# Genshin: A Generalized Framework with Software-Hardware Co-design and Pruned Fault Injection for Reliability Analysis

Quan Cheng[1,2], Hao-Yang Chi[3], Chien-Hsing Liang[3], Yu-Hong Chao[3], Huizi Zhang[2], Yuan Liang[2], Mingtao Zhang[1], Wang Liao[4], Jinjun Xiong[5,*], Jing-jia Liou[3,*], Masanori Hashimoto[1,*] Longyang Lin[2,*]

[1]Department of Communications and Computer Engineering, Kyoto University, Kyoto, Japan
[2]School of Microelectronics, Southern University of Science and Technology, Shenzhen, China
[3]Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan
[4]School of Systems Engineering, Kochi University of Technology, Kochi, Japan
[5]Department of Computer Science and Engineering, University at Buffalo, USA
[*]{jinjun@buffalo.edu, jjliou@ee.nthu.edu.tw, hashimoto@i.kyoto-u.ac.jp, linly@sustech.edu.cn}

***Abstract*—Reliability-demanding devices often require numerous fault injections (FIs) for reliability analysis in the product cycle. However, software-based FI typically demonstrates extremely low efficiency due to low simulation throughput, especially for large-scale designs, while hardware-based FI presents challenges related to complexity of setup and limited scalability. Additionally, FIs often occur in intervals where errors do not affect the system's outcome, e.g., after final read before next write, necessitating efficient pruning of non-impactful FIs. To address this, a general-purpose FI-specialized framework, Genshin, is proposed for rapid reliability analysis. On the hardware side, we provide an FI-specialized design, which works with Design Under Test (DUT) chips on PCB boards and supports FI control based on the scan chain (SC). An integrated programmable logic allows for flexible and custom FI pattern definitions. Furthermore, an architecturally correct execution (ACE) analysis generates pruned fault tables for DUTs. In Genshin, the SC logic achieves 3,802-65,388 cycles/FI across SC lengths ranging from 2,795 to 61,393 in different DUTs, while the programmable logic enables custom error patterns such as layout-aware multi-bit upset (MBU). Furthermore, the pruned fault tables achieve fault reduction rates from 45.80% to 83.21%.**

***Index Terms*—fault injection, reliability analysis, architecturally correct execution, software-hardware co-design**

## I. Introduction

In reliability-demanding applications, such as aerospace and autonomous systems, the threat of radiation-induced failures (e.g., single-event upset (SEU)) presents significant challenges [1]–[3]. Soft error arises when charged particles strike sensitive areas of semiconductor devices, potentially causing transient faults, data corruption, or system malfunctions [4]–[6]. To analyze the reliability of systems, fault injection (FI) is a preferred and solid solution for assessing the reliability, offering a practical, controlled, and cost-effective means of simulating radiation conditions in a laboratory environment [7].

However, existing FI techniques face several critical limitations. Simulation-based FI, while highly flexible, is often constrained by long simulation times, making it inefficient, especially for large-scale or complex designs where billions of cycles may be required to model realistic fault scenarios [8], [9]. This low efficiency significantly delays the analysis process and limits its applicability for time-sensitive projects. Hardware-based FI techniques, on the other hand, offer faster FI and testing capabilities, but come with significant setup complexity, cost, and limited scalability, often requiring custom hardware configurations that are challenging to reconfigure for different test cases [10]–[12]. Besides, when a Design Under Test (DUT) chip is designed for final validation or production, it typically lacks FI-specific functions due to area cost and security concerns. Using a large-scale integration (LSI) tester for FI is often impractical due to the extended occupation of expensive resources. Similarly, while FI and logic analysis can be implemented on FPGAs, such platforms often offer an excessive amount of logic and routing resources relative to the simplicity of FI tasks, resulting in inefficient resource utilization and high reconfiguration overhead.

On ther other hand, most reliability-demanding designs focus on SRAM, leveraging mature mitigation techniques such as Error Correction Code (ECC). However, the reliability of other components, including state machines and control logic, remains largely overlooked. Ensuring the reliability of these components is critical, as their failures can also significantly impact overall system stability, and their reliability can be analyzed through information obtained from Flip-Flops (FFs). When incorporating error detection and/or recovery mechanisms into chips and systems, it is necessary to verify and validate them through FI, as these mechanisms are not activated during normal operation.

In addition, hardware-based FI methods generally struggle to achieve comprehensive coverage of possible fault scenarios, limiting their effectiveness in providing a thorough reliability assessment [10], [11]. Meanwhile, not all FIs can trigger errors. Only bit flips that occur within the architecturally correct execution (ACE) interval can generate errors [13],

Regular Paper

2378-2250/25/$31.00 ©2025 IEEE
DOI 10.1109/ITC58126.2025.00008

[14]. The ACE interval refers to the critical period during which changes in system data or state can disturb system operation, typically starting with a write event that is not immediately followed by another write and continuing until the last read before the next write. Conversely, the non-ACE interval spans from a read event to a write event and may include redundant writes in between. Bit-flips during the ACE interval may affect the final output, while errors outside this interval (non-ACE interval) do not affect the system because they are overwritten by the following write event. Therefore, any FI within the non-ACE interval has no impact on the system. In many systems, over 50% of the timing intervals result in non-ACE intervals, leading to redundant FIs [13], [14]. Efficiently pruning these non-impactful FIs is crucial for enhancing the overall FI throughput.

To address these challenges, we introduce **Genshin**, a **GEN**eralized framework with **S**oftware-**H**ardware co-design and pruned fault **IN**jection to enable rapid, scalable, and flexible SEU analysis via an efficient software-hardware co-design approach. On the hardware side, Genshin integrates an FI-specialized design that supports common protocols and provides flexible clock and scan chain (SC) control for DUT chips, offering enhanced efficiency in SC-based FIs with minimal PCB connections. In addition, Genshin includes a programmable logic component that enables custom error pattern definitions, including complex scenarios such as layout-aware MBUs, thereby achieving high adaptability to various SEU cases. This allows for wide coverage of fault types while maintaining operational speed and minimizing reconfiguration overhead. On the software side, Genshin employs ACE-based analysis to generate pruned fault tables, thereby effectively reducing unnecessary FIs. This optimization significantly decreases the number of required injections while preserving comprehensive analysis capability. Importantly, Genshin is designed for lab-based pre-deployment testing rather than in-field runtime FI, enabling designers to evaluate and improve reliability during early design validation stages. Consequently, Genshin not only increases FI efficiency but also enhances fault coverage, supporting a broader range of error patterns and injection scenarios without the high resource demands of traditional hardware-only methods. Highlights of this paper are:

- **Genshin Framework for Fault Injection**: Genshin is a general-purpose FI platform designed to facilitate SEU analysis across a broad range of systems. By integrating software-hardware co-design for both FI circuits and DUTs, Genshin enables rapid and efficient FI, leveraging SC control and programmable logic for defining flexible error patterns. The ACE-based pruning approach ensures high fault coverage while significantly reducing FIs, making it highly adaptable to various testing environments.
- **Implementation and Evaluation for Genshin**: To validate the Genshin framework, we implement it as an ASIC in a 180nm CMOS process. The FI experiments of three DUTs (two microprocessors and one AI accelerator) demonstrate Genshin's efficiency, flexibility, and adaptability for FI, underscoring its practical value for high-reliability applications requiring rigorous fault analysis.

## II. Related Work

Fault injection techniques allow researchers to systematically explore a system's response to various fault scenarios, assess its fault tolerance mechanisms, and identify potential weaknesses. However, traditional FI simulation techniques have notable limitations. A major issue with software-based FI is its effectiveness. Eris [8], a fast C/C++ RTL simulation FI framework, offers high coverage at a low cost, but is limited to instruction-level FIs for processor designs and requires long simulation cycles for large designs. Similarly, Cheng et al. proposed a software-based FI to analyze an AI SoC with full coverage via backdoor access [9]. However, backdoor access and long simulation cycles severely hinder FI efficiency, resulting in experiments that may take months or even years to complete. In terms of hardware-based FI, the main challenge lies in the trade-off between coverage and hardware overhead. TensorFI [10], a high-level FI framework for TensorFlow applications, cannot directly inject faults into low-level hardware, limiting its utility for hardware-specific testing. Similarly, Fiji-FIN [11], a FI framework for evaluating deep learning models on IoT devices, is restricted to injecting faults into memory cells and cannot target registers, thus limiting its coverage for testing DUTs' resilience. Moreover, a FPGA-based FI framework [12] was proposed to mix various numbers of injected faults across multiple modeling levels (i.e., software, register, and FF). However, it suffers from high time and resource consumption, and the results may not fully reflect the behavior of modern, complex processors due to its reliance on the LEON3 architecture. Furthermore, these frameworks lack a specific definition of error patterns tailored to the requirements of different DUTs.

Meanwhile, fault reduction methods are essential to minimize unnecessary FIs in system reliability evaluations, allowing resources to focus on impactful errors. By targeting critical paths and sensitive components, these methods improve test efficiency and reduce FI time and computational demands. However, most existing fault reduction frameworks rely solely on RTL simulations, which can be inaccurate in both large and complex systems. For example, Raasch et al. [13] introduced an innovative approach that extracts circuit node trees from RTL designs to improve the precision of ACE analysis, yet this method does not simulate the actual netlist, which can result in inaccuracies. Similarly, Yang et al. [14] proposed an ACE-based propagation graph approach (ACE-Pro) for fault reduction using RTL analysis, achieving fault reductions ranging from 98.91% to 99.91%. However, the ACE-Pro method relies on pre-simulation and analysis at the RTL level, which also cannot accurately reflect the actual physical designs.

Both software-based and hardware-based FI frameworks have their own advantages. Therefore, a co-design FI framework that combines both software and hardware, with low overhead and high efficiency, is essential. Additionally, it

is necessary to develop components capable of accurately modeling the faults in the DUT under real conditions, allowing pruned FI and further improving FI efficiency.

## III. Genshin Framework

As shown in Fig. 1, a generalized FI framework (Genshin) is introduced, incorporating a lightweight RISC-V core for FI in the standalone mode and JTAG logic for FI in the interactive user mode. In standalone mode, FI is conducted solely using the RISC-V core without external interaction. This mode is suitable for large-scale FI experiments, especially with many DUT chips in parallel. In interactive mode, a computer communicates with Genshin through JTAG, issuing Tcl commands to perform interactive human-in-the-loop FI. A clock and scan management unit (CSMU) and a programmable FI unit (PFIU) similar to a tiny FPGA are implemented for SC-based FI with configurable error patterns. In addition, an ACE-based fault table generator is implemented for efficient fault reduction during FI.

### A. Hardware Architecture

As shown in Fig. 1(a), the hardware architecture in Genshin is built with four key components: 1) A 3-stage RV32IM RISC-V processor for standalone FI (Section III-B3), equipped with 1KB data buffer, and an affluent set of peripherals such as UART, (Q)SPI, and IIC. 2) A JTAG module designed for debugging and interactive FI (Section III-B3), having the privilege to access all on-chip components. 3) A clock and scan management unit (Section III-A1), supporting the control of SC-incorporated DUTs for SC-based FI with efficient clock management. 4) A programmable FI unit (Section III-A2), providing flexible error pattern definitions for users. Beside, the RISC-V processor (w/o on-chip instruction buffer) runs on Flash Execute-In-Place (FlashXIP) mode, where code can be executed directly on the external flash memory mode. Meanwhile, the JTAG module is based on typical 0.13 draft of the RISC-V Debug Support Specification [15].

*1) Clock and Scan Management Unit:* Most commercial off-the-shelf (COTS) integrated circuits incorporate SCs as a design-for-testability (DFT) feature during manufacturing testing and internal debugging. Although these SCs are typically disabled in the final commercial products to prevent security risks and protect proprietary design information, they remain fully accessible during pre-silicon verification, production testing, and early bring-up stages, providing a valuable opportunity to apply our technique for accurate FI and reliability evaluation before deployment. Therefore, our framework exploits these SCs for FI. To enable efficient SC-based FI and support an ACE-based fault table, the CSMU is implemented to manage clock signals and SC control.

The main logic of the CSMU is shown in Fig. 2. To better illustrate the logic, handshake functions are omitted between two clock domains in the figure but are present in the actual design. The timing of FI is randomly specified or swept within a period of interest. When the specified timing does not fall within the ACE intervals at the FI position, Genshin will skip the FI operation to improve the efficiency of SEU analysis. When Genshin activates the FI operation ($FI\_EN$) and determines a FI time point ($FI\_Timing$), the target device continues normal operation up to that point based on Sys_CLK signal. During this normal operation phase, the CSMU gives the normal clock signal (Sys_CLK) to DUTs and keeps track of the cycles. Once the designated FI time ($FI\_Timing$) is reached, the SC function is triggered. Signals such as SC_EN, SC_IN and SC_OUT are activated. During this process, the CSMU switches its output clock to the scan clock (SC_CLK), performing FI cycles that match the length of the SC ($SC\_Length$). The position of the FF for FI is randomly selected or specified to particular FFs in a focused test. Before reaching the designated FI point ($FI\_Position$), SC_OUT is directly assigned to SC_IN, maintaining a seamless signal flow. Upon reaching the FI point ($FI\_Position$), SC_OUT is inverted and then reassigned to SC_IN, effectively completing a bit-flip operation. This inversion introduces the intended fault by altering the signal, allowing precise control over FI within the SC. After injection is complete, this one-time FI operation is completed. Besides, additional rounds can be repeated in subsequent tests.

*2) Programmable Fault Injection Unit:* To provide flexibility in reconfigurable FI error patterns, a lightweight FPGA-like PFIU is designed specifically for FI purposes, as shown in Fig. 3. The programmability owing to FPGA-like array is crucial to perform FI at the speed of SC_CLK, which simplifies the clocking and improves the FI throughput. For example, supposing a RISC-V is handling a complex FI into the SC, the RISC-V needs tens of instructions for each scan clock since counters and some combinational logic are necessary to locate the FFs of interest. In this case, the RISC-V needs to operate with another clock instead of SC_CLK, which complicates the clocking and degrades the FI efficiency. Another approach is to enrich the CSMU functionality, but it is difficult to list all possible FI patterns in advance.

In systems equipped with ECC or Triple Modular Redundancy (TMR) mechanisms, direct single-bit upset (SBU) injection is often tolerable. When we need to evaluate the impact of rare yet catastrophic events, such as a multiple-bit upset (MBU) in a single word, the PFIU can be configured to inject errors into two adjacent bits simultaneously as shown in Fig 4(a). Moreover, in certain AI systems where buffers store model data, the inherent robustness of AI models necessitates extensive data flipping to effectively assess the reliability of the device. To address this, the PFIU can be configured to perform continuous bit-flips over a specified length of data as illustrated in Fig 4(b). On the other hand, the probability of occurrence of MBUs is considered to determine the test coverage of the main FI modes. Evaluating all modes equally in isolation may misrepresent system-level risk. By modeling the probability of fault occurrence, FI operations can match real-world fault scenarios, providing a more accurate and practical vulnerability assessment. Thus, users can customize error patterns in the PFIU to achieve different fault scenarios based on specific application needs. Besides, to achieve a

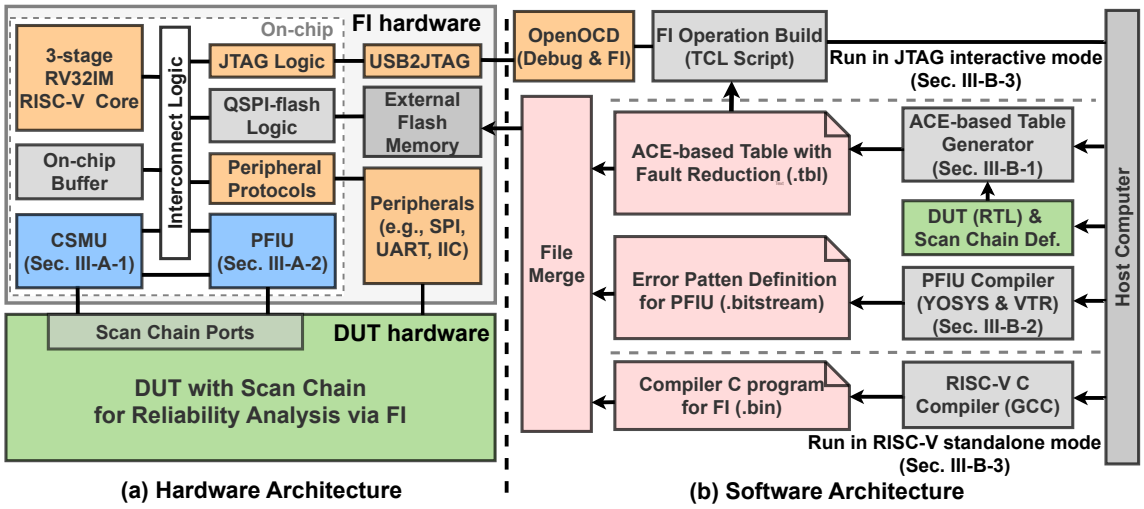


Fig. 1. Genshin Framework. (a) Hardware Architecture. (b) Software Architecture.

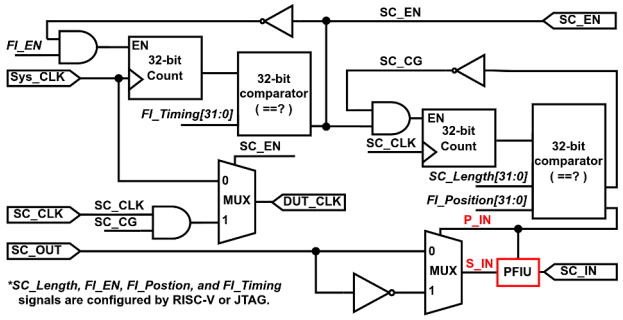


Fig. 2. Clock and Scan Management Unit.

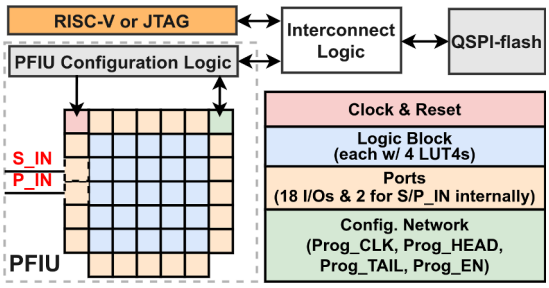


Fig. 3. Programmable Fault Injection Unit.

realistic reliability analysis, it is necessary to calculate the occurrence probability of these pattern-specialized errors. The probability estimation provides insight into how often such errors might happen in real-world conditions, enabling a more accurate assessment of the system's fault tolerance.

The design employs a 5×5 array structure constructed with LUT4 logic elements and FFs. Here, each LUT in a logic block accepts 4 inputs, enabling complex combinational logic by mapping these inputs to a desired 1-bit output. Meanwhile, the number of LUTs per logic block is 4, allowing for parallel logic functions or more complex functions by combining multiple LUTs within the same block. Furthermore, each routing wire segment spans 4 logic blocks, reducing routing hops and potentially lowering delay. Also, 15% of the inputs in each logic block can connect to a given routing channel, providing routing flexibility to the block inputs. Similarly, the output connection fraction, where 10% of the outputs from the logic block have direct connections to any specific routing channel, managing how outputs are routed in the array.

Each edge of the array incorporates 5 ports, totaling 20 ports, with two ports dedicated to S_IN and P_IN networks internally (Fig. 2) to facilitate the definition of various error patterns. Also, the bitstream downloaded via the configuration logic is 677B in size, offering efficiency advantages in error pattern switching scenarios. The reconfiguration process (error pattern switching) for the PFIU completes in approximately 110 µs at 50MHz, enabling quick switches between patterns. In contrast, conventional FPGA (e.g., Genesys2) setups often require several seconds and even longer to reload and reconfigure. This dramatic reduction in reconfiguration time significantly boosts FI throughput, especially in scenarios requiring frequent pattern changes, where delays in switching could severely impact overall testing efficiency.

To accommodate different error patterns, the bitstreams for each error pattern are stored in QSPI flash. Users can select different error patterns based on specific scenarios and download them into the PFIU through either JTAG or RISC-V.

### *B. Software Architecture*

As shown in Fig. 1(b), the software architecture for SEU impact evaluation via FI on the DUT, incorporates two operational modes: JTAG interactive mode and RISC-V standalone mode. In JTAG mode, the Tcl scripts build the FI operations, which relies on the ACE-based fault tables with fault reduction to enhance FI efficiency. The tables are generated using the

```
module PFIU_Pattern1 (
input wire SC_CLK,
input wire S_IN,
input wire P_IN,
output SC_IN
);
reg MBU;
always @(posedge SC_CLK) begin
    if (P_IN)
        MBU <= 1'b1;
    else
        MBU <= 1'b0;
    end
assign SC_IN = MBU ? !S_IN : S_IN;
endmodule
```

(a) PFIU_Pattern#1

```
module PFIU_Pattern2 (.....);
parameter N = 7;
reg [3:0] counter;   reg keep_flip;
always @(posedge SC_CLK) begin
    if (P_IN) begin
        counter <= N;
    end else if (counter > 0) begin
        keep_flip <= 'b1;
        counter <= counter - 1;
    end else begin
        keep_flip <= 'b0;
        counter <= 'd0;
    end
end
assign SC_IN = keep_flip ? !S_IN : S_IN;
endmodule
```

(b) PFIU_Pattern#2

Fig. 4. Examples of Custom Error Patterns Using PFIU.

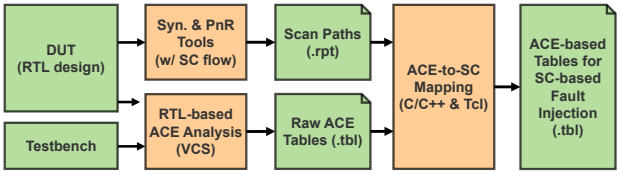


Fig. 5. ACE-based Fault Table Generator.

ACE-based table generator, the DUT and its SC definition. The DUT is then modeled and tested using the CSMU and PFIU. The PFIU is configured to define error patterns in bitstream format. Similarly, in the RISC-V standalone mode, a C program, compiled with the GCC compiler, coordinates the execution of FI processes. These processes involve merging files (e.g., compiled C program, error pattern definitions, and fault tables) before being transferred to the DUT for reliability analysis. The difference between these two modes is whether to use JTAG to implement FI or directly use RISC-V to implement FI. The rest is basically the same. After FI, result checking is essential, especially to ensure the repeatability and reliability of FI. In our setup, two result-checking strategies are employed: 1) coarse-grained observation through DUT peripherals such as UART, or GPIO to verify the functional correctness of program execution; and 2) fine-grained monitoring via SC extraction every cycle after FI, allowing detailed analysis of internal system behavior and correctness.

*1) ACE-based Table Generator:* Based on the ACE analysis flow mentioned in [13], [14], the ACE-based table generator is built as shown in Fig. 5. The process begins at the RTL level, where ACE analysis is conducted to identify relevant ACE intervals throughout the design. This involves analyzing the RTL code in the context of the system's logic and architecture. Specifically, trace assignments are inserted into the RTL to monitor key signal updates. During simulation, timestamped logs are generated using statements such as $fwrite whenever these assignments are triggered. By correlating these timestamps with register update events, the framework can accurately determine the clock cycles during which architectural state transitions occur, thereby identifying ACE intervals. However, ACE analysis at the RTL level may not fully capture the physical behavior of real hardware DUTs, since flip-flops can be removed, split, or merged during synthesis and place-and-route optimizations, leading to discrepancies between the RTL model and the final hardware implementation.

To generate ACE-based tables compatible with the DUT hardware platform for FIs, an additional mapping procedure based on C/C++ and Tcl is adopted. Once the ACE intervals are identified and raw ACE tables are generated, the system moves to the next stage, where the results are verified and mapped onto the SC definitions. In this stage, a Tcl script is constructed that utilizes built-in commands (e.g., report_scan_path) to list the names, paths, and numbers of all scan chain cells in the design. These details are stored in a file for further processing. In the subsequent stage, C/C++ code is used to implement a regular expression-based search function. This function will match the FF cell names in the raw ACE tables with the corresponding scan chain cell names extracted in the Tcl script. Once matched, the registers in the raw ACE tables are converted into their exact positions within the scan chain. This ensures that the ACE tables are mapped to the actual physical locations of the scan chain cells. Finally, the result is a scan chain cell index-based ACE table, which allows for accurate FI based on the actual configuration of the scan chain cells in the DUT hardware. These operations ensure that the generated ACE table corresponds to the actual SC configuration, enabling accurate FIs. The ACE table generated from the RTL code is then mapped onto the SC, enabling FI operation through the SC interface. Therefore, the final fault tables contain the location information of each FF in the SC and the distribution of its specific ACE intervals. This table can be converted into a matrix form for efficient FIs as follows:

$$St = \begin{bmatrix} St(1,0) & St(2,0) & \cdots & St(m,0) \\ St(1,1) & St(2,1) & \cdots & St(m,1) \\ \vdots & \vdots & \ddots & \vdots \\ St(1,n) & St(2,n) & \cdots & St(m,n) \end{bmatrix}$$

Here, $St(p,t) \in \{0,1\}$ represents the ACE information (i.e., one means ACE point, and zero means non-ACE point) of the $p$-th scan cell at time $t$. The first dimension $p$ represents the scan chain index, with $1 \leq p \leq m$. $m$ represents the total length of SCs. The second dimension $t$ represents time (i.e., system cycle), ranging from 0 to $n$. $n$ represents the overall execution cycle of system. This notation expresses the ACE transitions of the SC over time. The FI of non-ACE points will not affect the system's operation, so it can be directly reduced. This mapping ensures that faults can be injected in a precise and controlled manner, based on the verified ACE table, thus improving the fault reduction process and overall FI accuracy.

*2) PFIU Compiler:* Regarding the PFIU compiler, the bitstream compiled using YOSYS [16] and VTR [17] is employed to facilitate the selection and configuration of error patterns within the PFIU. YOSYS is an open-source framework for digital synthesis, which enables the synthesis of hardware description languages (HDLs) into gate-level representations suitable for FPGA implementation. It allows for the optimization and transformation of designs, providing

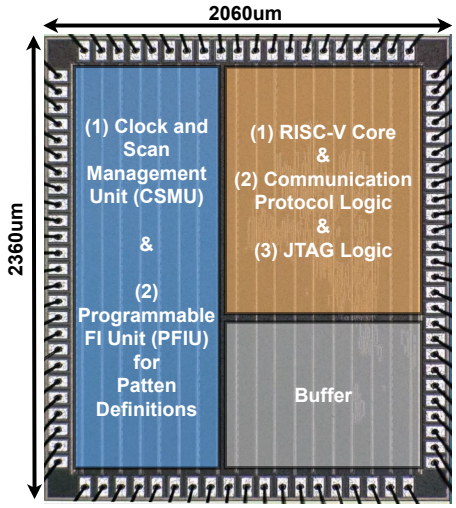


Fig. 6. Chip for Genshin Validation.

the flexibility to adapt various configurations according to specific requirements. VTR is a suite of tools designed to facilitate the process of FPGA design from high-level synthesis through to the final routing of the circuit. It takes synthesized designs and performs the placement and routing necessary for implementation on FPGAs, optimizing for area, delay, and power consumption.

*3) Operating Mode:* On the software level, Genshin provides two modes for FI. The first FI mode is a standalone mode based on RISC-V, designed for automated intensive FI experiments. Users need to develop C code for FI. In the C code, the relevant parameters for the SC and the FI error patterns need to be configured first. After that, faults can be injected through the CSMU and PFIU based on pseudo-random numbers or user-defined FI strategies. Finally, the output information from peripherals can be monitored to observe the system's operating status. If the DUT crashes, all SC information can be uploaded to the user via the CSMU for further analysis.

The second FI mode is an interactive mode based on JTAG module. Users need to construct a Tcl-based script package. This package should also include the relevant parameters for the SC and scripts for selecting and configuring the error patterns. Additionally, users can directly control JTAG via OpenOCD software [18] on the computer to perform similar FIs. In Genshin, JTAG can also directly control the CSMU and PFIU, allowing users to perform FIs interactively and monitor the operating status of the entire DUT from the host side.

## IV. Performance Evaluation

### A. Fabricated Chip for Reliability Analysis

Fig. 6 presents the chip fabricated using a 180nm CMOS process for Genshin Validation, covering an area of 4.86 $mm^2$. This work focuses on a comprehensive FI implementation with three main features. First, it supports FI control via RISC-V or a 4-port JTAG interface, providing flexible control types for FI. Second, it incorporates CSMU for precise clocking and scan signal control during fault injection, ensuring accurate fault simulation and injection. Third, it includes a PFIU, designed to define and apply customizable FI patterns. Moreover, the specifications of this chip are shown in Table I. The chip operates at a frequency range of 25–50 MHz, with power consumption ranging from 16.38 mW to 32.49 mW. It supports a core supply voltage of 1.3 V and an I/O supply voltage of 3.3 V. Although the system can support higher operating frequencies under higher pre-driver supply, the FI process is based on scan chain operations that rely on I/O bandwidth, which is limited to around 50 MHz. As a result, the chip has only been tested up to 50 MHz. However, the actual system can operate at significantly higher frequencies in non-FI scenarios. Additionally, the chip integrates a 1024B on-chip memory and supports external data access through a Q-SPI flash communication protocol with up to 256 Mb capacity, enabling flexible and efficient FI testing.

TABLE I
Chip Specifications

| Specification | Value |
|---|---|
| **Process** | 180nm CMOS |
| **Control Type** | RISC-V or JTAG |
| **Fault Injection Operation** | Clock and Scan (CSMU) |
| **Fault Pattern Configuration** | Programmable Pattern (PFIU) |
| **Core Supply [V]** | 1.3 |
| **I/O Supply [V]** | 3.3 |
| **Frequency [MHz]** | 25–50 |
| **Chip Area [$mm^2$]** | 4.86 |
| **Core Area [$mm^2$]** | 3.66 |
| **Power [mW]** | 16.38–32.49 |
| **On-chip Memory [Byte]** | 1024 |
| **Off-chip Flash [Bit]** | Up to 256M |

### B. Experimental Setup

To assess the effectiveness of Genshin, experiments are conducted with three DUTs: two RISC-V cores of PicoRV32 [19] and Ibex [20], and a downsized NVDLA-based [21] AI accelerator with 16×16 INT8 processing element array. These platforms reflect a range of complexity, from lightweight processors to high-performance AI designs, allowing for a comprehensive evaluation of Genshin's FI capabilities. Besides, as the FI-specialized chip has been fabricated as mentioned in Section IV-A, the experiments are conducted directly on hardware environment for fault analysis under real conditions.

The experimental setup used to evaluate the Genshin framework is conducted on our chip and a FPGA board (Xilinx ZCU102) as shown in Fig. 7, integrating multiple components to facilitate FI and analysis. At the core of the setup is the Genshin chip soldered on a tiny circuit board (Fig. 7, bottom-left), which interfaces with the aforementioned DUTs implemented on ZCU102 via PMOD ports. Note that these DUTs are on the FPGA to prepare multiple designs for the proposed FI system validation, but any custom chip DUTs with SCs can be tested. These DUTs are synthesized and

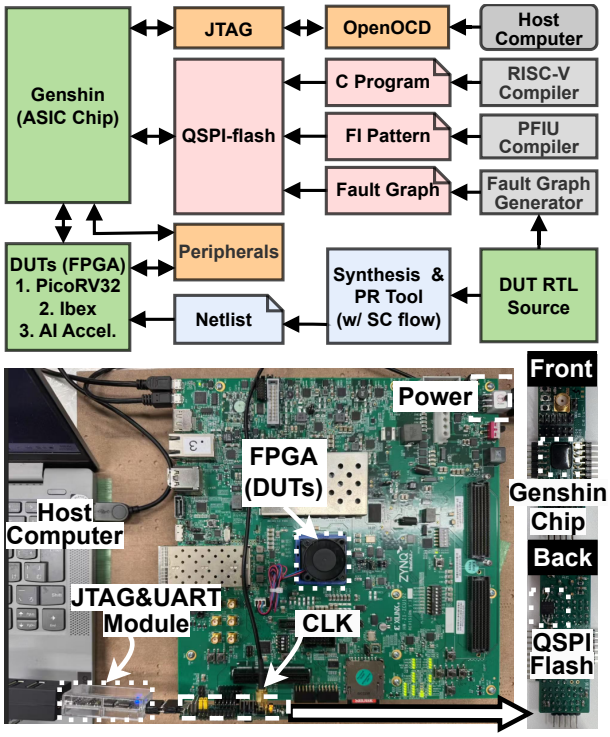


Fig. 7. Experimental Setup for Genshin.

routed using DFT flow for SC generation to produce a back-end netlist, starting from the RTL source of each DUT and utilizing Synopsys Design Compiler and ICC2. Subsequently, the back-end netlist, along with behavioral files, is imported into Vivado for synthesis and implementation. However, since some primitives described in the behavioral files (e.g., derived from standard cell libraries) are not natively supported by Vivado, slight modifications to these files are necessary to ensure compatibility during synthesis and implementation. This design transfer from Design Compiler to Vivado facilitates the introduction of SCs into DUTs on the FPGA, as SCs are generally absent in typical Vivado-mapped FPGA designs due to the availability of built-in debugging features.

To support pruned FI, the fault table generator creates ACE-based fault tables based on each DUTs and its SC definition. The custom FI patterns are then defined and compiled via the PFIU compiler. In standalone mode, the C program needs to be compiled. These elements are loaded into the QSPI flash. Moreover, some peripherals (e.g., UART, SPI) are incorporated to enable seamless interactions with Genshin. A host computer running OpenOCD acts as the primary control hub for the user in interactive mode, managing the FI and monitoring through the JTAG module (Fig. 7, left). The host can compile FI patterns and C programs via the PFIU and RISC-V compiler, coordinating with Genshin for FI. This setup allows for real-time FI and performance evaluation, effectively assessing Genshin's capabilities in handling SEUs across diverse DUTs, validating Genshin's adaptability and efficiency.

TABLE II
FAULT INJECTION PERFORMANCE.

| DUT | Pico RV32 | Ibex | NVDLA-based Accelerator |
|---|---|---|---|
| **SC Length** | 2795 | 4630 | 61393 |
| **Standalone Mode** | 3802 cycles (1.3603*) | 5637 cycles (1.2175*) | 62400 cycles (1.0164*) |
| **Interactive Mode** | 6790 cycles (2.4293*) | 8625 cycles (1.8629*) | 65388 cycles (1.0651*) |

*Normalization performed using cycles/scan chain length.

### C. Fault Injection Performance Analysis

We first evaluate the efficiency of Genshin by measuring the FI latency, defined as the number of clock cycles required for each FI operation (cycles/FI), and the FI throughput across three DUTs, excluding application execution time. In addition, note that in the throughput estimation, the overhead of result checking is not included, as it can vary significantly depending on the application and fault severity. For example, some faults may require detailed debugging or system rollback, while others may have negligible impact. Therefore, the reported throughput reflects the peak performance, assuming that no critical faults occur (i.e., no error logging is required) during execution. To align with the real hardware conditions, the chip runs at 50 MHz for FI evaluation. Also, the CSMU is configured to handle clock management and synchronization, ensuring that FIs occurred with minimal delay. Also, to test the fundamental FI efficiency of Genshin, the PFIU is compiled with direct input-to-output connection logic for SC_IN port.

The SC length of these DUTs are shown in Table II. In standalone mode, the FI latency for both PicoRV32 and Ibex core are 3.8k, and 5.6k cycles/FI, respectively. The AI accelerator demonstrates a higher latency of 62k cycles/FI due to its increased complexity and parallel computing structure. In interactive mode, due to the characteristics of serial communication in JTAG, the FI latency is about 3k cycles higher on average than in standalone mode due to the latency of JTAG configuration. After being normalized by dividing the total number of clock cycles required for a single FI by the SC length, which gives a normalized value representing the number of clock cycles required per scan bit, it can be found that when the number of scan FFs is relatively large, the latency of FI is basically proportional to the SC length (e.g., the accelerator design with a normalized latency of 1.0164 at standalone mode). In terms of throughput, Genshin achieved rates of 801 to 13,150 FIs/second on these DUTs in standalone mode and 765 to 7,363 FIs/second in interactive mode.

Here, the PFIU exhibits a significantly faster and more lightweight error pattern switching capability compared to conventional FPGA-based approaches. Although commercial FPGAs such as the Genesys board offer abundant LUT resources that support simultaneous placement of multiple circuits for FI, allowing one-cycle pattern switching via multiplexers, the cost is high in terms of resource usage and design complexity. In addition, the number of patterns placed simultaneously is limited by available logic resources and routing constraints. In contrast, our PFIU enables flexi-

TABLE III
PRUNED FAULT INJECTION EVALUATION.

| DUT | Pico RV32 | Ibex | NVDLA-based Accelerator |
|---|---|---|---|
| **Application** | Dhrystone (10 runs) | String Search | LeNet5 (MNIST) |
| **ACE Analysis Overhead (days)** | ∼2 | ∼3 | ∼24 |
| **Execution Cycle** | 19,424 | 21,180 | 83,506 |
| **Total Fault** | 54,245,360 | 98,063,404 | 625,737,938 |
| **ACE / non-ACE Fault** | 20,183,258 / 34,062,102 | 16,464,012 / 81,599,392 | 339,123,567 / 286,614,371 |
| **Fault Reduction Ratio** | 62.79% | 83.21% | 45.80% |
| **Total FI Overhead Reduction** | $^{s}$1.295e+11<br>$^{i}$2.313e+11 | $^{s}$4.600e+11<br>$^{i}$7.038e+11 | $^{s}$1.788e+13<br>$^{i}$1.874e+13 |

$^{s}$RISC-V standalone mode, $^{i}$ JTAG interactive mode.

ble reconfiguration of error patterns via compact bitstreams, achieving switching latency as low as 110 µs. Assuming a new error pattern is required for every FI cycle, the overhead accounts for less than $9\times10^{-9}$ of the application execution time, rendering it practically negligible. This efficiency allows the PFIU to support scalable and fine-grained FIs without the overhead or resource waste often associated with FPGAs.

Usually, to accelerate the reliability analysis of DUTs, high-intensity neutron or proton (e.g., 100,000x higher or more than the flux at ground level) irradiation experiments will be conducted. FF soft error rates (SERs) under these conditions typically range from $10^{-6}$ to $10^{-3}$ errors/sec/Mb for 65nm or smaller nodes [22]. When the maximum value of $10^{-3}$ errors/sec/Mb is applied to the accelerator DUT, the error rate is $5.855\times10^{-5}$ errors/sec. On the other hand, Genshin achieves the FI throughput of 765 FIs/sec. When supposing a 1-second application for reliability analysis, we need 170,800 application executions to obtain one FI sample in the irradiation experiment, while we can inject a fault every application execution. In this case, 170k times speed-up is possible in our framework, which contributes to a more comprehensive analysis without any special facilities. It should be mentioned that when the actual irradiation test, it is difficult to use higher intensity beams since multiple independent errors frequently occur in SRAMs in a short time, for example, in a recovery phase and rebooting process, while such errors will scarcely happen in real environments. On the other hand, our framework eliminates such unrealistic error patterns while keeping the high FI throughput. Note that our framework evaluates such extremely rare cases as well if necessary (e.g., highly redundant designs for mission-critical applications).

### *D. Pruned Fault Reduction Analysis*

Genshin enhances FIs through ACE-based pruning method. This approach targets and eliminates faults with minimal impact on system reliability, thereby drastically reducing the overall number of FIs. The effectiveness of this strategy is evident in Table III, which presents the pruned FI evaluation across three DUTs as mentioned in Section IV-B. Each DUT is tested with a specific application: Dhrystone for PicoRV32, String Search for Ibex, and LeNet5 for AI accelerator.

The ACE-based table generation, executed on the EPYC 7502P platforms with multiple threads for parallel analysis, presents a considerable run-time for ACE analysis. Therefore, only 10 runs of the Dhrystone benchmark have been conducted, alongside the String Search benchmark. Additionally, the LeNet5 model, a smaller and more manageable architecture, has been chosen. Despite these reductions, the analysis remains comprehensive. The applications contain numerous repeated computational units, which means that even with fewer runs or a smaller application size, the impact on the analysis is minimal. With PicoRV32 requiring approximately 2 days, Ibex around 3 days, and the NVDLA-based accelerator needing about 24 days. Further research for speeding-up the ACE analysis is demanded while it is beyond this work.

The pruned FI analysis across these three DUTs reveals significant reductions in fault numbers and total FI cycles. PicoRV32 exhibits an impressive 62.79% reduction in faults, equating to a decrease of approximately $1.295\times10^{11}$ cycles in standalone mode and $2.313\times10^{11}$ cycles in JTAG interactive mode with direct input-to-output connection logic of SC. Ibex follows with a 83.21% fault reduction, leading to a total FI cycle reduction of around $4.600\times10^{11}$ cycles in standalone mode and $7.038\times10^{11}$ cycles in interactive mode. Lastly, the NVDLA-based AI accelerator demonstrates a 45.80% fault reduction, resulting in a substantial total FI cycle reduction of approximately $1.788\times10^{13}$ cycles in standalone mode and $1.874\times10^{13}$ cycles in interactive mode. Idle computing units and memory-related operations are the main sources of non-ACE in the AI accelerator, and the proportion of control logic is relatively lower. The AI accelerator is designed to focus more on parallel computing and can significantly reduce the proportion of non-ACE, while Dhrystone and String Search contain more non-computation-related logic, resulting in a higher proportion of non-ACE. On the other hand, although our methodology primarily targets ACE intervals to efficiently reduce faults, evaluating FI during non-ACE intervals is also important to confirm the absence of unintended error propagation. While a comprehensive non-ACE analysis is outside the scope of this work, our framework supports such extensions and can be configured to perform specific FIs across both ACE and non-ACE windows for full functional validation.

### *E. Comparison with the State-of-the-Art FI Frameworks*

Table IV summarizes the measurements of Genshin framework and compares it with prior art [8]–[12]. Eris [8] and Cheng et al. [9] rely heavily on simulation techniques, while TensorFI [10] uses software-based TensorFlow graphs, and Fiji-FIN [11] and Cho et al. [12] utilize hardware emulation through FPGAs. In contrast, Genshin adopts a software-hardware co-design approach with an ASIC implementation, using a SC–based FI mechanism controlled by the CSMU to enable precise and controllable FI and monitoring.

Regarding FI efficiency, Eris [8] and Cheng et al. [9] are rated as low due to the limitations of simulation tools, while TensorFI [10] achieves moderate efficiency using multi-threading. Fiji-FIN [11] and Cho et al. [12] also show only

TABLE IV
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART FAULT INJECTION FRAMEWORKS

| | Eris [8] | Cheng et al. [9] | TensorFI [10] | Fiji-FIN [11] | Cho et al. [12] | This work |
|---|---|---|---|---|---|---|
| **Technique** | Simulation | Simulation (Synopsys VCS) | Software | Hardware (Xilinx FPGA) | Hardware Emulation (Xilinx FPGA) | **Software-Hardware Co-Design (ASIC)** |
| **FI Mechanism** | Instrumented C-model derived out of HDLs | Backdoor Access of FF & Memory | TensorFlow Graphs | Memory Access | BEE3 Emulation System | **Scan Chain (CSMU)** |
| **FI Efficiency** | Low | Low | Middle (Multi-thread) | High | Middle | **High** |
| **Fault Reduction** | No | No | No | No | No | **Yes (45.80–83.21%)** |
| **FI Pattern Definition** | No | No | Yes (Scalar&Tensor) | Yes (SEU&MBU) | No | **Yes (PFIU)** |
| **Target Designs** | Chisel+Verilog+ VHDL Designs | Netlist-level Designs | Tensorflow-based Programs | Limited ML/DL-Powered IoTs | LEON3 Processor (in-order SPARC) | **Any Hardware with Scan Chain(s)** |
| **Fault Tracking Cabability** | Yes | Yes | No | No | Yes (System Stack) | **Yes (Scan Chain)** |
| **Accessibility to Internal Modules** | High | High | Low | Middle | Middle | **High (All Registers)** |
| **Controllability** | High | Middle (Relying on Tools) | Middle (TensorFlow-based) | Low | Middle | **High (RISC-V&JTAG)** |
| **Repeatability** | Yes | Yes | NA | Yes | Yes | **Yes** |
| **Monitoring Time Resolution** | High | High | Middle | Low | High | **High (System Cycle)** |
| **Scalability** | Middle (Processor Level) | High | Low (Tensorflow Only) | Low (Limited QNNs) | Middle (Process Level) | **High** |
| **Fault Coverage** | High | High | High | Low | Middle | **High** |
| **Injection Granularity** | Middle (Instruction Level) | High (System Cycle) | Middle (Graph Level) | Low (Image Level) | Middle (3 Levels: FF, Register, Program) | **High (System Cycle)** |
| **Automation Capability** | Middle (Manual Intervention Required) | High (Backdoor Access Automatically) | High (Graph Copying, Duplication, and Logging | Middle (Complex Process) | High (w/ Comprehensive Propagation Patterns) | **High (Processor-based& Script-based)** |

moderate performance. In contrast, our framework achieves high efficiency through real-time FI enabled by SCs and tight integration of hardware and software components. For example, when running the same AI accelerator under Cheng et al.'s simulation-based framework [9] on VCS, each FI takes approximately 20,000 times longer than Genshin. Fault reduction is another key differentiator. While all other frameworks lack built-in fault reduction mechanisms, our system demonstrates fault reduction rates ranging from 45.80% to 83.21% across three DUTs (PicoRV32, Ibex, and an AI accelerator) due to the ACE-based pruning technique [13], [14].

In terms of FI pattern definition, most frameworks are either limited or completely lack this capability. Eris [8] and Cheng et al. [9] do not offer pattern definition, and TensorFI [10] is restricted to scalar and tensor fault patterns. Fiji-FIN [11] and Cho et al. [12] have minimal support for SEU and MBU faults. Genshin, however, incorporates a PFIU that can define comprehensive FI patterns, offering better insight and control over fault behavior. On the other hand, the range of target designs supported by each framework varies significantly. TensorFI [10] is limited to TensorFlow-based programs, Fiji-FIN [11] is tailored to IoT devices with limited machine learning (ML) capabilities, and Cho et al. [12] focused on the LEON3 processor. Genshin, in contrast, is highly adaptable and can be used with any hardware that incorporates SCs, making it a more flexible solution for FI across diverse platforms.

When considering tracking capability, several frameworks, such as TensorFI [10] and Fiji-FIN [11], do not offer tracking at all, while Eris [8], Cheng et al. [9], and Cho et al. [12] provide only basic tracking. Genshin excels in this area by utilizing SCs for real-time fault tracking. Besides, access to internal modules is another area where our framework demonstrates superiority. While TensorFI [10] provides low access and Fiji-FIN [11] offers moderate accessibility, Genshin ensures high access to all internal registers and modules, facilitated by the use of SCs.

Controllability varies across frameworks as well. TensorFI's controllability is constrained by TensorFlow's limitations [10], Cho et al. [12] relies on specific emulation systems, and Fiji-FIN [11] provides only low controllability. Our framework, on the other hand, offers high controllability through RISC-V and JTAG-based methods as mentioned in Section III-B3, allowing efficient FI control and monitoring. Repeatability is another critical factor, and while some frameworks like Eris [8] ensures consistent repeatability, TensorFI [10] and Fiji-FIN [11] fall short. Our framework supports high repeatability through comprehensive control features and monitoring, ensuring consistent results across repeated FIs.

Monitoring time resolution is critical for accurate fault analysis. TensorFI [10] is limited in this respect, and Fiji-FIN [11] provides low time resolution. Genshin, however, ensures high-resolution monitoring through efficient FI scheduling in CSMU, enabling precise tracking of fault behavior. Scalability is another area where our framework excels. While TensorFI

[10] has limited scalability and Fiji-FIN [11] only supports quantized neural networks (QNNs), Genshin is highly scalable across hardware domains, making it suitable for most systems.

Fault coverage injection granularity is an essential feature for accurate fault analysis. Fiji-FIN [11] is restricted to image-level granularity, TensorFI [10] covers graph-level granularity, and Cho et al. [12] reaches instruction-level granularity. Genshin surpasses these solutions by achieving high granularity at system cycle level, covering all FI points more effectively. Lastly, Genshin demonstrates clear superiority in automation capability. Eris [8] and Fiji-FIN [11] rely on manual intervention, TensorFI [10] requires complex processes, and Cheng et al. [9] provides automatic backdoor access. Our framework, however, demonstrates high automation through processor-based (RISC-V) and script-based (OpenOCD) automation processes, enabling efficient, repeatable FI and analysis.

In general, our framework outperforms other FI solutions in most categories. Its FI efficiency, fault reduction capability, granularity, tracking, controllability, scalability, automation, and pattern definition capabilities make it the ideal solution for reliable, high-performance FI in ASIC-based applications.

## V. Conclusion

In this work, we present Genshin, a generalized FI framework designed to address the challenges of reliability analysis in high-reliability systems. By integrating hardware-software co-design with a RISC-V/JTAG-controlled architecture, Genshin leverages a CSMU and a lightweight PFIU to enable precise, flexible, and high-throughput FI and SEU analysis across diverse DUTs. Another key advancement of the framework is its ACE-based fault pruning approach, which significantly reduces FI overhead by minimizing test duration while maintaining a high level of fault coverage. To further validate Genshin's effectiveness, we implemented a prototype on a 180nm CMOS chip, demonstrating its feasibility in real-world hardware scenarios. This physical chip implementation provided additional insights into Genshin's FI capability for reliability analysis under realistic conditions, reinforcing its practical applicability. Evaluations on multiple DUTs implemented on FPGA, including PicoRV32, Ibex, and a downsized NVDLA-based AI accelerator, demonstrate that Genshin achieves 3,802–65,388 cycles/FI and fault reduction rates ranging from 45.80% to 83.21%. These results underscore Genshin's efficiency, flexibility, and scalability, establishing it as an ideal solution for SEU analysis in space and other critical environments where SEUs are prevalent.

## Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant 62274081, the Grant-in-Aid for Scientific Research (S) from Japan Society for the Promotion of Science (JSPS) under Grant 24H00073, and the Grant-in-Aid for Early-Career Scientists from JSPS under Grant JP21K17721. Dr. Xiong's contribution to this work is limited to his interactions with Drs. Cheng and Hashimoto at Kyoto University, Kyoto, Japan.

## References

[1] H. Cho, E. Cheng, T. Shepherd, C. -Y. Cher and S. Mitra, "System-Level Effects of Soft Errors in Uncore Components," IEEE Trans. CAD, Sept. 2017.

[2] E. Cheng et al., "(Invited) Cross-Layer Resilience: Challenges, Insights, and the Road Ahead," Proc. DAC, 2019.

[3] 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," in IEEE Trans. on DMR, March. 2022.

[4] 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," in IEEE Trans. Nuclear Science, Jan. 2022.

[5] K. M. Girgis, T. Hada and S. Matsukiyo, "Estimation of Single Event Upset (SEU) rates inside the SAA during the geomagnetic storm event of 15 May 2005," 2021 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE), 2021.

[6] S. Z. Can, G. Yalcin, O. Ergin, O. S. Unsal, and A. Cristal, "Bit impact factor: Towards making fair vulnerability comparison," Microprocessors and Microsystems, Aug. 2014.

[7] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ace analysis reliability estimates using fault-injection," ACM SIGARCH Computer Architecture News, 2007.

[8] S. Nema et al., "Eris: Fault Injection and Tracking Framework for Reliability Analysis of Open-Source Hardware," 2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Singapore, Singapore, 2022, pp. 210-220, doi: 10.1109/ISPASS55109.2022.00027.

[9] Cheng et al. "How accurately can soft error impact be estimated in black-box/white-box cases?–a case study with an edge AI SoC–." Proceedings of the 61st ACM/IEEE Design Automation Conference. 2024.

[10] Z. Chen et al., "TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications," 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 2020.

[11] N. Khoshavi, C. Broyles, Y. Bi and A. Roohi, "Fiji-FIN: A Fault Injection Framework on Quantized Neural Network Inference Accelerator," Proc. ICMLA, 2020.

[12] H. Cho, S. Mirkhani, C. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in Proc. DAC'13, 2013, pp. 1–10.

[13] S. Raasch, A. Biswas, J. Stephan, P. Racunas, and J. Emer, "A fast and accurate analytical technique to compute the avf of sequential bits in a processor," in Proc. MICRO, 2015.

[14] D. -A. Yang, Y. -T. Chang, T. -S. Hsu, J. -J. Liou and H. H. Chent, "ACE-Pro: Reduction of Functional Errors with ACE Propagation Graph," 2021 IEEE International Test Conference (ITC), Anaheim, CA, USA, 2021.

[15] RISC-V External Debug Support. [Online]. Available: https://riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf

[16] Wolf, Clifford, Johann Glaser, and Johannes Kepler. "Yosys-a free Verilog synthesis suite." Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip). Vol. 97. 2013.

[17] Kevin E. Murray et al., 2020. "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling". ACM Trans. Reconfigurable Technol. Syst. 13, 2, Article 9 (June 2020), 55 pages. https://doi.org/10.1145/3388617

[18] D. Rath, "Open On-Chip Debugger," Diploma, Department of Computer Science, University of Applied Sciences Augsburg, 2005

[19] M. Jahnke, L. Bublitz and U. Kulau, "Performance Evaluation of PicoRV32 RISC-V Softcore for Resource-Constrained Devices," 2023 IEEE Nordic Circuits and Systems Conference (NorCAS), Aalborg, Denmark, 2023, pp. 1-6, doi: 10.1109/NorCAS58970.2023.10305479.

[20] Raveendran, Rahul, and Subhajit Bhuinya. "Customization of Ibex RISC-V Processor Core." Customization of Ibex RISC-V Processor Core (2021).

[21] F. Farshchi, Q. Huang and H. Yun, "Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim," Workshop on EMC2, 2019.

[22] K. Takami, Y. Gomi, S. Abe, W. Liao, S. Manabe, T. Matsumoto, and M. Hashimoto, "Characterizing SEU Cross Sections of 12- and 28-nm SRAMs for 6.0, 8.0, and 14.8 MeV Neutrons," Proceedings of International Reliability Physics Symposium (IRPS), 2023

Regular Paper