In-depth Analysis of the Effects of Electromagnetic Fault Injection Attack on a 32-bit MCU

Jinteng Jiao[1], He Li[2], Yanzhao Feng[3], Chengdong Qian[3], and Qiang Liu[1]
[1]Tianjin Key Laboratory of Imaging and Sensing Microelectronic Technology, School of Microelectronics, Tianjin University
[2]Trinity College, University of Cambridge, UK
[3]Phytium Technology Co., Ltd.

2022 IEEE 35th International System-on-Chip Conference (SOCC) | 978-1-6654-5985-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/SOCC56010.2022.9908097

***Abstract*—As semiconductor technology developing, current microcontrollers (MCUs) contain processor, memory, bus and peripherals, and become a low-power system-on-chip platform supporting various IoT applications. Although providing powerful real-time process capability, MCUs are vulnerable to fault injection attacks (FIAs) which target at the valuable data such as cipher key and personal information. While many research studies concentrated on successful FIAs on MCUs, this paper focuses on the low-cost and high-efficiency FIA technique, electromagnetic fault injection (EMFI), and its effects on the very widespread MCU (STM32F103ZET6) with ARM 32-bit Cortex™-M3 kernel. Interestingly, we observe a number of faults which consistently occur, such as the USART interface errors and the computation errors. By characterizing and reasoning the faults, we identify the vulnerabilities of the MCU and the EMPI fault models. Finally, countermeasures against the EMPI attacks are recommended.**

***Index Terms*—Microcontroller, Electromagnetic Fault injection, Fault Analysis**

I. Introduction

Current microcontrollers (MCUs) contain processor, memory, bus and peripherals, and become a low-power system-on-chip platform supporting various Internet of Things (IoT) applications, such as household appliances, wearable medical equipment and automotive vehicles. Due to the pervasive deployment on resource-constrained edge devices, the MCUs face a high risk from physical hardware attacks, which aim at escaping permission checks, controlling the systems and stealing confidential data [1].

The physical hardware attacks on MCUs can be classified into two categories, side channel attack (SCA) and fault injection attack (FIA). In SCA, the physical information, such as timing information, power consumption and electromagnetic, is collected from MCUs and used to compromise the security and privacy [2]. Unlike SCA, FIA is an active attack means, which induces faults into MCUs by tempering with the operating conditions such as clock and voltage. Existing FIA techniques include clock glitch [3], voltage glitch [4], laser injection [5] and electromagnetic pulse injection (EMPI) [6]. Among them, the clock glitch and voltage glitch FIAs are low-cost, but need to touch the devices and have low spatial resolution. The laser injection and EMPI have high spatial resolution. However, the laser injection needs advanced and expensive equipment. Therefore, this work focuses on the low-cost and non-invasive EMPI attack on MCUs.

Corresponding Author: Qiang Liu Email: qiangliu@tju.edu.cn

There exist several studies about the EMPI attacks on MCUs. It is demonstrated that instruction skipping on RISC-V can be caused by EMFI and the effect of the attack is related to temperature [7]. The vulnerability of single-chip microcomputer under electromagnetic pulse is proved [8]. Although demonstrating that EMFI gives rise to security threat to MCUs, these works mainly present the fault phenomenon such as instruction skipping, and do not provide a detailed analysis about the faults and the reasons behind the faults. Without identifying the reasons, it is difficult to design efficient countermeasures against the EMPI attack.

In this work, an EMFI attack experiment is designed and a 32-bit MCU with the ARM core is regarded as the attack target. Repeatable fault information is analyzed in detail to investigate the attack effect and the attack mechanism on the MCU. The main work of this paper is as follows:

- Three types of repeatable faults are observed by performing the comprehensive EMFI attack on the MCU and analysing the results.
- The reasons of the three types of faults are revealed, and the corresponding EMPI fault models and the vulnerabilities of the MCU are identified.
- Based on the reasoning results, possible countermeasures are discussed and suggested.

The rest of this paper is organized as follows: In Section II, the related work to EMFI attacks is introduced. In Section III, the designed EMPI attack experiment and the attack results are presented. In Section IV, the faults are analyzed and the causes of the faults are discussed. In Section V, the possible countermeasures are presented. Finally, conclusion is presented in Section VI.

II. Related Work

EMFI attack against MCUs can be divided into three categories: instruction skipping, cracking algorithms and information leakage. Instruction skipping is an error that skips an instruction of a program at runtime. Most work evaluates single instruction skipping [9]. There are also skips for multiple instructions [10], but the success rate is low. To crack the algorithm, the attacker always first analyzes the algorithm cracking method to find the sensitive instruction. Then, The attack is performed at the sensitive instructions to try to crack

the algorithm [6]. Information leakage refers to the appearance of information related to the MCU in the output data [7], such as address and instruction information. In this work, information leakage is also observed.

In general, during the runtime of a program on a MCU, there are a number of opportunities to induce a fault. This means that it is difficult to determine which instruction is attacked during the execution of the program. Meanwhile, for MCUs whose architecture is unknown such as the ARM core, it is also hard to determine which component is under attack. These difficulties hinder the fault analysis and countermeasure development. In this paper, we analyse the causes of the faulty phenomena in depth, and thus determine where and when the attacks are effective.

III. EMPA Attack and Results

In this section, the experiment designed for the EMFI attack on the MCU is presented first. Then, the experimental results are classified and briefly analyzed.

A. Attack experiment design

The EMPI attack platform consists of an electromagnetic pulse (EMP) generators, a CNC mobile platform, a oscilloscope, and a PC. The CNC mobile platform is used to place the attack object and can move in X and Y directions with an accuracy of 50 um. This generator has an adjustable charging voltage range of -400 V to 400 V. The width of the EMP is adjustable from 10 to 200 ns. In this experiment, the pulse intensity is used the maximum charging voltage. The pulse width is designed at 20 ns. The frequency of EMFI attack is approximately once in 0.6 s. The diameter of EM probe is 0.7 cm.

The EMPI attack object is the stm32f103zet6 enhanced MCU, which is widely information control field. It is packaged in an LQFP144 package and has an ARM 32-bit Cortex™-M core. It is also equipped with 64K SRAM, USART and a reset system. The reset system contains power-on reset(POR), power-down reset(PDR) and a reset button. The chip will reset when the power voltage is smaller than the threshold voltage. Threshold voltage $V_{POR/PDR}$ is 1.8 V. The power supply V_{DD} and V_{SS} is ranged in -0.3 V to 4.0 V. The chip size of the MCU is 20 cm $\times$ 20 cm. The USART is configured in the chip. The output data will be delivered to a PC via the USART serial port.

Given the small attack area below the probe, we divide the chip surface into a grid. In order to make every position of the chip under attack, the chip is divided into 60 $\times$ 60 small squares. The probe moves over the chip surface in a step of 0.33 cm. Each square is indicated by coordinate (m, n), where $1 \leq m \leq 60$ and $1 \leq n \leq 60$.

```
printf("Start\r\n");
while(1){
   long long int a=0;
   for(int i=0;i<255;i++){
      for(int j=0;j<255;j++){
          a++;}}
printf("ss %ll",a); printf("\n");}
```

Listing 1. Code executed on the MCU

The MCU stores the program in SRAM and starts it at power on. The program to be attacked is designed in advance and downloaded into the MCU. Listing 1 shows the program. It starts with printing "Start" and follows with a while loop. The while loop will first assign 0 to variable a. Then, two nested for loops will be executed to accumulate a. The result will be printed at the end of the program.

The program is designed in this way due to the following reasons. Firstly, the $while(1)$ loop is used to ensure at least an instruction is attacked. Secondly, two nested for loops are used to determine whether there is attack causing the loop to jump out. This is because the outer loop jump has more significant effect on the computation result. Thirdly, to ensure that the faults caused by two adjacent attacks do not affect each other, the execution time of the nested for loops is shorter than the interval of the two adjacent attacks. As a result, the upper bounds of the two loops are set to 255, which makes the two loops finished in 0.02 seconds. Fourthly, ss is designed as the label of starting the output. As long as the host computer receives an output of s, the data logging system will start recording the output characters, which are the experimental results.

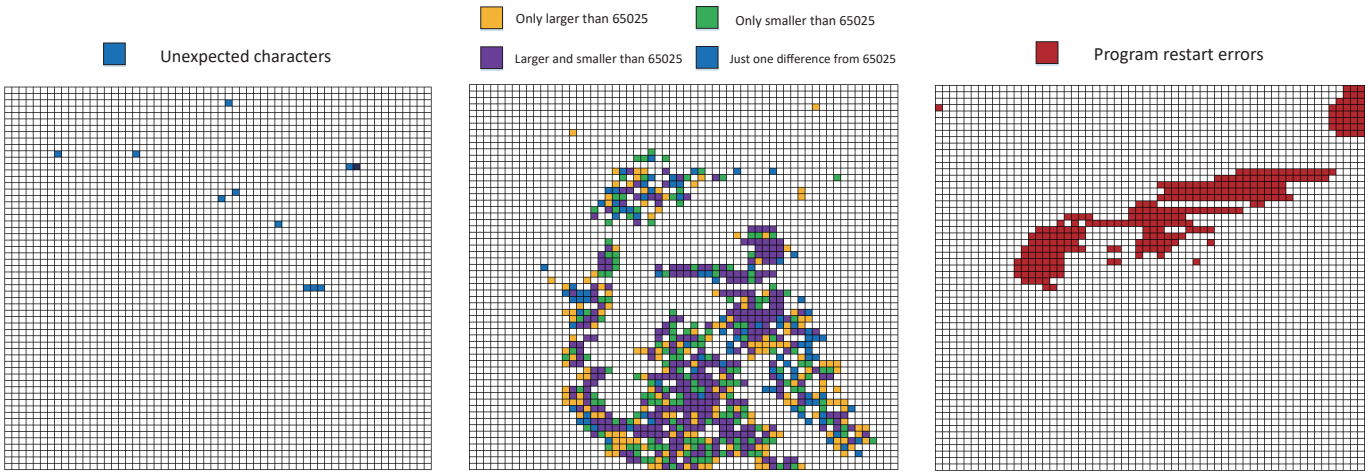
The attack procedure is the following. The MCU is connected to a host computer and the data logging system is turned on to record the data. The EMP probe starts from the (1,1) square for attack, and is placed close to the surface of the chip. At each square, the EMFI attack is performed 50 times. The probe moves according to the coordinates until it attacks the entire chip.

B. Attack result

The normal output of the program is "ss 65025". 65025 is the correct accumulation value of a. The output data that is not 65025 is considered as fault data. In total, the experiment performs 180000 times EMPI attacks. The number of faulty results is 3812. The faulty results can be divided into three types: output results with unexpected characters (0.29%), output results with numerical errors (51.00%), and program restart (48.51%). Next, we will present the three types of faulty results, respectively, in Fig. 1.

The blue points in Fig. 1 (a) indicate the positions, on which the EMPI attack leads to the results with unexpected character. For example, the faulty output of "6u025" at (11,8) shows that the character "u" is an unexpected character, which replaces "2" in the output. Table I shows the other unexpected characters. As shown in the figure, this type of fault is clearly fewer than the other two types.

Fig. 1 (b) shows distribution of positions on which the EMPI attacks induce the numerical errors. This type of faulty results is further classified in terms of the values of the results with respect to the correct value. The yellow points indicate that the attacks always lead to a larger than the correct value.



(a) Bitmap of unexpected character (b) Bitmap of numeric errors (c) Bitmap of program restart

Fig. 1. Bitmap for fault classification statistics

Green points indicate that the attacks always lead to a which is smaller than the correct value. Purple points are positions where the attacks could result in both larger and smaller values compared to the correct value. Blue points are special cases in which the resultant values are only one different from the correct value. This type of fault is the most among the three types.

There are a total of 1857 restart faults occurred. According to the designed program, "Start" will be printed in the first statement and will not be output again after entering the while loop. However, "Start" is exceptionally printed in the output result. The statistical results are shown in Fig. 1 (c). This indicates that the attack caused the program to restart.

The experimental results show clear patterns as illustrated in Fig. 1. These patterns demonstrate that (1) the EMPI attacks on different positions of the MCU chip surface have different effects and (2) the faults are repeatable and to some extent could be controllable. As a result, the EMPI attack shows severe threats to the MCU. In the next, we will analyze the three types of faults in detail and explore the reasons behind of the faults.

IV. Fault Analysis and Reasoning

In this section, the three types of faults are analyzed together with the faulty data statistics and the assembly code of the program executed on the MCU. The possible reasons for the occurrence of the faults are explored and determined.

A. Analysis of unexpected characters

Because this type of fault involves the output of characters, we pay attention on the printf function. The function prototype is:

int printf (const char *format, ...).

The execution of the printf function involves system calls and system interactions. Therefore, the corresponding assembly statements of this function are different in different devices. In the experiment platform, the USART interface is used to connect the MCU to PC. According to the transmission logic of the USART interface, the characters are passed in the form of the Unicode. Therefore, the printf function converts the binary data stored in a register into characters' Unicode with the specified format through a series of stack operations to complete the character output.

TABLE I
Unicode comparison of original and unexpected characters. The underlines highlight those bits which are flipped to generate the unexpected characters.

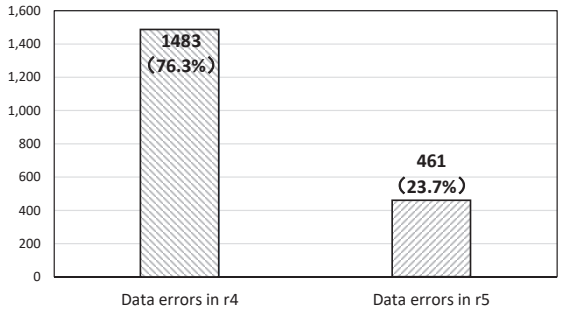
character		**Unicode**	
Original	***Unexpected***	***Original***	***Unexpected***
0	p	00110000	01110000
6	v	00110110	01110110
5	u	00110101	01110101
space char	$	00100000	00100100

Therefore, comparing the Unicode of the unexpected character and the original character can reveal the reason why the unexpected characters occur. Table I lists the Unicode of the original and unexpected characters in some experimental results. We use underlines to highlight the bits which are different. As shown in the table, the single-bit flip of the original characters leads to the unexpected characters. That is, the EMPI attack induces single-bit flip.

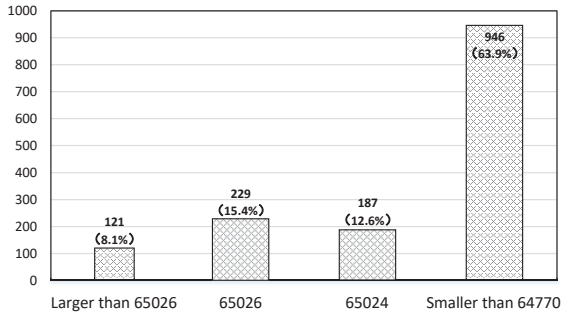
Now, we need to determine when and where the bit is flipped. The time span of the $while$ loop (from the beginning of variable a initialization to the final data output) execution can be divided into two stages in terms of the time when the Unicode is generated. Denote the time period before generation as T_a and after generation as T_b.

If an attack occurs in T_a, no matter what impact of the attack has on the chip, the error will eventually be reflected in the binary data. This erroneous binary data will be converted to decimal characters for output. In this case, decimal characters should be printed. However, in practice, unexpected letter characters occur. Therefore, the attack that causes unexpected characters output should happen in the time period T_b.

The operating components of the MCU in T_b are the stack storage unit, Advanced Peripheral Bus (APB), USART output



(a) The number of erroneous data in r4 and r5



(b) Four cases of erroneous data in r4

Fig. 2. Histogram of numerical errors caused by the attacks.

TABLE II
FAULTY VALUES OF a LEADING TO THE SYSTEM INFORMATION LEAKAGE.

Position	Faulty value	Machine Code
(44,30)	576476300085100033	0800 0E24 0000 FE01
(54,33)	2305850224758816257	2000 0690 0000 FE01

registers and output stream circuit. The latter three are all USART-related circuits. If the attack affects the stack memory cell, the distribution of such faults on the bitmap should be concentrated in a fixed range. However, as shown in Fig. 1 (a), the positions where the attacks induce this type of fault are scattering. Therefore, the cause of the faults with the unexpected characters is the EMPI attack on the USART-related circuits. That is, a bit of the original Unicode, which is processed in the USART-related circuits, is flipped by the EMPI attack.

B. Analysis of numerical errors

The variable a defined in the program is a 64-bit integer. It means two 32-bit registers are needed in the MCU to store this variable. Listing 2 shows the assembly code obtained by compiling the program in Listing 1. It can be seen that 1) a is represented by registers r5 (high 32-bit) and r4 (low 32-bit); 2) r4 is used to complete the accumulation operation and 3) r5 is always 0. However, in several faulty values of a, r5 not being 0 are found, as shown in Table II. The table lists two faulty values caused by the attacks on two positions. The 64-bit machine code shows the corresponding values of r5 and r4. "0000 FE01" is the correct result 65025 of r4 after the execution of the accumulation. "2000 0690" is the initial value of the Stack Pointer (SP) register of the MCU. "0800 0E24" is the address of the instruction "ADDS r1, r1, #1". As can be seen, the attack can cause the MCU information leakage. These information could be improperly exploited by attackers. According to Fig. 2 (a), the number of such errors is 461, accounting for 23.1%.

In addition to r5, data errors in r4 is the focus of the analysis. The value of r4 is directly related to the execution of the two for loops. Therefore, the assembly code of the for loops can be analyzed to explore the cause of the errors. The number of erroneous data in r4 is 1483 accounting for 76.3% of the total numerical errors. It can be further subdivided into 4 cases, as shown in Fig. 2 (b). The following will analyze the reasons of these four cases of data errors.

```
0x08000E10 2100 MOVS r1 , #0x00
0x08000E12 460C MOV  r4 , r1
0x08000E14 460D MOV  r5 , r1
0x08000E16 2000 MOVS r0 , #0x00
0x08000E18 E008 B    #0x08000E2C
0x08000E1A 2100 MOVS r1 , #0x00
0x08000E1C E003 B    #0x08000E26
0x08000E1E 1C64 ADDS r4, r4, #1
0x08000E20 F1450500 ADC r5, r5, 0x00
0x08000E24 1C49 ADDS r1, r1, #1
0x08000E26 29FF CMP  r1, #0xFF
0x08000E28 DBF9 BLT  #0x08000E1E
0x08000E2A 1C40 ADDS r0, r0, #1
0x08000E2C 28FF CMP  r0, #0xFF
0x08000E2E DBF4 BLT  #0x08000E1A
```

Listing 2. For the two-level loop in Listing 1

1) Exploration of reasons: The intuitive cause of the error to think of is the Program Counter (PC) register fault. The PC register will increase by a increment of the instruction length after the instruction finishes. If the PC increment is executed incorrectly, it may cause the instruction to be executed repeatedly or skip several instructions, leading to errors in the final value of a. In this MCU, the length of instructions is the even number, as shown in Listing 2. If the PC add an odd number, the MCU will fall in exception procedure. However, this phenomenon does not appear in our experiment. So, we think that the attack is less likely to lead a PC register fault.

In addition to the PC fault, there are also other possibilities which could lead to the four cases of data errors, by analyzing the assembly code. Specifically, the value smaller than 65024 is mainly related with the BLT instruction. Values of 65024 and 65026 are related with the ADDS and BLT instructions. Values larger than 65026 is mainly related with the MOV instruction. Next will analyse the four cases in detail.

a) Values smaller than 65024: In Listing 2, there are two BLT instructions corresponding to each of the for loops. It is the key instruction that controls the for loop. It jumps to the corresponding instruction when the result of the CMP

TABLE III
ASSEMBLY AND MACHINE CODE OF THE ADDS INSTRUCTIONS.

Assembly code	Machine code
ADDS r1, r1, #1	00011100<u>01</u>001001
ADDS r4, r4, #1	00011100<u>01</u>100100
ADDS r4, r4, #2	00011100<u>10</u>100100

instruction is negative. If a BLT instruction does not execute, the corresponding loop accidentally breaks, which causes a smaller value of the accumulated a. Besides, there is an exception that erroneous value 64770 could be also caused by the 0x08000E2A ADDS instruction fault. It makes the program skip the internal loop.

b) Values of 65024 and 65026: In the experiment results, the number of data with value "65026" is 229, accounting for 15.4% of the total number of erroneous data in r4. The number for "65024" is 187, accounting for 12.6%. The occurring possibility of both cases is basically the same. Also, It can be seen that both cases have a feature that the erroneous value is larger or less than the correct value by 1. It means that the accumulation of a is executed one more time or one less time. Looking at the code in Listing 2, there are two possibilities for this two erroneous cases.

The first possibility is the ADDS instruction fault. By looking into the ADDS instructions in Table III, it can be seen that bits 5 to 0 represent the register number, and the immediate number is stored in bits 7 to 6. If an attack causes bits 7 and 6 flipped, it will lead to an addition error. For example, there are two possible instruction faults lead to 65024. The first one is the 0x08000E1E instruction. If the immediate number changes from 1 to 0, it will make the accumulation to be performed one less time. The second one is the 0x08000E24 instruction. If the immediate number changes from 1 to 2, the internal loop is executed one time less. Because it is easier for the EMPI attack to flip one bit, the immediate number changing from 1 to 0 is more likely.

The second possibility is the BLT instruction fault. If the instruction 0x08000E28 does not jump during the penultimate execution of the internal loop, the last addition will not be executed. This situation can cause the output data to be smaller by one.

c) Values Larger than 65026: The reason of value larger than 65026 may be caused by the MOV instruction. If there is a fault in the execution of the 0x08000E12 MOV instruction, the initial value of r4 is not zero. The values larger than 65026 will appear, depending on the nonzero initial values. For example, during the experiment, there exist erroneous data of a whose initial value is equivalent to 0x01, 0x02, 0x04, etc. These initial values are just one bit different from 0x00. The EMPI attack could lead to these values by flipping one bit of 0x00. In addition, there is also an exception that the erroneous value 65280 can be also caused by the 0x08000E2A ADDS instruction fault, which makes the program repeat the internal loop.

2) Determination of reasons: Based on the above discussion, the faults in the BLT, ADDS and MOV instructions possibly lead to the erroneous data. In this section, the BLT and ADDS instructions are proved to be actually attacked by the EMPI.

If the attack makes the BLT instruction not jump, it has two effects on the final value of a. Let M be the value when the attack affects the internal for loop. Therefore, M is in the following range

$$64771 \leq M \leq 65024. \tag{1}$$

Let N be the value when the attack affects the external for loop and N should be

$$N = 255 * l \ (l = 1, 2, ..., 254). \tag{2}$$

By analysing the program, M is 255 times more likely to occur than N. Let the number of M and N be A and B, respectively. Define f_1 as (3) and it should be 255 theoretically.

$$f_1 = A/B = 255 \tag{3}$$

However, in the experiment results, A = 792, B = 4, and thus $f_1 = 198$. There is a deviation here. The reason for this deviation is that the erroneous values are not caused by the BLT instruction fault only, but also by the ADDS instructions, as analyzed above.

The similar situation also happens when we analyze the ADDS instructions. If the 0x08000E24 ADDS instruction has a fault, the output is 65024 or 65026. Let the number of occurring times of the two values be C and D, respectively. If the 0x08000E2A ADDS instruction has a fault, the output value is 64770 or 65280, whose occurring times are labelled by E and F, respectively. Then, the ratio of the number of times of the two instructions having faults is defined as f_2, which is also equal to 255 in theory.

$$f_2 = (C + D)/(E + F) = 255 \tag{4}$$

According to experiments, C = 187, D = 229, E = 3 and F = 0, and thus $f_2 = 138.7$. There is also a deviation. Therefore, it is needed to consider the BLT and ADDS instructions together.

In fact, A contains C and B contains E. Therefore, considering the two types of instruction fault, the ratio becomes:

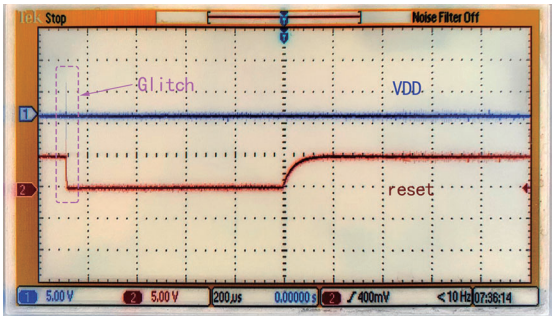
$$f_3 = (A + D)/(B + F) = 255 \tag{5}$$

Bring the experiment results, we obtain $f_3 = 255.3$, which is close to the 255. It proves that both BLT and ADDS faults actually occur in this experiment.

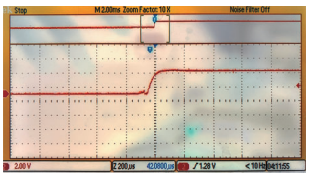
Because the output data in the experiment are not sufficient to complete the analysis of the MOV instruction fault, it cannot be proved that the EMPI attack actually affects the MOV instruction.

C. Analysis of program restart

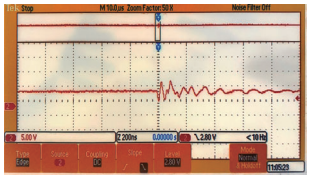
To analyse the reason of the program restart, the further experiment is done. The reset signal at the NRST pin and the power supply at the VDD pin of the MCU are monitored, using the oscilloscope, under the EMPI attack. The waveforms are shown in Fig. 3 (a), where the blue line is the VDD and the red line is the reset. We see a voltage glitch on the VDD and at the same time the reset signal falls. Then, after a short period, the reset signal rises. Note that the rising edge of the reset signal triggers reset of the MCU. To check the correctness of



(a) Waveform of VDD (blue) and reset (red) under attack



(b) Waveform of reset signal



(c) Waveform of VDD signal

Fig. 3. Waveforms of the VDD and reset signals.

the reset process, we also manually press the reset button and the monitored waveform is shown in Fig. 3 (b). The waveform shows the similar rising edge as the one in Fig. 3 (a). The VDD signal is monitored alone and the waveform is shown in Fig. 3 (c). It has a glitch at a nanosecond scale, which coincides with the width of EM pulse.

Based on the above observations, it can be concluded that EMFI can create a glitch at the VDD,which causes the falling and rising of the reset signal and resets the MCU. The related phenomenon is the program restart.

So far, we have analysed the three types of faulty results. Four EMPI attacks which lead to these faulty results are identified, including EMPI attack on the USART-related circuits, on the BLT and ADDS instructions and on the reset signal. Next, we will list some feasible countermeasures against these attacks.

V. Possible Countermeasures

According to the fault analysis and reasoning in the previous section, we recommend several countermeasures for the MCU design.

For the fault in USART related circuits, parity code-based method [11] can be used to detect one-bit flip caused by attacks.

For the fault in the BLT instruction, a shadow register [12] , which has the same function with xPSR, can be added behind the Program Status Register (xPSR). Before executing BLT instruction, MCU will compare the highest bit of these two registers. If they are not same, it proves an error happened.

For the faults affecting the ADDS instruction, *replay instruction* [13] may be a solution. This method protects instructions by copying one instruction into multiple. When the copied instructions are executed, they need to be compared with their own duplicated instructions. If the copied instructions are same with their own duplicated instructions, it proves the copied instructions correct.

For the glitch faults in the VDD net, it causes timing constraint violations or delay. Existing delay chain detector [14] is able to check for such timing errors.

VI. Conclusion

In this paper, a comprehensive EMFI attack on the MCU with the ARM architecture is designed and performed. Three types of repeatable faulty results are observed. By in-depth analysing the reasons behind the faulty results, four types of EMPI attacks and the related vulnerabilities of the MCU are identified. We recommend possible countermeasures against the EMPI attacks. It is believed that the vulnerabilities also exist in other MCUs. Therefore, this paper potentially provides a security evaluation platform and could help to improve the hardware security of MCUs in the IoT applications.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant 61974102 and the Tianjin University-Phytium IC Technology Joint Lab Program.

References

[1] C. Gaine, D. Aboulkassimi, S. Pontié, J.-P. Nikolovski, and J.-M. Dutertre, "Electromagnetic Fault Injection As a New Forensic Approach for SoCs," in *2020 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE, 2020.

[2] R. Wang, H. Wang, E. Dubrova, and M. Brisfors, "Advanced Far Field EM Side-Channel Attack on AES," in *Proceedings of the 7th ACM on Cyber-Physical System Security Workshop*, pp. 29–39, 2021.

[3] D. Zhou, P. Yang, and Q. Ou, "Analysis of Fault Characteristics Based on Clock Glitch Injection," in *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 5, pp. 785–790, 2021.

[4] C. Bozzato, R. Focardi, and F. Palmarini, "Shaping the Glitch: Optimizing Voltage Fault Injection Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 199–224, 2019.

[5] J. Breier, D. Jap, and C.-N. Chen, "Laser-based Fault Injection on Microcontrollers," in *Fault Tolerant Architectures for Cryptography and Hardware Security*, pp. 81–110, 2018.

[6] Liao, Haohao, "Electromagnetic Fault Injection on Two Microcontrollers: Methodology, Fault Model, Attack and Countermeasures," 2020.

[7] 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)*, pp. 206–212, IEEE, 2020.

[8] L. Keshun, Z. Xijun, and Z. Xing, "Research on Analysis and Classification of Vulnerability of Electromagnetic Pulse with a STM32 Single-Chip Microcomputer," *Scientific Programming*, vol. 2021, 2021.

[9] E. Trichina and R. Korkikyan, "Multi Fault Laser Attacks on Protected CRT-RSA," in *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 75–86, IEEE, 2010.

[10] 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.

[11] M. Medwed and S. Mangard, "Arithmetic Logic Units with High Error Detection Rates to Counteract Fault Attacks," in *2011 Design, Automation & Test in Europe*, pp. 1–6, 2011.

[12] C. Deshpande, B. Yuce, L. Nazhandali, and P. Schaumont, "Employing Dual-complementary Flip-flops to Detect EMFI Attacks," in *2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 109–114, 2017.

[13] N. A. Manssour, V. LapÔtre, G. Gogniat, and A. Tisserand, "Processor Extensions for Hardware Instruction Replay Against Fault Injection Attacks," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 26–31, 2022.

[14] M. Zhang and Q. Liu, "A Digital and Lightweight Delay-Based Detector Against Fault Injection Attacks," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.