Modeling Clock Glitch Fault Injection Effects on a RISC-V Microcontroller

Ihab Alshaer, Ahmed Al-Kaf, Valentin Egloff, Vincent Beroulle

Modeling Clock Glitch Fault Injection Effects on a RISC-V Microcontroller

Ihab Alshaer*, Ahmed Al-kaf*,Valentin Egloff*, Vincent Beroulle*
*Univ. Grenoble Alpes, Grenoble INP[1], LCIS, 26000 Valence, France

***Abstract*—Embedded systems face security concerns, vulnerable to physical attacks like fault injection. RISC-V processors are increasingly favored for their open-source architecture. In this article, we present practical fault models operating at the instruction encoding level, which effectively elucidate numerous observed faulty behaviors arising from clock glitch campaigns conducted on a 32-bit microcontroller (MCU) embedding a RISC-V core. We demonstrate that, owing to the variable-length encoding of instructions, the impact of these models at the execution level varies. Nevertheless, the proposed models consistently maintain their applicability irrespective of the encoding length. Furthermore, we illustrate that some of the observed faulty behaviors are comparable to those obtained when targeting Arm Cortex-M-based MCUs. In addition, we present new models that can explain new faulty behaviors. The presented models are able to explain more than 90 % of the observed faulty behaviors.**

***Index Terms*—fault injection attack, RISC-V, fault model**

I. Introduction

Recently, the adoption of RISC-V architectures for embedded systems has captured the attention of numerous entities within the design community. This is due to its openness, efficiency, simplicity, extensibility, and stability [1]–[3]. Yet, it is crucial to consider security, especially given the advancements in attack techniques and equipment. Fault injection is a major threat to embedded systems among physical attacks.

Understanding fault effects is essential to protect embedded systems. Fault models abstractly represent these effects across system abstraction levels, aiding hardware designers and software developers in identifying vulnerabilities. Accurate models enable the development of effective countermeasures, ensuring a balanced cost-performance ratio. Conversely, inaccurate models may lead to excessive or inadequate protections.

Several studies [4]–[9] focused on analyzing the fault injection effects at the instruction set architecture (ISA) level. As a result, they proposed various fault models including: instruction skip [7], [9], [10], instruction replay [7], [9], instruction corruption [4], [6], [8], and register corruption [5], [6], [8]. These models are rather generic and fail to accurately depict the true impact of fault injection. The term "corruption" lacks specificity in conveying the fault's effect, making it challenging to pinpoint vulnerabilities based solely on this information. Consequently, this leads to developing either inappropriate protections. Regarding works focused on RISC-V, [11] performed EM campaigns on a RISC-V 32-bit MCU. However, similarly to aforementioned works, they described the fault effect as instruction(s) corruption or skip.

[1]Institute of Engineering Univ. Grenoble Alpes

In [12], [13], authors analyzed clock and voltage glitch effects on 32-bit MCUs embedding Arm Cortex-M3 and -M4 cores. They showed how memory alignment influences faults. As a result, they introduced two fault models: *Skip* and *Skip and repeat* a specific bit of binary encoding. This number of bits relate to flash memory access size. These models allowed explaining many observed faults. However, applicability on MCUs with cores other than Cortex-M remains uncertain.

In this article, clock glitch campaigns have been conducted on a RISC-V MCU. As a result, we confirm that *Skip* and *Skip and repeat* fault models are also applicable to the obtained faulty behaviors from these campaigns. Moreover, we show that other faulty behaviors may occur and cannot be explained using these two models. Hence, additional models are proposed, at encoding level, to describe these behaviors. These models are *Replace 16 bits*, and *Combination*. To validate the inferred models, we utilized an approach akin to that of previous works in [14]–[16], in which, the results of physical injections and simulations are compared to confirm an inferred model.

II. Experimental setup

Two clock glitch campaigns have been conducted, targeting the programs in Listings 1 and 2. Both listings show the target programs instructions along with their encoding in hexadecimal format. Both programs are the same, except in the second Listing, a 16-bit `NOP` instruction has been added to make the code misaligned. All other instructions are 32-bit instructions. These instructions are selected, as examples, to simplify the analysis and allowing the detectability of any faults that may occur.

```
1 ADDI x28, x28, 0x3b // 0x03be0e13
2 ADDI x29, x29, 0xa  // 0x00ae8e93
3 ADDI x7, x7, 0x27   // 0x02738393
4 ADD x6, x28, x31    // 0x01fe0333
5 ADDI x6, x6, 0x6    // 0x00630313
6 ADDI x31, x31, 0xd  // 0x00df8f93
```

Listing 1. Target program with its encoding in hex. format

The target device is a SiFive 32-bit MCU that embeds an E31 RISC-V core [17]. E31 core is based on RISC-V architecture and supports the RV32IMAC instruction set. Supporting the Compressed (C) extension makes RV32I a variable-length instruction set that offers two encoding lengths: 16 and 32 bits. The flash memory access size is 32 bits, allowing for the retrieval of different configurations of 32 bits as elaborated in [12].

```
1 0x0e130001
2 0x8e9303be
3 0x839300ae
4 0x03330273
5 0x031301fe
6 0x8f930063
7 0x000100df
```

Listing 2. Misaligned code target program in hex. format, where each line is composed of 32 bits.

In this work, ChipWhisperer [18] environment has been employed to perform the clock glitch. In this setup, three parameters need to be configured: Width, Shift, and Delay. Two delay values were used, while 12 was used for Width, and −13 was used for Shift. For each combination of Shift, Width, and Delay, the experiments are repeated 10 000 times. Thus, the number of executions for each campaign is 20 000. The reasons for choosing such parameters are two-fold: firstly, maximizing the quantity and the diversity of the observed faulty behaviors. On the other hand, this is aiming at faulting different locations within the target program.

III. Experimental Results and Analysis

The outcome of an injection in an execution leads to either Silent, Crash, or a Faulty output. Table I presents the obtained percentages for each class for both campaigns.

TABLE I
PERCENTAGE OF SILENT, CRASH, AND FAULT OVER THE TWO CAMPAIGNS.

Class \ Campaign on	**aligned code**	**misaligned code**
Silent	99.42 %	99.345 %
Crash	0.005 %	0.005 %
Fault	0.575 %	0.65 %

Referring to the encoding depicted in Listing 1 and Listing 2, the observed faulty behaviors are explained by the following inferred fault models:

- *Skip 32 bits*: the 32 bits at line `i` are skipped and the execution resumes from line `i+1`. For more details, we refer the reader to [12].
- *Skip and repeat 32 bits*: the 32 bits at line `i+1` are skipped and the 32 bits at line `i` are repeated. Listing 3 shows an example for this model.

```
4 ...
5 MUL x6, x6, x7      // 0x02730333
6 ADD x6, x6, x6      // 0x00630333
7 ...
```

Listing 3. Observed execution as a result of skipping line 5 and repeating line 4 in Listing 2.

- *Replace 16 bits*: the most significant or least significant 16 bits at line `i` are replaced with either the corresponding 16 bits at line `i−1`, or the corresponding 16 bits at line `i+1`. Listing 4 shows an example for this model.

```
1 ADDI x0, x0, 0x0    // 0x0001
2 ADDI x28, x2, 0x0   // 0x00010e13
3 ...
```

Listing 4. Observed execution as a result of replacing the least half at line 2 with the corresponding half at line 1 in Listing 2.

- *Combination*: the observed faulty behavior is modeled by a combination of a single fault model or a combination of two different models from the above.
- *Other*: the observed faulty behavior cannot be modeled by the aforementioned fault models.

Table II shows the percentage of the observed faulty behaviors concerning each fault model over all obtained faults.

TABLE II
PERCENTAGE OF THE CLASSIFICATION OF THE OBSERVED FAULTY BEHAVIORS UNDER THE INFERRED FAULT MODELS FOR EACH CAMPAIGN

Fault model \ Campaign on	**aligned code**	**misaligned code**
Skip 32 bits	0.87 %	1.54 %
Skip and repeat 32 bits	82.60 %	53.08 %
Replace 16 bits	6.96 %	40.76 %
Combination	0.87 %	3.08 %
Other	8.70 %	1.54 %

The aforementioned experimental findings illustrate that the fault models introduced at the encoding level, namely *Skip* and *Skip and repeat*, as proposed in [12], are not only applicable to faulty behaviors observed in Cortex-M-based MCUs, but also extend to RISC-V MCU. However, it became necessary to propose additional models, specifically *Replace 16 bits* and *Combination*, to account for previously unexplained behaviors. It has been demonstrated that these models effectively explain more than **91 %** of the observed faulty behaviors, when the code is aligned, and around **98 %**, when the code is misaligned, as shown in Table II.

The presented faults have been confirmed to have corrupted the binary encoding of instructions, leading to the execution of entirely different instructions in certain scenarios. For example, executing `MUL` instead of `ADD`. This provides clear evidence that many of these faults manifest at the microarchitectural level prior to the Decode stage in the pipeline. Understanding this could aid in developing appropriate hardware-level countermeasures, as it restricts the fault analysis at lower levels of abstraction. Furthermore, this paves the way for exploring software-level countermeasures, especially at the compiler level. Finally, clock glitch was used in this study for fault injection, but the findings can apply broadly to other techniques that exploit timing violations, such as voltage glitch and EM fault injection.

IV. Conclusion and Future Work

In conclusion, more than **90 %** of the observed faulty behaviors were explained by the inferred fault models at encoding level. It has been shown that already proposed fault models, applied to Cortex-M MCUs, are also applicable to a RISC-V MCU. This is important, as it limits the efforts in designing device-independent protections against such faults. Nevertheless, other faulty behaviors required proposing new fault models, namely, *Replace 16 bits*, and *Combination*.

In terms of future works, an important perspective would be to investigate the observed faulty behaviors at hardware level using RISC-V core description, to make sure that their origin comes from the Fetch stage. Finally, an important future work is thinking of cost-effective countermeasures against the presented faults at hardware and/or software levels.

ACKNOWLEDGMENT

This work has been supported by ARSENE project (PEPR PP7 ARSENE — ANR-22-PECY-0004) and by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

REFERENCES

[1] S. Sharma, "RISC-V Architecture: A Comprehensive Guide to the Open-Source ISA," [Accessed: February 28, 2024]. [Online]. Available: https://www.wevolver.com/article/risc-v-architecture-a-comprehensive-guide-to-the-open-source-isa

[2] ——, "Understanding RISC-V: The Open Standard Instruction Set Architecture," [Accessed: February 28, 2024]. [Online]. Available: https://www.wevolver.com/article/understanding-risc-v-the-open-standard-instruction-set-architecture

[3] D. Banatao, "Driving the Future of Chip Innovation: Top Three Reasons to Adopt RISC-V," [Accessed: February 28, 2024]. [Online]. Available: https://www.computer.org/publications/tech-news/trends/reasons-to-adopt-risc-v

[4] 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*. IEEE Computer Society, 2013, pp. 77–88.

[5] O. Trabelsi, L. Sauvage, and J.-L. Danger, "Characterization of electromagnetic fault injection on a 32-bit microcontroller instruction buffer," in *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2020, pp. 1–6.

[6] N. Timmers, A. Spruyt, and M. Witteman, "Controlling pc on arm using fault injection," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2016, pp. 25–35.

[7] J. Proy, K. Heydemann, A. Berzati, F. Majéric, and A. Cohen, "A first ISA-level characterization of EM pulse effects on superscalar microarchitectures: A secure software perspective," in *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*. ACM, 2019, pp. 7:1–7:10.

[8] T. Trouchkine, G. Bouffard, and J. Clédière, "EM fault model characterization on socs: From different architectures to the same fault model," in *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE, 2021, pp. 31–38.

[9] V. Khuat, J.-L. Danger, and J.-M. Dutertre, "Laser fault injection in a 32-bit microcontroller: from the flash interface to the execution pipeline," in *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 2021, pp. 74–85.

[10] V. Werner, L. Maingault, and M. Potet, "An end-to-end approach for multi-fault attack vulnerability assessment," in *Workshop on Fault Detection and Tolerance in Cryptography*. Milan, Italy: IEEE, 2020, pp. 10–17.

[11] M. A. Elmohr, H. Liao, and C. H. Gebotys, "EM fault injection on ARM and RISC-V," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 206–212.

[12] 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*. Maspalomas, Spain: IEEE, Aug. 2022, pp. 464–471.

[13] I. Alshaer, "Cross-Layer Fault Analysis for Microprocessor Architectures (CLAM)," Theses, Université Grenoble Alpes [2020-....], Oct. 2023. [Online]. Available: https://theses.hal.science/tel-04417620

[14] L. Dureuil, M. Potet, P. de Choudens, C. Dumas, and J. Clédière, "From code review to fault injection attacks: Filling the gap using fault model inference," in *International Conference on Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, N. Homma and M. Medwed, Eds., vol. 9514. Bochum, Germany: Springer, 2015, pp. 107–124.

[15] I. Alshaer, B. Colombier, C. Deleuze, V. Beroulle, and P. Maistri, "Microarchitectural insights into unexplained behaviors under clock glitch fault injection," in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science. Springer Nature Switzerland, 2024, pp. 3–22.

[16] 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.

[17] SiFive, "SiFive E31 core complex manual v2p0," https://static.dev.sifive.com/SiFive-E31-Manual-v2p0.pdf, [Accessed: February 16, 2024].

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