SHADOWFI: An Open-source Framework For Fault Evaluation of Complex IC Designs Using Hyperscale Computing

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 November 2025, accepted 2 December 2025, date of publication 8 December 2025,
date of current version 18 December 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3641762

RESEARCH ARTICLE

SHADOWFI: An Open-Source Framework for Fault Evaluation of Complex IC Designs Using Hyperscale Computing

JUAN-DAVID GUERRERO-BALAGUERA[1], (Member, IEEE), ROBERT LIMAS-SIERRA[1], (Member, IEEE), OGUZ SENSOZ[1], JOSIE E. RODRIGUEZ CONDIA[1], (Member, IEEE), MAYNOR GIOVANNI BALLINA ESCOBAR[2,3], (Student Member, IEEE), MARIA LIZ CRESPO[2], SERGIO CARRATO[3], AND MATTEO SONZA REORDA[1], (Fellow, IEEE)

[1]Department of Control and Computer Engineering, Politecnico di Torino (DAUIN), Turin, Italy
[2]MLab, STI Unit, The Abdus Salam International Centre for Theoretical Physics (ICTP), Trieste, Italy
[3]Dipartimento di Ingegneria e Architettura (DIA), Universita degli Studi di Trieste (UNITS), Trieste, Italy

Corresponding author: Juan-David Guerrero-Balaguera (juan.guerrero@polito.it)

This work was supported in part by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing; and in part by ISCRA for the LEONARDO Supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA, Italy.

ABSTRACT Cutting-edge integrated circuits (ICs) are increasingly vulnerable to hardware faults, which can jeopardize the overall reliability of the system. Hence, it is crucial to evaluate the impact of faults to identify hardware vulnerabilities that later designers can use to devise fault-mitigation solutions during the circuit design stages. Fault injection (FI) through in-circuit emulation tackles the inherent complexity of fault evaluations by adopting FPGA emulation strategies. Although various works report frameworks using this FI approach, most of them are technology-dependent and hardly scalable when dealing with the increasing complexity of modern IC architectures. In addition, none of these frameworks is disclosed, which limits the adoption of fault-emulation strategies due to the inherent complexity and the required time to set up a functional FI environment. This work introduces SHADOWFI, a generic, open-source, and netlist-based fault-emulation framework that leverages the computational capabilities of hyperscale infrastructures for fault characterization and reliability estimation of complex IC designs. SHADOWFI offers two different functional workflows: *i)* simulation, which enables the parallelization of FI tasks on high-performance computing systems, and *ii)* emulation, which leverages the flexibility of FPGA cluster implementations. The framework automates saboteur insertion, FI campaign execution, and report generation, requiring minimal user configuration. Each SHADOWFI workflow was evaluated on a set of IC design benchmarks, demonstrating practical usability and significant speedup in fault injection. SHADOWFI is publicly available at https://github.com/divadnauj-GB/SHADOWFI.git

INDEX TERMS Hardware faults, fault injection, fault simulation, fault emulation, reliability, dependability, FPGA, hyperscale infrastructure.

I. INTRODUCTION

Cutting Edge silicon technologies have evolved significantly by aggressively scaling down the transistor sizes, promoting the delivery of high transistor densities, lower energy consumption, and significant computational capabilities. Today,

The associate editor coordinating the review of this manuscript and approving it for publication was Paolo Crippa.

many ICs exhibit remarkable complexity, incorporating multiple hardware structures and accelerators that range from general-purpose to application-specific architectures [1], [2]. Unfortunately, the outstanding capabilities of modern silicon technologies can be overshadowed due to serious reliability concerns. Recent studies have demonstrated that at small-scale integrations (i.e., 7 nm and below), the ICs can be significantly affected by faults (permanent or transient) either escaping the end-of-manufacturing test process, or arising during their operational life [3], [4], [5]. These faults may result from various phenomena, including process variation, manufacturing defects (e.g., test escapes), premature degradation, or aging, as well as sensitivity to temperature variations, and radiation effects [5], [6]. Moreover, hardware faults represent one of the primary sources of *Silent Data Errors* (SDEs) affecting modern silicon devices, posing a significant reliability challenge to the semiconductor industry today [7], [8], [9], [10]. In fact, the SDEs due to faults have become a serious reliability concern in datacenters (e.g., Google, Meta, Amazon [8], [9], [11], [12], [13]), where a defective integrated circuit leads to workload errors or malfunctions, reducing the quality of services (QoS), and in the end causing significant economic impact. Similarly, faults affecting integrated circuits can produce catastrophic results, either threatening human lives (e.g., in automotive, healthcare, or aerospace applications) or causing financial issues when they impact banking transactions. Consequently, it is crucial to characterize the fault effects on a given integrated circuit to devise effective countermeasures later.

In this scenario, Fault Injection (FI) is an essential experimental mechanism used to determine the impact of faults on microelectronic devices, facilitating the verification of functional safety and testing targets, as well as contributing to the fault characterization and identification of vulnerable structures and their subsequent mitigation of critical hardware vulnerabilities [14], [15], [16]. Moreover, fault injections applied at the early design stages (i.e, before the final chip manufacturing) play a crucial role in developing more robust and reliable IC devices, especially when used in mission-critical applications (e.g., automotive and aerospace) [15]. In this regard, it is crucial to have suitable tools and frameworks that enable the fast and accurate assessment of reliability with respect to faults affecting silicon devices.

Several FI approaches can be used to accurately evaluate the fault tolerance of modern ICs, such as software-based simulation or hardware-based emulation [15]. The software-based simulation—mainly based on logic simulation, leverages the software's flexibility to model and control different fault behaviors at the register-transfer or gate-level abstraction. However, this FI approach faces severe execution-time limitations that become prohibitive as the circuit complexity increases (e.g., months or years) [17], [18], [19]. Alternatively, hardware-based emulation overcomes simulation restrictions by inserting saboteur circuits within the integrated circuit under evaluation. This strategy, also known as in-circuit fault insertion, leverages reconfigurable hardware devices, such as Field Programmable Gate Arrays (FPGAs), thereby making accurate and faster assessments possible under realistic conditions [15], [20], [21].

To date, several works have presented various FPGA-based emulation frameworks, which combine specialized software and hardware tools to model and implement diverse fault behaviors [20], [21], [22], [23]. In-circuit instrumentation techniques have been widely adopted as a fault emulation approach for reliability assessment of IC designs [15], [20], [21]. However, most of the reported frameworks apply circuit instrumentation using vendor-dependent tools, preventing their use because they either are not freely available or already deprecated. In addition, the number of faults on an IC can increase substantially (i.e., millions of faults) when evaluating complex hardware designs and accelerators (e.g., AI hardware). Despite adopting statistical sampling methods to reduce the number of faults for analysis, maintaining representative confidence levels can still lead to a considerable amount of faults to be evaluated one at a time (e.g., thousands of faults), which nevertheless require prolonged evaluation times, even when relying on FPGA devices [24]. Therefore, it is crucial to adopt alternative or complementary paradigms that aim to reduce the duration of fault evaluations.

Currently, several initiatives in both industry and academic contexts [25] are being developed to create FPGA-based hyperscale infrastructures, aiming to provide energy-efficient and high-performance computing capabilities not offered by other high-performance systems. Such reconfigurable systems are promising platforms for deploying and accelerating fault emulation frameworks used in the reliability assessment of IC designs.

This work introduces SHADOWFI, a generic, open-source fault-emulation framework that, for the first time, leverages the parallelism capabilities offered by hyperscale infrastructures to accelerate the reliability estimation and fault characterization of complex microelectronic devices. SHADOWFI leverages the distributed computing paradigm to deploy parallel FI tasks using simulation and emulation workflows. In the first case, SHADOWFI provides support for deployment on high-performance computing (HPC) systems by distributing fault injections across multiple computational nodes in parallel. On the other hand, by employing FPGA-cluster infrastructures, SHADOWFI can be configured to emulate fault behaviors within the hardware structures of the targeted IC design, which is then deployed on independent FPGA nodes, enabling multiple FI tasks in parallel.

SHADOWFI was conceived as an open-source framework that facilitates the adoption of a netlist-based fault emulation strategy, addressing the main limitations of state-of-the-art emulation-based frameworks. Firstly, these fault-emulation frameworks primarily rely on commercial tools that are

typically tied to specific vendors or devices, which limits their scalability, flexibility, and portability. Additionally, most of them are private or have restricted access, hindering their usability in both academic and non-academic contexts, and potentially leading to obsolescence.

The key contributions of this paper can be summarized as follows:

- A generic and open-source FI framework for reliability estimation of complex microelectronic devices (SHADOWFI). The proposed framework supports the evaluation of permanent and transient faults, including stuck-at, single-event transient (SET), single-event upset (SEU), and multiple-event upset (MEU).
- A modular and flexible scan-chain-like saboteur architecture that combines RTL and netlist abstractions, offering flexible fault insertion across different hierarchical levels of the IC design, providing a simple Fault Injection Port (FIP) interface.
- An automated pipeline for the insertion of saboteur circuits in a target component's netlist of any IC design. The framework provides an automated end-to-end process that encompasses hardware description language (HDL) read and analysis, fault instrumentation, fault orchestration, results gathering, and report generation with minimal user intervention.
- A FI simulation workflow with support for high-performance computing deployment. This workflow can automatically orchestrate fault injections through parallel logic simulations.
- An FPGA-based fault emulation workflow with support for hyperscale infrastructures, leveraging the distributed computing strategies where the IC under evaluation is instrumented with saboteur circuits and deployed on multiple FPGA devices, enabling the evaluation of multiple faults in parallel at silicon speed. In particular, this workflow was deployed on the HyperFPGA system hosted by the ICTP; however, it can be extended to other FPGA cluster infrastructures.
- SHADOWFI was evaluated on a selection of benchmarks, demonstrating the framework's flexibility and effectiveness across different parallel configurations, providing relevant results while significantly speeding up the FI tasks.

The paper is structured as follows: Section II outlines related works regarding hardware-based emulation approaches. Section III introduces the proposed FI mechanism. Section IV describes the framework evaluation and reports the experimental results. Finally, Section V concludes the paper and describes future directions.

II. BACKGROUND AND RELATED WORKS

A. HARDWARE FAULTS AND RELIABILITY

The progress in CMOS technology has facilitated the development of smaller, faster, and more energy-efficient electronic devices such as Graphic Processing Units (GPUs), CPUs, and Domain-Specific-Architectures (DSA) hardware accelerators, which are utilized across various domains ranging from multimedia to AI applications. These improvements are largely due to the increased transistor densities on individual chips, in line with Moore's Law [5]. However, the ongoing miniaturization of these technologies—especially at 7nm and below—has raised concerns about reliability, as smaller components become more vulnerable to faults stemming from factors like aging, over-stress, harsh environmental conditions with high temperatures, or manufacturing defects. In fact, scaling down technology significantly raises the likelihood of hardware faults, including those caused by terrestrial radiation [4]. Numerous studies have shown that the failure rate increases along with technology scaling [1], [5], [26]. This rising failure rate in contemporary silicon technologies can considerably shorten device lifespans and compromise the reliability of many applications in use today.

Hardware faults occur due to physical phenomena that impact the electrical operation of an electronic circuit. They are generally classified as either permanent or transient, based on their characteristics and behavior during circuit operation.

Permanent faults arise from various causes, such as manufacturing defects, premature degradation, process variations, and semiconductor aging. These faults are persistent; once they occur and are activated in a circuit, they will consistently

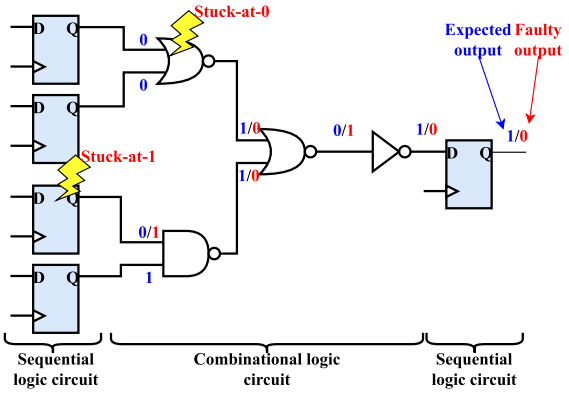


FIGURE 1. Representation of the stuck-at fault model for describing permanent faults.

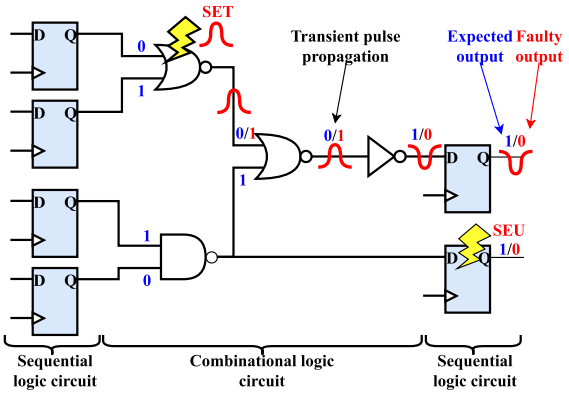


FIGURE 2. Illustration of Single-Event Transient (SET) and Single-Event Upset (SEU) faults on a basic circuit.

FIGURE 3. "Chain of treats" produced by faults on a computational system. A fault, when activated, generates errors. Errors propagate throughout the application or system operation, likely causing failures [29].

affect its operation. The stuck-at-fault model is the most commonly used method to describe and model permanent faults affecting individual signals in a circuit netlist [27], [28], [29]. This model forces certain signals to remain at fixed values of 0 or 1. Figure 1 illustrates a simple example of stuck-at faults within a basic combinational circuit.

On the other hand, transient faults correspond to temporary physical malfunctions caused by different phenomena such as electromagnetic interference, circuit degradation, manufacturing process variations, or due to the influence of external radiation effects (e.g., high-energy particle strikes). In a computer system, transient faults can instantaneously affect a single transistor or gate in the design that may corrupt pipeline states, leading to register and memory corruptions, pipeline deadlocks, or exceptions. These faults are commonly referred to as Single-event transients (SET) when a single transistor or gate is affected, and single-event upsets (SEUs) when a flip-flop or memory storage is flipped, and single-event multibit upsets (SEMUs or MBUs) when multiple gates or storage elements are impacted [30], [31], [32], [33]. Figure 2 depicts the effects of SEU and SETs on the behavior of a digital circuit. The SET fault model corresponds to a short pulse or glitch that can propagate through the circuit and potentially reach an output or be captured by a storage element; whereas a SEU corresponds to a bit-flip on storage elements or flip-flops. SEU effect remains until the flip-flop is overwritten.

When faults occur during the operation of a silicon device, they can lead to effects known as errors that propagate through the application execution, ultimately affecting the final system output. When these effects reach the final application, they can result in a failure. This fault propagation mechanism is referred to as the "chain of threats," as described in [29] and illustrated in Figure 3.

Specifically, when a fault appears in a computing device, it remains dormant until a set of functional or environmental conditions activate it, producing an error. These errors are often called Silent Data Errors (SDE) or Silent Data Corruption (SDC) [7], [10]. As the system continues to operate, these initial errors can transform into subsequent errors. In other words, errors caused by a fault in one component can propagate to other components, generating and transmitting additional internal errors. This chain of error propagation can ultimately reach the application's final output, leading to failures that jeopardize the entire system operation.

Faults that affect silicon devices pose a significant threat to the reliability and safety of advanced applications. Therefore, it is essential to develop more robust devices that can mitigate the impact of these faults. However, designing fault-tolerant circuits requires thorough fault assessments to identify the hardware vulnerabilities that need robustness. As hardware systems grow larger and more complex, conducting fault assessments becomes increasingly challenging. The rising number of potential faults can lead to extensive, and sometimes prohibitive, evaluation times when using traditional fault injection methods, mainly based on simulation strategies.

B. FAULT EMULATION FOR IC DESIGNS

Fault emulation is a fault injection strategy that relies on FPGA devices aiming to assess the impact of faults on IC designs. This fault injection technique enables high-speed and parallel execution, providing significantly faster fault evaluations, when compared to fault simulation approaches [14], [15]. In general, FPGA-based fault emulation approaches can be classified into reconfiguration-based [22], [23] and in-circuit instrumentation-based [20], [21], [24].

The reconfiguration-based approaches typically leverage the FPGA's dynamic reconfiguration properties and the bit-stream modification to inject faults [22], [23], [37], [38], [39], [40], [41], [42]. In this approach, for every fault, it is required to recompile and download the new bit-stream into the FPGA device. Although this strategy has limited or negligible hardware overhead requirements, it incurs significant time overhead during the preparation phase (i.e., hardware synthesis and FPGA reconfiguration), which in turn can take longer than the fault evaluation itself on the FPGA device. Furthermore, the technological dependency, the fault models (i.e., SEU only), and the evaluation scope (i.e., FPGA reliability), among other aspects, significantly constrain the straightforward adoption of this fault evaluation approach when considering different target configurations and fault models or deployment infrastructures.

On the other hand, the in-circuit instrumentation approach relies on special hardware structures, typically sabotage-based, inserted inside the circuit under evaluation [20], [21], [24], [36], [43]. The saboteur circuits can be designed to incorporate several fault models, which in turn, can be selected and activated during the runtime operation of the IC on an FPGA. In general, the in-circuit instrumentation can be conducted at RTL or netlist levels of abstraction.

The RTL instrumentation resorts to source code modification, inserting and interconnecting one or multiple FI components inside the design hierarchy [43], [44], [45], [46], [47], [48]. Unfortunately, this strategy can have limitations when the target IC design incorporates complex HDL descriptions (i.e, behavioral and parametric hardware descriptions), requiring significant customization for every IC design. Alternatively, the netlist instrumentation inserts saboteur circuits into the gate-level IC's netlist, which is obtained from the synthesis stage. In particular, this FI

TABLE 1. Summary of related works adopting fault emulation frameworks leveraging the netlist-based in-circuit instrumentation.

Work	Year	FI Level	FI Target	Fault Model	Netlist Gen. Tool	Simulation Support	Impl. Type	Current Status	
								OS(*)	UTD(**)
[24]	2008	Netlist	ASIC-Gtech	Stuck-at/delay/SEU	Design Compiler	✗	Standalone	✔	✗
[34]	2013	Netlist	FPGA-DFF	SET/SEU	Synplify Pro	✗	Standalone	✗	-
[35]	2014	Netlist	ASIC-Gtech-DFF	Stuck-at/SEU	-	✗	Standalone	✗	-
[36]	2015	Netlist	ASIC-Gtech	Stuck-at/delay/SEU	Synplify Pro	✗	Standalone	✗	-
[20]	2022	Netlist	ASIC-Gtech-DFF	Stuck-at/bit-flip	Vivado	✗	Standalone	✗	-
[21]	2025	Netlist	FPGA-Cells	Stuck-at/SEU/MEU	Vivado	✗	Standalone	✗	✔
[This Work]	2025	Netlist + RTL	ASIC-Gtech	Stuck-at/SET/SEU/MEU	Yosys	✔	- Hyperscale - Standalone	✔	✔

(*) Open-Source and freely available, (**) Up-to-date, DFF: D flip-flop, SEU: Single-Event-Upset, MEU: Multibit-Event-Upset, SET: Single-Event-Transient.

approach can be implemented by using cell replacement or net splitting methods. In the first case, all cells in the design are replaced by customized cells with FI capabilities [35], while in the second case, all nets in the design are opened and connected to saboteur structures. The latter method is more flexible and scalable, becoming widely adopted in most state-of-the-art emulation-based frameworks [20], [21], [24], [34], [36].

Table 1 reports the most recent state-of-the-art emulation-based frameworks adopting netlist instrumentation strategies by considering eight aspects of comparison: FI level, FI target, fault models, netlist generation tool, simulation support, type of deployment, and current status. It is worth noting that prior fault emulation frameworks share common characteristics regarding the target of fault injections, fault models, and hardware implementations. However, other aspects, such as availability and technology dependency, can hinder their adoption and portability. For instance, fault emulation frameworks are either proprietary or not openly disclosed, imposing significant restrictions on their use or potential extensions; likewise, they rely on Synopsys or AMD/Xilinx tools for netlist generation (e.g., design compiler, Synplify Pro, or Vivado), which significantly limits portability when adopting other FPGA devices, vendors, or any other FPGA-based infrastructure.

In general, netlist-only fault emulation offers flexibility in modeling and evaluating various fault behaviors, regardless of the IC's architecture or micro-architectural details. Nonetheless, this fault instrumentation approach can lead to significant hardware overhead, which can ultimately constrain the maximum number of FI circuits that can be inserted into the netlist design. In order to solve this problem, several works have proposed solutions by instrumenting only the flip-flop cells [20], [34], [35], while other works apply statistical sampling approaches across the flattened version of the design's netlist [24], [36].

Despite applying any of these fault sampling solutions, in the end, the required number of faults to be evaluated can still be significantly large. For example, as the IC designs grow in size, the number of faults to be inserted in the netlist also increases, especially when adopting statistical sampling, where maintaining proper confidence intervals is crucial for guaranteeing the statistical relevance of the fault evaluations [49]. Consequently, evaluating complex hardware designs and accelerators using netlist-based fault emulation results in a large number of faults to be evaluated, leading to a significant evaluation time, even when relying on FPGA devices. Hence, it is crucial to embrace alternative or complementary paradigms (e.g., distributed computing) aimed at speeding up the reliability assessment of digital circuits when using netlist-based instrumentation strategies.

In fact, the significant advances in reconfigurable computing have nowadays promoted the creation of FPGA clusters in both academic and industrial contexts [50]. The parallelism and reconfiguration properties offered by such systems meet the computational requirements demanded by emulation-based strategies, especially during the execution of fault injection tasks. Indeed, commercial FPGA cloud services, such as Amazon AWS FC1 and FC2[1] or Microsoft Catapult,[2] offer different infrastructures and software tools for deploying advanced reconfigurable computing applications on various FPGA platforms. Likewise, various academic initiatives, such as the Open Cloud TestBed (OCT) [25], cloudFPGA from IBM,[3] and HyperFPGA [51], [52] also provide novel hardware and software infrastructure for leveraging reconfigurable computing in hyperscale systems with FPGAs.

Although some efforts have been made to accelerate fault simulation tasks using HPC systems [53], to the best of our knowledge, there are no reported works leveraging the capabilities of hyperscale FPGA infrastructures for the deployment and acceleration of fault-emulation paradigms. Consequently, SHADOWFI is the first open-source framework that leverages the use of hyperscale infrastructures to speed up the fault characterization and reliability evaluation of IC designs using fault emulation strategies. In particular, SHADOWFI has been designed to support both HPC and FPGA-based computing clusters. This combination enables broader adoption and possibly longer lifespan compared to prior works. Unlike many works in the literature,

[1] https://aws.amazon.com/ec2/instance-types/f2/
[2] https://www.microsoft.com/en-us/research/project/project-catapult/
[3] https://research.ibm.com/projects/cloudfpga

SHADOWFI relies on the open-source Yosys framework [54] for the HDL analysis and netlist generation, which is fundamental for the fault injection infrastructure. In addition, SHADOWFI provides simulation support for both fault insertion verification and fault injection orchestration, utilizing logic simulations via Verilator [55].

SHADOWFI inserts saboteur circuits into any IC design under evaluation by combining RTL and netlist abstractions. First, the saboteur circuits are placed into the netlist of the selected components of the design. Then, at the RTL level, all saboteurs are interconnected across the design hierarchy using a scan-chain-like structure, exposing a simple interface, namely, the Fault Injection Port (FIP). Finally, the new fault-instrumented design is inserted in a testbed containing the IC test benchmark and a fault injection manager.

SHADOWFI provides an automated pipeline for conducting fault instrumentation, fault injection orchestration, results gathering, and report generation. This operational pipeline is implemented through two functional workflows that can be deployed on standalone or hyperscale infrastructures. The simulation workflow supports the deployment of fault injection tasks using HPC infrastructures. On the other hand, the emulation workflow supports the deployment of fault injection tasks on FPGA-based infrastructures. Although this work leverages the flexibility offered by HyperFPGA [51], [52], SHADOWFI has been envisioned to be extensible to any other FPGA-cluster infrastructure after a proper framework configuration.

Although, several recent works propose alternative AI-driven approaches to fault detection and robustness enhancement, such as graph-based transfer learning for analog circuits [56], SEU-resilient latch design [57], triple-node-upset-tolerant latches [58], and cost-optimized latch designs with aging mitigation [59]. These methods offer predictive or design-time robustness enhancements, but often provide limited assessment when considering multi-cycle fault propagation. SHADOWFI takes a complementary perspective by leveraging HPC and FPGA-based hyperscale infrastructures to accelerate detailed fault simulation and emulation, thereby enabling a closer analysis of hardware behavior under realistic fault scenarios.

III. SHADOWFI ARCHITECTURE AND DESCRIPTION

The SHADOWFI framework operates in two main stages: *i)* design preparation and *ii)* fault evaluation.

During the design preparation stage, a FI infrastructure is integrated into the circuit under test (CUT) using netlist instrumentation strategies, through a saboteur insertion flow.

In the fault evaluation stage, a FI campaign is conducted leveraging the fault instrumentation applied to the CUT. The fault injection is performed in parallel, creating multiple FI instances, where one fault is injected at a time through the FI infrastructure of each instance. The effects of the faults are assessed by comparing the outcomes of the CUT under faults with the outcomes observed in a fault-free scenario.

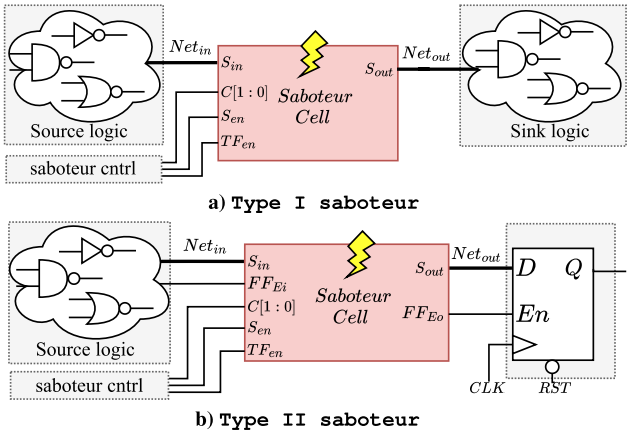


FIGURE 4. General representation of the saboteur architectures in SHADOWFI.

The following subsections provide a detailed description of each of the main elements incorporated by SHADOWFI.

A. FAULT INJECTION INFRASTRUCTURE

SHADOWFI adopts in-circuit instrumentation through netlist modification to emulate both permanent and transient fault behaviors at the gate-level netlist of a target IC under evaluation. The fault insertion is modeled by saboteur circuits, which integrate the functional behavior of different faults. These saboteur circuits are inserted into all target nets of the circuit, adding corruption logic based on the chosen fault model. We present five fault models, as described below; however, the framework can be extended by including additional fault models.

- *Stuck-at-0*: The target net is set to 0 after the fault activation
- *Stuck-at-1*: The target net is set to 1 after the fault activation
- *Single event transient (SET)*: The target net flips its value during one clock cycle
- *Single event upset (SEU)*: The target flip-flop changes its value after the fault activation
- *Multibit event upset (MEU)*: Multiple adjacent flip-flops change their values after the fault activation

The implementation of these fault models uses two saboteur circuits labeled as `Type I` and `Type II` saboteurs. The first one implements the stuck-at and SET fault models at any cell outputs of the design, whereas the second type is specially tailored to implement SEU on flip-flop components only. Figure 4 depicts the block diagram associated with each saboteur type. The saboteur circuits split the nets' interconnection between source and sink logic cells. Thus, the saboteur circuit receives as an input the Net_{in} value coming from the source logic and generates an output Net_{out} according to the fault model configuration, which is sent to the sink logic. The configuration of the `Type I` saboteur comprises four control signals, as depicted in Figure 4.(a). The C_0 and C_1 signals define the fault model behavior, while

TABLE 2. Functional operation of saboteurs `type I` & `II` in terms of the configuration and control inputs.

Saboteur type	Fault model	Saboteur cntrl			Saboteur out	
		TF_{en}&S_{en}	C_1	C_0	Net_{out}	FF_{Eo}
Type I	Stuck-at-0	1	0	0	0	-
	Stuck-at-1	1	0	1	1	-
	SET	1	1	dc	$\overline{Net_{in}}$	-
	No fault	0	dc	dc	Net_{in}	-
Type II	No fault	0	-	-	Net_{in}	FF_{Ei}
	SEU	1	-	-	$\overline{Net_{in}}$	1

dc : Don't care; - : Not used

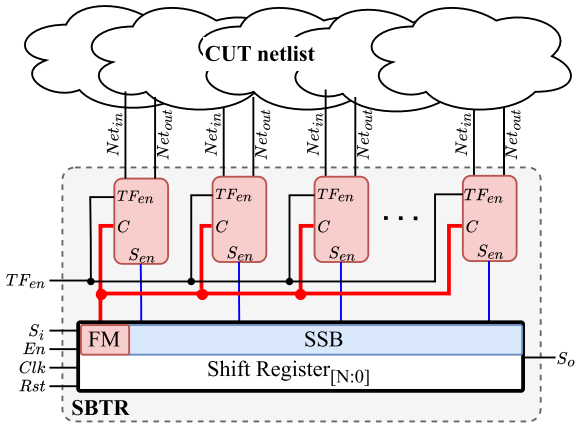


FIGURE 5. Illustration of the general architecture of a saboteur infrastructure (SBTR) composed of multiple saboteur circuits.

the S_{en} and TF_{en} correspond to the saboteur enable and the fault activation enable, respectively.

On the other hand, the `Type II` saboteur manipulates both the data (D) and enable (EN) signals of the flip-flop to induce a SEU fault, as shown in Figure 4.(b). Specifically, this saboteur causes a bit-flip at the flip-flop's input while simultaneously forcing it to load the erroneous value. The operation of the saboteur is controlled solely through the S_{en} and TF_{en} ports.

These saboteur circuits remain inactive until triggered by external control. Table 2 presents the configuration details for both saboteur circuits according to every fault model. It shows the saboteur configurations necessary to select and activate the saboteur operation for every fault model. Fault behaviors are effectively introduced on the circuit operation only when the saboteur selection and fault activation signals are active (i.e., $S_{en} = 1$ and $TF_{en} = 1$); otherwise, the saboteur does not produce any effect on the CUT. In the case of saboteur `Type I`, the additional control ports (i.e, C_1 and C_0) are used to select one specific fault model, either Stuck-at-0, Stuck-at-1, or SET. For example, when $C_1 = 0$ and $C_0 = 0$, the saboteur behaves as a Stuck-at-0, forcing the net to a zero value. On the contrary, the `Type II` saboteur circuit only requires $S_{en} = 1$ and $TF_{en} = 1$ to inject a SEU on the target flip-flop.

It is important to note that the TF_{en} signal is used to control both the fault activation time and the duration of the fault effect. These two configuration parameters are measured in system clock cycles. The fault activation time refers to the specific clock cycle within the operating window of the CUT when the fault is activated. On the other hand, the fault duration indicates how long, in clock cycles, the fault remains active. For instance, a permanent fault can be activated right from the first operational clock cycle of the CUT and will remain active for the entire duration of its operation. In contrast, a transient fault, such as a SET or SEU, can be activated randomly at any clock cycle but typically lasts for only one clock cycle.

These fault activation mechanisms are implemented within a specialized interconnection and control system that groups multiple saboteurs together, allowing for their individual configuration and activation. Figure 5 depicts a general block diagram of the proposed saboteur infrastructure (SBTR) used to insert and control the saboteur circuits within the netlist of any CUT. Initially, the number of saboteur units is determined by the number of nets or flip-flops in the target design netlists, ensuring that each saboteur unit is connected to a unique net or flip-flop of the design. The saboteur insertion splits the assigned net into Net_{in} and Net_{out} signals. Then the saboteur logic is inserted between the newly created signals, making Net_{in} the saboteur's input and Net_{out} the saboteur's output, respectively.

As the number of saboteur units increases, managing them becomes increasingly complex, necessitating a straightforward yet effective control mechanism. The proposed solution involves the integration of a shift register (SR) within the SBTR to connect all saboteurs' control signals, resulting in a unified fault injection interface. The SR offers a simple method for selecting and activating individual saboteur units, regardless of the number present in the design. In detail, the SR is divided into two main components: *i)* the fault model (FM) bits, and *ii)* the saboteur selection bits (SSB). The FM bits correspond to the two most significant bits of the SR and are shared among all saboteur units, allowing for the selection of the fault model behavior. The remaining SR bits (SSB) are individually connected to each saboteur unit, enabling the specific selection of saboteurs for fault injection. Finally, the fault activation signal TF_{en} is shared among all saboteur units and exposed as an input port for external handling.

It is important to note that the proposed SBTR architecture allows for easy manipulation of any number of saboteur units through a simple control interface, in our case, the FIP, that consists of the fault activation port TF_{en} along with the SR control ports (S_i, En, Clk, Rst, and S_o). The ports S_i and S_o function as the serial input and serial output ports, respectively, and enable seamless scan-chain-like interconnection of multiple SBTR modules while maintaining the same FIP.

Although this work introduces independent saboteur circuits `type I` and `type II`, the modular saboteur

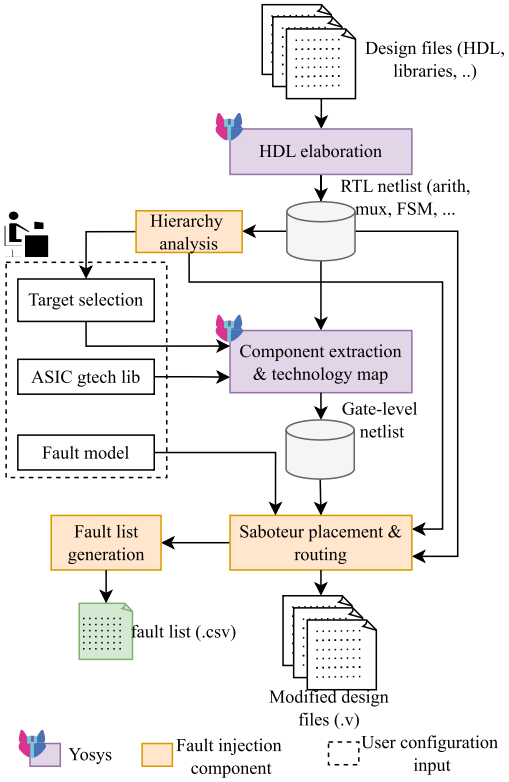


FIGURE 6. SHADOWFI saboteur insertion flow. This flow illustrates the automatic insertion of saboteur circuits on a given input HDL design.

architecture can be extended to represent additional classes of faults, including defect-related (e.g., bridging faults), soft errors, and even aging-related degradation. These models can coexist simultaneously within the same infrastructure, as each saboteur is parameterizable and independently activatable.

B. SABOTEUR INSERTION FLOW

SHADOWFI provides an automatic pipeline to insert the saboteur infrastructure into selected submodules of any CUT described in a hardware description language (HDL) (e.g., Verilog, SystemVerilog, or VHDL). The hardware instrumentation process performs several transformation steps on the input HDL design files of the CUT, producing new HDL design files that incorporate one or more SBTR modules. Figure 6 shows the detailed saboteur insertion flow implemented by SHADOWFI. In detail, the design transformation comprises five successive stages: *i)* HDL elaboration, *ii)* hierarchy analysis, *iii)* component extraction and technology mapping, *iv)* saboteur placement & routing, and *v)* fault list generation.

The HDL elaboration stage involves parsing the input HDL files to transform the complete CUT design into a RTL abstraction. This transformation preserves the original hierarchy of the CUT while simplifying various complex HDL abstractions. This stage is crucial because the HDL source files, in any CUT design, often contain different hardware description forms, such as parametric definitions, behavioral descriptions, and dynamic component instantiations, that

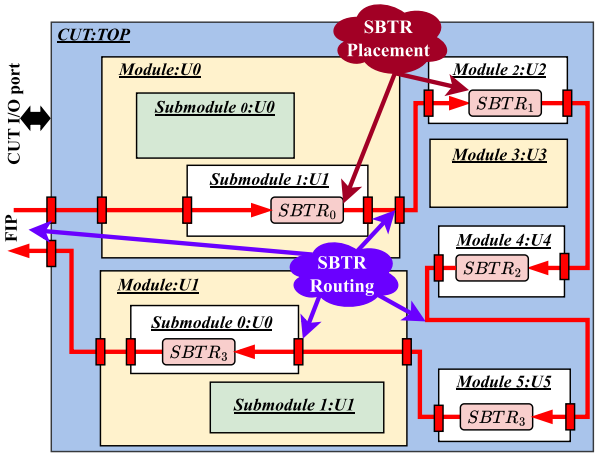


FIGURE 7. A general representation of the SBTR placement and routing. SHADOWFI automatically places the SBTR blocks across the design and routes them in a scan-chain-like architecture, simplifying their configuration.

make the insertion of saboteurs at these levels extremely complex.

Essentially, the HDL elaboration simplifies and transforms the input HDL design into a new representation that uses fundamental RTL components such as registers, adders, multipliers, multiplexers, memories, and finite state machines. SHADOWFI implements this stage using Yosys [54], an open-source synthesis tool that supports the conversion of full-featured Verilog/VHDL designs into simple RTL Verilog descriptions.

The hierarchy analysis stage processes the simplified RTL design generated in the previous stage and creates a hierarchical tree of the CUT's component interconnections. This hierarchical information is essential to select the parts or CUT's submodules to be instrumented with SBTR modules.

In the component extraction and technology mapping stage, several modules are extracted from the CUT and transformed into a gate-level netlist, preparing them for the insertion of saboteurs. The user determines the modules selected for extraction through three configuration options in SHADOWFI. The first option, known as the default configuration, randomly selects a single component instance from any level within the CUT hierarchy. The second option allows the user to specify the top entity in the hierarchy as the target component, which effectively collapses the entire design into a single CUT component. The third option is user-defined, enabling the test engineer to arbitrarily select specific component instances based on the hierarchy tree obtained in the previous stage.

At the same time, SHADOWFI generates the netlist of the extracted components using Yosys for synthesis. It is worth noting that the technology mapping process uses the internal Yosys cells. Nonetheless, the tool can be easily extended to be used with any other ASIC technology cell library by providing the appropriate liberty files. The outcome of this

process creates single Verilog files, where each of them contains the netlist of every extracted component.

Once the netlist is generated, the next step consists of inserting the saboteur circuits on every component of the hierarchy. SHADOWFI provides two fault instrumentation options: *full* or *statistical*. The full option inserts saboteur circuits exhaustively into every net inside the selected components within the IC design. The statistical option randomly selects a subset of nets where saboteurs are inserted. The number of sampled faults can be determined by the user. This feature enables the conduct of different statistical strategies depending on the user's needs. When selecting the statistical approach option, the framework by default uses fault sampling with a 95% of confidence level and 1% of error margin, aligning the established fault sampling strategies in the field [24], [49].

During the saboteur placement and routing stage, SHADOWFI inserts and interconnects SBTR modules into the components of the selected CUT. Figure 7 shows a general block diagram illustrating the insertion of the saboteur infrastructure targeting five modules within the CUT. The process begins by adding the SBTR modules to the netlists of the chosen components, following the SBTR architecture outlined in Figure 5. After modifying the components, the updated instances are placed within the RTL design of the CUT, replacing only the target components selected for fault injections. Once the new components are in place, the next step involves creating the interconnection infrastructure using a scan-chain-like architecture. In this setup, the output of one SBTR module is connected to the input of another SBTR module. This routing mechanism introduces additional FIPs and interconnection wires across different hierarchy levels. Creating a unique FIP at the top hierarchy component, which is utilized to manage the entire fault injection infrastructure.

Finally, at the last stage, SHADOWFI generates a file that provides detailed information about every possible fault to be injected into the fault injection infrastructure within the CUT. Each fault is defined by an eight-tuple descriptor as follows:

$<SBTR_{ID}$, $Path_{Inst}$, $Module_{name}$, Bit_{start}, Bit_{end}, Net_{ID}, $FaultModel$, $FaultAct_{time}>$

The $SBTR_{ID}$ indicates the position of the component in the SBTR chain where the fault will be injected. The $Path_{Inst}$ contains the full string path of the component instance within the CUT. The $Module_{name}$ specifies the name of the module in the design. The Bit_{start} and Bit_{end} define the positions of the components within the bit-string configuration chain. The Net_{ID} defines the index of a specific saboteur within the target component. The $FaultModel$ indicates the type of fault to be injected, and $FaultAct_{time}$ specifies the clock cycle when the fault is activated. For example, let's consider the fault descriptor <2, "TOP/U4", "Module4", 300, 365, 12, "S@1", 0> for the hypothetical fault injection infrastructure shown in Figure 7. The descriptor indicates a "Stuck-at-1" fault, activated from the first clock cycle, injected by the saboteur number 12 of the component "Module4". It corresponds to the $SBTR_2$, configured with the bits allocated between positions 300 and 365 of the saboteur infrastructure, within the instance "TOP/U4". The fault descriptor enables the creation of a unique bit string for each fault, allowing for the configuration of the fault injection infrastructure during fault injection campaigns.

C. FAULT EVALUATION WORKFLOW

This fault evaluation aims to estimate the impact produced by hardware faults on the CUT's operation. Therefore, the evaluation process consists of systematically injecting faults into the CUT using the inserted saboteur infrastructure. Afterward, a test application or a set of input stimuli is executed on the CUT, aiming to test the CUT's functionality in the presence of faults.

In detail, the fault evaluation procedure consists of two main steps. First, test stimuli are applied to the CUT under fault-free conditions. This process generates and stores reference outcomes, which are essential for the subsequent fault assessment. In the second step, known as fault injection, each fault is represented as a bit string that configures the saboteur infrastructure within the CUT. After this configuration, the CUT initiates a test operation by applying the test stimuli, resulting in outputs for the faulty scenario. These results are then compared with the reference outcomes. Based on this comparison, faults can be categorized as follows:

- *Detected Unrecoverable Error (DUE)*: The fault induced a critical condition on the CUT operation, such as CUT halt, crash, or timeout.
- *Silent Data Corruption (SDC)*: The fault changed the CUT operation, creating differences in the faulty outcomes with respect to the fault-free case.
- *Masked*: The fault did not impact the CUT operation; there is no difference between the faulty and fault-free outcomes. This definition naturally accounts for multi-cycle propagation and reconvergent fan-out in combinational circuits, since any latent error that eventually surfaces will be classified as SDC or DUE rather than masked.

It is worth noting that the fault outcome classification enables the derivation of standard reliability metrics, such as the Architectural Vulnerability Factor (AVF) or Soft Error Rate (SER). Hence, SHADOWFI results can be mapped to specific metrics required by functional safety standards; for example the Failure Mode and Effects Analysis (FMEA) in the ISO26262 standard from the automotive domain.

SHADOWFI incorporates the support for executing extensive fault injection campaigns using two independent workflows with parallel computation support. The first one consists of a simulation framework based on Verilator [55] implemented on HPC systems, while the second relies on FPGA emulation implemented on the HyperFPGA system infrastructure.

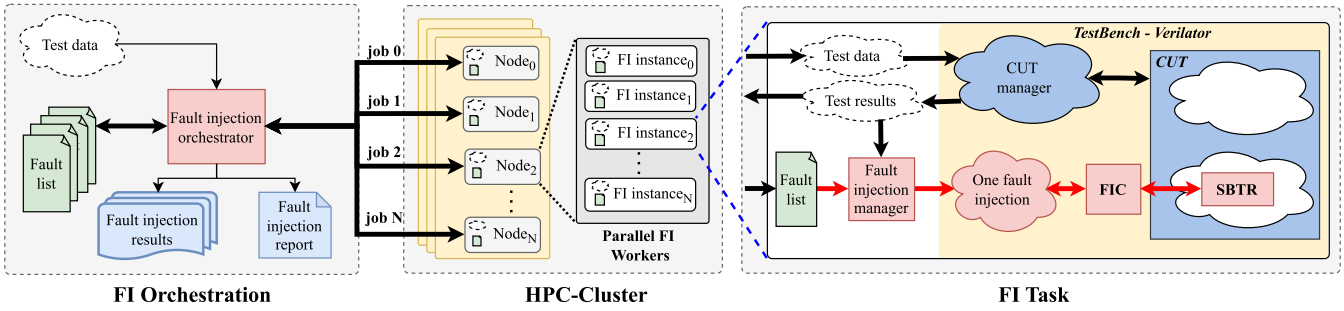


FIGURE 8. SHADOWFI's Simulation workflow architecture for deployment on HPC systems. This workflow distributes the fault injection across multiple computational nodes, where each node executes the simulation of a subset of faults.

1) SIMULATION WORKFLOW

The simulation workflow in SHADOWFI serves two primary purposes: verifying the operation of the CUT after the insertion of saboteur infrastructure, and executing the fault injection campaign. SHADOWFI relies on Verilator for logic simulations and Python scripts for issuing a fault injection campaign on the CUT.

The simulation workflow has been designed to be deployed on HPC infrastructures, enabling the execution of multiple fault injection campaigns in parallel, reducing execution time. Figure 8 illustrates the general overview of the parallel deployment of the simulation workflow on an HPC system. First, an orchestrator task splits the workload by launching several computational jobs in parallel, each using a different fault list. The jobs are executed across the HPC cluster, which consists of multiple nodes. In particular, every job is submitted for execution using a single node, with different available resources. Every job can have multiprocessing capabilities, enabling further workload partitioning into several fault injection tasks. This partitioning involves splitting the fault list assigned to the node to create multiple simulation instances of the same CUT, each running on a parallel core (or worker) with a different set of faults.

The FI task comprises two main parts: a testbench and a fault injection manager. The testbench includes two primary tasks. The first task manages the operation of the CUT during the simulation by applying input patterns and generating simulation results. The second task is responsible for injecting a single fault into the CUT. To facilitate this, the testbench incorporates an FIC and an additional logic interface designed to configure the fault injection infrastructure. SHADOWFI automatically inserts this second task during the simulation setup. The fault injection manager, on the other hand, oversees the fault injection campaign by generating a fault descriptor and a test stimulus for the simulation. Simultaneously, it reads the simulation results and performs the fault impact assessment.

Figure 9 illustrates a simplified fault injection flow using the simulation features provided by SHADOWFI. The simulation flow combines a Python-based orchestrator along

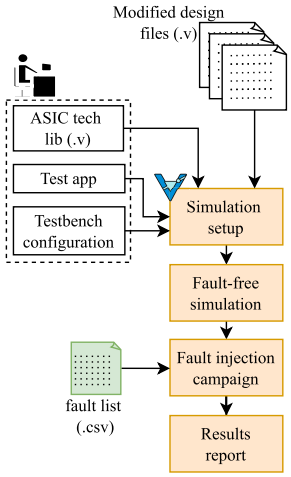


FIGURE 9. Fault injection flow using the SHADOWFI simulation setup.

with logic simulation using Verilator. The simulation flow encompasses four main stages: simulation setup, fault-free simulation, fault injection campaign, and report results.

The flow begins with the simulation setup, where the user must provide a CUT testbench, testbench configurations, Verilator configurations, additional simulation libraries, and stimulus data, if required. During this step, SHADOWFI automatically processes the testbench and inserts a set of functional procedures needed for configuring the saboteur infrastructure. This includes the insertion of a FIC and an external I/O interface for configuring the fault to be injected, as well as to generate output results. After the testbench instrumentation, Verilator compiles the new simulation framework and generates a simulation executable. Following this, the fault-free simulation stage, launches the logic simulation of the CUT with no faults injected. This enables the system to collect reference or golden results, which are used later to assess the impact of faults. The fault-injection campaign stage, evaluates the effect of every fault defined in the input fault list file. This evaluation follows a three-step sequence: first, a fault is selected from the fault list and inserted into the saboteur infrastructure; then, a logic simulation is performed, collecting the CUT faulty results;

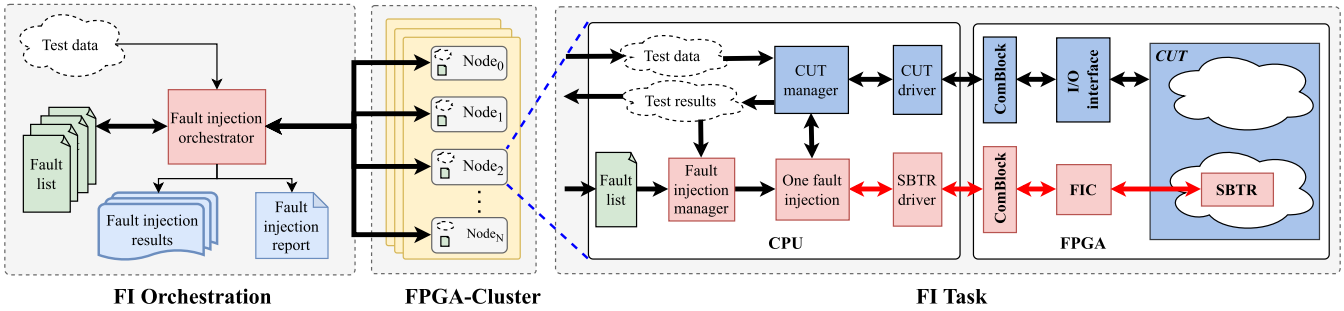


FIGURE 10. SHADOWFI's emulation workflow architecture for deployment on the HyperFPGA system. This workflow distributes the fault injection across multiple FPGA nodes, where each node emulates a subset of faults.

and finally, the faulty results are compared with the golden ones to assess the effect of faults. This process is repeated for all faults included in the fault list. The results report stage analyzes all the outcomes from the fault injection campaign and summarizes the results, categorizing faults according to their effect: Masked, SDCs, or DUEs.

2) FPGA EMULATION WORKFLOW

The FPGA-based fault emulation flow exploits the flexibility, reconfigurability, and programmability of hyperscale FPGA infrastructures. In particular, this work relies on the HyperFPGA system hosted by ICTP [51], [52], which integrates multiple MPSoC-FPGA nodes interconnected through a high-speed network. Each node can be independently reconfigured with user-defined hardware, enabling large-scale and parallel FI experiments. Although the implementation presented here uses HyperFPGA, SHADOWFI can be extended to other hyperscale FPGA clusters.

The HyperFPGA platform consists of sixteen MPSoC-FPGA nodes, each combining FPGAs, CPUs, GPUs, and high-speed general-purpose connectors. Two network topologies interconnect the system: a 2D grid connecting the FPGA I/O interfaces and an Ethernet star topology connecting the CPUs. Every node is accessible through a unique IP address and supports hardware–software integration via a Python-based programming interface.

Figure 10 illustrates the SHADOWFI emulation workflow on HyperFPGA. The FI orchestrator running on the host system distributes the workload across the available nodes. Each node is configured with the generated FPGA bitstream, a specific fault list, and the corresponding test data for the CUT. Once configured, each node executes its assigned FI tasks in parallel. The results from each injected fault are sent back to the orchestrator, which merges them, generates a final FI report, and releases all nodes for the next task.

At the node level, the FI architecture combines hardware and software components. The FPGA hosts the CUT and the FIC, which is connected to the FIP inside the CUT. On the CPU side, the FIM and CUT manager coordinate the FI process through ComBlock IP cores [60]. The CUT driver handles normal test data transfers, while the SBTR driver interacts directly with the FIC to inject specific faults.

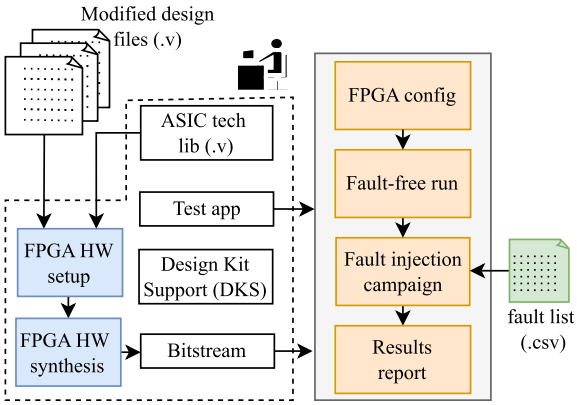


FIGURE 11. Fault emulation flow using the HyperFPGA setup.

The software stack operates on two abstraction levels: *(i)* low-level drivers that manage read/write transactions with the CUT (via ComBlock), and *(ii)* application-level modules that implement FI and CUT control algorithms.

SHADOWFI provides the software and hardware utilities for implementing the fault injection on the HyperFPGA systems. This includes the FIM, the Single fault injection, and the SBTR driver on the software side. Likewise, on the hardware side, SHADOWFI provides the full synthesizable description for the FIC.

Figure 11 illustrates the emulation workflows implemented by SHADOWFI. Initially, the fault evaluation process requires a proper configuration of the hyperscale systems through a development kit support (DKS). In particular, the HyperFPGA provides a seamless DKS compatible with Vivado Design Suite for AMD/Xilinx devices. This configuration facilitates the integration of the instrumented CUT produced by SHADOWFI within a specific hardware platform. As a result, the hardware implementation generates an FPGA bitstream configuration along with a device tree overlay for compatible devices within the system.

After generating the bitstream, the next step configures the nodes accordingly, and then a fault-free execution is performed to collect the reference CUT results. Then, a whole

fault injection campaign is executed across all the faults selected for the evaluation. In the end, all results are merged, and a final report is generated summarizing the fault injection according to the fault effects, classified as Masked, SDCs, or DUEs. This evaluation flow was implemented on the HyperFPGA system using Python scripts within a *JupyterHub* environment as described in [51] and [52].

D. SHADOWFI USER INTERFACE

SHADOWFI features a Command Line Interface (CLI) and a web-based graphical user interface (GUI) that streamlines the entire fault injection workflow, from design preparation to fault injection task. The GUI provides a user-friendly interface for automatically inserting the fault injection infrastructure and prepare the scripts for the subsequent fault injection orchestration using either the simulation or emulation procedures. Figure 12 depicts the GUI that is divided into five sections, as described below:

- **Load design**: This entry creates the fault injection project, which involves loading the HDL design files, link library paths, and setting parameter configurations specific to the CUT under evaluation.
- **HDL elaboration**: Initiates the HDL transformation from HDL source files to RTL netlist according to the selected top entity of the design.
- **SBTR place & route**: This section provides several configuration options, including the fault injection scope (i.e., selection of components inside the design: random, top or hierarchical), the fault model (e.g., stuck-at, SET, SEU, or MEU), and the number of saboteurs to be inserted across the selected components (i.e., *full*, or *statistical*.
- **Fault injection setup**: Prepares the CUT to initiate fault injections. This entry provides two configuration alternatives: Simulation Setup and Emulation Setup. The simulation setup receives the CUT's testbench configuration and compiles it using Verilator. The emulation setup generates an output directory including the modified HDL design files, the fault list, and the saboteur infrastructure hierarchy to be inserted within the FPGA's Design Support Kit provided by the hyperscale system.
- **Fault injection campaign**: Configures and initiates the fault injection task using either a standalone configuration on a local machine or a parallel execution on an HPC system. In the first case, fault injection initiates automatically. In the case of HPC systems, the graphical interface generates HPC runtime scripts (e.g., SLURM scripts) for automatically launching multiple jobs on an HPC infrastructure.

The SHADOWFI documentation provides a detailed user guide about the usage of both CLI and GUI interfaces.[4]

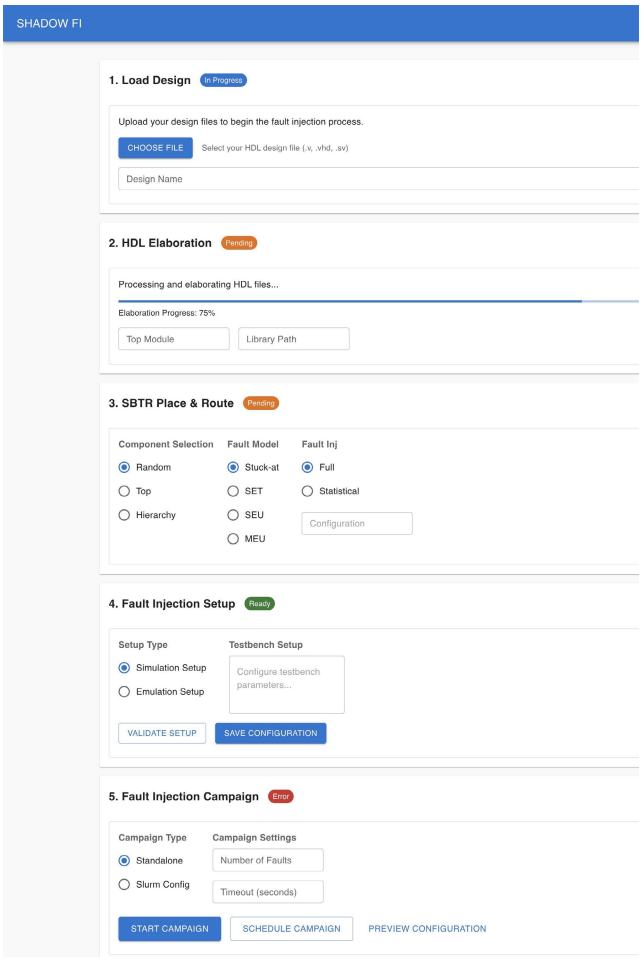


FIGURE 12. SHADOWFI web-based graphical user interface (GUI).

IV. EXPERIMENTAL SETUP AND RESULTS

The SHADOWFI's workflows were implemented in Python, which controls and interacts with different EDA tools, simulators, and hardware devices. In order to facilitate scalability and reproducibility, SHADOWFI leverages the usage of containers.[5] The container includes all necessary tools to enable the end-to-end operation of each workflow. Further technical details about the SHADOWFI implementation, including the tools versioning can be found in the framework repository.[6]

We evaluate the end-to-end SHADOWFI workflow by applying the complete saboteur insertion flow to different hardware accelerators used as CUTs. Every circuit was evaluated under different saboteur infrastructure dimensions, determining the maximum number of actionable faults, either single or multiple. For the experiments, we selected five fault injection infrastructure configurations, denoted as 3 k, 6 k, 12 k, 24 k, and 48 k, with capabilities to inject up to 3,000, 6,000, 12,000, 24,000, and 48,000 faults, respectively. Such saboteur configurations permit the analysis of the impact of

[4]https://shadowfi-userdocs.readthedocs.io/en/latest/

[5]https://docs.sylabs.io/guides/4.3/user-guide/introduction.html

[6]https://github.com/divadnauj-GB/shadowfi.git

inserting small (i.e, 3 k) to large (i.e., 48 k) fault injection infrastructures on the performance of the SHADOWFI. In fact, these experimental configurations aim to serve as illustrative benchmarks of SHADOWFI usage. However, it is important to point out that SHADOWFI can be configured to conduct any fault sampling strategy supporting rigorous statistical analyses if required by the user.

In particular, we profiled the execution time and memory usage for the main tasks in SHADOWFI. This work focuses on the analysis of five tasks as follows: *i)* HDL Elaboration, *ii)* Saboteur Placement & Routing, *iii)* Simulation Setup, *iv)* FPGA Implementation, *v)* FI Simulation-based, ot FI Emulation-based.

The first task transforms the HDL source design files of the CUT into a basic RTL netlist description, whereas the second inserts the saboteur infrastructure inside the CUT according to the user configuration. The third task generates a simulation executable from an evaluation testbench using Verilator. This executable is used as a verification step after the saboteur insertion, but it is also used as a fault injection instrument within the SHADOWFI simulation workflow. The fourth task corresponds to the FPGA compilation flow (i.e., synthesis, implementation, and bit-stream generation) used for generating the necessary hardware configuration files compatible with the HyperFPGA system. This process was conducted using Vivado Design Suite for the `HyperFPGA-3be11` part, which incorporates the Xilinx device `xczu3eg-sfvc784-1-e` [51]. Finally, the fifth task refers to the fault injection process, which can be performed using either the simulation or FPGA emulation workflows. Regardless of the selected evaluation flow, the maximum number of faults to be evaluated depends on the maximum number of saboteur circuits inserted inside the CUT.

The memory usage of every task was profiled by using the `memory-profiler` package through the `mprof` command, while the CPU time execution was obtained using timestamps captured at start/end of every executed task within the SHADOWFI using Python's `time` package. The FPGA runtime was measured as the total FPGA-SoC time using timestamps within the HyperFPGA system. Hence, the reported time in Table 3 includes the end-to-end execution time required to compute the test app on the target circuit. This execution time comprises OS overhead, data loading, CPU-FPGA data exchange, and final output generation. All tasks were profiled using the same strategy, except for the FPGA Implementation task, for which the memory usage and CPU execution time were obtained directly from the logs provided by the vendor EDA tool.

Further experiments were conducted to assess the overall time-per-fault, considering distributed computing configurations using 1, 2, 4, 8, and 16 parallel computing nodes. It is worth noting that all selected nodes implement an identical simulation/emulation engine; however, each node computes a different set of faults from the fault list. In fact, the complete fault list is divided into chunks, where every chunk of faults corresponds to the total number of faults divided by the number of selected compute nodes. In this case, the time per fault was calculated as the total fault injection time divided by the number of evaluated faults.

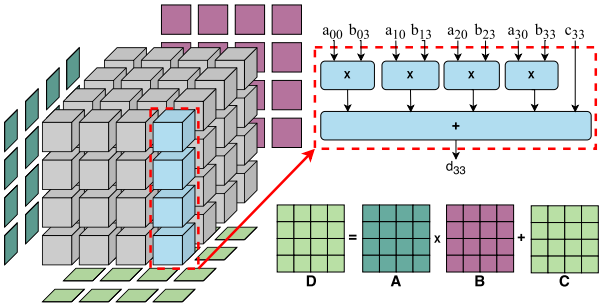


FIGURE 13. General illustration of a 4 × 4x4 Tensor Core architecture for matrix multiplication used as evaluation benchmark of SHADOWFI.

It is important to highlight that there are currently no open-source or SHADOWFI equivalent hardware fault-emulation platforms that provide end-to-end fault assessment for IC designs. Moreover, none of the prior work has adopted hyperscale computing to speed up the evaluation of fault injections as introduced by SHADOWFI. Consequently, a direct comparison with such frameworks is not possible. Instead, we compared the performance per fault provided by SHADOWFI with a commercial logic simulation tool enabled to simulate hardware faults [61]. This comparison enables assessing the speed up provided by the SHADOWFI's simulation or emulation workflows.

The experiments associated with the SHADOWFI's fault instrumentation tasks were conducted on a workstation HP Z2 G5 with an Intel Core i9-10800 CPU featuring 20 cores and 32 GB of RAM. The fault injection task was deployed into two different computational systems. First, the simulation workflow was implemented on the Leonardo HPC system from CINECA using 16 nodes and 8 tasks per node, whereas the FPGA emulation was deployed on the HyperFPGA system from the Multidisciplinary Laboratory (MLAB) at ICTP [52].

A. EVALUATION BENCHMARKS

Table 3 presents a detailed description of the benchmarks used to evaluate SHADOWFI. For the evaluations, four different hardware accelerators with different complexities in terms of component size granularity, hierarchical interconnections, and number of instantiated components were used. In detail, this work utilized a Tensor Core (TCU) [62], a Stereo Vision Core (SVC) accelerator [63], and two different versions of Special Function Units (SFUs) that implement trigonometric functions [28], [64]. Additionally, the table reports the general design hierarchy of each circuit, the number of instances per component, the circuit size in terms of generic gate cells, the number of FPGA resources, and the execution time of selected benchmarks, both in simulation and FPGA implementation.

TABLE 3. Main characteristics of the selected circuits and test benchmarks.

Circuit	Design hierarchy	Num. Inst.	Wires	Gtech Cells		Faults	FPGA resources		Fmax.	Test app.	Simulation time(*)	FPGA-system runtime
				Comb.	Seq.		LUT(%)	DFF(%)				
TCU	top	1	576,994	440,352	46,304	2,307,976	34,813 (49.34%)	46,304 (32.81%)	120 Mhz	4,096 MM	49s	1s
	- dot_unit_core	16	35,934	27,522	2,894	143,736						
	- fpadd	4	2,625	1,814	295	10,500						
	- intadder_27	1	369	258	54	1,476						
	- intadder_31	1	487	298	124	1,948						
	- leftshifter	1	301	187	27	1,204						
	- lzc	1	123	74	16	492						
	- rightshifter	1	218	165	20	872						
	- fpmult	4	6,222	4,930	292	24,888						
	- fpmult_pipe	1	5,561	4,667	292	22,244						
	-intadder33	1	570	302	166	2,280						
	- intmult	1	4,532	4,104	82	18,128						
	- inputieee	2	145	79	0	580						
	- outputieee	1	171	105	0	684						
SVC	top	1	500,000	59,264	425,966	2,000,000	28,599(**) (40.53%)	63,840 (45.24%)	200 Mhz	Three stereo images	600s	1.4s
	- census_transform	2	71,015	2,475	68,529	568,120						
	- lrcc	1	1,117	585	508	4,468						
	- shd	1	356,705	53,728	288,400	1,426,820						
	- disp_cmp	126	129	91	0	65,016						
	- num_ones	128	286	158	96	146,432						
	- window_sum	64	4,420	293	4,102	1,131,520						
SFUv1	top	1	28,288	23,464	0	113,152	3,031 (4.30%)	0 (0%)	23 Mhz	140,000 ops.	105s	1.4s
	- clz	1	312	248	0	1,248						
	- quadratic_interpolator	1	18,839	14,295	0	75,356						
	- fused_accum_tree	1	8,042	4,533	0	32,168						
	- lut_ops	1	9,721	8,883	0	38,884						
	- squaring	1	759	737	0	3,036						
	- sfu_exceptions	1	1,287	1,188	0	5,148						
SFUv2	top	1	51,414	46,805	106	205,656	4,447 (6.30%)	106 (0.08%)	54 Mhz	100,000 ops.	444s	0.9s
	- cordic	1	17,807	16,001	105	71,228						
	- cordic_ieee	1	130	29	3	520						
	- fpmult	2	4,542	4,478	0	18,168						
	- fpadd	3	2,379	2,075	0	9,516						
	- log2	1	5,980	5,328	0	23,920						
	- log2_ieee	1	124	63	0	496						
	- log2_lut_64x23	1	610	594	0	2,440						
	- intmult	1	3,528	3,482	0	14,112						
	- IntAdder	1	413	615	0	1,652						
	- clz	1	153	121	0	612						
	- exp2	1	5,965	4,868	0	23,860						
	- exp2_ieee	1	137	76	0	548						
	- exp2_lut_64x23	1	631	617	0	2,524						
	- intmult	1	3,528	3,482	0	14,112						
	- IntAdder	1	506	130	0	2,024						
	- rsqrt	1	21,115	20,261	0	84,460						
	- rsqrt_ieee	1	114	53	0	456						
	- fpmult	4	4,542	4,478	0	18,168						
	- fpadd	1	2,379	2,075	0	9,516						

(*) Simulation time using a commercial tool
(**) LUTs used for both combinational logic and memory storage

The circuit size was determined through a synthesis process using generic cells (Gtech) from Yosys, as well as an FPGA implementation. In the first case, the generic cell library of Yosys was used to determine the number of wires and the number of combinational and sequential cells. These steps provide insightful information about the circuit complexity as well as the number of possible faults to be evaluated.

Columns 8, 9, and 10 report the FPGA usage in terms of LUTs, flip-flops, and the maximum operational frequency. Column 11 lists the specific test application tailored to the target CUT. Columns 12 and 13 in Table 3 report the total execution time when using either simulation or FPGA implementation of every benchmark for each circuit under test. It is noticeable that the FPGA implementation takes less than two seconds to execute all benchmarks, while the RTL simulation, using commercial tools, can take from tens to hundreds of seconds per simulation, depending on the evaluated CUT.

It is important to note that the selected benchmarks were used to demonstrate the usability and flexibility provided by SHADOWFI. In fact, we demonstrated that regardless of the benchmark characteristics, SHADOWFI can be used to set up and execute a complete end-to-end fault injection campaign. In particular, SHADOWFI provides nine configuration steps using either the CLI or GUI modes. This demonstrates that the framework can be applied to any IC design available in any HDL form. Further technical information about the configuration details of SHADOWFI on each of the evaluated benchmarks is available in the SHADOWFI repository and the documentation resources.[7]

1) TENSOR CORE (TCU) DESCRIPTION

The first circuit under test corresponds to a TCU (i.e., also known as *Matrix Core*), a Domain-Specific Architecture (DSA) dedicated to performing in hardware 4×4 matrix

[7]https://github.com/divadnauj-GB/shadowfi

multiplications in Floating-Point (FP32) representation. The TCU corresponds to a fundamental AI accelerator which is part of modern processors and Graphic Processing Units (GPUs), implementing the matrix multiplication operation as $D = A \times B + C$ where A and B correspond to the input matrices to be multiplied, while C and D correspond to the accumulation and result matrices, respectively. Figure 13 illustrates the main architecture of the TCU under evaluation. In detail, the TCU comprises 16 Dot-Product-Units (DPUs), which in turn implement an input layer of multipliers followed by several layers of floating-point adders. It is worth noting that every FP32 adder and multiplier incorporates internal basic components, including shifters, lead-zero counters (LZC), and integer adders and multipliers. This TCU accelerator relies on VHDL hardware description language, and it is available in [62].

Table 3 reports the TCU size in terms of Gtech cells, comprising approximately 570,000 wires, 440,000 combinational cells, and around 463,000 sequential cells. The size of the TCU determines a significant number of fault locations, resulting in approximately 2.3 million faults across the entire accelerator. Instead, when it comes to the FPGA implementation, the synthesis results report around 49.3% of logic resources as LUTs and around 33% of flip-flops with a maximum operative frequency of 120 MHz.

For the experiments, we used 4,096 matrix multiplication (MM) samples from the inference of ResNet-18 on the CIFAR-10 dataset as a test benchmark for the TCU. The required simulation time for the selected test benchmark is approximately 49 seconds using an industrial logic simulation tool. At the same time, the FPGA implementation on the HyperFPGA system takes less than 1 second to perform the same number of operations.

In order to evaluate the functionality of SHADOWFI, 16 floating-point adders inside the TCU were selected to be instrumented with saboteur circuits. Specifically, we selected a single adder circuit per dot-product unit of the TCU. It is worth noting that SHADOWFI was configured to select automatically the number of fault locations to insert 3 k, 6 k, 12 k, and 48 k faults across the selected components.

2) STEREO VISION CORE (SVC) DESCRIPTION

The second circuit under test corresponds to the SVC, which is designed to calculate the disparity map of two stereo images in real-time using a stream processing architecture [65]. This computational feature enables the processing of stereo images in real-time, as the camera sensors capture the images. In detail, the SVC accelerator implements the stereo correspondence process by using the Census transform in combination with the sum of Hamming distances as described in [66].

Figure 14 depicts the general SVC architecture used in this work for the SHADOWFI evaluation. This accelerator incorporates two Census Transform modules, one for each stereo image (i.e., the left and right images). A set of D correspondence units is then used to obtain the correspondence of a neighborhood of pixels from D windows on the left image into a single window on the right image. The correspondence units are mainly composed of three main components: a pair of hamming distance calculators (number of ones), a window cost calculator, and a pair of disparity selectors (Disp cmp). Finally, the accelerator incorporates a left-to-right consistency check (LRCC) component, responsible for removing false-positive matching pixels in the occlusion regions of the disparity map. The SVC accelerator used in this work corresponds to the open-source core freely available in [63].

The SVC comprises 322 instances, distributed across two hierarchical levels, as shown in Table 3. The smaller component in terms of Yosys Gtech synthesis corresponds to the `disp_cmp` component (286 wires and 91 combinational cells), while the largest component corresponds to the `census_transform` (71,000 wires, 2,400 combinational cells, and 68,000 sequential cells). The total size of the SVC encloses around 500,000 wires, 59,000 combinational cells, and 426,000 sequential cells. Likewise, the significant size of the SVC accelerator implies a substantial number of possible faulty locations, which in turn account for approximately 2 million faults. Regarding the FPGA implementation of the SVC core, the synthesis results for the selected FPGA indicate around 41% and 45% of LUTs and Flip-flops usage, respectively. Also, the SVC reports a maximum operational frequency of up to 200 MHz.

The test benchmarks used for evaluating the SVC comprise three pairs of stereo images taken from the Middlebury dataset (Teddy, Cones, and Venus). The functional simulation of these benchmarks using the RTL description in VHDL language requires approximately 600 seconds on a commercial simulation tool. At the same time, the HyperFPGA implementation of the accelerator performs the required data processing in around the same time. 1.4s.

The fault injection experiments were conducted using SHADOWFI targeting all `window-sum` components and 64 `disp-cmp` components within the SVC accelerator. For the experiments, we selected a subset of these components to assess different saboteur infrastructure configurations, i.e., 3 k, 6 k, 12 k, 24 k, and 48 k.

3) SPECIAL FUNCTION UNIT (SFU) DESCRIPTION

This work selected two open-source special function units devoted to performing trigonometric and transcendental operations in FP32 format. These accelerators are essential computational units in modern graphics processing units (GPUs), enabling the acceleration of various computational workloads [67]. The first SFU core (hereafter referred as SFUv1), implements seven operations listed as follows: $\sin(x)$, $\cos(x)$, $\frac{1}{\sqrt{x}}$, 2^x, $log_2 x$, $\frac{1}{x}$, and $\sqrt{x}$, where x corresponds to a real number encoded as a IEEE754 standard for FP32. On the other hand the second SFU core (hereafter referred

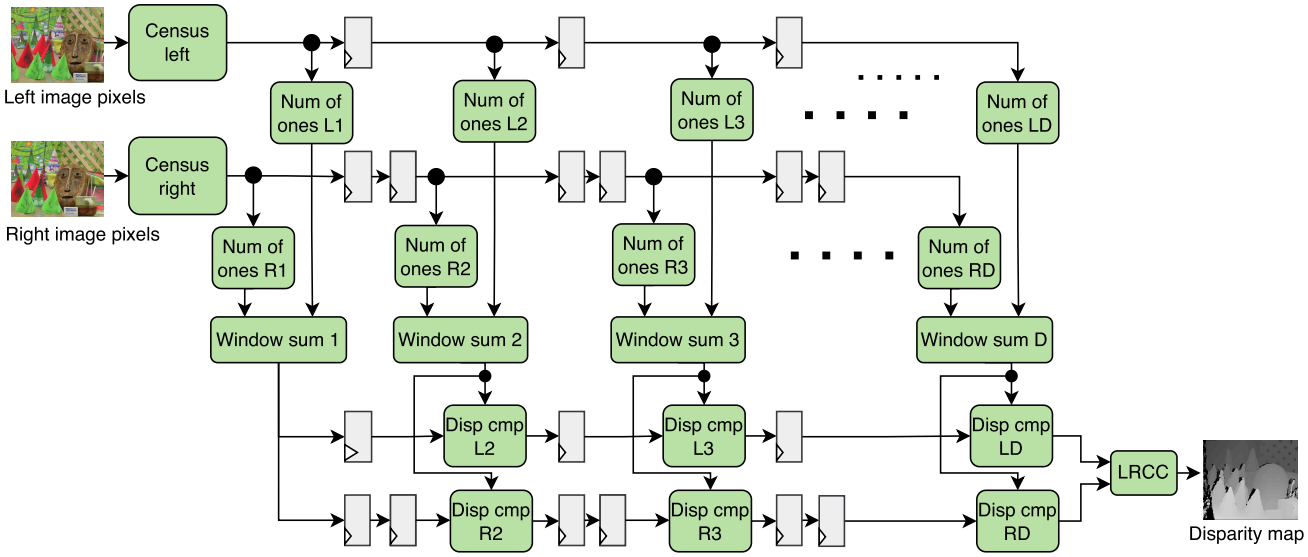


FIGURE 14. General architecture of a census-based stereo vision core (SVC) used as SHADOWFI's benchmark.

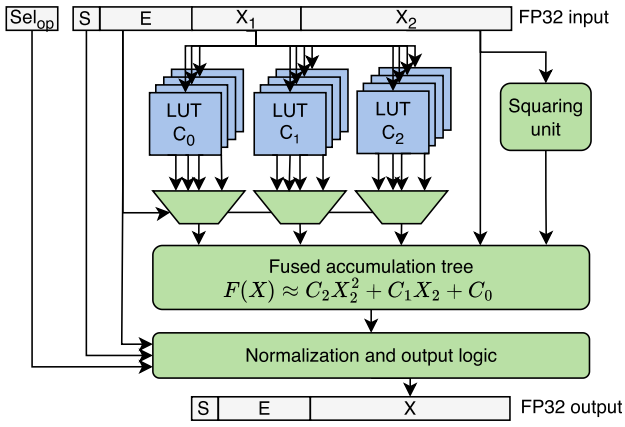


FIGURE 15. General architecture of the special function unit (SFU) based on piecewise quadratic approximation.

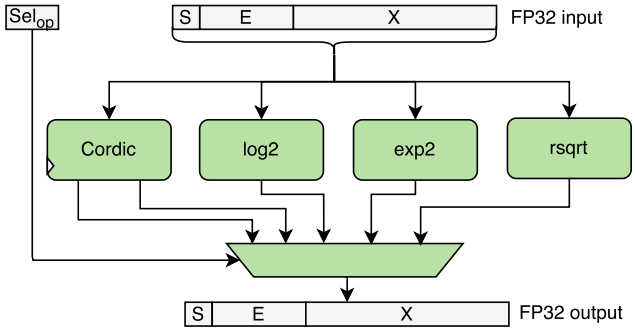


FIGURE 16. General architecture of a modular special function unit (SFU).

as SFUv2), implements five transcendental operations as follows: $\sin(x)$, $\cos(x)$, $\frac{1}{\sqrt{x}}$, 2^x, and $log_2 x$.

The SFUv1 implements the listed operations using the piece-wise polynomial approximation (PPA) as described in [68]. Figure 15 depicts the general architecture for the PPA SFUv1 architecture. The fundamental component in the SFUv1 corresponds to a quadratic interpolator unit, which incorporates a squaring unit, a fused accumulation tree, and a set of LUTs for storing approximation coefficients. This unit implements the polynomial approximations as $F(X) \approx C_2X_2^2 + C_1X_2 + C_0$, where $F(x)$ refers to the transcendental function result, C_0, C_1, and C_2 correspond to the approximation coefficients, and X_1 and X_2 correspond to the fractional part of the input X. The SFUv1 also incorporates additional normalization and output logic components. The SFUv1's components are organized in a two-level hierarchy accounting for six instances as reported in Table 3. The implementation of this core is open-source and fully synthesizable [64].

The size of the SFUv1 core in terms of Gtech cells corresponds to 28,000 wires and 23,500 logic cells, whereas on the FPGA implementation, it uses around 5% of the LUT resources. Also, the maximum operative frequency achieved on the FPGA corresponds to 23 MHz. The test benchmark for the SFUv1 corresponds to a dataset of 140,000 FP32 values inside the SFUv1's input ranges as described in [69]. This dataset is divided into seven smaller datasets, one per SFUv1 operation, comprising 20,000 data elements. This dataset requires approximately 104 seconds to be computed using a commercial simulation tool. In contrast, when utilizing a single node in the HyperFPGA system, the entire data calculation is completed in less than 1.4 seconds.

On the other hand, the SFUv2 corresponds to a modular architecture consisting of four independent computational units as depicted in Figure 16. The first unit implements the $\sin(x)$ and $\cos(x)$ operations using the rotation mode of the CORDIC algorithm. On the other hand, the $log_2(x)$ and 2^x operations are implemented by fast approximation

TABLE 4. Memory utilization and CPU Time execution for the main saboteur insertion tasks across the evaluated CUTs.

	Task	CUT	CET[s](*)	PMU[MiB](**)
Saboteur Insertion	HDL Elaboration	TCU	1.4	98.7
		SVC	344.9	323.3
		SFUv1	3.8	206.2
		SFUv2	2.1	164.3
	SBTR Place & Route	TCU	0.4	55.6
		SVC	0.8	66.1
		SFUv1	0.7	102.4
		SFUv2	1.7	66.4
Fault Injection Setup	Simulation Setup	TCU	16.2	2,007.2
		SVC	70.0	2,348.6
		SFUv1	9.9	1,885.4
		SFUv2	7.1	1,641.6
	FPGA Implement	TCU	538.9	5,322.1
		SVC	476.5	5,603.9
		SFUv1	505.2	5,077.0
		SFUv2	437.7	4,993.8
Fault Injection Orchestration	FI Simulation Verilator	TCU	1.4	39.8
		SVC	11.2	76.6
		SFUv1	4.7	43.3
		SFUv2	16.0	39.8
	FI Emulation HyperFPGA	TCU	1.1	650.5
		SVC	1.5	679.8
		SFUv1	1.5	641.3
		SFUv2	1.0	630.0

(*) CET: CPU Execution Time
(**) PMU: Host Peak Memory Usage

approaches based on piecewise linear approximations. The last hardware unit implements the $\frac{1}{\sqrt{x}}$ using a fast approximation as described in [67]. The SFUv2 incorporates a two-level hierarchical interconnection of 24 instantiated components, including adders, multipliers, and additional logic for normalization and data adjustment.

The size of this SFUv2 version in terms of Gtech cells corresponds to 52,000 wires, 47,000 combinational, and 106 sequential cells. According to the information reported in Table 3, the reciprocal of square root component (rsqrt) corresponds to the largest component in the design, followed by CORDIC, $log_2(x)$, and 2^x components, respectively. When it comes to the FPGA implementation, the SFUv2 uses around 6% of available LUTs and $< 1\%$ of flip-flops, while achieving a maximum operative frequency of 54 MHz.

The test benchmark used for evaluating the SFUv2 utilizes a dataset comprising 100,000 FP32 values within the SFUv2's input ranges, as described in [67]. This dataset is divided into five smaller datasets, one per SFUv2 operation, comprising 20,000 data elements. Running the SFUv2 using the test benchmark requires approximately 444 seconds when using commercial simulation tools, whereas implementing the same test benchmark on a single node in the HyperFPGA system takes less than one second.

For the experiments, SHADOWFI was applied across all components in the SFUv1 and SFUv2, ensuring different saboteur infrastructure sizes (i.e., 3 k, 6 k, 12 k, 24 k, and 48 k).

B. EXPERIMENTAL RESULTS

Table 4 reports the profiling results used for characterizing every SHADOWFI task divided into three main categories. The **Saboteur Insertion** corresponds to the tasks in charge of inserting the saboteur's circuits inside the CUT. The **Fault Injection Setup** encompasses tasks used for setting up the fault injection process in the simulation and emulation workflows. The **Fault Injection Orchestration** corresponds to tasks devoted to injecting faults using either simulation or emulation workflows. This table reports the characterization of the end-to-end flow of SHADOWFI when inserting the 3 k saboteur infrastructure across all evaluated CUTs. The table reports the CPU execution time (CET) and the peak memory usage (PMU) for each task, as well as the CUT under evaluation.

The results show that the saboteurs' insertion tasks are noticeably more lightweight in comparison with other task categories. Although these tasks displayed very short execution times (< 5 seconds) and memory usage (< 400 MiB) across most of the CUTs, the results also indicate that, in the case of the SVC, the HDL Elaboration task lasted approximately 340 seconds longer than for the other evaluated circuits. In fact, we observe that the high-abstraction levels and a significant number of components play a crucial role in the execution time of this task. In particular, the SVC includes more than 300 components, and some of them are described using complex behavioral descriptions, which demand more effort to convert such modules into a low-level RTL netlist. In contrast, the hardware descriptions of the other CUTs have fewer components with less complexity (e.g., multiplexers, adders), making this transformation process straightforward and consequently, faster.

On the other hand, the results indicate that placing and routing the saboteur circuits require less than 2 seconds to be accomplished, making this stage almost negligible in comparison with other tasks across the complete SHADOWFI's flow. It is important to note that the complexity of this specific task does not depend on the complexity or size of the evaluated circuit, but on the number of saboteur components to be inserted and routed across the CUT.

Once the saboteur circuits are inserted inside the CUT, the next step consists of setting up the fault injection workflow. In this case, there are two independent tasks; the first one corresponds to the simulation-based workflow, which generates the required configuration for Verilator to create a simulation executable, while the second task corresponds to the FPGA implementation using Vivado. In general, the resource utilization for both tasks is primarily dominated by the compilation processes of Verilator and Vivado, respectively. The results indicate that the size of the CUT slightly influences the execution time and memory usage. In particular, we observe longer execution times and higher memory usage for the largest CUT, whereas the opposite behavior is observed for the smallest CUT. Besides, the Simulation Setup task is approximately 7 times faster and utilizes approximately 2 times less memory than the FPGA Implementation task.

Finally, after setting up the fault injection, SHADOWFI comprises two tasks for the orchestration of the fault

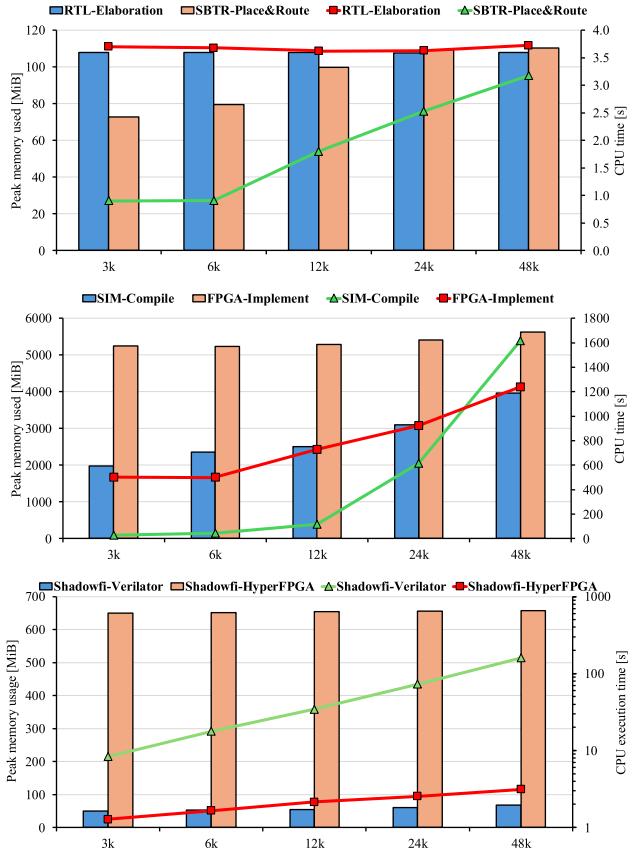


FIGURE 17. Memory usage and CPU time execution for the main SHADOWFI's tasks across different Saboteur infrastructures. Top: saboteur insertion tasks, Middle: fault injection setup, and Bottom: Fault injection orchestration. The bars report the memory utilization, while the line plots show the average execution time.

injections. The results for these tasks report the execution time and the memory utilization per injected fault, using a single computational node. These results show that the time per fault using the simulation workflow varies across the evaluated CUTs, ranging from 1.4 to 16 seconds for the TCU and SFUv2, respectively.

In contrast, for the emulation workflow, the execution time per fault is consistently lower than 1.6 seconds for all evaluated benchmarks. The FPGA emulation workflow makes the time per fault 3 to 16 times faster than the simulation workflow, leading to a significant speedup when evaluating a systematically large number of faults. Taking into account that the execution time and memory utilization of the SHADOWFI's tasks highly depend on the complexity associated with the CUT, in some cases, the simulation workflow offers similar performance features to the emulation workflow. For example, the simulation-based fault injection for the TCU exhibits a time-per-fault difference of just 0.4 seconds, with memory utilization approximately 10 times lower compared to the emulation workflow.

Figure 17 shows the performance impact of inserting saboteur infrastructures of various sizes using SHADOWFI, measured by average execution time and memory usage across five configurations (3 k, 6 k, 12 k, 24 k, and 48 k).

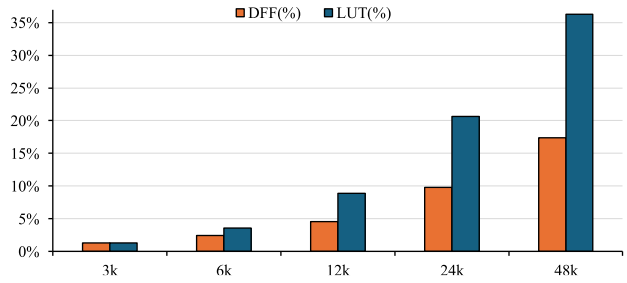


FIGURE 18. FPGA area overhead reported across different fault injection infrastructures sizes.

The figure is divided into three main graphs: the top graph presents results for saboteur insertion tasks; the middle graph reports results for fault-injection-setup tasks; and the bottom graph displays performance data for a single fault injection in both the simulation and emulation workflows of SHADOWFI. The bars report the memory utilization, while the lines denote the average execution time. We observed that as the saboteur circuit increases in size, the memory utilization and the execution time of every task also increase. Nonetheless, such increments are more noticeable for some tasks than others. In particular, when evaluating the SBTR Placing & Routing task, we observed an increment of around 40 MiB of memory usage between the insertion of the smaller FI infrastructure (3 k) and the larger one (48 k). Likewise, the execution time increased by around 2.5 seconds between 3k and 48k inserted saboteurs. Nonetheless, it is important to note that even for large saboteur infrastructure insertion (48 k), SHADOWFI keeps very low computational requirements compared with the other two task categories from the evaluation flow.

In particular, when evaluating the tasks from the fault-injection-setup category, i.e., Simulation Setup and FPGA implementation tasks, the results revealed a noticeable difference in task performance across the different saboteur configurations. For example, when inserting the 3k saboteur, the Simulation Setup utilizes approximately 2.0 GiB of memory while finalizing the task in under 25 seconds. However, for the case of the 48k saboteur, the same task increases the memory utilization to almost 4.0GB ($\approx$ 2X more than 3 k), and the execution time exceeds 1,600 seconds ($\approx$ 26 min) ($\approx$ 64X longer than 3k). Interestingly, for the FPGA implementation, the memory usage is always above 5.0 GiB, with a slight increase of 400 MiB between the 3k and 48k saboteurs. However, the execution time for the 48k saboteur doubles (1,200 seconds) that of the 3 k saboteur (600 seconds). In general, inserting large-scale fault-injection infrastructures within a given CUT significantly impacts the performance of the fault-injection setup tasks, in particular, the execution time for the Simulation Setup explodes for FI infrastructures beyond 48k configurations.

On the other hand, for the fault injection orchestration tasks (see the bottom graph in Figure 17), the execution time per injected fault increases progressively according to the size of the FI infrastructure. The simulation time per fault shifts from

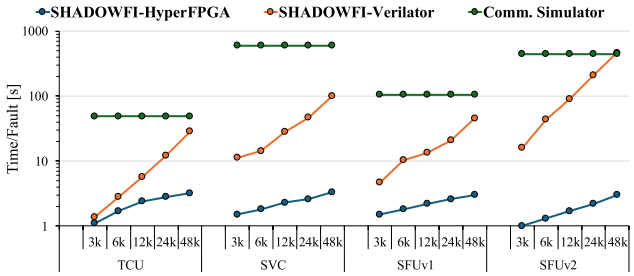


FIGURE 19. Time-per-fault evaluation using SHADOWFI across different saboteur's size configurations for all CUT under evaluation.

≈ 10 seconds in the case of the 3k configuration to more than 100 seconds in the case of the 48k configuration. Similarly, we observed a moderated behavior in the case of FPGA-emulation, where the execution time per fault increased from 1 second for the 3k configuration to 3 seconds in the case of the 48k configuration. These results demonstrate that while the time required to configure the large saboteur infrastructure increases with its size, this effect is notably more pronounced within the simulation-based workflow compared to the FPGA workflow. Therefore, instrumenting the CUT with extensive saboteur infrastructures can lead to diminishing returns, particularly when utilizing simulation for fault injection.

In addition, we evaluated the hardware overhead on the FPGA implementation produced when inserting different FI infrastructures on the evaluated CUT. These results are reported in Figure 18. The figure presents the percentage of FPGA resources, in terms of flip-flops (i.e., DFFs) and logic (i.e., LUTs), for every FI configuration. As expected, the area utilization grows proportionally to the size of the FI infrastructure. However, the LUTs utilization increases by approximately 2X compared with the DFF utilization for all evaluated FI configurations. In detail, the 3k and 6k configurations do not exceed 5% of the FPGA resources; and in particular, 3k has the lowest hardware overhead (<1.5%). In contrast, large configurations (i.e., 24k and 48k) introduced up to 17% and 38% of hardware overhead for the DFFs and LUTs, respectively. In the end, the FI configurations with less than 5% of FPGA resources overhead (3k and 6k) provide the best trade-off in terms of FPGA utilization, fault injection setup, and performance per injected fault.

It is important to note that the insertion of saboteurs has varying effects on the timing of FPGA implementations. We observed that the impact depends on the circuit design. In purely combinational designs, inserting a saboteur can increase the propagation delay by approximately 20%. However, in deeply pipelined accelerators, the additional delay is effectively hidden, resulting in a negligible timing overhead of less than 1%. Regardless, during fault assessment, the speed of evaluation is primarily influenced by the emulation environment, meaning that timing overhead does not significantly affect the execution of a fault injection campaign.

When it comes to the execution time assessment of the fault injection campaigns, the performance provided by SHADOWFI was compared against experiments using RTL simulations using ModelSim, a commercial simulator for HDL designs. It's important to note that a single fault injection, using SHADOWFI or Modelsim, involves activating a fault within the circuit at a specific location and time, during the circuit simulation, while capturing the outcomes to assess the fault's impact later. In this context, Figure 19 presents the experimental results concerning the performance of fault injections when using the SHADOWFI workflows (both simulation-based and emulation-based) as well as an RTL fault injection using Modelsim.

The figure illustrates the execution time per fault for five FI infrastructures applied across all evaluated designs. The results indicate that fault injections using SHADOWFI are significantly faster compared to those implemented directly in RTL simulations. Specifically, the duration of an RTL simulation ranges from 50 to 600 seconds per fault, depending on the CUT being evaluated. In contrast, the time required using SHADOWFI reduces to less than 4 seconds for the emulation-based workflow and less than 20 seconds for the simulation-based workflow. These findings highlight the effectiveness of SHADOWFI in accelerating the assessment of various hardware designs in relation to hardware faults.

On the other hand, when considering different saboteur configurations, the results also indicate that the time-per-fault on SHADOWFI increases as the size of the FI infrastructure increases. Although both SHADOWFI workflows present a similar time increment trend, the simulation workflow is more sensitive to size changes of the inserted saboteur infrastructure. For example, let's consider the SFUv2 using a 48k saboteur configuration. In the emulation workflow, the time per fault increased from 1 to 3.5 seconds compared to the 3k configuration. In contrast, the same circuit using the simulation workflow, reports an execution time increment from 15 seconds to almost 400 seconds under equivalent configurations. Therefore, depending on the selected SHADOWFI's workflows, the selection of the saboteur infrastructure may significantly impact overall performance during fault injections. All in all, it has been demonstrated that moderate-sized saboteur infrastructure (e.g., 3k or 6k configurations) yields a significant performance improvement for both workflows compared to RTL fault injections.

Regarding the overall execution time of a complete fault injection campaign, we accounted the time per fault and the total number of faults for each CUT under evaluation. Figure 20 reports the required execution time of the whole fault injection campaign, assuming the execution on a single computational node for both simulation and emulation workflows. These results include the estimated time required to accomplish the same fault injection campaign when adopting RTL simulation using a commercial simulator. Clearly, RTL fault injections lead to prohibitively long execution times, requiring more than 1,000 days to simulate large hardware circuits (i.e., TCU and SVC), and over

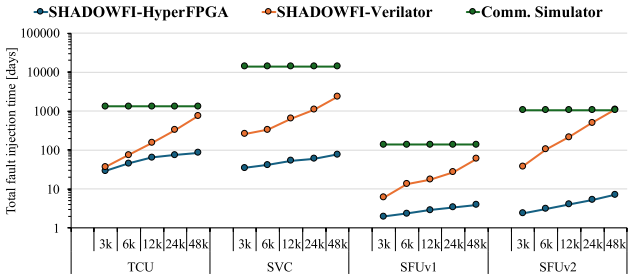


FIGURE 20. Execution time per fault injection campaign; impact of the saboteur's size configuration on the performance of SHADOWFI across the evaluated CUTs.

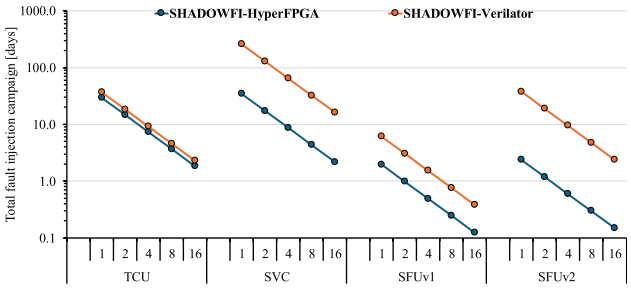


FIGURE 21. SHADOWFI parallel performance; execution time per fault injection campaign vs. 1, 2, 4, 8, and 16 parallel computational nodes.

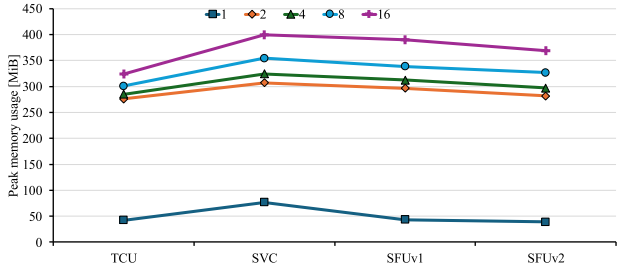


FIGURE 22. Memory utilization per node for the SHADOWFI's simulation-based workflow.

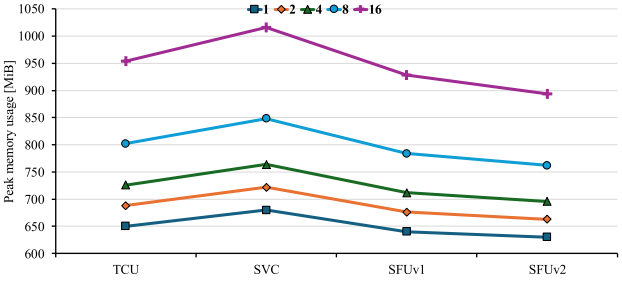


FIGURE 23. Host utilization for the SHADOWFI's emulation-based workflow.

100 days in the case of smaller circuits (i.e., SFUv1 and SFUv2). In contrast, SHADOWFI provides an outstanding reduction in fault injection time using a single computational node, requiring just 50 and 5 days for the largest and smaller CUTs, respectively.

SHADOWFI supports the use of a distributed computing paradigm to accelerate the execution of fault injection campaigns, both in simulation and emulation workflows. To evaluate this feature, we selected the best-performing saboteur configuration (i.e., 3k) and deployed parallel execution using five configurations with 1, 2, 4, 8, and 16 nodes. In addition, each node for the SHADOWFI's simulation workflow was configured in a multiprocessing mode to execute four processes in parallel.

We selected 48,000 faults for each evaluated CUT to calculate the time-per-fault for each parallel configuration. Figure 21 presents the speedup delivered by the SHADOWFI's workflows, considering the time-per-fault, the total number of faults on every CUT, and the level of parallelism. The results show that adopting a distributed computing architecture significantly speeds up the execution of fault injection campaigns by reducing the time from hundreds of days in a single node to one or fewer days when using 16 parallel nodes.

Although increasing the number of computational nodes reduces the execution time of the fault injection campaigns, we observed that using more than eight parallel nodes does not provide a substantial benefit for all evaluated scenarios. For example, let's consider the fault injections applied to the SVC using the emulation-based workflow, in this case, when using eight parallel nodes the fault injection time was reduced by approximately 46 days, nonetheless using 16 parallel nodes generates an improvement of roughly 48 days, this is just two days faster than the eight nodes configuration but it doubles the cost of the evaluations in terms of hardware resources. In the end, the adoption of parallel computation speeds up the SHADOWFI's performance by 1 to 3 orders of magnitude compared to RTL fault injections. In particular, the emulation-based workflow provides the best performance figures, especially when conducting fault evaluation of large hardware design circuits.

In addition, Figures 22 and 23 report the memory utilization across the parallel configurations for both simulation and emulation workflows. It is worth noting that the *simulation workflow* has low memory utilization per node (<400MiB). In fact, we noticed that this slight increase in memory utilization between parallel configurations is due to data transfers across all nodes during the initialization and finalization of the fault injection campaigns. The initialization involves moving data (i.e., fault list, simulation executable, test data) to separate directories. Then, when the task is completed, all the fault injection results, including log reports, are merged into a single directory.

On the other hand, the memory utilization in the *emulation workflow* increases proportionally to the number of parallel nodes under utilization. Also, such memory utilization increments depend on the evaluated CUT and the test data used during the fault injections. In fact, all parallel configurations indicate that the SVC has shown the highest memory utilization, followed by the TCU, SFUv1, and SFUv2, respectively. In particular, when using 16 parallel nodes, the memory profiler reported approximately 1 GiB

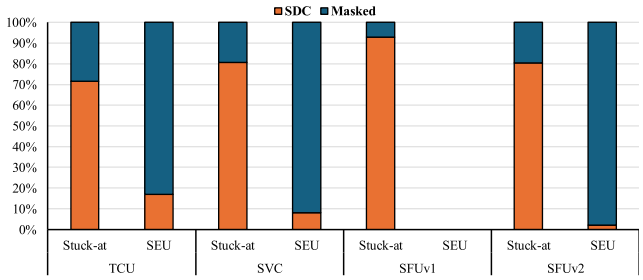


FIGURE 24. Fault classification results generated from SHADOWFI fault injections for Stuck-at and SEU fault models.

of memory utilization, while the TCU, SFUv1, and SFUv2 utilized between 900 and 950 MiB.

We observed that the memory increment occurs at the host side, as it must handle data transfer from and to every computational node in parallel. Let's consider several nodes implementing fault injection on the SVC benchmark. In this case, every node requires two input images as test data and the complete list of faults; each node handles a different fault list. Then, during the operational phase, every node returns data to the host. In the case of the SVC, the output data comprises multiple images, one image per evaluated fault. Consequently, the collected information requires additional memory allocation on the host side to ensure the correct processing, merging, and storage of the entire fault injection campaign.

After conducting the fault injection campaign, SHADOWFI generates a complete report indicating the effect of each injected fault during the operation of all evaluated CUTs. Figure 24 reports the fault injection results for 45,000 stuck-at faults and 3,000 SEUs applied to all circuits being assessed. The figure shows the percentage of faults classified as Silent Data Corruption (SDC) and Masked, according to the fault model. We reported the fault injection results obtained from the *emulation workflow*. Nonetheless, it is essential to note that the *simulation workflow* yields precisely the same results, as they implement the same functionality of the fault injection infrastructure, resulting in identical outcomes.

According to the results, the stuck-at fault model reports the highest percentage of SDCs for all evaluated CUTs compared to SEUs. More than 70% of stuck-at faults lead to data corruption at the outputs of the evaluated CUT, with the SFUv1 being the most sensitive to these faults. This fault model has a more noticeable impact since it persists throughout the entire CUT operation, increasing the likelihood of being activated and propagated as errors at the output of the application or observation point. In contrast, the SEU faults produced less than 20% of data corruption across the evaluated circuits, with the SFUv2 being the one with the lowest percentage of corruption. It is important to note that the SFUv1 does not include flip-flops in its implementation; therefore, we did not report results regarding SEU faults for this CUT, as these faults are only related to flip-flops or memory storage elements.

These results were obtained thanks to the flexibility, scalability, and fast fault evaluation provided by SHADOWFI on different hardware architectures. It is worth noting that this work does not delve into the details of fault impact evaluation. Instead, it focuses on introducing an open-source fault injection framework that contributes to speeding up the fault injection process through emulation and simulation workflows. The usage of SHADOWFI paves the way for obtaining accurate fault characterization results of different IC designs. These results are crucial for further analysis focused on the identification of faults with critical effects, which in turn are very helpful for the development of suitable fault countermeasures both at the hardware level and at the application level. Furthermore, SHADOWFI provides a straightforward way to evaluate the effectiveness of fault mitigation mechanisms, providing high evaluation accuracy since it relies on actual hardware implementation, utilizing RTL and gate-level descriptions.

C. DISCUSSION

The proposed fault emulation framework, SHADOWFI, demonstrates the effective use of hyperscale computing resources to enable scalable, automated, and rapid fault injection in distributed environments, including high-performance computing (HPC) systems and field-programmable gate array (FPGA) clusters. Experimental findings indicate that the framework can be applied to a wide range of integrated circuit (IC) designs while accommodating various fault instrumentation configurations.

To thoroughly illustrate the framework's usability, performance characterization, and scalability across different fault instrumentation sizes, we conducted a series of benchmark assessments. The results indicate that SHADOWFI achieves a reduction in time per fault by two to three orders of magnitude compared to conventional logic simulations executed with commercial tools. Furthermore, the findings reveal that the fault instrumentation incurs a moderate hardware overhead, remaining below 35% for the maximum number of instrumented faults observed throughout the experiments.

It is important to emphasize that SHADOWFI generates a comprehensive fault injection report for each fault, which can be further analyzed for thorough fault characterization and reliability studies. Although such detailed evaluations are not the primary focus of this work, we have included a summary of the fault assessments from each injection campaign across the evaluated benchmarks. Our findings indicate that permanent faults result in Silent Data Corruption (SDC) effects in more than 70% of cases, while transient faults lead to such effects in less than 20% of cases. Future research could delve deeper into assessing the impacts of these faults.

Overall, these results demonstrate that hyperscale infrastructures offer the necessary elasticity and flexibility to accelerate fault injection in IC designs while generating

detailed reports that are essential for comprehensive fault characterization and realistic reliability assessment.

When compared to previous work in the literature on fault emulation for hardware circuits, SHADOWFI stands out for three main reasons: (i) it is platform-independent, (ii) it leverages hyperscale computing, and (iii) it is open-source, which enhances reproducibility.

Firstly, SHADOWFI integrates saboteur infrastructures into the IC design netlist using open-source HDL processing tools. This approach makes it independent of specific FPGA vendors and tools, thereby increasing its portability across various platforms and vendors. In contrast, most fault emulation frameworks rely on vendor-specific tools for fault instrumentation, limiting their portability and risking obsolescence due to a lack of ongoing support and development.

Secondly, to the best of our knowledge, SHADOWFI is the only framework that harnesses the capabilities of hyperscale computing to accelerate fault injection campaigns for IC designs through fault-emulation strategies. Specifically, SHADOWFI provides a simulation and emulation workflow with parallel execution. The simulation workflow can be parallelized and distributed across multiple computational nodes in a high-performance computing (HPC) system, while the emulation workflow can be deployed on FPGA cluster infrastructures.

Finally, SHADOWFI and CrashTest [24] are the only open-source fault emulation frameworks that enable reproducibility of fault injection experiments for IC designs. However, CrashTest relies on outdated FPGA boards and legacy Xilinx tools, making it challenging to use in contemporary fault-injection experiments. As a result, conducting a direct experimental evaluation to compare SHADOWFI with other open-source fault injection frameworks is not feasible.

SHADOWFI demonstrates significant scalability and outstanding acceleration of fault injection tasks. However, such performance is achieved through the use of specialized computational infrastructure, which can sometimes be limited by resource availability when shared with other users. Hence, this constraint can limit the number of concurrent fault injection tasks, especially when the number of faults is large. On the other hand, although SHADOWFI is designed to be extensible for any FPGA cluster infrastructure, it still requires specialized Design Kit Support (DKS) tools for bitstream generation (such as Vivado, Quartus, etc.) and for orchestrating parallel tasks across multiple FPGA devices. Currently, SHADOWFI is fully compatible with the HyperFPGA system. However, it is still required to extend the framework to other FPGA as a Service (FaaS) or FPGA cloud infrastructures, including AWS EC2 F2 instances.

The simulation workflow in SHADOWFI can execute fault injection campaigns on any integrated circuit (IC) design, regardless of its size or whether it is fully synthesizable. In contrast, the emulation workflow is limited to designs that can fit on a single FPGA device. This limitation means that SHADOWFI can perform parallel fault execution by deploying multiple copies of the same IC design across different FPGA devices, applying different faults on each device simultaneously. Despite this constraint, it is important to note that SHADOWFI was designed to be extensible. This allows for the integration of complementary or alternative fault models and supports extremely large IC designs that require more than one FPGA device for operation.

Future work will expand SHADOWFI in three specific directions: (i) deployment on additional cloud FPGA platforms, such as AWS EC2 F2 and Azure Catapult; (ii) the inclusion of broader fault models, including timing violations and aging-related effects; and (iii) leveraging SHADOWFI capabilities for detailed fault characterization, vulnerability assessment, or coverage evaluation across various IC designs and new classes of accelerators, thereby enabling the development of fault tolerance solutions (e.g., selective hardening, error correction, or redundancy insertion).

Specifically, SHADOWFI can be extended to support the deployment of IC designs that require the collaborative operation of multiple FPGAs, facilitating fault evaluation in complex Network-on-Chip architectures, such as graphics processing units, vector accelerators, or multi-core systems. Furthermore, SHADOWFI can be utilized to investigate fault effects on a wide range of IC designs, from multimedia applications to artificial intelligence hardware, including bit-serial matrix–vector multipliers (MVMs) in LSTM accelerators, where temporal fault propagation presents unique challenges. [70], [71].

V. CONCLUSION

This work introduces SHADOWFI, a generic open-source and user-friendly fault injection framework that leverages the parallelism capabilities offered by hyperscale infrastructures to accelerate hardware fault characterization and reliability estimation of complex microelectronic circuits. SHADOWFI provides support for deploying parallel fault injection tasks using either simulation or emulation workflows. In the first case, SHADOWFI includes support for deployment in high-performance computing (HPC) systems by distributing fault injections in parallel across multiple computational nodes. On the other hand, employing FPGA-cluster infrastructures, SHADOWFI can be configured to issue multiple fault injection tasks in parallel on independent FPGA nodes.

SHADOWFI employs a fault instrumentation strategy by inserting saboteur structures into the design's netlist (i.e., at the gate level) of selected components across the integrated circuit design hierarchy, supporting both permanent and transient fault models, with capabilities to be extended to other fault models. The SHADOWFI's performance, scalability, flexibility and usability were evaluated trough four Domain-Specific-Architectures (DSA) hardware accelerators: one Tensor Core (TCU), one Stereo Vision Accelerator (SVC), and two Special Function Units (SFUs). The results showed that SHADOWFI can be applied to any IC design across the selected benchmarks which provide different circuit

features and configurations (i.e., size, hierarchical structure and functionality).

We measured the performance of SHADOWFI during fault injection preparation, focusing on execution time and memory use when inserting saboteurs and setting up for fault injection. The results showed that larger fault injection infrastructures increased both execution time and memory usage during system setup. In contrast, evaluations during fault injection orchestration showed that SHADOWFI's simulation and FPGA emulation workflows run faster than standard RTL fault injections in commercial tools. Moreover, using SHADOWFI with parallel computing resulted in a significant speedup of two to three orders of magnitude compared to RTL fault injections.

Future works envision the extension and usage of SHADOWFI in three main directions: (i) Extension of FPGA support to other platforms (e.g., AWS EC2 F2), (ii) exploration of new fault models, and (iii) comprehensive fault characterization on a variety hardware accelerators across multiple application domains (e.g., multimedia, AI, etc.), including industrial IP cores when possible. SHADOWFI is a crucial tool to obtain precise fault evaluations that later facilitate the development of effective fault countermeasures for complex and large hardware designs, such as AI accelerators or Network-on-Chip architectures. Also, as being open-source, SHADOWFI, can be enhanced or possibly cooperate with EDA tools to guide the insertion of selective hardening mechanisms on IC designs during the designs stages.

ACKNOWLEDGMENT

The authors acknowledge ICTP for awarding this work access to the HyperFPGA infrastructure.

REFERENCES

[1] *International Roadmap for Devices and Systems*, 2022.
[2] A. Evans, C. Fuguet, and D. Million, "OpenSource heterogeneous chipletbased computing architectures," in *Proc. 43rd IEEE/ACM Int. Conf. Comput.-Aided Design*, 2025, pp. 1–8, doi: 10.1145/3676536.3698874.
[3] A. J. Strojwas, K. Y. Y. Doong, and D. Ciplickas, "Yield and reliability challenges at 7nm and below," in *Proc. Electron Devices Technol. Manuf. Conf. (EDTM)*, 2019, pp. 179–181.
[4] J. Han, M. Meyyappan, and J. Kim, "Single event hard error due to terrestrial radiation," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Jan. 2021, pp. 1–6.
[5] I. Hill, P. Chanawala, R. Singh, S. A. Sheikholeslam, and A. Ivanov, "CMOS reliability from past to future: A survey of requirements, trends, and prediction methods," *IEEE Trans. Device Mater. Rel.*, vol. 22, no. 1, pp. 1–18, Mar. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9628165/
[6] R. T. Syed, F. L. Vargas, M. Andjelkovic, M. Ulbricht, and M. Krstic, "Aging and soft error resilience in reconfigurable CNN accelerators employing a multi-purpose on-chip sensor," in *Proc. IEEE 25th Latin Amer. Test Symp. (LATS)*, Apr. 2024, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10534625/
[7] A. D. Singh, "Silent error corruption: The new reliability and test challenge," in *Proc. IEEE 24th Latin Amer. Test Symp. (LATS)*, Mar. 2023, pp. 1–2.
[8] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021, *arXiv: 2102.11245*.
[9] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proc. Workshop Hot Topics Operating Syst.*, 2021, pp. 9–16.
[10] G. Papadimitriou, D. Gizopoulos, H. D. Dixit, and S. Sankar, "Silent data corruptions: The stealthy saboteurs of digital integrity," in *Proc. IEEE 29th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2023, pp. 1–7.
[11] R. Bonderson. (2021). *Training in Turmoil: Silent Data Corruption in Systems At Scale*. Accessed: Apr. 1, 2024. [Online]. Available: https://research.google/pubs/training-in-turmoil-silent-data-corruption-in-systems-at-scale/
[12] D. F. Bacon, "Detection and prevention of silent data corruption in an exabyte-scale database system," in *Proc. 18th IEEE Workshop Silicon Errors Logic-Syst. Effects*, Jun. 2022.
[13] H. Dixit, "Keytone: Silent data corruptions at scale," in *Proc. IEEE 29th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2023, pp. 1–2.
[14] S. K. Bukasa, L. Claudepierre, R. Lashermes, and J. L. Lanet, "When fault injection collides with hardware complexity," in *Foundations and Practice of Security (FPS 2018)* (Lecture Notes in Computer Science), vol. 11358. Cham, Switzerland: Springer, Nov. 2018, pp. 243–256. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-18419-316
[15] N. K. Salih, D. Satyanarayana, A. S. Alkalbani, and R. Gopal, "A survey on software/hardware fault injection tools and techniques," in *Proc. IEEE Symp. Ind. Electron. Appl. (ISIEA)*, Jul. 2022, pp. 1–7.
[16] G. Yu, G. Tan, H. Huang, Z. Zhang, P. Chen, R. Natella, Z. Zheng, and M. R. Lyu, "A survey on failure analysis and fault injection in AI systems," *ACM Trans. Softw. Eng. Methodol.*, May 2025, doi: 10.1145/3732777.
[17] A. Ruospo, A. Balaara, A. Bosio, and E. Sanchez, "A pipelined multi-level fault injector for deep neural networks," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9250866/
[18] J. E. R. Condia, J.-D. Guerrero-Balaguera, F. F. D. Santos, M. S. Reorda, and P. Rech, "A multi-level approach to evaluate the impact of GPU permanent faults on CNN's reliability," in *Proc. Int. Test Conf.*, Sep. 2022.
[19] J. D. Guerrero Balaguera, J. E. Rodriguez Condia, F. Fernandes Dos Santos, M. Sonza Reorda, and P. Rech, "Understanding the effects of permanent faults in GPU's parallelism management and control units," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2023, doi: 10.1145/3581784.3607086.
[20] Q. Liu, H. Tang, and P. Zhang, "Fault injection attack emulation framework for early evaluation of IC designs," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 1, pp. 1–25, Jan. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3480962
[21] M. Monopoli, M. Biondi, P. Nannipieri, S. Moranti, and L. Fanucci, "RADSAFiE: A netlist-level fault injection user interface application for FPGA-based digital systems," *IEEE Access*, vol. 13, pp. 28809–28823, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10877826/?arnumber=10877826
[22] J. Chaudhuri, H. Nassar, D. R. E. Gnad, J. Henkel, M. B. Tahoori, and K. Chakrabarty, "FLARE: Fault attack leveraging address reconfiguration exploits in multi-tenant FPGAs," 2025, *arXiv:2502.15578*.
[23] F. Ferlini, F. Viel, L. O. Seman, H. Pettenghi, E. A. Bezerra, and V. R. Q. Leithardt, "A methodology for accelerating FPGA fault injection campaign using ICAP," *Electronics*, vol. 12, no. 4, p. 807, Feb. 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/4/807
[24] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin, "CrashTest: A fast high-fidelity FPGA-based resiliency analysis framework," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2008, pp. 363–370. [Online]. Available: https://ieeexplore.ieee.org/document/4751886/
[25] M. Zink, D. Irwin, E. Cecchet, H. Saplakoglu, O. Krieger, M. Herbordt, M. Daitzman, P. Desnoyers, M. Leeser, and S. Handagala, "The open cloud testbed (OCT): A platform for research into new cloud technologies," in *Proc. IEEE 10th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2021, pp. 140–147.
[26] (2010). *Scaled CMOS Technology Reliability Users Guide*. Accessed: Apr. 1, 2024. [Online]. Available: https://nepp.nasa.gov/files/18209/091025JPLWhiteScaled%20CMOS%20Technology%20Reliability%20Users%20Guide%20110.pdf
[27] R. McWilliam, S. Khan, M. Farnsworth, and C. Bell, "Zero-maintenance of electronic systems: Perspectives, challenges, and opportunities," *Microelectron. Rel.*, vol. 85, pp. 122–139, Jun. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002627141830163X
[28] J.-D. Guerrero-Balaguera, C. Moreno, and J. E. R. Condia. (2020). *Special Functions Units (SFU)*. [Online]. Available: https://opencores.org/projects/specialfunctionsunit

[29] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[30] R. Shafik, A. Das, S. Yang, G. Merrett, and B. Al-Hashimi, "9-design considerations for reliable embedded systems," in *Reliability Characterisation of Electrical and Electronic Systems*. Oxford, U.K.: Woodhead Publishing, 2015, pp. 169–194. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781782422211000095

[31] J. Wadden and K. Skadron, "Chapter 22-advances in GPU reliability research," in *Advances in GPU Research and Practice*. San Mateo, CA, USA: Morgan Kaufmann, 2017, pp. 617–647. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128037386000227

[32] J. E. R. Condia, P. Rech, F. F. d. Santos, L. Carro, and M. S. Reorda, "An effective method to identify microarchitectural vulnerabilities in GPUs," *IEEE Trans. Device Mater. Rel.*, vol. 22, no. 2, pp. 129–141, Jun. 2022.

[33] S. Basu, "Chapter 2-treatise on hazards and risks," in *Plant Intelligent Automation and Digital Transformation*. New York, NY, USA: Academic, 2024, ch. 2, pp. 19–45. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128244579000017

[34] W. Mansour and R. Velazco, "An automated SEU fault-injection method and tool for HDL-based designs," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 4, pp. 2728–2733, Aug. 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6555963/

[35] R. Nyberg, J. Heyszl, D. Rabe, and G. Sigl, "Closing the gap between speed and configurability of multi-bit fault emulation environments for security and safety–critical designs," in *Proc. 17th Euromicro Conf. Digit. Syst. Design*, Aug. 2015, pp. 1119–1129. [Online]. Available: https://ieeexplore.ieee.org/document/6927234/

[36] C. Fibich, P. Rössler, S. Tauner, H. Taucher, and M. Matschnig, "A netlist-level fault-injection tool for FPGAs," *E I Elektrotechnik und Informationstechnik*, vol. 132, no. 6, pp. 274–281, Sep. 2015, doi: 10.1007/s00502-015-0315-4.

[37] E. Sanchez, L. Sterpone, and A. Ullah, "Effective emulation of permanent faults in ASICs through dynamically reconfigurable FPGAs," in *Proc. 24th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2014, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/6927478/

[38] C. Thurlow, H. Rowberry, and M. Wirthlin, "TURTLE: A low-cost fault injection platform for SRAM-based FPGAs," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2019, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/8994782/

[39] S. Di Carlo, P. Prinetto, D. Rolfo, and P. Trotta, "A fault injection methodology and infrastructure for fast single event upsets emulation on Xilinx SRAM-based FPGAs," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2014, pp. 159–164. [Online]. Available: https://ieeexplore.ieee.org/document/6962073/

[40] J. R. Azambuja, F. Kastensmidt, and J. Becker, "Configuration bit-stream fault injection experimental results," in *Hybrid Fault Tolerance Techniques To Detect Transient Faults in Embedded Processors*. Cham, Switzerland: Springer, 2014, pp. 69–74, doi: 10.1007/978-3-319-06340-96.

[41] G. L. Nazar and L. Carro, "Fast single-FPGA fault injection platform," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2012, pp. 152–157. [Online]. Available: https://ieeexplore.ieee.org/document/6378216/

[42] U. Legat, A. Biasizzo, and F. Novak, "Automated SEU fault emulation using partial FPGA reconfiguration," in *Proc. 13th IEEE Symp. Design Diag. Electron. Circuits Syst.*, Apr. 2010, pp. 24–27. [Online]. Available: https://ieeexplore.ieee.org/document/5491825/

[43] J. Grinschgl, A. Krieg, C. Steger, R. Weiß, H. Bock, and J. Haid, "Automatic saboteur placement for emulation-based multi-bit fault injection," in *Proc. 6th Int. Workshop Reconfigurable Commun.-Centric Syst.-Chip (ReCoSoC)*, Jun. 2011, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/5981521/

[44] B. Rahbaran, A. Steininger, and T. Handl, "Built-in fault injection in hardware–The FIDYCO example," in *Proc. 2nd IEEE Int. Workshop Electron. Design, Test Appl.*, Jan. 2004, pp. 327–332. [Online]. Available: https://ieeexplore.ieee.org/document/1409860/

[45] L. A. B. Naviner, J.-F. Naviner, G. G. dos Santos, E. C. Marques, and N. M. Paiva, "FIFA: A fault-injection–fault-analysis-based tool for reliability assessment at RTL level," *Microelectron. Rel.*, vol. 51, nos. 9–11, pp. 1459–1463, Sep. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271411002162

[46] S. Rudrakshi, V. Midasala, and S. Naga Kishore Bhavanam, "Implementation of FPGA based fault injection tool (FITO) for testing fault tolerant designs," *Int. J. Eng. Technol.*, vol. 4, no. 5, pp. 522–526, 2012. [Online]. Available: http://www.ijetch.org/show-45-324-1.html

[47] P. K. Lala, "Transient and permanent fault injection in VHDL description of digital circuits," *Circuits Syst.*, vol. 3, no. 2, pp. 192–199, Apr. 2012. [Online]. Available: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=18548

[48] A. R. Khatri, A. Hayek, and J. Borcsok, "RASP-FIT: A fast and automatic fault injection tool for code-modification of FPGA designs," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 10, 2018. [Online]. Available: https://thesai.org/Publications/ViewPaper?Volume=9&Issue=10&Code=IJACSA&SerialNo=4

[49] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 502–506. [Online]. Available: https://ieeexplore.ieee.org/document/5090716

[50] M. Leeser, S. Handagala, and M. Zink, "FPGAs in the cloud," *Comput. Sci. Eng.*, vol. 23, no. 6, pp. 72–76, Nov. 2021. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MCSE.2021.3127288

[51] W. F. Samayoa, M. L. Crespo, S. Carrato, A. Silva, and A. Cicuttin. (Aug. 2023). *HyperFPGA: An Experimental Testbed for Heterogeneous Supercomputing*. [Online]. Available: https://www.researchsquare.com/article/rs-3278560/v1

[52] F. Samayoa and W. Oswaldo. (2024). *HyperFPGA: SoC-FPGA Cluster Architecture for Supercomputing and Scientific Applications*. [Online]. Available: https://arts.units.it/handle/11368/3068428

[53] R. L. Sierra, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. S. Reorda, "Optimizing the analysis and evaluation of logic simulation workloads in HPC systems," in *Proc. IEEE 17th Int. Conf. Appl. Inf. Commun. Technol. (AICT)*, Jan. 2023, pp. 1–6.

[54] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free verilog synthesis suite," in *Proc. 21st Austrian Workshop Microelectron. (Austrochip)*, 2013.

[55] W. Snyder, "Verilator 4.0: Open simulation goes multithreaded," in *Proc. Open Source Digit. Design Conf. (ORConf)*, 2018.

[56] Z. Gao, A. Yan, Z. Huang, J. Cui, B.-H. Roh, G. Liu, P. Girard, and X. Wen, "Graph-based multitask transfer learning for fault detection and diagnosis of few-shot analog circuits," *IEEE Internet Things J.*, vol. 12, no. 12, pp. 21264–21279, Jun. 2025.

[57] R. Ma, S. Holst, H. Xu, X. Wen, S. Wang, J. Li, and A. Yan, "Highly defect detectable and SEU-resilient robust scan-test-aware latch design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 33, no. 2, pp. 449–461, Feb. 2025.

[58] A. Yan, J. Zhang, X. Xu, H. Li, N. Bai, Z. Huang, X. Wang, X. Wen, and P. Girard, "TUTPFL: Triple node upset-tolerant and single-event transient-filtered low-power latch with HSPICE and FPGA-based verifications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 33, no. 10, pp. 2783–2794, Oct. 2025.

[59] A. Yan, C. Hu, J. Li, N. Bai, Z. Huang, T. Ni, G. Patrick, and X. Wen, "Cost-optimized Double-Node-Upset-Recovery latch designs with aging mitigation and algorithm-based verification for long-term robustness enhancement," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 33, no. 6, pp. 1765–1773, Jun. 2025.

[60] K. S. Mannatunga, L. G. G. Ordóñez, M. B. Amador, M. L. Crespo, A. Cicuttin, S. Levorato, R. Melo, and B. Valinoti, "Design for portability of reconfigurable virtual instrumentation," in *Proc. 10th Southern Conf. Program. Log. (SPL)*, Apr. 2019, pp. 45–52.

[61] N. I. Deligiannis, J.-D. Guerrero-Balaguera, R. Cantoro, S. E. D. Habib, and M. Sonza Reorda, "A reliability evaluation flow for assessing the impact of permanent hardware faults on integer arithmetic circuits," *IEEE Access*, vol. 13, pp. 32177–32196, 2025.

[62] J. E. R. Condia. (2024). *Tensor Core Unit*. [Online]. Available: https://github.com/Jerc007/TCcore

[63] J.-D. Guerrero-Balaguera and W. J. Perez-Holguin. (Nov. 2024). *Stereo Vision Core*. [Online]. Available: https://github.com/divadnauj-GB/stereovisioncore

[64] E. J. P. Nuñes, J.-D. Guerrero-Balaguera, and J. E. R. Condia. (2022). *Sfu - Piecewise Polynomial Approximation*. [Online]. Available: https://github.com/edwar-vhd/SFU-Piecewise-Polynomial-Approximation

[65] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Hoboken, NJ, USA: Wiley, 2023.

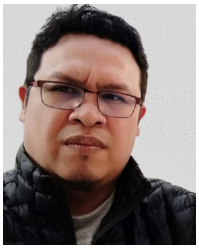
[66] W. S. Fife and J. K. Archibald, "Improved census transforms for resource-optimized stereo vision," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 60–73, Jan. 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6213095
[67] J. E. R. Condia, J.-D. Guerrero-Balaguera, C.-F. Moreno-Manrique, and M. S. Reorda, "Design and verification of an open-source SFU model for GPGPUs," in *Proc. Biennial Baltic Electron. Conf.*, 2020, pp. 1–6.
[68] J.-A. Pineiro, S. F. Oberman, J.-M. Müller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 304–318, Mar. 2005.
[69] J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. S. Reorda, "On the functional test of special function units in GPUs," in *Proc. 24th Int. Symp. Design Diagnostics Electron. Circuits Syst.*, Apr. 2021, pp. 81–86.
[70] M. T. Khan, H. E. Yantır, K. N. Salama, and A. M. Eltawil, "Architectural trade-off analysis for accelerating lstm network using radix-r obc scheme," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 1, pp. 266–279, Jan. 2023.
[71] M. A. Alhartomi, M. T. Khan, S. Alzahrani, A. Alzahmi, R. A. Shaik, J. Hazarika, R. Alsulami, A. Alotaibi, and M. Al-Harthi, "Low-area and low-power VLSI architectures for long short-term memory networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 4, pp. 1000–1014, Dec. 2023.

JUAN-DAVID GUERRERO-BALAGUERA (Member, IEEE) received the M.Sc. degree in electronics engineering from the Universidad Pedagogica y Tecnologica de Colombia (UPTC), Sogamoso, Colombia, in 2017, and the Ph.D. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2024. He is currently a Postdoctoral Research Fellow with the Politecnico di Torino. His research interests include digital design, fault-tolerant AI hardware, functional tests, artificial intelligence, parallel architectures, reconfigurable computing, and FPGAs.

ROBERT LIMAS-SIERRA (Member, IEEE) received the B.S. and M.Sc. degrees in electronic engineering from the Universidad Pedagógica y Tecnológica de Colombia (UPTC), Colombia, in 2019 and 2022, respectively. He is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy. His research interests include system functional testing, parallel architectures, GPGPUs, and hardware accelerators.

OGUZ SENSOZ received the B.S. degree in electronic engineering from Sabanci University, Istanbul, Turkey, in 2020, and the M.Sc. degree in embedded systems from the Politecnico di Torino, Turin, Italy, in 2025. His research interests include digital hardware design, FPGA development, embedded systems, fault tolerance, and hardware verification.

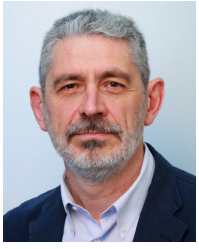
JOSIE E. RODRIGUEZ CONDIA (Member, IEEE) received the M.Sc. degree in electronics engineering from the Universidad Pedagogica y Tecnologica de Colombia (UPTC), Sogamoso, Colombia, in 2017, and the Ph.D. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2021. He is currently an Assistant professor with the Politecnico di Torino. His research interests include functional testing, parallel architectures, graphics processing Units, and embedded system design.



MAYNOR GIOVANNI BALLINA ESCOBAR (Student Member, IEEE) received the B.S. degree in electronic engineering from the Universidad de San Carlos de Guatemala (USAC), in 2019. He is currently pursuing the Ph.D. degree in collaboration with the International Centre for Theoretical Physics (ICTP), University of Trieste, Italy. His research interests include heterogeneous computing in MPSoC-FPGA systems and scientific instrumentation. He is also involved in the development of instrumentation for particle detectors with ECFM-USAC.

MARIA LIZ CRESPO received the Ph.D. degree in computer science from the National University of San Luis (UNSL), San Luis, Argentina, in 2004, with her thesis developed with The Abdus Salam International Centre for Theoretical Physics (ICTP) in collaboration with Italian National Institute of Nuclear Physics (INFN), Trieste, Italy. She is currently a Research Scientist and a Faculty Member with ICTP, where she is currently the Scientific Coordinator of the MLaboratory, Science, Technology, and Innovation (STI) Unit. She is also an Associate Researcher with INFN and a member of the COMPASS/AMBER collaboration with CERN. Her research focuses on advanced scientific instrumentation and methods for particle physics experiments, nuclear applications, supercomputing, and interdisciplinary experimental projects.

SERGIO CARRATO received the master's degree in electronic engineering and the Ph.D. degree in signal processing from the University of Trieste, Trieste, Italy. Then, he was with Ansaldo Componenti and Sincrotrone Trieste in the field of electronic instrumentation for applied physics. He joined the Department of Electronics, University of Trieste, where he is currently an Associate Professor in electronic devices.



MATTEO SONZA REORDA (Fellow, IEEE) received the M.Sc. degree in electronics and the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, in 1986 and 1990, respectively. He is currently a Full Professor with the Department of Control and Computer Engineering, Politecnico di Torino. He published more than 400 papers in the area of test and fault-tolerant design of reliable circuits and systems. He is involved in numerous research projects with companies and other research centers worldwide. He received several best paper awards at major international conferences.