Internal State Monitoring in RISC-V Microarchitectures for Security Purpose

Roua Boulifa, Giorgio Di Natale, Paolo Maistri

Internal State Monitoring in RISC-V Microarchitectures for Security Purpose

Roua Boulifa*, Giorgio Di Natale*, Paolo Maistri*
*Univ. Grenoble Alpes, CNRS, Grenoble INP[1], TIMA, 38000 Grenoble, France

***Abstract*—Embedded systems play a significant role in our everyday lives, making them prime targets for malicious actors. Consequently, ensuring the security of such systems becomes a crucial concern. Among various threats, fault injection attacks on microprocessors are particularly notable. Understanding the effects of these attacks within the microarchitecture is thus essential to assess their impact on overall security.**

In this paper, we present the Internal State Monitor (ISM), which provides observability at the software level of selected micro-architectural signals in RISC-V processors. ISM will be used at first to better understand the fault effects and their propagation patterns. This understanding is pivotal in designing effective countermeasures to protect processors against these attacks.

***Index Terms*—Hardware security, Fault injection, Risc-V security, Internal monitoring**

I. Introduction

In recent years, security has emerged as a critical concern, with a notable increase in attacks targeting design vulnerabilities in hardware, software, and protocols. Computer-based digital systems are often composed of software applications executed on a hardware architecture including a processor, which itself can be a direct target of physical attacks, known as hardware attacks. Hardware security aims to protect against these attacks, which may disrupt the normal system behavior and thus enable the leakage of sensitive information. In such cases, gaining a thorough comprehension of the events occurring within the processor both during and immediately after an attack becomes imperative. This is essential for accurately characterizing the repercussions to the internal components, and ultimately designing effective countermeasures.

A running system becomes comprehensible when it gives insight into its internal state, which helps to understand the propagation of events in the microarchitecture. For debugging purposes, therefore, providing visibility of program execution is essential. For instance, the trace encoder proposed in [12] allows tracking the execution of instructions from a known start address. This system is composed of a core with a trace interface that outputs all relevant information (instruction address, instruction type). This interface is connected to a hardware encoder that compresses the information into lower-bandwidth trace packets, which are transmitted via a transmission channel for storage. The decoder then reconstructs the program flow based on the trace packets and knowledge of the program binary. Although these data could be useful to debug and analyse the behavior of the program, Anthony et al. [1] used the trace encoder to provide information about the execution path of the user program, its focus (and the monitored elements) remains rather limited to Control Flow exceptions, without any insight to internal and hidden components of the microarchitecture.

In this paper, we introduce the Internal State Monitor (ISM), an additional module to the RISC-V core capturing flip-flop output signals for subsequent analysis. Compared to existing debugging units (which allow observing software details, the proposed ISM provides the observability at software-level of selected micro-architectural signals in RISC-V processors. Our primary core target, Pulpino, based on the RISC-V open-source ISA, will be used as test case for our approach. Moreover, this processor together with the proposed ISM will be implemented in an FPGA-based platform which enables physical attacks.

The goal of the proposed ISM is to better understand the fault effects and their propagation patterns. This improved understanding will help in designing effective countermeasures to protect processors against attacks.

This paper is organized as follows: section II describes a concise overview of fault modeling at the microarchitectural level and how faults are propagated to higher levels. Section III presents the used RISC-V core. Finally, Section IV details the proposal of the ISM and its hardware/software integration within the targeted processor.

II. Fault attacks and models

A. Fault Attacks

Fault Injection Attacks, or fault attacks, are a type of active attacks where the goal of the attacker is to disrupt the circuit functionality during its operation. The issue of faults was first known in the aerospace industry, where systems, less protected by our atmosphere, can be disturbed by particle impact. Unlike natural (uncontrolled) faults, the implementation of a fault attack is characterized by many parameters to be possibly controlled, such as: spatial granularity, i.e., the ability to precisely select the target position of the attack, as well as its extent (e.g., one or more bits); the effect of the fault on these bits (inversion, setting to 1, setting to 0, random transformation); their temporal precision, i.e., the ability to control the moment of injection (e.g., during the execution of an instruction); and their permanence, i.e., the duration of the perturbation or its effect over time (one cycle, several cycles, or permanent). When constructing an attack, the fault

[1]Institute of Engineering Univ. Grenoble Alpes

TABLE I
PHYSICAL ATTACK PRECISION COMPARISON

Physical Attack	Temporal Precision	Spatial Precision
Clock glitch	High	N/A
EM injection	High	Medium
Voltage glitch	High	N/A
Laser	Very High	High
X-ray	N/A	Very High

injection mechanism is chosen based on these criteria and the intended exploitation of the attack. Some attacks can be carried out with a low resolution in terms of precision, while others require maximum precision to avoid, for instance, triggering a countermeasure.

In fault injection attacks, various mechanisms are exploitable, such as clock and voltage glitching, electromagnetic (EM) interference, laser-induced faults, and X-ray.

The goal of a clock glitch is to shorten the period of one (or multiple) clock cycles, so that erroneous inputs are sampled by the flip-flops. This attack is possible when the clock can be easily manipulated by the attacker. Claudepierre et al. [2] propose a low-cost mechanism for injecting multiple faults using clock signal perturbations. Moreover, there are also low-cost fault injection systems marketed for the general public, such as the ChipWhisperer board [11].

With voltage glitching, the attacker manipulates the power supply causing voltage fluctuations that can lead to faulty behavior on the target device. It can be achieved by precisely creating variations in a power supply (either over- or under-powering). Korak et al [3] combine glitches on the clock signal with glitches on the power supply bus to improve the success rate of fault injection.

Electromagnetic fault injection is another technique to perturb the circuits: during an electromagnetic pulse the wire nets within the circuit are disturbed for the duration of the pulse. This disturbance particularly affects the synchronous elements of the circuits. Recently Elmohr et al [4] showed that EM fault injection can be injected in 320 MHz RISC-V processor leading to multiple instruction skip. As a result if the EM pulse voltage increase, the number of the consecutive instructions being skipped also increases.

More expensive and complex techniques exist. Laser fault injection injects errors due to the photoelectric effect resulting from its interaction with silicon. X-ray perturbation can be used to induce faults in the processor by modifying the electrical behaviour of a transistor using focused X-ray beams, by inducing permanent faults. The temporal and spatial characteristics of these techniques are summarized in Table I.

B. Fault Models

Fault models are proposed to describe the effect of the physical perturbations. The impact of a fault injection depends on the specific attack scenario and the targeted system or component. Different types of attacks may exhibit distinct fault effects based on their objectives and the vulnerabilities they exploit. Therefore, observed fault models depend on the type of physical injection. Moreover, fault model can be described at different abstraction levels, presenting trade-off between accuracy and simulation time [9]. The higher the level of abstraction, the less realistic and accurate the fault model becomes, but also the simpler to be simulated.

At the electrical level, Single Event Transients (SETs) model the effect of an ionizing particle striking the blocked junction of a MOSFET transistor. The resulting transient current pulse (SET) may propagate through logic gates, potentially causing data corruption or system failure. A Single Event Upsets (SEUs) models logical state modification in a memory point, which is a reversible effect. It can model the effect of a ionizing particles depositing charges on memory points or by the transformation of a transient upset into a logical error. Both SET and SEU can lead to Multiple Bit Upset (MBU).

At the logic/gate level, the modeling of fault injection involves directly the individual logic gates, seen as an atomic element. It comprises stuck-at faults (i.e., the inputs or the outputs of a gate are considered fixed at '0' or '1'), delay faults (i.e., the propagation time from input to output of a logic gate is altered), or transition faults (i.e., the output of a gate is not able to make a transition).

At the microarchitectural level, Trouchkine et al. [5] demonstrated that faults can manifest in different ways, affecting either the pipeline elements, registers, or memory. They can be classified into two main categories: those that influence data and those that impact instructions. Among the former, fault models describe either register or memory corruption, as well as wrongly fetched instructions. On the other hand, the latter faults can result in corruption within the pipeline micro architectural blocks (MABs), cache, or instruction bus.

At the ISA (i.e., assembly) level, the effects of fault injection are described as a modification of the instructions executed by the processor. Incorrect instruction execution may result in faulty computations, data corruption, or unexpected behavior, disrupting control flow by altering the program counter (PC) values, and skipping instructions, which can affect one [6] or multiple instructions [7]. Bosio and al [9] introduced three different models: (1) affecting the portion of the instruction responsible for the encoding of the destination register which model the data fault model; (2) affecting the instruction opcode and model the code fault model; and (3) affecting the condition flags of the instruction, which can result on having an erroneous flag impacting the control flow of the program.

At software level, hardware faults are translated into the software domain. They include data fault models, code fault models, and system fault models. Data fault models simulate faults corrupting data processed by software; code fault models affect the sequence of executed blocks in a program; and system fault models address timing, communication, or synchronization faults during software execution.

Lower level fault models are more accurate and they better represent the effects of perturbations and injections. Nevertheless, they are more complex and they require longer simulations. For processor-based systems, electrical and logical fault simulations are not affordable in reasonable times.

TABLE II
MODELING FAULT IN DIFFERENT LEVEL

Fault level	Fault model
Electrical level	SET, SEU, MBU, MET Double exponential Current injection
Logic gate level	Bit flip, Bit Set, Bit Reset Stuck-At Fault Delay Fault Transition Fault
Micro architectural level	Register Corruption Memory Corruption Instruction Bus Corruption Pipeline and MAB Corruption
ISA level	Instruction Skip, Skip and Repeat Register Corruption Incorrect Instruction Execution
Software level	Control Flow Error Variable Corruption

Hence, we need to rely on accurate fault models at higher levels (microarchitectural/ISA). Nowadays, however, the fault injection effects at these levels are not fully understood. This observation motivates our work which proposes to comprehend the fault propagation of actual physical fault injections by resorting to the observation of the internal state of the processor's microarchitecture.

III. RISC-V OVERVIEW AND IMPLEMENTATION

A. PULPino

In this paper, we use a RISC-V system as our target. RISC-V is an open Instruction Set Architecture developed by the University of California, Berkeley since 2010. Like its name implies, it is a Reduced Instruction Set Computer. It uses a load/store architecture, which means that memory and register operations are decoupled. It can operate on various data widths: 32, 64 or even 128 bits. The RISC-V architecture is modular: it is composed primarily of a base integer instruction set (I), to which several extensions can be attached.

We decide to use PULPino SoC coming from ETH Swiss because of its lightness, flexibility and relative easiness of understanding. PULPino is a SoC design based on the processor CV32E40P (RI5CY) and includes the complete peripheral interfaces (e.g., I2C, SPI) and AMBA Bus. Concerning the memory structure, it separate data and instruction memory and supports the external memory access via SPI. It supports two debug methods: JTAG Debug via OpenOCD and SPI Slave debug. CV32E40P is a small and efficient, 32 bit, in-order RISC-V core with a 4-stage pipeline written in System-Verilog. It implements the RV32IMC (i.e., Integer, Multiplication/division, and Compressed instructions), and optionally F (single-precision Floating point), and other custom extensions for achieving higher code density, performance, and energy efficiency. In particular, it supports the compressed instruction format (16-bits) which is a usual feature but also with a potential risk of introducing new vulnerabilities [8].

B. Hardware platform

The overall goal of this work is to understand the effect of physical fault injections at the microarchitectural level of a processor. We require therefore an evaluation board that facilitates both the physical fault injection process and the possibility to implement our target RISC-V core. We have identified the ChipWhisperer platform, which is a suitable choice meeting these requirements. Chipwhisperer is an embedded security analysis platform designed by NewAE, completely self-contained and requiring no additional hardware besides a computer [11]. It embeds a programmable FPGA, as well as a dedicated board equipped with intrinsic fault injection capabilities, based on manipulations of the clock signal. It provides a Python environment running on the host PC to configure, observe, and control the resource of the whole board. During our experiments, we use the CW305 board which equips a Xilinx FPGA which can carry the target SoC design, and an Atmel MCU acting as the communication bridge between PC and the target. Our experimental setup consists of two parts: the board CW305 equipped with the target design (PULPino and some necessary components, such as the PLL), and the CW1173 board used for communication and fault injection (which leverages a trigger signal coming from the target in order to generate a synchronized clock glitch from the internal clock of the target). Finally, both boards are interfaced and controlled through the PC. In this paper we focus on the description of the structure of the ISM, without details about the actual effects of the injections.

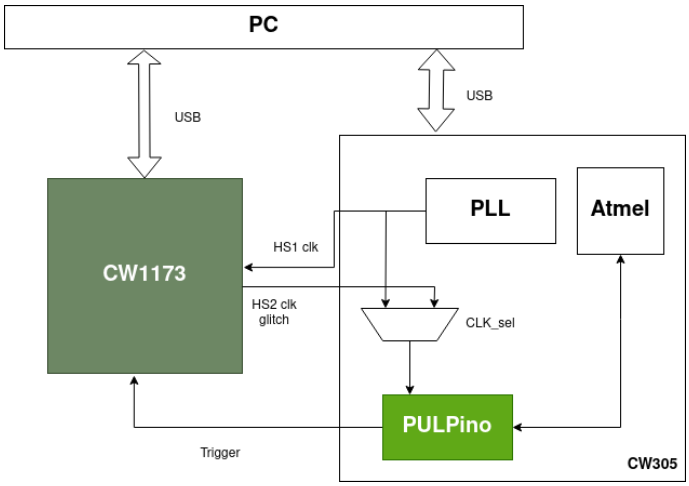

Fig. 1. The hardware platform

IV. INTERNAL STATE MONITOR

In the previous sections, we have demonstrated the importance of a thorough insight of the internal state of the CPU. Here, we describe the Internal State Monitor, a processor extension granting a software-level access to a precise snapshot, cycle by cycle, of a few selected components. Unlike the trace encoder described in the previous section, any architectural element can be monitored; on the other hand, the temporal

horizon that we aim for is limited to very few clock cycles, as monitoring the system for a long time would incur in a huge overhead in terms of additional memory. In our context, however, this is a reasonable limitation: our goal is to analyze the impact of fault attacks (techniques and parameters) on our target just after the attck, and not do long-term monitoring.

The internal monitor is based on AXI communication and is shown in Figure 2. The module consists of three state machines. The first state machine synchronizes the capture process with the injection, by monitoring the external trigger, together with the proper delay parameters associated with the glitching process. This synchronization is crucial to start filling the table at the precise cycles of interest. The second state machine populates the table with selected signals collected from the architecture. Currently, these signals come from a wrapper, which filters the selected signal in the post synthesis netlist, in particular sequential elements. In the future, the tool will be possibly extended to monitor generic signals as well. The selection of these signals may be based on an algorithm as described in [10], that identifies the minimum set of signals to monitor in order to achieve a certain level of detection. A third state machine allows reading the table through the AXI bus. Analyzing the content of the table will allow the designer to compare between a golden and a faulty execution. It is important to note that, in order to keep the size of the table at reasonable levels, a divide-and-conquer approach may be adopted, and the signals will be likely selected based on specific parts to be observed in the pipeline.

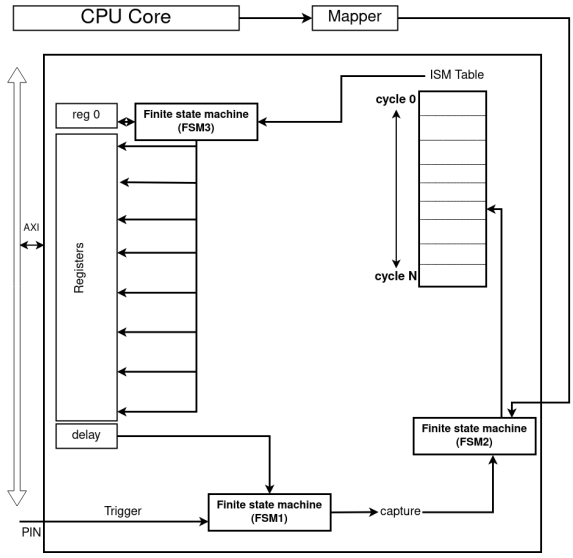

Fig. 2. Architecture of the Internal State Monitor

On the software side, communication between the CPU and the AXI bus is established through the use of write and read instructions. The CPU starts communication by sending a specific address to the AXI bus.

The trigger signal plays a crucial role in synchronizing the glitching process, in Figure 3, and initiating the onset of capturing the signals from the wrapper module to be the input of the ISM table in each clock cycle. The State Machine waits for the delay, a critical parameter that determines the time between arming the glitch and the actual execution of instructions. Once the delay is expired, the capturing process begins. During this phase, signals from the wrapper module are captured and utilized to populate the ISM table comprehensively. In the final step, the content of this table is transferred upon request by the software into a software accessible register by the third state machine, which initiates the address of the ISM table from which the write process starts.

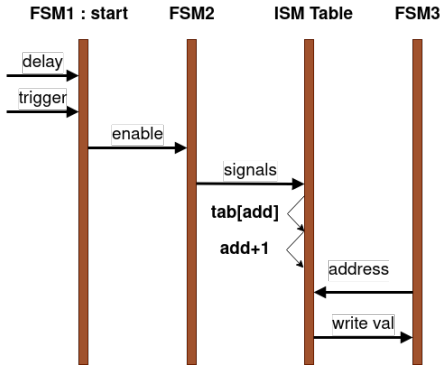

Fig. 3. Process Overview for the Internal State Monitor

V. CONCLUSION

This paper introduced the Internal Monitor State (ISM), a hardware processor extension designed to enhance the software observability of specific microarchitectural components in the RISC-V processor. The ISM module will contribute in obtaining a comprehensive understanding of the internal state after fault injection, with the goal to increase the security of the processor. Our focus is on advancing the monitoring module to enable the verification of control flow and instruction execution integrity.

ACKNOWLEDGMENTS

This work is supported by the ARSENE project, funded by the "France 2030" government investment plan managed by the French National Research Agency (ANR-22-PECY-0004).

REFERENCES

[1] A. Zgheib, et al., "A CFI Verification System based on the RISC-V Instruction Trace Encoder," 2022 25th Euromicro Conference on Digital System Design (DSD).
[2] L. Claudepierre, et al. "TRAITOR: A Low-Cost Evaluation Platform for Multifault Injection," in Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems.
[3] T. Korak et M. Hoefler,"On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms", in 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, sept.
[4] M. A. Elmohr, H. Liao and C. H. Gebotys, "EM Fault Injection on ARM and RISC-V," 2020 21st International Symposium on Quality Electronic Design (ISQED).
[5] Trouchkine, Thomas et al. "Fault Injection Characterization on Modern CPUs." 2019 Workshop in Information Security Theory and Practice .
[6] Yuce, Bilgiday et al. "Software Fault Resistance is Futile: Effective Single-Glitch Attacks." 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC) (2016): 47-58..

[7] V. Werner, et al. ”An End-to-End Approach for Multi-Fault Attack Vulnerability Assessment.” 2020 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC).
[8] I. Alshaer, et al.”Variable-Length Instruction Set: Feature or Bug?,” 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 2022.
[9] Natale, Giorgio Di et al., Cross Layer Reliability of Computing Systems, Ed. IET, ISBN 978-1785617973
[10] Stefano Di Carlo, Giorgio Di Natale, and Riccardo Mariani. On-Line Instruction-Checking in Pipelined Microprocessors. In Proceedings of the 2008 17th Asian Test Symposium (ATS ’08).
[11] NewAE Technology Inc. ”ChipWhisperer Documentation”.Available: https://chipwhisperer.readthedocs.io/en/latest/index.html.
[12] RISC-V International, Nov 2020 https://github.com/riscv-non-isa/riscv-trace-spec.