Functional Verification and Fault Modeling of a RISC-V Register File Using SystemVerilog

Camille Chang
Dept. of Electrical and Computer Engineering, University of California, Davis
tyuchang@ucdavis.edu

Chi-Yun (William) Wang
Dept. of Electrical and Computer Engineering, University of California, Davis
cywwang@ucdavis.edu

Yating Chang
Dept. of Electrical and Computer Engineering, University of California, Davis
cyating@ucdavis.edu

Zherong Yu
Dept. Electrical and Computer Engineering, University of California, Davis
zryu@ucdavis.edu

***Abstract*— We present the design, verification, and fault modeling of a RISC-V-compatible 32×32 register file using Verilog and SystemVerilog. The design enforces RISC-V constraints, including a hardwired-zero x0 register, and supports dual-read/single-write access. A SystemVerilog testbench with directed and randomized tests validates functionality and fault resilience. Fault injections (stuck-at, bit-flip) and coverage analysis reveal high fault detection rates, especially under multi-bit scenarios. Regression testing ensures robustness, demonstrating the value of fault-aware verification in RISC-V designs.**

Keywords*— *RISC-V, register file, functional verification, SystemVerilog, fault injection, bit-flip fault, stuck-at fault, coverage analysis, regression testing, golden model

I. INTRODUCTION

This project explores the functional verification and fault modeling of a RISC-V register file, a critical component in modern processor design. Built upon the RISC-V architecture—an open-source, modular instruction set architecture (ISA) based on the principles of Reduced Instruction Set (RISC)—this project benefits from RISC-V's flexibility, extensibility, and widespread applicability in computing systems. The register file designed in this project consists of 32 general-purpose 32-bit registers, with support for dual read ports and a single write port, adhering to the RISC-V constraint that register x0 is hardwired to zero. The project is structured into several key components: RTL design and parameterization of the register file, development of a SystemVerilog - based verification testbench, directed and randomized testing strategies, fault injection for both stuck-at and bit-flip faults, and automated coverage measurement and debugging. By combining design implementation with thorough verification and fault analysis, this project aims to demonstrate the robustness and reliability required in processor subsystems.

II. VERIFICATION ENVIRONMENT

To support the development and validation of our design, we established a two-stage test environment using industry-standard tools. EDA Playground was used for early-stage prototyping and iterative debugging, allowing rapid simulation of our SystemVerilog code and immediate waveform visualization. Once the testbench architecture and RTL design stabilized, we transitioned to Questa for full-scale verification. Questa enables comprehensive test execution, including random test generation, fault injection, and functional coverage analysis, providing advanced debugging features and reliable simulation performance. This hybrid setup ensured both agility during initial development and accuracy during final validation.

III. RTL DESIGN AND BASIC TESTBENCH DEVELOPMENT

A. RISC-V Reg File RTL Design

The register-file RTL was implemented in SystemVerilog as a fully RISC-V–compliant 32×32 storage array. The module is parameterized by a data width (WIDTH=32) and depth (DEPTH=32), corresponding to the 32 integer registers x0–x31. To enforce the RISC-V architectural requirement that register x0 always reads as zero, the design ignores any write attempts to address 0 and hard-wires read data at index 0 to zero. Two asynchronous read ports (rdata1, rdata2) deliver contents from the specified addresses immediately, while a single synchronous write port updates the storage on the rising edge of the clock. Importantly, "write-first" semantics ensure that a read and write to the same address within one cycle will return the newly written value rather than the old data.

B. Determinzed Test

A self-checking testbench was developed to validate each corner case prescribed by the RISC-V specification. Inside tb_regfile_cv.sv, a golden-model array of identical dimensions mirrors all writes under the same clocking conditions. On each clock edge when write_enable is deasserted, the DUT's outputs (rdata1, rdata2) are compared against the corresponding entries in golden_mem. Any mismatch triggers a runtime error, automatically flagging functional deviations.

We first applied four directed tests:

1. x0-pinning: Attempt to write a nonzero value into x0; subsequent read must return zero.

2. Basic write/read: Write a known pattern to a mid-range register (e.g., x5) and read it back in the next cycle.

3. Write-read collision: Simultaneously assert write and read on the same address (e.g., x7) within one cycle; the read ports must reflect the newly written data due to write-first semantics.

4. Multi-cycle overwrite: Perform back-to-back writes to the same register (e.g., x10), each with different data, then read and verify that the last value prevails.

All four scenarios produced the expected results with zero errors.

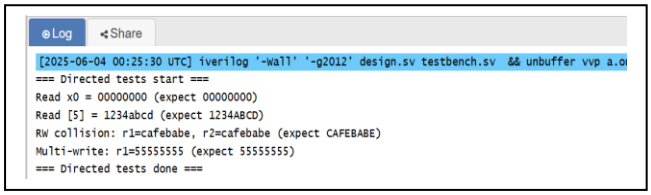

Fig.1 Directed Tests Results

C. *Randomized Test*

To increase confidence beyond hand-picked scenarios, we ran a randomized test sequence of 10 000 vectors. Each iteration generates random values for the write enable bit, write address, write data, and both read addresses. Over a sequence of 10 000 clock cycles, this exercise explores a broad cross-section of the design's state space. During this phase, the golden model is updated on every write, and the checker continuously compares DUT outputs on each read cycle. The absence of any $error messages over all 10 000 vectors confirms functional equivalence between the RTL and our reference model across a wide variety of stimulus patterns.

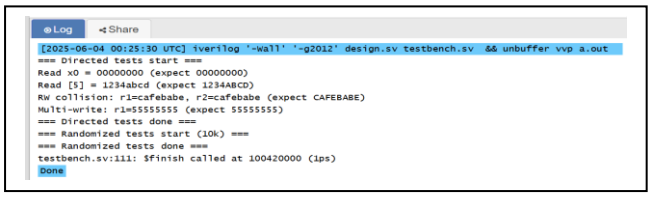

Fig.2 Randomized Tests Results

IV. FAULT INJECTIONS METHODOLOGY

To evaluate the robustness of the RISC-V register file against hardware-level errors, a fault injection framework was integrated into the verification environment. The methodology targets two primary fault types: **stuck-at faults** (stuck-at-0 and stuck-at-1) and **bit-flip faults**, with stuck-at-faults divided into single-bit tests and multi-bit tests.

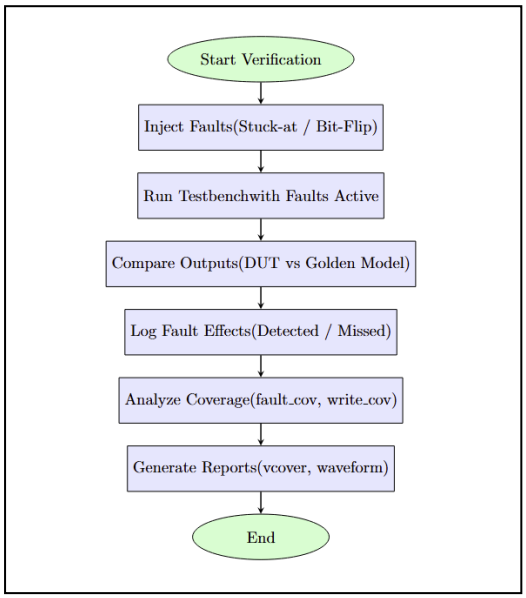

Fig.3 Fault Injection Methodology Flowchart

A. *Single-Bit Stuck-at-Faults*

Single-bit stuck-at fault injection targets common manufacturing defects where a logic signal is permanently stuck at '0' or '1'. In this methodology, each bit of every register (32 registers × 32 bits × 2 polarities = 2048) is individually forced to a constant value during normal operation, resulting in 2048 total fault scenarios. These faults were systematically injected one at a time while read and write transactions continued through the testbench. A golden model was used to compare outputs and identify any deviations caused by the stuck-at behavior. This method provides both deterministic and randomized coverage of all possible single-bit stuck conditions and is especially useful for uncovering faults that might be masked during normal functional tests. Coverage metrics such as fault_cov and write_cov were employed to track activation and detection across all register locations. For randomized tests, each number set of fault injections were run through multiple times and calculated their averages.

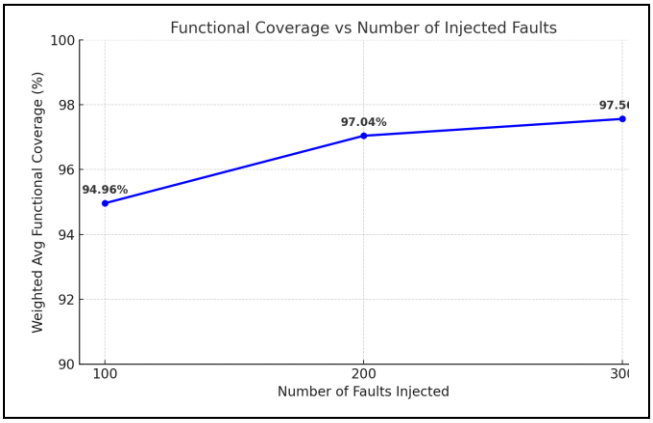

Fig.4 Line Graph of Functional Coverage for Randomized Tests

```
# Detected: 1060 / 2048
# Functional Coverage (2048 bins): 100.00%
# Missed faults:
# ** Note: $finish    : C:/questasim64_2024.1/examples/tb_regfile_all.sv(138)
#    Time: 40990 ns  Iteration: 0  Instance: /tb_regfile_all
```

Fig.5 Result of Deterministic Tests for Stuck-At-Faults

B. *Multi-Bits Stuck-at-Faults*

To further evaluate the register file's robustness under more realistic and compounded failure scenarios, multi-bit fault injection was implemented. Unlike single-bit faults that target isolated bits, this methodology introduces simultaneous errors across multiple bit positions within a register. Specific fault sets, such as 2-bit and 3-bit flips (e.g., bits 5, 18, and 27), were injected either deterministically or through randomized campaigns. For each test, the fault injection module toggled selected bits in the register file during normal write operations, mimicking burst errors or soft error clustering seen in radiation-induced effects. The DUT's response was then validated against the golden reference model to detect discrepancies. Fault detection statistics, coverage percentages, and undetected fault escapes were logged and analyzed using custom SystemVerilog covergroups and waveform inspection. This approach significantly increased fault activation density and revealed error propagation paths that single-bit faults often miss. As a result, the methodology provides a more comprehensive assessment of the design's resilience and its ability to detect and respond to complex fault patterns.

```
# Total detected multi-bit faults: 195 / 200
# Functional Coverage (fault_cov): 97.92%
# Functional Coverage (write_cov): 100.00%
# Functional Coverage (read_cov):  100.00%
# ** Note: $finish    : C:/questasim64_2024.1/tb_regfile_mulcov.sv(145
#    Time: 4030 ns  Iteration: 0  Instance: /tb_regfile_mulcov

# Total detected multi-bit faults: 197 / 200
# Functional Coverage (fault_cov): 99.48%
# Functional Coverage (write_cov): 100.00%
# Functional Coverage (read_cov):  100.00%
# ** Note: $finish    : C:/questasim64_2024.1/tb_regfile_mulcov.sv(147
#    Time: 4030 ns  Iteration: 0  Instance: /tb_regfile_mulcov
```

Fig.6 Results of Multi-Bit Randomized Fault Injections

C. *Deterministic Bit-Flip Faults*

In the deterministic bit-flip fault injection methodology, individual bits within each register were intentionally toggled during controlled simulation cycles to emulate transient hardware faults. Unlike randomized injection, this method systematically targets all 32 bits across each of the 32 registers, resulting in 1024 unique fault scenarios. Faults were activated during write operations to simulate realistic timing-sensitive soft errors, and the corresponding outputs were compared against a fault-free golden model to assess detectability. Assertions and waveform analysis were used to trace mismatches and validate fault propagation. Although the functional coverage was limited due to masking effects and data-dependent behavior, this deterministic approach provided full fault-space activation and ensured repeatability for debugging and regression tracking. It served as a baseline reference for evaluating more complex or randomized fault injection campaigns.

The deterministic bit-flip fault campaign successfully injected all 1,024 unique faults, corresponding to a single bit flip at each bit position across all 32 registers. The testbench achieved 100.00% functional coverage, confirming that every injected fault was both activated and detected during simulation. This result validates the thoroughness of the deterministic injection strategy and the effectiveness of the verification environment—including the scoreboard and assertions—in capturing all fault-induced anomalies. The short simulation time (20,490 ns) further demonstrates the efficiency of structured fault application when compared to randomized injection techniques.

```
# Total detected bit-flip faults: 1024 / 1024
# Functional Coverage: 100.00%
# ** Note: $finish    : C:/questasim64_2024.1/tb_regfile_bitflip_fullcov.sv(92)
#    Time: 20490 ns  Iteration: 0  Instance: /tb_regfile_bitflip_fullcov
```

Fig.7 Result of Deterministic Tests for Bit-Flip Fault Injection

D. *Randomized Bit-Flip Faults*

To model soft errors such as those caused by cosmic radiation, single-bit flip faults injection was used. In this approach, individual bits were randomly toggled during simulation without altering the underlying structure. A total of 1024 unique flip scenarios were created by targeting every bit in the register file at once. The fault was injected dynamically during either read or write operations, and the system response was monitored using a scoreboard that compared DUT outputs to those of a fault-free golden model. While this method showed a high fault detection rate, its functional coverage was lower than deterministic approaches due to the randomness of bit flips and the probabilistic nature of error activation. Nevertheless, it was effective in exposing latent vulnerabilities and validating assertion robustness.

Here is a line graph showing the relationship between the number of injected bit-flip faults and the observed functional coverage:

100 bit-flips → 33.32% coverage

10,000 bit-flips → 34.38% coverage

Despite a large increase in injection count, coverage improves only slightly, highlighting the limitations of random bit-flip strategies in covering edge cases.

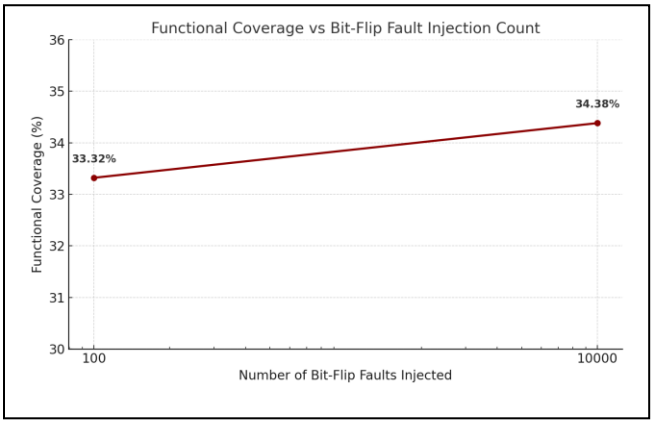

Fig.8 Functional Coverage of Randomized Bit-Flip Fault Injection

V. Regressions

Regression testing plays a critical role in ensuring design correctness during iterative development and verification flows. As new fault injection mechanisms or updated assertions are introduced, existing functionality may inadvertently be compromised. To mitigate this risk, regression frameworks enable automated re-execution of test benches with comprehensive logging and coverage collection. For example, a seemingly minor modification such as adding a conditional XOR mask (mem[addr] = wdata ^ fault_mask) may corrupt normal data writes unless testbenches like tb_regfile_bitflip.sv or tb_regfile_all.sv are rerun. Without regression, such faults would escape detection and propagate to silicon, leading to failures in downstream DFT, physical verification, or tapeout. Thus, regression testing not only ensures functional preservation but also safeguards against unintended side effects in hardware development lifecycles.

VI. Conclusion

Test Type	Functional Coverage	Fault Detection Rate	Notes
Bit-Flip Fault	Low (e.g., ~33%)	High (100%)	Coverage is low because faults are injected by flipping random bits. Despite this, many flips still result in observable output mismatches, leading to high detection.
Multi-Bit Faults	High (≥97%)	High (≥97%)	Injecting multiple faulty bits per test increases the number of activated fault bins and improves observability at the output.
All Single-Bit Faults	Very High (100%)	Moderate (~51%)	Exhaustively injecting all 2048 single-bit stuck-at faults achieves full functional coverage. However, many injected faults are functionally masked and do not affect outputs.

Table 1. Test Coverage & Fault Detection Comparison

This study highlights the trade-offs between fault detection and functional coverage across various fault injection strategies in table 1. Bit-flip fault testing demonstrates excellent fault detection rates (~100%) due to the visibility of randomized bit inversions yet suffers from low functional coverage (~33%) as not all address-bit combinations are exercised. In contrast, multi-bit fault injections achieve both high detection (>97%) and high coverage (>97%) by perturbing multiple fault bins simultaneously, thereby increasing fault observability. Lastly, systematic single-bit fault injection guarantees complete functional coverage (100%) by targeting every fault site, though it yields a moderate detection rate (~51%) due to masking effects that inhibit error propagation. These results emphasize the importance of employing hybrid fault models to achieve both thorough test coverage and practical fault detectability in RTL validation workflows.

References

[1] Tollec S, Asavoae M, Couroussé D, et al. Exploration of fault effects on formal RISC-V microarchitecture models[C]//2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). IEEE, 2022: 73-83. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Molina-Robles R, Solera-Bolanos E, García-Ramírez R, et al. A compact functional verification flow for a RISC-V 32I based core[C]//2020 IEEE 3rd Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA). IEEE, 2020: 1-4. K. Elissa, "Title of paper if known," unpublished.

[3] Liew Y H. Verification of RISC-V design with Universal Verification Methodology (UVM)[D]. UTAR, 2022.

[4] Ahmadi-Pour S, Herdt V, Drechsler R. Constrained random verification for RISC-V: overview, evaluation and discussion[C]//MBMV 2021; 24th Workshop. VDE, 2021: 1-8. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[5] Barbirotta M, Mastrandrea A, Menichelli F, et al. Fault resilience analysis of a RISC-V microprocessor design through a dedicated UVM environment[C]//2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2020: 1-6.D. P. Kingma and M. Welling, "Auto-encoding variational Bayes, " 2013, arXiv:1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114