EM Fault Injection on ARM and RISC-V

Mahmoud A. Elmohr, Haohao Liao, Catherine H. Gebotys
Department of Electrical and Computer Engineering
University of Waterloo
Ontario, Canada
Email: {mahmoud.elmohr, haohao.liao, cgebotys}@uwaterloo.ca

***Abstract*—Recently Electro-Magnetic Fault Injection (EMFI) techniques have been found to have significant implications on the security of embedded devices. Unfortunately, there is still a lack of understanding of EM faults and countermeasures for embedded processors. For the first time, this paper empirically shows that EMFI can cause skipping/faulting of more than one instruction on a 320MHz RISC-V processor, thus making it susceptible to a wider range of attacks. Additionally, empirical results on ARM Cortex M0 and RISC-V embedded processors show that EMFI is more susceptible at lower supply voltages and higher clock frequencies. Exception codes are also shown to be useful in understanding details of injected faults, providing further evidence that instructions have been corrupted in many cases. This research aims to enhance the understanding of faults, in order to better design countermeasures for embedded processors resistant to fault injection attacks.**

***Index Terms*—Fault Injection, Electromagnetic Pulse, RISC-V, ARM.**

I. Introduction

Although security is widespread in many embedded devices, the actual implementations must be resistant to physical attacks. For example, focusing electromagnetic waves or laser beams onto the device, or even applying voltage/clock glitches to the device, may cause a change in some computations or data within the device (referred to as a fault injection attack). Although research has examined fault injection attacks for several decades, there remains limited understanding of the types of faults on embedded processors and what factors affect the fault injection capabilities. In theory, if the results of a fault injection were to skip x instructions, then this could lead to a devastating attack. Consider a subset of code that calls an authentication algorithm followed by y instructions which jump to an error routine if authentication fails. If $y \ll x$, the fault injection does not need to be accurately timed, and the authentication and jump to error routine would be completely skipped leaving subsequent code to be executed with assumed authentication. This would support an attack that would completely skip authentication. Alternatively, cryptographic codes often load a constant to a register or perform additions representing the number of rounds or rounds remaining for the algorithm. If a fault injection were to change this value, it may lead to further attacks on the algorithm. Fault injection attacks have important consequences on the security of the embedded systems and thus detailed understanding of the threat is important in order to design appropriate countermeasures.

In this paper, we focus on Electro-Magnetic Fault Injection (EMFI) attacks since compared to other FI techniques it requires less access to the silicon die while having a moderate spatial and timing resolution [1].

Since the introduction of the EMFI concept, many studies on EMFI have been reported in the literature. A timing fault model was proposed in [2] where the researchers targeted both an 8-bit microcontroller and a Xilinx FPGA. In the timing fault model, the coupling between the EM pulse and the power ground network temporarily reduces the supply voltage and causes a setup time violation due to the increased propagation delay. However, the timing fault model was empirically proved not practical in [3] where the sampling fault model was proposed. In the sampling fault model, the EM pulse induces temporary perturbations to the circuit node. A metastability phenomenon occurs when this perturbation is within the setup and hold time window of the clock edge. A charge-based fault model was proposed in [4] based on the critical charge concept. In [5] a high precision EM pulse was able to induce repeatable faults into a microprocessor with ARM v7-M structure. In [6] the researchers observed that hardware exceptions were triggered while using EMFI to inject faults to an ARM Cortex M3 core processor. The exceptions were triggered due to illegal instructions. Moreover, other researchers have applied EMFI in attacking real applications such as secure boot [7], and AES [4]. The EMFI attacks were proposed targeting different microcontroller architectures including PIC [4], AVR [2] and ARM [6]. Some research papers pointed out that a fault injection attack could also be applied to RISC-V architecture [8] [9]. However, their attacks were based on simulations with no empirical results, thus lacking verification due to the unknown structure of the target chip and the complexity of the EM coupling. To thwart EMFI attacks, a glitch detector was designed to detect the fault injection attempt [10]. However, it turned out not to be efficient due to the local property of the EMFI. Some other software countermeasures to thwart EMFI were proposed in [11] [12] based on the assumption of an instruction skip fault model. In summary, there remains a lack of in-depth research in determining the Register Transfer Level (RTL) fault model. Additionally, the proposed countermeasures were either not efficient or had a low fault coverage [1]. Moreover, since RISC-V has shown promise for many applications in the near future, the study of EMFI attack on a RISC-V processor needs to be addressed.

In this paper, an in-depth analysis of EMFI is presented over NXP's LPC1114 chip featuring an ARM Cortex-M0

core and SiFive's FE310-G002 chip featuring an E31 RISC-V core. The previously proposed charge-based fault model [4] is empirically verified on both architectures. Furthermore, the RISC-V processor is found to be susceptible to multi-instruction skips and faults.

The rest of the paper is organized as follows: Section II describes the setup used for EMFI. Section III presents the experiments conducted on ARM as well as their results. And similarly in Section IV the experiments and results for RISC-V are presented. In Section V the charge-based fault model is described and empirically verified on both architectures. And finally, Section VI summarizes the results and concludes the paper.

II. EMFI Platform Setup

The core of the EMFI platform utilized a low-cost CNC machine, which worked as a movable XYZ stage, a burst power station from Langer EMV which generates a low rise time voltage pulse [13], EM probes, oscilloscope, and a function generator.

The function generator acted as the external clock source for the LPC1114, which was de-capsulated from the backside. For the FE310-G002 chip, the on-chip Phase-Locked Loop (PLL) was used to provide frequencies up to 384 MHz. Also due to its thin package, there was no need for the FE310-G002 chip to be de-capsulated. The oscilloscope was used to measure the delay between the trigger output from the chip and the associated pulse (approximately 270-290ns) with a jitter of 20ns. The rise time of the pulse was measured to be of 7 to 14 ns.

Several probe tips with different diameter sizes (1 mm, 2 mm, 3 mm) were made to inject faults to both the RISC-V and ARM Cortex-M0 cores. Initial experiments showed that only the probe tip with a 2 mm diameter could successfully inject faults to the ARM Cortex-M0 processor, while faults could be injected to the RISC-V processor with any of these probe tips. However, the 1 mm tip was used for RISC-V due to its higher resolution.

A python script was designed to synchronize all the equipment and read the results from the chip. For LPC1114, inside the assembly code, a fault handler was integrated to send the data out for off-chip analysis. However, for FE310-G002, the JTAG port was used for debugging by reading all general-purpose registers of the RISC-V core for off-chip analysis.

III. Experiments and Results on LPC1114

The ARM Cortex-M0 applies a 3-stage pipeline named as fetch, decode and execute with all the 16-bit Thumb-1 instruction set and a small portion of the 32-bit Thumb-2 instruction set. The processor fetches a word from the memory every other clock cycle. Most instructions take one cycle to execute. However, LDR/STR instructions take two cycles due to the pipeline architecture of the AHB bus [14].

A. Experiment and Target Program

The program running on the chip was designed in assembly to provide better timing information. Code 1 presents the targeting program. In the main loop, the processor writes the same data to registers R3 and R4 followed by a comparison. The **LDR R4, =(0xFEDCBA98)** is actually an LDR PC-relative instruction. The assembler first writes the constant to the literal pool and then calculates an offset from the literal pool to the current PC to load the constant from the literal pool. If the data in R3 and R4 are not equal, the program jumps to the FAULT loop where the faulty data inside R3 and R4 are sent out by the UART. A trigger signal is asserted before the target LDR PC-relative instruction. The number of NOPs between them could be adjusted accordingly based on the clock frequency and delay between the trigger to the pulse in order to deliver the EM pulse at correct timing. An initial experiment was done, and only with overclocking (while chip remained functional) could a fault be injected. The clock frequency was boosted to 104.4 MHz and supply voltage was the nominal 3.3 V.

Code 1. Test program targeting LPC1114

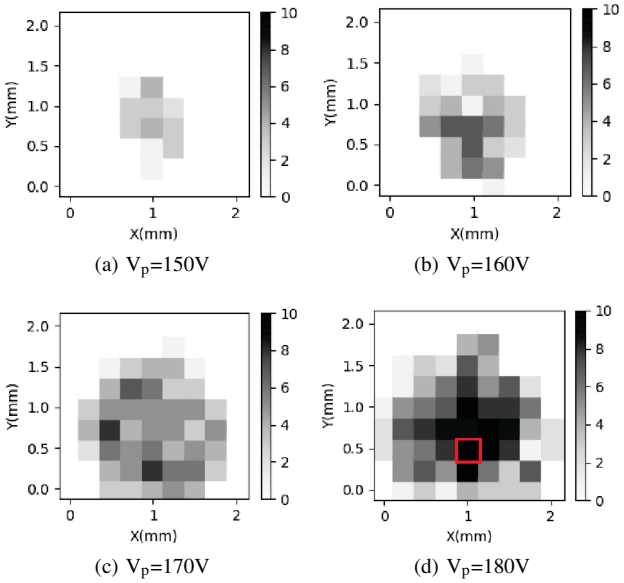
```
_MAIN:
    (Assert trigger)
    26 * NOPs
    LDR R3, = (0xFEDCBA98);
    LDR R4, = (0xFEDCBA98);
    CMP R3, R4;
    BNE _FAULT;
    (De-assert trigger)
    B _MAIN
```

After successfully injecting the fault with manual placement of the probe tip, the probe tip was attached to the CNC machine to run a scan over the chip. The objective of the scan was to determine the best spot for injecting the fault for consistency in further experiments. The scan was run for 10 rounds with different EM pulse voltages ranging from 150 V to 180 V over 9×9 coordinates for the 2×2 mm die with a resolution of 0.254 mm per step. The number of faults injected on each coordinate was reported and used to draw the shmoo plot in Fig. 1 identifying the best location for injecting the fault.

B. Analysis of Fault Type and Timing

The opcode for each instruction and the associated address is shown in Table I. The faulty data sent out by the UART shows that the value stored in R4 was changed to 0x480CD19A. The faulty data was consistent. Later it was found that 0xD19A was the opcode for **BNE _FAULT** instruction and 0x480C was the opcode for the subsequent instruction **LDR R0, [PC, #0x30]**. Thus, when executing the **LDR R4, [PC, #0x30]** instruction with the EM pulse, the address used to load the data was likely faulted. Consequently, the processor actually loaded the next two instructions into the register R4.

Based on Table I, to correctly load the data at address 0x00000A48 and 0x00000A4A, the PC address should be



(a) V_p=150V (b) V_p=160V

(c) V_p=170V (d) V_p=180V

Fig. 1. Shmoo plots for the number of faults injected at each point on LPC1114 with different EM pulse voltages

TABLE I
ADDRESSES AND OPCODES OF THE INSTRUCTIONS FOR THE TEST PROGRAM IN CODE 1

Address (hex)	Opcode (hex)	Instruction
0x00000A14	0x4C0C	$LDR\ R4,\ [PC, \#0x30]$
0x00000A16	0x42A3	$CMP\ R3,\ R4$
0x00000A18	0xD19A	$BNE\ 0x00000950$
0x00000A1A	0x480C	$LDR\ R0,\ [PC, \#0x30]$
0x00000A48	0xBA98	$DCW\ 0xBA98$
0x00000A4A	0xFEDC	$DCW\ 0xFEDC$

0x00000A18. At this PC address, the processor was fetching both the **BNE _FAULT** and **LDR R0, [PC, #0x30]** instruction while executing the **LDR R4, [PC, #0x30]** instruction in parallel as shown in Fig. 2. Therefore, the targeting address was likely faulted from 0xA48 to 0xA18 and then the **LDR R4, [PC, #0x30]** instruction loaded the opcode of **BNE _FAULT** and **LDR R0, [PC, #0x30]** instruction into register R4.

Similar behaviors were found when we made some modifications to our targeting program. First, in order to eliminate the possibility that the fault was injected to the instructions which were decoding or fetching in parallel with the **LDR R4, [PC, #offset]** instruction, numerous NOPs were added after the **LDR R4, [PC, #offset]** to push the **CMP R3, R4** and **BNE _FAULT** instruction away from the EM pulse. The same kind of fault was injected successfully. The address was likely faulted from 0xA74 to 0xA14 and the fault data was 0xBF004C17. Table II presents the associated address and opcode of each instruction.

Another experiment also showed that the fault was not affected by the instructions following the **LDR R4, [PC, #offset]**. The first NOP after the **LDR R4, [PC, #offset]** was replaced with **MOVS R0, #0** whose opcode was 0x2000. As expected, the faulty value received from the UART was changed to 0x20004C17. Further experiments indicated that the targeted address where the data should be loaded likely affected the faulty address where the faulty data was actually loaded from during fault injection. Assume the targeted address was 0xAMN where M and N are both 4 bits hex and N could only be 0, 4, 8, C since the memory is word-aligned, during fault injection the faulty address becomes 0xA1N. This property was verified for $0x48 \leq 0xMN \leq 0x80$.

TABLE II
ADDRESS, OPCODE OF THE INSTRUCTIONS FOR TEST PROGRAM ONE WITH MULTIPLE NOPS INSERTED

Address (hex)	Opcode (hex)	Instruction
0x00000A14	0x4C17	$LDR\ R4,\ [PC, \#0x5C]$
0x00000A16	0xBF00	NOP
0x00000A74	0xBA98	$DCW\ 0xBA98$
0x00000A76	0xFEDC	$DCW\ 0xFEDC$

IV. EXPERIMENTS AND RESULTS ON FE310-G002

The E31 core on FE310-G002 is a high-performance single-issue in-order execution five-stage pipeline processor. The five stages are: instruction fetch, instruction decode and register fetch, execute, data memory access, and register writeback. The core supports standard Multiply, Atomic, and Compressed RISC-V extensions (RV32IMAC ISA) [15].

Four different experiments were performed on the RISC-V core. The objectives of each experiment was as follows: 1) to identify the best spot over the chip where faults can be injected, 2) to determine the timing when faults can be injected, 3) to analyze how many instructions can be faulted at once, and 4) to see the effect of varying the clock frequency and the supplied voltage, which will be discussed later in Section V.

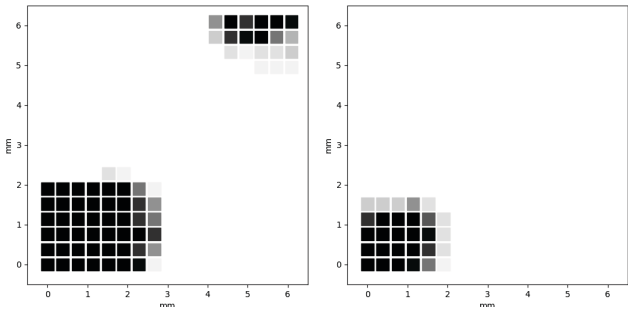
A. Identifying Best Spatial Position for EMFI

A scan over the chip using the CNC machine was performed for 10 rounds over 17 $\times$ 17 coordinates of the 6 $\times$ 6 mm chip with a resolution of 0.381 mm per steps using different voltages ranging from 150 V to 170 V. The number of faults injected on each coordinate was recorded and used to draw the shmoo plots shown in Fig. 3 identifying the best location for injecting the fault.

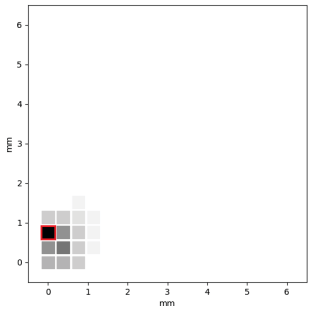
The code used for this experiment as illustrated in Code 2 has multiple ADDI instructions after triggering the EM system. The window of the ADDI instructions is large enough so that regardless of the accurate timing of the applied EM pulse, one of the instructions should be exposed to the EM pulse. If one of the instructions is faulted, then the final sum will be different from the expected correct sum and thus an injected fault is detected. Another scenario could be that the faulted instruction is corrupted causing an exception, which is detectable as well from the exception routine.

Cycles	0	1	2-22	23	24	25	26	27	28	29	30	31
Trigger PIO0_2												
	NOP2				NOP26		LDR R4			BNE _FAULT		
Fetch	NOP3		Fetch NOPs		LDR R3		CMP R3, R4		STALL	LDR R0	STALL	
Decode	NOP1	NOP2	Decode NOPs	NOP24	NOP25	NOP26	LDR R3	LDR R4		CMP R3, R4		BNE _FAULT
Execute	trigger	NOP1	Execute NOPs	NOP23	NOP24	NOP25	NOP26	LDR R3		LDR R4		CMP R3, R4

Fig. 2. Timing diagram of test program in Code 1



(a) V_p=170V (b) V_p=160V



(c) V_p=150V

Fig. 3. Shmoo plots for the number of faults injected at each point on FE310-G002 with different EM pulse voltages

Code 2. Test program targeting RISC-V to capture any fault injected to any of multiple ADDI instructions

```
li x22,10
MAIN:
    li x16,301
    li x17,1
    (Assert trigger)
    300*addi x17,x17, 1
    sub x14,x16,x17
    bne x14,x0,FAULT
    addi x22,x22,-1
    bne x22,x0,MAIN
```

For all experiments, the maximum functional frequency defined by the manufacturer (320 MHz) and the nominal supply voltage (1.8 V) were utilized.

B. Determining Timing Position of Target Instruction

The first experiment identified the hottest spot to inject faults, but still, the exact timing on which instruction is faulted is not yet identified. In this experiment as shown in Code 3, after the EM system trigger is asserted, 101 compressed NOP (C.NOP) instructions are used after the trigger (to account for the delay between the trigger and the EM pulse), followed by a window of 20 instructions. The 20 instructions utilize a single target ADD instruction among 19 other C.NOPs. Twenty experiments were run, where each experiment placed the ADD at a different position amongst the 19 C.NOPs (e.g. ADD + 19 C.NOPs, 1 C.NOP + ADD + 18 C.NOPs, ... , 19 C.NOPs + ADD). Experiments were repeated 100 times for statistical analysis.

Code 3. Test program targeting RISC-V to help identifying the best position for the target instruction

```
MAIN:
    li x16,7
    li x17,5
    li x13,2
    (Assert trigger)
    101*C.NOP
    - - - - - - - - - - - - - - - - - -
    add x17,x17,x13
    19*C.NOP
    - - - - - - - - - - - - - - - - - -
    10*C.NOP
    sub x14,x16,x17
    bne x14,x0,FAULT
```

Fig. 4 shows different types of faults injected to the target instruction and their frequencies. The target instruction is skipped more often than corrupted or causing an exception. Also due to the jitter of the EM pulse, surrounding instructions are often affected. As expected, the probability of fault injection increases as the EM pulse voltage increases.

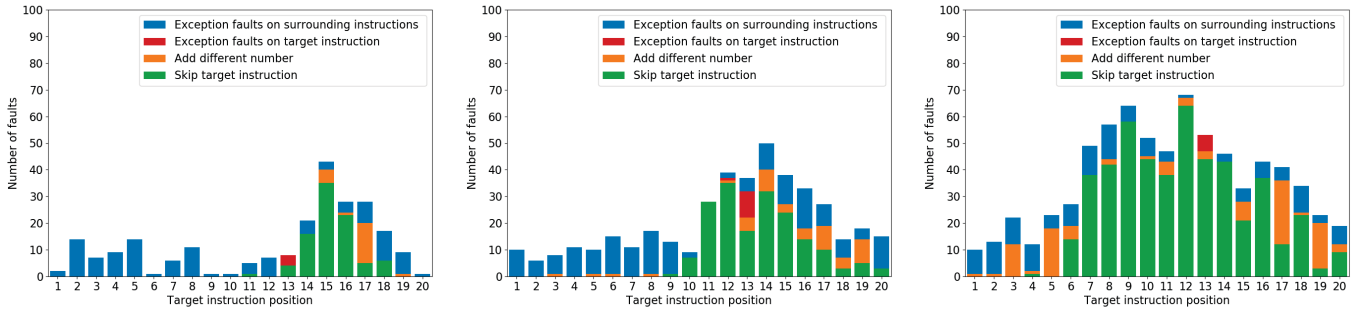
Although target instructions were more likely to be corrupted than to cause an exception, when they caused exceptions, the type of exception was illegal instruction. Nevertheless, surrounding instructions (C.NOPs) were more likely to cause exceptions with different exception types as in Table III.

The “mcause” register, one of the CSRs (Control and Status Registers) was used to provide the type of exceptions caused and reported in Table III.

As shown, most of the exceptions were illegal instruction exceptions, which provided evidence that instructions’ binaries had been corrupted during or before the decoding stage, when such an exception is caused. Given that C.NOP is encoded as “0000000000000001” and only flipping bit-0 will create the illegal instruction “0000000000000000”, it is likely to be the cause of these illegal instruction exceptions.

C. Examining Faulting Multiple Instructions

From the previous experiment, in some cases the target instruction would be faulted as well as an exception fault to



(a) EM pulse 150 V (b) EM pulse 160 V (c) EM pulse 170 V

Fig. 4. Timing analysis for target instruction positioning and types of faults injected with different EM pulse voltages (VDD=1.8, Freq=320)

TABLE III
DIFFERENT TYPES OF EXCEPTIONS TO C.NOPS SURROUNDING THE TARGET INSTRUCTION

	EM=150V		EM=160V		EM=170V	
	Freq	%	Freq	%	Freq	%
Total exceptions	121	100	191	100	198	100
Illegal instruction	108	89.3	145	76	191	96.5
Breakpoint	0	0	0	0	3	1.6
Load address misaligned	13	10.8	0	0	0	0
Load access fault	0	0	45	23.6	0	0
Store address misaligned	0	0	0	0	0	0
Store access fault	0	0	1	0.6	4	2.1

a following C.NOP is raised, which indicates two instructions being faulted at once. Also as it was shown that target instructions from the 7th to 16th position were more likely to get fault injected, this also supports the probability of having multiple instructions faulted at once. To test that, the program in Code 4 utilizes 10 target instructions in the window from the 7th to 16th position. That window of 10 target instructions specifically contained 10 ADD instructions using the same two input registers (x12,x13) but with different output registers (x14-18,24,28-31), which are expected to contain the same output if no fault is injected. Experiments were repeated 1000 times to apply statistical analysis.

Code 4. Test program targeting RISC-V exposing 10 ADD instruction to the range of the EM pulse

```
MAIN:
    li x12,3
    li x13,4
    li x14,5
    li x15,5
    (load other 8 registers with 5)
    (Assert trigger)
    101*C.NOP
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    6*C.NOP
    add x14,x12,x13
    add x15,x12,x13
    (assign other 8 registers with x12 + x13)
    4*C.NOP
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    10*C.NOP
```

Fig. 5 shows the frequency distribution of having X multiple instructions faulted at once, with bar 0 representing no faults at all and bar 1 representing only one instruction fault. The red (left) bars refer to multiple instruction faults in general including corrupted data and instruction skips. The purple (right) bars refer to only multiple instruction skips, which maybe more devastating and able to overcome many software countermeasures.

Fig. 5 shows histograms for the frequency of faulting multiple instructions at once, Fig. 5a shows that at lower EM pulse voltages such as 150 V, the probability of having no faults is more than 94%, and it is very unlikely (though possible) to have multiple faults up to 4 instructions at once.

Fig. 5b shows that by increasing EM pulse voltage to 160 V, the probability of having 2 instructions faulted at once is 29%, while still, it is very unlikely (though possible) to have up to 5 instructions faulted at once.

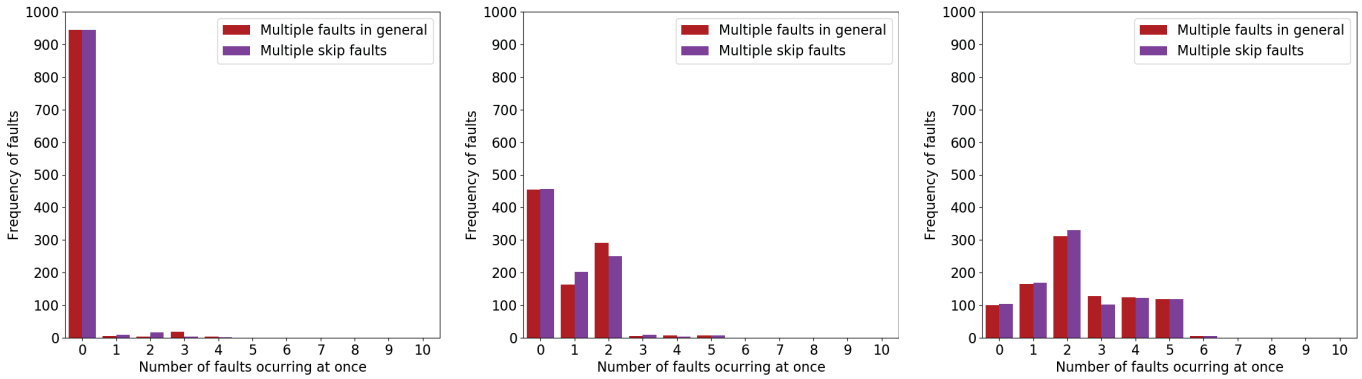
However, as shown in Fig. 5c when it comes to EM pulse voltage of 170 V, it is very probable to have multiple faults at once with probability of 31%, 13%, 12% and 12% for faulting 2, 3, 4 and 5 instructions at once respectively, with even a very low non-zero probability of having 6 instructions faulted at once.

V. CHARGE-BASED FAULT MODEL

In this section, the charge-based fault model is described with empirical results on both ARM and RISC-V processor.

A. Definition of the Charge-Based Fault Model

Unlike the previously proposed models in [2] [3], the charge-based fault model utilized the concept of critical charge. The critical charge is defined as the minimum amount of charge collected at a node to alter the normal behavior of this node or cause an ”upset” [16]. The EM pulse could induce charge (either via noise or power-ground network) accumulated at a specific node. If this induced charge is greater than the critical charge, a fault is injected. To reduce the critical charge, the attacker can reduce the supply voltage or increase the clock frequency. The circuit node will have less current or less time to charge/discharge the capacitor at the node.



(a) EM pulse 150 V (b) EM pulse 160 V (c) EM pulse 170 V

Fig. 5. Histogram for the number of ADD instructions faulted at once with different EM pulse voltages (VDD=1.8, Freq=320)

B. Verification of the Charge-Based Fault Model on ARM

The program used to validate the charge-based fault model was developed from the test program shown in Code 1 by adding 16 NOPs after the **LDR R4, [PC, #offset]** and 6 NOPs after the **CMP R3, R4** instruction. Initially, we set the clock frequency to 104.4 MHz and supply voltage to 3.3 V. When the delay from trigger to pulse was set to 284 ns, faults could be injected. The delay setting was then increased to 294 ns and then the reduced frequency was recalculated to keep the distance between the pulse and the associated edge the same.

TABLE IV
VALIDATING THE CHARGE-BASED FAULT MODEL ON LPC1114

FI/3.3V	FI/VDD	Freq (MHz)	Delay (ns)			
			D28	D29	D30	D31
Yes	Yes/3.3V	104.4	-15.8	-6.2	3.4	12.9
No	Yes/2.08V	100.65	-15.8	-	-	-
No	Yes/2.09V	100.77	-	-6.2	-	-
No	Yes/2.10V	100.89	-	-	3.4	-
No	Yes/2.11V	101.00	-	-	-	12.9

Experimental results on the verification of the charge-based fault model is presented in Table IV. Delay Di is the time between the EM pulse and the associated end of clock cycle i. A negative value indicates that the EM pulse was injected after the associated clock edge. For example, D29 is the delay from the EM pulse to the end of the 29th cycle (see Fig. 2). When the clock frequency was reduced, faults could no longer be injected even though the delay between the pulse and the associated edge (one of four in Table IV) was the same. Only by reducing the supply voltage, faults could be injected again. The fault type was the same as the result of the first row.

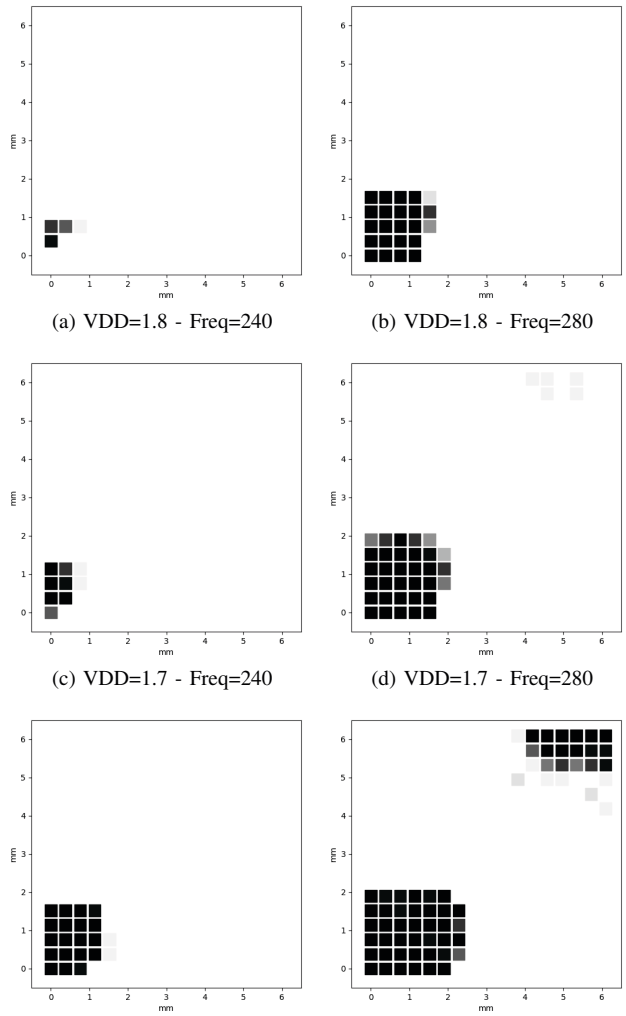
C. Verification of the Charge-Based Fault Model on RISC-V

Since multiple instructions could be faulted at once on the FE310-G002 chip, accurate timing was not necessary to verify the charge-based model. It was easier to verify the model by using the same experiment described in IV-A using Code 2 but with variant frequencies (240 MHz and 280 MHz), and functional supply voltages (1.8 V, 1.7 V, and 1.6 V), while keeping the EM pulse voltage the same (170 V). The chip remained functional at all combinations of settings as long as EMFI was not applied. As shown in Fig. 6, for both frequencies applied, reducing the supply voltage increased the number of faults injected. Also, for all the three supply voltages applied, the number of faults injected with the 280 MHz clock frequency was more than with 240 MHz.These results indicate that the susceptibility to EMFI increases with the increase of the clock frequency and the decrease of the supplied voltage, which complies with the charge-based model.

VI. DISCUSSIONS AND CONCLUSIONS

On the ARM processor, it has been shown that EMFI targeting PC-relative loads produces faults which load different values into registers, likely due to an address fault. This may support reduced round type of attacks. Additionally, several exceptions were resulting from the EMFI likely due to replacing an instruction with an illegal or undefined instruction. This research is unlike previous research that utilized clock glitches in [10] with fault injection on LDR Rt,[Rn,#imm] resulting in NOP or MOV faulty instruction replacement. In our case, a faulty instruction replacement could not be found equivalent to the address fault effect. In another ARM processor [17] the LDMIA was faulted using voltage glitching resulting in a fault that replaced the destination register by the PC, unlike our research.

On the other side, for the first time, EM fault injection has been empirically studied on a 320 MHz RISC-V processor, and it has been shown that multiple instruction skips are possible with a single EM pulse. Furthermore, as the EM pulse voltage increases, the number of consecutive instructions being skipped also increases. Unlike previous EMFI research which has not reported any multi-instruction skips, only laser FI has been shown to produce multi-instruction skips [18] [19], this research shows for the first time that EMFI on RISC-V also has this capability. Previous countermeasures such as duplication or triplication are not sufficient for this multi-instruction skip



(a) VDD=1.8 - Freq=240 (b) VDD=1.8 - Freq=280

(c) VDD=1.7 - Freq=240 (d) VDD=1.7 - Freq=280

(e) VDD=1.6 - Freq=240 (f) VDD=1.6 - Freq=280

Fig. 6. Shmoo plots for the number of faults injected over the chip using different supplied voltages and frequencies. Keeping pulse amplitude = 170 V

fault (given up to 6 instructions can be skipped with only a 170V EM pulse in this research). These new attacks may be devastating for many embedded processors since they are easier to launch in the field, unlike laser FI (since no decapping or silicon substrate access is required), and furthermore, timing does not need to be sufficiently accurate given the large number of instructions skipped with the EM pulse.

It is possible that exceptions could be used to detect possible attacks, thus acting as a possible countermeasure by restarting/halting security applications after an exception. For example, processors could be designed with "sparse" ISAs where the hamming distances between valid instructions are large and the hamming distances of a valid instruction to illegal instructions are small. This may increase the probability that an EMFI may lead to an illegal instruction and thus be detected. Other countermeasures may include high Vdd, slower clock frequencies and randomizing the order of real/redundant instructions.

ACKNOWLEDGMENT

This research was supported in part by grants from NSERC and XtremeEDA

REFERENCES

[1] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[2] A. Dehbaoui, J.-M. Dutertre, B. Robisson, and A. Tria, "Electromagnetic transient faults injection on a hardware and a software implementations of aes," in *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2012, pp. 7–15.

[3] S. Ordas, L. Guillaume-Sage, and P. Maurine, "Electromagnetic fault injection: the curse of flip-flops," *Journal of Cryptographic Engineering*, vol. 7, no. 3, pp. 183–197, 2017.

[4] H. Liao and C. Gebotys, "Methodology for em fault injection: Charge-based fault model," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 256–259.

[5] L. Riviere, 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 *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 62–67.

[6] 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*. IEEE, 2013, pp. 77–88.

[7] A. Cui and R. Housley, "{BADFET}: Defeating modern secure boot using second-order pulsed electromagnetic fault injection," in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[8] M. Werner, R. Schilling, T. Unterluggauer, and S. Mangard, "Protecting risc-v processors against physical attacks," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1136–1141.

[9] J. Laurent, V. Beroulle, C. Deleuze, and F. Pebay-Peyroula, "Fault injection on hidden registers in a risc-v rocket processor and software countermeasures," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 252–255.

[10] L. Zussa, A. Dehbaoui, K. Tobich, J.-M. Dutertre, P. Maurine, L. Guillaume-Sage, J. Clediere, and A. Tria, "Efficiency of a glitch detector against electromagnetic fault injection," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[11] N. Moro, K. Heydemann, A. Dehbaoui, B. Robisson, and E. Encrenaz, "Experimental evaluation of two software countermeasures against fault attacks," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2014, pp. 112–117.

[12] 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*. ACM, 2010, p. 7.

[13] *ICI 03 L-EFT set*, LANGER EMV-Technik. [Online]. Available: https://www.langer-emv.de/en/product/ic-side-channel-analysis/94/ici-03-l-eft-set-ic-em-pulse-injection-langer-pulse/1166

[14] J. Yiu, *The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors*. Academic Press, 2015.

[15] *SiFive E31 Manual*, SiFive, Inc., 6 2019, v19.05. [Online]. Available: https://www.sifive.cdn.prismic.io/sifive%2F2df24239-b9bf-42cd-b287-2d57030e91fa_e31-core-complex-manual-v19.05.pdf

[16] P. Dodd and F. Sexton, "Critical charge concepts for cmos srams," *IEEE Transactions on Nuclear Science*, vol. 42, no. 6, pp. 1764–1771, 1995.

[17] 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)*. IEEE, 2016, pp. 25–35.

[18] J. Breier, D. Jap, and C.-N. Chen, "Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on aes," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 99–103.

[19] K. Amin, C. Gebotys, M. Faraj, and H. Liao, "Analysis of dynamic laser injection and quiescent photon emissions on an embedded processor," *Journal of Hardware and Systems Security*, 2020.