FIawase: A SET Fault Injection Framework Towards Exhaustive System-Level Impact Evaluation

Mingtao Zhang, Quan Cheng, Masanori Hashimoto*
Graduate School of Informatics, Kyoto University, Japan
* Email: hashimoto@i.kyoto-u.ac.jp

***Abstract*—Single-event transients (SETs) threaten modern reliability-demanding SoCs equipped with error correction codes (ECC) for single event upset (SEU) mitigation. However, conventional gate-level SET fault injection (FI) remains prohibitively slow for practical reliability evaluation. This work presents FIawase, a high-throughput SET injection framework that enables comprehensive system-level evaluation of SET-induced soft errors. FIawase consists of two phases: a netlist-level inject-and-capture simulation, which systematically flips the output of every gate–cycle pair during program execution to record flip-flop changes one cycle later, and a scan-chain-based replay-and-measure emulation, which replays these patterns at hardware speed to quantify system-level impact. Implemented on an open-source RISC-V system, FIawase reduces a comprehensive SET injection campaign from decades of pure simulation to nearly a day, achieving over four orders of magnitude end-to-end speedup. FIawase takes a critical first step toward exhaustive, cycle-accurate SET analysis, enabling architectural and reliability research at previously infeasible scales.**

***Index Terms*—Reliability, fault injection, soft errors, single event transients**

I. Introduction

Every second, invisible cosmic-ray particles slice through the chips in our cars and satellites. When just one of these particles flips a single bit, software can crash and safety-critical systems can make life-threatening decisions [1]. Radiation-induced faults are typically classified into two types: bitflips on storage elements, known as *Single-Event Upsets (SEUs)*, and voltage glitches on logic gates, known as *Single-Event Transients (SETs)* [2]. If an SET propagates through combinational logic and is captured by one or multiple flip-flops (FFs), it can flip the stored values—a phenomenon referred to as *SET-induced SEU* [3]. While SEUs have been extensively studied and mitigated, SETs have received considerably less attention as the probability of a single glitch being latched is typically low [4]. However, their impact cannot be overlooked in systems where all FFs are protected against conventional SEUs [5], as these hardened FFs can still latch SET-induced errors indistinguishably from valid data. As node capacitance decreases and clock frequencies increase, susceptibility to SET-induced faults continues to rise [6], [7], underscoring the need to revisit SET analysis with renewed urgency.

Radiation testing provides accurate reliability assessment but is often impractical due to high cost, limited facility access, and long turnaround times [8]–[10]. This challenge is further amplified in the case of SET evaluation, as their low latching probability necessitates extensive irradiation to observe sufficient fault events. As a result, fault injection (FI) is expected to be an alternative, simulating radiation-induced faults through controlled bitflips. It plays a vital role in validating system behavior under faults and guiding protection strategies in fault-tolerant design [11]. Among various FI methods, finer granularity (from program-level to flip-flop-level, and gate-level) yields more accurate modeling of radiation effects [12]. While gate-level FI can reproduce SET propagation in sequential circuits [13], its high runtime overhead have constrained its use in large-scale designs.

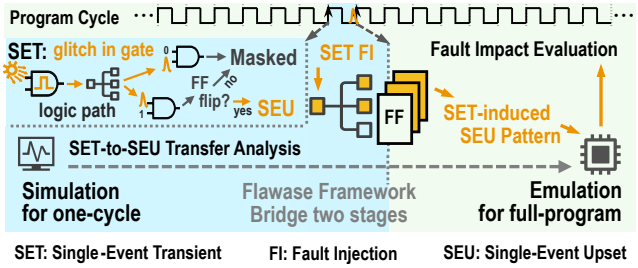

Fig. 1. Concept of Flawase: Bridging Single-Cycle SET Simulation with Full-Program SEU Emulation.

Most existing SET studies focus only on the likelihood that a glitch propagates through logic and is latched by FFs [14]. This process is referred to as *SET-to-SEU transfer* in this work. Although Monte Carlo [15] and probabilistic error propagation [16] methods can accelerate this process, they typically assume independent input distributions rather than realistic workloads. Moreover, their probabilistic outputs make system-level impact evaluations less reliable due to a lack of determinism. Simulation-based SET analysis remains the most accurate, but is prohibitively slow for system-level designs with realistic workloads. Consequently, system designers still lack and eagerly demand a practical and scalable SET-aware reliability analysis framework that satisfies two essential requirements: 1) the ability to inject faults at arbitrary logic gates and at any program cycle; and 2) the ability to exhaustively explore gate-cycle combinations and evaluate their system-level impact within a feasible runtime.

To address this gap, we propose *FIawase*: a high-throughput SET Fault Injection frAmework toWards exhAustive System-level fault impact Evaluation. As illustrated in Fig. 1, FIawase connects single-cycle netlist-level simulation, which extracts SET-induced SEU patterns, with high-speed emulation, which evaluates system-level impact over long-cycle program execu-

tion under the same workload. To our knowledge, FIawase is the first comprehensive SET FI framework capable of scaling to SoC-level designs while maintaining realistic workload contexts. Regarding the hardware emulation, we extend the scan-chain-based SEU injection technique from HachiFI [17] with our SET-aware control logic, and demonstrate the framework on an open-source RISC-V core [18] as the device under test (DUT). Compared to the HachiFI baseline, FIawase incorporates targeted optimization and acceleration strategies, achieving over $18\times$ improvement in SET FI throughput. For exhaustive FI campaigns, FIawase reduces the total runtime from several decades of pure simulation to just tens of hours.

The key contributions of this work are as follows:

- We propose a unified FI framework that includes SET-to-SEU transfer extraction and hardware-based SEU injection, bridging them via a shared design and workload.
- We optimize both simulation and emulation stages to enable fast, automatically-repeating FI execution.
- We demonstrate comprehensive SET FI on RISC-V system under realistic, program-driven conditions.

II. Related Work and Requirements for Comprehensive SET Analysis

Existing FI works typically focus on either SET analysis or SEU impact evaluation, lacking an end-to-end understanding of how SETs affect system behavior. We briefly review representative techniques for each stage and outline the requirements for bridging them within a unified framework.

A. SET analysis

Accurate SET modeling requires accounting for three masking effects: logic, electrical, and timing masking. Among them, logic masking has the greatest impact on analysis accuracy [19] and is the most computationally intensive, as it depends on specific input vectors that activate propagation paths. In contrast, electrical and timing masking can be partially approximated as attenuation ratios with comparatively smaller influence [20]. Recent works adopt signal-probability propagation techniques [21], [22], but their accuracy is significantly degraded by signal correlation, which frequently occurs in large netlists synthesized by commercial tools [23]. Additionally, these techniques often neglect logic dependencies caused by multi-bit correlated inputs, further reducing their effectiveness in large-scale designs with realistic workloads.

B. SEU evaluation

The second stage typically involves injecting faults into FFs and running the full program to observe system-level effects [24]. [25] perform SEU FI entirely in simulation, which becomes prohibitively time-consuming for large system; [26] use local simulations to estimate error rates but lack concrete fault impact; [27] inject faults on real silicon to measure persistent-SEUs but cannot capture run-time transient effects.

HachiFI [17] introduces a hardware-assisted SEU FI platform to address these limitations. It inserts a lightweight scan-chain control module to enable targeted bit flips during scan shift operations. FI parameters, such as the target FF index, are communicated via OpenOCD commands through the JTAG interface. By leveraging hardware emulation, HachiFI achieves speedups of several orders of magnitude over simulation-based SEU evaluation. However, HachiFI is not directly applicable to gate-level SET injection. Combinational logic gates cannot be instrumented with scan chains, and even if such instrumentation were possible, the required hardware overhead and control complexity would be prohibitive. Furthermore, HachiFI is designed for isolated, randomly injected SEUs, which do not reflect realistic SET propagation behaviors.

C. Requirements for Bridging SET and SEU

As mentioned above, SET-to-SEU transfer analysis, typically performed in simulation, and system-level SEU impact evaluation, often accelerated by hardware emulation, have traditionally been studied separately. However, to accurately assess the impact of SETs on system-level reliability, it is essential to bridge these two domains and develop a unified framework capable of comprehensively evaluating SET effects across both stages, as illustrated in Fig. 1.

Such a framework must ensure consistency in design, workload, and the spatial and temporal locations of FI between simulation and emulation. Moreover, comprehensive SET FI must cover a large number of gate-cycle combinations. Each logic gate at each clock cycle is a potential SET site, and every such event must be emulated to evaluate its system-level effect. As shown later, SEU evaluation on the emulator takes significantly more time than SET-to-SEU transfer analysis. To make this tractable, both the number of SEU injections (N_{SEUFI}) and the runtime per injection (T_{SEUFI}), which includes program execution and FI latency, must be minimized.

The key requirements for an effective SET FI framework are as follows:

- **R1** *Reduced SEU FI:* N_{SEUFI} must be reduced by filtering SEU patterns from SET-to-SEU transfer analysis.
- **R2** *Precise FI Control:* The hardware must support FI at precise cycles and specific flip-flop indexes defined by each SEU pattern.
- **R3** *Reliable FI Execution:* The FI mechanism must be robust against system crashes or JTAG errors to allow fault injection to proceed without host intervention.
- **R4** *Low Injection Latency:* Optimizing the FI process and communication overhead can reduce T_{SEUFI} by over 4×.

While HachiFI [17] provides a practical SEU FI interface, it does not satisfy the above requirements, and a substantial extension is needed. FIawase is designed to meet these requirements and enable fast, comprehensive SET fault injection, as described in detail in Section III.

III. Proposed FI Framework

A. Overall Framework

Fig. 2 illustrates the overall FIawase framework. Yellow boxes represent user-defined inputs, including the target DUT, test program, and SET model. Red boxes denote components generated during the flow, and gray boxes are executed using

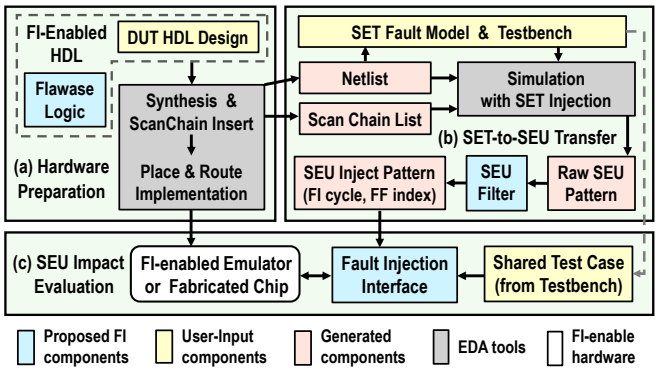

Fig. 2. Overall FIawase framework: (a) HDL is enhanced with FI modules (Section III-B); (b) SET simulation produces SEU patterns (Section III-C); (c) SEU patterns are mapped for impact evaluation (Section III-D).

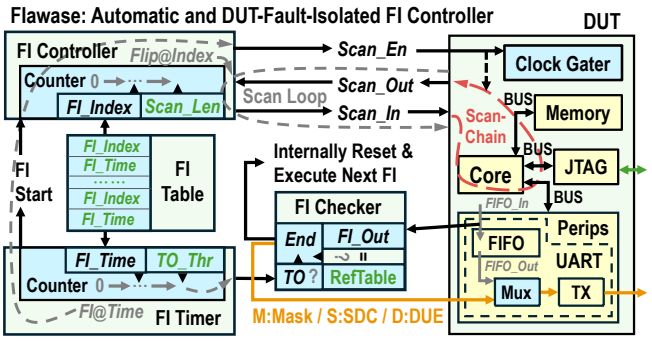

Fig. 3. FIawase HDL architecture enabling automatic fault injection and decoupling from DUT execution after initial load.

standard EDA tools such as DC and VCS. With the proposed components highlighted in blue, spanning all three stages and operating on the same design and test program, the framework effectively addresses **R1**-**R4**.

Stage (a) extends the user's HDL design with FIawase logic blocks that enable FI functionality. The augmented design is then processed through standard front-end and back-end flows with scan-chain insertion, to generate the netlist and ultimately an FI-enabled emulator or fabricated chip. During scan-chain insertion, untargeted modules such as SRAM and the JTAG debug interface must be marked with `set_dont_touch` and protected by clock-gated logic driven by the scan enable signal to avoid unintended functional disturbances caused by shift activity.

Stage (b) simulates SET faults by flipping logic wires in the netlist according to a user-defined SET fault model (e.g., pulse width, polarity, and masking effects), and extracts the resulting SEU patterns, including the injection cycle and affected FF indexes. An SEU pattern filter is applied to discard logic masked results or redundant identical FF bitflips, thereby reducing the number of SET-induced SEUs to be evaluated on hardware emulator, i.e., N_{SEUSI} **(R1)**.

In stage (c), the filtered N_{SEUSI} SEU patterns are injected into hardware via a dedicated FI interface (e.g., JTAG). FIawase bridges simulation and hardware by ensuring that stage (b) and stage (c) use the same test program and a netlist that structurally match each other at RTL **(R2)**. This consistency ensures that fault sites are structurally aligned and that program execution timing remains consistent across both simulation and hardware phases.

B. Hardware Preparation and Operation with FIawase Logic

The original HachiFI operates in an interactive JTAG mode, where each fault is injected via several host commands. However, this setup is fragile as an injected fault may crash the DUT or corrupt the communication link, halting the FI campaign and necessitating the DUT reboot. The DUT reboot significantly degrades the FI throughput. To improve robustness for high-throughput FI (**R3, R4**), FIawase adds an *Auto Mode* that requires host involvement only once for initialization.

Fig. 3 illustrates the additional hardware modules introduced by FIawase to support automatic and fault-isolated FI in Auto Mode. The four main blocks—FI Controller, FI Timer, FI Table, and FI Checker—are implemented outside the DUT, with key components enclosed in blue boxes. Inside the DUT, two lightly modified components (Clock Gater and Multiplexer, also in blue) support FI operations. Gray arrows indicate data/control flow to aid understanding.

1) FI initialization: During initialization, parameters shown in green in Fig. 3, such as the scan chain length (Scan_Len) and timeout threshold (TO_Thr) are latched into the FI Controller and FI Timer, respectively. A batch of fault schedules, including the target FF indexes (FI_Index) and injection cycle (FI_Time), is loaded into an on-chip FI Table via JTAG. In case of FPGA emulation, an affordable large FI table is recommended since it reduced overall runtime. After initialization, the system clock in the JTAG block is gated off during Auto Mode and re-enabled only after the current batch finishes, which prevents injected faults from disturbing the debug link.

2) FI operation: FI operation is controlled by two counters that coordinate fault injection in both time and space. The cycle counter in FI Timer advances with the DUT clock and is responsible for triggering the FI operation at the specified injection cycle (FI_Time) and for detecting timeouts caused by DUT hangs at TO_Thr. The latter functionality will be described in detail in the next section. The scan counter in FI Controller increments during scan operations to locate the target flip-flop before the fault is activated. Since both counters operate independently of the DUT's functional logic, they are unaffected by injected faults, ensuring reliable FI scheduling and timeout detection.

When the cycle counter reaches the scheduled injection time (FI_Time), the FI Controller asserts the Scan_En signal to activate the scan loop across all scan-chain-inserted FFs. During this loop, Scan_Out is routed through FI Controller and fed into Scan_In. If the scan counter matches FI_Index, the controller inverts Scan_Out before feeding it back; otherwise, the data is passed through unchanged. Considering that SETs can result in spatially clustered multi-bit SEUs [16],

FIawase supports injecting multiple faults by specifying several FI_Index entries within a single scan loop. To maintain program consistency, functional blocks not included in the scan chain are clock-gated during the scan loop. Once the scan completes, the original register state is restored except for the injected fault, and program execution resumes seamlessly from the same cycle.

3) FI result analysis and reporting: FIawase in this paper assumes that the DUT program reports execution results via UART, whereas other interfaces are also usable. To enable on-chip fault classification, the DUT first runs the test program once without FI and outside of Auto Mode. During this run, FIawase monitors all UART FIFO input to capture the expected output and stores it in a reference table (RefTable). In each fault-injected run, the FI Checker compares the DUT's FIFO writes against this RefTable. Based on this comparison, each FI is categorized as one of the following outcomes: *Mask*: fault has no observable effect on program output; *SDC* (Silent Data Corruption): program complete but produces incorrect output; or *DUE* (Detected Unrecoverable Error): program execution did not complete before the timeout threshold. Since the classification is performed on-chip, there is no need to wait for full UART transmission to the host, which reduces FI runtime (T_{SEUFI}). In Auto Mode, UART TX outputs one ASCII verdict character (*M*, *S*, or *D*) via a multiplexer.

Once the classification result transmission is complete, the FI Checker asserts an internal reset signal. This acts as a stop-and-restart mechanism, resetting only the DUT and counters while preserving FI control state. It enables the next fault to be loaded from FI Table and injected without host involvement. Through this structure, FIawase ensures long FI campaigns remain resilient and fully automatic, even if injected faults crash the DUT or disrupt standard interfaces. Since FIawase relies only on standard scan chains, JTAG, and UART output, and requires only minimal modification to functional HDL, it is broadly applicable to a wide range of system designs.

C. SET-to-SEU Transfer Analysis

Users are expected to provide application workloads that capture representative system behavior and are reused in both simulation and emulation. The same netlist with scan chains inserted is also used in both domains. For FPGA emulation, this netlist is treated as RTL by rewriting the standard-cell library into equivalent Boolean logic compatible with LUT-based synthesis. Although the FPGA implementation differs physically, it preserves the same logic structure, flip-flop count, and scan-chain configuration as in simulation, ensuring functional and temporal alignment between the two platforms.

In this paper, we focus solely on the worst-case SET scenario, where the SET pulse is assumed to be wide enough to be latched by FFs unless logically masked. In other words, we neglect electrical and temporal masking and consider only logic masking. This assumption eliminates the need for detailed timing or electrical-level information. Although SET pulse width can be modeled in logic simulation, whether an SET is captured follows a stochastic distribution, as both its timing and width vary randomly. Such probabilistic analysis is therefore not well suited to the objective of exhaustive FI.

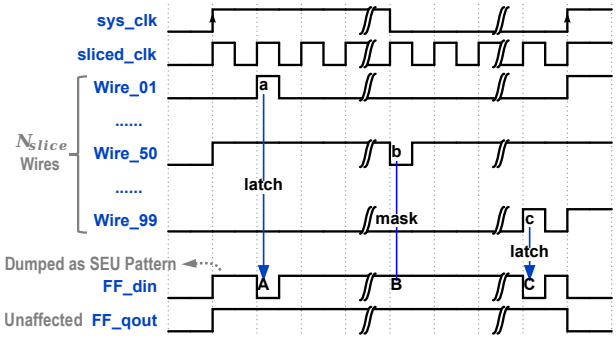

Fig. 4. Efficient simulation-based analysis of SET-to-SEU transfer.

For SET-to-SEU transfer analysis, only one-cycle propagation needs to be observed. Compared to conventional approaches that independently and repeatedly feed input vectors to evaluate bit propagation along the same logic paths, we adopt a time-sliced simulation strategy that minimizes input-handling overhead while fully leveraging commercial EDA tools. During simulation, a sliced clock (sliced_clk) is generated in the testbench to divide each system clock (sys_clk) cycle into N_{slice} finer-grained segments, enabling sub-cycle SET injection. Fig. 4 illustrates a configuration with $N_{slice} = 100$ time slices per sys_clk cycle. Within each time segment, one output wire of a logic gate is toggled to simulate the effect of a transient fault, while all FF inputs (FF_din) are monitored to detect SET-to-SEU propagation. To preserve correct program behavior, the first and last segments near the rising edge of sys_clk are left untouched. This ensures that FFs latch only unmodified data, and the FF output (FF_qout) remains unchanged throughout the fault evaluation. By maintaining natural timing at latching points, this approach achieves accurate observation of SET effects without disrupting program execution.

In this setup, each program run can evaluate N_{slice} wires per cycle across all program cycles N_{cycle}. Given N_{wire} total wires and a simulation time of T_{prog} per run, the total simulation cost is reduced to approximately $(N_{wire}/N_{slice}) \times T_{prog}$. This strategy achieves over $6000\times$ speedup compared to a Python-based one-cycle propagation analyzer, and reduces naive simulation runs by a factor proportional to the program's cycle count, eliminating simulation as a performance bottleneck.

The SET-induced bitflips in FF_din and the corresponding sys_clk cycle are recorded during simulation, and a Python script filters them by removing logically masked faults (i.e., those not captured by any flip-flop) and redundant patterns (i.e., different gates that induce the same SEU pattern in the same clock cycle). This filtering effectively reduces the number of SEU patterns while preserving the overall coverage of distinct SET behaviors.

D. SEU Impact Evaluation

The final output of Stage (b) is a refined set of SEU injection patterns $\mathcal{P}$, each pattern consisting of an injection

cycle and corresponding FF indexes. These patterns reproduce meaningful SET-to-SEU propagation effects and guide FI on hardware during Stage (c). In this stage, the host communicates with the emulator via JTAG using OpenOCD [28] commands issued through Tcl scripts, enabling DUT halt, program loading, and FI configuration. The injection campaign starts with a golden run that collects a reference output, which is stored in RefTable for later comparison. Subsequently, FI proceeds in batches, each autonomously initiated by enabling Auto Mode. In every batch $\mathcal{B}$, the host selects a subset of SEU patterns $\mathcal{B} \subset \mathcal{P}$ and loads them into the on-chip FI Table. Each pattern defines a fault configuration, consisting of FI_Index and FI_Time, where FI_Index specifies one or more fault locations. These configurations enable injection of SET-induced multi-bit SEUs at the specified FI_Time. During this process, batch-level parameters such as Scan_Len and TO_Thr are also configured and kept consistent across batches. While the DUT executes the batch in Auto Mode, the host remains passive, only monitoring UART messages sent by the on-chip FI Checker. For each injected pattern, a single verdict is returned: *M*, *S*, or *D*. Once the FI schedule in the FI Table has been fully read out and all verdicts for the current batch have been received, the system exits Auto Mode and returns to interactive mode. The host then halts the DUT, loads the next batch, and resumes execution.

IV. Experimental Results

A. Experimental Setup

Platform: Experiments are conducted on the open-source tinyriscv SoC [18], used as the DUT and integrated with FIawase logic for FI support. To accurately reflect physical implementation effects, the DUT is synthesized and placed-and-routed using the open-source SkyWater 130 nm PDK. During synthesis, scan chains are inserted into all flip-flops within the DUT, except those located in memory blocks, within the JTAG or UART interfaces, or in the FIawase control logic. These excluded modules are clock-gated during scan operations to prevent unintended behavior caused by shift activity. This configuration enables fault accessibility representative of the DUT's combinational logic while avoiding disturbance to critical control paths. The resulting post-layout netlist is mapped to a Digilent Genesys-2 FPGA board (Xilinx Kintex-7 XC7K325T) operating at a 25 MHz system clock.

Workloads: We evaluate two representative workloads with contrasting execution characteristics: (i) *Hash*: a compute-intensive FNV-1a hash function; (ii) *BubbleSort*: a control-flow-heavy algorithm with frequent branching. Each program outputs its final result via UART. In fault-free runs, both workloads complete in approximately 60 k CPU cycles, enabling exhaustive SET FI within practical time constraints.

Timing parameters: In this experiment, the inserted scan chain includes 5,024 flip-flops, resulting in approximately 5,000 system clock cycles per scan operation. UART transmission introduces an additional latency of around 4,000 system cycles per transmitted ASCII byte due to serial I/O overhead. Furthermore, a basic JTAG command issued from the host requires approximately 3,000 system cycles to propagate and take effect within the emulator.

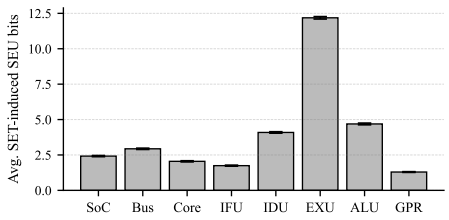

Fig. 5. SET-induced bitflip statistics for Hash and Sort.

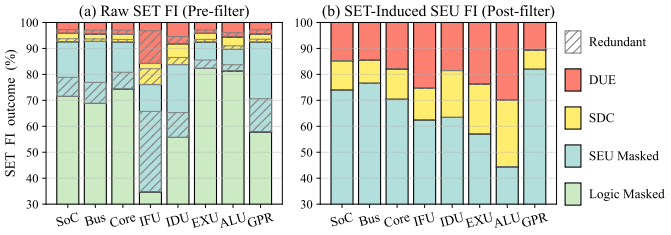

Fig. 6. SET Fault Outcome classification for Hash.

B. System-level Fault Evaluation

Multi-bit Flip Statistics: SET-induced bitflips often result in multiple simultaneous SEUs due to glitch propagation along wide datapaths. Fig. 5 presents the average number of flipped bits per SET across various microarchitectural modules, aggregated over the Hash and Sort workloads. These modules are actively exercised by the workloads and represent typical hotspots in SoC execution. Notably, the EXU module shows the widest SEU spread, with an average of 12.2 flipped bits per event, likely due to its extensive connectivity and central role in computation. In contrast, the SoC-wide average is approximately 2.4 bits. This highlights the non-uniform distribution of SET-induced multi-bit effects, with datapath-intensive modules exhibiting significantly higher susceptibility.

Module-level vulnerability: Fig. 6 shows the SET fault evaluation outcomes across different microarchitectural modules under the Hash workload. The left plot (a) presents the raw SET FI results, where all gate–cycle pairs are considered. It includes both logic-masked and redundant patterns, which together account for approximately 80% of all injected faults, demonstrating the effectiveness of the pre-filtering process. The right plot (b) shows the FI outcomes of the remaining SET-induced SEU patterns after filtering. Although redundant patterns are not re-injected, their counts are proportionally reflected once the corresponding injected pattern produces an outcome.

Notably, the IFU exhibits the lowest masking rate but the highest portion of identical SEU patterns, likely due to its relatively simple combinational logic. At the same time, it contributes to a higher-than-average rate of DUEs, as faults in this module can quickly affect control flow. In contrast, datapath-heavy modules such as EXU and ALU exhibit higher masking rates, but faults that escape masking in these modules have a greater likelihood of causing harmful outcomes (SDC

TABLE I
PER-OPTIMIZATION SPEEDUP COMPARED TO HACHIFI

#	Technique	Baseline	Speed-up Gain
1	Pattern Filter	Inject all fault patterns	4.5–5.3×
2	1-Byte UART	Full output	3.2–3.4×
3	Fault Batching	One setup per injection	1.4–1.5×
4	Crash Recovery	Manual global reset	$> 60.6\times$[a]
Total (w/o Crash)		$\sim$ 4.4ms per injection	$\sim 18\times$
Total (w/ Crash)		> 0.34s per injection[a]	$> 655\times$[a]

[a] Manual crash recovery leads to unpredictable and unbounded delays.

TABLE II
TIME NECESSARY FOR EXHAUSTIVE SET INJECTION (HASH).

	Method	Time and Speed-up
SET-to-SEU Transfer	FIawase	1h (VCS) + 6h (script)
1 Program Run with SET FI	Pure Sim.	2.12 s
	FIawase	1.19 ms (**1782x**)
Exhaustive SET FI	Pure Sim.	$\sim$61.5 years
	FIawase	35.66 h (**15106x**)

Note: DUT has 15219 wires and 60035 cycles for Hash workload.

or DUE). However, this trend does not align directly with the bit-flip magnitude shown in Fig. 5, suggesting that the system-level impact of SETs cannot be inferred from low-level metrics alone. This underscores the necessity of performing system-level FI rather than relying solely on gate-level glitch modeling. FIawase enables such end-to-end analysis efficiently, allowing rapid identification of vulnerable modules under realistic fault conditions and providing concrete guidance for targeted resilience strategies.

C. Throughput and Robustness

Baseline specify: While the core contribution of FIawase is bridging netlist-level SET-to-SEU transfer and hardware emulation for exhaustive system-level fault evaluation, several optimisations have also been introduced to improve both speed and reliability compared to the HachiFI baseline [17]. In this experiment, HachiFI runs 60 k-cycle test programs and issues ten OpenOCD commands to configure the fault site (10 × 3k cycles per FI). Each SET induces 2.4 SEUs on average as observed in our experiments, resulting in 19k cycles of overhead for JTAG writes and scan loops (2.4 × (3k + 5k)). As a result, each injection consumes about 109 k cycles, or 4.36 ms on a 25 MHz emulator. Furthermore, injected faults crash the system in 15% of runs as shown in Fig. 6. Assuming a 2-second manual global reset, constant supervision, and no idle time, the reset overhead increases the average injection time by roughly 69× (i.e., 15% × 2 s ÷ 4.36 ms).

Speed Improvements over Baseline: FIawase addresses these issues through the following optimisations: *(i)* An SEU filter in the simulation stage discards 80% of fault patterns that are logic-masked or redundant, avoiding unnecessary emulation; *(ii)* By skipping full program output over UART and using a single-byte result flag, the test program length is reduced from 60 k to 18 k cycles; *(iii)* Faults are loaded in batches of 256, amortizing the 30 k-cycle setup overhead to approximately 0.1 k cycles per injection; *(iv)* Finally, a crash recovery timer resets the core automatically after 218 k cycles, which is twice the baseline injection time. This replaces the manual global reset and reduces the reset overhead from 69× down to just 0.3× (15% × 2). Together, these optimisations significantly improve overall FI throughput, as shown in Table I, achieving a speedup of approximately 18× in non-crashing cases, and over 655× when accounting for crash-induced reset overhead.

Realtime for Exhaustive Evaluation: Table II summarizes the end-to-end speedup achieved by FIawase on the Hash workload. Specifically, 15,219 logic wires in the DUT are involved in the injection campaign, and the Hash program spans 60,035 cycles.

In the SET-to-SEU transfer stage, one FIawase simulation run processes 100 wires over a full program execution in 19 seconds using Synopsys VCS, with an additional 2 minutes for post-processing and SEU pattern filtering. Given this setup, the complete SET-to-SEU transfer across all wires completes in approximately 7 hours. As a result, simulation is no longer the performance bottleneck. In comparison, a Python-based SET propagation analyzer takes approximately 0.02 seconds per gate-cycle pair, leading to a total estimated runtime of over two years. This translates to a 6,316× speedup with FIawase when excluding script processing time.

In the emulation stage, a single program run with one injected fault takes only 1.19ms on FPGA, providing a 1,782× speedup compared to 2.12s in netlist-level simulation. Together with the proposed optimization strategies, the total time for an exhaustive SET FI campaign is reduced from more than 61 years using pure simulation to just 36 hours. This results in an overall speedup of 15,106×. In summary, FIawase transforms exhaustive SET FI at the system level from an impractical long-term task into a process that completes in just 36 hours. These results validate the practicality of FIawase for exhaustive fault analysis on SoCs under realistic workloads.

V. CONCLUSION

In this work, we present FIawase, a high-throughput SET fault injection framework that, to our knowledge, is the first to enable scalable, practically exhaustive fault impact evaluation by bridging simulation-based SET-to-SEU transfer with emulation. By filtering unnecessary full-program runs on the emulator and executing the FI campaign automatically and reliably at runtime, FIawase enables efficient and comprehensive fault analysis at the system level. Our implementation achieves over 15,000× speedup compared to traditional pure simulation, reducing an otherwise 61-year exhaustive campaign to just 36 hours. FIawase establishes a practical foundation for future SET FI research, enabling integration with advanced SET modeling or acceleration techniques at the SoC scale.

REFERENCES

[1] T. Tanaka, et al. Impact of Neutron-Induced SEU in FPGA CRAM on Image-Based Lane Tracking for Autonomous Driving: From Bit Upset to SEFI and Erroneous Behavior. *IEEE Transactions on Nuclear Science*, vol. 69, no. 1, pp. 35-42, Jan. 2022.

[2] D. Munteanu, et al. Modeling and Simulation of Single-Event Effects in Digital Devices and ICs. *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 1854-1878, Aug. 2008.

[3] D. M. Fleetwood. Radiation Effects in a Post-Moore World. *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 509-545, May 2021.

[4] M. Ebrahimi, et al. Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, pp. 1-6.

[5] M. Hashimoto and W. Liao. Soft Error and Its Countermeasures in Terrestrial Environment. In Asia and South Pacific Design Automation Conference (ASP-DAC), 2020, pp. 617-622.

[6] L. Artola, et al. Modeling Single Event Transients in Advanced Devices and ICs. *IEEE Transactions on Nuclear Science*, vol. 62, no. 4, pp. 1528-1539, Aug. 2015.

[7] P. S. Rajakumar, et al. A Comprehensive Review of Single Event Transients on Various MOS Devices. *IEEE Access*, vol. 12, pp. 154760-154777, 2024.

[8] L. M. Luza, et al. Emulating the Effects of Radiation-Induced Soft-Errors for the Reliability Assessment of Neural Networks. *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 4, pp. 1867-1882, 1 Oct.-Dec. 2022.

[9] Y. Li, et al. Experimental Study of Proton-Induced Radiation Effects on DDR5 Modules. *IEEE Transactions on Nuclear Science*, vol. 72, no. 6, pp. 1907-1918, June 2025.

[10] K. Takami, et al. Validating Terrestrial SER in 12-, 28- and 65-nm SRAMs Estimated by Simulation Coupled with One-Time Neutron Irradiation. *IEEE Transactions on Nuclear Science*, doi: 10.1109, 2025.

[11] R. Vadlamani, et al. Multicore soft error rate stabilization using adaptive dual modular redundancy. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010.

[12] H. Cho, et al. Quantitative evaluation of soft error injection techniques for robust system design. In ACM/IEEE Design Automation Conference (DAC), 2013, pp. 1-10.

[13] M. Kooli and G. Di Natale. A survey on simulation-based fault injection tools for complex systems. In International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2014, pp. 1-6.

[14] D. Holcomb, et al. Design as you see FIT: System-level soft error analysis of sequential circuits. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2009, pp. 785-790.

[15] B. Liu and L. Cai. Monte Carlo Reliability Model for Single-Event Transient on Combinational Circuits. *IEEE Transactions on Nuclear Science*, vol. 64, no. 12, pp. 2933-2937, Dec. 2017.

[16] Zhu Kexin, et al. Efficient Sublogic-Cone-Based Switching Activity Estimation using Correlation Factorn. In Asia and South Pacific Design Automation Conference (ASP-DAC), 2024, pp. 638–643.

[17] Q. Cheng, et al. HachiFI: A Lightweight SoC Architecture-Independent Fault-Injection Framework for SEU Impact Evaluation. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2025, pp. 1-7.

[18] K. Liang. tinyriscv: A simple implementation of RISC-V CPU in Verilog. Available: https://github.com/liangkangnan/tinyriscv.

[19] N. Mohyuddin, E. Pakbaznia and M. Pedram. Probabilistic error propagation in logic circuits using the Boolean difference calculus. In IEEE International Conference on Computer Design (ICCD), 2008, pp. 7-13.

[20] M. Anglada, et al. Fast and Accurate SER Estimation for Large Combinational Blocks in Early Stages of the Design. *IEEE Transactions on Sustainable Computing*, vol. 6, no. 3, pp. 427-440, 1 July-Sept. 2021.

[21] E. Esmaieli, et al. Fanout-Based Reliability Model for SER Estimation in Combinational Circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 1, pp. 228-240, Jan. 2025.

[22] Weihua Xiao and Weikang Qian. ASPPLN: Accelerated Symbolic Probability Propagation in Logic Network. In IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2022, pp. 1–9.

[23] Z. Shi, et al. Identifying Reliability High-Correlated Gates of Logic Circuits With Pearson Correlation Coefficient. *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 4, pp. 2319-2323, April 2024.

[24] S. S. Mukherjee, J. Emer and S. K. Reinhardt. The soft error problem: an architectural perspective. In High-Performance Computer Architecture (HPCA), 2005, pp. 243-247.

[25] R. Leveugle, et al. Statistical fault injection: Quantified error and confidence. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2009, pp. 502-506.

[26] S. Mirkhani, et al. Efficient soft error vulnerability estimation of complex designs. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 103-108.

[27] N. A. Harward, et al. Estimating Soft Processor Soft Error Sensitivity through Fault Injection. In Annual International Symposium on Field-Programmable Custom Computing Machines, 2015, pp. 143-150.

[28] Open On-Chip Debugger. OpenOCD: Open On-Chip Debugger. Available: http://openocd.org/