Tenpura: A General Transient Fault Evaluation and Scope Narrowing Platform for Ultra-fast Reliability Analysis

Quan Cheng[1,2], Huizi Zhang[2], Chien-Hsing Liang[3], Mingtao Zhang[1],
Jing-jia Liou[3], Jinjun Xiong[4], Longyang Lin[2], Masanori Hashimoto*[1]

[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]Department of Computer Science and Engineering, University at Buffalo, USA
*{hashimoto@i.kyoto-u.ac.jp}

***Abstract*—For reliability-critical silicon systems, transient errors caused by cosmic rays necessitate comprehensive and efficient reliability analysis before product deployment. Fault injection (FI) serves as a cost-effective alternative to expensive irradiation experiments for evaluating system robustness. However, simulation-based FI is constrained by the performance of the underlying hardware platform, making it impractical for large-scale designs, where achieving high fault coverage can take months or even years. Furthermore, most transient errors have no impact on system functionality, and filtering out these insignificant errors in advance can significantly enhance the efficiency of reliability analysis. To address these challenges, we propose Tenpura, a fault evaluation platform designed for ultra-fast reliability analysis. In Tenpura, a transient fault scope narrowing method is introduced to narrow the FI scope via the proposed scan-based activity tracing flow, further optimizing fault analysis and improving overall efficiency. By leveraging FPGA emulation and scan chain-based fault analysis at the pre-silicon stage, Tenpura achieves high-efficiency fault reduction (88.49–96.26% across three design under tests (DUTs) including RISC-V cores and NVDLA-based AI accelerator) within one month, delivering over an order of magnitude faster fault analysis compared to SOTA methods.**

***Index Terms*—fault injection, reliability analysis, scan chain, scope narrowing**

I. Introduction

In silicon-based systems, especially those deployed in safety-critical environments (e.g., space tasks, autonomous driving), cosmic rays present a significant challenge to system reliability [1], [2]. These high-energy particles can induce transient faults, commonly known as soft errors, within the silicon substrate [3]. Such errors manifest as momentary disruptions in circuit behavior, potentially compromising system integrity and functionality. These faults are particularly concerning for reliability-demanding applications, where even brief disruptions can lead to system failure or mission compromise [4], [5]. Evaluating transient faults early in the product lifecycle is critical for ensuring the reliability of silicon chips before they reach production. Early-stage transient fault evaluation not only helps mitigate potential failures, but also accelerates product deployment by proactively identifying and addressing vulnerabilities [6], [7]. Furthermore, with the growing demand for edge computing and AI-enhanced platforms in fail-safe fields, comprehensive fault analysis has become a cornerstone of modern silicon systems [8], [9].

Fault injection (FI) has attracted considerable attention as a powerful tool for analysis of system-level reliability [10]–[13]. It allows the simulation of soft errors by deliberately introducing faults into the system, enabling engineers to observe how the system reacts and whether error-handling mechanisms function as intended. However, traditional FI methods encounter significant limitations. Hardware emulation-based FI often struggles with insufficient coverage, limiting its ability to comprehensively test system reliability across all potential fault scenarios [12]. On the other hand, software simulation-based FI suffers from severe inefficiency. This inefficiency stems from two main causes. First, the sheer number of possible fault locations and injection time points leads to a combinatorial explosion in test cases, making exhaustive simulation impractical. Second, many injected faults turn out to be inconsequential. They do not propagate to the system output or violate correctness, but still consume computational resources during simulation [14]. This results in large portions of the FI campaign contributing little to meaningful analysis, wasting both time and compute cycles. Consequently, full-coverage simulation-based FI can take months or even years to complete for complex systems [10]. This inefficiency is not just a matter of computational cost. It also hinders timely reliability evaluation and rapid design iterations. By narrowing the FI scope (i.e., identifying and filtering out faults that do not affect the system), it is possible to reduce redundant injections, accelerate analysis, and achieve ultra-fast yet meaningful reliability assessment.

To address the aforementioned issues, this paper introduces **Tenpura**: A General **T**ransient Fault **E**valuation and Scope **N**arrowing **P**latform for **U**ltra-fast **R**eliability **A**nalysis. The proposed framework incorporates three key points to address the challenges of traditional FI methods for reliability analysis:

- **Scan-based Activity Tracing Flow**: A scan-based activitiy tracing flow is implemented by reconfiguring the scan chain (SC) to enhance the fault analysis capabilities. In mode 1, the SC is utilized for FI and operational correctness verification, allowing users to systematically

inject faults or detect defects into various flip-flops (FFs) on the chip. Mode 2 switches to capturing the write state of FFs, enabling the collection of detailed information about how data propagates and changes.

- **Fault Injection Scope Narrowing Flow**: Leveraging the SC's ability of both modes to efficiently capture FF write states, data flipping states, and the mapping of FF interconnections, a C/C++-based program is developed to implement FI scope narrowing (SN). This process significantly reduces the number of faults that need to be injected by filtering out faults that are unlikely to affect the behavior of the system. By focusing on high-impact faults, the proposed platform speeds up the reliability analysis process without sacrificing coverage.
- **FPGA Emulation for Fast Data Acquisition**: To quickly obtain the operating status of design under test (DUT), the emulation platform is deployed to FPGA. On the FPGA, the dual-mode SC realized in activity tracing flow is used to obtain the write states and data of FFs and to perform FIs to obtain the states of data flipping. Combining the above data and analyzing it through a FI SN flow can efficiently accelerate the fault reduction analysis.

In summary, Tenpura provides a comprehensive solution for transient fault evaluation and FI scope narrowing, addressing the limitations of traditional FI methods. Its contributions in SC reconfiguration, FI SN and FPGA emluation enable ultra-fast, efficient, and precise reliability analysis for silicon-based systems operating in harsh environments such as space.

II. Related Work

Fault injection is essential for assessing system reliability, particularly for analyzing transient faults in silicon systems under radiation or environmental disturbances. Traditional FI methods can be broadly classified into hardware-based and software-based approaches, each facing distinct limitations.

Software-based (simulation-based) FI offers flexibility without hardware modifications but is often inefficient, with comprehensive campaigns taking months or even years for complex systems with many FFs. The vast number of fault scenarios further increases computational demands. Cheng et al. [5] implemented software-based FI for an AI SoC using backdoor access, but the long simulation times significantly impact efficiency. Similarly, Eris [10], a C/C++ RTL simulation FI framework, achieves high coverage at low cost but is limited to instruction-level FI and suffers from long simulation times for large designs. Hardware-based FI, such as FPGA emulation, enables high-speed and high-fidelity testing but struggles with comprehensive fault coverage due to the complexity of modern ICs. Many FI techniques fail to account for all transient fault scenarios, and their setup is often resource-intensive. Fiji-FIN [12], for example, only injects faults into memory cells, missing registers and limiting DUT resilience evaluation. Similarly, an FPGA-based FI framework [15] introduces faults at multiple levels (software, register, FF) but suffers from high resource consumption and limited applicability to modern processors due to its reliance on LEON3. Moreover, existing frameworks often lack well-defined error patterns tailored to specific DUT requirements. Another key issue with hardware and software FI methods is the irrelevance of fault impact. Not all faults will lead to serious system failures, resulting in unnecessary overhead and wasted simulation or emulation time. Current methods for screening irrelevant errors are basically based on software extraction, which faces limitations.

FI scope narrowing techniques, such as filtering non-impact single event upsets (SEUs) faults, have been introduced to improve the efficiency of reliability analysis without compromising comprehensiveness. For example, Raasch et al. [14] improved architecturally correct execution (ACE) analysis accuracy by extracting circuit node trees from RTL designs. However, their approach is inefficient and lacks netlist-level simulation, leading to potential inaccuracies. Since ACE analysis determines whether a bit-flip could affect system behavior, it is crucial to comprehensively evaluate errors only within the ACE interval (i.e., the critical window between a write and the final read before the next write). Yang et al. proposed ACE-Pro, an RTL-based propagation graph method [6] up to 99.91% fault reduction, but it relies on pure simulation. Also, due to the low efficiency of simulation, the design takes several weeks to simulate dozens of system cycles. Moreover, Huang et al. proposed an FPGA-based acceleration method for transient fault reduction analysis [16]. While it enhances efficiency by accelerating certain steps, many time-consuming processes such as ACE analysis still depend on software simulation, limiting overall speed improvements. In addition, the above research works do not improve and optimize FI and accelerate fault reduction analysis from a circuit perspective.

To address these challenges, an FI platform that seamlessly integrates the advantages of both software and hardware is essential for efficient collaborative design and ultra-fast reliability analysis. Additionally, an effective platform for fault reduction analysis is essential to further accelerate system's reliability evaluation.

III. Tenpura Framework

Tenpura is a comprehensive platform aimed at conducting ultra-fast reliability analysis for digital systems, particularly under transient fault conditions. Tenpura accelerates FI scope narrowing by assisting ACE analysis [17] and Def/Use Pruning (DUP) [18]–[20] with FPGA-based DUT tailored for reliability analysis, whereas earlier studies performed both ACE analysis and DUP entirely in software simulation [6], [14]. The DUT is implemented on an FPGA with enhanced scan chains that provide additional functionality required for hardware-assisted FI scope narrowing, thereby drastically reducing the otherwise time-consuming simulation effort.

Fig. 1 shows the original DUT design, followed by FPGA-based implementation of DUT with reliability analysis functionalities. The flow starts from the HDL design (Verilog or SystemVerilog), which integrates standard cell and IP libraries, design constraints (SDC), and technology files. The first step involves traditional ASIC design flow (i.e., synthesis and place-and-route (PR) with design-for-testability (DFT),

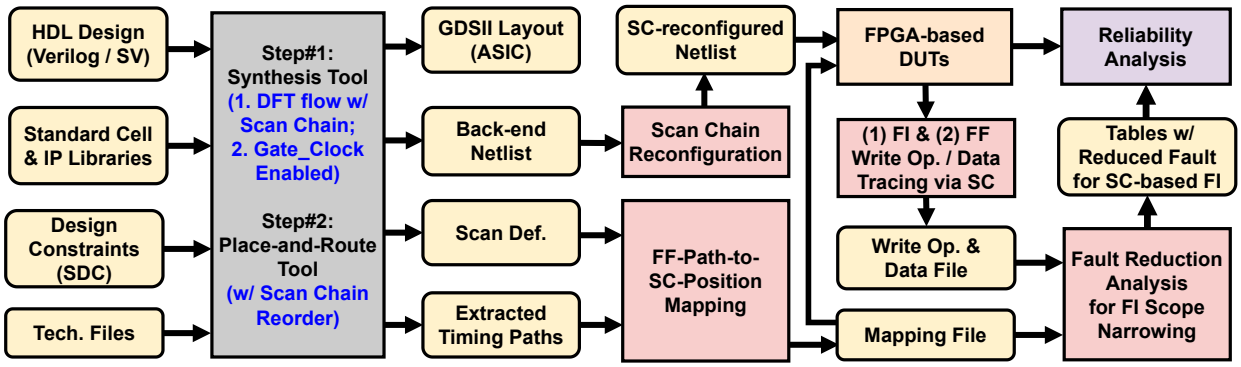

Fig. 1. Flow of Tenpura Platform for Ultra-fast Reliability Analysis.

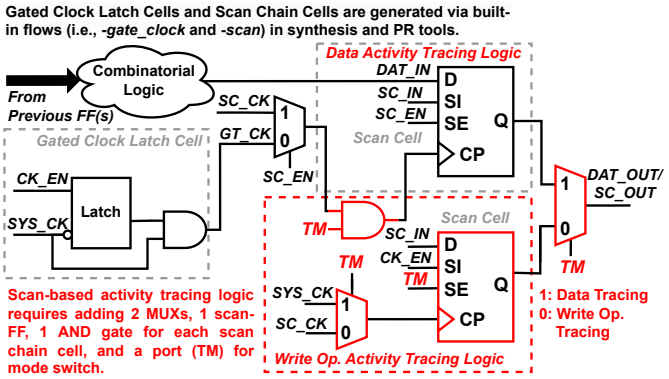

Fig. 2. Scan-based Activity Tracing Logic of FFs.

enabling scan chains and clock gating). This process produces a GDSII layout for ASIC fabrication, a back-end netlist for the following scan-based activity tracing detailed in Section III-A, SC definitions, and extracted register-to-register (R2R) logic paths only with start points and end points necessary for the following SC position mapping detailed in Section III-B. Moreover, the design after reconfiguring the SC for improved reliability analysis will be deployed to the FPGA as DUT for FI and data extraction detailed in Section III-C. Then, the data extracted by the SC on the FPGA will be analyzed and the corresponding fault reduction tables will be generated to achieve FI SN detailed in Section III-D.

A. Scan-based Activity Tracing

Fault reduction analysis, especially ACE analysis, relies on tracking the read/write operations and the stored data of FFs to determine whether a bit-flip can propagate and affect system behavior. However, traditional DFT flows generate SCs that only support FI and data extraction at each time step, without capturing the specific read/write operations of FFs. To address this limitation, this paper proposes a scan-based activity tracing mechanism implemented using Tcl scripts to restructure the SC(s), enabling the extraction of both write enable signals and data for all FFs at every time step for fault reduction analysis as shown in Fig. 2.

The synthesis and PR tools come with built-in flows capable of generating the gated clock latch cells through the gate-clock flow and the scan cells through the DFT flow as shown in the upper part of Fig. 2. Based on the generated results from the above steps, we used some commands of commercial EDA tools (e.g., *create_cell*, *connect_pin*, *disconnect_net*, *report_timing*) to realize the reconfiguration of SC. The SC reconfiguration part is shown in the lower part of Fig. 2. The AND gate in red serves to implement clock gating for the upper scan cell for FF write operation tracing, preventing its clock signal from contaminating data stored in the upper FF. The newly added lower scan FF is designed for extracting write operations to the upper scan FF (via the CK_EN signal). When $CK\_EN = 1$, it indicates the upper FF is being updated with new data, while $CK\_EN = 0$ maintains the current stored value without updates.

Basically, the reconfigured SC supports two modes: Data tracing mode (TM=1): The system operates in normal functional mode or SC mode, and the upper scan FF becomes active to shift out the data of all FFs in the current clock cycle. Write operation tracing mode (TM=0): The lower scan FF is enabled to scan through the chain, specifically recording the write operating state (CK_EN) of the upper FF in the current clock cycle. Once the write state is captured with TM=1, the write state scan starts by setting TM=0. Thus, the data update behavior can be observed through the SC output. This dual-mode implementation enables data extraction and operational status monitoring (via clock-gated enable signals) for subsequent fault reduction analysis while maintaining backward compatibility with legacy SC functionality. Additionally, the scan-based activity tracing logic is merely an example. Users can customize the reconfiguration process based on specific requirements to enable additional functionality.

B. FF Connectivity to Scan Chain Mapping

Fault reduction analysis and reliability analysis are based on the connectivity among FFs to extract the fault propagation graph. In this design, both analyses rely on operations within the SC. Therefore, it is crucial to map the FF connections to specific locations of FFs within the SC for accurate analysis.

This process, implemented by Tcl scripts, focuses on accurately mapping FF connections to specific SC locations to enhance the precision of FI, fault reduction analysis, and reliability analysis. Commercial EDA tools, for example, are used to extract all R2R paths through commands such as

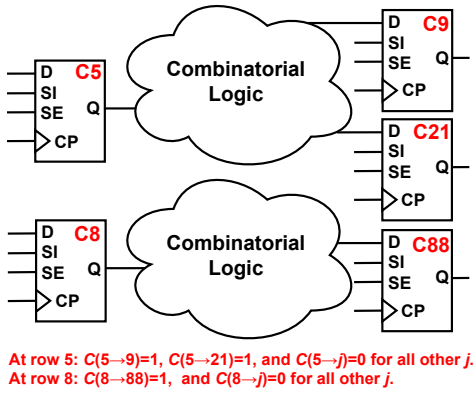

Fig. 3. Example of FF Connectivity to SC Mapping Logic.

report_timing. The start and endpoints of these R2R paths correspond to FFs. Typically, each FF can serve as either a starting or ending point in multiple R2R paths, resulting in multiple connections. By mapping these extracted FFs to their positions within the SC, a matrix representation of their connectivity can be constructed. Based on this, we can express the connection relationship more clearly using the notation $C(i \rightarrow j)$, which represents a physical connection from start point i to end point j:

$$C = \begin{bmatrix} C(1 \rightarrow 1) & C(1 \rightarrow 2) & \cdots & C(1 \rightarrow m) \\ C(2 \rightarrow 1) & C(2 \rightarrow 2) & \cdots & C(2 \rightarrow m) \\ \vdots & \vdots & \ddots & \vdots \\ C(m \rightarrow 1) & C(m \rightarrow 2) & \cdots & C(m \rightarrow m) \end{bmatrix} \quad (1)$$

Where $C(i \rightarrow j) \in \{0, 1\}$, $C(i \rightarrow j) = 1$ indicates a logic connection from start point i to end point j expressed by R2R path, while $C(i \rightarrow j) = 0$ means no connection exists. Each row i represents all the connections originating from scan cell i. Assuming the SC length is m, C is an $m \times m$ matrix that captures the R2R dependencies between all scan cells. This naming convention clearly reflects the start and end connection relationships for each scan cell, making it more intuitive for reliability analysis and FI applications. Besides, all scan chains in a design can be organized into a single C matrix.

The simple example shown in Fig. 3 depicts a mapping of physical connections on a SC, where scan cells (e.g., C5, C9, C21, C8, C88) are interconnected through combinatorial logic blocks. Each scan cell includes FF components with inputs (D, SI), control signal (SE), clock signal (CP), and an output (Q). Connections between cells are defined by R2R paths: for example, scan cell C5 at position 5 sends signals to cells C9 and C21, while C8 connects to C88. These links are represented in a binary matrix C as illustrated above, where a "1" at position $C(i \rightarrow j)$ indicates a physical connection from cell i to j, and "0" denotes no logic connection.

C. FPGA Emulation for Fast Data Acquisition

To speed up the analysis of DUT, we use an FPGA-based emulation platform. The SC-reconfigured netlist and some behavior files will be deployed to Xilinx FPGA to build the DUT as shown in Fig. 1. In addition, we need to rewrite the behavior models (e.g., all standard cells). The behavior models provided by the manufacturer usually contain some primitive syntax and some declarations that Vivado does not support. Therefore, we need to rewrite and match them with the netlist. Also, the platform uses SC to efficiently capture the write status and data of FF in each cycle and execute FI. The entire data acquisition process mainly includes two steps:

In the first step, the system extracts only the write states of the FFs through the SC's write operation tracing mode. For each system clock cycle to proceed, the write states of all the FFs are captured simultaneously and then scanned out. This process is repeated for the number of system clocks under analysis. This write state data is essential for analyzing the ACE intervals in the preliminary fault reduction analysis of FI SN flow (Algorithm 1 in Section III-D). In the second step, based on the DUP technique, FI is performed. The DUP method selects the fault to be injected at the cycle just before the last read cycle within an equivalent region, as any faults injected earlier in the cycle would have the same faulty effect until the read cycle occurs (Details will be explained in Section III-D2 with an example). Following the FI, the system captures both the write states of the FFs and the corresponding data after the fault has been injected. The SC's two modes are employed during this step: one mode extracts the write state, while the other captures the data post-fault injection. This enables a detailed evaluation of the fault effects, including identifying any masked faults (Algorithm 2 in Section III-D).

By combining FPGA emulation with SC-based FI and the two-step data extraction process, Tenpura enables efficient fault reduction analysis. The initial analysis of ACE intervals and the subsequent analysis of DUP-based fault reduction, significantly expediting the DUT's reliability analysis.

D. Fault Injection Scope Narrowing Flow

The FI SN process consists of two main components: (1) ACE analysis and (2) DUP-based fault reduction. ACE analysis relies on the data extracted in step 1 of the aforementioned FPGA emulation process, referred to as scan write states WC. The implementation of DUP-based fault reduction depends on the data obtained after FIs in step 2, referred to as scan FF write states WC_{fi} and scan data DC_{fi}. Also, a matrix DC with the same dimension as WC is defined to represent the actual data stored in each scan FF, which is used as a golden reference to compare the fault injected scan data DC_{fi}.

1) ACE Interval Analysis for Fault Reduction: Algorithm 1 depicts the ACE analysis procedure, aiming to optimize FI by identifying time intervals where injected faults have no impact on system behavior. Algorithm 1 analyzes FF write states extracted by SC and searches non-impact intervals for every startpoint FF based on its endpoints' behavior.

Initially, a matrix ACE_m with all ones (i.e., no unACE intervals) is constructed, where each row corresponds to a scan FF in the design, and each column corresponds to a specific clock cycle during application execution in sequential order. Then, the algorithm iterates through each scan FF as the startpoint and retrieves its corresponding endpoints from the connection matrix C as mentioned in Section III-B. These

Algorithm 1 ACE Analysis for FI Scope Narrowing

Require: Connection matrix C, scan chain states $WC(i,t)$, number of scan FFs m, total cycles T
Ensure: Searching ACE intervals for fault reduction

Initialize ACE matrix ACE_m of the same size as WC to all ones
for each FF i from 1 to m **do**
  Find endpoints $Endpoints(i) \leftarrow \{j \mid C(i \rightarrow j) = 1\}$
  if $Endpoints(i) \neq \emptyset$ **then**
    Initialize list $ZeroIntervals \leftarrow []$
    $start = -1$ ▷ Track start of global zero interval
    for each cycle t from 1 to T **do**
      if $\forall j \in Endpoints(i), WC(j,t) = 0$ **then**
        if $start = -1$ **then**
          $start \leftarrow t$ ▷ Start of zero interval
        end if
      else
        if $start \neq -1$ **then**
          Add $(start, t-1)$ to $ZeroIntervals$
          $start \leftarrow -1$
        end if
      end if
    end for
    if $start \neq -1$ **then**
      Add $(start, T)$ to $ZeroIntervals$
    end if
    for each zero interval $(n, n+k) \in ZeroIntervals$ **do**
      Find last $WC(i,t) = 1$ in range $(1, n-1)$, set t_{last}
      if t_{last} exists **then**
        Mark interval $(t_{last}+1, n+k)$ as unACE intervals
        $ACE_m(i, t_{last}+1 : n+k) \leftarrow 0$
      end if
    end for
  end if
end for
return ACE matrix ACE_m, and $Endpoints$ array.

endpoints represent the FFs can be influenced by the current FF (i.e., startpoint) in the SC. If an FF has no endpoints, it is skipped. Otherwise, for each scan FF with valid endpoints, the algorithm scans all time cycles using the FF write states WC to identify contiguous time intervals. The notation $WC(i,t)$ for scan write state, can be expressed as follows:

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

$WC(p,t) \in \{0,1\}$ becomes 0 when all endpoint FFs starting from the p-th scan cell are in hold operation at time t. Conversely, if at least one of those endpoint FFs is in the write state at time t, $WC(p,t)$ is 1. The first dimension p represents the SC index, with $1 \leq i \leq m$. The second dimension t represents time (i.e., system cycle), ranging from 0 to n. This notation expresses the state transitions of the SC over time.

In the search process, the algorithm searches for all endpoint FFs that remain in a zero state. These zero intervals indicate periods when no write operations occur at the endpoints, meaning a fault injected in the originating FF (i.e., startpoint) may be masked. The algorithm records each detected zero interval as $(n, n+k)$. Once all zero intervals are determined, the algorithm examines each interval $(n, n+k)$ and searches for the last occurrence of a write operation in the originating FF before n. This timestamp (t_{last}) marks the last meaningful

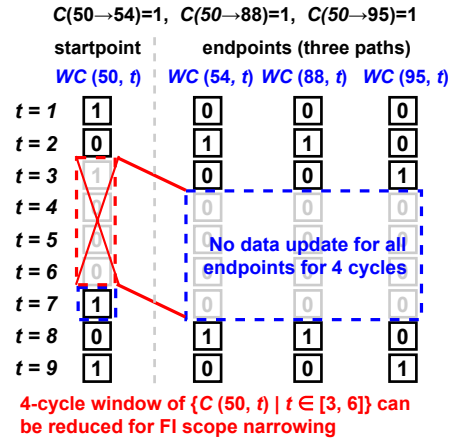

Fig. 4. Example of FI Scope Narrowing for ACE Analysis.

operation before the zero interval. If such an operation exists, the interval $(t_{last}+1, n+k)$ is classified as an unACE interval, meaning that FIs during this period do not propagate and can be excluded from FI analysis. By systematically detecting and marking unACE intervals into matrix ACE_m, the algorithm refines the FI process, significantly reducing unnecessary FIs while preserving accuracy in fault coverage analysis.

A simple example is shown in Fig. 4 for better understanding, $WC(50,t)$ is the startpoint, with timing connections to three endpoints: $WC(54,t)$, $WC(88,t)$, and $WC(95,t)$. The connection matrix indicates that $C(50 \rightarrow 54) = 1$, $C(50 \rightarrow 88) = 1$, and $C(50 \rightarrow 95) = 1$, meaning that any transition at $WC(50,t)$ should propagate to these endpoints. However, in the highlighted window from $t = 4$ to $t = 7$, all endpoint states remain at zero and the last write operation of the startpoint is found at $t = 7$. This indicates that during the period ($t \in [3,6]$), the state transitions at $WC(50,t)$ do not propagate to any of its connected endpoints. Following the algorithm, this forms an unACE interval where no data updates occur at the endpoints, making it a candidate for FI scope reduction. By identifying such intervals, the FI scope can be refined to avoid non-impact FIs into scan cells that do not affect the design, thereby improving FI efficiency and accuracy in reliability analysis.

2) DUP Reduction Analysis for Fault Reduction: To further reduce the faults, Algorithm 2 is designed to analyze fault propagation by systematically identifying DUP faults. The essence of DUP lies in the observation that faults injected before the last use (read) cycle often produce the same effect. Therefore, instead of injecting faults at every possible cycle, DUP identifies the last cycle before the value is used, and injects only one representative fault at the beginning point, significantly pruning redundant FIs while virtually preserving the accuracy and comprehensiveness of fault analysis.

Algorithm 2 operates by iterating over all scan FFs and injecting errors at specific timing points, referring to matrix WC obtained by Algorithm 1, to evaluate their impact on system behavior. If these errors have no impact on the system and do not propagate along the R2R path, then it can be determined that these errors can be removed. In the analysis

Algorithm 2 DUP Reduction for FI Scope Narrowing

Require: Connection matrix C, scan chain write states $WC(i,t)$, scan chain data $DC(i,t)$, number of scan FFs m, total cycles T, ACE matrix ACE_m and $Endpoints$ array from Algorithm 1
Require: $InjectError()$ ▷ Function for FI via SC
Require: $GetWriteStateAndData()$ ▷ Function to fetch FF write state and data via SC
Ensure: Analyzing DUP faults after fault injection
1: Initialize final fault matrix $Total\_Faults \leftarrow ACE_m$
2: $DUP\_Reduced\_Faults \leftarrow \emptyset$
3: **for** $i = 1$ to m **do** ▷ Iterate over all startpoints
4: $t_s \leftarrow$ First occurrence of $WC(i,t) = 1$ ▷ t_s<T
5: **while** $t_s \neq \emptyset$ **do**
6: $t_e \leftarrow$ Next occurrence of $WC(i,t) = 1$ after t_s
7: **if** $t_e = \emptyset$ **then**
8: $t_e = T$ ▷ Next write point is T
9: **end if**
10: Define interval (t_s, t_e)
▷ Check if all $Endpoints$(i) in $(t_s + 1, t_e)$ are 0
11: **if** $\exists j$ such that $WC(j,t) \neq 0$ for some $t \in (t_s + 1, t_e)$ **then**
12: $InjectError(i, t_s)$ ▷ Bit-flip at (i, t_s)
13: Data acquisition (scan write states WC_{fi} and scan data DC_{fi}) $\leftarrow GetWriteStateAndData(t_s, t_e)$
14: **if** No changes (WC_{fi} == WC && DC_{fi} == DC) **then**
15: $DUP\_Reduced\_Faults(i) \leftarrow DUP\_Reduced\_Faults(i) \cup (t_s, t_e - 1)$
16: $Total\_Faults(i, (t_s, t_e - 1)) \leftarrow 0$
17: **end if**
18: **end if**
19: $t_s \leftarrow t_e$ ▷ Move to the next write point
20: **end while**
21: **end for**
22: **return** $DUP\_Reduced\_Faults$ for DUP statistics, and fault matrix $Total\_Faults$ with reduced faults.

process, for each scan cell i, the algorithm identifies the first occurrence of $WC(i,t) = 1$, denoted as t_s, which marks a write operation. It then searches for the next occurrence of $WC(i,t) = 1$, defining an interval (t_s, t_e) where t_e represents the next write occurrence. If no subsequent write is found, t_e is set to the final time step T. Within each identified interval (t_s, t_e), the algorithm examines whether some endpoint scan cells remain at one throughout the interval. If any endpoint has a non-zero state in $(t_s + 1, t_e)$, an error is injected into scan cell i at t_s. It should be noted that the FI here is to change the data of scan FF, that is, bit-flip, without changing the write state of scan FF. Then, the system runs, and the write state (WC_{fi}) and data (DC_{fi}) from t_s to t_e are collected via SC(s).

If the injected error leads to a change in write state or output data, the interval $(t_s, t_e - 1)$ is classified as an impactful interval, indicating that faults in this range are architecturally significant. Otherwise, the interval is categorized as a masked fault interval and can be removed, meaning that faults in this range are masked and do not propagate to affect system behavior. These masked faults are fused with the ACE_m matrix in Algorithm 1 to generate the final fault reduction matrix. By iterating through all scan cells and analyzing these intervals, the algorithm effectively differentiates between fault-masking and unACE scenarios, optimizing FI analysis and improving fault characterization in scan-based testing.

A simple example (Fig. 5) illustrates the DUP fault reduction process for FI using SC. It shows how a FI (bit-flip) affects data propagation and endpoint updates. The process

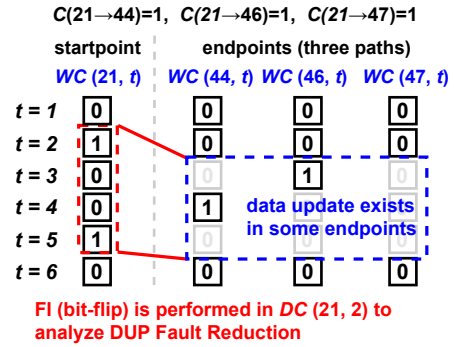

Fig. 5. Example of FI Scope Narrowing for DUP Reduction.

TABLE I
TYPICAL COMMANDS IN TENPURA.

Tenpura Command	Description
Init	Setting up parameters
MapFFToSCPos	Mapping FFs to SCs' positions
GetFFConnect	Retrieving FF connectivity details
SCofFFConnect	SC-represented FF connectivity details
ScanReconfiguration	Reconfiguring the SC cells
ACEAnalysis	Searching ACE intervals
DUPAnalysis	Evaluting DUP reduction

begins by identifying a starpoint where two consecutive states are one, in this case, from $t = 2$ to $t = 5$. Next, we check whether any endpoint registers a non-zero value (i.e., a write operation) within the interval ($t = 3$ to $t = 5$). As shown in the figure, WC(44,4) and WC(46,3) contain write operations, indicating that an FI is necessary. To perform FI, we flip the bit at DC(21,2) at $t = 2$ and allow the system to continue running, capturing the resulting values WC_{fi} and DC_{fi} till $t = 5$. Finally, we compare the outputs. if WC is equal to WC_{fi} and DC is equal to DC_{fi} from ($t = 3$ to $t = 5$), the entire starpoint interval from $t = 2$ to $t = 5$ can be considered a removable fault, as it does not affect system behavior. This approach optimizes fault analysis by filtering out inconsequential faults, improving efficiency.

E. Main Commands of Tenpura Platform

To implement the functionalities described above, we have developed a set of dedicated commands using Tcl and C++, as summarized in Table I. These commands facilitate system initialization, FF connectivity analysis, SC reconfiguration, and FI reduction. The $Init$ command initializes system parameters and configurations, ensuring that all necessary variables and data structures are correctly configured for subsequent operations. The $MapFFToSCPos$ command establishes the mapping between scan FFs and SC positions, ensuring proper alignment of FFs within the SC for FI testing, and structural analysis. The $GetFFConnect$ command retrieves connectivity details between FFs expressed by specific R2R paths, providing insights into their interconnections within the system, which is essential for FF-to-SC-position mapping. The $SCofFFConnect$ command further refines the connectivity analysis by representing FF connections in SC-based format, making FI and fault propagation dependencies more explicit. The $ScanReconfiguration$ command is used to reconfigure the SC structure to enable extraction of write status and data

TABLE II
DUT SUMMARY.

	Humming Bird E203	Tiny RISC-V	NVDLA-based Accelerator
FF number	11775	5025	61393
Application	Dhrystone (10 runs)	CoreMark (1 run)	LeNet5 (MNIST) (1 run)
Execution Cycle	19,408	529,513	83,506

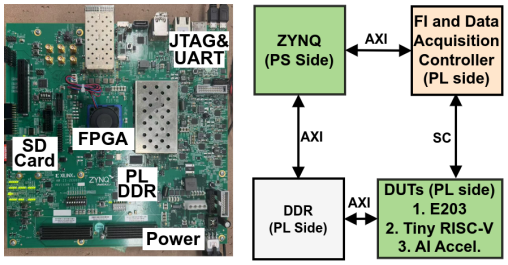

Fig. 6. Experimental Setup for Tenpura Evaluation.

of FFs. The $ACEAnalysis$ command identifies ACE intervals that represent critical time windows where faults affect system behavior, thereby improving the accuracy of FI scenarios. The $DUPAnalysis$ command evaluates DUP reduction and further analyzes the errors that can be reduced. Overall, these commands establish a systematic framework for FI analysis, improving fault localization, reliability analysis efficiency.

IV. EVALUATION OF TENPURA

A. Experimental Setup

To evaluate the effectiveness of Tenpura, we conduct experiments on three DUTs: Hummingbird E203 [21], TinyRISC-V [22] and a downsized NVDLA-based AI accelerator [23] featuring a 16×16 INT8 processing element array. These DUTs cover a broad spectrum, from lightweight RISC-V processors to complex AI accelerators, ensuring a comprehensive assessment of Tenpura's FI and fault reduction analysis capabilities.

The experimental setup for Tenpura is implemented on an FPGA board (Xilinx ZCU102) as shown in the Fig. 6, integrating multiple components to facilitate FI and analysis. The system consists of a ZYNQ SoC, where the Processing System (PS) side communicates with the Programmable Logic (PL) side via AXI interfaces. The DUTs are synthesized and routed using the DFT, gate-clock, and the proposed scan-based activity tracing flow, where the reconfigured SCs are automatically inserted to facilitate FI and fault reduction analysis. Table II summarizes the DUTs in terms of scan FF count, benchmark applications, and execution cycles. E203 includes 11,775 FFs and runs Dhrystone (10 runs) in 19,408 cycles. Tiny RISC-V has 5,025 FFs, executing CoreMark (1 run) in 529,513 cycles. The NVDLA-based accelerator contains 61,393 FFs and runs LeNet5 on MNIST (1 run) in 83,506 cycles. Also, these three DUTs (Back-end netlists with SC reconfiguration) are implemented on the PL side running at 50MHz. We construct only one SC for a fair evaluation. Besides, a simple FI and data acquisition controller, developed by Verilog, is responsible for injecting faults, collecting WC and DC information as mentioned in Section III-D2, and enabling cycle-by-cycle monitoring of system behavior.

TABLE III
TIME OVERHEAD FOR TENPURA FRAMEWORK STEPS.

Tenpura Step (unit: minute)		Humming Bird E203	Tiny RISC-V	NVDLA-based Accelerator
FF-SC mapping*		319.76	304.12	337.94
SC reconfiguration*		326.71	122.95	14937.40
ACE	**Acquisition**	0.08	0.89	1.71
	Analysis	92.29	552.58	4864.14
DUP	**Fault Injection**	47.87	205.33	5466.58
	Acquisition	151.74	581.08	13393.13
	Analysis	105.41	597.52	4050.30
Total		1043.86 (∼1 day)	2364.47 (∼2 days)	43051.2 (∼1 month)

*Only executed once during the ASIC design process.

To support pruned FI, the ACE analysis and DUP reduction, discussed in Section III-D, are employed. The entire system is controlled via a host computer, which interacts with the ZYNQ SoC through JTAG & UART interfaces to configure the FI and data acquisition controller. The fault analysis results are stored in SD card by PS. Additionally, the DDR memory module on the PL side is used to store intermediate computation results in the AI accelerator. This hardware-based experimental setup allows for a comprehensive assessment of Tenpura's FI and fault analysis capabilities, validating its efficiency in handling SEUs across different DUT architectures.

B. Runtime Analysis in Tenpura

Table III shows the time overhead for each step in the Tenpura framework across three different DUTs. The workflow consists of several stages: FF-SC mapping, SC reconfiguration, ACE analysis, and DUP analysis. For a fair evaluation, all steps requiring desktop CPU computation are executed on an Intel i9-10850K processor. The FF-SC mapping and SC reconfiguration steps are preliminary processes that establish the relationship between FFs and scan chains. These steps are essential to enable accurate FI and analysis later on. However, they are only required once during the ASIC design process, which means their cost is amortized over the entire lifetime of the design. For instance, SC reconfiguration for the NVDLA-based accelerator takes significantly longer than for the other DUTs due to its large scale and more complex scan structure.

The ACE phase consists of two parts: acquisition and analysis. Acquisition is performed using FPGA platforms to collect runtime execution data under normal operation. This step is relatively fast, taking less than 2 minutes for all DUTs. The analysis phase, however, is conducted on a desktop workstation and consumes most of the time (552.58 minutes for Tiny RISC-V, 92.29 minutes for HummingBird E203, and 4864.16 minutes for NVDLA-based accelerator). The DUP step requires FI, data acquisition and analysis. The analysis phase of DUP includes both the analysis of FI locations and timing and the examination of the collected data. This stage identifies and eliminates masked faults by injecting faults into the system and observing whether they have distinct effects. DUP introduces a substantial execution time, especially for larger designs. For instance, FI for the NVDLA-based accelerator takes 5466.58 minutes, with acquisition and analysis times of 13393.13 and 4050.30 minutes, respectively.

TABLE IV
FAULT RATIO WITH FAULT REDUCTION ANALYSIS.

	Humming Bird E203	Tiny RISC-V	NVDLA-based Accelerator
Total Fault	25,498,784	222,394,107	625,737,938
unACE	12,347,810 (48.43%)	78,444,347 (35.27%)	286,614,371 (45.80%)
DUP	12,196,068 (47.83%)	122,583,632 (55.12%)	267,127,536 (42.69%)
Remained Fault	954,906 (3.74%)	21,366,128 (9.61%)	71,996,031 (11.51%)

In summary, the total time is approximately 1043.86 minutes (∼1 day) for HummingBird E203, 2364.47 minutes (∼2 days) for Tiny RISC-V, and 43051.2 minutes (∼1 month) for the NVDLA-based accelerator. However, this execution time is entirely acceptable within the overall ASIC design process.

C. Fault Reduction Results

Table IV presents the fault analysis results for the DUTs shown in Table II. Initially, each design is subjected to a large number of total possible faults. However, by applying ACE analysis, a significant portion of faults is identified as unACE (i.e., faults that do not affect the architectural state and therefore do not need to be injected). For example, in the HummingBird E203, 48.43% of the total faults are identified as unACE, while the NVDLA-based accelerator and Tiny RISC-V have 45.80% and 35.27% unACE faults, respectively. Subsequently, DUP reduction is performed to eliminate masked faults that would otherwise produce the same effects. This further reduces the number of FIs. For instance, DUP analysis removes 47.83% of the faults in E203, 55.12% in Tiny RISC-V, and 42.69% in the NVDLA-based accelerator.

As a result of these two steps, the final set of faults that actually need to be injected for accurate evaluation is dramatically reduced. Specifically, only 3.74% of the original faults remain in HummingBird E203, 9.61% in Tiny RISC-V, and 11.51% in the NVDLA-based accelerator. This substantial reduction highlights the effectiveness of combining ACE and DUP analyses to focus FI efforts on only the architecturally and functionally significant cases, thereby saving considerable simulation time and computational resources.

D. Comparison with Fault Evaluation SOTAs

Compared to prior fault evaluation approaches as shown in Table V, Tenpura demonstrates significant advantages in both efficiency and practicality. In [7], although FI is performed through simulation, it requires extremely long runtimes (approximately 1500 years, scaled to the same CPU, i9-10850K) if one intends to inject faults across all FFs in a full application cycle. In [6], while fault reduction is supported, the evaluation is conducted solely on small-scale applications without considering large or complex workloads, making the evaluation incomplete and inaccurate. When scaled to full-application coverage to the same desktop CPU, this work would still require approximately 35 years to complete FI and reduction analysis. The above two SOTAs are mainly limited by the impact of software simulation efficiency. In [13], a FI platform based on software-hardware co-design can effectively solve this problem. However, due to the lack of hardware-software-based ACE and DUP fault analysis, it cannot effectively filter out errors that have no impact. It can only perform comprehensive FI to analyze system reliability, so the cost also takes about 1 year. In contrast, our proposed software-hardware co-designed approach achieves comparable FI and fault reduction analysis within just one month ranging from lightweight MCU designs to complex AI accelerators, offering orders-of-magnitude improvement in analysis time while maintaining high efficiency and broad applicability.

TABLE V
COMPARISON WITH FAULT EVALUATION SOTAS.

	DAC'24 [7]	ACE-Pro [6]	HachiFI [13]	This work
Technique	Simulation (VCS)	Simulation (Verilator)	Software-Hardware Co-Design	**Software-Hardware Co-Design**
FI Mechanism	Backdoor Access	Backdoor Access	Scan Chain	**Reconfigured Scan Chain**
FI Efficiency	Low	Low	High	**High**
Fault Reduction	No	Yes[1] (98.91-99.91%)	No	**Yes (88.49-96.26%)**
FI/Reduction Analysis Cost	1500 years[2] (AI Accel.)	35 years[2] (PicoRV32)	1 year[2] (AI Accel.)	**<1 month (TinyRISC-V, E203, AI Accel.)**
Target Designs	Netlist Designs	RTL Designs	Netlist Designs w/ SC(s)	**Netlist Designs w/ SC(s) and Gate Cells**

[1]Only simple applications are evaluated (e.g., String Search, Qsort).
[2]Estimated time for full-coverage FI and/or analysis of the application.

V. CONCLUSION

This paper presents Tenpura, a novel fault evaluation platform that enhances the efficiency and accuracy of system reliability analysis. By addressing the limitations of traditional FI methods, Tenpura introduces a scan-based activity tracing mechanism, an FI scope narrowing with obtained activity traces, and an FPGA-based emulation platform to significantly improve fault evaluation processes. The activity tracing mechanism enables precise FI operations and cycle-by-cycle tracking of FF write states. The FI scope narrowing based on ACE and DUP analysis further enhances efficiency by systematically filtering out faults that do not impact system behavior, reducing unnecessary FIs. Additionally, FPGA emulation accelerates data acquisition, allowing rapid validation of fault propagation effects. Experimental results demonstrate that Tenpura achieves ultra-fast reliability analysis (<1 month ranging from lightweight MCU designs to complex AI accelerator design) while maintaining comprehensive fault coverage, making it a promising solution for large-scale fault injection studies.

ACKNOWLEDGMENT

This work was supported in part by the Grant-in-Aid for Scientific Research (S) from Japan Society for the Promotion of Science (JSPS) under Grant 24H00073, by JST CREST, Japan, under Grant JPMJCR19K5; the National Natural Science Foundation of China under Grant 62274081; Grant 2023QN10X177. Dr. Xiong's contribution to this work is limited to his interactions with Drs. Cheng and Hashimoto at Kyoto University, Kyoto, Japan.

REFERENCES

[1] P. Rech, "Artificial Neural Networks for Space and Safety-Critical Applications: Reliability Issues and Potential Solutions," in IEEE Transactions on Nuclear Science, vol. 71, no. 4, pp. 377-404, April 2024, doi: 10.1109/TNS.2024.3349956.

[2] J. Gutiérrez-Zaballa, K. Basterretxea, and J. Echanobe, "Evaluating single event upsets in deep neural networks for semantic segmentation: An embedded system perspective," in Journal of Systems Architecture, vol. 154, 103242, 2024.

[3] S. S. Mukherjee, J. Emer and S. K. Reinhardt, "The soft error problem: an architectural perspective," 11th International Symposium on High-Performance Computer Architecture, San Francisco, CA, USA, 2005, pp. 243-247, doi: 10.1109/HPCA.2005.37.

[4] T. Garrett, S. Roffe and A. George, "Soft-Error Characterization and Mitigation Strategies for Edge Tensor Processing Units in Space," in IEEE Transactions on Aerospace and Electronic Systems, vol. 60, no. 4, pp. 5481-5498, Aug. 2024, doi: 10.1109/TAES.2024.3393929.

[5] Q. Cheng et al., "Reliability Exploration of System-on-Chip With Multi-Bit-Width Accelerator for Multi-Precision Deep Neural Networks," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 70, no. 10, pp. 3978-3991, Oct. 2023, doi: 10.1109/TCSI.2023.3300899.

[6] 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, pp. 10-19, doi: 10.1109/ITC50571.2021.00008.

[7] Q. 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 –," 2024 61st Design Automation Conference (DAC).

[8] C. Bolchini, L. Cassano, A. Miele and A. Toschi, "Fast and Accurate Error Simulation for CNNs Against Soft Errors," in IEEE Transactions on Computers, vol. 72, no. 4, pp. 984-997, 1 April 2023, doi: 10.1109/TC.2022.3184274.

[9] Q. Xu, M. Tanvir Arafin and G. Qu, "Security of Neural Networks from Hardware Perspective: A Survey and Beyond," 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 2021, pp. 449-454.

[10] 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.

[11] S. Laskar, M. H. Rahman and G. Li, "TensorFI+: A Scalable Fault Injection Framework for Modern Deep Learning Neural Networks," 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Charlotte, NC, USA, 2022, pp. 246-251, doi: 10.1109/ISSREW55968.2022.00074.

[12] N. Khoshavi, C. Broyles, Y. Bi and A. Roohi, "Fiji-FIN: A Fault Injection Framework on Quantized Neural Network Inference Accelerator," 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 2020, pp. 1139-1144, doi: 10.1109/ICMLA51294.2020.00183.

[13] Q. Cheng, W. Liao, R. Zhang, H. Yu, L. Lin and M. Hashimoto, "HachiFI: A Lightweight SoC Architecture-Independent Fault-Injection Framework for SEU Impact Evaluation," 2025 Design, Automation & Test in Europe Conference (DATE), Lyon, France, 2025, pp. 1-7, doi: 10.23919/DATE64628.2025.10993139.

[14] 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," 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Waikiki, HI, USA, 2015, pp. 738-749, doi: 10.1145/2830772.2830829.

[15] H. Cho, S. Mirkhani, C. -Y. Cher, J. A. Abraham and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 2013, pp. 1-10.

[16] Z. -M. Huang, D. -A. Yang, J. -J. Liou and H. H. Chen, "FPGA-Based Emulation for Accelerating Transient Fault Reduction Analysis," 2022 IEEE 31st Asian Test Symposium (ATS), Taichung City, Taiwan, 2022, pp. 144-149, doi: 10.1109/ATS56056.2022.00037.

[17] A. Biswas, P. Racunas, J. Emer and S. Mukherjee, "Computing Accurate AVFs using ACE Analysis on Performance Models: A Rebuttal," in IEEE Computer Architecture Letters, vol. 7, no. 1, pp. 21-24, Jan. 2008, doi: 10.1109/L-CA.2007.19.

[18] R. Barbosa et al., "Assembly-Level Pre-injection Analysis for Improving Fault Injection Efficiency," in Dependable Computing–EDCC 5, Berlin, Heidelberg: Springer, 2005, pp. 246–262.

[19] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock and J. Haid, "Efficient fault emulation using automatic pre-injection memory access analysis," 2012 IEEE International SOC Conference, Niagara Falls, NY, USA, 2012, pp. 277-282, doi: 10.1109/SOCC.2012.6398361.

[20] J. Guthoff and V. Sieh, "Combining software-implemented and simulation-based fault injection into a single fault injection method," Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers, Pasadena, CA, USA, 1995, pp. 196-206, doi: 10.1109/FTCS.1995.466978.

[21] Y. Tong and Y. Xia, "Implementation of Hummingbird E203 Based Electric DC Motor Controller," 2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS), Changchun, China, 2023, pp. 1220-1223, doi: 10.1109/EIECS59936.2023.10435587.

[22] J. Zhou, G. Qin, L. Li, C. Guo and W. Wang, "ISA Extensions of Shuffling Against Side-Channel Attacks," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 3, pp. 761-773, March 2024, doi: 10.1109/TCAD.2023.3323165.

[23] F. Farshchi, Q. Huang and H. Yun, "Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim," 2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), Washington, DC, USA, 2019, pp. 21-25, doi: 10.1109/EMC249363.2019.00012.