Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 15, No. 3, 2026

Research on Fault Injection based on Lockstep Architecture Processors

Jie Lei *

Xi'an Shiyou University, Xi'an Shaanxi, 710000, China
*** Corresponding author:** Jie Lei

Abstract: This paper focuses on a lockstep architecture processor and employs a software-based fault injection method to construct a system-level verification model. By implementing data consistency fault injection and high-concurrency stress testing, the processor's loss-of-lock detection mechanism and operational stability were comprehensively evaluated. The test results successfully verified that the lockstep comparator can accurately identify and report data discrepancies, and does not generate false alarms under extreme loads. This proves that both its data error tolerance and system anti-interference capabilities meet design expectations, providing a key verification basis for the chip's functional safety.

Keywords: Software Fault Injection; Lockstep Architecture Processor; System-Level Verification.

1. Introduction

The increasing complexity in aviation and the growing demand for high reliability necessitate advanced fault-tolerant solutions. The lock-step function in lock-step architecture processors, which employs multi-core redundancy for parallel execution and real-time result comparison, significantly enhances system fault tolerance and safety margins. This technology is critical for ensuring the reliable operation of key functions such as flight control and navigation. As the core computational unit of modern avionics systems, the functionality of processor chips directly impacts flight safety and mission reliability[1]. Therefore, strict verification of the functional integrity, failure detection effectiveness, and system-level reliability of lock-step architecture processors is essential. Traditional testing methods often struggle to cover potential fault modes under complex operating conditions. In this context, fault injection technology, which actively introduces simulated faults, has become a core method for evaluating system fault detection, diagnosis, and recovery capabilities. It provides critical insights for design optimization and system robustness validation.

2. Common Fault Injection Methods for Processors

(1) Simulation-Based Fault Injection

Simulation-based fault injection is a key technique widely used in processor design verification and system reliability assessment. This method involves artificially introducing error states into a processor's simulation model (such as cycle-accurate RTL-level models or system-level architectural simulators) to simulate various types of faults that may occur in physical hardware[2]. This allows for the evaluation of the effectiveness of fault-tolerant mechanisms at an early stage, prior to chip fabrication.

Its core lies in leveraging the controllability and observability of software models to precisely interfere with the internal states of the model. Common technical approaches include establishing bit-flip models to simulate the impact of transient faults like single-event upsets on storage units (e.g., registers, caches, memory); forcibly altering logic values of internal data paths, control signals, or clock networks through signal tampering to simulate logic errors or timing violations; and manipulating instruction streams to simulate fetch or decode errors. These injection actions can be triggered dynamically during simulation runtime or statically implanted at specific time points.

The significant advantage of this method is its high degree of controllability and repeatability. It allows researchers to conduct large-scale, exhaustive fault scenario testing without relying on physical hardware and at a very low cost, making it particularly suitable for vulnerability analysis and fault-tolerant algorithm prototyping in the early stages of design iteration. However, its effectiveness is fundamentally limited by the functional and timing accuracy of the simulation model itself. The higher the level of model abstraction, the greater the potential deviation from the actual behavior of the final silicon chip. Therefore, simulation-based fault injection is generally regarded as a crucial preliminary step in the reliability assessment workflow. Its conclusions often need to be combined with verification methods closer to the hardware to form a complete evidence chain for evaluation.

(2) Hardware-Based Fault Injection

Hardware-based fault injection is a verification method that directly interferes with an actual operating processor chip through physical means to induce functional or timing errors within it. Since this method acts directly on the silicon hardware, the observed system responses and failure modes possess the highest degree of authenticity and credibility. It is often regarded as the final verification benchmark for evaluating a processor's resistance to interference in high-reliability applications (such as aerospace, automotive electronics, and critical infrastructure).

Its implementation primarily relies on sophisticated experimental equipment and physical intervention techniques. Typical technical approaches include using heavy ions or pulsed lasers to irradiate specific areas of the chip to study its susceptibility to single-event effects[3]; disrupting its power integrity and timing through near-field electromagnetic interference or directly injecting voltage glitches into power pins; and employing invasive methods like micro-probing to directly manipulate the electrical levels of the chip's internal

nodes. These means can accurately simulate various transient and permanent faults caused by space radiation, electromagnetic pulses, and harsh operating environments on a nanosecond timescale.

The fundamental advantage of this method lies in its ability to reveal the most intrinsic vulnerabilities of the processor under real silicon processes and physical effects. Its results are crucial for defining the device's safe operating area and verifying the effectiveness of its built-in fault-tolerant mechanisms (e.g., lockstep cores, ECC). However, its limitations are also significant: experiments typically require expensive equipment like particle accelerators, laser positioning systems, or precision fault injection devices, and the procedures are complex and time-consuming; physical intervention on the chip can be destructive[4]; furthermore, its controllability and observability over the processor's deep internal nodes (such as critical signals not routed to the package pins) remain limited. Therefore, hardware-based fault injection is generally considered the "gold standard" in the reliability assessment workflow. However, due to its cost and complexity, it is mostly used in the certification phase of critical systems or for calibrating and correcting higher-level simulation models, thereby constructing a complete and credible evaluation chain from virtual to physical entities.

(3) Software-Based Fault Injection

Software-based fault injection is an efficient verification method that simulates the effects of hardware faults by executing specific software programs while the processor system is running. This approach does not rely on physical hardware or underlying simulation models. Instead, it creates system-level error states—similar to those caused by physical faults such as single-event upsets or timing violations—through controlled interference at the operating system or application level. This enables the evaluation of the behavior and resilience of the target software stack and system services under abnormal conditions.

Its core implementation mechanism involves the dynamic manipulation of the processor's visible state via software interfaces. Key technical approaches include using debugging and performance monitoring units (e.g., via JTAG, ARM Core Sight, or Intel PT) for direct read/write operations on runtime registers and memory states; inserting instrumentation code into critical paths of the target application or operating system kernel to trigger errors such as data corruption, control-flow hijacking, or resource exhaustion; and performing injection at the virtualization layer or hypervisor level to assess the cascading impact on multiple guest operating systems or the entire virtualization platform[5]. These operations can be triggered after precise instruction cycles or specific memory access events, achieving high controllability and repeatability of fault scenarios.

The prominent advantages of this method are its extremely low implementation cost, high flexibility, and seamless integration with large-scale test automation processes. It makes large-scale, statistically significant reliability testing on mass-produced hardware feasible, and is particularly suitable for evaluating fault-tolerance algorithms and recovery mechanisms at the operating system, middleware, and application layers. However, its fundamental limitation lies in the relatively high abstraction level of the fault model. The injected faults are confined to the processor state visible through the Instruction Set Architecture (e.g., general-purpose registers, memory) and cannot directly simulate deep-seated faults at the microarchitecture level (e.g., inside execution units, pipeline registers, predictor states). Its coverage inherently has boundaries. Therefore, software-based fault injection is generally regarded as a crucial tool for system-level reliability assessment and robustness testing. Its conclusions are often complemented by results from lower-level hardware or simulation-based injections, collectively forming a complete reliability evidence system spanning from micro to macro perspectives.

(4) Hybrid Fault Injection

Hybrid Fault Injection is a systematic methodology that aims to construct a multi-level, cross-abstraction collaborative verification framework. It overcomes the inherent limitations of individual methods—such as coverage, fidelity, and efficiency—by systematically integrating simulation-based, hardware-based, and software-based fault injection techniques. Its core philosophy is not merely the parallel application of these techniques, but rather emphasizes closed-loop interaction and mutual calibration among injection methods at different abstraction levels[6]. This enables a more comprehensive and credible assessment of the reliability of complex processor systems.

The typical technical workflow of this framework embodies an iterative and synergistic process. First, simulation-based injection is used for large-scale, rapid preliminary screening of vulnerabilities and localization of sensitive points within high-level models, generating a guiding fault list. Subsequently, based on this list, hardware-based injection is employed on FPGA prototypes or physical chips to perform physical verification of key sensitive units, obtaining authentic electrical and timing fault response data. Simultaneously, software-based injection utilizes fault signature patterns (e.g., error manifestations following specific instruction sequences) collected from hardware injection or Performance Monitoring Counters (PMCs) to conduct high-frequency, targeted testing at the operating system and application levels, evaluating system-level fault tolerance and recovery mechanisms. Continuous data exchange and model calibration occur between these levels. For instance, hardware injection results are used to correct the fault propagation parameters of simulation models, and the refined models then guide the generation of more efficient software injection scenarios.

The fundamental advantage of this method lies in its ability to construct a complete verification evidence chain spanning from microarchitecture to system software, and from virtual simulation to physical implementation. It maintains the efficiency of large-scale testing while ensuring the credibility of results by anchoring them to real hardware behavior, thereby providing higher assessment confidence for design verification, safety-critical system certification, and resilience evaluation. However, its implementation also faces significant engineering complexity challenges. These include the need to develop a unified fault description language and cross-platform management tools, integrate heterogeneous toolchains and data formats, and coordinate simulation and testing processes operating at different time scales. Consequently, Hybrid Fault Injection represents a systems engineering philosophy that pursues an optimal balance between assessment completeness and efficiency. Its effective implementation relies on meticulous upfront framework design and the standardization of interfaces between the various levels.

3. Lockstep Mechanism of Lockstep Architecture Processors

The lockstep mechanism of lockstep architecture processors is a closed-loop fault-tolerant process designed with the core objective of ensuring computational reliability. Its operation begins with clock alignment and signal delay compensation for external input instructions, guaranteeing strict synchronization of the instruction streams fed to the dual cores. Under the premise of this input synchronization, the two processor cores execute identical instructions in locked step.

Upon completion of execution, the system performs real-time comparison of the outputs from both cores. This constitutes the core detection stage of the mechanism. Only when the output results from the dual cores are entirely consistent will the system deem the computation correct and output the final result[7]. If the input synchronization stage fails, or if any inconsistency is detected during the output comparison phase, the mechanism will determine a "loss of lock," immediately trigger an interrupt, and report error information, thereby preventing the propagation of erroneous results.

Through mandatory synchronized execution and result comparison, the entire mechanism achieves real-time detection and isolation of transient faults, forming a complete reliability assurance closed loop that spans from input and computation to output.

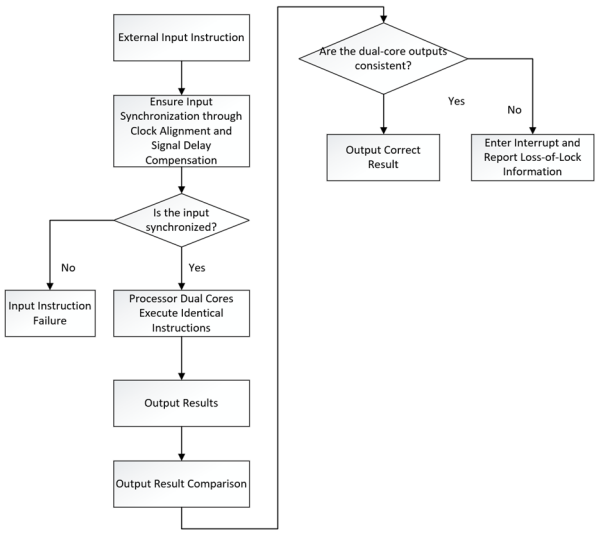

Figure 1. Lockstep Flowchart

4. Construction of the Fault Model

When conducting comprehensive loss-of-lock verification for all functional modules of a lockstep architecture processor, this solution adopts a software-based fault injection method to construct the core fault model, aiming to achieve high test coverage for the numerous internal buses and functional modules of the chip.

However, in the verification of the lockstep mechanism via software-based fault injection, due to its inability to directly manipulate underlying hardware timing, it is not feasible to construct precise timing window timeout faults[8]. Therefore, the developed fault model focuses on the data consistency dimension and system-level stress testing, with the goal of systematically validating the comparison logic and overall stability of the lockstep architecture through software-triggered fault conditions.

Table 1. Lockstep Verification Fault Model

Verification Dimension	Test Category	Core Scenario Description	Injection Method and Trigger Condition
Data Consistency	Generic Bus Comparison	Any mismatch in the PAYLOAD between channels X and Y.	Forcefully modify the payload of either channel by writing to registers via software.
Data Consistency	Generic Bus Comparison	The PAYLOAD of channels X and Y is completely consistent	Execute various bus transactions in normal mode.
System Stability	Contention Access Stress	High-load concurrent DDR access by multiple master devices (A664/CPU/DMA)	Simultaneously activate the A664 network engine, multiple CPU cores, and DMA controllers, and orchestrate a high-bandwidth, high-probability-of-conflict concurrent access sequence to perform continuous read/write operations on the shared DDR

5. Design and Implementation of the Fault Injection Platform

(1) Design and Implementation of the Hardware Platform Architecture

The hardware platform constructed in this research is an FPGA-based prototyping system designed for the verification of high-reliability lockstep processors. Its core objective is to accurately implement the target avionics processor's lockstep architecture and its complete System-on-Chip (SoC) runtime environment on programmable logic devices. This provides the physical foundation for software fault injection, loss-of-lock scenario construction, and precise performance testing.

The platform adopts a hierarchical core-periphery design approach. At its core, integrated within the FPGA, is the target

lockstep processor system[9]. This system comprises the dual-core processor forming the lockstep pair, various levels of cache, and an on-chip interconnect network linking all subsystems. Surrounding this core, the platform integrates essential peripheral support circuits. These include Flash memory for program storage and non-volatile data, DDR3 SDRAM serving as the main operating memory, low-latency on-chip SRAM, and an A664 end-system module for implementing specific avionics communication protocols. Furthermore, the platform is equipped with comprehensive debugging and monitoring interfaces, such as JTAG and a dedicated debug link (CK-Link Lite), used for system initialization, control, status monitoring, and real-time data acquisition.

The entire platform's design strictly adheres to two primary principles: modularity and observability[10]. Modularity is reflected in the clear functional partitioning and interface definition of the processor cores, the lockstep comparison unit, various memory controllers, peripheral IPs, and the debug subsystem. This not only facilitates initial integration and verification but also enables subsequent testing and fault injection targeting specific lockstep points. Observability is crucial for achieving the research objectives of this thesis. By reserving abundant internal state access points during the hardware design phase and routing all critical signals to debug interfaces or status registers[11], it ensures the ability to monitor and record the processor's internal behavior and performance metrics non-intrusively while test programs are running. This satisfies the verification requirements for constructing scenarios ranging from simple data comparison errors to complex contention access scenarios.

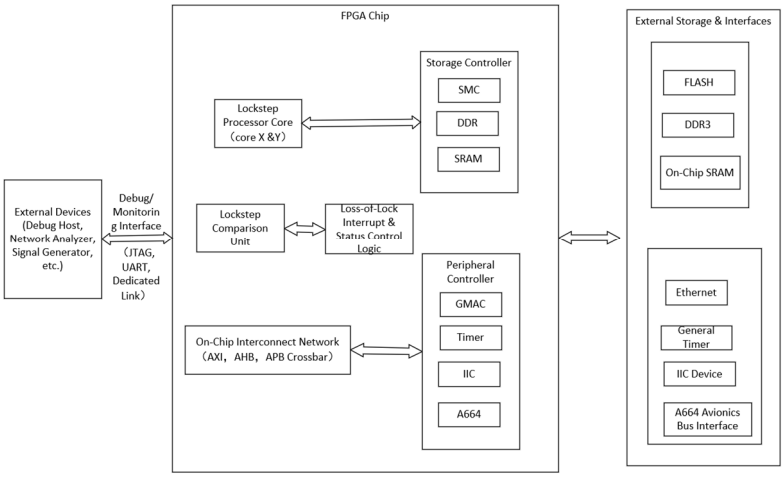

Figure 2. FPGA Platform Architecture Diagram

(2) Design and Implementation of the Software System

To achieve systematic injection of loss-of-lock faults and performance evaluation for the lockstep processor, this research designed and implemented a layered, modular software testing system[12]. The overall architecture of the system, from top to bottom, can be divided into the test control layer, the functional module layer, and the hardware interface layer, which work in concert to accomplish automated testing tasks. The test control layer serves as the core scheduler, responsible for parsing test plans, serially executing test cases, and managing the entire test lifecycle. The functional module layer includes key modules such as fault injection control and result recording. Each module interacts with the test control layer through well-defined interfaces and communicates with the processor and peripherals on the FPGA platform via the hardware interface layer. The hardware interface layer encapsulates low-level access operations to processor-specific registers[13], memory-mapped I/O space, and debug ports, ensuring hardware independence and portability for the upper-layer software.

6. Implementation and Result Analysis of Software Fault Injection

(1) Implementation of Software Fault Injection

Environment and Target Preparation: Start the operating system and the driver under test on the hardware platform equipped with the lockstep processor, ensuring the lockstep function is enabled. Define the fault model to be verified (e.g., data consistency fault) and specify the precise injection target (e.g., a specific AXI bus transaction, memory address, or register).

Injection Point Selection and Triggering: Insert the fault at predetermined "injection points" within the target code flow via debug interfaces, kernel modules[14], or test programs. Common methods include dynamically modifying data words during transmission, flipping address bus bits, or tampering with control signals.

Fault Execution and Monitoring: After triggering the injection, the fault is executed along with the normal instruction stream. The system continuously monitors the lockstep status registers and predefined interrupt vectors to capture whether a "Data Comparison Loss-of-Lock" interrupt is generated, along with associated error information.

Result Verification and Analysis: Compare the monitored system response (e.g., interrupt trigger time, error codes) with the expected behavior. Analyze logs to confirm whether the fault was accurately detected and reported, and evaluate its impact on the integrity of system functionality.

(2) Collection of Test Results

The following key data was recorded.

Fault Detection Latency: The delay from fault injection to the triggering of the loss-of-lock interrupt was within 10 clock cycles, meeting the design specification.

Error Information Accuracy: The error address and channel identifier recorded by the interrupt service routine completely

matched the injected fault information.

Stress Test Metrics: During a continuous 30-minute stress test, the system processed 24 billion concurrent transactions without any data errors or loss-of-lock events, with an average bandwidth utilization reaching 85%.

Table 2. Test Results

Test Scenario Description	Expected System Response	Actual Observed Result	Test Conclusion
Any mismatch in the PAYLOAD between channels X and Y	Trigger the "Data Comparison Loss-of-Lock" interrupt	The interrupt was correctly triggered, and error information was recorded completely	Pass
The PAYLOAD of channels X and Y is completely consistent	No loss-of-lock interrupt	The system ran stably with no false alarms	Pass
High-load concurrent DDR access by multiple master devices (A664/CPU/DMA)	No loss-of-lock, normal functionality/performance	No loss-of-lock, business data integrity maintained, performance met targets	Pass

(3) Analysis of Test Results

The test results indicate that the processor's lockstep mechanism demonstrates a high degree of functional completeness and reliability within the defined verification scope. Regarding fault detection capability[15], all data inconsistency faults introduced via software injection were accurately captured and triggered interrupts, proving that its core comparison logic is functionally correct at the circuit level, and the error reporting pathway is complete and effective. In terms of anti-interference and stability, the system did not generate any false loss-of-lock alarms under high-concurrency stress scenarios, indicating that its synchronization and comparison mechanisms possess good robustness. They can tolerate timing disturbances caused by normal bus contention and scheduling delays without affecting system performance under functional safety requirements.

The reliability of this software-based fault injection test is built upon a clearly defined and controllable fault model. The test successfully verified the data domain consistency protection capability, confirming that this method is an effective means for large-scale functional module verification[16]. However, it is necessary to point out its verification boundaries: deep hardware behaviors such as timing window tolerance, which cannot be covered by software injection, require supplementation through lower-level verification methods. In summary, under the currently defined and accessible fault models, the processor's lockstep function has demonstrated reliable fault-tolerant characteristics that meet design expectations.

7. Conclusion

This study, through software-based fault injection, has verified that the lockstep architecture processor meets design expectations in terms of data consistency protection and system stability. Testing demonstrates that the lockstep mechanism can accurately detect and report data inconsistency faults, and exhibits no false loss-of-lock alarms under high-pressure contention access, confirming its effective fault detection and anti-interference capabilities. However, limited by the software injection approach, the verification of deeper hardware behaviors such as timing tolerance requires supplementation through lower-level methods. Overall, the lockstep processor performs reliably under the currently defined data-domain fault models, satisfying the fault-tolerant design requirements of high-reliability systems.

Acknowledgments

The authors gratefully acknowledge the assistance and support provided by Xi'an Shiyou University during the course of their research.

References

[1] Zhao G L, Jing M. Safety Testing of Radar Software Based on Fault Injection [J]. Information Technology and Informatization, 2024, (09): 179-183.

[2] Song J C, Liu X Z, Zhao C Y. Research on Fault Injection Methods for Embedded Software Testing [J]. Science and Technology & Innovation, 2024, (18): 166-168. DOI:10. 15913/ j.cnki.kjycx.2024.18.049.

[3] Zhang C Y, Wang M Y, Yu Z Y, et al. Design of a Low-Overhead Superscalar Dual-Core Lockstep Processor Architecture for Automotive Functional Safety [J]. Chinese Journal of Automotive Engineering, 2024, 14(02): 313-320.

[4] Bai J, Fu Z, Xie K, et al. Testing Error Handling Code With Software Fault Injection and Error-Coverage-Guided Fuzzing[J]. IEEE Transactions on Dependable and Secure Computing, 2024, 21(4): 1724-1739. DOI:10.1109/ TDSC. 2023. 3288874.

[5] Qiao B J, Liu B, Yi Z P, et al. Research on System Verification and Test Evaluation Technology Based on Software Fault Injection [J]. Aviation Maintenance & Engineering, 2023, (12): 36-40. DOI:10.19302/j.cnki.1672-0989.2023.12.012.

[6] Wang H, Tan Z H, Chen S Y. Implementation of Extreme Evaluation for Application Verification of Domestic Aviation Processors [J]. Aeronautical Computing Technique, 2023, 53(02): 101-104.

[7] Mao C, Xie Y, Wei X, et al. FPGA-based fault injection design for 16K-point FFT processor [J]. The Journal of Engineering, 2019, 2019(21): 7994-7997. DOI:10.1049/joe.2019.0703.

[8] Yao F, Shi G, Fu X W. Research on Software Testing Method at Spacecraft System Level Based on Fault Injection Technology [J]. Spacecraft Engineering, 2019, 28(01): 130-136.

[9] Qiu Z Y. Research on Safety Computer Testing Based on Fault Injection [D]. Beijing Jiaotong University, 2018.

[10] Zhang X C. Research on Fault Models of Embedded Systems Based on Dynamic Fault Trees [D]. Nanjing University of Aeronautics and Astronautics, 2017.

[11] Zhou X, Li P, Han Q. A Lockstep Processor Architecture Based on 60x Bus [J]. Aeronautical Computing Technique, 2015, 45(01): 127-130.

[12] Embedded Systems; Investigators from Technical University Target Embedded Systems (A fault-injection methodology for the system-level dependability analysis of multiprocessor embedded systems) [J]. Journal of Engineering, 2014.

[13] Jiang L J. Research on Digital IC Fault Models and Design of a Fault Injection Platform [D]. Harbin Institute of Technology, 2013.

[14] [Wang C, Fu Z C, Chen H S, et al. Low-Cost Lockstep EDDI: A Transient Fault Detection Mechanism for Processors [J]. Chinese Journal of Computers, 2012, 35(12): 2562-2572.

[15] Fu A Y, Zhou J J. Research on Fault-Tolerant Technology Based on Lockstep [J]. Science Mosaic, 2012, (07): 70-73.

[16] Sun R, Zhang T, Xiao D Y, et al. Research on Fault Injection for Fault-Tolerant Computers Based on FPGA [J]. Microcomputer Information, 2010, 26(14): 116-118