Implementation and evaluation of countermeasures implementation in a DIFT against Fault Injection Attacks

William PENSEC^1, Vianney LAPÔTRE^1 and Guy GOGNIAT^1
^{1}Université Bretagne Sud, UMR 6285, Lab-STICC, Lorient, France
`firstname.lastname@univ-ubs.fr`

1 Introduction

Software attacks such as buffer overflows, and malware can be detected using Dynamic Information Flow Tracking (DIFT) techniques. These techniques tag and track information containers during runtime to detect malicious activities. Various DIFT implementations, including hardware, software, and hybrid approaches, have been studied [1]. Information containers can range from files to registers.

This study focuses on the D-RI5CY processor as proposed in [2]. Our objective is to assess the impact of Fault Injection Attacks (FIAs) on the D-RI5CY DIFT mechanism and to propose effective countermeasures. We use fault injection simulations to evaluate the sensitivity of DIFT and identify vulnerable registers by using a new open-source tool designed to perform automated fault injection campaigns.

FIAs can be induced via power supply glitches, clock perturbations, or laser shots. Laser injections offer precise control, while power and clock perturbations affect the entire circuit. Critical systems are vulnerable to such attacks, which can bypass protection mechanisms and hijack systems, as shown in [3].

Figure 1 presents an overview of the D-RI5CY processor with DIFT components highlighted. These components manage tags during execution. The security policy is configured via Control and Status Registers (CSRs), TPR, and TCR.

We present three different countermeasures to enhance the DIFT mechanism against FIAs. We briefly explain the implementation of these countermeasures and provide a brief presentation of the results. Our analysis aims to develop a more robust DIFT mechanism that can counter both software and physical attacks. .

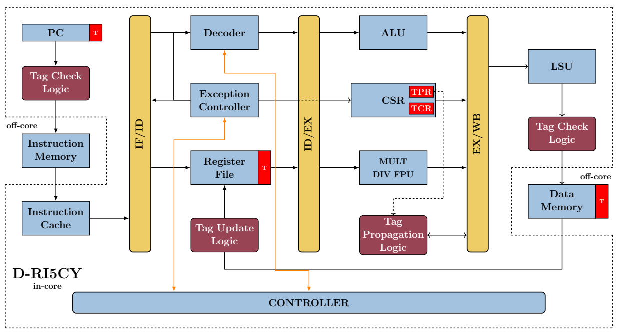

Figure 1: *Overview of the D-RI5CY processor*

2 Countermeasures implementations

We have shown that DIFT is vulnerable against FIA and the main threat that it faces is bit-flips. For an attacker able to attack specific bits, he would be able to bypass the DIFT computation.

To protect our DIFT, we propose different countermeasures from being able to detect only single-bit fault to be able to detect two-bits faults and correct single-bit errors. At first, we choose to implement simple parity which will be able to detect single-bit fault. With this countermeasure, we have shown that we can detect every single-fault and we can stop the faulted application at cycle accurate. But with it, we only detect errors. For that, we choose to implement Hamming Code into our registers. For that, we tried different implementation from one that reduce the number of redundandy bits by grouping small registers together, to protecting our registers by pipeline stage or even protecting every register with Hamming code. This induce a bigger overhead but with better results than the first implementation against two-bits errors. The last implementation is the more complex and split every registers. This lead to an overhead of 3.98% in terms of LUTs. To face two-bits errors, we choose to use another version of Hamming code with an extra bit for parity (SECDED for single-error correction, double-errors detection), under the same implementation than Hamming code.

Finally, we compare the results associated to each implementation in terms of area, performance and efficiency against different fault models from simple one (single-bit fault into one register at a given cycle) to more complex ones (multi-bits faults into two registers at a given cycle).

References

[1] Kejun Chen et al. “Dynamic Information Flow Tracking: Taxonomy, Challenges, and Opportunities”. In: *Micromachines* (2021). DOI: `10.3390/mi12080898`.

[2] Christian 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`.

[3] Niek Timmers et al. “Controlling PC on ARM Using Fault Injection”. In: *Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2016. DOI: `10.1109/FDTC.2016.18`.