Clock-glitch Fault Characterization by Timing Comparison with Laser Fault Injection

Ludovic Claudepierre, Edna Rocio Ferrucho-Alvarez, Laurent Le Brizoual, Laurent Pichon
Univ Rennes, CNRS, IETR (Institut d'Electronique et des Technologies du numéRique)
UMR 6164, Rennes, France
ludovic.claudepierre.1@univ-rennes.fr

***Abstract*—Since clock-glitching and the voltage-glitching are non-localized techniques of fault injection, it is difficult to accurately characterize their fault effects. In contrast, laser fault injection is a localized technique in which the targeted element of the microprocessor is known. In this article, we present a method that exploits the advantages of laser fault injection to characterize the glitch fault effects. We first make an hypothesis about the physical location of the sensitive part that induces the fault by glitch injection. Then we compare the fault timing and execution timing for glitch and laser platforms. If the delay between the fault and the execution is the same for both platforms, the initial hypothesis is supported.**

Keywords-Fault attack, Laser fault injection, Clock glitch, Fault characterization.

I. INTRODUCTION

Fault injection is defined as the introduction of a physical disturbance at microelectronic level with the objective of inducing a malfunction at physical level called a fault. This fault propagates through logic and microarchitecture to finally reach software level to provoke alteration on the program executed by the microprocessor. Fault injection can be used to analyze the robustness of electronic circuits and embedded systems or to exploit vulnerabilities. Faults have the capacity to corrupt data or instructions which can weaken security mechanisms or leads the program to crash. There are several techniques to physically disrupt a microprocessor, including voltage glitch [1], clock glitch [2] [3], electromagnetic fault injection [4] [5], and laser fault injection [6]. Voltage and clock glitches are global injection methods as the perturbation propagates though the whole microprocessor (power line or clock tree). The two methods involve the injection of a brief disruption, either into the voltage line or the clock signal. This disruption propagates through the circuit until it reaches a critical element whose failure results in a fault. Electromagnetic fault injection is a localized method in which the microprocessor is exposed to a local short-range electromagnetic pulse inducing a fault. Laser fault injection involves pointing a focused laser beam at a designated area of the circuit. This causes a transient photocurrent responsible for the switching of a transistor leading to a disruption at logic level.

Manuscript received May 08, 2026; revised July 14, 2026; accepted July 13, 2026. Published August 25, 2026.
Issue category: Special Issue on DSD/SEAA 2026 on Works in Progress (WiP) Session, Krakow, Poland, Sept. 2026.
Paper category: Full
DOI: doi.org/10.64552/wipiec.v12i2.135

Fault effects can be observed at different levels of abstraction. The characterization step consists in understanding and modeling the impact of fault injection at a designated abstraction level. Frequently, the observation is only made on the registers at ISA (Instruction Set Architecture) level. Consequently, the fault model is only based on the comparison of the registers between nominal execution and execution with fault. This step facilitates the elaboration of cyberattacks potentially complex and hard to prevent. However, performing deeper analysis is really challenging. In particular, the propagation mechanism from the physical level to the software level is hard to explain with certainty. This step is even more challenging for glitch attack as the precise area which is failing is unknown. The method presented in this paper aims to address this challenge by using the advantage of the localized laser injection to better describe the fault glitch mechanism.

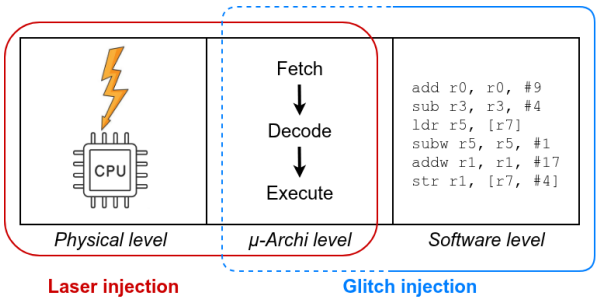


Figure 1: Layer information according to the injection techniques.

When performing glitch fault injection, an hypothesis at the microarchitectural level can be proposed to explain specific fault effects at ISA level. (Figure 1). In particular, an assumption can be made about the pipeline stage sensitive to the glitch injection. Accordingly, a list of the physical elements involved in that stage that could be critical can be made (flash memory, RAM memory, registers). Then, the validity of this hypothesis can be verified by using laser injection. Indeed, as a located fault injection technique, the fault effects observed at ISA level are directly associated with a specific physical element and consequently with certain pipeline stages. Since glitch and laser injection do not result in the same fault effect, the comparison is made by analyzing the fault timing. The new

method developed to perform such analysis is called GARLIC (Glitch chARacterization by Laser Injection Comparison).

In section II, related works on characterization methods and also on clock-glitch capabilities are presented. Then, every step of the GARLIC method is detailed in section III. In section IV and V, the glitch platform TRAITOR and the laser platform are presented, along with the characterization of the fault models chosen to apply GARLIC. In section VI, the conditions of the application of our method to characterize the skip-repeat fault of TRAITOR are detailed. Finally, in section VII, results and conclusions of this study are exposed.

II. Related Work

A. Characterization

In order to construct a fault attack on instruction flow or data flow, it is necessary to characterize the fault effect so as to understand the mechanism at a given level of abstraction. Furthermore, in some cases, we can get information about the propagation mechanism through the different abstraction levels that finally induces a faulty behavior at software level. The precise architecture of the targeted microcontroller is not accessible for devices on-the-shelf. Therefore, the utilization of a simplified model is the only way to depict the fault. The characterization of the observed effects can be made at physical level [7], RTL [8] [9] and ISA level [10].

When characterizing a fault effect, the selection of the test code is a crucial step. The instructions of the test code have to be chosen according to the expected effect - whether it is a skip, skip-repeat, or instruction corruption. Alle Monne *et al.* [11] propose an automated methodology for synthesizing characterization programs using fault model checking on CPU RTL design. Trouchkin *et al.* [12] propose a methodology for determining, in the case of instruction corruption, which part of the instruction is corrupted and in which stage of the pipeline the fault occurs. Moro *et al.* [13] compare the results of fault injection campaigns with simulation where instructions are replaced by alternative ones to determine if there is a match at ISA level. Instrumentation of the program can facilitate the characterization process, for example by enabling automatic identification of the most common fault models [14].

B. Clock-glitch fault injection

Tools such as the ChipWhisperer [15] inject glitches that can provoke faults that are generally described as a consequence of a timing violation or a setup violation. The effects can include instruction skip [16], skip and repeat or instruction alteration as demonstrated by Alshaer *et al.* [17]. Claudepierre *et al.* [3] proposed the platform TRAITOR using a different way to perform a clock glitch by interrupting a rising edge of a clock signal. Regarding this type of glitch, Marotta *et al.* [18] explain that for flip-flops to correctly sample incoming data, the clock signal must reach a given threshold; otherwise, a fault occurs.

III. Method

The objective of the GARLIC method is to obtain information at physical level regarding the fault effects observed when performing fault injection with non-localized injection techniques. This method is applied to a specific fault effect observed on a specific platform. For the experiments a GPIO is used as a trigger to synchronize the injection platform and the target. This signal is the reference for timing measurements and every fault timing is expressed as the delay between this trigger and the fault injection. In order to know the instruction execution duration *i.e* the duration that the instruction remains in the execute pipeline stage, we can either look at the instruction set documentation or make a measurement for example by using the DWT (Data WatchPoint and Trace Unit) debug register if the microprocessor is ARM. Making a measurement is the more accurate option because it takes into account potential stalls, unlike the documentation data which only provides theoretical values. First, we establish a list of terms used throughout this article. We only consider cases where instructions are aligned. Consequently, a word is defined as a group of complete instructions.

- Execution duration of an instruction: The number of cycles that the instruction remains in the execute stage of the pipeline, as described in the documentation.
- Execution duration of one word: The sum of the execution duration of the instructions contained in the word.
- Execution timing of a targeted code: The list of the execution duration for the instructions contained in this code.
- Fault delay: Delay between the trigger and the fault. Depending on the injection type, it can be a number of clock cycles or a delay in nanoseconds.
- Fault timing: List of the fault delays for the instructions composing the test code.

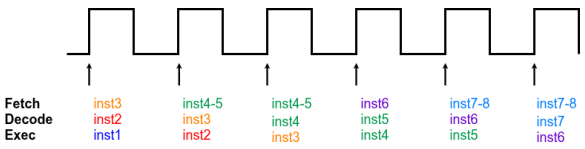


Figure 2: Example of instructions progression in a 3 stage pipeline.

A. Method overview

Laser fault injection is characterized by its spatial locality. The targeted element is used in some specific pipeline stages, for example flash memory is only used during the prefetch. Therefore, faults occur at specific pipeline stages or during the transfer between two stages. Figure 2 illustrates the example of the progression of instructions in the 3 stage pipeline of a Cortex-M3 microprocessor with a 32 bit prefetch buffer and using Thumb-2 instruction set allowing 16 bits instructions. A different color is attributed to each word, thus two instructions with the same color are stored in the same word (for example instructions 4 and 5). An instruction remains in the execute stage during a given number of clock cycles. This imposes on the following instruction to remain in the decode stage for the

same amount of time. Consequently, it also imposes the number of clock cycles that the next instructions remain in the fetch stage. Considering this, the delay between faults on two successive instructions at a given stage depends on the execution duration of the instruction presently in the execute stage. Therefore, by comparing the fault delays for every instruction in the test code with the execution timing, we should be able to determine which instruction is in the execute stage when a fault occurs.

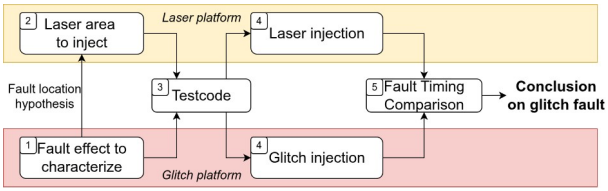


Figure 3: GARLIC method overview.

As illustrated in Figure 3, the method consists of 5 steps. First, the glitch platform and the fault effect we want to characterize are selected. A hypothesis on the potential mechanism responsible for this effect is formulated. From this, the potential fault location is deduced. Second, the location for laser injection is chosen based on this assumption. Third, the test code is written taking into account the constraints imposed by the glitch fault effect to analyze and the laser injection area. Then, the injection campaigns on both platforms are performed. Finally, the fault timing of both injection campaigns are compared. By examining the differences and the similarities, conclusions about the fault mechanism can be drawn, leading either to the support or the rejection of the initial hypothesis.

B. *Glitch platform and type of fault*

The first step is to select a non-localized injection platform to work with. It could be a clock glitch or a voltage glitch platform. The type of fault to be analyzed must be repeatable with a given set of parameters and observable on multiple instructions with different execution duration. At this stage, a hypothesis is formulated concerning the pipeline stage disrupted by the glitch injection.

C. *Area for Laser fault injection*

The hypothesis on the pipeline stage affected by the glitch fault guides the choice of the laser spot location. Therefore, we have to select a physical area whose sensitivity is associated with a specific element (flash memory, RAM, registers). In addition, this element has to be used in a single stage of the pipeline to easily be linked to the fault timing and the involved pipeline stage.

D. *Test code*

The elaboration of the test code for the method application is an essential part of the process. This test code has to be designed to ensure the observation of faults effects with both glitch and laser platforms. Furthermore, the method relies on the comparison between the execution duration of instructions and the delay between the fault of two successive instructions. To obtain a pattern, the test code has to contain successive instructions with different execution duration. As shown in Figure 4, this pattern is observable for execution timing and fault timing with a delay between them. For example, there is one cycle between the moment when instruction 3 is faulted and the moment instruction 4 is faulted. Thus, we know that the instruction executed during that time stays only 1 cycle in the execute stage. By making this comparison for each faulted instruction, we can make the connection between execution duration and fault delay. The execution pattern has to be non-periodic to ensure a single match between execution and fault timing. We note that the timing can be observed for instructions or words, depending on the fault effect on which `GARLIC` is applied.

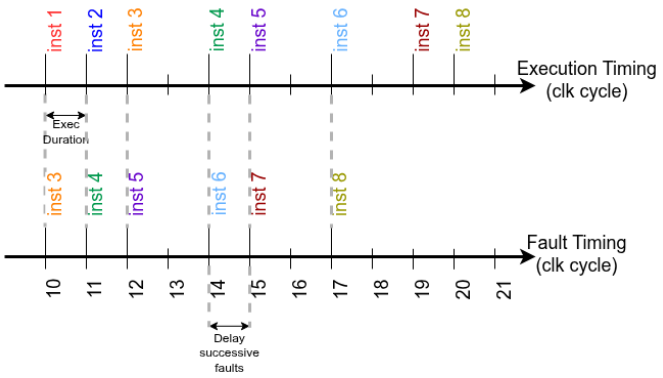


Figure 4: Principle of comparison between fault timing and execution timing.

E. *Injection Campaign and results*

Prior to the method application, a test on the glitch injection platform is performed. The objective of this experiment is to determine the parameter values necessary to achieve the fault effect under study. Also, laser injection parameters are defined to make timing located faults with a precision inferior to one clock cycle. After these steps, the fault injection campaigns can be performed on the test code for both platforms. The campaign sweeps the injection delay such that every instruction is faulted.

First, for laser and glitch injection, the delay between the fault and the execution is measured for every instruction. Then, we can determine which instruction is in the execute stage when faults are observed. This can give an indication of the pipeline stage targeted by the fault. Finally, the fault timing for the laser and the glitch injection are compared with an analysis of similarities and differences between them.

IV. CLOCK-GLITCH PLATFORM: `TRAITOR`

In this part the clock-glitch injection platform `TRAITOR` is presented and the fault effects observable when attacking simple codes are detailed.

A. *Hardware setup*

`TRAITOR` is a multi clock-glitch platform implemented on an Artix-7 FPGA (Figure 5). The output signal replaces the clock source of the targeted microcontroller [3]. Also, as PLL

is a countermeasure to clock glitches, it has to be deactivated. In this example and for the rest of this article, the device under test (DUT) is a STM32 Discovery board with a STM32F100RB Cortex-M3 processor. The clock is provided by a 8 MHz quartz crystal corresponding to a 125 ns clock period.

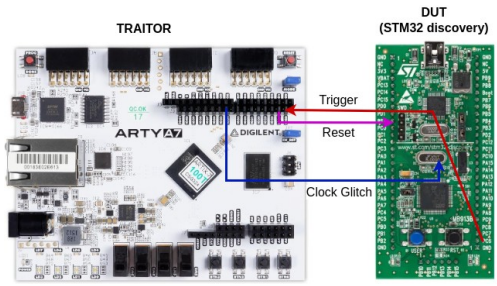


Figure 5: TRAITOR platform and connection to the DUT.

The glitch of `TRAITOR` is synchronous and can be described as an interrupted rising edge. Each glitch is defined by its amplitude, which is the value image of the physical voltage of the glitch (from 0 V to Vcc), and the delay when the glitch is injected (Figure 6). This glitch results of a combination of logic operations on two delayed clock signals.

In the context of multi-glitch attacks, the number of successive glitches is an additional parameter to consider. In that case, the attacker programs `TRAITOR` with the amplitude (identical for all the glitches of the attack), the delay and the size of each group of successive glitches. The most commonly observed effect when attacking with `TRAITOR` [3] is the skip of all the instructions contained in one word and the repeat of all the instructions contained in the previous word. As a matter of concision we will talk about skipped word and repeated word, respectively. Consequently, the following observations have been made regarding words and not instructions alone.

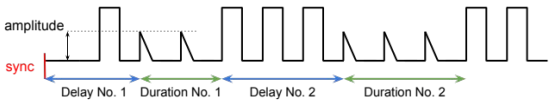


Figure 6: TRAITOR output signal characterized by the glitch amplitude, the delay of injection for each glitch burst and its duration.

B. Fault effect observed

First, the various fault effects obtained by injecting faults with `TRAITOR` have to be analyzed. A fault campaign is launched on two simple codes compiled into the Thumb-2 instruction set only containing arithmetic instructions: one code with only 16 bits instructions (Listing 1) and the other one with only 32 bits instructions (Listing 2). The effects of the amplitude over the fault effect is investigated for each delay between the first and the last faulted instruction. For these two codes, we chose aligned instructions to avoid any hybridization of instructions as observed by Alshaer *et al.* [17]. Indeed, this effect makes the identification of fault models more complicated and has no interest in our study. We also chose non-idempotent instructions to make possible the observation of potential repeated instructions when faulting. For this experiment, faults are injected at every clock cycle such as each word is faulted. The range of glitch amplitude values is from 420 to 530.

```
...
nops
nops
adds r0, r0, #11
subs r3, r3, #4
adds r1, r1, #13
subs r4, r4, #5
...
nops
nops
...
```

Listing 2: Code with 16 bits instructions.

```
...
nopw
addw r0, r0, #11
subw r3, r3, #4
addw r1, r1, #13
subw r4, r4, #5
...
nopw
...
```

Listing 1: Code with 32 bits instructions.

The results of the injection campaign with `TRAITOR` are plotted in Figure 7. The different types of fault are represented by different colors, with respect to the delay and amplitude values. First, we observe that regardless of the instruction size, the fault effect is most of the time a skip with some repeat side effects. The *n* value is related to the index of the skipped word.

In the case of 16 bits instructions (Figure 7b), the only observable fault model is the $skip_n$ /$repeat_{n-1}$ (red). As the words contain two instructions of 16 bits, each one executed in 1 cycle, this fault can only occur every 2 cycles. This type of fault is observable for amplitude values from 440 to 500.

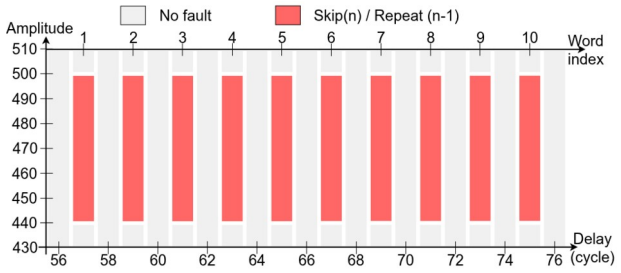
In the case of 32 bits instructions (Figure 7a), fault can be observed at every cycle. It makes sense as each word contains only 1 instruction of 32 bits that is executed in 1 cycle. The $skip_n$/$repeat_{n-1}$ (red) is also observable every cycle but in a smaller amplitude range of 440 to 455. Every 2 cycles, $skip_n$/$repeat_{n-2}$ (blue) appears for amplitudes of 455 to 465. A $skip_n$/$repeat_{n-4}$ type of fault is also observable every 4 cycles for higher amplitudes. Other unidentified effects are detected for higher amplitudes as well.

Among the various fault effects, the only one that is observable regardless the instruction size is the $skip_n$ /$repeat_{n-1}$. For the application of `GARLIC` method, it is necessary to write a test code containing words with variable execution duration. We made the choice to only characterize glitches on addition and subtraction. Consequently, the test code must contain a mix of instructions with different sizes to obtain words with variable execution duration. For that reason, we chose to study the $skip_n$ /$repeat_{n-1}$ fault. In the remainder of this article, we will refer to that type of fault as *skip/repeat*.

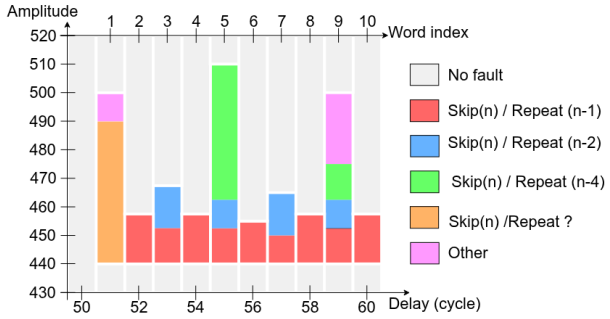
C. Hypothesis on the fault mechanism

Before going further, it is necessary to make an assumption about the physical location of the fault induced by the clock glitch on the selected target. First, the skip/repeat appears on a whole word and not on separate instructions. Thus, the possibility of faulting the decode and execution stages can be excluded. Two transfer steps could be disrupted: the transfer

from the flash memory to the prefetch buffer and the transfer from the the prefetch buffer to the fetch stage. The location of the flash memory on our microcontroller is easily identifiable. Consequently, the fetch of instructions from the flash memory to the prefetch buffer can easily be faulted by laser injection. For these reasons, we chose to investigate on the hypothesis that `TRAITOR` glitch disrupts the transfer from the flash memory to the prefetch buffer.



a. Faults on 16 bits arithmetic instructions



b. Faults on 32 bits arithmetic instructions

Figure 7: Type of faults for 16 and 32 bits arithmetic instructions with n the word index.

V. Laser Fault Injection

A. Laser bench and target

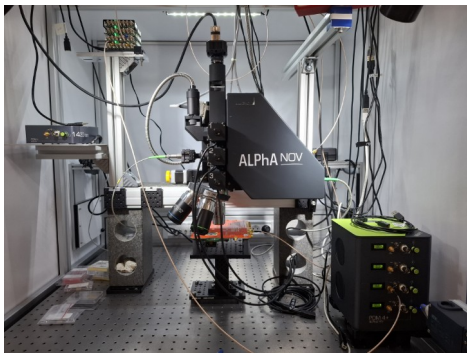


Figure 8: Laser fault injection bench.

The laser bench used in our experiments (Figure 8) has a 1064 nm wavelength source that is used with a 5x objective. This optical part is mounted on a three-axis motorized turntable. The maximum power output is 5 W. The injection delay and the pulse width can be adjusted with a precision of 0.1 ns.

The targeted microcontroller is the same as that used in the clock-glitch attack (STM32F100RB Cortex-M3). The processor is interfaced on a Donjon Scaffold board. The clock source is an 8 MHz quartz crystal which corresponds to a 125 ns clock period. Figure 9 shows an infrared image of the target after mechanical decapsulation. The main parts of the microcontroller have been identified: the analog part (in blue) with the power and clock management; the flash memory (red) where the instructions of the targeted code are stored; the logic part (yellow) where the instructions are launched from flash memory, treated, and executed; the RAM memory (green) where data are temporarily stored during the code execution.

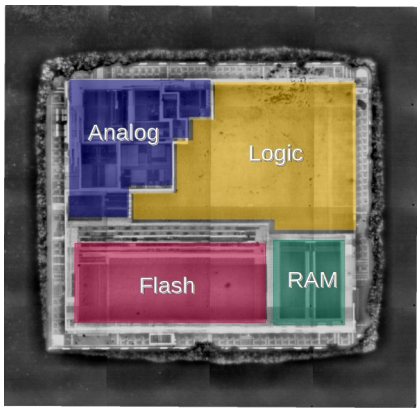


Figure 9: Infrared picture of the STM32F100 microcontroller and identification of areas

B. Parameters calibration

For the experiment, faults are injected into the flash memory of the microcontroller. Colombier *et al.* showed that this type of experiment resulted in a bit-set over the instruction opcode [19]. Since the objective of the study is a timing analysis, we have to find the best parameters to obtain temporally localized faults with a good success rate. The parameters have to be set such that one pulse can only fault one word.

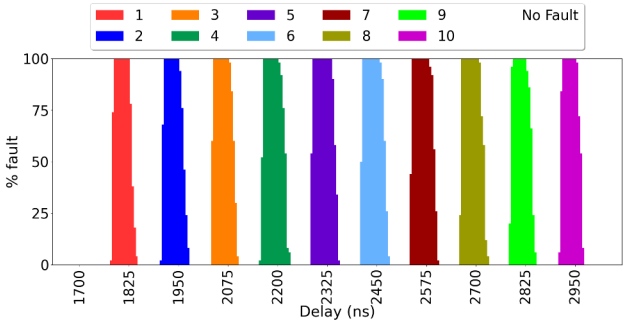


Figure 10: Percentage of faults with respect to the injection delay with power of 100 mW and pulse width of 285 ns. The colors index is related to the faulted instruction.

These conditions are met with a pulse width of 285 ns and a power of 100 mW. We tested these parameters on the piece of code used in TRAITOR characterization (Listing 2) constituted of 32 bits ADDW and SUBW instructions. The laser injection is performed on the 24th bit of the word. The fault is a bit-set

on the immediate number that consequently adds or subtracts 128 (0x80) to the register involved in the addition or subtraction, respectively. The percentage of faults with respect to the injection delay is plotted in Figure 10. We observe that the delay range for which a fault is possible over a word is around 70 ns which is less than the clock period (only 0.56 period). Therefore, the timing precision is sufficient to discriminate the faults injected on two distinct clock rising edges.

VI. Application

The GARLIC methodology is applied to determine the location of the disrupted stage when injecting fault with TRAITOR and observing a skip/repeat. This section details the parameters used for the experiment.

A. Test code

For the application of GARLIC, a sequence of instruction words with alternating execution time and with no periodicity is necessary. The instructions have also to be selected so that faults by glitch injection and laser injection are observable. Considering the arithmetic instructions used for the characterization step, we chose to write the test code with a mix of 32 bits and 16 bits addition and subtraction instructions to obtain words executed in 1 and 2 cycles, respectively. These non-idempotent instructions make the skip/repeat fault effect observable

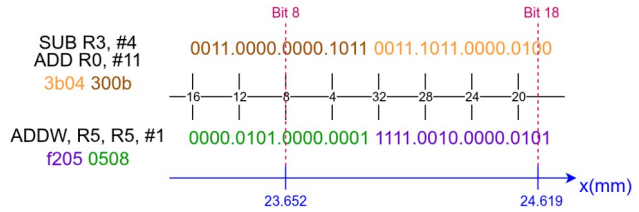


Figure 11: Localization of the targeted bits and instruction organization in the flash memory.

Concerning the laser fault injection, three constraints must be considered. First, the same bit has to be targeted all along the experiment. This means that initially, this bit has to be at '0' on every word. Second, since some of the words contain two instructions, two experiments on two different bits have to be performed: one in the 16 low-bits and the other one in the 16 high-bits. Finally, in order to make possible the identification of the fault, the fault injection must not result in a crash. The instructions are stored into the flash memory such as depicted in Figure 11. Consequently, the test code has been written so that bits 8 and 18 are set to '0' for each word. Faulting the 8th bit adds 128 to the immediate value of the instructions. Faulting the 18th bit adds 2 to the immediate value for the 16 bits instructions or add 2 on the register index Rn for the 32 bits instructions. The test code used for the experiment is presented in Listing 3 with the corresponding execution duration for each word. The instructions contained in the same word have the same color.

VII. Results

A fault injection attack campaign has been conducted on the test code using the TRAITOR platform and the laser fault injection platform. The results are presented and compared below.

```
...                 ...
adds R0,R0,#9       2 cycles
subs R3,R3,#4
subs R2,R2,#13      2 cycles
adds R6, R6,#5
subw R5,R5,#1       1 cycle
addw R1,R1,#17      1 cycle
subw R0,R0,#19      1 cycle
adds R3,R3,#20      2 cycles
subs R2,R2,#24
subw R4,R4,#2       1 cycle
addw R0,R0,#31      1 cycle
adds R5,R5,#29      2 cycles
subs R6,R6,#41
subs R4,R4,#37      2 cycles
adds R3,R3,#51
adds R2,R2,#57      2 cycles
subs R1,R1,#61
addw R5,R5,#6       1 cycle
subs R6,R6,#44      2 cycles
adds R2,R2,#53
...                 ...
NOPS                2 cycles
```

Listing 3: Test code used for the GARLIC application with the execution duration for each word.

A. TRAITOR injection results

To perform the attack, we set the amplitude for TRAITOR glitches to 445 which makes possible the skip/repeat fault on every word of the test code regardless the instruction size. The results are plotted for the delays ranging from 61 to 74 cycles, respectively when the first and the last words are faulted. The attack was repeated 50 times with the same set of parameters giving the same results. In the following Figures, the color and index *n* of each word is related to the test code (Listing 3). The fault timing is defined as the delay when the word n is skipped.

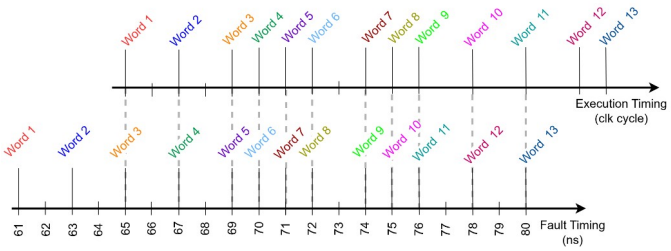


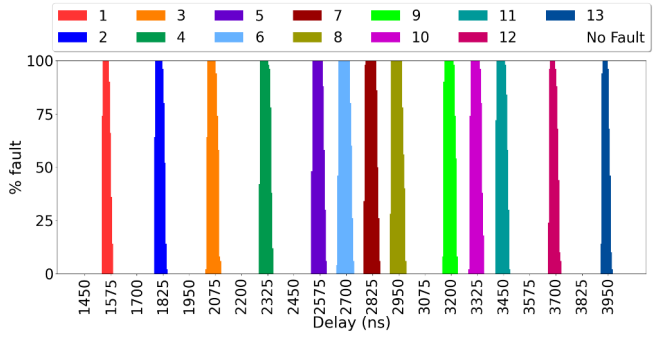
Figure 12: Execution timing and fault timing for each word when attacking the test code with TRAITOR.

In Figure 12 the execution timing is depicted at the top and the fault timing is represented at the bottom. First, we can observe the same pattern for both execution timing and fault timing. This means that for every fault we can identify the instruction word being executed. For example, after faulting the word 3 we need to wait 2 cycles to obtain a fault on the word 4. During that time a word is executed in 2 cycles. This

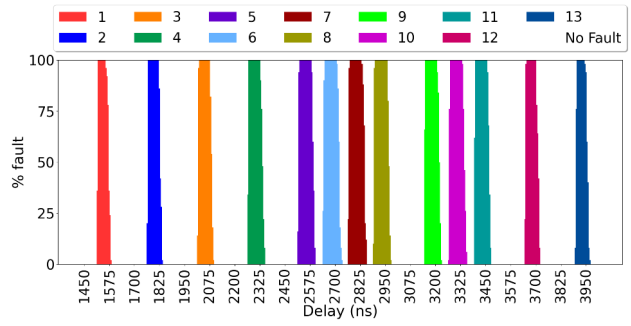
indicates that the fault occurs before the execution stage. By doing this comparison for every fault and as the execution timing pattern is non periodic, we can conclude that the fault on the word n occurs during the execution of the word (n-2). This supports the initial hypothesis that the skip/repeat is a disruption during the instruction transfer from the flash memory to the prefetch buffer.

B. Laser injection results

Two experiments have been performed using laser injection. We selected two bits to be faulted, one in the low part of the word (bit 18) and one in the high part of the word (bit 8) such that, with a word containing two instructions of 16 bits, faults on both instructions can be observed. The fault probability with respect to the delay is plotted in Figure 13a for the bit 8 and in Figure 13b for the bit 18. First, we can see that the fault timing is the same for both bits attacked. Second, we observe that the delay between each fault is approximately 125 ns or 250 ns which is equivalent to one or two times the clock period value. The alternation between 1 and 2 periods corresponds to the same pattern as the execution timing.



a. Faults on bit 8.



b. Faults on bit 18.

Figure 13: Percentage of faults obtain by attacking two different bits of each word at different delay.

The schematic of this pattern for the execution timing and for the laser fault timing is represented in Figure 14. We can see the presence of a delay between the execution pattern and the fault pattern. We can make the same observation that we previously made on the TRAITOR results: the fault appears on word n during the execution of the word (n-2).

C. Comparison and conclusions

When looking at the results of both fault injection campaigns with TRAITOR and with laser, we draw the same conclusion. In both cases we retrieve the same fault pattern that is shifted by 2 words. This means that the fault on the word n occurs during the execution of the word (n-2). Given this result, we can conclude that the fault occurs on the same stage while making fault injection with TRAITOR and with laser. As the laser injection is performed on the flash memory thus it disrupts the transfer from the flash memory to the prefetch buffer. Therefore, the analysis supports the initial assumption that the TRAITOR skip/repeat is due to the disruption of the transfer from the flash memory to the prefetch buffer.

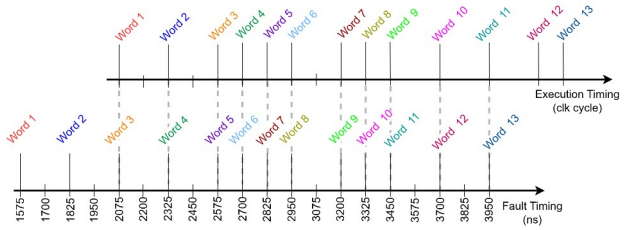


Figure 14: Execution timing and fault timing for laser attacking the test code.

D. Discussion

GARLIC is presented here as a proof of feasibility on a simple microcontroller for a well-defined fault model. The laser injection has been performed on the flash memory which is easy do identify on the microcontroller. To expand the method to other fault effects it is necessary to have the possibility of making fault injection on other parts (RAM memory, logic, registers). The RAM memory is also easily identified and considering the state of the art, it is now possible to obtain fault on registers using laser [20]. As RAM and registers are used in the execute stage, we could apply GARLIC on glitch effect that could happens during the execution.

Cortex-M microprocessors are commonly used to study fault injection because of their simple architecture with no countermeasures against physical attack [17] [19]. It makes possible the characterization of fault effects and the elaboration of complex attacks. Some works have been made on SoC showing that the presence of cache memory makes the study much more complicated [21], [22]. We could expect different effects of TRAITOR glitch in this case. As the cache memory is not fetch periodically the timing analysis could be complicated. Nevertheless we still could make two separate studies with and without cache activated and distinguish different fault effects [4].

VIII. Conclusion and Perspectives

The characterization of the effects observed when performing fault injection is made at ISA level. In particular, accessing to the mechanism of fault propagation is often challenging and leads to uncertain descriptions. Glitch fault injection is a non-located injection technique. In this case, the investigation over the fault propagation from the physical level to the software level is even more complicated. The GARLIC

method presented in this article aims to overcome this problem. This method is based on a comparison between glitch and laser fault timing. This method has been applied on `TRAITOR` clock glitch that induces skip/repeat. The `GARLIC` application on the skip/repeat effect obtained with `TRAITOR` supports the hypothesis that it is due to a disruption during the transfer of instructions from the flash memory to the prefetch buffer.

In further work, the method could be applied on other fault effects such as the ones observed during the `TRAITOR` characterization (section IV). `GARLIC` could also be applied to fault effects expected to occur during the decode or the execute stage considering the possibility of making fault injection on RAM memory and registers.

ACKNOWLEDGMENT

The authors thank the CYBER-ELEC platform with IETR (CNRS 6164) for laser fault injection facilities.

REFERENCES

[1] N. Beringuier-Boher, K. Gomina, D. Hély, J. Rigaud, V. Beroulle, A. Tria, J. Damiens, P. Gendrier, and P. Candelier, "Voltage glitch attacks on mixed-signal systems," in 17th Euromicro Conference on Digital System Design, DSD 2014, Verona, Italy, August 27-29, 2014. IEEE Computer Society, 2014, pp. 379–386.

[2] B. Yuce, N. F. Ghalaty, H. Santapuri, C. Deshpande, C. Patrick, and P. Schaumont, "Software fault resistance is futile: Effective single-glitch attacks," in 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016. IEEE Computer Society, 2016, pp. 47–58.

[3] L. Claudepierre, P.-Y. Péneau, D. Hardy, and E. Rohou, "TRAITOR: A low-cost evaluation platform for multifault injection," in ASSS '21: Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems, Virtual Event, Hong Kong, 7 June, 2021,

[4] L. Rivière, Z. Najm, P. Rauzy, J.-L. Danger, J. Bringer, and L. Sauvage, "High precision fault injections on the instruction cache of ARMv7-M architectures," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015. IEEE Computer Society, 2015, pp. 62–67.

[5] 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, WESS 2010, Scottsdale, AZ, USA, October 24, 2010. ACM, 2010, p. 7.

[6] M. Dumont, P. Moëllic, R. A. C. Viera, J. Dutertre, and R. Bernhard, "An overview of laser injection against embedded neural network models," in 7th IEEE World Forum on Internet of Things, WF-IoT 2021, New Orleans, LA, USA, June 14 - July 31, 2021. IEEE, 2021, pp. 616–621.

[7] M. Dumont, M. Lisart, and P. Maurine, "Modeling and simulating electromagnetic fault injection," IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., vol. 40, no. 4, pp. 680–693, 2021.

[8] I. Alshaer, G. Burghoorn, B. Colombier, C. Deleuze, V. Beroulle, and P. Maistri, "Cross-layer analysis of clock glitch fault injection while fetching variable-length instructions," J. Cryptogr. Eng., vol. 14, no. 2, pp. 325–342, 2024.

[9] J. Laurent, C. Deleuze, F. Pebay-Peyroula, and V. Beroulle, "Bridging the gap between RTL and software fault injection," ACM J. Emerg. Technol. Comput. Syst., vol. 17, no. 3, pp. 38:1–38:24, 2021.

[10] M. S. Kelly, K. Mayes, and J. F. Walker, "Characterising a CPU fault attack model via run-time data analysis," in 2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2017, McLean, VA, USA, May 1-5, 2017. IEEE Computer Society, 2017, pp. 79–84.

[11] Jonah Alle Monne, Guillaume Bouffard, "Synthesis of rtl-based characterization programs for fault injection," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2026, Washington, USA, May 4-7, 2026, 2026.

[12] T. Trouchkine, G. Bouffard, and J. Clédière, "Fault injection characterization on modern cpus," in Information Security Theory and Practice - 13th IFIP WG 11.2 International Conference, WISTP 2019, Paris, France, December 11-12, 2019, Proceedings, ser. Lecture Notes in Computer Science, M. Laurent and T. Giannetsos, Eds., vol. 12024. Springer, 2019, pp. 123–138.

[13] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller," in 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013, W. Fischer and J.-M. Schmidt, Eds. IEEE Computer Society, 2013, pp. 77–88.

[14] A. Gicquel, L. Claudepierre, D. Hardy, K. Heydemann, and E. Rohou, "Chapati: End-to-end methodology to conduct multi-faults injection campaigns," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2026, Washington, USA, May 4-7, 2026.

[15] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers, ser. Lecture Notes in Computer Science, E. Prouff, Ed., vol. 8622. Springer, 2014, pp. 243–260.

[16] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus," in 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, DTC 2011, Tokyo, Japan, September 29, 2011, L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, Eds. IEEE Computer Society, 2011, pp. 105–114.

[17] I. Alshaer, B. Colombier, C. Deleuze, V. Beroulle, and P. Maistri, "Variable-length instruction set: Feature or bug?" in 25th Euromicro Conference on Digital System Design, DSD 2022, Maspalomas, Spain, August 31 - Sept. 2, 2022. IEEE, 2022, pp. 464–471.

[18] A. Marotta, R. Lashermes, G. Bouffard, O. Sentieys, and R. Dafali, "Characterizing and modeling synchronous clock-glitch fault injection," in Constructive Side-Channel Analysis and Secure Design - 15th International Workshop, COSADE 2024, Gardanne, France, April 9-10, 2024, Proceedings, ser. Lecture Notes in Computer Science, R. Wacquez and N. Homma, Eds., vol. 14595. Springer, 2024, pp. 3–21.

[19] B. Colombier, A. Menu, J.-M. Dutertre, P.-A. Moëllic, J.-B. Rigaud, and J.-L. Danger, "Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019. IEEE, 2019, pp.1–10.

[20] H. Perrin, J. Dutertre, and J. Rigaud, "Betrayed by light: How photon emission microscopy empowers register bit-level laser attacks on microcontrollers," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2025, San Jose, CA, USA, May 5-8, 2025, pp. 35–45.

[21] T. Trouchkine, S. K. Bukasa, M. Escouteloup, R. Lashermes, and G. Bouffard, "Electromagnetic fault injection against a system-on-chip, toward new micro-architectural fault models," CoRR, vol. Abs/1910.11566, 2021.

[22] S. Casavecchia, D. Aboulkassimi, J. Clédière, J.-M. Dutertre, and S. Pontié, "Exploring laser fault injection in system-on-chips: A new approach for cpu and cache attacks," in IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2026, Washington, USA, May 4-7, 2026.