Case Study: Fault-Injection Vulnerability Assessment at RTL Level

Azim Uddin, Sujan Kumar Saha, Farimah Farahmandi and Mark Tehranipoor
Department of Electrical and Computer Engineering, University of Florida
Emails: {azim.uddin, sujansaha}@ufl.edu, {farimah, tehranipoor}@ece.ufl.edu

Motivation

• As hardware design grows in complexity and global distribution, embedded devices face significant security challenges, especially against fault injection attacks that can alter control flows, compromise device integrity, and reveal sensitive data.
• Most fault-injection studies focus on gate- or layout-level vulnerabilities, but this approach limits early detection and mitigation opportunities.
• RTL-level assessments offer a proactive approach, enabling early-stage vulnerability identification and secure design practices.

Contribution

• Developed a security property-oriented fault-injection assessment framework at the RTL level.
• Modeled the impact of different types of fault injection techniques at the RTL level.
• Translated design-specific security properties to observable strobe points.
• Performed a formal feasibility analysis of injected faults and determined the design's vulnerability for specific security properties.
• Developed local countermeasure recommendations, allowing designers to implement specific protections at critical points identified through the analysis.

Fault Injection Background

Fault injection attacks are active physical attacks where an attacker intentionally injects a fault into a device to compromise its security. It can be used to bypass security mechanisms, access secret data, or disrupt normal operation.

Types of Fault Injection:
Voltage Glitching: Alters circuit behavior by briefly changing the supply voltage.
Clock Glitching: Disrupts timing through irregular clock signals.
Laser Fault Injection: Uses a laser to manipulate specific transistor states.
Electromagnetic Interference (EMI): Applies electromagnetic fields to interfere with operations.
Focused Ion Beam (FIB): Modifies internal structures using a precise ion beam.

Related Works

• Most existing studies focus on fault injection at the gate-level or layout-level, which limits early vulnerability identification and mitigation options.
• Some frameworks, like SoFI, focus on security property checks to evaluate vulnerabilities at the gate level, emphasizing specific security properties to drive assessments but lacking early RTL-level evaluation.

Proposed Research

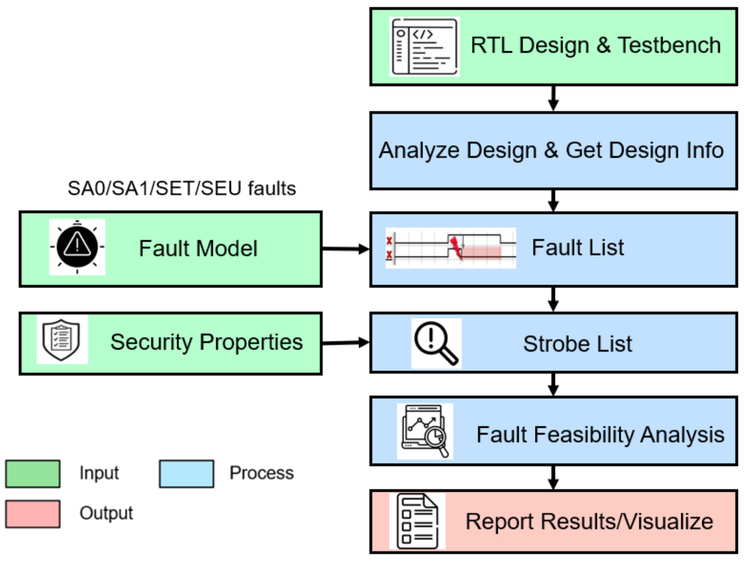

Figure: Proposed Framework for Fault Injection Vulnerability Assessment

Testbench Development: The testbench is crafted to cover all operational scenarios and edge cases for robust RTL assessment. It should include security-sensitive scenarios, such as encryption and decryption, to evaluate fault injection impacts.
Fault Modelling: Impacts of different fault injection techniques are translated into equivalent fault models at RTL level such as stuck-at faults (SA0, SA1), single event upsets (SEUs), and single event transients (SETs).

Technique	Fault Category	Fault Type	Fault Duration
Clock Glitching	Global	Bit-flip	Transient
Voltage Glitching	Global	Bit-flip	Transient
EM	Global	Bit-flip	Transient
Laser	Local	Bit-flip	Transient
FIB	Local	Bit-flip Stuck-at	Transient Permanent

Security Property based Strobe List Generation: After developing security properties for the RTL design, Specific signals or registers—known as strobe points—are selected to monitor these security properties. Strobe points are refined with conditions, specifying when and under what values they should be checked, ensuring accurate monitoring aligned with security requirements.
Fault Feasibility Analysis: We utilize formal methods to assess fault injection vulnerability. The analysis involves two instances of the design—a "good machine" (no faults) and a "bad machine" (faults injected)—to observe differences in behavior and identify potential security property violations.
Fault Classification: Faults are classified into one of three categories based on the impact at strobe points:

- **Dangerous:** Faults causing value mismatches at strobe points between the good and bad machines, violating security properties.
- **Safe:** Faults that do not cause any mismatch at strobe points.
- **Unknown**: The impact of the fault through formal Analysis could not be determined through formal analysis, treated as Dangerous for completeness.

For doing this classification, three types of structural analysis are performed.

- **Out-of-Cone-of-Influence (COI):** Faults outside the COI of strobe points are considered safe as they cannot influence critical signals.
- **Activatability Analysis:** Faults on constant signals that cannot be activated are marked safe.
- **Propagatability Analysis:** Faults that activate but do not reach strobe points are also marked safe, reducing false positives.

We used the Cadence JasperGold Functional Safety Verification (FSV) App for formal fault analysis, which generates a report that classifies each fault and offers interactive debugging and visualization for detailed fault information, such as propagation paths. Our framework further analyzes these reports to identify critical fault scenarios, impacts, and mitigation recommendations.

Experimental Setup

For this case study, our proposed fault injection framework was applied to three implementations of AES IP (from OpenTitan, FreeCores and SecWorks) and three implementations of SHA IP (from SecWorks, OpenTitan and OpenCores), each developed with varying degrees of security considerations.
We assume the attacker can inject faults at any location at any time. Although this assumption isn't practical for large designs, it enables thorough fault vulnerability assessment. After that, we define two general security properties for those designs: SP1: The *done/valid* signal should not be high before crypto operation is complete and SP2: Any intermediate states should not be skipped during crypto/hashing operation.

Results

Design Type	Fault Type	Total Faults	Dangerous Faults (%)		
			SP1	SP2	Any
SHA-SW	SA0	3999	6 (0.15%)	5 (0.13%)	6 (0.15%)
	SA1	3999	23 (0.58%)	19 (0.48%)	24 (0.60%)
	SEU	1033	8 (0.77%)	7 (0.68%)	8 (0.77%)
	SET	3999	27 (0.68%)	24 (0.60%)	28 (0.70%)
SHA-OT	SA0	3852	250 (6.49%)	279 (7.24%)	279 (7.24%)
	SA1	3852	253 (6.57%)	278 (7.22%)	278 (7.22%)
	SEU	1181	88 (7.45%)	88 (7.45%)	88 (7.45%)
	SET	3852	281 (7.29%)	281 (7.29%)	281 (7.29%)
SHA-OC	SA0	1760	16 (0.91%)	19 (1.08%)	19 (1.08%)
	SA1	1760	14 (0.80%)	19 (1.08%)	22 (1.25%)
	SEU	1103	10 (0.91%)	9 (0.82%)	10 (0.91%)
	SET	1760	21 (1.19%)	19 (1.08%)	21 (1.19%)

Design Type	Fault Type	Total Faults	Dangerous Faults (%)		
			SP1	SP2	Any
AES-SW	SA0	11711	76 (0.65%)	76 (0.65%)	93 (0.79%)
	SA1	11711	90 (0.77%)	70 (0.60%)	105 (0.90%)
	SEU	2470	29 (1.18%)	16 (0.65%)	29 (1.18%)
	SET	11711	117 (1.00%)	79 (0.68%)	131 (1.12%)
AES-FC	SA0	2133	5 (0.23%)	5 (0.23%)	6 (0.28%)
	SA1	2133	4 (0.19%)	6 (0.28%)	7 (0.33%)
	SEU	554	5 (0.90%)	4 (0.72%)	5 (0.90%)
	SET	2133	8 (0.38%)	6 (0.28%)	8 (0.38%)
AES-OT	SA0	17136	8 (0.05%)	28 (0.16%)	28 (0.16%)
	SA1	17136	12 (0.07%)	18 (0.11%)	22 (0.13%)
	SEU	703	0 (0.00%)	0 (0.00%)	0 (0.00%)
	SET	17136	149 (0.87%)	206 (1.20%)	206 (1.20%)

Mitigation Strategy

From the results, we can see only about 1-2% of fault locations are dangerous. If global countermeasure like Triple Module Redundancy (TMR) is used for whole design, it will incur ~200% overhead. Using error detection and correction codes (EDAC) will add less overhead but will add latency. However, we can use our framework to identify the locations where low-overhead local countermeasures can be applied. The following figure illustrates that 6 of the dangerous SEU faults for SHA-SW are associated with **t_ctr_reg**, which stores the round counter value.

Fault Table

Filter on Node

ID	Node	Type	Injection	Constant	FCOI	CCOI
8384	dut.digest_valid_reg	SEU	0:$	Activated	In	Unpr...
8514	dut.sha256_ctrl_reg[1]	SEU	0:$	Activated	In	Unpr...
8515	dut.t_ctr_reg[0]	SEU	0:$	Activated	In	Unpr...
8516	dut.t_ctr_reg[1]	SEU	0:$	Activated	In	Unpr...
8517	dut.t_ctr_reg[2]	SEU	0:$	Activated	In	Unpr...
8518	dut.t_ctr_reg[3]	SEU	0:$	Activated	In	Unpr...
8519	dut.t_ctr_reg[4]	SEU	0:$	Activated	In	Unpr...
8520	dut.t_ctr_reg[5]	SEU	0:$	Activated	In	Unpr...

We can apply local countermeasure where only the portion responsible for updating the register is duplicated three times, which drastically reduces dangerous faults with minimal overhead.

Design	Flops	Gates	SA0	SA1	SET	SEU
Base Design	35	332	6	23	27	8
Global countermeasure	105 (+200%)	1019 (+207%)	3(-50%)	4(-83%)	5(-81%)	0(-100%)
Local countermeasure	37 (+5.7%)	340 (+2.4%)	5 (-16.7%)	17 (-26.1%)	20 (-25.9%)	2 (-75%)

Conclusion & Future Plan

Our RTL-level fault injection framework effectively identifies critical vulnerabilities and enables early-stage security enhancements with minimal overhead. Future work will focus on automating strobe list generation and expanding the framework to support multi-fault scenarios. Applying the approach to diverse designs will further validate its robustness and adaptability.