Laser Fault Injection in a 32-bit Microcontroller: from the Flash Interface to the Execution Pipeline

Vanthanh Khuat*‡, Jean-Luc Danger* and Jean-Max Dutertre†
*LTCI, Télécom Paris, Institut polytechnique de Paris, France
Email: {khuat, jean-luc.danger}@telecom-paris.fr
†Mines Saint-Etienne, CEA, Leti, Centre CMP, F - 13541 Gardanne France
Email: dutertre@emse.fr
‡Faculty of Information Technology, Le Quy Don Technical University, Hanoi, Vietnam
Email: van-thanh.khuat@lqdtu.edu.vn

***Abstract*—In this paper, we report on a method for obtaining faults using Laser Fault Injection (LFI) in a 32-bit Microcontroller (MCU) from the Flash interface to the execution pipeline via the AHB bus. Different fault behaviors were obtained at six positions along the instruction channel. Instruction(s) were observed to be faulted with a reproducibility of 100% at each position. By collecting the faults on all the positions together and analyzing their behaviors, the faults were identified and characterized. The faults on the Flash interface buffer are different depending on the cache operation modes. When the cache is disabled, the fault is related to a block of 32 bits, whereas when the cache is enabled, the fault is related to a block of 64 bits. Two fault models, namely, replay and skip of instructions block were obtained depending on the injection position. The fault happening at the AHB bus is with a block of two instructions in both cache operation modes. Depending on the injection position, two fault models of replay and skip of two instructions were also observed. The faults on the core pipeline are related to a single instruction. The fault behavior is such that both the fetch and execution stages were faulted. The skip of a single instruction was obtained by faulting either the fetch or the execution stage of the core pipeline. By increasing the Pulse Width (PW), tens to more than one hundred of instructions were faulted at each position. The impacts of LFI parameters such as the PW and the power on the faults were studied. In addition, we compared skip fault models achieved at different positions. Our results illustrate the spatial and temporal accuracy of the LFI, thus pointing out the vulnerable positions and unveiling information of the device architecture.**

***Index Terms*—Laser fault injection, Fault models, Characterization, Microcontroller.**

I. Introduction

Recently, with the development of the Internet of Things (IoT), the demand for using Microcontrollers (MCU) has witnessed a considerable rise. Data being processed by MCU including password, account, etc. are sensitive and valuable. Because the devices are physically accessible, in addition to cyberattacks, many physical techniques have been developed for extracting unauthorized data. Among these techniques, Fault Injection (FI), an active side-channel attack, poses a significant threat to MCUs. In this method, by introducing a physical stress into the device, the adversaries can alter the way it functions; hence, it produces a faulted result which is further used in Differential Fault Analysis (fault vs no-fault) to extract the unauthorized data.

The most common techniques used for Fault Injection Attacks (FIA) are: Clock or Voltage tampering [1], [2], [18], Electromagnetic Fault Injection (EMFI) [3], [11], [13], [14], Optical Fault Injection [6], [17], [19]. The attacks can be classified into invasive, semi-invasive, and non-invasive attacks. Clock or Voltage tampering is relatively easy to implement with a simple and low-cost setup, however, it has a global impact on the device under test. In contrast, EMFI and LFI only have a local impact on a specific part of the device. Specifically, LFI has an extremely high spatial resolution which makes it possible to achieve a local effect on a single element with even the most advanced technology [5].

LFI can be classified into semi-invasive techniques because the device needs to be unpackaged to guarantee that the light can reach the circuit layer. Optical Fault Injection is the method in which an optical source (including the laser) is used to induce ionization in electronic circuit devices, causing the targets to behave differently. The disadvantages of this method are: having a relatively high cost of equipment and requiring well-trained staff to operate.

Since the first implementation of optical attack in [19], there have been plenty of works focusing on examining the characteristics of the optical attack, understanding the mechanism underneath, and developing the countermeasures against it. Many of them are dedicated to studying different kinds of laser-induced faults in MCU.

LFI is reported of being able to cause faults on data stored in volatile memories (Static Random Access Memory (SRAM) or registers) [10], [15], [16], with the different underlying technologies and data while moving from one to another component without affecting the data stored in non-volatile source memories [3], [11], [22]. Specifically, LFI can be used to target and modify instructions, causing them to be corrupted, skipped, etc. Recently, Dutertre et al. [6] reported a powerful LFI-induced multiple instructions skip fault model, in which the authors were able to skip an arbitrary number of instructions with a maximum of 300 instructions by using a laser pulse with a relatively long PW.

In MCU, instructions are stored in program memories and passed through several stages or components before being executed. Most of the available works demonstrated the ability

to induce change into instructions being loaded from memories, resulting in instructions modification or skip. In our previous work [9], with the same target, we were able to fault instructions at the Flash interface buffer, and achieved two fault models namely, skip and replay of a block of instructions using EMFI. Recently, we reported on the replay of a block of instructions fault model caused by laser-induced prevention of Flash interface buffer update [8].

In this paper, by using LFI, we were able to fault instructions in several stages such as the Flash interface buffer, the AHB data bus, the fetch and execution stages in the core pipeline. The faults were induced at different positions of the laser spot. To our best knowledge, no studies have been reported of being able to fault the core pipeline and analyzed the fault on instructions following their flow from the Flash interface to the execution pipeline.

The main contributions of this paper are:

- being able to fault instructions from the Flash interface to the core pipeline in a 32-bit MCU such as fault on instructions updated into the Flash buffer, fault on instructions loaded into the AHB bus, fault on the core fetch and execution stages;
- identifying the faults in different stages and their behaviors, and characterizing the fault models at instruction level and bit level;
- investigating the impact of LFI parameters such as the PW and the power on the faults;
- comparing the instruction(s) skip fault models obtained with LFI at different positions.

The rest of the paper is organized as follows. Section II describes the experimental setup, target, test code, fault definition, and methodology. Section III describes the faults of instructions on different stages from the Flash interface to the core execution pipeline. Section IV describes the impact of the PW on the number of faulted instructions. Section V investigates the impact of the laser power on different fault models. Section VI characterizes the fault model at bit level. Section VII compares multiple instructions skip fault models obtained at different positions. Finally, section VIII provides the main conclusions and perspectives.

II. Experimental setup and methodology

A. Laser bench

Fig. 1 shows the experimental setup we used for conducting LFI on the MCU. The laser platform, as shown in Fig. 1(a), consists of a laser source, a microscope, an XYZ stage, an Infrared (IR) camera, and a computer. The laser source can produce laser pulses with a wavelength of 1,064 nm which allows the light to pass through several hundreds of μm of silicon. The laser PW is tunable in the range from 50 ns to 1 s. In addition, the laser source allows obtaining a programmable delay, and a power ranging from 0 to 3 W. The light is conducted to and focused by a microscope. In our experiments, we used the microscope with a 5× objective to focus the laser beam on the transistors of the device under test.

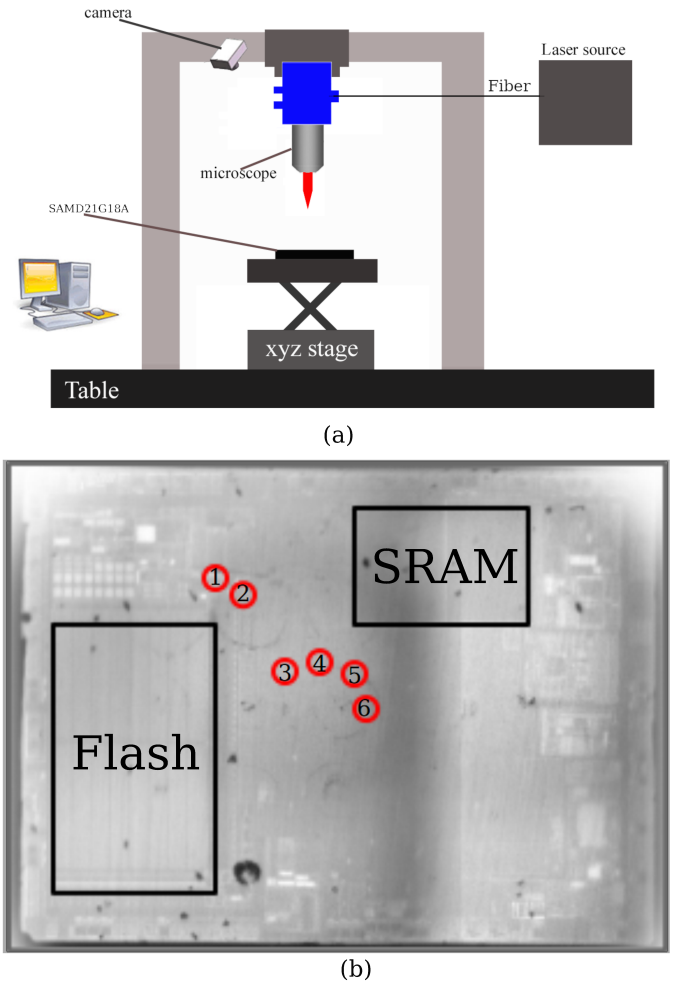


Fig. 1. Experimental setup: (a) laser bench schematic, (b) target backside image taken using an IR camera

The diameter of the laser spot was 20 μm. The device under test was mounted on the XYZ stage which allows controlling the position of the laser spot with an accuracy of 0.1 μm. The IR camera was used to observe the active part of the device and the location of the laser spot through the bulk (FIA were carried out through the target backside). The computer was used to control the laser pulse parameters as well as communicate with the device under test.

B. Device under test and targeted regions

Our target was a 32-bit MCU: a SAMD21G18A [7] which uses an ARM Cortex-M0+ core (2-stage pipeline). This core implements an ARMv6 thumb architecture and the Thumb-2 ISA of which most of the instructions are 16-bit length [12]. The MCU is equipped with a 256 Kb Flash, and a 32 Kb SRAM; the data transfer between the memories and the processor is performed via 32-bit AHB and APB buses. A cache of 8× 64-bit lines is added to improve the performance of the MCU.

The MCU was unpackaged from the backside to ensure that the light can reach the transistor layer. Notice that the

laser power is strong enough to reach the circuit layer of the MCU without requiring it to be thinned down. Fig. 1(b) shows the image taken with an IR camera from the backside of the MCU. The positions of the Flash and SRAM memories are marked with black rectangular shapes. Other structures in the circuit layer can also be seen. The positions marked with red circular shapes are the points of interest we considered for LFI in this work. For all the reported experiments, the MCU was configured to work at 12 MHz with zero wait state which guarantees that there is no delay during the data read operation from the Flash memory. The MCU was debugged using an Atmel-ICE Debugger which allows stopping the program at a breakpoint and collecting registers data for further analysis.

C. Test code and fault definitions

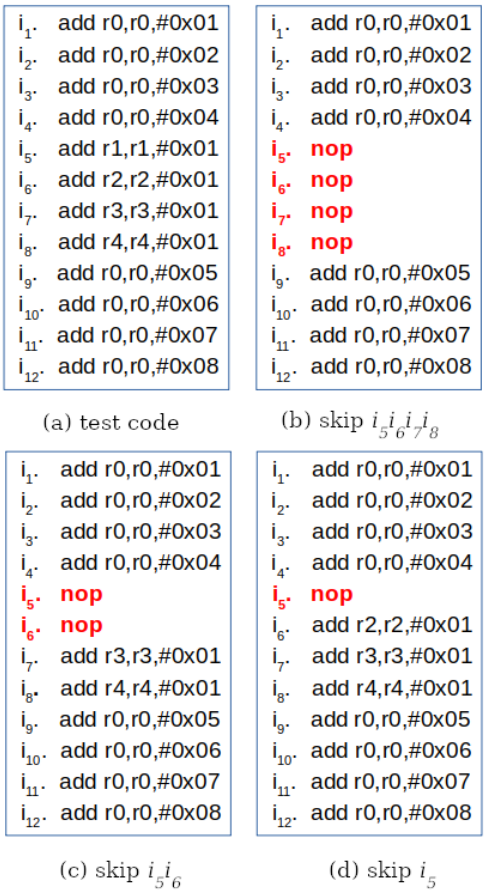


Fig. 2. Test code and skip fault definitions: (a) test code, (b) skip $i_5i_6i_7i_8$, (c) skip i_5i_6 (d) skip i_5

The main part of our test code which consists of twelve instructions is shown in Fig. 2(a); it consists in additions of immediate values in registers `r0` to `r4`. For convenience, the instructions are denoted as (i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}, i_{11}, i_{12}). The initial values of the related registers are as follows: `r0:0x7F8000FF`, `r1:0x01FE0000`, `r2:0x3FC00000`, `r3:0x001FE000`, `r4:0x0007f800`. The following fault definitions are used in this paper.

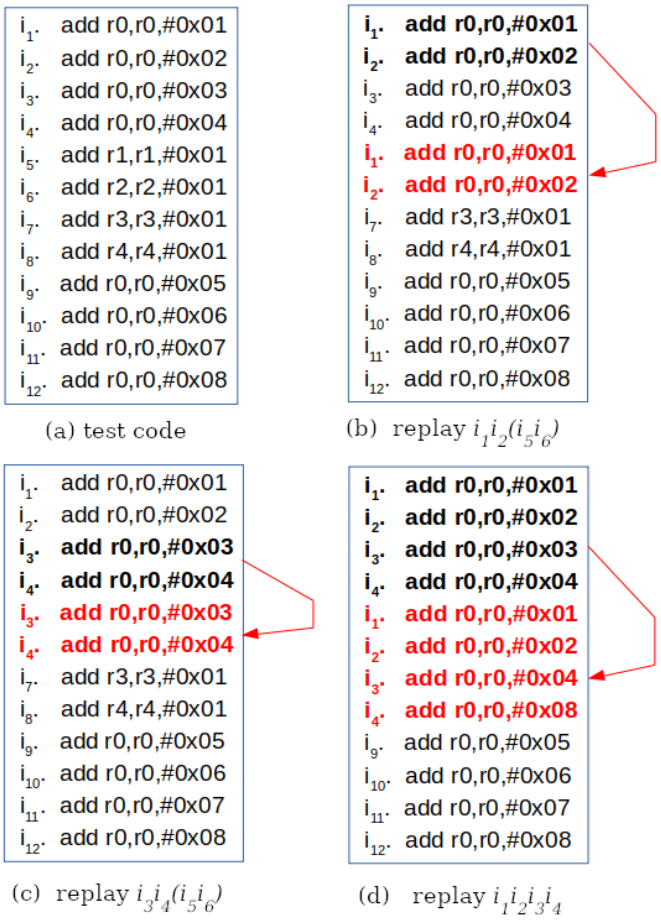


Fig. 3. Test code and replay fault definitions: (a) test code, (b) replay $i_1i_2(i_5i_6)$, (c) replay $i_3i_4(i_5i_6)$, (d) replay $i_1i_2i_3i_4$

- **skip** i_ai_b**...**: instructions (i_a, i_b...) being replaced by no-operation instructions (`nop`, `nop`,..), with a, b being 1, 2, ..., 12;
- **replay** $i_ai_b(i_ci_d)$: instructions (i_c, i_d) being overwritten by instructions (i_a, i_b), with a, b , c, d being 1, 2, ..., 12;
- **replay** $i_1i_2i_3i_4$: instructions (i_5, i_6, i_7, i_8) being overwritten by (i_1, i_2, i_3, i_4);
- **replay** $i_5i_6i_7i_8$: instructions (i_9, i_{10}, i_{11}, i_{12}) being overwritten by (i_5, i_6, i_7, i_8);
- **other**: instruction(s) modification, system fault, and others.

In this paper, mainly two types of fault models were observed: (1) instruction(s) skip, (2) instructions replay. Fig. 2 shows skip fault definitions. Notice that concerning instruction(s) skip fault model, we assume that the instruction is replaced by an instruction equivalent to no-operation (`nop`).

- Fig. 2(a) shows the test code.
- Fig. 2(b) demonstrates the skip $i_5i_6i_7i_8$ in which (i_5, i_6, i_7, i_8) are turned into instructions equivalent to (nop, nop, nop, nop).
- Fig. 2(c) demonstrates the skip i_5i_6 in which (i_5, i_6) are turned into (nop, nop).

- Fig. 2(d) demonstrates the skip i_5 in which (i_5) is turned into (nop).

Fig. 3 shows the replay fault definitions.

- Fig. 3(a) shows test code.
- Fig. 3(b) demonstrates the replay $i_3 i_4 (i_5 i_6)$ in which (i_5, i_6) are overwritten by (i_3, i_4).
- Fig. 3(c) demonstrates the replay $i_1 i_2 (i_5 i_6)$ in which (i_5, i_6) are overwritten by (i_1, i_2).
- Fig. 3(d) demonstrates the replay $i_1 i_2 i_3 i_4$ in which (i_5, i_6, i_7, i_8) are overwritten by (i_1, i_2, i_3, i_4).

It should be pointed out that to differentiate between the replay and skip fault models, `nop` instructions should not be used in the test code, because the replay and skip of a block of `nop` instructions are equivalent. Here, for convenience of post processing, instructions: `add r0,r0,#value` (with `value` being `0x01`,...,`0x08`) were used for (i_1, i_2, i_3, i_4, i_9, i_{10}, i_{11}, i_{12}); and (i_5, i_6, i_7, i_8) are simply the operations to `add 0x01` to the registers `r1`, `r2`, `r3` and `r4` .

D. Test procedure and sensitive regions identification

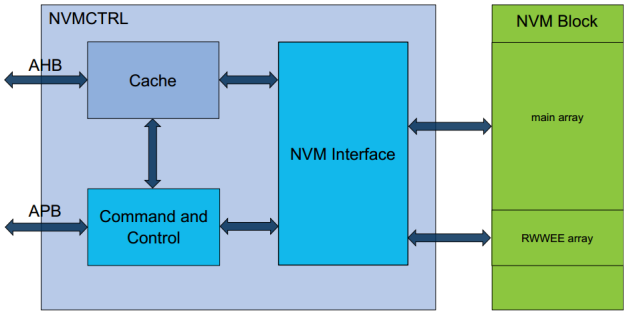


Fig. 4. SAMD21 NVM interface [7]

To detect and roughly classify the induced faults, it is of great importance to have an overview of the flow of the instructions from the Flash memory to the core pipeline. Fig. 4 shows the block schematic of the Nonvolatile Memory (NVM) of the SAMD21G18A. Instructions, normally stored in the NVM such as the Flash, are loaded into the Flash interface buffer before being transferred to the AHB bus. And according to [20], up to 32 bits of data corresponding to 2×16-bit instructions are loaded into the AHB bus every two clock cycles. The instructions then go through the pipeline one after another. Recently, it was demonstrated that the Flash interface buffer can be faulted using EMFI [9] and LFI [8]. In these works, we also noticed that the buffer sizes are different depending on the cache operation mode. As the cache is disabled, the buffer size is 32 bits corresponding to 2×16-bit instructions. As the cache is enabled, the buffer size is 64 bits corresponding to 4×16-bit instructions. Based on the number of faulted instructions and the fault behavior reported when the cache is disabled and enabled, we were able to identify at which stages the instructions were being faulted.

Concerning the experimental process, we first performed a scan of all the chip surface to identify its sensitive regions. For the Flash region, the fault tends to be some specific bits on the opcode of the instruction resulting in instruction modification or skip as reported in [4]. In addition, six regions were found with different fault behaviors. For each region, we focused our experiments to the location that reported the highest fault rate to collect data for further analysis. The positions were marked with red circular shapes numbered from `1` to `6` as shown in Fig. 1(b).

All registers were initialized to a known value at the beginning of all the tests to ensure fault traceability. One test iteration follows three main steps: (1) a target reset is performed to initialize all the core registers; (2) the trigger for laser pulse generator is set, and the test code is then executed; (3) registers content harvesting is performed as the program reaches the configured break-point, or when an interrupt routine is performed. During the campaign, 100 tests were performed for each configuration of FI parameters. Before each test, we collected the contents of all registers at the configured breakpoint to confirm that the program functions correctly in normal conditions and used the values as the reference to detect the faults after each LFI.

III. Fault on Instructions: From the Flash Interface to the Execution Pipeline

A. Fault behavior

Fig. 5 shows the fault models and the fault rates achieved through the Flash interface to the execution pipeline with six different positions designated as P1, P2, P3, P4, P5, and P6 (see Fig. 1 for their exact locations). For each position, we studied the injected faults in both cache disabled and enabled modes. The laser power was set to 1.5 `W`, the PW was 50 ns. All the results were taken in the same time window: from 1,030 ns to 1,585 ns. For all positions, a perfect fault rate of 100% was achieved by finely tuning the laser spot location.

At P1 and P2, the faults are related to a block of two and four instructions depending on the cache operation mode. Two fault models, namely, replay and skip of block of instructions were obtained. As the cache is disabled, the block is with 2×16-bit instructions or 32 bits data. As the cache is enabled, the block is with 4×16-bit instructions corresponding to 64 bits data. The fault behavior is almost the same as the result obtained in [9], in which the faults were ascribed to Electro-magnetic (EM)-induced fault on the Flash interface buffer, and the fault mechanism was discussed. In addition, the locations of P1 and P2 are very closed to the Flash memory. Therefore, we also ascribed the faults obtained at P1 and P2 to laser-induced fault on the Flash interface buffer. The replay fault is caused by laser-induced prevention of the Flash interface buffer update, while the skip fault is caused by laser-induced corruption of one or more bits in the buffer content.

The differences between the faults obtained with EMFI [9] and LFI are as follows. First, in EMFI, the replay and skip fault models were obtained by using different PW at the same probe position. While, in LFI the two fault models were obtained at different positions and with the same laser parameters. Second, the faults were achieved at almost the same delay time in

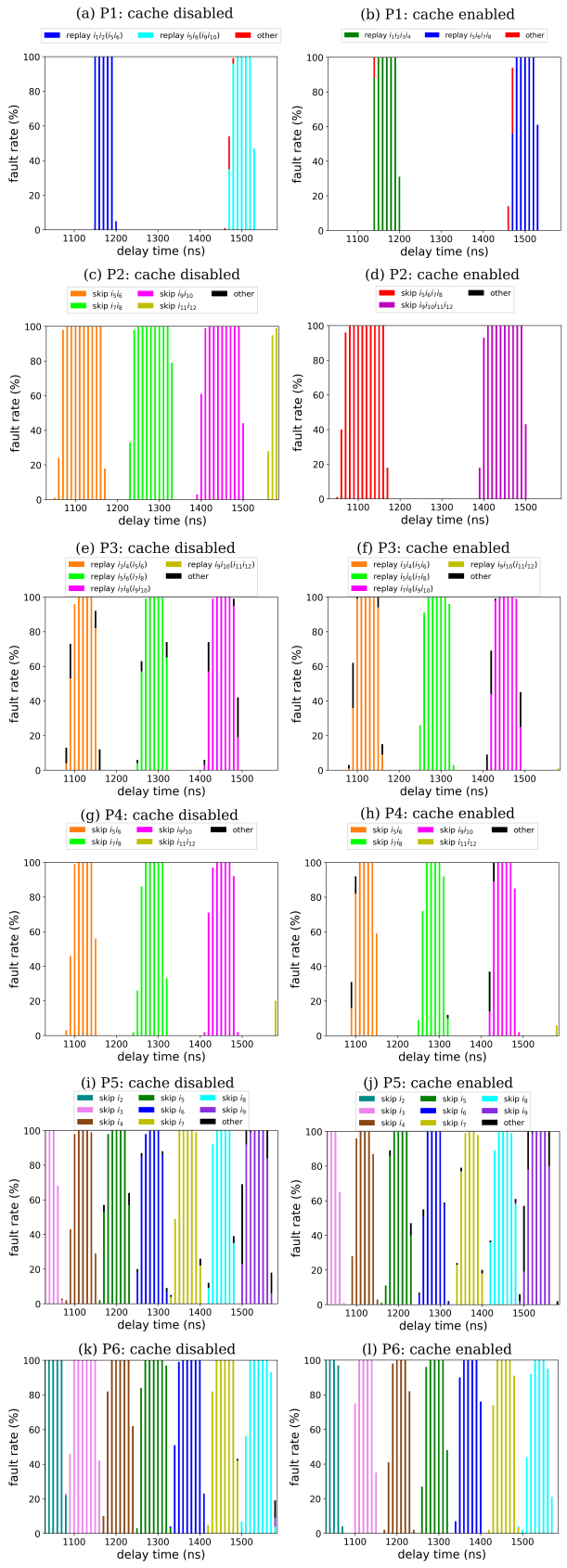


Fig. 5. Laser-induced fault on MCU from the Flash interface to the execution pipeline: replay fault on the Flash interface buffer (a) cache disabled, (b) cache enabled, skip fault on the Flash interface buffer (c) cache disabled, (d) cache enable, replay fault on AHB bus (e) cache disabled, (f) cache enabled, skip fault on AHB bus, (g) cache disabled, (h) cache enabled, skip fault on the core pipeline fetch (i) cache disabled, (j) cache enabled, skip fault on the core pipeline execution (k) cache disabled, (l) cache enabled.

EMFI, while in LFI as shown in Fig. 5(a) to (d), the faults are achieved with a shift in time. This is probably because the laser has a higher local effect as compared to EM pulses. Hence, at each position, the laser pulse only has an impact on one subpart of the Flash interface buffer. Whereas an EM pulse (at a given probe position) has a simultaneous effect on several parts of the Flash interface buffer; which varies for different settings of the PW, thus injecting different fault types. For the faults at P3 to P6, as shown in Fig. 5(e) to (l), there is no obvious difference between the faults achieved with cache disabled and cache enabled. The faults at P3 and P4 are related to a block of two instructions; it repeats every two clock cycles. Considering the case of the cache disabled mode at positions P2 and P4, an instructions skip fault model is obtained in both cases with the same periodicity (two clock cycles) but there is a reduction in the width of the sensitivity time window for P4 w.r.t. P2. However, when considering the case with cache enabled mode, the difference is obvious: at P2, a block of four instructions is skipped while a block of two instructions is skipped at P4. The faults induced at P5 and P6 are single instruction skips with one clock cycle periodicity.

P1 and P2 are close to the Flash, while P3, P4, P5 and P6 are located between the Flash and the SRAM. Because the chip is opened from its backside, it is not possible to see the circuits at these locations, however, P3, P4, P5 and P6 are most likely to be close to the location of the Processor Core (PC).

Notice that in the same target, it was not able to observe the faults obtained at positions from P3 to P6 when using EMFI [9]. Especially, there is no available work reporting on being able to fault the core pipeline or the progress of the instructions as they travel from the Flash interface to the core pipeline. Therefore, it is of great importance to further analyze the mechanism underneath.

B. *Fault mechanism hypothesis*

To understand the mechanism of the fault at the positions from P3 to P6, it is necessary to have a close look at the pipeline of an Arm Cortex M0+. As shown in Fig. 6 (a), in Arm Cortex M0+, data loaded from the Flash interface go through several steps: (1) loaded into AHB bus, (2) the core fetch, (3) the core execution. Noted that according to [21], Arm Cortex M0+ has a two-stage pipeline: stage 1 (fetch and predecode) and stage 2 (main decode and execution). For convenience, in this work, we simply used fetch for stage 1 and execution for stage 2. As an example, here, we consider the execution of instructions (i_5, i_6) of our test code, from their loading from the Flash memory to their execution, that lasts four clock cycles from the transfer of the address of instruction i_5 (denoted as a_5) to the execution of i_6. First, 32-bit data corresponding to 2×16-bit instructions (i_5, i_6) are loaded from the Flash interface into the AHB bus and buffered in HRDATA of which the size is 32 bits. Then the instructions go through the fetch and execution stages of the core pipeline. We can see that data loading into the AHB bus happens every two clock cycles, while the fetch and execution

	CLOCK	1	2	3	4
AHB access	HTRANS	NESQ	IDLE	NESQ	IDLE
	HADDR	a_5		a_7	
	HRDATA		i_5 i_6		i_7 i_8
Core pipeline	Fetch	i_4	i_5	i_6	i_7
	Execute	i_3	i_4	i_5	i_6

(a) Normal execution

	CLOCK	1	2	3	4
AHB access	HTRANS	NESQ	IDLE	NESQ	IDLE
	HADDR	a_5		a_7	
	HRDATA		i_5 i_6		i_5 i_6
Core pipeline	Fetch	i_4	i_5	i_6	i_5
	Execute	i_3	i_4	i_5	i_6

(b) Laser-induced replay of two instructions

	CLOCK	1	2	3	4
AHB access	HTRANS	NESQ	IDLE	NESQ	IDLE
	HADDR	a_5		a_7	
	HRDATA		i_5 i_6		i'_7 i'_8
Core pipeline	Fetch	i_4	i_5	i_6	i'_7
	Execute	i_3	i_4	i_5	i_6

(c) Laser-induced modification of two instructions

	CLOCK	1	2	3	4
AHB access	HTRANS	NESQ	IDLE	NESQ	IDLE
	HADDR	a_5		a_7	
	HRDATA		i_5 i_6		i_7 i_8
Core pipeline	Fetch	i_4	i_5	i'_6	i_7
	Execute	i_3	i_4	i_5	i'_6

(d) Laser-induced fault on core pipeline fetch stage

	CLOCK	1	2	3	4
AHB access	HTRANS	NESQ	IDLE	NESQ	IDLE
	HADDR	a_5		a_7	
	HRDATA		i_5 i_6		i_7 i_8
Core pipeline	Fetch	i_4	i_5	i_6	i_7
	Execute	i_3	i_4	i'_5	i_6

(e) Laser-induced fault on core pipeline execution stage

Fig. 6. Hypothesis on the faults on the AHB bus and the core pipeline: (a) normal execution process, (b) laser-induced replay of two instructions ($i_5 i_6$), (c) laser-induced modification of two instructions (i_7, i_8), (d) laser-induced fault on the core fetch pipeline stage, (e) laser-induced fault on the core pipeline execution stage

in the core pipeline happen every clock cycle simultaneously for two successive instructions (note that we only considered here the 16-bit instructions having the execution time of one clock cycle).

Fig. 6 (b) and (c) show the hypothesis of faults effects (highlighted in red) on data loaded into the AHB bus. According to the description in [21], data from program memories are read into the AHB bus every two clock cycles for better performance. First, we consider a laser pulse during clock cycle 3 that prevents instructions (i_7, i_8) from being loaded into HRDATA. As a result, because HRDATA is not updated, instructions (i_5, i_6) are replayed and instructions (i_7, i_8) are discarded without being executed (as if they were overwritten by (i_5, i_6)). Notice that the replay of two instructions is not the same as the replay of two instructions observed at P1 with cache disabled. At P3, (i_5, i_6) overwrites (i_7, i_8); while at P1 (i_5, i_6) overwrites (i_9, i_{10}).

In the second case, the content of HRDATA is faulted (the update process is completed but with faulted instructions denoted as (i'_7, i'_8)). During our experiments, (i'_7, i'_8) were not recognized by the core, they were considered as (`nop`, `nop`) instructions, resulting in the skip of a block of two instructions. Notice that as the cache is disabled, the fault at P4 is quite the same as that obtained at P2. However, as the cache is enabled, the faults at the two positions can be differentiated by the number of faulted instructions and the fault periodicity: (1) for the fault on the Flash buffer, the block affected consists of four instructions and the fault repeats every four clock cycles, (2) for the fault on AHB bus, the block affected consists of two instructions and the fault repeats every two clock cycles.

From Fig. 5 (a-d) and (e-h), we identified respectively: (1) faults induced on the Flash interface buffer at positions P1 and P2, and (2) faults on data in the AHB bus at positions P3 and P4, which share a common behavior of injecting faults on a block of 2 or 4 instructions. In both cases, the instructions skip (which were probably the modifications of instructions at the opcode level) and replay fault models were observed at close locations. It was in cache enabled mode that skip and replay of block of four instructions were obtained at positions P1 and P2.

Fig. 6 (d) and (e) show the hypothesis of the effects of faults induced on the core pipeline fetch and execution stages for a laser shot at clock cycle 3. As mentioned above, the ARM Cortex-M0+ core pipeline consists of two stages, namely, fetch and execution. At position P5, the core fetch stage is faulted: as a result instruction i_6 is turned into a faulted instruction i'_6 which executes as a (`nop`). While at position P6, the core execution stage is faulted: as a result instruction i_5 is turned into a faulted instruction i'_5 which executes as a (`nop`). This behavior illustrates the progression of instructions into the pipeline stages and the ability to fault different consecutive instructions depending on the laser beam location (targeting the fetch or execution stages) for the same injection time. At positions P5 and P6, the laser PW made it possible to skip a single instruction with a fault rate of 100%.

Notice that there is no paper reporting of being able to fault both the stages pipeline of the MCU.

Through the above analysis, the following conclusions are reached:

- **Position P1**: the replay of a block of instructions due to laser-induced prevention of the Flash interface buffer updating process;
- **Position P2**: the modification of a block instructions (including skip) due to laser-induced bit corruption of instruction's opcodes in the Flash interface buffer;
- **Position P3**: the replay of two instructions due to laser-induced prevention of loading data into the AHB bus;

- **Position P4**: the modification of two instructions (including skip) due to laser-induced bit(s) corruption of instructions loaded into the AHB bus;
- **Position P5**: the modification of a single instruction (including skip) due to laser-induced fault in the core pipeline fetch stage;
- **Position P6**: the modification of a single instruction (including skip) due to laser-induced fault in the core pipeline execution stage.

It is also worth mentioning that in cache enabled mode (as we consider the cache should be used for better performance), the number of faulted instructions and the fault periodicity, as fault injection locations progress from the Flash interface to the core pipeline, are: four instructions every four clock cycles (Flash interface buffer), two instructions every two clock cycles (data on AHB bus), and one instruction every clock cycle (core pipeline: fetch or execution stage).

Notice that the fault behaviors described previously were obtained with cache miss. Further investigations showed that in case of cache hit, the faults at P1 and P2 were not observed, while the faults induced at P3 to P6 were the same. This makes sense because in case of a cache hit there is no instruction loaded from the Flash memory, instructions are instead loaded into the AHB bus directly from the cache memory. Hence, targeting the Flash interface buffer does not effect the executed instructions.

C. Proposed core architecture

Based on the faulty behavior obtained from LFI used as a reverse engineering tool, we drew in Fig. 7(a) schematic of the targeted core architecture. When the cache is disabled, instructions are loaded from the Flash into the Flash interface buffer which consists of two 32-bit (corresponding to 2×16-bit instructions) buffers as shown in Fig. 7(a); the two buffers are updated with new data every four clock cycles with a phase shift of two clock cycles. Then, the instructions from the Flash buffer are loaded into the AHB bus two by two every two clock cycles. Finally, the instructions enter the core pipeline, which consists of two stages: fetch and execution, one after another every clock cycle. In this case, instructions can be faulted at positions from P1 to P6.

When the cache is enabled and cache miss occurs (meaning that the instructions to be executed are not found in the cache memory), the scenario is almost the same with the cache disabled mode. The only difference is that there exists only one 64-bit buffer at the Flash interface (corresponding to 4×16-bit instructions), and it is updated with new data every four clock cycles. Fig. 7 (b) depicts the flow of instructions when cache miss happens. In this case, instructions can be faulted at positions ranging from P1 to P6.

When the cache is enabled and cache hit occurs (meaning that instructions are found in the cache), 32-bit data corresponding to 2×16-bit instructions are loaded into AHB bus directly from the cache and enter the core pipeline. In this case, instructions can be faulted at positions from P3 to P6, and no fault occurs at P1 and P2.

To summarize, thanks to the local effect of the laser pulse when interacting with the MCU, we were able to fault instructions in a 32-bit MCU at different stages from the Flash interface through the execution pipeline and found six positions with different fault behaviors. The faults also reveal helpful information to understand the architecture of the MCU.

In the following sections, the impact of the laser parameters such as its PW and power on the faults are studied.

IV. Impact of the PW

In this section, we studied the impact of the PW on the faults induced at positions from P1 to P6. The MCU was configured to run in cache enabled mode. For each position, by increasing progressively the PW, we were able to increase the number of faulted instructions. Fig. 8 shows how the number of faulted instructions increases with the increase in the PW. For the fault on the Flash interface buffer at P1 and P2, with each load of instructions block into the buffer being covered by the laser pulse, the number of faulted instructions increases by four. Fig. 8(a) illustrates the case in which the laser pulse has a PW long enough to cover the loads of (i_1, i_2, i_3, i_4), and (i_5, i_6, i_7, i_8), resulting in the fault of eight successive instructions.

For the faults on the AHB bus at positions P3 and P4, with each load of instructions from the Flash interface buffer to the AHB bus being covered by the laser pulse, the number of faulted instructions increases by two. Fig. 8(b) illustrates the case in which a laser pulse with a PW long enough to cover the loads of (i_1, i_2), (i_3, i_4), and (i_5, i_6), resulting in the fault of six successive instructions.

For the faults on the core pipeline at position P5 and P6, with each fetch (or execution) of the pipeline being cover by the pulse, the number of faulted instructions increases by one. Fig. 8(c) illustrates the case in which a laser pulse with a PW long enough to cover the fetch (execution) of (i_1), (i_2), (i_3), (i_4), (i_5), resulting in the fault of five successive instructions.

It can be seen that the increment of the number of faulted instructions is different depending on the laser shot position (or the part being faulted). With a same PW, the number of faulted instructions is different for each position.

For each position, we gradually increased the PW until the MCU stopped working to acknowledge the limitation of the number of successive faulted instructions. It is reported in Fig. 9. The experiments confirmed that it is possible to extend the number of faulted instructions by using a longer PW.

Fig. 9(a) shows the results obtained when faulting the Flash interface buffer at P1 and P2. At P1, up to five blocks of four instructions were overwritten by the replayed instructions corresponding to 20 instructions not being loaded into the Flash interface buffer. At P2, skips of up to 20 instructions in a row were achieved. For the Flash interface buffer, we noticed that the MCU stopped working as the number of faulted instructions reached 20. Fig. 9(b) reports the results obtained when faulting the AHB bus at positions P3 and P4. At P3, up to 55 blocks of two instructions were overwritten corresponding to 110 instructions not being loaded into the AHB bus (i.e.

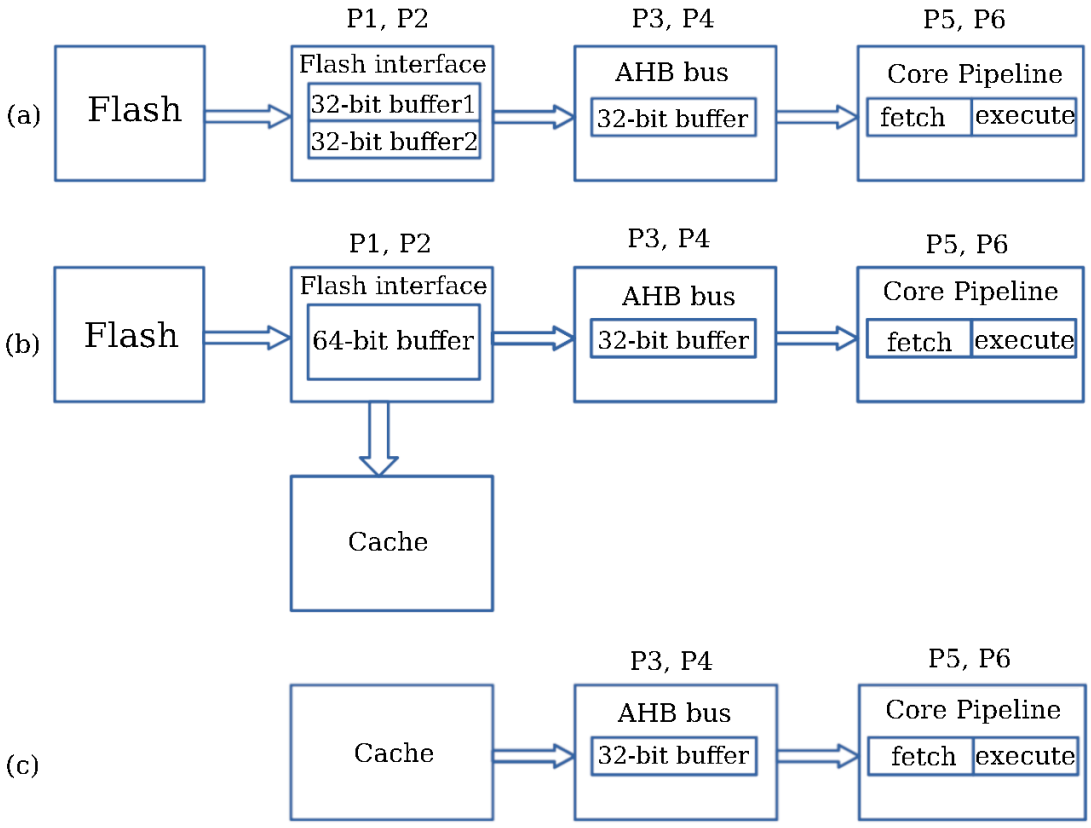


Fig. 7. Hypothesis on the core architecture based on fault behavior: (a) cache disabled, (b) cache enabled: cache miss, (c) cache enabled: cache hit

being overwritten by the same set of two instructions preceding the faulted ones). At P4, we also tested until 110 instructions were skipped in a row. After that the MCU stopped working.

Finally, Fig. 9(c) depicts the results obtained when faulting the core pipeline at positions P5 and P6. For both P5 and P6, we tested until 115 instructions were faulted in a row. After that the MCU stopped working.

Our experiments assess that in all the cases a large number of instructions can be faulted in a row. However for each position, there is a limitation on the number of faulted instructions, which is different for each position. This is probably because of the side effect of longer pulse and its different impact at each position. For one thing, longer pulse causing the MCU to stop working is maybe due to the side effects of the pulse, such as thermal effect. For the other thing, the difference in the maximum number of faulted instructions is maybe because the pulse side effect on each position is different. However, a great deal of effort in the future is still required to achieve a clear explanation for these differences.

It is worth mentioning that the change in the PW of the laser pulse has no impact on the fault model achieved at each position. This is different from the results achieved with EMFI in [9], in which different fault models were obtained as the EM pulses with different PW were used with a same probe position. In addition, as the PW was increased, a smaller laser power was needed to achieve a fault rate of 100%.

To sum up, the PW has a direct impact on the number of faulted instructions. Tens to more than a hundred of instructions can be faulted in a row by choosing a proper PW. This makes the corresponding fault models even more threatening.

V. IMPACT OF THE POWER

We also studied the impact of the laser power on the injected faults. The PW was set to 50 ns, the laser power was first set to 0.3 W, and then increased to 0.5 W, 0.7 W, 0.9 W, 1.1 W, 1.3 W, 1.5 W and 1.7 W. The MCU was configured to work in cache enabled mode. For each position, we measured the maximum fault rate obtained with each laser power. It is reported in Fig. 10 for the laser positions we studied. It can be seen that as the laser power increases, the fault rates increase accordingly. With a laser power of 0.3 W, no fault is observed. The laser power needed to induce faults on the Flash interface buffer is relatively smaller than at the AHB bus or the core pipeline. Starting from a laser power of 0.5 W, faults on the Flash buffer are observed. The replay fault rate at P1 is 60%, while the skip fault rate at P2 is about 10%. With the laser power set at 0.7 W, there is still no fault at P3 to P6, while the replay fault rate at P1 is 80%, and the skip fault rate at P2 is 30%. With the laser power set at 0.9 W, the fault rates at P1 and P2 can reach 100% and 90% respectively. Also, starting from a laser power of 0.9 W, faults at P3 to P6 start to appear with the following

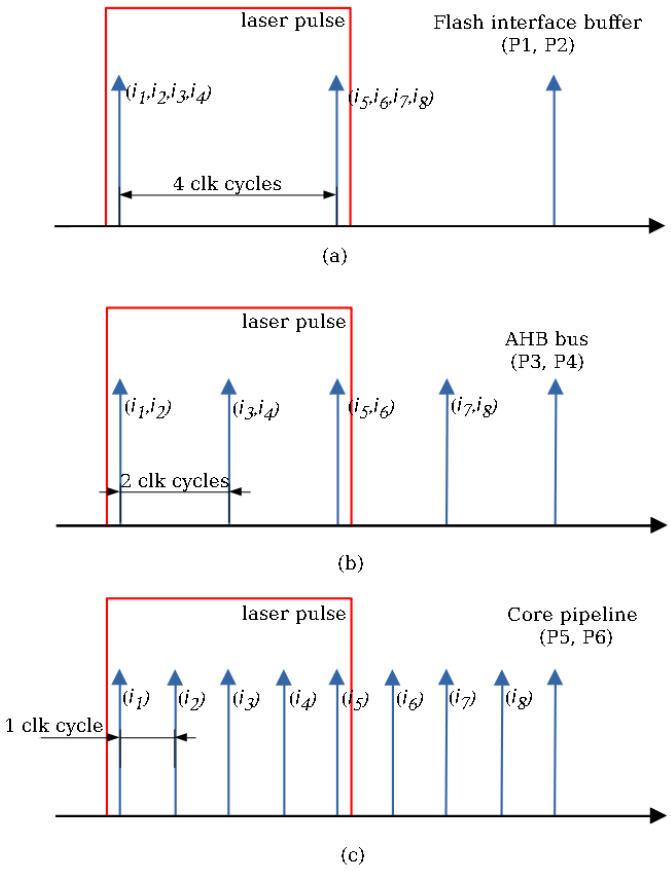


Fig. 8. Fault scenarios at different positions with a long pulse , (a) the Flash interface buffer: P1 and P2, (b) the AHB bus: P3 and P4, (c) the core pipeline: P5 and P6.

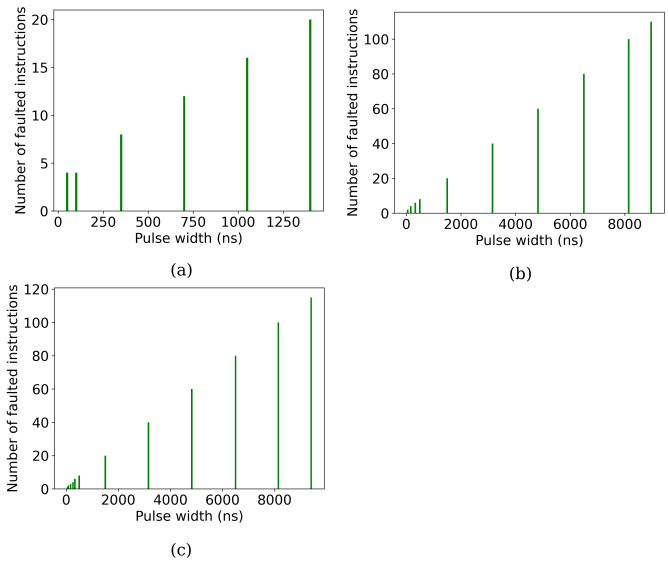


Fig. 9. Impact of the PW on the number of faulted instructions at different positions, (a) P1 and P2, (b) P3 and P4, (c) P5 and P6.

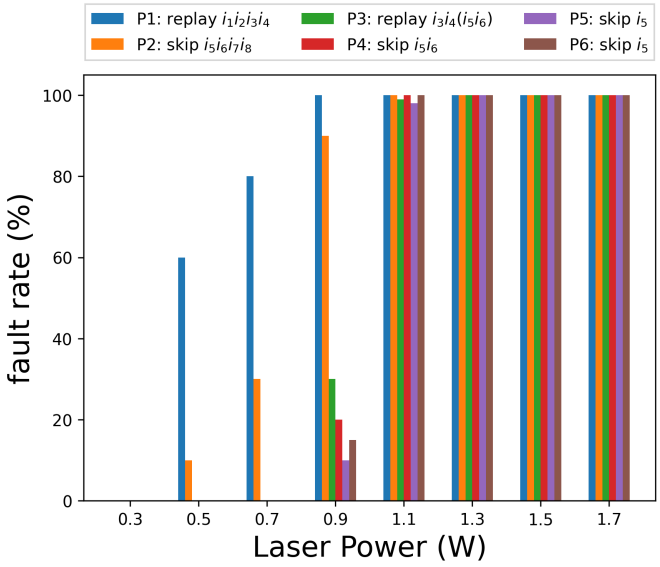


Fig. 10. Impact of the laser power on the fault rate

fault rates: replay of two instructions (P3) 30%, skip of two instructions (P4) 20%, skip of a single instruction (P5) 10%, skip of a single instruction (P6) 15%. With the laser power set at 1.1 `W` and higher, the fault rates mostly reach 100% at all the positions. It is also worth mentioning that as the laser power increases, the fault windows widen accordingly.

In short, the laser power has a direct impact on the fault rate obtained at each position. The Flash interface buffer seems to be more sensitive to the laser pulse than the AHB bus or the core pipeline. Smaller laser power is needed to induce fault on it as compared to the AHB bus or the core pipeline. With a proper laser power, fault rate of 100% can be achieved with all six positions. Whereas the change in the laser power has no effect on the number of faulted instructions.

VI. Fault at Bit Level Characterization

As mentioned previously, skip of instructions obtained at positions P2 and P4 were ascribed to laser-induced corruption of one or more bits of the instructions opcode while being loaded into either the Flash interface buffer or the AHB bus. To identify the fault model at bit level, specific test codes were used to identify whether the induced faults were bit-set, bit-reset, or both. First, to detect bit-set faults, the buffer was filled with all bits at `0`. This was done by using a test code consisting of successive iterations of the same instruction `lsl r0,r0,#0x00`, of which the opcode is `0x0000`. Similarly, bit-reset faults can be detected by filling the buffer with all bits at `1`. Because there is no instruction with such an opcode, the test code consisted of several times the same instruction `sub r7,r7,#0xff` of which the opcode is `0x3fff` or most of the bits are `1`.

The laser power was set to 1.5 `W`, and the PW to 50 ns. The MCU was configured to work in cache enable mode, meaning that the Flash interface buffer size is 64-bits and the HRDATA

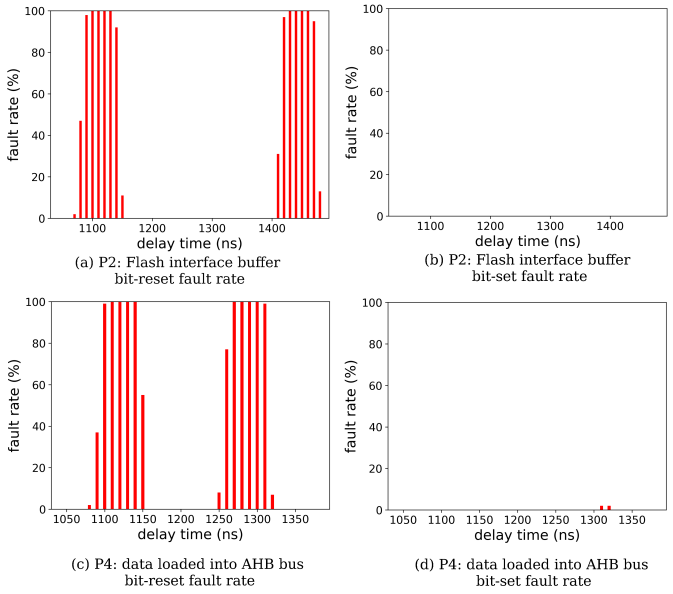


Fig. 11. Bit level characterization: Flash interface buffer (a) bit-reset fault rate, (b) bit-set fault rate, data loaded into AHB bus (c) bit-reset fault rate, (d) bit-set fault rate.

size is 32-bit. Fig. 11 reports the fault rates obtained for these two data buffers. In Fig. 11(b) and (d) corresponding to a test code with its instructions with their bits at `0` , almost no faults were induced. While in Fig. 11(a) and (c) corresponding to the test code with its instructions with most of their bits are `1`, many faults were observed (with a success rate of 100% for some timings. Through this test, we assume that at bit level, the LFI-induced faults are bit-reset rather than bit-set. This result is quite the same with the result obtained with EMFI in [9].

VII. Comparison of Different Skip Instruction Fault Models Obtained with LFI

Skip of instruction(s) is a very powerful fault model. It has drawn great attention of the adversaries because it is easy to use for further exploitation. As presented above, it is possible to achieve the skip fault models by faulting instructions at different stages from the Flash interface to the execution pipeline: (1) the Flash interface buffer at P2, (2) the data loaded into the AHB bus at P4, (3) the core pipeline fetch at P5, (4) the core pipeline execution at P6. However, there are some differences among these faults. In this section, we draw an in-depth analysis of the skip fault models obtained in this work.

The test code used is the same as shown in Fig. 2(a) except that (i_5, i_6, i_7, i_8) are replaced by load instructions: `ldr rx, \#address`. Notice that the instruction `ldr rx, \#address` has a execution time of two clock cycles. The MCU was configured to work in cache enabled mode. At positions P2, P4, P5 and P6, we were able to skip all four `ldr` instructions (the skips were confirmed by checking the register values) with a fault rate of 100%. Fig. 12 shows the signals taken from the oscilloscope, where the green waveform is an image of the laser pulse, the dark pink waveform provides the test code time window (i.e. its duration) which includes the four instructions, the brown waveform is the trigger signal used to synchronize the laser shot, and the dark green waveform is the pulse command. Notice that the MCU was configured to operate at 12 MHz, meaning one clock cycle is approximately 83.2 ns. The black vertical bars in Fig. 12(b), (c), (d), (e) represent the expected falling edge of the test code window in normal execution conditions.

Note that we used different laser PW during the reported experiments. Faulting the Flash interface buffer was accomplished with a PW of 50 ns which was enough to fault a block of four instructions. While a longer pulse was needed to fault the AHB bus or the core pipeline.

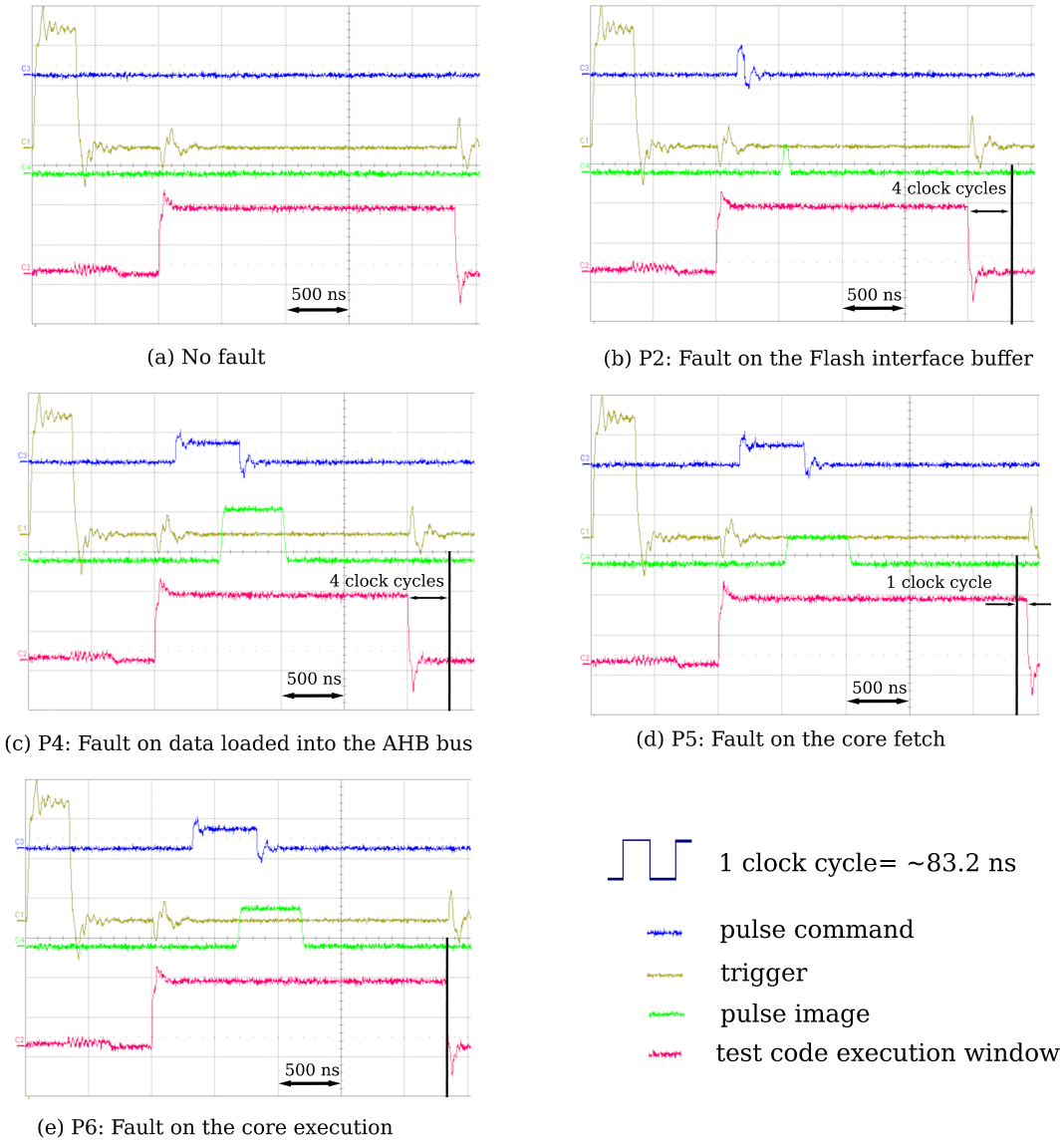
Fig. 12(b) corresponds to the skip of four instructions obtained from a short laser pulse (50 ns PW) targeting the Flash buffer (P2): the test code execution time is accordingly reduced by four clock cycles (4 $\times$83.2 ns) as four `ldr rx, \#address` instructions are replaced by four `nop` instructions (of which the execution time is one clock cycle). A longer laser pulse is used to skip the four load instructions in a row while targeting the AHB bus (location P4) as reported in Fig. 12(c). Similarly, the test code duration decreased by four clock cycles.

The same 500 ns laser PW was used to target the core fetch and execution stages (locations P5 and P6 respectively) with the intent to fault four instructions in a row. In both cases, the readback of the registers content showed that the load instructions were not executed, as if these instructions were effectively skipped. But as revealed in Fig. 12(d) and (e), the test code time window was slightly increased by one clock cycle when targeting the fetch stage and unmodified when targeting the execution stage. The load instructions were faulted and replaced by other (unknown) instructions having the execution time of two clock cycles (and possibly an instruction having the execution time of three clock cycles with the fault on the core fetch stage). This illustrates that the instruction skip fault model does not necessarily involve the replacement of the skipped instructions by `nop` instructions.

It is also worth mentioning that as compared to faults injected at other locations, the faults induced in the pipeline stages are more precise and flexible. Indeed, faults induced in the Flash interface buffer or in the AHB bus buffer encompass a block of at least several instructions (starting at two). Hence, faulting a single instruction is not achievable which reduces the ability of an attacker to tailor precise attacks. While faults induced in the pipeline allow to fault a single instruction (or a larger number with an elementary increment of one) as instructions are processed one by one in the pipeline stages.

VIII. Conclusion & Perspectives

In conclusions, by using LFI, we were able to fault the instructions executed by a 32-bit MCU as they traveled from the Flash interface to the execution pipeline. Instructions were found to be faulted at: (1) the Flash interface buffer, (2) data



(a) No fault

(b) P2: Fault on the Flash interface buffer

(c) P4: Fault on data loaded into the AHB bus

(d) P5: Fault on the core fetch

(e) P6: Fault on the core execution

Fig. 12. Comparison of multiple skip fault models obtained at different positions with images taken from oscilloscope, black vertical bar indicates the expected terminating time of test code window, one clock cycle is approximately 83.2 ns: (a) No fault, (b) fault on the Flash interface buffer at P2, (c) fault on data loaded in to the AHB bus at P4, (d) fault on the core pipeline fetch at P5, (e) fault on the core pipeline execution at P6.

loaded into the AHB bus, (3) the core pipeline fetch and execution stages. Six positions with different fault behaviors were obtained.

For the Flash interface buffer, replay and skip of instructions block fault models with a fault rate of 100% were obtained at positions P1 and P2, respectively. The block size is related to the buffer size which depends on the cache operation mode. As the cache is disabled, the two 32-bit buffers of the Flash interface are updated independently every four clock cycles with a phase shift of two clock cycles. Hence, these operations can be faulted independently (from one to several blocks of two instructions are faulted depending on the laser PW). Whereas in cache enabled mode, the 64 bits of the Flash interface buffer are updated simultaneously every four clock cycles. Block of four instructions can be faulted.

Instructions are loaded into the AHB bus buffer by blocks of 2×16-bit instructions (either in cache enabled or disabled mode). Two fault models of replay and skip of two instructions with a fault rate of 100% were obtained at the corresponding positions P3 and P4, respectively. The replay of instructions block at P1 and P3 is ascribed to laser-induced prevention of block data update, resulting in the replay of the previously stored block of data. While the skip of instructions block at P2 and P4 is due to laser-induced bit corruption (bit-reset) in the block data being updated. The fault on core pipeline happens with each single instruction. For the same instruction, there is a shift of one clock cycle in injection time of the faults between the fetch and the execution processes. Single instruction skip with a fault rate of 100% was achieved by faulting either the fetch or the execution.

The impact of laser parameters such as the PW and laser power on the induced faults were systematically studied. For all six positions, we were able to achieve from tens to more than one hundred of faulted successive instructions by increasing the PW. The laser power has a direct impact on the fault rate of the LFI-induced faults. The fault only occurs when the laser power reaches a certain value. The Flash interface buffer seems to be more sensitive to the laser pulse as the fault on it occurs with smaller powers. As the laser power increases, the fault rates increase accordingly.

We further used specific test codes with the opcodes having maximum bits of `1` and `0` to characterize the fault at the Flash interface buffer and the data loaded into the AHB bus at bit level. The fault model was identified to be bit-reset rather than bit-set.

In addition, we compared the skip fault models obtained at different position to each other by monitoring the width of the related signals. The skip obtained by faulting the Flash interface buffer and AHB bus data involves the replacement of instructions by `nop` operations, while the skip obtained by faulting the fetch and execution stages in the core pipeline involves the replacement of instructions by other (unknown) operations.

Our future work will be centering on the validation of the faults obtained in this work on other devices.

ACKNOWLEDGMENT

This work was partly funded by the SPARTA project, which has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement number 830892.

REFERENCES

[1] Josep Balasch, Benedikt Gierlichs, and Ingrid 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*, pages 105–114. IEEE, 2011.

[2] Alessandro Barenghi, Guido Bertoni, Emanuele Parrinello, and Gerardo Pelosi. Low voltage fault attacks on the rsa cryptosystem. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 23–31. IEEE, 2009.

[3] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, Ingrid Verbauwhede, Saki Osuka, Masahiro Kinugawa, Daisuke Fujimoto, and Yuichi Hayashi. Characterization of em faults on atmega328p. In *International Symposium on Electromagnetic Compatibility. IEEE*, 2019.

[4] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. *IACR Cryptology ePrint Archive*, 2018:1042, 2018.

[5] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Regis Leveugle, Paolo Maistri, et al. Laser fault injection at the cmos 28 nm technology node: an analysis of the fault model. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–6. IEEE, 2018.

[6] Jean-Max Dutertre, Timothé Riom, Olivier Potin, and Jean-Baptiste Rigaud. Experimental analysis of the laser-induced instruction skip fault model. In *Nordic Conference on Secure IT Systems*, pages 221–237. Springer, 2019.

[7] Microchip Technology Inc. SAM D21/DA1 Family. In *SAM D21/DA1 Family*.

[8] Vanthanh Khuat, Jean-Max Dutertre, and Jean-Luc Danger. Analysis of a laser-induced instructions replay fault model in a 32-bit microcontroller. In *2021 Euromicro DSD/SEAA 2021 Conference (acceptted)*, 2021.

[9] Vanthanh Khuat, Oualid Trabelsi, Laurent Sauvage, and Jean-Luc Danger. Multiple and reproducible fault models on micro-controller using electromagnetic fault injection. In *2021 Joint IEEE Internationnal Symposium on Electromagnetic Compatibility, Signal Power Intergrity, and EMC Europe (acceptted)*, 2021.

[10] Marc Lacruche, Nicolas Borrel, Clement Champeix, Cyril Roscian, Alexandre Sarafianos, Jean-Baptiste Rigaud, Jean-Max Dutertre, and Edith Kussener. Laser fault injection into sram cells: Picosecond versus nanosecond pulses. In *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*, pages 13–18. IEEE, 2015.

[11] Ronan Lashermes, Marie Paindavoine, Nadia El Mrabet, Jacques JA Fournier, and Louis Goubin. Practical validation of several fault attacks against the miller algorithm. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 115–122. IEEE, 2014.

[12] ARM Limited. Arm®v6-m architecture reference manual. In *ARM Limited*. ARM Limited, 2017.

[13] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88. IEEE, 2013.

[14] Lionel Riviere, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent 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)*, pages 62–67. IEEE, 2015.

[15] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault model analysis of laser-induced faults in sram memory cells. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 89–98. IEEE, 2013.

[16] Jörn-Marc Schmidt and Michael Hutter. *Optical and em fault-attacks on crt-based rsa: Concrete results*. na, 2007.

[17] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on aes: A threat in violet. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 13–22. IEEE, 2009.

[18] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical setup time violation attacks on aes. In *2008 Seventh European Dependable Computing Conference*, pages 91–96. IEEE, 2008.

[19] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *International workshop on cryptographic hardware and embedded systems*, pages 2–12. Springer, 2002.

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

[21] Joseph Yiu. The anatomy of the arm cortex-m0+ processor. 2016.

[22] Bilgiday Yuce, Nahid Farhady Ghalaty, Harika Santapuri, Chinmay Deshpande, Conor Patrick, and Patrick Schaumont. Software fault resistance is futile: Effective single-glitch attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 47–58. IEEE, 2016.