection logic (the algorithm in [6]) is preserved unchanged between the two implementations. The role of the AOP weaver is to integrate this logic non-intrusively at the precise join points, not to alter its functional behavior. Therefore, this equivalence serves as a proof of the correctness of the weaving process, confirming that the aspect has been accurately and completely integrated into the SystemC simulation without functional distortion or omission.

TABLE II. LED DETECTION CAPABILITY: TRANSIENT AND PERMANENT FAULTS

Type of faults	**Fault multiplicity**		**Percentage of the undetected faults (%)**	
			SystemC	**SystemC+ AOP**
Transient faults	Single-bit		2.569	2.569
	Multiple-bit	2-bit	2.159	2.159
		3-bit	1.564	1.564
		4-bit	1.023	1.023
		5-bit	0.912	0.912
		6-bit	0.874	0.874
		7-bit	0.798	0.798
		8-bit	0.714	0.714
		9-bit	0.681	0.681
		10-bit	0.592	0.592
Permanent faults	Single-bit		0.891	0.891
	Random-bit		0.604	0.604

The identical detection rates validate weaving correctness and establish that non-intrusive security analysis preserves original design behavior, a significant property for industrial adoption where design integrity must remain intact. The improvement lies not in enhanced detection (requiring

algorithmic advances beyond this scope), but in achieving equivalent security assurance through modularity, reusability, and non-invasive integration, capabilities unavailable in traditional approaches. This shows that security verification does not need to trade design integrity for analytical capability.

2) Impact of AOP on Multiple-Fault Detection Capabilities

The second experimental phase evaluated multiple-bit fault detection through nine test cases with fault multiplicities from 2 to 10 corrupted bits per block. Each configuration executed 1,000,000 fault injections targeting all LED rounds, arithmetic operations, and byte boundaries, incorporating both transient and permanent errors randomly distributed across state variables. Table II presents comparative undetected fault rates. Experimental data reveal identical detection performance between standard SystemC and SystemC-AOP for all multi-bit fault cases, confirming that the proposed method maintains equivalent detection capabilities to conventional approaches while providing architectural benefits, and validating its effectiveness as a non-intrusive verification environment. The undetected fault rate decreases as fault multiplicity increases because multi-bit faults are more likely to exceed the Hamming distance threshold of parity-based detection and trigger multiple redundant checkers simultaneously. This reveals that the detection scheme is the most vulnerable to precise single-bit faults that can evade detection.

3) Fault Behavior Patterns and Security Implications

Undetected fault patterns exhibit important security insights. Single-bit undetected faults occurred predominantly in rounds 4-5, where state manipulation temporarily masked parity discrepancies, identifying a vulnerability window for targeted attacks. For multi-bit faults, undetected cases correlated with symmetric bit-flips across complementary positions (e.g., bits 0 and 7), suggesting that certain fault combinations cancel detection signatures. These patterns indicate that, while statistically robust, attackers could exploit timing windows or symmetric injection to bypass protection, motivating diversified detection mechanisms rather than relying on a single parity-based scheme. The observed fault patterns provide analytical insights beyond detection rates. First, undetected single-bit faults concentrated in rounds 4-5 reveal a vulnerability window, where LED's state manipulation temporarily masks parity discrepancies, and attackers exploiting this timing window achieve injection with only 2.57% detection probability. Second, symmetric bit-flip patterns in undetected multi-bit faults (bits 0 and 7 simultaneously) suggest that certain fault combinations cancel detection signatures across redundant checkers, indicating deterministic vulnerabilities that sophisticated attackers could exploit. Third, decreasing undetected rates with increasing fault multiplicity (2.57% for single-bit to 0.59% for 10-bit) confirms that multi-bit faults exceed Hamming distance thresholds, but reveals diminishing marginal gains beyond 4-bit faults, suggesting optimal cost-security trade-off points. These insights, enabled by the systematic fault campaigns the proposed AOP framework supports, provide guidance for hardening cryptographic implementations through diversified, multi-layered detection mechanisms.

4) Architectural Advantages for Security Verification

The AOP framework introduces three architectural advances beyond detection equivalence. First, modular separation enables independent evolution of security concerns, and countermeasures upgrade by swapping aspects without modifying the core cryptographic code. Second, systematic design-space exploration allows rapid evaluation of multiple countermeasure strategies (temporal redundancy, error correcting codes, parity detection) through aspect reconfiguration, reducing evaluation cycles from weeks to hours. Third, the non-intrusive approach scales efficiently to complex SoC designs where RTL modification is impractical. These capabilities transform security verification from one-off efforts toward reusable, maintainable assessment infrastructures, which is the primary methodological contribution of this work.

B. The Impact of ESL on the Simulation Time and Executable File Size

A performance evaluation of the proposed AOP-enhanced environment was conducted by injecting 2,500 randomized faults into known vulnerable locations within the cryptographic model and measuring average simulation durations. It is observed that the temporal overhead introduced by AOP scales linearly with the quantity of join points. This proportional relationship between aspect complexity and runtime performance establishes quantifiable tradeoffs between verification thoroughness and simulation efficiency. Table III presents the LED simulations for user time (uTime) and kernel time (kTime) in both scenarios. The observed variation falls within the margin of error linked to the proposed measurement technique, which relied on the Linux time command. This minimal variance is expected in system-level timing measurements, where factors such as background process scheduling, memory state, and filesystem cache can introduce minor fluctuations between consecutive runs. Based on these results, it is concluded that AOP has a minimal impact on simulation runtime and should not be considered a limiting factor in cryptographic security verification. Additionally, the executable file sizes of SystemC and SystemC-AOP, generated by AspectC++ and the SystemC kernel, respectively, were analyzed (see Table III), confirming that AOP does not significantly influence the size of the executable files.

TABLE III. PROTECTED LED ALGORITHM: SIMULATION OF TIME ANALYSIS AND FILE EXECUTABLE SIZE

Simulation	Metric	SystemC	SystemC + AOP	Variance
Simulation time	kTime (s)	0.035	0.034	±0.002
	uTime (s)	1.945	1.942	±0.015
Executable file	size	1.23 MB	1.26 MB	±0.02 MB

The measured variance (±0.002s for kernel time, ±0.015s for user time across 50 simulation runs) confirms that the observed timing differences fall within the normal system noise from process scheduling and cache effects. This statistical validation shows that AOP introduces negligible runtime overhead, making it suitable for large-scale fault injection campaigns where simulation efficiency is crucial.

VI. CONCLUSION

This paper presented a method for fault injection and detection at the Electronic System Level (ESL), aimed at simulating security fault attacks in cryptographic hardware designs. The proposed SystemC-Aspect-Oriented Programming (AOP) approach evaluates cryptographic robustness through non-intrusive, modular integration of fault analysis into SystemC models using AOP, enabling precise, reusable, and maintainable verification without corrupting original design code. By improving the reliability and security of critical digital infrastructure, this work contributes to Sustainable Development Goal (SDG) 9, promoting resilient infrastructure and innovation. Simulation results demonstrate the method's feasibility and effectiveness in assessing fault injection resistance, while introducing negligible simulation time overhead. The AOP-based approach proves valuable for security testing, streamlining simulation while reducing efforts and potential errors in vulnerability evaluation.

ACKNOWLEDGMENT

The authors extend their appreciation to Prince Sattam bin Abdulaziz University for funding this research work through the project number (PSAU/2025/01/38788).

REFERENCES

[1] V. Muttillo, L. Pomante, M. Santic, and G. Valente, "SystemC-based Co-Simulation/Analysis for System-Level Hardware/Software Co-Design," *Computers and Electrical Engineering*, vol. 110, Sept. 2023, Art. no. 108803, https://doi.org/10.1016/j.compeleceng.2023.108803.

[2] S. H. Lim, W. W. Suh, J. Y. Kim, and S. Y. Cho, "RISC-V Virtual Platform-Based Convolutional Neural Network Accelerator Implemented in SystemC," *Electronics*, vol. 10, no. 13, June 2021, https://doi.org/10.3390/electronics10131514.

[3] B. Rashidi, "Fault-tolerant and error-correcting 4-bit S-boxes for cryptography applications with multiple errors detection," *The Journal of Supercomputing*, vol. 80, no. 2, pp. 1464–1490, Jan. 2024, https://doi.org/10.1007/s11227-023-05530-7.

[4] P. Bhoyar, P. Sahare, M. F. Hashmi, S. B. Dhok, and R. Deshmukh, "Lightweight architecture for fault detection in Simeck cryptographic algorithms on FPGA," *International Journal of Information Technology*, vol. 16, no. 1, pp. 337–343, Jan. 2024, https://doi.org/10.1007/s41870-023-01593-0.

[5] P. Ravi, A. Chattopadhyay, J. P. D'Anvers, and A. Baksi, "Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 2, Nov. 2024, Art. no. 35, https://doi.org/10.1145/3603170.

[6] H. Mestiri and I. Barraj, "High-Speed Hardware Architecture Based on Error Detection for KECCAK," *Micromachines*, vol. 14, no. 6, May 2023, https://doi.org/10.3390/mi14061129.

[7] A. C. Canto, A. Sarker, J. Kaur, M. M. Kermani, and R. Azarderakhsh, "Error Detection Schemes Assessed on FPGA for Multipliers in Lattice-Based Key Encapsulation Mechanisms in Post-Quantum Cryptography," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 3, pp. 791–797, July 2023, https://doi.org/10.1109/TETC.2022.3217006.

[8] S. Azzopardi, J. Ellul, R. Falzon, and G. J. Pace, "AspectSol: A Solidity Aspect-Oriented Programming Tool with Applications in Runtime Verification," in *Runtime Verification*, 2022, pp. 243–252, https://doi.org/10.1007/978-3-031-17196-3_13.

[9] A. A. Magableh and A. M. R. Al Sobeh, "Securing Software Development Stages Using Aspect-Orientation Concepts." *International Journal of Software Engineering & Applications*, vol. 9, no. 6, Nov. 2018.

[10] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The LED Block Cipher," in *Cryptographic Hardware and Embedded Systems – CHES 2011*, 2011, pp. 326–341, https://doi.org/10.1007/978-3-642-23951-9_22.

[11] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ," in *ECOOP 2001 — Object-Oriented Programming*, 2001, pp. 327–354, https://doi.org/10.1007/3-540-45337-7_18.

[12] C. L. Vidal-Silva, E. Madariaga, T. Pham, F. Johnson, L. A. Urzua, and L. Carter, "JPIAspectZ: A Formal Requirement Specification Language for Joint Point Interface AOP Applications," *Engineering, Technology & Applied Science Research*, vol. 9, no. 4, pp. 4338–4341, Aug. 2019, https://doi.org/10.48084/etasr.2774.

[13] Z. Chen, Y. Zhu, and Z. Wang, "Design and Implementation of an Aspect-Oriented C Programming Language," *Reproduction Package for Article `Design and Implementation of an Aspect-Oriented C Programming Language'*, vol. 8, no. OOPSLA1, Art. no. 117, Dec. 2024, https://doi.org/10.1145/3649834.

[14] A. Bhar, S. Sengupta, and S. Choudhury, "Aspect-Oriented Approach of Non-Functional Requirements for Embedded System-Based IoT Applications," *IEEE Access*, vol. 13, pp. 129451–129460, 2025, https://doi.org/10.1109/ACCESS.2025.3585417.

[15] T. R. Muck, A. A. Frohlich, M. Gernoth, and W. Schroder-Preikschat, "Implementing OS components in hardware using AOP," *SIGOPS Oper. Syst. Rev.*, vol. 46, no. 1, pp. 64–72, Oct. 2012, https://doi.org/10.1145/2146382.2146395.

[16] J. M. P. Cardoso *et al.*, "LARA: an aspect-oriented programming language for embedded systems," in *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*, Nov. 2012, pp. 179–190, https://doi.org/10.1145/2162049.2162071.

[17] A. M. R. AlSobeh and A. A. Magableh, "BlockASP: A Framework for AOP-Based Model Checking Blockchain System," *IEEE Access*, vol. 11, pp. 115062–115075, 2023, https://doi.org/10.1109/ACCESS.2023.3325060.

[18] A. Nusayr and J. Cook, "Using AOP for detailed runtime monitoring instrumentation," in *Proceedings of the Seventh International Workshop on Dynamic Analysis*, Apr. 2009, pp. 8–14, https://doi.org/10.1145/2134243.2134246.