Peña-Fernández, M., et al. Microprocessor error diagnosis by trace monitoring under laser testing. In: *IEEE transactions on nuclear science, 68(8), Aug. 2021, Pp. 1651-1659*

DOI: https://doi.org/10.1109/TNS.2021.3067554

Microprocessor Error Diagnosis by Trace Monitoring under Laser Testing

M. Peña-Fernandez, A. Lindoso, L. Entrena, I. Lopes, V. Pouget

***Abstract*—This work explores the diagnosis capabilities of the enriched information provided by microprocessors trace subsystem combined with laser fault injection. Laser fault injection campaigns with delimited architectural regions have been accomplished on an ARM Cortex-A9 device. Experimental results demonstrate the capability of the presented technique to provide additional information of the various error mechanisms that can happen in a microprocessor. A comparison with radiation campaigns presented in previous work is also discussed, showing that laser fault injection results are in good agreement with neutron and proton radiation results.**

***Index Terms*— ARM, microprocessor trace, fault tolerance, error diagnosis, laser fault injection.**

I. Introduction

IN complex devices, such as microprocessors, the capabilities to understand fault effects and diagnose errors are crucial to design effective error mitigation solutions. In the particular case of Commercial Off-The-Shelf (COTS) devices, such capabilities are strongly needed to assess their use in radiation environments, cope with radiation effects and eventually meet dependability requirements in sensitive applications. Contrary to radiation-hardened devices, for which RHBP (Radiation-Hardening by Process) or RHBD (Radiation-Hardening by Design) techniques are systematically applied to all components of the circuit irrespective of its criticality, COTS devices generally require ad-hoc techniques that are specifically applied to each component or to mitigate particular types of errors. As radiation induced soft errors cannot generally be avoided for COTS devices, the goal is to identify the source of errors and provide for an appropriate recovery in each case.

COTS microprocessors are becoming increasingly attractive for space applications [1]. Compared to their radiation-hardened counterparts, COTS microprocessors provide higher performance, lower power consumption, lower cost and wider availability, yet they are susceptible to radiation effects. Therefore, they can be a good choice for low cost applications or less critical parts of missions, provided that radiation effects can be reduced to an acceptable level. To achieve this goal, we need to know how COTS microprocessors fail and where they fail. However, fault diagnosis is a difficult task because many of the internal components of a microprocessor are not directly observable. Radiation test experiments can be used to estimate the error cross-section, but they generally provide very limited information about the vulnerabilities of the microprocessor components [2].

Error diagnosis approaches are generally based on observing errors and performing some sort of cause-effect analysis to deduct where errors were originated [3], [4], [5], [6]. The capability of observing the internal state of a microprocessor is key for this purpose. In a previous work [7], trace monitoring was used to observe microprocessor behavior under proton and neutron irradiation, detect errors and collect information for diagnosis. This approach takes advantage of the trace infrastructures that are commonly included in modern microprocessors to support debugging and profiling.

In this work we extend the error diagnosis approach proposed in [7] by analyzing errors produced by laser fault injection. The use of laser can provide further insight into the diagnosis capabilities that can be achieved by trace monitoring. In addition, the capability of the laser to focus on specific components of the microprocessor, such as the caches or control logic, enables to categorize the type of errors that are produced in each of these components. Experimental results demonstrate that the effects of faults on these components are indeed quite different and have a distinctive pattern.

Trace information is very rich to perform error diagnosis. We show how particular errors can be related to particular code instructions or variables by analyzing the obtained trace information.

Pulsed lasers have been used for a long time by the radiation effects community [8]. In this work we seek to evaluate laser injected errors using the high diagnosis capabilities supported by the trace monitoring approach. Finally, laser scanning can also be used to obtain geometrical sensitivity mappings. With the use of trace monitoring, we can further relate the sensitivity maps to the types of errors.

The remaining of this paper is as follows. Section II summarizes related work. Section III describes the evaluation

This work has been supported in part by the Spanish Ministry of Science and Innovation under project PID2019-106455GB-C21 and by the Community of Madrid under grant IND2017/TIC-7776.

M. Peña-Fernández is with Arquimea Ingenieria SLU., Leganes, Madrid, Spain (email: mpena@arquimea.com).

A. Lindoso, L. Entrena and M. García-Valderas are with the Department of Electronic Technology, Universidad Carlos III de Madrid, Avda. Universidad 30, E-28911 Leganes, Madrid, Spain (e-mail: alindoso@ing.uc3m.es; entrena@ing.uc3m.es; mgvalder@ing.uc3m.es).

I. Lopes and V. Pouget are with the IES—UMR UM/CNRS 5214, Université de Montpellier, F-34097 Montpellier, France (e-mail: dacostalopes@ies.univ-montp2.fr; vincent.pouget@ies.univ-montp2.fr).

methodology than we have used in this work. Section IV shows the experimental results. Finally, section V presents the conclusions of this work.

II. Related work

Microprocessors have complex architectures that present a wide variety of erroneous behaviors when they are exposed to radiation. Radiation can affect any part of the architecture, but its effects deeply depend on the application in execution [9]. The executed application uses specific parts of the architecture within a predetermined execution order and timing, increasing with it the complexity of error diagnosis.

The analysis of errors in microprocessors is a challenging task. When execution takes place in harsh environments, we can easily observe the effects of errors, but it is difficult to determine the part of the architecture that was affected by ionized particles. Being able to correctly characterize the microprocessors error mechanisms can lead to the isolation of incorrect behavior and can reverberate in improvements for mitigation techniques.

The problem of fault diagnosis of microprocessor systems has been a matter of research since the beginning of the microprocessor era [10]. It has been addressed mainly for permanent errors, such as those produced by manufacturing defects or ageing. Approaches include Design for Test (DfT) techniques, such as scan-based BIST (Built-In Self Test) [11], Software-Based Self-Test [12], or a combination of both [13]. In complex circuits, BIST structures are often included on chip to support test functions and can also be used in the field. DfT techniques usually provide high fault coverage but have to be highly compacted to reduce the impact on hardware area. Software-Based Self-Test execute special codes that check critical parts of the system, such as the memories or the register file. They are often used in critical systems to ensure integrity when the system is put in operation or before carrying out a critical task.

Self-testing codes are specialized for each module and are intended to be executed on-line, but in a non-functional mode. This approach has also been used to analyze the susceptibility of different parts of a system to radiation-induced soft errors [14]. To this purpose, a special code is used to continuously stimulate and inspect some particular modules, such as memories, internal registers or pins, when the circuit is under the beam. Nevertheless, it is not generally possible to ascertain the origin of errors when the test is performed with a broad beam because of the random nature of the ion strikes and the limited observability of the system. To control fault locations, pulsed laser [8], heavy-ion micro-beam [15] and neutron micro-beam [16] have been proposed as alternatives to broad-beam ion tests.

In the literature we can find approaches that rely on intensive fault injection to determine the relationship between errors and the architectural parts affected by radiation. The architectural vulnerability factor (AVF) of the architectural elements can be determined [17], which represents the probability of occurrence of a system error given a bit flip in a storage cell. With massive fault injection campaigns, errors can be classified and linked to the architectural location of the injected fault. This approach has been used in [18] and [19]. An extended approach is used in [20] to analyze the AVF due Multi-Bit Upsets (MBUs). The recent work [26] proposes to overcome the limitations of AVF combining simulation campaigns with radiation results by means of a mathematical model which provides a reliability figure for the different blocks of the processor. In the case of microprocessors, this kind of approaches can provide incorrect results because the very same observed error can be due to misbehaviors in different parts of the architecture. For instance, microprocessor hangs or crashes are commonly observed when a microprocessor is exposed to radiation, but this type of error can be due to particle impacts in several elements of the architecture.

Intensive fault injection campaigns can be used to create fault dictionaries which could be used for diagnosing errors when the circuit is exposed to radiation. This approach is used in [3] and [4]. However again, we face the same problem as exposed before, for several architectural components misbehavior can match with the observed error.

In [21], a methodology is developed to analyze actual field data collected from error logs of information computing systems. The subsystem where errors occurred could be localized with the help of the Machine Check Architecture (MCA). The MCA is an internal subsystem included in the studied microprocessor which can detect and capture errors. With the collected information, it was possible to point out the possible cause of some soft errors.

Artificial Intelligence (AI) could be also used to solve the relationship between observed errors and architectural damaged locations [22]. AI techniques that have been used for fault diagnosis include expert systems, neural networks, Petri nets and fuzzy logic.

In general, all possible solutions increase their matching capabilities when the amount of information gathered increases. For that reason, new methods to obtain information may improve diagnosis results.

Today, the trace subsystem is part of the architecture of most microprocessors. It is used for debugging purposes when the application is under development and is no longer needed during regular execution. In [23], an IP is presented that gathers trace information and observes a high-end microprocessor (ARM Cortex-A9) behavior. In [7], this IP increases its observation capabilities and is used for diagnosis purposes under proton and neutron irradiation.

In this work we combine the diagnosis capabilities of the information provided by the microprocessor's trace subsystem with a laser fault injection experiment, which provides accurate fault location. With the combination of both laser injection and enriched trace information observation we seek to improve the understanding of errors mechanisms in microprocessors.

III. Evaluation Methodology

A. Device under test

The Device Under Test (DUT) used in the experiments was a Xilinx Zynq XC7Z030-1SBG485, a monolithic programmable System-on-Chip (SoC) manufactured in 28nm planar bulk CMOS technology. The DUT embeds a dual-core ARM Cortex A9 processor, an SRAM-based FPGA, and many peripherals. The DUT selection was driven by the fact that this

Table I. Zones used for laser fault injection

Zone	Description	Injection method	Width (µm)	Height (µm)	Tested on CUT1	Tested on CUT2	Tested on CUT3
Z1	L1 Data cache – CPU0	Sequential	210	390	Yes	No	No
Z2.1	L1 instruction cache1–CPU0		244	164	Yes	No	No
Z2.2	L1 instruction cache2–CPU0		210	425	Yes	No	No
Z3	Registers (unidentified)		230	215	Yes	Yes	Yes
Z4.1	L2 Cache - 1	Random	863	883	Yes	No	No
Z4.2	L2 Cache - 2		828	1424	Yes	No	No
Z5.1	OCM 1		803	416	No	No	Yes
Z5.2	OCM 2		802	416	No	No	Yes

particular device is packaged without plastic cover, so it can be used for laser testing with no package processing. During the tests, the DUT was mounted on a PicoZed development board, and connections were performed through the PicoZed FMC carrier board. Both boards are manufactured by AVNET and are commercially available.

B. Software case study

A matrix multiplication benchmark was executed on ARM Cortex-A9 core #0 of the XC7Z030 device, running at the nominal 650 MHz clock frequency. The benchmark enters an infinite loop performing two consecutive phases: firstly, result matrix R is computed by multiplying two input matrices A and B; secondly, matrix R correctness is checked. The correctness of result matrix is checked by comparing it with a previously calculated golden reference, and additionally by performing consistency checks on triplicated data. We use triplicated data to emulate the typical approach for software hardening in a realistic application, in which a golden reference is not commonly available. Triplication allows us to identify and determine the faulty value and eventually the faulty bit if the error only affects one of the three copies. In addition, golden reference is used only to cover the cases in which more than one data copy is corrupted.

The benchmark used 32x32 matrix dimension composed by 32-bit integer elements. We tested three different software versions:

- CUT1: All caches were enabled during execution. This can be considered as the baseline benchmark.
- CUT2: All caches were disabled.
- CUT3: All caches were disabled. Program data and stack were stored on On-Chip Memory (OCM) instead of DRAM.

Maximum optimization effort was used for the compilation of all software versions. This choice is frequently found in deployed applications and can be considered as a worst-case scenario for error diagnosis. With this configuration, the compiler modifies the structure of the code to optimize execution performance, so it can be more difficult to stablish a cause-effect relation. In addition, it makes a higher use of resources, particularly all core registers, resulting in a more intensive exercise of the architecture.

C. Trace monitoring approach

This work uses an external Trace Monitor IP based on [7] that observes the trace information provided by the trace subsystem. The IP is located in the programmable logic (PL) and connected to the processor by EMIO (Extended Multiplexed I/Os) interface available on the device. The IP can be configured as a peripheral through AXI bus, and it observes the microprocessor through the trace interface.

We leverage the CoreSight subsystem, which is available on almost every ARM processor or microcontroller nowadays, and particularly on the Zynq device. CoreSight is a set of components provided by ARM to support debug and trace purposes. Among the available components on Zynq device, we have selected and configured two trace subsystem cells: the Program Trace Macrocell (PTM) [24] and the Instrumentation Trace Macrocell (ITM) [25]. These two cells can send information about the executed application and the microprocessor behavior through the trace interface to the Trace Monitor IP located in PL.

The PTM generates information about processor behavior by providing the Program Counter (PC) address values during execution. To optimize bandwidth, the PTM only generates PC address information related with the points on the execution where the flow can be altered, such as branches or exceptions. PTM information is used to track the correctness of the execution flow. The Trace Monitor IP triggers an error signal if the microprocessor reaches an instruction outside of the allowed address range. Data errors are tracked thanks to the information provided by the ITM. The ITM can export through the trace interface any 32-bit value from the program data just by writing it from the software in the ITM stimulus registers. Triplicated data are sent through the trace interface for the IP to check data consistency. When a data error is found, the IP triggers an error signal.

The trace information provided by both ITM and PTM is monitored on-line by the Trace Monitor IP and stored in two buffers to enable further analysis. One buffer is the Embedded Trace Buffer, which is part of the CoreSight subsystem, and the other is a custom design implemented in programmable logic. When an error is detected, the buffer contents are exported and the system also collects information about the execution context, namely relevant microprocessor memory contents and stack state. Error data logs are sent to the host through a serial communication port to be analyzed afterwards. The analysis can identify wrong addresses and wrong data by inspecting the last executed instructions before the error was detected. The trace information can eventually be used to reproduce the

execution in order to diagnose the error.

D. Laser fault injection

Laser testing was performed at the laser facility of the University of Montpellier, using single-photon absorption. The laser wavelength is 1064 nm with a pulse duration of 30 ps. The beam is focused down to spot size of 1.0µm through the backside of the 700µm thick substrate by using a 100X lens. The laser energy is set to 300pJ, approximately twice above the threshold to generate single bit flips. The setup of the DUT under the laser beam is presented on Fig. 1, in which both the test board and the laser-focusing lenses can be observed.

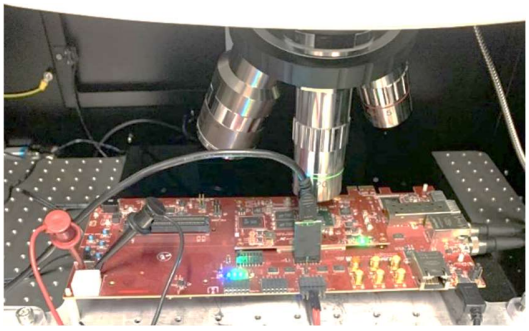

Fig. 1. Experimental setup detail

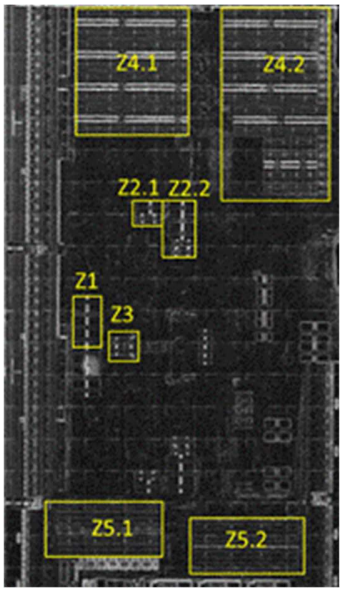

Fig. 2. Infrared microphotograph of the processing system of the DUT showing the fault injection zones.

The selected zones to be injected are defined in Table I and their localizations in the layout of the device are shown in Fig. 2. These zones cover the L2 cache, OCM, L1 instruction and data caches and an unidentified area of registers related with the CPU #0 core of the DUT. For each software version, we only tested the resources in use. For example, we avoided injecting in a cache when the executed code was not using it. The exception for this rule was the Z3 region because the resource associated to that zone is unidentified, so we injected it in all versions.

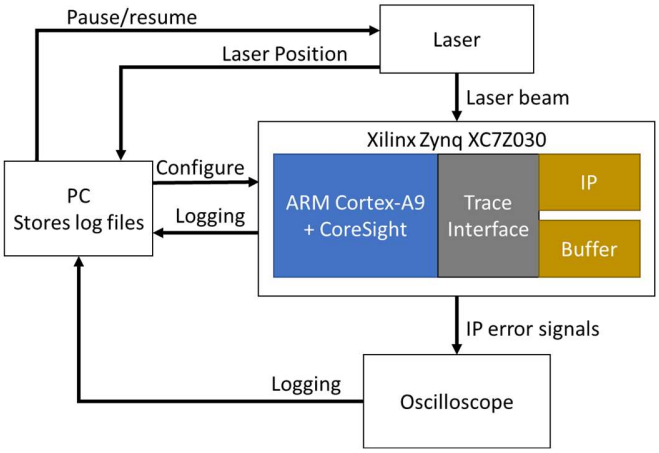

Fig. 3. Block diagram of the experiment

Laser scanning is performed repeatedly on each zone with a step size of 4µm on X and Y axes. Small regions, Z1 to Z3, were entirely scanned in a sequential manner injecting faults in each step. For larger regions, Z4 and Z5, injection was performed in random locations using a bouncing ball scheme. In any case, laser scanning and triggering is asynchronous with the benchmark loop execution, providing randomness in the fault injection. After each laser pulse, the error signal from the Trace Monitor IP is sampled by an oscilloscope to detect errors, while the benchmark output by the serial port is continuously logged. Scans are automatically paused while reconfiguring the DUT in the case of any error. A block diagram of the experiment is displayed in Fig 3.

E. Relation to previous work

In this work, we extend diagnosis experiments conducted under proton and neutron irradiation, presented in [7], by testing a similar infrastructure under laser fault injection. The experiment has been designed to maximize the similarities between both tests in order to compare results. For this purpose, the selected software application is the same in both works and code versions are equivalent. The IP core used to detect faults through the trace interface and the detection and diagnosis criteria are also the same. The tested devices belong to the same Xilinx Zynq family in both cases. However, they are not exactly the same. While an XC7Z010 device is used in [7], an XC7Z030 is used in this work. This difference is motivated by the fact that the ZC7Z030 device can be obtained in a package without plastic cover, which is more convenient for laser testing. Nevertheless, the main architectural difference between both chips resides in the size of the programmable logic, but the dual-core processor, which is actually the tested region of the chip, is exactly the same.

IV. Experimental Results and Discussion

Fault injection campaigns have been accomplished in all zones described in section III.D. We have defined the following error categories to classify observed errors:

- Prefetch Abort (PA): Exception caused by a forbidden

memory access when an instruction is fetched.
- Data Abort (DA): Exception caused by a forbidden data access attempt.
- Undefined Instruction (UI): Unrecognized instruction code.
- Invalid PC (IPC): The PC address got a value which was far outside the expected range.
- Result total errors (R): Data stored in matrix R is wrong. This category is divided in two subcategories attending the error multiplicity:
  - Result few errors (Rf): Data stored in matrix R is wrong in less than 25 matrix positions.
  - Result many errors (Rm): Data stored in matrix R is wrong in more than 25 matrix positions.
- Timeout (To): Processor communications timed out.
- Comm. (C): Processor communications were corrupted. Underlying errors, if any, cannot be classified.

A total number of 20,776 faults were injected in Z1 (L1 Data Cache) and 372 errors (1.79%) were obtained using CUT1. This relatively high error rate can be associated to an intensive use of this memory. Matrices are bigger than the cache: each matrix has 32x32 elements of 32 bits, and 9 matrices (3 copies of each matrix A, B and R) are used, giving a total figure of 36kB, while the available L1 cache is only 32kB. Thus, L1 Data Cache is fully used in each iteration of the benchmark. Injected faults in Z1 give more than 99% of errors in the results (R) as depicted in Fig. 4 for 372 errors in CUT1. More than 77% of errors affected only few elements in the matrix while around 22% affected more than 25 elements. The last case is probably related to a value or a pointer stored in the cache that got corrupted and it was used afterwards in many calculations in the benchmark. Errors in few elements may be related to three phenomena: 1) a fault in an input matrix value that was refreshed before affecting many values in the result; 2) a cache corruption in a result matrix value; or 3) an error in the cache controller returning a wrong value, but the cached value remained untouched.

L1 Data Cache results are in good agreement with those presented in [7]. In that work, several versions of the same benchmark were irradiated with different configurations. Particularly, the difference of the version with only L1 Data Cache enabled compared to the version with no cache enabled was mainly related to data errors.

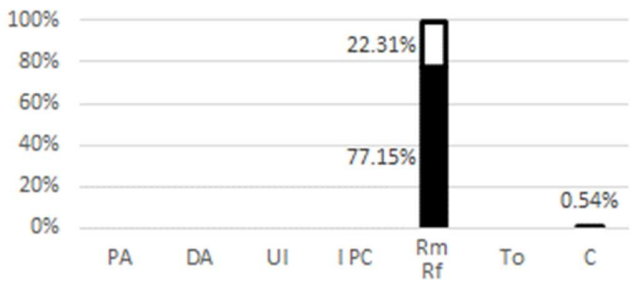

Fig. 4. Error histogram for Z1 (CUT1) - L1 Data Cache

In the case of Z2 (L1 Instruction Cache), 27,832 faults were injected, obtaining 82 errors (0.29%) using CUT1. The code size for the used application is low so just a small part of the L1 Instruction Cache is used, giving a lower error rate. Z2 error distribution is clearly dominated by errors in which it is not possible to continue with execution, also known as functional interrupt errors. These errors represent 93.90% of the total, and involve exception, timeout, communication and invalid PC errors, which are likely produced by the corruption of the benchmark code. The contribution of each type of error is shown in Fig.5 for the 82 errors observed in CUT1. In particular, communication errors may also be related to the corruption of the code associated to the serial port, which typically produced unexpected and eventually extremely long messages. All errors categorized as data errors in the result matrix are related to many wrong values. They have been probably produced due to a faulty operation code or an incorrect execution flow which did not produce a functional interrupt but affected the computations in a dramatic manner.

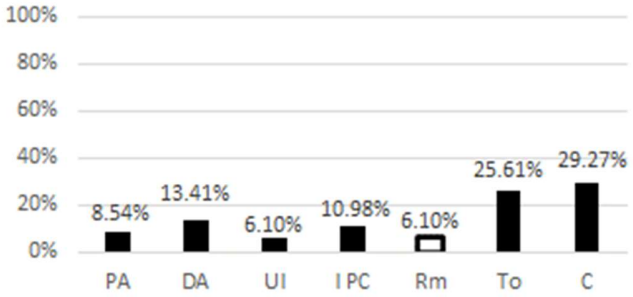

Fig. 5. Error histogram for Z2 (CUT1) - L1 Instruction Cache

L1 Instruction Cache contribution to errors was also evaluated under proton and neutron irradiation in [7] and it was found to be an increment of functional interrupt errors between the version with all caches disabled and the version with only L1 Instruction Cache enabled. This tendency is confirmed in the present work.

Regarding Z4 (L2 Cache), we injected 102,681 faults and obtained 119 errors (0.12%) using CUT1. The size of the L2 cache is 512kB, much bigger than the data used by the application, resulting in a low error rate as most of L2 Cache area is not used by the application. Every error found in Z4 was a Data Abort as indicated in Fig. 6 for 119 errors in CUT1. It is known that L2 Cache is used to store data values so it could be expected to produce also Result errors, but it is not the case. Data Abort errors occur mainly when there is a memory access to a memory location which is restricted by the Memory Management Unit (MMU). Injected faults are probably interfering with the address translation mechanism of the cache, resulting in a wrong memory access. We suspect that this mechanism is more sensitive to faults than the cache memory cells themselves, so every error in cache contents ends up masked by an earlier error in the cache control circuitry. Because of the big size of the L2 Cache, it was not completely injected, and only some points were randomly chosen for

injection using a bouncing ball scheme. Once the number of observed errors was enough to obtain statistically significant results, we stopped injection in this area. The extremely high incidence of Data Abort errors in L2 cache was also found in the radiation experiments [7] and is in good agreement with the results in this work.

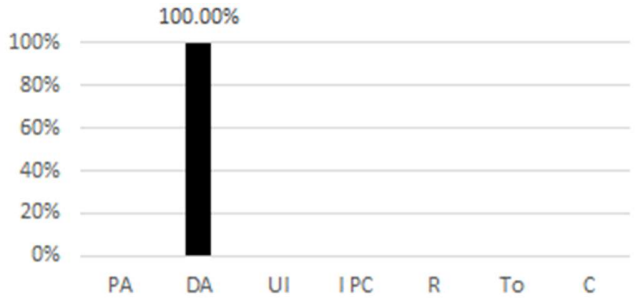

Fig. 6. Error histogram for Z4 (CUT1) - L2 Cache

We injected 41312 faults in Z5 (OCM), obtaining 143 errors (0.35%) using CUT3. Fig. 7 contains the error distribution of Z5 (OCM) for 143 errors in CUT3. It is important to note that in CUT3, OCM is only storing program data and stack, which, compared to the 256kB size of the OCM, turns out that much of it is unused, resulting in a low error rate. Every observed error in this zone was a data error in the Result matrix. Errors involving few wrong matrix positions are probably related with a fault injected into the result matrix directly. However, the cases where many faulty values are found can be related with an erroneous input value, or an error in a pointer stored in the stack. This is an expected behavior, because only data and stack are stored in OCM. Obtained results are in line with those obtained under radiation in [7] using a similar benchmark.

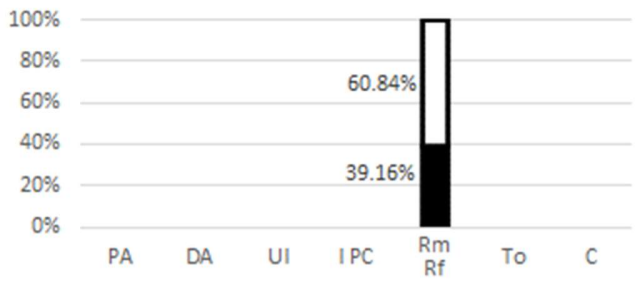

Fig. 7. Error histogram for Z5 (CUT3) - OCM

In the case of Unidentified Registers area, Z3, we do not know which resource is being injected, so we try to deduct it by exercising the architecture with all versions of the application. We injected 4,159 faults for CUT1, obtaining 61 errors (1.47%), which means that this resource may be intensively used by the benchmark. In addition, 3,808 faults were injected in Z3 for CUT2, and 5,715 faults for CUT3, obtaining 0 errors (0%) in both cases. When injecting faults in the Z3 area using CUT1, more than 81% of the obtained errors are related with data, although some Data Abort exception errors are also observed. Fig. 8 summarizes the error distribution in this zone for the three versions of the benchmark. However, Z3 area only shows sensitivity to errors when executing CUT1 (caches enabled), obtaining 61 errors, and no error was observed when caches were disabled (CUT2 and CUT3). Considering this behavior, we can deduce that this zone is probably related in some way to the cache operation, so in the case that caches are not used, this zone is unused as well. Geometrically, this zone is very close to L1 Data Cache and presents a similar error rate, which reinforces the deduction that it is related with it. Observed error distribution in this zone shows a high number of Result errors (similar to L1 Data Cache error distribution) but also a moderate number of Data Abort errors. We hypothesize that this region could be part of the L1 Data Cache controller as it is only used when the L1 Data Cache is used and if a fault is injected in it, it could become either a Result error or a Data Abort depending on whether the fault affects cache memory cells or the address translation mechanism, respectively.

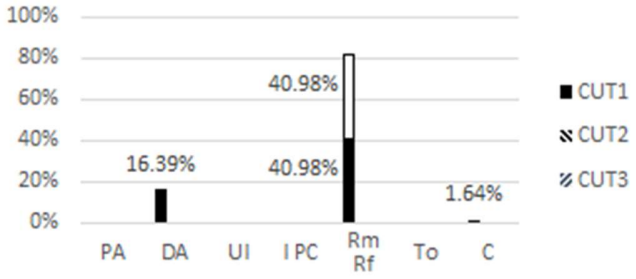

Fig. 8. Error histogram for Z3 - unidentified registers region

Obtained trace information is very rich to perform error diagnosis. In the aforementioned cases, the majority of errors are similar, so we categorize them in predefined error categories for the sake of brevity. However, further information can be extracted by carefully analyzing the collected evidences about the errors.

As an example, we analyze in detail one of the Invalid PC errors on L1 Instruction Cache. From the trace information we can obtain the PC values before the error. Table II shows the values for the PC in the moment of the error, normalized as t, and for two instants earlier, namely t-1 and t-2. It is important to note that the trace only exports PC values in points where the program flow can be changed (mainly branches and exceptions). By comparing the PC progression with the disassembled code depicted in Table III we can see that the error is a bit-flip in an instruction code provoking a change of `0x00100000` in the offset of the `beq` instruction located at address `0x00130240.` This change results in an instruction code of `0x0a100013` instead of `0x0a000013,` causing a branch to `0x00230294` instead of `0x00130294`.

Another example of trace diagnosis capabilities is that an out-of-range loop index was found among the Timeout errors on L1 Instruction Cache. In four error evidences, it was found that the matrix multiplication loop index was out of the 0-31 range, causing an excessive execution time leading to a timeout error.

We could observe this situation because we inserted specific instructions in the code for the ITM to export the loop index value through the trace.

TABLE II
PC PROGRESSION TO INVALID VALUE

t-2	t-1	t
0x00130244	0x00130234	0x00230294

TABLE III
DETAILED ADDRESS CONTENTS

Address	Content	Instruction	
0x00130230	ea000003	b	130244
0x00130234	e2822010	add	r2, r2, #16
0x00130238	e28cc010	add	ip, ip, #16
0x0013023C	e1520004	cmp	r2, r4
0x00130240	**0a000013**	**beq**	**130294**
0x00130244	e992000a	ldmib	r2, {r1, r3}
0x00130248	e592000c	ldr	r0, [r2, #12]
0x0013024C	e5871000	str	r1, [r7]
0x00130250	e1530000	cmp	r3,r0
0x00130254	01510003	cmpeq	r1, r3
0x00130258	e5863000	str	r3, [r6]
0x0013025C	13a0e001	movne	lr, #1
0x00130260	03a03001	moveq	r3, #1
0x00130264	13a03000	movne	r3, #0
0x00130268	e5850000	str	r0, [r5]
0x0013026C	11a0800e	movne	r8, lr
0x00130270	1affffef	bne	130234
0x00130274	e59c0004	ldr	r0, [ip, #4]

Errors on matrix data can also be analyzed in detail. When injecting L1 Data Cache and OCM, errors in a single value, including bit-flips, can be identified by comparing triplicated data sent through the trace. Such errors may be associated to a fault injected on a stored result value. However, we found at least two cases in which none of the three copies of a value matched when injecting L1 Data Cache and one more in OCM. By comparing the downloaded memory contents in the moment of those errors with the golden reference we obtained that in one case (L1 Data Cache) there was one correct value and two incorrect values while in the other two cases none of the value copies were correct. As demonstrated, when introducing the golden reference, we can refine the case granularity and discover more cases. This is particularly relevant attending to results obtained for Z3 region. In seven cases, some of the matrix redundant values were equal, but wrong, in all the copies at the same time. This case is unique for Z3 so it could allow to link similar errors to a fault within this region. Additionally, at least two cases in which two redundant values were equal, but wrong, were found on L1 Data Cache and Z3 regions.

The trace information about errors can also be used to obtain not only particular-case results, as in the last two examples, but also to extract error tendencies. As we have a continuous stream of trace information, we can know what happened before an error. In Fig. 9, it is presented the last instruction before 119 Data Abort errors related to L2 Cache when executing CUT1. It can be found that some addresses are much more prone to Data Abort error than others. In fact, the instruction at address `0x00130164` accumulates almost 50% of error incidence. Attending to such information, a software designer could put more effort to harden the code in the most problematic parts.

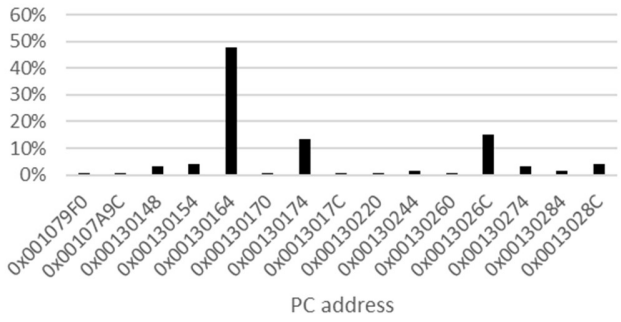

Fig. 9. Last instructions before Data Aborts related to L2 Cache (CUT1)

Results shown in this work are in good agreement with those obtained in previous radiation experiments [7]. Nevertheless, it must be noted that radiation testing and laser injection are quite different, so it is important to be careful when comparing results from both types of experiments. In radiation testing, it is possible to enable or disable some resources to isolate the contribution of each one to the final error distribution as we did in [7]. However, there are other resources, such as the registers, the pipeline, the arithmetic unit or the processor core itself, that cannot be disabled, because they are needed for the device to perform properly. Thus, radiation testing will always provide a result of the combined contribution of each resource to the total number of errors, given by the sensitivity of each resource. In contrast, laser testing can completely focus on specific resources and provide error results which can be closely related to them. We can conclude that the similarity of the results between radiation testing and laser scanning will be affected by the sensitivity of the resources under test. In the radiation campaigns performed in [7], L2 Cache, OCM, L1 Instruction Cache and L1 Data Cache were found to have a high sensitivity as they incremented the cross section by one to four orders of magnitude depending on the case. That high error sensitivity is probably a major reason for the agreements found in the results of both works.

Laser scanning allows us to plot error distribution on an XY graph, so it is possible to evaluate if error occurrence follows any recognizable pattern. Fig. 10 show the geometrical distribution for some of the previously presented errors, namely for L1 Data and Instruction caches for CUT1, while Fig. 11 shows the geometrical distribution by category for the Z3 region also for CUT1.

It is remarkable that in the case of L1 Data Cache, Fig. 10 a), errors are spread across the entire studied area so no correlation between the layout geometry and error appearance can be inferred. Matrices are bigger than the cache, so the cache is entirely used and often refreshed. Consequently, errors are expected to have a similar effect on execution in any position.

The case of L1 Instruction Cache is the opposite of the L1

Data Cache, as the executed code is very short and fits completely in the instruction cache. It can be observed in Fig. 10 b) that only some parts of the memory are producing errors. Particularly, there are many errors accumulated in a horizontal line at -1530µm.

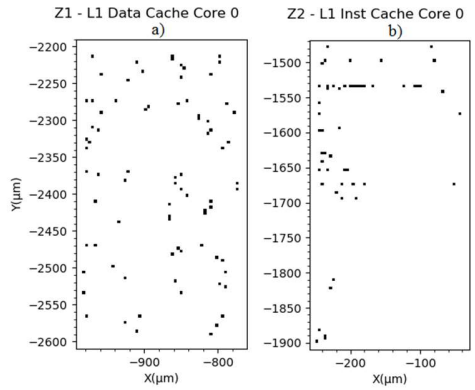

Fig. 10. Error geometrical distribution (CUT1)

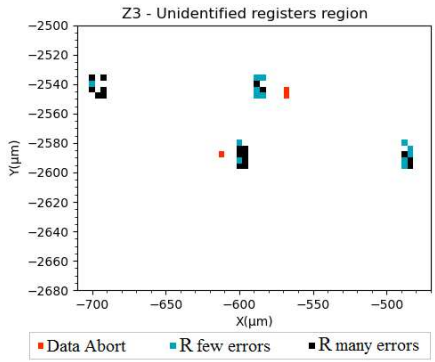

Fig. 11. Error geometrical distribution by category (CUT1)

The geometrical error distribution in Z3 region (unidentified registers) presents very sharply delimited error regions. To clarify the impact of each area in the error count, we take advantage of the capability of this method to generate sensitivity maps by correlating error types with the geometry of the device. A colored plot of the error geometrical distribution for Z3 zone is presented in Fig. 11. We distinguish in it 3 types of errors: Data Abort exceptions, Result few errors and Result many errors. It can be clearly observed that there are two zones contributing only to Data Abort errors and four zones contributing to errors in result matrix, in which the geometrical distribution of few errors and many errors on result matrix can be discriminated.

V. Conclusions

This work demonstrates the use of trace information to diagnose laser-induced errors. Diagnosis information obtained from the memory and the trace interface allows us to classify the observed errors in several categories and enables a rich analysis. By correlating the laser position with diagnosis information, we obtained that different error types are found depending on the injected area. Moreover, their geometrical and statistical distribution has a highly distinctive pattern, like a fingerprint.

Obtained diagnosis results allowed to identify functional regions on a Cortex-A9 processor thanks to the observed error statistical distribution. It must be pointed out that this information is not generally available for COTS and we are able to obtain it by combining accurate laser fault injection and intensive trace information observation.

Moreover, the presented approach has been demonstrated to be highly accurate and allow to identify the faulty bit on detected control-flow errors.

Laser fault injection results are in good agreement with radiation test results and confirm some of the hypotheses made in previous work [7]. By combining both works, we demonstrate that the proposed technique can effectively identify which processor resource has the highest sensitivity to each error type.

References

[1] M. Pignol, "COTS-based applications in space avionics," 2010 Design, Automation & Test in Europe Conf. (DATE 2010), pp. 1213-1219, Dresden, 2010.

[2] H. Quinn, T. Fairbanks, J. L. Tripp, G. Duran and B. Lopez, "Single-Event Effects in Low-Cost, Low-Power Microprocessors," 2014 IEEE Radiation Effects Data Workshop (REDW), pp. 1-9, 2014.

[3] S. Wei, F. Tongshun, and D. Mingfang. "Research for digital circuit fault testing and diagnosis techniques". Proc. Intl. Conf. on Test and Measurement, vol. 1, pp. 330-333, Dec. 2009.

[4] F. A. Bower, D. J. Sorin, and S. Ozev. "Online diagnosis of hard faults in microprocessors". ACM Trans. on Architecture and Code Optimization (TACO), vol. 4, no. 2, Article 8, June 2007.

[5] J. M. Mogollon, J. Napoles, H. Guzman-Miranda, and M. A. Aguirre. "Real Time SEU Detection and Diagnosis for Safety or Mission-Critical ICs Using HASH Library-Based Fault Dictionaries". Proc. RADECS, paper J-3, pp.705-710, Sept. 2011.

[6] J. M. Mogollon, J. Napoles, H. Guzman-Miranda, and M. A. Aguirre. "Metrics for the Measurement of the Quality of Stimuli in Radiation Testing Using Fast Hardware Emulation". IEEE Trans. on Nuclear Science, vol. 60, no. 4, pp. 2456-2460, Aug. 2013.

[7] M. Peña-Fernandez, A. Lindoso, L. Entrena and M. Garcia-Valderas, "The Use of Microprocessor Trace Infrastructures for Radiation-Induced Fault Diagnosis," IEEE Trans. on Nuclear Science, vol. 67, no. 1, pp. 126-134, Jan. 2020.

[8] S. P. Buchner, F. Miller, V. Pouget, and D. P. McMorrow. "Pulsed-Laser Testing for Single-Event Effects Investigations". IEEE Trans. on Nuclear Science, vol. 60, no. 3, pp. 1852–1875, June 2013.

[9] L. Entrena, A. Lindoso, M. Garcia-Valderas, M. Portela and C. Lopez-Ongil, "Analysis of SET Effects in a PIC Microprocessor for Selective Hardening", IEEE Trans on Nuclear Science, vol. 58, no. 3, pp. 1078-1085, Feb. 2011.

[10] V. P. Srini, "Special Feature: Fault Diagnosis of Microprocessor Systems," Computer, vol. 10, no. 1, pp. 60-65, Jan. 1977.

[11] M. Elm and H. Wunderlich, "BISD: Scan-based Built-In self-diagnosis," 2010 Design, Automation & Test in Europe Conf. (DATE 2010), pp. 1243-1248, 2010.

[12] P. Bernardi, R. Cantoro, S. De Luca, E. Sanchez, A. Sansonetti and G. Squillero, "Software-Based Self-Test Techniques for Dual-Issue Embedded Processors," IEEE Trans. on Emerging Topics in Computing, vol. 8, no. 2, pp. 464-477, April-June 2020.

[13] M. Ulbricht, M. Schölzel, T. Koal and H. T. Vierhaus, "A new hierarchical built-in self-test with on-chip diagnosis for VLIW processors," 14th IEEE Intl. Symp. on Design and Diagnostics of Electronic Circuits and Systems, pp. 143-146, Cottbus, 2011.

[14] G. M. Swift, F. F. Fannanesh, S. M. Guertin, F. Irom and D. G. Millward, "Single-event upset in the PowerPC750 microprocessor". IEEE Trans. on Nuclear Science, vol. 48, no. 6, pp. 1822-1827, Dec. 2001.

[15] F. W. Sexton. “Microbeam Studies of Single-Event Effects”. IEEE Trans. on Nuclear Science, vol. 43, no. 2, pp. 687–695, Apr. 1996.
[16] Y. Xu et al. “An Accelerator-Based Neutron Microbeam System for Studies of Radiation Effects”. Radiation Protection Dosimetry, vol. 145, no. 4, pp. 373–376, Dec. 2011.
[17] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor". Proc. 36th Annual IEEE/ACM Intl. Symp. on Microarchitecture (MICRO-36), pp. 29-40, Dec. 2003.
[18] R. Velazco, S. Rezgui and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection". IEEE Trans. on Nuclear Science, vol. 47, no. 6, pp. 2405-2411, Dec. 2000.
[19] R. Mansour and Velazco, “An automated SEU fault-injection method and tool for HDL-based Designs”. IEEE Trans. on Nuclear Science, vol. 60, no. 4, pp. 2728–2733, Aug. 2013.
[20] A. Chatzidimitriou, G. Papadimitriou, C. Gavanas, G. Katsoridas and D. Gizopoulos, "Multi-Bit Upsets Vulnerability Analysis of Modern Microprocessors," 2019 IEEE Intl. Symp. on Workload Characterization (IISWC), pp. 119-130, 2019.
[21] S. Z. Shazli, M. Abdul-Aziz, M. B. Tahoori and D. R. Kaeli, "A Field Analysis of System-level Effects of Soft Errors Occurring in Microprocessors used in Information System”, IEEE Intl. Test Conf., paper 24.3, Oct. 2008.
[22] D.V. Kodavade, S.D.Apte, “Troubleshooting Microprocessor Based System using An Object Oriented Expert System”. Intl. Journal of Adv. Computer Science and Applications, vol. 3, no.5, pp. 111-116, 2011.
[23] M. Peña-Fernandez, A. Lindoso, L. Entrena, M. Garcia-Valderas, Y. Morilla and P. Martín-Holgado, "Online error detection through trace infrastructure in ARM microprocessors". IEEE Trans. on Nuclear Science, vol. 66, no. 7, pp. 1457-1464, July 2019.
[24] “CoreSight Program Flow Trace. Architecture Specification”, ARM Ltd., IHI0035B, 2011.
[25] “CoreSight Components. Technical Reference Manual”, ARM Ltd., DDI0314H, 2009.
[26] A. Serrano-Cases, L. M. Reyneri, Y. Morilla, S. Cuenca-Asensi and A. Martínez-Álvarez, "Empirical Mathematical Model of Microprocessor Sensitivity and Early Prediction to Proton and Neutron Radiation-Induced Soft Errors," IEEE Trans. on Nuclear Science, vol. 67, no. 7, pp. 1511-1520, July 2020.