Automating Fault Injection through CABA Simulation for Vulnerability Assessment

William Pensec, Vianney Lapôtre, Guy Gogniat

Automating Fault Injection through CABA Simulation for Vulnerability Assessment

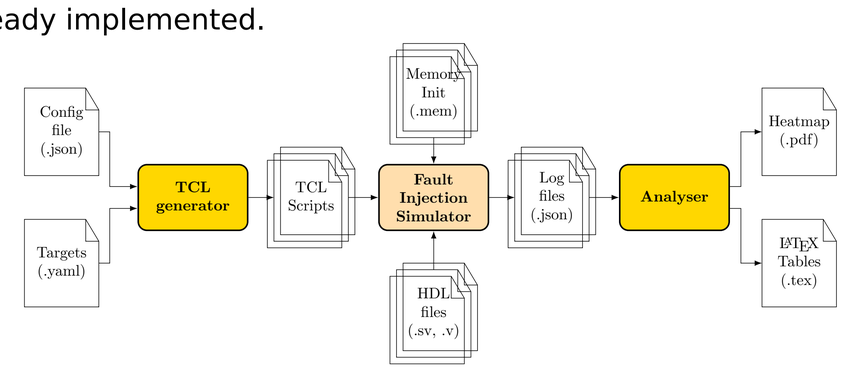
William PENSEC, Vianney LAPÔTRE, Guy GOGNIAT
Université Bretagne Sud, UMR 6285, Lab-STICC, Lorient, France
firstname.lastname@univ-ubs.fr

Context

Internet of Things (IoT) devices have revolutionised data collection and analysis, yet their proximity raises concerns about physical attacks like fault injection attacks (FIA). Numerous studies have highlighted critical system vulnerabilities to FIAs [1, 2]. This poster presents FISSA (Fault Injection Simulation for Security Assessment), automating circuit design robustness evaluation against FIA through early-stage simulations using existing HDL simulators.

FISSA - Fault Injection Simulation for Security Assessment

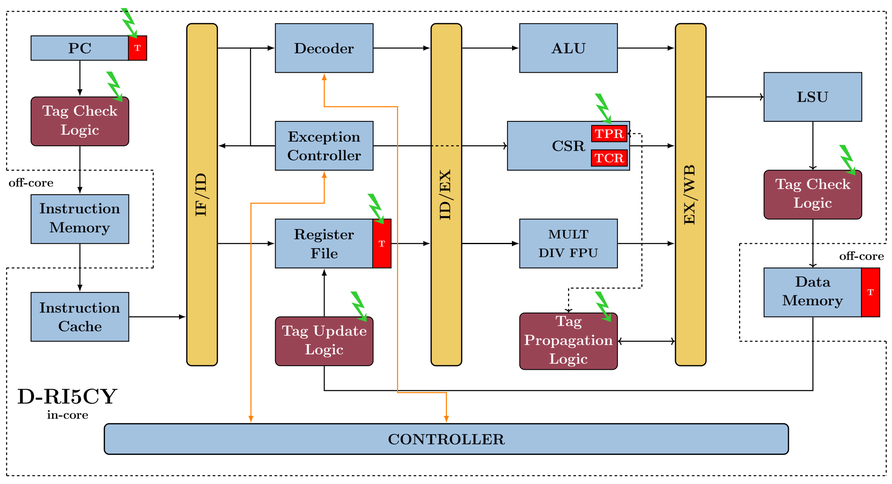
FISSA is an open-source [3] tool based on an HDL simulator such as Questasim to perform fault injection attack campaigns in simulation in order to assess the security of designs during the development phase. Its main characteristics are: highly configurable, easy to integrate, high control over fault scenarios, and multiples fault models already implemented.



Software architecture of FISSA

D-RI5CY

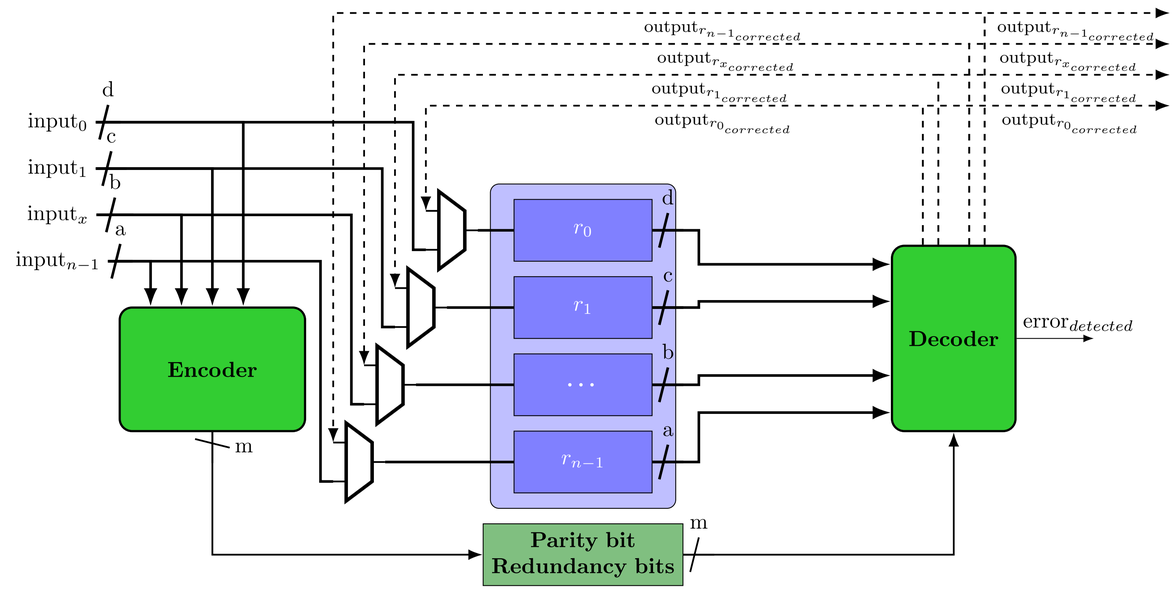
We study the D-RI5CY [4] which introduces a Dynamic Information Flow Tracking (DIFT) mechanism to protect the processor against software attacks such as buffer overflows, SQL injections, etc. DIFT-related elements are represented in red.



D-RI5CY processor architecture overview

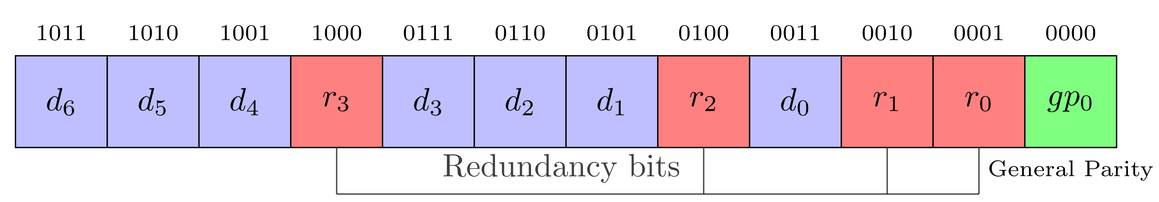
Evaluated protection mechanism

Hamming Codes are a class of linear error-correcting codes invented by Richard W. Hamming [5] in 1950. The main use of these codes is to detect and correct errors. They are mostly used in digital communication and data storage systems as error control codes. The main disadvantage is they work only for one error.



Proposed scheme for code-based protections of a set of independent registers

In addition of Hamming Code to correct one error and detect two errors, we decided to implement another implementation of Hamming Code. This implementation is named as SECDED for Single Error Correction Double Error Detection. This code use Hamming Code and a parity bit to detect double errors.
The general parity bit "gp_0" is calculated on all other bits with a XOR ($\oplus$) operator.



SECDED redundancy and general parity bits positions.

Experimental results

Table 1: Results for single bit-flip in two registers at a given clock cycles

		Crash	Silent	Delay	Detection	Single Error Correction	Double Errors Detection	Success	Total
Buffer overflow	No protection	0	45,097	1,503	–	–	–	1,406 (2.93%)	48,006
	Simple Parity	0	10,551	134	40,952	–	–	239 (0.46%)	51,876
	Hamming Code	0	0	575	–	67,829	–	452 (0.66%)	68,856
	SECDED	0	2,436	0	–	59,424	11,616	0 (0.00%)	73,476
Format string	No protection	0	55,589	5,035	–	–	–	3,384 (5.29%)	64,008
	Simple Parity	0	13,361	450	54,590	–	–	767 (1.11%)	69,168
	Hamming Code	0	0	1,709	–	89,010	–	1,089 (1.19%)	91,808
	SECDED	0	0	0	–	82,480	15,488	0 (0.00%)	97,968

Table 1 shows protection schemes against buffer overflow and format string vulnerabilities. Without protection, success rates are 2.93% and 5.29%. Simple Parity and Hamming Code reduce these significantly. SECDED is the most effective, reducing success rates to 0%.
The results in Table 2 show that the three implementations have no impact on performance and the area overhead is negligible (less than 8%) to allow the correction of single errors and the detection of double errors.

Table 2: Implementation results of the three different protections

	LUT	FF	Max Frequency
No protection	6,911	2,335	47.619 Mhz
Simple Parity	7,011 (1.45%)	2,337 (0.09%)	47,619 Mhz (0%)
Hamming Code	7,283 (5.38%)	2,361 (1.11%)	47.447 Mhz (-0.36%)
SECDED	7,428 (7.48%)	2,366 (1.33%)	47.170 Mhz (-0.95%)

Conclusion and Perspectives

In this work, we present a configurable open-source tool to automate fault injection simulations campaigns. We illustrate FISSA thanks to a simple use case. FISSA can be used with well-known HDL simulators such as Questasim. It generates TCL scripts for simulation and produces JSON log files for security analysis.
In future work, we plan to support new HDL simulators, extend the fault models supported and improve integration into the design workflow.

References

[1] S. Nashimoto *et al.*, "Bypassing Isolated Execution on RISC-V using Side-Channel-Assisted Fault-Injection and Its Countermeasure," 2021. DOI: 10.46586/tches.v2022.i1.28-68.

[2] J. Laurent *et al.*, "Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures," in *Design, Automation & Test in Europe Conference (DATE)*, 2019. DOI: 10.23919/DATE.2019.8715158.

[3] W. Pensec, *Fault Injection Simulation for Security Assessment*. [Online]. Available: https://github.com/WilliamPsc/FISSA.

[4] C. Palmiero *et al.*, "Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications," in *High Performance Extreme Computing*, 2018. DOI: 10.1109/HPEC.2018.8547578.

[5] R. W. Hamming, "Error detecting and error correcting codes," 1950. DOI: 10.1002/j.1538-7305.1950.tb00463.x.