

A Hardware Trojan-Resilient FPGA Framework with Adaptive Runtime Reconfiguration

maryam amini

University of Isfahan

shahram Etemadi Borujeni

etemadi@eng.ui.ac.ir

University of Isfahan

Siamak Mohammadi

University of Tehran

Mansoureh Labafniya

University of Isfahan

Research Article

Keywords: Hardware Trojan, Configurable Look-Up Tables, Evolvable Hardware, Virtual Reconfigurable Circuit, FPGA

Posted Date: October 26th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11063481/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

A Hardware Trojan-Resilient FPGA Framework with Adaptive Runtime Reconfiguration

Maryam Amini¹, Shahram Etemadi Borujeni^{a,1}, Siamak Mohammadi², Mansoureh Labafniya³

¹ Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran

² Faculty of Electrical and Computer Engineering, University of Tehran, Tehran, Iran

³ MSI Research Center, Tehran, Iran

Received: date / Accepted: date

Abstract The increasing deployment of Field-Programmable Gate Arrays (FPGAs) in safety- and security-critical applications has heightened concerns over hardware Trojans (HTs), which can compromise circuit functionality during runtime. Although existing runtime mitigation techniques provide partial protection, they often suffer from hardware overhead, limited adaptability, or prolonged recovery after Trojan detection. To address these limitations, this paper presents a dynamic hardware security framework that restores correct circuit functionality through adaptive runtime reconfiguration following Trojan detection. The proposed framework employs two complementary reconfigurable architectures, namely Evolvable Hardware (EH) based on Virtual Reconfigurable Circuits (VRCs) and an enhanced Configurable Look-Up Table (CFG-LUT) architecture. Critical circuit modules are implemented using reconfigurable structures, enabling compromised configurations to be replaced with functionally valid alternatives during runtime. To maintain functional correctness during the reconfiguration process, a compensation mechanism temporarily provides the expected output of the affected module. Furthermore, the enhanced CFG-LUT architecture incorporates input-selection multiplexers, enabling simultaneous logic reconfiguration and input-path adaptation. An enhanced Genetic Algorithm (GA) is employed to accelerate the configuration search and reduce recovery latency. Experimental results demonstrate up to a 53.12% reduction in convergence time, while introducing only 2.46% and 1.8% additional LUT utilization for the VRC and CFG-LUT-with-MUX architectures, respectively, compared with their corresponding baseline implementations. Furthermore, compared with the reference approach, the proposed framework achieves up to a 9.22% reduction in LUT

utilization, demonstrating an effective balance between recovery performance and hardware cost.

Keywords Hardware Trojan · Configurable Look-Up Tables · Evolvable Hardware · Virtual Reconfigurable Circuit · FPGA

1 Introduction

FPGAs have become one of the dominant implementation platforms for modern embedded and high-performance systems due to their flexibility, parallel processing capability, and support for runtime reconfiguration. These features have enabled the widespread adoption of FPGAs in industrial automation, aerospace systems, military infrastructures, communication platforms, and other safety- and security-critical applications requiring high adaptability and reliable operation. Nevertheless, the increasing complexity of FPGA-based systems, together with the extensive use of third-party Intellectual Property (IP) cores and dynamic reconfiguration mechanisms, has significantly increased hardware security concerns and expanded the attack surface of such systems.

Among existing hardware threats, HTs represent one of the most critical and destructive security challenges. HTs are malicious modifications intentionally inserted into a circuit during different stages of the design or supply chain process. Typically, an HT consists of a trigger condition and a hidden payload designed to remain inactive during conventional testing procedures, thereby avoiding detection while enabling malicious behavior during runtime operation [1]. The detection of HTs in FPGA-based systems is particularly difficult because runtime reconfiguration can dynamically alter circuit behavior and obscure Trojan activity [2]. Since the security risks associated with HTs were formally highlighted by the U.S. Department of Defense in 2005, HT mit-

^aCorresponding author: Shahram Etemadi Borujeni, e-mail: etemadi@eng.ui.ac.ir

igation and trustworthy hardware design have attracted considerable attention in the hardware security community [3].

Existing HT detection techniques, including side-channel analysis, logic testing, and formal verification approaches, often suffer from limited effectiveness against stealthy Trojans with small hardware footprints, rare activation conditions, and behavior highly correlated with normal circuit activity. Moreover, many conventional methods mainly focus on offline detection and provide limited support for runtime recovery after Trojan activation. Consequently, adaptive runtime protection mechanisms capable of maintaining system functionality under Trojan attacks have become increasingly important for modern reconfigurable systems. To address these challenges, this paper presents a reconfigurable security framework for the detection, mitigation, and prevention of HTs in FPGA-based systems.

The proposed framework maintains parallel implementations of critical circuit modules on two independent reconfigurable architectures—VRCs and enhanced Configurable Look-Up Tables with input-selection multiplexers (CFG-LUT with MUX). In addition to their implementation, the behavioral characteristics and structural evolution of these architectures under runtime reconfiguration are systematically analyzed. These architectures operate as runtime protection layers, introducing structural diversity alongside dynamic reconfiguration capability. By leveraging dynamic structural adaptability and evolutionary reconfiguration, the system supports the online replacement of compromised circuit components with validated healthy configurations, thereby maintaining operational continuity without service interruption.

For Trojan detection, the Golden Reference functionality is stored in an on-chip embedded memory block. The fundamental assumption is that implanting HTs into internal FPGA memory resources is considerably more challenging than targeting configurable logic fabric. This assumption is supported by the intrinsic structural protection and built-in reliability mechanisms of embedded memories. In particular, the Block RAM (BRAM) modules utilized in this implementation provide error detection and correction capabilities (e.g., ECC), enhancing their trustworthiness as secure storage elements. Detection is performed through periodic comparison and systematic monitoring of selected circuit modules against the stored reference model, enabling timely identification of functional deviations.

Upon detection of a HT, a compensation mechanism is immediately activated and deployed as a temporary functional substitute to maintain uninterrupted system operation. The compromised module is logically isolated from the system to prevent fault propagation and preserve operational integrity.

Accordingly, the proposed framework integrates real-time detection, module isolation, attack tolerance, and secure recovery within a unified architectural defense strategy. Build-

ing upon and extending these concepts, this paper makes the following contributions:

- To the best of our knowledge, this paper is the first to investigate the independent use of two reconfigurable architectures, namely VRC and an enhanced CFG-LUT equipped with multiplexers, for implementing security-critical modules to mitigate hardware Trojan effects and prevent their propagation throughout circuit operation.
- Hardware Trojan detection is achieved through a detection module that continuously monitors the outputs of security-critical modules during runtime and periodically compares them with the corresponding reference outputs to identify Trojan-induced anomalies.
- A concurrent compensation mechanism is introduced to maintain uninterrupted system operation after attack detection. Once a malicious modification is identified, the controller immediately redirects the output path from the compromised module to a trusted compensation unit, thereby preventing the propagation of Trojan-induced effects and ensuring continuous generation of valid outputs during the recovery and reconfiguration interval.
- An enhanced GA-based recovery framework is developed to accelerate runtime reconfiguration and improve convergence efficiency. The proposed optimization strategy incorporates inheritance-aware population initialization derived from circuit truth tables, hybrid parent-selection policies, fitness-guided crossover operations, and controlled mutation mechanisms to reduce recovery latency and improve reconfiguration quality.
- Finally, the FPGA resource overhead, including logic utilization, power consumption, reconfiguration latency, as well as the advantages and limitations of each proposed architecture, is comprehensively evaluated.

The remainder of this paper is organized as follows. Section 2 reviews the fundamental concepts related to HTs in digital circuits. Section 3 presents a comprehensive survey of related work, with a focus on advanced detection, prevention, and dynamic reconfiguration techniques, particularly in the context of *EH* and *CFG-LUT* architectures. Section 4 details the proposed architecture and its implementation on both VRC and multiplexer-enhanced *CFG-LUT* platforms, along with the runtime reconfiguration mechanism facilitated by the GA. Section 5 reports the implementation results and provides a comprehensive comparison of the two architectures in terms of resource utilization, reconfiguration speed, and algorithm convergence using standard benchmark circuits. Finally, Section 6 concludes the paper and outlines directions for future work.

2 Preliminaries

2.1 Hardware Trojans

HTs are stealthy malicious modifications intentionally inserted into integrated circuits ICs during different stages of the design, development, fabrication, or supply-chain process. These modifications may compromise system functionality, degrade reliability, leak confidential information, or create hidden backdoors for unauthorized access. Due to their covert nature and complex activation mechanisms, HTs are extremely difficult to detect or eliminate after fabrication [4]. Unlike software Trojans, which can often be detected using conventional cybersecurity techniques, HTs operate at the hardware level and are typically embedded in rarely activated circuit paths or low-switching-activity regions. They are triggered only under specific operational conditions or rare input patterns, significantly reducing their observability during conventional testing and making post-deployment detection particularly challenging [5, 6].

FPGA-based systems are especially vulnerable to HT insertion because of their reconfigurable architecture and complex design flow, which includes hardware description, synthesis, implementation, bitstream generation, and deployment [7]. Potential attack vectors include malicious design houses or third-party IP vendors, untrusted fabrication facilities, compromised Electronic Design Automation (EDA) toolchains [8], and bitstream-level attacks during bitstream generation, transmission, storage, or deployment [9].

2.2 Evolvable Hardware

EH combines reconfigurable hardware with Evolutionary Algorithms (EAs) to enable adaptive, fault-tolerant, and self-repairing digital systems. As the demand for resilient and autonomous computing platforms continues to grow, EH has become an attractive approach for developing hardware capable of dynamically adapting to environmental changes and operational faults. In contrast to conventional hardware design, where circuit structures are predetermined during the design phase, EH defines only the required input–output functionality, while the internal circuit configuration is automatically evolved through heuristic optimization using EAs. By exploring a large configuration space, this evolutionary design process can discover efficient circuit implementations that are difficult to obtain using traditional design methodologies [10, 11].

EH is commonly implemented on reconfigurable hardware platforms, particularly FPGAs, where circuit configurations generated by EAs are mapped onto programmable logic resources. Depending on the evaluation strategy, EH implementations are generally classified into three categories: extrinsic evolution, intrinsic evolution, and Virtual VRC based

evolution. In extrinsic evolution, candidate configurations are assessed using software based simulation, providing high design flexibility but limited runtime accuracy. By contrast, intrinsic evolution evaluates each candidate directly on the target FPGA through Dynamic Partial Reconfiguration (DPR), resulting in more accurate fitness evaluation at the expense of higher reconfiguration latency and implementation complexity [12].

Among the existing EH implementations, the VRC-based approach provides a more efficient solution by executing the entire evolutionary process directly on the FPGA. A VRC introduces an overlay reconfigurable architecture consisting of programmable Processing Elements (PEs) connected through a configurable interconnection network. EAs manage a secondary reconfiguration layer that updates both the functionality of the PEs and their routing connections without requiring complete FPGA reconfiguration, thereby significantly reducing reconfiguration overhead [13]. Furthermore, the PE architecture, routing organization, and overall VRC structure can be tailored to the requirements of a specific application. Each PE acts as a compact programmable logic element, commonly realized using multiplexer-based combinational logic, where the implemented logic function and input selection are controlled by multiplexer configuration bits. By arranging programmable PEs into a regular array, the VRC architecture can implement a wide variety of circuit structures through dynamic reconfiguration controlled by an embedded controller or an EA. This capability enables the rapid evaluation of numerous candidate configurations during runtime, considerably accelerating the evolutionary optimization process compared with conventional extrinsic and intrinsic EH approaches. Because the VRC architecture is defined during the design stage using a Hardware Description Language (HDL), it is independent of the target FPGA architecture and can be readily integrated with a hardware-implemented EA on the same FPGA. Furthermore, when the chromosome encoding is directly aligned with the VRC configuration interface, new configurations can be applied within only a few clock cycles, providing reconfiguration speeds that are significantly higher than those achievable using conventional FPGA reconfiguration techniques [14].

Genetic Algorithms (GAs) are among the most widely used EAs in EH due to their effectiveness in exploring large hardware design spaces. Starting from an initial population of candidate configurations, GAs iteratively improve circuit implementations by applying genetic operators such as selection, crossover, and mutation under the guidance of a predefined fitness function. Their effectiveness has been demonstrated in numerous FPGA-based digital circuit optimization and design applications [15]. Implementing GAs directly on FPGA platforms further exploits hardware parallelism and reconfigurability, enabling high-speed online evolution that is difficult to achieve using software-based

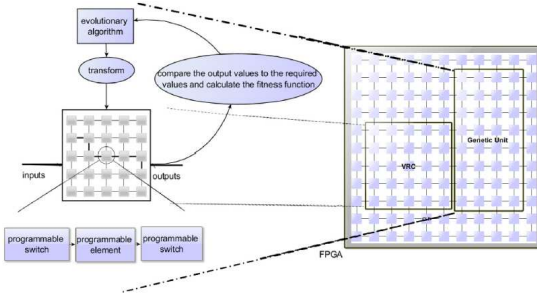


Fig. 1 The structure of VRCs and GA on FPGA [13].

approaches. Within such EH systems, the EA module performs chromosome generation, fitness evaluation, and evolutionary optimization, whereas the VRC provides the reconfigurable hardware platform for realizing and validating candidate configurations. As shown in Figure 1, the EA and VRC modules are integrated on the same FPGA, where the VRC consists of an array of programmable PEs interconnected through configurable routing resources.

2.3 Configurable Look-Up Tables (CFG-LUTs)

The fundamental computational unit in FPGA architectures is the logic slice, which is primarily composed of LUTs and flip-flops. Each LUT contains an initialization parameter, commonly referred to as the *INIT* value, which defines the truth table and corresponding logical functionality of the implemented combinational circuit. In conventional FPGA architectures, the *INIT* contents are programmed through the configuration bitstream during device initialization and remain fixed throughout system operation unless a complete device reconfiguration is performed.

Recent FPGA technologies have introduced runtime reconfigurable LUT structures that enable selective modification of LUT functionality during normal system operation without requiring full FPGA reprogramming. These dynamically reconfigurable LUTs, commonly known as CFG-LUTs, provide the capability to adaptively modify internal logic behavior while maintaining uninterrupted FPGA functionality [16]. In Xilinx FPGA devices, this feature is supported through the *CFGLUT5* primitive, which was originally introduced in the Virtex-5 architecture. Each *CFG-LUT* can be dynamically updated within approximately 32 clock cycles, enabling extremely low-latency runtime logic reconfiguration. Unlike conventional FPGA reconfiguration approaches, this process modifies only the internal logical contents of the LUT without affecting the global routing resources of the FPGA fabric. Functionally, CFG-LUTs exhibit behavior similar to Shift Register LUTs (*SRLs*), which also support runtime modification of internal logical contents [17].

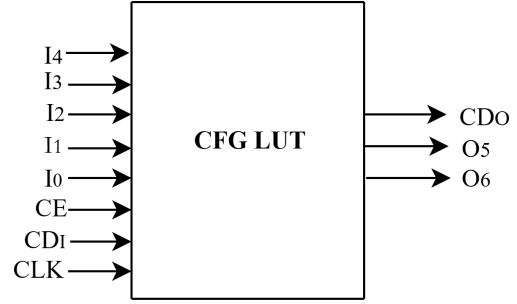


Fig. 2 Block diagram of CFG-LUT.

As illustrated in Fig. 2, a CFG-LUT includes logical inputs (I_0 – I_4) and outputs (O_5 and O_6). Reconfiguration is performed through a serial configuration interface, where new configuration data are supplied via the CDI (Configuration Data Input) port, and previously stored configuration data are shifted out through the CDO (Configuration Data Output) port. The reconfiguration process is enabled by the CE (Configuration Enable) signal and synchronized by the configuration clock (CLK). A distinguishing feature of the CFGLUT primitive is its chaining capability, which allows multiple CFG-LUT instances to be serially connected through their CDI and CDO ports. This feature facilitates the design of scalable and high-performance reconfigurable architectures, making CFG-LUTs particularly well suited for adaptive, secure, and fault-tolerant FPGA-based systems.

Both **CFG-LUT** and **VRC** architectures are designed to provide reconfigurability in FPGA systems; however, the level, nature, and scope of this reconfiguration fundamentally differ. In CFG-LUT architectures, the basic processing unit is a hardware LUT, typically implementing a five-input, one-output logic function. The LUT input structure is fixed and cannot be altered, while only its truth table can be rewritten at runtime. Consequently, although the logical functionality can be modified, the input switching network remains unchanged, and reconfiguration is limited to the functional level. In contrast, **VRC**-based architectures consist of an array of PEs. Each PE is typically implemented using multiplexer-based combinational logic and can dynamically adjust its behavior through the configuration bits of the multiplexer select lines. Unlike CFG-LUTs, which are embedded as hard cores within the FPGA, VRCs are fully designer-defined soft cores that do not inherently exist in the FPGA. In VRC-based designs, the designer can freely configure the number of inputs and outputs, the number of PEs, their interconnections, switching mechanisms, data paths, and even the overall block topology. This flexibility allows VRCs to modify both logical functionality and underlying structural architecture. As a result, VRCs offer a higher level of reconfigurability, enabling modifications to topology, data paths, and architectural structure at runtime. In summary, the key distinction between these two architectures reflects a

trade-off between architectural flexibility and hardware simplicity: VRCs support architectural reconfiguration, runtime evolution, and structural adaptability, whereas CFG-LUTs provide efficient and low-overhead functional reconfiguration within a fixed physical structure.

3 Related Work

Ensuring the resilience of digital systems against security threats requires methodologies grounded in the principles of Design for Security (DfS). In the context of hardware-level attacks, particularly HTs, robust protection depends on both preventive strategies that reduce the likelihood of insertion and detection mechanisms capable of accurately identifying and localizing malicious modifications.

Recent studies have investigated automated detection frameworks based on large language models (LLMs). For example, the seCRT-LLM framework exploits GPT-3.5-turbo to improve Trojan detection accuracy in code-level designs. By generating a synthetic dataset of 10,000 samples and applying In-Context Learning (ICL) with multiple prompt templates, the framework enables scalable detection without requiring additional fine-tuning [18]. Extending this idea, prompt-based analysis techniques such as ARC-FSM have been applied to finite state machine (FSM) verification, where detected anomalies are automatically translated into assertion-based design modifications [19]. Graph-based vulnerability analysis has also been explored, where circuit representations are constructed using tools such as VeriLAVD. In this approach, Depth-First Search (DFS) and Breadth-First Search (BFS) traversals are employed to extract vulnerability patterns, while LLMs generate corresponding detection rules that are validated through VeriPG-based traversal for automated vulnerability identification [20]. In a related direction, it has been shown that general-purpose LLMs can also be misused to assist in hardware Trojan insertion; for instance, a Trojan targeting a modern RISC-V microarchitecture was successfully implemented on an FPGA, demonstrating potential risks to system integrity and availability [21]. Beyond LLM-based approaches, alternative detection techniques rely on side-channel information such as power consumption, propagation delay, and electromagnetic emissions to identify abnormal circuit behavior [22].

In evolutionary hardware approaches, software-based GAs have been used to generate initial VRC configurations with improved convergence behavior through adaptive mutation control [9]. Building on this concept, GA-driven VRC frameworks have been proposed to prevent Trojan propagation by dynamically reconfiguring affected regions, replacing compromised PEs with spare units to maintain system integrity [10]. Similar evolutionary strategies have been extended to transistor-level reconfigurable architectures such as Field-Programmable Transistor Arrays (FPTAs), where software-implemented GAs

enable adaptive fault-tolerant circuit evolution [11]. A VRC-based circuit supporting runtime reconfiguration was periodically updated using a GA, enabling automatic convergence toward a valid configuration even under hardware Trojan conditions [13]. A hardware-implemented hybrid evolvable unit combining VRC and EA has been introduced, enabling autonomous synthesis and optimization of digital circuits such as adders and multipliers within seconds through direct environmental interaction [14]. Similarly, an EH platform incorporating BIST has been proposed, where test-time error detection triggers a hardware-implemented GA for dynamic reconfiguration, while permanent faults are addressed through replacement of faulty PEs with spare units [23]. Runtime error mitigation techniques for VRC structures have also been explored, where outputs are continuously compared against a golden reference stored in BRAM, and detected discrepancies are corrected through GA-based reconfiguration to restore correct functionality [24, 25]. Beyond classical fault recovery, software-based evolutionary approaches have been applied to adapt neuronal structures in VRC-based neural networks, aiming to prevent error propagation during operation [26]. In addition, hybrid optimization strategies combining Genetic Algorithms and Memetic algorithms have been utilized to enhance the fault tolerance of VRC architectures [27]. Self-healing VRC-based designs have further demonstrated autonomous recovery capabilities, where configuration bitstreams are dynamically generated to support adaptive repair and post-fault restoration [28]. EH frameworks have been extended to autonomous circuit synthesis, where evolved configurations are capable of generating digital image processing circuits that outperform conventional filters in tasks such as edge detection and noise suppression, while maintaining strong robustness under diverse noise conditions [29].

The inherent reconfigurability of CFG-LUTs has been widely exploited in recent studies for both security enhancement and malicious design manipulation. One line of research has demonstrated that this flexibility can be used to enable low-overhead hardware Trojan insertion, where a circuit initially operates in a clean state and is later modified at runtime through CFG-LUT reconfiguration [16]. Similarly, CFG-LUT-based implementations have been used to achieve rapid runtime adaptation in digital signal processing applications, such as FIR filters, by enabling efficient reconfiguration of logic functions during operation [17]. In cryptographic applications, CFG-LUTs have been employed to explore multiple runtime reconfiguration strategies. For instance, dynamically loading different LUT structures for each encryption round enables per-operation variability in cryptographic circuits, while the same reconfigurability has also been leveraged to introduce runtime hardware Trojans within encryption modules [30]. Extending this concept, programmable cellular automata architectures based on CFG-

LUTs have been proposed, supporting real-time configurability of neighborhood functions and offering dual operational modes for normal execution and testing scenarios [31].

Further applications in cryptography include CFG-LUT-based implementations of the PRESENT cipher, where substitution and permutation layers are independently reconfigured at each encryption round. This continuous structural variation improves resistance against side-channel analysis by making internal behavior harder to model or predict [32]. In another approach, portions of a circuit are randomly mapped onto CFG-LUTs while configuration data are stored in BRAM and dynamically restored at runtime, introducing a hardware obfuscation mechanism [33].

Evolutionary and learning-based reconfiguration strategies have also been integrated with CFG-LUT architectures. For example, a hardware-implemented GA has been used to dynamically update LUT truth tables during AES encryption, enabling per-cycle adaptation of cryptographic logic [34]. Beyond cryptographic optimization, CFG-LUTs have been combined with Physically Unclonable Functions (PUFs) to strengthen IP protection mechanisms [35]. However, the same reconfigurable properties have also been exploited for stealthy hardware Trojan insertion with minimal overhead [36], while other works have utilized CFG-LUT structures to interfere with AES execution through internal FPGA sensor manipulation [37].

In previous studies, including [18–21], structure detection and correction were performed using LLMs. The main limitation of these approaches is that the models can only detect Trojans inserted at the design level (RTL) and are unable to identify Trojans injected at the bitstream level or EDA. Furthermore, these models rely on ICL, which is constrained by the number of input tokens, limiting the learning capability and reducing the accuracy of detection and correction. In other approaches, such as [11, 13, 14, 23–28], circuits were implemented on a VRC structure to prevent error propagation and enable reconfiguration. However, the reconfiguration algorithms in these methods are executed in software, which, due to the speed mismatch between software and hardware, results in slow reconfiguration. Additionally, as the reconfiguration time is determined based on the fault occurrence, prolonged reconfiguration renders the circuit unavailable and significantly reduces overall system performance. In [31, 32, 34, 37], only the internal CFG-LUT structures were modified, while routing and switching remained fixed. Moreover, the CFG-LUT contents were predetermined, allowing an attacker to potentially manipulate these values and compromise the circuit functionality.

In our proposed approach, the reconfigurability of the CFG-LUT-based architecture is extended beyond conventional LUT content modification to enhance both implementation efficiency and structural security. Unlike traditional schemes that confine reconfiguration to modifying the inter-

nal truth tables of CFG-LUTs, the proposed framework additionally enables dynamic adaptation of routing paths and switching resources during runtime. By tightly integrating CFG-LUT structures with multiplexing elements, the architecture achieves enhanced runtime adaptability. This integration supports not only logic-level redefinition but also controlled restructuring of interconnection networks, thereby allowing structural transformations at multiple abstraction levels. HT detection is performed periodically in both proposed architectures. Upon identification of anomalous behavior, the affected circuit region is immediately reconfigured using an enhanced reconfiguration algorithm GA. The recovery process is executed concurrently with normal system operation, thereby preventing service interruption and maintaining functional continuity. Furthermore, since both candidate circuit structures and the reconfiguration process are dynamically generated at runtime by the proposed algorithm GA, the resulting configurations are neither predefined nor statically stored. Consequently, potential adversaries cannot predict or pre-characterize the structural implementation that will ultimately be deployed, significantly increasing resistance against targeted and structurally aware attacks.

4 Proposed Approach

The proposed approach introduces a comprehensive and coherent architectural framework for the detection, containment, and mitigation of HTs in FPGA-based systems across two reconfigurable architectures: VRC and CFG-LUT-with-MUX. During the design phase, security-critical regions of the original circuit are identified, including components the manipulation of which would directly affect correct system functionality, such as cryptographic cores and memory control units, as well as regions characterized by low testability. Since hardware Trojans are typically inserted in either low-testability areas or structurally influential regions with high impact on system behavior, this analysis enables the identification of design portions that are most susceptible to adversarial manipulation. Once identified as critical blocks, each region is independently mapped onto the VRC and CFG-LUT-with-MUX architectures under two design scenarios, whereas non-critical portions are realized using a static implementation to minimize unnecessary architectural overhead.

This selective deployment strategy enables rapid adaptation of critical regions through runtime architectural reconfiguration in response to malicious perturbations, functional deviations, or hardware Trojan-induced anomalies. However, conventional CFG-LUT architectures suffer from a fundamental limitation in that their reconfiguration capability is restricted to modifications of LUT truth tables, without the ability to alter routing structures. As reported in [17], such

logic-level-only reconfiguration is insufficient to mitigate Trojans that manipulate primary inputs or induce stuck-at faults along signal paths. To address this limitation, the proposed work adopts an enhanced CFG-LUT architecture augmented with input-selection multiplexers. The integration of multiplexing elements introduces additional reconfigurable routing resources, enabling dynamic selection among alternative signal paths and multiple input sources. This capability allows selective bypassing of compromised connections, dynamic modification of internal signal paths, and localized reconstruction of the circuit topology during runtime. As a result, the enhanced CFG-LUT architecture significantly expands the reconfiguration space, enabling Trojan mitigation at both the logic and routing levels.

Figure 3 presents the proposed framework under both the VRC- and CFG-LUT-with-MUX-based implementation scenarios. In both implementations, the framework comprises four major components: (1) the primary circuit under protection, (2) the controller, (3) the detection module, and (4) the Genetic unit. To validate the proposed approach, the MemCtrl-T100 benchmark from the Trust-Hub repository is adopted as the case study. Within the memory controller, two sub-modules are identified as security-critical components and selected for protection. These submodules are implemented in parallel using both the VRC and CFG-LUT-with-MUX architectures, enabling a direct comparison of their effectiveness in runtime adaptation and Hardware Trojan mitigation.

Flash Sleep Control Unit: Generates the control signals required to enable or disable access to the flash interface.

Data Output Control Unit: Controls the operating mode of the data bus by determining when it is configured for data output.

In the proposed framework, both security-critical sub-modules are implemented in parallel on two reconfigurable architectures, namely VRC and CFG-LUT-with-MUX, to enable structural adaptation and runtime functional recovery. As illustrated in Fig. 3, in the VRC-based implementation, the logic gates belonging to the security-critical regions are mapped onto an array of PEs. Each PE consists of a multiplexer-based configurable logic structure that enables runtime modification of both its implemented logic function and input connections. The configuration of the PEs is controlled by the GA unit through dedicated configuration signals. This mechanism enables rapid structural reconfiguration of the circuit, allowing compromised configurations to be replaced without requiring full device reconfiguration.

In the CFG-LUT-with-MUX implementation, the logic gates of the security-critical regions are mapped onto a set of FEs, which are functionally equivalent to the PEs employed in the VRC architecture. Each FE integrates a CFG-LUT with input-selection multiplexers, providing runtime reconfigurability. Similar to the VRC architecture, the GA

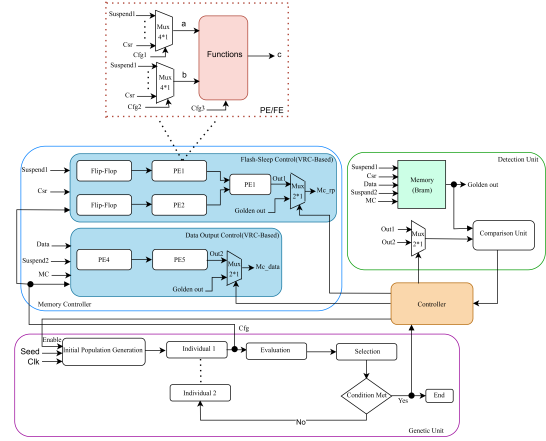


Fig. 3 Overall architecture of the proposed framework based on VRC and CFG-LUT-with-MUX architectures.

unit generates the configuration signals required by each FE. These signals simultaneously determine both the multiplexer input selections and the logic function implemented by the CFG-LUT.

To establish a unified configuration mechanism across both architectures, each PE and FE receives three configuration signals generated by the GA unit. In the VRC architecture, the first two signals determine the multiplexer input selections, while the third signal specifies the logic function implemented by the PE. Likewise, in the CFG-LUT-with-MUX architecture, the same three configuration signals jointly determine the multiplexer input connections and the configuration of the CFG-LUT logic.

The detection unit is based on a golden reference model stored in BRAM. During runtime, it continuously monitors the outputs of the security-critical regions and periodically compares them with the corresponding reference outputs. This mechanism enables reliable detection of functional deviations potentially caused by hardware Trojan activation.

The controller serves as the central management unit, coordinating the operation of all system components and maintaining overall system integrity. During runtime, it continuously analyzes the outputs of the detection unit to identify and isolate Trojan-affected modules, thereby preventing the propagation of compromised outputs. Once a HT is detected, the controller activates the GA unit to initiate the recovery process. The GA unit then explores alternative configurations for the affected regions implemented on the VRC and CFG-LUT-with-MUX architectures, selecting an optimized configuration that restores correct circuit functionality with minimal reconfiguration overhead.

4.1 Security Enhancement Through a Detection Unit

In the proposed architecture, a golden-model-based detection unit is integrated into the system. In both implementa-

tion scenarios, the golden model comprises the truth tables associated with the selected security-critical regions, which are stored in on-chip BRAM.

The detection process is illustrated in Fig. 3. A pseudo-random number generator (PRNG) embedded in the controller randomly selects one of the security-critical output blocks for evaluation, preventing attackers from predicting which region will be monitored at a given time. For the selected block, the controller retrieves the corresponding reference output from BRAM and compares it with the actual circuit output in real time. Any discrepancy between the two outputs is treated as a potential indication of hardware Trojan activity. Upon detection of a mismatch, a detection flag is generated and forwarded to the controller, which subsequently initiates the localization, isolation, and runtime re-configuration procedures.

4.2 Runtime Fault Recovery and Stable Operation Assurance in Reconfigurable Architectures

Once an HT is detected, the controller isolates the affected region to prevent the propagation of erroneous outputs to the remainder of the system. During this period, the golden model, which has already generated the correct reference outputs for the compromised security-critical region during the detection phase, is temporarily employed as a trusted output source. This enables uninterrupted system operation while the affected module is isolated and undergoing re-configuration. As illustrated in Fig. 3, the outputs *mc_rp* and *mc_data* are redirected to the golden model, allowing the system to continue producing correct outputs while bypassing the compromised implementation.

Since the convergence time of the GA depends on the search process and cannot be determined a priori, the affected module remains unavailable until a valid configuration is identified. Consequently, maintaining uninterrupted and functionally correct system operation throughout the recovery interval is essential, particularly for security-critical modules whose temporary failure may compromise overall system reliability. The golden model therefore acts as a compensation mechanism, ensuring correct system operation until the GA converges to a valid configuration. Once the GA identifies an optimized configuration, the controller updates the affected region and redirects the data path from the golden model to the recovered implementation. The system then resumes normal operation, while the golden model continues to serve as the reference for runtime monitoring and subsequent HT detection, thereby preserving long-term system security and functional integrity.

4.3 Initial Population Generation and Genetic Diversity Enhancement

Upon the detection of a HT, the controller activates the Genetic unit to initiate runtime recovery. The GA generates an initial population of chromosomes, where each chromosome represents a candidate configuration for reconstructing the compromised security-critical region. The chromosome encoding is architecture-dependent and is tailored to the configuration interface of the underlying reconfigurable implementation.

For the VRC-based architecture, each chromosome is encoded as a binary string representing the configurable multiplexer selection bits associated with the affected circuit region. In contrast, the proposed CFG-LUT-with-MUX architecture employs a more expressive chromosome representation that jointly encodes both signal routing and logic functionality. In addition to the selection bits of the input multiplexers, which determine the connectivity among circuit signals, each chromosome includes the configuration bits required to program the CFG-LUT truth tables through the CDI interface. Consequently, a single chromosome specifies both the routing configuration and the logic function implemented by the reconfigurable region. This joint representation substantially increases the chromosome length compared with the VRC-based encoding and, consequently, enlarges the corresponding configuration search space. While this increased representation capability provides greater flexibility for runtime structural adaptation, it also makes the evolutionary search more challenging. Therefore, architecture-specific population initialization strategies are adopted for the VRC and CFG-LUT-with-MUX implementations to generate suitable initial search populations and improve the efficiency of the subsequent evolutionary process.

For the VRC architecture, the initial population is generated using a Linear Feedback Shift Register (LFSR) as a pseudo-random number generator. In each generation, nine chromosomes are produced using different LFSR seeds to generate distinct configuration patterns. This initialization strategy increases population diversity during the early stages of evolution, thereby reducing the likelihood of premature convergence and improving the exploration capability of the GA.

Given the substantially larger configuration search space of the CFG-LUT-with-MUX architecture, purely random population initialization may result in an ineffective exploration of the search space and provide limited functional guidance to the evolutionary process. To address this challenge, the proposed framework employs a hybrid, truth-table-guided initialization strategy, as shown in Algorithm 1. Instead of generating chromosomes entirely at random, the proposed method preserves a controlled portion of the functional information extracted from the original circuit while intro-

ducing stochastic perturbations into the remaining configuration bits. This approach provides the initial population with circuit-specific information while retaining sufficient diversity for evolutionary exploration. The proportion of inherited information is adaptively determined from structural and statistical characteristics of the original truth table, including its entropy, maximum run length (*Max_Run*), distribution of logical zeros and ones, logic-state balance, and number of bit transitions. These characteristics are jointly used to derive a *Fidelity* metric, which determines the inheritance ratio and, consequently, the fraction of chromosome bits retained from the original truth table. For each chromosome, the remaining bits are generated through controlled stochastic assignment or bit-flipping operations, thereby introducing genetic diversity while preserving a degree of functional similarity to the original circuit.

By integrating circuit-specific functional information with stochastic variation, the proposed initialization strategy establishes a controlled balance between exploitation and exploration. The resulting population provides the GA with informative initial candidate configurations without restricting the search to a narrowly defined region of the configuration space.

Algorithm 1 Adaptive Truth-Table–Guided Seed Generation

Require: Truth table TT

Ensure: Initial population $\mathcal{P}$

```

1:  $TT \leftarrow \text{READTRUTHTABLE}(\text{truthtable.txt})$ 
2:  $F \leftarrow \text{ANALYZETRUTHTABLE}(TT)$ 
    $F = \{\text{num\_ones}, \text{num\_zeros}, \text{balance}, \text{entropy}, \text{max\_run}\}$ 
3: if  $F.\text{entropy} < 0.8$  then
4:    $\text{fidelity} \leftarrow 0.6$ 
5: else if  $F.\text{max\_run} > 10$  then
6:    $\text{fidelity} \leftarrow 0.7$ 
7: else
8:    $\text{fidelity} \leftarrow 0.85$ 
9: end if
10:  $N_{\text{chr}} \leftarrow 9$ 
11:  $L \leftarrow 128$ 
12:  $TT_{\text{ext}} \leftarrow \text{EXTEND}(TT, L)$ 
13:  $\mathcal{P} \leftarrow \emptyset$ 
14: for  $i \leftarrow 1$  to  $N_{\text{chr}}$  do
15:   Initialize chromosome  $C_i$ 
16:   for  $j \leftarrow 1$  to  $L$  do
17:     if  $\text{RANDOM}() < \text{fidelity}$  then
18:        $C_i[j] \leftarrow TT_{\text{ext}}[j]$ 
19:     else
20:        $C_i[j] \leftarrow 1 - TT_{\text{ext}}[j]$ 
21:     end if
22:   end for
23:    $\mathcal{P} \leftarrow \mathcal{P} \cup \{C_i\}$ 
24: end for
25:  $\text{GENERATECONFIGURATION}(\text{seed\_config.vhdl}, F, \mathcal{P})$ 
26: return  $\mathcal{P}$ 

```

4.4 Fitness Evaluation

After generating the chromosomes using the GA and applying them to reconfigure the target structures under both architectural scenarios, namely **VRC** and **CFG-LUT-with-MUX**, the output of the reconfigured circuit is forwarded to the evaluation stage. At this stage, the fitness of each chromosome is computed independently. During the evaluation phase, the output generated by the reconfigured circuit is systematically compared with the corresponding output in the reference truth table. This comparison quantifies the functional agreement between the reconfigured output $S(x)$ and the reference output $F(x)$ for each input vector x .

During the evaluation phase, the output generated by the reconfigured circuit is systematically compared with the corresponding output in the reference truth table. This comparison provides a quantitative measure of the functional agreement between the reconfigured output $S(x)$ and the corresponding reference output $F(x)$ for each input vector x . Accordingly, a binary evaluation function $G(x)$ is defined to formally assess functional correctness by determining whether the reconfigured architecture produces the expected reference output, as given in Eq. (1). The function returns a value of 1 if the output generated by the reconfigured structure matches the reference output; otherwise, it returns 0. Therefore, $G(x)$ provides a binary metric for assessing correctness at each input instance.

$$G(x) = \begin{cases} 1, & \text{if } S(x) = F(x), \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Subsequently, the fitness value of each chromosome is computed according to Eq. (2) by summing the binary evaluation function $G(x)$ across all input vectors.

$$\text{Fitness} = \sum_{x=0}^{2^n-1} G(x) \quad (2)$$

4.5 Combining Two Chromosome Selection Strategies

After fitness evaluation, the selection phase identifies candidate chromosomes for the next generation and plays a critical role in steering the evolutionary process toward improved solutions. In this paper, a hybrid selection strategy combining elitism and tournament selection is employed to balance convergence speed and population diversity. Elitism preserves a small set of the highest-fitness chromosomes unchanged, ensuring retention of the best solutions and accelerating convergence [25]. In parallel, tournament selection randomly selects a subset of individuals and promotes the

fittest among them for reproduction, providing all chromosomes with a non-zero chance of selection and maintaining diversity, albeit with a potential trade-off in convergence speed [38].

4.6 Intelligent Crossover Based on Chromosome Fitness

After the selection phase, a fitness-proportional single-point crossover is applied to generate the next generation. In this scheme, chromosomes with higher fitness have a greater influence on gene inheritance and participate in more crossover operations. This approach preserves high-quality genetic material while limiting the impact of low-fitness individuals, thereby maintaining population diversity.

4.7 Adaptive Mutation Strategies for Sustaining Genetic Variation

Mutation is a fundamental operator in EAs, playing a critical role in maintaining genetic diversity and preventing premature convergence to local optima. In this paper, a dynamic and adaptive mutation strategy is adopted to balance exploration and exploitation throughout the evolutionary process. Mutation is performed by randomly altering selected gene positions within a chromosome to introduce new genetic variations. Unlike fixed-rate mutation schemes, the mutation rate is adaptively adjusted according to the generation index and chromosome length. A higher mutation rate ($\mu_{\max}$) is applied in early generations to enhance exploration, while it is gradually reduced to a lower bound ($\mu_{\min}$) in later stages to improve convergence stability and solution accuracy.

4.8 Proposed GA Evaluation and Stopping Criteria

In each generation of the GA, the chromosomes with the highest fitness values are selected as the primary candidate solutions. If the output of the reconfigured circuit by a chromosome fully matches the expected output, the evolutionary process terminates, and that chromosome is accepted as the final configuration of the circuit. Otherwise, crossover and mutation operators are applied to the selected chromosomes, and a new population is generated for the next generation. A maximum of 300 generations is considered for each run of the algorithm; however, in practice, convergence typically occurs within far fewer generations.

Table 1 presents the encoding scheme for input and logical functions in the proposed VRC architecture. In this architecture, the chromosome bits that determine the multiplexer selector specify which input path is chosen and which logical operation is applied. Each VRC unit has two inputs and one output, and can dynamically switch between inputs

Table 1 Encoding scheme for genotype representation in the VRC.

Input	Encoding	Function	Encoding
Input0	00	XOR	000
Input1	01	OR	001
Input2	10	AND	010
Input3	11	NAND	011
Input4	00	XNOR	100
Input5	01	NOR	101
Input6	10	(Input1)	110
Input7	11	(Input2)	111

to perform a set of basic logic functions, including XNOR, XOR, NAND, NOR, AND, OR, NOT, and BUFFER. This flexible capability enables dynamic reconfiguration and supports the implementation of a wide range of logic structures. Similarly, in the CFG-LUT-with-MUX architecture, the initial chromosome is partitioned into two functional segments. The first segment encodes the multiplexer input selection bits, whereas the remaining bits represent the configuration values stored in the CFG-LUT. The multiplexer input selection is encoded using the same strategy adopted in the VRC architecture.

However, the fundamental distinction between the two architectures lies in the mechanism used to realize the logical function. In the VRC architecture, the set of candidate logical functions is predetermined at design time. During runtime, the GA can only select one function from this fixed function pool. Therefore, the search space is restricted to a limited and predefined subset of Boolean functions.

In contrast, in the CFG-LUT-with-MUX architecture, the contents of the CFG-LUT—and consequently the implemented logical function—are directly determined by the GA during the reconfiguration phase. This removes the restriction of a predefined function library and enables dynamic function synthesis. This exponential functional capacity dramatically enlarges the search space available to the GA. Consequently, the CFG-LUT-with-MUX architecture provides significantly higher expressive power and functional flexibility compared to the VRC approach, enabling exploration of a substantially broader range of logical implementations.

5 Design Implementation and Experimental Results

To comprehensively evaluate the proposed architectures, three benchmark circuits from the Trust-Hub repository, namely **MemCtrl-T100**, **MemCtrl**, and **AES-128**, are employed as case studies. The objective of the experimental evaluation is to assess the practical feasibility and effectiveness of the proposed runtime reconfigurable architectures under realistic operating conditions. For this purpose, each benchmark

is implemented and evaluated on both the VRC- and CFG-LUT-with-MUX-based architectures using an identical experimental setup. This ensures a fair and unbiased comparison by eliminating the influence of implementation-dependent factors.

The comparative evaluation focuses on the key performance metrics that directly affect the efficiency of runtime recovery, including reconfiguration latency, reconfiguration methodology, GA convergence time, hardware resource utilization, and overall system performance. Through this comprehensive analysis, the strengths and limitations of the two architectures are systematically examined, providing quantitative insights into their capability to support efficient runtime recovery and Hardware Trojan mitigation.

To provide a comprehensive evaluation of the proposed runtime recovery framework, both the **VRC-based** and **CFG-LUT-with-MUX-based** architectures integrated with the GA were implemented on FPGA platforms.

The experimental evaluation considers a diverse set of HTs exhibiting different triggering mechanisms and payload behaviors. These Trojans are strategically inserted at various locations within the benchmark circuits, including primary inputs, primary outputs, and internal signals, in order to comprehensively assess the capability of both architectures to detect, isolate, and recover from hardware attacks through runtime reconfiguration. As a representative example, the **MemCtrl-T100** benchmark incorporates two distinct Trojan variants. The first Trojan permanently forces the *Flash Sleep (FS)* control signal to logic zero, creating a stuck-at-1 fault that prevents access to the flash interface. The second Trojan is activated by a 3-bit counter and manipulates internal circuit signals, thereby altering the functional behavior of the memory controller. These two Trojan variants represent different attack models and are used to evaluate the robustness and adaptability of the proposed runtime recovery architectures under both externally observable and internally activated attack scenarios.

The **AES-T2300** benchmark provides a hardware implementation of the AES cryptographic algorithm and includes a hardware Trojan that maliciously alters the encrypted output by inserting an additional XOR gate.

When runtime reconfiguration is initiated for a compromised region, the architectural constraints of the VRC implementation play a crucial role in the convergence behavior of the GA. In the VRC architecture, the logic functions supported by each PE are fixed during the design phase, while runtime reconfiguration is limited to modifying the interconnections among PEs and selecting from this predefined set of logic functions. Consequently, the available hardware resources directly determine the flexibility of the evolutionary search. For functionally complex or large-scale circuits, allocating only a small number of PEs may severely restrict the search space, preventing the GA from converg-

ing to a feasible solution. In contrast, the same resource allocation may be sufficient for smaller or less complex circuits, where the required functionality can be reconstructed with fewer processing elements. To accommodate the varying complexity of different benchmark circuits while avoiding excessive hardware overhead, an adaptive resource allocation strategy is adopted. Similar to conventional VRC-based approaches [10, 13], the reconfiguration process initially allocates the minimum number of PEs required for the target region. If the GA fails to converge within a predefined convergence criterion, the number of allocated PEs is progressively increased, thereby expanding the available configuration space. At each allocation stage, chromosome evaluation is performed with respect to the current hardware configuration of the affected region, ensuring consistency between resource availability and fitness evaluation. This adaptive allocation mechanism enables efficient utilization of reconfigurable resources while improving the probability of convergence across circuits with different levels of structural complexity. The proposed VRC- and CFG-LUT-with-MUX-based architectures were implemented using the Xilinx Vivado Design Suite and validated on a Xilinx Zynq UltraScale+ FPGA (XCZU3EG-SFVC784-1-I). All experimental results reported in this work were obtained using this hardware platform.

5.1 Reconfiguration Time of the Affected Module

When a Trojan-affected module enters the reconfiguration phase, the GA is triggered to reconstruct the circuit structure. The total time required for this recovery process depends on several timing parameters that govern the execution flow of the GA. These parameters are defined as follows:

- Pop_{size} : Number of chromosomes generated per generation.
- T_{access} : Time required to generate and access each chromosome within the population.
- T_{config} : Time required to load a chromosome into the reconfigurable architecture (VRC or CFG-LUT-with-MUX) during runtime reconfiguration.
- T_{fit} : Time allocated for evaluating the fitness of each chromosome.
- T_{EA} : Time required to perform evolutionary operations, including selection, crossover, and mutation.

Based on these parameters, the reconfiguration latency incurred by a single GA generation is calculated using Eq. (3). Considering the total number of generations required for convergence, denoted by Gen , the overall reconfiguration latency is subsequently determined according to Eq. (4).

$$T_{gen} = Pop_{size} \times (T_{access} + T_{config} + T_{fit}) + T_{EA} \quad (3)$$

$$Total_Time = \sum_{i=1}^{Gen} T_{gen_i} \quad (4)$$

Since the GA is implemented entirely in hardware, genetic operations, including selection, crossover, and mutation, are executed concurrently across the entire population. Consequently, the evolutionary processing time (T_{EA}) is largely independent of the population size and is primarily determined by the latency of the hardware implementation rather than the number of chromosomes. A fundamental distinction between the VRC and CFG-LUT-with-MUX architectures lies in the configuration latency, denoted as T_{config} . In the VRC architecture, reconfiguration is accomplished by updating the multiplexer selection signals that define the interconnections and logic functions of the PEs.

Since these configuration signals are applied simultaneously, all PEs within the target reconfigurable region are updated concurrently in a single clock cycle, resulting in negligible configuration latency. In contrast, the CFG-LUT-with-MUX architecture requires reconfiguration of both the routing paths and the logic contents of the CFG-LUTs. Each CFG-LUT is implemented using an SRL-based structure and programmed through an independent CDI interface. Since the CDI interface operates serially, only one configuration bit can be loaded into a CFG-LUT during each clock cycle. Consequently, the configuration latency is proportional to the number of bits required to program the CFG-LUT, resulting in a significantly larger T_{config} than that of the VRC architecture. Nevertheless, the use of an independent CDI interface for each CFG-LUT enables multiple CFG-LUTs to be programmed concurrently. Therefore, although the configuration process of an individual CFG-LUT is inherently serial, the overall reconfiguration latency is substantially reduced through parallel programming of multiple CFG-LUT instances, partially compensating for the serial loading overhead.

5.2 Memctrl-T100 with Signal-Forcing HT and GA-Driven VRC Recovery

In the MemCtrl-T100 benchmark, the FS control logic is selected as one of the security-critical functional blocks for runtime protection. The original implementation of this module is illustrated in Fig. 4(a), while its corresponding mapping onto the proposed VRC architecture is shown in Fig. 4(b). In the original design, each logic gate is mapped to a dedicated PE, allowing the complete functionality of the FS control circuit to be realized using a minimum of three PEs. As described in the previous section, if the GA does not converge within the predefined convergence criterion, additional PEs are progressively allocated to enlarge the search

space and improve the likelihood of obtaining a feasible recovery solution.

A key feature of the proposed VRC architecture is the incorporation of a multi-level input selection mechanism within each PE to improve resilience against signal contamination caused by HTs. Instead of restricting each PE to receive inputs solely from the immediately preceding logic level, every PE can dynamically select its inputs from one of three alternative sources: (i) the outputs of the immediately preceding logic level, (ii) the outputs of the two preceding logic levels, or (iii) the primary inputs of the protected circuit. The rationale behind this design is that Hardware Trojans often compromise internal signals rather than only modifying the functionality of individual logic gates. Consequently, one or more intermediate nodes may become permanently corrupted through malicious signal manipulation or stuck-at faults. If the reconstructed circuit continues to rely on these compromised nodes, the malicious behavior may persist even after successful logic reconfiguration because the corrupted signals continue to propagate through the reconstructed circuit. To address this challenge, the proposed multi-level input selection mechanism enables the GA to explore alternative signal paths that bypass contaminated intermediate nodes. By allowing each PE to directly receive inputs from earlier logic levels or from the primary inputs, the reconstructed circuit is no longer constrained by the original signal dependencies. This capability significantly reduces the propagation of compromised signals, increases the probability of recovering the correct circuit functionality, and expands the feasible search space available to the GA. As a result, the proposed architecture provides greater robustness against both logic-level and signal-level Hardware Trojan attacks while improving the effectiveness of runtime recovery.

Fig. 4(c) illustrates a representative HT injected into the Flash-sleep control logic. However, in the attack scenario, an adversary embeds a stealthy counter within the circuit structure, which serves as the Trojan's trigger mechanism. Once the counter reaches a predefined threshold value (8'h77), the Trojan is activated. Upon activation, the malicious logic forces the signal *fs* to remain permanently at logic level '1', regardless of the actual value of *csr_r*.

Consequently, all connected flash devices are unintentionally driven into the sleep mode. For example, when the input combination (*suspend1*, *csr*) = (0,0) is applied, the correct circuit output is expected to be '1'. However, after Trojan activation, an erroneous value is propagated to *fs*, which in turn forces the signal *out1* to incorrectly transition to '0'. This behavior demonstrates that the Trojan not only induces a persistent fault at an internal node but also disrupts the entire dependent logic path, thereby compromising the functional correctness of the module.

In the proposed architecture, when the Flash-sleep function is selected for monitoring by the detection unit, the cor-

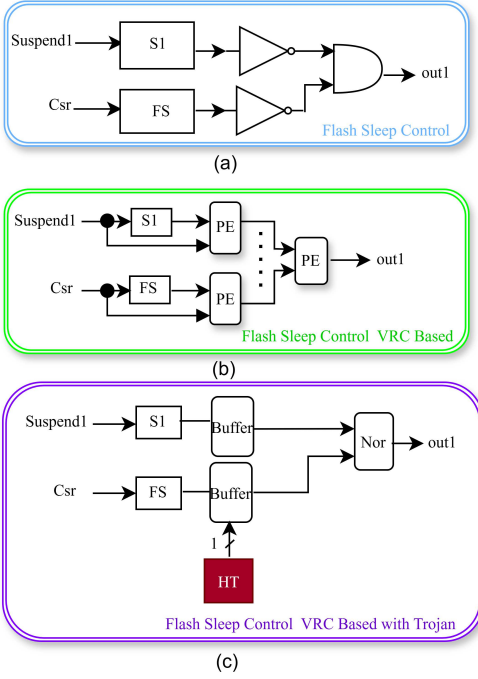


Fig. 4 (a) Original circuit, (b) mapping of the original circuit onto the VRC architecture, and (c) selective runtime reconfiguration following a hardware Trojan attack.

responding golden model is first retrieved from BRAM, and its output is compared with the observed signal *out1*. Upon anomaly detection, the controller immediately redirects the data path *mc_rp* to the golden model output, effectively preventing the propagation of corrupted signals to the rest of the system. Simultaneously, a GA-based reconfiguration process is initiated to restore the functionality of the compromised VRC structure. Experimental results indicate that the GA converges to a valid functional configuration after 15 generations. As shown in Fig. 4(c), in the final evolved structure, the PEs in the first layer select their inputs directly from the primary inputs rather than from the outputs of the preceding logic layer. This configuration decision effectively eliminates dependencies on Trojan-affected signals and prevents the re-propagation of malicious behavior. In the final layer, the corresponding PE receives its inputs from the previous layer and performs a NOR operation, which matches the correct reference logic illustrated in Fig. 4(a). As a result, the reconfigured structure is capable of generating the correct output for all possible input combinations, even in the presence of the activated Trojan.

5.3 Memctrl-T100 with Signal-Forcing HT and GA-Driven CFG-LUT-With-MUX Recovery

The original circuit structure corresponding to the *Flash Sleep Controller*, previously discussed in Section 5.2, is herein

evaluated by mapping it onto the proposed CFG-LUT-with-MUX architecture. Fig. 5(a) illustrates the *Flash Sleep Control* logic mapped onto a CFG-LUT-with-MUX architecture constructed using FEs to enhance structural security.

In the proposed CFG-LUT-with-MUX architecture, the target logic is implemented using three FEs, representing the minimum number of CFG-LUT units required to realize the original circuit functionality. Similar to the VRC-based architecture, each FE incorporates input-selection multiplexers that provide flexible connectivity to different signal sources. The inclusion of these multiplexers constitutes a key architectural feature for improving resilience against HT attacks. In conventional CFG-LUT-based implementations, modifying only the truth-table contents is insufficient to restore the correct functionality when the inputs of a logic block have already been compromised. For example, a Trojan may permanently force an internal signal to logic '0' or '1' or maliciously manipulate an upstream signal that subsequently propagates through the circuit. Under these circumstances, reconfiguring the logic function of the CFG-LUT alone reproduces the same erroneous behavior because the corrupted input signals remain unchanged. To address this limitation, each FE is augmented with input-selection multiplexers that enable the GA to dynamically establish alternative signal paths during runtime reconfiguration. By selecting inputs from different logic levels or directly from the primary inputs, the reconstructed circuit can bypass contaminated intermediate nodes rather than relying on compromised signal paths. Consequently, the proposed architecture simultaneously reconfigures both the implemented logic functions and their corresponding input connections. This dual reconfiguration capability substantially enhances the effectiveness of runtime recovery by preventing the propagation of Trojan-contaminated signals, increasing the flexibility of the evolutionary search, and improving the probability of restoring the intended circuit functionality. Experimental results show that the GA successfully converges to a valid configuration after 17 generations. As depicted in Fig. 5(b), in the final evolved structure, the GA configures the FEs in the first column such that each FE directly receives its inputs from the primary circuit inputs (*csr*) rather than relying on signal such as *fs*, which may be compromised by the Trojan. Furthermore, the GA configures the internal CFG-LUTs of the two FEs in the first column to implement NOT operations, while in the final column, each FE selects its inputs from the preceding column and performs an AND operation. This configuration precisely matches the correct reference logic of the original circuit shown in Fig. 5(a), thereby guaranteeing correct outputs for all possible input combinations. It is important to note that if only CFG-LUT reconfiguration were available without the MUX-based input selection layer, the GA would not be able to recover correct functionality. Since the Trojan directly corrupts the signal *fs*—which serves as

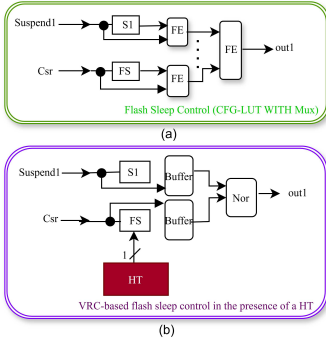


Fig. 5 (a) Mapping of the original circuit onto the proposed CFG-LUT-with-MUX architecture and (b) selective runtime reconfiguration following a hardware Trojan attack.

an input to the CFG-LUT—any modification of the internal LUT truth table alone would remain ineffective, as *fs* would continue to be forced to a constant value.

5.4 Inversion Trojan in MemCtrl and GA-Driven VRC Recovery

Another experimental scenario is conducted using the **MemCtrl** benchmark to evaluate the effectiveness of the proposed framework against a different class of HT. Unlike the attack scenario described in Section 5.2, where the Trojan induces a *stuck-at* fault by forcing an internal signal to a constant logic value, the Trojan considered in this experiment performs dynamic logic manipulation by inverting the intermediate signal *s3* within the target circuit. The Trojan is activated when an internal counter reaches a predefined trigger value. Upon activation, the value of *s3* is inverted, and the resulting erroneous signal propagates through the circuit to the output *out1*. For example, when the input vector (*suspend1*, *csr*) = (0, 1) is applied, the fault-free circuit produces the expected output value of '0'. However, after Trojan activation, the inversion of *s3* causes *out1* to incorrectly change to '1'.

The final evolved configuration, shown in Fig. 6, is obtained by the GA after 29 generations. During the evolutionary process, the GA treats the Trojan as a persistent component of the operational environment and attempts to synthesize a compensatory logic structure that restores functional correctness. Accordingly, in the final VRC configuration, a NOT operation is assigned to the PE affected by the Trojan in the first column, thereby canceling the inversion imposed on *s3*. Subsequently, in the final column, a NOR operation is applied to the signals *s2* and *s3* to reconstruct the correct logical functionality of the original circuit. As a result, the reconfigured structure guarantees correct output generation for all possible input combinations, even in the presence of the activated Hardware Trojan.

S1

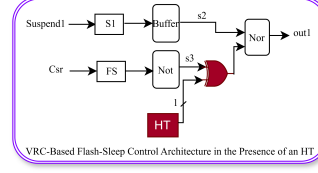


Fig. 6 Runtime reconfiguration process of the proposed VRC architecture following a hardware Trojan attack.

5.5 Inversion Trojan in MemCtrl and GA-Driven CFG-LUT-With-MUX Recovery

In the evaluated **MemCtrl** scenario (Section 5.4), the attack recovery mechanism was previously analyzed for the VRC architecture. In this section, the same target logic and identical attack conditions are re-examined using an FE-based implementation in order to evaluate the recovery capability of the alternative architecture. According to the final evolved configuration shown in Fig. 7, the GA converges to a stable and functionally valid configuration after 33 generations. Throughout the evolutionary process, the GA treats the HT as a permanent component of the operational environment and optimizes the internal configuration of the computational elements accordingly.

In the final configuration, a buffer function is assigned to the CFG-LUT whose output corresponds to the signal *s3*, which is compromised by the Trojan. This design choice is motivated by the fact that the Trojan inverts *s3*. Consequently, the combination of the buffer operation with the Trojan-induced inversion effectively reproduces the behavior of a logical NOT gate. In the final stage, an AND operation is applied to the signal *s2* and the corrected version of *s3*, thereby reconstructing the intended logical functionality of the original circuit. As a result, the reconfigured architecture guarantees correct output generation for all possible input combinations, despite the continued presence of the activated Hardware Trojan.

5.6 Analysis of the AES-T2300 and GA-Driven VRC Recovery

Another benchmark considered in this study is **AES-T2300**. As illustrated in Fig. 8, the *AddRoundKey* module in the final AES encryption round is selected as the security-critical region to evaluate the effectiveness of the proposed runtime recovery framework in a large-scale cryptographic application. Since this module produces the final ciphertext, any

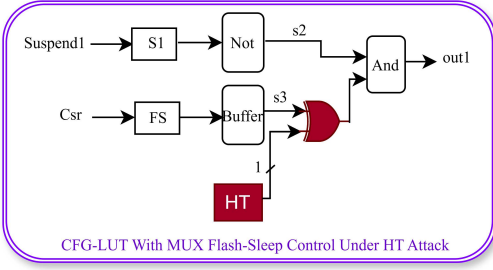


Fig. 7 Runtime reconfiguration process of the proposed CFG-LUT-with-MUX architecture following a logic inversion hardware Trojan attack.

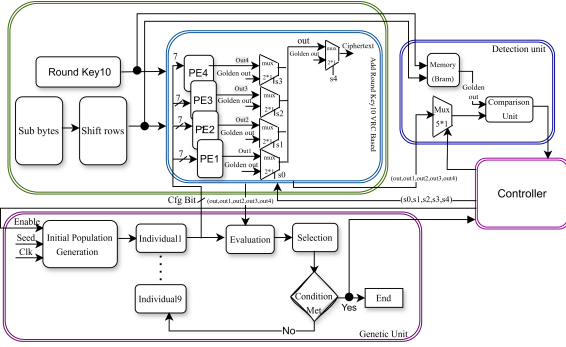


Fig. 8 Mapping of the AES-T2300 benchmark onto the proposed VRC architecture.

error introduced at this stage directly affects the encryption output. Therefore, protecting the *AddRoundKey* module is essential for preserving the correctness and integrity of the encrypted data.

The *AddRoundKey* operation receives a 128-bit plaintext state, corresponding to the output of the ninth AES round, together with a 128-bit round key. Both inputs are partitioned into four 32-bit segments, and each PE is assigned one 32-bit plaintext segment and the corresponding 32-bit round-key segment. Each PE independently performs a bit-wise XOR operation, and the final 128-bit output (*out*) is obtained by concatenating the outputs of the four PEs. This partitioning enables parallel execution while preserving the original functionality of the *AddRoundKey* operation.

Upon detecting malicious behavior, the controller immediately replaces the compromised output with the corresponding output generated by the golden model, thereby maintaining correct system operation throughout the runtime recovery process. Subsequently, a bit-level comparison is performed between the circuit output and the reference output to accurately localize the compromised PE.

As illustrated in Fig. 9, the **AES-T2300** benchmark incorporates a HT into the *AddRoundKey* module of the final AES encryption round. The Trojan is activated by the control signals *Trigger1* and *Trigger2* and inserts a malicious

XOR gate into the data path, causing the first bit of the final ciphertext to be inverted while leaving the remaining bits unaffected. Following Trojan activation, the proposed detection module performs a bit-level comparison between the circuit output and the corresponding golden-model output.

Since only the first bit of the ciphertext is corrupted, the generated comparison vector contains a logic '1' only in its first position, while all remaining bits remain equal to '0'. As explained previously, the final ciphertext is obtained by concatenating the outputs of the four PEs. Therefore, an error occurring in the first ciphertext bit can be directly mapped to the first 32-bit partition of the *AddRoundKey* operation, which is processed by **PE1**. Based on the comparison result, the controller accurately identifies **PE1** as the faulty processing element and confines the runtime recovery procedure exclusively to this PE. Without the controller's fault-localization analysis, it would be impossible to determine which processing element is responsible for the observed error, since the final ciphertext is formed by concatenating the outputs of all four PEs. Consequently, the recovery mechanism would be forced to evolve and reconfigure all four PEs simultaneously rather than only the faulty one. By localizing the fault prior to initiating the recovery process, the proposed architecture restricts the evolutionary reconfiguration to the affected PE only. As a result, the proposed selective recovery strategy substantially reduces the reconfiguration overhead, preserves the configurations of the unaffected PEs, limits the search space explored by the GA, and improves its convergence efficiency without disturbing the normal operation of the healthy processing elements.

The reconfiguration process is initially performed using the minimum number of PEs required to implement the target functionality. If the GA fails to converge within 33 generations, the controller dynamically allocates an additional PE to enlarge the structural search space. As shown in Fig. 9, **PE5** is introduced to cooperate with **PE1** during the reconstruction of the first 32-bit partition, while the recovered output is generated by **PE5**. To further enhance the exploration capability of the evolutionary search, **PE5** is instantiated with a structural configuration different from those of the existing processing elements. This controlled architectural diversity enables the GA to explore candidate circuit structures that are unattainable under the original resource constraints, thereby increasing the expressive capability of the VRC architecture and improving the probability of successful runtime recovery. The input design of **PE5** follows three fundamental principles. First, it can directly access the primary inputs of the *AddRoundKey* stage, namely the round key and plaintext. Second, outputs from the previous layer remain selectable as candidate inputs. Third, the comparison result vector is introduced as an auxiliary input. Since each bit in the *AddRoundKey* architecture is processed independently, the comparison vector provides a fine-grained

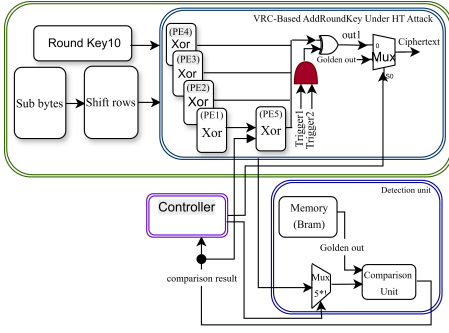


Fig. 9 Convergence behavior of the proposed GA for runtime reconfiguration of the AES-T2300 benchmark under a hardware Trojan attack on the proposed VRC architecture.

correctness indicator: only the attacked bit is marked as ‘1’, while all unaffected bits remain ‘0’. This mechanism enables selective correction of the corrupted bit without disturbing structurally correct bits. In effect, the comparison signal functions as a high-resolution damage map.

According to the final configuration selected by the GA, the primary XOR operation between ROUNDKEY and PLAINTEXT is assigned to **PE1**, forming the core functionality of the *AddRoundKey* operation. The output of **PE1**, together with the comparison vector, is then fed into **PE5**, where a final XOR operation is performed. This operation selectively inverts only the corrupted bit identified by the comparison vector, while leaving all other bits unchanged. The output of **PE5** subsequently propagates through the original data path and undergoes a second inversion caused by the Trojan. These two consecutive inversions cancel each other out, ultimately restoring the correct and original ciphertext output.

5.7 Analysis of the AES-T2300 and GA-Driven CFG-LUT With MUX Recovery

In Section 5.6, the AES-T2300 attack mechanism and the corresponding recovery process using the **VRC** architecture were analyzed in detail. In this section, the same target logic and attack scenario are re-evaluated using the *CFG-LUT-with-MUX* architecture to provide a consistent comparison of the structural characteristics and hardware implications of the two approaches.

As discussed previously, the *AddRoundKey* operation is performed in the final round of the AES encryption process, where a 128-bit round key and a 128-bit state vector are provided as inputs. Both vectors are partitioned into four independent 32-bit segments to enable parallel hardware implementation. Within each 32-bit segment, every bit of the round key is independently combined with the corresponding state bit through a bitwise XOR operation.

Consequently, the hardware realization of each 32-bit segment requires 32 independent LUTs operating in parallel, with each LUT implementing a single-bit XOR function. Since a CFG-LUT is functionally equivalent to a conventional LUT, the proposed FE-based architecture implements each 32-bit *AddRoundKey* segment using 32 parallel CFG-LUTs.

As illustrated in Fig. 10, each FE comprises 32 CFG-LUT instances operating in parallel, thereby preserving the inherent parallelism of the original *AddRoundKey* operation and maintaining the throughput of the AES encryption process. In addition to the CFG-LUT array, each FE incorporates two input-selection multiplexers for configuring the corresponding 32-bit input vectors. The multiplexer selection signals are determined by the chromosome generated by the GA, enabling the multiplexers to dynamically select the appropriate 32-bit segments of the round key and AES state as inputs to the FE. A key feature of the proposed architecture is the independent configurability of all 32 CFG-LUTs within each FE through dedicated CDI interfaces. This fine-grained configuration capability allows individual CFG-LUTs to be reconfigured at runtime without altering the configuration of the remaining LUTs in the same FE. Consequently, the architecture provides enhanced flexibility for localized runtime recovery while minimizing unnecessary reconfiguration of unaffected logic.

In the evaluated attack scenario, the injected hardware Trojan introduces a stealthy XOR gate that flips only the first bit of the final ciphertext. Consequently, the malicious behavior is localized to **CFG-LUT#1** within **FE1**, while all other CFG-LUTs remain functionally correct. Based on the analysis of the comparison vector, the controller accurately identifies the compromised CFG-LUT and subsequently invokes the GA to initiate the runtime recovery procedure for **FE1**. During this process, only the affected CFG-LUT is reconfigured, whereas all CFG-LUTs in the remaining FEs preserve their existing configurations and continue performing their previously assigned XOR operations without interruption.

- *CFG-LUTs #2–#32* in **FE1** retain their original XOR functionality.
- *CFG-LUT#1* is reconfigured from XOR to XNOR to compensate for the Trojan-induced inversion.

Following reconfiguration, *CFG-LUT#1* is configured to implement an XNOR function, which compensates for the inversion introduced by the hidden Trojan XOR gate. The Trojan subsequently inverts the XNOR output, and the two consecutive inversions cancel each other, resulting in the same functional behavior as the original XOR operation. Therefore, although the internal configuration of *CFG-LUT#1* is modified to XNOR, the overall recovered path is functionally equivalent to the fault-free XOR-based implementation.

Consequently, the proposed framework compensates for the Trojan-induced error by modifying only a single CFG-LUT, avoiding unnecessary reconfiguration of unaffected resources while introducing negligible additional hardware overhead.

This observation highlights a fundamental difference between the proposed CFG-LUT-with-MUX and VRC architectures in terms of their reconfiguration capabilities and evolutionary search behavior. In the **CFG-LUT-with-MUX** architecture, the GA has direct access to the internal configuration memory of the CFG-LUTs through the dedicated CDI interfaces, enabling dynamic modification of their stored truth tables during runtime reconfiguration. This fine-grained logic-level programmability enables each FE to realize a wide range of logical functions without requiring additional hardware replication. Consequently, the architecture inherently provides a larger functional search space, reducing the necessity for allocating additional FEs during the recovery process and minimizing the associated hardware overhead. Conversely, the PE-based VRC architecture relies on a fundamentally different reconfiguration mechanism. In this approach, each PE is composed of a predefined set of logic functions that are determined during the design phase. During runtime recovery, the GA can only modify the interconnection pattern and select among these predefined functions; it cannot alter the internal logic behavior of a PE. Therefore, the functional diversity available to the GA is inherently constrained by the initial design assumptions. Since the exact characteristics and activation mechanisms of HTs cannot be fully predicted during the design phase, the predefined function library may be insufficient when encountering unforeseen malicious modifications. Under such circumstances, the evolutionary search may fail to converge within the available structural resources because the existing PEs cannot provide an adequate set of candidate functions. To overcome this limitation, additional PEs must be introduced to enlarge the structural search space and increase the probability of obtaining a valid recovery configuration. However, this approach increases resource utilization and introduces additional hardware overhead compared with the CFG-LUT-with-MUX architecture, where functional diversity is primarily achieved through runtime modification of LUT contents.

5.8 Threat Model

In this section, the detectable HT categories and the corresponding threat models addressed by the proposed architecture are discussed. The proposed framework is designed to mitigate the effects of the following classes of HT attacks.

Manufacturing-Level Trojans: Introduced during fabrication by modifying silicon characteristics. The architecture

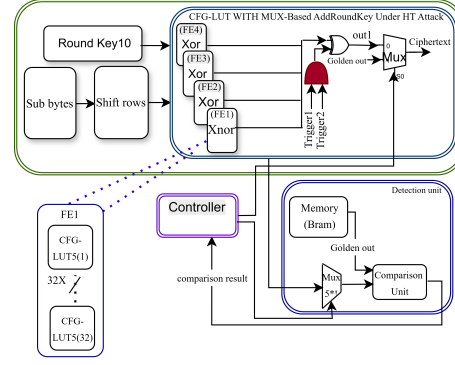


Fig. 10 Convergence behavior of the proposed GA for runtime reconfiguration of the AES-T2300 benchmark under a HT attack on the proposed CFG-LUT-with-MUX architecture.

ture mitigates their effect through targeted reconfiguration of affected FPGA regions.

EDA-Level Trojans: Injected during synthesis, placement, or routing. Runtime monitoring detects functional deviations, allowing recovery via dynamic configuration updates.

Bitstream-Level Trojans: Malicious logic embedded within the FPGA configuration bitstream. The proposed approach mitigates such attacks by selectively updating LUT contents in CFG-LUT or reconfiguring PE in VRC interconnections, restoring correct circuit functionality without requiring a full device reprogramming.

5.9 Evaluation Results

This section presents a comprehensive evaluation of the proposed framework under both implementation scenarios, namely the **VRC** and **CFG-LUT-with-MUX** architectures. The objective of this evaluation is to systematically analyze and compare the effectiveness of both approaches under Hardware Trojan attack scenarios. The assessment focuses on key performance metrics, including hardware resource utilization, GA convergence behavior, recovery success rate, and the capability of the controller to exploit Trojan detection information for accurate localization and selective recovery of the compromised modules. This comparative analysis provides a thorough insight into the effectiveness of the proposed approach in detecting hardware Trojans and restoring correct circuit functionality.

Table 2 summarizes the hardware resource utilization, GA convergence behavior, and power consumption of the proposed framework for the **MemCtrl-T100**, **MemCtrl**, and **AES-T2300** benchmarks under both the VRC and CFG-LUT-with-MUX architectures. For the **MemCtrl-T100** benchmark, the VRC implementation converged after 15 generations, resulting in a design that utilizes 1080 LUTs, 1000

registers, and two RAM blocks. Under the CFG-LUT-with-MUX architecture, the GA converged after 17 generations, while the final implementation required slightly fewer LUTs (1073) with the same number of registers and RAM blocks. This result indicates that the enhanced configurability of the CFG-LUT-based architecture can achieve a modest reduction in logic resource utilization with only a marginal increase in convergence time. A similar trend is observed for the **MemCtrl** benchmark. The VRC implementation converged after 29 generations, whereas the CFG-LUT-with-MUX architecture required 33 generations. Although the CFG-LUT-based implementation exhibits a slightly slower convergence rate, it consistently reduces LUT utilization while maintaining the same register count. Compared with the baseline implementation, which requires 1054 LUTs and 968 registers, both reconfigurable architectures introduce additional hardware overhead due to the resources required for runtime recovery and adaptive reconfiguration.

The difference between the two architectures becomes more pronounced in the AES-T2300 benchmark. The VRC implementation reached convergence after 33 generations and required 2708 LUTs and 4096 registers. In contrast, the CFG-LUT-with-MUX implementation converged after 113 generations but reduced LUT utilization to 2610 while maintaining the same register count. Compared with the baseline AES implementation, which occupies 2562 LUTs and 3968 registers, both approaches incur additional resource overhead to support runtime recovery. Nevertheless, the CFG-LUT-with-MUX architecture achieves a more area-efficient implementation at the expense of a larger evolutionary search space, which increases the number of generations required for convergence.

The results reveal a clear architectural trade-off between the proposed VRC and CFG-LUT-with-MUX implementations. Across all evaluated benchmarks, the VRC architecture consistently requires more LUT resources than the CFG-LUT-with-MUX architecture. This difference is primarily attributed to the underlying organization of the reconfigurable logic in the two architectures. The VRC architecture employs a coarse-grained reconfigurable structure in which each PE is designed to support multiple logic functions. To provide this flexibility, each PE incorporates additional multiplexers, configuration bits, and routing resources that enable runtime selection among different logic operations. Although this organization simplifies runtime reconfiguration and reduces the complexity of the evolutionary search process, it also introduces additional hardware overhead. Consequently, even when only a single logic function is active, the logic resources required to support the remaining functions remain physically allocated within the PE, leading to increased LUT utilization. In contrast, the proposed CFG-LUT-with-MUX architecture employs a fine-grained reconfiguration mechanism in which the required logic function is

directly implemented by dynamically updating the contents of the CFG-LUTs at runtime. Instead of embedding multiple logic functions within each reconfigurable element, only the truth table stored in the CFG-LUT is modified to realize the desired function. This eliminates the need for much of the additional logic required by the VRC architecture, thereby reducing the overall LUT utilization while preserving the same functionality.

Table 3 summarizes the GA parameters used to optimize the runtime reconfiguration process for both the VRC-based and the CFG-LUT with MUX architectures. The same GA settings are employed for all evaluated benchmarks, including Memctrl-T100, Memctrl, and AES-T2300, ensuring a fair comparison between the two architectures. In all experiments, the GA operates with a fixed population size of nine chromosomes and a maximum of 300 generations. The chromosome representation depends on the underlying reconfigurable architecture. In the VRC-based architecture, each chromosome encodes the configuration of the target PEs, whereas in the CFG-LUT with MUX architecture, it represents the configuration of the corresponding FE. Both PEs and FEs are constructed using a two-input, single-output logic structure; however, their configuration mechanisms differ significantly.

For the AES-T2300 benchmark implemented on the VRC architecture, the chromosome length is only 14 bits, corresponding to the configuration of two PEs (*PE1* and *PE5*). Each PE requires seven configuration bits: two bits for selecting the first input, two bits for selecting the second input, and three bits for selecting the logic function through the output multiplexer. In contrast, the chromosome length increases to 132 bits in the CFG-LUT with MUX architecture. This increase originates from the fine-grained configuration strategy adopted in each FE. Specifically, every FE contains 32 CFG-LUTs, each implementing the logic associated with one plaintext bit and one key bit. Since each CFG-LUT must define the output values for all four possible input combinations, four configuration bits are required per CFG-LUT, resulting in a total of 128 configuration bits. Furthermore, each FE includes two input multiplexers, each requiring two additional configuration bits. Consequently, the overall chromosome length reaches 132 bits for a single FE. The CFG-LUT with MUX architecture consistently exhibits a higher runtime reconfiguration latency than the VRC architecture. For instance, in the Memctrl-T100 benchmark, the reconfiguration time increases from 1275 ns to 1500 ns. The difference in reconfiguration latency is primarily determined by the distinct configuration mechanisms employed by the two architectures. In the VRC architecture, once the optimal chromosome has been identified, the configuration information of the VRC units can be applied in parallel within a single clock cycle. Thus, the configuration of the affected region does not require sequential transfer of individual configura-

Table 2 Comparison of resource utilization under different attack and protection scenarios.

Application	Resource	Util.	Used	Avail.
<i>Memctrl-T100 VRC</i>	Reg.	0.71	1000	140000
	LUT	1.54	1080	70560
	BRAM	0.90	2	216
	IO	–	143	252
	Power	–	0.74+0.30	–
<i>Memctrl-T100 CFG-MUX</i>	Reg.	0.71	1000	140000
	LUT	1.52	1073	70560
	BRAM	0.90	2	216
	IO	–	143	252
	Power	–	0.74+0.30	–
<i>Memctrl VRC</i>	Reg.	0.71	1000	140000
	LUT	1.54	1080	70560
	BRAM	0.90	2	216
	IO	–	143	252
	Power	–	0.74+0.30	–
<i>Memctrl CFG-MUX</i>	Reg.	0.71	1000	140000
	LUT	1.47	1073	70560
	BRAM	0.90	2	216
	IO	–	143	252
	Power	–	0.74+0.30	–
<i>AES-T2300 VRC</i>	Reg.	2.92	4096	140000
	LUT	3.80	2708	70560
	BRAM	40.74	88	216
	IO	–	170	252
	Power	–	0.12+0.14	–
<i>AES-T2300 CFG-MUX</i>	Reg.	2.92	4096	140000
	LUT	3.60	2610	70560
	BRAM	40.74	88	216
	IO	–	170	252
	Power	–	0.12+0.90	–

tion bits. In contrast, the CFG-LUT-with-MUX architecture employs an SRL-based configuration mechanism in which the configuration data of the CFG-LUTs are transferred serially through the CDI interfaces. Consequently, multiple clock cycles are required to load the complete configuration before it can be applied to the affected region. As defined in Eq. (3), the remaining stages contributing to the duration of each GA generation, including fitness evaluation and the associated evolutionary operations, are identical for both architectures. Therefore, the difference in the resulting reconfiguration latency is attributable to the configuration-time component, T_{config} . The larger T_{config} of the CFG-LUT-with-MUX architecture reflects the additional latency introduced by its serial configuration mechanism, and consequently increases the total runtime recovery latency compared with the VRC architecture.

Table 4 presents four runtime reconfiguration strategies evaluated using the AES benchmark on both the VRC and CFG-LUT-with-MUX architectures. Among these, Strategies 1 and 2 correspond to the proposed recovery approaches.

In both strategies, a dedicated controller analyzes the comparison vector immediately after Hardware Trojan detection to accurately localize the compromised module. Based on this localization, the reconfiguration process is confined exclusively to the affected component, thereby avoiding unnecessary modification of healthy resources. By restricting the optimization process to the faulty module, the controller substantially reduces the search space explored by the GA, leading to faster convergence and lower computational overhead.

Strategy 1 is implemented on the VRC architecture, where the chromosome length is limited to only 14 bits because the recovery process targets a single affected PE. Owing to this compact search space, the GA converges after only 33 generations. Moreover, the proposed approach achieves a recovery success rate of 5%, indicating that five independent valid configurations capable of restoring the correct circuit functionality are discovered during the evolutionary process.

Strategy 2 is implemented on the CFG-LUT-with-MUX architecture. Similar to Strategy 1, the controller exploits the comparison vector to localize the attack and confines the recovery process to the affected FE only. Although the chromosome length is considerably larger due to the fine-grained configuration of the CFG-LUTs, the reduced search space still enables the GA to converge after 113 generations while achieving a recovery success rate of 2%.

In contrast, Strategies 3 and 4 evaluate the recovery process without employing the proposed localization controller. Consequently, the GA has no information regarding the location of the compromised module and must simultaneously evolve all four PEs. This dramatically enlarges the search space by including both faulty and healthy components in the optimization process, preventing the GA from converging to a valid solution even after the maximum limit of 300 generations.

Overall, the experimental results demonstrate that integrating a fault-localization controller is essential for efficient runtime recovery. By accurately identifying the compromised module prior to reconfiguration, the proposed controller minimizes the search space of the GA, reduces the optimization complexity, preserves the configurations of unaffected modules, and enables reliable convergence to valid recovery solutions.

Table 5 presents a comprehensive evaluation of different mutation-rate strategies for the proposed GA-based runtime recovery framework. The results consistently demonstrate that gradually decreasing the mutation rate during evolution provides a better balance between exploration and exploitation, leading to faster convergence and higher recovery success rates than fixed mutation-rate configurations.

For the *Memctrl-T100* benchmark implemented on the VRC architecture with a 13-bit chromosome, the most effective strategy employs an adaptive mutation schedule that

Table 3 Key parameters used in the genetic algorithm configuration for VRC and CFG-LUT-with-MUX optimization.

Parameter	Mem-Controller-T100 (VRC)	Mem-Controller-T100 (CFG-LUT-MUX)	Mem-Controller (VRC)	Mem-Controller (CFG-LUT-MUX)	AES-T2300 (VRC)	AES-T2300 (CFG-LUT-MUX)
Number of inputs (n_i)	2	2	2	2	2	2
Number of outputs (n_o)	1	1	1	1	1	1
Population size	9	9	9	9	9	9
Chromosome length (bits)	13	16	13	16	14	132
Termination criterion (generations)	300	300	300	300	300	300
Reconfiguration time (ns)	1,275	1,500	2,175	3,300	4,851	25,404
Number of active VRC modules	4	–	4	–	5	–

Table 4 Comparison of reconfiguration strategies for mitigating AES-T2300 attacks.

Method	Chr.	Gen.	Success (%)
AES-T2300 with Controller in VRC	14	33	5
AES-T2300 with Controller in CFG-LUT with MUX	132	113	2
AES-T2300 without Controller in VRC	28	–	–
AES-T2300 without Controller in CFG-LUT with MUX	132	–	–

decreases from 15.38% (two-bit mutations) during the early generations to 7.69% (single-bit mutations) in the later stages. This configuration enables the GA to converge after only 15 generations while achieving a recovery success rate of 7%. Increasing the initial mutation rate to 23.07% or adopting a fixed mutation rate either delays convergence or prevents the GA from converging altogether. A similar trend is observed for the CFG-LUT with MUX implementation of the *Memctrl-T100* benchmark. With a 16-bit chromosome, an adaptive mutation schedule decreasing from 8% to 6.25% results in convergence after 17 generations with a recovery success rate of 4%. In contrast, larger initial mutation rates (e.g., 18.75%) or constant mutation rates increase the number of generations required for convergence and reduce recovery effectiveness. The same behavior is evident for the *Memctrl* benchmark. In the VRC architecture, reducing the mutation rate from 15.38% to 7.69% enables convergence after 29 generations with a recovery success rate of 5%. Likewise, in the CFG-LUT with MUX architecture, decreasing the mutation rate from 8% to 6.25% leads to convergence after 33 generations while achieving a recovery success rate of 3%. These results confirm that adaptive mutation improves evolutionary stability by promoting broad exploration during the early generations and refined exploitation during the later stages.

For the *AES-T2300* benchmark, the influence of chromosome length becomes more pronounced. In the VRC architecture, where the chromosome contains 14 bits, gradually reducing the mutation rate from 14.2% to 7.14% enables convergence after 33 generations with a recovery success rate of 5%. In contrast, the CFG-LUT with MUX architecture employs a substantially larger chromosome of 132 bits, requiring a multi-stage adaptive mutation schedule. In this case, progressively decreasing the mutation rate from 3.78% to 1.51% and finally to 0.75% allows the GA to converge

Table 5 Comparison of genetic algorithm behavior with different mutation rates.

Method	Mutation Rate	Success (%)	Gen.
Memctrl-T100 (VRC)	(15.38%, 7.69%)	7	15
	(23.07%, 7.69%)	3	18
	(30.76%, 7.69%)	–	–
	7.69%	–	–
Memctrl-T100 (CFG-LUT-MUX)	(8%, 6.25%)	4	17
	(18.75%, 6.25%)	3	19
	(18.75%, 8%)	1	36
	6.25%	–	–
Memctrl (VRC)	(15.38%, 7.96%)	5	29
	(23.07%, 7.96%)	–	–
	7.96%	5	44
Memctrl (CFG-LUT-MUX)	(8%, 6.25%)	3	33
	(18.75%, 6.25%)	–	–
	(18.75%, 8%)	–	–
	6.25%	3	38
AES-T2300 (VRC)	(14.2%, 7.14%)	5	33
	(21.42%, 7.14%)	1	37
	7.14%	1	36
AES-T2300 (CFG-LUT-MUX)	(3.78%, 1.51%, 0.75%)	2	113
	(4.54%, 1.51%, 0.75%)	–	–
	(3.03%, 1.51%, 0.75%)	2	178
	(2.27%, 1.51%, 0.75%)	–	–
	(1.51%, 0.75%)	–	–
	0.75%	–	–

after 113 generations while maintaining a recovery success rate of 2%. The lower mutation rates adopted for this architecture prevent excessive disruption of promising candidate solutions while still providing sufficient diversity to effectively explore the considerably larger search space.

Overall, the experimental results demonstrate that the proposed adaptive mutation strategy consistently outperforms fixed mutation-rate schemes across all evaluated benchmarks. By dynamically reducing the mutation rate as evolution progresses, the GA achieves an effective balance between global exploration and local exploitation, thereby accelerating convergence, improving recovery success, and enhancing the robustness of the runtime reconfiguration process.

Table 6 summarizes the experimental evaluation of different crossover strategies employed in the proposed GA-based recovery framework. The primary objective of this

study is to investigate how different gene inheritance ratios between parents with unequal fitness influence the convergence characteristics and recovery capability of the GA. The experimental results consistently show that biasing the crossover operator toward the fitter parent significantly improves the evolutionary process. Except for the conventional 50%/50% crossover, all evaluated strategies assign a larger proportion of genes to the higher-fitness parent while preserving a limited contribution from the lower-fitness parent. This strategy exploits high-quality genetic material while maintaining sufficient diversity to avoid premature convergence.

For the *Memctrl-T100* benchmark implemented on the VRC architecture, the best performance is obtained when 61.53% of the offspring genes are inherited from the higher-fitness parent and the remaining 38.46% from the lower-fitness parent. Under this configuration, the GA converges after only 15 generations and achieves the highest recovery success rate of 7%. A similar trend is observed for the corresponding CFG-LUT with MUX implementation, where a crossover ratio of 68%/32% enables convergence after 17 generations with a recovery success rate of 4%.

The same behavior is observed for the *Memctrl* benchmark. In the VRC architecture, the optimal crossover ratio is 69.23%/30.76%, resulting in convergence after 29 generations and a recovery success rate of 5%. Likewise, the CFG-LUT with MUX architecture achieves its best performance using a crossover ratio of 62.5%/37.5%, whereas equal crossover or larger contributions from the lower-fitness parent noticeably degrade the convergence performance.

For the *AES-T2300* benchmark, the optimal crossover ratio in the VRC architecture is 64.28%/35.72%, allowing the GA to converge after 33 generations while maintaining a recovery success rate of 5%. In contrast, increasing the contribution of the lower-fitness parent degrades convergence, demonstrating that excessive inheritance from inferior individuals disrupts the propagation of beneficial genetic material. A similar tendency is observed in the CFG-LUT with MUX architecture, where a crossover ratio of 64.28%/35.72% enables convergence after 113 generations with a recovery success rate of 2%.

Overall, the experimental results demonstrate that allowing the offspring to inherit approximately 60%–70% of its genes from the higher-fitness parent provides the most effective trade-off between convergence speed and recovery reliability. This crossover policy efficiently exploits high-quality candidate solutions while preserving sufficient genetic diversity through the remaining contribution of the lower-fitness parent. Consequently, the proposed crossover strategy accelerates convergence, improves recovery success, and enhances the robustness of the evolutionary runtime reconfiguration process.

Table 6 Comparison of Genetic Algorithm Performance with Different Crossover Rates.

Method	Crossover (High, Low)	Gen.	Success (%)
<i>Memctrl-T100</i> (VRC)	(61.53, 38.46)	15	7
	(69.23, 30.76)	19	7
	(53.84, 46.15)	21	2
	50/50	19	7
<i>Memctrl-T100</i> (CFG-LUT-MUX)	(68, 32)	17	4
	(75, 25)	41	4
	(62, 38)	25	1
	50/50	29	4
<i>Memctrl</i> (VRC)	(69.23, 30.76)	29	5
	(61.53, 38.47)	32	5
	(76.92, 23.08)	–	–
	(53.84, 46.16)	42	5
	50/50	43	5
<i>Memctrl</i> (CFG-LUT-MUX)	(62.5, 37.5)	33	3
	(68, 32)	39	3
	(56.25, 43.75)	35	3
	50/50	33	3
<i>AES-T2300</i> (VRC)	(64.28, 35.72)	33	5
	50/50	34	4
	(57.14, 42.86)	33	5
	(71.42, 28.58)	34	5
	(42.85, 57.15)	47	3
<i>AES-T2300</i> (CFG-LUT-MUX)	(64.28, 35.72)	113	2
	(59.09, 40.91)	116	2
	(66.66, 33.34)	–	–
	50/50	–	–

Table 7 compares the hardware resource utilization of the proposed framework with the VRC-based approach reported in [13] for the *Memctrl/T100* benchmark. In the method presented in [13], only selected portions of the circuit are implemented on the VRC architecture and periodically reconfigured at runtime using a conventional GA. However, the recovery process relies solely on evolutionary reconfiguration and does not incorporate an explicit mechanism for HT detection or temporary compensation of corrupted outputs during the recovery phase. In contrast, the proposed framework extends the VRC-based architecture by integrating several additional security and optimization mechanisms. Specifically, it incorporates a dedicated comparison module for HT detection, an attack-compensation unit that preserves system output correctness during runtime recovery, and an enhanced GA that employs multiple optimization strategies, including population seeding, a multi-stage adaptive mutation scheme, a fitness-biased crossover operator, and a hybrid selection strategy. Despite these additional security and optimization mechanisms, the proposed framework maintains efficient hardware resource utilization. The results demonstrate an approximately 8.63% reduction in LUT utilization compared with the implementation in [13]

for the *Memctrl-T100* benchmark. Furthermore, under the same number of PEs, the proposed GA achieves approximately 53.12% faster convergence. This improvement is primarily attributed to the enhanced evolutionary operators, which provide a more effective balance between exploration and exploitation and enable the GA to identify valid configurations within fewer generations. Consequently, the proposed framework not only strengthens the security and resilience of the reconfigurable system but also improves the efficiency of the runtime recovery process.

Table 7 Comparison of resource utilization for different architectures in the *Memctrl-T100* benchmark.

Architecture	Parameter	Usage
Proposed Architecture (T100-VRC)	Register	1,000
	LUT	1,080
	Generation	15
Proposed Architecture (T100-CFG-LUT-MUX)	Register	1,000
	LUT	1,073
	Generation	17
Article [13]	Register	990
	LUT	1,182
	Generation	32

6 Conclusion

This paper presents a novel protection framework for mitigating Hardware Trojans based on two reconfigurable architectures, namely VRC and CFG-LUT-with-MUX, integrated with an enhanced GA. The proposed approach enables fast and reliable recovery through dynamic reconfiguration, ensuring correct system operation even in the presence of malicious modifications. By mapping security-critical modules onto both architectures, the proposed framework effectively replaces compromised configurations with valid ones while maintaining uninterrupted system functionality during the recovery process through the Compensation Module.

Furthermore, the enhanced CFG-LUT architecture with multiplexer-based input selection improves reconfigurability and robustness against complex Hardware Trojan attack scenarios. Experimental results demonstrate that the proposed framework reduces the number of generations required for GA convergence by an average of **53.12%**, while introducing only a limited LUT overhead of **2.46%** for the VRC architecture and **1.8%** for the CFG-LUT-with-MUX architecture. The substantial reduction in evolutionary search effort, together with the low hardware overhead, indicates that the proposed approach can achieve more efficient runtime recovery while preserving the resource efficiency of the underlying reconfigurable architectures. These findings highlight the effectiveness of integrating hardware Trojan de-

tection, output compensation, and architecture-aware evolutionary reconfiguration within a unified runtime framework, providing a practical approach to improving the resilience of FPGA-based systems against hardware Trojan-induced functional disruptions.

Funding

No funding was received for conducting this research or for the preparation of this manuscript.

References

1. Babu, P., Parthasarathy, E., Reconfigurable FPGA architectures: A survey and applications, *Journal of The Institution of Engineers (India): Series B*, 102 (2021) 143–156.
2. Bhunia, S., Hsiao, M.S., Banga, M., Narasimhan, S., Hardware Trojan attacks: Threat analysis and countermeasures, *Proceedings of the IEEE*, 102 (8) (2014) 1229–1247.
3. Defense Science Board Task Force, High Performance Microchip Supply, Defense Technical Information Center (DTIC), Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, USA, 2005.
4. Naveenkumar, R., Sivamangai, N.M., Hardware Trojans Detection and Prevention Techniques Review, *Wireless Personal Communications*, 136 (2) (2024) 1147–1182.
5. Xiao, J., Shi, Z., Yang, X., Lou, J., BM-RCGL: Benchmarking approach for localization of reliability-critical gates in combinational logic blocks, *IEEE Transactions on Computers*, 71 (5) (2021) 1063–1076.
6. Tehranipoor, M., Salmani, H., Zhang, X., Design Vulnerability Analysis, in: *Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection*, 2014, pp. 125–145.
7. Sunkavilli, S., Zhang, Z., Yu, Q., New security threats on FPGAs: From FPGA design tools perspective, in: *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2021, pp. 278–283.
8. Ahmed, Q.A., Wiersema, T., Platzner, M., Post-configuration Activation of Hardware Trojans in FPGAs, *Journal of Hardware and Systems Security*, 2024, pp. 1–15.
9. Wang, J., Lee, C.-H., Virtual reconfigurable architecture for evolving combinational logic circuits, *Journal of Central South University*, 21 (5) (2014) 1862–1870.
10. Li, Z., Wang, J., Huang, Z., Luo, N., Wang, Q., Towards trust hardware deployment of edge computing: Mitigation of hardware Trojans based on evolvable hardware, *Applied Sciences*, 12 (13) (2022) 6601.

11. Keymeulen, D., et al., Fault-tolerant evolvable hardware using field-programmable transistor arrays, *IEEE Transactions on Reliability*, 49 (3) (2002) 305–316.
12. Chandra, K., Jagtap, A.P., Srivastava, S., Evolvable hardware state of the art, in: *Proceedings of the International Conference on Computing and Communication Systems: I3CS 2020, NEHU, Shillong, India*, Springer Singapore, Singapore, 2021.
13. Labafniya, M., et al., On the feasibility of using evolvable hardware for hardware Trojan detection and prevention, *Applied Soft Computing*, 91 (2020) 106247.
14. Sekanina, L., Friedl, Š., An evolvable combinational unit for FPGAs, *Computing and Informatics*, 23 (5–6) (2004) 461–486.
15. Chakraborty, S., et al., Towards the Detection of Hardware Trojans with Cost Effective Test Vectors using Genetic Algorithm, *Journal of Electronic Testing*, 40 (3) (2024) 371–385.
16. Roy, D.B., Schaumont, P., Narasimhan, S., The conflicted usage of RLUTs for security-critical applications on FPGA, *Journal of Hardware and Systems Security*, 2 (2) (2018) 162–178.
17. Kumm, M., Möller, K., Zipf, P., Dynamically reconfigurable FIR filter architectures with fast reconfiguration, in: *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, IEEE, 2013, pp. 1–8.
18. Saha, D., Yahyaie, K., Saha, S.K., Tehranipoor, M., Farahmandi, F., Empowering hardware security with LLM: The development of a vulnerable hardware database, in: *Proceedings of the 2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, May 2024, pp. 233–243.
19. Saha, D., Tarek, S., Yahyaie, K., Saha, S.K., Zhou, J., Tehranipoor, M., Farahmandi, F., LLM for SoC security: A paradigm shift, *IEEE Access*, 2024.
20. Long, X., Zhang, Y., Li, H., Wang, J., Chen, Y., VerilogLAVD: LLM-Aided Rule Generation for Vulnerability Detection in Verilog, *arXiv preprint arXiv:2508.13092*, 2025.
21. Kokolakis, G., Moschos, A., Keromytis, A.D., Harnessing the Power of General-Purpose LLMs in Hardware Trojan Design, in: *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*, Springer Nature Switzerland, Cham, 2024.
22. Agrawal, D., et al., Trojan detection using IC fingerprinting, in: *2007 IEEE Symposium on Security and Privacy (SP'07)*, IEEE, 2007, pp. 296–310.
23. Malhotra, G., Duraiswamy, P., Kishore, J.K., Self-repair embryonic architecture with GA evolved configuration data, in: *International Conference on Soft Computing and Pattern Recognition*, Springer Nature Switzerland, Cham, 2022, pp. 1–12.
24. Deepanjali, S., Noor Mohammad Sk., Self-healing controllers to mitigate SEU in the control path of FPGA-based systems: A complete intrinsic evolutionary approach, *Journal of Electronic Testing*, 38 (5) (2022) 547–565.
25. Wang, J., Kang, J., Hou, G., Real-time fault repair scheme based on improved genetic algorithm.
26. Jian, G., Mengfei, Y., Evolutionary fault tolerance method based on virtual reconfigurable circuit with neural network architecture, *IEEE Transactions on Evolutionary Computation*, 22 (6) (2017) 949–960.
27. Chandrasekharan, R., Rani, S.P.J.V., Performance analysis of intrinsic embedded evolvable hardware using memetic and genetic algorithms, *International Journal of Bio-Inspired Computation*, 15 (1) (2020) 43–51.
28. Sakali, R.K., Sk, N.M., Fault-tolerant multiplier using self-healing technique, *Microelectronics Reliability*, 160 (2024) 115458.
29. Salvador, R., Ruben, R., et al., Self-reconfigurable evolvable hardware system for adaptive image processing, *IEEE Transactions on Computers*, 62 (8) (2013) 1481–1493.
30. Roy, D.B., Bhasin, S., Guilley, S., Danger, J.L., Mukhopadhyay, D., Ngo, X.T., Najm, Z., Reconfigurable LUT: A double edged sword for security-critical applications, in: *Security, Privacy, and Applied Cryptography Engineering: 5th International Conference, SPACE 2015, Jaipur, India, October 3–7, 2015, Proceedings*, Springer, 2015, pp. 248–268.
31. Palchaudhuri, A., Dhar, A.S., Testable architecture design for programmable cellular automata on FPGA using run-time dynamically reconfigurable look-up tables, *Journal of Electronic Testing*, 36 (4) (2020) 519–536.
32. Sasdrich, P., Moradi, A., Mischke, O., Güneysu, T., Achieving side-channel protection with dynamic logic reconfiguration on modern FPGAs, in: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2015, pp. 130–136.
33. Labafniya, M., Borujeni, S.E., An Obfuscation Method Based on CFGLUTs for Security of FPGAs, *ISecure*, 13 (2) (2021).
34. Labafniya, M., Borujeni, S.E., Mentens, N., Evolvable hardware architectures on FPGA for side-channel security, in: *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings*, Springer, 2020.
35. Skubich, C., Reichel, P., Reichenbach, M., Increasing the Robustness of TERO-TRNGs Against Process Variation, *ACM Transactions on Reconfigurable Technology and Systems*, 16 (3) (2023) 1–29.

- 36. Kutzner, S., Poschmann, A., Stöttinger, M., TROJANUS: An ultra-lightweight side-channel leakage generator for FPGAs, in: *2013 International Conference on Field-Programmable Technology (FPT)*, IEEE, 2013.
- 37. Ng, X.T., et al., Integrated sensor: A backdoor for hardware Trojan insertions?, in: *2015 Euromicro Conference on Digital System Design*, IEEE, 2015.
- 38. Razali, N.M., Geraghty, J., Genetic algorithm performance with different selection strategies in solving TSP, in: *Proceedings of the World Congress on Engineering (WCE)*, vol. 2, Hong Kong, China, 2011, pp. 1–6.