Exploring Fault Injection Attacks on CVA6 PMP Configuration Flow

Kévin Quénéhervé, William Pensec, Tanguy Philippe, Rachid Dafali, Vianney Lapotre

Exploring Fault Injection Attacks on CVA6 PMP Configuration Flow

Kévin Quénéhervé
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
kevin.queneherve@univ-ubs.fr

William Pensec
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
william.pensec@univ-ubs.fr

Philippe Tanguy
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
philippe.tanguy@univ-ubs.fr

Rachid Dafali
DGA MI
Rennes, France
rachid.dafali@def.gouv.fr

Vianney Lapôtre
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
vianney.lapotre@univ-ubs.fr

***Abstract*—Fault Injection Attacks (FIA) pose significant threats to the security and reliability of embedded systems. FIAs can be used to target an embedded processor by manipulating its clock signal, power supply or by using electromagnetic pulses. In this study, we analyze FIA on the Physical Memory Protection (PMP) configuration flow of a CVA6 RISC-V core. Fault injection campaigns targeting an FPGA implementation on an ARTY A7-100T board are performed to characterize the fault effects. For that purpose, we rely on clock glitches. Moreover, in order to further characterize the induced faults, Error-Correction Code (ECC) is considered. We extend the *ID* pipeline stage with hardware modules to filter faults using Hamming code.**

Experimental results demonstrate that FIA has multiple effects on the PMP configuration registers. By classifying these effects in regards with injection parameters, we highlight that a given effect can be obtained with high probability by an attacker. Furthermore, thanks to integrated ECC modules used as filters, we confirm that single bit-flips is a prevalent effect in our experiments. Particularly, results demonstrate that numerous fault effects observed in the PMP configuration registers are caused by single bit-flips in the *ID* stage of the CVA6 core.

***Index Terms*—Hardware Security, Fault Injection Attacks, RISC-V, Physical Memory Protection, Hamming codes, FPGA**

I. Introduction

Many implementations of processors based on RISC-V open-source ISA have been proposed since the first specification was released in 2011 [1]. They have been used in many embedded systems, with applications ranging from low-resource-embedded systems running bare-metal applications to embedded systems capable of running a Linux operating system.

As security is a major concern in embedded systems, the RISC-V foundation has a Security Standing Committee to develop a consensus on best security practices. Several working groups have discussed security extensions to the ISA, such as the Cryptographic Extensions Task Group and the Trusted Execution Environment (TEE) Task Group. Privileged instructions and Physical Memory Protection (PMP) are described in the specifications [2]. The PMP mechanism is optional, but as it plays an important role in the security of systems, it is supported by many processor implementations. To illustrate, the PMP is a crucial security mechanism in the construction of TEE, such as Keystone [3] and Hex-Five [4].

Embedded systems are subject to many types of physical attacks. Among them, Fault Injection Attacks (FIA) [5], [6] pose a significant threat to the security and reliability of embedded systems. Several injection fault techniques exist like voltage pulse, laser and ElectroMagnetic (EM) pulse, temperature and clock glitching. In this context, Nashimoto et al. [7] have shown the possibility to modify the PMP configuration registers on a RISC-V processor through FIA using clock glitching. Although PMP was not originally designed to withstand physical attacks, we think that it would be beneficial to investigate in more details the effect of FIA on PMP. Indeed, many security applications depend on PMP robustness. It is also worth noting that in the majority of cases an attacker does not have a direct access to the clock because it is integrated. Nevertheless, clock glitching remains an interesting case study to characterize faults and countermeasure against FIA. As demonstrated in [8], the faults caused by EM Fault Injection are similar to those caused by timing glitches like the clock glitch. In [9], a injection fault technique using the clock signal is used in order to emulate fault caused by an EM pulse.

The process of developing solutions to protect and mitigate the effects of fault injection on microarchitectures involves two essential phases. The exploration phase allows the nature of the fault and its impact on the architecture and execution flow to be identified. The modelling phase allows the construction of the appropriate model. Exploring the effects of faults on a hardware implementation of a microarchitecture is complicated for the following reasons: (i) The physical effect of the injection observed may vary from one campaign to another over time due to variations in certain factors due to changes in ambient temperature, component aging, etc.; (ii) The impossibility of recovering all the values of all the signals and registers during

an injection, because the logic used to recover this data may also be affected by the effect of the injection; (iii) It is also very difficult to compare two implementations, one that is immune and another that suffers the effects of a fault on an FPGA component. Indeed the architecture of the FPGA favours the diffusion of the fault. These observations led us to consider a new methodology for exploring the effects of faults on a microarchitecture. This exploration is based on filtering to classify observed faults effects.

The main contributions of this paper are :

- we study the impact of clock glitching on the PMP configuration flow of the RISC-V CVA6 core [10];
- we classify the observed impacts on PMP configuration registers regarding both fault effects and injection parameters;
- we introduce a novel approach based on fault filtering to better characterize fault effects at the microarchitecture level;
- we highlight the interest of our proposal by demonstrating the prevalence of single bit-flips in the *ID* pipeline stage of the CVA6 core by relying on ECC hardware modules used as filters.

This article is organized as follows: Section II presents related works. Section III presents the considered threat model and provides background information on CVA6 PMP configuration. Section IV presents our experimental environment and results. Section V evaluates the use of an ECC-based filter in characterizing the effects of faults. Section VI discusses the proposed work. Section VII draws some perspectives. Finally, Section VIII concludes this article.

II. Related work

Fault injection attacks pose a significant threat to the reliability and security of embedded systems. [11]–[13] present the different techniques of fault injection based on voltage glitch, electromagnetic emanation, clock glitch, laser pulse and temperature manipulation.

These types of FIA can disrupt systems in a number of ways, such as controlling the Pointer Counter (PC) [11], bypass secure boot [14] and modifying the memory hierarchy and MMU [15]. Different types of Fault Injection Attacks can lead to comparable faults. For example, faults induced by electromagnetic injection could produce effects related to those caused by clock offsets as shown in [9]. Moreover, its crucial to distinguish the various effects of faults caused by FIA, as demonstrated in [16].

In [7], Nashimoto et al. show that the PMP configuration of an RISC-V processor is vulnerable to physical attacks by FIA using clock glitch. Despite proposing a set of countermeasures, none have been evaluated.

In our study, we used the same FIA technique to target the PMP configuration on an RISC-V processor. However, in contrast to [7], we detail the effects of the injected faults on the PMP configuration registers in order to help the designer to implement adapted protection schemes. Indeed, various protection methods exist against FIA [5]: error code correction/detection [17]–[19], information and temporal redundancy [20]–[23], randomization [24], [25], Control Flow Integrity (CFI) [26], etc. However, some of these countermeasures can be complex and/or expensive in term of silicon area. Consequently, to propose an efficient implementation, a designer must be aware of the actual fault injection effects in order to to select the most area and timing efficient countermeasures.

A number of different approaches are available to characterize faults effects in the microarchitecture of processors. In [27], authors use simulation and an FPGA implementation extended with an integrated logic analyzer to characterize fault in RISC-V core. In [28], Trouchkine et al. propose a method based on an ISA fault model to characterize fault at the microarchitectural level for a modern processor. Tollec et al. in [29] introduce a formal workflow for modeling software/hardware systems in order to explore the effects of fault injections and they apply their method to RISC-V core. Our paper examines the interest of an Error-Correction Codes-based approach, not as a countermeasure, but to further characterize actual fault effects. To demonstrate the interest of the proposed approach, we rely on Hamming code to filter the occurrences of single bit-flips in the *ID* pipeline stage of the CVA6 core during the PMP configuration process.

III. Background

This section describes the considered threat model and provides background information regarding the CVA6 architecture and its PMP configuration flow.

A. Threat Model

We assume that an attacker aims to bypass PMP mechanism to access sensitive data or execute an arbitrary code. We consider that the attacker program does not have the privilege to modify the PMP configuration. We assume that the attacker has physical access to the target device and can perform FIA by relying on clock manipulation. We consider the attacker is able to precisely synchronize its attacks by relying, for instance, on GPIO activity. In this study, we do not consider side-channel analysis, reverse engineering, software, and network-based attacks.

B. CVA6 architecture and PMP configuration flow

Figure 1 shows an overview of the CVA6 core. It is a 6-stage, single issue, in-order core that implements RV32GC or RV64GC extensions with three privilege levels, M, S, and U, to fully support a Unix-like operating system. In this study, we consider a 64-bits version of this processor. RISC-V processors rely on a hardware module called Physical Memory Protection (PMP) to enforce access permissions to a set of memory regions, called *PMP regions*.

The RISC-V processor can configure a maximum of 16 PMP regions. They are configurable using Control Status Registers (CSR). These registers must be configured in high-privilege (M). PMP consists of a set of configuration registers called `pmpcfg` and address registers called `pmpaddr`.

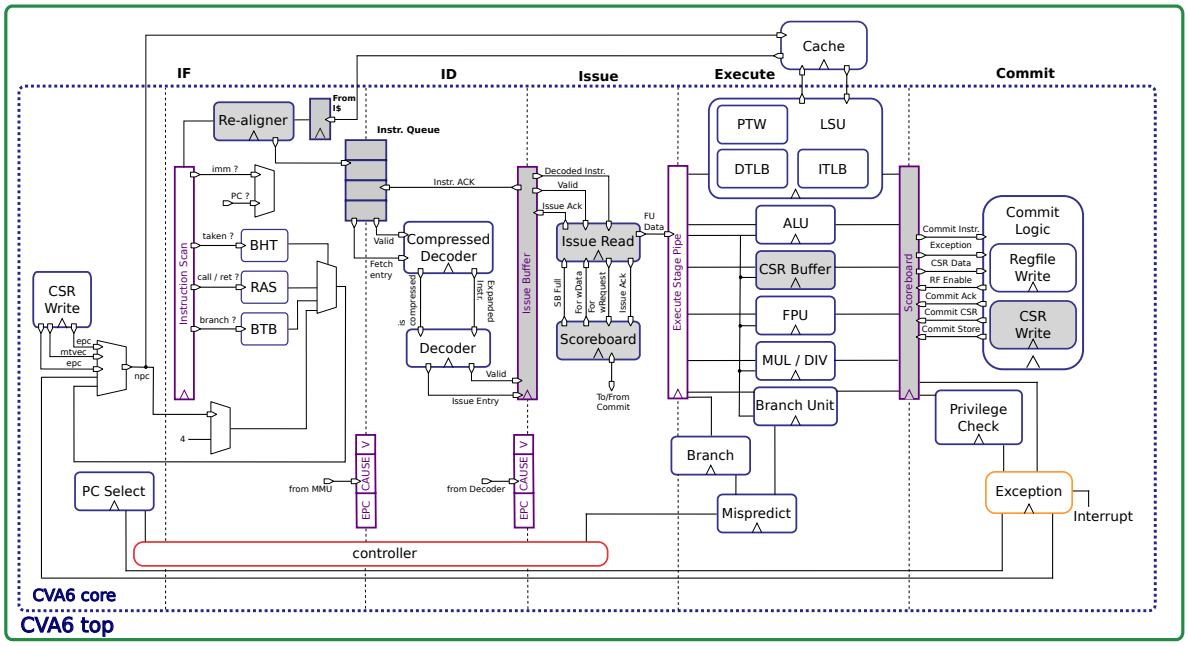

Fig. 1: CVA6 core architecture overview. Boxes in grey highlight registers involved into the PMP configuration flow along the CVA6 pipeline. ID: Instruction Decode. IF: Instruction Fetch

`pmpcfg` define the access permissions to a set of memory region which is defined through *0* or to *2*, `pmpaddr` registers depending on the addressing mode.

The RISC-V specification [2] defines the structure of `pmpcfg`. Each PMP region is configured through an 8-bit vector called `pmpicfg` with $0 \leq i \leq 15$. 8 configurations are grouped, forming a 64-bit register refers as `pmpcfgj`. The 64-bit CVA6 core implements 2 `pmpcfg` registers named `pmpcfg0` and `pmpcfg2`. Each `pmpicfg` configures the following permissions: L, A, X, W and R. Among them, X, W and R represent execute, write and read access permissions respectively. L signifies PMP region configuration is locked, i.e. any attempt to write in the corresponding configuration register is ignored. Locked PMP regions remain locked until the hart is reset. When the L bit is set, access permissions are enforced for all privilege modes. The A bit represents the addressing mode. Usually either NAPOT (Naturally Aligned Power-Of-Two) or TOR (Top Of Range) mode is used. In NAPOT mode, `pmpaddri` stores both memory region size and base address. In TOR mode, `pmpaddri` stores the address of the top of the memory region, and `pmpaddri-1` stores the address of the bottom of the memory region.

PMP configuration registers are stored in CSR registers in the *Commit* stage of the CVA6 core (module *CSR Write* in Figure 1). A *CSR* instruction is used to write these registers. For each load/store operation performed by the Load/Store Unit (LSU), the source/destination memory address is confronted with the PMP configuration to check the access permissions.

Figure 1 highlights, with grey boxes, hardware modules involved into the PMP configuration flow along the CVA6 pipeline.

In the *Instruction Fetch* (*IF*) stage, a CSR instruction is fetched from the memory cache, re-aligned if necessary, and stored for later decoding. Then, in the *Instruction Decode* (*ID*) stage, the CSR instruction is decompressed if necessary and decoded. During the *Issue* stage, components such as the *scoreboard* and *issue read* manage further processing, facilitating instruction scheduling and dependency resolution. In the *Execute* stage, the index of the targeted CSR register is determined and stored in the *CSR Buffer* register. Finally, in the *Commit* stage, the instruction payload is stored in the corresponding PMP configuration register included in the *CSR Write* module.

This analysis shows that each pipeline stage is involved for processing CSR instructions in order to configure the CVA6 PMP. Therefore, a fault injection at any stage can affect the PMP configuration. The next section details our experiments to further analyze the effects of fault injections on the CVA6 PMP configuration flow.

IV. Experiments

This section describes the experimental setup and the scenario for fault evaluation during PMP configuration. Then, it analyzes the impacts of faults encountered when targeting the CVA6 core. To deepen our analysis, we propose to extend the *ID* stage with ECC modules to filter single bit-flips at the microarchitectural level. To achieve this, we rely on Hamming code-based ECC. Finally, we analyze the impact of the proposed filtering methodology on observed fault effects at the PMP configuration level.

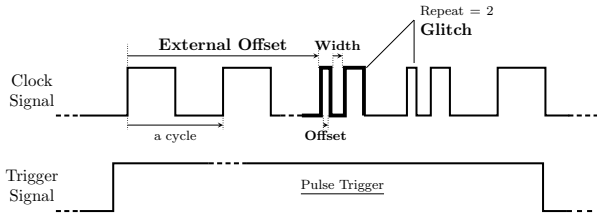

Fig. 2: Glitch clock principle *ex. Repeat = 2*

A. Evaluation setup

We built a system-on-chip (SoC) with the Litex framework [30]. The SoC includes a CVA6 core, a RAM memory, GPIOs, and an UART. This SoC is implemented on a Digilent Arty A7-100T FPGA board [31].

Fault injection is realized using the Chipwhisperer Lite [32] and its associated software suite [33]. In order to produce clock gliching the Chipwhisperer Lite [32] provides a 25 MHz clock signal to the target FPGA board.

For each fault injection campaign, we explore the injection parameters presented in Figure 2: 1) *Width* refers to the percentage of a single clock period ranging from -49 to 49 that determines the width of the the glitch; 2) *Offset* refers to the percentage of a single clock period ranging from -49 and 49 that determines the placement of the glitch within the glitched period; 3) *External Offset* determines the duration of the delay between a trigger and a glitch, expressed in clock cycle from 0 to 200; 4) *Repeat* refers to the number of times the glitch will be repeated on subsequent clock cycles, as shown in Figure 2 with *Repeat* equals 2. In our experiments, this parameter is set to 5. As a result, a fault injection campaign leads to a total of 1,970,001 injections.

The software running on the CVA6 core used in this evaluation has been instrumented to generate a *trigger* signal via a dedicated GPIO. Furthermore, the UART is used for communicating with the target device to read the PMP configuration state following each fault injection.

B. Evaluation scenario

For this evaluation, we consider a dedicated software, running in M-MODE, that first configures a protected PMP region in NAPOT mode as in [7]. For that purpose, CSR instructions are executed to writes `0x99` into `pmpcfg0` and `0x800018F` into `pmpaddr0`. The other PMP configuration registers are not configured (i.e. their value is null). This configuration fixes a read-only permission on a 128-bytes memory region starting from address `0x20000600`. Moreover, it locks the PMP configuration. Thus, following these CSR instructions, the PMP configuration cannot be modified anymore. Once the configuration process is done, the software attempts to write into the protected memory region (this action is considered to be a malicious operation that is being performed by an attacker). This operation leads to a exception as it is not allowed by the PMP configuration. However, in presence of faults disturbing the PMP configuration flow, the resulting configuration could allow accesses to the protected memory region. Finally, the software communicates, for each execution, the state of the PMP configuration through a the serial communication.

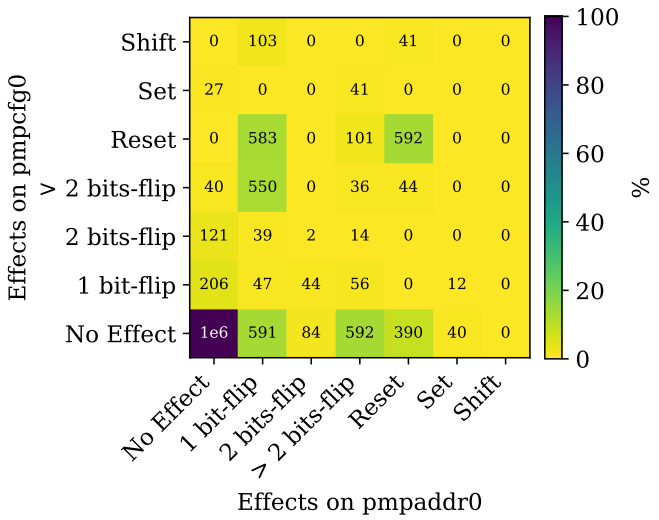

Fig. 3: Fault injection effect combinations - groups `G2` & `G3`

It is worth noting that the fault injection timing is triggered through a GPIO raised before the PMP configuration. In the next section, we rely on this scenario to analyze the effect of faults injected during the PMP configuration.

C. Effects of FIA on PMP configuration

The study conducted by [7] highlighted the vulnerability of the PMP configuration to FIA, but it does not provide a detailed analysis of the fault effects on PMP configuration registers. Consequently, we performed a fault injection campaign relying on the experimental setup and the scenario presented in section IV-A and IV-B respectively. Among 1,970,001 fault injections, 5,561 lead to a modification of the PMP configuration including 4,267 allowing the executed program to write into the protected memory region. We group these 5,561 fault injections in three groups.

The first group (`G1`) gathers faults that lead to *complex* effects leading, for instance, to the storage of *random* values into one or several PMP configuration registers, the storage of a faulty value into several PMP configuration registers or the storage of the correct value in a PMP configuration register which is not the targeted one. 1,165 fault injections over 5,561 are classed in this group.

The second group (`G2`) gathers faults that impact a single register among all PMP configuration registers. In this group, either `pmpcfg0` or `pmpaddr0` are impacted (these two registers are the ones configured by the executed software) while others PMP configuration are not impacted. 2,091 fault injections over 5,561 are classed in this group. The First column and the last row of Figure 3 details, for this group, the number of fault injections leading to a specific fault effect. It shows that a fault injection can lead to bit(s)-flip and register set or reset. We observe that for both `pmpcfg0` and `pmpaddr0`, single bit-flip is the most common effect. The

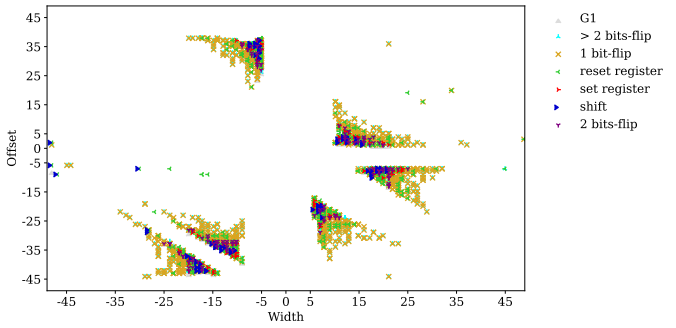

Fig. 4: Fault effects in regards with injection parameters between Width and Offset

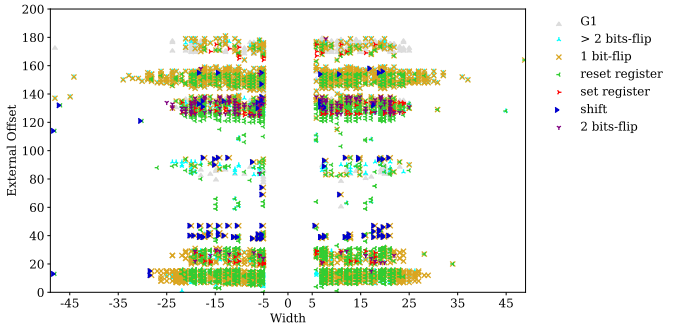

Fig. 5: Fault effects in regards with injection parameters between Width and External Offset

second and third most common effects are multiple bits-flip and register reset.

The third group (`G3`) gathers faults that impact both `pmpcfg`*0* and `pmpaddr`*0* but do not impact other PMP configuration registers. 2,305 fault injections over 5,561 are classed in this group. Figure 3 shows that a fault injection can lead to couple of effects in the two configured registers. These couple of effects are built around bit(s)-flip, register reset or value shift in the configured register. It is worth noting that 1,322 injections over 2,305 lead to single bit-flips in a least one register.

Figure 4 highlights the relation between the observed fault effects and the *Width* and *Offset* injection parameters. We observe 6 sensitive zones where all fault effects can be observed (two symbols are used for fault effects of group `G3`, i.e. one symbol per faulty register). Furthermore, Figure 4 shows that in each sensitive zone, sub-zones corresponding to parameters leading to a specific effect can be delimited. This is particularly true for the single bit-flip effect which mainly appears on the border of each zone.

Figure 5 illustrates the relationship between observed fault effects and the injection parameters *Width* and *External Offset* and Figure 6 shows the relation between observed fault effects and the *Offset* and *External Offset* injection parameters. These figures facilitate the observation of the impact induced by the *External Offset* parameter on fault injection effects. As presented in Figure 2, this parameter allows an attacker to reach specific clock cycles from the *trigger* signal. Figure 5 and Figure 6 show that the occurrences of some effects are highly

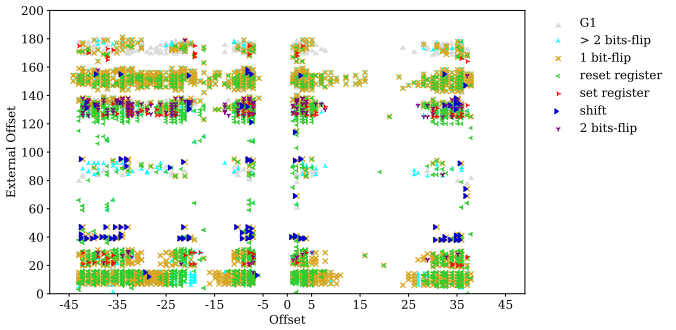

Fig. 6: Fault effects in regards with injection parameters between Offset and External Offset

correlated to this parameter and then, the set of instruction treated in each pipeline stages when a fault injection occurs. Moreover, by analysing Figure 4, Figure 5 and Figure 6, one can conclude that an attacker would be able, with a high probability, to produce the desire effect by tweaking injection parameters. For instance, by setting *External Offset* between 80 and 100, *Offset* close to 25 and *Width* close to -5, an attacker would expect a multi-bits fault effect while, by setting *External Offset* around 40, *Offset* close to 35 and *Width* close to -5, he would expect a value shift effect.

In this section, we have studied the fault injection effects on the PMP configuration registers. However, the previous analysis does not provide information regarding the faults locations and effects at the microarchitectural level leading to the observed effects in the PMP configuration registers. In order to further study these effects, the next section introduces an original approach based on fault filtering used to link fault effects observed at the PMP configuration registers level with a given fault effect at the microarchitectural level.

V. Hamming Code as a filter in the PMP configuration flow

The previous section has shown that FIA, relying on clock glitching, can lead to numerous effects in the resulting PMP configuration. In this section, we consider a lightweight ECC based on Hamming Code to correct single bit-flips impacting registers participating to the PMP configuration flow of the CVA6 core. In the community, this approach is commonly used to protect registers from the effects of fault injections. Our objective is different. Indeed, we use it to filter faults at different location in the CVA6 *ID* pipeline stage in order to better characterize the observed fault effects by linking a fault effect at the microarchitecture level to observed faults at the PMP configuration level. Therefore, the rest of this article will not refer to protection but to *filtering*.

Hamming Code is a class of linear error-correcting codes proposed by Richard W. Hamming [17] in 1950. The main use of this code is to detect and correct errors. It is mostly used in digital communication and data storage systems as error control codes. Hamming Code can detect and correct single-bit errors or detect double-bits errors. Regarding hardware

implementation, we rely on the solution presented in [34]. Such a solution relies on an *Encoder* module in charge of generating the redundancy, a register to store this information and a *Decoder* to detect and eventually correct faults.

As presented in III-B, the PMP configuration flow involves a set of components in all pipeline stages (see Figure 1, grey boxes). In order to further analyze the cause of the observed fault effects, we choose to focus on the *ID* stage since instruction decoding is a highly sensitive process. Thus, we extend *ID* stage registers involved in the configuration flow of the PMP. To demonstrate the interest of the proposed approach we focus, in this paper, on single bit-flips by relying on a Hamming Code-based solution.

Table I lists the *ID* stage registers that are extended to filter single bit-flips induced by fault injections. Table II shows implementation results targeting the Xilinx Artix-7 of the Digilent Arty A7-100T development board for Hamming Code-based filter implemented in *ID* stage only. Implementation results are obtained using Vivado 2019.1. Results presented in Table II show that the resources overhead of the filters implemented in the *ID* stage is 5.6% for LUTs and 1.9% for FFs.

In order to evaluate the interest of the proposed filtering approach to further analyze the fault injection effects at the microarchitectural level, we perform an additional fault injection campaign targeting the extended version of the CVA6 core and relying on the evaluation setup and scenario presented in section IV-A and IV-B. Table III compares fault injection results for the baseline CVA6 core and a version in which single bit-flips are filtered in the *ID* stage. *Crash* column indicates the number of system failures encountered during the experiment. *Silent* column represents the number of silent fault, i.e. fault injections that did not cause a faulty PMP configuration on `pmpcfg0` and `pmpaddr0`. *Faults* indicates the number of faults observed on PMP configuration registers (i.e. `pmpcfg0` and `pmpaddr0`).

Results presented in Table III shows that filtering single bit-flits in the *ID* pipeline stage allows to reduce the total number of observed faults in PMP configuration registers from 5,561 to 1,783 (-68%). These results confirm that the single bits-flip effect prevails in the *ID* stage since the proposed Hamming Code-based filters single bit-flips only. Among the 1,970,001 fault injections, 1,783 leads to a modification of the PMP configuration. We group these 1,783 fault injections in three groups as presented in section IV-C.

The first group (`G1`) gathers faults that lead to *complex* effects. 248 fault injections over 1,783 are classed in this group. The second group (`G2`) gathers faults that impact a single register, either `pmpcfg`*0* or `pmpaddr`*0*. 487 fault injections over 1,783 are classed in this group. The third group (`G3`) gathers faults that impact both `pmpcfg`*0* and `pmpaddr`*0*. 1,048 fault injections over 1,783 are classed in this group. Table III allows a comparison with results obtained considering the CVA6 baseline core. We observe that filtering single bit-flips in the *ID* stage leads to a significant reduction of faults occurrences in each groups (`G1`, `G2`, `G3`).

Figure 7 compares the different types of fault effects of groups `G2` and `G3` considering the baseline CVA6 core and the CVA6 core extended with the proposed filters in the *ID* pipeline stage. Results show that filtering single bit-flips in the *ID* stage leads to reducing the occurrences of almost all fault effects observed in the PMP configuration registers. For instance, the "`No Effect/>2 bits-flip`" fault effect decreases from 592 to 49 occurrences. Such results demonstrate that single bit-flips in the *ID* stage actually lead to more complex faults in the PMP configuration registers. Consequently, implementing such filter provides a promising solution to identify the actual fault effect at the microarchitectural level and develop tailored countermeasures.

VI. Discussion

The results provided in Section IV demonstrate that FIA relying on clock glitches has multiple effects on the final PMP configuration of the CVA6 core. Our work shows that observed fault effects can be associated to a set of injection parameters. Furthermore, it demonstrates that single bit-flips are a common effect of such injections.

The set of experiments proposed in this paper show that using ECC can help to characterize fault effects. Indeed, relying on Hamming Code allows us to confirm that single bit-flips in the *ID* stage are one of the fault effects leading to divers PMP configuration divergences including fault effects such as register set, multiple bit-flips, value shift, etc. Thus, we believe that a methodology for fault effect characterization relying on similar approaches should be explored (e.g., relying on more complex ECC, such as BCH, Reed-Solomon, etc [17]–[19]).

VII. Future works

In this paper, we focus our analysis on the single bit-flip fault effect to demonstrate the interest of the proposed approach. We are conscious that additional studies are necessary to reinforce and extend this work to tackle multiple bit-flips, shift, register set/reset and complex (group `G1` in the paper) fault effects. Regarding register reset, the *instruction skip* fault model can also explain this behavior. In this case, we believe that a dedicated detection mechanism could allows to identify this fault effect. Considering the complex fault effects, faulty control signals or register indexes could be the main causes. The proposed ECC-based filters should also be able to filter most of these fault effects as demonstrated by our experiments considering the *ID* stage in which the number of complex fault effects has been reduced from 1,165 to 248 using single bit-flip filters. However, additional experiments are required to confirm these claims. We plan to apply the proposed approach to other pipeline stages of the CVA6 core, consider more complex filters to enhance the proposed characterization methodology and consider different processor cores and implementations. Furthermore, based on the proposed characterization we will investigate the design of dedicated FIA countermeasures taking into account area and energy constraints.

TABLE I: Summary of Filtered Registers by Hamming Codes in *ID* stage

Registers	width	Meaning
Valid	1	issue entry is valid
rs1	6	register source address 1
rs2	6	register source address 2
trans _id	3	this can potentially be simplified, we could index the scoreboard entry
use_imm	1	should we use the immediate as operand b?
use_zimm	1	use zimm as operand a
use_pc	1	set if we need to use the PC as operand a, PC from exception
valid	1	-
cf	3	type of control flow prediction
predict_address	64	branch predict scoreboard data structure
cause	64	cause of exception
op	7	operation to perform in each functional unit
pc	64	PC of instruction
rd	6	register destination address
result	64	for unfinished instructions this field also holds the immediate
valid	1	is the result valid
tval	64	additional information of causing exception (e.g.: instruction causing it), address of LD/ST fault
fu	4	functional unit to use
is_compressed	1	signals a compressed instructions, we need this information at the commit stage if we want jump accordingly
is_ctrl_flow	1	the instruction we issue is a control flow instructions
Total	363	
Number of Redundancy Bits for Hamming Code	10	

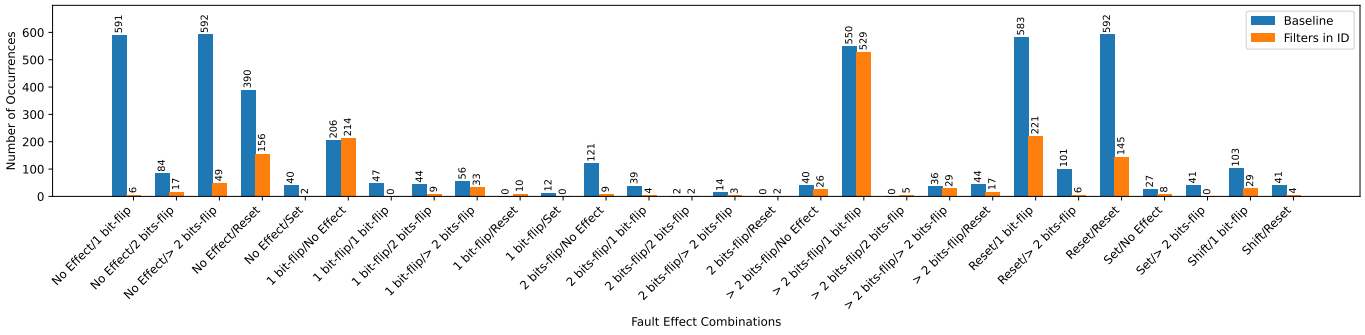

Fig. 7: Fault injection effects on PMP registers - group `G2` & `G3` with Hamming Code-based filters in *ID* stage

TABLE II: FPGA implementation results in LUTs and FFs

CVA6 core	Number of LUTs	Number of FFs
Baseline	47,955	29,303
Filters in *ID*	50,627 (+5.6%)	29,849 (+1.9%)

TABLE III: fault injection results comparison between baseline and filtered CVA6 core - with 1,970,001 faults injected per campaign

CVA6 core	Crash	Silent	Faults			
			`G1`	`G2`	`G3`	Total
Baseline	50,146	1,914,294	1,165	2,091	2,305	5,561
Filters in *ID*	122,047	1,846,171	248	487	1,048	1,783

VIII. Conclusion

This paper provides an analysis of the effect of FIA relying on clock glitches on the CVA6 PMP configuration. It highlights that such attacks lead to multiple effects that can be exploited by an attacker. Furthermore, this study analyzes the efficiency of a Hamming Code ECC to filter faults in the *ID* stage of a CVA6 core. Results show that such approach can help during the characterization phase when multiple fault effects are observed. This tool is also interesting because it has a little impact on resource used in FPGA when it comes to instrument a design. Indeed, in our case study the FPGA resources increased by 5.6% for LUTs and 1.9% for FFs compared with the baseline design.

References

[1] A. Waterman, Y. Lee, D. A. Patterson, and A. Krste. "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA," RISC-V. (), [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.pdf.

[2] "Volume 2, Privileged Specification version 20211203," RISC-V. (), [Online]. Available: https://riscv.org/technical/specifications/.

[3] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An Open Framework for Architecting Trusted Execution Environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, Association for Computing Machinery, 2020. DOI: 10.1145/3342195.3387532.

[4] "Hex-Five security," HexFive. (), [Online]. Available: https://hex-five.com/.

[5] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, 2006. DOI: 10.1109/JPROC.2005.862424.

[6] Karaklajić, Duško and Schmidt, Jörn-Marc and Verbauwhede, Ingrid, "Hardware Designer's Guide to Fault Attacks," *IEEE Transactions on Very Large Scale Integration Systems*, 2013. DOI: 10.1109/TVLSI.2012.2231707.

[7] S. Nashimoto, D. Suzuki, R. Ueno, and N. Homma, "Bypassing Isolated Execution on RISC-V using Side-Channel-Assisted Fault-Injection and Its Countermeasure," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021. DOI: 10.46586/tches.v2022.i1.28-68.

[8] R. Nabhan, J.-M. Dutertre, J.-B. Rigaud, J.-L. Danger, and L. Sauvage, "A tale of two models: Discussing the timing and sampling em fault injection models.," in *2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 2023. DOI: 10.1109/FDTC60478.2023.00010.

[9] P. Maistri and J. Po, "A low-cost methodology for em fault emulation on fpga," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022. DOI: 10.23919/DATE54114.2022.9774507.

[10] F. Zaruba and L. Benini, *The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology*, 2019. DOI: 10.1109/TVLSI.2019.2926114. [Online]. Available: https://github.com/openhwgroup/cva6.

[11] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM Using Fault Injection," in *Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2016. DOI: 10.1109/FDTC.2016.18.

[12] B. Yuce, P. Schaumont, and M. Witteman, "Fault attacks on secure embedded software: Threats, design, and evaluation," *Journal of Hardware and Systems Security*, 2018. DOI: 10.1007/s41635-018-0038-1.

[13] Breier, Jakub and Hou, Xiaolu, "How Practical Are Fault Injection Attacks, Really?" *IEEE Access*, 2022. DOI: 10.1109/ACCESS.2022.3217212.

[14] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, "Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2017. DOI: 10.1109/FDTC.2017.18.

[15] T. Trouchkine, S. K. K. Bukasa, M. Escouteloup, R. Lashermes, and G. Bouffard, "Electromagnetic Fault Injection Against a Complex CPU, toward new Micro-architectural Fault Models," *Journal of Cryptographic Engineering*, 2021. DOI: 10.1007/s13389-021-00259-6.

[16] O. Trabelsi, L. Sauvage, and J.-L. Danger, "Characterization of electromagnetic fault injection on a 32-bit micro-controller instruction buffer," in *Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2020. DOI: 10.1109/AsianHOST51057.2020.9358270.

[17] Hamming, R. W., "Error detecting and error correcting codes," *The Bell System Technical Journal*, 1950. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[18] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proc. of IEEE*, 2012. DOI: 10.1109/JPROC.2012.2188769.

[19] R. Schilling, P. Nasahl, S. Weiglhofer, and S. Mangard, "Secwalk: Protecting page table walks against fault attacks," in *Proc. International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2021. DOI: 10.1109/HOST49136.2021.9702269.

[20] N. Theißing, D. Merli, M. Smola, F. Stumpf, and G. Sigl, "Comprehensive analysis of software countermeasures against fault attacks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013. DOI: 10.7873/DATE.2013.092.

[21] A. Barenghi, L. Breveglieri, I. Koren, G. Pelosi, and F. Regazzoni, "Countermeasures against fault attacks on software implemented aes: Effectiveness and cost," in *Proceedings of the 5th Workshop on Embedded Systems Security*, Association for Computing Machinery, 2010. DOI: 10.1145/1873548.1873555.

[22] V. B. Thati, J. Vankeirsbilck, J. Boydens, and D. Pissort, "Selective duplication and selective comparison for data flow error detection," in *International Conference on System Reliability and Safety (ICSRS)*, IEEE. DOI: 10.1109/ICSRS48664.2019.8987731.

[23] P. Kiaei, D. Mercadier, P.-E. Dagand, K. Heydemann, and P. Schaumont, "Custom instruction support for modular defense against side-channel and fault attacks," in *Constructive Side-Channel Analysis and Secure Design*, Springer International Publishing, 2021. DOI: 10.1007/978-3-030-68773-1_11.

[24] G. Leplus, O. Savry, and L. Bossuet, "Insertion of random delay with context-aware dummy instructions generator in a risc-v processor," in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2022. DOI: 10.1109/HOST54066.2022.9840060.

[25] B. Wang, L. Liu, C. Deng, M. Zhu, S. Yin, and S. Wei, "Against double fault attacks: Injection effort model, space and time randomization based countermeasures for reconfigurable array architecture," *IEEE Transactions on Information Forensics and Security*, 2016. DOI: 10.1109/TIFS.2016.2518130.

[26] A. Zgheib, O. Potin, J.-B. Rigaud, and J.-M. Dutertre, "A CFI Verification System based on the RISC-V Instruction Trace Encoder," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022. DOI: 10.1109/DSD57027.2022.00067.

[27] J. Sharma and N. Rao, "The characterization of errors in an fpga-based risc-v processor due to single event transients," *Microelectronics Journal*, 2022. DOI: https://doi.org/10.1016/j.mejo.2022.105392.

[28] T. Trouchkine, G. Bouffard, and J. Clédière, "Fault injection characterization on modern cpus: From the isa to the micro-architecture," in *Information Security Theory and Practice: 13th IFIP WG 11.2 International Conference, WISTP 2019, Proceedings*, Springer-Verlag, 2020. DOI: 10.1007/978-3-030-41702-4_8.

[29] S. Tollec, M. Asavoae, D. Couroussé, K. Heydemann, and M. Jan, "Exploration of fault effects on formal risc-v microarchitecture models," in *2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 2022. DOI: 10.1109/FDTC57191.2022.00017.

[30] E. Digital. "Enjoy Digital Litex framwork." https://github.com/enjoy-digital/litex, Litex. (2019), [Online]. Available: https://github.com/enjoy-digital/litex.

[31] "Arty A7," Digilent. (2019), [Online]. Available: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual?redirect=1.

[32] "Chipwhisperer Lite," NewAETech. (2019), [Online]. Available: https://rtfm.newae.com/Capture/ChipWhisperer-Lite/.

[33] "Chipwhisperer tools," NewAETech. (2019), [Online]. Available: https://github.com/newaetech/chipwhisperer.

[34] W. Pensec, F. Regazzoni, V. Lapotre, and G. Guy, "Defending the Citadel: Fault Injection Attacks against Dynamic Information Flow Tracking and Related Countermeasures," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024. [Online]. Available: https://hal.science/hal-04620057.